ANGLIA RUSKIN UNIVERSITY

FACULTY OF SCIENCE AND TECHNOLOGY

CONCEPTUAL DEVELOPMENT OF A NOVEL DIGITAL CONTROLLER FOR OPTIMISED OPERATION OF A HYBRID RENEWABLE ENERGY SYSTEM

JOHN DARVILL

A thesis in partial fulfilment of the requirements of Anglia Ruskin University for the degree of Doctor of Philosophy

Submitted January 2018

Acknowledgements

I would like to express my deep gratitude to my supervisor, Dr Alin Tisan, for his consistent support and guidance over the last five years. I would also like to offer my thanks to my second supervisor, Professor Marcian Cirstea, not only for his support during my PhD but also for inspiring me during my undergraduate degree and sparking my interest in this topic. This project would not have been possible without the invaluable input you both offered.

ANGLIA RUSKIN UNIVERSITY

ABSTRACT

FACULTY OF SCIENCE AND TECHNOLOGY

CONCEPTUAL DEVELOPMENT OF A NOVEL DIGITAL CONTROLLER FOR OPTIMISED OPERATION OF A HYBRID RENEWABLE ENERGY SYSTEM

JOHN DARVILL

January 2018

The popularity of renewable energy has increased dramatically in the last decade as it becomes increasingly apparent that an alternative to fossil fuel is required. In order for this market to continue to grow, there are a number of technical issues which must be overcome. One such issue is the instability of power supplies which rely on unpredictable natural resources. A popular solution to this problem is the use of a number of complementary renewable power sources to create a stable power network serving a small geographical area. In this project, a novel control solution for such a system is proposed and modelled using a new approach.

A comprehensive review of the existing research literature was used to place in context the hybrid renewable energy concept. This included identification of a system topology as well as the novel control solution. A model of the system has been developed using a combination of PSim and MATLAB. This model allowed for the operation of the system to be simulated and subsequently verified. Upon completion of successful system level simulation testing, VHDL implementations of the control solution were created and incorporated into the model. Further simulations were then carried out based on a specific hardware target, an All Programmable System-on-Chip (APSOC) device featuring a dual-core CPU and an FPGA fabric.

The novel control solution is primarily aimed at the optimal control of the system's power converters. This controller employs a Neuro-Fuzzy algorithm to provide gain scheduling for a PI type controller. The test results indicate that improvements were achieved in the stability of the power converters in comparison to more traditional approaches, offering improved response times and a reduction of the output voltage error. Moreover, a state-based algorithm is utilised, which was demonstrated to ensure that the renewable energy sources are optimally complementary.

Table of Contents

List of Figures

List of Tables

List of Appendices

1 Introduction

Traditional power generation techniques rely on the burning of natural resources, such as coal or gas, to create kinetic energy which can be transformed into electricity. Using this approach, power can be reliably supplied over a great area using large, centralised power plants. However, in recent decades it has become abundantly clear that the burning of such natural materials is responsible for the release of pollutants into the atmosphere. As the environmental damage caused by these practices has become more apparent, it has become increasingly important for new approaches to be adopted. The use of renewable energy sources, such as solar photovoltaic (PV) cells, wind turbines and hydropower, has become established as a preferred alternative to the traditional generation methods. As a result of this, the popularity of renewable energy sources, such as wind and solar, has increased considerably in the last decade. In order to drive the continued growth of these markets, further technological advances must be made to overcome inherent challenges.

As renewable energy sources rely upon the behaviour of unpredictable natural resources, one major challenge is the reduction of instability which is inherent in power supplies which utilise these technologies. One approach which is becoming increasingly popular is the use of a number of complementary sources and storage devices within a system. This has the effect of improving both the stability and the security of the supply. This methodology requires power to be delivered to a smaller area, with the control becoming decentralised in comparison to the current electrical grid. This decentralised approach, which requires a higher number of distributed power generation stations, is commonly referred to as Distributed Energy Resources (DER).

In this research project, a novel approach to the modelling of a DER type system is presented. This system makes use of a solar photovoltaic (PV) cell, a wind turbine and storage elements. This project specifically aims to contribute to the growing research into the control of such systems. In particular, a novel approach to the control of power converters for the renewable energy sources is designed, featuring the use of a new intelligent control solution to improve the performance of the PI algorithm with non-linear plants. The DER model is primarily utilised to demonstrate the capabilities of this novel control approach.

1.1 Research Objectives

This research project has a total of four objectives which are intended to guide the overall aim of initiating a novel control solution which is suitable for use in power converters within DER systems. These objectives relate not only to the creation of the novel control concept but also to the implementation within a hardware device. Each of these objectives is defined in more detail in the rest of this section.

The first objective of this project is the creation of a model for a hybrid renewable energy system using a novel modelling approach. This model encapsulates the behaviour of the power electronics, the renewable energy devices and the control solution. However, this model doesn’t consider the transmission lines, the communication buses between the renewable energy converters and the dump load. To meet with this objective, a comprehensive review of the current state of the art is firstly carried out, to ascertain the system topology and an optimal control solution. In order to determine the best methodology for the model of the system, a review of existing modelling solutions is then carried out.

The second objective of this project is the extensive simulation of the system in order to validate its performance. These simulations firstly consider the individual power converters, to demonstrate the characteristics of the novel control solution. A number of realistic use case scenarios can also be created, which are used to demonstrate how the renewable energy sources complement one another in this system. As part of the methodology review carried out for the first objective, suitable modelling tools are selected to streamline this process.

Once the model of the system has been proven at a conceptual level, the next objective is the realisation of the novel aspects of the control solution within a suitable hardware device. An FPGA is chosen as the primary hardware platform for this task, as it offers a high level of parallel processing capabilities. The VHDL modelling language is utilised to create a model of the novel digital controller, which can be used to prove the performance of the design. Using the Vivado design suite, a free EDA tool from Xilinx, this implementation is further analysed to ascertain its temporal performance and logic utilisation requirements.

The final objective of this research project is the validation of the digital controller implementation. This covers simulations of the model, firstly in isolation and then in consideration with the power electronics. Modern co-simulation techniques are utilised as part of this objective, allowing for SPICE based simulations of the power electronics to be carried out in conjunction with the VHDL model. Testing of the real silicon device is also considered as a part of this objective.

1.2 Methodology

As was discussed in the previous section, the primary objective of this research project is the creation of a novel control solution for power electronic converters. In order to demonstrate the capabilities of this new controller, a DER type system is modelled and simulated. As a part of this work, there are three distinct modelling elements which must be considered – the system level controller, the renewable energy sources and the power electronics. The PSim software tool is used in the modelling of the power electronics and renewable energy sources. This is a specialist tool which is designed to simplify the modelling of power electronics circuits, featuring a number of pre-defined component models. This represents an efficient, rapid prototyping solution for these elements of the system.

The Matlab software tool is utilised as the primary tool in the modelling of the controller, which includes the design of several Artificial Intelligence (AI) based algorithms. This software package is chosen as it includes a number of pre-packaged tools which are intended to simplify the task of designing such algorithms, with the Fuzzy Logic toolbox being one example. In addition to this, Matlab features a powerful yet relatively simple syntax which makes it ideal for the modelling of control algorithms. Using this tool, it is also possible to create a Simulink model which can interface with the PSim software. This allows for co-simulation between the two packages, which helps to optimise the modelling environment.

Another objective of this research project is the realisation of the controller algorithms within a suitable hardware device. The Xilinx Zynq All Programmable System-on-Chip (APSOC) is chosen as the final hardware target for this purpose. Therefore, the controller is primarily implemented using VHDL, with the C programming language being used for supporting functionality. In order to model the functionality of the controller, the QuestaSim simulation software is utilised. This is a powerful environment which is designed for the simulation of hardware modelling languages. A key benefit of this is the ability of PSim to interface with this environment, meaning that the power electronics and controller can be considered in a single environment. The synthesis, layout and programming file generation shall be carried out using the Xilinx Vivado tool, which also allows for further analysis of the implementation to be carried out.

1.3 Original Contributions

As part of this research work, there have been a number of original research contributions, some of which have resulted in peer-reviewed publications. These research papers are included in the appendices, whilst their accomplishments are summarised below.

The first original knowledge contribution stems from the use of a new modelling environment for the tuning of PI controllers. This environment utilises co-simulation between PSim and Matlab to create a rapid modelling approach. The research paper which resulted from this work considers the case of a buck converter, fed by a solar PV cell, which is controlled using a PI algorithm. The tuning of the PI controller is carried out using an artificial intelligence based algorithm known as Particle Swarm Optimisation (PSO). The flow of this algorithm is well suited to the Matlab modelling tool, whilst the power electronics and solar PV can be modelled more efficiently using PSim. This paper illustrates how co-simulation between these tools creates a rapid modelling solution for the PSO algorithm.

The next original contribution to knowledge is the creation of a novel gain scheduling control solution which is utilised in the power converters. This methodology features the use of a Neuro-Fuzzy control algorithm to provide gain scheduling for a PI-type controller. Adopting this approach improves the performance of the essentially linear PI algorithm with a highly non-linear plant, such as a power converter. The research paper in which this was presented considered a boost type power converter fed by a solar PV cell. Comparisons were made showing how this approach reduces output errors in comparison to a more traditional fuzzy logic based gain scheduling controller.

The final original contribution relates to the implementation of the gain-scheduling controller within an FPGA. Following a review of the existing architectures for the Neuro-Fuzzy algorithm, inefficiencies in the current approaches have been identified. A new architecture has been devised which reduces the calculation time whilst requiring only a small increase in the logical utilisation of the FPGA. The research paper in which this is presented considers a basic use case and makes comparisons with the existing methodology from which it is derived. The paper concludes that the scalability of the architecture is improved, whilst there are a number of simulations presented which demonstrate how the latency of the calculation is greatly reduced.

1.4 Structure of the Thesis

In chapters 2 and 3 of this thesis, detailed background information is introduced for the field of DER systems and control solutions for power converters respectively. As part of this background discussion, a literature review is also presented on both of these topics. In the following chapter, the methodology is presented. This begins with a review of the different modelling tools and hardware solutions which are commonly used before the approach utilised in this project is presented and discussed. The model of the DER system is introduced in chapters 5 and 6, which includes the presentation of a number of simulations to demonstrate the effectiveness of the novel control solution. This begins with a discussion of the power converters and their individual controllers, in chapter five, before the entire system is considered in chapter 6. The FPGA implementation of the main novel control solution is discussed in detail in chapter 7, including simulation and hardware testing. Finally, chapter 8 demonstrates the behaviour of the FPGA implementation in the context in which it will be utilised. A number of conclusions are then drawn at the end of the thesis, to illustrate how the objectives of the research project have been met and to highlight possible areas of further work.

2 Review of Existing Renewable Power System Technologies and Advanced Research Trends

Traditional energy generation schemes rely on the burning of fossil fuels such as coal and gas, an approach which results in the emission of carbon and other pollutants into the atmosphere. As the damage caused by these sources became apparent, a global trend has emerged which seeks to reduce carbon emissions generated from such practices. In the last few years, this has become a headline-grabbing trend thanks to high-profile initiatives such as the Paris climate agreement (Clark and Stothard (2015)). A number of European countries, such as the UK (Pickard and Campbell (2017)), and France (Chrisafis and Vaughan (2017)) have similarly announced an upcoming ban on carbon producing diesel and petrol engines. It seems likely that others, most notably Germany (Briscoe (2017)), will also follow suit shortly. In order to drive this trend, it is obvious that alternative, clean energy sources must be considered. Renewable energy sources, such as wind turbines and solar photovoltaic (PV) cells, are increasingly becoming a preferred alternative to the traditional sources.

The renewable energy market has enjoyed substantial growth over the last twenty-five years, with energy generation growing at an average of 2.5% in OECD countries, nearly double that of conventional energy generation (1.4%) (International Energy Agency (2017)). This growth rate is especially pronounced in the areas of Solar PV cells and Wind turbines, which have recorded growth rates of 43.3% and 21.4% respectively during this period, as shown in Fig 1. In absolute terms, the total installed power for wind and solar PV increased by 443.1TWh and 35 TWh respectively (International Energy Agency (2017)). Despite this rapid growth in these markets, hydropower continues to be the most widely used renewable energy source, accounting for more than half of renewable energy production as is shown in Fig 2. Hydropower, which is obtained from water energy, is a very well-established method of power production. It has the flexibility to meet with peak supply demands as well as being capable of acting as a storage system, making it a stable source of power. In 2016, the total energy generated by hydropower amounted to 1401.9TWh, compared to 599.4TWh for wind power and 218.3TWh for solar PV power (International Energy Agency (2017)).

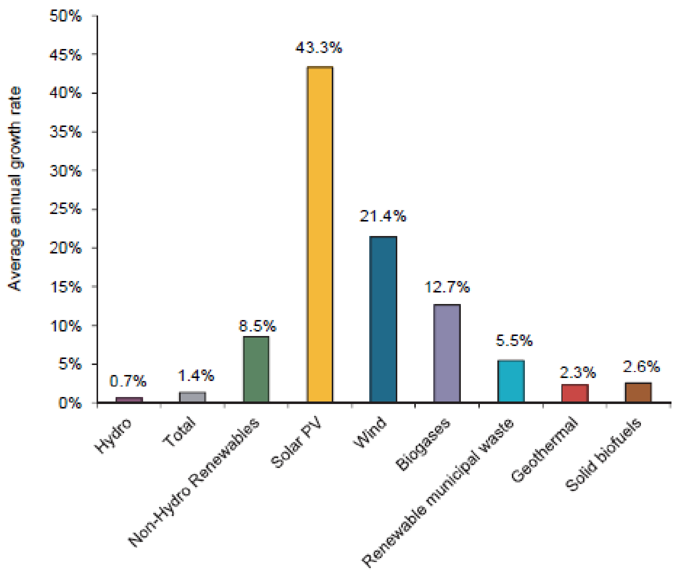


Fig 1 Annual Growth Rates of Electricity Production from 1990 to 2016 in OECD Countries (International Energy Agency (2017))

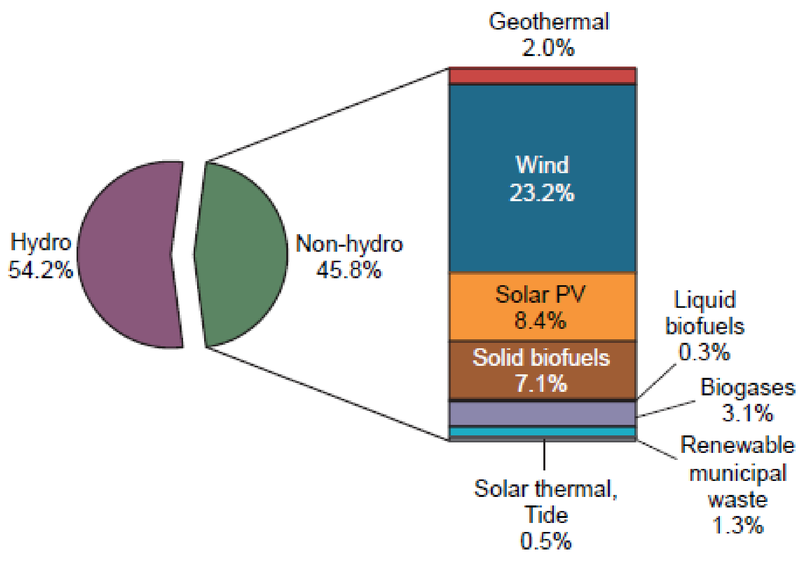


Fig 2 Shares in OECD Renewable Electricity Production for 2016 (International Energy Agency (2017))

In the last decade, the solar PV market has seen a shift away from Europe and towards China. This has been true not only in consumption terms but also in manufacturing terms, with China supplanting Germany as the largest manufacturer of solar PV modules. Fig 3 shows how production has rapidly increased since 2005 and how the manufacture of modules has shifted towards China, as well as Asia more generally. This shift towards Asian manufacturing bases has helped to reduce production costs, a trend which has been exacerbated by the accompanying increase in manufacturing volume. As a result of this, the installation price for solar PV energy has dropped significantly in the past decade. Mature markets such as Italy and Germany are now price competitive with traditional energy generation (International Energy Agency (2014)). The continuing fall in production prices, combined with strong interest from large industrialising countries such as China and India (International Energy Agency (2016)), means that the solar PV market is forecast to continue its strong growth rates. According to the International Energy Agency (2014), solar PV could potentially account for 16% of global energy production by 2050. This would represent a marked increase from the current market share of less than 10% (International Energy Agency (2014)).

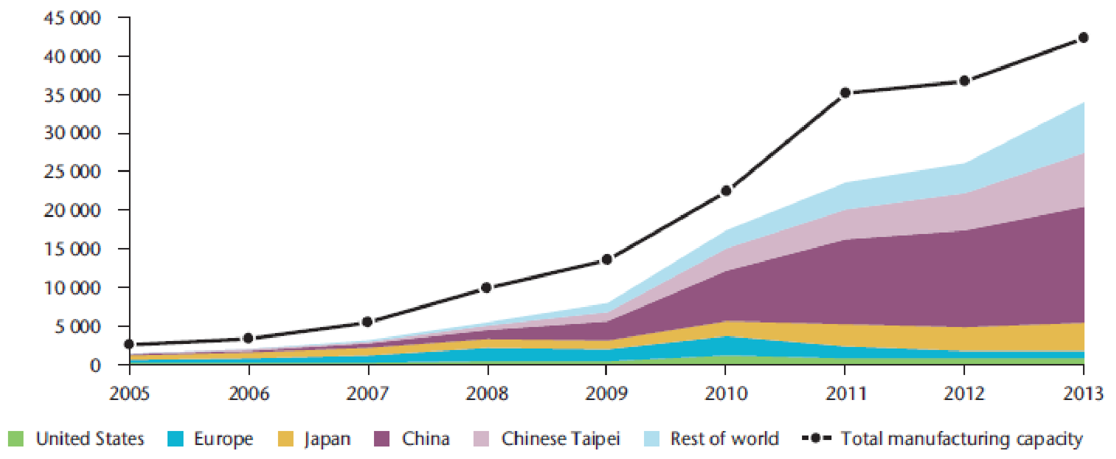


Fig 3 Solar PV Manufacturing by Country (SPV Market Research (2014 cited by International Energy Agency (2014)))

Wind power has seen a similar surge in popularity, with installed capacity more than doubling between 2009 and 2013 (International Energy Agency (2013)). Fig 4 shows how the cumulative installed wind power increased in the period between 1995 and 2012. Wind turbine technology has seen rapid development over the past few years, with improvements in tower height and blade length, in particular, helping to increase their power capacities (International Energy Agency (2013)). Fig 5 shows how the capacity factor of wind turbines increased in the ten years from 2002. This improvement in technology, coupled with falling production costs, means that wind power is becoming increasingly competitive with traditional power sources. In mature markets with good wind resources, such as Brazil, Australia and New Zealand, on-shore wind farms are already cost competitive with newly built power plants (International Energy Agency (2013)). The off-shore market remains less price competitive than the on-shore market, owing largely to the extra complexity of maintenance and installation. According to the International Energy Agency (2013), the wind market is set to continue exhibiting a high growth rate. Forecasts show that by 2050 wind power could be responsible for meeting around 15% of global power demand. This would represent a substantial increase in the current level of around 2.5% as measured in 2013 (International Energy Agency (2013)).

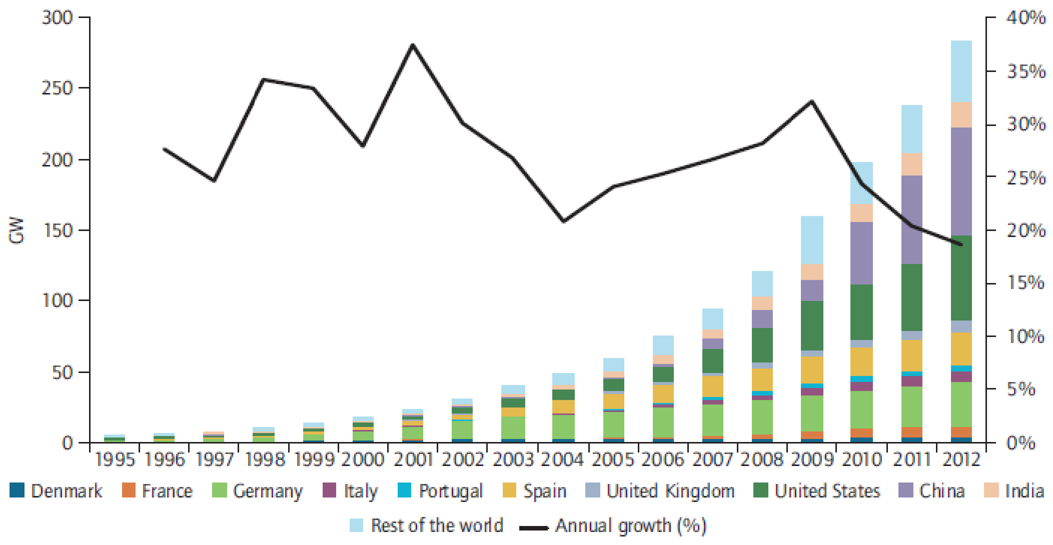


Fig 4 Global Cumulative Growth of Wind Power Capacity (International Energy Agency (2013))

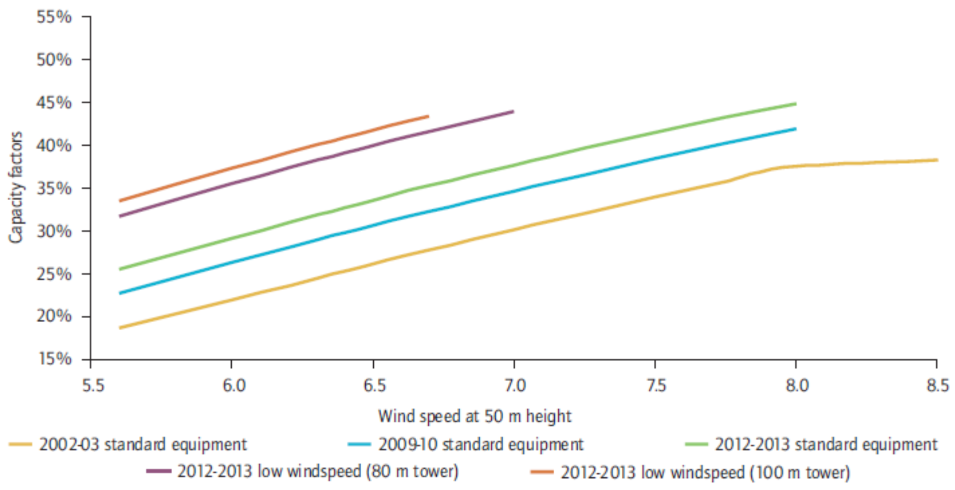


Fig 5 Capacity Factor History of Selected Wind Turbines (Wiser and Bolinger (2013))

Whilst not as popular as the more mature renewable energy technologies, there is still a market for bioenergy and geothermal power. Bioenergy focuses on the creation of biomass materials, which can support the use of coal in traditional power plants. The co-firing of turbines using a mix of coal and biomass is the most cost-efficient way of producing electricity from biomaterials. This stems largely from the fact there is no need to invest in new equipment and existing infrastructure, as traditional power plants are sufficient for the deployment of bioenergy. As well as the co-firing method, which has limitations on the amount of biofuel that can be used, there are a number of other methods which are being developed. However, none of these systems are currently widely deployed. There has been considerable growth in bioenergy take up since 1991, with output more than doubling up to 2009 to contribute around 1.24% (Brown, Muller and Dobrotkova, 2011) of the world energy supply. Geothermal power relies on the earth's heat, estimated to be 5,500°C, to produce energy. This heat is generally drawn from naturally occurring sources – hydrothermal (volcanic) resources, gaps in the Teutonic plate and hot rock. Whilst the required volcanic resources are limited by location, it is possible to use hot rock in all locations. These hot rocks can be used to produce steam, which in turn is used in steam-based generators. It is anticipated by the International Energy Agency (2011) that the market for geothermal power will show robust growth over the coming decades, with forecasts suggesting it could account for 3.5% of global power by 2050 compared to less than 1% now, as shown in Fig 6.

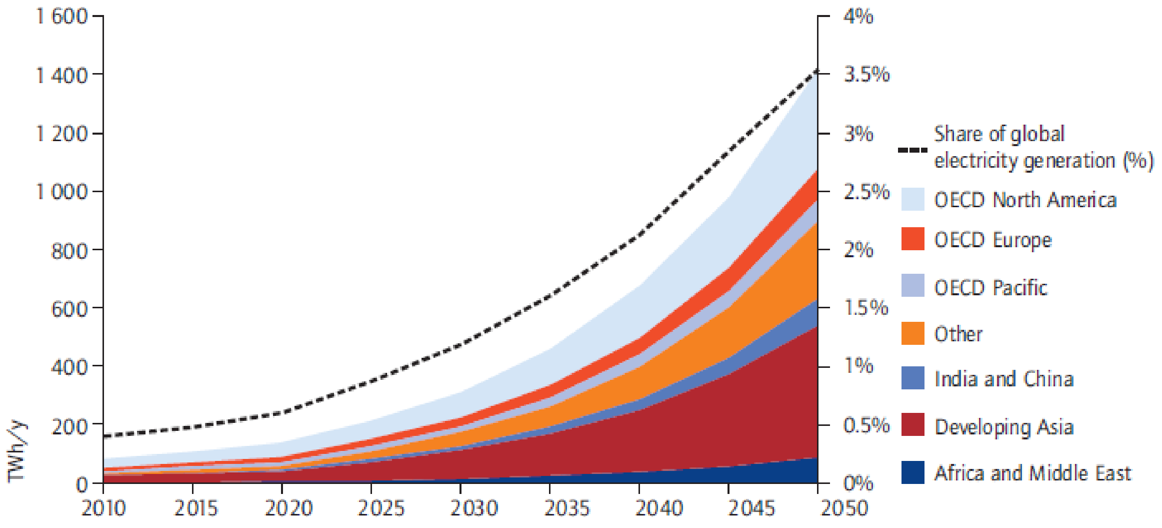


Fig 6 Forecast Growth in Geothermal Power Production by Region (International Energy Agency (2011))

Despite the high growth rates for wind and solar energy, there still remain a number of challenges which need to be overcome to continue to drive this development. Price remains perhaps the main barrier for mass penetration, although the International Energy Agency (2016) anticipates that by 2040 most renewable energy will be price competitive without subsidies. There are also a number of technical challenges which must be overcome, particularly within the manufacturing and technologies for new sources. Work is being done in improving the efficiency of currently expensive manufacturing processes, such as PV cell fabrication and large wind turbine construction, which will be essential in ensuring that the popularity of renewable energy continues to increase.

One major technical challenge with renewable systems is controlling the inherently unstable nature of power supplies, which rely on unpredictable natural resources. One solution to this problem that has gained popularity is the use of a number of complementary sources together, such as wind generators and solar PV cells, which help to smooth fluctuations in the supply. Using this methodology, the energy system will provide power to a smaller area and the control will consequently become decentralised, an approach known as distributed energy resources (DER). Although the idea behind DER systems is well established, there is still a need for enhancements to be made before the use on a large scale becomes a reality. A major area which needs improvement is the control of such a system, which must be able to produce a stable and secure supply, which meets the tight power quality needs of the grid.

In this section, a review is conducted into the major technologies involved in the development of a new DER concept. This starts with the consideration and review of the renewable power generation technologies and then also considers power converters, which form a key part of the system. Finally, a review is conducted into the current research being done in relation to renewable energy systems and power converter technology.

2.1 Renewable Energy Sources and Technologies Employed

There are a number of different technologies used in the generation of power from renewable sources, covering aspects of mechanical, electrical and electronic engineering. In this section, some of the key energy sources and technologies are introduced.

2.1.1 Wind Systems

The potential for wind energy is vast, with the relative maturity of the technology meaning they are already a popular renewable energy source. Conversion of wind power into mechanical motion is a centuries-old concept, dating back to the use of windmills for the grinding of wheat. Although modern wind machines are much more sophisticated, with larger and more aerodynamic blades, this same basic mechanical principle is at the heart of the modern wind turbine. When wind comes into contact with the large blades of a wind turbine, the wind power forces the turbine to rotate, with the rate of rotation being proportional to the speed of the wind. This motion is then carried through to the generator, via a gearbox, where it forces the rotor into motion and thus starts to produce power. A typical block diagram for a wind turbine system is shown in Fig 7.

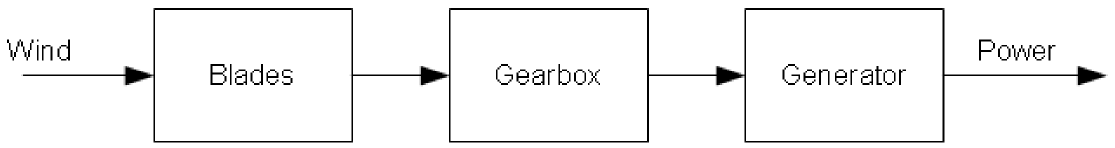


Fig 7 Wind turbine block diagram

The blades play a critical role in the wind turbine system as they are the primary component in converting the potential energy of the wind into rotating mechanical motion which can be used by the generator. As a result of this, the blades play a big role in the efficiency of the system and their design is critical to the performance. In order to achieve greater efficiency, there are three key considerations - size, number of blades and aerodynamics. The size of the blade, or rather the length, dictates how much wind will come into contact with the blade. This means that bigger blades are better, as more energy hitting the blades equates to greater energy production. Similarly, better aerodynamics allows for higher rotational speed which again results in higher power generation. The aerodynamics of the blades have received much attention and modern blades now closely resemble the construction of aeroplane wings, featuring aerofoil type shapes to reduce drag and maximize lift as far as possible. The number of blades is actually limited by other factors as it becomes increasingly difficult to produce aerodynamically efficient blades which are adequately large and robust as more are added to a turbine. Therefore, the best balance has been found with three blades per turbine.

Electrically speaking, the generator is the core component of the wind turbine system as this is where the potential of the mechanical energy is transformed into electrical energy. There are a number of different generators which can be used in a wind system but the most commonly used types are permanent magnet synchronous machines (PMSM) and induction generators. DC generators are also used occasionally, especially to integrate with DC systems such as those which are employed with PV cells, but they generally require much higher revolution speeds to generate power. Due to this inefficiency, it is more common and practical to use an AC generator with a power converter to achieve DC from the wind turbine.

The induction generator is popular for use in wind turbines due to its ability to produce power at a range of frequencies. The stator of the induction generator is constructed from slotted iron cores, with wires being wound into the slots to form the electromagnet. The rotor is made up of cylindrical conductive bars evenly spaced around the periphery. In order to produce power, the speed of the rotor must be greater than the stator's synchronous speed, which is dictated by the frequency of the stator's AC supply. The induction generator is a fairly simple form of generator and is therefore relatively simple to control, as well as being rugged and inexpensive. All of these factors contribute to their ongoing popularity in wind turbines.

The key defining feature of PMSM generators is the presence of permanent magnets in the construction rather than the use of an electromagnet, as is required in most generator technologies. In a PMSM generator, the stator is again constructed from a core which features a number of windings, whilst the rotor features fixed permanent magnets. The operation of the generator is fairly simple, with the rotation of the rotor mounted magnets creating an electromagnetic field in the stator. This, in turn, generates an electrical output through electromagnetic induction. PMSM generators are a popular choice of generator in wind turbine systems, largely owing to the fact they require no excitation or gearbox meaning they are simple to integrate. The main disadvantage of these devices is that the magnetic flux density of permanent magnets is less than that of electromagnets, effectively limiting the power capabilities. In addition to this, they also tend to be more expensive than other generator technologies.

2.1.2 PV Systems

Conversion of solar power to electrical energy encompasses a broad range of technologies, all of which are relatively mature. Whilst some of these technologies, such as concentrated solar, rely heavily on favourable environmental conditions, photovoltaic (PV) cells are able to produce power as long as there is at least some irradiance. PV cells are therefore a very popular renewable energy source, which in turn has led to them becoming relatively inexpensive. It is because of this popularity that this type of energy source has been chosen as a key component in the energy system designed later in this paper.

The operation of PV cells is based on a principle known as the photoelectric effect. Materials which exhibit this property have electrons which can be displaced when photons collide with their surface. This action causes an electrical current to be incurred, with the electrons becoming free to flow through an externally attached circuit. In order to facilitate this operation, PV cells are constructed from two differently doped semiconductors known as a p-n junction. In this structure, a p-type material is created through the addition of ‘acceptor’ impurities, which means there are more electron holes – gaps in the atomic structure which have the potential to be filled by electrons. Conversely, n-type materials are created by adding ‘donor’ impurities resulting in more free electrons in the material. These two different materials are separated by a depletion zone in a p-n junction. This is caused by the holes and electrons diffusing into the opposite material type, creating a small region where there is no longer a neutral charge. As the charge in this zone is in opposition in the two different material types, this prevents any further diffusion until there is sufficient excitation to overcome this barrier. In a PV cell, the source of this excitation is the photon displacing an electron within the semiconductor and allowing it to move freely once again. When this semiconductor is connected to an external circuit, this free electron (and a complementary hole) can move through the external circuit acting as a power source. Whilst traditional production of PV cells uses silicon as the base material, there is also an emerging trend towards other thin film materials such as copper and selenium. Although these materials are inefficient compared to their silicon counterpart, they offer considerable cost savings. Therefore, they have an important role to play in bringing PV cells to a wider audience.

The equivalent circuit of the PV cell is shown in Fig 8, with the input being a light-dependent current source. This would indicate that more irradiance produces more power at the output and this is, to an extent, true. However, it must also be considered that according to the thermal characteristics of the device, the extra heat associated with areas of high irradiance actually reduces the efficiency. This reduction in efficiency is the reason that areas with high irradiance and heat, such as the desert states in the USA, are areas where concentrated solar is often preferred to PV cells. The diode in this circuit models the previously discussed operating principle of the semiconductor material. Like the PV cell, diodes are constructed from a simple p-n junction, allowing them to model the operating principle of the semiconductor material, as previously discussed. The two resistive components in the equivalent circuit are included to model the characteristic resistance of the solar cell. The series resistance of the cell, modelled as Rs, is derived from the contacts of the cell and the parasitic resistance of the semiconductor. These characteristics have the effect of reducing the amount of useful power which the cell can produce. The shunt resistance, Rsh, is caused by slight defects in the material and the manufacturing process. These defects also have the effect of reducing the power which can be used, especially at low levels of irradiance. The output of a single PV cell is relatively small, at just 0.5V, meaning a number of cells must be utilised to create higher power panels. Series connection of cells is used to achieve higher nominal voltage, whilst parallel connection increases the operational current.

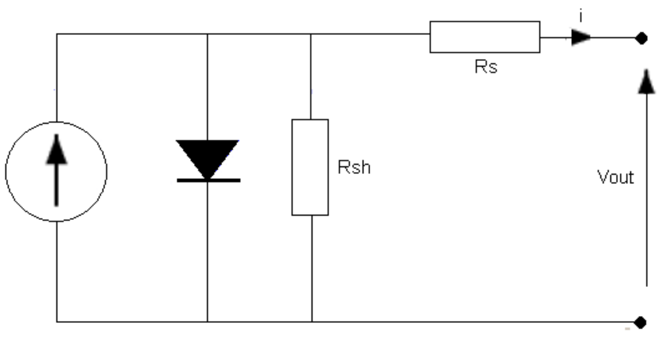


Fig 8 Solar cell equivalent circuit

2.1.3 Fuel Cells

One of the key problems in renewable systems is energy storage, which is a crucial element in ensuring the stability of the supply. Excess power generated during times of high production must be captured in order to be utilised during periods of lower production. Whilst batteries are currently a popular solution to this problem, these technologies are environmentally unfriendly. As a result, they negate a central advantage of using renewable energy systems. Hydrogen fuel cells present an alternative to batteries and are becoming increasingly popular, despite the relative expense of the technology. These fuel cells consume only hydrogen and produce only water as a waste product. This makes them an ideal storage element for use within renewable energy systems.

The basic structure of a typical cell is shown in Fig 9 and the operation is straightforward. There are two inputs to the cell - one being hydrogen, which passes over the anode, and oxygen, which passes over the cathode. The anode has a catalyst layer between itself and the hydrogen input, which acts to split the electrons from the ions. These loose particles then seek to bond with the oxygen which is present on the cathode. The ions (H^+) flow through the electrolyte membrane, which blocks the electrons and forces them to flow through an external circuit. Upon reaching the cathode the hydrogen particles fuse back together, as well as bonding with the oxygen to form water. As with the previously discussed PV cell, this cell only produces a small voltage, usually around 1V under no load conditions, meaning a number of cells must be cascaded to form a higher power device.

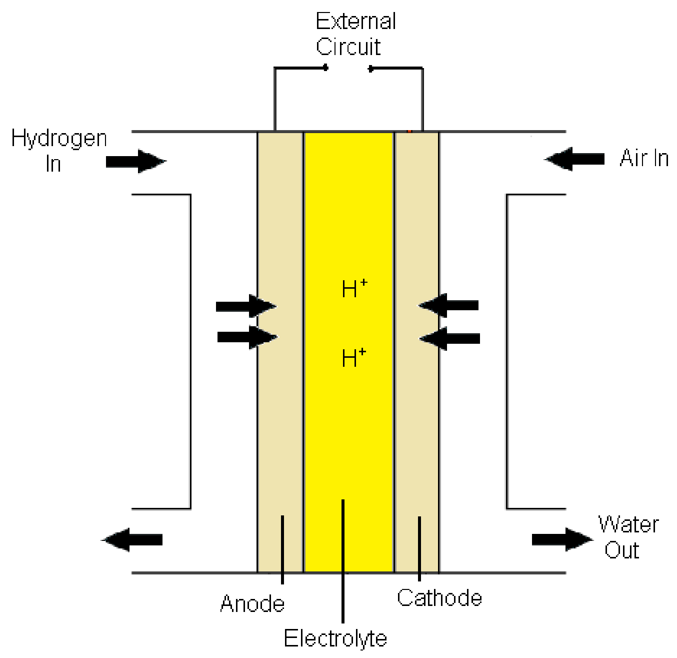


Fig 9 Hydrogen fuel cell structure

The fuel cell does suffer from some inefficiency in its structure, as is demonstrated by the equivalent circuit shown in Fig 10. There are three elements in the fuel cell, all modelled as resistors, which limit the power which can be produced. The first of these is modelled as Ract, which symbolizes the activation losses of the cell, caused by the amount of energy which is required to force the reactions to take place. The next parameter is Rohm, which represents ohmic losses due to the electrical resistance of the materials in the cell. Finally, the concentration losses, modelled as Rcon, are associated with the transport of the gases around the cell.

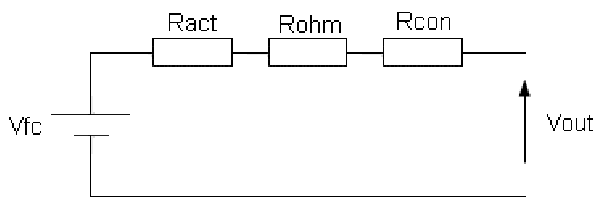


Fig 10 Fuel Cell Equivalent Circuit

The major barrier to mass penetration of the fuel cell technology lies with the production of hydrogen itself. Although one of the most prevalent elements in the world, its propensity for bonding with other compounds means naturally occurring hydrogen is not readily available. The most popular method for production is the use of a reverse fuel cell reaction to remove hydrogen compounds from water, a process known as electrolysis. To achieve this, the same structure as the one presented in Fig 9 is utilised. However, a power supply is attached to the anode and cathode and water is fed into the cell. This induces the reverse reaction and has the effect of creating hydrogen. Therefore, a typical hydrogen fuel cell system consists of an electrolyser, the fuel cell itself and a storage tank. This method of producing hydrogen is time-consuming, expensive and highly inefficient, making the fuel cell an unattractive option for wide-scale use at present. However, as hydrogen production techniques improve, the fuel cell is likely to challenge batteries as the dominant power storage element in electrical systems.

2.1.4 Other Sources

A number of additional renewable energy sources exist, with the most prevalent of these being hydropower. These power plants exploit the potential energy of flowing water to generate electricity. This is typically achieved using a reservoir which feeds into another water source, typically an outflow river. The reservoir is designed to have a higher elevation than the outflow source, meaning that there is a constant flow of water in one direction. By placing a propeller in this stream, the flow of water can be converted to rotational energy, which is utilised to drive the rotor of a generator. Although the use of a reservoir is common, it is also possible to use other methods to create the flow of water. Other popular methods include using pumps to generate a flow or exploiting the natural water flow in rivers. Hydropower plants tend to have large power producing capabilities, with large plants, such as the Three Gorges Dam in China, being capable of delivering greater than 10GW of power (Acker (2009)). Although hydropower plants can form part of a DER system, it is much more common for them to be connected directly to the grid.

Another popular renewable energy source is concentrated solar power, which aims to harness the potential energy of the sun. These energy sources consist of two major parts: a set of reflectors to collect solar energy and a generator. The reflectors are utilised to focus the light emitted from the sun and consequently convert it into heat. This heat energy is then utilised by the generator to create electrical power. The generator takes the form of a heat engine such as a steam-based generator. Thermochemical reactions are also being researched, which may provide a future method of converting the heat into electric power, although no such systems have yet been deployed (Wald (2013)). In order to be most effective, concentrated solar plants require locations which receive a great deal of bright sunlight. The most popular locations for concentrated solar plants are the desert regions of California and Spain (Ren12 (2017)). These plants tend to be grid-connected rather than forming part of a DER system. This is largely down to their requirement for deployment in areas which tend to be unsuitable for other sources such as wind turbines.

2.2 Main Power Converter Topologies used in Renewable Applications

In order for renewable energy sources to produce useable power, it is necessary to regulate the output, be it for connection to the grid or for use within standalone systems. This regulation is performed by electronic power converters which can be classified in one of four ways – AC to DC (rectifiers), DC to AC (inverters), DC to DC and AC to AC. In this section, the basic operating principles and some key topologies are presented for each of these different classifications. Additionally, a special class of converters, which features bi-directional power flow, making them suitable for use with storage elements, is discussed.

2.2.1 History of Power Converters

Since the establishment of the electric grid, there has been a need for circuits which produce a regulated DC output. In early power supplies, this was done by linear type regulators such as that shown in Fig 11. In this system, the first stage increases or decreases the AC voltage through the use of a transformer. A rectifier circuit is then utilised to convert the voltage into DC. Although there are more modern ways of rectifying the AC voltage, the most popular method remains the use of diodes arranged as a full bridge, as is discussed later in this chapter.

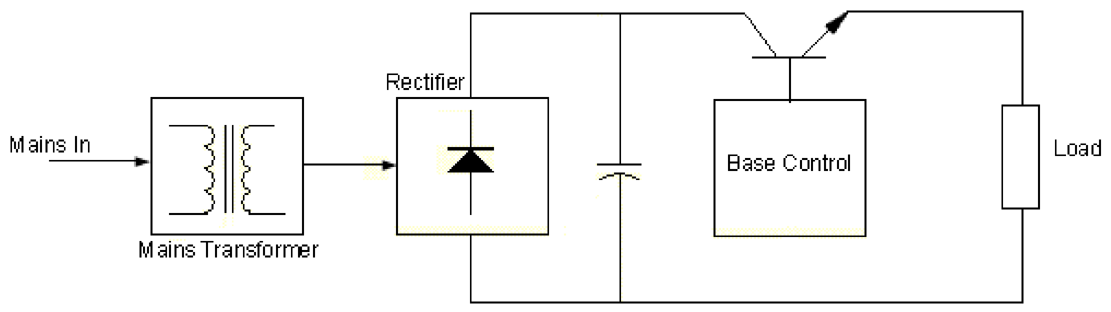


Fig 11 Traditional linear supply

In the traditional power supply, DC regulation is often achieved through the use of a linear regulator. In this circuit, the transistor is run in its active area, behaving as a variable resistor in order to dissipate energy. Although this method is simple to implement, there are a number of limitations associated with its usage. In modern power supplies, it is common for the efficiency of a power converter to be over 80 or even 90 percent. In contrast to this, a simple linear regulator is often less than 50% efficient. The other major problem with this type of regulator is that the lost energy is dissipated as heat, meaning careful thermal management is required. If this is improperly managed, the supply can overheat and consequently cease to function. This also creates a need for the inclusion of bulky heat sinks, which often add significant cost.

As semiconductor technology has developed, switch mode power supplies have become more popular. This technology use switches to repeatedly connect and disconnect the supply voltage to the output. Although this has the disadvantage of introducing harmonics into the system, a much higher level of efficiency is achievable. As energy is not being dumped through an artificial load, there is also much less heat dissipation. Using switching methodologies, it is also possible to increase the frequency of the AC input prior to rectification. This allows for smaller, less expensive transformers to be utilised. Given these advantages, it is clear why switching converters are now the preferred method for power conversion applications.

2.2.2 Rectifiers

Rectifiers are used to convert AC voltages into DC using devices which allow the flow of electrons in one direction. This can either be a diode (uncontrolled) or a semiconductor device which doesn’t feature natural commutation, such as a thyristor or transistor (controlled). Rectifiers are generally classified as being either single or three phase, depending on the type of input.

Single phase rectifiers are typically easier to implement than their three-phase counterparts and are usually uncontrolled, utilising diodes as the rectifying component. The most widely used single phase rectifier is the full bridge circuit which features four diodes, as shown in Fig 12. In this topology, there are two diodes on at any one time, with D1 and D2 being switched on during the positive half cycle of the input whilst D3 and D4 are in conduction during the negative half cycle. This results in a rough DC output, which fluctuates between zero and the peak amplitude of the input. It is typical to have a capacitor on the output of this circuit to smoothen these fluctuations.

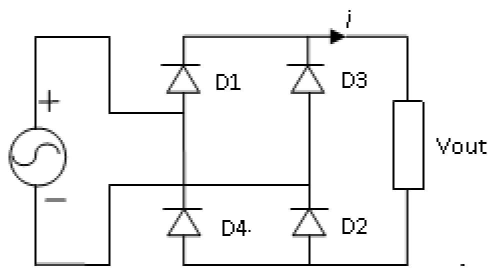


Fig 12 Single phase full bridge rectifier

The most commonly used three-phase rectifier is the uncontrolled bridge, which is an extension of the single phase bridge. In this topology, an extra pair of diodes are utilised and arranged in the same manner as those shown in Fig 12. One phase is then fed into each of the three diode pairs. The top three diodes of this circuit form one group, which are in conduction during the positive half of the input cycle. Conversely, the bottom three diodes are in conduction during the negative half cycle and form a separate group. One diode in each of these groups is in conduction at any one time, albeit never on the same input pair (leg). This topology relies on natural commutation, with switching occurring whenever the voltage across any diode is greater than the voltage across the others in its group. As a result of this, each of the diodes will conduct for a period of 120° per cycle. This topology is often referred to as the six pulse rectifier as there are six segments in its operation and six pulses in the rectified voltage for each period of the supply voltage.

If the diodes used in these circuits are replaced by thyristors, then it is possible to control the commutation. Utilising this controlled topology makes it possible for the thyristors to be triggered later in the cycle by delaying the triggering pulse by a firing angle, α. The result of this is that the average DC output can be controlled as is shown by the equation given in Eq. 1 . Despite the fact that extra control of the regulated output is possible, this topology remains less popular than the simple diode solution. This is mainly down to the fact that DC-DC converters are often included downstream, offering simpler control and improved power quality.

$$V_{out} = 0.9 V_{in} \cos\alpha \qquad \textit{Eq. 1}$$

2.2.3 Inverters

Inverters are closely related to rectifiers and are used to convert DC power into AC, either three phase or single phase. These converters are especially important in grid-connected renewable energy systems, where they provide an interface between the renewables and the grid. The basic topology of a single phase inverter is shown in Fig 13, with the three-phase variant requiring an extra pair of transistors to function. Each of the three legs in the inverter, consisting of a pair of transistors, constitutes one output for topologies requiring a three-phase output.

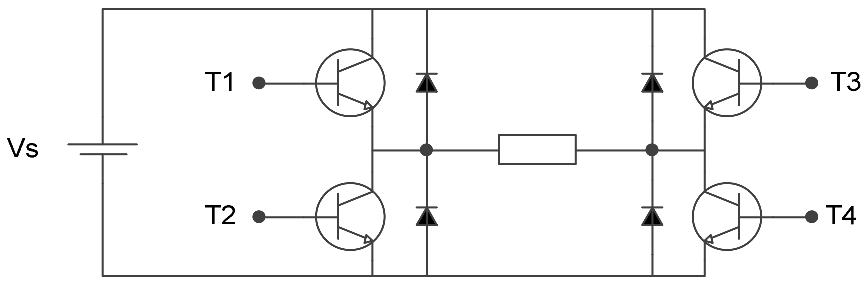


Fig 13 Full Bridge Inverter

A commonly used control methodology for the single-phase inverter topology is known as bipolar switching. When this is employed, the controller features a comparator fed with a triangular waveform signal and a sinusoidal reference. When this reference is greater than the triangular wave, T1 and T4 are in conduction. This has the effect of creating the positive half cycle of the AC output. Similarly, when the reference is less than the triangular wave, the negative half cycle is created by forcing T2 and T3 into conduction. Therefore, this methodology creates a bipolar voltage across the load which oscillates between plus and minus V_s.

Another commonly used control methodology is known as unipolar switching. The controller for this methodology is similar to that utilised for the bipolar switching mode, except that an extra sinusoidal reference is employed. The two reference signals are 180 degrees out of phase with one another, with one reference controlling the switching of T1 and T2 and the other T3 and T4. This scheme means that when the duty cycle of leg A (T1 and T2) is at its highest, the duty cycle of leg B (T3 and T4) will be at its lowest and vice versa. Consequently, the voltage generated by leg A is greatest when the voltage on leg B is at its lowest and vice versa. As the voltage across the load is equivalent to the difference between the voltage generated in each leg, this has the effect of producing an alternating waveform which oscillates between plus and minus half of V_s. Although this is more difficult to implement, it effectively doubles the harmonic frequencies. As this moves the harmonics further from the fundamental frequency of the output, this simplifies the output filter.

As previously discussed, three-phase inverters feature three switching legs. Despite this, the control is similar to that employed in the unipolar switching technique utilised in the single-phase inverter. However, the controller features three sinusoidal reference signals which are 120 degrees out of phase. Each of these reference signals regulate one leg in the inverter, creating half a cycle which switches between zero and half the input voltage and another half cycle which switches between zero and minus half the input voltage Although the bipolar switching method can be employed for three-phase inverters, it is not commonly used as it requires twelve separate switches. This obviously increases both the complexity and the expense without delivering any significant benefits.

It is important to consider that the inverter output is not a perfect sine wave and as such will contain a number of harmonics of the fundamental output frequency, f_{out}. This must be taken into account during the design of the converter in order to optimise the power quality. A particularly significant characteristic of the circuit is the relationship between the triangular and sinusoidal waveforms used in the controller. Of particular interest is the frequency modulation, which is a ratio between the fundamental frequency and the PWM switching frequency (f_s), as is shown in Eq. 2 . The output harmonics are located at intervals of the frequency modulation, showing how the switching frequency can be utilised to manipulate their location. As it is simpler to dampen out higher frequency signals, it is desirable to have a higher switching frequency and, consequently, harmonics. However, the downside to this is that the switch utilisation is inversely proportional to its switching frequency, illustrating how the design is a delicate balancing act.

$$F_{\mathrm{mod}} = \frac{fs}{fout} \qquad \textit{Eq. 2}$$

Inverters form a core component of renewable energy systems and the need to improve the quality of their output has been a research theme which has created more sophisticated switching techniques. One popular method which has been developed, especially for three-phase inverters, is the use of space vector modulation (SVM). In the three-phase topology, only eight switching states are allowed, when the consideration is taken that the input voltage should never be short-circuited to the ground. Of these states, six give a non-zero state and these, therefore, form the basis of the SVM technique. The system state space is split into six distinct quadrants, using these switch states as the boundaries. The output voltage can then be transformed into a Cartesian vector, using the αβ transform, which can be plotted within this space. This vector represents a rotating point which is dependent upon the three phase voltages. The PWM signal is produced by switching between one of the zero voltage vectors and the two vectors that form the boundary of the active quadrant. Although more difficult to implement than the sinusoidal method, SVM is a popular choice as it benefits from improved utilisation of the input voltage and a reduction in harmonic distortion. The implementation and improvement of SVM techniques is currently a topical and popular research theme (Lee et al (2010), Saqib and Kashif (2010) and Quach et al (2012)).

2.2.4 AC to AC Converters

AC to AC converters are more complex due to the fact that there are two variables which must be controlled - the frequency and the voltage. There are three commonly used topologies for AC - AC conversion – DC link, cycloconverters and matrix converters. The most common topology for three-phase systems is the DC link which consists of a pair of converters. In this topology, the AC input is rectified into a regulated DC voltage using a rectifier circuit. This voltage is then applied to an inverter which converts the DC back into AC, meaning that the conversion is effectively AC-DC-AC. This topology, which is also referred to as an indirect AC – AC converter, is popular for two reasons. Firstly, it is made of two fairly simple converters, making it easy to implement. Secondly, the use of transistors in the switching stages means that power can flow in either direction, making it suitable for grid integration.

Another widely used topology in AC to AC converters is the cycloconverter, which performs the conversion in one stage, unlike in the DC link. The most basic topology is the one phase to one phase converter which is shown in Fig 14. This consists of back to back bridge rectifiers in anti-parallel with one another, with thyristors being used as the rectifying component. The operation of this converter can be split into two sections, with only one of the anti-paralleled rectifiers acting during each half of the output period. The two rectifiers are never in conduction at the same time. Instead, one provides the negative portion of the output waveform and the other provides the positive portion. This converter is primarily utilised to transform the frequency, which is relatively simple to achieve. For example, if an input is applied with a period of 2π and the desired output frequency is a quarter that of the input, both the positive and negative converters are active for a period of 4π. However, this produces a non-sinusoidal output, as is shown in Fig 15b, with the average DC level instead producing a bipolar square wave. A common solution to this issue involves controlling the firing angle of the thyristor so that the average DC level is also sinusoidally variable, as is shown in Fig 15c. This technique has the added benefit of reducing the number of harmonics which are present in the output.

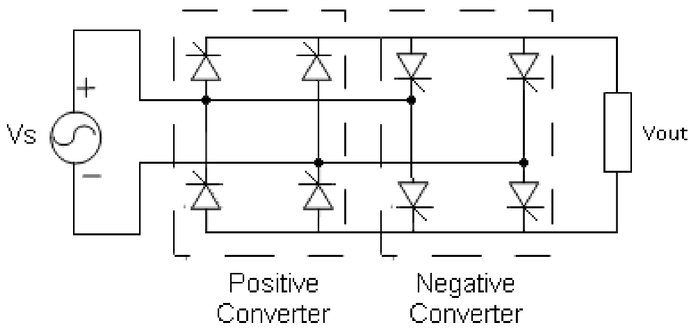


Fig 14 Single phase to single phase cycloconverter

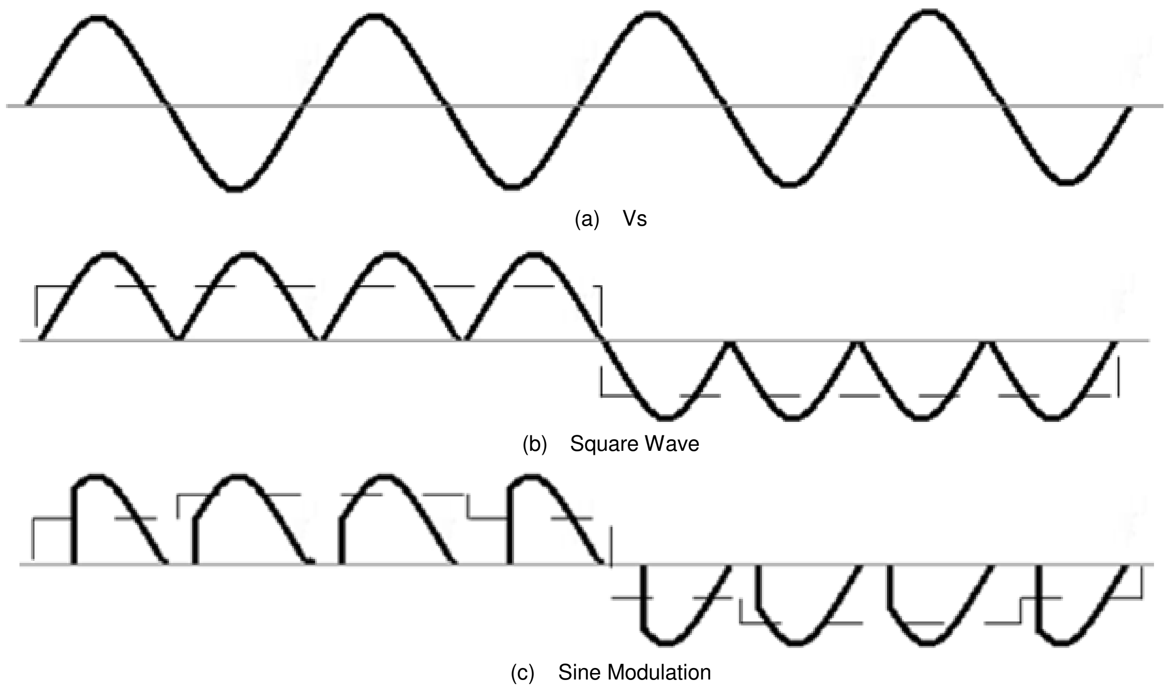


Fig 15 Single phase cycloconverter waveforms

In order to utilise a three-phase input with a single phase output, some modifications to the cycloconverter topology are required. The major difference in this instance is the requirement for an extra pair of thyristors in each of the anti-parallel rectifiers. Each of the three phases is then applied to one leg, which is made up of two series thyristors, in each of the converters. The operating principle of three phase cycloconverters is similar to that previously discussed for the single phase variant. The two rectifiers are again utilised in opposite half cycles whilst the firing angle is modulated sinusoidally to improve power quality. The only major difference is that a slightly different switching sequence must be utilised to facilitate the extra phases and thyristors. Although this methodology is more complex to implement, the resultant output is smoother than in the single phase input approach. Three phase to three phase cycloconverters operate on the same principle except that three parallel rectifiers are required to generate the three output phases.

The third family of AC to AC converters are known as Matrix converters, as shown in Fig 16, which are a relatively new topology. This type of converter is designed to work with a three-phase input and can have either a single or three-phase output. This circuit features bi-directional switches, which are typically comprised of anti-parallel transistors, and offers a number of advantages over the other AC to AC converters. Unlike in the previously discussed methodologies, both the input and output current are sinusoidal in nature (Karaca and Akkaya (2012)). This results in a better quality of power and a reduction in harmonics which are present in the output. The use of bi-directional switches also allows for full power factor control and bi-directional power flow. In comparison to the DC link method, there is also the advantage that the smoothing capacitor, which tends to be bulky and has a limited lifetime, is no longer required. Despite these advantages, there are several disadvantages which have limited the popularity of the matrix converter. The most notable drawback is the lack of a truly bidirectional switching device for use in this circuit. This effectively doubles the number of switches required, which has an impact on both the cost and efficiency of the circuit. However, the biggest challenge that this presents is a difficulty in commutating current between the switches. There are two methods for achieving this, with there being either a short period of time where no switch is in conduction or a short period where two switches are in conduction. Although the advantages of the matrix converter are currently outweighed by the disadvantages, there is still a great deal of research interest in this topology.

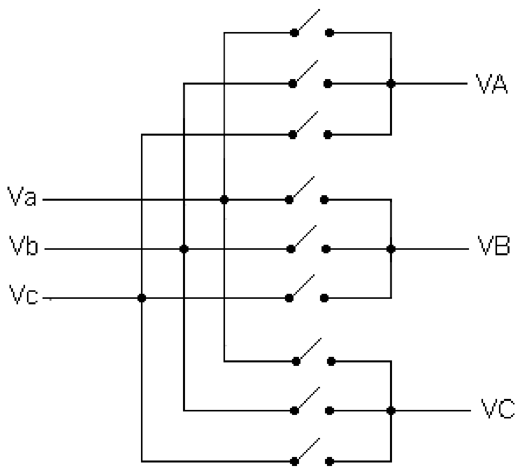


Fig 16 Three-Phase Matrix Converter

The operating principle of the matrix converter is similar to the three-phase inverters which have been discussed previously. The three inputs are sequentially sampled for a small period, with a reconstructed AC output being made of a number of these samples. The bidirectional switch conducts in one direction to generate the positive portion of the output, whilst the negative portion is created with the switches conducting in the opposite direction. The output, which consists of a number of high-frequency samples, is then filtered to give a smooth sinusoidal output. As with the cycloconverter, the firing angle can also be modulated in order to improve power quality. The control of the matrix converter is a popular research topic (Hajbani et al (2012), Rivera et al 2012 and Yusoff et al (2012)) and there is no well-established approach for the modulation of this signal. One widely used methodology is similar to the inverter method, whereby a sinusoidal reference signal is compared to a triangular wave. Another commonly employed technique involves the use of SVM, whereby the switching states are mapped as vectors. The switches are then modulated according to their position within the state space. This method is again similar to the methodology previously discussed for the three-phase inverter.

2.2.5 DC to DC Converters

Another popular family of power converters are DC-DC converters, which are used to convert DC voltages from one level to another, with the input often being unregulated. These circuits can be used to either increase or decrease a DC voltage and typically use PWM to avoid the use of a capacitive or inductive storage element. When applied to DC converters, PWM refers to the periodic connection/interruption of the input supply voltage to the output. Although not an actively used topology, the circuit shown in Fig 17 gives the basic operating principle of the PWM modulation scheme. By controlling the switch in such a way that there are controlled periods of disconnection, it is possible to decrease the average DC level and this forms the basis of the PWM controlled converter.

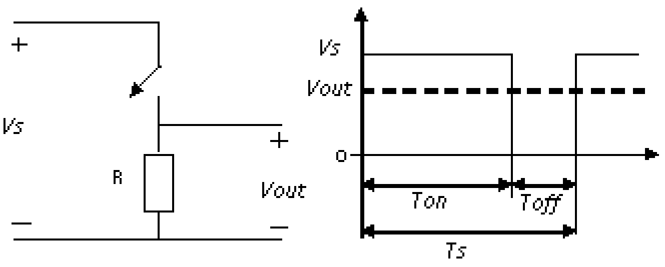


Fig 17 DC-DC Converter Principle

Although not as commonly used as PWM, another technique which has some advantages is a technique known as pulse frequency modulation (PFM). In this methodology, the duty cycle of the controlled switch is constant and the average DC level is controlled by the frequency of pulses. Higher frequency pulse trains result in a higher average DC voltage than is produced at lower frequencies. A closely related variant of the PFM mode of control is a burst like operation, whereby the transistor is modulated when the voltage falls below a given setpoint. The transistor is then switched off when the output capacitor becomes sufficiently charged. This is a popular method when there is only a very small amount of load current. Although the PFM mode of operation is generally less efficient than PWM, at light loads this technique does exhibit greater efficiency. Therefore, a common approach is the utilisation of a dual PWM-PFM control scheme for maximum efficiency (Zhang and Maksimovic (2010)).

In practical applications, the circuitry for a DC power converter is more complex than that shown in Fig 17. In order to smoothen the switched DC voltage and improve the power quality, an inductor and capacitor are required. The characteristics of the inductor have an important effect on the performance of the converter, particularly on the current flow from the inductor during periods when it is discharging. The most desirable operating range for a DC-DC converter is known as continuous conduction mode, where the inductor is never allowed to become fully discharged. In this mode of operation, the system performance will become mathematically simple and the system exhibits a high degree of stability. When there is a period where the inductor becomes fully discharged, a condition known as discontinuous conduction, the system's performance becomes more complex. When this occurs, a dependence on the inductor current is introduced and the stability of the system is hindered. Therefore, it is generally good practice to design converters to operate in the continuous conduction mode of operation. However, this can be difficult if simple linear control is utilised and a wide range of output currents are required.

One of the most common applications for DC-DC converters is reducing a DC voltage to some pre-defined level. The most popular topology for this is the buck converter, which is shown in Fig 18. This circuit uses the same principle as shown in Fig 17, with the PWM controller switching in such a way that the average output is dropped to a defined level. There are two extra components which must be included to ensure correct operation - the diode and the LC filter. The diode is used to overcome the problem of stored inductive power during the period that the switch is off, effectively acting as protection for the switch. The LC filter is a low pass filter which is designed to average out the switched waveform to give a good quality DC output. When the transistor is in conduction, the input voltage supplies power to both the inductor and the load. The diode allows the energy stored in the inductor to flow through the load to give a continuous supply of current when the transistor is no longer in conduction. The varying current in the inductor, caused by constantly charging and discharging the device, has an effect on the output of the circuit as it creates a small voltage ripple. The amplitude of this can be calculated using the equation shown in Eq. 3 . It can be observed from this that by making the LC filter cut-off frequency (fc) much smaller than the switching frequency (fs), the ripple can be kept to a minimum.

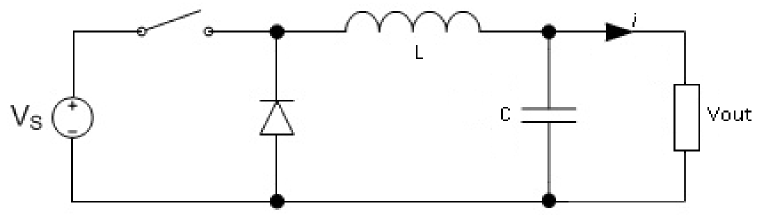


Fig 18 Buck converter topology

$$\Delta V_0 = \frac{\pi}{2}(1-D)(\frac{f_c}{f_s})^2$$ *Eq. 3*

Another popular function of DC-DC converters is to increase the input voltage to a higher regulated output. The most widely used topology for this is the boost converter, which is shown in Fig 19. In this circuit, the diode acts as an isolator when the switch is in conduction. During this period, the input charges the inductor whilst the output is supplied solely by the capacitor. This often means that a large output capacitor is required for stable operation, particularly with heavy loads. When the transistor is switched off, the diode is forward biased which allows the stored energy in the inductor to be released into the capacitor. This extra energy has the effect of increasing the output voltage. The ripple of the circuit can be modelled using the equation shown in Eq. 4 , where the time constant (τ) is equivalent to the load multiplied by the capacitance and D is the duty ratio of the switch. This shows that the voltage ripple has a dependence on the output current of the circuit meaning it can be difficult to minimise the ripple when a wide range of currents is required.

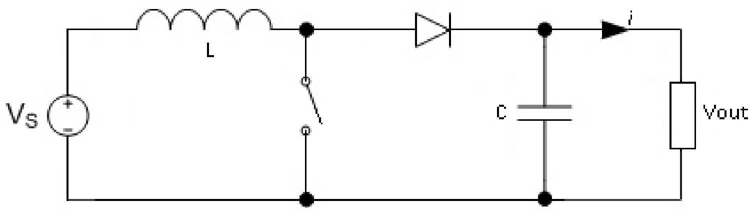


Fig 19 Boost converter topology

$$\Delta V_0 = D\frac{T_s}{\tau} \qquad \text{Eq. 4}$$

The DC-DC converter topologies which have been presented so far feature no form of galvanic isolation. The full bridge converter, which is shown in Fig 20, is the most commonly utilised converter when galvanic isolation is required. As with the inverter topology, this type of circuit can be controlled in one of two ways, either using bipolar or unipolar switching. In bipolar mode, the switching is carried out in such a way that the output is a square wave which swings from Vin to –Vin. In contrast, unipolar switching creates a square wave output which ranges from Vin to ground. The use of unipolar switching has two advantages. Firstly, it reduces harmonic content as the switching frequency is effectively doubled, which also makes filtering easier. Secondly, no rectifier is required after the transformer in the circuit, as the output is already DC. Despite these advantages, the ease of controlling and implementing the bipolar mode means this is a much more widely used methodology.

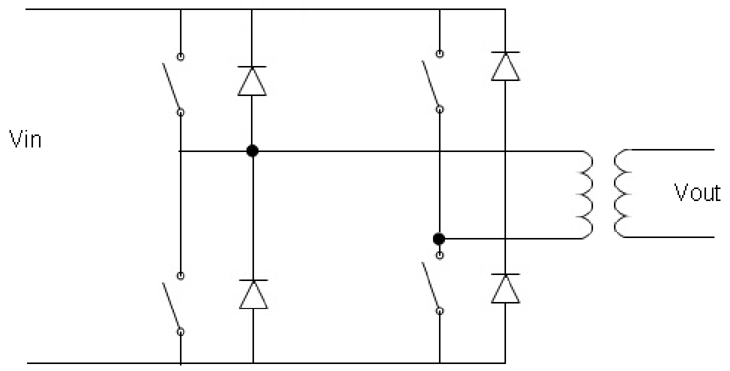


Fig 20 Full bridge converter topology

2.2.6 Bidirectional Power Converters

One important consideration of any renewable energy system is the ability to store energy as this helps to smoothen out the inherent fluctuations in the supply. Whilst there are several devices capable of storing energy within the system, the most commonly used component is the battery. The requirement for bi-directional power flow in these storage elements has led to the growth of a new class of converters which feature bidirectional DC-DC conversion. This is a newer classification of power converter, meaning that there has been less work done in establishing standard topologies. Despite this, there are a number of popular circuits which are typically used. These can be broadly classified as being either isolated or non-isolated. In this section, the main types of bidirectional converter are presented and their operation discussed.

The simplest of the bidirectional converters is shown in Fig 21. This acts as a boost converter in one direction and a buck in the other. In this topology, a pair of transistors are utilised in order to allow for current to flow in both directions. In order for power to flow from the high voltage side to the low voltage side, T1 is used as the main modulating transistor. When the power must flow in the opposite direction, then T2 is used as the main control switch. In both instances, the second transistor is designed to operate in a manner which is complementary by conducting when the main control transistor is switched off. In order to prevent a short circuit, a small period of ‘dead time’ is required with neither transistor being in conduction when switching between states. Other than this subtlety, the circuit behaves in the same manner as the previously described buck and boost circuits, depending on the direction of power flow.

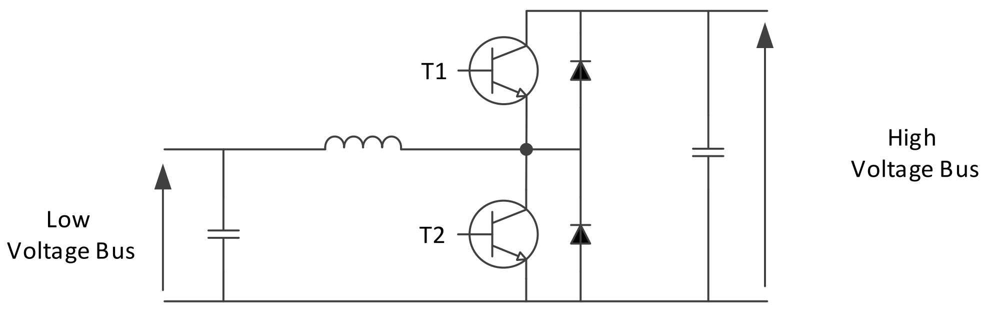


Fig 21 Basic Bidirectional Converter

Whilst the circuit shown in Fig 21 is a useful converter in scenarios where one of the voltage buses is always higher than the other, in practical systems this can’t always be guaranteed. There are a number of different circuits which have been proposed for use when the two buses have a similar voltage. One popular methodology features four switches, as is shown in Fig 22, which allows for the configuration of the converter to be altered depending on the current operating conditions. This converter can change the way in which the inductor is connected to the rest of the circuit so that it operates as either a Buck or a Boost type. If the case is considered where the power flows from Va to Vb, T1 can be switched on and T2 switched off to configure the circuit as a boost converter. The control of the circuit is then similar to the previously discussed bidirectional converter, with T3 providing the main control and T4 complementing it. In contrast, when the buck mode of operation is required, T3 is switched on and T4 off. In this case, the control is provided by T1 whilst T2 acts as the complementary transistor. An additional mode for this converter occurs when the two voltage buses are roughly the same, then all four switches must be used to regulate the voltage levels.

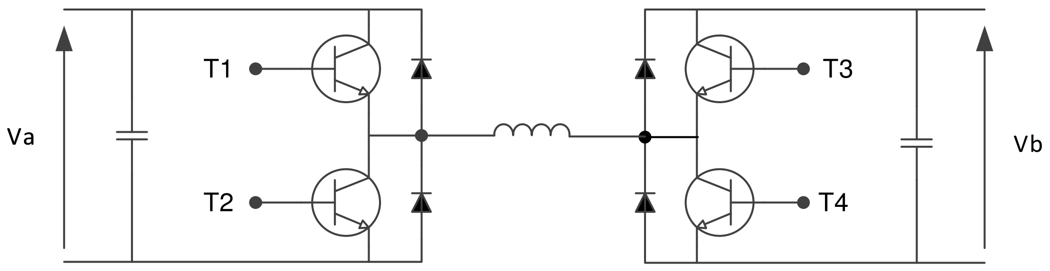


Fig 22 Bidirectional converter with buck and boost capabilities

The bidirectional converters considered thus far have no form of galvanic isolation, which is often a desirable feature. Whilst there are a number of different approaches which have been proposed, the two most common topologies are the half bridge and full bridge converters. To achieve bidirectional power conversion, a pair of cascaded full bridge converters, which were previously discussed in section 2.2.5, can be utilised as shown in Fig 23. The basic behaviour of this circuit is very similar to the unidirectional variant, with the switches being used to oscillate the voltage on the transformer windings between a positive and negative polarity. The regulated voltage output is determined by both the duty cycle of the PWM modulator and the winding ratio of the transformer. In order to achieve power flow in both directions, it becomes important to control the phase difference between the two sides of the transformer. The current in this circuit will always flow from the leading phases into the lagging phase, with the amount of power transferred being proportional to the phase angle. As this necessitates control of the phase angle, the implementation of this circuit is typically more difficult to achieve than the non-isolated varieties presented previously.

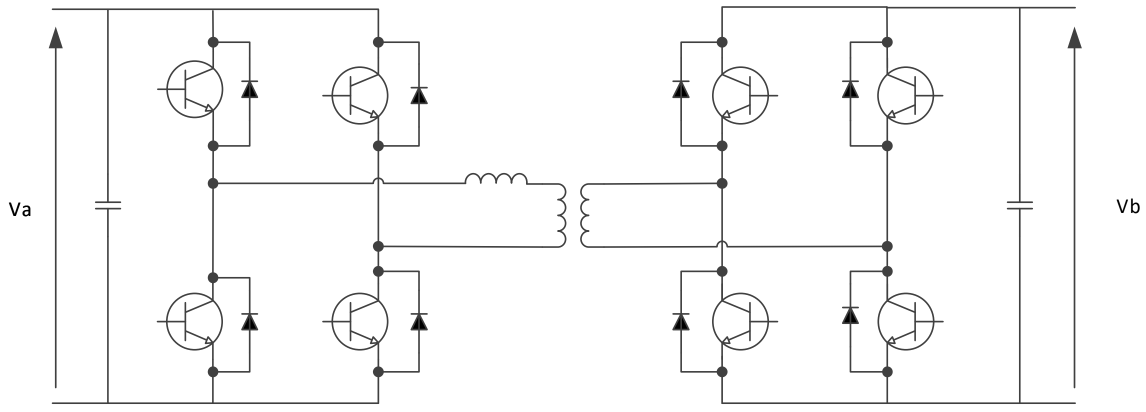


Fig 23 Full Bridge Isolated Bidirectional Converter

The half bridge converter, which is shown in Fig 24, is another popular topology which features galvanic isolation. When current flows from V_a into V_b, the performance of this converter is similar to the full bridge. The voltage regulation is a function of both the duty cycle and the transformer turns ratio. However, a pair of capacitors are utilised in favour of the transistors in the full bridge converter. This creates a voltage divider, meaning that the voltage across the windings is divided by two. When the current flows in the opposite direction, from V_b to V_a, the performance of the circuit is much the same, with the voltage still being a function of the windings ratio and duty cycle. However, in this instance, a centre tapped transformer is utilised which removes the requirement for additional capacitors. In both instances, the circuit relies on the phase lag created between the two sides to create bidirectional power flow. As with the full bridge converter, the current flows from the leading phase into the lagging phase meaning the phase angle must be controlled.

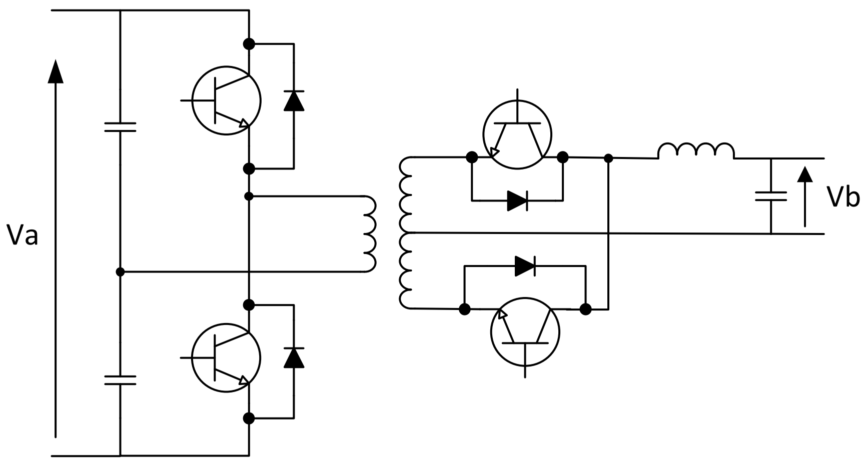


Fig 24 Half Bridge Bidirectional Converter

2.3 *Modern Trends in Wind and PV Systems*

The topic of DER systems is one which is beginning to receive greater attention among researchers as the strengths of such systems make them an attractive option for future energy systems. Although there is a variety of different areas which are currently being investigated, systems presented in literature typically feature one or more renewable energy source and some form of storage, such as a battery or fuel cell. One approach which has gained some popularity and attention is the use of multi-input power converters (Yang et al (2012), Bae and Kwasinkski (2012) and Chen et al (2013)) as these can combine power from different input sources. Although this approach has some advantages, such as a reduced component count and potentially simpler control, it also often involves sacrificing the modularity of the system which is an important feature of DER systems. These converters are still in the research phase and are yet to find widespread adoption, meaning there are no well-established topologies. For these reasons, as well as giving consideration to the fact that the major theme of this project is the design of an optimal controller, such systems will not be reviewed in this section. The main focus of much of the literature, which includes strong control emphasis, is the management of system loads. The load management of DER systems is a considerably more complex issue than in traditional grids due to the unpredictable nature of the power sources and the need for supplementary storage components. A number of these systems are presented in this section, with a focus on optimal control of the resources to ensure the delivery of reliable power.

Wind turbines and PV cells represent two of the more mature renewable energy sources, meaning that they are often included in DER systems. Batteries are perhaps the most popular of the available storage elements, which is largely due to their technological maturity. These three power sources are brought together in the paper presented by Kumaravel and Ashok (2011), where a feed-forward neural network is used as the basis for the controller. In this system, the controller fulfils two roles, ensuring that the maximum power is derived from both sources and charging the battery. The simulations of this system demonstrate its ability to maximise the power generation from all sources. However, there is little consideration of the quality of the power produced and the regulation is fairly poor in this case. Another popular storage element in DER systems is the fuel cell, owing to the fact that it offers environmentally friendly energy when it is charged by renewable power sources. One example of this type of system is presented by Gyawali et al (2011), whereby a wind turbine is utilised as the main power source. A fuel cell is included to provide backup power when the turbine produces insufficient energy on its own. The electrolyser is used to absorb energy during periods of excess power generation. This system features an additional storage element known as superconducting magnetic energy storage (SMES). As the dynamic response of the fuel cell is relatively slow, this device is used to provide voltage compensation when fast transitions occur. This system demonstrates a number of favourable characteristics, such as a well regulated AC output and reliable delivery of power under peak demands. Although the DC bus is well regulated, in steady-state operation there are some noticeable fluctuations when loaded. This is most likely caused by the use of relatively simple linear PI controllers, rather than a more sophisticated solution. One drawback of this system topology is the use of the SMES, which is not currently a well-established technology.

Supercapacitors are a commonly used power source for applications which require the fast release of energy owing to the fact they have a high energy density. This characteristic makes them ideal for voltage compensation during transient conditions. One such system, which also includes a fuel cell, is presented by Thounthong et al (2013). In this paper, a PV cell is used in conjunction with the storage elements to provide DC power. The supercapacitor is utilised as the main source in this system so that its energy density can be exploited. The controller, which is implemented using fuzzy logic, then uses the renewable energy sources to ensure that its level of charge is adequately maintained. The test results in this paper illustrate the effectiveness of this approach, with the output bus demonstrating excellent stability even under fluctuating conditions. This concept has been utilised to good effect in a number of similar systems, such as those presented by Shen and Khaligh (2015) and Somayajula and Crow (2014).

Another way in which the shortcomings of the renewable energy sources can be compensated for is through the use of a diesel generator in conjunction with the renewables. Examples of such systems are presented by Sekhar and Mishra (2016) and Safari and Sarvi (2014). Although this methodology includes a component which is powered using fossil fuels, the diesel generator is purely designed to supply backup power. This means the renewable energy sources generate clean power for the majority of the time the system is operating. The inclusion of an element which has more predictable operation is intended to reduce the complexity of the control solution in these systems.

One advantage of renewable energy systems, and in particular PV cell based systems, is the fact that they are highly modular in nature. However, the systems presented up to this point fail to take advantage of this fact, instead being based on a fairly rigid design. There has been some research interest in creating a more modular approach which would allow for a more flexible type of DER system. One example of such as system is presented by Rodriguez et al (2013). In this system, the interface to the main DC bus is handled by a number of local 500W modules, each of which is responsible for their own power management. These modules also include a communication bus, allowing them to talk to one another to optimise the system's control. Whilst this system has obvious potential, it suffers from poor performance as a converter. The 500W size of each module is also restrictive as it introduces the need for a number of modules to be used for higher power generation.

Whilst there is plenty of literature which explores the use of DER systems, often including detailed models and simulations, there are not many systems which have been practically implemented. The paper presented by Nehrir et al (2011) outlines a few selected locations, such as Hawaii Island, Utsira Island and Hachinohe, but these are still relatively small-scale installations of less than a few hundred kW in total size. One example of a real system, demonstrating the potential of the DER approach, is presented by Papaefthymiou et al (2010). In this paper, a pumped hydro plant and a wind farm are combined to provide power to the island of Ikaria. This highlights one of the major advantages of a DER system – the ability to generate power locally. When compared with the traditional centralised approach, this makes it considerably easier to create power systems for remote locations. This means that DER systems have the potential to provide power to many isolated places, such as islands or in rural areas of developing countries like China and India.

3 Review of Control Technologies and Advanced Research Trends

The field of control theory, and the related topic of feedback analysis, can be traced back to the late 19th century and in particular to the work published by Maxwell (1868). It was during this period that many of the fundamentals which are still in use today were developed. The early 20th century saw further studies conducted into control theory, building the fundamentals of what is today known as Classical Control. It was during the first half of the 20th century that the main components of classical control design and analysis, such as frequency analysis, root locus and Bode plots were introduced. One of the most important developments of this era was the proportional-integral-derivative (PID) controller. This has enabled the creation of a standard, simple to implement family of controllers, which give excellent performance and continue to be widely used even today.

There were two important developments which stimulated the emergence of modern control theory – the advent of the digital computer and a rise in the use of ordinary differential equations to describe system behaviour. These developments allowed for much more sophisticated controllers, which were aimed at improving performance when used with more complex plants. Modern control theory relies heavily on having a mathematical model of the plant integrated into the controller architecture, allowing for adaptations depending upon the state of the plant. Popular examples of this type of controller, commonly known as adaptive control, include model predictive control, auto-tuning and sliding mode control.

The major limiting factor of modern control theory is the need for a mathematical model of the plant. This is often unknown, ill-defined or too complex to allow for the creation of a model with sufficient accuracy. To overcome this limitation, systems based on artificial intelligence (AI) techniques are becoming increasingly popular. Intelligent controllers typically utilise methods such as artificial neural networks (ANN), fuzzy logic (FL) and genetic algorithms. Many systems feature a hybrid mix of a number of these methods, most notably neuro-fuzzy systems, which utilise both FL and ANN. The rise of this type of controller has been made possible by the advances in processing technologies and the advent of FPGAs in particular. These devices provide the fast parallel processing capability required for many of these techniques to be implemented.

In this chapter, the key background topics relating to control theory are presented and explained. This includes consideration of modern intelligent control based systems. A literature review is then conducted to examine the current state of research into modern AI systems. This review places an emphasis on applications in the fields of power electronics and renewable energy systems.

3.1 Control Systems for Power Converters

In this section, some important features of control theory are introduced and explained, starting with important system classifications and characteristics. The classical PID controller is then presented before the main intelligent control methods are discussed.

3.1.1 Control Principles and Traditional Systems

When it comes to the design of control systems, one of the most important concepts is the classification of the systems, based on a number of different factors, which is important in understanding a given systems behaviour. A number of different components are considered in the classification of a system; three of the most important ones are discussed below.

One key system classification is the domain in which the controller operates - either analogue or digital. In the early period of control theory development, there was a lack of affordable, fast digital processing devices. Therefore, analogue based systems were much more prevalent despite being generally more complex to design. As more affordable processing power has become available, both in the form of microcontrollers and FPGAs, digital control techniques have gained more popularity. This method is now much more widely used than its analogue counterpart and offers the advantages of improved speed, high levels of accuracy and greater flexibility.

The nature of the system behaviour, being either linear or non-linear, is another important consideration of controller design. There are two principles which must be satisfied in order for a system to be classified as linear. The first of these, known as homogeneity, states that if the output is y for any given input x, then for an input Kx the output should be Ky, where K is some constant. The second principle, known as superposition, dictates that if inputs x1 and x2 produce the outputs y1 and y2, then x1+x2 will produce the output y1+y2. The classical control methodology was primarily concerned with linear systems, which have much simpler system equations and are consequently simpler to manage. Non-linear systems are much more complex to model and to accurately control. As a result of this, most of the recent developments in control theory, such as intelligent controllers, adaptive control and sliding mode control, have been aimed at improving performance with these plants.

The use of feedback is another system classification which has a large effect on system performance. In a closed-loop system, in which negative feedback is utilised to improve the performance of the controller, the output, or a part of it, is fed back into the controller. This allows the control effort to be adjusted accordingly. Implementation of open-loop systems, which feature no feedback loop, is comparatively simple and inexpensive. Whilst more complex to design, closed loop systems offer a number of significant advantages. This method creates greater stability and accuracy, whilst reducing the susceptibility to disturbances. These desirable characteristics mean that feedback is popular in control systems.

In any control system, regardless of the application area or system classification, there are a set of common characteristics which are used to define and analyse the system's performance. These indicators can be split into two distinct categories. Static characteristics are measured when the system is not being subjected to disturbance, either from an input of another source, and the output has settled into a constant, steady state. Dynamic performance indicators, on the other hand, are considered when the system is in a transient state, either due to a change of input or some other disturbance.

Under static operating conditions, when the input is not changing and the output remains in a constant state, stability is a critical consideration. Any system which exhibits instability is considered to be sub-optimally controlled. After a disturbance is applied, there is a finite period of time after which the output of a given system remains at a constant value. This state of operation is known as the steady state. When this state is unreachable, with the output instead oscillating, this is considered an indication of system instability. There are a number of ways in which this undesirable characteristic can be rectified. Common techniques include the use of a compensator and the application of negative feedback. The other important static characteristic is the accuracy of the output, which indicates the deviation of the actual output from the expected output. The measure of this value, expressed as a percentage, is known as steady state error. Control solutions which feature an integral parameter in the control law, such as PI or PID, have a steady state error which is theoretically zero.

Under dynamic operating conditions, when the system output is in a transient state, the speed of the system becomes the main consideration. Response time is the measure of how quickly the system returns to a steady state after a disturbance is applied to the system. Through inspection of undershoot and overshoot in response to a step input, the stability and damping of a system can also be analysed. In an overdamped system, which is also the most stable of systems, the response time will be slow but there will be no overshoot or ringing. In contrast, an underdamped system, where there is less stability, will exhibit greater responsiveness although some decaying oscillations will be present. It is possible to infer from these states that optimally damped conditions will provide a good balance between stability and speed. However, it can also be deduced that increasing response speed has the effect of reducing stability and vice versa. As the transient response to a step input can tell us so much about the state of the system, it is a very common tool for system testing and analysis.

3.1.2 PID Control

One of the most important developments of the classical control era was the PID controller. This is a simple algorithm which consists of three separate actions - proportional, integral and derivative. The PID algorithm can also be used with some of the actions removed, for example, PI or PD controllers are not uncommon. The proportional element is the simplest of the three actions and can be implemented by applying a gain to the fedback error. If a solely proportional element is used, the control is very simple but the trade-off is poor system performance. Typically, this will result in the presence of an appreciable amount of steady-state error. The integral action effectively acts as a history of the previous errors in the system and can be used to reduce the steady-state error to zero. However, the dynamic behaviour of the system is still suboptimal, with errors occurring with a constantly changing input such as a ramp. The derivative action measures the rate of change in the system output, adding a value to the control law which is proportional to this. However, as the derivative in a system may be influenced by measurement error or noise, this term often reduces the noise immunity of the controller. Therefore, it is common for the derivative term to be excluded from the control law.

Although the PID algorithm, as shown in Eq. 5 , has been applied to a variety of different systems with good effect, it does require manipulation in order to meet the exact requirements of a system. There are three variables in the algorithm – k, Tr and Td – which must be tuned in order to get the best performance. The k term is the gain of the controller and has a bearing on the proportional contribution to the system. Tr is integral reset time which controls how quickly the integral term responds to changes. Finally, Td is the derivative time and determines the contribution which is made by the derivative action. Tuning of PID controllers is a complex task and there is no method which suits every situation perfectly, although there are a number of popular methods such as the Ziegler-Nichols or Cohen-Coon. There are also many software packages, such as MATLAB, which offer automatic optimal tuning of these parameters, which simplifies the deployment of this algorithm.

$$y = k(x + \frac{1}{T_r}\int x.dt + T_d \frac{dx}{dt}) \qquad \textit{Eq. 5}$$

Although the PID is a well-established control technique, there are many limitations which have helped to drive the development of modern control solutions. As has already been mentioned, the tuning of the PID is a complex task and there is still much research being conducted in this area. Another limitation, which arises from tuning, can be the effect that different loads have on the system. As tuning is generally performed at one set point, it doesn't capture the full range of operation of the plant. As a result of this, performance can be optimised for one operating condition whilst being sub-optimal in other conditions. One example of this is in a power converter with a purely resistive load, which in practice will vary over time. If the controller is tuned with no load or a light load, then the performance is not representative of the condition where a heavy load is present and consequently much greater damping. The PID algorithm is essentially linear in nature, which means another major issue is the performance with non-linear plants. These limitations of the PID controller have led to the growth of other controllers, such as adaptive and AI-based methods. However, none of these has replaced the popularity of the PID for simple control solutions. Instead, many methods now aim to improve the performance of the PID through the use of AI to adaptively tune the algorithm, for example.

3.1.3 Intelligent Controllers

The most recent set of developments in the optimisation of controllers has been the introduction of AI techniques. These controllers offer the advantage of improved system performance, especially for non-linear systems, whilst there is no need for the use of a mathematical plant model. The most common AI based controllers use fuzzy logic, neural networks, evolutionary algorithms or some combination of these techniques. In the rest of this section, these control methodologies are discussed.

3.1.3.1 Fuzzy Control

In traditional Boolean logic systems, it is conventional to have statements which are either true or false, depending on a number of variables. This black and white approach differs from the human world whereby there is the possibility for grey areas. For example, in a binary logic system the statement 'is it too hot' can either be answered yes or no, whereas a human will likely give a more qualitative answer such as 'it's a little cold'. Fuzzy logic systems, as well as controllers built on fuzzy logic, are designed to behave like a human expert by taking decisions based on less absolute data, known as fuzzy data logic. The block diagram in Fig 25 shows the general topology of a fuzzy logic controller (FLC).

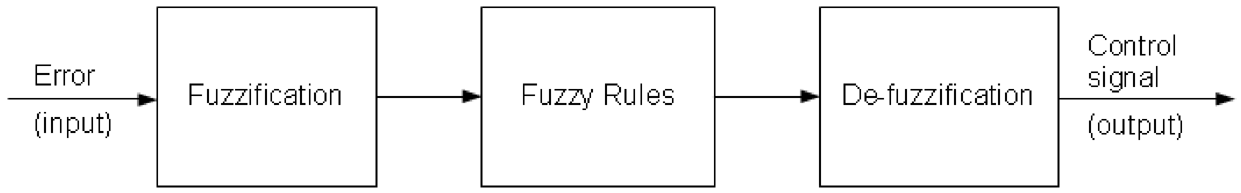


Fig 25 Fuzzy Logic Controller Topology

In these controllers, fuzzification refers to the transformation of data from crisp, absolute input values into fuzzy data sets. This is achieved by defining a number of membership functions, which are created by splitting the range of possible input values into a set of distinct groups, such as the example shown in Fig 26. In classical logic systems, it is possible to have only true or false values, corresponding to 0 or 1, whereas a fuzzy logic system will have varying degrees of membership from 0 to 1. Let's consider an example where heights are taken as input with a membership set called 'tall'. Consider the point at which a person becomes tall in a classical system and there would be a distinct cut off point of say 190cm, whereas in a fuzzy system the set would be weighted so that a person could be considered quite tall. In a classical logic system, this means that a person of 189cm is not considered tall, or has a membership of 0, whilst one who is 190cm is considered tall, having a membership value of 1. By comparison, in a fuzzy logic system, a person who is 189cm will have a degree of membership in the tall set of around 0.9, making them quite tall. This more gradual change in membership gives a much more real-world representation and is one of the reasons why FLC systems are good at making decisions. In a practical system, each input must be associated with at least two membership functions.

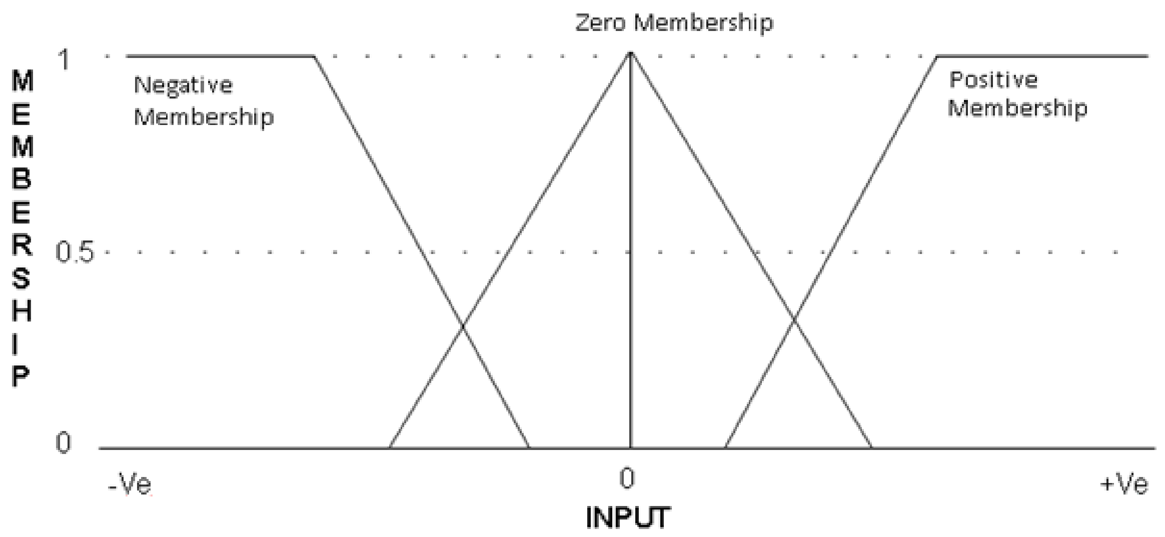


Fig 26 Example of fuzzy membership functions

Another important feature to consider with the fuzzification process is the shape of the membership functions. The most common membership functions are triangular, as shown in Fig 26, trapezoidal and Gaussian (bell-shaped). In traditional fuzzy systems, the triangular and trapezoidal membership functions are considerably more popular than Gaussian shapes. This is due to functional simplicity, making them easy to implement, which is particularly attractive for simple control solutions. However, as systems have become more complex, coupled with the adaptation of neural features into fuzzy architectures, the Gaussian function has become increasingly popular. This popularity stems from the fact that the straight edges in the membership functions of the triangular and trapezoidal types mean smooth switching is more difficult to accomplish than in Gaussian functions. There is no established rule for choosing the membership function which is best suited to the system but there is no advantage in using an unnecessarily complex function.

Once a set of inputs has been fuzzified, the next stage in the process is to apply the rule base which acts as the decision-making portion of the controller. The main building block of these is the IF – THEN function, which behaves exactly as the name suggests. If the previous example of ‘tall’ membership is taken, an example of a simple IF-THEN rule would be IF a person is tall THEN they are heavy. In a practical system, the number of inputs to the system must be at least two, meaning that the rule would take the general form IF A AND B THEN C. In order to implement IF-THEN functions with more than one input, it is necessary for the classical logic operators – NOT, AND and OR – to be mapped into an equivalent fuzzy logic operator in the following way:

- AND becomes minimum, eg A AND B is equivalent to min(μ_A,μ_B)
- OR becomes maximum, eg A OR B is equivalent to max(μ_A,μ_B)
- NOT becomes one minus the value, e.g. NOT A is equivalent to 1- μ_A

The rules themselves are then interpreted through a method known as implication. There are many different types of implication in fuzzy logic but the two most commonly used in FLC systems are known as Mamdani and Takagi-Sugeno-Kang (often shortened to Sugeno or TSK). The Mamdani implication method is the most commonly used of the two and there are two stages to this. Firstly, the inputs to the rule are compared and the min of these is taken, giving the firing strength for a particular rule. In order to determine the shape of the fuzzy output over its entire range, the maximum value of all the weighted rules is then taken. This output distribution must then be converted back into a crisp value using a process known as defuzzification. There are numerous different methods for performing this defuzzification but the most common methods are the “centroid of area” and the “weighted average” methods, both of which fundamentally take the crisp output value to the average of the membership values.

In order to demonstrate the operation of a typical Mandani system, consider an example with two inputs (height and weight) with this information being used to determine the likelihood of a person suffering from type-2 diabetes. An example of a fuzzy rule would be IF a person IS average height AND average weight THEN there is a moderate risk of getting diabetes. If a person has a degree of membership of 0.6 in the medium height group and 0.4 in the average weight group then the firing strength would be 0.4 for the moderate risk output membership. This shows how the AND function is mapped to the rule min [μ_A(x),μ_B (y)] in order to determine the rule strength in a Mamdani inference system. Assuming that there is another implication rule which has some degree of membership, the shape of the output is determined using the rule max[(μ_A,μ_B)] as is demonstrated in Fig 27.

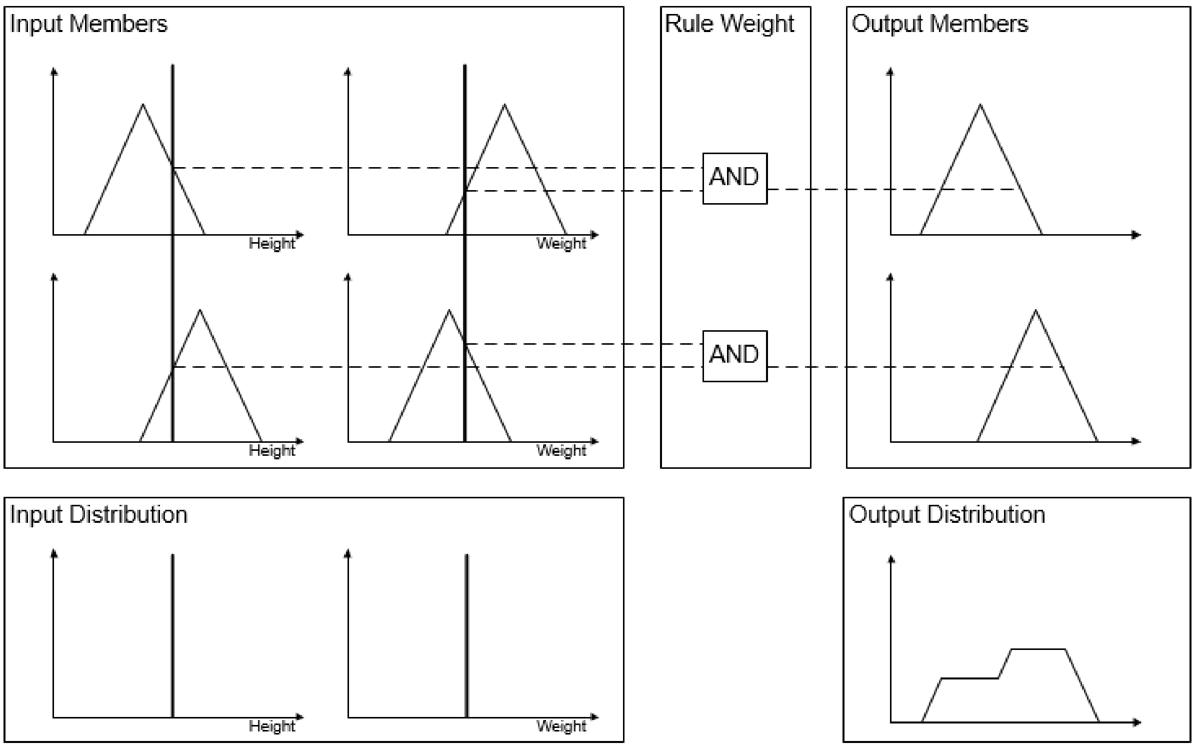


Fig 27 Operation of a Mamdani Based Inference System

Although traditionally not as popular as the Mamdani method, the TSK method is still widely used. This is largely owing to the fact that no defuzzification is required, making the algorithm more computationally efficient and also giving a smoother output surface. Whilst tuning was traditionally a problem for TSK implication, several system modelling tools, such as Matlab, now offer easy automatic optimisation algorithms. Unlike the Mamdani method, the TSK algorithm uses a linear algorithm to determine the values of the output members. In a system with two inputs, given as x and y, this algorithm can be expressed as $z = ax + by + \mathrm{k}$. In its simplest form, the TSK algorithm has both a and b set to zero, becoming a zero-order system. This means that the output is simply some constant k and the performance is very similar to Mamdani type systems. When the system is not zero order, then the performance relies on a and b, meaning careful tuning of these variables is needed and is one of the reasons for the additional complexity of the TSK method. In a TSK system, the membership values of each output function must be calculated and the output of the system becomes the weighted average of these functions. As the weighted average of membership values is the output of the system, it is clear that the output here is crisp rather than fuzzy and this is why there is no defuzzification required.

The use of FLC is both well established and popular for good reason. This type of controller is simple to implement for most applications, can incorporate human intuition and doesn’t require a complex model of the plant. The lack of a mathematical model is one of the major advantages of the FLC when applied to complex non-linear systems as the mathematical model tends to become more difficult to implement accurately as the complexity increases. This means that systems which rely on a model have inherent errors whereas a fuzzy logic system can avoid such issues. However, whilst the controller is generally easy to implement, as the system becomes more complex so does the tuning of the controller. Expert input is often needed for the most complex applications. Additionally, this type of controller differs from many non-linear controllers in that it lacks any learning ability. This further complicates the tuning and design when compared to the more simplistic learning method of ANN-based control.

3.1.3.2 Neural Networks

Artificial neural networks have seen a rise in popularity as fast processing devices have become more readily available and affordable. There are a number of applications for neural networks, such as image processing and voice recognition, but they have also proven to be useful tools when applied to the control of complex systems. Neural networks are so called because they emulate the behaviour of the neurons found in the human nervous system and the artificial neuron forms the backbone of neural network based controllers. The basic structure of an artificial neuron is shown in Fig 28.

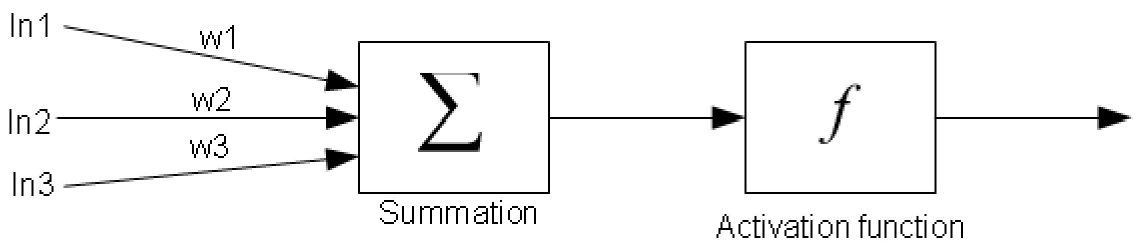


Fig 28 McCulloch and Pitts Neuron model

The basic structure of the neuron is not overly complex. There are several inputs to each neuron and these are weighted by their importance and then summed together. The output of the summation block is passed to the activation function, which decides upon the output value of the neuron and so has a big effect on its performance. There are several types of activation function but the threshold logic unit (TLU), log-sigmoid and Gaussian functions are popular examples. The shape of all three functions is given in Fig 29 and these all share the common goal of crunching the data into a new range (such as -1 to 1 or 0 to1). The activation function will change its output (typically to 1) when its value exceeds a certain threshold. Using the simple case of the TLU give in in Fig 29, the output of this function would change to 1 when the value of the input is greater than 0. In addition to the basic activation function, a bias can also be added to move the function to the left or right within the input space. Considering the TLU example again, adding a bias of two would move the activation function give in Fig 29 to the right by a value of 2. This simply means that in order to trigger the output, the input would require a value of 2 rather than 0.

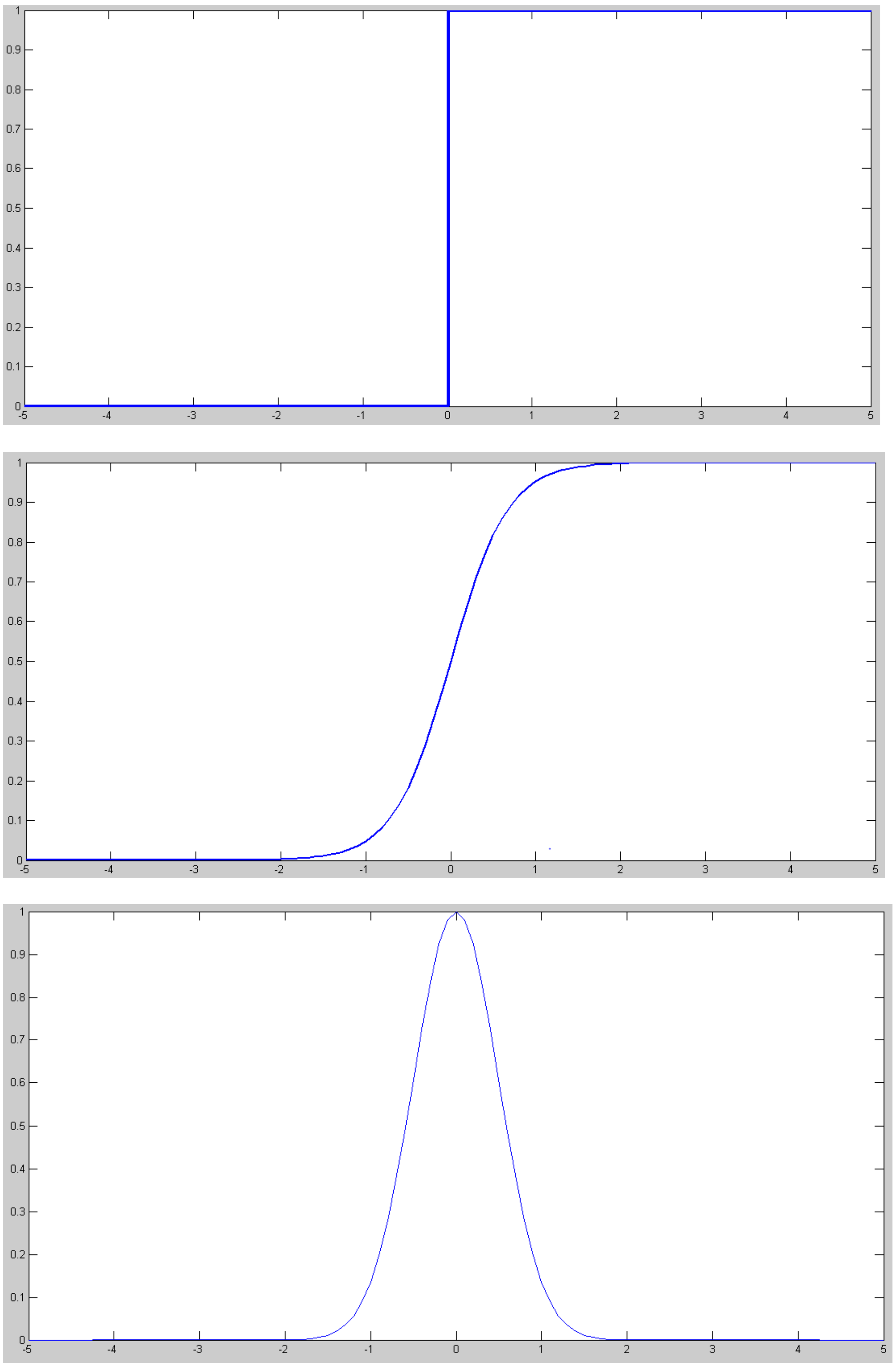


Fig 29 Typical Neural Network Activation Functions – TLU (top), Log-Sigmoid (middle) and Gaussian (bottom)

Another key feature of ANN is the type of structure which they employ. There are two main neural network structures, feedforward and recurrent, the basic architectures of which are shown in Fig 30 and Fig 31. Feedforward networks don't feature any form of feedback, meaning that the inter-neuron communication is limited. This is in contrast to recurrent networks, which are capable of communicating with one another through inherent feedback mechanisms. This feature of the recurrent architecture makes them good at recognising patterns and they are primarily used for this application. Feedforward networks are easier to implement than the recurrent network type and, because of this, they tend to be the architecture of choice for control solutions.

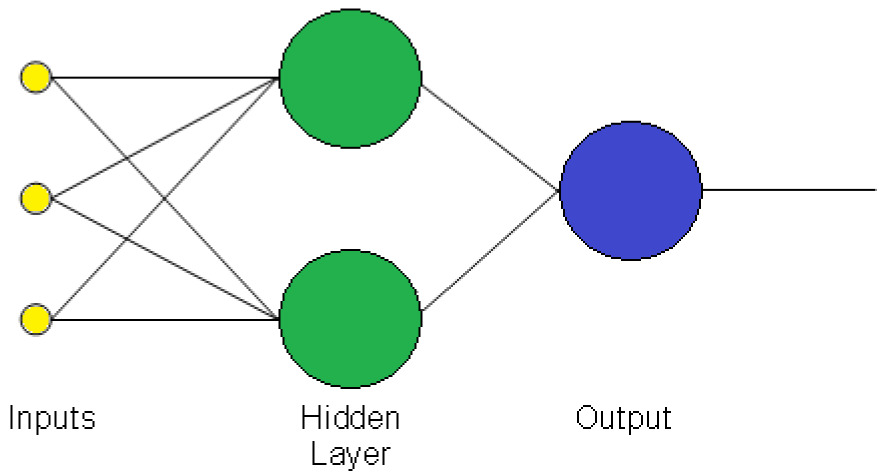


Fig 30 Feed-Forward Neural Network Architecture

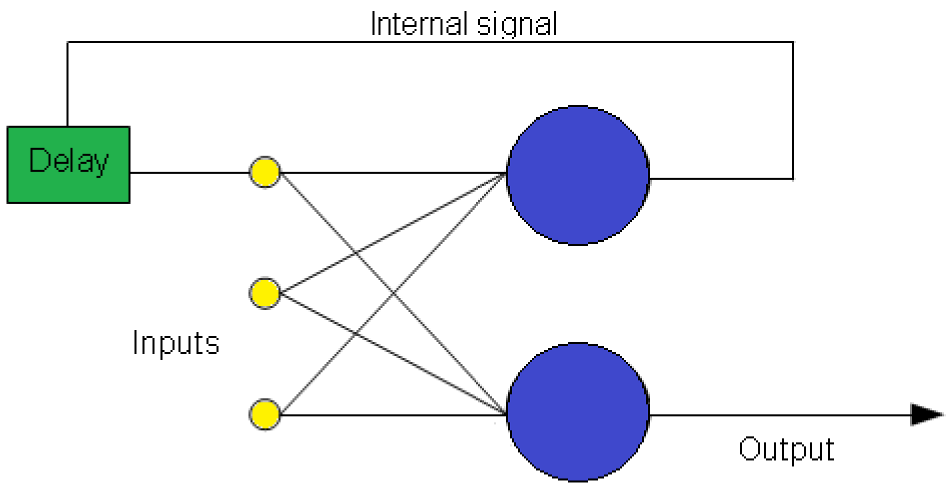


Fig 31 Recurrent Neural Network Architecture

One of the major features of ANNs is the ability to learn, or to be trained, and therefore improve their functionality. There are numerous methods which are used for training ANNs but they can broadly be classified as either constructive or non-constructive. In the constructive method all features of the network, including its architecture, are modified. On the other hand, non-constructive training adapts only the neurons weighting. In control applications, constructive algorithms are rarely used and the most popular non-constructive algorithm is the backpropagation algorithm, which is applied to feedforward networks. In order to train the ANN, a set of training data must be provided. This may consist purely of a set of inputs or can include inputs and the expected outputs. The nature of the training set dictates the nature of the learning algorithm. Data which consists of input-output pairs is known as supervised, whilst data sets consisting of only inputs are known as non-supervised. In control applications, the non-supervised method, which is commonly used for pattern recognition, is not popular. The supervised method is widely applied in control applications, as it gives more accurate results. The popular back propagation method is a supervised algorithm, which aims to minimise the sum squared error by adjusting the parameters of the neurons. This algorithm continues the process until the network has an error which is deemed negligible.

When applying a neural network structure to a control system, there are a number of different approaches which can be taken. Perhaps the most popular way in which ANNs are applied to control systems is through the modelling of the plant. This approach is particularly popular for non-linear systems where the plant model is both extremely complex and difficult to correctly ascertain. This procedure involves looking at the error between the plant and the existing model and using this to train the neural network until the error is minimized. The obtained neural based model can then be used as an integrated part of the controller and this method provides a good solution to the non-linear control problem. There are, however, some disadvantages of ANNs, particularly when compared to fuzzy logic systems. The most obvious limitation of ANNs is the fact that they don't feature explicit rules which can be interpreted by a human user. This is a major strength of fuzzy systems and obviously a very useful feature for a control system to have.

3.1.3.3 Neuro-Fuzzy Control

Whilst FLC and ANN are both powerful control tools, they have some major disadvantages. In the case of FLC, this is the inability of the system to learn whilst for the ANN the underlying rules are not known by the designer. A growing trend in controller research is the use of Neuro-Fuzzy (NF) systems which combine the features of both ANN and FLC, theoretically making use of the strengths of both types of system (Wu et al(2011)). A typical methodology in this approach is the use of a neural network arranged as a Fuzzy controller. The most popular architecture for NF systems is the Adaptive Neural Fuzzy Inference System (ANFIS). Much like fuzzy systems, this architecture is either classified as Mamdani or TSK. As it tends to exhibit greater levels of accuracy, the TSK method, as is shown in Fig 32, has gained greater popularity than the Mamdani method.

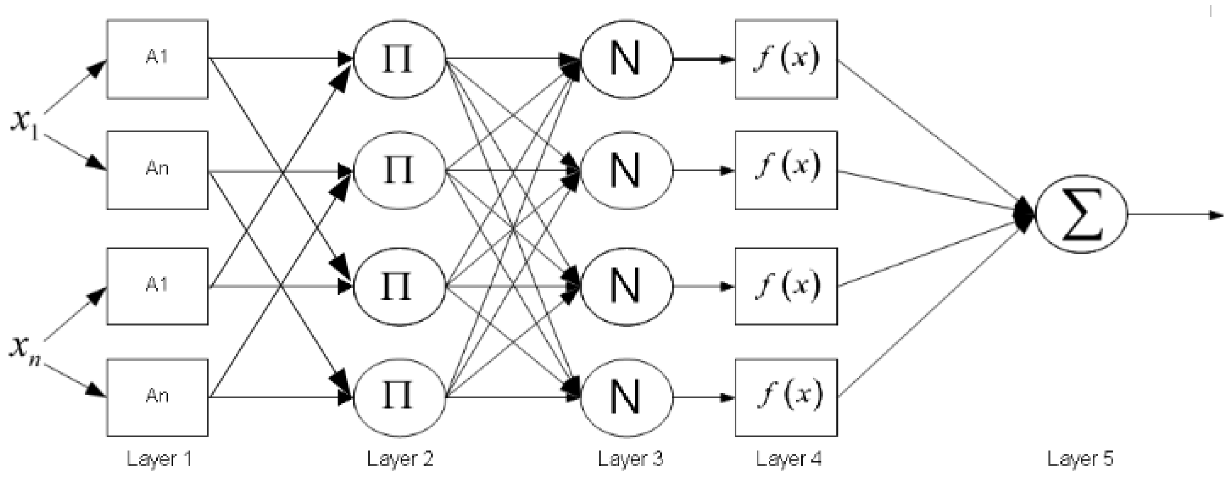


Fig 32 ANFIS Structure

In the ANFIS structure, layer one acts as the system fuzzifier, converting the crisp inputs into fuzzy sets as in the FLC type of system. An operational characteristic of the TSK ANFIS architecture is that there must be at least two inputs for the inference system to work correctly. This means that an additional input is required and is usually defined as the difference between the current error and the previous error. Layers two and three in the architecture perform simple math operations, layer two is a weighted multiplication of the inputs and layer three performs normalization so that the outputs are all in the range of 0 to 1. Layer four in the architecture is dependent on the specifics of the systems and plays an important role in the characterisation of the system's behaviour. In the TSK architecture, this layer takes the form of a polynomial equation of type ax+by+z and the order of this polynomial is used to define the type of system, with a zero order polynomial being a type zero and a first-order polynomial giving a type one system. Finally, layer five is a weighted sum of all the outputs from layer four.

The ANFIS structure, like ANN, is trained to tune the performance of the system. The methodology for training the ANFIS network is much the same as the methods used for ANN. A set of training data, consisting of inputs and the expected outputs, is presented to the network and the weightings are tuned until the sum squared error becomes negligible. Although this is a complicated procedure, the popularity of the ANFIS structure means there are several software packages, most notably MATLAB, which offer quick and easy solutions for the tuning of ANFIS networks.

3.1.3.4 Evolutionary Algorithms

Evolutionary algorithms are a class of algorithms which mimic some form of natural evolution to help solve a problem, often being employed to solve complex optimisation problems. The algorithms evolve a population of potential solutions to reduce the weaker fits and increase the number of promising potential solutions. The most famous evolutionary algorithm is the genetic algorithm (GA), which was developed during the 1960s and uses the principles of mutation of potential solutions. This approach is based on the principle of competition, as observed in genetics, with the stronger solutions eliminating the weaker ones. The other major class of evolutionary algorithm is based on swarm intelligence. The most well known of this classification of algorithms is the particle swarm optimisation (PSO) one, which was developed by Eberhart and Kennedy(1995). The PSO algorithm is based on flocking behaviour, such as seen in migrating birds, with all elements in a population being given freedom to search the solution space in a controlled manner. Unlike the GA, PSO is based on the principle of co-operation, with all the elements working together to track a solution. The flow and basic principles of the GA and PSO are explained further in this section.

The main flow of the GA is shown in Fig 33 and consists of four main stages – initialisation, calculation of fitness, selection of the strongest members and recombination of the strongest members into a new population. The first stage of the algorithm is self-explanatory, as the population is filled with a selection of random solutions. The next stage is to calculate the fitness of each member by evaluating how well each member of the population solves the problem. The degree of suitability to solving the problem is known as fitness, with members who give a good solution being said to have a better fitness than members which provide a poor solution to the problem. There are a number of popular algorithms utilised for the calculation of this value, such as means squared error (MSE) and integral squared error (ISE).

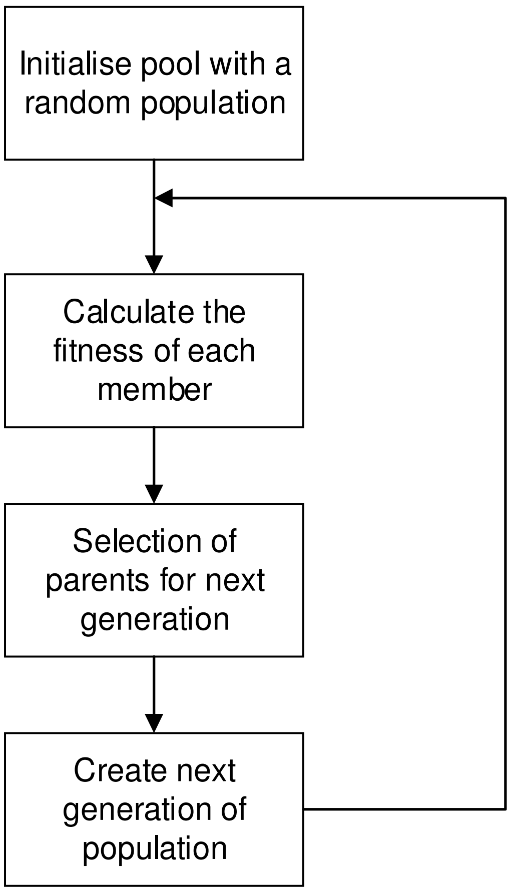

Initialise pool with a random population
Calculate the fitness of each member
Selection of parents for next generation
Create next generation of population

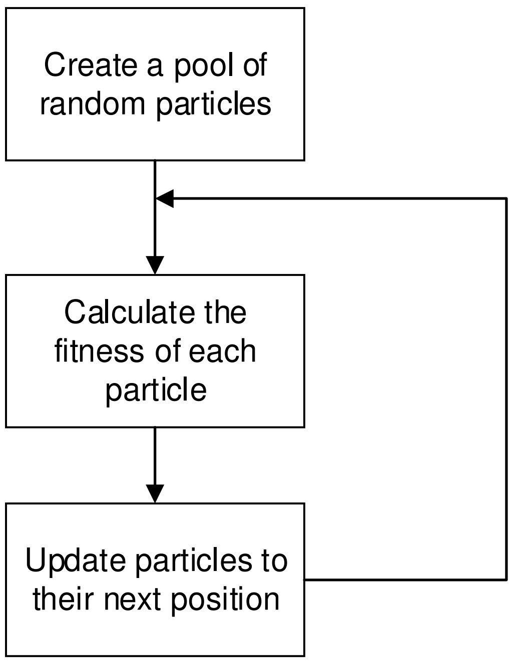
Fig 33 Flow Diagram of the Genetic Algorithm

The process of selection and recombination of members is at the heart of the genetic algorithm and forms the most complex part of the algorithm. Once the fitness has been evaluated, the most promising solutions are given a chance to propagate into the next generation. There are two commonly used methods for selecting which members of the population get a chance to become parents in the next generation – elitism and roulette selection. The simplest method to use is elitism, which takes the top 50% of the population according to fitness and allows the other members to die off. Although this solution is very simple to implement, one drawback can be a loss of diversity in the population, reducing the amount of solution space which is explored. Roulette selection is a probabilistic approach, whereby members which have a greater fitness have a greater chance of moving to the next generation. As an example, if there are three members in a generation, with one having a higher fitness than the others, then the member with a high fitness will be given a 50% chance, whereas the others will only have a 25% chance. This method ensures a degree of randomness remains in the population, which also promotes greater diversity. However, this approach is more complex to implement.

Once the parents have been selected, they are then used to create new members for the next population. In order to maintain and promote diversity within the population, it is necessary to create new unique members from the parents. There are three methods which are used for this - mutation, inversion and crossover. Mutation and inversion are very similar to one another and involve inverting bits of the parent. In the case of mutation, this means inverting single bits, whilst in the case of inversion sections of the bit string are inverted. The crossover process involves combining two members and this can be achieved by taking a number of bits from one and combining it with a number of bits from the other. Crossover can occur at one instance, such as the first four bits of parent A and the second four bits of parent B, or can occur at multiple points. It is common to move the member with the highest fit straight into the next population and then perform the recombination process on the remaining members to create a new population.

One other nuance of the GA structure is the termination of the algorithm, which is not shown in the general flow as presented in Fig 33. The most common way to terminate the algorithm is to exit the main loop when the solution fits with the designers' criteria. As an example, the designer may specify a required fitness of at least 99.9% and so the algorithm would terminate when this condition is met. However, there can also be instances when suboptimal solutions can be provided due to a lack of diversity in the population. In order to overcome this limitation, it is also often necessary to create a termination criterion which forces the algorithm to exit when there is a lack of diversity in the population.

The other major class of evolutionary algorithm is focused on swarm intelligence, with the most popular example of this being the PSO. The flow of this algorithm is shown in Fig 34. The PSO is simpler than the GA and there are only three stages in the design flow: initialisation of the particles, calculation of fitness and updating of the particles. Whilst the population of each generation for the GA is simply comprised of potential solutions, for the PSO the term particle is more encompassing. Each particle consists of a position vector, which is the current value of the solution to be tested, as well as a velocity vector and a personal best fit vector. The velocity vector is used to update the position of the particle and is discussed in more detail later in this section. The personal best fit vector stores the position coordinates which gave the best fit for a given particle as well as the value of fitness at this point. In addition to the personal best, there is also a value for the global best. This represents the best position found in any particle and both of these values are used to update the position of the particles. The first and second stages of the PSO algorithm flow are the same as detailed in the GA flow, with particle initialisation being the first phase and calculation of fitness the second stage.


Create a pool of random particles
Calculate the fitness of each particle
Update particles to their next position
Fig 34 Flow of the PSO

Once the fitness of each particle has been computed, it is necessary to create a new set of particles to evaluate. The first stage of this process is to update the position vector by an amount which is determined by the velocity vector. The second stage of the process is to update the value of the velocity vector and is more complex to achieve. This vector is produced by taking the current velocity and adding the difference between the current fit and the global and personal best fit. Both of these values are weighted with a random number to help promote diversity. The full algorithm used to update the velocity is shown in Eq. 6 , with GX_n being global best fit, PX_n being personal best fit, V_n being the current velocity and X_n being the current position. The two C coefficients are acceleration constants which dictate how much the personal best and global best influence the next velocity.

$$V_{N+1} = V_N + (C_1.Rand.(GX_N - X_N)) + (C_2.Rand.(PX_N - X_N)) \qquad \text{Eq. 6}$$

As with the GA flow, there is also a need for termination criteria to prevent the algorithm from looping infinitely. This task is commonly achieved using one of two methods - either the fitness of the algorithm meets the designers' constraints or a maximum number of iterations is exceeded. If the maximum number of iterations is exceeded, this is often indicative of there being a lack of diversity in the particles, meaning that some local maximum has instead been focused on rather than a global maximum. In this instance, running the algorithm again may find a better solution.

3.2 AI-Based Controllers for Integrated Renewable Systems

A significant amount of research has been conducted into the control of non-linear systems. This has led to the rise of intelligent controllers as a promising method. Intelligent controllers can be classified as Fuzzy logic based, Neural Network based, Fuzzy-Neuro hybrid and Evolutionary Algorithm based. In this section, all of these controllers are reviewed and their application to power converters and hybrid renewable energy is considered.

3.2.1 Fuzzy Control

The use of fuzzy controllers is well established in a variety of different applications and the application to renewable energy systems is also well researched. A popular application for fuzzy controls is load management (Zerkaoui (2012), Althubaiti (2017) and Thounthong et al (2013)). A particularly good example of this is shown in the system presented by Thounthong. In this example, the sources are comprised of a PV cell, a fuel cell and a supercapacitor. The FLC in this design prioritises how the sources supply the load and to charge the storage elements. In this system, the PV is prioritised for steady state operation whilst the supercapacitor is used for dynamic states, with the fuel cell supporting both of these sources. The simulations provided in this paper show the smooth, high-quality DC supply which is achieved as a result of the optimal control of the selected system and also shows how the decision-making capabilities of FLC can be exploited. Although the system achieves a steady DC bus, there are still some noticeable fluctuations in the bus during transient periods showing there is still room for improvement within this system.

An important consideration of DER systems is maximizing the power from the input sources in order to ensure the best possible efficiency. In PV cells, maximum power point tracking (MPPT) is a widely researched theme and FLC, whilst not as popular as other techniques, does have some uses in this area. The paper presented by Alajmi et al (2013) is a prime example of an MPPT for a PV system which uses a fuzzy controller to extract maximum energy from the system. By tuning the system using a traditional algorithm and then teaching the tuned parameters to the controller, this system is able to extract maximum power even when the array is partially shaded. A number of similar systems exist in literature, such as those presented by Dounis et al (2015) and El Khateb et al (2014). These papers outline a similar approach using an FLC to provide MPPT algorithms, illustrating the popularity of this method. Neural networks are a more commonly used technique for MPPT but this method shows how the FLC can be used to enhance performance in difficult conditions where decision-making abilities are a major advantage.

One paper which shows how FLC can be used to maximise power output is presented by Nasser et al (2009), whereby a hydropower plant and a wind turbine are controlled to give optimal complementary power output. The simulations run without any controller on the system show how the power output is fluctuating excessively under normal conditions. The introduction of an FLC helps to ensure a smoother, more predictable power flow over a longer period, which is essential for ensuring power stability. The complexity of the system is reflected in the controller design, with several smaller controllers being used for different time periods. Whilst it is expected that an increase in system complexity will increase the controller complexity, this highlights a major weakness of the FLC as a great deal of data will need to be analysed by an expert before a controller of this nature can be implemented.

In the control of DC-DC converters, simple linear PI(D) controllers are a well-established technique. However, the inherent non-linearity of switch mode converters means this is a sub-optimal solution. Fuzzy logic controllers provide a non-linear control solution and the use of these controllers in conjunction with PI(D) controllers has emerged as a strong research theme. Sahin and Okumus (2011) compare the performance of a solar panel connected DC-DC converter showing the difference in performance between the PI and the FLC approaches. In this paper, it is demonstrable that the performance of the FLC on its own is quite poor, with a noticeable steady-state error. Similarly the PI controller on its own exhibits poor characteristics with a slow response speed and a noisy output. The concept of a combined controller is then introduced and this shows that using the two control strategies together results in improved performance. In the examples presented by Rabbani et al (2012), Sarvi and Namazy Pour (2008) and Chang et al (2016), a more interesting approach is taken as the FLC is used to adapt the gains associated with the PID. As fluctuating loads equate to fluctuating levels of damping, this method has a great deal of potential as different parameters will be required for different loads. This approach gives an improved system performance compared to either the PI or FLC but there is still room for improvement under dynamic conditions. The paper presented by Elshaer at al (2011) improves on the performance of the adaptive controllers already discussed by exploiting the dynamic behaviour of the derivative portion of the PID controller. As the derivative is only useful for improving dynamic performance, the output current is monitored and the derivative is only applied when there is a sufficient change in load. By making the time constant of the output filter slower than the controller, it is possible for the derivative to act before the capacitor is discharged excessively. The resultant performance of this controller is excellent, with there being virtually no disturbances in the output due to load fluctuations. The system presented by Abbas et al (2011) shows another expanded method for the Fuzzy-PID controller by integrating a pole-placement controller. In this method, the Fuzzy-PID is used to tune the positions of the system poles in order to achieve stability. Although this is perhaps an overly complicated method, the results of the system simulations show that the controller exhibits excellent characteristics, providing not only a stable steady state but also showing minimal fluctuations under rapidly changing loads.

3.2.2 Neural Networks

The learning ability of neural networks makes them a particularly popular choice for MPPT algorithms in PV systems (Baek et al (2010), Tsai at al (2012) and Sheraz and Abido (2012)). A good example of a typical type of MPPT algorithm is presented in the system developed by Elobaid et al (2012). This system uses the ANN to produce a model of the plant using voltage and current measurements. This information is then utilised to implement an MPPT algorithm which is demonstrably superior to the widely used perturb and observe (PO) method. This system proves to have excellent characteristics in both steady state and dynamic operating conditions. Additionally, the need for temperature and irradiance measurements is eliminated by the use of ANN and this is a good example of an application in which the use of ANN is highly optimal. Another common approach for MPPT algorithms is the use of an ANN as a function approximator which replaces a traditional algorithm, such as in the method presented by Xu et al (2011). In this paper, the neural network is trained to behave like the well known incremental conductance method, with the experimental results showing that there is an improved dynamic performance when employing this method over traditional algorithms. ANN-based controllers can also be used for extracting maximum power from wind systems and there are a number of methods presented (Lin et al (2011), Cirrincione et al (2013) and Ren and Bao (2010)), all of which feature similar techniques. One good example of a wind turbine based MPPT is shown by Lin and Hong (2011). This methodology uses a novel recurrent ANN structure, known as an Elmann type ANN, to control the pitch angle of the blades in order to extract maximum power. A genetic algorithm is firstly used to calculate the pitch angle for different wind conditions, with this data being used as training data for the ANN. The comparison is drawn with a traditional PI controller and a fuzzy logic controller and shows the favourable performance compared to both of these techniques. However, there is a great deal of extra complexity, not only in the implementation but also in the design, for a relatively small gain in performance compared to the FLC.

The use of ANN in the control of power converters is not as widely researched or popular as the use of fuzzy logic. One system which uses the ANN as a standalone control solution is shown by Utomo et al (2011). In this paper, the controller takes the form of a pure ANN which is used to derive the duty cycle of the PWM signal in a buck-boost converter. The training data for this is taken directly from the plant in an online environment and the structure of the network is basic. This controller exhibits better dynamic performance than a PI controller, with which it is compared, and is not overly complex to implement. This controller is still far from optimal, however, with some oscillations being present in the transient response and not a great deal of speed improvement compared to the PI controller.

In the control of power converters, it is a more popular and promising solution to create an ANN-based system model which acts as a reference in the controller. Examples of such an approach are given by Kurokawa et al (2012), Abbas et al (2011) and Zhu et al (2015). In the paper presented by Abbas et al (2011), the controller takes the form of a model predictive controller. This is a common adaptive controller architecture and requires the controller to have an integrated mathematical model of the plant. The ANN in this system is trained in an online environment to behave like the plant and this controller is proven to exhibit better system characteristics than a purely mathematical model. Despite the improved performance offered by the neural network, there is still no protection against inaccuracies in the model due to interferences such as thermal drift. The papers presented by Kurokawa et al (2012) and Zhu et al (2015) again use an ANN to obtain the model of the system which can be used as the predictive element of the controller. In these cases, the model is used to improve the performance of a PID controller. Again, this controller uses online training and the data acquired is based on real data which is obtained during an output disturbance. In this example, the training of the controller is continuous. This helps to eliminate any model inaccuracies which can occur when training is performed under only one operating condition. Using the adaptive PID architecture does result in an improvement of the controller response and the continued training shows continuous improvements in the transient response as a result of the learning capability of the ANN. However, this approach does have one major drawback as this is a very complex set of algorithms and the controller can be viewed as being inefficient compared to the use of an FLC which yields similar results. The use of an intelligent controller to adaptively tune the PID is a popular approach, as was outlined in the previous section discussing FLC. The use of an ANN for this is more difficult to implement and yields no particular benefits, so the FLC is more suited to this task. However, the use of ANN does make the tuning of the system a simpler task and the online training method is highly desirable. The use of Neuro-Fuzzy controllers, discussed in the next section, combines the benefits of both approaches.

3.2.3 Neuro-Fuzzy Control

The use of NF controllers in power systems is a newer concept which has not been as widely employed as either ANN or fuzzy systems. Despite this, there is some promising work being done on the application of NF in the domain of power electronics and renewable energy. One popular area of research for NF controllers is in the use of MPPT algorithms, as their inherent learning ability makes these controllers ideal for this solution. One such system is presented by Afghoul and Krim (2012), where the MPPT is used to achieve the maximum power from a PV cell with the controller being based on the ANFIS architecture. The system presented in this paper is compared to the popular Perturb and Observe methodology and demonstrates reduced steady-state error and faster dynamic response. The comparison is also drawn with the implementation of an FLC and the ANFIS implementation exhibits some features which make it a superior choice. Using ANFIS makes the controller more portable and the design method also means expert knowledge is not required to achieve optimal control. It would also be beneficial to have seen a comparison made with an ANN-based controller as this control solution doesn't seem to improve greatly on the results seen in other works which use this approach. There are several similar papers on the use of ANFIS based MPPT algorithms (Haibin and Jinging (2010), Iqbal et al (2010), Rouzbehi et al (2012), Khosrojerdi et al (2016) and Chikh and Chandra (2015)), all of which present similar techniques. This is perhaps the most well-researched area of applications for NF control in renewable energy systems.

The use of NF in power converters, particularly DC/DC converters, is not well established as a research theme but there are some papers which have started to explore this possibility. One such paper is presented by Hajizadeh and Golkar (2009) and looks at the use of NF control in a hybrid energy system. This paper uses an interesting approach as the power flow between the system components is managed by an FLC, such as in previous papers, whilst the control of the individual converters is performed by the NF controller. This approach makes sense, as the power flow is relatively simple and a fuzzy system is more than capable of meeting this demand, whilst the optimal control of the power converters is a more complicated task. Although this paper presents some simulations, they merely highlight the efficiency of the power flow, a testament to the FLC, and don't focus on the performance of the NF control. Although this detail is lacking, the paper still presents a novel control approach; with the use of FLC for power flow being well established and proven, it makes sense to use this approach whilst using an NF for the more complex parts of the system. Using this combination of approaches represents an efficient way of producing an optimal control solution.

Another interesting concept is presented by Kumarawadu et al (2006), whereby a buck converter is controlled using firstly a PI, then an FLC and then finally the FLC membership functions are tuned using an ANFIS type training tool. Despite not being a full ANFIS controller, the system still exhibits improved dynamic performance compared to the PI and FLC systems, with reductions in settling time and overshoot. The use of PI/PID controllers is well established for DC/DC power converters, so it is perhaps surprising that there has not been much research into the use of ANFIS-PID controllers in the same way that Fuzzy-PID systems have been researched. Tuning of PID controllers for power converters with wide current operating parameters is especially difficult, as the load creates a variable level of damping in the system. This means that a PID which is tuned for a mid-range operating current will not be optimised for low or high levels of current and can even result in system instabilities. This gives the ANFIS-PID architecture an advantage, as the tuned parameters of the PID become adaptive, meaning that the PID can be optimised over a wide range of damping coefficients. These characteristics make this control strategy an interesting option for power converters which require a wide operating range.

3.2.4 Evolutionary Algorithms in Control

The most popular and widely used of the evolutionary algorithms are the GA and the PSO, which were previously discussed in detail. Despite these algorithms being popular and widely used for many applications, in the context of control systems, the evolutionary algorithm is fairly limited in use. One popular use of both types of algorithm is parameter tuning, making them an ideal tool for use in the optimisation of PI(D) controllers. Therefore, it is little surprise that this is their main application area within the context of control systems. The paper presented by Yousefi et al (2008) highlights an implementation example where both of the algorithms are considered. In this paper, the manually tuned PID controller is compared to controllers which have been tuned by both the GA and the PSO algorithm. In both instances, the performance of the controller is improved, with the settling time being reduced and an increase in the system stability being observed. Although the GA and PSO algorithms give a similar system response, the PSO implementation is perhaps more widely adopted (Femmy Nirmal and Jeraldin Auxillia (2013), Sharaf and El-Gammal(2010), Li et al (2013), Wang et al (2014) and Ghosh et al (2015)), due to the fact that it is much simpler to implement.

4 Modelling and Hardware Tools

The traditional electronics design flow consisted of creating a theoretical design and building hardware to test its validity, an approach which is both time consuming and expensive. As affordable, powerful computing devices became more readily available, this was quickly replaced with the modern day approach of modelling the system to give confidence before the design is committed to hardware. Whilst the creation of complex mathematical models isn't a trivial task, the use of modelling tools has resulted in reduced development time. As this approach has become more popular, several standard models have also become available. Common circuit components are now almost universally available, whilst models of other popular components are also readily available in various tools. A further by-product of the increased popularity of the modelling approach is a greater influx of tools hitting the market, offering a range of advantages over one another. Whilst this inevitably means that there is a software package which is ideal for any system, the array of tools available means that careful consideration is required to ensure the best modelling environment is chosen. In this chapter, some modelling tools which would be suitable for this project are presented and discussed before the proposed approach for this project is introduced.

4.1 Review of Modelling Tools

4.1.1 Matlab

One of the most commonly used modelling tools for engineering systems is the Matlab environment. Originally designed as a software tool for solving equations, Matlab has grown to include a wide array of features which make it a highly flexible tool. At the heart of Matlab is a simple, yet powerful, mathematical syntax, which uses the same basic engines as the original versions. The simplicity of developing highly complex algorithms is one of the key strengths of Matlab and perhaps the biggest reason for its continued popularity. In its modern guise, Matlab also offers a number of plug-ins, such as Simulink, which help to expand the range of applications which it can be used in. A number of these plug-ins not only increase the range of applications but also make it possible to rapidly create components to use in system models. The highly flexible nature of Matlab makes it an excellent tool for system-level modelling and it has subsequently found use in a wide range of different application areas.

As expected from such a popular tool, Matlab makes an excellent choice for system-level modelling. The simple yet powerful syntax of Matlab is ideal for the modelling of renewable sources (Lakshmi et al (2013), Biju and Ramchand (2013) and Bayindir et al (2013)). As Matlab offers co-simulation between a number of models and even different modelling languages, it is an ideal candidate for running system level simulations of engineering systems. Another important consideration with Matlab is the Fuzzy logic and Neural Network toolboxes, which allow for rapid creation of controllers or other components based on these techniques. These toolboxes make the otherwise highly complex task of designing, training and validating systems which feature Fuzzy logic or Neural Networks considerably simpler, by offering a set of generic tools which can quickly perform the associated algorithms for the user. The use of an additional plug-in, the Simpowersystems toolbox, allows for electronic components to be simply imported into a Simulink model. These features mean that Matlab offers an environment which can be used for rapidly creating full system-level models for renewable energy systems, such as those presented by Ramya et al (2013), Othman et al

(2015) and Suh et al (2016). This makes Matlab an extremely powerful modelling tool in the creation of renewable energy systems.

Despite these features, there are some drawbacks to the use of Matlab in comparison with other modelling tools. In this project, there are two notable disadvantages – the need to create custom models for the renewable energy sources and the lack of a simple SPICE simulator for electronics simulation. Both of these represent significant weaknesses of the tool, as they introduce a deal of uncertainty into the model. Whilst it is not overly difficult to create models of the renewable energy sources in Matlab, it doesn't offer a verified, generic solution in the way that other tools do. This means that extra development effort must be dedicated not only to create but also to verify these models. Although this is not an overly onerous process, this approach is obviously inferior to having predefined models ready to deploy. Another, less significant, drawback is the lack of simple SPICE support for electronic circuit simulation. Simulations based on this approach are very popular and widely used within industry, offering simplicity and reliability. The other advantage of SPICE simulations is that precise models of components can be created, allowing for a model which is closer to the final hardware target. The lack of such an engine is mitigated somewhat by the fact that Matlab offers a simulation environment within Simulink which features electronic circuit simulation, although this tool is less flexible.

4.1.2 SystemC and Handel-C

A recent development in the design of FPGA based hardware is the use of C like languages to design the embedded digital control solution. The most popular of these languages are Handel-C and SystemC, which both offer libraries building concurrency into the standard C language syntax. In doing this, they become capable of modelling the parallel features of the FPGA, which isn't otherwise supported by the sequential nature of the C programming language and its derivatives. Although these languages are targeting FPGA development, they are not inherently hardware description languages (HDL) such as VHDL and Verilog. These languages are instead intended to allow for the designer to exploit the parallel features of the FPGA, whilst using a high-level software development language, with the compiler being responsible for the generation of the detailed hardware implementation from the abstract source code. These two languages offer one significant advantage over the more popular HDL languages and that is the ability to easily integrate C and C++ code into the development process, such as in the DK design suite, where these languages are natively supported. The advantage of this is that C/C++ are both well suited for the development of algorithms and mathematical models, meaning that it is possible to create an abstract system model and the hardware model in one environment. This approach is also suited to a rapid development process, as there is no need to specifically create a hardware model once the high-level system-level model is complete, owing to the FPGA being developed simultaneously and/or generated through the compiler.

The two C languages are powerful modelling tools and a good choice for rapid prototyping solutions. However, there are some drawbacks to their usage which make them a less appealing tool compared to other approaches. When it comes to system-level modelling, one issue is that it becomes necessary to create a new model for every part of the system using either C or C++. Whilst this isn't always a complex issue, other tools are available which offer a number of predefined models which can be used, reducing the development time in both instances. Other tools also offer a more graphical approach, without compromising any of the mathematical processing power, which makes them a simpler to use alternative. So whilst SystemC and Handel-C are useful, and will no doubt become viable alternatives as they mature, at the moment the system level modelling capabilities are not as favourable as other tools. The other use of these C languages is in the design of FPGA hardware and, in this respect, the more traditional HDL languages are generally favourable. The main reason for this is that these languages have been inherently designed for hardware simulation, meaning that they are aimed specifically at gate-level logic. This is in contrast to the C type languages, which are effectively a bolt on to a software programming language, giving a higher level of abstraction. For this reason, there are syntax features in the VHDL and Verilog languages which can't be easily replicated in the C languages, even though these languages are better for abstract features such as algorithm development.

4.1.3 VHDL and Verilog

In the design of modern industrial controllers, one of the most popular hardware implementation solutions is the FPGA. These devices are readily available, highly customisable and offer true parallelism, making them a good fit for AI-based controllers which also feature parallel architectures. The most popular languages for creating FPGA designs are the two HDL languages – Verilog and VHDL. In both instances, the design flow for creating the hardware is the same. Firstly, a functional model is created and simulated, verifying that the code is correct. Once this stage is satisfactorily completed, the design can then be synthesized, transforming the functional code into an array of gates level macros. Carrying out post-synthesis simulation means that the specific hardware design is being simulated to verify that the final FPGA will behave as intended. Once the synthesis simulations have been satisfactorily completed, the design can then be fit to the specific device and programmed into the target FPGA. The next stage of the process is the post-layout simulations which are intended to mimic the array of gates which has been fitted exactly to the intended device. It is also becoming increasingly common for vendors to offer debugging solutions which allow for continued low-level data capture of the design once in the hardware target. These techniques typically make use of JTAG capabilities which are built into most modern devices and examples of this include the Hardware In the Loop (HIL) offerings from Xilinx for the Zynq family. The entire flow for an FPGA design is shown in Fig 35.

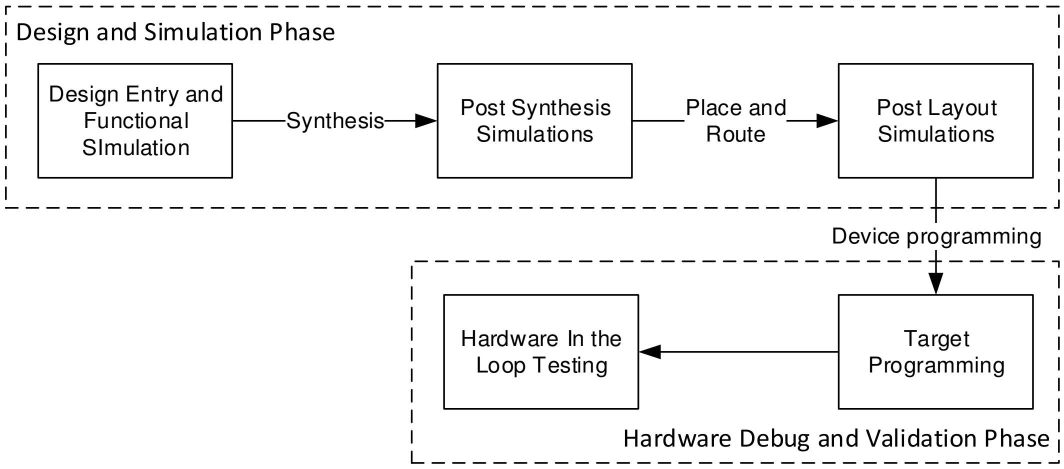

Design and Simulation Phase
Design Entry and Functional SImulation
Synthesis
Post Synthesis Simulations
Place and Route
Post Layout Simulations
Device programming
Hardware In the Loop Testing
Target Programming
Hardware Debug and Validation Phase

Fig 35 FPGA Design Flow

Both VHDL and Verilog were created during the 1980s, with VHDL being produced by the United States government and Verilog being privately developed. Both of these were developed using features of existing programming languages; in the case of VHDL the constructs are based around Ada whilst Verilog is based around a combination of Ada and C. As HDLs gained popularity, these languages have been standardised by the IEEE, with VHDL being covered by the 1076 standard and Verilog being covered by the 1364 standard. The intention of both HDL languages is to allow the user to model and simulate programmable logic devices. As such, they feature a number of similar features to one another, whilst also being inherently different from other programming languages. Perhaps the key defining feature of VHDL and Verilog is the inbuilt concurrent statements, which are essential to the modelling of hardware which they are describing. This is in contrast to traditional languages such as C, which only offers a sequential syntax and is a reason HDL is more suited for hardware design. Both VHDL and Verilog also offer a lower level of abstraction, featuring a syntax which is much more suited to Register Transfer Level (RTL) development than to abstract algorithm development. These features mark out these languages as superior options to tools such as SystemC for designing detailed FPGA systems, as they are more closely aligned to the hardware circuit being designed.

When discussing the two HDL languages, it is also important to consider the differences between them. Whilst these two languages are different from a syntax point of view, they do offer similar levels of performance and there are only subtle differences between the two. One of the key differences between them is the data types that they offer. VHDL is a heavily typed language, with numerous predefined types such as integers, std_logic and std_logic_vector which can be used by the designer. On top of this, it is also possible for the user to define their own types or subtypes, which can be enumerated or extensions of existing types. This is in contrast to the Verilog language, where there are very few types which can be used and it is only possible to create parameters rather types. In reality, the data which is being described is closer to the Verilog method, as it will be binary ones or zeroes in the hardware target, but the usage of types can help to create more readable and maintainable code. In comparison to Verilog, VHDL also has greater features for the support of code and module reuse. In Verilog, there is no functional equivalent of the VHDL ability to create packages and libraries which feature reusable functions or procedures. Similarly, VHDL offers the generate statement, which can be used to structurally create designs featuring a number of modules of one common design. As an example, in VHDL an array of RAM blocks is created from one description of RAM behaviour and then multiple instances can be called using the generate statement, a feature which isn’t equivalently supported in Verilog. Although the general logical constructs of the two languages are the same for gate level modelling, Verilog does offer some useful extensions to this such as the ability to easily create truth tables. Despite these differences, the performance of both languages is similar and, in most instances, the choice of which language is used for a design often comes down to the availability of suitable tools and the preference of the user.

4.1.4 VHDL-AMS and System Verilog

Whilst the previously discussed C like programming languages act as a means to bridge the gap between the system modelling environment and the hardware design, both of the HDL languages also have extensions which are intended to serve this same purpose. In the case of VHDL, this extension is known as VHDL-AMS which includes syntax for modelling analogue components. This extension was originally intended for the development of mixed signal ASIC designs, meaning it is primarily aimed at the modelling of discrete components rather than being a general modelling language. Despite this, the VHDL-AMS language features a strong syntax for the solution of differential equations, meaning it can be used for general system modelling. As with the Handel-C and SystemC languages, this has the obvious benefit of allowing for the final target and the system to be created simultaneously. This allows for rapid prototyping of the system and the fact that VHDL-AMS is supported by industry standard tools such as QuestaSim and Modelsim means that most FPGA engineers will not need to become familiar with an additional software tool. Despite these strengths, the VHDL-AMS language does have some rather significant drawbacks to its usage which has limited its appeal. The first issue with the language is that it is necessary to manually create models for everything which is used in the system. This process is time consuming compared to other tools which contain a number of predefined, generic models. This problem is worsened by the limitations of the VHDL-AMS language, as the syntax is not similar to other programming languages. Compared to the syntax used in the Matlab environment, for example, the VHDL-AMS syntax is difficult to understand and not as easy to write code for. The basis of the syntax is in discrete components, which means that anything which isn't modelled as a discrete component within a model can also be complex to effectively simulate.

The Verilog equivalent of the VHDL-AMS language is known as System Verilog and this is designed to act alongside the existing Verilog language and improve its functionality. The additional features of System Verilog are primarily aimed towards improved verification abilities but many of the features offered allow for some degree of system-level modelling. These enhancements include improved data types and processes, improvements to procedures and syntax, which is closer to software languages in order to create more realistic test benches. One other major feature of the language is the ability to easily interface with other programming languages through the use of a Direct Programming Interface. This is a highly usable feature for modelling of systems, as it means that components can be developed in another appropriate language independently and verified before being simulated together with the final hardware target. As a product, System Verilog has gained a lot of industry interest but this is because of the enhancements in verification ability rather than in the system modelling abilities. In order to achieve a complete model using this approach, it is still necessary to create models using another language such as C. As it has already been discussed, this is time-consuming and inefficient in comparison to the use of specialist tools which generally offer a much more graphical approach and feature a number of predefined models.

4.1.5 PSim

A more recent development in the field of renewable power system modelling is the PSim simulation environment. Like the other tools discussed here, this is not a piece of software that has been designed specifically for the development of hybrid renewable systems but is instead aimed at being a general power and control modelling tool. PSim is a more graphical tool which relies on schematic entry to create the system. There are numerous toolboxes which feature generic schematic symbols and models for a range of uses, most notably power electronic components, renewable sources, generators and blocks of C or VHDL code. It also allows for quick simulations allowing for the analysis of voltages and current in all parts of the system as well as allowing for spectrum measurements, allowing a way of measuring system harmonics.

There are a number of reasons why PSim is an excellent tool for the modelling of renewable energy systems. One big advantage that PSim offers is the ability to rapidly model renewable energy sources. Included in the renewable toolbox are basic models of a wind turbine, a solar panel and batteries which can be adapted easily to meet the specification of commercially available products. This is in comparison to Matlab/Simulink, SystemC, Handel-C or VHDL-AMS, which require the creation of a mathematical model in order to replicate their behaviour. Another feature of PSim is the close integration of the controller design to the power electronics. PSim allows for the integration of VHDL and C blocks so that they can be simulated in the same environment as the power electronics, giving the designer confidence that the code is functional. Additionally, this allows for the quick exporting of the controller code, as it can easily be transferred into the compiler of choice. PSim, therefore, offers a great range of flexibility and allows for the simulation and modelling of the full renewable system being designed.

Whilst PSim is a useful tool for modelling renewable systems, there are drawbacks to the software. The chief problem with this tool is that there are only two renewable sources – wind and solar – and the introduction of other renewables relies on the creation of Matlab models. Whilst these are the most commonly used renewables, it would be useful to be able to rapidly model other sources as well, especially fuel cells, which are often required for stabilizing the fluctuations from renewable energy supplies. However, the PSim environment allows for co-simulation with other tools, most notably Matlab. This means that the ability of Matlab to rapidly model any component which isn't included in PSim, can be exploited, and this lack of generic models is relatively easy to overcome.

4.2 Potential Hardware Solutions

Once the modelling of the system has been completed, it is important to ensure that the hardware provides an appropriate means of implementing the controller for the system. Four separate ways in which the controller could be implemented can be easily identified – in pure hardware using discrete components, using an FPGA, using a processor or using an ASIC. The approach of using discrete components is dated for many reasons - it is generally more expensive, more difficult to implement and debugging designs will require iterative hardware solutions which is a time-taking and expensive development practice. Therefore, the use of either a processor, ASIC or an FPGA will be considered for this project.

4.2.1 Microprocessors and DSPs

Microprocessors and microcontrollers are now a well-established method of designing electronics for a wide range of applications. Microprocessors and microcontrollers are an obvious candidate for many solutions as they are available in a range of different sizes, from a simple device with a handful of pins to multiple core devices with several hundred pins. As they have become increasingly popular, the affordability of such devices has also greatly improved. Microprocessors also typically feature an array of peripheral cores, ranging from USB interfaces to simple I2C controllers, which can simplify the implementation of a great deal of functionality. Typically, processor-based systems are programmed using the C language and, as already discussed, this is apposite for a number of tasks. This language is well suited to writing simple sequential algorithms, such as the PID controller, compared to the equivalent development in VHDL. It can also simplify the development of common hardware buses, such as UART or I2C based devices. On the flip side of this, is the inherent lack of parallelism in any processor, although multi-core devices can alleviate this shortcoming to a degree. This feature can be detrimental to control algorithms such as NF or ANN which feature a highly parallel architecture and will result in a loss of efficiency when written using a sequential language.

A further subset of processors, developed from traditional analogue processors, are specialist Digital Signal Processing (DSP) cores. Whilst most CPU devices are aimed at being adaptable and affordable, the DSP class of devices is instead specifically designed to perform fast math operations on digital signals, placing a premium on computational speed. As these devices are intended to offer fast, streamlined processing ability, they are often the preferred choice in real-time systems, especially when signals within the system are complex. There are two main features of a typical DSP which distinguish it from the more general CPU – the chip architecture and the operation of the Central Arithmetic Logic Unit (ALU). In most CPUs, the architecture is based on the Von Neumann approach, which means that there is a common memory used for both data storage and program storage. This is in contrast to the Harvard architecture, which is employed in most DSP devices, where there are separate memory spaces for data and program storage. The main motivation for this different architecture is the ability to reduce memory access times, leading to faster program execution times. This is especially pertinent in most DSP applications, as digital filters tend to require a large portion of data which can be quickly accessed from memory. The use of digital filters is also an important consideration in the deployment of additional, more efficient multipliers and accumulators in the main ALU. As digital filters use a large number of these operations, the use of these fast cores can reduce execution time for complex algorithms. These extra specialist features make the DSP subset of processors ideal for use in complex control systems (Wang et al (2013), Rodrigues et al (2013), Mekhilef and Kadir (2013) and Rezkallah et al (2017)). However, these more niche devices are often more expensive than the more general CPUs. In addition to this, FPGAs offer all the features of the DSP class of devices, whilst also benefitting from greater flexibility.

4.2.2 FPGAs

The Field Programmable Gate Array (FPGA), a programmable logic device, has been around as a device since the 1980s and has steadily gained in popularity since then. The basic structure of the FPGA consists of a matrix of reconfigurable logic blocks, as well as IO blocks and a network providing interconnection. Although the exact structure of the reconfigurable logic blocks varies between vendors and devices, typical architectures consist of a few inputs, look-up tables (LUT), multiplexers and flip-flops. There is also a degree of variation in the IO block structure between vendors but a typical device will contain some simple registers such as D-type flip-flops and tri-state buffers. Once programmed, the configuration of the logic and the interconnections are stored in memory on the device. In most modern devices, most notably those offered by Xilinx and Altera, this takes the form of SRAM due to its speed and its ability to be reconfigured several times. However, the SRAM does have to be reconfigured each time the device is powered on, as it is volatile memory, and so flash is becoming a popular alternative, although it is slower to configure and is more limited in the number of times it can be configured. As flash memory is more tolerant of single event upsets (SEU), it is also popular for use in safety-critical applications, such as Aerospace, and is offered in a selection of FPGAs such as the ProASIC family by Microsemi. Modern FPGA devices also feature several more advanced blocks which are used within digital systems, most commonly embedded RAM memory and dedicated, high-speed DSP blocks. A typical structure for a basic modern FPGA is shown in Fig 36.

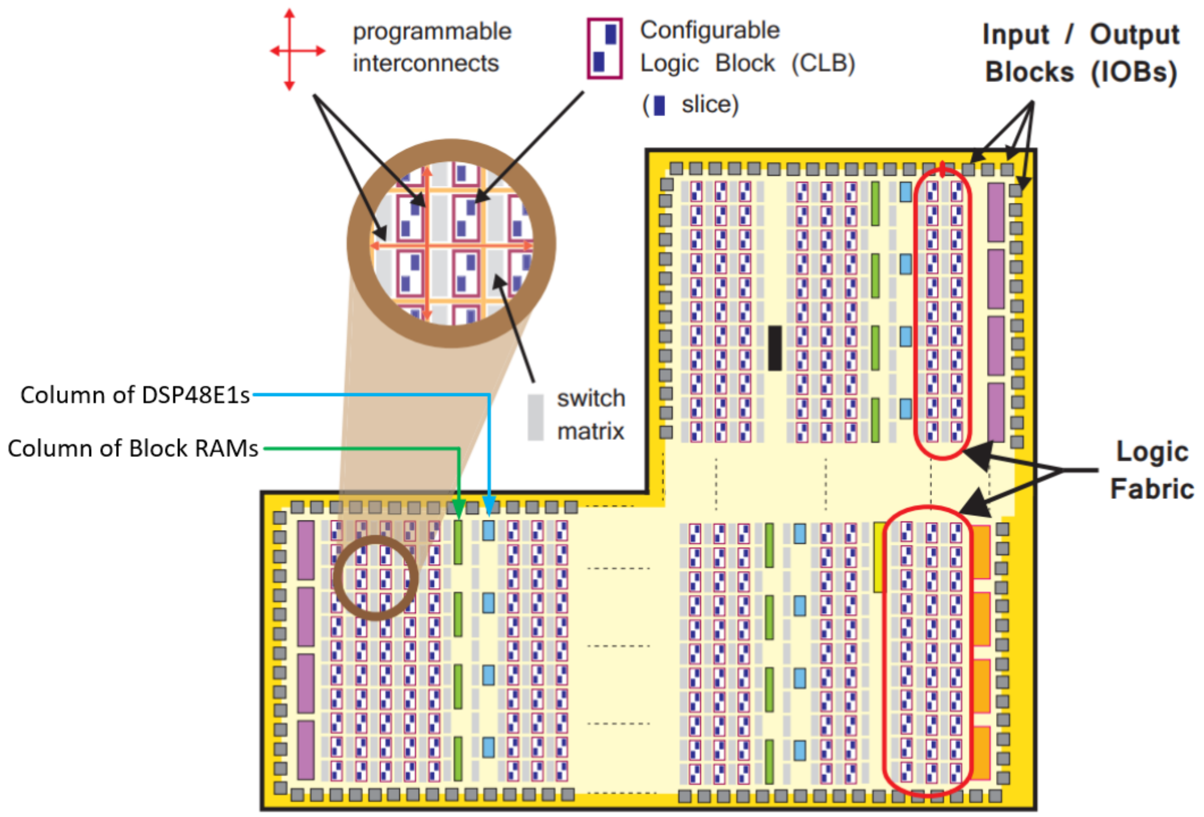

programmable interconnects
Configurable Logic Block (CLB)
(▮ slice)
Input / Output Blocks (IOBs)
Column of DSP48E1s
Column of Block RAMs
switch matrix
Logic Fabric

Fig 36 Example of a Typical FPGA structure (Crockett et al, 2014)

As FPGAs represent a pure piece of hardware, one major advantage they offer over a CPU is true parallelism. This feature is perhaps the key reason that it is an excellent choice for the design of intelligent controllers as it matches up well with the inherently parallel nature of these architectures. The FPGA can also offer tighter timing control and reduced latency when compared with the CPU, making it a good choice for high-speed applications or applications with tight timing constraints. Although the FPGA is well suited to the implementation of AI-based algorithms, it does have a few drawbacks which can make its use a little more complex compared to a processor core. As already mentioned, the C language is the language of choice for CPU programming and compared to VHDL this is a much simpler language to program sequential algorithms in. A good example of this is the PID controller, which in C would consist of just one or two lines of code whereas in VHDL the same function would likely take tens of lines to implement correctly. Similarly, the relatively trivial implementation of peripheral buses (such as SPI or I2C) in C languages, can become more complex in the VHDL language.

4.2.3 ASIC

Another popular technology for the implementation of digital circuits and systems, once they have been prototyped in an FPGA, is the Application Specific Integrated Circuit (ASIC). Unlike CPU or FPGA technologies, which are mainly general purpose, ASICs are designed to perform one specific function. Ordinarily, these devices follow a similar design flow to the FPGA, with a hardware description language being used to describe the functionality. However, whilst an FPGA design is then committed to a general purpose array of gates, ASIC designs are implemented by generating a highly customised design to be placed into an entirely bespoke IC. This means that ASICs can take two forms when being considered for a design – either an entirely new design can be created (and optimised) or an existing off the shelf solution could be selected. For more complex systems and designs, it tends to be more common for an entirely new chip to be created. This is normally attributable to the need for a highly niche device, which is unlikely to have been previously realised.

Although not generally as popular as FPGA based designs, ASICs do offer some distinct advantages which make them a good choice for certain applications. The greatest strength of the ASIC comes in the ability to create a more bespoke chip layout. This means that the device can be more optimized, either through physically placing interconnected circuits closer to one another or through the use of more specialized logic cells. As shown by Nurvitadhi et al (2016) and Lin et al (2014), allowing for a more bespoke chip layout means an order of magnitude improvement in the efficiency in terms of performance per watt. This means that ASICs can be run at higher frequencies than an equivalent FPGA, as shown by Cai et al (2015). An additional benefit of the bespoke layout is that a reduction in chip area is achievable. This stems from the fact that more specialized cells can be utilized, which results in less complex logic chains. In addition to this, FPGA implementations typically utilize less than the entire fabric of the device, meaning that a large amount of the chip can become redundant.

Whilst ASICs offer performance enhancements, there are some major disadvantages to their usage which make them unsuitable for deployment in most cases. The primary drawback is the cost involved with producing the actual IC itself. As these devices are less generic than FPGAs, with the fabrication of a new chip often being required, the economies of scale are no longer enjoyed. This means that the expense of setting up new production lines or methods, as well as associated chip level testing facilities, have to be incorporated into the development price. As chip manufacture is a far from trivial process, these expenses tend to be significant. Another knock-on effect of the need to develop a new chip is a more onerous development cycle in comparison to FPGAs. Whilst an FPGA can be programmed multiple times and different hardware configurations tested, this process isn't possible in an ASIC due to it being hardwired into the chip. This often means that a design will require several re-spins of the chip in order to fully guarantee the performance of the device, which can further increase cost. These features mean that the use of ASIC devices is only suitable for applications which have a high volume or require performance beyond that which could be achieved with a conventional FPGA.

4.2.4 All Programmable System-on-Chip (APSOC)

As FPGA technology has matured, allowing for bigger and more complex devices, many leading vendors now offer their own IP cores, which are functionally equivalent to an embedded CPU core. Some examples of these predefined models, known as soft cores, are the Nios family from Altera and the MicroBlaze and PicoBlaze families from Xilinx. Whilst these cores tend to include less computational power than high-end CPUs, as well as generally only allowing for a small size program, they can nonetheless run RISC instructions on them. This gives a greater level of flexibility and allows for any given device to feature some functionality which would previously only be possible on CPUs. Although these cores extend the versatility of the FPGA device, they are somewhat limited compared to a hard CPU core. Unlike most cheap microprocessors, the array of peripheral busses is somewhat limited. Although the vendors offer freely available HDL models for these communication buses, these require resources from the FPGA. Similarly, the use of these soft cores also requires a large portion of the FPGA to be used and, given the limitations of the cores, it is often price comparable to use a separate and more powerful general purpose processing device in addition to an FPGA.

In the last few years, a new approach to silicon design has been introduced by a number of vendors, which sees an FPGA and a CPU core integrated onto one chip, often known as an All Programmable System-on-Chip (APSOC) approach. Some examples of these offerings are the Smart Fusion family by Microsemi, the Altera Arria family and the Xilinx Zynq series of devices (which offers a dual-core ARM Cortex core package alongside an FPGA fabric). These APSOC solutions offer an excellent medium for the design of controllers, as they are capable of exploiting the advantages of both the CPU and the FPGA. This means that the FPGA core can be used for the complex parallel architecture required for the AI based controllers, whilst the CPU can be used for tasks which have lower throughput requirements. The block diagram in Fig 37 shows the architecture of the popular Zynq chip from Xilinx. It can be seen how this APSOC integrates predefined cores which are capable of implementing complex communication buses such as UART or I2C. These APSOC devices are an ideal candidate for the implementation of the system controller, as they offer all the advantages of the CPU and the FPGA in one simple silicon device, allowing for a more rapid creation of the prototype.

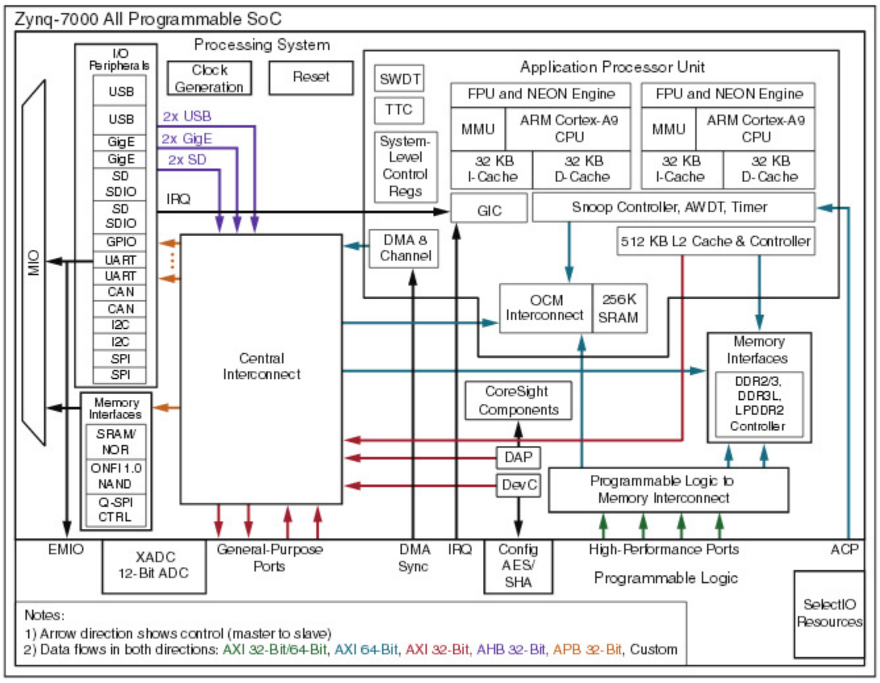


Fig 37 Xilinx Zynq APSOC Architecture (Xilinix (2013))

4.3 Methodology and Tools Used in this Project

In considering the tools for this project, there are three modelling and design elements which must be considered – the system level controller, the renewables and the power electronics - and, finally, the overall hardware solution. The first modelling consideration will be the system level controller design. In this process, which is outlined in more detail in the following chapter, several ANFIS based controllers are created and the tuning of PI controllers using the PSO algorithm is addressed. As discussed earlier in this chapter, the Matlab simulation environment offers a set of tools which are custom designed for achieving these tasks. The Fuzzy logic toolbox offers an easy to use package for quick design and verification of the ANFIS based models and is, therefore, chosen for this element of the project. Similarly, for the tuning of the PI controllers, Matlab features a powerful yet relatively simple syntax which lends itself very well to the flow of this algorithm. An additional feature which can be utilised in the Matlab environment is the ability to create a Simulink model which can interface with the PSim software. This means that the PSim tool can be used for the modelling of the power electronics, for which it is highly suited, thus creating an optimised rapid prototyping approach. The diagram in Fig 38 shows the general tool setup for the tuning of a PI controller, which will be an interim step used in the creation of the controllers for each renewable energy source.

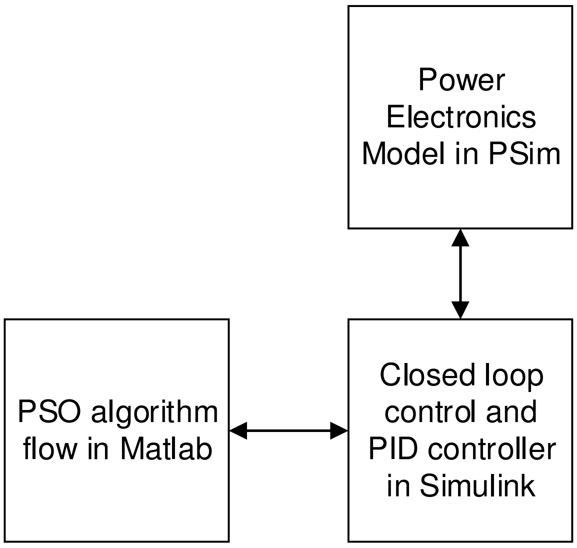

Power Electronics Model in PSim
PSO algorithm flow in Matlab
Closed loop control and PID controller in Simulink

Fig 38 PSO PID Tuning Tool Flow

The other main consideration of the system level model is formed by the renewable energy elements. In this system, there will be four separate sources - a solar panel, a wind turbine, a battery and a supercapacitor. The PSim simulation software is the ideal tool for this task, offering generic models of the solar panel, battery and wind turbine. This can aid in the streamlining of the modelling and verification process for these components. A relatively simple equivalent circuit can be developed for the supercapacitor, as is discussed in more detail in the following chapter. This makes use of resistors and capacitors to create performance which is analogous to a supercapacitor. A straightforward SPICE based approach, using existing PSim components, can be used for this purpose. The full system level modelling approach is shown in Fig 39 as part of the overall tool flow of the project. This highlights the models of the renewable energy sources and power electronics in PSim, as well as the controller in Matlab.

In the final modelling stage, the main AI based novel controller algorithm is implemented in a hardware model for simulation alongside the power electronics circuitry. The final hardware platform is discussed in further detail in the next section. This stage of the project will be primarily implemented using VHDL, with the C programming language being used for supporting functionality. For simulations of the VHDL code, the modelling tool used shall be the QuestaSim environment from Mentor Graphics. This is an advanced and powerful industry standard tool which offers extensive support for the VHDL language. Support is also offered for the TCL scripting language, which can help to reduce the development time by automating the tool flow. The other main attraction of QuestaSim is the ability to be interfaced with the PSim software. This allows for the VHDL model to be simulated alongside the power electronics, allowing for closer consideration of the algorithm's performance. The diagram in Fig 39 shows how the controller algorithm will be transformed, in stages, from an abstract model into an array of gates for the final hardware target. The synthesis, layout and programming file generation shall be carried out using the Xilinx Vivado tool.

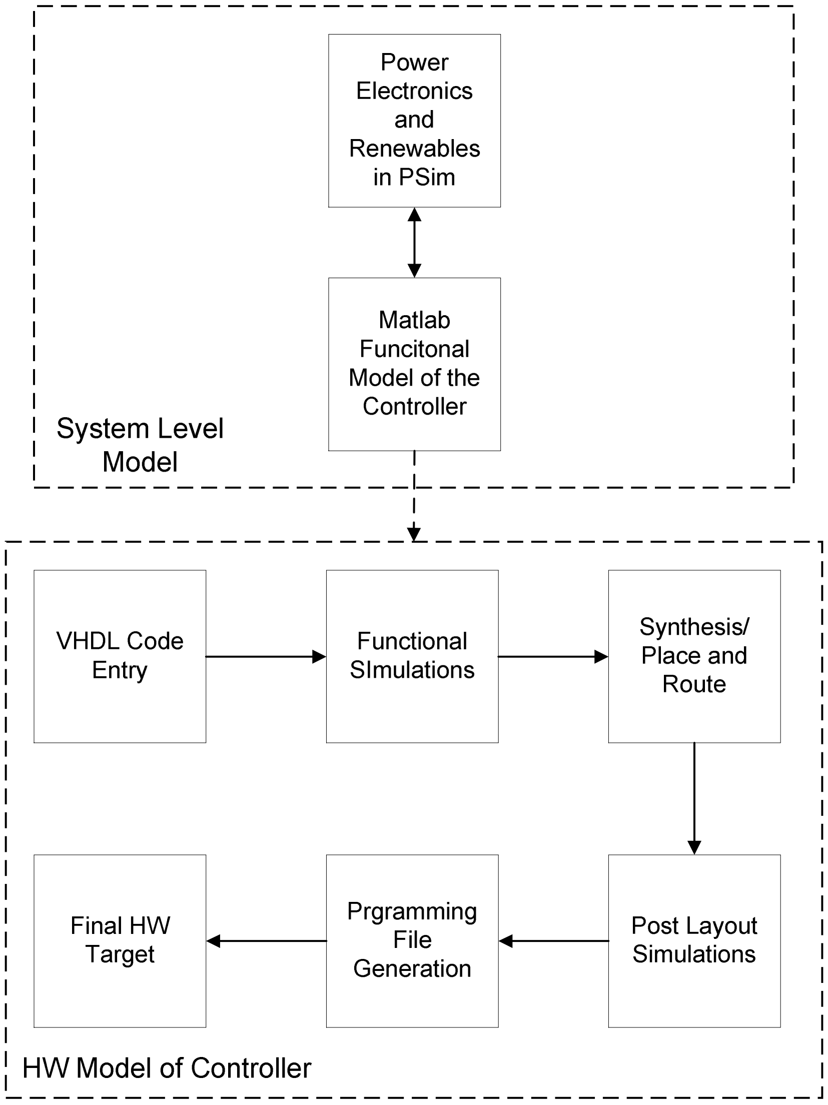

Power Electronics and Renewables in PSim
Matlab Funcitonal Model of the Controller
System Level Model
VHDL Code Entry
Functional SImulations
Synthesis/ Place and Route
Final HW Target
Prgramming File Generation
Post Layout Simulations
HW Model of Controller

Fig 39 Model Tools and Flow Utilised in this Project

The final consideration for this system is the choice of hardware platform for the controller implementation. As has already been mentioned, the fuzzy logic and the ANFIS controllers will be implemented using an FPGA. This choice is based on the fact architectures of both of these controllers lend themselves well to the parallelism that can be achieved within the FPGA. In addition to these control loops, it will also be necessary for the final hardware to interface with several ADC's for the monitoring and control of the system. A number of FPGA cores, especially those offered by Xilinx, feature onboard ADCs which can be used for this purpose. A simple parallel interface is offered with this approach, simplifying access for the FPGA. This methodology also maximises throughput when compared to using a sequential method for ADC access. A number of other basic control functions, such as look-up-table (LUT) management and supervision of the main controllers, can be more easily realised using a CPU. As the optimal solution for this hardware implementation features both an FPGA and a processor, the platform used will be the Zedboard, as shown in Fig 40, which utilises the Zynq-7020 APSOC from Xilinx. This device features an integrated Artix 7 FPGA offering approximately 1.3 million logic gates and a Dual ARM Cortex A9 processor.

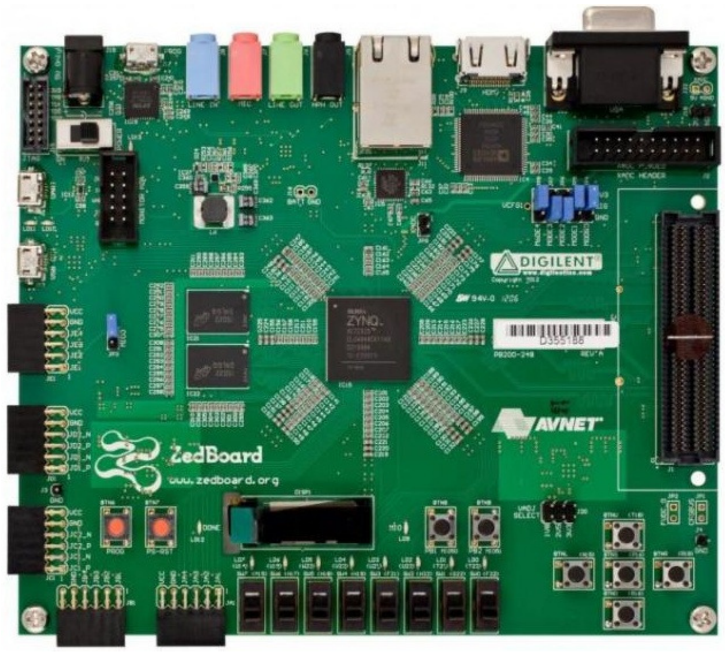


Fig 40 The Zedboard Hardware Platform

5 Modelling of Local Renewable Power Converters

In the previous chapters, some of the key concepts used in the implementation of DER systems were introduced. This included a review of power converters and intelligent control systems which are fundamental to DER systems. In this chapter, a new approach to the modelling of such a system is introduced and the creation of suitable power converters and associated controllers is discussed. These models feature co-simulation between Matlab and PSim, as described in the previous chapter, with PSim being used to model the power electronics and Matlab to model the controllers.

The main renewable energy sources used for this system will be a PV cell, a BP3230N rated at 230W, and a wind turbine, an Ampair 100 rated at 100W. The maturity of these technologies means that they are often utilised within renewable energy systems. Wind and irradiance also offer the advantage of being well suited as complementary resources, as there is a tendency for periods of low irradiance to coincide with high wind speeds and vice versa. In order to ensure the security of the power supply, it is important that the system includes a means for storing energy. When the renewable energy sources produce more power than is required by the loads, the surplus energy can be used to charge the storage components. This power can then be utilised in periods when the renewables are producing insufficient power, helping to extend the efficiency of the system. There are two devices which are commonly used for power storage in DER systems – batteries (Garcia et al (2014), Brenna et al (2017) and Hussain et al (2017)) and fuel cells (Sekhar and Mishra (2016), Biswas and Bajpal (2014) and Moré et al (2015)). Whilst fuel cells offer the advantage of environmentally clean power, the difficulties in producing hydrogen mean that this technology is inefficient in comparison to battery technologies. Therefore, the more mature battery technology is utilised as the main power storage element within this system

One characteristic which can create issues within DER system is the relatively slow response time of the renewables and associated storage elements, owing to the low power density of these devices. As supercapacitors have a superior response time under dynamic conditions, due to their higher power density, it has become common for these devices to be utilised for voltage compensation (Thounthong et al (2013), Sikkabut et al (2016) and Perqueroles-Queralt et al (2015). One particularly good example of such a system is presented by Thounthong et al (2013), where the supercapacitor is the main source of output power in the system. The renewable energy sources are then utilised in order to ensure that the supercapacitor has adequate charge to supply the system loads. The results presented in this paper demonstrate how this topology is capable of delivering excellent stability even under fluctuating conditions. A similar example is presented by Shamra and Mihra (2017), whereby the renewable energy sources in the system are used in order to provide steady state power whilst a supercapacitor is utilised in order to smooth out fluctuations and transients. Further examples of this approach are given by Sikkabut et al (2016) and Mane et al (2016), each of which utilises the supercapacitor in the system to provide voltage compensation under load transients. Therefore, the supercapacitor is used as the primary source of power for the loads in this system, with the wind turbine, solar PV cell and battery being used to maintain the state of charge for this device. In order to facilitate this strategy, the renewable energy sources are tied into an intermediate bus which supplies power to the supercapacitor via a buck converter. The system loads are then supplied directly by the supercapacitor through a boost converter which allows the supercapacitor to source power into the loads, thus allowing for improved voltage compensation.

The system diagram shown in Fig 41 illustrates how each of the power sources will be interconnected to produce a controlled 48VDC bus. A power controller is included in the system in order to manage the way in which the energy flows from the renewable sources to the various system loads. The primary objective of this controller, which is discussed in depth in the next chapter, is the maintenance of the supercapacitor state of charge.

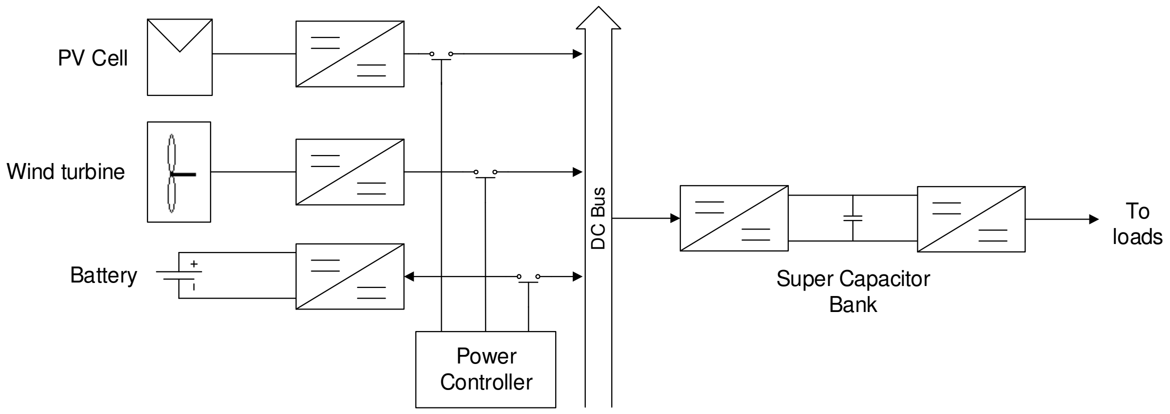


Fig 41 System Block Diagram

The relatively small output power of the renewable energy sources, coupled with the fact that no AC power is available, means that this topology can be viewed as somewhat academic. In a more practical application, it is likely that the power capacity of the renewable energy sources would be several MW. In addition, the DC bus would have a much higher voltage, most likely several hundreds of volts, and there would more than likely be an AC output and a grid connection. However, this system has been primarily chosen to demonstrate the performance of the novel control solution which is extensively detailed in sections 5.1.1 and 5.1.2 of this chapter. Despite this, there are a number of applications which require a 48VDC bus, for example in lighting systems (Chew et al (2015) and Koh et al (2017)) and telecommunications data centres (Usui et al (2015) and Schmidt and Myhre (2015)), illustrating that there are some practical applications for this system. As a further stage of the development work, once the concept is proven with this model, it would be possible to implement a more practical application using the ideas presented. This would involve increasing the voltage bus to a higher level, such as the commonly used 400VDC (Fukui et al (2010), Lisy et al (2014) and Zhang (2017)), increasing the power generation capabilities of the renewable energy sources and adding an inverter. Although the power capacity of the system would be increased in this scenario, the control principles which are applied to the 48VDC system in this project could still be utilised. Alternatively, given that many systems require low voltage buses, the 48VDC bus could be used as an intermediate bus with a further step-up DC to DC converter being employed and an inverter added. A similar approach to this is presented by George and Ang (2016), albeit the 400VDC bus acts an intermediary to the lower voltage DC loads. The design, implementation and modelling of the system's local power converters are considered in depth in the rest of this chapter.

5.1 Photovoltaic Power Converter

In this section, the power converter design and the control algorithm utilised in the PV sub-system is presented. This is followed by a discussion of the model before the simulation results are detailed.

5.1.1 Topology

The solar PV cell used in this system has an output voltage range of approximately 26 to 36V. In order for this voltage source to feed the 48VDC bus, it is necessary for a boost type power converter to be utilised. The minimum value for the inductor (L_{min}) for the boost converter is determined using the equation given in Eq. 7 where D is the duty cycle, V_{out} is the regulator output, I_{min} is the minimum output current of the converter and F_s is the switching frequency. The value for the capacitor (C) is selected using the equation given in Eq. 8 where I_{out} is the maximum output current of the converter and ΔVout is the desired ripple voltage. The values are chosen as 22uH and 470uF, giving a theoretical maximum ripple current of 500mA through the inductor and a maximum voltage ripple of 5mV on the output.

$$L_{\min} = \frac{V_{out} D(1-D)^2}{2F_s I_{\min}} \qquad \textit{Eq. 7}$$

$$C = \frac{I_{out} D}{F_s \Delta V_{out}} \qquad \textit{Eq. 8}$$

There are two distinct control algorithms employed in the solar PV sub-system. The role of the first control algorithm is to provide maximum power point tracking (MPPT), ensuring that the PV cell delivers power at near to its maximum operating point in any given state. This is accomplished through the use of a load balancing algorithm which iteratively adjusts the output load until the MPP is deemed to be achieved. The second control algorithm is responsible for the regulation of the power converter output voltage.

5.1.1.1 Control Strategy

The boost converter circuit is highly non-linear and can be modelled as a second-order system. This means that optimum control can be difficult to achieve and requires a non-linear control solution. One method which is commonly employed to simplify the control of the circuit is the use of cascaded PI controllers, as shown in Fig 42. In this approach, the outer PI controller is influenced by the bus voltage, whilst the inner PI controls the current through the inductor. The major advantage of adopting this approach is that it has the effect of turning the inductor into a current source. As a result of this, the complexity of the transfer function can be reduced so that the system is considered as a first-order type.

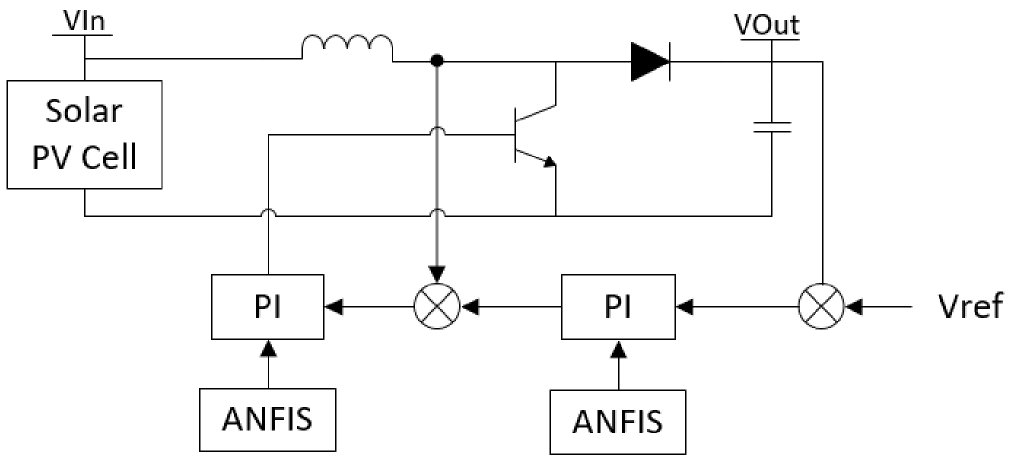


Fig 42 Control Strategy for the Solar PV Boost Converter Circuit

In the previous chapter, one well-researched method for power converter control was the use of an FLC to provide gain scheduling for a PI(D) controller. In the papers presented by Rabbani et al (2012), Abhinav and Sheel (2012) and Chang et al (2016) comparisons are made between a buck controller using a traditional PID and a gain-scheduled PID. In the presented systems, all of the key performance characteristics are improved. The rise time and settling time are up to ten times smaller whilst the ripple is up to a hundred times smaller. This shows that complementing the operation of a PI(D) controller with an intelligent gain scheduling algorithm can result in improved performance. In systems which feature a non-linear plant, such as a switching power converter, the use of gain scheduling control is even more conducive to the optimal performance of the system. Using this approach allows the system to be effectively split into a number of linear operating regions, thus improving the effectiveness of the essentially linear PI(D).

An improved intelligent control solution, which combines the benefits of the fuzzy logic controller and the learning ability of the NN, is known as an ANFIS controller. The ANFIS architecture was first proposed by Jang (1993) as a means to exploit the advantages of both ANN and FL. Whilst the ANFIS algorithm is around two decades old, little work has been done on integrating this architecture into a DC-DC power converter controller. One of the key issues with the ANFIS architecture has traditionally been the large amount of processing power which is required in comparison to other intelligent controllers. However, advances in modern semiconductor technologies mean that the implementation is now more viable. In this system, ANFIS controllers will be employed instead of fuzzy logic to provide gain scheduling for the PI controllers, as is shown in Fig 42.

5.1.1.2 Soft Start Circuit

Without a supervised start-up routine, the boost converter suffers from a large in-rush current as the capacitor is charged up. This initial in-rush current can cause damage to the power converter components and requires mitigation. An additional problem that takes places during the start-up of the boost converter occurs when the output voltage is below the input voltage as this causes the control loop to become unstable. In order to overcome these issues, it is necessary to incorporate a “soft start” function. This is designed to limit the in-rush current and slowly ramp up the output voltage before the controller begins normal operation.

During the first stage of the soft start function, a constant current source is used to charge the capacitor at a fixed rate. This is switched off once the capacitor voltage has reached the same value as the input voltage. One advantage of this method is that it acts as a current limit for the inductor as well as the capacitor. This means that there is no need to have a separate pre-charge stage, something which is commonly implemented. The power converter transistor is left open during this mode of operation, meaning the boost circuit is not operational.

In the second phase of the soft start function, a PWM signal is applied to the gate of the transistor and the regulator starts to operate. However, the control loop remains inactive and the PWM duty cycle is set to a fixed rate. This is continued until the output voltage reaches 48V, at which point the controller begins operating. When the converter begins operating in its normal mode of operation, a p-MOSFET is activated to connect the output load. The state machine shown in Fig 43 gives a graphical representation of the flow of the soft start function.

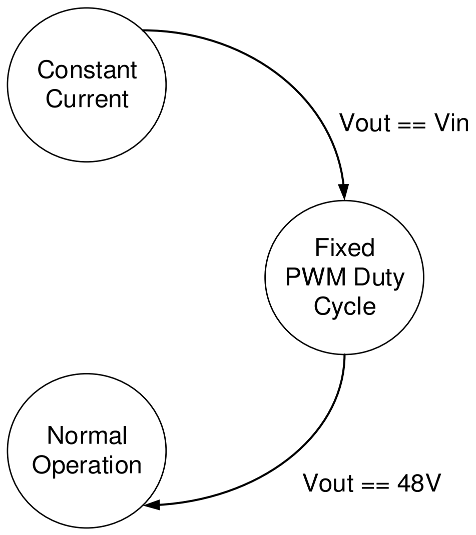


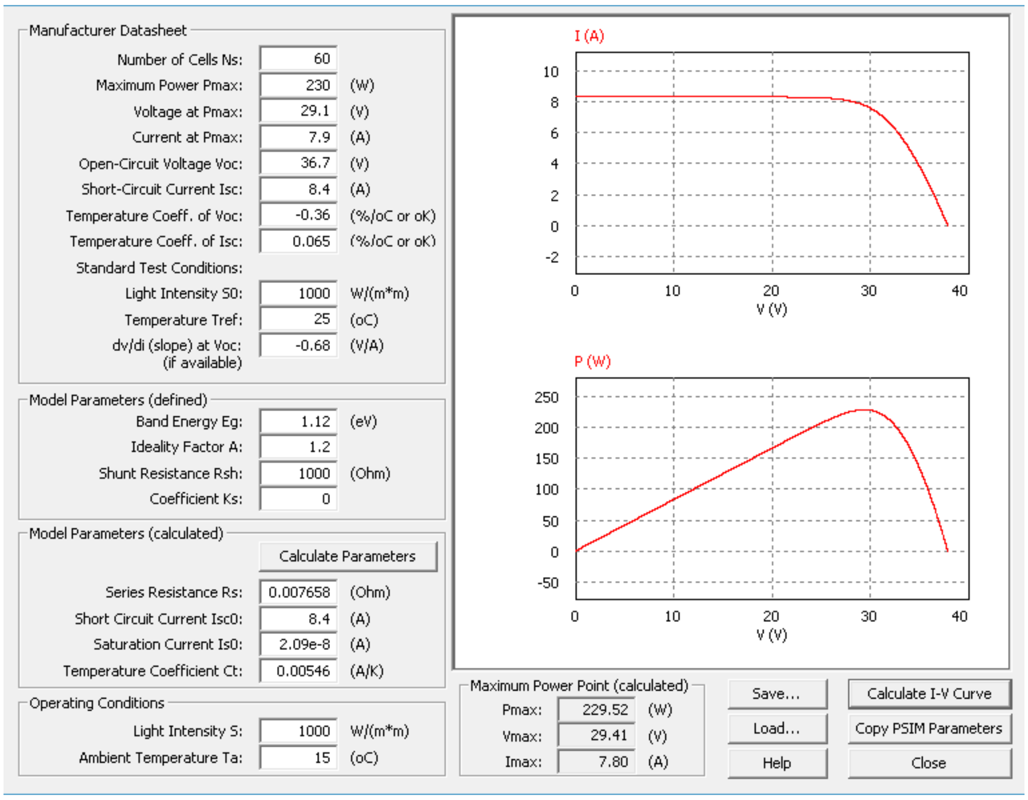
Fig 43 Soft Start State Machine

5.1.2 Modelling

The modelling of the PV sub-system has three main considerations – the solar PV cell, the MPPT algorithm and the regulator control loop. The modelling of the solar PV cell is done exclusively in the PSim simulation tool using pre-packaged wizards, allowing for a rapid modelling approach. The load balancing algorithm, which is used for MPPT, is developed using C code for easy portability and is also modelled in PSim. Finally, the model of the boost converter and the associated controller utilises both Matlab and PSim. The rest of this section details more closely the modelling of this subsystem.

5.1.2.1 Solar PV Cell

The modelling of solar PV cells can be a time-consuming task. However, one major advantage of the PSim software is the inclusion of a utility which creates a model of the PV cell using datasheet parameters. The screenshot given in Fig 44 illustrates how this tool can also be used to generate and validate power curves. Once this model has been created and validated, it can then be imported into the PSim system model for simulation.



Manufacturer Datasheet

Number of Cells Ns:	60	
Maximum Power Pmax:	230	(W)
Voltage at Pmax:	29.1	(V)
Current at Pmax:	7.9	(A)
Open-Circuit Voltage Voc:	36.7	(V)
Short-Circuit Current Isc:	8.4	(A)
Temperature Coeff. of Voc:	-0.36	(%/oC or oK)
Temperature Coeff. of Isc:	0.065	(%/oC or oK)
Standard Test Conditions:		
Light Intensity S0:	1000	W/(m*m)
Temperature Tref:	25	(oC)
dv/di (slope) at Voc: (if available)	-0.68	(V/A)

Model Parameters (defined)

Band Energy Eg:	1.12	(eV)
Ideality Factor A:	1.2	
Shunt Resistance Rsh:	1000	(Ohm)
Coefficient Ks:	0	

Model Parameters (calculated)

Series Resistance Rs:	0.007658	(Ohm)
Short Circuit Current Isc0:	8.4	(A)
Saturation Current Is0:	2.09e-8	(A)
Temperature Coefficient Ct:	0.00546	(A/K)

Operating Conditions

Light Intensity S:	1000	W/(m*m)
Ambient Temperature Ta:	15	(oC)

Maximum Power Point (calculated)

Pmax:	229.52	(W)
Vmax:	29.41	(V)
Imax:	7.80	(A)

Fig 44 PSim Model of the BP3230N Solar PV Cell

The physical model of the solar PV cell which is utilised within PSim uses the equivalent circuit given in Fig 45 below. The total output current of the cell is given by Eq. 9 (Powersim (2017)), where i is the output current, i_{ph} is the current generated due to the photoelectric effect, i_d is the current lost in the diode and i_r is the current lost through the shunt resistor (R_{sh}).

$$i = i_{ph} - i_d - i_r \qquad \text{Eq. 9}$$

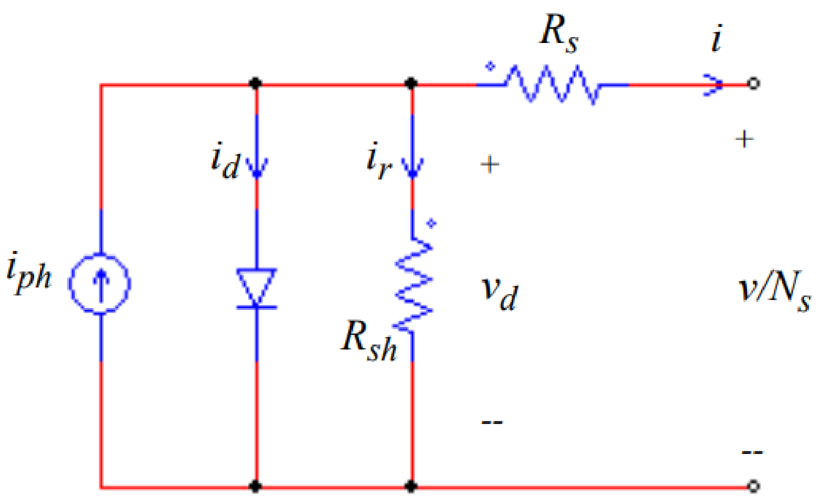


Fig 45 PSim Solar Cell Equivalent Circuit (Powersim (2017))

The value of i_{ph} is given by the equation in Eq. 10 (Powersim (2017)), where I_{sc0} is the short circuit current, S is the intensity of the light, S0 is the light intensity under standard test conditions, C_t is a temperature coefficient and T_{ref} is the temperature under standard test conditions. The values of Tref, S and Ct can normally be obtained from a device's datasheet. The value of T is equivalent to the current temperature of the cell and can be calculated using the equation given in Eq. 11 (Powersim (2017)), where k is the Boltzmann constant and T_a is the ambient temperature.

$$i_{ph} = I_{sc0}.\frac{S}{S_0} + C_t.(T - T_{ref}) \qquad \textit{Eq. 10}$$

$$T = T_a + kS \qquad \textit{Eq. 11}$$

The value of i_d can be calculated using the equation given in Eq. 12 (Powersim (2017)), where q is the electron charge, k is the Boltzmann constant and A is a constant which depends upon the type of material used in the cell. The values of I_0 and v_d are calculated using the equations given in Eq. 13 (Powersim (2017)) and Eq. 14 (Powersim (2017)) respectively, where I_{s0} is the saturation current of the diode, E_g is a constant which depends upon the type of material used, v is the output of the solar panel, N_s is the number of individual cells which make up the solar panel and R_s is the value of the series resistance as shown in Fig 45.

$$i_d = I_0.\left(e^{\frac{qv_d}{AkT}} - 1 \right) \qquad \textit{Eq. 12}$$

$$I_0 = I_{s0}.\left(\frac{T}{T_{ref}} \right)^3 .e^{\frac{qE_g}{Ak}\left(\frac{1}{T_{ref}} - \frac{1}{T} \right)} \qquad \textit{Eq. 13}$$

$$v_d = \frac{v}{N_s} + i.R_s \qquad \textit{Eq. 14}$$

Finally, the value of i_r can be calculated using the equation which is given in Eq. 15 (Powersim (2017)) with R_{sh} being the value of the shunt resistor in Fig 45.

$$i_r = \frac{v_d}{R_{sh}} \qquad \textit{Eq. 15}$$

5.1.2.2 Load Balancing MPPT Algorithm

The power produced by solar PV cells is influenced by both the solar irradiation and the operating temperature. When the solar panel is operating below its MPP, the excess power is transformed into heat which can further reduce the efficiency of the panel. This characteristic means it is important that an MPPT algorithm is implemented to ensure that the maximum available power is extracted. There are a number of algorithms which have been applied to this task, with the most popular being Perturb and Observe (P&O), Incremental Conductance (INC) and ANN based approaches. Using ANN to approximate the behaviour of the solar PV cell is a relatively new approach. This involves training the network so that it can model the behaviour of the cell and provide information on the current operating conditions of the system. According to Khanaki et al (2013), El Telbany et al

(2014) and Sunny et al (2016) this approach can be shown to perform favourably compared to the other algorithms. This is shown to offer faster tracking, greater precision and fewer oscillations around the maximum power point. However, there are some major drawbacks to the use of ANN for MPPT. Compared to the other methods, the ANN is the most computationally expensive and difficult to implement. Additionally, this algorithm is technology dependent, meaning that the network must be trained for each device to which it is applied. As the characteristics of solar PV cells also change over time, the NN will either eventually become inaccurate or will require periodic retraining. Whilst the ANN approach offers high performance, the associated drawbacks make it an unattractive option compared to the other approaches.

The P&O and INC algorithms are both examples of hill climbing algorithms, so called because they rely on traversing the power curve of the solar PV cell. The P&O algorithm works by slightly adjusting the output voltage. If this change causes an increase in power then the voltage is adjusted further in that direction, otherwise the voltage point is moved in the opposite direction. The main advantages of the P&O algorithm are its efficiency, computational simplicity and the fact that it is technology independent. However, the P&O algorithm does suffer from oscillations around its MPP, caused by the perturbation in voltage the algorithm uses to track power.

The INC algorithm is similar to the P&O algorithm in so much as it relies on creating a perturbation in the operating voltage and monitoring the effect on the output power. However, the INC algorithm also monitors the derivatives of the output power (dP) and the output voltage (dV). The algorithm is then able to determine when the MPP is reached by exploiting the fact that dP/dV is equal to zero when this occurs. This relationship can also be used to determine if the power is below MPP, when $dP/dV < 0$, or above the MPP, when $dP/dV > 0$. This allows for the INC algorithm to track the MPP with greater efficiency. Fig 46 shows how the power-voltage graph can be divided into the three sections depending on the state of dP/dV. As this algorithm offers an effective, relatively simple, technology independent solution this algorithm is chosen as the basis for the MPPT algorithm utilised in this system.

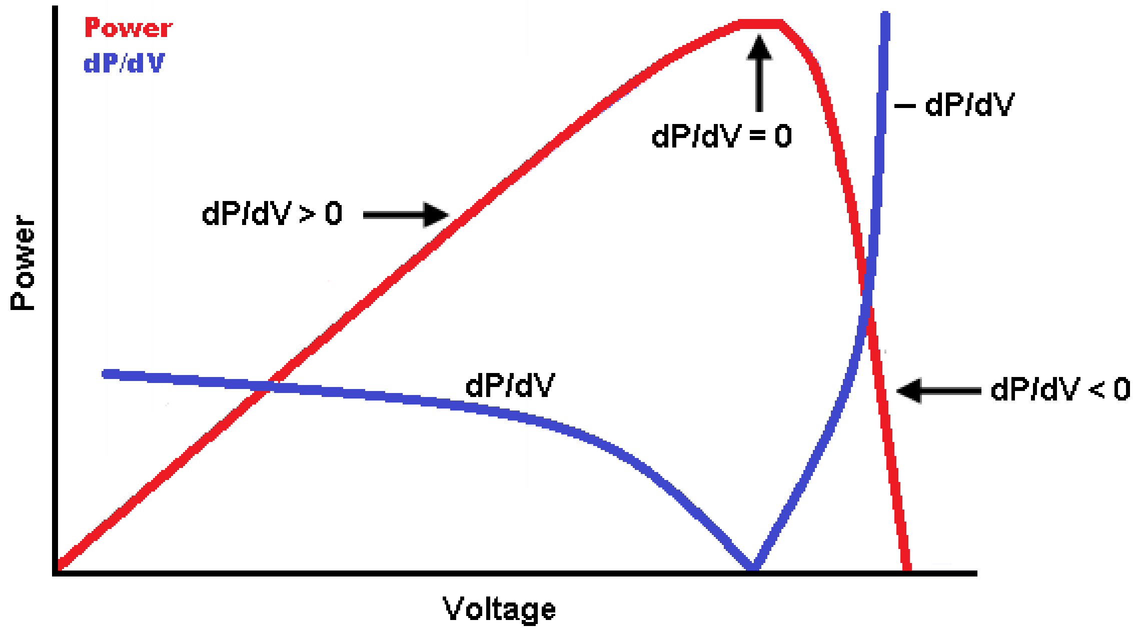


Fig 46 Example Solar PV Power-Voltage Curve

The relationship between dP/dV can also be expressed in terms of the array voltage and current, as is shown in Eq. 16 . As it is computationally less expensive to express in terms of voltage and current, the INC algorithm splits the curve into three portions based on these values. This means that the algorithm increases the operating voltage by a set step size when dI/dV > - I/V and decreases it by a set step size when dI/dV < -I/V. In addition to this, if dV is zero then the operating voltage increases when dI is positive and decreases when it is negative. When dV/dI is equal to –I/V or both dV and dI are zero then the operating voltage is unchanged. The step size which is used to alter the operating voltage is an important feature in determining the performance of the algorithm. Using a larger step size increases the speed at which the MPP is tracked but means that the accuracy may be affected. This can also cause oscillations to be generated when the MPP is reached. Using a larger step size can also cause oscillations to be generated when the MPP is reached. On the other hand, using a smaller step size will result in slower tracking but means that the accuracy is improved and reduces oscillations.

$$\frac{dP}{dV}=\frac{d(VI)}{dV}=I\frac{dV}{dV}+V\frac{dI}{dV}=I+V\frac{dI}{dV} \qquad \textit{Eq. 16}$$

An elegant solution to the step size issue is presented by Liu et al (2008) with a variable step size being employed. This approach means that the fast dynamic tracking characteristics can be preserved without adversely affecting the steady-state operation. To facilitate this variable step size, the algorithm exploits the fact that dP/dV reduces to a very small value as the MPP is approached. These features mean that this value is an ideal choice for determining the size of the variable step. With the addition of a scaling factor N, the size of the step can be calculated as is shown in Eq. 17 where D denotes the duty cycle. The scaling factor N has a big influence in determining the performance of the MPPT system. Utilising larger values leads to faster dynamic tracking but increases the oscillations around the MPP. In order to ensure suitable performance of the tracking algorithm, it is recommended that a fixed step size, Dmax, is initially used to promote fast-tracking. The value of N should then be selected such that it satisfies the condition given in Eq. 18 .

$$D(k) = D(k-1) \pm N.\left|\frac{dP}{dV}\right| \qquad \text{Eq. 17}$$

$$D_{\max} > N.\left|\frac{dP}{dV}\right| \qquad \text{Eq. 18}$$

One drawback associated with the use of the INC algorithm in this system is that it can't operate properly when the power converter has a fixed output voltage. This makes the INC algorithm inappropriate for use in this system and a modified version is instead employed as a load balancing algorithm. This means that the algorithm modifies the load applied to the converter rather than directly influencing the power supplied. Using this mechanism, the output power of the solar PV cell can be manipulated until the MPP is reached. The basic premise of the load balancing algorithm is that the load is increased or decreased according to how the last iteration affected the output. When the power increases between iterations, the next step is in the same direction as the previous step. When the opposite is true, with power decreasing, then the load is moved in the opposite direction. However, in order to avoid oscillations in the output, it is necessary to consider the relationship presented in Fig 46. This means that when dV/dI is equal to –I/V then the load remains unchanged, as is the case with the INC algorithm. The load balancing algorithm employs the same methodology as is utilised by Liu et al (2008) for the determination of the step. Therefore, the step size is variable and is determined using the same methodology as presented in Eq. 17 . However, rather than modifying the duty cycle of the converter, the load is instead altered. The flowchart of the full load balancing algorithm is given in Fig 47.

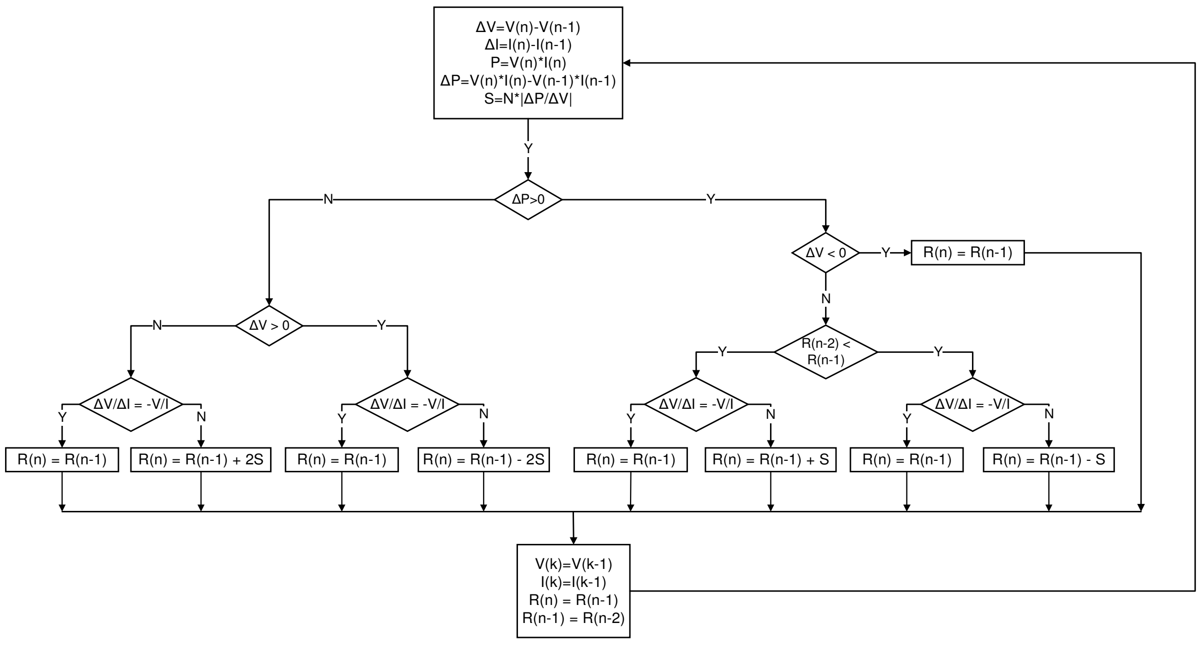


Fig 47 Load Balancing MPPT Algorithm

5.1.2.3 PI(D) Tuning

In order to obtain the training data for the ANFIS controllers, it's necessary to tune the PI controllers in a number of different operating scenarios. These scenarios depend on the state of the load current and take place at numerous steady state set points as well as with a number of load transients. The tuning of the PI controllers is done using the PSO algorithm which was discussed in the previous chapter. The use of this algorithm is a relatively new concept but it has been shown to yield good results compared to traditional methods (Chen et al (2015), Bevrani et al (2012), Wai et al (2011) and Hanifah et al (2017)). Unlike with the traditional methods, which only consider the steady-state operation, this approach also allows for the consideration of transient conditions. This feature makes the PSO algorithm an ideal choice for the gain scheduling tuning. As the control loop uses cascaded PI algorithms, the inductor controller must be tuned first with the voltage controller being considered once this is complete.

The PSO algorithm is designed to move a number of potential solutions, known as particles, through the problem search space in co-operation with one another, as is shown in Fig 48. When one of these particles finds a promising solution, the others are encouraged to move towards it. The particle is, therefore, the key component within the PSO algorithm. The particle actually consists of three separate components – the current potential solution, the velocity and the personal best fit. The velocity is a key parameter in determining how the problem space is searched as it controls how fast and in which direction a solution is accelerated. The equation for updating the particle is given in Eq. 20 , where Rand represents a random number between 0.0 and 1.0, and this plays a key role in the performance of the PSO algorithm. The social and cognitive coefficients, shown by C1 and C2 respectively, have the effect of pulling the solution towards either its personal best fit (cognitive) or the global best fit (social). These coefficients, therefore, control how much the algorithm converges towards a global maximum or searches around promising potential solutions. Achieving a good balance between the two parameters is crucial to optimal performance. One widely used concept was introduced by Ratnaweera et al (2004) and has been shown to give favourable results. This methodology involves linearly decreasing the value of the cognitive coefficient from 2.5 to 0.5 whilst simultaneously increasing the social coefficient from 0.5 to 2.5. This is achieved using the formulas given in Eq. 21 and Eq. 22 . Using this method means a wider search space can be explored initially before the algorithm focuses in on a global best fit. Another important parameter of the particle velocity is the inertia weight, as shown by ω in Eq. 20 which was a later development introduced by Eberhart and Shi (1998). This concept is intended to reduce stagnant particles which focus on local, rather than global, minima. Therefore, inertia weight is an important factor in promoting convergence and improving the algorithm efficiency. Setting this parameter to a high value promotes a wider search space, giving the algorithm the opportunity to consider a wider selection of potential solutions. Setting the value to a smaller value, on the other hand, forces the algorithm to search around a smaller solution area, in effect giving an element of fine tuning. As it is beneficial to initially have a large search space before reducing the search area to help find a true optimum, a popular strategy is to begin with a large value which is iteratively reduced. The papers presented by Zhan et al (2009), Park et al (2010) and Bansal et al (2011)) show that the best solution is to linearly decrease the inertia weight between the values of 0.9 and 0.4. This approach can be implemented using the equation in Eq. 23 where ω_{max} is equal to 0.9 and ωmin is equal to 0.4.

The final important consideration of the PSO algorithm is the method by which the particles fitness is assessed. This necessitates the selection of a function which can evaluate how well the potential solution solves the problem being explored. There are a number of different methods which can be used for this task, with one of the most popular being the mean square error (MSE). This algorithm calculates the mean of all errors squared and offers a robust, simple method of measuring fitness quality. This algorithm is implemented as shown in Eq. 19

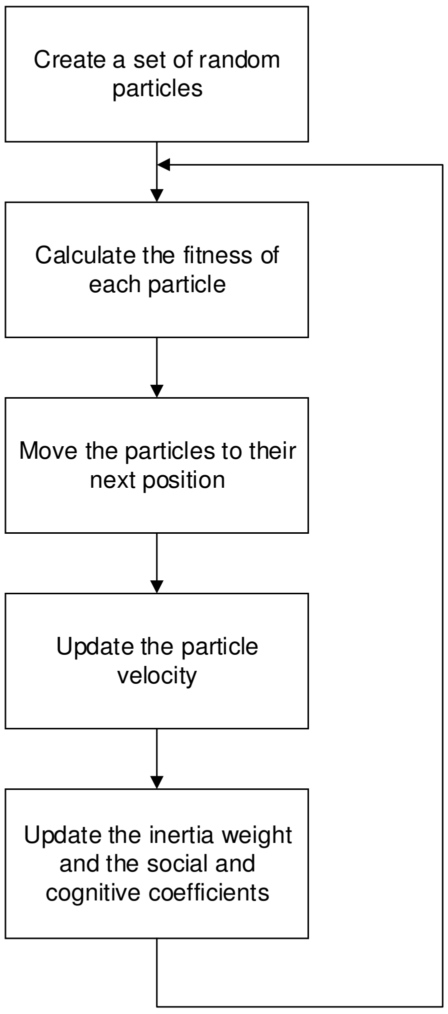

Create a set of random particles
Calculate the fitness of each particle
Move the particles to their next position
Update the particle velocity
Update the inertia weight and the social and cognitive coefficients

Fig 48 Particle Swarm Optimization Flow Chart

Mean Square Error Algorithm

$$MSE = \frac{1}{n}\sum_{x=1}^{n} E_x^{\,2} \qquad \text{Eq. 19}$$

Equation for Updating the Particle Velocity

$$V_{N+1} = \omega V_N + (C_1.Rand.(GX_N - X_N)) + (C_2.Rand.(PX_N - X_N)) \qquad \text{Eq. 20}$$

Equations for Updating the Social and Cognitive Coefficients

$$C_1 = c_{\max} - (\frac{c_{\max} - c_{\min}}{iteration_{\max}}) \times iteration \qquad \text{Eq. 21}$$

$$C_2 = c_{\min} + (\frac{c_{\max} - c_{\min}}{iteration_{\max}}) \times iteration \qquad \text{Eq. 22}$$

Equation for Updating the Inertia Weight

$$\omega = \omega_{\max} - \frac{\omega_{\max} - \omega_{\min}}{iter_{\max}} \times iteration \qquad \text{Eq. 23}$$

5.1.2.4 ANFIS Controller

The ANFIS controller plays a crucial role in the control of the local power converters, providing gain scheduling to improve the linearity of the PI(D) controllers. In the previous chapter, the algorithm was introduced and a brief discussion presented. In this section, a more detailed discussion of the algorithm background is presented before the modelling solution is considered.

5.1.2.4.1 Algorithm Background

The ANFIS algorithm consists of a five-layer feed-forward neural network which is trained to behave like a Fuzzy logic inference system. When trained, the behaviour of the ANFIS is analogous to a Sugeno type system. This family of algorithms are universal approximators which are capable of approximating non-linear functions, with the accuracy being solely limited by the number of rules.

An example of a typical ANFIS is shown in Fig 49. For simplicity, the network features two inputs, both of which have two members, and one output. The rule base takes the same format as that utilised in a Sugeno type inference system. This means that the IF-THEN rules of the system can be expressed as shown in Eq. 24 . In this rule base, the values of A and B are the antecedent membership values and f_n is the crisp output of the system. During the training phase, which is discussed later in this chapter, the values of p, q and r are tuned for each of the output members. The functionality and output of each of the five layers are discussed in more depth below.

$$If\ X1\ is\ A_n\ and\ X2\ is\ B_n,\ then\ \mathrm{f}_n = \mathrm{p}_n X1 + \mathrm{q}_n X2 + \mathrm{r}_n \qquad \text{Eq. 24}$$

In the first layer of the network, which effectively performs the fuzzification, each of the nodes are adaptive and can be calculated using the formula given in Eq. 25 . The output of this node, O, is a value between 0 and 1 which denotes the extent to which input X satisfies the membership function A. The exact function utilised in this project will be a Gaussian type, as is given in Eq. 26 . The values of δ and c, known as the premise parameters, determine the exact shape of this function and are tuned during the training phase.

The next two layers in the network are fixed nodes which are used in conjunction with one another to determine the firing strength of the rule. The output of layer two is the product of all the incoming signals as is shown in Eq. 27 . Layer three then performs a normalisation on the output of the second layer as is shown in Eq. 28 .

The fourth layer in the network is an adaptive node which implements the rule base. The output of this node is as given in Eq. 29 where w_i is the output of layer three and p,q and r are the parameters of the node, known as the consequence parameters. During the training phase, these parameters are tuned to give optimal performance.

The final layer consists of one fixed node which gives the overall output of the ANFIS network. The output of this node is a weighted average of all incoming values as is shown in Eq. 30 .

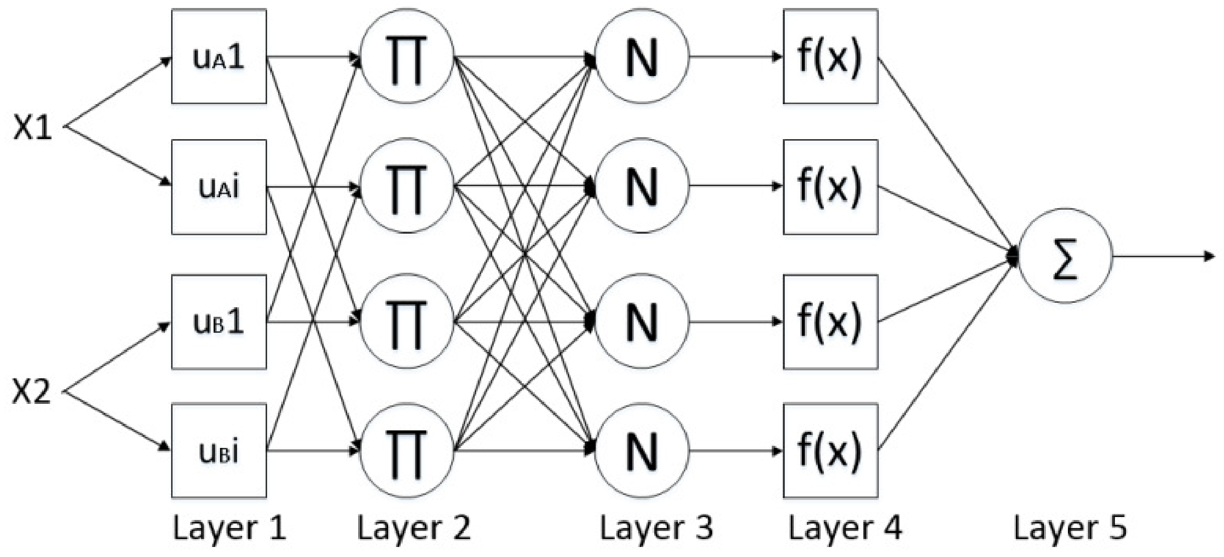


Fig 49 Example ANFIS Network Architecture

Layer One Outputs

$$O_{1,i} = \mu A_i(\mathrm{X}) \qquad \text{Eq. 25}$$
$$O_{1,i} = \mu B_i(\mathrm{X})$$

$$\mu A_i = \exp\left(\frac{-(\mathrm{X}-c)^2}{2\sigma^2}\right) \qquad \text{Eq. 26}$$

Layer Two Output

$$O_{2,i} = w_i = \mu_{Ai}(X)\mu_{Bi}(X) \qquad \text{Eq. 27}$$

Layer Three Output

$$O_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2} \qquad \text{Eq. 28}$$

Layer Four Output

$$O_{4,i} = \overline{w}_i f_i = \overline{w}_i(px_1 + qx_2 + r) \qquad \text{Eq. 29}$$

Layer Five Output

$$O_5 = \frac{\sum w_i f_i}{\sum w_i} \qquad \text{Eq. 30}$$

In order for the ANFIS to work optimally with any given system, the adaptive nodes in layers one and four must be trained. To facilitate this, a vector of inputs and expected outputs is created, known as the training data. An algorithm is then employed which aims to minimize the error between the actual and expected outputs. The training for the ANFIS network uses a hybrid learning approach, meaning separate algorithms are employed for each of the adaptive nodes. The result of this is a requirement for two distinct stages, with a forward pass phase consisting of feeding the data through the network and using the least squares algorithm to update the consequence parameters. The error is then propagated back through the network during a backward pass phase, with the premise parameters being updated using the gradient descent methodology. A high-level flow of the algorithm is shown in Fig 50 and Table 1.

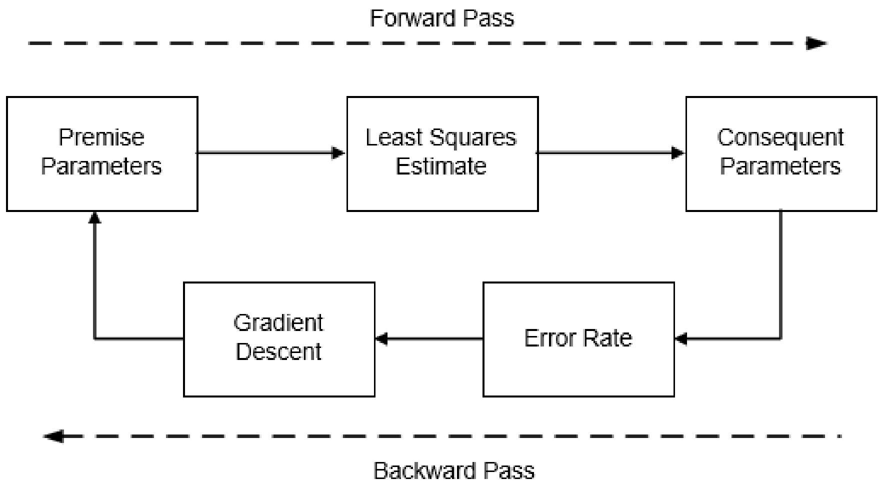


Fig 50 Flow Chart for the ANFIS Training Algorithm

	Forward Pass	Backward Pass
Consequent Parameters	Least Squares Estimate	Fixed
Premise Parameters	Fixed	Gradient Descent

Table 1 Parameter Settings for the Training Algorithm

During the first phase of training, the premise parameters are set to a fixed value in order to calculate the consequence parameters. Initially, these values are set in such a way that they uniformly cover the input membership space. These values are consequently updated during the backward pass phase but are always constant during subsequent forward pass iterations. This allows the input vector to flow through the network up to the fourth layer. At this point, the values of the consequence parameters can be calculated using the least squares estimate method. A simple model of the ANFIS is given in Eq. 31 (Jang (1993)) where I is the input vector and S is a vector of the network parameters. This vector consists of both the premise and consequence parameters, meaning it can be split into constituent parts – S1 (premise) and S2 (consequence). Given that the premise parameters are fixed during the forward pass phase, the fourth layer can be modelled with the equation given in Eq. 32 (Jang (1993)) where A and B are the input and output vectors respectively. The least squares estimate method can then be used to minimise the squared error of the layer, which is calculated as $\|A.S2 - B\|^2$. The most commonly used equation for the least squares estimate is given in Eq. 33 (Jang (1993)) where A^T is the transpose of the input matrix A.

$$out = F(I, S) \qquad \textit{Eq. 31}$$

$$B = AS_2$$ *Eq. 32*

$$S_2 = \left(A^T A\right)^{-1} A^T B$$ *Eq. 33*

Once the consequence parameters have been calculated, the output of the network can be calculated and the backward pass phase can begin. The premise parameters are calculated using the gradient descent method during this phase of the training. This algorithm alters the parameters by an amount that is proportional to the error rate. Assuming that the training dataset has P entries, the error can be calculated by taking the square of the error between the expected and the actual outputs. The overall error for the network can, therefore, be calculated using Eq. 34 (Jang (1993)) where m denotes the entry in the training vector, Y is the expected output and O^L is the real output. In order to utilise the gradient descent method, the error rate of the layers must be determined. If the dependent nodes are represented by the vector S and α is used to denote one of the parameters in the adaptive node, the error rate with respect to α can be calculated as in Eq. 35 (Jang (1993)). Using this information it is then possible to perform an iterative increment which is proportional to this rate, as is given in Eq. 36 (Jang (1993)) where η is a constant known as the learning rate. This coefficient determines how well the algorithm converges to the minimum error, with larger values resulting in faster tracking but less precision. The learning rate for the ANFIS network is typically defined as given in Eq. 37 (Jang (1993)) where k is a step size which determines the convergence rate of the algorithm.

$$E_p = \sum_{m=1}^{p} \left(Y_m - O_m^L\right)^2$$ *Eq. 34*

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha}$$ *Eq. 35*

$$\Delta \alpha = -\eta \frac{\partial E}{\partial \alpha}$$ *Eq. 36*

$$\eta = \frac{k}{\sqrt{\sum \alpha \left(\frac{\delta E}{\partial \alpha}\right)^2}}$$ *Eq. 37*

5.1.2.4.2 Modelling

The training of the ANFIS network can be achieved using the Neuro-Fuzzy toolbox in Matlab, allowing automation of the previously discussed hybrid learning algorithm. In addition to this, it is also possible to perform validation using this tool. Using this methodology, both the number and type of input members can be customised for each of the ANFIS networks. In the outer control loop, the two inputs are load current and the derivative of load current. For the inner control loop, the two inputs are the inductor current and the inductor current derivative. In both instances, the ANFIS network utilises 4 input members and consequently has 16 fuzzy rules. The PSO algorithm is used to generate a total of 110 sets of training data for each of the ANFIS blocks, with half of the sets being used to train the ANFIS and the remaining sets being used as validation data. The primary objective of the PSO algorithm was improving the response to a step change in load current with input disturbance rejection being of less concern. The performance of the system was considered over a period of 1 ms in each iteration of the PSO algorithm. Once the ANFIS network has been trained and validated, a model can then be generated and imported into Simulink for simulation within the full system.

5.1.3 Simulations

A number of simulations have been carried out on the solar PV subsystem model to prove the effectiveness of the control methodology. These simulations are intended to evaluate the performance of both the MPPT algorithm and the DC-DC regulator. The subsystem is tested under steady-state operating conditions and with fluctuations in irradiance. It should be noted that the switching frequency of the PWM signal is chosen as 1MHz for these simulations, to more thoroughly exercise the novel control solution which is utilised in the power converter topology. However, in a practical application, it is unlikely that such a high frequency would be utilised as the switching losses would be excessive.

When considering the performance of the MPPT load balancing algorithm, the most important criteria is the tracking accuracy. The worst-case error observed during simulation was slightly over 10%, as is shown in Fig 51. This occurs when the power being produced by the solar PV cell is low and is a result of the relatively fine step size which is required for the algorithm to work over its full operating range. However, this means that the power change caused by the disturbance can be discarded as noise when operating close to the MPP. In the rest of the results, the MPP is tracked with an accuracy of over 90% with the worst case error observed being 6%, as is shown by Fig 52. Similarly, there are several simulations which show errors of less than 1%, such as those shown in Fig 53 and Fig 54. A number of more sophisticated MPPT algorithms are available, such as those presented by Sellami et al (2016) and Yang et al (2017), which are shown to offer a tracking accuracy of over 95%. The simulation results given here show that the relatively simple algorithm employed offers similar performance under a majority of operating conditions. Whilst a more sophisticated algorithm could be employed to improve the accuracy, the primary benefit of this approach is that it removes the need for an additional power converter to achieve the MPP and reduces the overall complexity of the system.

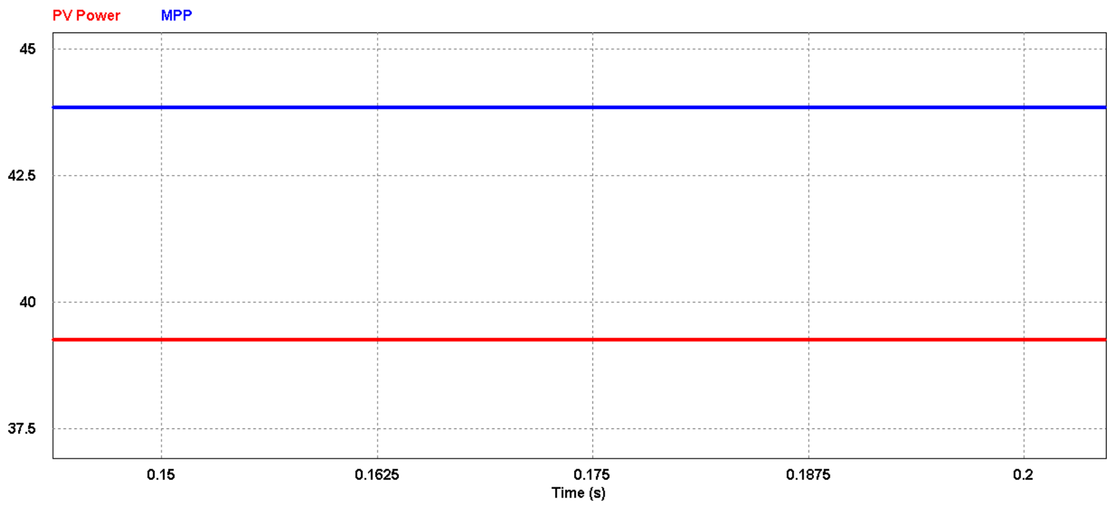


Fig 51 MPPT Tracking at 200 W/m^2 Steady State Showing the MPP (blue) and the Actual Output Power (red)

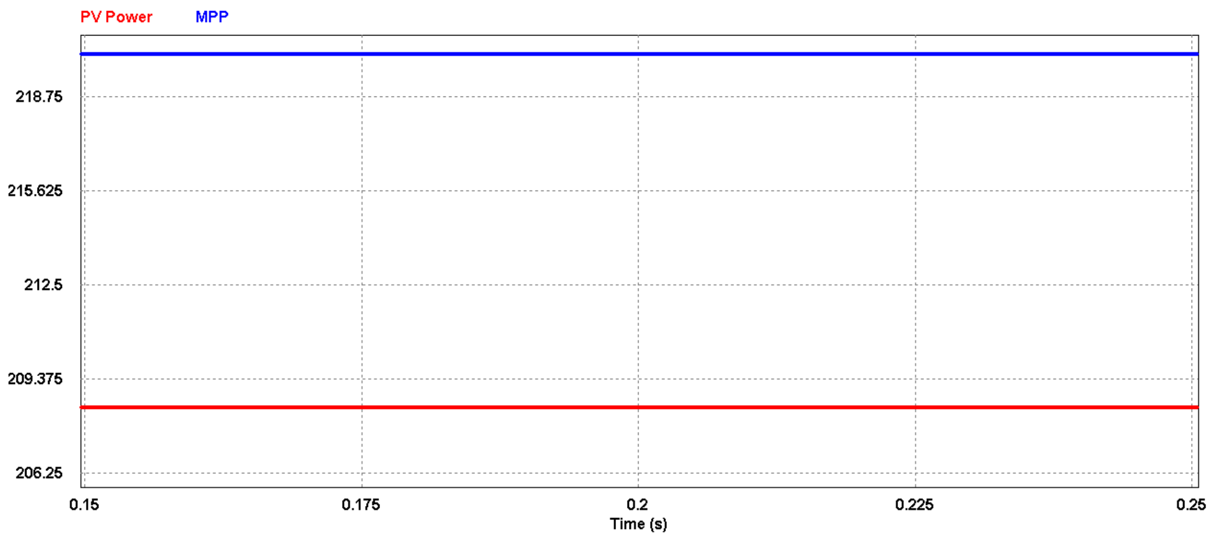


Fig 52 MPPT Tracking at 1000 W/m^2 Steady State Showing the MPP (blue) and the Actual Output Power (red)

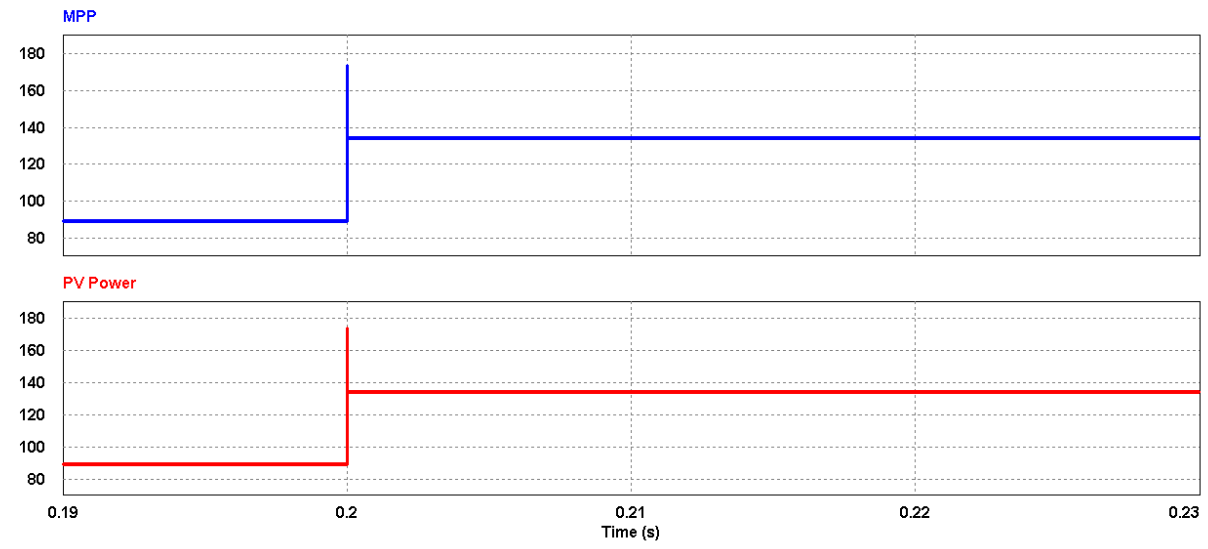


Fig 53 MPPT Tracking with 200 W/m^2 Step Increase showing the MPP (blue) and the Actual Output Power (red)

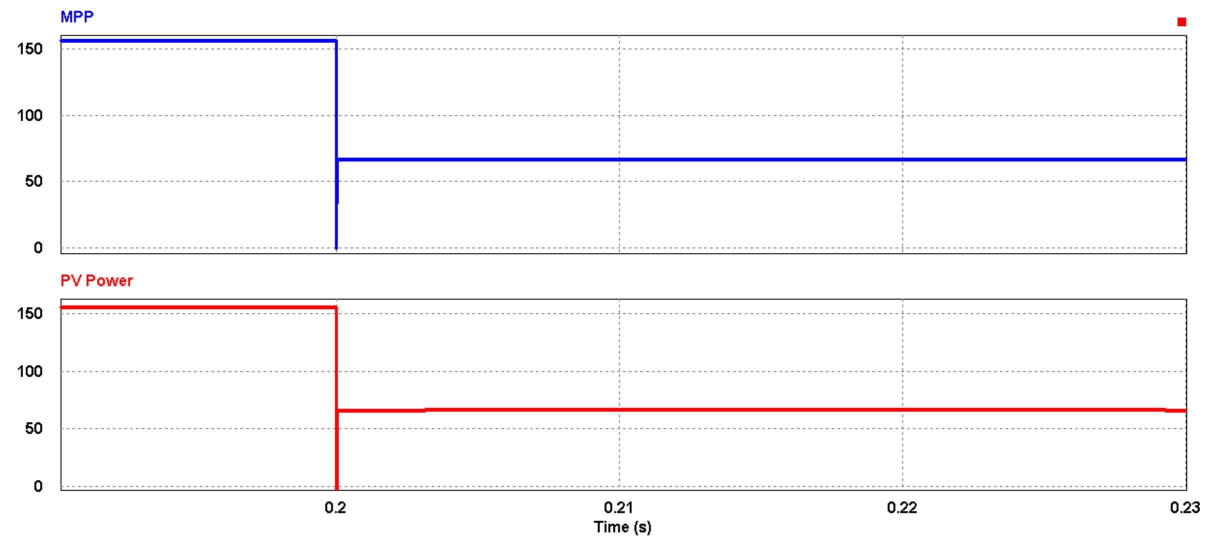


Fig 54 MPPT Tracking with 400 W/m2 Step Decrease showing the MPP (blue) and the Actual Output Power (red)

The simulations of the PV sub-system are also used to evaluate the performance of the DC-DC regulator. In order to establish the steady state characteristics of the system, the irradiance is set to a fixed value. Once the system has settled, the output voltage is then measured to determine the steady state error and the ripple voltage. This is repeated with a number of different irradiances to establish how the load affects the output. All of these simulations result in a steady state error of 0%. When the irradiance is set to 1000 w/m^2 and a heavy load applied, as is shown in Fig 56, ripple voltage of just 3mV is observed. Similarly, when the irradiance is set to 200 w/m^2, as is shown in Fig 55, the ripple voltage is less than 1mV. These simulations, the results of which are tabulated in Table 2, demonstrate there is a high degree of stability in the output voltage when the circuit is run in steady state.

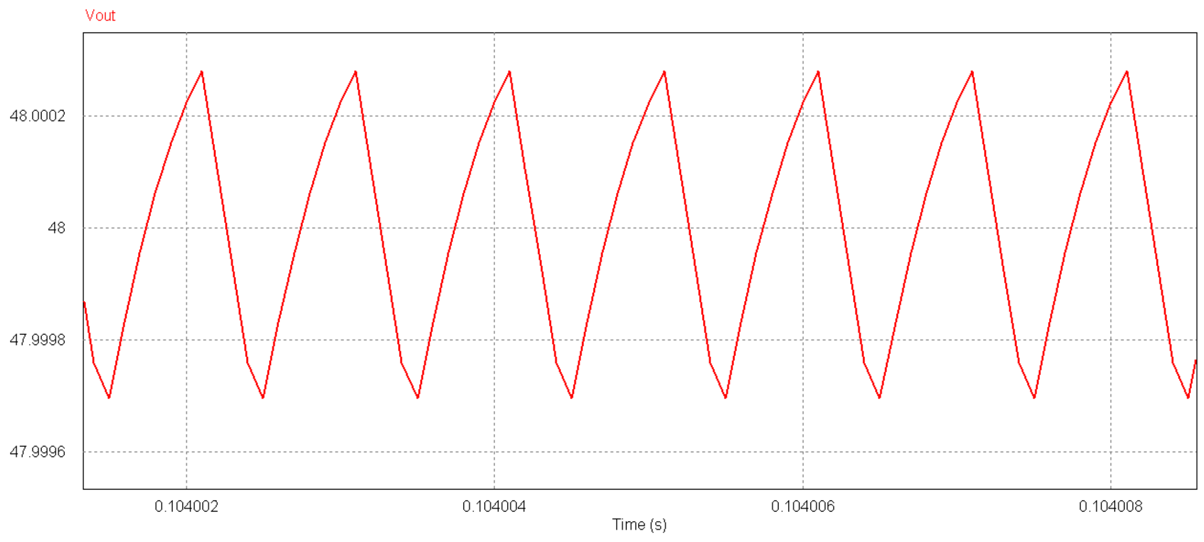


Fig 55 48VDC Regulator Output Voltage Ripple at 200 W/m^2 irradiance

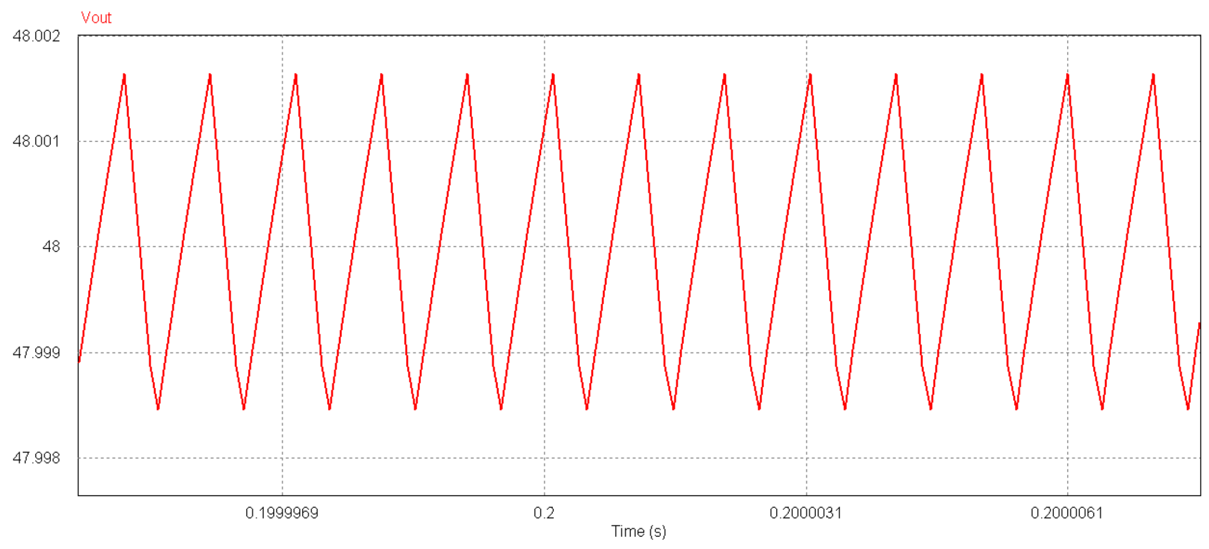


Fig 56 48VDC Regulator Output Voltage Ripple at 1000 W/m^2

Irradiance	Peak to Peak Ripple (mV)	Steady State Error (%)
200 W/m^2	800µV	0
4000 W/m^2	2mV	0
800 W/m^2	2.5mV	0
1000 W/m^2	3mV	0

Table 2 48VDC Regulator Steady State Performance Summary

The dynamic characteristics of the system must also be established to ascertain the performance of the novel control solution. In order to do this, the output voltage is observed when there is a step increase or decrease in solar irradiance. The output voltage can then be measured to determine the maximum error and the settling time. The results of these simulations illustrate that the observable error is between 13 mV (0.027 %), as shown in Fig 57, and 75 mV (0.156%), as shown in Fig 58. Similarly, the settling time varies from a maximum of 45 µs to a minimum of just 20 µs. These characteristics, which are tabulated in Table 3, demonstrate that the novel control solution is capable of achieving fast response times and a high degree of accuracy.

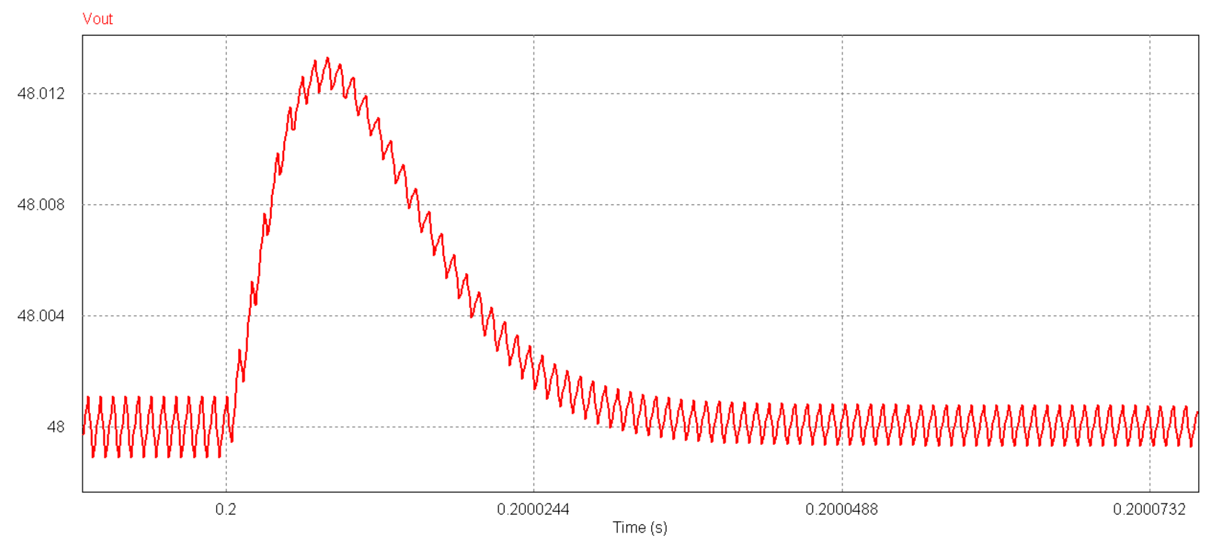


Fig 57 48VDC Regulator Output Voltage with a 200 W/m^2 Step Decrease

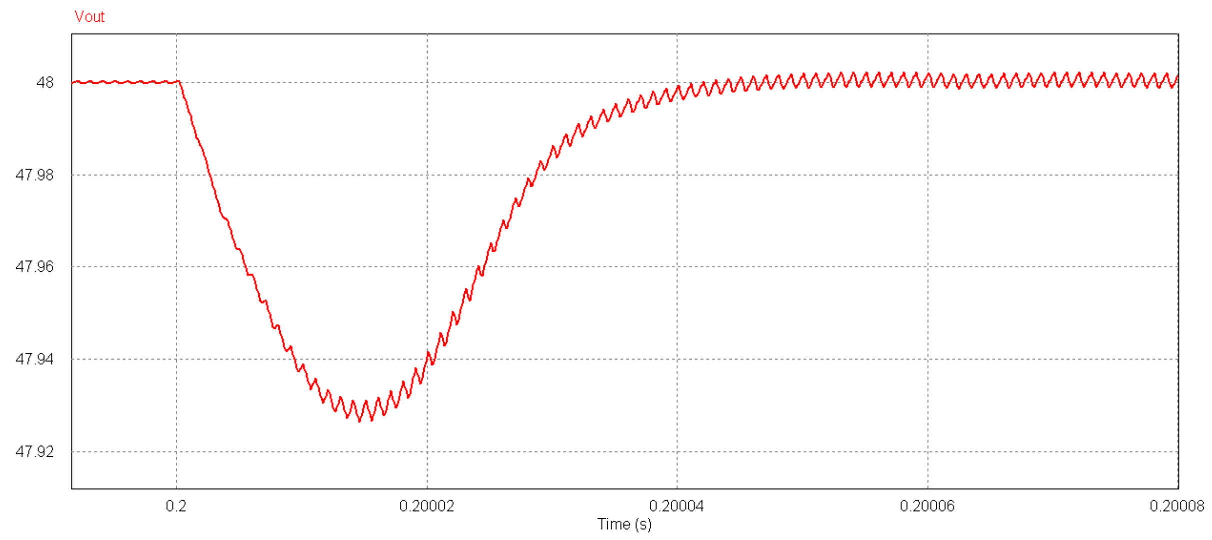


Fig 58 48VDC Regulator Output Voltage with an 800 W/m^2 Step Increase

Load	Max Error		Settling Time (μs)
	mV	%	
200 W/m^2 Increase	17	0.035	25
200 W/m^2 Decrease	13	0.027	20
400 W/m^2 Increase	40	0.83	40
400 W/m^2 Decrease	24	0.05	30
800 W/m^2 Increase	75	0.156	40
800 W/m^2 Decrease	53	0.11	45

Table 3 48VDC Regulator Dynamic Performance Summary

A final important consideration of the PV sub-system is the performance of the boost converter soft-start circuit. This is important as it provides a safeguard against potential damage to the converter caused by the in-rush current at startup. The waveform shown in

Fig 59 illustrates that the boost converter has a controlled startup sequence, with the voltage slowing increasing to prevent damage to the converter components.

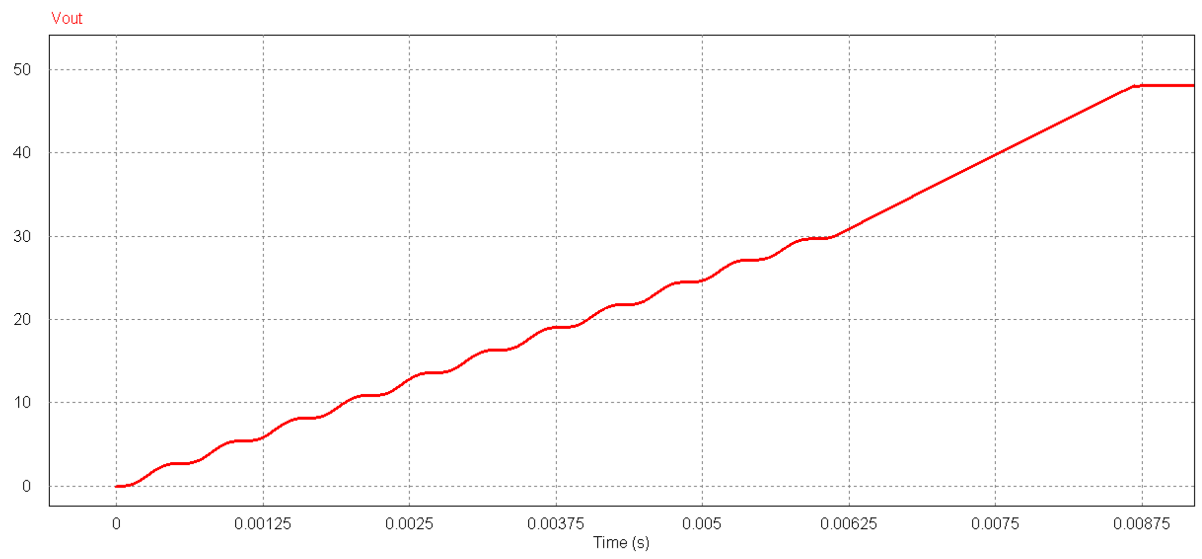


Fig 59 Simulation of the Boost Converter Soft Start Showing Regulator Output Voltage

5.2 Wind Power Converter

In this section, the design and modelling of the power converter and associated controller for the wind turbine subsystem are discussed and simulation results are presented.

5.2.1 Topology

In order to interface with the main 48VDC bus, it is necessary to employ a boost type converter to regulate the power output at the required level. The inductor and the capacitor can be chosen using the equations shown in Eq. 7 and Eq. 8 . These values are selected as 22uH and 330uF, giving a maximum inductive ripple current of 550mA and a maximum ripple voltage of 4mV. The control of the regulator utilises the same methodology as the 48VDC boost regulator in the PV subsystem, which was discussed at length in section 5.1.1.1.

5.2.2 Modelling

The wind turbine used in this system model is an Ampair 100 device, which is a six blade turbine featuring a permanent magnet synchronous machine (PMSM) with an operating range of 3 to 12 m/s. An inbuilt regulator is included within the selected turbine, meaning the power is generated at a nominal voltage of 24VDC, giving a maximum output current of approximately 5A. As PSim features pre-defined models of these components, this part of the system is simple to model as is shown in Fig 60.

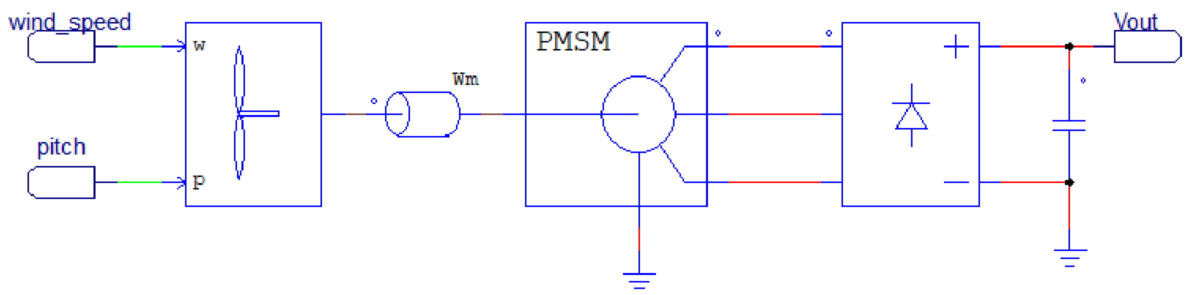


Fig 60 Wind Turbine Model

As has been previously discussed, the renewable energy system in this project has primarily been chosen to demonstrate the capabilities of the novel ANFIS-PI control algorithm. Therefore, the wind turbine model presented in Fig 60 has been simplified through the inclusion of a mechanical load which regulates the speed of rotation. The mechanical load is varied using a LUT which determines the rotational speed based on the given wind speed. In a practical system, it would not be possible to implement this methodology and an additional regulator and controller would be required to control the generator output. In addition to this, it should also be noted that the inductance of the stator is considered negligible in this model meaning that the current steps up or down with little opposition.

The output power generated by the wind turbine is determined using the equation given in Eq. 38 (Powersim (2017)), where A is the area of the blades, v_{wind} is the wind speed in m/s, ρ is a constant which gives an approximation of the air density and C_p is a power coefficient. This coefficient can be calculated using the equation shown in Eq. 39 (Powersim (2017)), where β is the blade pitch angle and λ is the tip speed ratio. The equation given in Eq. 40 (Powersim (2017)) shows how the tip speed ratio can be calculated, with ω_m being the rotational speed of the rotor and R_{blade} being the radius of the blade in metres.

$$P = \frac{1}{2}.A.{v_{wind}}^{3}.\rho.C_p \qquad \textit{Eq. 38}$$

$$C_p = 0.5.(116.\lambda - 0.4\beta - 5).e^{21\lambda} + 0.01\lambda \qquad \textit{Eq. 39}$$

$$\lambda = \frac{\omega_m.R_{blade}}{v_{wind}} \qquad \textit{Eq. 40}$$

The modelling of the PI controllers and ANFIS network follows the same methodology employed in the PV system, as was detailed in sections 5.1.2.3 and 5.1.2.4. In the outer control loop, the ANFIS uses load current and the derivative of load current as inputs. In the inner control loop, inductor current and its derivative are used as the inputs. In both cases, four input members are used which results in a total of 16 different fuzzy logic rules. The PSO algorithm is used to generate a total of 120 sets of training data, with half of these being used to train the algorithm and half to validate its performance. As discussed in section 5.1.2.4.2, the primary objective of the PSO was improving the step response of the converter. Each iteration of the PSO uses a simulation of 1 ms to determine the performance of the algorithm. Once the networks are trained, they are then exported into Simulink so that they can form a part of the system model.

5.2.3 Simulation

In order to characterise the performance of the wind subsystem, a number of simulations have been carried out. As has been stated in section 5.2.2, several simplifications have been made to the model of the wind turbine to better demonstrate the capability of the ANFIS-PI based control algorithm. As part of this model, an assumption was made that the inductance of the generator windings in the stator and the cables could be considered negligible. This means that the current transients occur without any opposition, acting in a manner which is nearer to an ideal power source. In addition to this, a 1MHZ switching frequency has been utilised in the power converter, which would not be practically realisable due to the switching losses at such a high frequency. However, this value has been chosen for the simulations since the novel control solution is the major focus of this work and a higher switching frequency more thoroughly exercises this area of the design. The final assumption which is made as part of these simulations is that the generator is producing more power than is being utilised by the load. This allows for step changes in the load to be made, allowing for the dynamic characteristics to be ascertained.

In order to establish the steady state characteristics of the system, the load is set to a fixed value. The converter output is then allowed to settle before the output voltage is observed to determine both the steady-state error and the voltage ripple. This is repeated with a load current of 1A and 2A. In both instances, the observed steady-state error is 0%, as is shown in Fig 61 and Fig 62, whilst the ripple voltage is measured at 1mV and 2.5mV. These simulations, the results of which are tabulated in Table 4, demonstrate a high level of stability during steady state operation of the circuit.

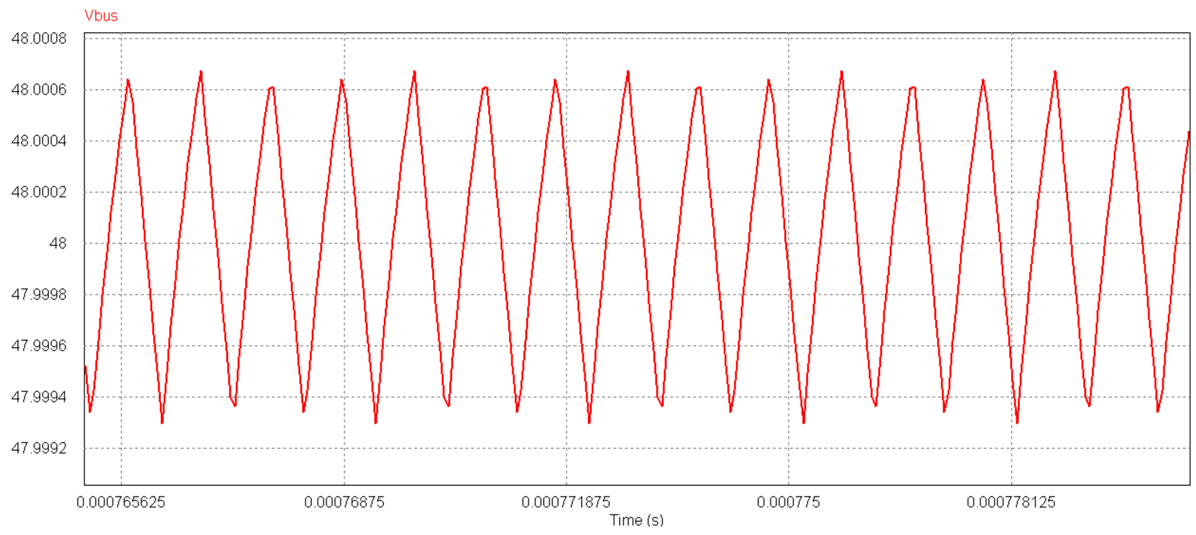


Fig 61 Wind Boost Converter Output with 1A Load

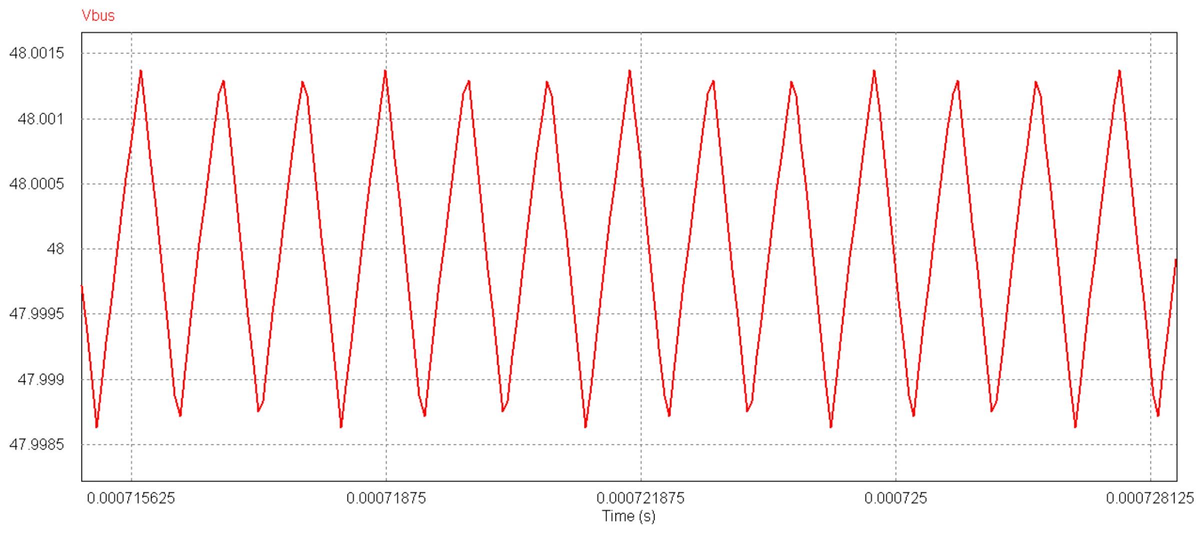


Fig 62 Wind Boost Output Converter with 2A Load

Load	Peak to Peak Ripple (mV)	Steady State Error (%)
1A	1	0
2A	2.5	0

Table 4 Wind Converter Steady-State Performance

Simulations have also been carried out with load steps of 500mA and 1.5A in order to ascertain the dynamic characteristics of the converter. The output is measured to determine the maximum error and the settling time. The results of these tests, which are shown in Fig 63 to Fig 66, illustrate that the observed absolute error is between 7.5mV (0.0156%) and 35mV (0.0729%). Similarly, the settling time varies from a minimum value of 13 μs to a maximum of 31 μs. These characteristics, which are tabulated in Table 5, demonstrate that the novel control solution is capable of delivering a high level of accuracy and a quick settling time.

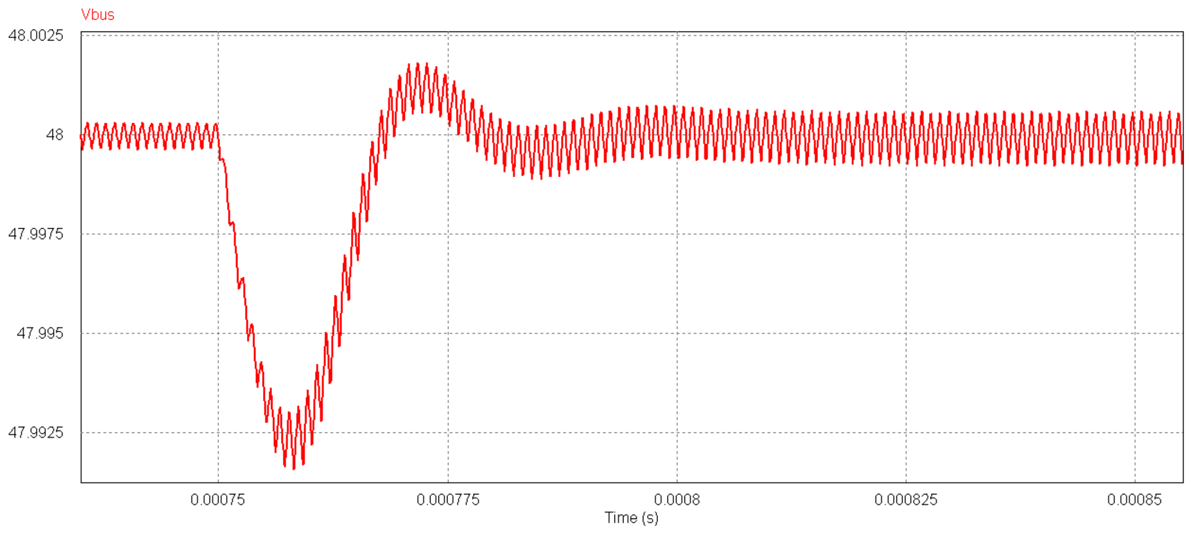


Fig 63 Wind Converter Output with 500mA Step Increase in Load

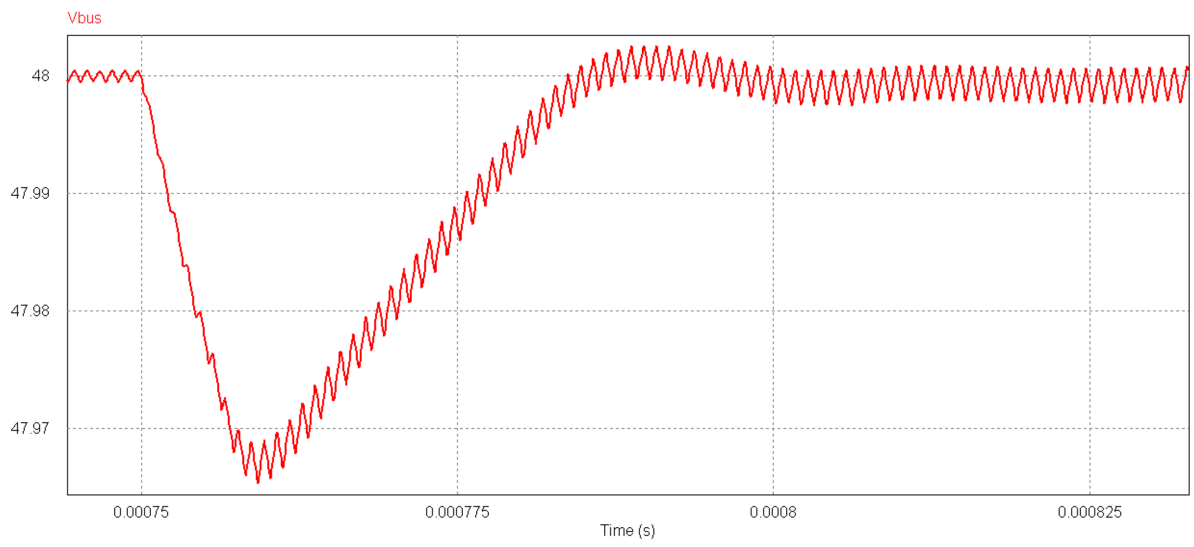


Fig 64 Wind Converter Output with 1.5A Step Increase in Load

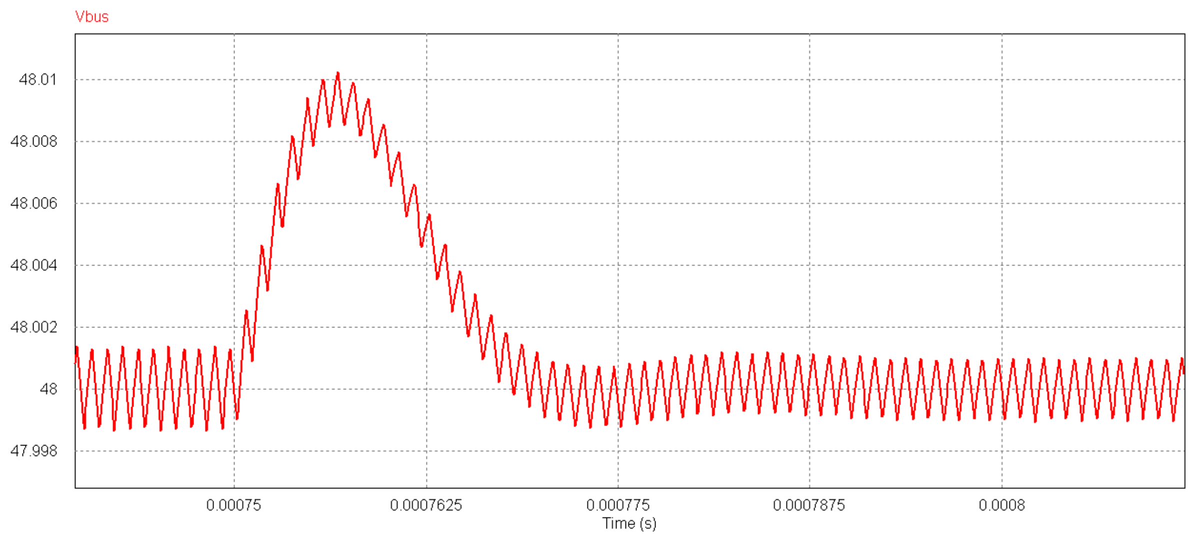


Fig 65 Wind Converter Output with 500mA Step Decrease in Load

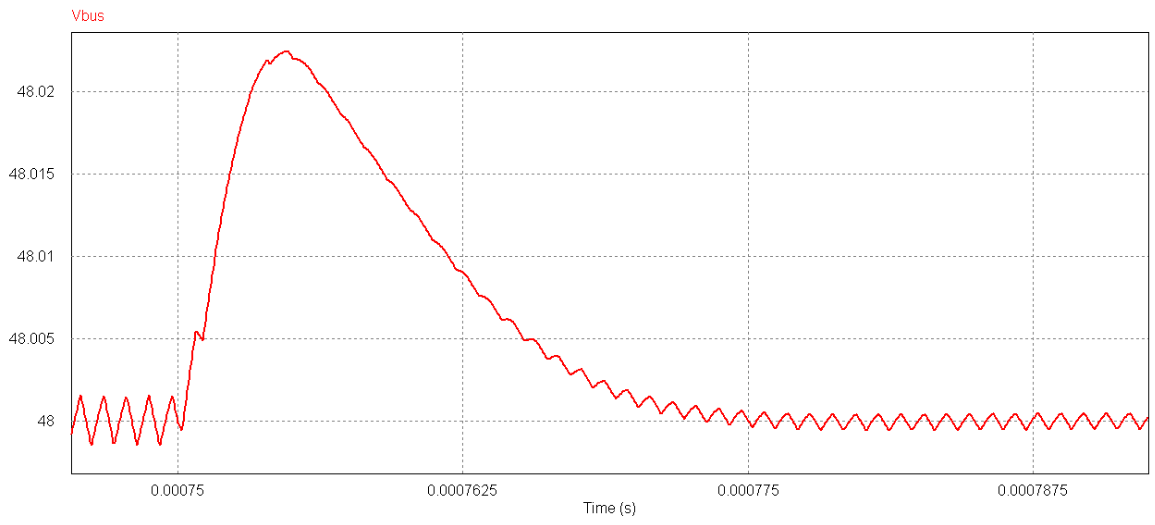


Fig 66 Wind Converter Output with 1.5A Step Decrease in Load

Load	Max Error		Settling Time (us)
	mV	%	
500mA Increase	7.5	0.0156	13
500mA Decrease	10	0.0208	13
1.5A Increase	35	0.0729	31
1.5A Decrease	22	0.045	16

Table 5 Wind Converter Dynamic Performance Summary

5.3 Super Capacitor Power Converter

The supercapacitor subsystem uses a pair of Maxwell BMOD0058 E016 devices, each of which has a maximum voltage of 16V. In the rest of this section, the design and modelling of the power converter and associated controller are discussed.

5.3.1 Topology

The two supercapacitors will be connected in series, giving a maximum voltage of 32VDC. This means that a boost converter is required to interface the supercapacitor with the 48VDC output bus. The supercapacitor must additionally sink power from the renewable power sources to maintain its state of charge, meaning that a buck regulator is also required. The full topology for the supercapacitor subsystem, including the control solutions, is shown in Fig 67.

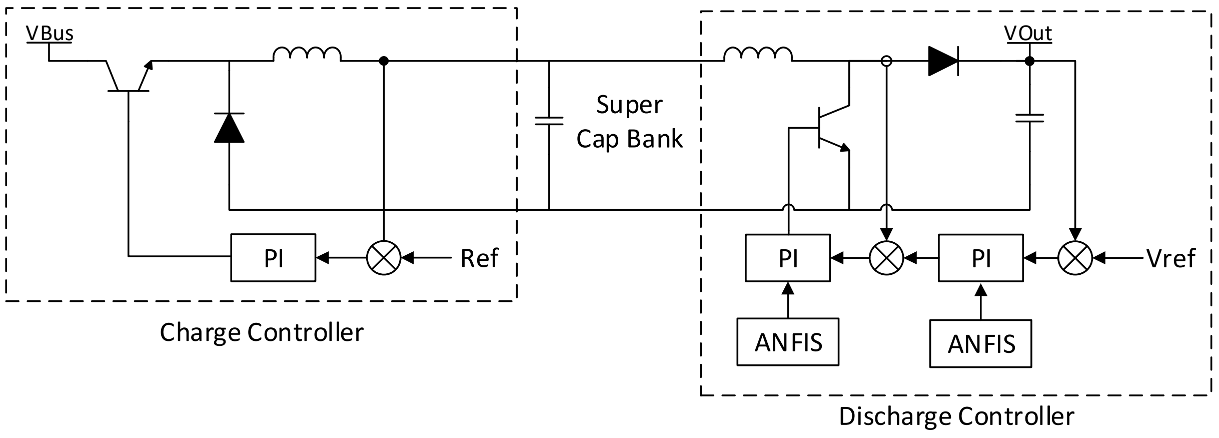


Fig 67 Super Capacitor Sub System

When considering the boost converter circuit in Fig 67, the capacitor and inductor can be selected based on the equations given in Eq. 7 and Eq. 8 . In this case, the inductor is chosen as 22uH whilst the output capacitance is chosen as 2200uF. This gives a maximum output ripple voltage of around 2 mV and a maximum ripple current of 480mA. The equations used to determine the value of capacitance and inductance in the buck converter circuit are given in Eq. 41 and Eq. 42 . The inductor for the buck converter is again chosen as 22uH which also results in a maximum ripple current of 480mA. However, as this circuit is sourcing power to the supercapacitor bank it is unnecessary to include an additional output capacitor in this case.

$$L_{\min} = \frac{D(\mathrm{V}_{\mathrm{in}} - \mathrm{V}_{\mathrm{out}})}{F_s I_{\min}} \qquad \text{Eq. 41}$$

$$C = \frac{I_{out} D}{F_s \Delta V_{out}} \qquad \text{Eq. 42}$$

5.3.1.1 Super Capacitor Charging

The charging of the supercapacitor is split into two phases – constant current and constant voltage. In the first phase of charging, when the capacitor voltage is much lower than the nominal value, the constant current mode is active. This means that the controller must regulate the current which is delivered into the supercapacitor. The exact rate of charge depends on the available power from the renewable energy sources. When the capacitor voltage reaches a level which is close to the nominal level, the charge controller switches to constant voltage mode. When this occurs the power converter starts to regulate the super capacitor voltage. This means that the voltage is held at 32V, with the current slowly decreasing down to zero as the device is charged. Once the supercapacitor is fully charged, the charger algorithm ceases operating and power is no longer sunk into the device. Fig 68 gives a generalised view of the full supercapacitor charge profile. Starting from empty, a constant current is applied until a nominal voltage level is reached. When this happens, the supercapacitor is then charged with a constant voltage.

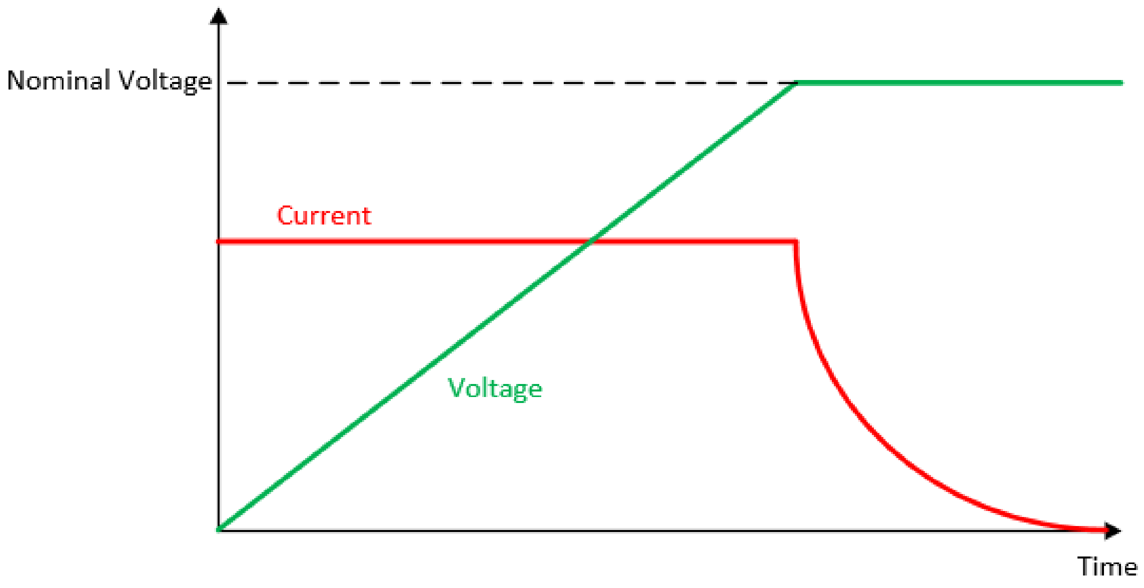


Fig 68 Super Capacitor Charge Profile

The charging algorithm utilises a simple PI-type controller. In the constant current mode, the PI controller is responsible for regulating the charge current. Once the capacitor voltage reaches approximately the nominal voltage, the PI controller starts to regulate the capacitor voltage. Once the charge current becomes very small, the PI controller is reset and the capacitor charging ceases. This mode is then maintained until the supercapacitor is discharged below 30V. The state diagram in Fig 69 shows how the supercapacitor sub-system switches between the different charging modes.

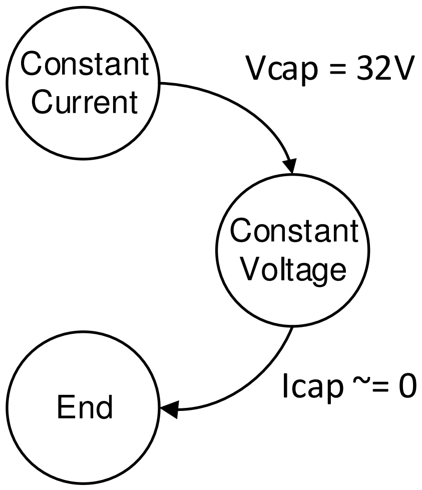


Fig 69 Super Capacitor Charger States

5.3.1.2 Super Capacitor Discharge

The controller which regulates the supercapacitors 48VDC output bus will be provided by a PI controller which has gain scheduling provided by an ANFIS algorithm. This methodology is the same as that presented and discussed in section 5.1.1.1.

5.3.2 Modelling

The modelling of the supercapacitor can be split into three distinct sections – the supercapacitor model, the charge controller and the discharge controller. In the rest of this section, each of these aspects are discussed in more detail.

5.3.2.1 Super Capacitor Model

A number of different equivalent circuits have been proposed for supercapacitor modelling, ranging from a simplistic classical model, which considers only the device ESR and capacitance, through to complex multi branch circuits which consider a much larger array of parameters. The use of a model consisting of three parallel RC branches has proven to be a popular choice, providing a good balance between complexity and accuracy as shown by Weddel et al (2011), Ge et al (2017) and Zubieta and Bonert (2000). Therefore, the model presented by Miller et al (2003), as is shown in Fig 70, will be utilised. In this approach, the three branches represent the fast (C_f, R_f), medium (C_m, R_m) and slow (R_s, C_s) temporal characteristics of the supercapacitor device. The leakage current of the device is also considered through the addition of the resistor R_L. This equivalent circuit is implemented within PSim, owing to the simplicity of creating SPICE based models with this tool.

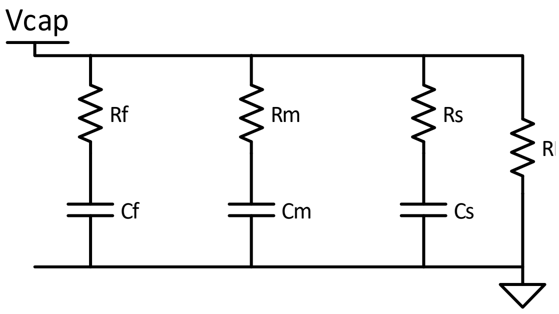


Fig 70 Three Branch Super Capacitor Model

A number of different methods have been proposed for the calculation of the parameters in the supercapacitor model. These methods tend to be experimental in nature, requiring a number of different test scenarios to establish the different characteristics of the device. Whilst good accuracy can be achieved using these experimental results, Miller et al (2003) present a faster approach whereby the parameters can be determined from the system datasheet. Table 6 shows the equations used to determine the parameters in the model, where N is the number of cells in the supercapacitor module, Co is the nominal capacitance of the supercapacitor, whilst Φ, k and j are constants with values of 8, 2 and 0.618 respectively. The Maxwell BMOD0058 E016 supercapacitor is comprised of six Maxwell BCAP0350 cells, which have a nominal capacitance of 350F and an ESR of 3.2mΩ. Using this information, it is possible to calculate the component values as shown in Table 7.

Fast		Medium		Slow		Leakage	
R_f	$\frac{2N}{3}ESR$	R_m	$\frac{2N}{3}\phi^{-(2k-1)}ESR$	R_s	$\frac{2N}{3}\phi^{-(2k+1)}ESR$	R_L	$\frac{NV_r}{I_{leak}}$
C_f	$\frac{1.05}{N}C_o$	C_m	$\frac{1.05}{N}\phi^{2j+1}C_o$	C_s	$\frac{1.05}{N}\phi^{2j-1}C_o$		

Table 6 Super Capacitor Parameter Selection Equations (Miller et al (2003))

Fast		Medium		Slow		Leakage	
R_f	13mΩ	R_m	17.45Ω	R_s	45.7Ω	R_L	680Ω
C_f	61F	C_m	5.5F	C_s	14.45F		

Table 7 Super Capacitor Calculated Parameters

5.3.2.2 Charge Controller

There are two considerations for the modelling of the charge controller – the tuning of the PI controller and the implementation of the state machine shown in Fig 69. The tuning is simple to achieve using the PSO algorithm which was previously discussed in section 5.1.2.3. The PI controller will firstly be tuned for constant current mode and then for constant voltage mode. One of the main considerations for stable control is the transition point between the constant current and voltage modes. In order to help stabilise this transition, the tuning of the voltage phase will be done as it enters this mode from the constant current phase. This means that the supercapacitor will be pre-set to a value just below the nominal voltage. The tuning is then carried out at this transition point.

The required state machine is simplistic in nature, consisting of only three states. There are a number of different methodologies which could be used for this task but the method employed here utilises the C compiler within PSim. This methodology is chosen as it improves portability whilst the simplicity of the state machine means that the coding is straightforward.

5.3.2.3 Discharge Controller

The modelling of the discharge controller follows the same methodology presented in sections 5.1.2.3 and 5.1.2.4. The outer control loop utilises load current and the derivative of load current as the inputs to the ANFIS network. In the inner control loop, inductor current and its derivative are utilised instead. In both cases, four input members are used which results in a total of 16 different fuzzy logic rules. The PSO algorithm is used to create a total of 200 sets of training data, with half of these being used for training and half for validation. As discussed in section 5.1.2.4.2, the main consideration of the PSO algorithm is improving the response of the converter when subjected to a step change in load current. In each iteration of the PSO algorithm, a simulation time of 1 ms is used when calculating the MSE, as defined in Eq. 19 . Once the ANFIS networks have been fully trained and validated, models are generated which can be imported directly into Simulink for incorporation into the system model.

5.3.3 Simulation

The supercapacitor subsystem model has been simulated to prove the performance of the different system elements under a number of operating conditions. The simulations are split into two sections – the charging circuit and the discharge controller – both of which are discussed in this section.

5.3.3.1 Charge Controller Simulation

Although the charge controller doesn't represent a novel approach, its operation is integral to the performance of the overall system. Therefore, a number of simulations have been carried out to verify its performance. In the first set of tests, the constant current mode of operation is evaluated with currents of 1A, 8A and 15A, as is shown in Fig 71 to Fig 73, and there is no steady state error observed. A simulation is then carried out where the supercapacitor is taken from a fully discharged state through a full charging cycle. The waveforms obtained from this are shown in Fig 74, verifying that the constant current mode continues until the voltage reaches a level of approximately 32V. This voltage is then maintained as the current drops down to zero, as is expected and previously outlined in Fig 68.

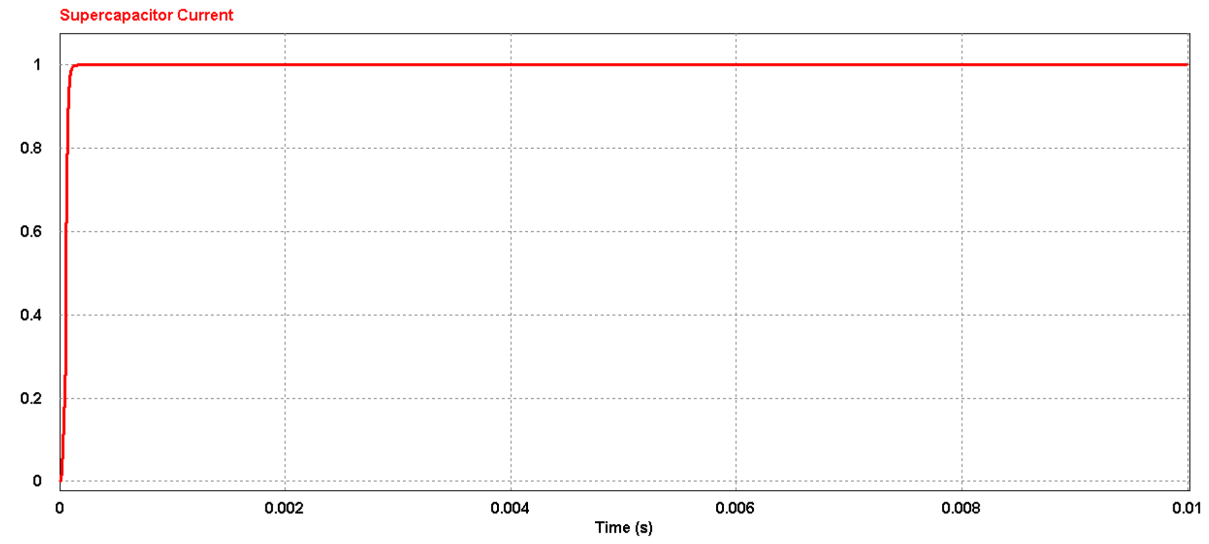


Fig 71 Supercapacitor Constant Current at 1A

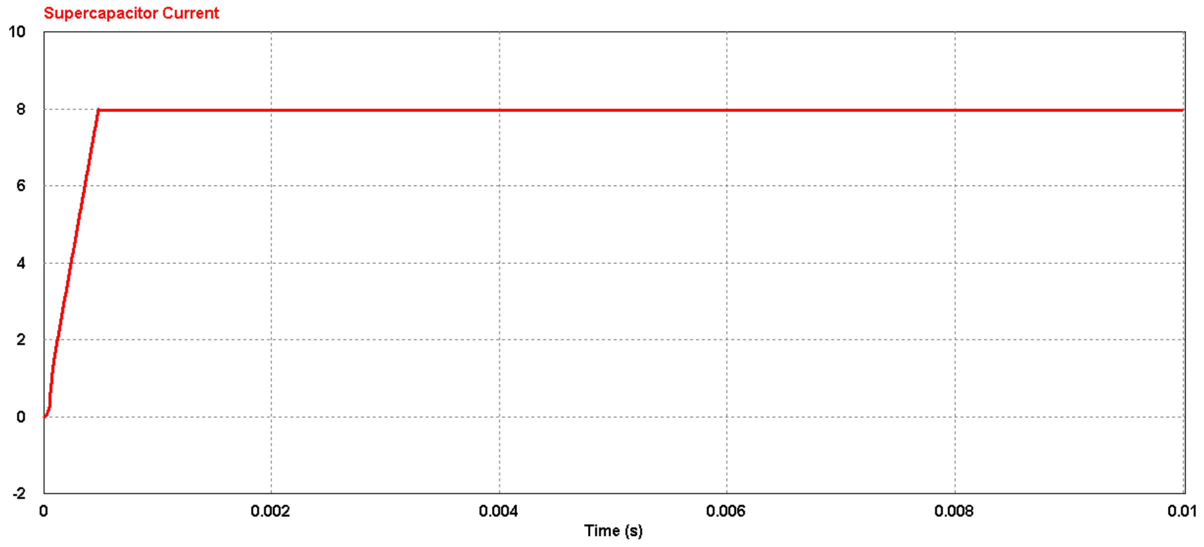


Fig 72 Supercapacitor Constant Current at 8A

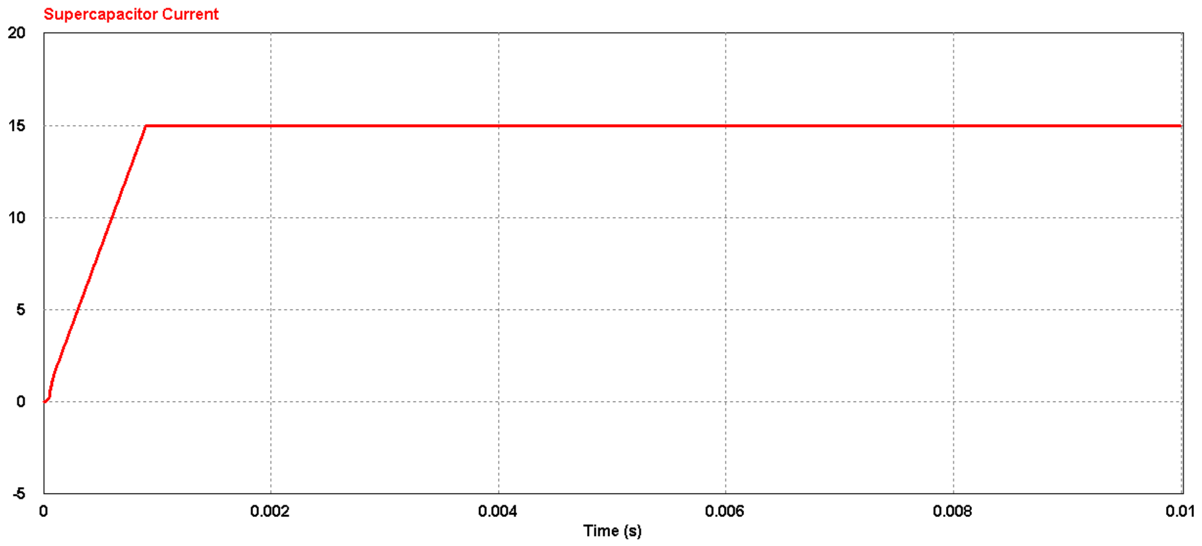


Fig 73 Supercapacitor Constant Current at 15A

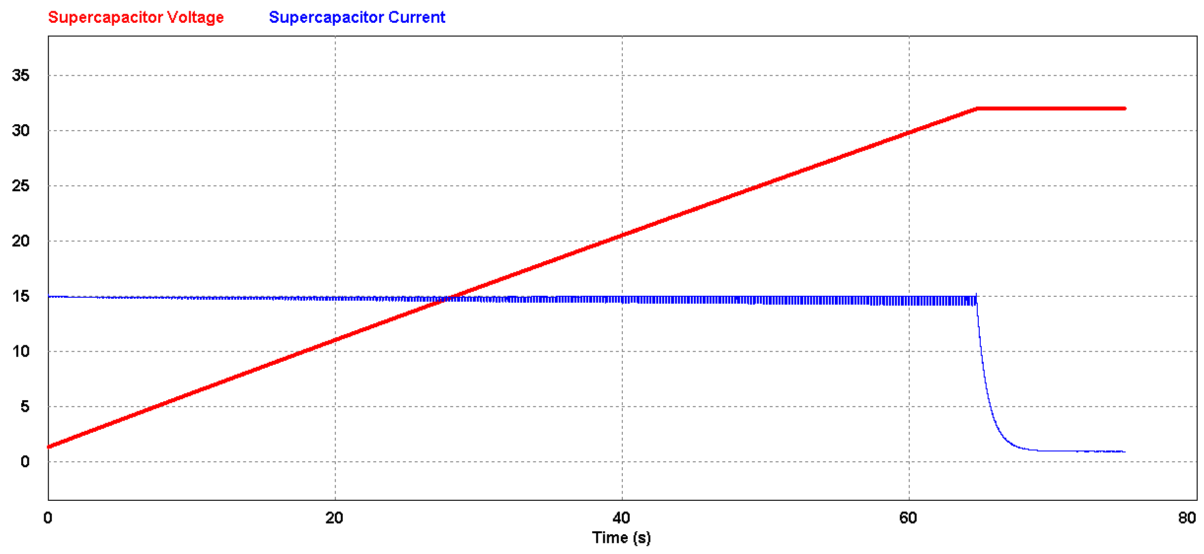


Fig 74 Supercapacitor Charge Curve at 15A

5.3.3.2 Discharge Control Simulation

In order to characterise the performance of the supercapacitor discharge controller, a number of simulations have been conducted. One important point in these simulations is that the switching frequency of the power converter has been selected as 1MHz. In a practical application, a lower frequency would likely be utilised as the switching losses are proportional to switching frequency. However, this decision has been made as it better demonstrates the capabilities of the ANFIS-PI control solution which is one of the main accomplishments of this project.

To assess the steady-state performance, the load is set to a fixed value and the steady state error and ripple voltage are measured. This is repeated with loads of 1A, 2.5A and 4A. The results of these tests, as shown in Fig 75 to Fig 77, indicate that the steady-state error is 0% in all instances whilst the ripple voltage is just 1mV. The results of these simulations are tabulated in Table 4, showing how the controller is able to deliver both accuracy and stability in the steady state mode of operation.

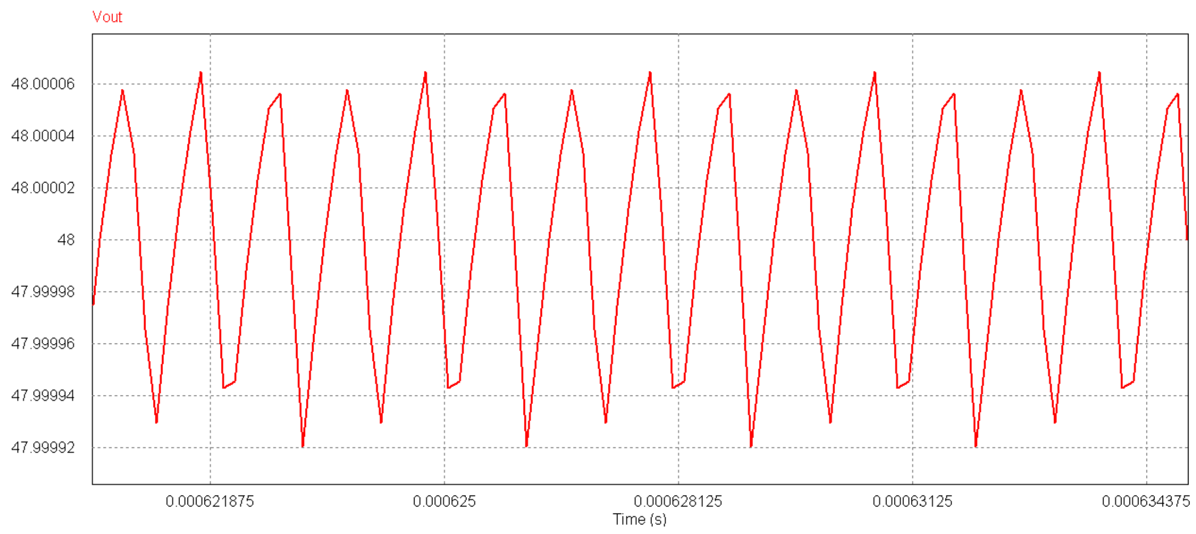


Fig 75 Super Capacitor Discharge with a Steady 1A Load

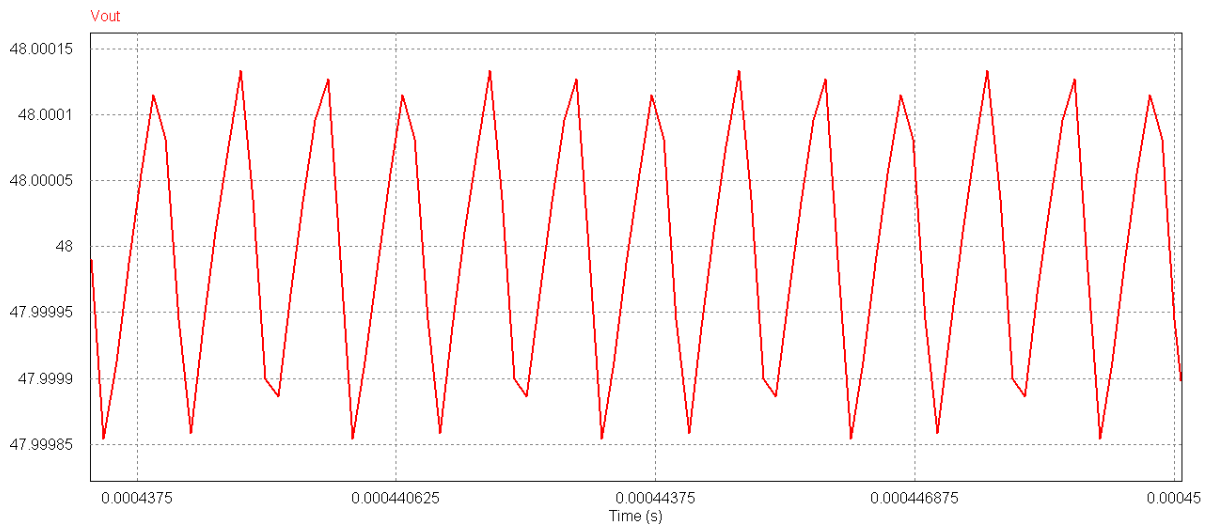


Fig 76 Super Capacitor Discharge with a Steady 2A Load

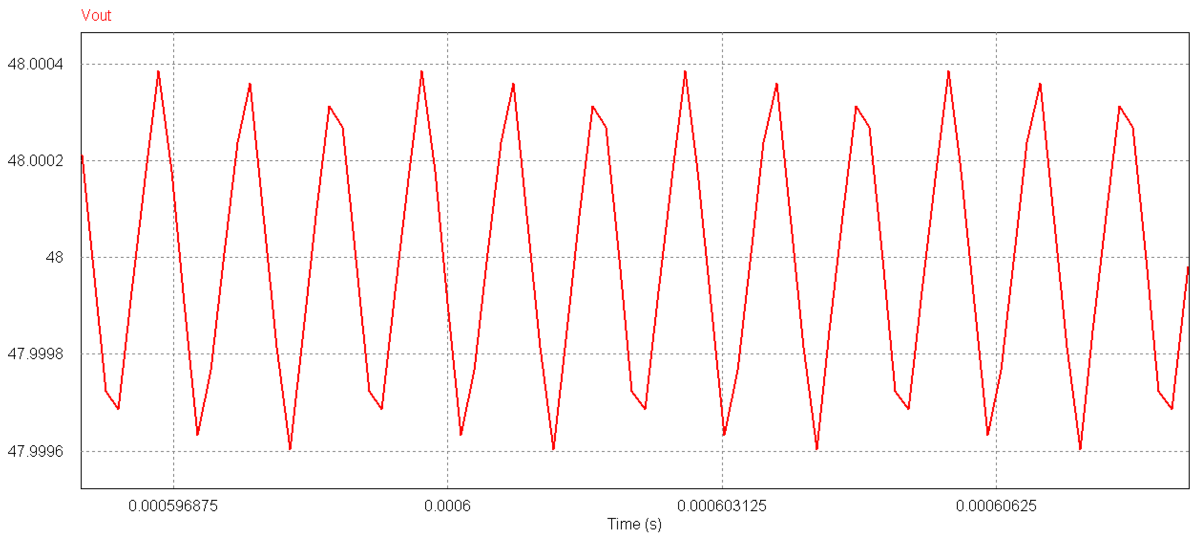


Fig 77 Super Capacitor Discharge with a Steady 4A Load

Load	Peak to Peak Ripple (mV)	Steady State Error (%)
1A	1	0
2A	1	0
4A	1	0

Table 8 Super Capacitor Steady-State Performance

In order to evaluate the performance of the model under dynamic operating conditions, it is necessary to conduct a number of simulations with fluctuations in the load current. A total of six simulations have been carried out, with load steps of 1A, 2A and 4A. The output voltage can then be measured to determine the maximum error and the settling time. The results of these tests illustrate that the observed absolute error is between 3.5mV (0.0073%), as is shown in Fig 78, and 15mV (0.038%), as shown in Fig 79. The results of these simulations, which are summarised in Table 9, also indicate that the maximum observable settling time is just 138 μs.

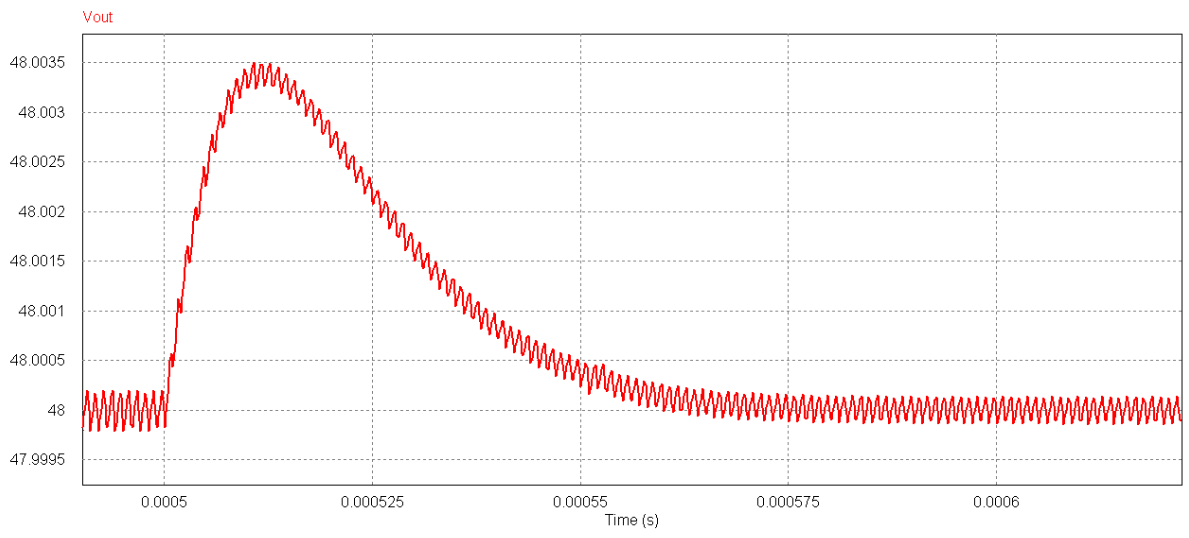


Fig 78 Super Capacitor Discharge with a 1A Load Step Decrease

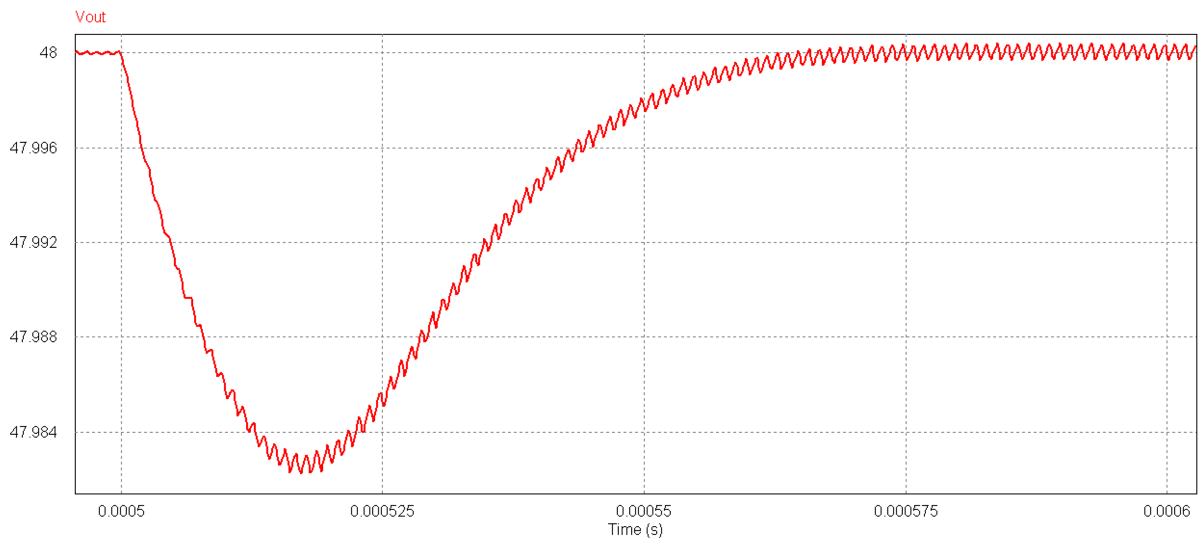


Fig 79 Super Capacitor Discharge with a 4A Load Step Increase

Load	Max Error		Settling Time (µs)
	mV	%	
1A Increase	4.2	0.0088	N/A
1A Decrease	3.5	0.0073	N/A
2A Increase	9	0.0188	44
2A Decrease	7.5	0.0156	21
4A Increase	18	0.038	43
4A Decrease	15	0.031	138

Table 9 Super Capacitor Dynamic Performance Summary

5.4 Battery Power Converter

The battery subsystem is a vital component of the DER system as it provides a means of storing energy. This is crucial as it allows for excess energy to be stored during times of high production from the renewable sources and then utilised during periods of insufficient power production. The battery is, therefore, critical to ensuring the stability of the power produced by the DER system. In this system, the battery power is supplied by a pair of 12VDC lead acid batteries connected in series to give a nominal 24VDC. In the rest of this section, the modelling and simulation of this element is discussed.

5.4.1 Topology

As the battery needs to be able to both source and sink current, it is necessary to have a bidirectional power converter to interface between the battery and the main DC bus. As the battery voltage will always be lower than the main bus, the basic power converter presented in section 2.2.6 can be utilised. There will be two controllers for this circuit which operate depending on the battery state – one for charging and one for discharging. A simple PI controller is utilised to charge the battery, whilst the discharge controller consists of the gain scheduling ANFIS-PI methodology as discussed in section 5.1.1.1. A diagram of this circuit, including the control solutions, is given in Fig 80.

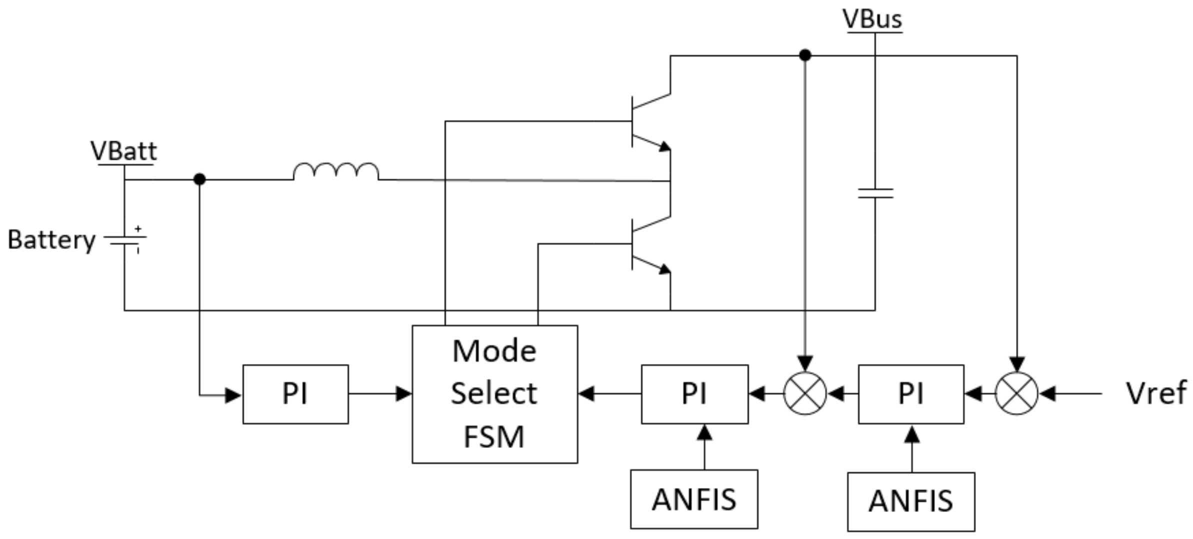


Fig 80 Battery Subsystem Circuit

As previously mentioned, the components for the boost mode of operation can be selected using the equations given in Eq. 7 and Eq. 8 . Similarly, the components for the buck can be selected using the equations given in Eq. 41 and Eq. 42 . Using these equations, the inductor is selected as 22uH, giving a maximum ripple current of 480mA. The boost side capacitor is selected as 470uF, which gives a maximum voltage ripple of 6 mV. On the buck side of the converter, the power quality is less important as on the output bus side. For this reason, a small 10uF output capacitor is included purely to improve stability.

5.4.1.1 Battery Charging

The battery charging can be split into three separate phases – constant current, constant voltage and float. The first phase occurs when the battery is deeply discharged and the controller must regulate the amount of current which flows into the battery. Once the battery voltage reaches 29V, the charging mode switches to constant voltage. Subsequently, the battery voltage is maintained and the charge current slowly decreases. The final phase of the charging process occurs when the charge current drops to a level of 1A or less. When this scenario occurs, the power converter is switched off and the battery charging is complete. Once in the floating mode of operation, the charging of the battery will only begin again when the battery has sufficiently self-discharged. An example of the full charge profile for a lead acid battery is shown in Fig 81.

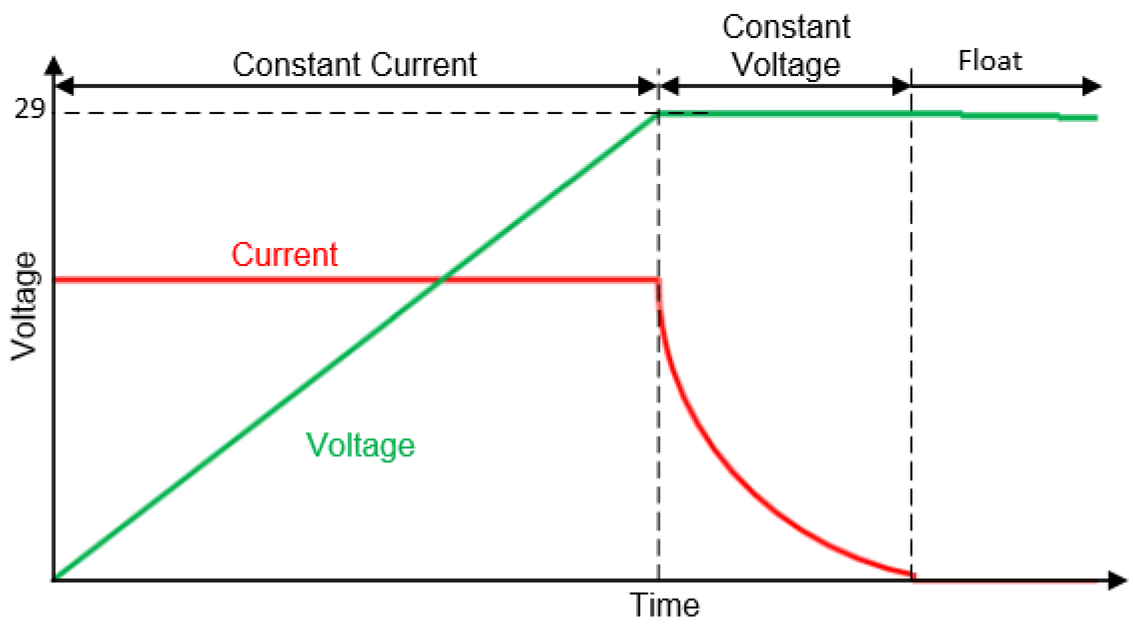


Fig 81 Battery Charging Profile

As was previously discussed, the charging of the battery can be achieved through the use of a simple PI controller. During the first phase of the charging process, the charge current is regulated by the PI. The exact current sunk into the battery during this stage is dependent on the available power in the system. Once the battery voltage reaches 29V, the PI controller then begins to regulate the battery voltage rather than the charge current. This mode of operation is continued until the charge current reaches 1A at which point the charger enters the float mode. In this mode of operation, the controller is disabled and the battery is fully disconnected from the voltage bus. The charging controller is also reset whenever the battery starts to discharge. The state diagram in Fig 82 shows how the battery sub-system switches between each of the three charging states, as well as the discharging state.

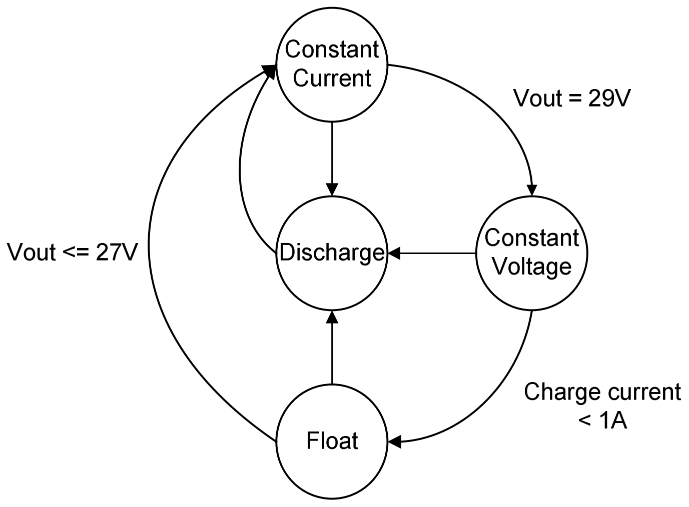


Fig 82 Battery Charger State Machine

5.4.1.2 Battery Discharging

When the renewable energy sources are unable to supply sufficient power to the loads, the battery is used to source power. When this occurs, the power converter operates in the boost mode of operation, with power flowing from the battery into the main DC bus. The controller for this mode of operation is provided by a pair of PI controllers with gain scheduling provided by an ANFIS network. This methodology is the same as that presented and discussed in section 5.1.1.1 for the solar PV subsystem regulator.

5.4.2 Modelling

As with the previously discussed power converters, the modelling of the battery subsystem is split between the PSIM and Matlab simulation environments. The PSim software is utilised for the power electronics, PI controller and battery, whilst Matlab is used for the ANFIS algorithm. As PSim features a number of predefined battery devices, the modelling of the battery is achieved using a drag and drop component and doesn't warrant further discussion. Therefore, the rest of this section details the modelling of the controllers used in the battery subsystem.

5.4.2.1 Battery Charger

The charge controller features a state-dependent PI controller, with the gain being determined by the active charging mode. This means there are two elements which must be considered when modelling the battery charger – the tuning of the PI controllers and the implementation of the state machine. The first of these tasks is achieved using the PSO algorithm which was discussed in detail in section 5.1.2.3, with the current controller being tuned first. An important consideration for stable control is the transition between the constant current and voltage modes. Therefore, the tuning of the voltage controller is performed at this transition point meaning the battery voltage will be pre-set to a value just below the nominal voltage.

The state machine associated with the battery charger has previously been outlined in Fig 82. This is a simple machine, consisting of only four states, which will be modelled using the C compiler which is included in PSim. Whilst there are a number of potential solutions for the state machine model, this method is chosen as it improves portability of the code, in particular to CPU cores. As the state machine consists only of four states, the coding of the FSM is also straightforward. Simulations of the battery charger model, including the tuned PI controller and the FSM, are shown in section 5.4.3.1.

5.4.2.2 Discharge Controller

The modelling of the discharge controller follows the same methodology presented in sections 5.1.2.3 and 5.1.2.4. The ANFIS network in the outer control loop utilises load current and the derivative of load current as its inputs, whilst the inner loop utilises inductor current and it's derivative. Both of these networks use four input members, requiring a total of 16 logic rules per ANFIS. The PSO algorithm is used to create a total of 110 training sets for each network, with half being used for training and half being used for validation. As discussed in section 5.1.2.4.2, the main consideration of the PSO algorithm is improving the response of the converter when subject to a step change in load current. A simulation of 1 ms is used in each iteration of the PSO algorithm in order to calculate the MSE, as is defined in Eq. 19 . Once the two networks have been fully trained and validated, they are exported into Simulink for incorporation into the overall system model.

5.4.3 Simulation

In order to prove the operation of the battery subsystem, a number of simulations have been carried out on the model under differing operating conditions. These simulations are split into two different sections – the charge controller and the discharge controller - and are presented in this section.

5.4.3.1 Charge Control Simulations

The battery charger performs a vital function within the system, even though it contains no novelty. This necessitates the verification of its performance through simulation. The first set of tests, which are shown in Fig 83 and Fig 84, are intended to evaluate the performance during constant current operation, with a reference current of 1A and 10A. The second set assesses the constant voltage operating mode with the same reference currents, as shown in Fig 85 and Fig 86. In all cases, the simulations result show how the controller exhibits 0% steady state error, highlighting the stability which is achieved by the controller.

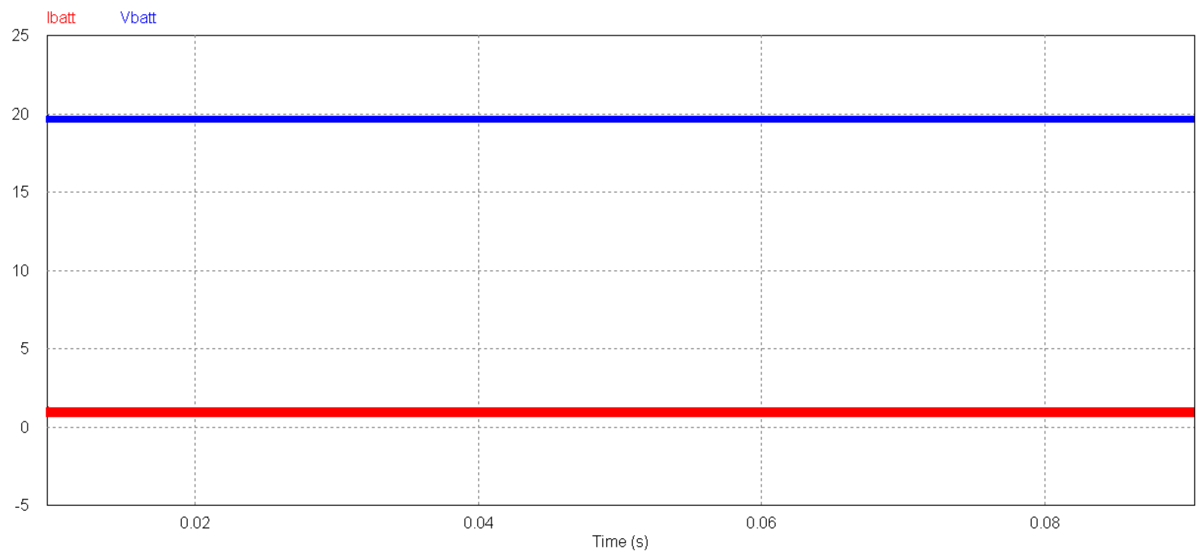


Fig 83 Battery Charge Constant Current at 1A

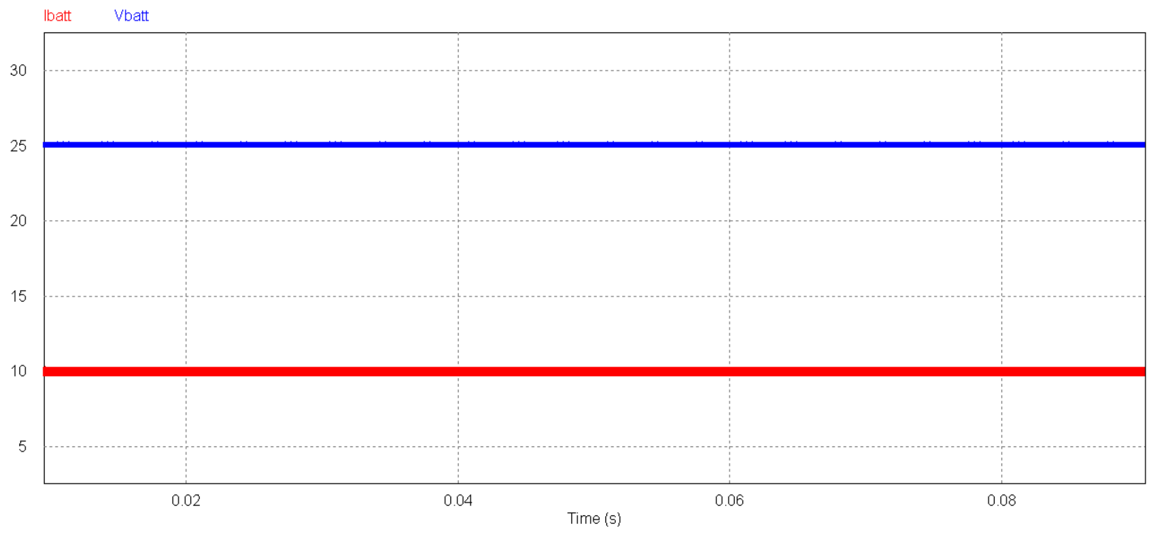


Fig 84 Battery Charge Constant Current at 10A

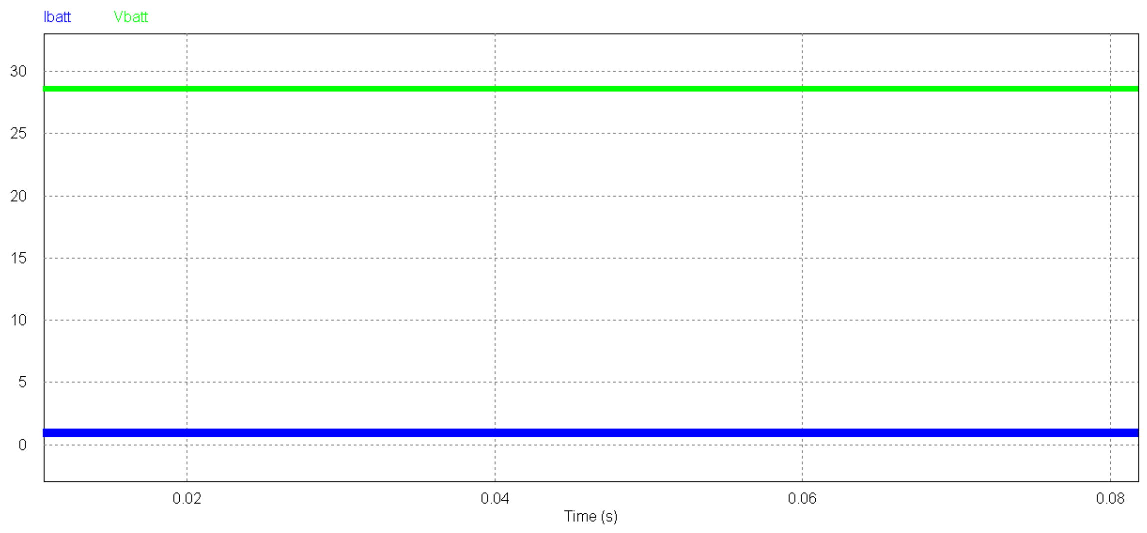


Fig 85 Battery Charge Constant Voltage at 1A

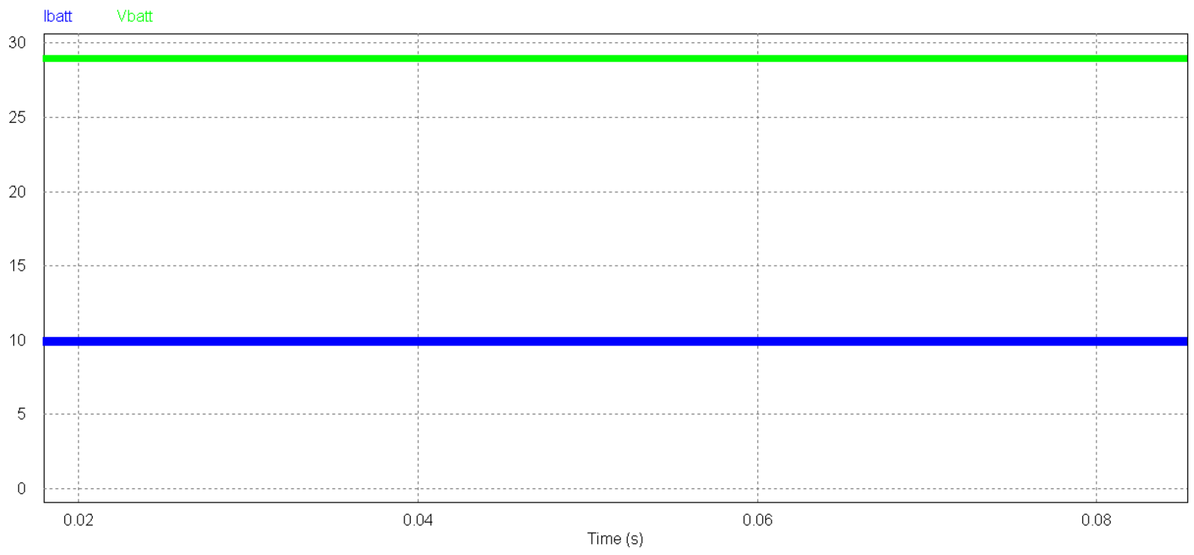


Fig 86 Battery Charge Constant Voltage at 10A

5.4.3.2 Discharge Controller Simulation

In order to characterise the battery discharge controller, simulations have been conducted under both dynamic and steady-state operating conditions. In these simulations, the PWM switching frequency is selected as 1MHz to more thoroughly exercise the ANFIS-PI control algorithm. In a real application, this would be likely to lead to excessive switching losses meaning it is unlikely such a high frequency would be employed. However, this approach has been taken to more thoroughly exercise the developed ANFIS-PI control solution which is one of the major achievements presented in this work.

To assess the steady state, fixed loads of 1A, 2A and 4A are utilised. Once the output has settled, the steady-state error and ripple voltage are then measured. The results of these tests are tabulated in Table 10, showing that the steady state error is 0% in all instances. Furthermore, the ripple voltage varies from a minimum of 1mV at 1A, as shown in Fig 87, to a maximum of 3mV at 4A, as shown in Fig 88.

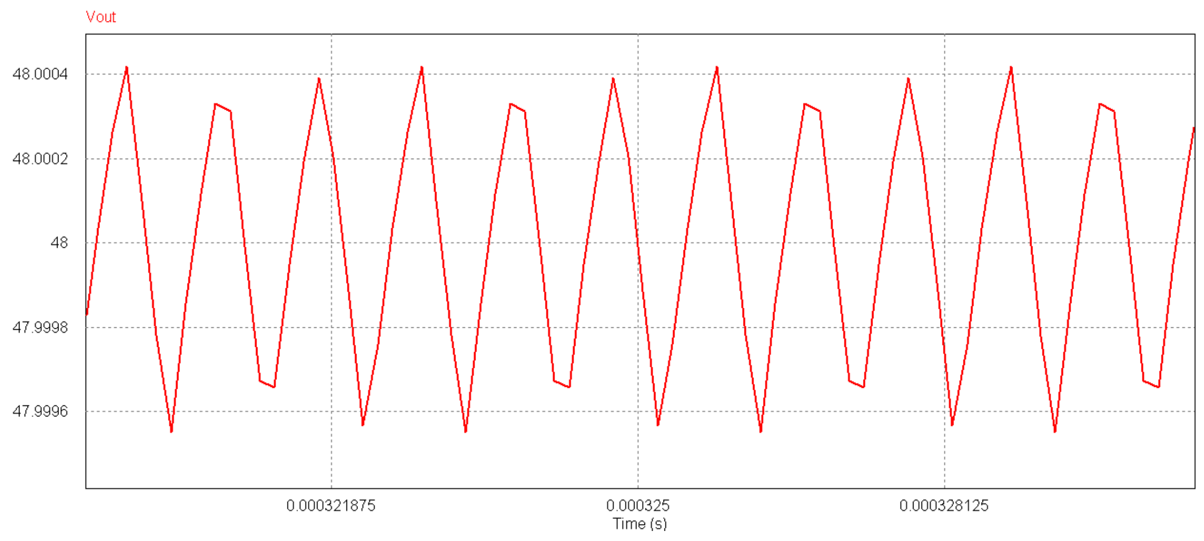


Fig 87 Battery Discharge Controller with a 1A Load

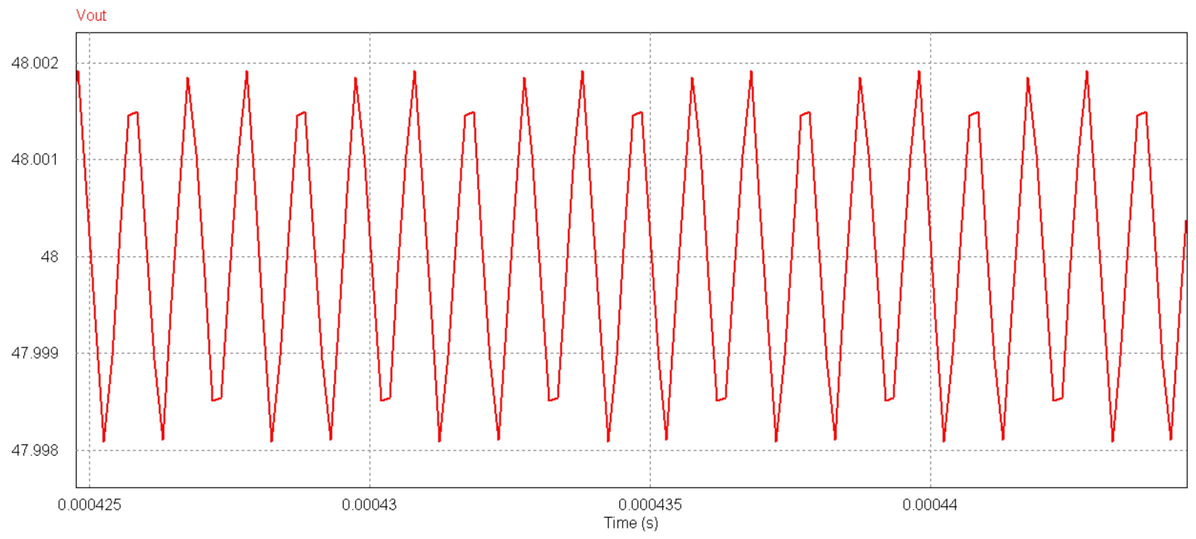


Fig 88 Battery Discharge Controller with a 4A Load

Load	**Peak to Peak Ripple (mV)**	**Steady State Error (%)**
1A	1	0
2A	1	0
4A	3	0

Table 10 Battery Discharge Steady-State Performance

In order to evaluate the performance of the battery system under dynamic operating conditions, simulations have been carried out with steps of 1A, 2A and 4A in load current. The output voltage can then be measured to determine the maximum error and the settling time of the output. The results of these tests illustrate that the observed absolute error varies from a minimum of 16mV (0.033%), as shown in Fig 89 to a maximum of 90mV (0. 188%), as shown in Fig 90. The results of these simulations, which are summarised in Table 11, also indicate that the maximum observable settling time is just 148 µs.

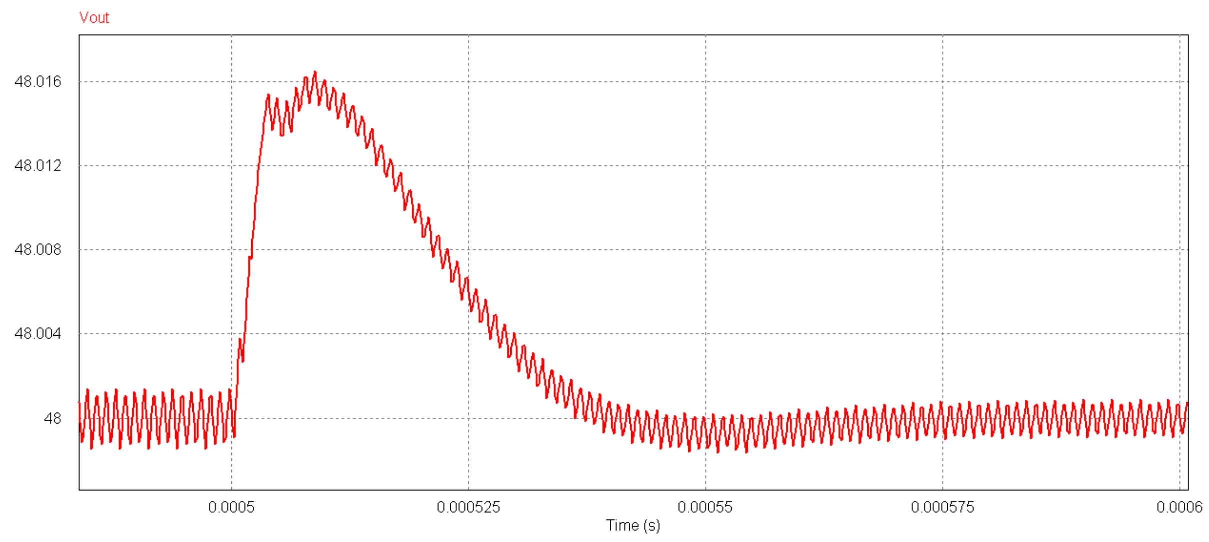


Fig 89 Battery Discharge with a 1A Load Step Decrease

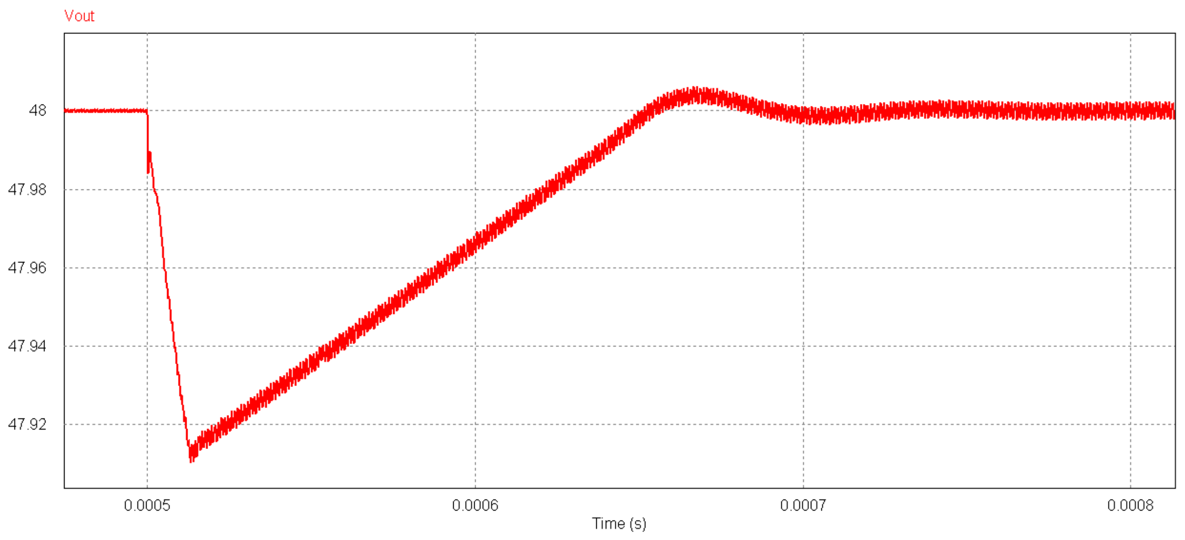


Fig 90 Battery Discharge with a 4A Load Step Increase

Load	Max Error		Settling Time (µs)
	mV	%	
1A Increase	21	0.043	34
1A Decrease	16	0.033	28
2A Increase	38	0.079	40
2A Decrease	33	0.069	27
4A Increase	90	0.188	145
4A Decrease	82	0.177	104

Table 11 Battery Dynamic Performance Summary

5.5 Conclusions

In this chapter, the overall topology of the system is briefly presented before the modelling of the individual power converters is discussed in detail. This incorporates a discussion of the power electronics, renewable energy sources and the individual control loops. Simulations have been carried out to assess the performance of each of the power converters, using co-simulation between Matlab and PSIM.

The main accomplishment presented in this chapter is the creation of a novel power converter control solution. This approach utilises the ANFIS algorithm to improve the non-linear performance of the PI controller. This methodology is based on the FLC-PI based approach which has been presented in the literature. The use of the ANFIS algorithm creates an improved design flow, removing the need for expert knowledge during the training phase. This is optimised further through the utilisation of the PSO algorithm for the tuning of the PI controllers. The simulation results which are presented show absolute errors of between 3.5mV and 90 mV during a load transient, whilst the maximum settling time observed is just 145 µs. These results attest to the validity of the novel control solution, showing that this approach yields a high level of stability and a fast response time.

6 Modelling of the Power Controller

In the previous chapter, the operation of the individual power converters, which regulate the power from the renewable energy sources and storage elements, was considered. However, in order for any DER system to achieve optimal operation, it is important for the renewable energy sources to complement one another. To facilitate this, a second controller is included in the system which is responsible for the way in which the energy flows from the renewable sources to the various system loads. The primary objective of this controller is to ensure that adequate power is supplied to the main load at all times. In addition to this, it is responsible for the management of the storage elements' state of charge (SoC). The modelling and simulation of this controller, which is shown as the Power Controller in the block diagram given in Fig 91, is discussed in detail in the rest of this chapter.

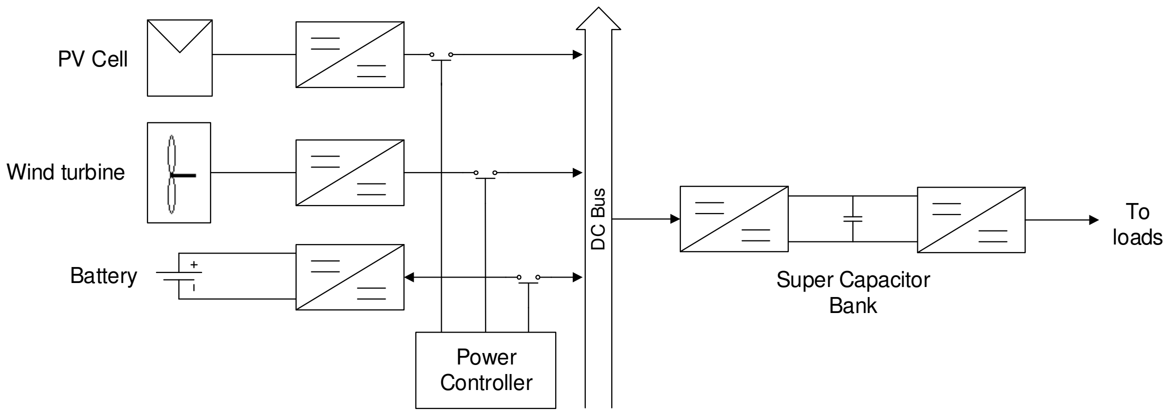


Fig 91 System Block Diagram

6.1 Operation and Modelling

As has been previously discussed, the power produced by the DER system is dependent on the wind speed and the solar irradiance. Due to the uncontrollability of these factors, it can become difficult for DER systems to consistently source sufficient power to the load. This creates a need for storage elements within the system which can smooth out power delivery during periods when the renewable elements provide inadequate power. In this system, two storage elements are utilised for different purposes – a battery and a supercapacitor. The battery is included as a long-term storage solution and is designed to boost power availability when the renewable sources produce insufficient power. The supercapacitor is used as the main output source in the system as it has the ability to deliver power quickly. This means it provides better compensation for the output voltage than the other sources, as shown by Sikkabut et al (2016), Mane et al (2016) and Han et al (2007). The power controller is responsible for the charging of both of these elements, as well as determining when the battery should be discharged.

As the supercapacitor is used to source the system's loads, the primary objective of the power controller is to prevent the supercapacitor from becoming deeply discharged. In order to achieve this goal, charging of the supercapacitor is given priority over the battery bank. This means that the battery is only actively charged during periods when the supercapacitor can't sink all the power being generated by the wind turbine and solar PV cell. In addition to this, the battery is also required to source power under certain operating conditions. This will only occur when the renewable sources aren't producing adequate power to prevent the supercapacitor from becoming deeply discharged. This scenario demonstrates how the battery is utilised to extend the useful capacity, and stability, of the DER system. The nature of the power controller means that it is ideally suited to a state-based algorithm. This has proven to be effective in a number of systems, such as those presented by Xu et al (2016), Zhang et al (2017), Mendis et al (2014) and Sekhar and Mishra (2016). The flow chart shown in Fig 92 demonstrates the full functionality of the state-based algorithm which is utilised in this system

As the power controller interfaces with the power converters introduced in the previous chapter, the modelling approach for this algorithm must be compatible with the Matlab and PSim environments. Therefore, this controller is modelled using the C programming language, which gives the potential for porting of the design into the APSOC processing system. This code can then be compiled and run in the PSim environment, with Matlab co-simulation being offered through the Sim Coupler plug-in module. Simulations of the power controller are presented in the next section as a part of the overall system testing.

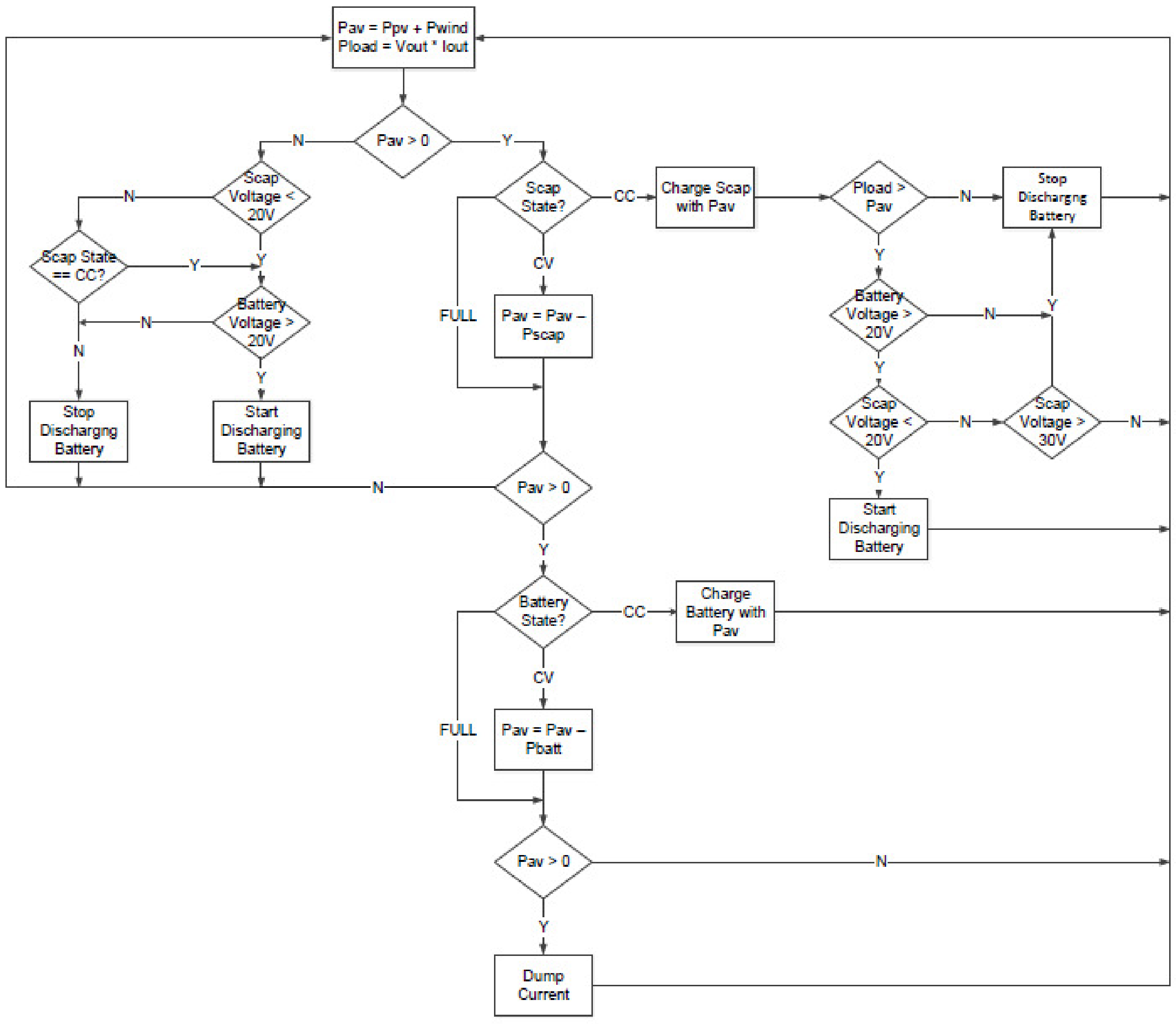


Fig 92 Power Control Algorithm Flow Chart

The first stage of the algorithm presented in Fig 92 is the determination of the power being produced by the renewable energy sources. In order to prevent excessive fluctuations in this value, a 32-sample moving average is utilised. In the case of the wind turbine, this power is determined through multiplication of the measured current and output voltage, as is denoted by Pwind in Fig 92. The operation of the load balancing algorithm, as presented in section 5.1.2.2, depends on an ability to alter the load applied to the solar PV cell. The change in power produced is then utilised to determine when the MPP is achieved. Therefore, the value of Ppv in Fig 92 is determined using this load balancing algorithm.

Once the total power available in the system has been determined, as given by Pav in Fig 92, the controller then begins calculating where this should be routed. Assuming that Pav is greater than 0, this means sinking power into one of the storage elements with the supercapacitor taking precedence over the battery. As discussed in the previous chapter, the charging of the supercapacitor uses a constant current (CC) mode followed by a constant voltage (CV) mode. When operating in the CC mode, the power controller sinks all of the available power into this device. However, when operating in CV mode, the current required by the supercapacitor is both limited and variable. As it gets closer to being fully charged, the current demand reduces until it is close to zero. Therefore, it becomes necessary for the supercapacitor charge algorithm to indicate how much current is required for operation. The power controller can then determine if there is any excess power which can be used by the rest of the system – which ordinarily means the battery. In the case of the supercapacitor being fully charged, all of the available power is utilised elsewhere in the system.

Once the power controller has determined the amount of power to route to the supercapacitor, it then considers the charging of the battery. In the case where the supercapacitor utilises all the power in the system, the battery is left in its floating mode of operation. This means that no charging or discharging takes place and the bus switch for the battery is left open to prevent power flow into or out of the device. Much like the supercapacitor, the battery charges in a CC mode followed by a CV mode and this mode of operation must be considered. If the battery operates in the CC mode, then it is capable of sinking all of the available power in the system. When operating in CV mode, the battery assesses its power requirements and reports this to the controller. In this instance, the power controller would utilise a dump load if any amount of usable power remains in the system after the battery has taken its share. When the battery is fully charged, all of the available power should be sunk into a dump load.

The methodology utilised in the management of the storage elements' SoC is one of the key benefits of the state-based approach employed in this system. Using this tactic means that it becomes unnecessary to precisely measure the SoC for either of the storage components. Achieving an accurate measure of this value can prove difficult. A number of algorithms of varying complexity have been proposed for this, such as extended Kalman filtering (Chen et al (2016)), more complex variations of the Kalman filter (Piao et al (2010) and Charkgard and Farrokhi (2010)) or through the use of genetic algorithms (Chaoui et al (2015)). Instead, the power controller only requires knowledge of the operating mode for the storage elements, so that it may determine how power is routed within the system. These states can be easily determined through voltage and current measurements of the respective elements. The exact determination of the state of both the supercapacitor and the battery is determined as detailed in Table 12 and Table 13 below. It is also important to note that, to avoid oscillating between states, the flow charts presented in Fig 69 and Fig 82 Battery Charger State Machine are observed. This effectively adds hysteresis to the state transitions which in turn improves stability.

State	Current	Voltage
Charging in Constant Current	N/A	< 32V
Charging in Constant Voltage	> 500 mA	32V
Fully Charged	< 500 mA	32V

Table 12 Supercapacitor State Determination

State	Current	Voltage
Charging in Constant Current	N/A	< 29V
Charging in Constant Voltage	> 500 mA	29V
Fully Charged	< 500 mA	29V

Table 13 Battery State Determination

In addition to charging the storage elements, the power controller also determines when the battery is discharged in order to provide stability to the power supply. This occurs when the renewable energy sources are producing insufficient power and the supercapacitor is deeply discharged. There are two different scenarios in which this can arise – when no power is being produced and when the power utilised by the supercapacitor, as given by Pload in Fig 92, is greater than Pav. In both cases, the battery starts discharging when the supercapacitor voltage is less than 22V. This continues until either the supercapacitor enters the CV mode or the renewable energy sources begin producing sufficient power. When no power is being produced, the battery's rate of discharge is 10% higher than Pload. In the case where power is being produced, the battery discharge rate is equal to Pload. It is also important to prevent the battery from becoming fully discharged, therefore the battery voltage must be greater than 20V before it can begin charging the supercapacitor.

6.1.1 Switch Topologies

Although a relatively trivial task, the methodology applied to connect and disconnect devices from the main DC bus warrants some deliberation. In particular, the topology of the switching circuits must be considered. In the case of the renewable energy sources and the supercapacitor circuit, unidirectional switching is required. A simple P channel MOSFET device is the ideal solution for this as it reduces the complexity of the driving circuit in comparison to an NMOS device. In order to drive the PMOS circuit directly from a logic level, such as an FPGA or processor pin, it is necessary to utilise an NMOS transistor in the driver circuitry. The NMOS is then used to drive the gate to a sufficiently low voltage to switch off the PMOS, as is shown in Fig 93. A simulation of this circuit has been carried out using the popular LTSpice circuit simulation tool. The results which are given in Fig 94 demonstrate how a 48VDC bus is connected to an external load through the use of a logic level (3.3V) input control signal.

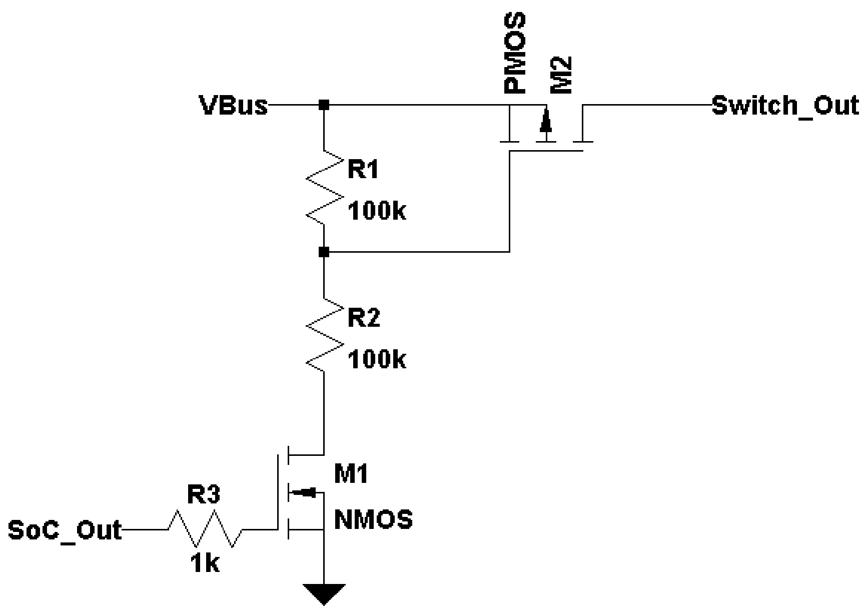


Fig 93 High Side PMOS Based Switch Circuit

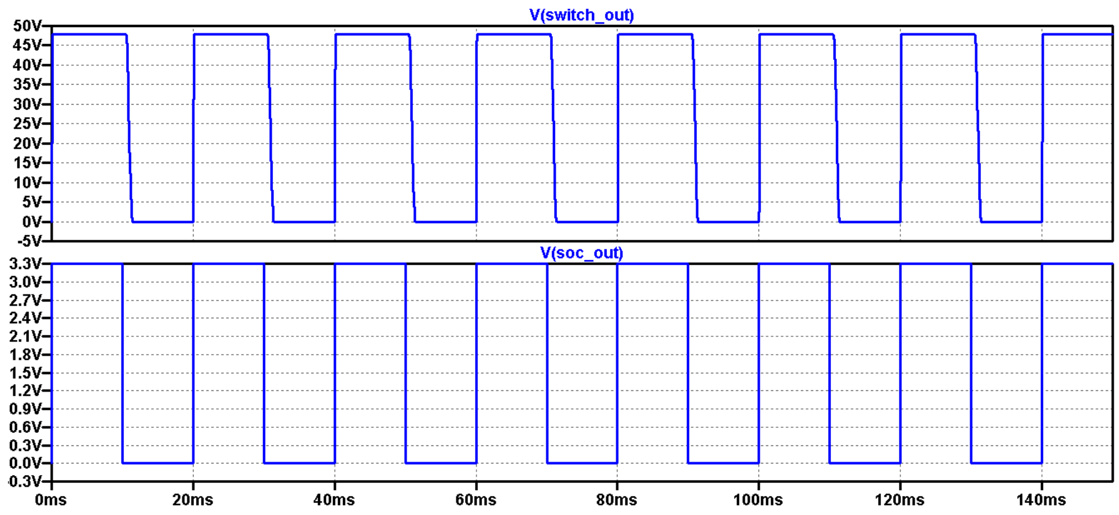


Fig 94 Simulation Results of the High Side PMOS Based Switch Circuit

Unlike the other components in the system, the battery requires a connection with the common voltage bus which allows bi-directional power flow so that it can both source and sink power. This is a more complex task to achieve, largely due to the fact that a suitable silicon switching device doesn't exist. One simple solution, which has been demonstrated by Castellazzi et al (2013) and Mori et al (2013), is the use of a pair of PMOS transistors with serial diodes. Using this approach, there is one PMOS for each direction of flow, with the series diode preventing bidirectional conduction within the silicon device. An example of this circuit is shown in Fig 95, noting that this excludes the driver circuitry for the PMOS transistor. For a fully operational circuit, the NMOS based driver circuit presented in Fig 93 would be required for each of the switches. In order to show the operation of this circuit, LTSpice has been used to simulate its operation. In Fig 96, it can be seen how current flows through D1 from bus1 to bus 2 when the bottom PMOS (M2) is in conduction. Conversely, current flow from bus2 to bus1 and through D2 when the top PMOS (M3) is in conduction, as is shown in Fig 97.

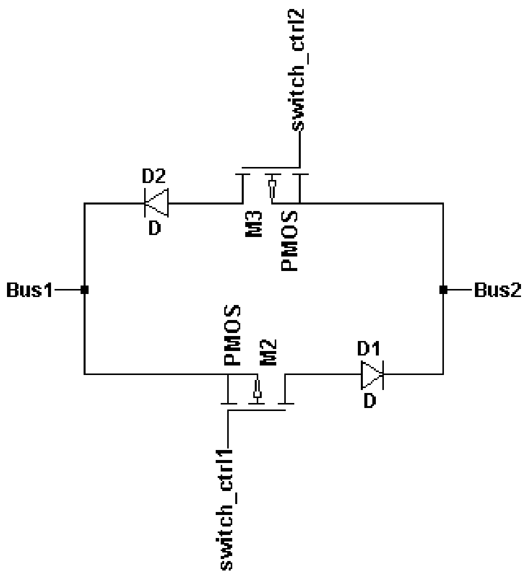


Fig 95 Bidirectional PMOS Based Circuit

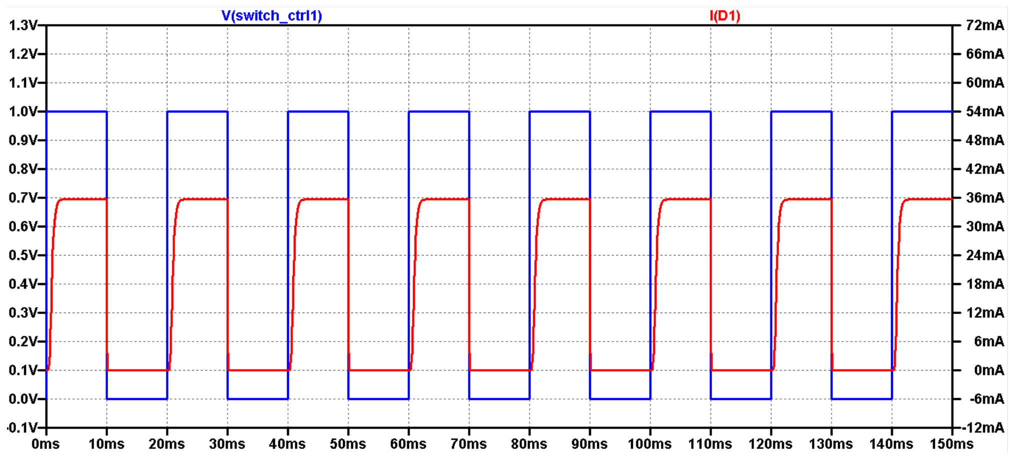


Fig 96 Bidirectional Switch Simulation with Current Flowing from Bus1 to Bus2

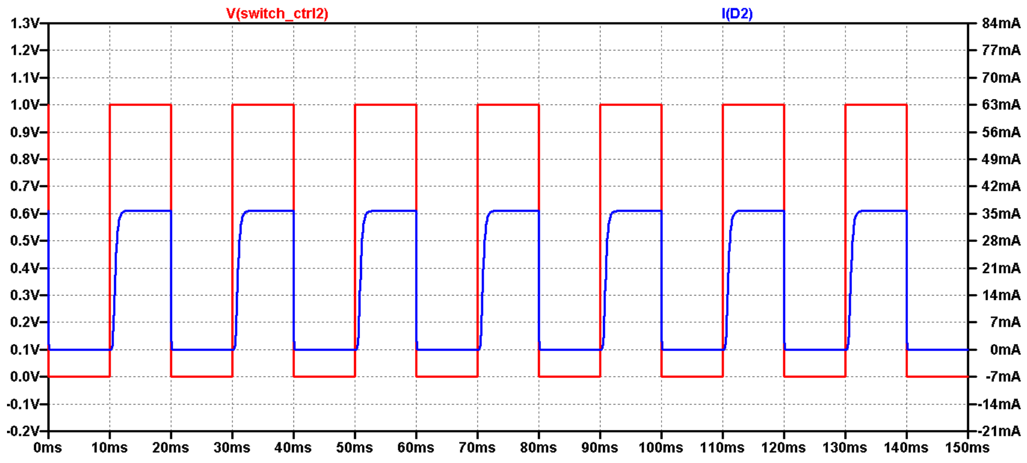


Fig 97 Bidirectional Switch Simulation with Current Flowing from Bus2 to Bus1

In the previous chapter, a discussion was undertaken concerning the methodology used to create a soft start function for the boost converter topology. However, a similar strategy is also required for the buck converters which are used to charge the storage elements in the system. As the main aim of the soft start is to prevent large inrush currents on the capacitor, this can be achieved by slowly ramping up the converters reference current when the converter initially beings operating. An example of this is given in Fig 98, whereby the current is ramped up at a rate of 50mA every 1 µs which has the effect of reducing inrush current into the converter.

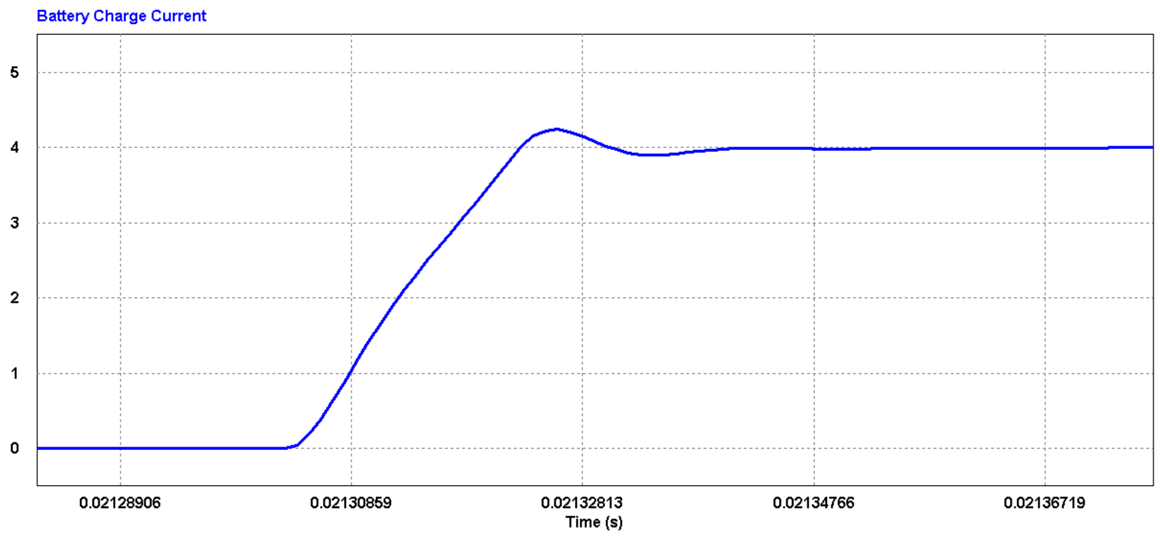


Fig 98 Example of the Buck Converter Soft Start Function

6.2 System Simulations

In order to characterise the behaviour of the DER system, a number of simulations have been carried out. These tests are split into three separate components - the complementation of the renewable energy sources, the performance of the supercapacitor when discharging and the battery sub-system. In the rest of this section, the three elements are all discussed in more detail.

6.2.1 Complementation of the Renewable Energy Sources

One of the major aims of the power controller is to ensure that the renewable energy sources are optimally complementary to one another. In order to facilitate this, the solar PV cell and wind turbine sink power into the storage elements via a common voltage bus. In the majority of instances, the supercapacitor is the component which stores the power from the renewables. This is owing to the fact that this device sources the external loads and must be kept charged. In order to demonstrate this functionality, the system is simulated with a number of different wind and irradiance values under both steady state and dynamic conditions.

The first simulation scenario considers the instance where only the solar PV cell is producing power. To achieve this, the irradiance is set to a steady value and a light load is applied to the output. This is repeated with two different levels of irradiation so that a small and large amount of power is sunk into the supercapacitor. In Fig 99, the first test is carried out with 300 w/m^2 of irradiance. An important characteristic which is shown in these waveforms is the manner in which the power is routed from the solar PV cell into the supercapacitor. It can be observed that the solar PV cell produces a total of 63.7 W, once it has settled into its steady state of operation. The supercapacitor then sinks a total of 61.1 W, which demonstrates a total efficiency of 96.4%. It is also possible to characterise the performance of the load balancing MPPT algorithm, which was discussed in some detail in the last chapter. Here, we observe a theoretical MPP of 66.7 W, meaning that the MPPT algorithm has a tracking accuracy of 95.5%. It can also be deduced that the system exhibits an efficiency of 92.1 % against the theoretical maximum under these operating conditions.

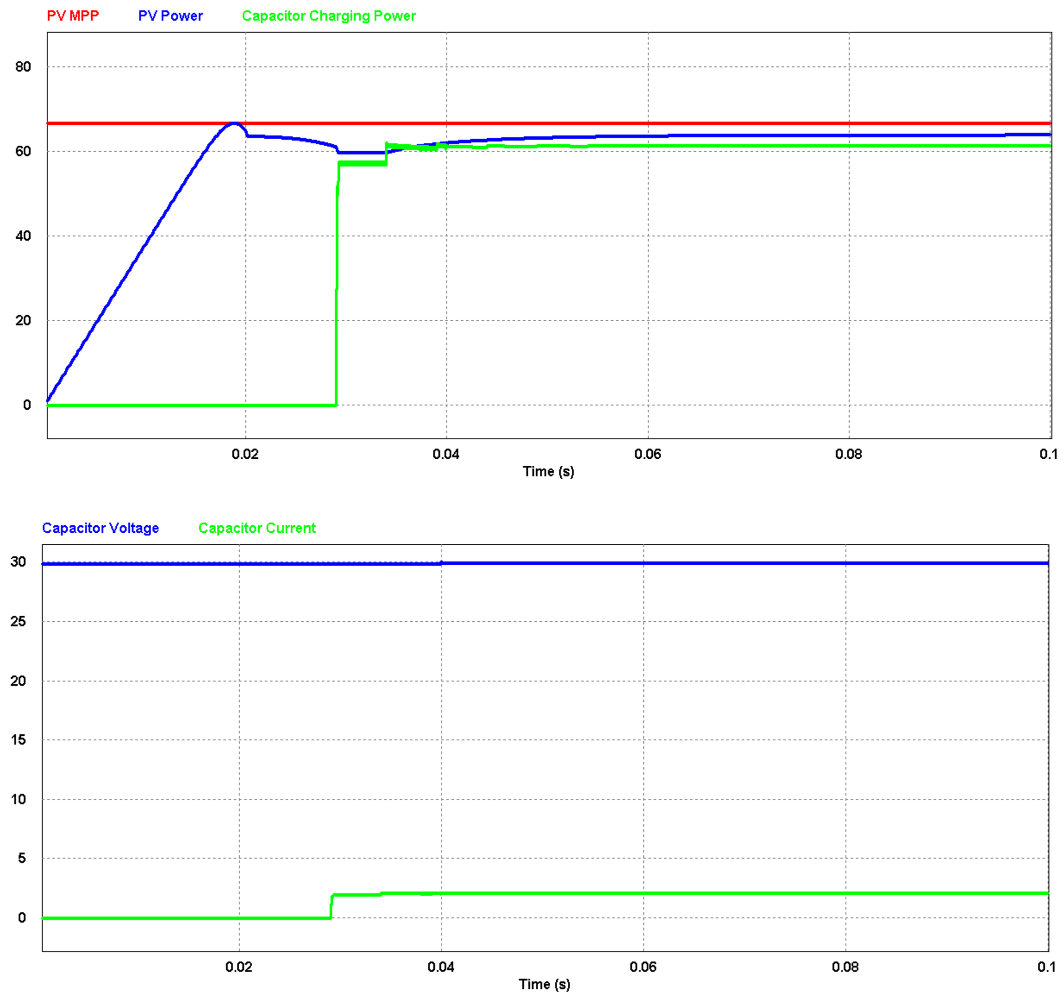
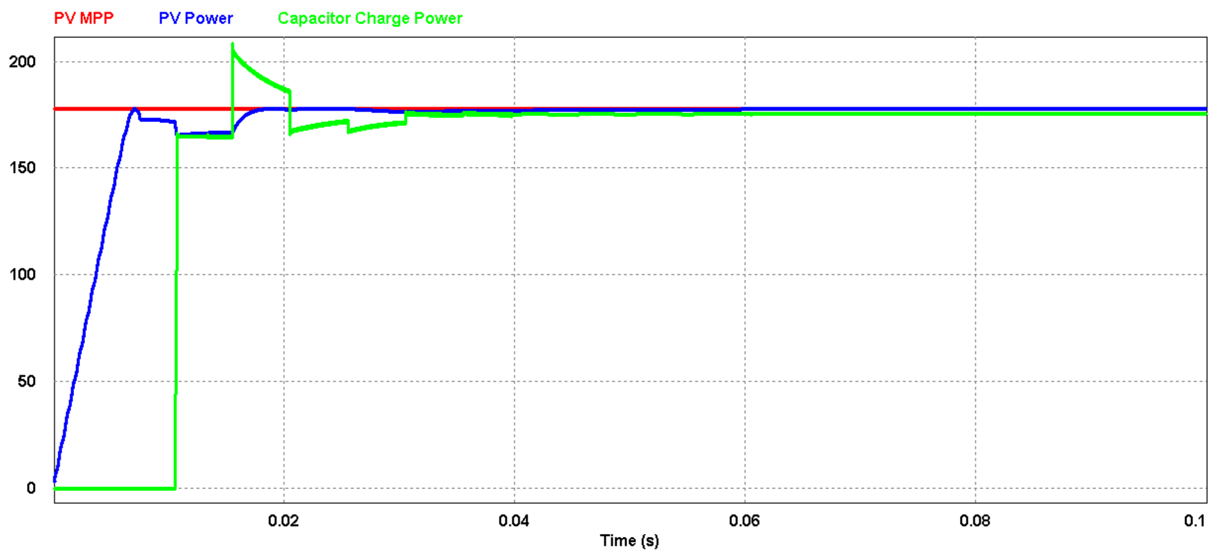


Fig 99 Solar PV Cell Power and Supercapacitor Charging Power with 300 w/m^2 Irradiance

In the second simulation test, the solar irradiance is set to a fixed value of 800 w/m^2. A fixed load is then applied to the output, as was the case in the previous test. From Fig 100 it can be observed that the supercapacitor sinks 175.7 W after it has settled into its steady state of operation. At this point, the solar PV cell is generating a total of 177.8 W of power, meaning the supercapacitor is charged at an efficiency of 98.8%. The MPPT tracking accuracy can also be observed and evaluated in this simulation. Under these operating conditions, the theoretical MPP is given as 178W, meaning that the MPPT algorithm has an accuracy of 99.8%. Finally, it can be deduced from the simulation results that the supercapacitor charging rate gives an efficiency of 98.4% against the theoretical maximum.



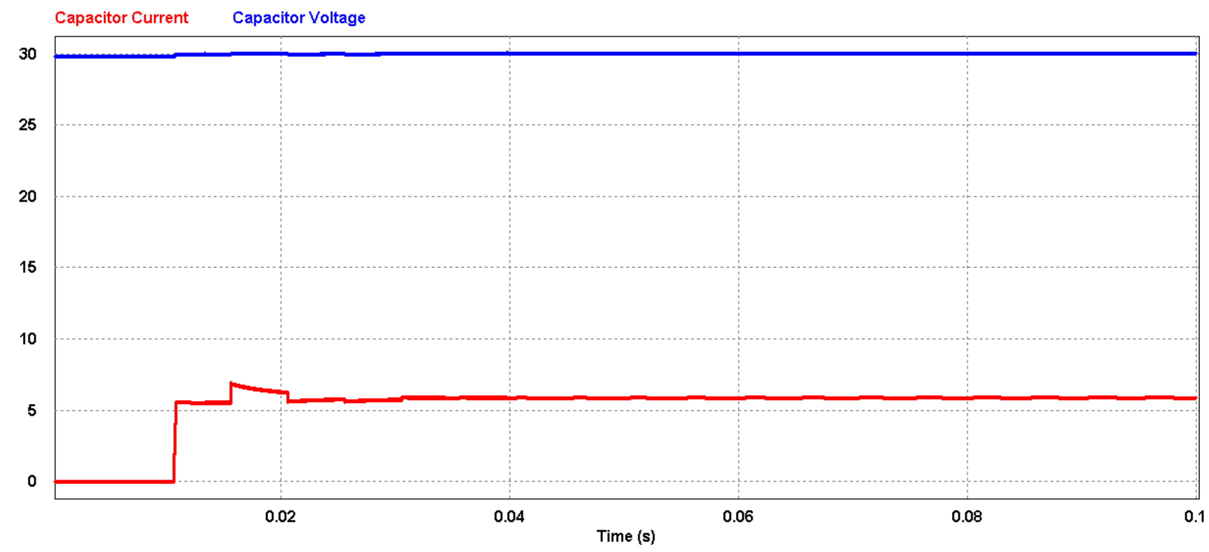
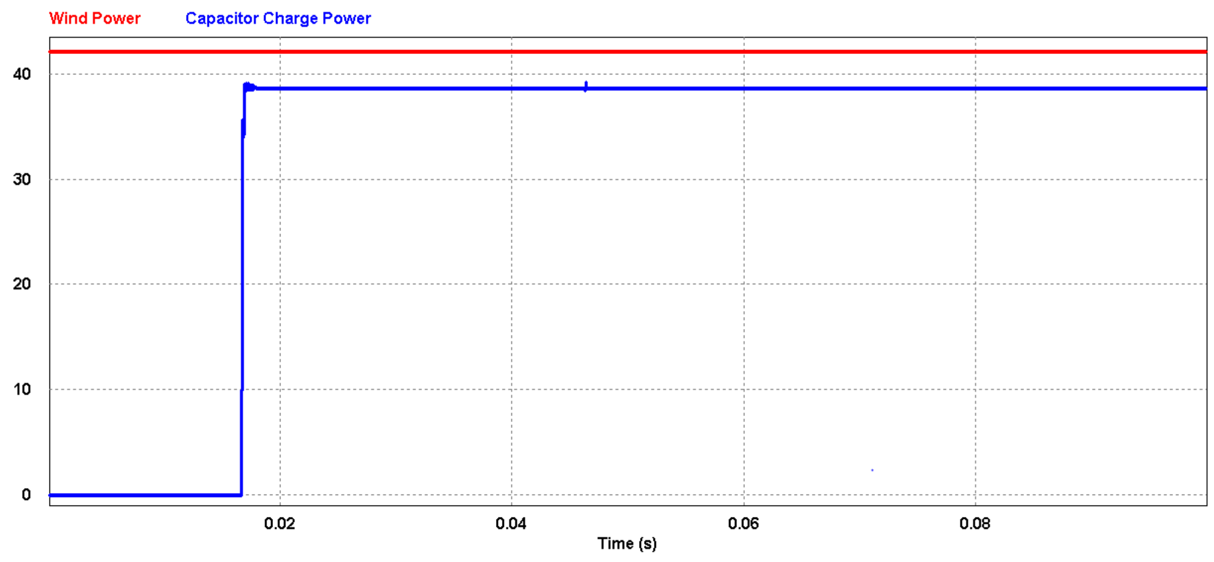


Fig 100 Solar PV Cell Power and Supercapacitor Charging Power with 800 w/m2 Irradiance

In the next set of tests, the wind turbine is used as the sole source of power for the system. In this operating mode, the wind speed is set to a fixed value and a 2A load is applied to the output. This is repeated with speeds of 6 m/s and 12 m/s, corresponding to a small and large amount of power production. As with the previously presented simulations, the main consideration here is the efficiency with which the power is routed into the supercapacitor from the wind turbine. It can be observed from Fig 101 that the power available with the slower wind speed is 42.1 W, of which 38.7 W is utilised by the capacitor. In the second test case, 92.8 W of power is produced and 86.9 W of this is sunk into the supercapacitor, as shown in Fig 102. This represents an efficiency of 91.9% and 93.6% respectively.



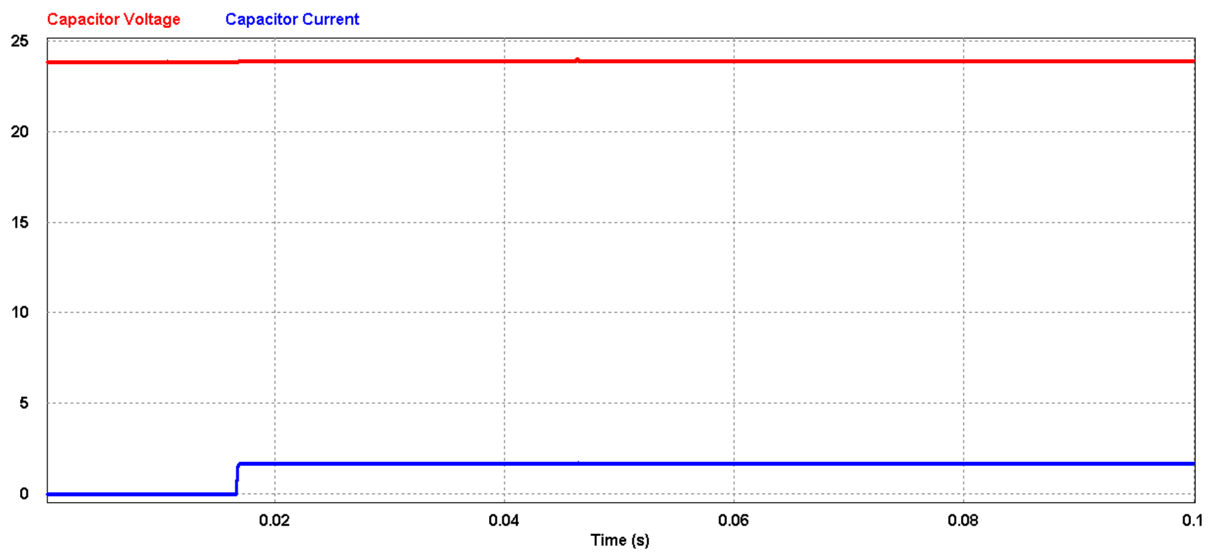


Fig 101 Wind Turbine Power and Supercapacitor Charging Power with 6 m/s Wind Speed

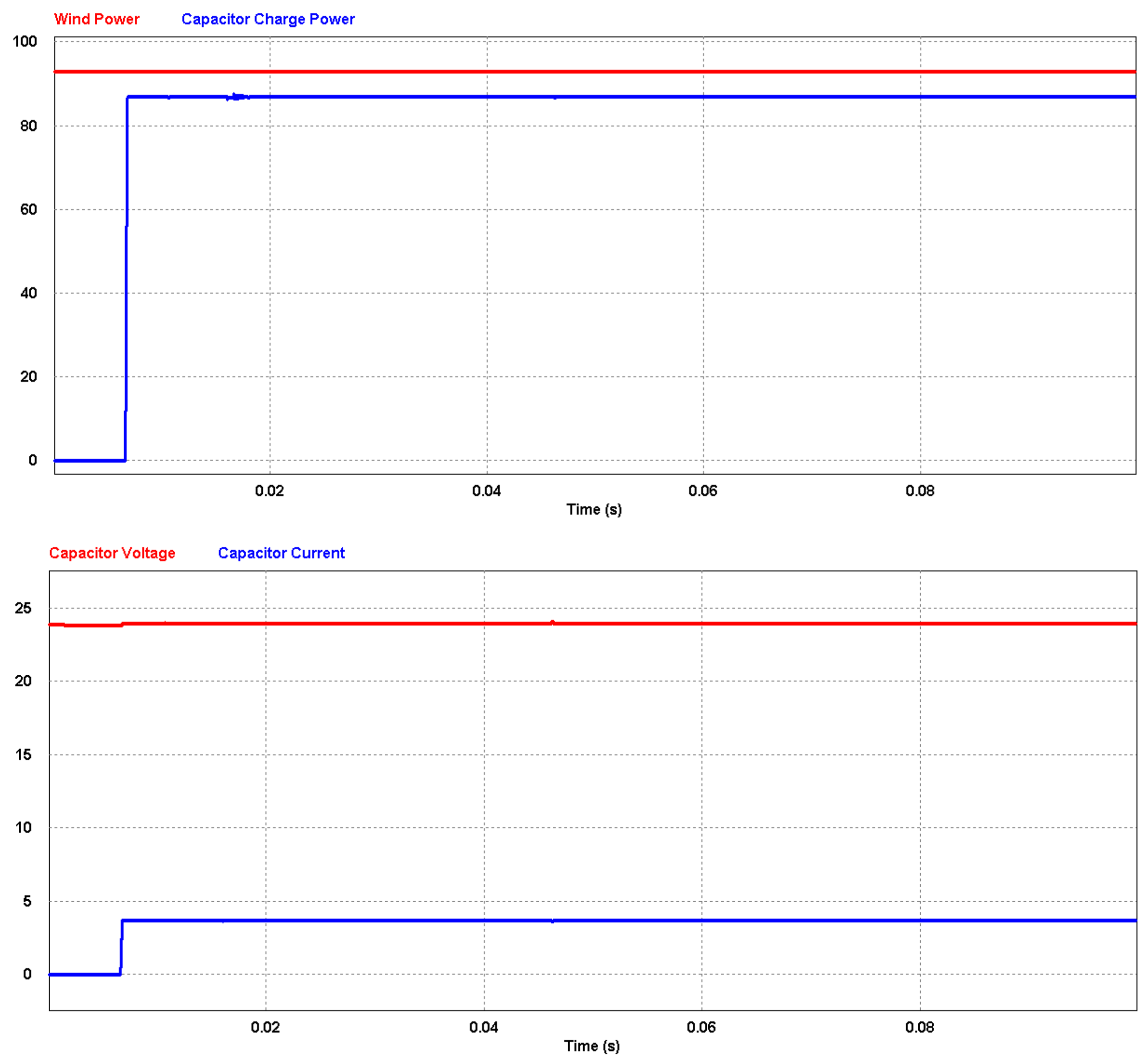


Fig 102 Wind Turbine Power and Supercapacitor Charging Power with 12 m/s Wind Speed

The third set of simulations in this section considers the performance of the system when both of the renewable sources are generating power. This is repeated with both a large amount of power being produced and a small amount. As with the previous simulations in this section, the intention is that the power is utilised to charge the supercapacitor. In the first test scenario, with the wind speed and irradiance set to low values, the total power being produced is 143.8 W. The supercapacitor sinks 135.3 W of this, as is shown in Fig 103, representing an efficiency of 94.1%. In the second scenario, 240 W of power is produced by the renewable energy sources, as is shown in Fig 104. The supercapacitor utilises 226.8 W of this power, giving a total efficiency of 94.5%.

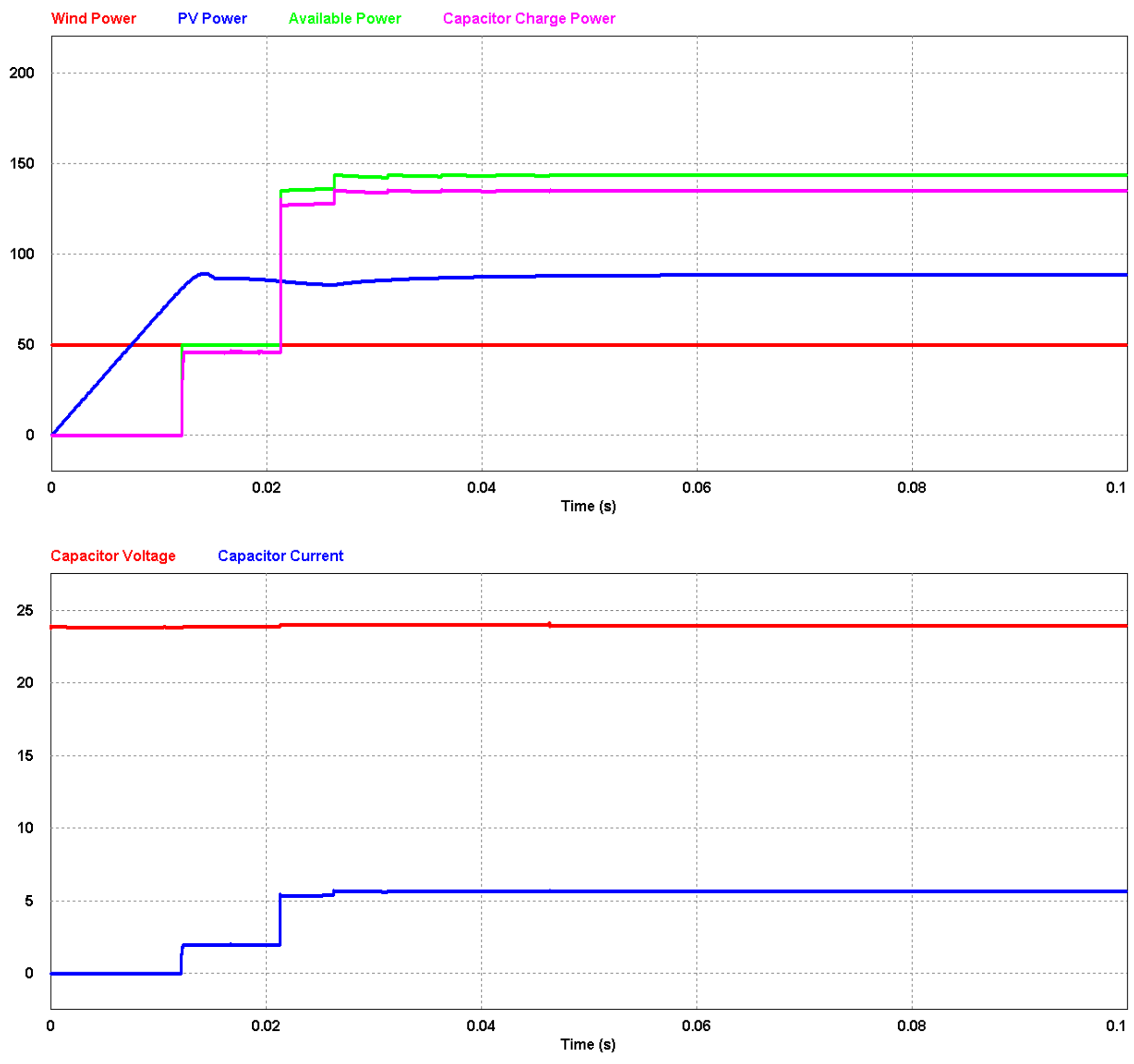


Fig 103 Wind Turbine, Solar PV Cell and Supercapacitor Charging Power with 6 m/s Wind Speed and 400 w/m^2 Irradiance

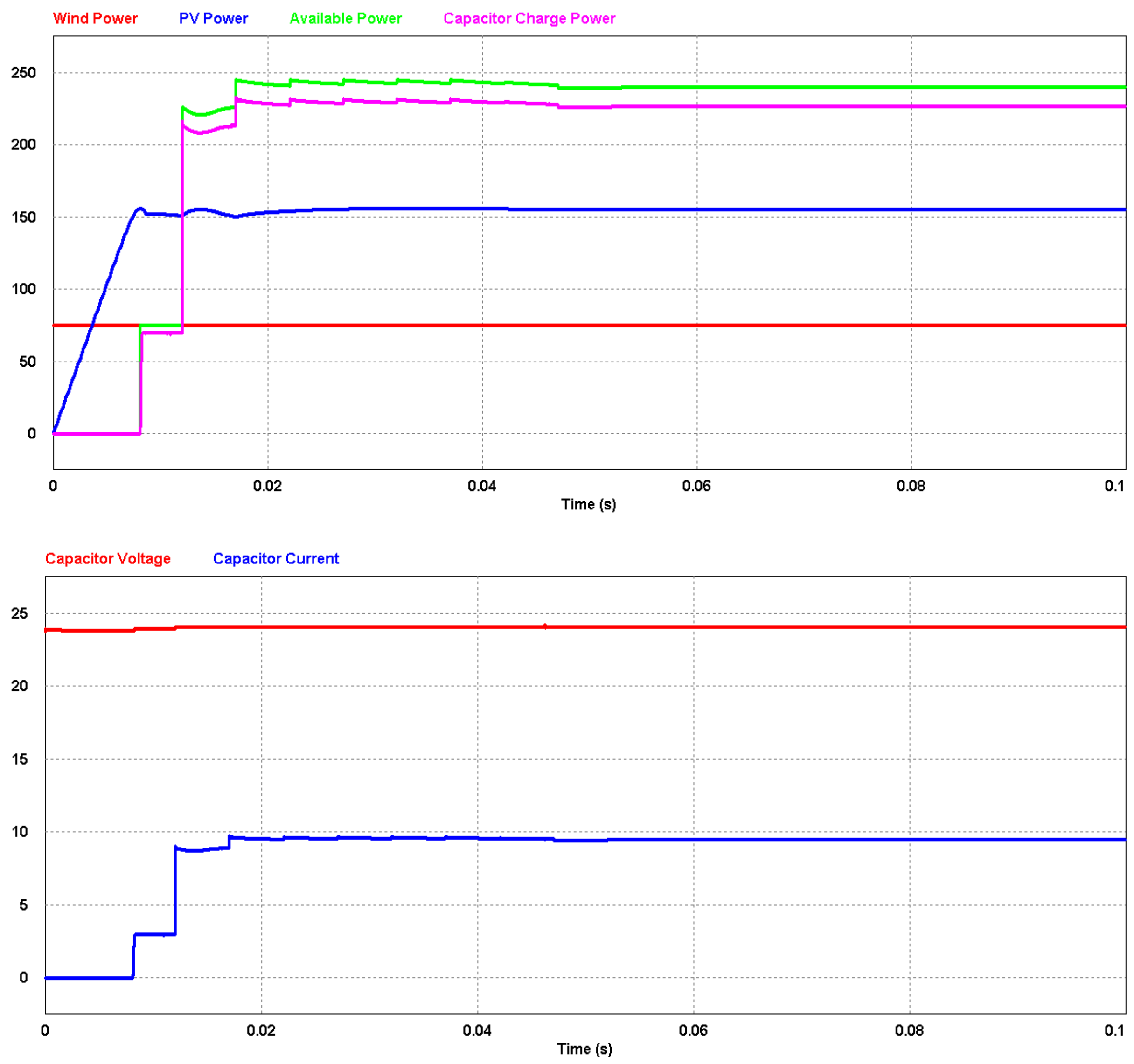
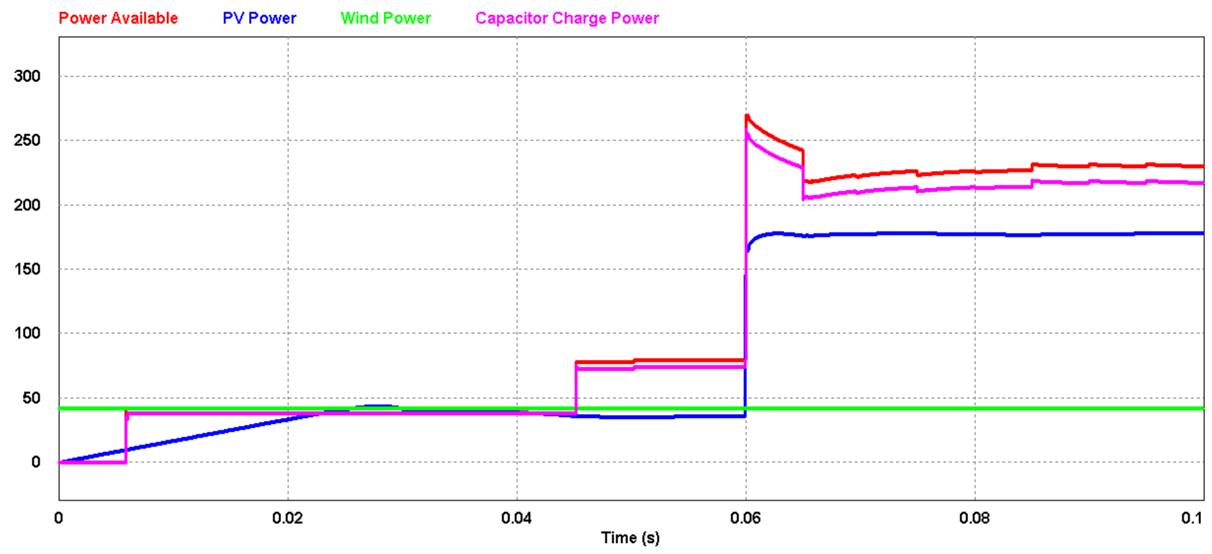


Fig 104 Wind Turbine, Solar PV Cell and Supercapacitor Charging Power with 10 m/s Wind Speed and 800 w/m^2 Irradiance

The test scenarios presented up until this point have only considered the operation of the system with steady, constant levels of irradiance and wind speed. However, it is also important to consider the dynamic operation, which occurs when there are fluctuations in the power produced by the renewables. Therefore, the final set of simulation scenarios looks at the operation with step changes in the irradiance and then in the wind power. These tests also provide an opportunity to examine the behaviour of the regulated voltage buses under fluctuating conditions.

The first dynamic test considers the case where there is a step increase of 600 w/m^2 in the solar irradiance, with the wind speed set to a constant value of 6 m/s. In the early part of this simulation, as shown in Fig 105, it can be seen that the supercapacitor is charged with a fixed power of 38.7W. This is drawn exclusively from the wind turbine at this point, which produces 42.1 W. The solar PV cell begins operating after approximately 45ms, at which point the supercapacitor is charged with 74.52 W of the available 79.84 W. The step increase is then applied after 60 ms, with the system taking approximately 25 ms to settle back into a steady state. At the end of this period, the supercapacitor is being charged with

218.1 W of the available 231 W. This illustrates that the total efficiency of the system begins at 91.92 %, before moving to 93.34 % and finally finishing at 94.41 %.



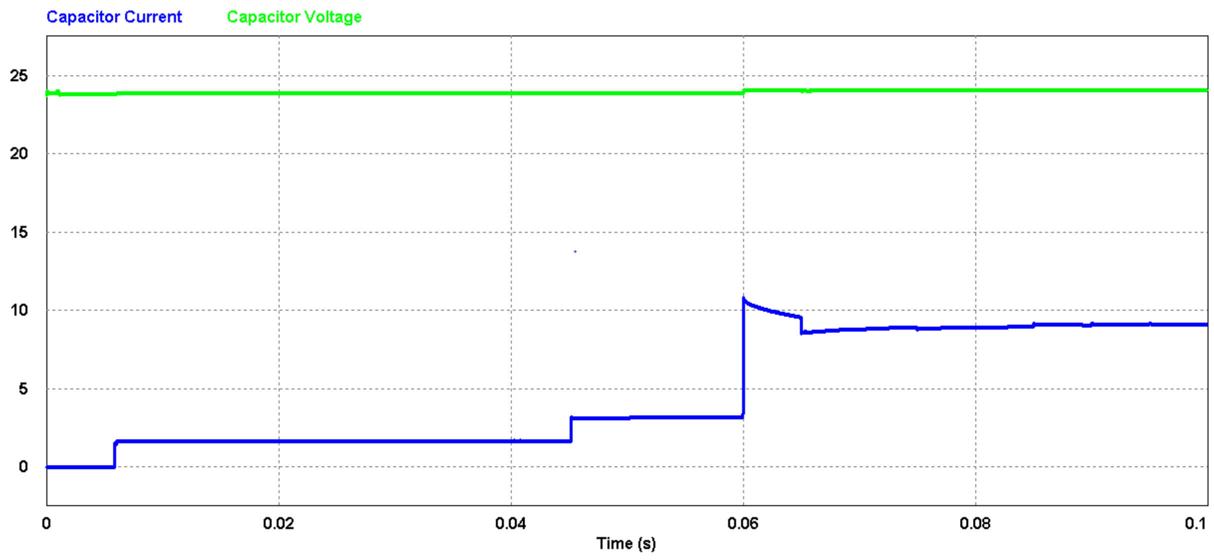


Fig 105 Wind Turbine, Solar PV Cell and Supercapacitor Charging Power with 6 m/s Wind Speed and a 600 w/m^2 Step Increase in Irradiance

This test case also provides an opportunity to consider the stability of the DC busses as the power produced by the renewables varies. In the first instance, the effect of the solar PV coming online can be considered. This occurs at approximately 45 ms into the simulation, at which point both the renewables are sourcing power to the load. In Fig 106, it can be seen how this causes an absolute error of 14 mV (0.029 %) in the common DC bus. However, the output voltage settles back to its reference point after 58 µs. As is expected, this also affects the output of the system with a small disturbance of less than 1 mV being observed. An additional disturbance occurs when the irradiance increases after 60 ms, as is shown in Fig 107. It can again be seen how this creates an error of 72.5 mV (0.15%) in the DC bus before settling back at the reference voltage after 90 µs. Again, this also affects the regulation of the system output, as is seen by a small divergence of less than 1mV.

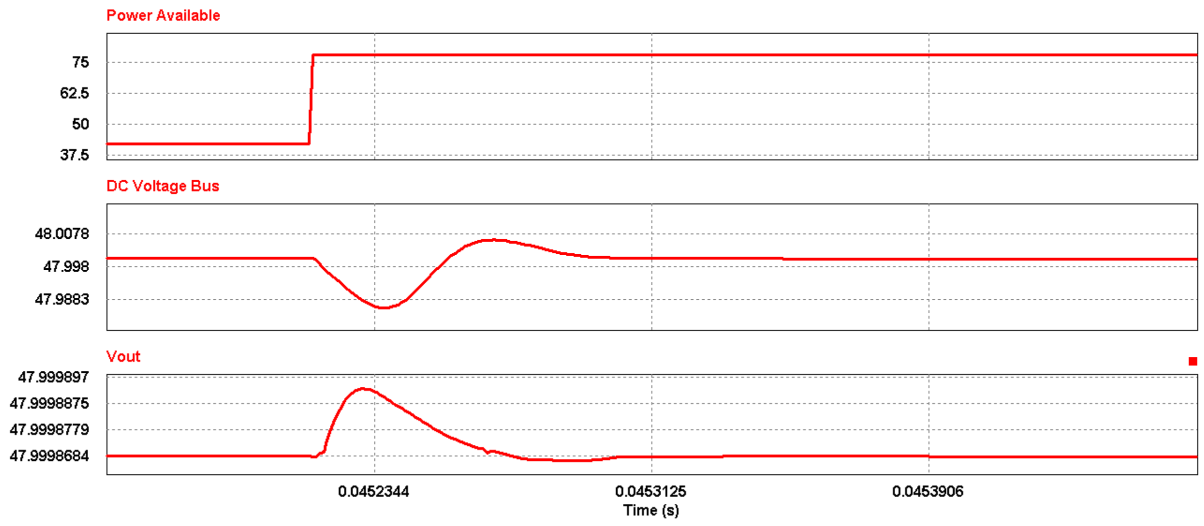


Fig 106 Voltage Regulation when PV Solar Cell Begins Outputting Power

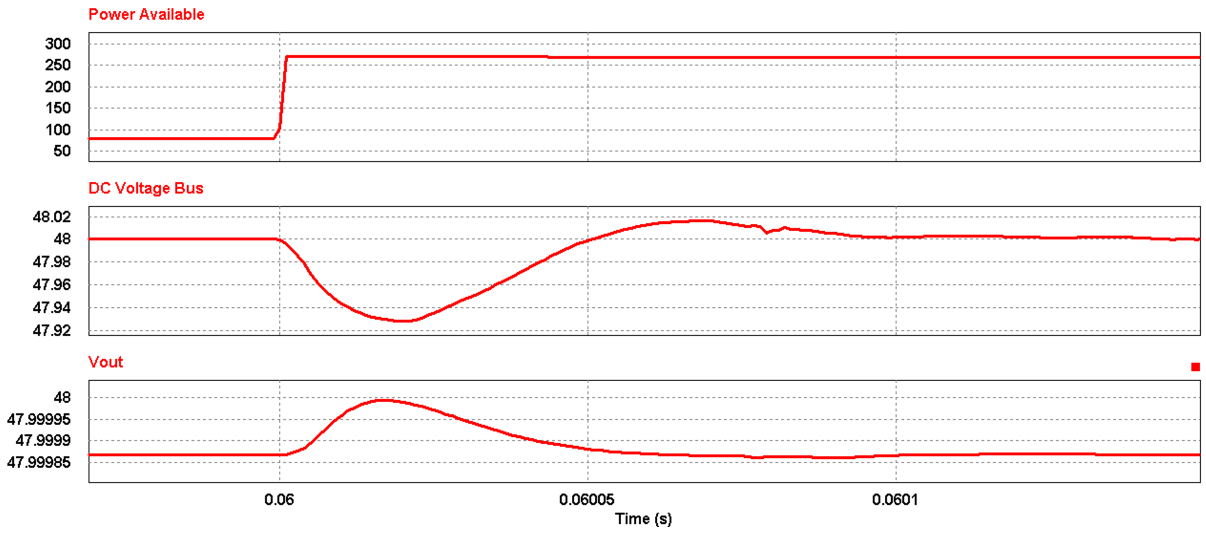
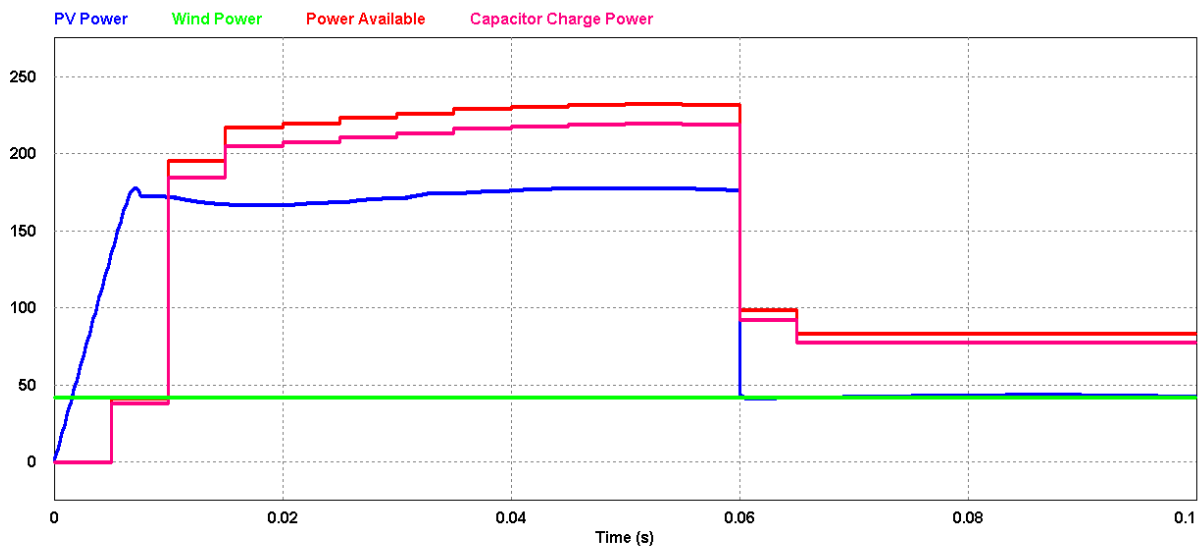


Fig 107 Voltage Regulation when Step Decrease Occurs in PV Solar Cell Power Production

The second dynamic simulation uses the same setup as the previous test, with a fixed wind speed of 6 m/s and 600 w/m^2 step in the solar irradiance. However, the solar irradiance is decreased rather than increased in this case, as is shown in Fig 108. As with the previous test, the wind turbine sources all of the power for the supercapacitor in the first part of the simulation. In this phase, the wind turbine produces 42.1 W of power, with the supercapacitor sinking a total of 38.7 W to give an efficiency of 91.9%. The solar PV cell then begins to produce useable energy after approximately 10 ms, when the available power slowly increases to a total of 232 W. As is expected, this results in a corresponding increase in the supercapacitor charge rate to 219.6 W at an efficiency of 94.6 %. Finally, the step in irradiance occurs at 60 ms, at which point the supercapacitor is charged with 78.1 W of the total 83.6 W that is available. This means that the efficiency at the end of this simulation is equal to 93.4 %.



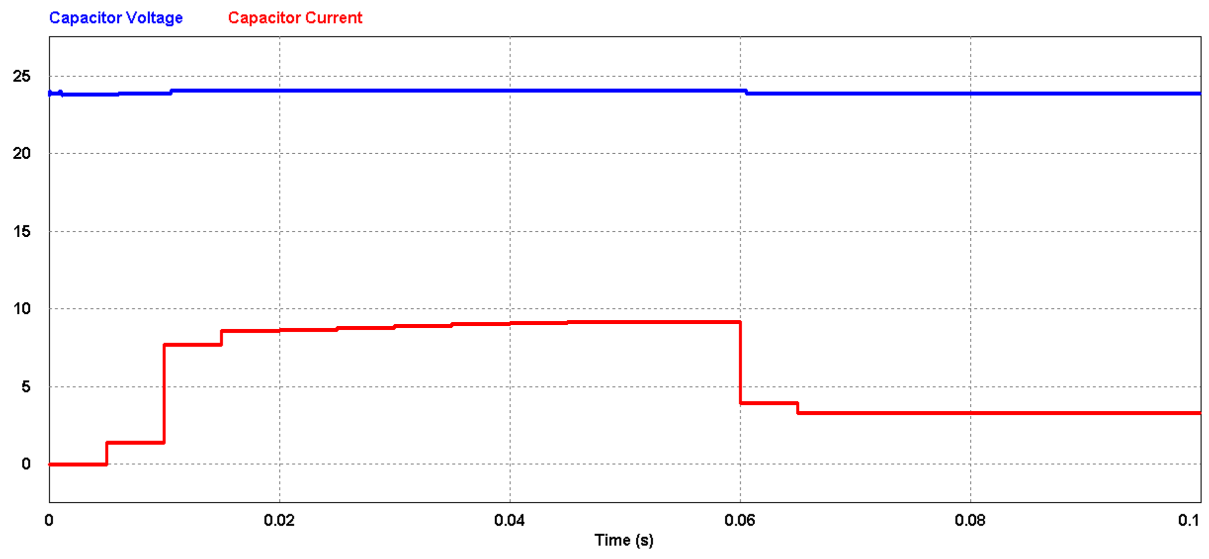


Fig 108 Wind Turbine, Solar PV Cell and Supercapacitor Charging Power with 6 m/s Wind Speed and a 600 w/m^2 Step Decrease in Irradiance

In this test, there are two points at which the DC buses in the system are subjected to disturbances. The first of these occurs when the solar PV cell starts outputting power on the bus, as is shown in Fig 109. This happens after approximately 10 ms, when an absolute error of 75 mV (0.156%) can be seen on the common DC bus. The output voltage settles back to its reference point after 260 µs. A corresponding deviation can be observed on the systems output bus, although this is less than 1 mV, as shown in Fig 109. A second disturbance takes place at 60 ms when there is a step in the solar irradiance, as is shown in Fig 110. This causes an absolute error of 46 mV (0.095%) in the DC voltage bus, whilst the settling time in this instance is 33 µs. This also creates a small fluctuation of less than 1 mV in the output of the systems, as can be observed in Fig 110.

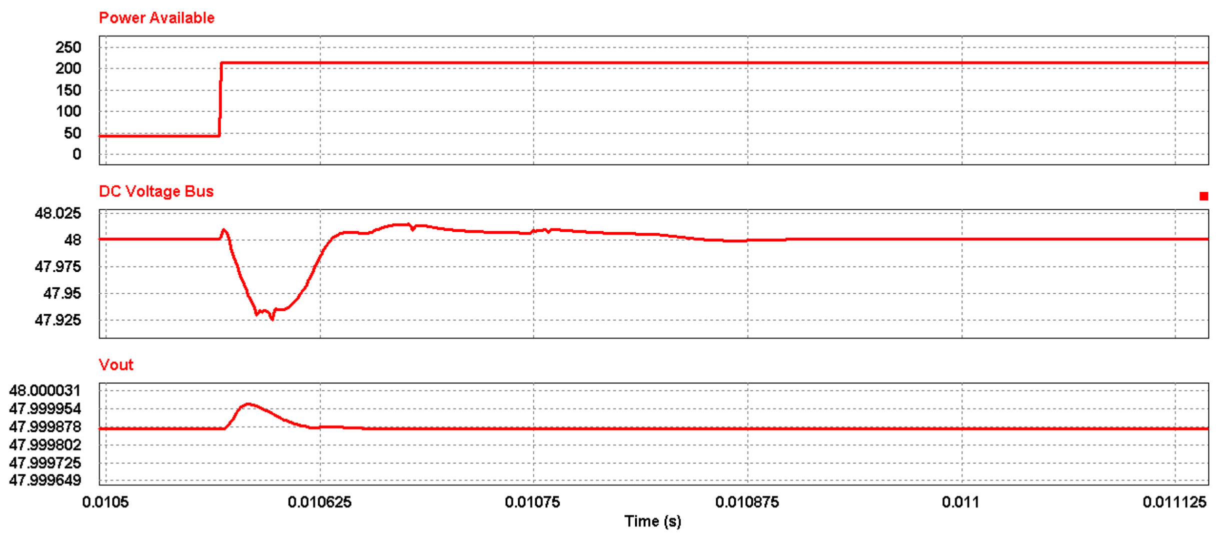


Fig 109 Voltage Regulation when PV Solar Cell Begins Outputting Power

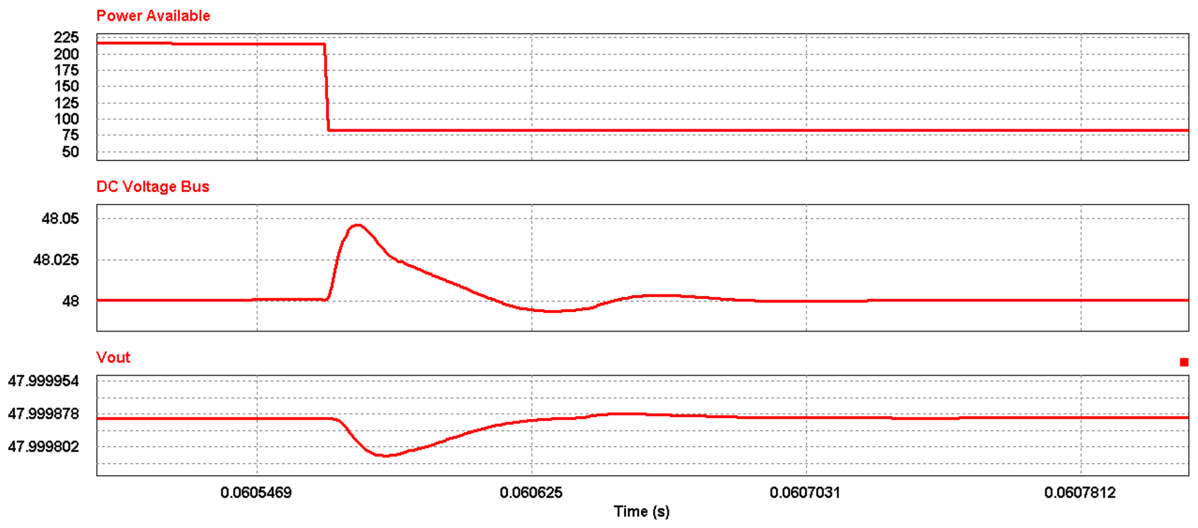
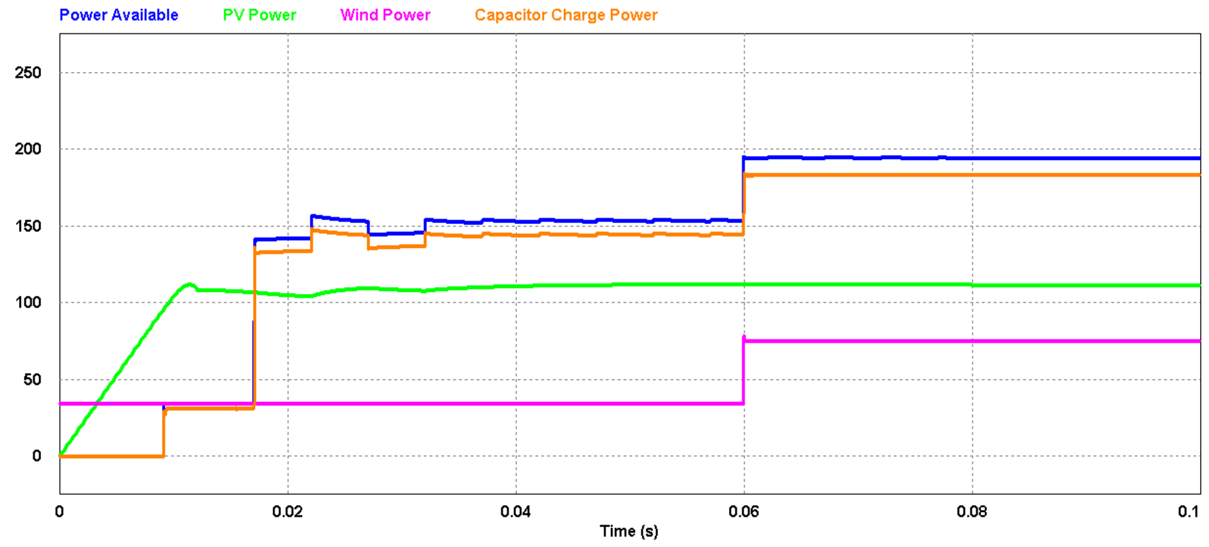


Fig 110 Voltage Regulation when Step Decrease Occurs in PV Solar Cell Power Production

The dynamic tests presented up until this point have considered the operation of the system when there is a step in the solar irradiance. In the final set of simulations, the performance of the system with fluctuating wind speed is demonstrated. In the first of these tests, a step increase of 5 m/s in wind speed is applied to the system whilst the solar irradiance is set to a steady value of 500 w/m^2. In the first part of this simulation, as shown in Fig 111, the supercapacitor is charged with a power of 31.4W. This power is sourced entirely from the wind turbine, which is producing a total of 34.5W. This means that the efficiency of the system is equal to 91% at this point. At approximately 17 ms, the startup sequence of the solar PV cell completes and the system has a total available power of 154.25 W. The supercapacitor charge power increases to 145.21W as a result of this, giving an efficiency of 94 %. The increase in wind speed takes place 60 ms into the simulation, when the total available power increases to 194.59 W. The supercapacitor charge power simultaneously increases at this point to a total of 183.5 W, giving a charging efficiency of 94 %.



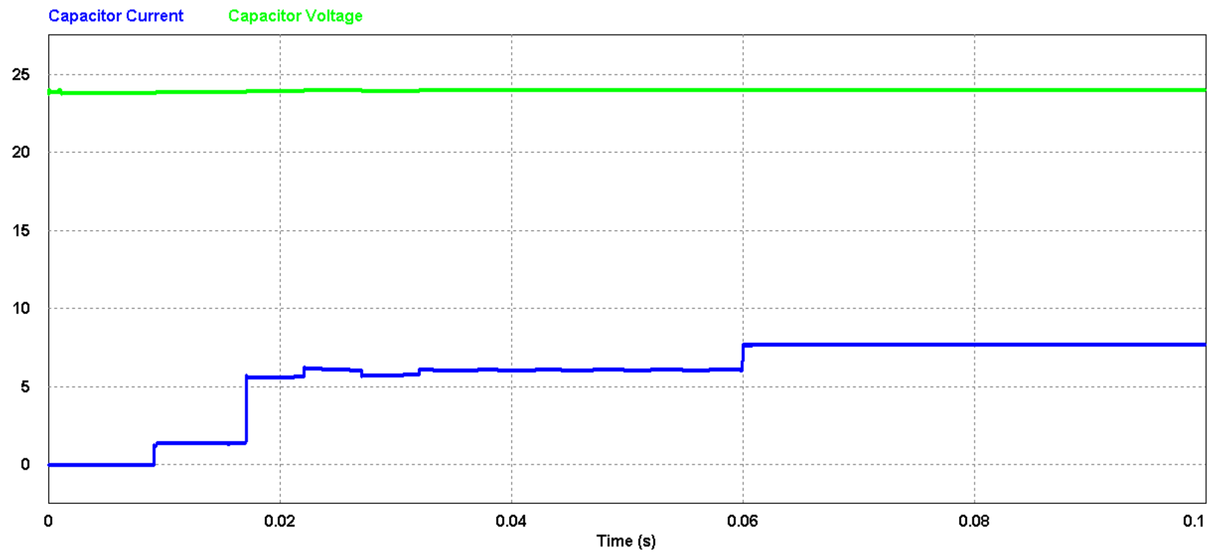


Fig 111 Wind Turbine, Solar PV Cell and Supercapacitor Charging Power with 500 w/m^2 Irradiance and 5 m/s Step Increase in Wind Speed

As was the case with the previous dynamic tests, this simulation provides a chance to examine the regulation of the system's DC busses under dynamic operating conditions. At approximately 17 ms into the simulation, the solar PV cell begins sourcing power into the common DC bus. Fig 112 shows how this causes an error in the voltage regulation of 37mV (0.077%) which takes 172 µs to return to the reference voltage. This fluctuation also generates a small error of less than 1 mV in the system's output. The step in wind speed, which occurs at 60 ms into the simulation, causes another deviation as can be seen in Fig 113. In this case, the error is smaller, at just 8 mV, with the output returning to the reference voltage after 68 µs. As with the previous disturbance, this also generates a small error in the output bus which is less than 1 mV.

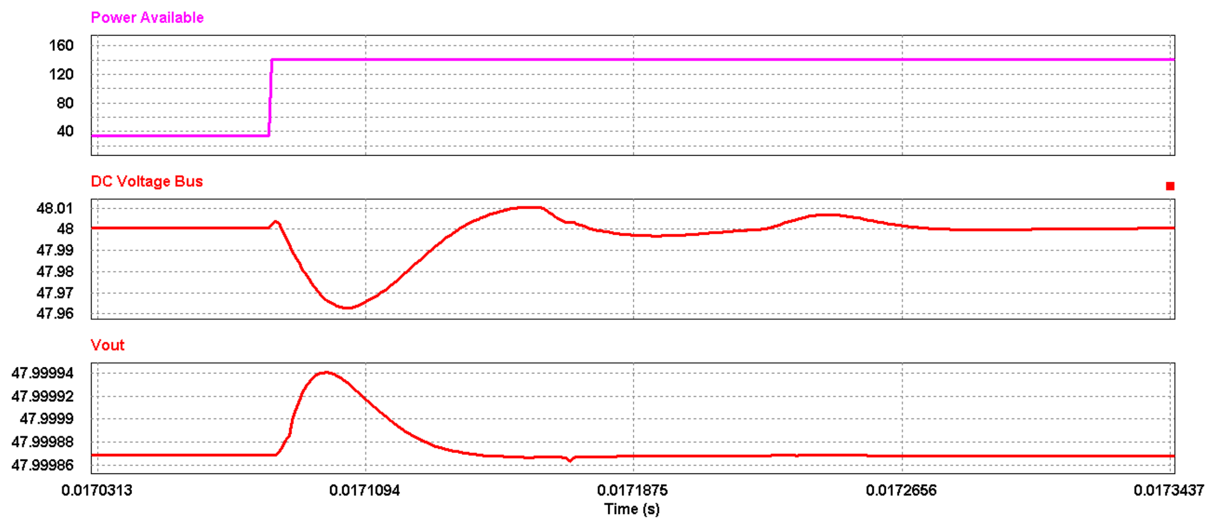


Fig 112 Voltage Regulation During Wind Step Increase Test when PV Solar Cell Begins Outputting Power

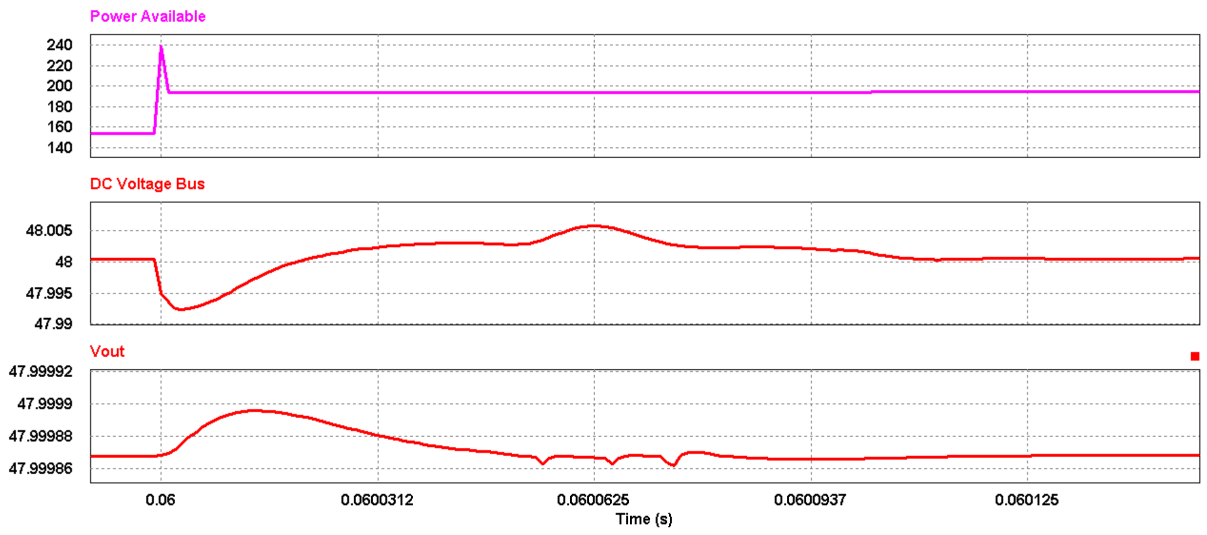
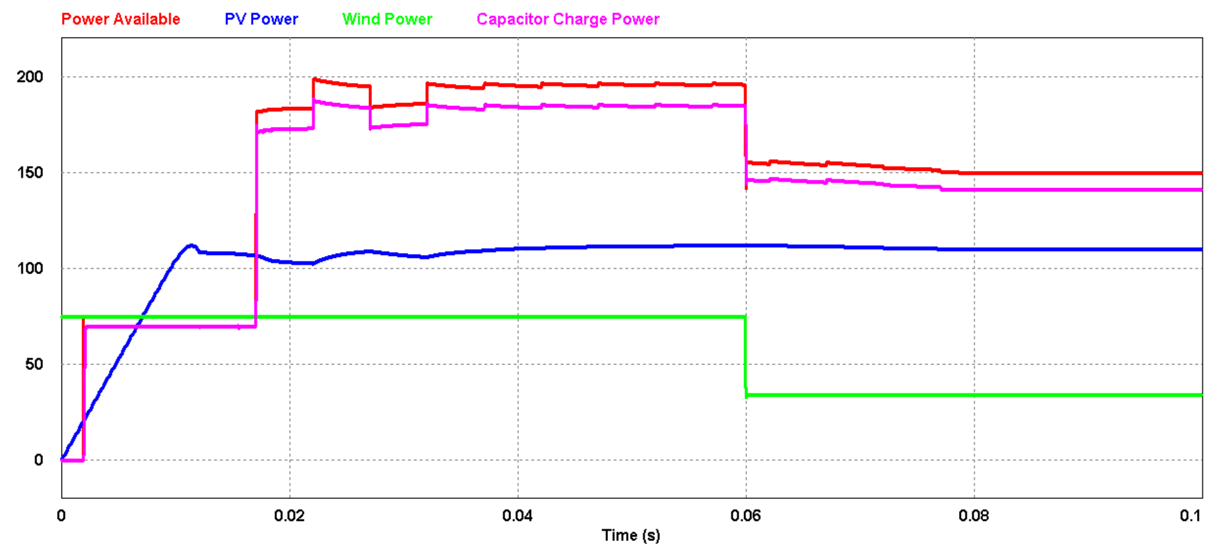


Fig 113 Voltage Regulation when Step Increase in Wind Speed

The final dynamic test scenario considers the operation of the system when there is a step decrease of 5 m/s in wind speed, whilst the solar irradiance is set at a fixed value of 500 w/m^2. In the first part of the simulation, as is shown in Fig 114, the supercapacitor charge power is sourced entirely from the wind turbine. At this stage, the wind turbine is producing a total of 75 W and the supercapacitor is being charged with 69.91 W, giving an efficiency of 93.2%. The solar PV cell becomes active after approximately 18 ms, when the available power increases from 75 W up to 195.8 W. The supercapacitor charge rate also increases at this point, to a total of 184.75 W which gives an efficiency of 94.35%. Finally, the wind speed is decreased after 60 m/s, at which point the available power reduces to 149.96 W. At the same time, the supercapacitor charge power reduces to a total of 141.2 W, giving a final efficiency of 94.16%.



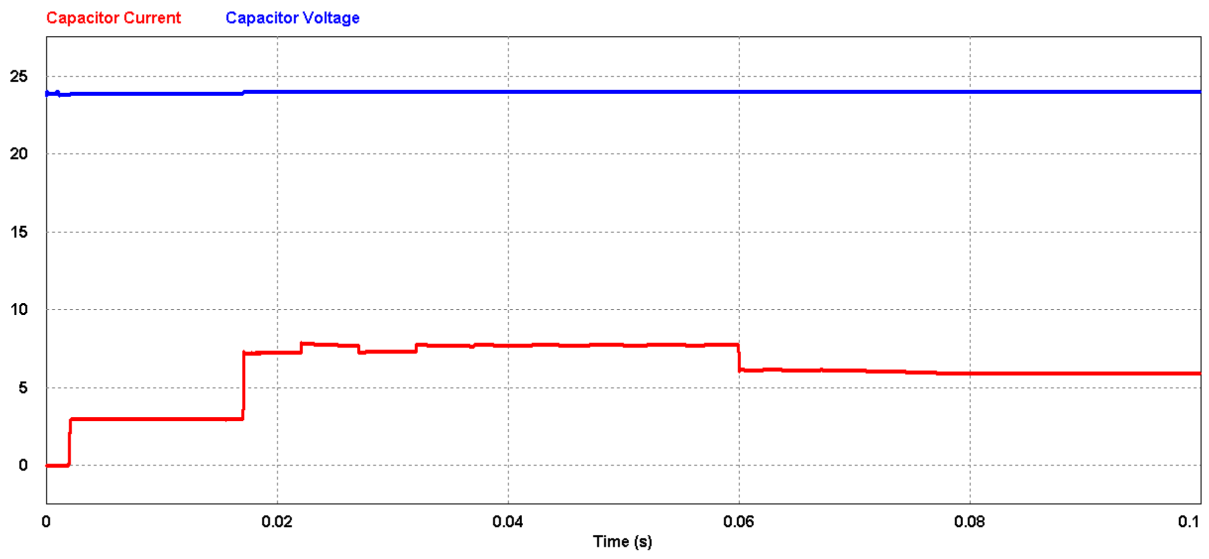


Fig 114 Wind Turbine, Solar PV Cell and Supercapacitor Charging Power with 500 w/m^2 Irradiance and 5 m/s Step Decrease in Wind Speed

Finally, this simulation once again provides an opportunity to demonstrate the regulation of the DC buses in the system. At around 18 ms into the simulation, the solar PV cell is connected to the DC bus, which obviously creates a disturbance. This is shown in Fig 115, where it can be seen that an absolute error of 14 mV (0.029%) occurs, with the output voltage returning to its reference level after 81 µs. It can also be seen from this simulation that this creates a very small disturbance on the output voltage, although this is less than 1 mV. Another disturbance occurs when the wind speed decreases after 60 ms in the simulation. This can be seen in Fig 116, where an error of 8 mV (0.016%) is visible on the DC bus, with this settling back into the steady state of operation after 1 µs. In this scenario, there is also a small disturbance of less than 1 mV on the output bus.

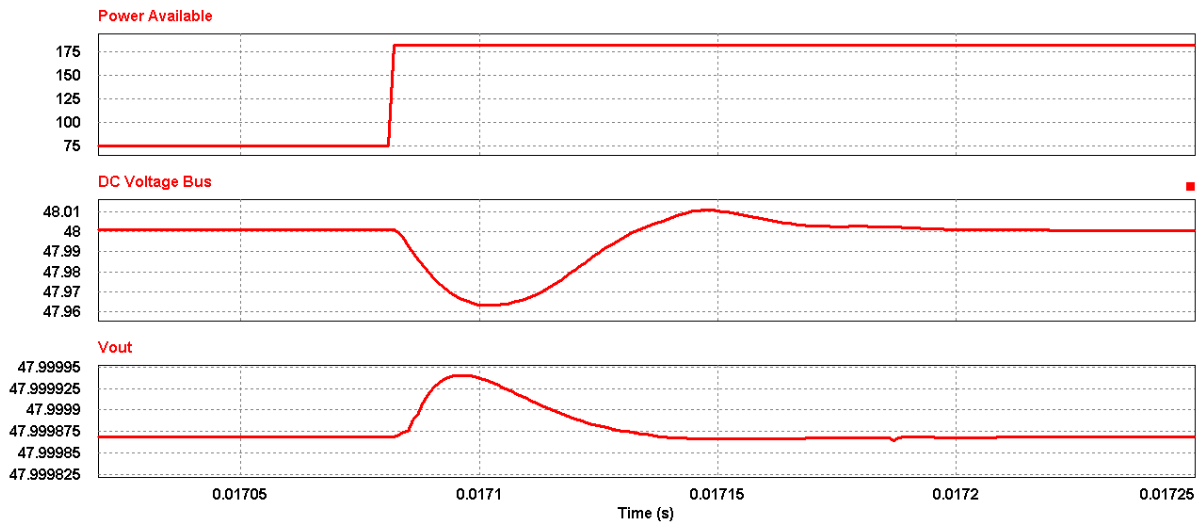


Fig 115 Voltage Regulation During Wind Step Decrease Test when PV Solar Cell Begins Outputting Power

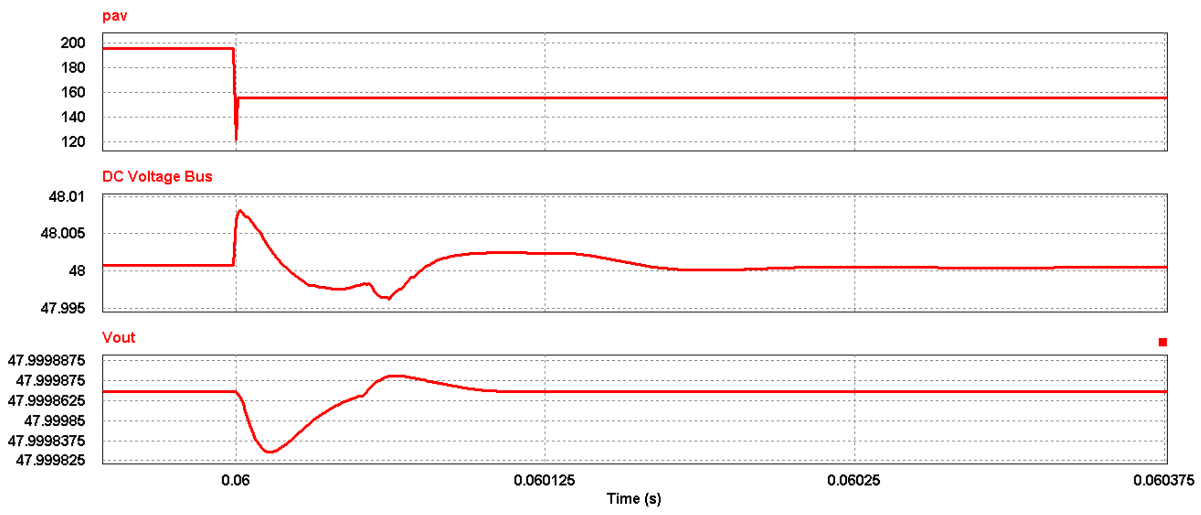


Fig 116 Voltage Regulation when Step Decrease in Wind Speed

6.2.2 Super Capacitor Simulations

As the output of the system is primarily supplied from the supercapacitor, this component serves a crucial function in the system. In this section, a number of simulations are carried out to ascertain the performance of this element. As was stated in the previous chapter, there are a number of assumptions which are made relating to the model of the system. As a result of these assumptions, the inertia of the renewable energy sources is negligible as the intention is to test the performance of the novel control algorithm. In a real system, the inertia of the renewable energy sources would be increased which in turn would increase the response time of the intermediate bus. This is down to the fact that the renewable sources couldn't realistically source the required power as quickly. As a result of this, the intermediate bus would be less stable and this would have a negative effect on the performance of the supercapacitor power converter. In the first simulation, the dynamic characteristics are considered with a small load step of 1A. For this test, the wind speed is set to a fixed value of 6 ms, whilst 500 w/m^2 of irradiance is applied to the solar PV cell. This allows for the system algorithm to charge the supercapacitor with the power being produced by the renewable energy sources and is designed to imitate a typical operating scenario. Once the system has settled into a steady state of operation, a step of 1A is applied to the load. This is firstly done with an increase in current, as shown in Fig 117, where an absolute error of 4.5 mV, or 0.009 %, is seen. This is then repeated with a corresponding decrease in current, as is shown in Fig 118, which results in an absolute error of 3.1 mV, or 0.006 %. As with the tests previously presented in section 5.3.3, the settling time is unmeasurable due to the small size of the error. Furthermore, the test results match with those previously obtained in section 5.3.3 which illustrates how the operation of the power controller doesn't cause a degradation of the power converter operation.

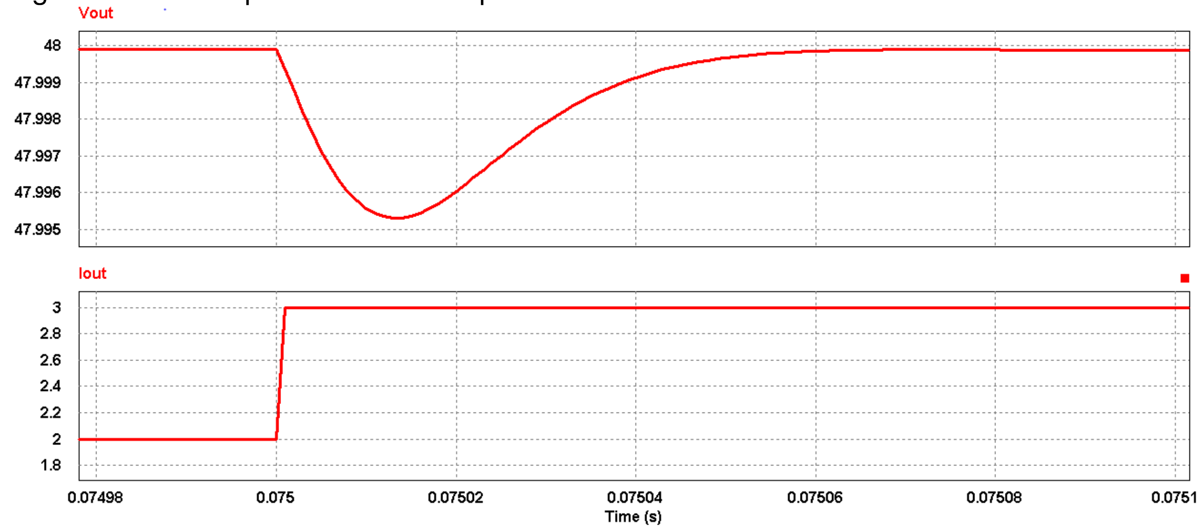


Fig 117 System Output Voltage with a 1A Load Increase

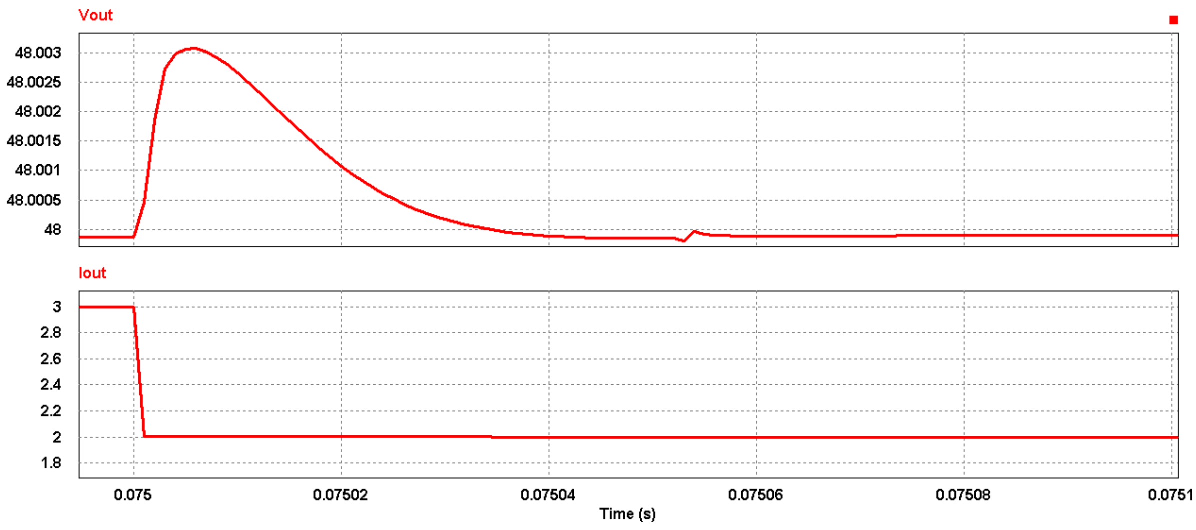


Fig 118 System Output Voltage with a 1A Load Decrease

The second set of simulations for the supercapacitor considers the performance of the converter with a larger load step. As with the previous tests in this section, the renewable energy sources are set to fixed values in order to imitate a typical operating scenario. In this case, the wind speed is set to 10 m/s and the solar irradiance to 800 w/m^2. After the system has settled into its steady state of operation, a 3A load step is then applied to the output. In the first case, this is carried out with an increase in load current, as is shown in Fig 119. This results in an absolute error of 17.5mV, or 0.036%, whilst the output returns to the reference voltage after 45 µs. In the instance where a load decrease is applied, as shown in Fig 120, an error of 11 mV, or 0.023%, is observed whilst the settling time is measured at approximately 20 µs. In both instances, this yields a similar result to that observed in section

5.3.3 which again shows how the power converter operation isn't impacted by the operation of the power controller.

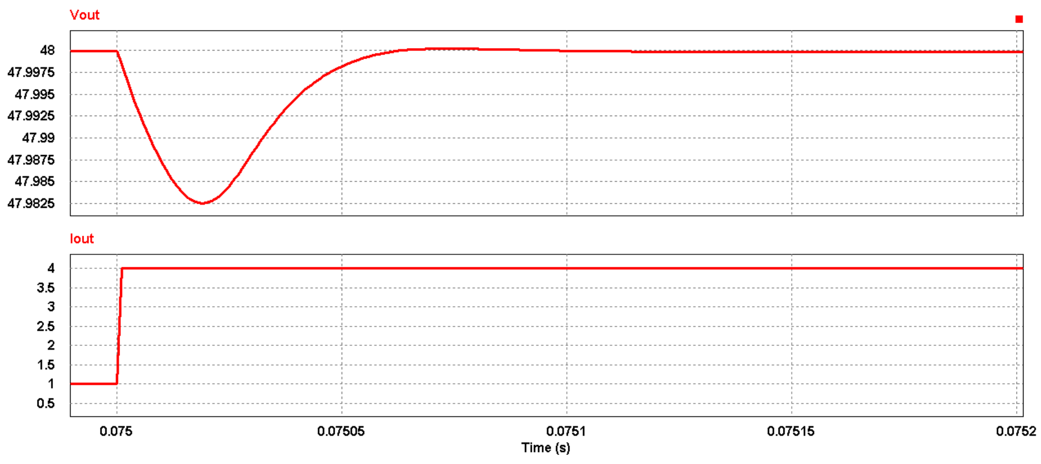


Fig 119 System Output Voltage with a 3A Load Increase

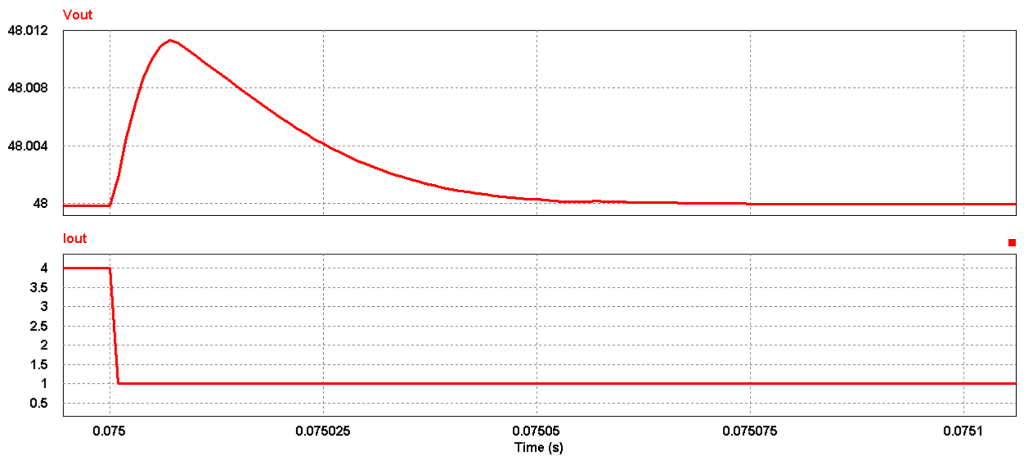


Fig 120 System Output Voltage with a 3A Load Decrease

6.2.3 Battery Simulations

As has previously been discussed, the battery performs an important function within the system as it provides a means of storing energy which in turn improves the power security. In this section, a number of simulations are conducted to demonstrate the performance of the battery controller. In the first set of tests, the charging performance is considered. In this scenario, the supercapacitor is pre-set so that it appears fully charged. The wind speed and the solar irradiance are set to constant values, as is the system load. This is repeated with both a moderate load, which yields a modest rate of charge, and with a small load, which creates a larger charging current.

The results of these tests, as shown in Fig 121 and Fig 122, illustrates that the battery begins charging after approximately 20 ms. As the supercapacitor is pre-set to 32V in this case, this is the time it takes for charging of this device to complete. Once this period has passed, the battery then begins to sink the surplus power which is available within the system. In Fig 121, where only 128 W of power is produced, the battery is charged with

121.7 W at an efficiency of 95%. In the second instance, where 228 W of power is being produced, the battery is charged at a higher rate of 221.9 W with an efficiency of 97.3%.

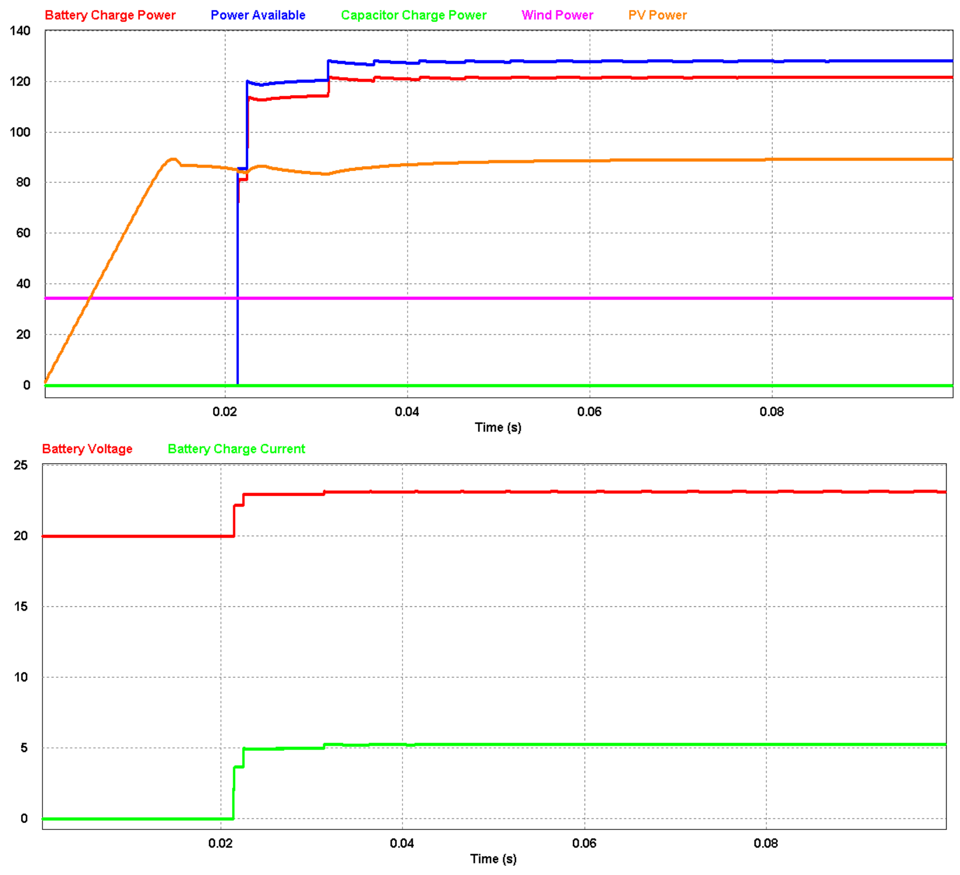


Fig 121 Battery Charging Simulation with a Small Charge Current

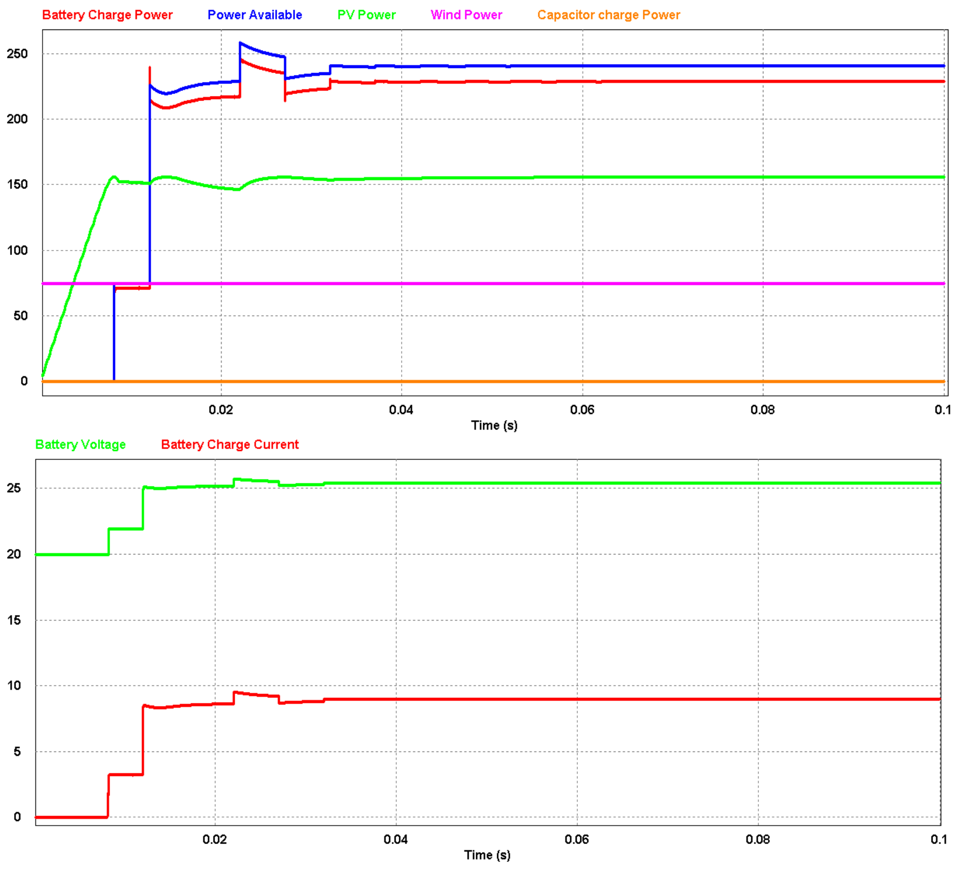
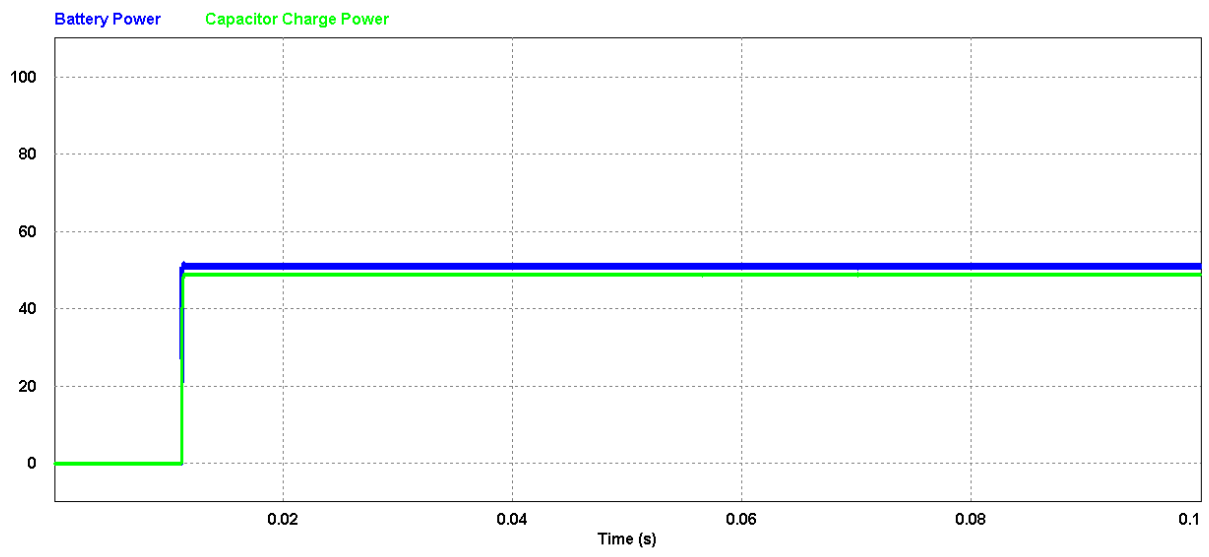


Fig 122 System Battery Charging Simulation with a Large Charge Current

The second set of simulations for the battery subsystem is concerned with the controller performance when the renewable energy sources produce no power. According to the algorithm flowchart presented in Fig 92, the battery should be discharged at a rate which is proportional to the output power of the system when this occurs. This begins when the supercapacitor voltage drops to a sufficiently low level and ends when it enters the constant voltage charging mode. Therefore, the supercapacitor is pre-set to a deeply discharged state as part of these simulations. The renewable energy sources are then set so that they generate no power and the battery discharging can be observed. This is firstly carried out with a 1A load, as is shown in Fig 123, which results in the battery charging the capacitor at a rate of approximately 50W. In the second scenario, a larger load of 4A is applied to the system, as shown in Fig 124, which results in the supercapacitor being charged at 200 W. The results for both of these validate the performance of the algorithm in Fig 92, showing how the battery is discharged at a rate which is sufficient to keep the supercapacitor in a state of charge when the renewables are not producing power.



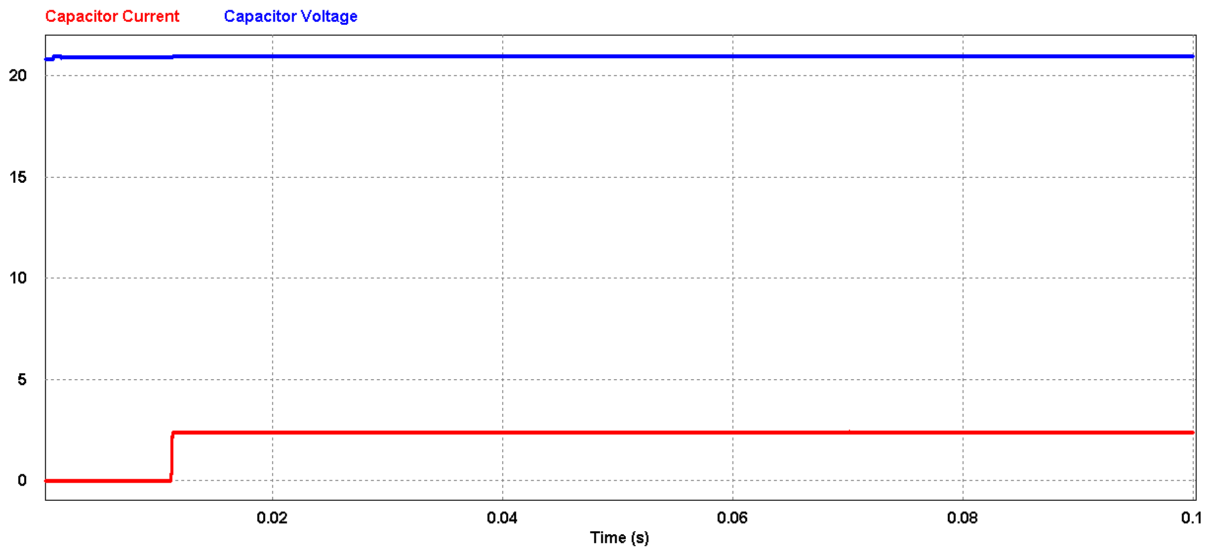


Fig 123 Supercapacitor Being Charged at a Rate of 50W by the Battery

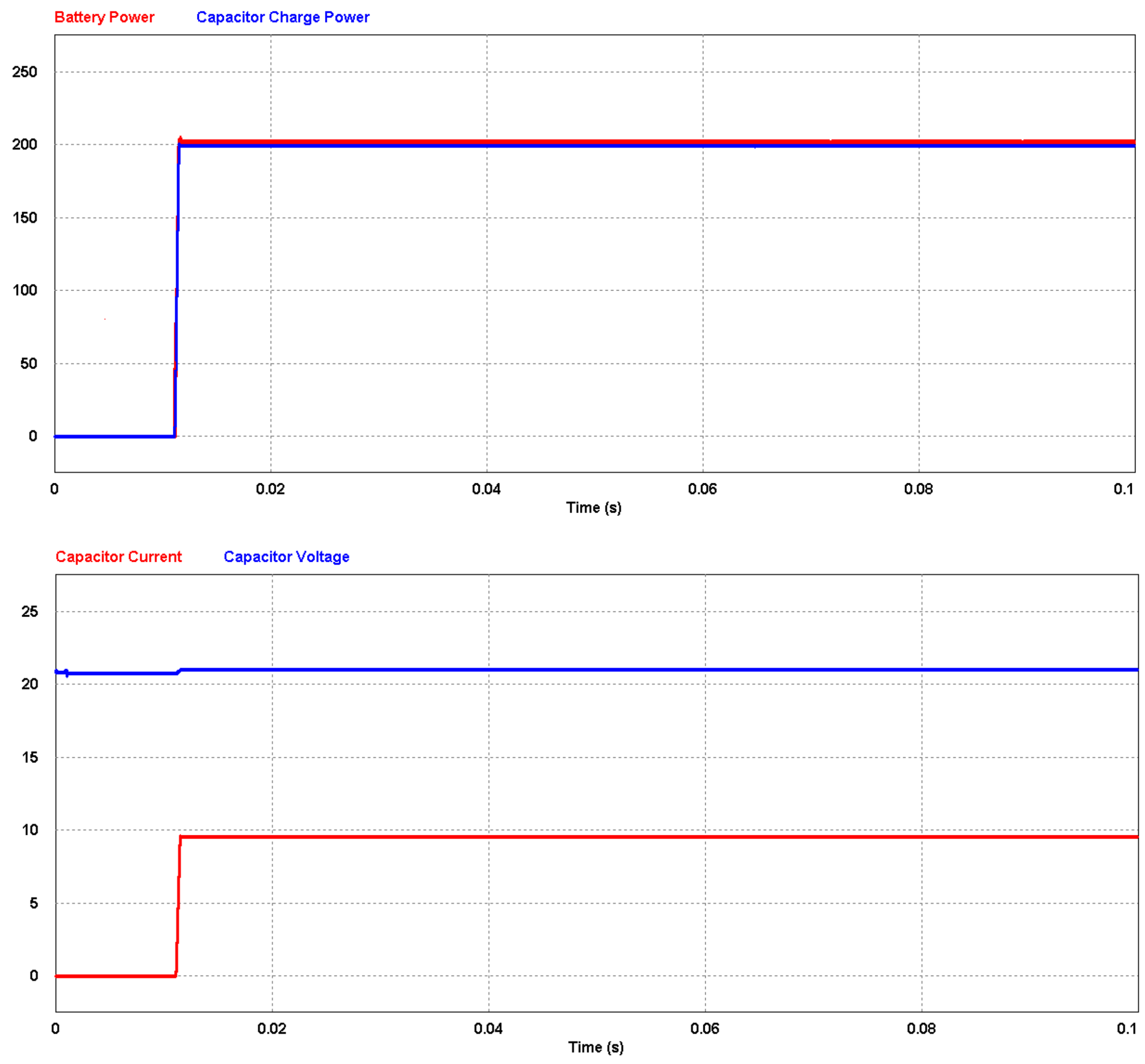
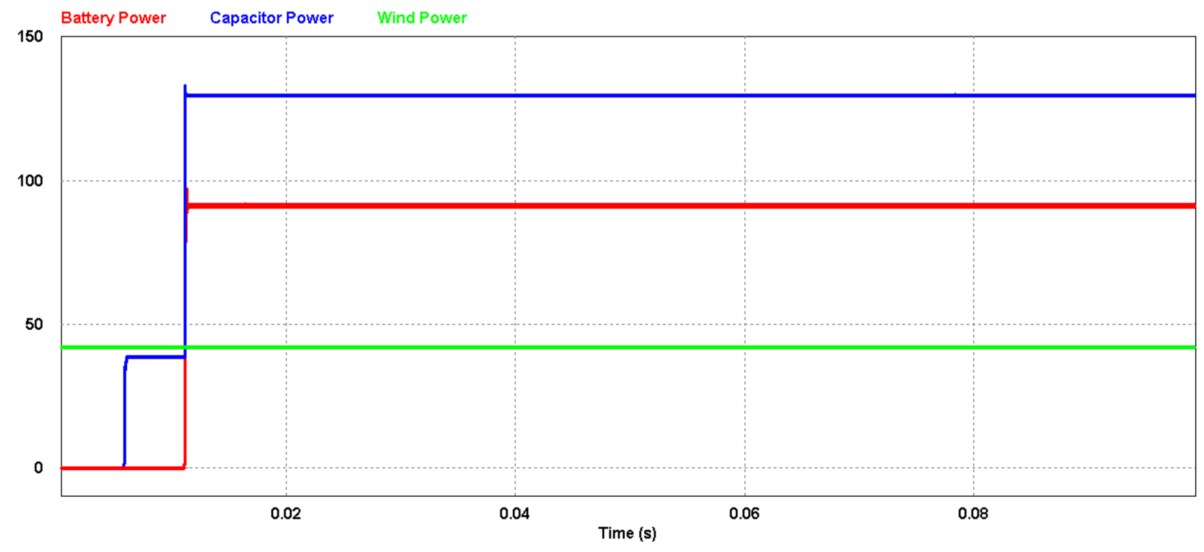


Fig 124 Supercapacitor Being Charged at a Rate of 200W by the Battery

In the final set of simulations for the battery subsystem, the performance of the system is considered when the load is greater than the power being produced by the renewables. In this scenario, the flow chart given in Fig 92 indicates that the battery should begin discharging in order to maintain the supercapacitor SoC. For the first of these tests, the renewables are configured so that the wind turbine produces a small amount of power whilst the solar PV cell contributes no power. In this simulation, as shown in Fig 125, the load is set to 2A to give an output power of 96 W. The supercapacitor is initially charged at a rate of 38 W, all of which is sourced from the wind turbine. At approximately 11 ms, the battery starts discharging at a rate of 92 W which is approximately equal to the load power. The rate of charge consequently increases to 129 W at this point, meaning the supercapacitor is charged with an efficiency of 97 %.



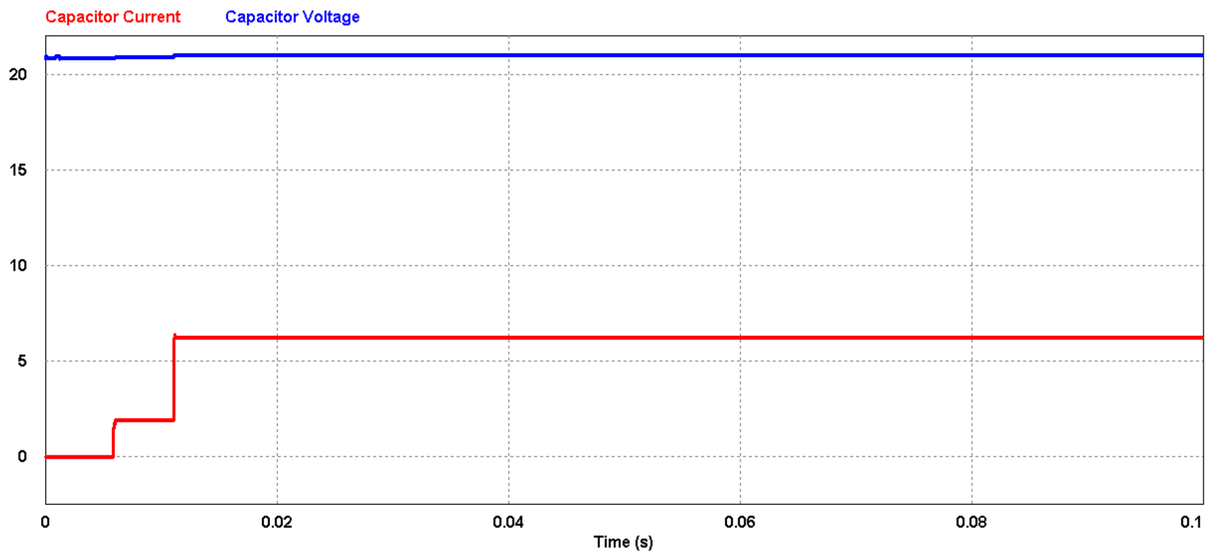
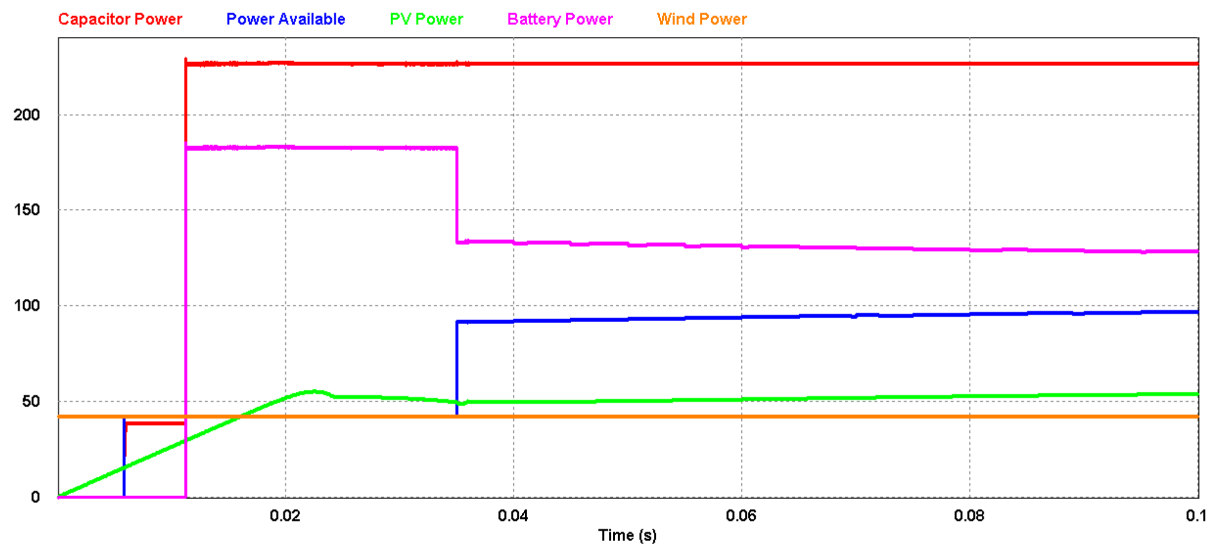


Fig 125 Supercapacitor Being Charged at a Rate of 130W by the Battery and Wind Turbine

For the final simulation in this chapter, the wind turbine is set to 6 m/s and the solar irradiance is set to 200 w/m^2. The supercapacitor voltage is then set to a low value, in order to allow the battery to discharge, and the load is set to 5A. In this scenario, it is expected that the supercapacitor will be charged by all of the power sources in the system. The result of this test, as shown in Fig 126, indicates that the supercapacitor is charged at a rate of 38 W during the early part of the simulation, with this power being sourced solely from the wind turbine. At approximately 11 ms, the battery begins discharging at a rate of 184 W. At this point, the supercapacitor is charged at a rate of 221 W, which gives a total efficiency of 98 %. Finally, the solar PV cell begins producing power after 35 ms, at which point the battery discharge rate decreases in proportion to the PV output, with the supercapacitor rate of charge remaining constant.



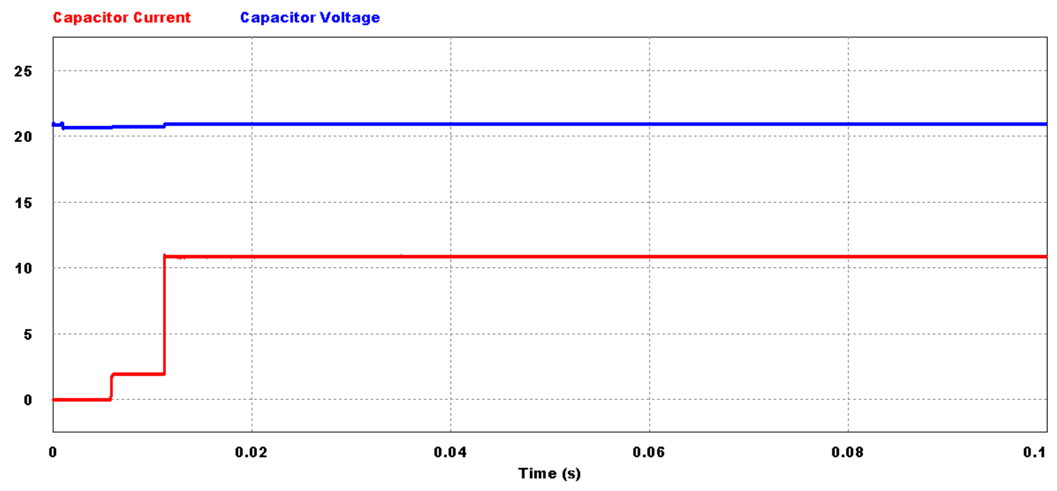


Fig 126 Supercapacitor Being Charged at a Rate of 221W by the Battery, Solar PV Cell and Wind Turbine

6.3 Conclusions

In this chapter, the modelling of the system's power controller has been introduced. The primary objectives of the algorithm have been described and the justification for the use of a state-based approach is discussed. The operation and modelling of this controller are then considered more closely before extensive simulations are carried out. This begins with demonstrations of the performance when only the solar PV cell generates power, giving a worst-case efficiency of 95.5% and a best-case of 98.4%. The system is then considered with only the wind turbine generating power, with an efficiency of between 91.2% and 93.6% being observed. Finally, the operation is considered with a combination of sources generating power, which yields a maximum efficiency of 94.41% and a minimum of 93.4%.

Furthermore, the system's DC buses are observed under a number of transitory conditions as part of the simulations. The system features an intermediary bus, which is used as an interconnection between the renewables, and the main output bus. As the renewable sources are connected and disconnected, the intermediary bus exhibits a worst-case error of 75mV (0.156%) and a maximum settling time of 172 µs. Perhaps more importantly, the output of the system exhibits a worst-case error of 17.5mV (0.036%) and worst-case settling time of 45 µs even under transitory conditions. By way of comparison, the systems presented by Sikkabut et al (2016) and Thoungthong et al (2013) utilise a similar topology to that presented in this project, albeit with a higher power capacity. In both instances, the presented test results show an error of several volts and a settling time of several ms under transitory operating conditions.

The simulation test results illustrate how an efficiency of between 91.2% and 98.4% is achieved within the system. Whilst it's not common for this analysis to be extensively detailed in the research literature, the papers presented by Shen and Khaligh (2015), Anvari-Moghaddam et al (2016), Jia et al (2016) and Shen and Khaligh (2017) indicate that achieving an efficiency of greater than 90% is desirable. Therefore, it can be concluded that the state-based algorithm delivers good efficiency, especially considering the relatively simple nature of this algorithm. It can also be concluded that utilising the supercapacitor to provide compensation for the main output results in a high power quality output, especially when coupled with the novel power converter control solution discussed in the previous chapter.

7 Design of the Novel Power Converter Controller for Digital Hardware Implementation

In the previous two chapters, the model of the DER system has been presented, simulated and discussed in detail. The next stage of the project is the implementation of the novel ANFIS-PI based power converter control algorithm in a digital device. As has previously been discussed, the hardware will be prototyped using a Digilent Zedboard which features a Xilinx Zynq XC7Z020 APSOC running at 100MHz. This chip includes programmable logic (PL) with the same architecture as an Artix-7 FPGA and a dual-core Cortex-A9 ARM based processing system. The majority of the control algorithm is implemented in the FPGA fabric, owing to the improvements in throughput which can be achieved. The CPU is used to provide supervisory functionality, primarily through the configuration of the ANFIS algorithm logic. The architecture of the SoC, including contextualization within the overall system, is shown in Fig 127. A detailed discussion regarding this implementation, including simulation and test results, is presented in the rest of this chapter.

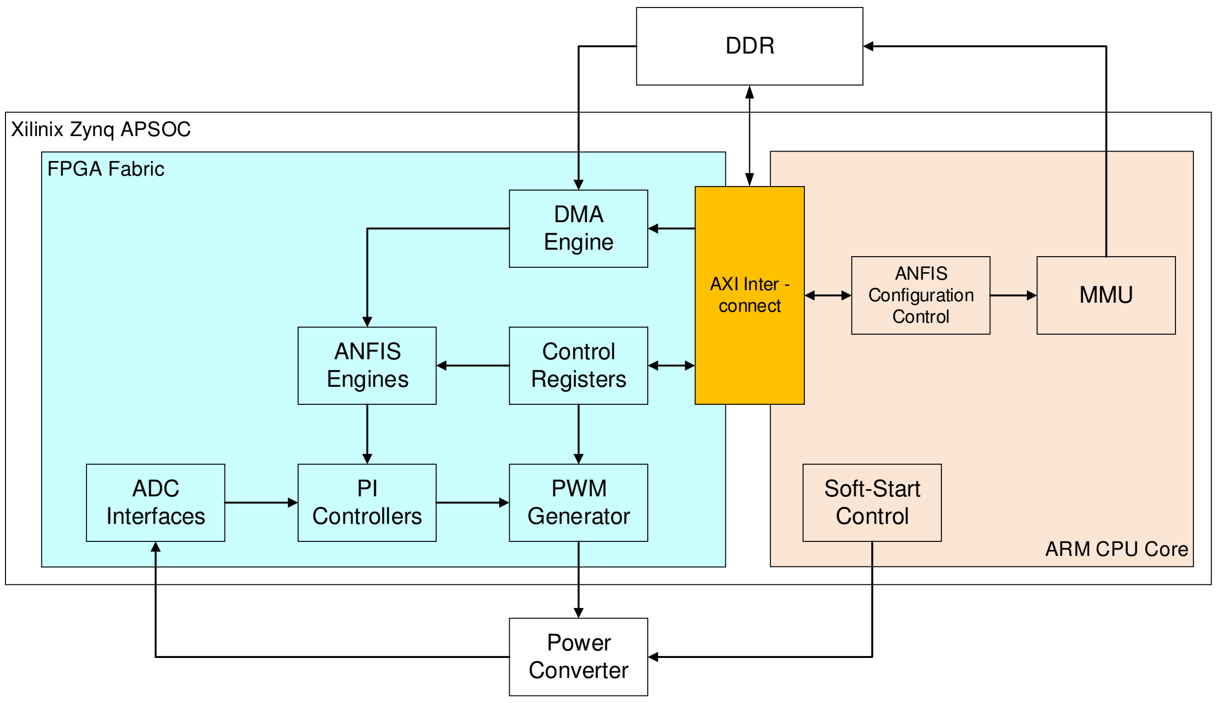


Fig 127 Block Diagram of the APSOC Architecture

7.1 CPU Functionality

The ARM CPU core within the Zynq device is utilised to perform a number of housekeeping functions. There are a number of options available for the development of the CPU which can be broadly classified as “bare metal” or OS-based approaches. When using an OS system, the Linux kernel is the most commonly used tool, although there are popular alternatives. In this section, these different methodologies are discussed before the chosen solution is outlined.

The bare metal methodology benefits from a lack of overhead allowing it to achieve fast program execution. However, as this approach is less widely deployed, there are fewer supported libraries and tools which means the realisation of complex programs can be time-consuming. Conversely, the Linux OS is extremely popular and has found wide use on a number of different platforms. As a result of this, there is a greater amount of supported tools and libraries which can simplify the development of complex applications. The kernel is also designed to be general purpose and highly customisable, meaning it offers a great deal of flexibility. However, Linux based approaches require a much larger amount of overhead than the bare metal method. This leads to a reduction in program execution speed. The Linux kernel features a time slicing scheduler, meaning that tasks are apportioned time slots on the shared CPU resources. Whilst higher priority tasks receive more processing time, this feature makes it unsuitable for applications with tight timing constraints. By comparison, Real-Time Operating Systems (RTOS), such as the popular FreeRTOS software, utilise a scheduler which cedes control of the processing resources to the task with the highest priority. This makes it more suitable for applications which are time sensitive, as tasks which require a fast response can be given unimpeded access to the chip's resources. Systems based on RTOS sit somewhere between bare metal and Linux, offering less flexibility than Linux but also faster execution.

The main task the CPU performs in this application involves configuring the ANFIS algorithm logic in the FPGA fabric. This means reading ADC values from the FPGA fabric, using this data to configure a Direct Memory Access (DMA) controller and finally dispatching control words back to the FPGA fabric. The CPU is also responsible for the control of the soft start circuitry, writing additional control words to the FPGA fabric for this purpose. The exact methodology by which the CPU and FPGA cores communicate with one another is discussed in more detail in section 7.3.

Whilst the use of a full OS based system has a number of advantages, the creation of such systems is also more complex. Given that the functionality of the software for this application is relatively simple, the use of a full OS based system is deemed unnecessary. Therefore, the bare-metal approach is favoured for this application.

7.2 FPGA Design

The FPGA fabric is the primary core used for the implementation of the local power converter controller. This core is preferred to the processing system as it offers higher throughput, which is a desirable characteristic for the ANFIS implementation. The block diagram shown in Fig 128 illustrates the architecture which is used in the FPGA fabric for each of the power converters. As was discussed in chapter 5, the ANFIS algorithms in the power converter feature two inputs – one for the current and one for the derivative of the current. Both of these inputs have four Gaussian input members, meaning there are a total of 16 fuzzy rules in the inference engine. The individual blocks which make up the ANFIS engine within the FPGA are discussed in more detail in the rest of this section.

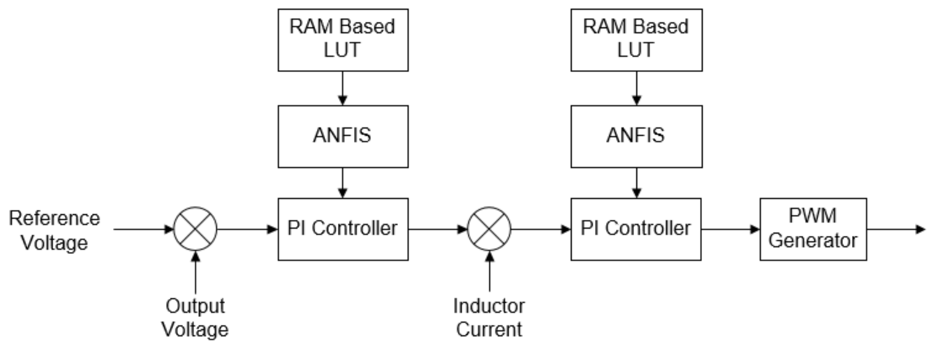


Fig 128 Block Diagram of the Power Converter Controller as Implemented in the FPGA Fabric

7.2.1 ANFIS Algorithm Architecture

There are two main types of ANFIS architecture which have been utilised in FPGAs. One implementation utilises a parallel approach, such as that presented by Gomez-Castaneda et al (2014), whilst the other uses a sequential based approach, such as that presented by Saldana and Silva-Cardenas (2012) and Pande et al (2013). The parallel methodology offers a faster response time than the sequential based approach although this increase in speed requires a much greater logic utilisation. This means that the parallel implementation approach is best suited to applications which require a high throughput. As the throughput requirements for this project are relatively modest, a sequential based implementation approach is favoured. The sequential architecture, as shown in Fig 129, splits the algorithm into four subsystems – a fuzzifier, a permutator, the inference engine and finally a de-fuzzifier, each of which is discussed further in the rest of this section.

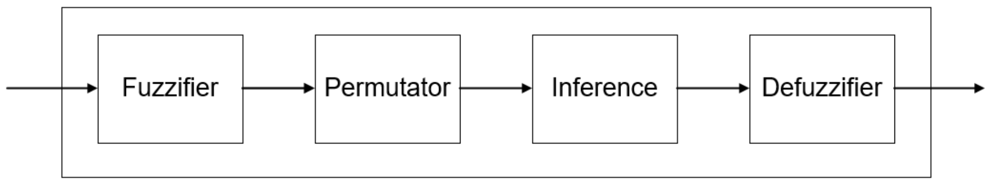


Fig 129 Block Diagram of the Sequential ANFIS Architecture for use in FPGAs

7.2.1.1 Fuzzifier

The fuzzifier block in the ANFIS architecture is responsible for transforming the crisp input values into a set of fuzzy logic values. The fuzzifier in this project uses Gaussian type input members which are implemented using lookup tables. These values are stored in the off-chip RAM rather than in the FPGA fabric BRAM. This is necessary as there is insufficient onboard memory in the FPGA fabric to implement the entire lookup table. The off-board memory is accessed through the DMA controller within the FPGA fabric as is described and modelled in section 7.3.2.

7.2.1.2 Permutator

The permutator block generates every possible permutation of the input members for downstream processing. In the methods presented by Saldana and Silva-Cardenas (2012) and Pande et al (2013), the permutator block takes the form of a pair of circular shift registers which iterates over each of the permutations in turn. This means that the total number of iterations required is equal to the number of input members squared. Whilst this approach is acceptable for applications which use only a small number of membership inputs, this can cause long processing times. This method is also inefficient as it is unlikely that any input will ever occupy all of the possible input members. As members which have a value of zero don't contribute to the output, this introduces a number of redundant clock cycles in the calculation latency. As the ANFIS algorithm in this project utilises four members, meaning that there could potentially be a substantial amount of redundancy, a novel methodology is created with the aim of reducing the latency of the algorithm implementation.

The novel permutator solution utilised in this project replaces the two circular shift registers with a pair of multiplexors. A sequencer circuit is then included which is used to drive the address bits of the two multiplexors. The sequencer is responsible for ensuring that all valid permutations are iterated over. The sequencer doesn't drive the multiplexors directly but instead drives a pair of priority encoders which convert the control logic to a valid address. The full circuit for the novel permutator solution is given in Fig 130.

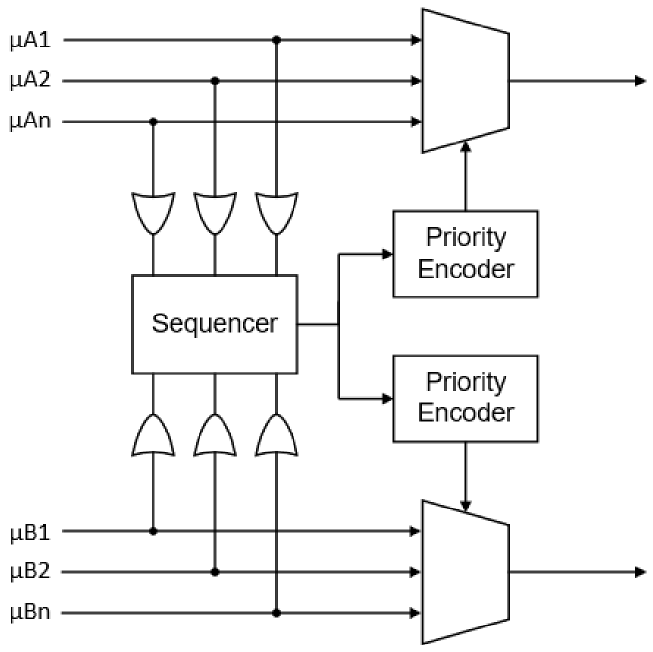


Fig 130 Block Diagram of the Novel Permutator

The first stage in the novel permutator is the identification of input members which have a valid, non-zero value. This is achieved by taking an OR of all of the bits in each of the input members. The result of these OR gates gives a member map which consists of one bit per input member. To facilitate this operation, a simplistic unary OR function is created in VHDL. This function takes a VHDL vector type, such as an unsigned or std_logic_vector, and converters it to a single bit type. An example of such as function is given in Fig 131, showing how all the bits of a std_logic_vector are subjected to an OR operation to produce a std_logic type output. The VHDL code in Fig 132 illustrates how this function can then be utilised to generate the member map.

```
function or_reduce(a : std_logic_vector) return std_logic is
    variable ret : std_logic := '0';
begin
    for i in a'range loop
        ret := ret or a(i);
    end loop;
    return ret;
end function or_reduce;
```

Fig 131 VHDL Function to determine when a Signal is a Non-Zero Value

```
vkp_register_members:
process (fis_clk) is
begin
    if rising_edge(fis_clk) then
        if (resetn  = '0') then
            vkp_stdy_map    <= (others => '0');
        elsif (dma_en_sync = '1') then
            -- Create the map showing the non-zero input members
            vkp_stdy_map(3)     <= or_reduce(vkp_stdy_mem4);
            vkp_stdy_map(2)     <= or_reduce(vkp_stdy_mem3);
            vkp_stdy_map(1)     <= or_reduce(vkp_stdy_mem2);
            vkp_stdy_map(0)     <= or_reduce(vkp_stdy_mem1);
        end if;
    end if;
end process vkp_register_members;
```

Fig 132 VHDL Code Showing How a Member Map is Created That Indicates Which Members are Non-Zero Values

The sequencer within the permutator registers this member map and then clears one of the bits which are set to 1b in each clock cycle. Once all the bits in the first member map have been cleared, it is restored to the original value and one of the bits in the second map is cleared. This process is repeated until all of the valid permutations have been iterated over. An example of the operation of this sequencer is shown in Table 14, with the x1 input having three active members and the x2 input two. This example requires a total of six clock cycles to iterate over each of the possible combinations. It can be seen how the sequencer clears one bit of the x1 input in each of the first three clock cycles. At this point, one of the x2 input bits is cleared and the value of x1 is restored to its original state. The sequence is then repeated on the x1 input until all of the possible permutations have been iterated over.

Clock Cycles	x1 Sequence	x2 Sequence
1	01110	01100
2	01100	01100
3	01000	01100
4	01110	01000
5	01100	01000
6	01000	01000

Table 14 Example of the Sequencer Outputs for a Two Input Five Member ANFIS

The multiplexer inputs are driven via a priority encoder which gives a binary representation of the least significant bit which is set in the sequencer output. The logic table given in Table 15 show the priority encoder operation for a four-bit input. The VHDL snippet in Fig 133 further shows how the priority encoder is implemented and how the multiplexer address bits are subsequently driven.

In4	In3	In2	In1	In0	Out2	Out1	Out0
0	0	0	0	0	0	0	0
x	x	x	x	1	0	0	1
x	x	x	1	0	0	1	0
x	x	1	0	0	0	1	1
x	1	0	0	0	1	0	0
1	0	0	0	0	1	0	1

Table 15 Five Input Priority Encoder Logic Table

```
-- priority encoder for the steady input
steady_addr <=  ADDR_ONE when steady_temp_addr(0) = '1' else
                ADDR_TWO when steady_temp_addr(1) = '1' else
                ADDR_THREE when steady_temp_addr(2) = '1' else
                ADDR_FOUR when steady_temp_addr(3) = '1' else
                (others => '0');

-- priority encoder for the delta input
delta_addr  <=  ADDR_ONE when delta_temp_addr(0) = '1' else
                ADDR_TWO when delta_temp_addr(1) = '1' else
                ADDR_THREE when delta_temp_addr(2) = '1' else
                ADDR_FOUR when delta_temp_addr(3) = '1' else
                (others => '0');

-- Mux for the steady membership
with steady_addr select steady_temp <=
    steady_mem_reg(0) when ADDR_ONE,
    steady_mem_reg(1) when ADDR_TWO,
    steady_mem_reg(2) when ADDR_THREE,
    steady_mem_reg(3) when ADDR_FOUR,
    (others => '0') when others;

-- Mux for the delta membership
with delta_addr select delta_temp <=
    delta_mem_reg(0) when ADDR_ONE,
    delta_mem_reg(1) when ADDR_TWO,
    delta_mem_reg(2) when ADDR_THREE,
    delta_mem_reg(3) when ADDR_FOUR,
    (others => '0') when others;
```

Fig 133 VHDL Code Showing the Implementation of the Novel Permutator Multiplexers and Priority Encoders

7.2.1.3 Inference Engine

The inference engine performs the majority of the arithmetic in the algorithm, calculating the values of both the numerator and denominator. There are three distinct functions which are performed within this block. The first of these is the calculation of the denominator value. This is realised through the use of a digital multiplier circuit which performs a multiplication of the permutator circuit outputs.

The second function of the inference engine is the calculation of the consequence parameters in the ANFIS algorithm. In this block, a ROM LUT table is utilised to retrieve the values which are required for the calculation. The permutator passes the priority encoder output to the inference engine to use as the address in the LUT. This value is passed through an address decoder and used to retrieve the relevant parameters from the ROM.

Basic digital processing blocks then utilise this data in order to calculate the consequence parameters. The full circuit used in this block is given in Fig 134.

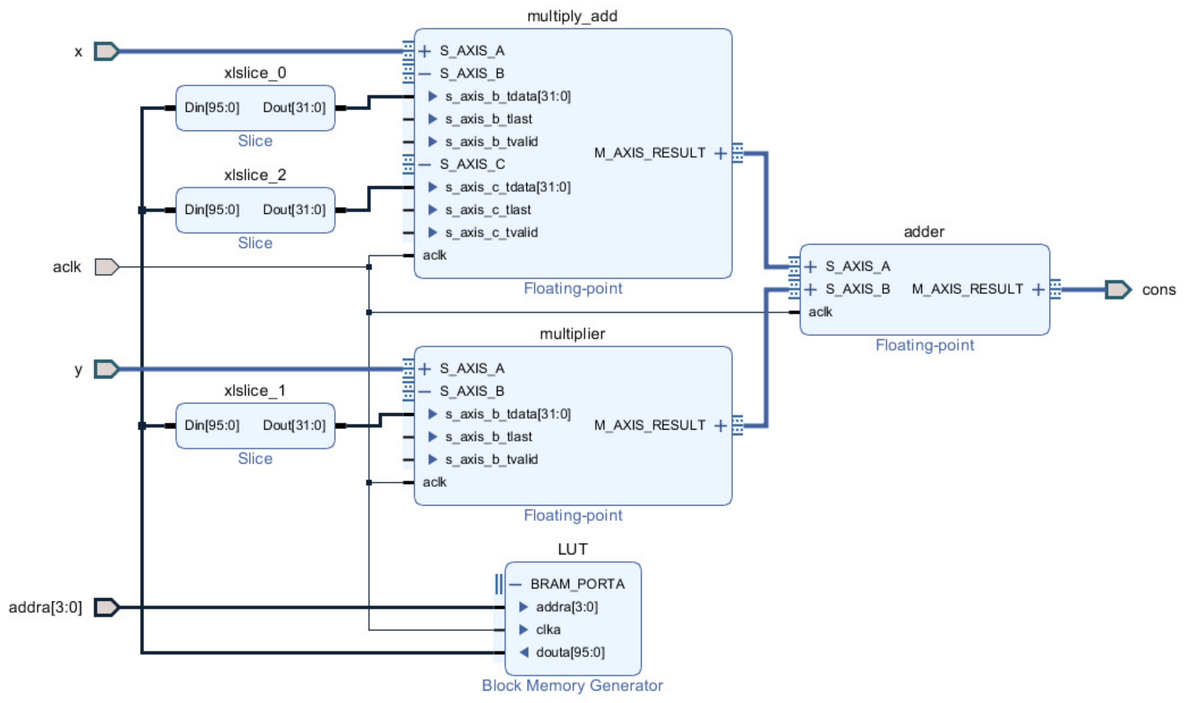


Fig 134 Consequence Parameter Circuit Diagram Taken from Xilinx Vivado Software

The final function of the inference engine is the calculation of the numerator value. This function is simple to implement, consisting of a single digital multiplier circuit. This multiplier takes the output of the inference engine as its inputs. The block diagram shown in Fig 135 gives the full functionality of the inference engine in the digital ANFIS block.

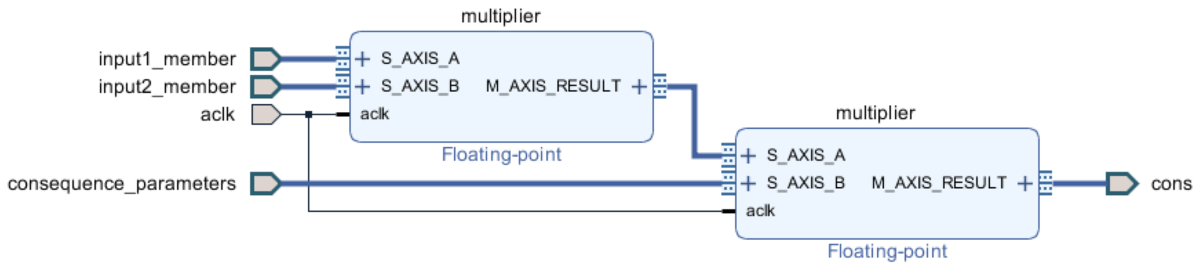


Fig 135 Inference Engine Circuit Diagram Taken from Xilinx Vivado Software

7.2.1.4 Defuzzifier Block

The final block in the digital ANFIS architecture is the defuzzifier circuit which generates the output of the algorithm. This block accumulates the numerator and the denominator as calculated by the inference engine. Once all iterations of the permutator have been performed, a divider block performs the final piece of arithmetic using the accumulated values. The block diagram for the defuzzifier is given in Fig 136.

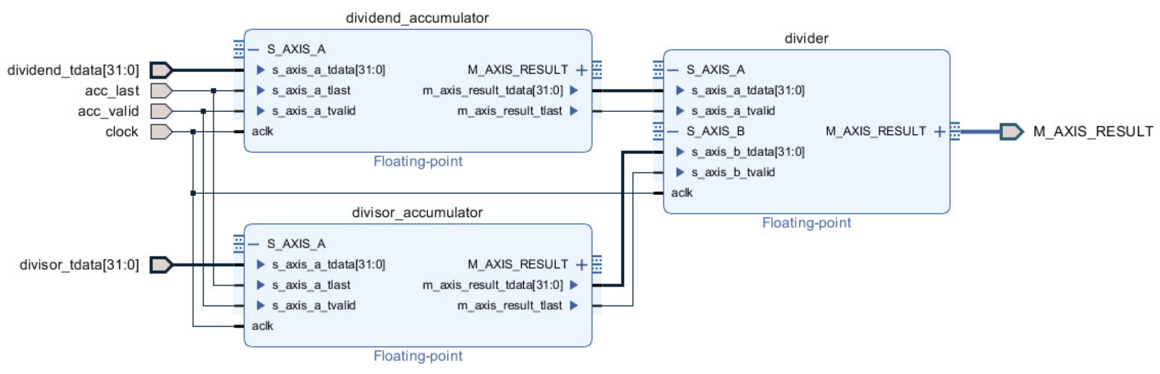


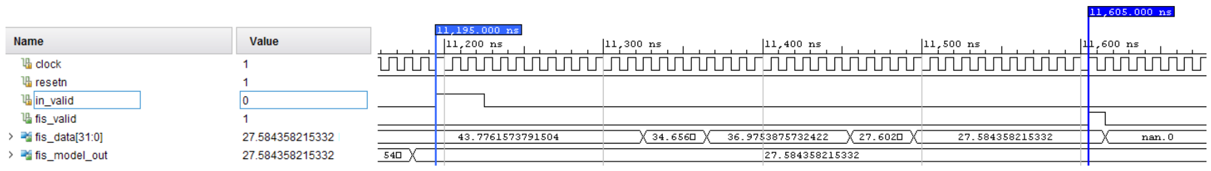
Fig 136 Defuzzifier Circuit Block Diagram as Taken from Xilinix Vivado Software

7.2.1.5 Simulation and Hardware Testing

In order to verify the novel ANFIS architecture, a VHDL model of this has been developed. In addition to this, a model of the existing approach has been additionally created so that the performance benefits of the new technique can be demonstrated. Simulations of both of these are carried out using the Xilinix Vivado design suite. This is a free tool from the chip vendor which allows for the synthesis of VHDL code, the integration of pre-existing IP blocks and includes a simulation environment called XSim. The inputs to the ANFIS are set to random values by the stimuli, with the output of the block being compared to a master model which has equivalent behaviour to algorithms developed within MATLAB. In this section, just one of the systems ANFIS engines is considered, the proportional gain scheduler for the PV subsystem. However, as the basic structure is consistent, the results are representative of all the engines in the system. The ANFIS considered here has two inputs both of which have four membership groups.

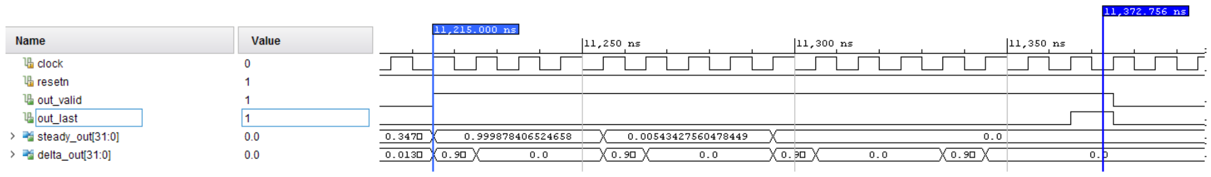
One key feature of the novel ANFIS approach which is demonstrated in these simulations is the reduction in clock cycles required per calculation. In the sequential methodologies presented by Saldana and Silva-Cardenas (2012) and Pande et al (2013), a fixed number of clocks will be required. This is demonstrated in Fig 137, where it can be seen that a total of 41 clock cycles are required for a calculation. Conversely, the total clock cycles required for the novel approach is variable between a minimum of 27, as shown in Fig 139, and a maximum of 37, as shown in Fig 140. The contrasting operation here is entirely attributable to the different permutator architectures. In the existing methodology, a total of 16 iterations are required to give every possible input combination, as is shown in Fig 138. In contrast, the number of iterations required in the novel approach is dependent upon the values of the membership groups for any given input. In the minimum case, there are non-zero values for one group in the first input and one in the second input. This means that the total number of valid iterations is equal to two in this case, as is demonstrated in Fig 141. In the maximum case, there are non-zero values for all four groups in the first input and three members in the second input. This means a total of twelve iterations are requisite for the permutator operation in this instance, as shown in Fig 142. It can be seen from this that the reduction in the latency of the ANFIS calculation is traceable to the removal of between fourteen and four redundant clock cycles in the permutator hardware.

As the number of members used in this example is relatively small, the clock saving is fairly modest. This is mainly attributable to the fact that the number of redundant clock cycles is proportional to the number of input members. This is due to the fact that they must cover a greater amount of the input area meaning that a higher proportion must contain at least some degree of membership for any given input. However, for more complicated applications with more input members, the input mappings cover a smaller amount of the input space and a smaller proportion of these members are active for any input. Consequently, there will be a greater amount of redundancy and, therefore, a greater reduction in latency.



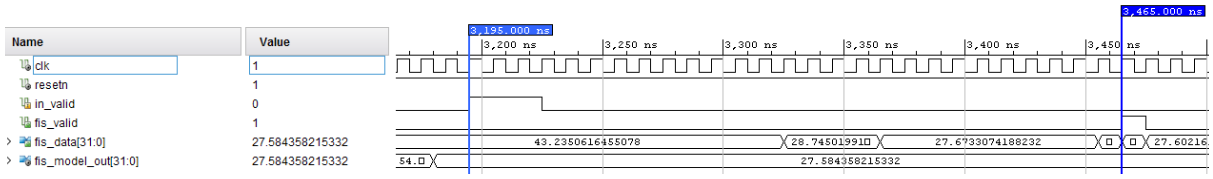
Name	Value
clock	1
resetn	1
in_valid	0
fis_valid	1
fis_data[31:0]	27.584358215332
fis_model_out	27.584358215332

Fig 137 Existing ANFIS Calculation Showing 41 Clock Cycles Between the Start and End Calculation Pulses



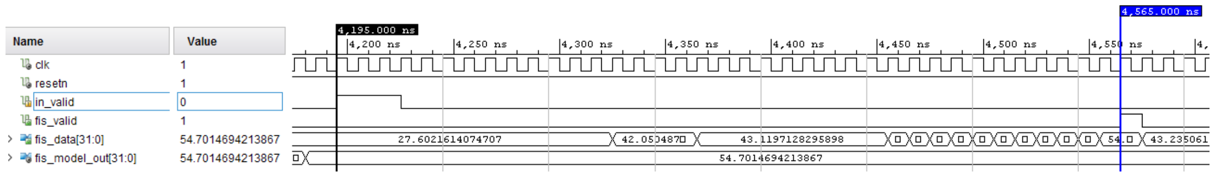
Name	Value
clock	0
resetn	1
out_valid	1
out_last	1
steady_out[31:0]	0.0
delta_out[31:0]	0.0

Fig 138 Existing Permutator with Sixteen Clock Cycles, as shown by the Assertion Time of the Out Valid Signal



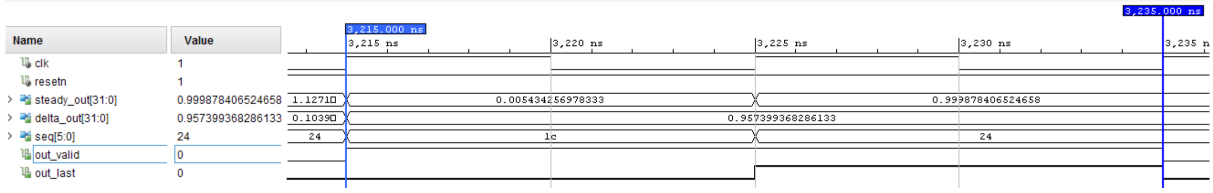
Name	Value
clk	1
resetn	1
in_valid	0
fis_valid	1
fis_data[31:0]	27.584358215332
fis_model_out[31:0]	27.584358215332

Fig 139 Novel ANFIS Calculation Showing 27 Clock Cycles Between the Start and End Calculation Pulses



Name	Value
clk	1
resetn	1
in_valid	0
fis_valid	1
fis_data[31:0]	54.7014694213867
fis_model_out[31:0]	54.7014694213867

Fig 140 Novel ANFIS Calculation Showing 37 Clock Cycles Between the Start and End Calculation Pulses



Name	Value
clk	1
resetn	1
steady_out[31:0]	0.999878406524658
delta_out[31:0]	0.957399368286133
seq[5:0]	24
out_valid	0
out_last	0

Fig 141 Novel Permutator with Two Clock Cycles, as shown by the Assertion Time of the Out Valid Signal

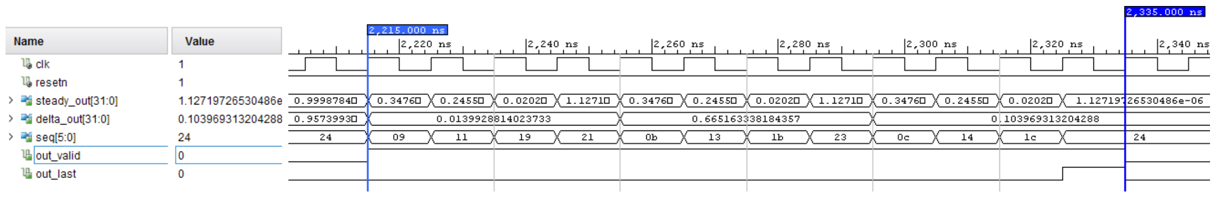


Fig 142 Novel Permutator with Fourteen Clock Cycles, as shown by the Assertion Time of the Out Valid Signal

The Vivado tool additionally includes an Integrated Logic Analyser (ILA) IP core which allows for simplified FPGA hardware testing. This core is capable of capturing the FPGA signals and outputting this data to a PC using the JTAG interface. An onboard stimulus generator has been created, which emulates the behaviour of the VHDL test bench. This is then committed to silicon and the ILA is utilised to capture and display the resultant logical waveforms in Vivado. The results of the hardware tests are shown in Fig 143 and Fig 144 confirming that the hardware operation is analogous to the VHDL model.

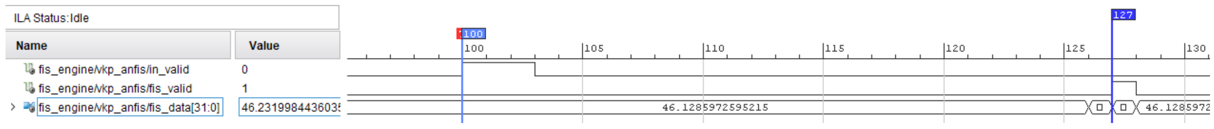

ILA Status:Idle
Name
Value
fis_engine/vkp_anfis/in_valid
fis_engine/vkp_anfis/fis_valid
fis_engine/vkp_anfis/fis_data[31:0]
100
127

Fig 143 Novel ANFIS Calculation Showing 27 Clock Cycles Between the Start and End Calculation Pulses During Hardware Testing

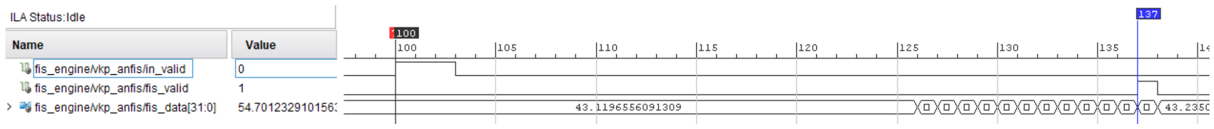

ILA Status:Idle
Name
Value
fis_engine/vkp_anfis/in_valid
fis_engine/vkp_anfis/fis_valid
fis_engine/vkp_anfis/fis_data[31:0]
100
137

Fig 144 Novel ANFIS Calculation Showing 37 Clock Cycles Between the Start and End Calculation Pulses During Hardware Testing

7.2.1.6 Timing and Utilisation Analysis

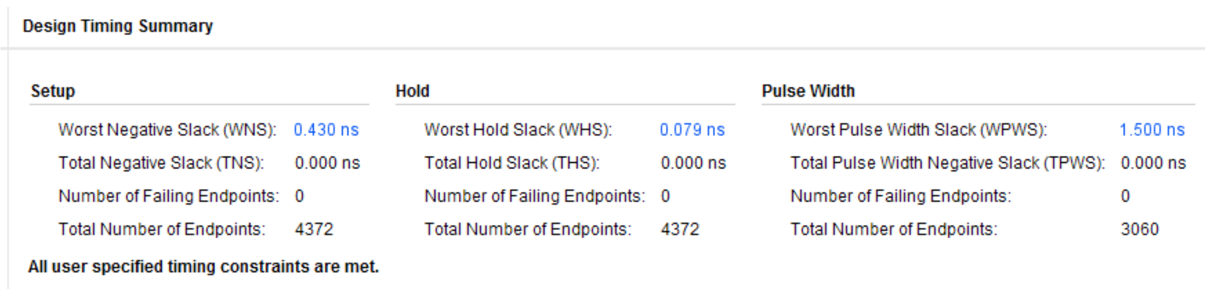
Using the Xilinix Vivado software, it is possible to transform the previously developed VHDL models into an actual array of gates for the FPGA fabric. Analysis can then be performed on the resultant hardware, allowing for a comparison to be made between the existing and the novel implementations. The results of these analyses are detailed in this section.

The FPGA fabric on the zynq device is primarily composed of logical slices, all of which consist of a number of LUTs, multiplexors and registers. The FPGA fabric additionally features a number of DSP slices, which are designed to perform fast math operations, dedicated block RAM, clock management circuits and a pair of ADCs. The utilisation figures for these different resources are given in Table 16 for both the new and the existing ANFIS implementations. As both of these implementation techniques have similar structures, it is unsurprising that both approaches have similar utilisation figures. However, the permutator in the novel implementation technique requires a slight increase in combinatorial logic, as was previously discussed, to remove the redundant cycles. The synthesis shows that this increase is only small, amounting to an increase of just 2.19% in LUTs and 0.38% in the number of registers required. This demonstrates that the novel approach can improve the performance of the ANFIS algorithm without requiring more than a modest increase in the amount of logic which must be employed.

	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	DSPs
Novel Permutator	3868	2640	8	1	21
Existing Permutator	3785	2630	8	1	21
Increase	83	10	0	0	0
Percentage Increase	2.19	0.38	0	0	0

Table 16 Logic Utilisation for the Different ANFIS Implementation Approaches

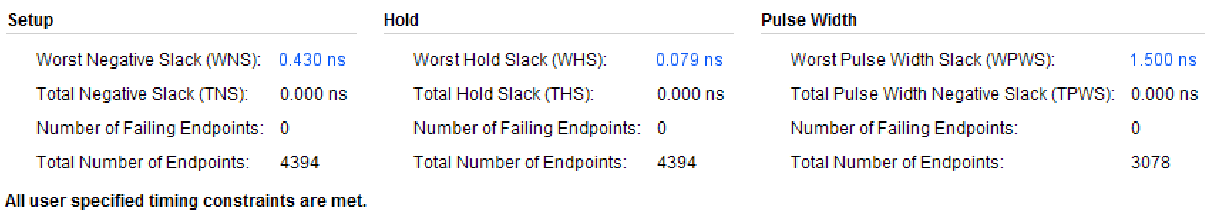
The FPGA fabric on the Zedboard is clocked at a rate of 100MHz. Using this information, it is possible to perform a static timing analysis (STA) of both implementation techniques. This is a particularly important consideration for the novel approach as it introduces extra combinatorial logic chains within the permutator block to speed up its operation. This typically leads to degradation in the temporal performance of circuits within FPGA devices. In Fig 145 a summary of the STA for the existing architecture is presented, giving a worst case slack of 430 ps for flip-flop setup time and a corresponding hold time slack of 79 ps. This is identical to the values observed in the STA for the novel implementation approach, as is given in Fig 146. Similarly, the pulse width slack for both of these implementations is given as 1.5 ns. Furthermore, the longest logic chains in the design are located within the IP blocks which are used to implement the floating-point DSP functions. As an example of this, the multiply-add block given in Fig 134 is deemed to include the logic chain with the worst timing performance in the FPGA fabric, as is shown in Fig 147. As these IP blocks allow for further pipelining stages to be added, this implementation technique could equally be targeted at a higher frequency design with little additional effort. It can be concluded from this STA analysis that the extra combinatorial logic required in the novel permutator doesn’t have any ill effect on the timing performance of the device. It can also be concluded that both designs are equally capable of meeting the timing constraints associated with the 100MHz FPGA clock, with extra pipelining being required for higher clock speeds.


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.430 ns	Worst Hold Slack (WHS):	0.079 ns	Worst Pulse Width Slack (WPWS):	1.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4372	Total Number of Endpoints:	4372	Total Number of Endpoints:	3060

All user specified timing constraints are met.

Fig 145 Summary of the STA for the Existing ANFIS Implementation


Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.430 ns	Worst Hold Slack (WHS):	0.079 ns	Worst Pulse Width Slack (WPWS):	1.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4394	Total Number of Endpoints:	4394	Total Number of Endpoints:	3078

All user specified timing constraints are met.

Fig 146 Summary of the STA for the Novel ANFIS Implementation

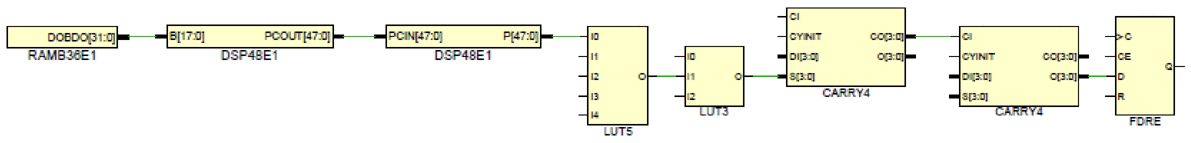


Fig 147 Circuit Diagram of the Worst Timing Path in the ANFIS FPGA Implementations

7.2.2 PWM Generator

The PWM generator module provides the interface between the digital FPGA circuitry and the analogue power electronics. The PWM signal has an operating frequency of 1MHz, whilst a high resolution is also required to achieve optimal control of the system. In a practical application, a 1MHz frequency is unlikely to be utilised as the switching losses in MOSFET transistors are affected by the switching frequency. Therefore, it is preferable to employ a slower switching rate, typically less than 100KHz, so that the efficiency of the power converter is improved. However, as the focus of this research work is the creation of a novel control solution, a higher switching frequency has been used to more thoroughly challenge the developed algorithms.

A number of methods exist and have been presented in the research literature for the generation of high-frequency PWM signals. These methods include the classical counter-comparator based approach, such as that presented by Ishizuka et al (2014) and Corradini et al (2010), delay line approaches (Li et al (2007) and Trescases et al (2005)) and hybrid approaches (Patella et al (2003) and Krug et al (2015)). However, these architectures all have inherent limitations which make them unsuitable for this application. In the case of the counter-comparator approach, there is a need for an extremely high frequency in the counter if a high resolution is to be achieved. The delay line and hybrid approaches both require delay cells, which are not readily available in the FPGA and would also occupy a large amount of area.

A newer approach which has emerged utilises the clock management capabilities of the FPGA. An example of this approach is presented by Batarseh et al (2009), in which the digital clock management (DCM) circuits are used to generate phase shifted clocks of different frequencies. These clocks are then used to reset the flip-flop which drives the PWM output, with a multiplexor selecting the required signal. A low-resolution counter-comparator circuit is used to set the flip-flop, with the counter utilised in this paper being two bits wide. A similar approach to this is presented by Guo et al (2009), with the addition of a sigma-delta modulator on the input. This has the effect of causing small oscillations in duty cycle from one cycle to the next. These small oscillations allow for the averaged duty cycle to be more precisely controlled which improves the effective resolution. The PWM generator utilised in this project is based on this approach, with the block diagram for the PWM circuit being shown in Fig 148. It should be noted that in a practical implementation, this PWM architecture would also require a block which limits the minimum pulse period to avoid damaging the transistor.

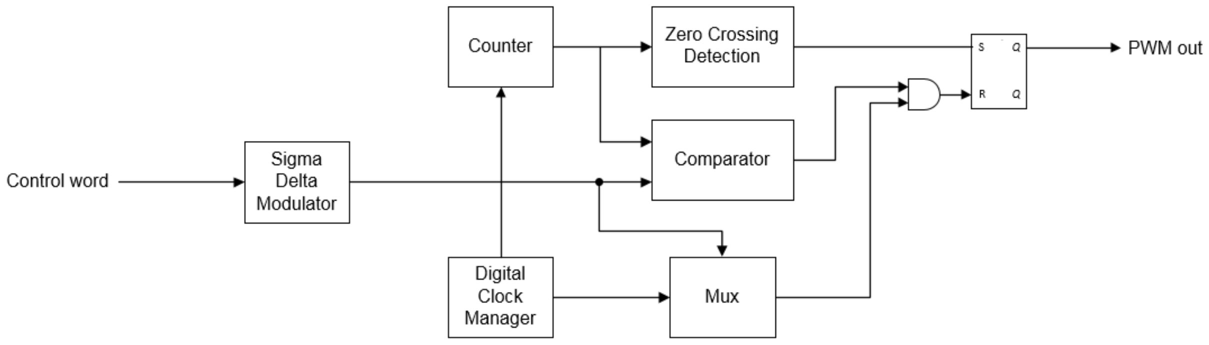


Fig 148 PWM Generator Block Diagram

Lukic et al (2007) and Li et al (2011) indicate that the sigma-delta modulator can introduce so-called idle tones, which are caused by steady-state instability. To overcome this issue a multi-stage modulator, based on that presented by Guo et al (2009), is used in the PWM generator. This approach sees a pair of first-order modulators cascaded to create a second-order modulator. According to Guo et al (2009), this method offers improved performance compared to the use of a single second-order modulator with equivalent performance. Through the use of this cascaded architecture, it is possible to achieve superior noise shaping capabilities whilst maintaining stability. The PWM generator is controlled through a 12-bit input signal. The circuit for the sigma-delta modulator is given in Fig 149 and the RTL code implementing this circuit is given in Fig 150.

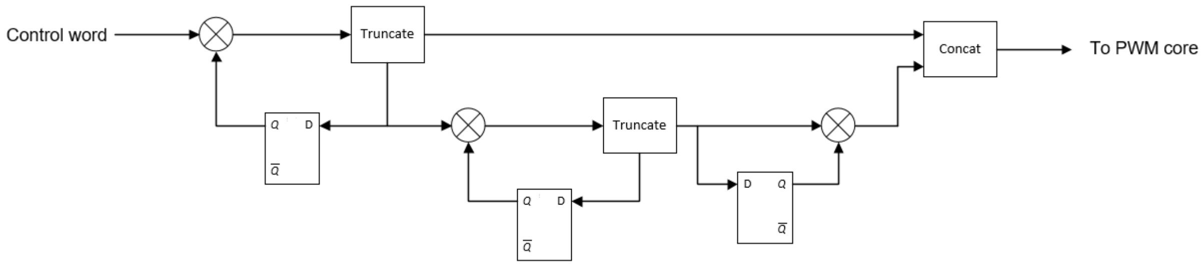
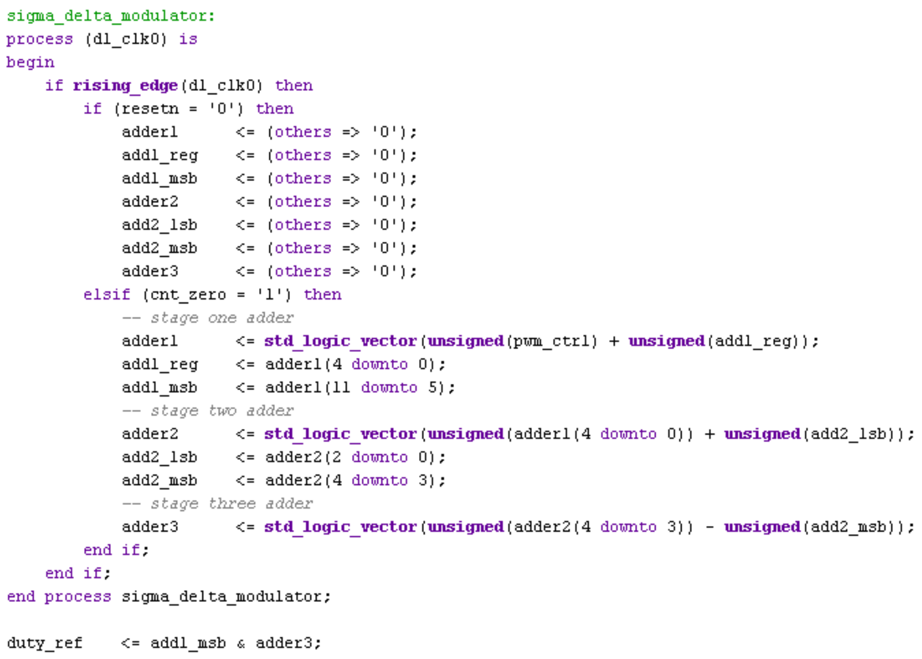


Fig 149 Sigma Delta Modulator Circuit



```vhdl
sigma_delta_modulator:
process (dl_clk0) is
begin
    if rising_edge(dl_clk0) then
        if (resetn = '0') then
            adder1      <= (others => '0');
            add1_reg    <= (others => '0');
            add1_msb    <= (others => '0');
            adder2      <= (others => '0');
            add2_lsb    <= (others => '0');
            add2_msb    <= (others => '0');
            adder3      <= (others => '0');
        elsif (cnt_zero = '1') then
            -- stage one adder
            adder1      <= std_logic_vector(unsigned(pwm_ctrl) + unsigned(add1_reg));
            add1_reg    <= adder1(4 downto 0);
            add1_msb    <= adder1(11 downto 5);
            -- stage two adder
            adder2      <= std_logic_vector(unsigned(adder1(4 downto 0)) + unsigned(add2_lsb));
            add2_lsb    <= adder2(2 downto 0);
            add2_msb    <= adder2(4 downto 3);
            -- stage three adder
            adder3      <= std_logic_vector(unsigned(adder2(4 downto 3)) - unsigned(add2_msb));
        end if;
    end if;
end process sigma_delta_modulator;

duty_ref    <= add1_msb & adder3;
```

Fig 150 VHDL Code Implementing the Sigma Delta Modulator Circuit

The DCM is used to generate a reset signal which introduces an additional small delay after the counter-comparator indicates that the PWM can be deasserted. Using this approach effectively increases the resolution of the counter comparator, allowing for a lower frequency counter to be utilised. The DCM circuit uses a pair of mixed-mode clock management (MMCM) primitives within the FPGA fabric. In both of these, four main clocks are generated with a phase shift of ninety degrees between all of them. The first primitive generates these clock signals at a frequency of 32MHz with a duty cycle of 12.5%, as well as generating a 128MHz clock for the second primitive. This is then used to create four further signals at the same frequency which are ninety degrees out of phase with one another and have a duty cycle of 25%. These signals, as well as an additional 256MHz clock which is generated within the second primitive, are then fed into a bank of AND gates which creates the required reset pulse. The full circuit for the PWM clock management block is given in Fig 151, whilst the VHDL code for one of the MMCM blocks and the associated logic (AND gates and multiplexer) and is given in Fig 152. The waveforms given in Fig 153 further illustrate the operation of this part of the system.

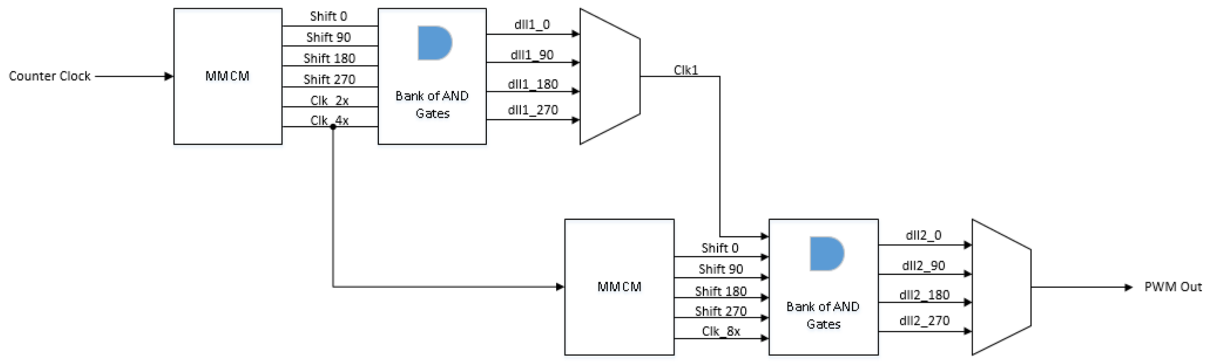


Fig 151 Digital Clock Manager Block Diagram

```
-- First DMC cell instatiation
i_dcm_14 : component dcm_14
    port map (
        clk_inl => clk,
        clk_0   => dl_clk0,
        clk_90  => dl_clk90,
        clk_180 => dl_clk180,
        clk_270 => dl_clk270,
        clk_2x  => dl_clk2,
        clk_4x  => dl_clk4
    );

-- Bank1 of AND gates to generate pulses
dll_l_0     <= dl_clk2 and dl_clk4 and dl_clk0;
dll_l_90    <= not dl_clk2 and dl_clk4 and dl_clk90;
dll_l_180   <= dl_clk2 and dl_clk4 and dl_clk180;
dll_l_270   <= not dl_clk2 and dl_clk4 and dl_clk270;

-- Multiplexer to select the input to the second DCM AND bank
with sclk_ref select dll_l_clk <=
    dll_l_0 when "00",
    dll_l_90 when "01",
    dll_l_180 when "10",
    dll_l_270 when others;
```

Fig 152 RTL Code Showing Instantiation of the DCM, Bank of AND Gates and Multiplexor Required to Generate the Reset Pulse

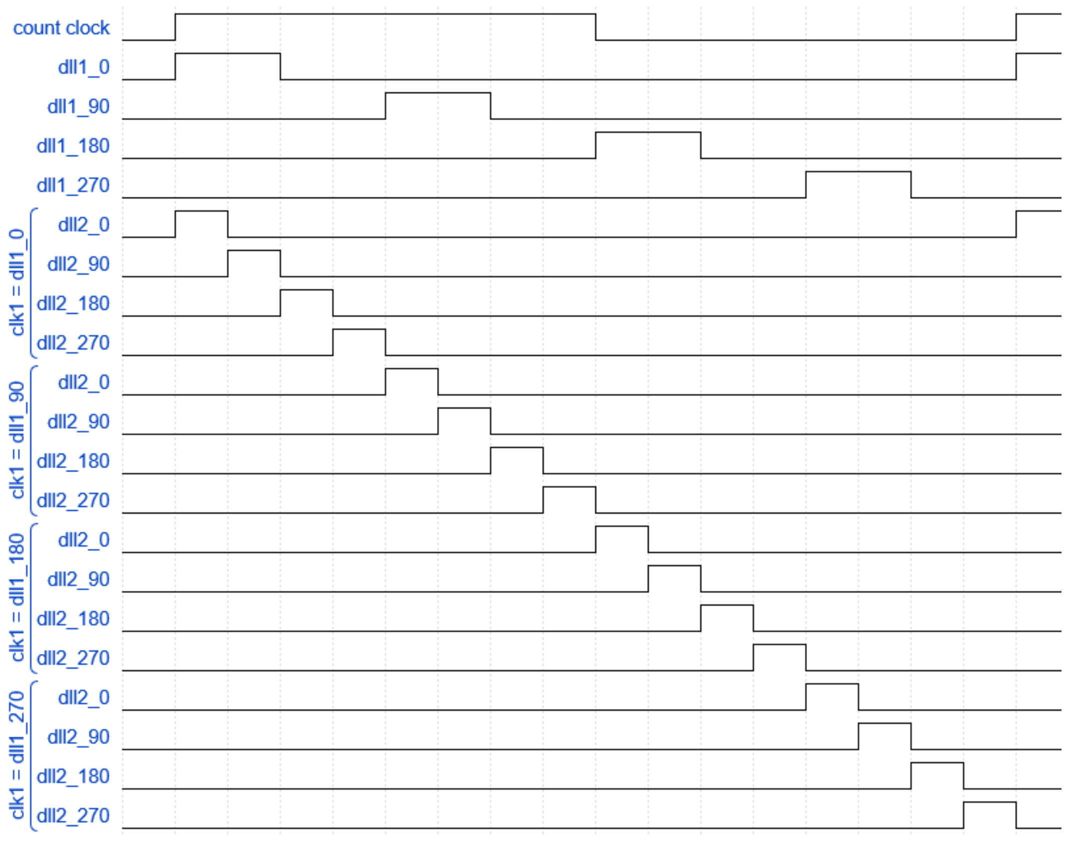


Fig 153 Digital Clock Manager Waveforms

The final block of the PWM generator circuit is the counter comparator circuit, which is used to drive the set pin of the output flip-flop. This circuit utilises a five-bit counter, clocked at

32MHz, and a digital comparator. The output flip-flop is only reset when the counter value is greater than the input control word, as determined by the digital comparator. The RTL code for this part of the PWM generator is shown in Fig 154.

```
counter_comparator:
process (dl_clk0) is
begin
  if rising_edge(dl_clk0) then
    if (resetn_sync = '0') then
      pwm_cnt   <= (others => '0');
      cnt_zero  <= '0';
      cnt_comp  <= '0';
    else
      -- Increment the counter and determine when it reaches zero
      pwm_cnt     <= pwm_cnt + 1;
      cnt_zero    <= and_reduce(pwm_cnt);
      -- Determine when the counter and the reference are about to become equal
      if (op_mode /= "00" and (pwm_cnt + 1) = unsigned(count_ref)) then
        cnt_comp  <= '1';
      else
        cnt_comp  <= '0';
      end if;
    end if;
  end if;
end process counter_comparator;
```

Fig 154 RTL Code for a Basic Five Bit Counter Comparator Circuit as Used to Drive the Power Converter PWM Output

7.2.2.1 Simulation and Testing

In order to verify the operation of the PWM generator block, a VHDL model is simulated using the Xilinix Vivado design suite. The PWM generator is tested under five different operating conditions to ensure that the input control word can generate the expected duty cycle. The control word is set to generate outputs of ten percent, twenty-five percent, fifty percent, eighty percent and ninety-five percent. Perhaps the most important consideration of these simulations is the way in which the sigma-delta modulator extends the effective resolution of the PWM generator. This is particularly prevalent in the simulation shown in Fig 155 which demonstrates the operation with a duty cycle of 80%. In this simulation, it can be seen how the signal duty_ref, which is the output of said modulator, oscillates between minutely different values. As intended, this has the effect of causing small oscillations in the duty cycle from one cycle to the next, with the average duty being tabulated in Table 17. These results demonstrate how the use of a sigma-delta modulator in the PWM generator block achieves a high degree of accuracy, exhibiting a maximum error of just 0.39% as shown in Fig 155.

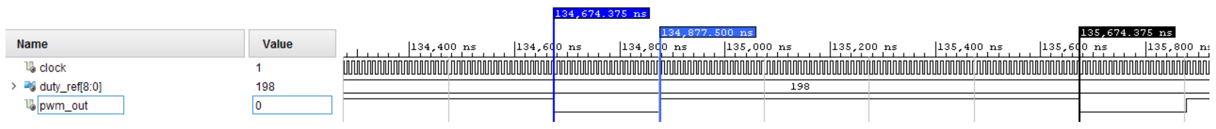


Fig 155 PWM Simulation with 80% Duty Cycle

PWM Control Word (hex)	Duty Cycle (%)	
	Expected	Measured
190	10	9.97
400	25	25
800	50	50
CC0	80	79.68
F30	95	95.11

Table 17 PWM Simulation Results

As discussed earlier in this chapter, an ILA core can be used to capture FPGA signals and perform hardware testing. Using this approach, the model has been further tested through the creation of an on-board stimulus generator which emulates the VHDL test bench. The outcomes of these tests are tabulated in Table 18 and illustrate how the hardware behaviour is analogous to the VHDL model simulations.

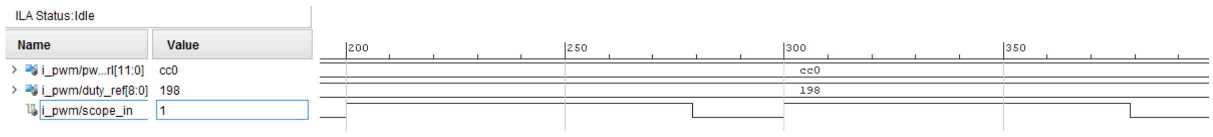


Fig 156 Hardware Test with 80% Duty Cycle

PWM Control Word (hex)	Duty Cycle (%)	
	Expected	Measured
190	10	10
400	25	25
800	50	50
CC0	80	80
F30	95	95

Table 18 PWM Hardware Test Results

7.2.3 PI Controller

The PI controller algorithm is relatively simple to realise in an FPGA, as it can be implemented using multipliers and adders, as shown in Fig 157. The output flip-flop and multiplexer are employed to limit the PI output to a predefined value. One important consideration is the prevention of integral windup which is caused by the value of the algorithms integral increasing to a large value whilst the output is clamped. This can result in both sub-optimal control and long response times. In order to prevent this from occurring, a technique known as conditional integration is utilised. When using this method, the integral is calculated on a cycle by cycle basis, but the value is disregarded if the output is above the output saturation point. This circuit is formed by the multiplexer shown in Fig 157, the output of which is registered and fed back into the DSP blocks. When the output has saturated, the flip-flop output is routed back to its input and the integrator value isn't updated. When the value of the output isn't saturated then the output of the adder is fed to the flip-flop and the integrator value is updated.

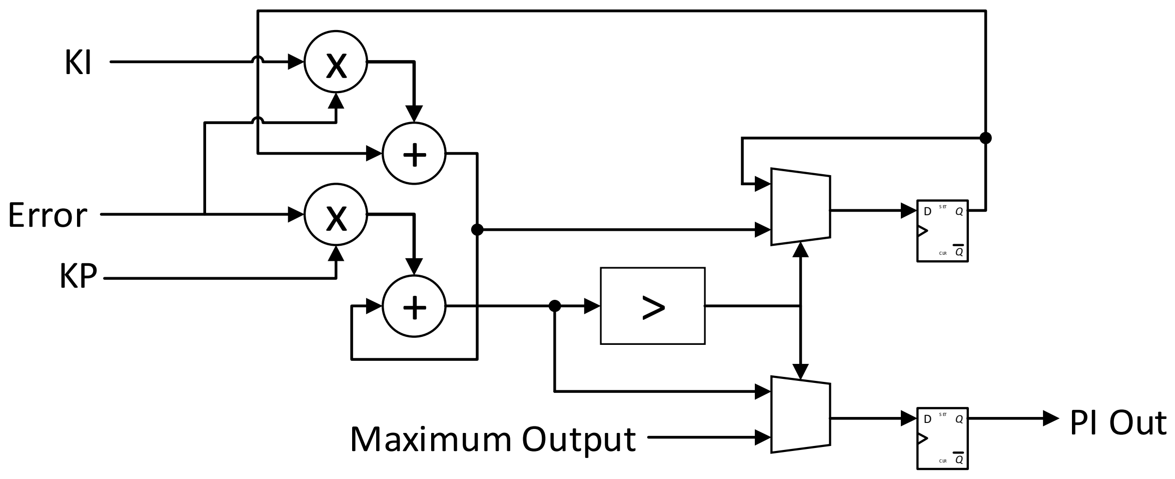


Fig 157 FPGA PI Controller Block Diagram

7.3 Inter-core Communication

The final consideration of the APSOC design is the methodology employed for communication between the CPU and FPGA cores. The standard method used for this, and recommended by Xilinx (2011), is the AXI4 bus. This is a bidirectional memory mapped interface which allows up to 256 data transfers in a single burst. This protocol is part of the ARM Advanced Microcontroller Bus Architecture (AMBA) (ARM (2010)), which is a series of buses introduced by ARM in the mid-1990's. A number of subsequent updates have been made to this standard, with the AXI protocol being introduced in the third version in 2003. There are three main variants of AXI4 interface which can be employed within the Zynq APSOC – AXI, AXI-Lite and AXI-Stream. The AXI-Lite is a stripped-down version which is suitable for devices with lower throughput requirements and transfers only one packet of data in a burst. The AXI-Stream is the most lightweight version of the interface, featuring uni-directional point to point data transfers. This version doesn't utilise addressing, featuring only a data bus and simple handshaking signals.

There are two mechanisms which are used for transferring data between the ARM CPU and the FPGA fabric, both of which use variations of the AXI buses. The first mechanism is concerned with the transfer of the currents and voltages read from the external ADCs. This data has a low throughput and is, therefore, transferred using a general purpose AXI bus. The second mechanism concerns the transfer of the data for the ANFIS membership values. This data is determined using look-up tables which are stored in the off-chip DDR RAM, owing to the fact that these LUTs occupy a large amount of RAM. In order to support this, a DMA IP core is utilised and controlled by software running on the CPU. An AXI stream type interface is used to interface between this core and the FPGA fabric. The communication between the FPGA and CPU cores is discussed further in the rest of this section.

7.3.1 AXI Control Registers

As one of the functions of the ARM CPU is to perform the system housekeeping, a number of memory mapped registers must be included within the FPGA fabric to allow the CPU to control its functionality. In addition to this, a number of registers are also required for the CPU to read the ADCs values as this data is contained in the FPGA fabric to improve throughput. The full ADC data set consists of three 32-bit words, two 12-bit words and one 16-bit word. This data is then used by the CPU to configure the ANFIS engines, which requires the writing of five 32-bit words back into the FPGA fabric. This data is updated at a slow rate of around 200 kHz to allow the CPU sufficient time to process these commands. These characteristics mean that a generic AXI-Lite interface is the most logical means of transferring data between the two chip segments.

The AXI-Lite bus consists of five separate channels – a write address channel, a write data channel, a read address channel, a read data channel and a write response channel. Each of these utilises a pair of handshaking signals – with valid being generated by the source of the transfer and ready by the sink. When this data is output, the valid signal is asserted with the ready signal being asserted by the recipient to indicate a successful transfer. In addition to the address and data transfer, there are also signals to indicate the response of the destination device. On the read channels, this is integrated into the data channel and transmitted at the same time as the data is read back. The write channels have a separate set of signals to handle the transfer of this response, replete with its own set of handshaking signals. In both instances, the response channel is set to a binary value of 00 when the transfer has been successfully completed. The full operation of the AXI-Lite interface is shown in the waveforms presented in Fig 158 and Fig 159.

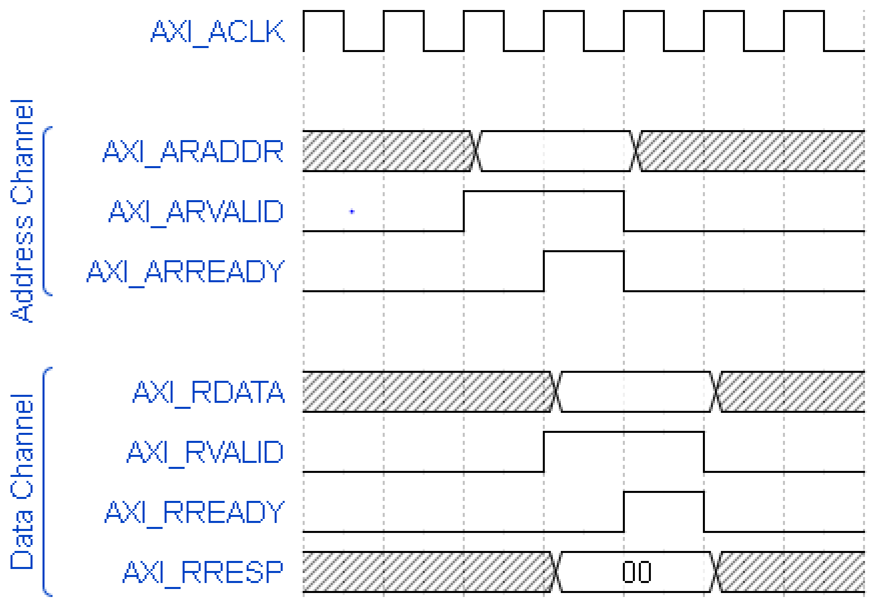


Fig 158 AXI-Lite Bus Behaviour During a Read

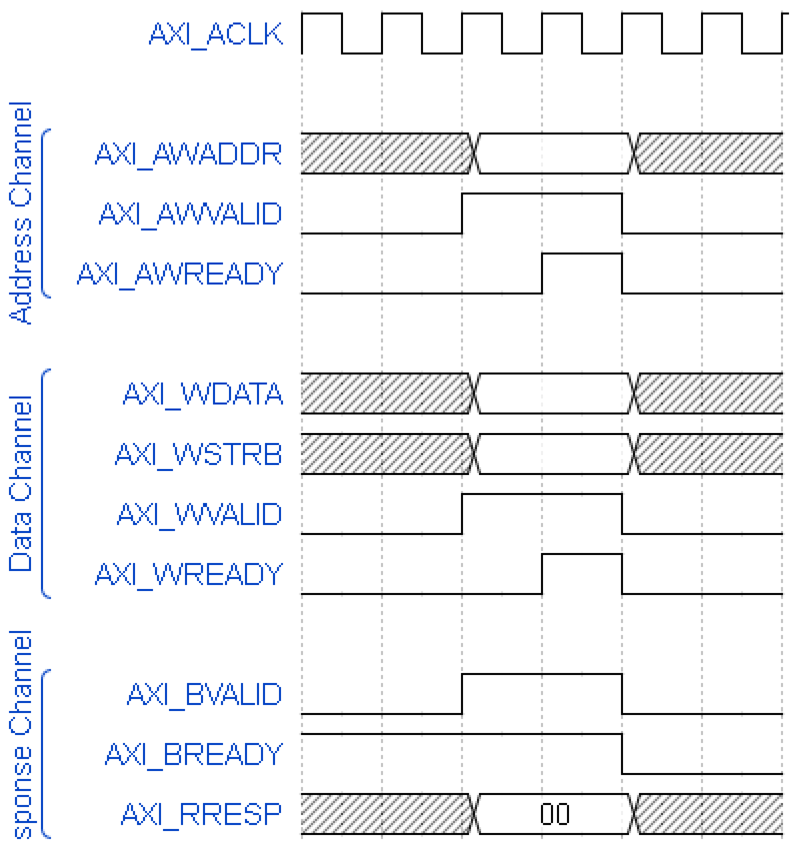
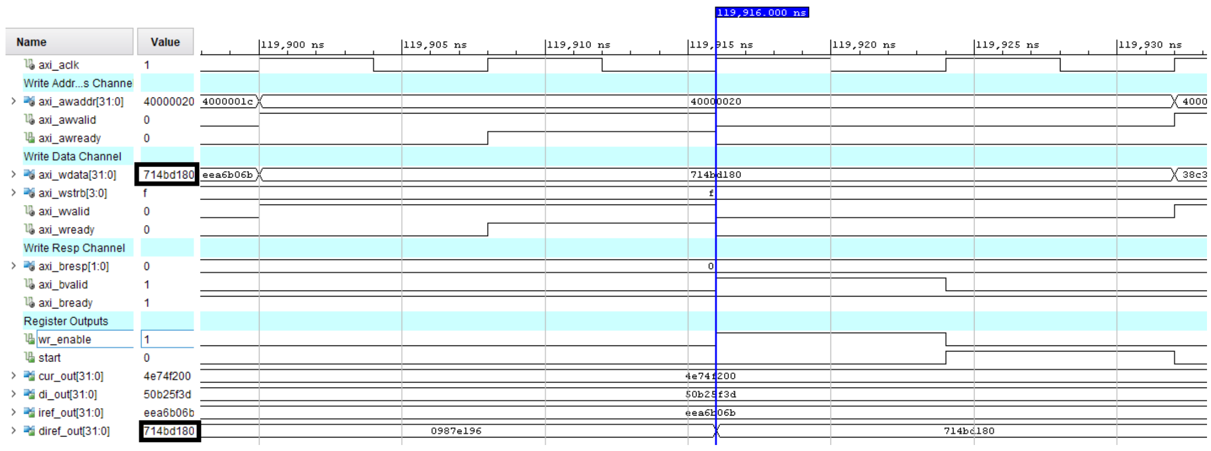


Fig 159 AXI-Lite Bus Behaviour During a Write

The AXI bus is a widely used standard which is the preferred method of intra-chip communication on Xilinx devices. As such, the design and implementation aren't novel concepts. However, this is an important part of the hardware without which the controller algorithms would not correctly function. In order to ensure that data is correctly transferred between the CPU and FPGA cores, VHDL models have been created which are used to simulate the operation of this interface. Additionally, on-chip testing is also carried out using the previously discussed Xilinx ILA IP core. The results of both of these tests are shown in Fig 160 and Fig 162 for a write transaction and Fig 161 and Fig 163 for the read transactions. In both instances, it can be observed how data is written into the register with address 4000020h which is then subsequently read back.



Name	Value
axi_aclk	1
Write Addr...s Channe	
axi_awaddr[31:0]	40000020
axi_awvalid	0
axi_awready	0
Write Data Channel	
axi_wdata[31:0]	714bd180
axi_wstrb[3:0]	f
axi_wvalid	0
axi_wready	0
Write Resp Channel	
axi_bresp[1:0]	0
axi_bvalid	1
axi_bready	1
Register Outputs	
wr_enable	1
start	0
cur_out[31:0]	4e74f200
di_out[31:0]	50b25f3d
iref_out[31:0]	eea6b06b
diref_out[31:0]	714bd180

Fig 160 AXI Register Write Simulation

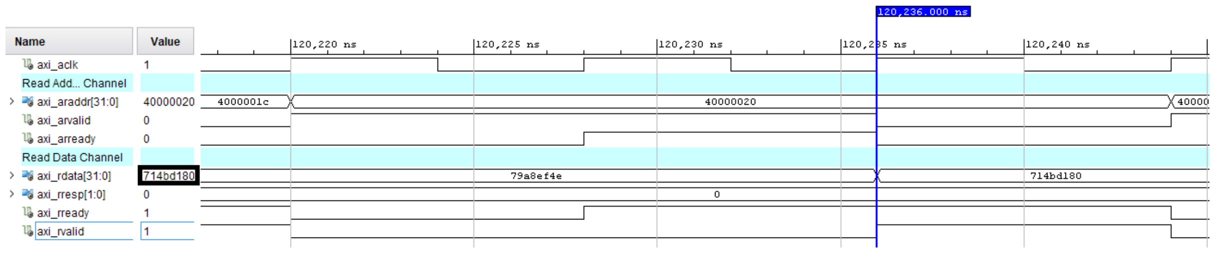
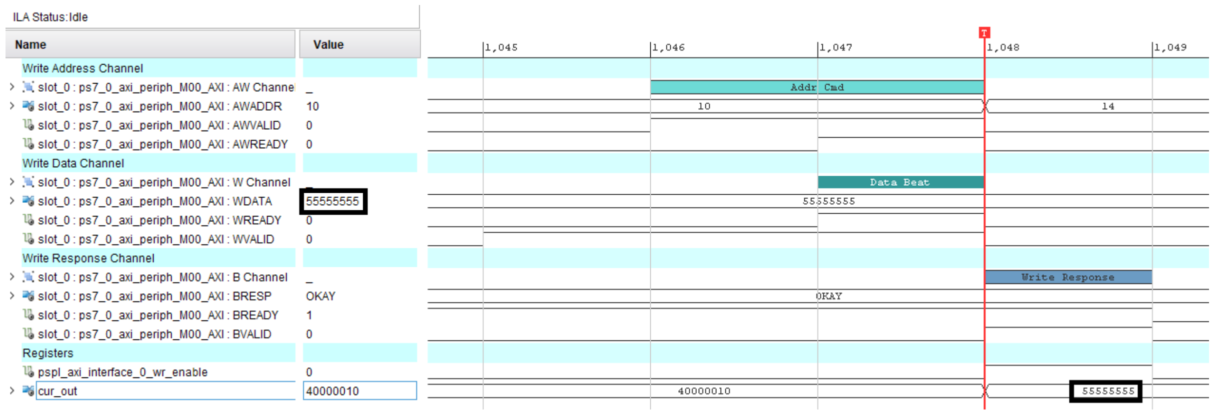


Fig 161 AXI Register Read Simulation



ILA Status:Idle
Name
Value
Write Address Channel
slot_0 : ps7_0_axi_periph_M00_AXI : AW Channel
slot_0 : ps7_0_axi_periph_M00_AXI : AWADDR
slot_0 : ps7_0_axi_periph_M00_AXI : AWVALID
slot_0 : ps7_0_axi_periph_M00_AXI : AWREADY
Write Data Channel
slot_0 : ps7_0_axi_periph_M00_AXI : W Channel
slot_0 : ps7_0_axi_periph_M00_AXI : WDATA
slot_0 : ps7_0_axi_periph_M00_AXI : WREADY
slot_0 : ps7_0_axi_periph_M00_AXI : WVALID
Write Response Channel
slot_0 : ps7_0_axi_periph_M00_AXI : B Channel
slot_0 : ps7_0_axi_periph_M00_AXI : BRESP
slot_0 : ps7_0_axi_periph_M00_AXI : BREADY
slot_0 : ps7_0_axi_periph_M00_AXI : BVALID
Registers
pspl_axi_interface_0_wr_enable
cur_out
Addr Cmd
Data Beat
Write Response
55555555
40000010
OKAY

Fig 162 AXI Register Write HW Test

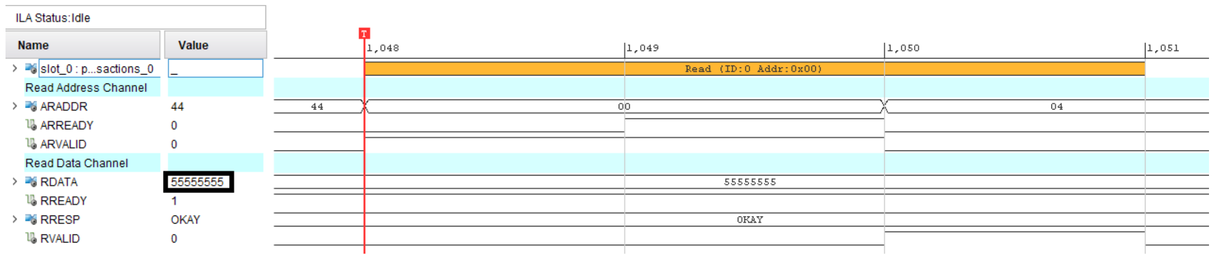


Fig 163 AXI Register Read HW Test

7.3.2 DMA Data Transfer

Both the inner and the outer control loop use a pair of ANFIS algorithms, one to calculate the value of KP and the other to calculate the value of KI. This means that there is a total of four ANFIS algorithms in the controller, each having two inputs with four input members. Given that the outer loop uses a 16-bit ADC and the inner loop ADC uses a 12-bit ADC, a total of approximately 2MB of RAM would be required to implement a LUT for each of these membership groups. However, as several inputs will give a zero degree of membership, these members can be removed from the LUT. An address decoder can then be utilised which returns a fixed value of zero when one of these inputs is selected. This approach reduces the total required RAM down to 1.5 MB. There are other techniques which could be utilised to further reduce the amount of memory required. One example would be to reduce the size of the LUT, so that it doesn't cover the entire input space, and use interpolation to calculate any intermediate values. However, achieving the most efficient usage of the RAM resources was not a major consideration as this implementation is instead focused on improving the latency of the ANFIS calculation within the FPGA fabric.

As has been discussed, the membership LUTs require approximately 1.5MB of memory. As the FPGA fabric features only 600KB of BRAM, it is necessary for them to be stored in the off-chip DDR RAM. The contents of the LUTs will be programmed by the software during initialisation of the APSOC. The data can then be transferred as required from the RAM into the FPGA fabric using a dedicated DMA core, which is a Xilinx IP core. The CPU core controls the operation of this core, deciding which DDR data to transfer and when. As the membership LUTs are organised in non-sequential addresses within the DDR RAM, the so-called scatter gather engine must be utilised. This engine means that a string of descriptors is additionally required and stored in a separate memory block. These contain a string of pointers to the area of DDR memory which must next be read for a given packet. Each descriptor contains a base address to be read from the DDR and an indicator of the number of bytes to read. A typical interconnection for a scatter-gather based DMA system is given in Fig 164. In this diagram, it can be seen how there is a separate block RAM (BRAM) which is additionally added to the design. It is this piece of memory which houses the descriptors for the DMA engine.

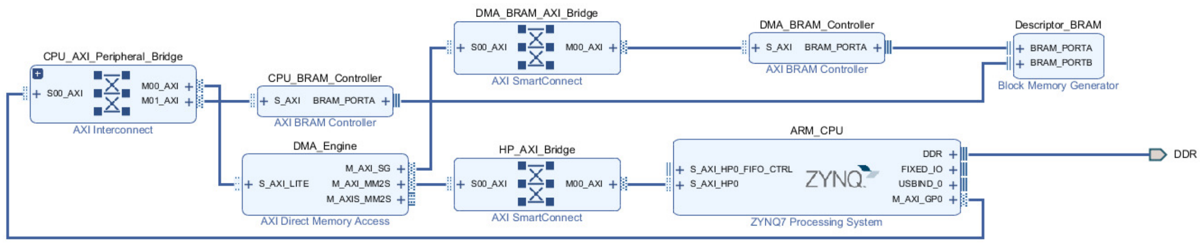


Fig 164 Scatter-Gather DMA System

There are three buses utilised within the architecture shown in Fig 164, all of which use an AXI-Lite interface. The first of these buses is a master from the CPU to both the DMA and BRAM. This is used to write the descriptors into the BRAM and to control the DMA transactions. When the DMA starts to operate, it must read the descriptors from the BRAM module in order to start configuring the memory transfers. This data is handled by the second bus which is also an AXI-Lite master-type bus. As well as reading the memory, the DMA also stores status updates into the BRAM as the descriptors are processed. The final interface is between the DDR and DMA, which also takes the form of an AXI-Lite bus. Within the Zynq device, this interface features fast FIFOs to allow buffering from the DDR RAM. This garners a higher throughput than the general purpose ports used in the rest of the system.

The final consideration of the DMA core is the methodology by which the packets are transferred from the engine itself to the rest of the logic on the FPGA fabric. An AXI-Stream interface, featuring non-addressed unidirectional data transfers, is utilised for this purpose. This requires three handshaking signals – valid, ready and last. This simple interface allows for a packet to be transferred in a single clock cycle, resulting in a high bandwidth. The waveforms given in Fig 165 demonstrate how the data is transferred using this interface.

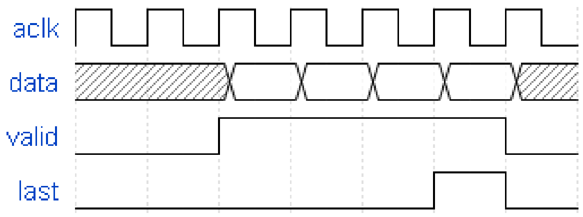


Fig 165 AXI-Stream Bus Behaviour

The DMA core performs a crucial function within the APSOC, meaning it is important to validate its performance through simulation and hardware testing. This core communicates with a custom logic block which is in turn used to write the data into the relevant ANFIS engine. This logic is comprised of a small FSM which routes data to the correct ANFIS as it is received. An output signal called ”wr_enable” is asserted for one clock period to latch the data into the ANFIS registers. This block also generates a signal called “start_anfis” which is used to start an ANFIS calculation once all data has been received from the CPU. A handshaking signal, called “anfis_rdy”, is received from the ANFIS logic which acknowledges that it has started performing a calculation. The operation for this can be seen in Fig 166, which shows a simulation of a single packet transfer, and Fig 167, which shows the same thing as captured on the APSOC. Note that in these waveforms only the first and last set of register writes are shown for brevity.

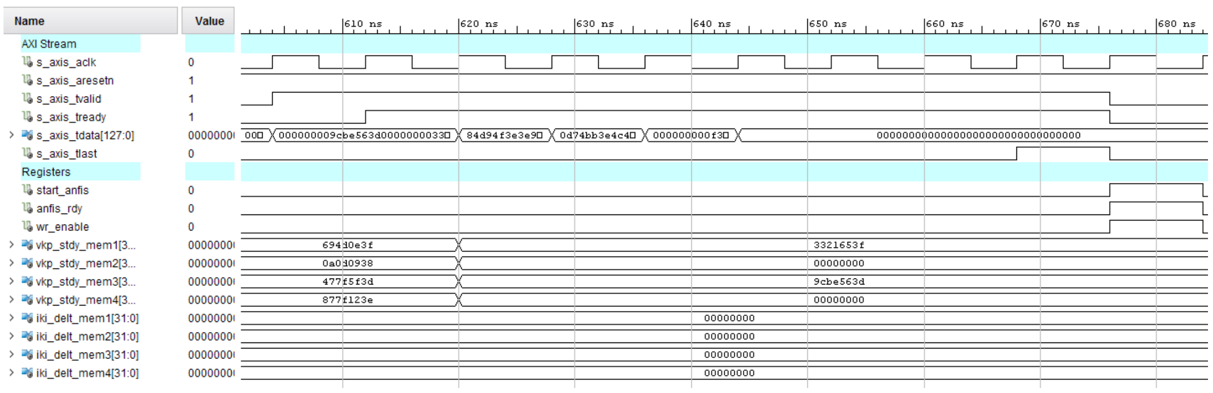


Fig 166 DMA Data Transfer Simulation

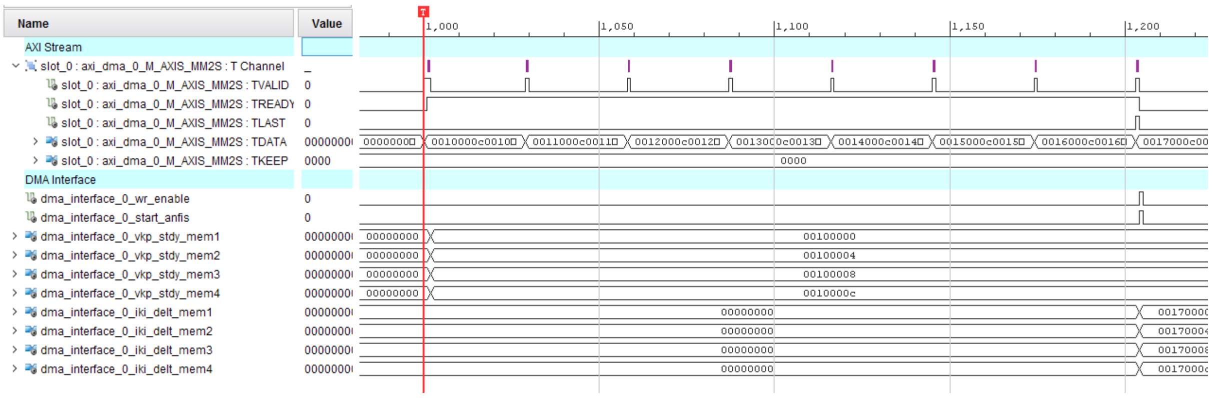

Name
Value
1,000
1,050
1,100
1,150
1,200
AXI Stream
slot_0 : axi_dma_0_M_AXIS_MM2S : T Channel
slot_0 : axi_dma_0_M_AXIS_MM2S : TVALID
slot_0 : axi_dma_0_M_AXIS_MM2S : TREADY
slot_0 : axi_dma_0_M_AXIS_MM2S : TLAST
slot_0 : axi_dma_0_M_AXIS_MM2S : TDATA
slot_0 : axi_dma_0_M_AXIS_MM2S : TKEEP
DMA Interface
dma_interface_0_wr_enable
dma_interface_0_start_anfis
dma_interface_0_vkp_stdy_mem1
dma_interface_0_vkp_stdy_mem2
dma_interface_0_vkp_stdy_mem3
dma_interface_0_vkp_stdy_mem4
dma_interface_0_iki_delt_mem1
dma_interface_0_iki_delt_mem2
dma_interface_0_iki_delt_mem3
dma_interface_0_iki_delt_mem4

Fig 167 DMA Data Transfer HW Test

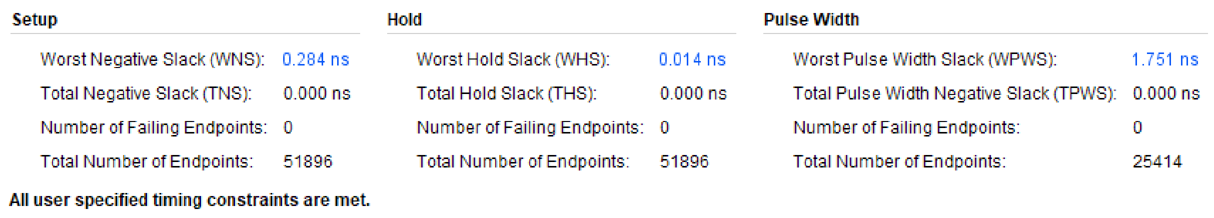
7.4 Timing and Utilisation Analysis

The FPGA fabric is synthesized and built using the Xilinx Vivado design suite. Once the RTL has been transformed into its equivalent array of gates, the aforementioned tool is then used to perform further analysis of the design. As has previously been mentioned, the FPGA fabric is primarily composed of a number of logical slices consisting of LUTs, registers and muxes. In addition to this, the FPGA fabric also features DSP cores, 4.9Mb of RAM and a number of MMCM blocks for clock generation. The total logic utilisation for each of these different types of cells is shown in Fig 168, which indicates that the design utilises less than half of the total resources in the selected device. This means that the design could be equally implemented in a smaller Zynq device, such as a Z-7015 device. It would also require minimal effort to port the design to other suitable devices, such as the Cyclone-V or Arria-V chips from Altera. This shows how the design, in particular the novel ANFIS implementation, is not technology dependent.

	Slice LUTs	**Slice Registers**	**F7 Muxes**	**F8 Muxes**	**DSP**	**RAM**	**MMCM**	**XADC**
Used	24485	24081	69	8	100	11.5	2	1
Available	53200	106400	26600	13300	220	140	4	1
Utilisation (%)	45.49	22.59	0.26	0.06	39.09	8.21	50.00	100

Fig 168 Logic Utilisation for the Novel Power Converter Controller FPGA Core

In order to optimise the operation of the different functions within the FPGA fabric, there are two main clock domains. The majority of the logic in the device is clocked using the main on-board oscillator, which runs at 100MHz. The AXI busses feature a higher clock rate of 150MHz, which is designed to improve the throughput. This clock is sourced by one of the devices PLL cores, with the exact frequency having been determined on the basis that this is the fastest clock speed which can meet with the timing constraints of the device. In addition to these clock domains, the PWM generator utilises a pair of MMCM primitives which generate a number of clocks for use in this block. The exact functionality of these clocks is discussed in section 7.2.2, with the majority being used only to generate the reset signal for the PWM output. However, the sigma-delta modulator is clocked at 1MHz whilst the counter comparator circuit runs at 32MHz. In order to avoid metastability issues when crossing from one clock domain to the other, handshaking signals are generated and passed between domains using double flip-flop circuits as recommended by Li et al (2010). Using this information, a set of timing constraints have been developed to identify false paths and an STA has been performed on the device. In Fig 169 a summary of the STA is presented, giving a worst-case slack of 284 ps for flip-flop setup time and a corresponding hold time slack of 14 ps. The slack time for a pulse is also analysed and given as 1.751 ns, showing how the FPGA design is capable of meeting with the devices timing constraints. As is discussed in section 7.2.1.6, the longest timing delays in the FPGA are caused by the IP blocks used for the floating-point DSP operations. As additional pipelining stages can be added to these blocks, it is possible for the FPGA design to be targeted at higher clock speeds.



Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.284 ns	Worst Hold Slack (WHS):	0.014 ns	Worst Pulse Width Slack (WPWS):	1.751 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	51896	Total Number of Endpoints:	51896	Total Number of Endpoints:	25414

All user specified timing constraints are met.

Fig 169 Summary of the STA for the FPGA Design

7.5 Conclusions

In this chapter, the implementation of the novel power converter controller has been considered. This is primarily realised within the FPGA fabric of a Zynq APSOC device, with the CPU core being used for basic housekeeping tasks. The full control loop is discussed, including the implementation of the PWM block using a state of the art technique. Additionally, the interaction between the FPGA and the CPU cores is modelled and tested on the real device. However, the main accomplishment presented in this chapter is the creation of a novel FPGA implementation technique for the ANFIS algorithm. It is shown how this methodology can be utilised to remove redundant clock cycles within the existing architectures, resulting in reduced calculation latency. The analysis of the utilisation reports shows that this comes at the expense of only a very modest increase in the amount of logic required within the FPGA. The STA results for the two methods also show that the novel methodology has no temporal impact on the implementation. Although the design is targeted at a 100MHz clock rate, it is also shown through analysis how it would be straightforward to target higher clock rates. In the next chapter, this FPGA realisation is validated through the use of PSim-QuestaSim co-simulation and hardware testing.

8 Power Converter Circuit Design and Experimental Testing

In the previous chapter, the APSOC implementation of the novel control solution for the power converters was presented and the critical elements of the device were simulated. The next stage in the project is the validation of this controller within a realistic operational context. The block diagram given in Fig 170 shows the test setup, including hardware interfaces, which is used to validate the performance of the controller. The design of the interface circuitry is briefly discussed in the rest of this chapter before test results are presented. In the first instance, this involves simulation of the RTL models presented in the previous chapter alongside the power electronics. This is achieved using co-simulation between the QuestaSim modelling tool and PSim. Finally, some testing of the APSOC control solution is presented which considers the operation of the critical blocks in the controller.

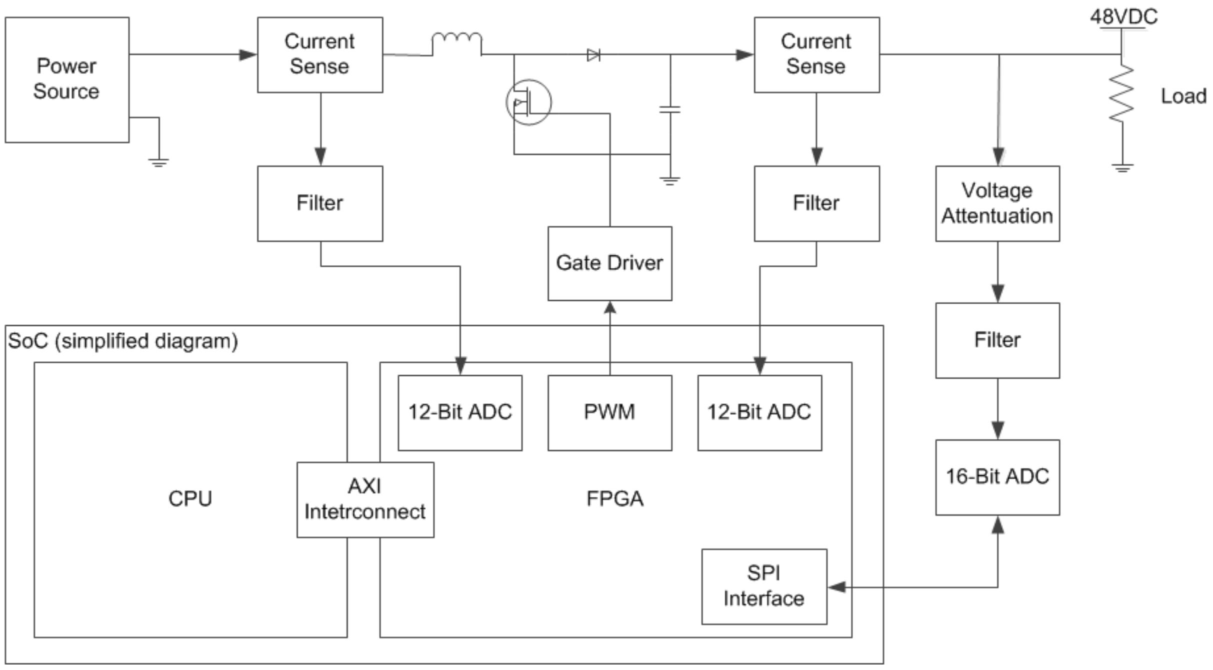


Fig 170 Test Setup for the Novel Power Converter Control Solution

8.1 Interface Circuit Design

In order to properly regulate the power converter output, the APSOC requires three ADC interfaces – one each for the load current, the inductor current and the output voltage. The Zynq chip features a pair of 12-bit ADCs with a sample rate of approximately 1MHz which can be utilised for the current measurements, giving a resolution of less than 1mA in both instances. As the onboard ADCs have a maximum input voltage of 1V, they are not suitable for measuring the output voltage of the converter. Therefore, an additional 16-bit ADC with a maximum voltage of 5V is instead utilised for this purpose. Using a simple voltage divider circuit, the output is divided by ten and fed into the ADC, giving a resolution of approximately 1mV for the output voltage. This ADC is also sampled at a rate of 1MHz and interfaces with the APSOC using a simple SPI bus.

In order to monitor the currents in the power converter, it is necessary to convert the current into a voltage. In principle, this can be achieved by measuring the voltage drop across a series resistor. An op-amp based multiplier can then be used to give the required voltage output. However, this is somewhat difficult to achieve in reality owing to the need for a high-performance amplifier and a high degree of noise immunity. Therefore, it is best to utilise a dedicated IC for this purpose, such as the AD8212 from Analog Devices (2017). An example of a typical circuit is given in Fig 171, illustrating how the only external components required are a gain setting resistor (R2) and a sense resistor (R1).

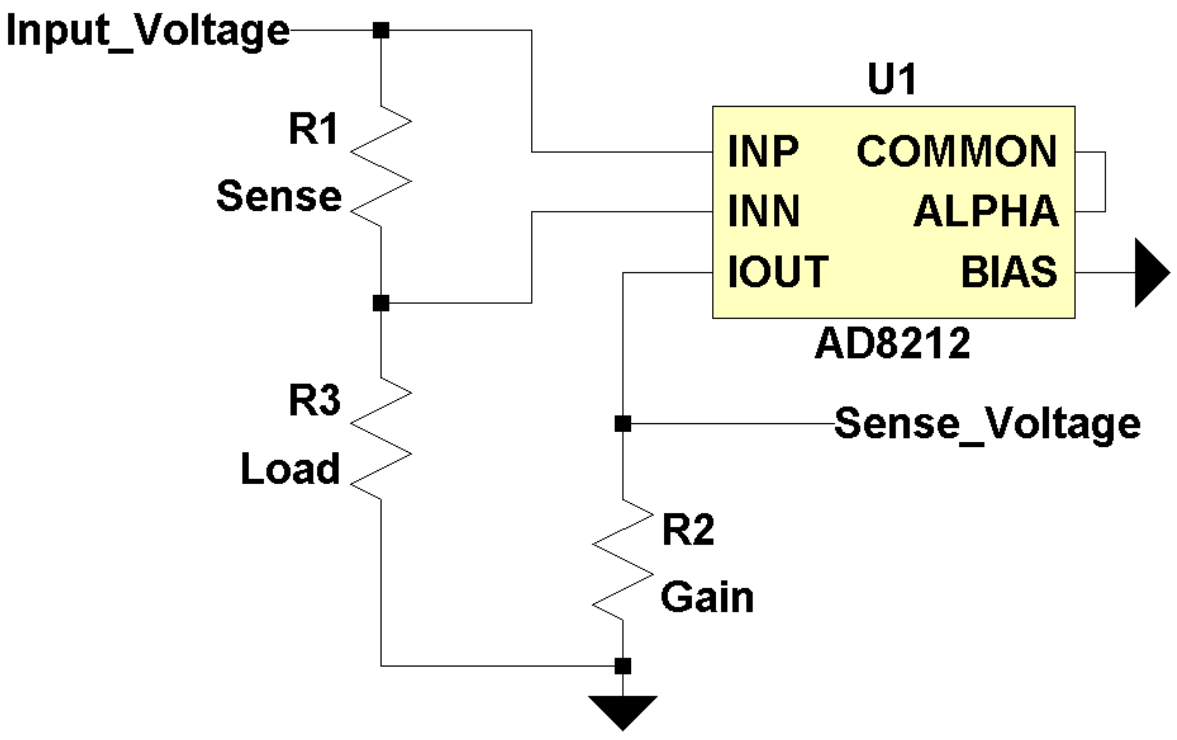


Fig 171 High Side Current Sense Circuit using the AD8212 IC

The circuit given in Fig 171 is simulated using the LTSpice software to ascertain its validity. To measure the load current, a sense resistor of 20 mΩ can be utilised and the gain of the amplifier set to 10, giving a maximum output of 1V corresponding to 5A. A simulation showing this is given in Fig 172, where it can be seen how the output voltage increases in proportion to the load current. It is simple to modify this circuit to measure the inductor current by reducing the gain of the amplifier. If this is reduced to five, the resultant circuit has an output of 1V corresponding to 10A.

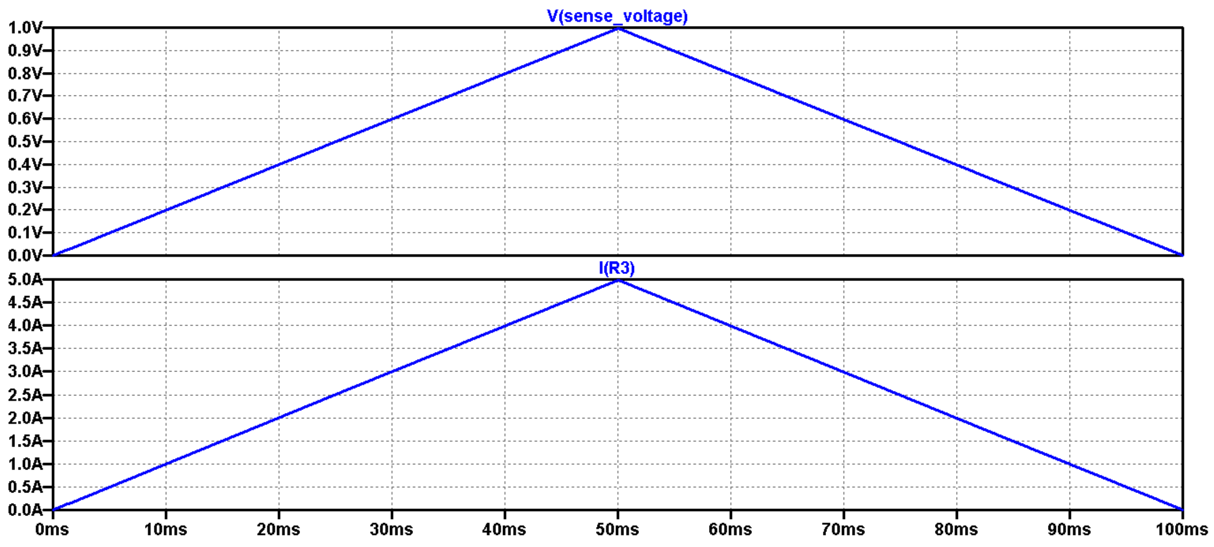


Fig 172 Simulation of the Load Current Sense Circuit Showing Current (blue) and Sensor Circuit Output Voltage (Red)

In order to prevent aliasing from occurring as part of the ADC sampling, it is important to include a low pass filter stage. In order to satisfy the Nyquist theorem, the cut-off of these aliasing filters must be twice the frequency of the ADC sampling rate. Therefore, a second-order active filter with a cut off of 2MHz should be included as part of the interface circuit. Using the Sallen-Key topology, the circuit for this filter is given in Fig 173. The resistor values in this circuit are deliberately kept below 1k in order to reduce the thermal noise which is introduced into the circuit. As this noise is proportional to resistance, the accuracy of the measurement system is improved by keeping resistive values as low as possible. The frequency response of the Anti-Aliasing filter is shown in Fig 174. As expected with a second order filter, the roll-off rate is measured at 12dB per octave. The attenuation of the circuit is also approximately 6dB at 2MHz, again in keeping with the desired performance of a second order filter of this kind.

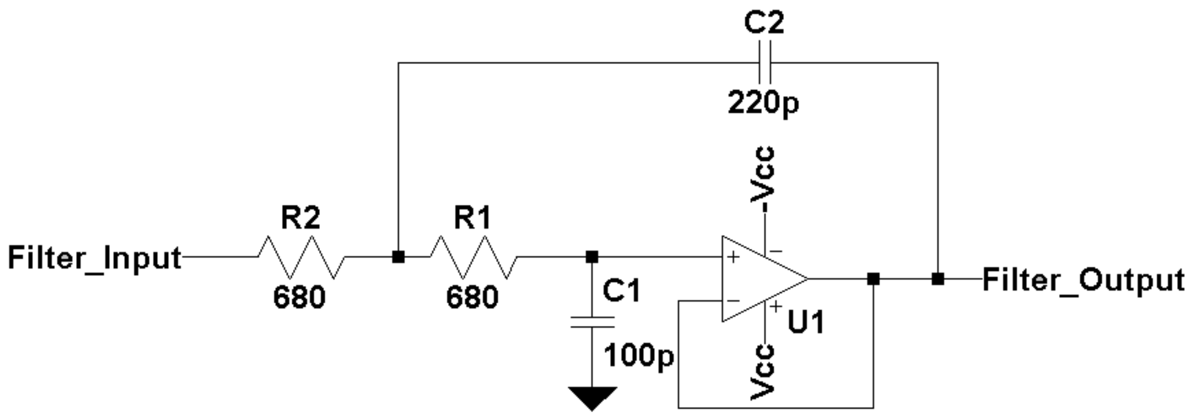


Fig 173 2MHz Low Pass Sallen Key Anti-Aliasing Filter Circuit

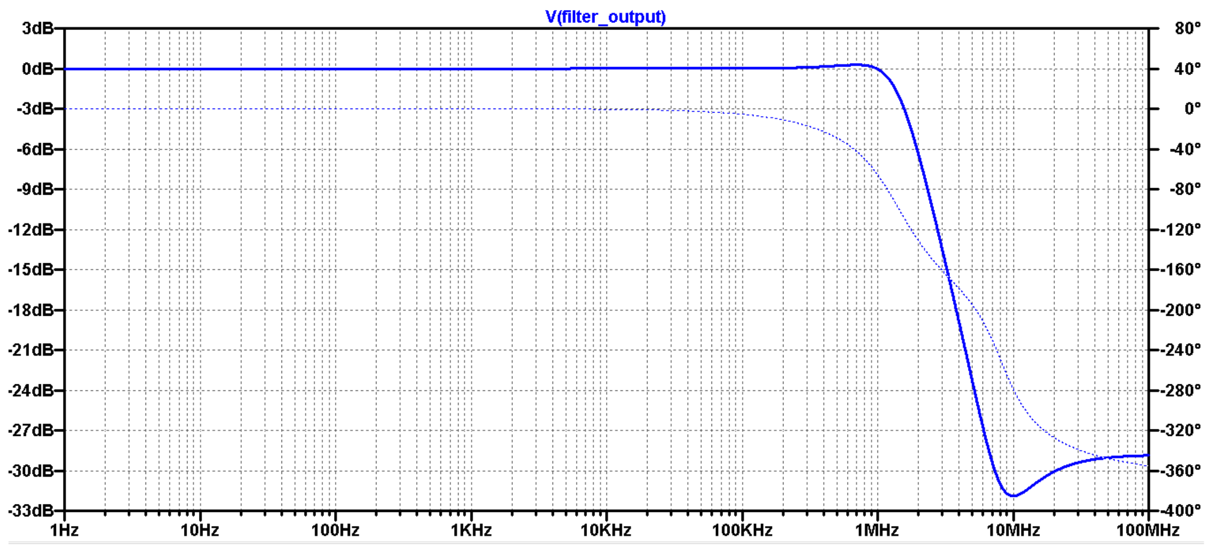


Fig 174 Frequency Response of the Anti-Aliasing Filter

The final consideration of the interfacing circuits is the switching of the MOSFET in the power converter. Whilst it is possible to find devices which can be directly driven from the APSOC output, high currents are usually required to ensure fast switching within power converters. This makes it impractical to drive these switches directly from an FPGA fabric. This is especially true with high frequency converters which require a large inrush current to force the MOSFET into conduction. Therefore, an interfacing circuit is required for this purpose, primarily to protect the APSOC power supplies but also to reduce switching delays. Although this can be achieved using discrete components, for example through the utilisation of a push-pull driver, the availability of inexpensive driver ICs simplify this task. An example of this is shown in Fig 175, where an LTC1693 driver IC is used to drive a MOSFET, in this case an IRF1010 device. This circuit is simulated using the LTSpice simulation software, with the results given in Fig 176 demonstrating how this circuit can switch the MOSFET at a rate of 1MHz.

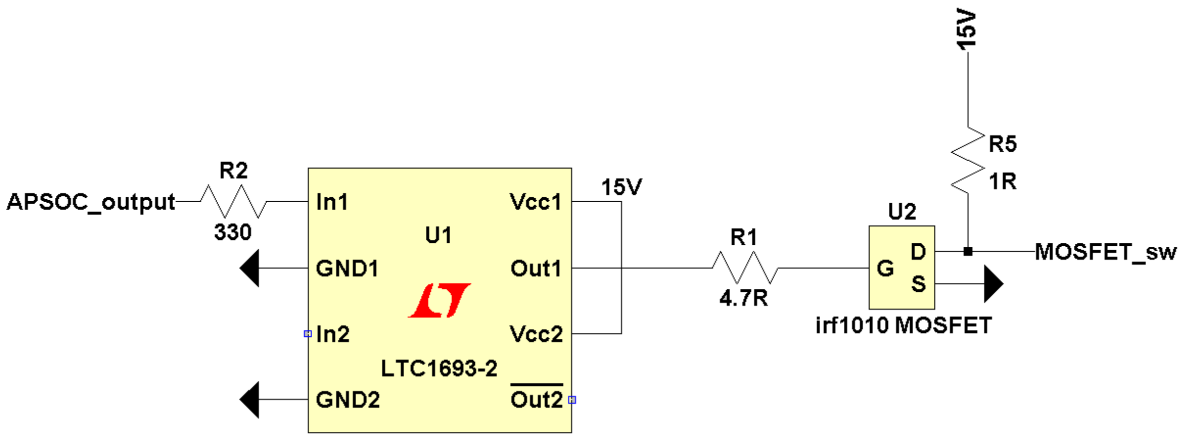


Fig 175 MOSFET Driver Example Circuit

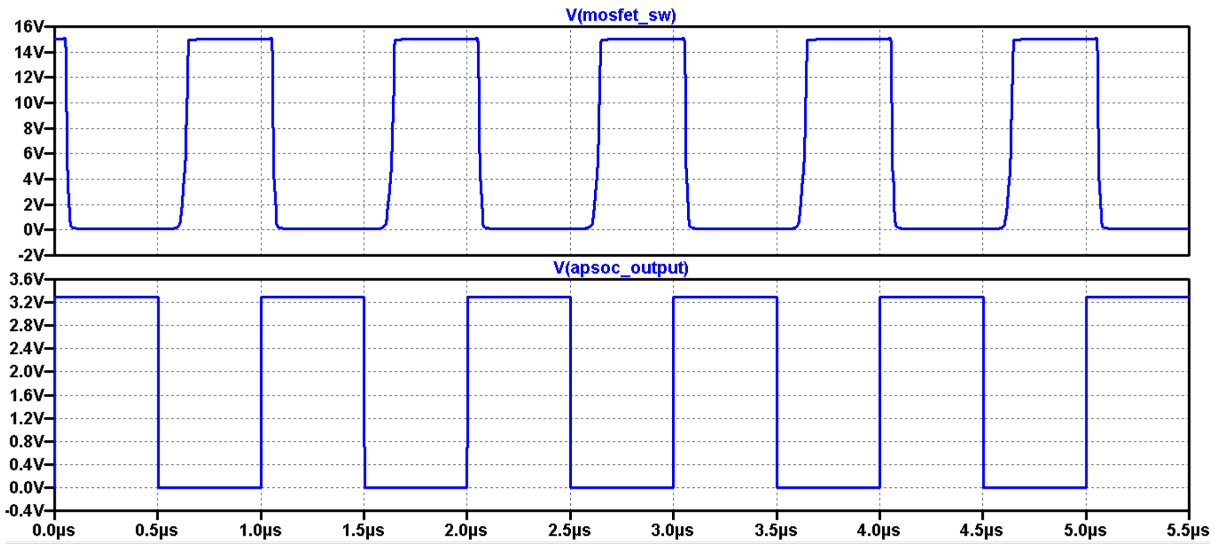


Fig 176 MOSFET Driver Circuit Simulation

8.2 Validation through Simulation

As has previously been discussed, PSim allows for co-simulation with the QuestaSim modelling tool. This means that the VHDL based model of the APSOC, as described in the previous chapter, can be simulated alongside the power electronics. Using this methodology, it is possible to examine the effect of the digital implementation on the control loop in closer detail. A boost converter model has been developed within PSim in order to allow for this validation. This model is based on the analogue power converter which was previously used for the solar PV cell in the analogue simulations given in chapter 5. This means that the ANFIS parameters, which are intended to be stored within the DDR RAM and are subsequently easy to modify, are programmed to give the same performance as the previously developed MATLAB model. However, the underlying APSOC architecture is easily adaptable for different applications, as discussed in the previous chapter. The model is simulated under a number of different operating conditions, with the steady-state and dynamic characteristics being examined. These simulations allow for the overall performance to be compared with that seen in the original simulations given in chapter 5.

In the PSIM – QuestaSim model, the CPU functions are run within the C compiler whilst the behaviour of the intercore AXI bus and ADCs are modelled using VHDL. The FPGA generates an interrupt whenever a new set of measurements is ready for processing by the CPU. When this occurs, a full set of data is read from the FPGA fabric using the AXI interface. These measurements, which consist of a full set of ADC values and the inductor reference current, as calculated in the outer PI control loop, are then used to determine how the ANFIS engine is configured. Once this has been calculated, the CPU then passes this data back into the FPGA fabric using both the AXI bus and the DMA engine.

In Fig 177, the basic operation of the AXI bus model is shown during the initial read sequence, as taken from the QuestaSim software. In this simulation, it can be seen how the inductor reference current is transferred into the CPU core, using the s_axi_rdata signal, when the s_axi_araddr signal is set to 10h. The operation of the four handshaking signals – s_axi_arvalid, s_axi_arready, s_axi_rvalid and s_axi_rready - is also demonstrated within this simulation, with the transfer only being effective when both the valid and ready signals are asserted for a given channel. Although only one transfer is shown here, for brevity, the operation is observed to be identical for each of the ADC registers within the FPGA fabric.

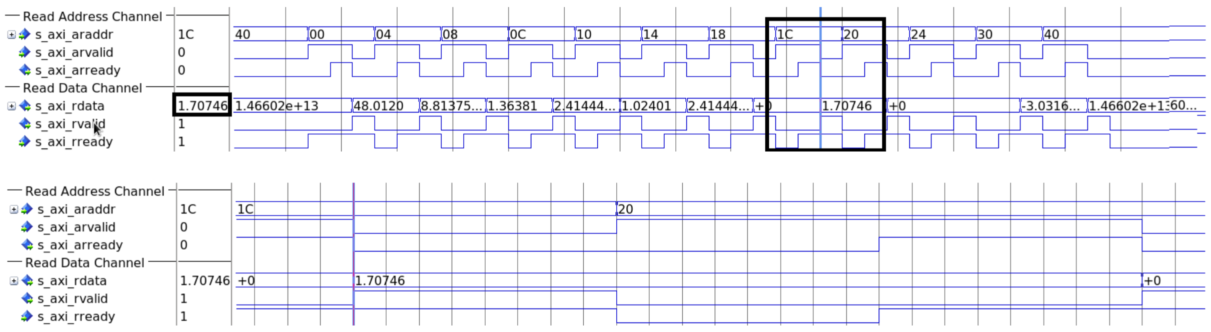


Fig 177 Inductor Reference Current being Read from the FPGA Core using the AXI-Lite Bus

Once the CPU has read the measurement data and performed its calculations, it then passes data back into the FPGA fabric to configure the ANFIS hardware. The waveforms for the write transactions on the AXI bus are shown in Fig 178, as taken from the QuestaSim software. There are two important aspects which must be considered in this waveform. Firstly, the behaviour of the AXI bus write channels is demonstrated to operate exactly as required. In Fig 178 the address is set using the s_axi_awaddr signal, with handshaking handled by s_axi_awvalid and s_axi_awready. Similarly, the data is set using the s_axi_wdata signal, whilst handshaking is performed using the s_axi_wvalid and s_axi_wready signals. When the data or address is set, the associated valid signal is asserted and is then acknowledged by the FPGA core using the appropriate ready signal.

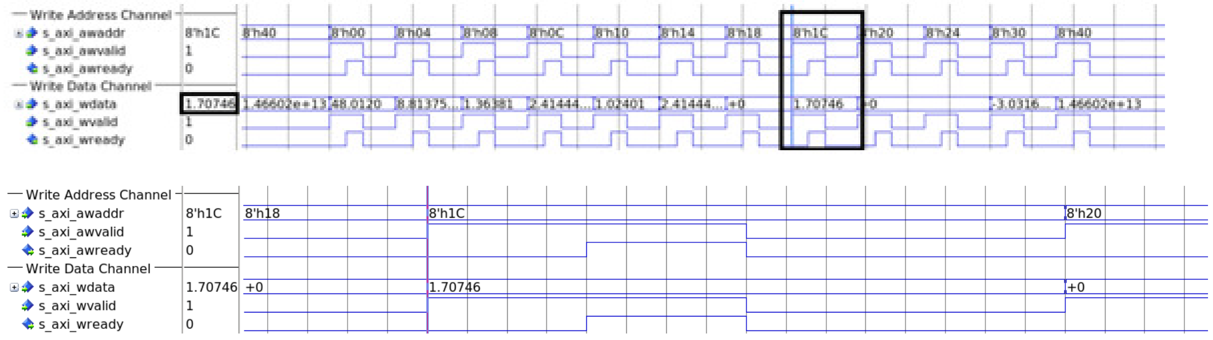


Fig 178 CPU Writing Configuration Data into the FPGA Core using the AXI-Lite Bus

The other important consideration of the inter-core communication bus is the way in which the data received over the AXI bus is interpreted by the FPGA fabric. This operation is illustrated in Fig 179, where it can be seen that the wr_enable signal is pulsed after each write transaction. This serves as a register enable signal within the ANFIS block, with the internal control registers being updated when this signal is asserted. It can also be seen how one of the register values, such as the iref_out signal in Fig 179, is updated after each of the AXI write transactions, showing how the data values are routed into the ANFIS block.

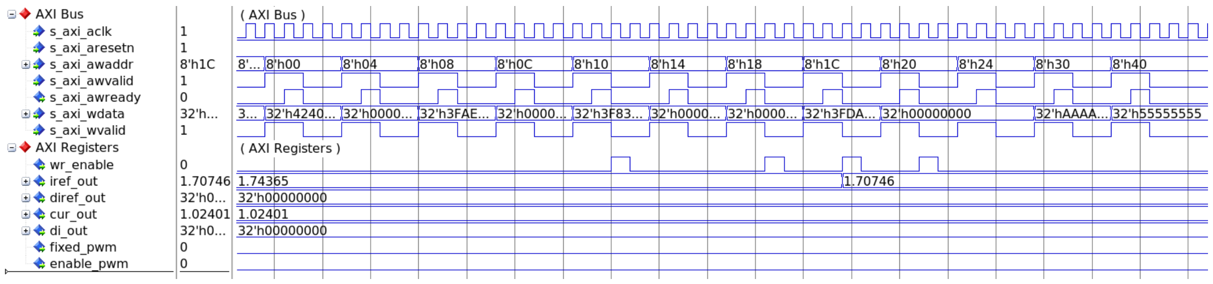


Fig 179 Update of Registers Following Write of Data using AXI-Lite Bus

As is discussed in the previous chapter, the ANFIS hardware uses RAM based LUTs to determine the values of its input members. Due to the amount of memory required to implement these tables, the off-chip DDR RAM is utilised for this purpose. A DMA engine is used to transfer the data from this memory space into the FPGA fabric in order to configure the ANFIS hardware. In the PSim-QuestaSim model, the DMA engine is modelled using VHDL and is controlled using a function written in C. This is analogous to the hardware design where the DMA is a hardware block within the FPGA fabric which is controlled by the CPU core.

There are a total of four ANFIS engines, one each for the proportional and integral gain in both of the PI controllers. These ANFIS engines each have two inputs with four members. This means that the DMA is required to look up a total of 32 different members and transfer this data to the FPGA core. This data is transferred as eight groups of 128 bits, with each group representing four single precision floating point numbers. As is discussed in the previous chapter, the DMA passes data to the FPGA fabric using an AXI-Stream bus and this transfer is shown in Fig 180. In this simulation, data is transferred across when the two handshaking signals, s_axis_tvalid and s_axis_tready, are both asserted. It can be seen in Fig 180 how the two handshaking signals indicate an active data transfer for a total of eight groups of data. When the FPGA fabric receives this data, it decomposes the 128-bit data, given in s_axis_tdata, into four separate floating-point numbers, as can be seen in Fig 180 by the vkp_stdy_mem1 to vkp_stdy_mem4 signals. Finally, the wr_enable signal is pulsed high for one clock period when s_axis_tlast is asserted in order to signal to the ANFIS engines that new data is available.

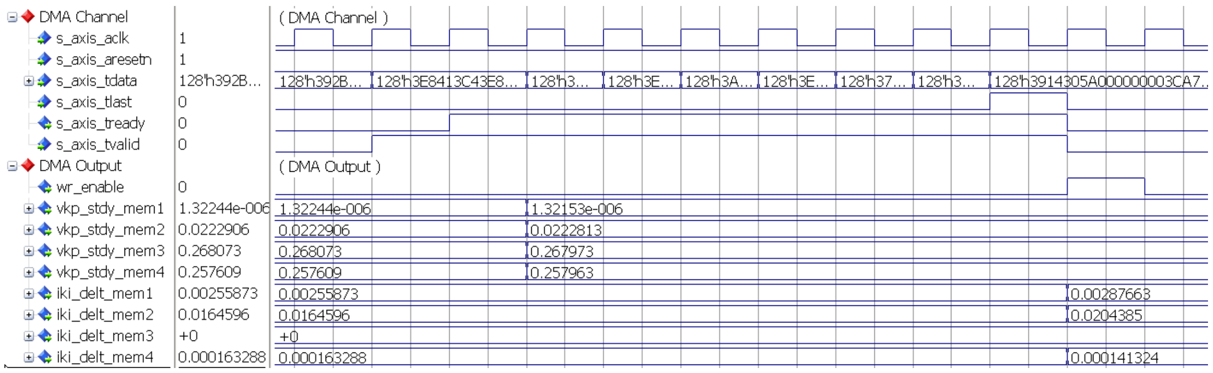


Fig 180 ANFIS Membership LUT Table Being Written into the FPGA Core from the DMA IP Core using the AXI-Stream Bus

The final thing which can be demonstrated through the examination of the AXI based communication buses is the way in which step changes are handled. In the previously developed model of the controller in Matlab, any fluctuations in load current are latched for 50 μs. This value was chosen experimentally as part of the previous modelling because it gives the best response to step changes. Therefore, it is important for the CPU core to replicate this feature when it configures the ANFIS engines. The waveforms in Fig 181 and Fig 182 are taken from the QuestaSim software when there is a 1A change in current. In both instances, it can be seen how the pv_di signal is held at approximately one (Fig 181) and minus one (Fig 182) for a total duration of 50 μs. This signal is the delta value for the load current, as written into the ANFIS blocks, and shows how the step changes are handled using the same methodology as the Matlab model.

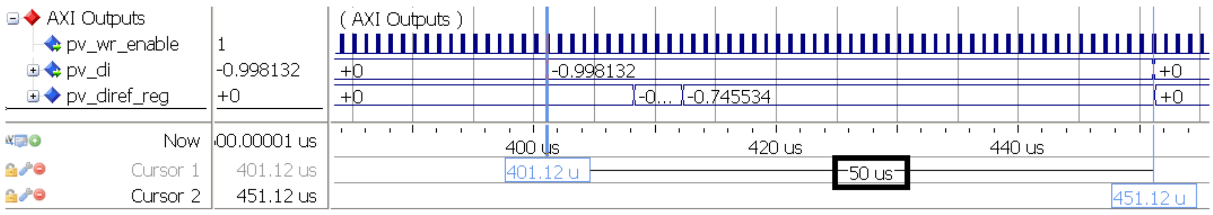


Fig 181 Delta Data for the Inductor Current and Load Current as Transferred over the AXI-Lite Bus Showing the Values being Held for 50 us During a 1A Load Current Decrease

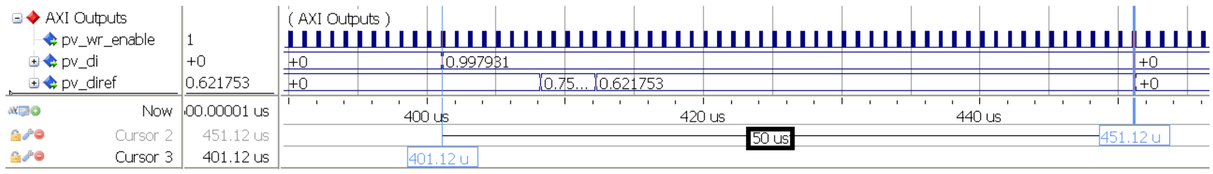


Fig 182 Delta Data for the Inductor Current and Load Current as Transferred over the AXI-Lite Bus Showing the Values being Held for 50 us During a 1A Load Current Increase

In the previous chapter, the operation of the ANFIS algorithm within the FPGA fabric was discussed in detail, simulated and tested on HW. This showed conclusively how the novel ANFIS architecture is capable of reducing the latency of the calculation without excessively increasing the logic utilisation. As part of these simulations, a comparison was made with an established technique which showed that the total latency for the existing architecture is 41 clock cycles. In comparison, the novel ANFIS architecture required between 27 and 37 clocks to perform a calculation, depending on the exact inputs. This reduction in latency is again witnessed in the PSim-QuestaSim simulation model, where it has been observed that in all of the simulations between 29 and 37 clocks are required for an ANFIS calculation. This is shown in Fig 183 and Fig 184 where the in_valid signal indicates the start of a calculation when it is asserted and the fis_valid signal indicates the conclusion. The minimum number of clocks is slightly higher than that demonstrated in the previous chapter due to the fact that the requisite input combination was not observed. Despite this, the latency of the ANFIS model in this model still represents a clock saving of between four and twelve clocks in comparison to the existing implementation techniques.

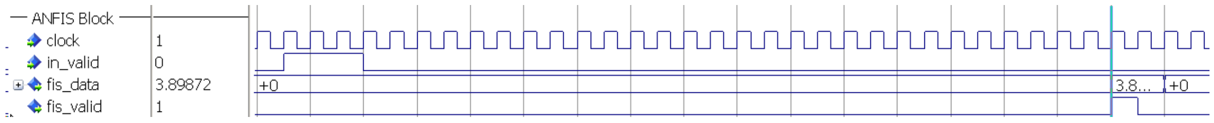


Fig 183 Novel ANFIS Calculation Showing 29 Clock Cycles Between the Start and End Calculation Pulses (in_valid and fis_valid respectively)

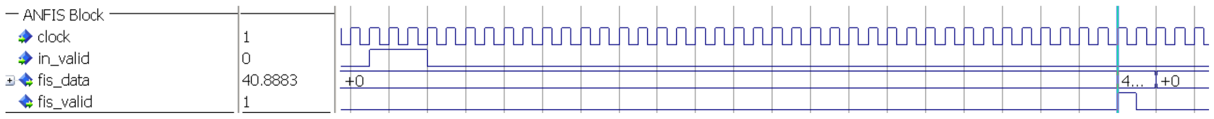


Fig 184 Novel ANFIS Calculation Showing 37 Clock Cycles Between the Start and End Calculation Pulses (in_valid and fis_valid respectively)

The final part of the digital controller which can be closely considered is the PWM generator. This block has three different modes of operation depending on the current state of the converter – disabled, fixed 50% duty cycle or enabled. These operating modes are designed to accommodate the operation of the soft start circuit. When the output capacitor is charged with a fixed current during the first phase of the soft start operation, the PWM needs to remain inactive. Once the input and output voltages are roughly equal, the soft start circuit routine then requires a fixed 50% duty cycle. Finally, when the output voltage has safely been increased to an approximate operating voltage of 48V, the PWM generator is enabled and controlled by the main loop. This operation is demonstrated in Fig 185, where it can be seen that a 50% duty cycle is produced when the fixed_pwm control signal is asserted. Once the soft start is completed, the fixed_pwm signal is de-asserted and the enable_pwm signal becomes active. It can be observed in Fig 185 how, when this occurs, the 50% duty cycle PWM is no longer active and the PWM output starts to ramp up as controlled by the PI algorithms. Prior to this, both of the control signals are inactive and the PWM output is inactive.

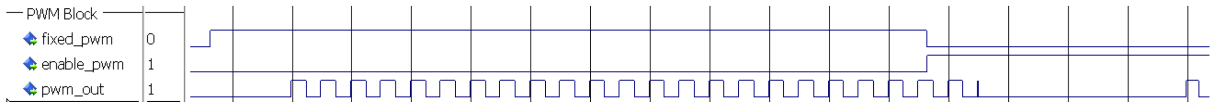


Fig 185 Simulation of the PWM Block with a Fixed 50% Duty Cycle during Soft Start Mode as Controlled by the “fixed_pwm” Signal

In the previous chapter, the operation of the PWM was described in detail. One of the key features of this was the MASH modulator, which extends the effective resolution of the PWM generator. This modulator causes small variations in the duty cycle of the output, with the average duty cycle value having a higher resolution than would otherwise be achievable. This operation can again be observed in this simulation model. The expected duty cycle of the PWM signal can be determined using the formula given in Eq. 43 . Considering the case where a 28V input is used, this would require a duty cycle of 41.667% for a 48V output. The simulation given in Fig 186 shows the case where the power converter is in a steady state of operation with a 28VDC input. The reference input to the PWM driver logic, denoted by the signal pwm_ctrl_sync, is 6AAh which is approximately 41.667% of the maximum 12-bit input. The output of the MASH modulator circuit is given by the signal duty_ref, which can be seen to oscillate between different reference points. The full range of values seen at the output of the MASH circuit is given in Table 19 along with the duty cycle at that point. Taking the average of the measured duty cycles, which gives 41.956%, shows how the duty cycle error is just 0.69%. This further demonstrates how the oscillations in the MASH output can be used to create a high-resolution PWM output signal.

$$D = 1 - \frac{Vin}{Vout} = 1 - \frac{28}{48} = 41.667\% \qquad \text{Eq. 43}$$

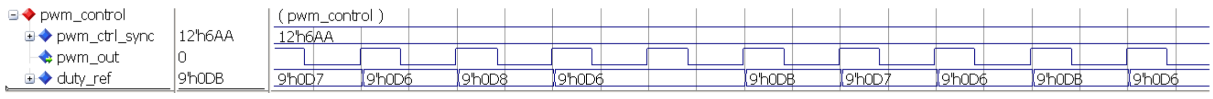


Fig 186 PWM Simulation with 41.956% Duty Cycle Showing Operation of the MASH Modulator ("duty_ref" Signal)

Duty Ref Value	**Duty Cycle (%)**
0D6h	41.8
0D8h	42.18
0D6h	41.8
0D6h	41.8
0D8h	42.18
0D7h	41.98
Average	41.956
Expected	41.667

Table 19 PWM Operation

The simulation results presented thus far have illustrated the validity of the digital implementation of the novel power converter controller. One of the major advantages of the PSim-QuestaSim co-simulation approach is that the analogue circuitry which is being driven can be considered alongside the digital controller. This allows for the performance to be analysed and contrasted with the wholly analogue results which were previously discussed. Six simulations have been carried out to this end, considering the steady-state and dynamic operation of the converter.

The first set of tests considers the instances where the converter has a fixed load current, firstly of 1A and then of 4A. In the previous results, the comparable ripple voltages were measured as 1mV and 3mV respectively. It can be observed from Fig 187 that this figure has increased somewhat for the digital implementation, with the ripple in both instances now being 4mV. Under heavy load conditions, this is attributable to the reduction in the resolution of the PWM block as well as the increased latency of the controller algorithm. In the original simulations, the PWM circuit is implementable using an op-amp based comparator and a triangle wave. This means that the resolution is effectively unlimited, considering that the op-amp model exhibits ideal behaviour. Likewise, the error signal is updated in every time step of the simulator, meaning that the latency of the controller is only limited by the simulation environment. The digital controller, conversely, has a limited resolution PWM module and is only updated once every microsecond. The result of this is that the controller is slightly less sensitive to errors and, therefore, takes longer to apply corrective actions.

In the case of the 1A load, the increase in the ripple voltage is more directly attributable to the sampling which is necessarily introduced by the control loop. The original measurement is slightly below the minimum resolution of the ADC (~1mV). As a direct result, the output can exhibit a small amount of drift which isn't visible to the digital controller. It's only when this voltage drift is greater than the resolution of the ADC that the control loop applies a corrective action. This has the effect of creating a small amount of oscillation around the voltage set point, increasing the observable ripple. This could be mitigated or reduced by increasing the number of bits in the ADC conversion. However, this would require an increase in expense for the ADC as well as increasing the complexity of the signal conditioning circuitry by demanding greater noise immunity. Given that a real circuit would exhibit output noise which is already likely to be greater than 1mV, owing to impurities such as capacitive equivalent series resistance (ESR), there would be little to be gained from increasing the ADC resolution in reality. This oscillation in is not seen under heavy loading as the voltage ripple is instead dominated by the ripple current of the inductor.

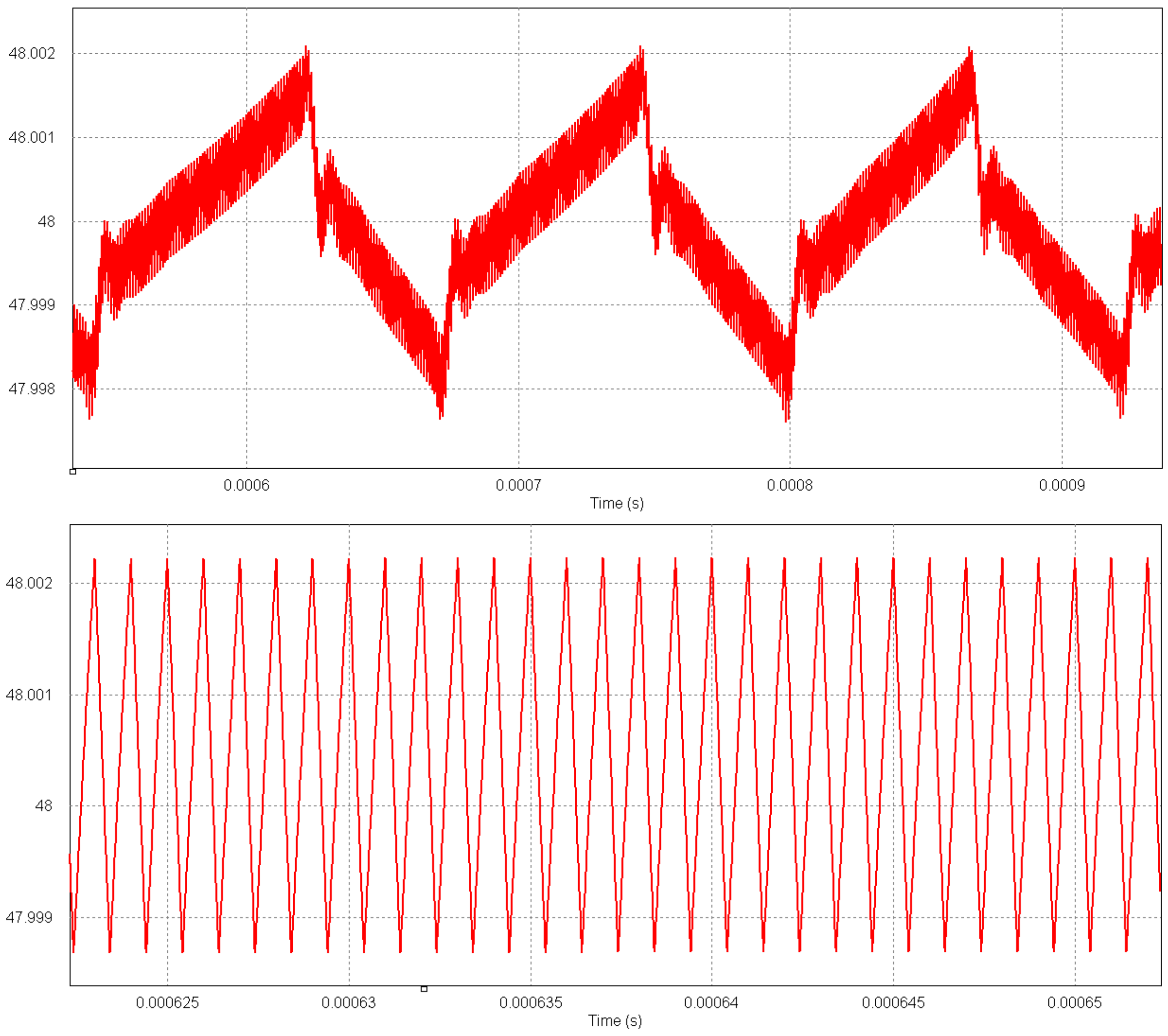
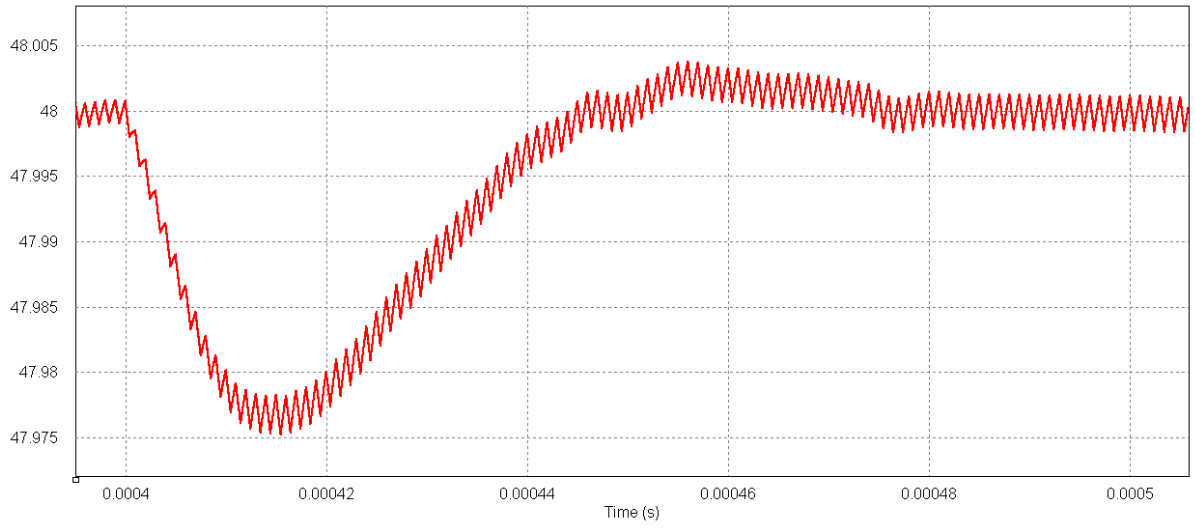


Fig 187 Boost Converter Ripple Voltage with a Steady 1A Load (Top) and 4A Load (Bottom)

In addition to the steady state mode of operation, it is also important to consider the dynamic operation of the power converter. To this effect, simulations are carried out with both a 1A and 4A step in load current. In the original simulations, the 1A step increase yielded a maximum error of 21mV whilst a decrease of the same size caused an error of 16mV. In both instances, this has been increased through the implementation of the controller in the digital domain, as is shown in Fig 188, with the errors now becoming 25mV and 20mV respectively. Likewise, the settling time has also increased, from 34 us to 40 us in the case of the step increase and from 28 us to 35 us in the case of the decrease.



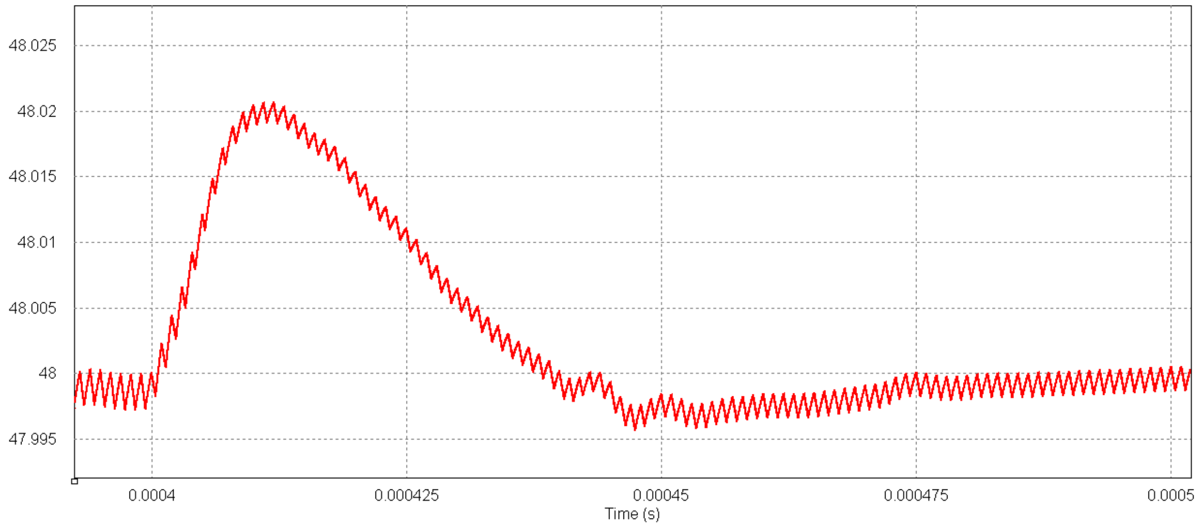
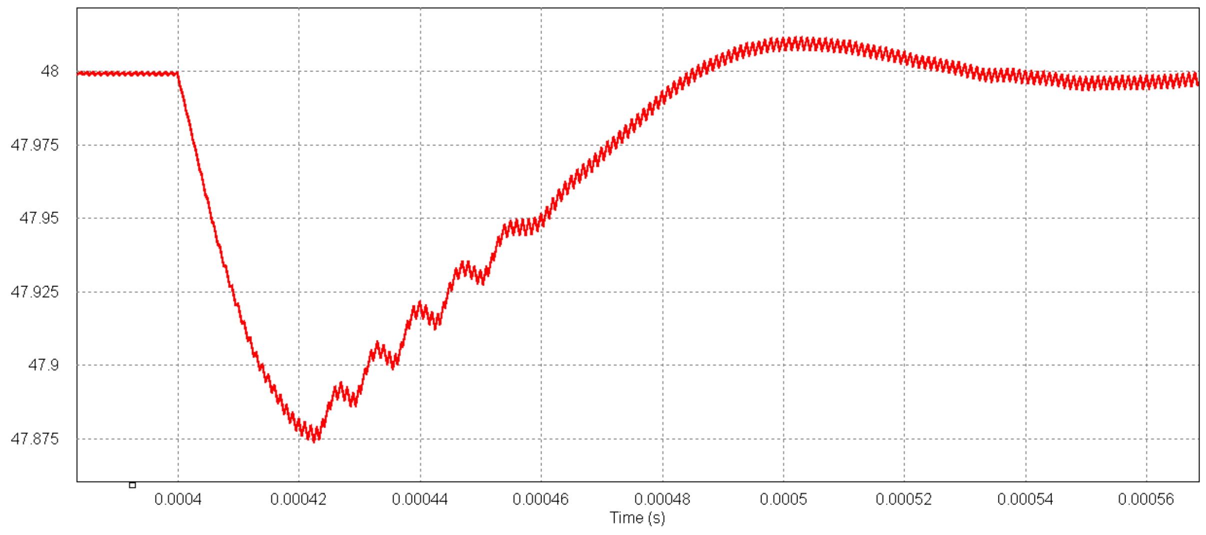


Fig 188 Boost Converter Output Voltage with a 1A Decrease in Load Current (Top) and 1A Decrease in Load Current (Bottom)

In Fig 189, the simulation results for the 4A decrease and increase in load current are shown. In these results, the maximum error is observed as 125mV for the increase and 90mV for the decrease. As with the previous simulations which considered the 1A step change, this represents a slight degradation in comparison to the previous model which had an absolute error of 90mV and 82mV respectively. This can be attributed to the fact that the sampling rate of the ADC means the controller can’t react as quickly to disturbances, as has previously been discussed. In contrast to this, the settling time observed has reduced from 145 µs to 120 µs, in the case of the step increase, and from 104 µs to 90 µs for the step decrease. This is attributable to the fact that the larger initial error necessitates a greater control effort, resulting in a reduced settling time. However, it can be observed that there is an increase in the instability of the system as a result of this, as shown in Fig 189 by the oscillations and undershoot as the output voltage recovers.



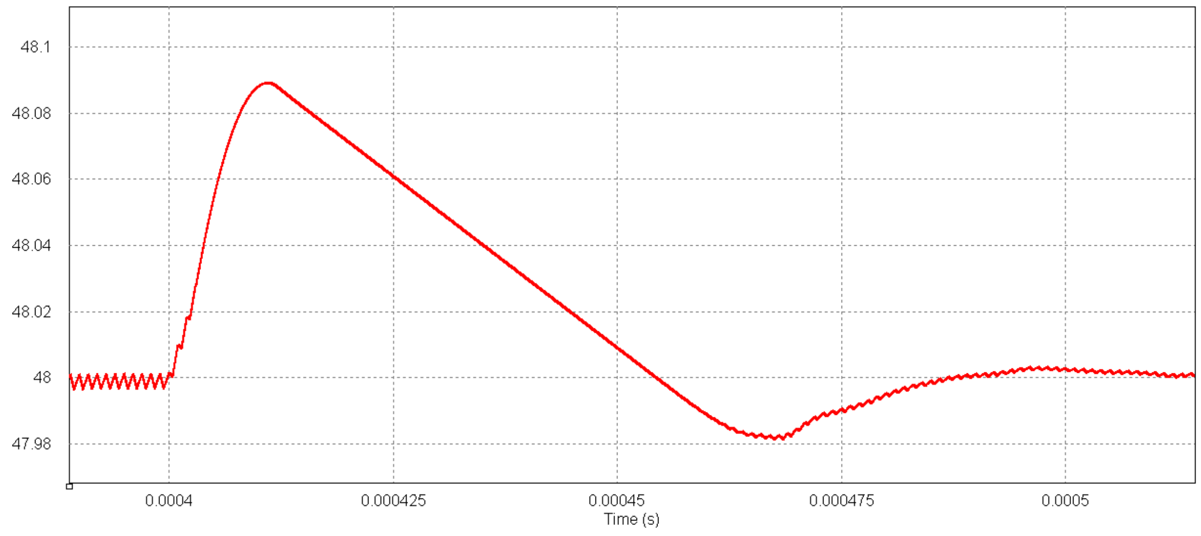


Fig 189 Boost Converter Output with a 1A Increase in Load Current (Top) and 4A Increase in Load Current (Bottom)

The differences between the original simulations and those featuring the digital implementation, which is tabulated in Table 20, can be ascribed to the increased latency of the control loop in the digital domain. As was previously discussed in the steady-state results, the digital control loop is updated once every 1 µs, compared to every simulation time step for the previous simulations. This has the obvious effect of increasing the response time of the controller which in turn leads to a greater absolute error. Despite this slight reduction in performance, the power converter is still capable of delivering a high quality of power. Even under dynamic operating conditions, the worst observable error is considerably less than 1% whilst the worst measured settling time is 125 µs and, in all other instances, this value is less than 100 µs.

Load	Digital Implementation			Original Simulations		
	Max Error		Settling Time (µs)	Max Error		Settling Time (µs)
	mV	%		mV	%	
1A Increase	25	0.052	40	21	0.043	34
1A Decrease	21	0.043	35	16	0.033	28
4A Increase	125	0.26	120	90	0.188	145
4A Decrease	90	0.188	90	82	0.177	104

Table 20 Comparison between the Original Simulations and the Digital Implementation Simulations

8.3 Experimental Testing

The final stage of the validation of the novel power converter controller is the experimental validation of the APSOC. In order to do this, the ANFIS engine and the PWM generator are tested in isolation within the device to ensure their operation. These tests are presented in the rest of this section.

8.3.1 PWM Testing

In the previous chapter, the PWM block was simulated and some tests were carried out using the ILA core. The architecture used for the generation of the PWM signal utilises a combination of the DCM approach, as presented by Batarseh et al (2009), together with a sigma-delta modulator to increase the effective resolution, as presented by Guo et al (2009). As this is a relatively new approach, which has an important influence on the performance of the converter, this block is tested in isolation to ensure its correct performance. To do this, the input reference is set to emulate the steady state performance with an input of 24V, 26V, 28V and 30V. The output of the PWM block is then captured using an oscilloscope to measure the duty cycle.

In Table 21, the measured results of the PWM output circuit are recorded and given against the expected duty cycle values. Using this measurement, the error of the duty cycle is then calculated and is additionally given in this table. It can be observed from this that the maximum measured error is 0.51%, as is shown in Fig 182. The errors in the PWM duty cycle can be attributed partially to the PWM block, which was observed to give small errors owing to a non-infinite resolution in the previous chapter. It should also be considered, however, that the oscilloscope is also likely to have some inaccuracy in its measurement which contributes to this. In any case, the captured waveforms demonstrate the PWM functions with a high degree of accuracy.

Input Voltage	Expected Duty Cycle	Measured Duty Cycle	Error
24	50.00	50.05	0.10
26	45.83	45.82	0.02
28	41.60	41.62	0.05
30	37.50	37.69	0.51

Table 21 PWM Test Results

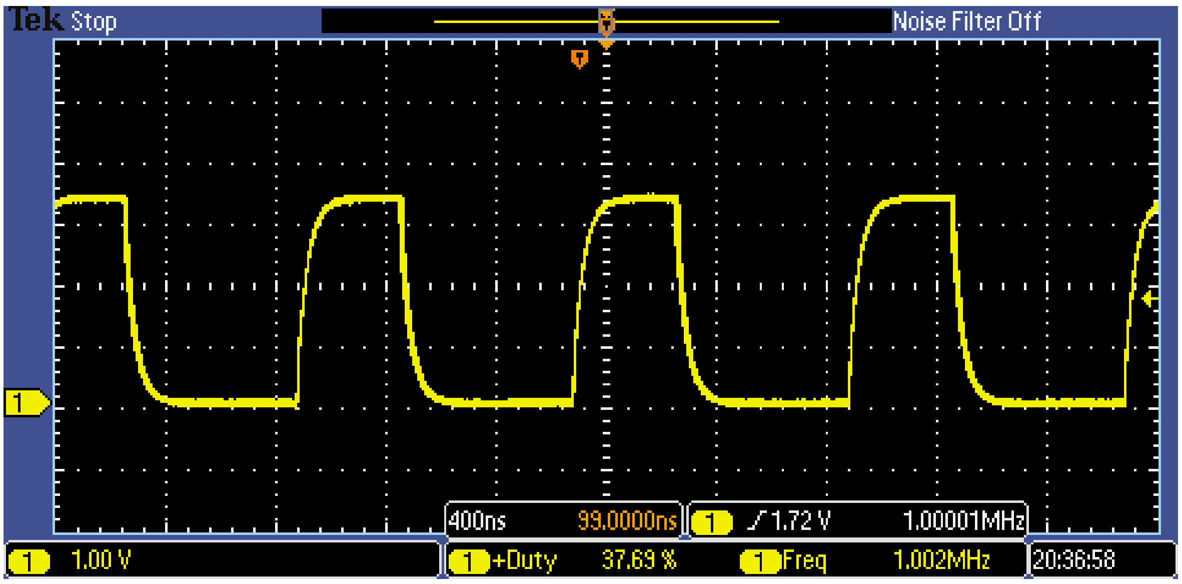


Fig 190 30VDC Emulated PWM Output with a 37.69% Duty Cycle

8.3.2 ANFIS HW Testing

In the previous chapter, the implementation of the ANFIS engine within the FPGA fabric was discussed and analysed. This included the presentation of simulation models and basic hardware testing. As the ANFIS presents a major area of novelty, as well as performing a crucial element of the control loop, its operation is again validated as part of the overall APSOC behaviour. In order to achieve this, a number of test patterns are generated within the CPU and subsequently used to stimulate the ANFIS engines. The outputs of these engines can then be captured using the onboard ILA. The captured data is subsequently imported into MATLAB and comparisons are made with the output of the MATLAB model. This is done for each of the four ANFIS blocks, as is shown in Fig 191 to Fig 194, with the full set of captured data from the FPGA core being given in Appendix D. These results show a maximum error of 0.07, which can largely be attributed to the fact that the FPGA uses single precision floating point. In comparison, the underlying data types in MATLAB utilise double precision floating point numbers, meaning that there is a loss of precision which is propagated through the logic. In order to simplify its operation, the LUTs used in the calculation of the input members also removes any values which are less than 1e^{-6} as they have a minimal effect on the calculated output values. This serves to increase the calculation speed of the logic but does result in a small increase in error.

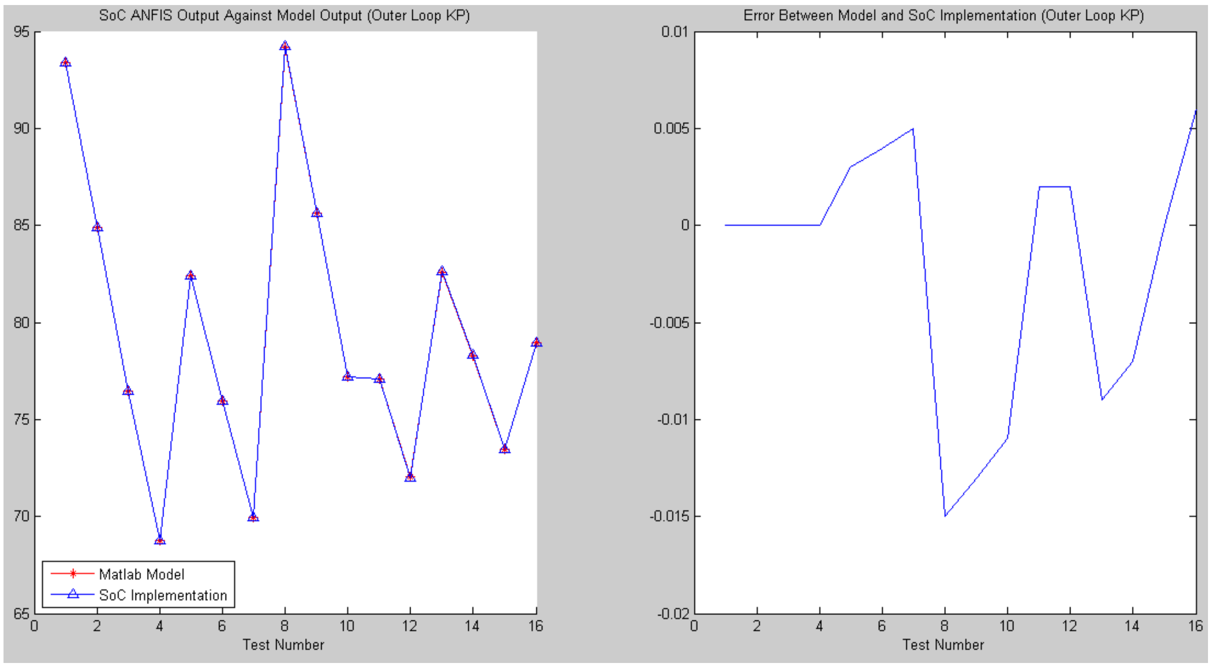


Fig 191 ANFIS Test Results for the Outer Loop KP Calculator

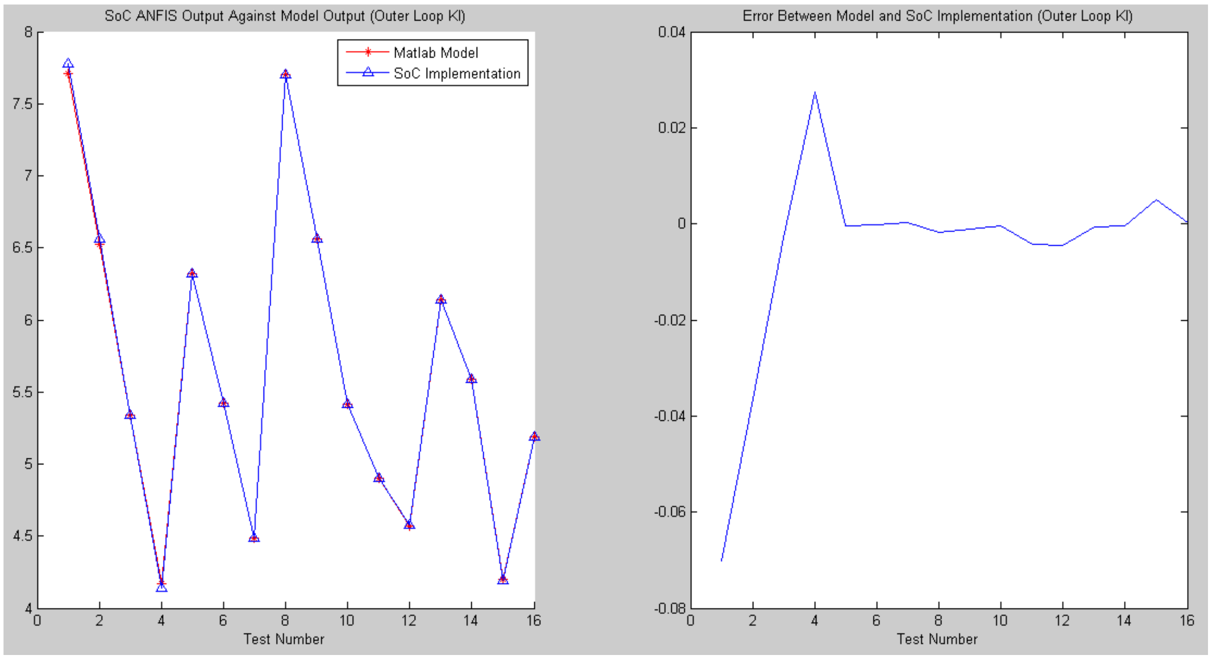


Fig 192 ANFIS Test Results for the Outer Loop KI Calculator

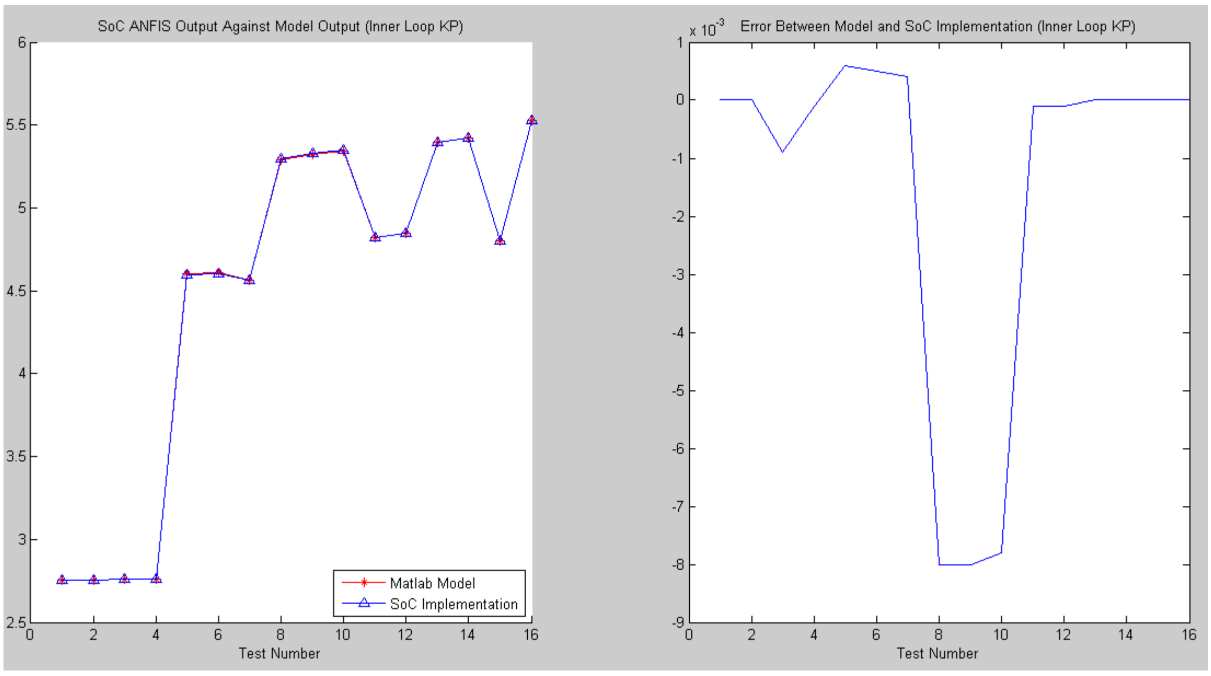


Fig 193 ANFIS Test Results for the Inner Loop KP Calculator

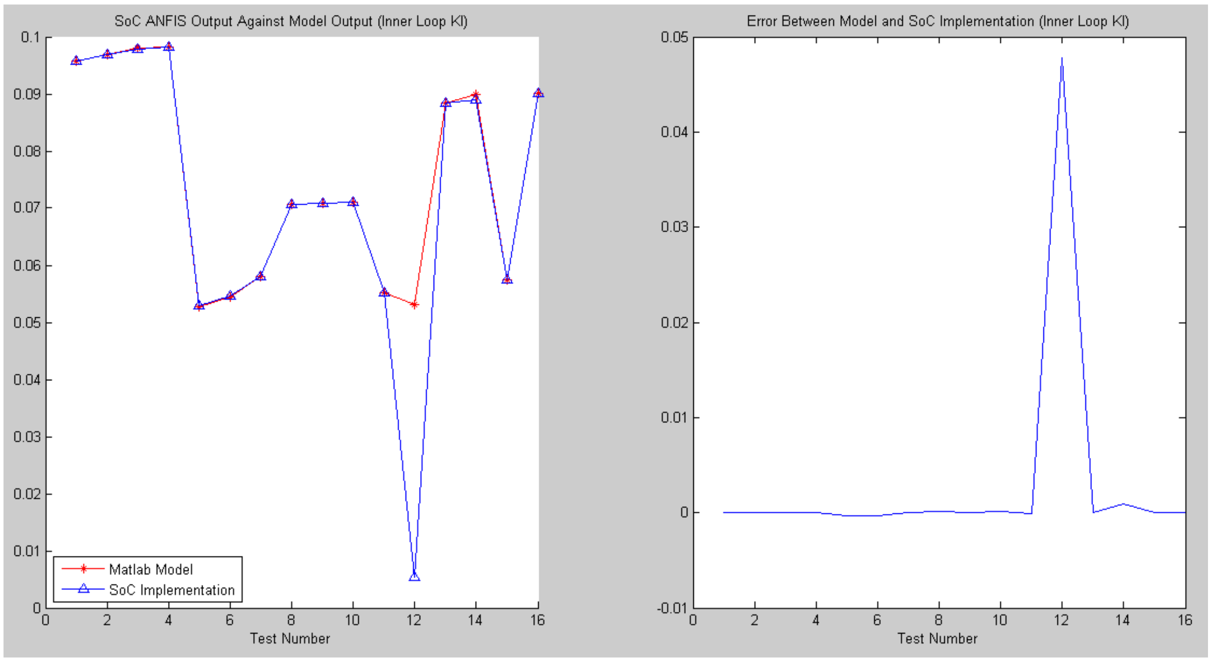


Fig 194 ANFIS Test Results for the Inner Loop KI Calculator

8.4 Conclusions

In this chapter, some validation of the novel power converter control solution is presented with the aim of demonstrating the performance of the areas of novelty within the digital controller. This begins with a discussion of the signal conditioning circuits which are required to interface the power converter with the APSOC. A number of simulations are presented as a part of this discussion to illustrate the performance of these circuits and contextualise how the digital controller interacts with the analogue plant. Co-simulation between QuestaSim and PSim is then utilised in order to validate the VHDL models, which were discussed in detail in the previous chapter, alongside a model of the power electronics. The results of these tests show that the digital implementation causes some degradation of the converter performance in comparison to the analogue models. Despite this, the controller is still shown to produce excellent regulation of the output voltage, showing a maximum error of 125mV (0.26%) even under large load transients. The settling time of the output voltage is also measured at just 125 µs. Finally, the key elements of the novel digital controller were tested using the Zedboard Hardware Platform. These tests demonstrate the capability of the PWM block to generate a high-frequency PWM signal with a high accuracy. In addition to this, it is demonstrated that the novel architecture employed for the ANFIS engines give analogous behaviour to the models which have previously been developed using Matlab.

9 Conclusions

This research project proposed and developed a novel control solution suitable for use in a power electronic converter within a DER type system. Such a system was modelled, implemented and verified through extensive simulation and experimental testing in order to demonstrate the performance of this controller. The novel solution utilises an ANFIS based algorithm to improve the performance of the essentially linear PI controller with a highly non-linear plant. This was shown to improve the regulation of the power converters, which subsequently improved the quality of the power delivered. This section highlights the achievements and originality of this project before some areas of further work are presented.

9.1 Summary of Achievements

As was briefly mentioned in the introduction, there are a total of four objectives for this research project. The achievements related to these objectives are more closely considered in this subsection and are justified through the underpinning research work and results presented earlier in this thesis.

The first objective of this project was the creation of a model for a hybrid renewable energy system using a novel modelling approach. The work towards this objective began with a comprehensive literature review, summarised in chapters 2 and 3, which was carried out to identify existing trends in the published research. Using this information, a topology was devised and a novel control solution formulated, both of which are discussed in depth in chapters 5 and 6. A review of the existing modelling techniques was carried out, as presented in chapter 4, which has enabled the formulation of an efficient modelling approach. Using all of this information, a model (including sub-models) has been created for the hybrid renewable energy system. These models utilise co-simulation between PSim and Matlab, which provides an efficient modelling approach capable of exploiting the advantages of both of these tools.

The second objective of this project was the extensive simulation of the system model, aiming to validate its performance. This aspect is initially covered in chapter 5, where the individual power converters within the system are considered. The results of these tests show how the novel control solution is capable of delivering a high degree of accuracy and a fast response time. The performance of the system as a whole is then considered in chapter 6, with the system being simulated under a range of different operating scenarios. The tests carried out demonstrate the way in which the renewable energy sources complement one another towards achieving an improved efficiency of the system, with between 91.2% and 98.4% of the power generated by the renewable energy sources being utilised.

The next objective of the project was the implementation (virtual and physical) of the control solution within a suitable hardware device, specifically an APSOC device. The realisation of this objective is presented in chapter 7. The architecture of the device is discussed at length, including the introduction of a new methodology for the realisation of the ANFIS algorithm. The VHDL modelling language is utilised to create a model of the hardware implementation, which is then extensively simulated within the Xilinx Vivado software environment. This same tool is also used to synthesise the design, allowing for temporal and logical utilisation analyses to be carried out. Finally, a number of tests are conducted on the programmed APSOC, which demonstrate the correct functional operation of the digital hardware implemented solution.

The final objective of this project was the validation of the digital controller implementation, which is considered in chapter 8. This begins with some discussion of the signal conditioning circuitry which is required to interface the power electronics and APSOC. Co-simulation between PSim and QuestaSim is then utilised in order to model the power electronics alongside the digital controller. These simulations demonstrate how the digital implementation exhibits the same ability to regulate the power converter output as was observed in the system model. Finally, the crucial blocks of the APSOC are tested within the device, validating their performance in relation to the previously developed models.

9.2 Contributions to Knowledge

In this project, a total of three original knowledge contributions have been identified. Whilst these are briefly discussed in the introduction, in the rest of this section they are considered in more detail with reference to the work which has been discussed.

The first original knowledge contribution is the use of a new modelling approach for the tuning of the PI controller. This started with the consideration of the different modelling tools outlined in chapter 4. As part of this discussion, the PSim and Matlab simulation environments were considered in detail. Given that these tools offer co-simulation with one another, a rapid prototyping methodology was created using these tools. In chapter 5, the PSO algorithm is discussed as a means for tuning PI controllers. This methodology offers a number of benefits, such as flexibility, automation and ease of implementation. The novel modelling approach is then detailed in section 5.1.2, showing how the benefits of Matlab are exploited for the PSO algorithm, whilst spice models are developed within PSim.

The next original contribution to knowledge was the creation of a novel gain scheduling control algorithm. The formulation of this idea is rooted in the literature review carried out in chapter 3, where one popular research trend was the use of a FLC based gain scheduling algorithm. The ANFIS algorithm was also considered as part of this background review, with explanations offered as to how it improves on the performance of fuzzy logic based algorithms. Finally, the ANFIS-PI algorithm is conceptually developed in section 5.1 in relation to a boost converter circuit. The performance of this new approach is considered in detail in chapter 5, where a number of simulations are presented demonstrating an absolute error of between 3.5mV and 90mV during load transients, whilst a maximum settling time of 145 µs is observed. These results show how this algorithm benefits from a high degree of stability and fast response times.

The third original contribution relates to the implementation of the gain-scheduling controller within an FPGA. In chapter 7, the existing architectures which are utilised for ANFIS algorithms implemented in FPGA fabrics are reviewed. As part of this discussion, inefficiencies are identified which increase the latency of the calculation and have a negative impact on scalability. A new approach is then presented at the conceptual level, offering explanations of how the efficiency of the architecture can be improved by making a few modifications to the underlying logic. Finally, a number of simulations and analyses are presented, which demonstrate that the novel approach is capable of reducing the calculation latency by up to 14 clocks (34 %) whilst requiring an increase in logic utilisation of just 2.19% for LUTs and 0.38% for flip-flops.

9.3 Further Work

There are a number of different ways in which the research presented in this thesis could be further expanded; some of these are discussed in this section. This further work is broadly split into two areas – enhancements to the existing system and potential areas of further investigation.

9.3.1 System Enhancements

In chapter 8, validation of the novel power converter controller was carried out with the intention of demonstrating the viability of the gain scheduling algorithm. Whilst this included some testing of the system algorithms within the real hardware, one obvious area of future work would be in conducting further testing of the system. In the first instance, this would relate to the boost converter circuit and APSOC, which was considered as part of chapter 8. By implementing the converter in a real hardware platform, it would be possible to repeat the PSim-QuestaSim tests to further validate the operation of the ANFIS-PI solution. In addition to this, testing of the full renewable energy system could be carried out with the intention of validating the performance of the state-based algorithm presented in chapter 6.

Another future consideration for this work would lie in increasing the power capabilities of the renewable energy sources. The main aim of this project has been the creation of a new prototype system as a pilot at a low power level, to demonstrate the validity of the ideas and concepts developed. However, power demands of real hybrid renewable energy systems are much greater than those presented in this project. Therefore, the technologies / solutions presented could be utilised to develop a higher power system, which would have more direct real-world applications.

Another way the system could be enhanced, in order to make it more directly usable in a real-world situation, is through the inclusion of a grid connection. This would necessitate the inclusion of a grid-tied inverter circuit but would bring a number of advantages. Firstly, this would allow for the stability of the system to be improved as power can be readily sourced from the grid as needed. Whilst this power would obviously be produced from environmentally unclean sources, the control strategy could be designed so that the renewables generate power the majority of the time. This would bring a further benefit by allowing the system to sink surplus energy into the grid when it can’t be utilised by either of the storage elements. This is obviously favourable to the utilisation of a dump load to remove excess power.

The final improvement to the system mentioned here relates to the hardware implementation of the control algorithms. The APSOC approach offers many benefits, such as flexibility and reduced cost, which have previously been discussed in more depth. However, an ASIC based implementation approach would offer a number of benefits, despite the extra costs. One particular benefit which could be achieved using this technology is the increase in speed which is possible within such devices. As a more optimal layout can be achieved in comparison to FPGAs, it is typically possible for the ASIC devices to run at a much higher frequency. This methodology would also allow for closer integration of the signal conditioning circuits, as analogue components could be incorporated into the chip. This could potentially allow for the incorporation of all the ADC circuitry or some of the MOSFET driver circuitry, for example.

9.3.2 Areas of Further Investigation

In chapter 6, a state-based algorithm was presented, which is intended to supervise and manage the power within the system. This had a number of advantages, such as simplicity of implementation, but a number of other options could also have been explored for this controller. One trend which has grown in popularity as this research project has been carried out is the use of the ANFIS algorithm for such controllers. This has been shown to offer improvements in efficiency when compared to the state-based approach (Garcia et al (2014) and Mahmud et al (2017)). Therefore, one way in which future work could expand on the ideas presented in this project is through the utilisation of an ANFIS algorithm rather than a state-based approach. This would be an especially relevant development within the context of this project, given that one of the major accomplishments has been the creation of a more efficient ANFIS implementation approach for FPGA fabrics.

As well as improvements to the state-based algorithm, another area of potential future work would be the investigation of improvements to the novel ANFIS-PI based controller. In the current approach, a simple feedback approach is utilised, whereby the control law depends wholly upon the measured output current and voltage. Given that the renewable energy sources are highly variable and unpredictable, adding a feedforward element to the control law could further optimise the performance of the regulator by compensating for this unpredictability. This could take the form of a more complex PI-based algorithm, which features feedforward control from the renewable sources that contribute to the control law. Another approach would be the expansion of the ANFIS algorithms so that the PI controller gains also have a dependency on the current state of the renewable energy sources.

The PI algorithm is favoured over the PID approach in this project owing to the fact that it offers a higher degree of noise immunity. However, the inclusion of a derivative action in the control loop offers the benefit of improved response times when transients occur. Therefore, another idea which could be explored is the inclusion of such a derivative action within the control law. Given that the ANFIS algorithm offers the ability to alter the gain depending on the state of the system, the derivative could be managed so that it only contributes to the control output during dynamic operation. This way the noise immunity would be unaffected during the steady-state of operation, whilst the transient response could potentially be further improved through the exploitation of the derivative action.

Another technique which is commonly used in the control of power converters is the H-infinity method (Zhou and Hu (2015) and Khayat et al (2017)). As with the PID controller, this method relies on being tuned to give optimal performance with a given application. As was discussed earlier in this thesis, this can be difficult in power converters as the variable loads mean the damping of the system is not fixed. Therefore, another possible area of further investigation would be the use of the ANFIS algorithm in conjunction with the H-infinity method. This approach could be utilised to adaptively tune the weight of the H-infinity based controller and improve its performance with a non-linear plant such as a power converter. This would be another area of potential further research based on the approach developed in this research project.

Finally, another way in which the ideas presented in this project could be expanded on is through the application to different power converter topologies. The ANFIS-PI control solution utilised for the boost converters in chapter 5 has been demonstrated to offer a high degree of stability and a fast response time. This methodology is equally suitable for other topologies and this is another area which could be explored. This extends to more complex DC-DC converters, such as the full bridge or interleaved boost converter schemes, as well as inverters and AC-AC converters, such as matrix converters or cycloconverters.

References

Abbas, G., Farooq, U. and Asad, M.U., 2011. Application of Neural Network Based Model Predictive Controller to Power Switching Converters. International Conference and Workshop on Current Trends in Information Technology (CTIT), pp.132-136. Available at: http://ieeexplore.ieee.org/ [Accessed February 2013]

Abbas, G., Farooq, U. and Asad, M.U., 2011. Fuzzy Logic Based Robust Pole-Placement Controller for DC-DC Buck Converter. International Conference on Information and Communication Technologies, pp. 1-6. Available at: http://ieeexplore.ieee.org/ [Accessed May 2013]

Abhinav, R. and Sheel, S., 2012. An Adaptive, Robust Control of DC Motor using Fuzzy-PID Controller. 2012 IEEE International Conference on Power Electronics, Drives and Energy Systems. Available at: http://ieeexplore.ieee.org/ [Accessed July 2018]

Acker, F., 2009. Taming the Yangzte. IET Engineering and Technology, [e-journal] 4(4). Available at: https://eandt.theiet.org/content/articles/2009/03/taming-the-yangtze/ [Accessed August 2017]

Afghoul, H. and Krim, F., 2012. Intelligent Energy Management in a Photovoltaic Installation using Neuro-FuzzyTechnique. 2012 IEEE International Energy Conference and Exhibition (ENERGYCON), pp. 20-25. Available at: http://ieeexplore.ieee.org/ Accessed March 2013

Alajmi, B.N., Ahmed, K.H., Finney, S.J. and Williams, B.W., 2013. A Maximum Power Point Tracking Technique for Partially Shaded Photovoltaic Systems in Microgrids. IEEE Transactions on Industrial Electronics, 60(4), pp. 1596-1606. Available at: http://ieeexplore.ieee.org/ [Accessed April 2013]

Althubaiti, M., Bernard, M. and Musilek, P., 2017. Fuzzy Logic Controller for Hybrid Renewable Energy System with Multiple Types of Storage. IEEE 30th Canadian Conference on Electrical and Computer Engineering. Available at: http://ieeexplore.ieee.org/ [Accessed November 2017]

Analog Devices, 2017. AD8212 High Voltage Current Shunt Monitor (Rev. C). Available at: www.analog.com/media/en/technical-documentation/data-sheets/AD8212.pdf [Accessed September 2017]

Anvari-Moghaddam, A., Guerrero, J.M., Vasquez, J.C., Monsef, H. and Rahimi-Kian, A., 2016. Efficient Energy Management for a Grid-Tied Residential Microgrid. IET Generation, Transmission and Distribution, 11(11), pp. 2752-2761, Available At: http://ieeexplore.ieee.org/ [Accessed January 2018]

ARM Limited, 2010. ARM Information Centre. [online] Available at: http://infocenter.arm.com/help/index.jsp. [Accessed March 2017]

Bae, S. and Kwasinkski, A., 2012. Dynamic Modelling and Operation Strategy for a Microgrid With Wind and Photovoltaic Resources. IEEE Transactions on Smart Grid, 3(4), pp. 1867-1876. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Baek, J.W., Ko, J.S., Choi, J.S., Kang, S.J. and Chung, D.H., 2010. Maximum Power Point Tracking Control of Photovoltaic System using Neural Network. 16th International

Conference on Intelligent System Application to Power Systems, pp. 638-643. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Bansal, J.C., Singh, P.K., Saraswat, M., Verma, A., Jadon, S.S. and Abraham, A., 2011. Inertia Weight strategies in Particle Swarm Optimization. Third World Congress on Nature and Biologically Inspired Computing, pp. 633-640. Available at: http://ieeexplore.ieee.org/ [Accessed October 2013]

Batarseh, M. G., Al-Hoor, W., Huang, L., Iannello, C. and Batarseh, I., 2009. Window-Masked Segmented Digital Clock Manager FPGA Based Digital Pulsewidth Modulator Technique. IEEE Transactions on Industrial Electronics, 24(11), pp. 2649-2660, Available at: http://ieeexplore.ieee.org/ [Accessed November 2016]

Bayindir, R., Colak, I. and Kaplan, O. and Can, C., 2011. MATLAB/GUI based simulation for photovoltaic systems. International Conference on Power Engineering, Energy and Electrical Drives, pp. 1-4. Available at: http://ieeexplore.ieee.org/ [Accessed November 2013]

Bevrani, H., Habibi, F., Babahajyani,P., Watanabe, M. and Mitani, Y., 2012. Intelligent Frequency Control in an AC Microgrid: Online PSO-Based Fuzzy Tuning Approach. IEEE Transactions on Smart Grid, 3(4), pp. 1935-1944, Available at: http://ieeexplore.ieee.org/ [Accessed May 2014]

Biju, K. and Ramchand, R., 2013. Modelling and simulation of single phase five level inverter fed from renewable energy sources. International Conference on Microelectronics, Communications and Renewable Energy, pp. 574-579. Available at: http://ieeexplore.ieee.org/ [Accessed November 2013]

Biswas, I. and Bajpal, P., 2014. Control of PV-FC-Battery-SC Hybrid System for Standalone DC Load. 18th National Power Systems Conference, pp. 1-6. Available at: http://ieeexplore.ieee.org/ [Accessed September 2017]

Brenna, M., Longo, M., Yaici, W. and Abegaz, T.D., 2017. Simulation and Optimization of Integration of Hybrid Renewable Energy Source and Storages for Remote Communities Electrification. IEEE PES Innovative Smart Grid Technologies Conference Europe. Available at: http://ieeexplore.ieee.org/ [Accessed January 2018]

Briscoe, M. 2017. Germany Demands Electric-Car Quotas as India to be All Electric by 2030. Irish Times. [online] May 3 2017. Available at: http://www.irishtimes.com/life-and-style/motors/germany-demands-electric-car-quotas-as-india-to-be-all-electric-by-2030-1.3069798 [Accessed July 2017]

Brown, A., Muller, S. and Dobrotkova, S., 2011. Renewable Energy Markets and Prospects by Technology. [pdf]. Available at: http://www.iea.org/ [Accessed May 2013]

Cai, Y., Mai, Y., Mai, K. and Mutlu, O., 2015. Comparative Evaluation of FPGA and ASIC Implementations of Bufferless and Buffered Routing Algorithms for On-Chip Networks. International Symposium on Quality Electronic Design, pp. 475-484. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]Castellazzi, A., Dai, T., Li, J., Solomon A., Trentin, A. and Wheeler, P., 2013. Integrated Matrix Converter Switch. IEEE 10th International Conference on Power Electronics and Drive Systems, pp. 525 – 530. Available At: http://ieeexplore.ieee.org/ [Accessed November 2017]

Chang, C., Yuan, Y., Jiang, T. and Zhou, Z., 2016. Field Programmable Gate Array Implementation of a Single Input Fuzzy Proportional-Integral-Derivative Controller for DC-DC Buck Converters. IET Power Electronics, 9(6), pp. 1259-1266. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Chaoui, H., Miah, S., Oukaour, A. and Gualous, H., 2015. State of Charge and State of Health Prediction of Lead Acid Batteries with Genetic Algorithms. 42nd Annual Conference of the IEEE Industrial Electronics Society, pp. 1-6. Available at: http://ieeexplore.ieee.org/ [Accessed January 2017]

Charkgard, M. and Farrokhi, M., 2010. State of Charge Estimation for Lithium-Ion Batteries using Neural Networks and EKF. IEEE Transactions on Industrial Electronics, 57(12), pp. 4178-4187. Available At: http://ieeexplore.ieee.org/ [Accessed November 2017]

Chen, G.Y., Perng, J.W. and Ma, L.S., 2015. DSP Based BLDC Motor Controller Design with Auto Tunning PSO-PID Algorithm. IEEE International Symposium on System Integration. pp. 766-770, Available at: http://ieeexplore.ieee.org/ [Accessed September 2017]

Chen, S., Kang, C., Zhang, Z. and Zhu, H., 2016. A Method for SOC Estimation for lead-acid Battery Based on Adaptive Extended Kalman Filtering Estimation. 42nd Annual Conference of the IEEE Industrial Electronics Society, pp. 18-24. Available at: http://ieeexplore.ieee.org/ [Accessed January 2017]

Chen, Y.M., Huang, A.Q. and Yu, X., 2013. A High Step-Up Three-Port DC–DC Converter for Stand-Alone PV/Battery Power Systems. IEEE Transactions on Power Electronics, 28(11), pp. 5049-5062. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Chew, S. H., Tseng, K. J. and Nguyen, H. T., 2010. An Energy Efficient 48VDC Bipolar LED Lighting System in a High-Rise Building. The 2015 International Conference in Power Electronics and Drive Systems. Available At: http://ieeexplore.ieee.org/ [Accessed September 2018]

Chikh, A., and Chandra, A, 2015. An Optimal Maximum Power Point Tracking Algorithm for PV Systems with Climatic Parameters Estimation. IEEE Transactions on Sustainable Energy, 6(2), pp. 644-652. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Chrisafis, A. and Vaughan, A., 2017. France to Ban Sales of Petrol and Diesel Cars by 2040. The Guardian. [online] July 2 2017. Available at: https://www.theguardian.com/business/2017/jul/06/france-ban-petrol-diesel-cars-2040-emmanuel-macron-volvo [Accessed July 2017]

Cirrincione, M., Pucci, M. and Vitale, G., 2013. Neural MPPT of Variable-Pitch Wind Generators With Induction Machines in a Wide Wind Speed Range. IEEE Transactions on Industry Applications, 49(2), pp. 942-953. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Clark, P. and Stothard, M., 2015. COP21: Paris Agreement Formally Adopted. Financial Times, [online] December 12 2015. Available at: https://www.ft.com/content/8677562c-a0c0-11e5-8d70-42b68cfae6e4 [Accessed July 2017]

Corradini, L., Bjeletić, A., Zane, R. and Maksimović, D., 2010. Fully Digital Hysteretic Modulator for DC-DC Switching Converters. IEEE Transactions on Power Electronics, 26(10), pp. 2969-2979. Available at: http://ieeexplore.ieee.org/ [Accessed December 2016]

Crockett, L., Elliot, R., Enderwitz, M., Stewart, R., 2014. The Zynq Book, Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC, Strathclyde Academic Media.

Dounis, A.I., Stavrinids, S., Kofinas, P. and Tseles, D., 2015. Fuzzy-PID Controller for MPPT of PV System Optimized by Big Bang-Big Crunch Algorithm. IEEE International Conference on Fuzzy Systems. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Eberhart, R. and Shi, Y., 1998. A Modified Particle Swarm Optimizer. IEEE International Conference on Evolutionary Computation Proceedings, pp. 69-73. Available at: http://ieeexplore.ieee.org/ [Accessed October 2013]

El Khateb, A., Rahim, N.A., Selvaraj, J. and Uddin, M.N., 2014. Fuzzy-Logic-Controller SEPIC Converter for Maximum Power Point Tracking. IEEE Transactions on Industry Applications, 50(4), pp. 2349-2358. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

El Telbany, M.E., Youssef, A. and Zekry, A.A., 2014. Comparison of ANN and P&O MPPT Methods for PV Applications under Changing Solar Irradiation. International Conference on Artificial Intelligence with Applications in Engineering and Technology, pp. 17-22, Available at: http://ieeexplore.ieee.org/ [Accessed September 2017]

Elobaid, L.M., Abdelsalam, A.K. and Zakzouk, H.I., 2012. Artificial Neural Network Based Maximum Popwer Point Tacking Technique for PV Systems. IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society, pp. 937-942. Available at: http://ieeexplore.ieee.org/ [Accessed February 2013]

Elshaer, M., Mohamed, A. and Mohammed, O.A., 2011. Smart Optimal Control of DC-DC Boost Converter for Intelligent PV Systems. 16th International Conference on Intelligent System Application to Power Systems, pp. 1-6. Available at: http://ieeexplore.ieee.org/ [Accessed March 2013]

Femmy Nirmal, J. and Jeraldin Auxillia, D., 2013. Adaptive PSO based tuning of PID controller for an Automatic Voltage Regulator system. International Conference on Circuits, Power and Computing Technologies, pp. 661-666. Available at: http://ieeexplore.ieee.org/ [Accessed July 2013]

Fukui, A., Takeda, T., Hirose, K. and Yamasaki, M., 2010. HVDC Power Distribution Systems for Telecom Sites and Data Centers. The 2010 International Power Electronics Conference. Available At: http://ieeexplore.ieee.org/ [Accessed September 2018]

Garcia, P., Garcia, C.A, Fernandez, L.M., Llorens, F. and Jurado, F., 2014. ANFIS-Based Control of a Grid-Connected Hybrid System Integrating Renewable Energies, Hydrogen and Batteries. IEEE Transactions on Industrial Informatics, 10(2), pp. 1107-1117. Available at: http://ieeexplore.ieee.org/ [Accessed January 2018]

Ge, R., Lin, Z., Gong, N. and Wang, J., 2017. Design and Performance Analysis of Energy Harvesting Sensor Networks with Supercapacitor. IEE International Midwest Symposium on Circuits and Systems, pp. .64-67. Available at: http://ieeexplore.ieee.org/ [Accessed December 2017]

George, K. and Ang, S., 2016. Topology Survey for GaN-Based High Voltage Step-Down Single-Input Multi-Output DC-DC Converter Systems. IEEE 4th Workshop on Wide Bandgap Power Devices and Applications. Available At: http://ieeexplore.ieee.org/ [Accessed September 2018]

Ghosh, A., Banerjee, S., Sarkar, M.K. and Dutta, P., 2015. Design and Implementation of Type-2 and Type-3 Controller for DC-DC Switched Mode Boost Converter by Using K-Factor Approach and Optimisation Techniques. IET Power Electronics, 9(5), pp. 938-950. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Gomez-Castaneda, F., Tornez-Xavier, Flores-Nava, L.M., Arellano-Cardenas, O. and Moreno-Cadenas J.A., 2014. Photovoltaic Panel Emulator in FPGA Technology Using ANFIS Approach. International Conference on Electrical Engineering, Computing Science and Automatic Control, pp. 1-6, Available at: http://ieeexplore.ieee.org/ [Accessed November 2016]

Guo, S., Lin-Shi, X., Allard, B., Li, B. and Ruan, Y., 2009. High Resolution Digital PWM Controller for High Frequency Low Power SMPS. European Conference on Power Electronics and Applications, Available at: http://ieeexplore.ieee.org/ [Accessed November 2016]

Gyawali, N., Ohsawa,Y. and Yamamoto, O., 2011. Power Management of Double-Fed Induction Generator-Based Wind Power System with Integrated Smart Energy Storage having Superconducting Magnetic Energy Storage/Fuel-Cell/Electrolyser. IET Renewable Power Generation, 5(6), pp. 407-421. Available at: http://digital-library.theiet.org/content/journals/iet-rpg/ [Accessed June 2013]

Haibin, S. and Jingjing, B., 2010. Maximum Power Point Tracking Algorithm Based On Fuzzy Neural Networks for Photovoltaic Generation System. International Conference on Computer Application and System Modelling, pp. 353-357. Available at: http://ieeexplore.ieee.org/ [Accessed May 2013]

Hajbani, F., Babaei, E., Hosseini, S.H. and Nouri, T. 2012. A new control method for single-phase to three-phase matrix converter based on SPWM. 20th Iranian Conference on Electrical Engineering, pp. 427-432. Available at: http://ieeexplore.ieee.org/ [Accessed May 2013]

Hajizadeh, A. and Golkar, M.A., P., 2009. Fuzzy neural control of a hybrid fuel cell/battery distributed power generation system. IET Renewable Energy Power Generation, 4(3), pp. 402-414. Available at: http://digital-library.theiet.org/content/journals/iet-rpg [Accessed April 2013]

Hanifah, R.A., Toha, S.F., Ahmad, S. and Hassan M. H., 2017. Swarm Intelligence Tuned Reduction for Power-Assisted Steering Control in Electric Vehicles. IEEE Transactions on Industrial Electronics, 65 (9), pp. 7202-7210. Available At: http://ieeexplore.ieee.org/

[Accessed August 2018] Hussain, S., Al-Ammari, R., Iqbal, A., Jafar, M. and Padmanaban, S., 2017. Optimisation of Hybrid Renewable Energy System using Iterative Filter Selection Approach. IET Renewable Power Generation, 11(11), pp. 1440-1445. Available at: http://ieeexplore.ieee.org/ [Accessed January 2018]

International Energy Agency 2013. Technology Roadmap: Wind Energy. [pdf] Available at: http://www.iea.org/publications/freepublications/publication/technology-roadmap-wind-energy---2013-edition.html [Accessed August 2017]

International Energy Agency 2014. Technology Roadmap: Solar Photovoltaic. [pdf] Available at: http://www.iea.org/publications/freepublications/publication/technology-roadmap-solar-photovoltaic-energy---2014-edition.html [Accessed August 2017]

International Energy Agency 2016. World Energy Outlook 2016. [pdf] Available at: http://www.iea.org/publications/freepublications/publication/world-energy-outlook-2016---executive-summary---english-version.html [Accessed August 2017]

International Energy Agency, 2011. Technology Roadmap: Geothermal Heat and Power. [pdf] Available at: http://www.iea.org/publications/freepublications/publication/Geothermal_Roadmap.pdf [Accessed August 2017]

International Energy Agency, 2017. Renewables Information: Overview. [pdf] Available at: http://www.iea.org/publications/freepublications/publication/RenewablesInformation2017Overview.pdf [Accessed August 2017]

Iqbal, A., Abu-Rub, H. and Ahmed, S.M., 2010. Adaptive neuro-fuzzy inference system based maximum power point tracking of a solar PV module. 2010 IEEE International Energy Conference and Exhibition (ENERGYCON), pp. 51-56. Available at: http://ieeexplore.ieee.org/ [Accessed April 2013]

Ishizuka, Y., Mii,K., Takenami, F., and Kanemoto, D., 2014. 1MHz Switching Frequency POL with a Fast Response Digital Controller. IEEE Applied Power Electronics Conference and Exposition, pp. 447-454. Available at: http://ieeexplore.ieee.org/ [Accessed December 2016]

Jang, J.S.R., 1993. ANFIS: Adaptive-Network-Based Fuzzy Inference System. IEEE Transactions on Systems, Man and Cybernetics, 23(3), pp. 665-685. Available at: http://ieeexplore.ieee.org/ [Accessed November 2013]

Jia, K., Chen, Y., Bi, T., Lin, Y., Thomas, D. and Sumner, M., 2017. Historical-Data-Based Energy Management in a Microgrid with a Hybrid Energy Storage System. IEEE Transactions on Industrial Informatics, 13(5), pp. 2597-2605, Available At: http://ieeexplore.ieee.org/ [Accessed January 2018]

Karaca, H. and Akkaya, R. 2012. Modeling, Simulation and Analysis of Matrix Converter Using Matlab & Simulink. International Journal of Modeling and Optimization, 2(3), pp. 328-332. Available at: http://ijmo.org/papers/137-C135.pdf [Accessed July 2017]

Kennedy, J. and Eberhart, R., 1995. Particle Swarm Optimization. International Conference on Neural Networks, pp.1942-1948. Available at: http://ieeexplore.ieee.org/ org [Accessed July 2013]

Khan, T.A., Taj, T.A., Asif, M.K. and Ijaz, I., 2012. Modeling of a standard Particle Swarm Optimization algorithm in MATLAB by different benchmarks. Second International Conference on Innovative Computing Technology, pp. 271-274. Available at: http://ieeexplore.ieee.org/ [Accessed October 2013]

Khanaki, R., Radzi, M.A.M. and Marhaban, M.H., 2013. Comparison of ANN and P&O MPPT Methods for PV Applications under Changing Solar Irradiation. IEEE Conference on Clean Energy and Technology, pp. 287-292, Available at: http://ieeexplore.ieee.org/ [Accessed November 2014]

Khayat, Y., Naderi, M., Shafiee, Q., Batmani, Y., Fathi, M., and Beyrani, H., 2017. Robust Control of a DC-DC Boost Converter: H2 and H-Infinity Techniques. 8th Power Electronics, Drive Systems and Technologies Conference. Available at: http://ieeexplore.ieee.org/ [Accessed September 2018]

Khosrojerdi, F., Taheri, S., and Cretu, A.M., 2016. An Adaptive Neuro-Fuzzy Inference System-Based MPPT Controller for Photvoltaic Arrays. IEEE Electrical Power and Energy Conference, pp. 1-6. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Krug, M., Nuber, F., and Bretthauer, G., 2015. Variable Frequency Digital PWM Controller for Low Power Buck Converters. IEEE International Conference on Industrial Technology. Available at: http://ieeexplore.ieee.org/ [Accessed November 2016]

Kumaravel, S. and Ashok, S., 2011. Adapted multilayer feedforward ANN based power management control of solar photovoltaic and wind integrated power system. IEEE PES Innovative Smart Grid Technologies, pp. 223-228. Available at: http://ieeexplore.ieee.org/ [Accessed December 2012]

Kumarawadu, S., Amaratunga, D.S., Piyasinghe, L.P., Prasanga, W.M.B. and Wijeratne, D.S., 2006. Intelligent Controller (Adaptive Fuzzy) for High Performance Power Electronic Converters. 2006 International Conference on Information and Automation, pp. 33-38. Available at: http://ieeexplore.ieee.org/ [Accessed February 2013]

Kurokawa, F., Maruta, H., and Motomura, M. 2012. A Novel Neural Network Based Control Method with Adaptive On-Line Training for DC-DC Converters. International Conference on Machine Learning and Applications (ICMLA), pp. 503-508. Available at: http://ieeexplore.ieee.org/ [Accessed March 2013]

Lakshmi, A.V., Rajan, S.E. and Vengatesh, R.P., 2013. Performance evaluation of a magnetically coupled DC -DC converter for photovoltaic energy systems. International Conference on Circuits, Power and Computing Technologies, pp. 368-275. Available at: http://ieeexplore.ieee.org/ [Accessed November 2013]

Lee, H.H., Dzung, P.Q., Phuong, L.M., Khoa, L. D., and Vu, N.T.D., 2010. The Space Vector PWM for Voltage Source Inverters using Artifical Neural Networks Based on FPGA. International Forum on Strategic Technology, pp. 396-401. Available at: http://ieeexplore.ieee.org/ [Accessed May 2013]

Li, B., Guo, S., Lin-Shi, X., and Allard, B., 2011. Design and Implementation of the Digital Controller for Boost Converter Based on FPGA. IEEE Internation Symposium on Industrial Electronics. Available at: http://ieeexplore.ieee.org/ [Accessed November 2016]

Li, J., Qiu, Y., Sun, Y., Huang, B, Xu, M., Ha, D.S. and Lee, F.C., 2007. High Resolution Digital Duty Cycle Modulation Schemes for Voltage Regulators. IEEE Applied Power Electronics Conference, pp. 871-876. Available at: http://ieeexplore.ieee.org/ [Accessed November 2016]

Li, X., Chen, M. and Tsutomu, Y., 2013. A method of searching PID controller's optimized coefficients for Buck converter using particle swarm optimization. IEEE Electric Power and Energy Conference, pp. 238-243. Available at: http://ieeexplore.ieee.org/ [Accessed July 2013]

Li, Y., Nelson, B. and Wirthlin, M., 2010. Synchronization Techniques for Crossing Multiple Clock Domains in FPGA-Based TMR Circuits. IEEE Transactions on Nuclear Science, 57(6), pp. 3506-3514.Available at: http://ieeexplore.ieee.org/ [Accessed September 2017]

Lin, T. J., Zhang, W. and Jha, N.K., 2014. A Fine-Grain Dynamically Reconfigurable Architecture Aimed at Reducing the FPGA-ASIC Gaps. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 22(12), pp. 2607-2620. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Lin, W.M. and Hong, C.M., 2011. A New Elman Neural Network-Based Control Algorithm for Adjustable-Pitch Variable-Speed Wind-Energy Conversion Systems. IEEE Transactions on Power Electronics, 26(2), pp. 473-481. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Lin, W.M., Hong, C.M. and Chen, C.H., 2011. Neural Network Based Control of a Stand-Alone Hybrid Power Generation System. IEEE Transactions on Power Electronics, 26(12), pp. 3571-3581. Available at: http://ieeexplore.ieee.org/ [Accessed February 2013]

Lisy, S.M., Sonnenberg, B. J., and Dolan, J., 2014. Case Study of Deployment of 400V DC Power with 400V/-48VDC Conversion. IEEE International Telecommunications Energy Conference. Available At: http://ieeexplore.ieee.org/ [Accessed September 2018]

Liu, F., Duan, S., Liu,B. and Kang, Y., 2008. A Variable Step-Size INC MPPT Method for PV Systems. IEEE Transactions on Industrial Electronics, 55(7), pp. 2622-2628. Available at: http://ieeexplore.ieee.org/ [Accessed December 2014]

Lukic, Z., Rahman, N., and Prodie, A., 2007. Multibit Σ-Δ PWM Digital Controller IC for DC-DC Converters Operating at Switching Frequencies Beyond 10MHz. IEEE Transactions on Power Electronics, 22(5), pp. 1693-1707.Available at: http://ieeexplore.ieee.org/ [Accessed November 2016]

Mahmud, N., Zahedi, A. and Mahmud, A., 2017. A Cooperative Operation of Novel PV Inverter Control Scheme and Storage Energy Management System Based ANFIS for Voltage Regulation of Grid-Tied PV System. IEEE Transactions on Industrial Informatics, 13(5), pp. 2657-2668.Available at: http://ieeexplore.ieee.org/ [Accessed January 2018]

Mane, S., Mejari, M., Kazi, F. and Singh, N., 2017. Improving Lifetime of Fuel Cell in Hybrid Energy Management System by Lure-Lyapunov-Based Control formulation. IEEE Transactions on Industrial Electronics, 64(8), pp. 6671-6679. Available At: http://ieeexplore.ieee.org/ [Accessed November 2017]

Maxwell, J.C., 1868. On Governors. Proceedings of the Royal Society of London, Vol 16. Available at: http://www.jstor.org/stable/112510?seq=2 [Accessed March 2013]

Mekhilef, S. and Kadir, M.N.A., 2013. Novel Voltage Control of 18 Level Multilevel Inverter. 9th Asian Control Conference, pp. 1-6. Available at: http://ieeexplore.ieee.org/ [Accessed December 2013]

Mendis N., Muttaqi, K.M. and Perera, S., 2014. Management of Battery-Supercapacitor Hybrid Energy Storage and Synchronous Condenser for Isolated Operation of PMSG Based Variable-Speed Wind Turbine Generating Systems. IEEE Transactions on Smart Grid, 5(2), pp. 944-953. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Miller, J., McCleer, P.J. and Cohen, M., 2003. Ultracapacitors as Energy Buffers in a Multiple Zone Electrical Distribution System. [pdf] Maxwell Technologies. Available at: www.sciencemadness.org/talk/files.php?pid=162431&aid=8647 [Accessed April 2014]

Moré, J.J., Puleston, P.F., Kunusch, C. and Fantova, M.A., 2015. Development and Implementation of a Supervisor Strategy and Sliding Mode Control Setup for Fuel-Cell-Based Hybrid Generation Systems. IEEE Transactions on Energy Conversion, 30(1), pp. 218-225. Available at: http://ieeexplore.ieee.org/ [Accessed January 2018]

Mori, S., Aketa, M., Sakaguchi, T., Nanen, Y., Asahara, H., Nakamura, T. and Kimoto, T., 2017. High-Temperature Characteristics of 3-kV 4H-SiC Reverse Blocking MOSFET for High-Performance Bidirectional Switch. IEEE Transactions on Electron Devices, 64(10), pp. 4167-4174, Available At: http://ieeexplore.ieee.org/ [Accessed November 2017]

Nasser, M., Vergnol, A., Sprooten, J. and Robyns, B., 2009. A Global Supervision for Wind/Hydro Power Plant and Storage System Connected to AC Grid. 13th European Conference on Power Electronics and Applications, pp. 1-10. Available at: http://ieeexplore.ieee.org/ [Accessed March 2013]

Nehrir, M. H., Wang, C., Strunz, K., Aki, H., Ramakumar, R., Bing, J., Miao, Z. and Salameh, Z., 2011. A Review of Hybrid Renewable/Alternative Energy Systems for Electric Power Generation: Con figurations, Control, and Applications. IEEE Transactions on Sustainable Energy, 2(4), pp. 392-403. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Nurvitadhi, E., Sheffield, D., Sim, J., Mishra, A., Venkatesh, G. and Marr, D., 2016. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU and ASIC. International Conference on Field-Programmable Technology. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Othman, M.M., Abdelaziz, A.Y., Hegazi, Y.G. and El-Khattam, W., 2015. Approach for Modelling Stochastically Dependent Renewable Energy-Based Generators using Diagonal Band Copula. IET Renewable Power Generation, 9(7), pp. 809-820. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Pande, P.R., Paikrao, P.L. and Chaudhari, D.S., 2013. Digital ANFIS Model Design. International Journal of Soft Computing (IJSCE), pp. 314-318, Available at: http://citeseerx.ist.psu.edu [Accessed November 2016]

Papaefthymiou, S.V., Karamanou, E.G., Papathanassiou, S.A. and Papadopoulos, M.P., 2010. A Wind-Hydro-Pumped Storage Station Leading to High RES Penetration in the Autonomous Island System of Ikaria. IEEE Transactions on Sustainable Energy, 1(3), pp. 163-172. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Park, J.B., Jeong, Y.W., Shin, J.R. and Lee, K.Y., 2010. An Improved Particle Swarm Optimization for Nonconvex Economic Dispatch Problems. IEEE Transactions on Power Systems, 25(1), pp. 156-166. Available at: http://ieeexplore.ieee.org/ [Accessed October 2013]

Patella, J., Prodic, A., Zirger, A. and Maksimović, D., 2003. High Frequency Digital PWM Controller IC for DC-DC Converters. IEEE Transactions on Power Electronics, 18(1), pp. 438 - 446. Available at: http://ieeexplore.ieee.org/ [Accessed November 2016]

Perqueroles-Queralt, J., Bianchi, F.D. and Gomis-Bellmunt,O., 2014. A Power Smoothing System Based on Supercapacitors for Renewable Distributed Generation. IEEE Transactions on Industrial Electronics, 62(1), pp. 343-350. Available at: http://ieeexplore.ieee.org/ [Accessed April 2015]

Piao, C., Sun, Z., Liang, Z. and Cho, C., 2010. SOC Estimation of Lead Acid Batteries Based on UKF. International Conference on Electrical and Control Engineering, pp. 1968 - 1972. Available at: http://ieeexplore.ieee.org/ [Accessed January 2017]

Pickard, J. and Campbell, P., 2017. UK Plans to Ban Sale of New Petrol and Diesel Cars by 2040. Financial Times, [online] July 26 2017. Available at: https://www.ft.com/content/7e61d3ae-718e-11e7-93ff-99f383b09ff9 [Accessed July 2017]

Powersim Inc., 2017. PSim User's Guide. [pdf] Available at: https://powersimtech.com/drive/uploads/2017/11/PSIM-User-Manual.pdf [Accessed June 2018]Quach, D., Yin, Q., Shi, Y. and Zhou, C. 2012. Design and implementation of three-phase SVPWM inverter with 16-bit dsPIC. 12th International Conference on Control Automation Robotics & Vision, pp. 1181-1186. Available at: http://ieeexplore.ieee.org/ [Accessed May 2013]

Rabbani, M., Maruf, H.M.M., Ahmed, T., Kabir, M.A. and Mahbub, U., 2012. Fuzzy Logic Driven Adaptive PID Controller for PWM Based Buck Converter. International Conference on Informatics, Electronics & Vision (ICIEV), pp. 958-962. Available at: http://ieeexplore.ieee.org/ [Accessed February 2013]

Ramya, S., Napolean, A. and Manoharan, T., 2013. A novel converter topology for stand-alone hybrid PV/Wind/battery power system using Matlab/Simulink. International Conference on Power, Energy and Control, pp. 17-22. Available at: http://ieeexplore.ieee.org/ [Accessed November 2013]

Ratnaweera, A., Halgamuge, S. and Watson, H.C., 2004. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Transactions on Evolutionary Computation, 8(3), pp. 240-255. Available at: http://ieeexplore.ieee.org/ [Accessed October 2013]

Ren, Y.F. and Bao, G.Q., 2010. Control Strategy of Maximum Wind Energy Capture of Direct-Drive Wind Turbine Generator Based on Neural-Network. Asia-Pacific Power and

Energy Engineering Conference, pp. 1-4. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Ren21, 2017. Renewables 2017 Global Status Report [pdf]. Available at: http://www.ren21.net/wp-content/uploads/2017/06/170607_GSR_2017_Full_Report.pdf [Accessed August 2017]

Rezkallah, M., Hamadi, A., Chandra, A. and Singh, B., 2017. Design and Implementation of Active Power Control with Improved P&O Method for Wind-PV-Battery Based Standalone Generation System. IEEE Transactions on Industrial Electronics, PP(99), pp. 1. Available at: http://ieeexplore.ieee.org/ [Accessed November 2017]

Rivera, M., Rodriguez, J., Wheeler, P.W., Rojas, C.A., Wilson, A. and Espinoza, J.R. 2012. Control of a Matrix Converter With Imposed Sinusoidal Source Currents. IEEE Transactions on Industrial Electronics, 59(4), pp. 1939-1949. Available at: http://ieeexplore.ieee.org/ [Accessed May 2013]

Rodrigues, D.B., Costa, A.V., Brito Lima, G. and de Freitas, L.C., 2013. DSP-Based Implementation of Control Strategy for Sinusoidal Input Line Current Imposition for a Hybrid Three-Phase Rectifier. IEEE Transactions on Industrial Informatics, 9(4), pp. 1947-1963. Available at: http://ieeexplore.ieee.org/ [Accessed December 2013]

Rogriguez, M., Stahl, G., Corradini, L. and Maksimovic, D., 2013. Smart DC Power Management System Based on Software-Configurable Power Modules. IEEE Transactions on Power Electronics, 28(4), pp. 1571-1586. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Rouzbehi, K., Miranian, A., Luna, A. and Rodriguez, P., 2012. Identification and maximum power point tracking of photovoltaic generation by a local neuro-fuzzy model. IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society, pp. 1019-1024. Available at: http://ieeexplore.ieee.org/ [Accessed April 2013]

Safari, M. and Sarvi, M., 2014. Optimal Load Sharing Strategy for a Wind/Diesel/Battery Hybrid Power System Based on Imperialist Competitive Neural Network Algorithm. IET Renewable Power Generation, 8(8), pp. 937-946. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Sahin, M.E. and Okumus, H.I., 2011. Fuzzy Logic Controlled Buck-Boost DC-DC Converter for Solar Energy-Battery System. International Symposium on Innovations in Intelligent Systems and Applications (INISTA), pp. 394-297. Available at: http://ieeexplore.ieee.org/ [Accessed February 2013]

Saldana, H. J. S. and Silva-Cardenas, C., 2012. A Digital Hardware Architecture for a Three-Input One-Output Zero-Order ANFIS. Third Latin American Symposium on Circuits and Systems, pp. 1-4, Available at: http://ieeexplore.ieee.org/ [Accessed November 2016]

Saqib, M.A.; Kashif, S.A.R., 2010. Artificial neural network based space vector PWM for a five-level diode-clamped inverter. 20th Australasian Universities Power Engineering Conference. pp. 1 -6. Available at: http://ieeexplore.ieee.org/ [Accessed May 2013]

Sarvi, M. and NamazyPour, N., 2008. A New PID-Fuzzy Controller for DC/DC Converters. 43rd International Universities Power Engineering Conference, pp. 1-4. Available at: http://ieeexplore.ieee.org/ [Accessed April 2013]

Schmidt, O. R. and Myhre, E., 2015. 380VDC/48VDC/3 kw DC/DC Converter with 98.2% Efficiency. IEEE International Telecommunications Energy Conference. Available At: http://ieeexplore.ieee.org/ [Accessed September 2018]

Sekhar, P.C. and Mishra, S., 2016. Storage Free Smart Energy Management for Frequency Control in a Diesel –PV-Fuel Cell Based Hybrid AC Microgird. IEEE Transactions on Neural Networks and Learning Systems, 27(8), pp. 1657-1671. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Sellami, A., Kandoussi, K., El Otmani, R., Eljouad, M., Hajjaji, A. and Lakrami, F., 2016. Improvement of Perturb and Observe Method for PV Array under Partial Shading Conditions. International Renewable and Sustainable Energy Conference. Available at: http://ieeexplore.ieee.org/ [Accessed January 2018]

Sharaf, A.M. and El-Gammal, A.A., 2010. A Novel Efficient PSO-Self Regulating PID Controller for Hybrid PV-FC-Diesel-Battery Micro Grid Scheme for Village/Resort Electricity Utilization. IEEE Electric Power and Energy Conference, pp. 1-6. Available at: http://ieeexplore.ieee.org/ [Accessed July 2013]

Sharma, R.K. and Mishra, S., 2017. Dynamic Power Management and Control of a PV PEM Fuel-Cell-Based Standalone AC/DC Microgrid using Hybrid Energy Storage. IEEE Transactions on Industry Applications, 54(1), pp. 526-538. Available at: http://ieeexplore.ieee.org/ [Accessed August 2018]

Shen, J. and Khaligh, A., 2015. A Supervisory Energy Management Control Strategy in a Battery/Ultracapacitor Hybrid Energy Storage System. IEEE Transactions on Transportation Electrification, 1(3), pp. 223-231. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Shen, J. and Khaligh, A., 2017. Design and Real-Time Controller Implementation for a Battery-Ultracapacitor Hybrid Energy Storage System. IEEE Transactions on Industrial Informatics, 12(5), pp. 1910-1918, Available At: http://ieeexplore.ieee.org/ [Accessed January 2018]

Sheraz, M. and Abido, M.A., 2012. An Efficient MPPT Controller using Differential Evolution and Neural Network. 16th IEEE International Conference on Power and Energy, pp. 378-282. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Sikkabut, S., Mungporn, P., Ekkaravarodome, C., Bizon, N., Tricoli, P., Nahid-Mobarakeh, B., Pierfederici, S., Davat, B. and Thounthong, P., 2016. Control of High-Energy High-Power Densities Storage Devices by Li-Ion Battery and Supercapacitor for Fuel Cell/Photovoltaic Hybrid Power Plant for Autonomous System Applications. IEEE Transactions on Industry Applications, 52(5), pp. 4395-4407. Available at: http://ieeexplore.ieee.org/ [Accessed September 2017]

Somayajula, D. and Crow, M. L., 2014. An Ultracapacitor Integrated Power Conditioner for Intermittency Smoothing and Improving Power Quality of Distribution Grid. IEEE

Transactions on Sustainable Energy, 5(4), pp. 1145-1155. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Suh, J., Yoon, D.H., Cho, Y.S. and Jang, G., 2016. Flexible Frequency Operation Strategy of Power System With High Renewable Penetration. IEEE Transactions on Sustainable Energy, 8(1), pp. 192-199. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Sunny, M.S.H., Ahmed, A.N.R. and Hasan, M.K., 2016. Design and Simulation of Maximum Power Point Tracking of Photovoltaic System using ANN. International Conference on Electrical Engineering and Information Communication Technology. Available at: http://ieeexplore.ieee.org/ [Accessed September 2017]

Thounthong, P., Luksanasakul, A., Koseeyaporn, P. and Davat, B., 2013. Intelligent Model-Based Control of a Standalone Photovoltaic/Fuel Cell Power Plant With Supercapacitor Energy Storage. IEEE Transactions on Sustainable Energy, 4(1), pp. 240-249. Available at: http://ieeexplore.ieee.org/ [Accessed April 2013]

Trescases, O., Wei, G. and Ng, W.T., 2005. A Segmented Digital Pulse Width Modulator with Self-Calibration for Low Power SMPS. IEEE Conference on Electron Devices and Solid State Circuits. Available at: http://ieeexplore.ieee.org/ [Accessed November 2016]

Tsai,M.F., Tseng,C.S., Hong,G.D. and Lin,S.H., 2012. A Novel MPPT Control Design for PV Modules using Neural Network Compensator. International Symposium on Industrial Electronics, pp. 1742-1747. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Usui, L. H., Babasaki, T., Hirose, K. and Yoshida, Y., 2017. Dual-Voltage Output Power Supply System Toward Parallel Use of 380VDC and 48VDC. IEEE International Telecommunications Energy Conference. Available At: http://ieeexplore.ieee.org/ [Accessed September 2018]

Utomo, W.M., Bakar, A., Ahmad, M., Taufik, T. and Heriansyah, R., 2011. Online Learning Neural Network Control of Buck-Boost Converter. Eighth International Conference on Information Technology: New Generations, pp. 485-489. Available at: http://ieeexplore.ieee.org/ [Accessed April 2013]

Wald, M. 2013. New Solar Process Gets More Out of Natural Gas. New York Times [online], 10 April 2013. Available at: http://www.nytimes.com [Accessed August 2017]

Wai, R.J., Lee, J.D. and Chuang, K.L., 2011. Real-Time PID Control Strategy for Maglev Transportation System via Particle Swarm Optimization. IEEE Transactions on Industrial Electronics, 58(2). Available at: http://ieeexplore.ieee.org/ [Accessed July 2018]

Wang, A., Huang, Z., Lei, C., Chao, Z. and Wu, E., 2014. Parameters Optimization Study and Analysis of PID Controller in Buck Converter Based on Fuzzy Particle Swarm Optimization Algorithm. International Conference on Information Science, Electronics and Electrical Engineering, pp. 1562-1566. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Wang, G., Yang,R. and Xu, D. 2013. DSP-Based Control of Sensorless IPMSM Drives for Wide-Speed-Range Operation. IEEE Transactions on Industrial Electronics,60(2) pp. 720-727. Available at: http://ieeexplore.ieee.org/ [Accessed December 2013]

Weddell, A.S., Merrett, G.V., Kazmierski, T.J. and Al-Hashimi,B.M., 2011. Accurate Supercapacitor Modeling for Energy Harvesting Wireless Sensor Nodes. IEEE Transactions on Circuits and Systems 2: Express Briefs, 58(1), pp. 911-915. Available at: http://ieeexplore.ieee.org/ [Accessed April 2015]

Wiser, R. and Bolinger, M. 2013. 2012 Wind Technologies Market Report. [pdf] Lawrence Berkley National Laboratory, Berkley, CA. Available at: https://www1.eere.energy.gov/wind/pdfs/2012_wind_technologies_market_report.pdf [Accessed August 2017]

Wu, Y., Zhang, B., Lu, J. and Du, K.L., 2011. Fuzzy Logic and Neuro-Fuzzy Systems: A Systematic Introduction. International Journal of Artificial Intelligence and Expert Systems (IJAE), 2 (2), pp. 47-80. Available at: http://www.researchgate.net [Accessed March 2013]

Xilinx, 2011. AXI Reference Guide. [online] Available at: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf [Accessed March 2017]

Xilinx, 2013. Zynq-7000 All Programmable SoC Overview [pdf] Available at: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf [Accessed December 2013]

Xu, J., Shen, A., Yang, C., Rao, W. and Yang, X., 2011. ANN Based on IncCond Algorithm for MPP Tracker. Sixth International Conference on Bio-Inspired Computing: Theories and Applications, pp.129-134. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Xu, Z., Yang, P., Song, S., Zhang, Y., Peng, J. and Zheng,Q., 2016. Wind/PV/Battery micro-grid hybrid simulation research on operation control based on RTDS and controller. IEEE Innovative Smart Grid Technologies – Asia, pp. 902-907. Available at: http://ieeexplore.ieee.org/ [Accessed January 2017]

Yang, D., Yang, M. and Ruan, X., 2012. One Cycle Control for a Double Input DC/DC Converter. IEEE Transactions on Power Electronics, 27(11), pp. 4646-4655. Available at: http://ieeexplore.ieee.org/ [Accessed June 2013]

Yang, Z., Duan, Q., Zhong, J., Mao, M. and Xun, Z., 2017. Improvement of Perturb and Observe Method for PV Array under Partial Shading Conditions. IEEE Chinese Control and Decision Conference, pp. 549-553. Available at: http://ieeexplore.ieee.org/ [Accessed January 2018]

Yousefi, M.R., Poudeh, M.B. and Eshtehardiha, S., 2008. Improvement performance of step-down converter through intelligent controllers. International IEEE Conference on Intelligent Systems, pp. 20-24. Available at: http://ieeexplore.ieee.org/ [Accessed July 2013]

Yusoff, S., De Lillo, L., Zanchetta, P. and Wheeler, P. 2012. Predictive Control of a direct AC/AC matrix converter power supply under non-linear load conditions. 15th International Power Electronics and Motion Control Conference. Available at: http://ieeexplore.ieee.org/ [Accessed May 2013]

Zerkaoui, S., Hajjaji, A.E. and Bosche, J., 2012. On-line control strategy for instantaneous power management of hybrid power system based on dynamic fuzzy logic controller. IEEE

Conference on Industrial Electronics and Applications, pp. 1130-1136. Available at: http://ieeexplore.ieee.org/ [Accessed March 2013]

Zhan, Z.H., Zhang, J., Li , L. and Chung, H.S.H., 2009. Adaptive Particle Swarm Optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 39(6), pp. 1362-1381. Available at: http://ieeexplore.ieee.org/ [Accessed October 2013]

Zhang, X. and Maksimovic, D., 2010. Multimode Digital Controller for Synchronous Buck Converters Operating Over Wide Ranges of Input Voltages and Load Currents. IEEE Transactions on Power Electronics, 25(8), pp. 1958-1965. Available at: http://ieeexplore.ieee.org/ [Accessed May 2013]

Zhang, Y., Qin, W., Wang, P., Han, X., Wang, J. and Lei, D., 2017. Hierarchical Control Strategy of Hybrid Energy Storage System in Bipolar Type DC Micro Grid. IEEE Future Energy Electronics Conference, pp. 2223-2227. Available at: http://ieeexplore.ieee.org/ [Accessed August 2017]

Zhang, Y., Bezawada, Y., Fu, R., Tian, W. and Winter, R. M., 2017. Study of a 3kW High-Efficient Wide-Bandgap DC-DC Power Converter for Solar Power Integration in 400V DC Distribution Networks. IEEE Conference on Power Electronics and Drive Systems. Available At: http://ieeexplore.ieee.org/ [Accessed September 2018]

Zhou, Y. and Hu, S., 2011. H Infinity Control for DC Servo Motor in the Network Environment. IEEE Advanced Information Technology, Electronic and Automation Control Conference. Available at: http://ieeexplore.ieee.org/ [Accessed September 2018]

Zhu, Z., Wang, L. and Duan, S., 2015. Memristor-Based Neural Netowrk PID Controller for Buck Converter. International Conference on Intelligent Control and Information Processing, pp.36-41. Available at: http://ieeexplore.ieee.org/ [Accessed February 2013]

Appendix A – PSO Algorithm Script for Matlab

```
function [gain, resettime, bfit] = PSO_PID(pmax, imin, current, cur_step)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Initialise the variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%iteration parameters
PAR = 50;
MAX_ITER = 50;

%maximums
rmax = 1e-6;
RMAX_FINAL = 1e-4;

%inertia weight
w = 0.9;

%Personal best and global best variables
Pb = zeros(1, PAR);
Pb_pos = zeros(2, 1, PAR);
Gb = inf;
Gb_pos = zeros(2, 1);

%set the values of acceleration coefficients
C1 = 0.5;
C2 = 2.5;

%fill the vel and pos vectors
vel = zeros(2, PAR);
pos = zeros(2, PAR);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Create the random particles
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%initial random particle positions
posp = randi([0 pmax], 1, PAR);
postr = rand(1, PAR);
postr = postr * rmax;

%initial random velocities
velp = randn(1, PAR) * (pmax / 2);
veltr = randn(1, PAR) * (rmax / 2);

%increase the spread of veltr and veltd
for j = 1:PAR
   q = rand;
   q2 = rand;
   if q < 0.33
      postr(1, j) = postr(1, j) / 1000;
   elseif q > 0.66
      postr(1, j) = postr(1, j) * 1000;
```

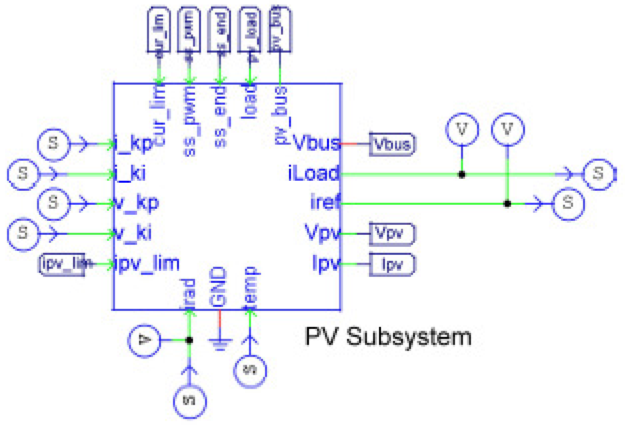
```
    end;
    if q2 < 0.33
        veltr(1, j) = veltr(1, j) / 1000;
    elseif q2 > 0.66
        veltr(1, j) = veltr(1, j) * 1000;
    end;
end;

clear q;
clear q2;

%increase maximums to final value
rmax = RMAX_FINAL;

%correct for out of bounds
for j = 1:PAR
    if velp(1, j) > pmax
        velp(1, j) = pmax;
    elseif velp < (-1 * pmax)
        velp(1, j) = -pmax;
    end;
    if posp(1, j) > pmax
        posp(1, j) = pmax;
    elseif posp < 1e-9
        posp(1, j)= 1e-9;
    end;
    if veltr(1, j) > rmax
        veltr(1, j) = rmax;
    elseif veltr(1, j) < -rmax
        veltr(1, j) = -rmax;
    end;
    if postr(1, j) > rmax
        postr(1, j) = rmax;
    elseif postr(1, j) < imin
        postr(1, j) = imin;
    end;
end;

%create the particles
for j = 1:PAR
    pos(1, j) = posp(1, j);
    vel(1, j) = velp(1, j);
    pos(2, j) = postr(1, j);
    vel(2, j) = veltr(1, j);
end;

% free up some memory
clear veltr;
clear postr;
clear velp;
clear posp;

%calculate the fitness
for j = 1:PAR
```

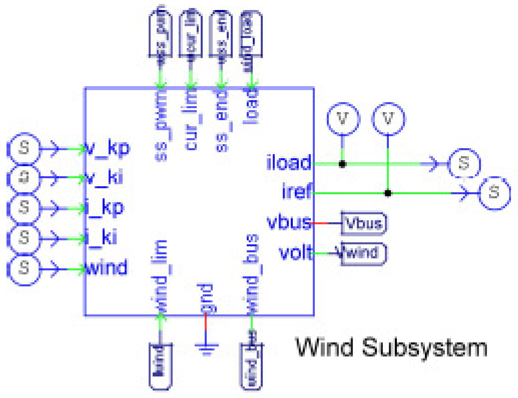
```
    %%%%%%%%%%%%%%%%%%%%%%%%
    %simulate the system and calculate fitness
    %%%%%%%%%%%%%%%%%%%%%%%%
    kp = pos(1, j);
    tr = pos(2, j);

    if current > 0
        if cur_step > 0
            sim('pvboostcur', 0.6e-3, [], [1, kp, tr, cur_step]);
        else
            sim('steadycur', 0.6e-3, [], [1, kp, tr]);
        end
    else
        if cur_step > 0
            sim('pvboostv', 1e-3, [], [1, kp, tr, cur_step]);
        else
            sim('steadyv', 1e-3, [], [1, kp, tr]);
        end
    end

    fit = yout(end);

    % adjust out false positives
    if fit < 10e-6
        fit = 1e6;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Check for global and personal best
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if fit < Gb
        Gb = fit;
        Gb_pos = [kp , tr];
        Pb(1,j) = fit;
        Pb_pos(1, 1, j) = kp;
        Pb_pos(2, 1, j) = tr;
    else
        Pb(1, j) = fit;
        Pb_pos(1, 1, j) = kp;
        Pb_pos(2, 1, j) = tr;
    end
    fprintf('Best fit %d \n', Gb);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Main Loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%
for iter = 1:MAX_ITER

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Update position
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i = 1:2
        for j = 1:PAR
            pos(i, j) = pos(i, j) + vel(i, j);
        % Correct for any out of bounds
        if i == 1
            if pos(i, j) > pmax
                pos(i, j) = pmax;
            elseif  pos(i, j) < 1e-9
                pos(i, j) = 1e-9;
            end
        else
            if pos(i, j) > rmax
                pos(i, j) = rmax;
            elseif  pos(i, j) < imin
                pos(i, j) = imin;
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%simulate the system and
%calculate fitness
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j = 1:PAR
    kp = pos(1, j);
    tr = pos(2, j);

    fprintf('iteration: %d \n', iter);

    if current > 0
        if cur_step > 0
            sim('pvboostcur', 0.6e-3, [], [1, kp, tr, cur_step]);
        else
            sim('steadycur', 0.6e-3, [], [1, kp, tr]);
        end
    else
        if cur_step > 0
            sim('pvboostv', 1e-3, [], [1, kp, tr, cur_step]);
        else
            sim('steadyv', 1e-3, [], [1, kp, tr]);
        end
    end

    fit = yout(end);

    % adjust out false positives
    if fit < 10e-6
        fit = 1e6;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Check for global and personal best
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if fit < Gb
        Gb = fit;
        Gb_pos = [kp , tr];
```

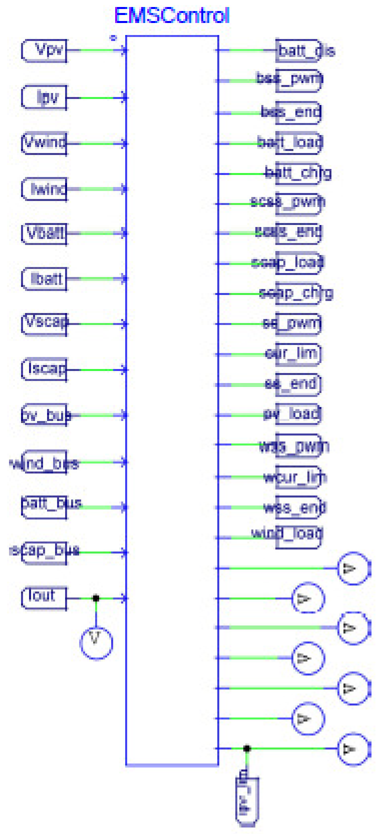
```
            Pb(1, j) = fit;
            Pb_pos(1, 1, j) = kp;
            Pb_pos(2, 1, j) = tr;
        elseif fit < Pb(1,j)
            Pb(1, j) = fit;
            Pb_pos(1, 1, j) = kp;
            Pb_pos(2, 1, j) = tr;
        end
        fprintf('Best fit %d \n', Gb);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Update the velocity
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i = 1:2
        for j = 1:PAR
            new_gb_pos = (C1 * rand) * (Gb_pos(1, i) - (pos(i, j)));
            new_pb_pos = (C2 * rand) * (Pb_pos(i, 1, j) - (pos(i, j)));

            vel(i, j) = (w * vel(i, j)) + new_gb_pos + new_pb_pos;

            %clamp the maximum velocities
            if i == 1
                if vel(i, j) > pmax
                    vel(i, j) = pmax;
                elseif vel(i, j) < -pmax
                    vel(i, j) = -pmax;
                end
            else
                if vel(i, j) > rmax
                    vel(i, j) = rmax;
                elseif vel(i, j) < -rmax
                    vel(i, j) = -rmax;
                end
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Update the inertia value
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    w = 0.9 - ((0.5 / MAX_ITER) * iter);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Update the C1 and C2
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    C1 = 0.5 + ((2 / MAX_ITER) * iter);
    C2 = 2.5 - ((2 / MAX_ITER) * iter);

end

gain = Gb_pos(1, 1);
resettime = Gb_pos(1, 2);
bfit = Gb;
```

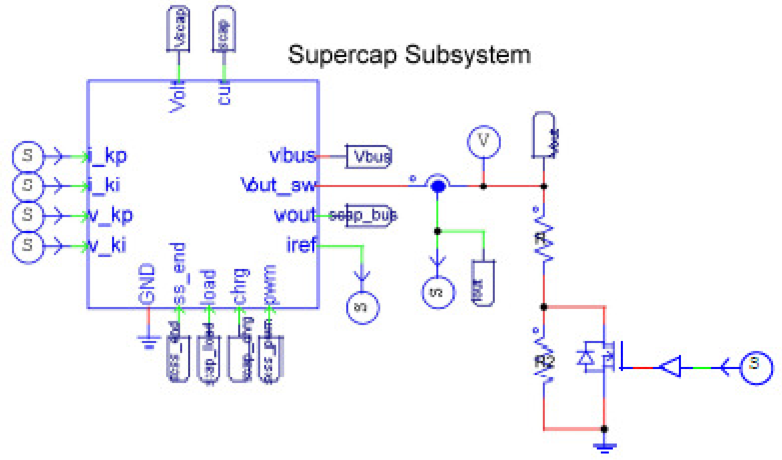
Appendix B – Psim Schematics for the Hybrid Renewable Energy System

Appendix B.1 – Top Level Schematic

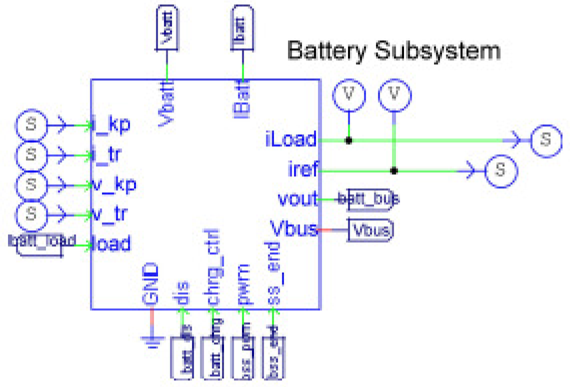






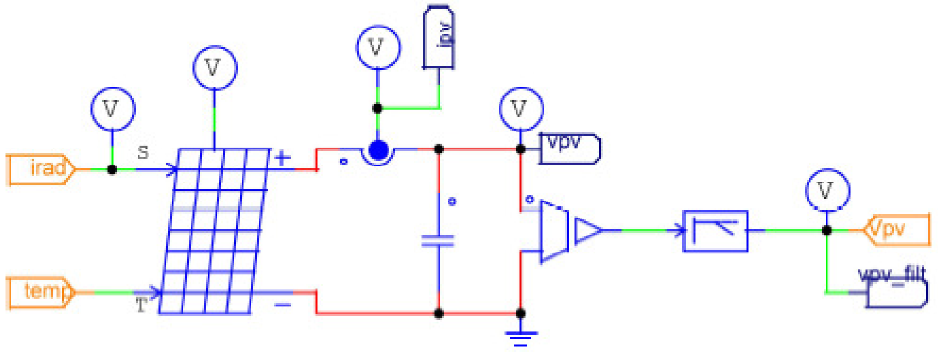


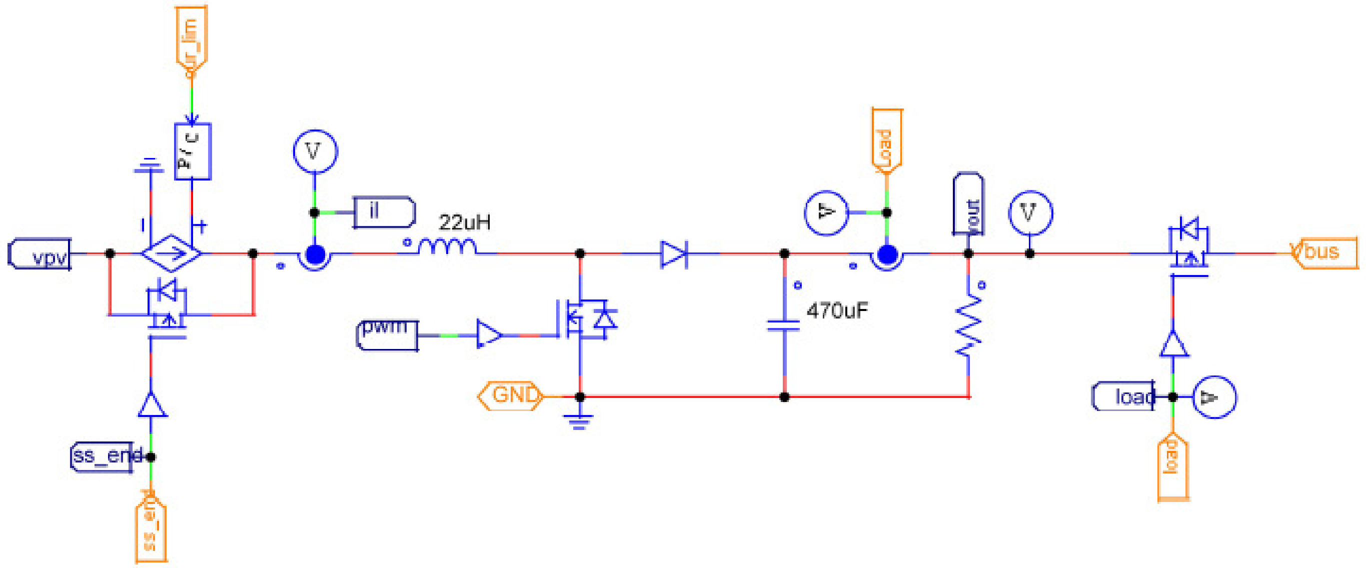
Supercap Subsystem
Vscap
Iscap
Volt
cur
i_kp
i_ki
v_kp
v_ki
vbus
Vout_sw
vout
iref
Vbus
scap_bus
GND
ss_end
load
chrg
pwm
V
S

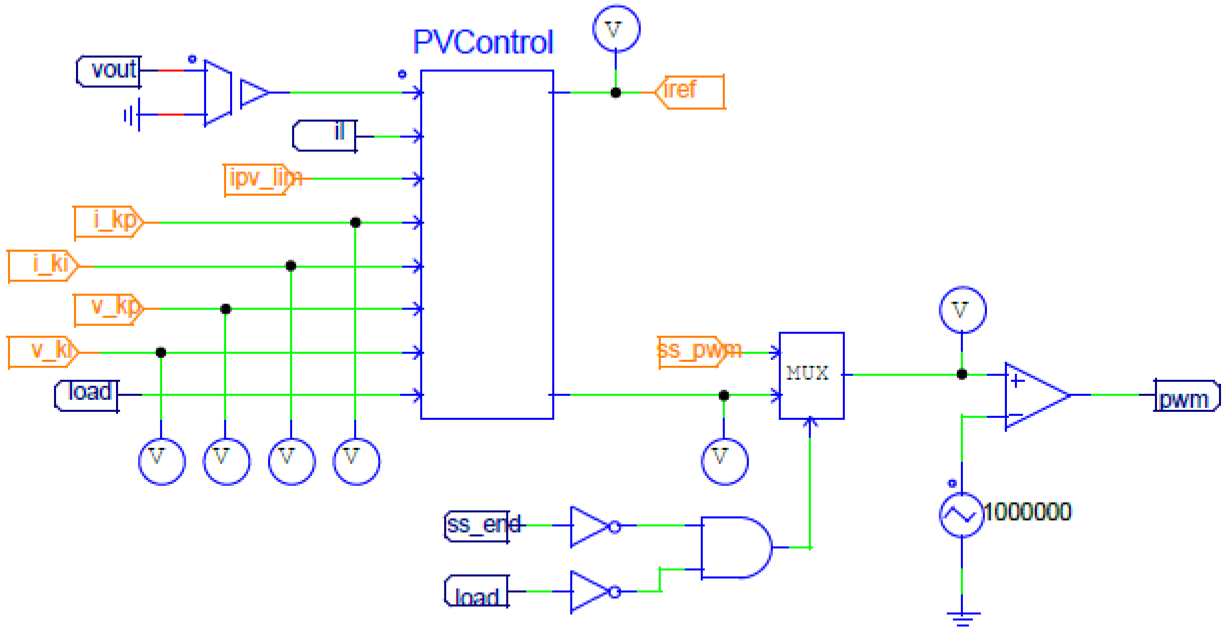


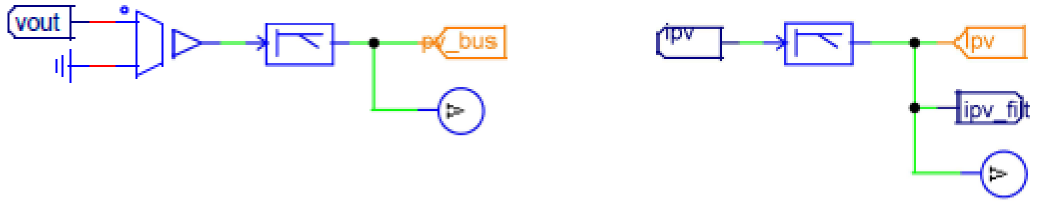
Battery Subsystem
Vbatt
Ibatt
VBatt
IBatt
i_kp
i_tr
v_kp
v_tr
load
Ibatt_load
iLoad
iref
vout
Vbus
batt_bus
GND
dis
chrg_ctrl
pwm
ss_end
batt_dis
batt_chrg
V
S

Appendix B.2 – Solar PV Subsystem Schematic

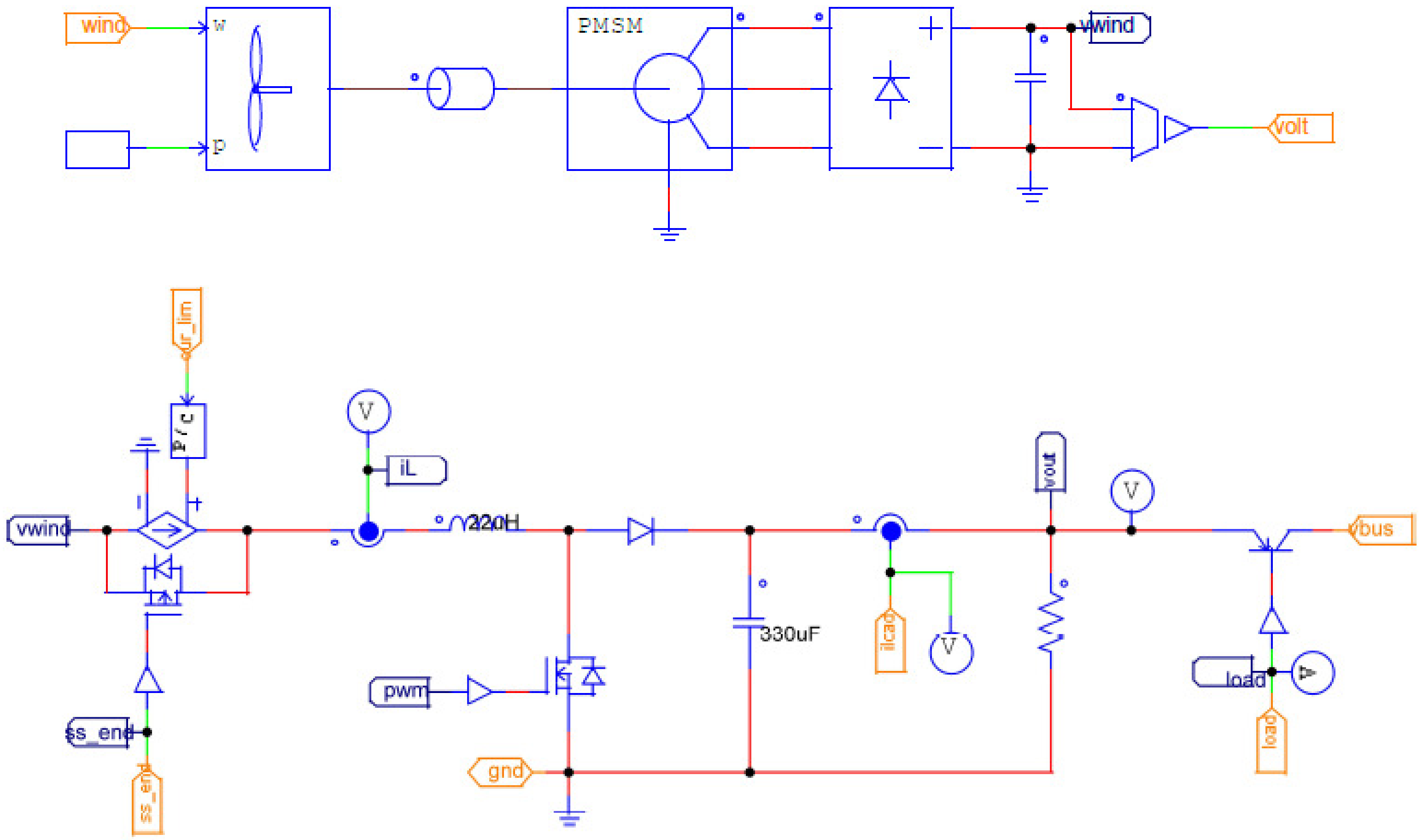


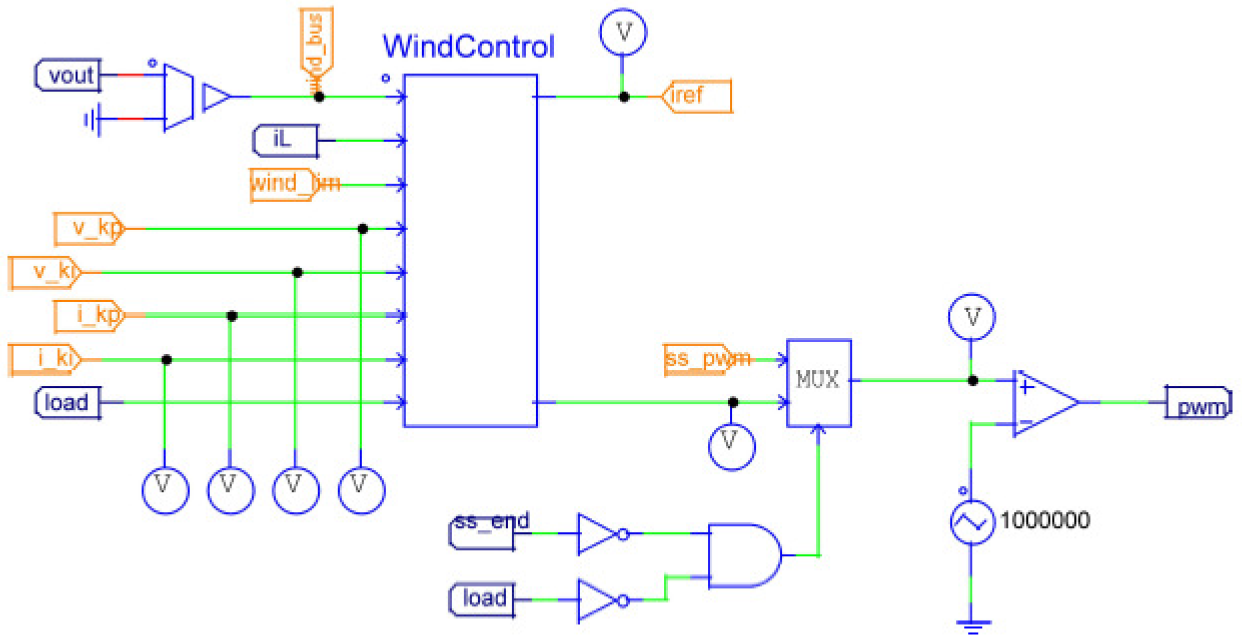



PVControl
vout
il
ipv_lim
i_kp
i_ki
v_kp
v_ki
load
iref
ss_pwm
MUX
pwm
ss_end
load
1000000

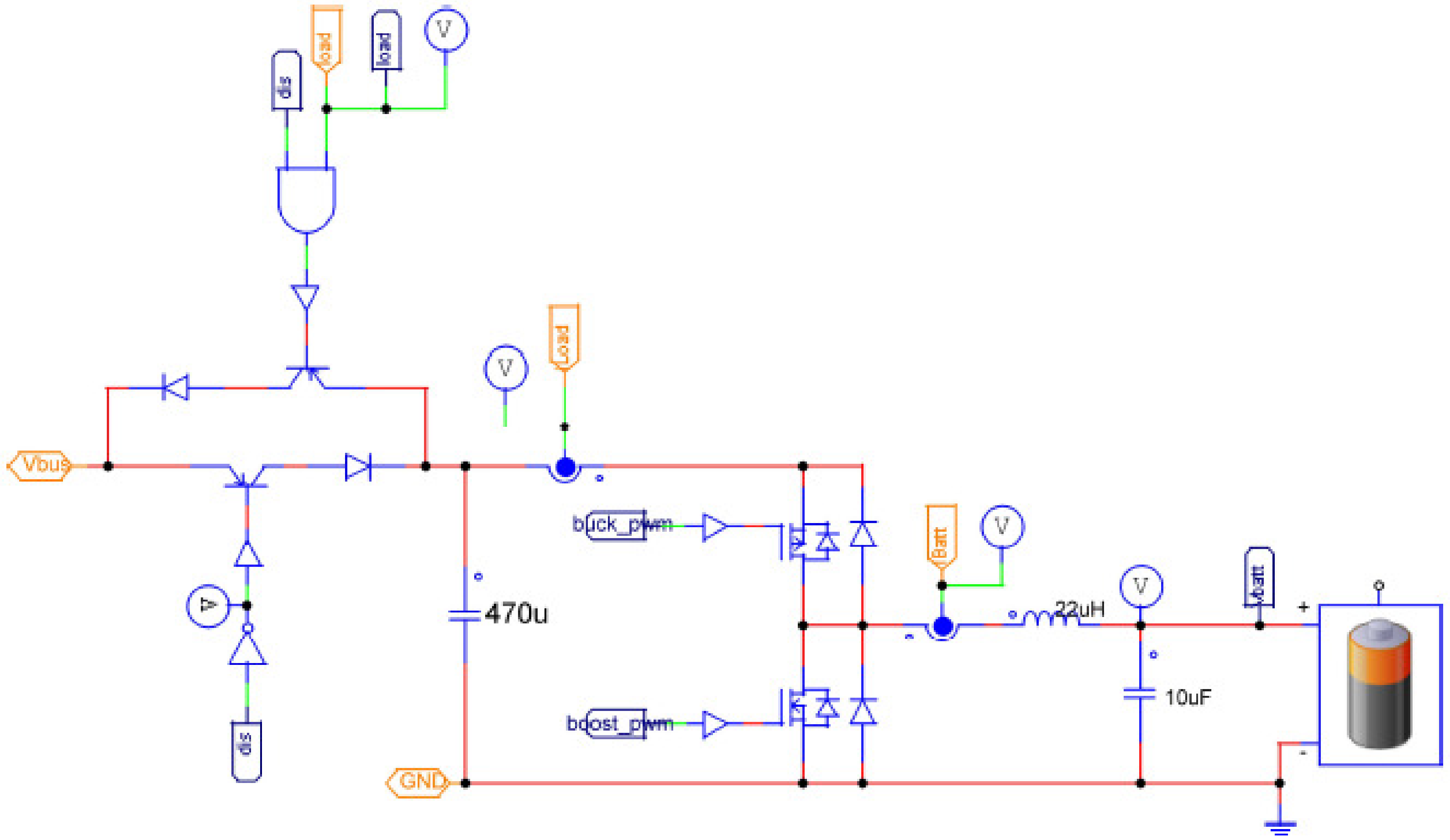

vout
pv_bus
ipv
ipv
ipv_filt

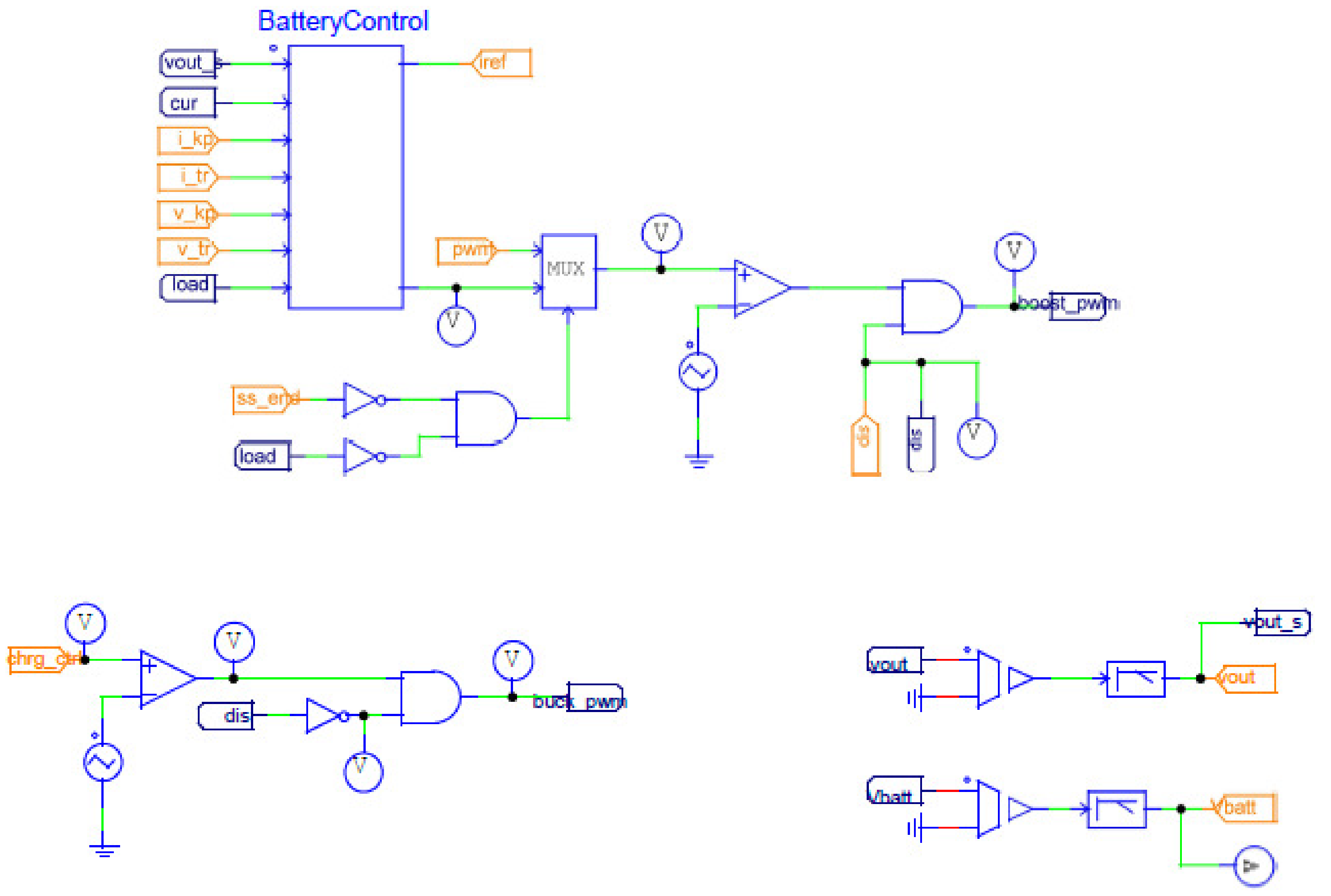
Appendix B.3 – Wind Subsystem Schematic



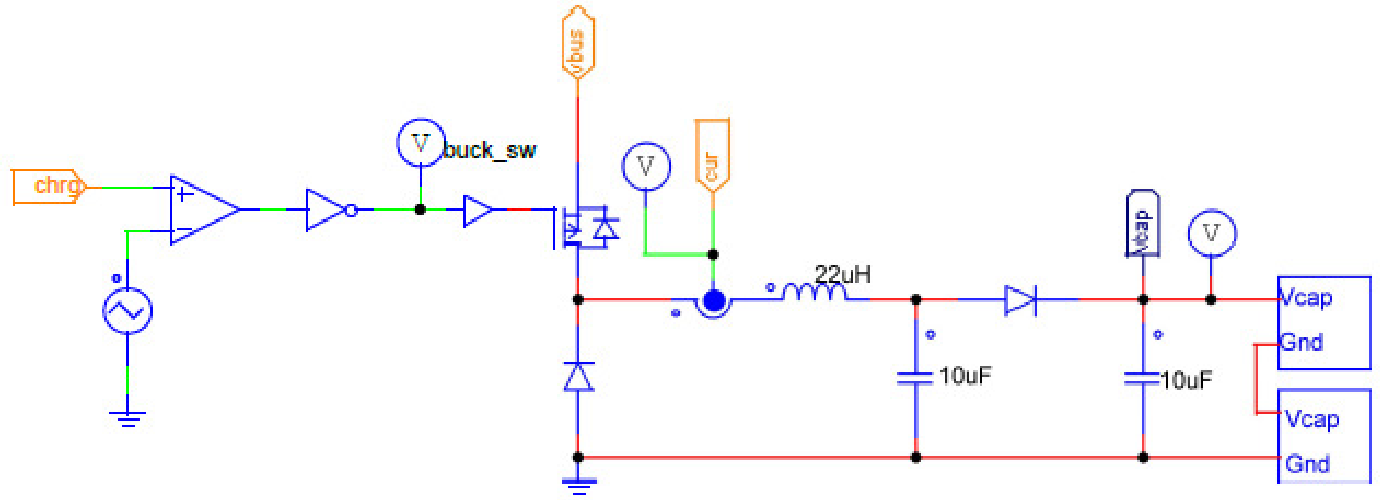

WindControl
vout
wind_bus
iref
iL
wind_lim
v_kp
v_ki
i_kp
i_ki
load
ss_pwm
MUX
pwm
ss_end
load
1000000
V

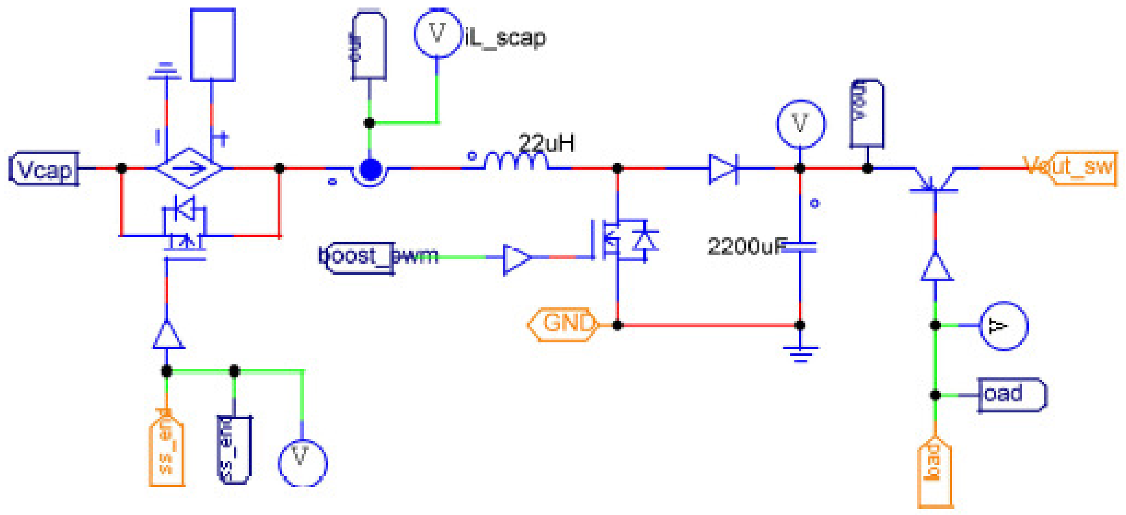
Appendix B.4 – Battery Subsystem

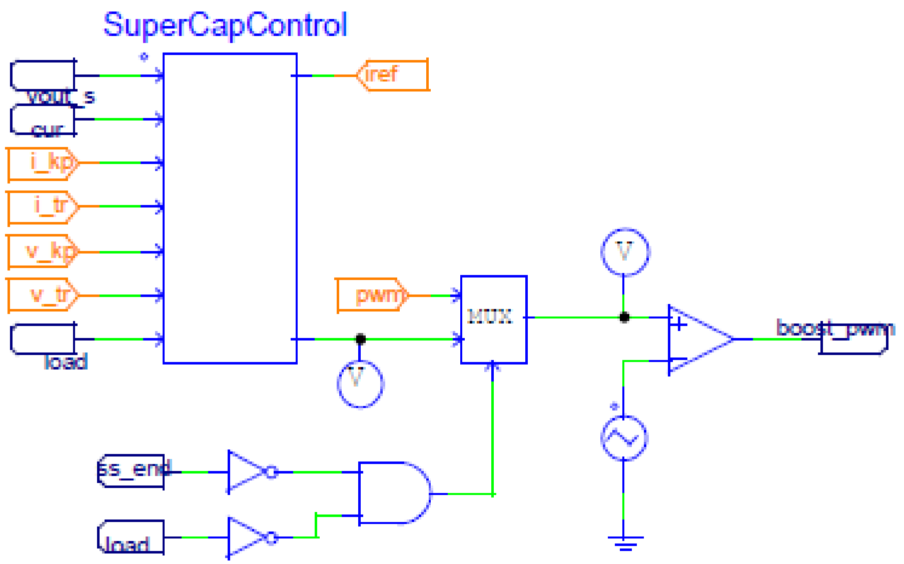


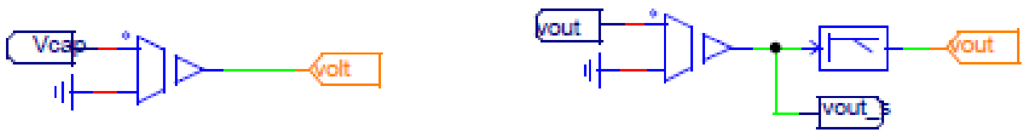

BatteryControl
vout_s
cur
i_kp
i_tr
v_kp
v_tr
load
iref
pwm
MUX
V
boost_pwm
ss_end
load
dis
dis
chrg_ctrl
dis
buck_pwm
vout
vout_s
vout
Vbatt
Vbatt

Appendix B.5 – Super Capacitor Subsystem






SuperCapControl
vout_s
cur
i_kp
i_tr
v_kp
v_tr
load
iref
pwm
MUX
V
V
boost_pwm
ss_end
load


Vcap
volt
vout
vout
vout_s

Appendix C – C Code used in the Hybrid Renewable Energy System Model

Appendix C.1 – Main.c

```
#include <Stdlib.h>
#include <String.h>
#include <math.h>
#include <Psim.h>

#include "general_control.h"
#include "renewable_control.h"
#include "storage_control.h"
#include "ems.h"

/* Inputs for readability */
#define PVVOLT in[0]
#define PVCUR in[1]
#define WINDVOLT in[2]
#define WINDCUR in[3]
#define BATT_VOLT in[4]
#define BATT_CUR in[5]
#define SCAP_VOLT in[6]
#define SCAP_CUR in[7]
#define PV_BUS in[8]
#define WIND_BUS in[9]
#define VBUS_BATT in[10]
#define VBUS_SCAP in[11]
#define ILOAD in[12]

/* Outputs for readability */
#define DIS_BATT out[0]
#define BATT_PWM out[1]
#define BATT_SSEND out[2]
#define BATT_EN out[3]
#define BATT_CHRG_CTRL out[4]
#define SCAP_PWM out[5]
#define SCAP_SSEND out[6]
#define SCAP_EN out[7]
#define SCAP_CHRG_CTRL out[8]
#define PV_PWM out[9]
#define PV_CLIM out[10]
#define PV_SSEND out[11]
#define PV_EN out[12]
#define WIND_PWM out[13]
#define WIND_CLIM out[14]
#define WIND_SSEND out[15]
#define WIND_EN out[16]
#define PAV out[17]
#define BATT_STATE out[18]
#define BATT_DIS_STATE out[19]
#define SCAP_STATE out[20]
#define SCAP_DIS_STATE out[21]
```

```
#define PV_RES out[22]
#define IPV_OUT out[23]

#define NUMBER_SAMPLES 32

const int Step = 25;
const double LoadTickTime = 5e-3;
const double FastTickTime = 20e-6;
const double IPVTickTime = 1e-3;

INIT_LOADS(sys_loads)
INIT_VOLTS(sys_volts)
INIT_PARAMS(pv_params)
INIT_SWITCHES(pv_switches)
INIT_SWITCHES(wind_switches)
INIT_STORAGE(batt_store)
INIT_STORE_SW(batt_sw, 10)
INIT_STORAGE(scap_store)
INIT_STORE_SW(scap_sw, 15)
INIT_ACTIVE(ren_act)

/* Moving averages */
float v_av = 0;
float i_av = 0;
float vscap_av = 0;
float iscap_av = 0;
float vbatt_av = 0;
float ibatt_av = 0;
float v_av_arr[NUMBER_SAMPLES] = {0};
float i_av_arr[NUMBER_SAMPLES] = {0};
float sv_av_arr[NUMBER_SAMPLES] = {0};
float si_av_arr[NUMBER_SAMPLES] = {0};
float bv_av_arr[NUMBER_SAMPLES] = {0};
float bi_av_arr[NUMBER_SAMPLES] = {0};

/* PV Load Balancing */
float dp = 0;
float power = 0;
float pv_res = 0;
double last_tick = 0;
double fast_tick = 0;

/* EMS control variables */
float pav = 0;

/////////////////////////////////////////////////////////////////////
// FUNCTION: SimulationStep
//   This function runs at every time step.
//double t: (read only) time
//double delt: (read only) time step as in Simulation control
//double *in: (read only) zero based array of input values. in[0] is the first node, in[1] second
input...
//double *out: (write only) zero based array of output values. out[0] is the first node, out[1]
second output...
```

```
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set the error
message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationStep(
        double t, double delt, double *in, double *out,
         int *pnError, char * szErrorMsg,
         void ** reserved_UserData, int reserved_ThreadIndex, void * reserved_AppPtr)
{

    /* Input averaging for the PV */
    v_av = fMovingAverage(v_av_arr, NUMBER_SAMPLES, PVVOLT);
    i_av = fMovingAverage(i_av_arr, NUMBER_SAMPLES, PVCUR);

    /* Input averaging for the super cap */
    vscap_av = fMovingAverage(sv_av_arr, NUMBER_SAMPLES, SCAP_VOLT);
    iscap_av = fMovingAverage(si_av_arr, NUMBER_SAMPLES, SCAP_CUR);

    /* Input averaging for the battery */
    vbatt_av = fMovingAverage(bv_av_arr, NUMBER_SAMPLES, BATT_VOLT);
    ibatt_av = fMovingAverage(bi_av_arr, NUMBER_SAMPLES, BATT_CUR);

    /*************************
    ** Update the measurements
    ***************************/
    sys_volts.vpv = v_av;
    sys_volts.vwind = WINDVOLT;
    sys_loads.iwind = WINDCUR;
    sys_volts.vbatt = BATT_VOLT;
    batt_store.volt = vbatt_av;
    sys_loads.ibatt = BATT_CUR;
    batt_store.cur = ibatt_av;
    sys_volts.vscap = SCAP_VOLT;
    scap_store.volt = vscap_av;
    sys_loads.iscap = SCAP_CUR;
    scap_store.cur = iscap_av;
    sys_loads.iload = ILOAD;

    /*****************************
    ** PV Load balancing
    ******************************/

    /* Power interupts - dop this on a faster loop for quicker tracking */
    if ((t - fast_tick) >= FastTickTime) {
        /* store the tick time */
        fast_tick = t;

      /* calculate the power & delta */
      dp = fabs(power - (PVVOLT * PVCUR));
     }

    /* Determine the power for this cycle */
    power = v_av * i_av;

    /* Main algorithm */
```

```
if (PVVOLT == 0 || PVCUR == 0) {
   pv_res = 1; // this may not strictly be neccessary
}
else if (pv_switches.state != OPERATING) {
   /* Just for continuity */
   vDetermineStep(&pv_params, v_av, i_av);
   pv_res = fBalanceLoad(&pv_params, power, pv_res);

   /* set the actual resistance */
   pv_res = (48 * 48) / power;
   if (pv_res < 9.7) {
      pv_res = 9.7;
      sys_loads.ipv = PVCUR;
   }
   else {
      sys_loads.ipv = v_av / pv_res; //((48 / pv_res) * 48) / PVVOLT;
   }
}
else {
   /* Main load balancing algorithm */
   if (((t - last_tick) >= LoadTickTime) || (dp > Step)){
      /* store the tick time */
      last_tick = t;

      /* Perform the load balancing algoriithm*/
      if (dp > Step) {
         vDetermineStep(&pv_params, PVVOLT, PVCUR);
         pv_res = fBalanceLoad(&pv_params, (PVVOLT * PVCUR), pv_res);
      }
      else {
         vDetermineStep(&pv_params, v_av, i_av);
         pv_res = fBalanceLoad(&pv_params, power, pv_res);
      }

      /* Limit the resistor*/
      if (pv_res < 9.7) {
         pv_res = 9.7;
         sys_loads.ipv = PVCUR;
      }
      else {
        sys_loads.ipv = ((48 / pv_res) * 48) / v_av;
      }
   }
}

/*****************************
** Renewable energy control
*******************************/
vPVControlTask(&pv_switches, sys_volts.vpv, PV_BUS);
vWindControlTask(&wind_switches, sys_volts.vwind, WIND_BUS);

/**************************
** Update the EMS
**************************/
if (wind_switches.connect_load == 0) {
    ren_act.wind_active = 1;
}
else {
    ren_act.wind_active = 0;
}

if (pv_switches.connect_load == 0) {
    ren_act.pv_active = 1;
}
else {
    ren_act.pv_active = 0;
}

if (batt_sw.connect_load == 0) {
    ren_act.batt_active = 1;
}
else {
    ren_act.batt_active = 0;
}

pav = vUpdateEMS(&sys_loads, &sys_volts, &batt_store, &scap_store, ren_act);

/****************************
** Perform the storage tasks
*****************************/

/* Used the non-averaged inputs for the control loop */
batt_store.volt = BATT_VOLT;
batt_store.cur = BATT_CUR;
scap_store.volt = SCAP_VOLT;
scap_store.cur = SCAP_CUR;

vSupercapChrgTask(&scap_store, delt);
vStorageTask(&scap_sw, SCAP_VOLT, VBUS_SCAP, 1);
vBatteryChrgTask(&batt_store, delt);
vStorageTask(&batt_sw, BATT_VOLT, VBUS_BATT, batt_store.dis);

/****************************
** Set the outputs
*****************************/
DIS_BATT = batt_store.dis;
BATT_PWM = batt_sw.pwm;
BATT_SSEND = batt_sw.cc_enable;
BATT_EN = batt_sw.connect_load;
BATT_CHRG_CTRL = batt_store.pwm_ctrl;
SCAP_PWM = scap_sw.pwm;
SCAP_SSEND = scap_sw.cc_enable;
SCAP_EN = scap_sw.connect_load;
SCAP_CHRG_CTRL = scap_store.pwm_ctrl;
PV_PWM = pv_switches.pwm;
PV_CLIM = PVCUR;
PV_SSEND = pv_switches.cc_enable;
PV_EN = pv_switches.connect_load;
WIND_PWM = wind_switches.pwm;
WIND_CLIM = WINDCUR;
```

```
    WIND_SSEND = wind_switches.cc_enable;
    WIND_EN = wind_switches.connect_load;

    /* Debug/measurement signals */
    PAV = pav;
    BATT_STATE = batt_sw.state;
    BATT_DIS_STATE = batt_store.dis;
    SCAP_STATE = scap_sw.state;
    SCAP_DIS_STATE = scap_store.dis;
    PV_RES = pv_res;
    IPV_OUT = sys_loads.ipv;
}

/////////////////////////////////////////////////////////////////////
// FUNCTION: SimulationBegin
//   Initialization function. This function runs once at the beginning of simulation
//   For parameter sweep or AC sweep simulation, this function runs at the beginning of each
simulation cycle.
//   Use this function to initialize static or global variables.
//const char *szId: (read only) Name of the C-block
//int nInputCount: (read only) Number of input nodes
//int nOutputCount: (read only) Number of output nodes
//int nParameterCount: (read only) Number of parameters is always zero for C-Blocks.
Ignore nParameterCount and pszParameters
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set the error
message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationBegin(
        const char *szId, int nInputCount, int nOutputCount,
         int nParameterCount, const char ** pszParameters,
         int *pnError, char * szErrorMsg,
         void ** reserved_UserData, int reserved_ThreadIndex, void * reserved_AppPtr)
{
// ENTER INITIALIZATION CODE HERE...

}

/////////////////////////////////////////////////////////////////////
// FUNCTION: SimulationEnd
//   Termination function. This function runs once at the end of simulation
//   For parameter sweep or AC sweep simulation, this function runs at the end of each
simulation cycle.
//   Use this function to de-allocate any allocated memory or to save the result of simulation
in an alternate file.
// Ignore all parameters for C-block
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationEnd(const char *szId, void ** reserved_UserData, int reserved_ThreadIndex,
void * reserved_AppPtr)
{


}
```

Appendix C.2 – renewable_control.h

```
#ifndef SRC_RENEWABLE_CONTROL_H_
#define SRC_RENEWABLE_CONTROL_H_

/* State machine constants */
#define VOLTLOW 0
#define CONST_CUR 1
#define VOLT_RAMP 2
#define OPERATING 3

#define POW2LOAD(V, P) (V * V) / P

/* Struct for storing the calculated INC values */
typedef struct {
   float di;
   float dv;
   float dp;
   float didv;
   float i_div_v;
   float step;
   float dpdv;
   int state;
} pv_params_t;

#define INIT_PARAMS(X) pv_params_t X = {0, 0, 0, 0, 0, 0};

/* Struct for the soft start switches */
typedef struct {
   float pwm;
   int cc_enable;
   int connect_load;
   int state;
} ren_switches_t;

#define INIT_SWITCHES(X) ren_switches_t  X = {0, 0, 0, 0};

/* MPPT &  Load Balancing Functions */
void vDetermineStep (pv_params_t * params, float vpv, float ipv);
float fBalanceLoad(pv_params_t * params, float power, float load);
void vPVControlTask(ren_switches_t * switches, float vpv, float vout);
void vWindControlTask(ren_switches_t * switches, float vwind, float vout);


#define MAX(x,y) (((x) > (y)) ? (x) : (y))
#define MIN(x,y) (((x) < (y)) ? (x) : (y))

const float N = 0.2;
const float DpDvNoise = 0.1;
const float Noise = 0.01;
const float LargeStep = 25;
const float StepMax = 10;

const float VrefSquared = 2304; // ie 48 squared
const float RMax = 96;
const float RMin= 9.6;

const float PVUnderVoltage = 10;
const float MinPV = 30;
const float WindUnderVoltage = 15;
const float MinWind = 20;
const float Vref = 48;

const float PWMOff = 0;
const float PVHalfPWM = 5;
const float WindHalfPWM = 2.5;

/*----------------------------------------------------------*/

void vDetermineStep (pv_params_t * params, float vpv, float ipv) {
  /* static values to determine history */
  static float old_p = 0;
  static float old_v = 0;
  static float old_i = 0;

  /* Algorithm paramaters */
  float power = ipv * vpv;

  /* calculate all the derivatives */
  params->di = ipv - old_i;
  params->dv = vpv - old_v;
  params->dp = power - old_p;

  /* Step size calculation*/
  if (params->di == 0 || params->dp == 0 || params->dv == 0
      || ipv == 0 || vpv == 0) {
    /* Avoid divide by zero erros */
    params->step = 0;
    params->state = 0;
  }
  else {
    /* Calculate the v/i parameters */
    params->didv = params->di / params->dv;
    params->i_div_v = -1 * (ipv / vpv);
    params->dpdv = params->dp / params->dv;

    if ((fabs(params->didv - params->i_div_v) < Noise)
        || (fabs(params->dpdv) < DpDvNoise)) {
      /* More or less at the MPP, don't change the load */
      params->step = 0;
      params->state = 1;
    }
    else {

      /* Calculate the exact load for current operating point */
      float res = VrefSquared / power;

      /* Calculate the step size, which is worked out as being
        a percentage of the resistance. The percentage is determined
```

```
       by the value of dpdv and a scaling factor N */
      params->step = MIN((res / 100) * (N * fabs(params->dpdv)), StepMax);
      params->state = 2;
    }
  }

  /* history for next iteration */
  old_i = ipv;
  old_v = vpv;
  old_p = power;
}
/*----------------------------------------------------------*/

float fBalanceLoad(pv_params_t * params, float power, float load) {
  float new_load = 0;
  static float old_load = 0;

  if (fabs(params->dp) > LargeStep) {
    /* When a large step occurs, simply re-calcualte the load
      to provide fast changes at transition points*/
    new_load = POW2LOAD(Vref, (0.98 * power));
    old_load = new_load;
  }
  else if (params->dp > 0) {
    /* Carry on moving the load in the same direction if a
      positive change in power occurs*/
    new_load = (old_load < load) ?  load + params->step :
                                load - params->step;
    old_load = load;
  }
  else {
    /* When power decreases, the direction is determined by dV */
    new_load = (params->dv > 0) ? load - (params->step) :
                              load + (params->step);
    old_load = load;
  }

  //old_load = load;

  return MAX(MIN(new_load, RMax), RMin);
}
/*----------------------------------------------------------*/

void vPVControlTask(ren_switches_t * switches, float vpv, float vout){

  /* State machine for control of the PV boost system*/
  switch (switches->state) {
    case VOLTLOW:
      /* State for when the pv output is too low */

      /* set the switches (ie disconnect everything) */
      switches->cc_enable = 0;
      switches->connect_load = 5;
      switches->pwm = PWMOff;
      /* determine next state */
      if (vpv > MinPV) {
          switches->state = CONST_CUR;
      }
      break;

    case CONST_CUR:
      /* First stage of soft start - charge the inductor */

      /* Connect the PV cell and turn on the const current source */
      switches->cc_enable = 5;

      /* determine next state */
      if (vout > MinPV){
          switches->state = VOLT_RAMP;
      }
      break;

    case VOLT_RAMP:
      /* State to ramp up the voltage during soft start */

      /* Set the PWM output to 0.5 */
      switches->pwm = PVHalfPWM;

      /* determine next state */
      if (vout >= Vref){
          switches->state = OPERATING;
      }
      break;

    case OPERATING:
      /* Connect the output load*/
      switches->cc_enable = 0;
      switches->connect_load = 0;
      switches->pwm = PWMOff;

      /* Check the PV cell hasn't gone undervoltage */
      if (vout < PVUnderVoltage) {
          switches->state = VOLTLOW;
      }
      break;

    default:
        printf("Entered unreachable state");
  }
}
/*-------------------------------------------------------*/

void vWindControlTask(ren_switches_t * switches, float vwind, float vout){

  /* State machine for control of the PV boost system*/
  switch (switches->state) {
    case VOLTLOW:
      /* State for when the wind output is too low */
      /* set the switches (ie disconnect everything) */
      switches->cc_enable = 0;
      switches->connect_load = 5;
      switches->pwm = PWMOff;

      /* determine next state */
      if (vwind > MinWind) {
         switches->state = CONST_CUR;
      }
      break;

    case CONST_CUR:
      /* First stage of soft start - charge the inductor */

      /* Connect the PV cell and turn on the const current source */
      switches->cc_enable = 5;

      /* determine next state */
      if (vout > vwind){
         switches->state = VOLT_RAMP;
      }
      break;

    case VOLT_RAMP:
      /* State to ramp up the voltage during soft start */

      /* Set the PWM output to 0.5 */
      switches->pwm = WindHalfPWM;

      /* determine next state */
      if (vout >= Vref){
         switches->state = OPERATING;
      }
      break;

    case OPERATING:
      /* Connect the output load*/
      switches->cc_enable = 0;
      switches->connect_load = 0;
      switches->pwm = PWMOff;

      /* Check the wind turbine hasn't gone into undervoltage */
      if (vout < WindUnderVoltage) {
         switches->state = VOLTLOW;
      }
      break;

    default:
      printf("Wind control task - Entered unreachable state");
  }
}

#endif /* SRC_RENEWABLE_CONTROL_H_ */
```

Appendix C.3 – storage_control.h

```
#ifndef SRC_STORAGE_CONTROL_H_
#define SRC_STORAGE_CONTROL_H_

#include "general_control.h"

/* Supercapacitor charging state machine constants */
#define SCAP_FLOAT 0
#define SCAP_CC 1
#define SCAP_CV 2

/* Battery charging state machine constants */
#define BATT_FLOAT 0
#define BATT_CC 1
#define BATT_CV 2

/* Soft start control FSM constants */
#define FLOAT 0
#define SOFT_START_CC 1
#define SS_VOLT_RAMP 2
#define STORE_OPERATING 3

/* Threhsold limits for charge/discharge */
const float BattVoltLimit = 29;
const float BatCVEnd = 0.5;
static const int CapVoltage = 32;
static const int CapFloatCur = 0.5;
static const float VbusRef = 48;
static const float StoreVoltLow = 20;

/* State storage struct & macro */
typedef struct {
   int state;
   int dis;
   float volt;
   float cur;
   float cur_ref;
   float pwm_ctrl;
} storage_t;

#define INIT_STORAGE(X) storage_t X = {0, 0, 0, 0, 0, 0};

/* Struct for the soft start switches */
typedef struct {
   float pwm;
   int cc_enable;
   int connect_load;
   int max_pwm;
   int state;
} store_switches_t;

#define INIT_STORE_SW(X, MAX) store_switches_t X = {0, 0, 0, MAX, 0};

/* Statemachine tasks for the control of the charging components */
```

```
void vSupercapChrgTask(storage_t * params, float delta);
void vBatteryChrgTask(storage_t * params, float delta);
void vStorageTask(store_switches_t * switches, float vin, float vout, float dis);

/*-----------------------------------------------------------*/

/* Supercapacitor charging constants & PI settings */
static const float ScapKPCC = 4;
static const float ScapTRCC = 1.5e-5;
static const float ScapKPCV = 100;
static const float ScapTRCV = 1e-6;

/* Battery charging constants & PI settings */
static const float BattKPCC = 1.7;
static const float BattTRCC = 2.2e-6;
static const float BattKPCV = 2.8;
static const float BattTRCV = 1.5e-5;

/*-----------------------------------------------------------*/
void vSupercapChrgTask(storage_t * params, float delta){
   static INIT_PI(scap_pi, 15)
   static float cur_ramp = 0;
   scap_pi.delt = delta;

   /* State machine for control of the supercapacitor subsystem */
   switch (params->state) {
      case SCAP_FLOAT:
         /* When the super capacitor is not being charged
            then the switches must be reset and the integral cleared*/

         /* Reset the PI controller */
         if (params->cur_ref == 0) {
            cur_ramp = 0;
            scap_pi.error = 0;
            scap_pi.integral = 0;
            params->pwm_ctrl = 0;
         }

         /* Set the PI gain values */
         scap_pi.kp = ScapKPCC;
         scap_pi.tr = ScapTRCC;

         /* Soft start ramp up when the converter is turned on */
         /* To prevent inrush current the current is ramped up slowly */
         if (cur_ramp < params->cur_ref) {
            cur_ramp = cur_ramp + 500e-6;//(params->cur_ref * 0.1e-3);
            scap_pi.error = cur_ramp - params->cur;
            params->pwm_ctrl = fCalcPI(&scap_pi);
         }
         else if (params->cur_ref > 0) {
            /* Move onto normal operation when ramp up is done */
            params->state = SCAP_CC;
            scap_pi.error = params->cur_ref - params->cur;
            params->pwm_ctrl = fCalcPI(&scap_pi);
         }
```

```
            params->cur_ref = cur_ramp;

            break;

        case SCAP_CC:
            /* State for the constant current charging mode */

            /* Clear the soft start ramp variable */
            cur_ramp = 0;

            /* Update the PWM control signal */
            scap_pi.kp = ScapKPCC;
            scap_pi.tr = ScapTRCC;
            scap_pi.error = params->cur_ref - params->cur;
            params->pwm_ctrl = fCalcPI(&scap_pi);

            /* Move to the next state when required */
            if (params->volt > CapVoltage) {
                params->state = SCAP_CV;
            }
            else if (params->cur_ref == 0) {
                scap_pi.integral = 0;
                params->state = SCAP_FLOAT;
            }
            break;

        case SCAP_CV:
            /* State for the constant voltage charging mode */

            /* Update the PWM control signal */
            scap_pi.kp = ScapKPCV;
            scap_pi.tr = ScapTRCV;
            scap_pi.error = CapVoltage - params->volt;
            params->pwm_ctrl = fCalcPI(&scap_pi);

            /* Return to Float mode when fully chargedor interrupted */
            if (params->cur < CapFloatCur || params->cur_ref == 0) {
                params->state = SCAP_FLOAT;
            }
            break;

        default:
            printf("Super cap FSM entered unreachable state");
    }
}
/*------------------------------------------------------------*/

void vBatteryChrgTask(storage_t * params, float delta) {
    static INIT_PI(batt_pi, 10)
    static float cur_ramp = 0;
    batt_pi.delt = delta;

    /* When discharging goto float mode */
    if (params->dis > 0) {
```

```
    params->state = BATT_FLOAT;
}

/* State machine for control of the supercapacitor subsystem */
switch (params->state) {
    case BATT_FLOAT:
        /* When the super capacitor is not being charged
            then the switches must be reset and the integral cleared*/

        /* Reset the PI controller */
        if (params->cur_ref == 0 || params->dis > 0) {
            cur_ramp = 0;
            batt_pi.error = 0;
            batt_pi.integral = 0;
            params->pwm_ctrl = 0;
        }

        /* Set the PI gain values */
        batt_pi.kp = BattKPCC;
        batt_pi.tr = BattTRCC;

        /* Soft start ramp up when the converter is turned on */

        /* To prevent inrush current the current is ramped up slowly */
        if (cur_ramp < params->cur_ref) {
            cur_ramp = cur_ramp + (params->cur_ref * 1e-3);
            batt_pi.error = cur_ramp - params->cur;
            params->pwm_ctrl = fCalcPI(&batt_pi);
        }
        else if (params->cur_ref > 0) {
            /* Move onto normal operation when ramp up is done */
            params->state = SCAP_CC;
            batt_pi.error = params->cur_ref - params->cur;
            params->pwm_ctrl = fCalcPI(&batt_pi);
        }
        break;

    case BATT_CC:
        /* State for the constant current charging mode */

        /* Clear the soft start ramp variable */
        cur_ramp = 0;

        /* Update the PWM control signal */
        batt_pi.kp = BattKPCC;
        batt_pi.tr = BattTRCC;
        batt_pi.error = params->cur_ref - params->cur;
        params->pwm_ctrl = fCalcPI(&batt_pi);

        /* Move to Constant Voltage if required */
        if (params->volt > BattVoltLimit) {
            params->state = BATT_CV;
        }
        else if (params->cur_ref == 0) {
            params->state = BATT_FLOAT;
        }
        break;

    case BATT_CV:
        /* State for the constant voltage charging mode */

        /* Update the PWM control signal */
        batt_pi.kp = BattKPCV;
        batt_pi.tr = BattTRCV;
        batt_pi.error = BattVoltLimit - params->volt;
        params->pwm_ctrl = fCalcPI(&batt_pi);

        /* Move to float mode when charge is fully complete */
        if (params->cur < BatCVEnd || params->cur_ref == 0) {
            params->state = BATT_FLOAT;
        }
        break;

    default:
        printf("Battery FSM entered unreachable state");
  }
}
/*---------------------------------------------------------*/

void vStorageTask(store_switches_t * switches, float vin, float vout, float dis){

  /* When not discharging the battery should be floated */
  if (dis == 0) {
    switches->state = FLOAT;
  }

  /* State machine for control of the  Battery discharge subsystem*/
  switch (switches->state) {
    case FLOAT:
        /* State to disconnect the battery when not discharging */

        /* When the battery isn't discharging disconnect everything */
        switches->cc_enable = 0;
        switches->connect_load = 5;
        switches->pwm = 0;

        if (dis != 0 && vin > StoreVoltLow) {
            switches->state = SOFT_START_CC;
        }
        break;

    case SOFT_START_CC:
        /* First stage of soft start - charge the inductor */

        /* Connect the PV cell and turn on the const current source */
        switches->cc_enable = 5;

        /* determine next state */
        if (vout > (0.95 * vin)){
            switches->state = SS_VOLT_RAMP;
        }
        break;

    case SS_VOLT_RAMP:
        /* State to ramp up the voltage during soft start */

        /* Set the PWM output to 0.5 */
        switches->pwm = switches->max_pwm / 2;
        //switches->cc_enable = 0;

        /* determine next state */
        if (vout >= VbusRef){
            switches->state = STORE_OPERATING;
        }
        break;

    case STORE_OPERATING:
        /* Connect the output load*/
        switches->cc_enable = 0;
        switches->connect_load = 0;
        switches->pwm = 0;

        /* Check the PV cell hasn't gone undervoltage */
        if (vout < StoreVoltLow) {
            switches->state = FLOAT;
        }
        break;

    default:
        printf("Entered unreachable state in vStorageTask");
  }
}

#endif /* SRC_STORAGE_CONTROL_H_ */
```

Appendix C.4 – general_control.h

```
#ifndef SRC_GENERAL_CONTROL_H_
#define SRC_GENERAL_CONTROL_H_

/* PI Controller struct and macro */
typedef struct {
    float error;
    float kp;
    float tr;
    float td;
    float integral;
    float error_old;
    int max;
    int min;
    float delt;
} pid_t;

#define INIT_PI(NAME, MAX) pid_t NAME = {0, 0, 0, 0, 0, 0, MAX, 0, 0};
```

```
/* Range Calculation struct and macro */
typedef struct{
   float min_val;
   float max_val;
   float range;
} range_t;

#define INIT_RANGE(X) range_t X = {100 ,0 ,0};

/* Variable input filter struct and macro */
typedef struct {
   int count;
   int ready;
   float max_delt;
   float last_val;
   float filt_value;
} filt_t;

#define INIT_FILT(NAME, MAX) filt_t NAME = {0, 0, MAX, 0, 0};

/* PI Function */
float fCalcPI (pid_t *params);
float fMovingAverage(float arr[], int arr_size, float meas);
float fNormalise(float val, float max, float min);
float fCalcVoltFitness (float error, double step_time, double time);
float fMSE (float error);
void vGetRange(range_t *meas, float value);

#define NOT_STARTED 0
#define STEADY 1
#define STEP 2
#define POST_STEP 3

const float ErrorMin = 0.005;

/*-----------------------------------------------------------*/

float fCalcPI (pid_t *params){
   /* get the gain parameters */
   float ki = (params->kp)/(params->tr);

   /* integral */
   float t_integral = params->integral + (ki*(params->error)*(params->delt));

   /* main pid algorithm */
   float prop = (params->kp)*(params->error);
   float t_pi = prop + t_integral;

   /* conditional integration */
   if (t_pi > (params->max)){
      return params->max;
   }
   else if (t_pi < (params->min)){
      return params->min;
```

```
    }
    else{
        params->integral = t_integral;
        return t_pi;
    }
}
/*----------------------------------------------------------*/

float fDiscCalcPI (pid_t *params){

    /* integral */
    float t_integral = params->integral + (params->tr * params->error);

    /* main pid algorithm */
    float prop = (params->kp)*(params->error);
    float t_pi = prop + t_integral;

    /* conditional integration */
    if (t_pi > (params->max)){
        return params->max;
    }
    else if (t_pi < (params->min)){
        return params->min;
    }
    else{
        params->integral = t_integral;
        return t_pi;
    }
}
/*----------------------------------------------------------*/

float fMovingAverage(float arr[], int arr_size, float meas){
    int i = 0;
    float average = 0;

    /* Update the array contents */
    for (i = arr_size - 1; i > 0; i--){
        arr[i] = arr[i - 1];
    }
    arr[0] = meas / arr_size;

    /* now get the average from the array */
    for (i = 0; i < arr_size; i++) {
        average += arr[i];
    }

    return average;
}
/*----------------------------------------------------------*/

float fNormalise(float val, float max, float min) {
    if (val <= min) {
        return 0;
    }
    else if (max <= val) {
        return 1;
    }
    else {
        return (val - min) / (max - min);
    }
}
/*------------------------------------------------------------*/

float fCalcVoltFitness (float error, double step_time, double time){
    static int state = 0;
    static int count = 0;
    static float peak_error = 0;
    static float cum_error = 1e9;

    /* Essentially an MSE alogrithm that penalises under / overshoot */
    switch (state) {
        case NOT_STARTED:
            if (time >= step_time) {
                state = STEP;
            }
        break;

        case STEP:
            if (fabs(error) > peak_error) {
                peak_error = fabs(error);
            }
            if (fabs(error) < ErrorMin) {
                cum_error = peak_error;
                state = POST_STEP;
            }
            break;

        case POST_STEP:
            count++;
            cum_error = (fabs(error) > ErrorMin) ?  cum_error + (5 * fabs(error)) :
                                            cum_error + fabs(error);
            break;
    }

    return (count > 0) ? cum_error/count : cum_error;
}
/*------------------------------------------------------------*/

float fMSE (float error){
    static int count = 0;
    static float cum_error = 0;

    /* Perform the MSE algorithm */
    count++;
    cum_error = cum_error + sqrt(error*error);
    return cum_error/count;
}
/*------------------------------------------------------------*/

void vGetRange(range_t *meas, float value) {
    /* Basic functiont to get the range of any measured value */
    if (value > meas->max_val){
        meas->max_val = value;
    }
    if (value < meas-> min_val){
        meas->min_val = value;
    }

    meas->range = meas->max_val - meas->min_val;
}
/*--------------------------------------------------------*/

void fInputFilter(filt_t *filt_ctrl, float adc_val) {
    float delt = filt_ctrl->last_val - adc_val;

    if (filt_ctrl->count == 0) {
        /* First value, reset the filter value */
        filt_ctrl->count++;
        filt_ctrl->filt_value = adc_val * 0.125;
    }
    else if (filt_ctrl->count >= 7) {
        /* Last value in the averaging filter */
        filt_ctrl->ready = 1;
        filt_ctrl->count = 0;
        filt_ctrl->filt_value += adc_val * 0.125;
        filt_ctrl->last_val = filt_ctrl->filt_value;
    }
    else {
        /* Last value in the averaging filter */
        filt_ctrl->count++;
        filt_ctrl->filt_value += adc_val * 0.125;
    }
}

#endif /* SRC_GENERAL_CONTROL_H_ */
```

Appendix C.5 – EMS.h

```
#ifndef EMS_H_
#define EMS_H_

#include "storage_control.h"

#define CUR2POWER(V, I) ((V) * (I))
#define POWER2LOAD(V, P) P / V

/* Constants for controlling the storage charge/discharge */
static const float CapLowVolt = 22;
static const float BattLowVolt = 20;
static const float ScapFullVolt = 31;
static const float BattFullVolt = 28;
static const float PowerMax = 240;

/* Structs and subsequent macro declarations */
```

```
typedef struct{
   float ipv;
   float iwind;
   float iscap;
   float ibatt;
   float iload;
} loads_t;

#define INIT_LOADS(X) loads_t X = {0, 0, 0, 0, 0};

typedef struct{
   float vpv;
   float vwind;
   float vbatt;
   float vscap;
   float vout;
} voltages_t;

#define INIT_VOLTS(X) voltages_t X = {0, 0, 0, 0, 0};

typedef struct {
   int wind_active;
   int pv_active;
   int batt_active;
} active_t;

#define INIT_ACTIVE(X) active_t X = {0, 0, 0};

/*** Main EMS funcitons ***/
float vUpdateEMS(loads_t *loads, voltages_t *volts, storage_t*batt_chrg,
   storage_t* scap_chrg, active_t act);

/*-----------------------------------------------------------*/
static float prvCalcPower(loads_t *loads, voltages_t * volts, active_t act) {

   /* Get the power into and out of the system */
   float power = 0;

   if (act.wind_active > 0) {
      power += loads->iwind * volts->vwind;
   }

   if (act.pv_active > 0) {
      power += loads->ipv * volts->vpv;
   }

   /* Calculate the power surplus/deficit */
   return power; // - itotal;
}

/*-----------------------------------------------------------*/
static void prvUpdateSCapStore(storage_t *charge, float Pav) {

   if (charge->state != SCAP_FLOAT) {
     /* Determine the available current, allowing a degree of
          error to allow for ripple */
     charge->cur_ref = charge->state == SCAP_CV ? charge->cur * 1.05 :
                                         POWER2LOAD(charge->volt, Pav) * 0.95;
   }
   else if (charge->volt < ScapFullVolt) {
     charge->cur_ref = POWER2LOAD(charge->volt, Pav) * 0.95;
   }
   else {
     charge->cur_ref = 0;
   }
}

/*---------------------------------------------------------*/
static void prvUpdateBattStore(storage_t *charge, float Pav) {

  if (charge->state != BATT_FLOAT) {
   /* Determine the available current, allowing a degree of
      error to allow for ripple */
     charge->cur_ref = charge->state == BATT_CV ? charge->cur * 1.05 :
                                         POWER2LOAD(charge->volt, Pav) * 0.95;
  }
  else if (charge->volt < BattFullVolt) {
   charge->cur_ref = POWER2LOAD(charge->volt, Pav) * 0.95;
  }
  else {
   charge->cur_ref = 0;
  }
}

/*---------------------------------------------------------*/
float vUpdateEMS(loads_t *loads, voltages_t *volts, storage_t* batt_chrg,
  storage_t* scap_chrg, active_t act) {

  /* Step one - determine the power in the system */
  float pav = prvCalcPower(loads, volts, act);
  float pload = 48 * loads->iload;

  /* Step three - perform the EMS algorithm */
  if (pav > 0) {

   if ((pav < pload) && (volts->vbatt > BattLowVolt) &&
     (volts->vscap < CapLowVolt)) {
     /* Discharge the battery when there is insufficient power
        being produced by the renwables */

     /* Update the available power, waiting for the
         battery converter to come online first*/
     if (act.batt_active > 0) {
      pav = MIN(pav + pload, PowerMax);
     }

     /* Set the battery up to discharge */
     batt_chrg->cur_ref = 0;
     batt_chrg->dis = 1;
   }
   else if ((volts->vbatt <= BattLowVolt) || (volts->vscap >= ScapFullVolt)
     || (pav >= pload)) {
     /* Disable the battery discahrging function by default */
     batt_chrg->dis = 0;
   }
   else if ((batt_chrg->dis == 1) && (act.batt_active > 0)) {
     /* Update the power to carry on discharging the battery */
     pav = MIN(pav + pload, 240);
   }

   /* Update the scap charging parameters */
   prvUpdateSCapStore(scap_chrg, pav);

   if (scap_chrg->cur_ref != 0) {
     pav = scap_chrg->state == SCAP_CV ? pav - (scap_chrg->cur * scap_chrg->volt) : 0;
   }

   if (pav > 0) {
     /* If still surplus power, charge battery */

     prvUpdateBattStore(batt_chrg, pav);
     pav = pav - (batt_chrg->volt * batt_chrg->cur_ref);

     /* If still surplus power, dump current */
     if (pav > 0) {
      printf("Need to add a dump load");
     }
   }
   else {
     batt_chrg->cur_ref = 0;
   }
  }
  else if ((scap_chrg->state == SCAP_CC || volts->vscap < CapLowVolt) &&
   (volts->vbatt > BattLowVolt)) {
   /* Start charging the supercapacitor from the battery when it is
      deeply discharged. Charge with half maximum current */

   /* Set the battery up to discharge */
   batt_chrg->cur_ref = 0;
   batt_chrg->dis = 1;

   /* Start discharging the battery into the cap when
      it has reached the correct output voltage */
   if (act.batt_active > 0) {
     prvUpdateSCapStore(scap_chrg, MIN(1.1 * pload, PowerMax));
   }
   else {
     scap_chrg->state = SCAP_FLOAT;
     scap_chrg->cur_ref = 0;
   }
  }
  else {
   /* Float the supercap when no power available */
   batt_chrg->cur_ref = 0;
```

```
    scap_chrg->state = SCAP_FLOAT;
  }

  return prvCalcPower(loads, volts, act);
}

#endif /* EMS_H_ */
```

Appendix D – FPGA Design Files

Appendix D.1 – der_controller_top.vhd

```
library ieee;
	use ieee.std_logic_1164.all;
library der_controller_lib;
	use der_controller_lib.components.all;

entity der_controller_top is
	port (
		adc_sdo		: in std_logic;
		adc_sck		: out std_logic;
		adc_conv	: out std_logic;
		pwm_out		: out std_logic
	);
end entity der_controller_top;

architecture struct of der_controller_top is

	signal aclk			: std_logic;
	signal aresetn			: std_logic_vector(0 downto 0);
	signal fis_clk			: std_logic;
	signal fis_resetn		: std_logic_vector(0 downto 0);
	signal op_mode			: std_logic_vector(1 downto 0);

	signal wr_enable		: std_logic;
	signal fis_start		: std_logic;
	signal fis_ready		: std_logic;
	signal fis_cur			: std_logic_vector(31 downto 0);
	signal fis_dcur			: std_logic_vector(31 downto 0);
	signal fis_iref			: std_logic_vector(31 downto 0);
	signal fis_diref		: std_logic_vector(31 downto 0);

	signal adc_intrpt		: std_logic;
	signal adc_meas_valid		: std_logic;
	signal adc_volt			: std_logic_vector(31 downto 0);
	signal adc_volt_raw		: std_logic_vector(15 downto 0);
	signal adc_cur			: std_Logic_vector(31 downto 0);
	signal adc_cur_raw		: std_logic_vector(11 downto 0);
	signal adc_ind			: std_logic_vector(31 downto 0);
	signal adc_ind_raw		: std_logic_vector(11 downto 0);

	signal iref_en			: std_logic;
	signal iref_fpga		: std_logic_vector(31 downto 0);

	signal dma_axis_tdata		: std_logic_vector(127 downto 0);
	signal dma_axis_tkeep		: std_logic_vector(15 downto 0);
	signal dma_axis_tlast		: std_logic;
	signal dma_axis_tready		: std_logic;
	signal dma_axis_tvalid		: std_logic;

	signal dma_wr_enable		: std_logic;
	signal dma_start			: std_logic;
	signal vkp_stdy_mem1		: std_logic_vector(31 downto 0);
	signal vkp_stdy_mem2		: std_logic_vector(31 downto 0);
	signal vkp_stdy_mem3		: std_logic_vector(31 downto 0);
	signal vkp_stdy_mem4		: std_logic_vector(31 downto 0);
	signal vkp_delt_mem1		: std_logic_vector(31 downto 0);
	signal vkp_delt_mem2		: std_logic_vector(31 downto 0);
	signal vkp_delt_mem3		: std_logic_vector(31 downto 0);
	signal vkp_delt_mem4		: std_logic_vector(31 downto 0);
	signal vki_stdy_mem1		: std_logic_vector(31 downto 0);
	signal vki_stdy_mem2		: std_logic_vector(31 downto 0);
	signal vki_stdy_mem3		: std_logic_vector(31 downto 0);
	signal vki_stdy_mem4		: std_logic_vector(31 downto 0);
	signal vki_delt_mem1		: std_logic_vector(31 downto 0);
	signal vki_delt_mem2		: std_logic_vector(31 downto 0);
	signal vki_delt_mem3		: std_logic_vector(31 downto 0);
	signal vki_delt_mem4		: std_logic_vector(31 downto 0);
	signal ikp_stdy_mem1		: std_logic_vector(31 downto 0);
	signal ikp_stdy_mem2		: std_logic_vector(31 downto 0);
	signal ikp_stdy_mem3		: std_logic_vector(31 downto 0);
	signal ikp_stdy_mem4		: std_logic_vector(31 downto 0);
	signal ikp_delt_mem1		: std_logic_vector(31 downto 0);
	signal ikp_delt_mem2		: std_logic_vector(31 downto 0);
	signal ikp_delt_mem3		: std_logic_vector(31 downto 0);
	signal ikp_delt_mem4		: std_logic_vector(31 downto 0);
	signal iki_stdy_mem1		: std_logic_vector(31 downto 0);
	signal iki_stdy_mem2		: std_logic_vector(31 downto 0);
	signal iki_stdy_mem3		: std_logic_vector(31 downto 0);
	signal iki_stdy_mem4		: std_logic_vector(31 downto 0);
	signal iki_delt_mem1		: std_logic_vector(31 downto 0);
	signal iki_delt_mem2		: std_logic_vector(31 downto 0);
	signal iki_delt_mem3		: std_logic_vector(31 downto 0);
	signal iki_delt_mem4		: std_logic_vector(31 downto 0);

	signal vkp_addr			: std_logic_vector(5 downto 0);
	signal vkp_cons			: std_logic_vector(95 downto 0);
	signal vki_addr			: std_logic_vector(5 downto 0);
	signal vki_cons			: std_logic_vector(95 downto 0);
	signal ikp_addr			: std_logic_vector(5 downto 0);
	signal ikp_cons			: std_logic_vector(95 downto 0);
	signal iki_addr			: std_logic_vector(5 downto 0);
	signal iki_cons			: std_logic_vector(95 downto 0);

	signal fis_valid			: std_logic;
	signal vkp_data			: std_logic_vector(31 downto 0);
	signal vki_data			: std_logic_vector(31 downto 0);
	signal ikp_data			: std_logic_vector(31 downto 0);
	signal iki_data			: std_logic_vector(31 downto 0);

begin

	cpu_sub_sys: component cpu_block_wrapper
		port map (
			aclk				=> aclk,
			aresetn				=> aresetn,
                fis_clk                         => fis_clk,
                fis_resetn                      => fis_resetn,
                -- DMA AXI stream
                dma_axis_mm2s_tdata             => dma_axis_tdata,
                dma_axis_mm2s_tkeep             => open,
                dma_axis_mm2s_tlast             => dma_axis_tlast,
                dma_axis_mm2s_tready            => dma_axis_tready,
                dma_axis_mm2s_tvalid            => dma_axis_tvalid,
                -- ADC Interface
                core0_nfiq                      => adc_intrpt,
                meas_valid                      => adc_meas_valid,
                voltage                         => adc_volt,
                raw_volt                        => adc_volt_raw,
                current                         => adc_cur,
                raw_current                     => adc_cur_raw,
                inductor                        => adc_ind,
                raw_inductor                    => adc_ind_raw,
                -- Control loop interface
                iref_en                         => iref_en,
                iref_fpga                       => iref_fpga,
                -- AXI register outputs
                wr_enable                       => wr_enable,
                start                           => fis_start,
                ready                           => fis_ready,
                cur_out                         => fis_cur,
                di_out                          => fis_dcur,
                iref_out                        => fis_iref,
                diref_out                       => fis_diref,
                fixed_pwm                       => op_mode(1),
                enable_pwm                      => op_mode(0)
        );

adc_if: component adc_interface
        port map (
                clock           => aclk,
                resetn          => aresetn(0),
                -- Output to the voltage ADC
                adc_sdo         => adc_sdo,
                adc_sck         => adc_sck,
                adc_conv        => adc_conv,
                -- CPU interrupt
                cpu_intrpt      => adc_intrpt,
                -- Raw data output bus
                rdat_valid      => adc_meas_valid,
                r_volt_data     => adc_volt_raw,
                r_ind_data      => adc_ind_raw,
                r_load_data     => adc_cur_raw,
                volt_data       => adc_volt,
                ind_data        => adc_ind,
                load_data       => adc_cur
        );

dma_if: component dma_interface
        port map(
                s_axis_aclk             => aclk,
		s_axis_aresetn		=> aresetn(0),
		s_axis_tvalid		=> dma_axis_tvalid,
		s_axis_tready		=> dma_axis_tready,
		s_axis_tdata		=> dma_axis_tdata,
		s_axis_tlast		=> dma_axis_tlast,
		wr_enable		=> dma_wr_enable,
		vkp_delt_mem1		=> vkp_delt_mem1,
		vkp_delt_mem2		=> vkp_delt_mem2,
		vkp_delt_mem3		=> vkp_delt_mem3,
		vkp_delt_mem4		=> vkp_delt_mem4,
		vkp_stdy_mem1		=> vkp_stdy_mem1,
		vkp_stdy_mem2		=> vkp_stdy_mem2,
		vkp_stdy_mem3		=> vkp_stdy_mem3,
		vkp_stdy_mem4		=> vkp_stdy_mem4,
		vki_delt_mem1		=> vki_delt_mem1,
		vki_delt_mem2		=> vki_delt_mem2,
		vki_delt_mem3		=> vki_delt_mem3,
		vki_delt_mem4		=> vki_delt_mem4,
		vki_stdy_mem1		=> vki_stdy_mem1,
		vki_stdy_mem2		=> vki_stdy_mem2,
		vki_stdy_mem3		=> vki_stdy_mem3,
		vki_stdy_mem4		=> vki_stdy_mem4,
		iki_delt_mem1		=> iki_delt_mem1,
		iki_delt_mem2		=> iki_delt_mem2,
		iki_delt_mem3		=> iki_delt_mem3,
		iki_delt_mem4		=> iki_delt_mem4,
		iki_stdy_mem1		=> iki_stdy_mem1,
		iki_stdy_mem2		=> iki_stdy_mem2,
		iki_stdy_mem3		=> iki_stdy_mem3,
		iki_stdy_mem4		=> iki_stdy_mem4,
		ikp_delt_mem1		=> ikp_delt_mem1,
		ikp_delt_mem2		=> ikp_delt_mem2,
		ikp_delt_mem3		=> ikp_delt_mem3,
		ikp_delt_mem4		=> ikp_delt_mem4,
		ikp_stdy_mem1		=> ikp_stdy_mem1,
		ikp_stdy_mem2		=> ikp_stdy_mem2,
		ikp_stdy_mem3		=> ikp_stdy_mem3,
		ikp_stdy_mem4		=> ikp_stdy_mem4,
		-- ANFIS module control signals
		anfis_rdy		=> fis_ready,
		start_anfis		=> dma_start
	);

anfis_comp: component anfis_ctrl
	port map (
		aclk			=> aclk,
		aresetn			=> aresetn(0),
		fis_clk			=> fis_clk,
		resetn			=> fis_resetn(0),
		axi_wr_enable		=> wr_enable,
		cur			=> fis_cur,
		cur_delta		=> fis_dcur,
		ind			=> fis_iref,
		ind_delta		=> fis_diref,
		mem_wr_enable		=> dma_wr_enable,
                    vkp_delt_mem1         => vkp_delt_mem1,
                    vkp_delt_mem2         => vkp_delt_mem2,
                    vkp_delt_mem3         => vkp_delt_mem3,
                    vkp_delt_mem4         => vkp_delt_mem4,
                    vkp_stdy_mem1         => vkp_stdy_mem1,
                    vkp_stdy_mem2         => vkp_stdy_mem2,
                    vkp_stdy_mem3         => vkp_stdy_mem3,
                    vkp_stdy_mem4          => vkp_stdy_mem4,
                    vki_delt_mem1         => vki_delt_mem1,
                    vki_delt_mem2         => vki_delt_mem2,
                    vki_delt_mem3         => vki_delt_mem3,
                    vki_delt_mem4         => vki_delt_mem4,
                    vki_stdy_mem1         => vki_stdy_mem1,
                    vki_stdy_mem2         => vki_stdy_mem2,
                    vki_stdy_mem3         => vki_stdy_mem3,
                    vki_stdy_mem4         => vki_stdy_mem4,
                    iki_delt_mem1         => iki_delt_mem1,
                    iki_delt_mem2         => iki_delt_mem2,
                    iki_delt_mem3         => iki_delt_mem3,
                    iki_delt_mem4         => iki_delt_mem4,
                    iki_stdy_mem1          => iki_stdy_mem1,
                    iki_stdy_mem2         => iki_stdy_mem2,
                    iki_stdy_mem3         => iki_stdy_mem3,
                    iki_stdy_mem4         => iki_stdy_mem4,
                    ikp_delt_mem1         => ikp_delt_mem1,
                    ikp_delt_mem2         => ikp_delt_mem2,
                    ikp_delt_mem3         => ikp_delt_mem3,
                    ikp_delt_mem4         => ikp_delt_mem4,
                    ikp_stdy_mem1         => ikp_stdy_mem1,
                    ikp_stdy_mem2         => ikp_stdy_mem2,
                    ikp_stdy_mem3         => ikp_stdy_mem3,
                    ikp_stdy_mem4         => ikp_stdy_mem4,
                    axi_start             => fis_start,
                    dma_start             => dma_start,
                    anfis_rdy             => fis_ready,
                    anfis_valid           => fis_valid,
                    vkp_data              => vkp_data,
                    vki_data              => vki_data,
                    ikp_data              => ikp_data,
                    iki_data              => iki_data,
                    vkp_addr              => vkp_addr,
                    vkp_cons              => vkp_cons,
                    vki_addr              => vki_addr,
                    vki_cons              => vki_cons,
                    ikp_addr              => ikp_addr,
                    ikp_cons              => ikp_cons,
                    iki_addr              => iki_addr,
                    iki_cons              => iki_cons
          );

vkp_lut: component pv_vkp_rom
          port map (
                    clock    => fis_clk,
                    addr     => vkp_addr,
                    data     => vkp_cons
```

```
        );

    vki_lut: component pv_vki_rom
        port map (
            clock   => fis_clk,
            addr    => vki_addr,
            data    => vki_cons
        );

    ikp_lut: component pv_ikp_rom
        port map (
            clock   => fis_clk,
            addr    => ikp_addr,
            data    => ikp_cons
        );

    iki_lut: component pv_iki_rom
        port map (
            clock   => fis_clk,
            addr    => iki_addr,
            data    => iki_cons
        );

    hw_controller: component hw_ctrl
        port map (
            clk             => fis_clk,
            resetn          => fis_resetn(0),
            aclk            => aclk,
            aresetn         => aresetn(0),
            mode            => op_mode,
            gain_valid      => fis_valid,
            vkp             => vkp_data,
            vki             => vki_data,
            ikp             => ikp_data,
            iki             => iki_data,
            meas_valid      => adc_meas_valid,
            volt            => adc_volt,
            cur             => adc_ind,
            iref_en         => iref_en,
            iref_fpga       => iref_fpga,
            pwm_out         => pwm_out
        );

end architecture struct;
```

Appendix D.2 – adc_interface.vhd

```
library ieee;
    use ieee.std_logic_1164.all;
library der_controller_lib;
    use der_controller_lib.components.all;
    use der_controller_lib.der_controller_utils.all;

entity adc_interface is
```

```
        port (
                clock           : in std_logic;
                resetn          : in std_logic;
                -- Output to the voltage ADC
                adc_sdo         : in std_logic;
                adc_sck         : out std_logic;
                adc_conv        : out std_logic;
                -- CPU interrupt
                cpu_intrpt      : out std_logic;
                -- Raw data output bus
                rdat_valid      : out std_logic;
                r_volt_data     : out std_logic_vector(15 downto 0);
                r_ind_data      : out std_logic_vector(11 downto 0);
                r_load_data     : out std_logic_vector(11 downto 0);
                volt_data       : out std_logic_vector(31 downto 0);
                ind_data        : out std_logic_vector(31 downto 0);
                load_data       : out std_logic_vector(31 downto 0)
        );
end entity adc_interface;

architecture struct of adc_interface is

        constant CUR_WIDTH          : integer := 12;
        constant VOLT_WIDTH         : integer := 16;
        constant AV_SAMPLES         : integer := 8;

        constant IND_CONV           : std_logic_vector(31 downto 0) := x"3a800000";
        constant LOAD_CONV          : std_logic_vector(31 downto 0) := x"3a000000";
        constant VOLT_CONV          : std_logic_vector(31 downto 0) := x"3a480000";

        signal xadc_addr            : std_logic_vector(6 downto 0);
        signal xadc_data            : std_logic_vector(15 downto 0);
        signal xadc_den             : std_logic;
        signal xadc_drdy            : std_logic;
        signal xadc_eoc             : std_logic;
        signal xadc_valid           : std_logic;
        signal xadc_ready           : std_logic;
        signal reset                : std_logic;

        signal r_ind_data_buf       : std_logic_vector(r_ind_data'range);
        signal r_load_data_buf      : std_logic_vector(r_load_data'range);
        signal r_volt_data_buf      : std_logic_vector(r_volt_data'range);
        signal r_ind_data_av        : std_logic_vector(r_ind_data'range);
        signal r_volt_data_av       : std_logic_vector(r_volt_data'range);
        signal r_ind_data_pad       : std_logic_vector(31 downto 0);
        signal r_load_data_pad      : std_logic_vector(31 downto 0);
        signal r_volt_data_pad      : std_logic_vector(31 downto 0);

        signal av_valid             : std_logic;
        signal valid_reg            : std_logic_vector(2 downto 0);
        signal ind_valid            : std_logic;
        signal ind_ready            : std_logic;
        signal load_valid           : std_logic;
        signal ld_ready             : std_logic;
        signal volt_valid           : std_logic;
```

```
	signal volt_ready		: std_logic;

	signal intrpt_shift		: std_logic_vector(4 downto 0);

begin

	-- active high rest for the XADC
	reset <= not resetn;

	-- Instantiate the XADC component
	i_xadc: component xadc_device
		port map (
			daddr_in	=> xadc_addr,
			den_in		=> xadc_den,
			di_in		=> (others => '0'),
			dwe_in		=> '0',
			do_out		=> xadc_data,
			drdy_out	=> xadc_drdy,
			dclk_in		=> clock,
			reset_in	=> reset,
			vauxp0		=> '1',
			vauxn0		=> '1',
			vauxp8		=> '1',
			vauxn8		=> '1',
			busy_out	=> open,
			channel_out	=> open,
			eoc_out		=> xadc_eoc,
			eos_out		=> open,
			alarm_out	=> open,
			vp_in		=> '0',
			vn_in		=> '0'
		);

	i_xadc_intfc: component xadc_intfc
		port map (
			clock		=> clock,
			resetn		=> resetn,
			eoc		=> xadc_eoc,
			den		=> xadc_den,
			drdy		=> xadc_drdy,
			addr		=> xadc_addr,
			data		=> xadc_data,
			xvalid		=> xadc_valid,
			xready		=> xadc_ready,
			ind_data	=> r_ind_data_buf,
			load_data	=> r_load_data_buf
		);

	i_vadc: component ad7983_intfc
		port map (
			clk		=> clock,
			resetn		=> resetn,
			start_conv	=> xadc_eoc,
			read_data	=> r_volt_data_buf,
			adc_sdo		=> adc_sdo,
                adc_sclk_o     => adc_sck,
                adc_cnv_o      => adc_conv
        );

-- Perform the averaging
i_ind_av: component moving_av
        generic map(
                DATA_WIDTH          => CUR_WIDTH,
                NUM_SAMPLES         => AV_SAMPLES
         )
         port map (
                clock          => clock,
                resetn         => resetn,
                en             => xadc_eoc,
                data           => r_ind_data_buf,
                av_valid       => av_valid,
                av_data        => r_ind_data_av
         );

i_volt_av: component moving_av
        generic map (
                DATA_WIDTH          => VOLT_WIDTH,
                NUM_SAMPLES         => AV_SAMPLES
        )
        port map (
                clock          => clock,
                resetn         => resetn,
                en             => xadc_eoc,
                data           => r_volt_data_buf,
                av_valid       => open,
                av_data        => r_volt_data_av
        );

-- Assign the output of the XADC for transmision
r_ind_data_pad         <= x"00000" & r_ind_data_av;
r_load_data_pad        <= x"00000" & r_load_data_buf;
r_ind_data             <= r_ind_data_av;
r_load_data            <= r_load_data_buf;

-- Assign the output of the volt ADC for transmission
r_volt_data_pad        <= x"0000" & r_volt_data_av;
r_volt_data            <= r_volt_data_av;

-- Convert the raw ADC values into floating point numbers
i_ind_conv: component adc_to_float
        port map(
                aclk                    => clock,
                aresetn                 => resetn,
                s_axis_a_tdata          => r_ind_data_pad,
                s_axis_a_tready         => ind_ready,
                s_axis_a_tvalid         => av_valid,
                s_axis_b_tdata          => IND_CONV,
                m_axis_result_tdata     => ind_data,
                m_axis_result_tvalid    => ind_valid
        );
```

```
i_load_conv: component adc_to_float
        port map(
                aclk                    => clock,
                aresetn                 => resetn,
                s_axis_a_tdata          => r_load_data_pad,
                s_axis_a_tready         => ld_ready,
                s_axis_a_tvalid         => xadc_valid,
                s_axis_b_tdata          => LOAD_CONV,
                m_axis_result_tdata     => load_data,
                m_axis_result_tvalid    => load_valid
        );

i_volt_conv: component adc_to_float
        port map(
                aclk                    => clock,
                aresetn                 => resetn,
                s_axis_a_tdata          => r_volt_data_pad,
                s_axis_a_tready         => volt_ready,
                s_axis_a_tvalid         => av_valid,
                s_axis_b_tdata          => VOLT_CONV,
                m_axis_result_tdata     => volt_data,
                m_axis_result_tvalid    => volt_valid
        );

xadc_ready <= ld_ready and ind_ready and volt_ready;

-- Assert the valid output signal when all ADC conversions are completed
set_valid:
process (clock) is
begin
        if rising_edge(clock) then
                if (resetn = '0') then
                        valid_reg <= (others => '0');
                else
                        if (and_reduce(valid_reg) = '1') then
                                valid_reg <= (others => '0');
                        else
                                -- Latch the ADC output valid signals
                                if (ind_valid = '1') then
                                        valid_reg(0)    <= '1';
                                end if;
                                if (load_valid = '1') then
                                        valid_reg(1)    <= '1';
                                end if;
                                if (volt_valid = '1') then
                                        valid_reg(2)    <= '1';
                                end if;
                        end if;
                end if;
        end if;
end process set_valid;

rdat_valid <= and_reduce(valid_reg);
```

```
	-- Generate the CPU interrupt once every 4 ADC samples
	cpu_interrupt:
	process (clock) is
	begin
		if rising_edge(clock) then
			if (resetn = '0') then
				intrpt_shift(0) <= '1';
				intrpt_shift(intrpt_shift'high downto 1) <= (others => '0');
			elsif (and_reduce(valid_reg) = '1') then
				intrpt_shift(0) <= intrpt_shift(intrpt_shift'high);
				for i in intrpt_shift'high downto 1 loop
					intrpt_shift(i) <= intrpt_shift(i - 1);
				end loop;
			end if;
		end if;
	end process cpu_interrupt;

	cpu_intrpt <= intrpt_shift(intrpt_shift'high);

end architecture struct;
```

Appendix D.3 – xadc_intfc.vhd

```
library ieee;
	use ieee.std_logic_1164.all;

entity xadc_intfc is
	port (
		clock		: in std_logic;
		resetn		: in std_logic;
		-- Interface to the XADC
		eoc		: in std_logic;
		den		: out std_logic;
		drdy		: in std_logic;
		addr		: out std_logic_vector(6 downto 0);
		data		: in std_logic_vector(15 downto 0);
		-- Stream the data out
		xvalid		: out std_logic;
		xready		: in std_logic;
		ind_data	: out std_logic_vector(11 downto 0);
		load_data	: out std_logic_vector(11 downto 0)
	);
end entity xadc_intfc;

architecture rtl of xadc_intfc is

	constant IND_ADDR		: std_logic_vector(6 downto 0) := "0010000";
	constant LOAD_ADDR		: std_logic_vector(6 downto 0) := "0011000";

	type xadc_type is (idle, wait_adc1, read_adc2, wait_adc2, handshake);
	signal xadc_intfc_state : xadc_type;

begin

	read_xadc:
```

```
process (clock) is
begin
if rising_edge(clock) then
        if (resetn = '0') then
                den                     <= '0';
                xvalid                  <= '0';
                xadc_intfc_state        <= idle;
                ind_data                <= (others => '0');
                load_data               <= (others => '0');
        else
                case xadc_intfc_state is
                when wait_adc1 =>
                        -- Wait for valid data on the bus
                        den <= '0';
                        if (drdy = '1') then
                                ind_data                <= data(15 downto 4);
                                xadc_intfc_state        <= read_adc2;
                        end if;

                when read_adc2 =>
                        -- Read the next ADC
                        den                     <= '1';
                        addr                    <= LOAD_ADDR;
                        xadc_intfc_state        <= wait_adc2;

                when wait_adc2 =>
                        -- Wait for valid data on the bus
                        den <= '0';
                        if (drdy = '1') then
                                load_data               <= data(15 downto 4);
                                xadc_intfc_state        <= handshake;
                        end if;

                when handshake =>
                        -- Output the data with handshaking
                        if (xready = '1') then
                                xvalid                  <= '1';
                                xadc_intfc_state        <= idle;
                        end if;

                when others=>
                        -- Idle - wait for a valid sample
                        xvalid  <= '0';
                        if (eoc = '1') then
                                den                     <= '1';
                                addr                    <= IND_ADDR;
                                xadc_intfc_state        <= wait_adc1;
                        else
                                den <= '0';
                        end if;
                end case;
        end if;
end if;
end process read_xadc;
end architecture rtl;
```

Appendix D.4 – ad7983_intfc.v

```
`timescale 1ns/1ns

module ad7983_intfc
(
                /* clock and reset signals */
                input clk,                      // 150MHz AXI clock
                input resetn,                   // active low reset signal

                /* Interface to the FPGA core */
                input start_conv,               // Pulsed high to start a conversion
                output [15:0] read_data,        //data read from the ADC

                /* Interface to the ADC */
                input adc_sdo,                  //ADC SDO signal
                output adc_sclk_o,              //ADC serial clock
                output adc_cnv_o                //ADC CNV signal
);

        /* ADC states */
        parameter ADC_IDLE_STATE                = 3'b001;
        parameter ADC_START_CNV_STATE           = 3'b010;
        parameter ADC_DELAY_STATE1              = 3'b100;
        parameter ADC_DELAY_STATE2              = 3'b101;
        parameter ADC_DELAY_STATE3              = 3'b110;
        parameter ADC_READ_CNV_RESULT           = 3'b111;

        /* ADC FSM registers */
        reg [2:0] adc_state;
        reg [2:0] adc_next_state;

        /* ADC conv pulse time - gives 500 ns with 150MHz input */
        parameter [6:0] ADC_CNV_CNT = 75;

        /* Conversion pulse generation register */
        reg [6:0] adc_tcnv_cnt;

        /* ADC serial clock generation parameters */
        parameter [2:0] SCLK_CNT = 3;
        parameter [2:0] EDGE_CNT = 2;
        parameter [2:0] SAMP_CNT = 1;
        parameter ADC_SCLK_PERIODS = 5'd16;

        /* Serial clock generaiton & control regsiters */
        reg sample;
        reg sclk_clk;
        reg [2:0] sclk_clk_cnt;
        wire adc_clk_en;

        /* Sampling registers */
        reg [15:0] adc_data_s;
        reg [4:0] samp_cnt;
```

```
/* Internal buffering & averaging registers */
reg [1:0] av_shift [3:0];
reg [3:0] moving_av;
reg [15:0] filt_data;
reg adc_cnv_s;

/* Assign the outputs */
assign adc_cnv_o = adc_cnv_s;
assign adc_sclk_o = sclk_clk;
assign read_data = adc_data_s;

/* Counter for the conversion pulse */
always @(posedge clk)
begin
	if (resetn == 1'b0)
	begin
		adc_tcnv_cnt  <= ADC_CNV_CNT;
	end
	else
	begin
		if(adc_state == ADC_START_CNV_STATE && adc_tcnv_cnt != 0)
		begin
			adc_tcnv_cnt <= adc_tcnv_cnt - 1;
		end
		else
		begin
			adc_tcnv_cnt <= ADC_CNV_CNT;
		end
	end
end

/* Counter for the clock generation */
always @(posedge clk)
begin
	if (resetn == 1'b0)
	begin
		sample		<= 1'b0;
		sclk_clk	<= 1'b0;
		sclk_clk_cnt	<= SCLK_CNT;
	end
	else if (adc_clk_en == 1'b1)
	begin
		if (sclk_clk_cnt == EDGE_CNT)
		begin
			sample		<= 1'b0;
			sclk_clk	<= 1'b1;
			sclk_clk_cnt	<= sclk_clk_cnt - 1;
		end
		else if (sclk_clk_cnt == SAMP_CNT)
		begin
			sample		<= 1'b1;
			sclk_clk_cnt	<= sclk_clk_cnt - 1;
		end
		else if (sclk_clk_cnt < 1)
```

```
			begin
				sample		<= 1'b0;
				sclk_clk		<= 1'b0;
				sclk_clk_cnt	<= SCLK_CNT;
			end
			else
			begin
				sample		<= 1'b0;
				sclk_clk_cnt	<= sclk_clk_cnt - 1;
			end
		end
		else
		begin
			sample		<= 1'b0;
			sclk_clk		<= 1'b0;
			sclk_clk_cnt	<= SCLK_CNT;
		end
	end

	assign adc_clk_en = ((adc_state == ADC_READ_CNV_RESULT) && (samp_cnt !=
					0)) ? 1'b1 : 1'b0;

	/* Sample the data from the ADC */
	always @ (posedge clk)
	begin
		if (resetn == 1'b0)
		begin
			adc_data_s	<= 16'h00;
			samp_cnt	<= ADC_SCLK_PERIODS;
		end
		else if (sample == 1'b1 && adc_state == ADC_READ_CNV_RESULT &&
			samp_cnt > 0)
		begin
			adc_data_s	<= {adc_data_s[14:0], adc_sdo};
			samp_cnt	<= samp_cnt - 1;
		end
		else if (adc_state == ADC_START_CNV_STATE)
		begin
			samp_cnt <= ADC_SCLK_PERIODS;
		end
	end

	/* Drive the conv signal according to the current state */
	always @(posedge clk)
	begin
		if (resetn == 1'b0)
		begin
			adc_cnv_s	<= 1'b0;
			adc_state	<= ADC_IDLE_STATE;
		end
		else
		begin
			adc_state <= adc_next_state;
			case (adc_state)
				ADC_START_CNV_STATE:
```

```
                /* Drive the conv signal high during conversion phase */
                begin
                        adc_cnv_s <= 1'b1;
                end
                default:
                /* In all other states drive the conv signal low */
                begin
                        adc_cnv_s <= 1'b0;
                end
            endcase
    end
end

/* Next state control for the FSM */
always @(adc_state, start_conv, adc_tcnv_cnt, samp_cnt)
begin
    adc_next_state <= adc_state;
    case (adc_state)
            ADC_START_CNV_STATE:
            /* Wait for the conversion phase to complete */
            begin
                if(adc_tcnv_cnt == 0)
                begin
                        adc_next_state <= ADC_DELAY_STATE1;
                end
            end
            ADC_DELAY_STATE1:
            /* Delay to allow the ADC to set its first value */
            begin
                adc_next_state <= ADC_DELAY_STATE2;
            end
            ADC_DELAY_STATE2:
            /* Delay to allow the ADC to set its first value */
            begin
                adc_next_state <= ADC_DELAY_STATE3;
            end
            ADC_DELAY_STATE3:
            /* Delay to allow the ADC to set its first value */
            begin
                adc_next_state <= ADC_READ_CNV_RESULT;
            end
            ADC_READ_CNV_RESULT:
            /* Wait for the aquisition phase to complete */
            begin
                if(samp_cnt == 0)
                begin
                        adc_next_state <= ADC_IDLE_STATE;
                end
            end
            default:
            /* Idle state, wait for a conversion to be initiated */
            begin
                if (start_conv == 1'b1)
                begin
                        adc_next_state <= ADC_START_CNV_STATE;
                        end
                    end
                endcase
            end

endmodule
```

Appendix D.5 – moving_average.vhd

```
library ieee;
        use ieee.std_logic_1164.all;
        use ieee.numeric_std.all;
        use ieee.math_real.all;

entity moving_av is
        generic (
                DATA_WIDTH          : integer := 12;
                NUM_SAMPLES         : integer := 8
        );
        port (
                clock           : in std_logic;
                resetn          : in std_logic;
                en              : in std_logic;
                data            : in std_logic_vector(DATA_WIDTH-1 downto 0);
                av_valid        : out std_logic;
                av_data         : out std_logic_vector(DATA_WIDTH-1 downto 0)

        );
end entity moving_av;

architecture rtl of moving_av is

 constant DIV_BITS    : integer := integer(log2(real(NUM_SAMPLES)));
 constant SUM_BITS  : integer := DATA_WIDTH + DIV_BITS;

 type samp_array_t is array(NUM_SAMPLES-1 downto 0) of
        std_logic_vector(DATA_WIDTH-1 downto 0);

 signal samp_array     : samp_array_t;
 signal sum_ptr         : integer range 0 to NUM_SAMPLES-1;
 signaL sum_reg         : std_logic_vector(SUM_BITS-1 downto 0);
 signal sum_comb       : unsigned(SUM_BITS-1 downto 0);
 signal sum_done        : std_logic;
 signal sum_hist         : std_logic;

begin


        sum_proc:
        process (clock) is
        begin
                if rising_edge(clock) then
                        if (resetn = '0') then
                                sum_ptr         <= 0;
                                sum_done        <= '0';
                                sum_reg         <= (others => '0');
```

```
                                        samp_array     <= (others => (others => '0'));
                              else
                                        if (en = '1') then
                                                  -- Kick off a new summer process by adding
                                                  -- the new data sample to the array
                                                  samp_array(0) <= data;
                                                  for i in samp_array'high downto 1 loop
                                                            samp_array(i) <= samp_array(i-1);
                                                  end loop;
                                                  -- Reset the sequential summer registers
                                                  sum_done       <= '0';
                                                  sum_ptr        <= NUM_SAMPLES-1;
                                                  sum_reg        <= (others => '0');
                                        elsif (sum_ptr > 0) then
                                                  -- Perform accumulation until all samples are done
                                                  sum_reg <= std_logic_vector(sum_comb);
                                                  sum_ptr <= sum_ptr - 1;
                                        else
                                                  -- Flag that the average is done
                                                  sum_done <= '1';
                                        end if;
                              end if;
                    end if;
          end process sum_proc;

          -- Perform the sumamtion combinatorially
          sum_comb <= unsigned(sum_reg) + unsigned(samp_array(sum_ptr));

          -- Register the output when valie
          out_reg:
          process (clock) is
          begin
          if rising_edge(clock) then
                    if (resetn = '0') then
                              av_valid       <= '0';
                              sum_hist       <= '0';
                              av_data        <= (others => '0');
                    else
                              -- Register to detect an edge
                              sum_hist <= sum_done;
                              -- Set outputs when new sum is complete
                              if (sum_hist = '0' and sum_done = '1') then
                                        av_valid       <= '1';
                                        av_data        <= std_logic_vector(sum_comb(SUM_BITS-1
                                                                 downto DIV_BITS));
                                        else
                                                  av_valid <= '0';
                                        end if;
                              end if;
                    end if;
          end process out_reg;

end architecture rtl;
```

Appendix D.6 – dma_interface.vhd

```
library ieee;
	use ieee.std_logic_1164.all;

entity dma_interface is
	port (
		s_axis_aclk		: in std_logic;
		s_axis_aresetn		: in std_logic;
		-- AXI Stream Signals
		s_axis_tvalid		: in std_logic;
		s_axis_tready		: out std_logic;
		s_axis_tdata		: in std_logic_vector(127 downto 0);
		s_axis_tlast		: in std_logic;
		-- Membership registers
		wr_enable		: out std_logic;
		vkp_stdy_mem1		: out std_logic_vector(31 downto 0);
		vkp_stdy_mem2		: out std_logic_vector(31 downto 0);
		vkp_stdy_mem3		: out std_logic_vector(31 downto 0);
		vkp_stdy_mem4		: out std_logic_vector(31 downto 0);
		vkp_delt_mem1		: out std_logic_vector(31 downto 0);
		vkp_delt_mem2		: out std_logic_vector(31 downto 0);
		vkp_delt_mem3		: out std_logic_vector(31 downto 0);
		vkp_delt_mem4		: out std_logic_vector(31 downto 0);
		vki_stdy_mem1		: out std_logic_vector(31 downto 0);
		vki_stdy_mem2		: out std_logic_vector(31 downto 0);
		vki_stdy_mem3		: out std_logic_vector(31 downto 0);
		vki_stdy_mem4		: out std_logic_vector(31 downto 0);
		vki_delt_mem1		: out std_logic_vector(31 downto 0);
		vki_delt_mem2		: out std_logic_vector(31 downto 0);
		vki_delt_mem3		: out std_logic_vector(31 downto 0);
		vki_delt_mem4		: out std_logic_vector(31 downto 0);
		ikp_stdy_mem1		: out std_logic_vector(31 downto 0);
		ikp_stdy_mem2		: out std_logic_vector(31 downto 0);
		ikp_stdy_mem3		: out std_logic_vector(31 downto 0);
		ikp_stdy_mem4		: out std_logic_vector(31 downto 0);
		ikp_delt_mem1		: out std_logic_vector(31 downto 0);
		ikp_delt_mem2		: out std_logic_vector(31 downto 0);
		ikp_delt_mem3		: out std_logic_vector(31 downto 0);
		ikp_delt_mem4		: out std_logic_vector(31 downto 0);
		iki_stdy_mem1		: out std_logic_vector(31 downto 0);
		iki_stdy_mem2		: out std_logic_vector(31 downto 0);
		iki_stdy_mem3		: out std_logic_vector(31 downto 0);
		iki_stdy_mem4		: out std_logic_vector(31 downto 0);
		iki_delt_mem1		: out std_logic_vector(31 downto 0);
		iki_delt_mem2		: out std_logic_vector(31 downto 0);
		iki_delt_mem3		: out std_logic_vector(31 downto 0);
		iki_delt_mem4		: out std_logic_vector(31 downto 0);
		-- ANFIS module control signals
		anfis_rdy		: in std_logic;
		start_anfis		: out std_logic
	);
end entity dma_interface;

architecture rtl of dma_interface is
```

```
        type dma_state is (     vkp_stdy_reg, vkp_delt_reg, vki_stdy_reg, vki_delt_reg,
                                ikp_stdy_reg, ikp_delt_reg, iki_stdy_reg, iki_delt_reg,
                                error, idle);

        signal dma_fsm          : dma_state;
        signal kick_fis          : std_logic;

begin

        dma_state_machine:
        process (s_axis_aclk) is
        begin
        if rising_edge(s_axis_aclk) then
                if (s_axis_aresetn = '0') then
                        s_axis_tready           <= '0';
                        wr_enable               <= '0';
                        kick_fis                <= '0';
                        dma_fsm                 <= idle;
                        vkp_stdy_mem1           <= (others => '0');
                        vkp_stdy_mem2           <= (others => '0');
                        vkp_stdy_mem3           <= (others => '0');
                        vkp_stdy_mem4           <= (others => '0');
                        vkp_delt_mem1           <= (others => '0');
                        vkp_delt_mem2           <= (others => '0');
                        vkp_delt_mem3           <= (others => '0');
                        vkp_delt_mem4           <= (others => '0');
                        vki_stdy_mem1           <= (others => '0');
                        vki_stdy_mem2           <= (others => '0');
                        vki_stdy_mem3           <= (others => '0');
                        vki_stdy_mem4           <= (others => '0');
                        vki_delt_mem1           <= (others => '0');
                        vki_delt_mem2           <= (others => '0');
                        vki_delt_mem3           <= (others => '0');
                        vki_delt_mem4           <= (others => '0');
                        ikp_stdy_mem1           <= (others => '0');
                        ikp_stdy_mem2           <= (others => '0');
                        ikp_stdy_mem3           <= (others => '0');
                        ikp_stdy_mem4           <= (others => '0');
                        ikp_delt_mem1           <= (others => '0');
                        ikp_delt_mem2           <= (others => '0');
                        ikp_delt_mem3           <= (others => '0');
                        ikp_delt_mem4           <= (others => '0');
                        iki_stdy_mem1           <= (others => '0');
                        iki_stdy_mem2           <= (others => '0');
                        iki_stdy_mem3           <= (others => '0');
                        iki_stdy_mem4           <= (others => '0');
                        iki_delt_mem1           <= (others => '0');
                        iki_delt_mem2           <= (others => '0');
                        iki_delt_mem3           <= (others => '0');
                        iki_delt_mem4           <= (others => '0');
                else
                        case dma_fsm is
                        when vkp_stdy_reg =>
                                if (s_axis_tvalid = '1') then
                vkp_stdy_mem1        <= s_axis_tdata(31 downto 0);
                vkp_stdy_mem2        <= s_axis_tdata(63 downto 32);
                vkp_stdy_mem3        <= s_axis_tdata(95 downto 64);
                vkp_stdy_mem4        <= s_axis_tdata(127 downto 96);
                dma_fsm              <= vkp_delt_reg;
        end if;

when vkp_delt_reg =>
        if (s_axis_tvalid = '1') then
                vkp_delt_mem1        <= s_axis_tdata(31 downto 0);
                vkp_delt_mem2        <= s_axis_tdata(63 downto 32);
                vkp_delt_mem3        <= s_axis_tdata(95 downto 64);
                vkp_delt_mem4        <= s_axis_tdata(127 downto 96);
                dma_fsm              <= vki_stdy_reg;
        end if;

when vki_stdy_reg =>
        if (s_axis_tvalid = '1') then
                vki_stdy_mem1        <= s_axis_tdata(31 downto 0);
                vki_stdy_mem2        <= s_axis_tdata(63 downto 32);
                vki_stdy_mem3        <= s_axis_tdata(95 downto 64);
                vki_stdy_mem4        <= s_axis_tdata(127 downto 96);
                dma_fsm              <= vki_delt_reg;
        end if;

when vki_delt_reg =>
        if (s_axis_tvalid = '1') then
                vki_delt_mem1        <= s_axis_tdata(31 downto 0);
                vki_delt_mem2        <= s_axis_tdata(63 downto 32);
                vki_delt_mem3        <= s_axis_tdata(95 downto 64);
                vki_delt_mem4        <= s_axis_tdata(127 downto 96);
                dma_fsm              <= ikp_stdy_reg;
                        end if;

when ikp_stdy_reg =>
        if (s_axis_tvalid = '1') then
                ikp_stdy_mem1        <= s_axis_tdata(31 downto 0);
                ikp_stdy_mem2        <= s_axis_tdata(63 downto 32);
                ikp_stdy_mem3        <= s_axis_tdata(95 downto 64);
                ikp_stdy_mem4        <= s_axis_tdata(127 downto 96);
                dma_fsm              <= ikp_delt_reg;
        end if;

when ikp_delt_reg =>
        if (s_axis_tvalid = '1') then
                ikp_delt_mem1        <= s_axis_tdata(31 downto 0);
                ikp_delt_mem2        <= s_axis_tdata(63 downto 32);
                ikp_delt_mem3        <= s_axis_tdata(95 downto 64);
                ikp_delt_mem4        <= s_axis_tdata(127 downto 96);
                dma_fsm              <= iki_stdy_reg;
        end if;

when iki_stdy_reg =>
        if (s_axis_tvalid = '1') then
                iki_stdy_mem1        <= s_axis_tdata(31 downto 0);
```

```
                        iki_stdy_mem2           <= s_axis_tdata(63 downto 32);
                        iki_stdy_mem3           <= s_axis_tdata(95 downto 64);
                        iki_stdy_mem4           <= s_axis_tdata(127 downto 96);
                        dma_fsm                 <= iki_delt_reg;
                    end if;

                when iki_delt_reg =>
                    if (s_axis_tvalid = '1') then
                        iki_delt_mem1<= s_axis_tdata(31 downto 0);
                        iki_delt_mem2<= s_axis_tdata(63 downto 32);
                        iki_delt_mem3<= s_axis_tdata(95 downto 64);
                        iki_delt_mem4<= s_axis_tdata(127 downto 96);
                        -- Either move onto error mode or finish & wait for anfis
                        if (s_axis_tlast = '1') then
                            kick_fis        <= '1';
                            wr_enable       <= '1';
                            s_axis_tready <= '0';
                            dma_fsm         <= idle;
                        else
                            dma_fsm <= error;
                        end if;
                    end if;

                when error =>
                    -- Error with data synchronisation, wait for stream to end
                    if (s_axis_tlast = '1') then
                        s_axis_tready <= '0';
                        dma_fsm         <= idle;
                    end if;

                when others =>
                    -- ie Idle state, wait for a transaction to begin
                    kick_fis <= '0';
                    if (s_axis_tvalid = '1') then
                        s_axis_tready <= '1';
                        dma_fsm         <= vkp_stdy_reg;
                    end if;
                end case;
        end if;
end if;
end process dma_state_machine;

anfis_hand_shake:
process (s_axis_aclk) is
begin
        if rising_edge(s_axis_aclk) then
                if (s_axis_aresetn = '0') then
                    start_anfis <= '0';
                else
                    if (kick_fis = '1') then
                        start_anfis <= '1';
                    elsif (anfis_rdy = '1') then
                        start_anfis <= '0';
                    end if;
                end if;
```

```
		end if;
	end process anfis_hand_shake;

end architecture rtl;
```

Appendix D.7 – anfis_ctrl.vhd

```
library ieee;
	use ieee.std_logic_1164.all;
library der_controller_lib;
	use der_controller_lib.der_controller_utils.all;
	use der_controller_lib.components.all;

entity anfis_ctrl is
	port (
		-- Axi clock signal (125MHz)
		aclk			: in std_logic;
		aresetn			: in std_logic;
		-- FIS clock domain signals
		fis_clk			: in std_logic;
		resetn			: in std_logic;
		-- Data values from AXI stream
		axi_wr_enable		: in std_logic;
		cur			: in std_logic_vector(31 downto 0);
		cur_delta		: in std_logic_vector(31 downto 0);
		ind			: in std_logic_vector(31 downto 0);
		ind_delta		: in std_logic_vector(31 downto 0);
		-- Membership registers from the DMA
		mem_wr_enable		: in std_logic;
		vkp_stdy_mem1		: in std_logic_vector(31 downto 0);
		vkp_stdy_mem2		: in std_logic_vector(31 downto 0);
		vkp_stdy_mem3		: in std_logic_vector(31 downto 0);
		vkp_stdy_mem4		: in std_logic_vector(31 downto 0);
		vkp_delt_mem1		: in std_logic_vector(31 downto 0);
		vkp_delt_mem2		: in std_logic_vector(31 downto 0);
		vkp_delt_mem3		: in std_logic_vector(31 downto 0);
		vkp_delt_mem4		: in std_logic_vector(31 downto 0);
		vki_stdy_mem1		: in std_logic_vector(31 downto 0);
		vki_stdy_mem2		: in std_logic_vector(31 downto 0);
		vki_stdy_mem3		: in std_logic_vector(31 downto 0);
		vki_stdy_mem4		: in std_logic_vector(31 downto 0);
		vki_delt_mem1		: in std_logic_vector(31 downto 0);
		vki_delt_mem2		: in std_logic_vector(31 downto 0);
		vki_delt_mem3		: in std_logic_vector(31 downto 0);
		vki_delt_mem4		: in std_logic_vector(31 downto 0);
		ikp_stdy_mem1		: in std_logic_vector(31 downto 0);
		ikp_stdy_mem2		: in std_logic_vector(31 downto 0);
		ikp_stdy_mem3		: in std_logic_vector(31 downto 0);
		ikp_stdy_mem4		: in std_logic_vector(31 downto 0);
		ikp_delt_mem1		: in std_logic_vector(31 downto 0);
		ikp_delt_mem2		: in std_logic_vector(31 downto 0);
		ikp_delt_mem3		: in std_logic_vector(31 downto 0);
		ikp_delt_mem4		: in std_logic_vector(31 downto 0);
		iki_stdy_mem1		: in std_logic_vector(31 downto 0);
		iki_stdy_mem2		: in std_logic_vector(31 downto 0);
		iki_stdy_mem3		: in std_logic_vector(31 downto 0);
		iki_stdy_mem4		: in std_logic_vector(31 downto 0);
		iki_delt_mem1		: in std_logic_vector(31 downto 0);
		iki_delt_mem2		: in std_logic_vector(31 downto 0);
		iki_delt_mem3		: in std_logic_vector(31 downto 0);
		iki_delt_mem4		: in std_logic_vector(31 downto 0);
		-- ANFIS control signals
		axi_start		: in std_logic;
		dma_start		: in std_logic;
		anfis_rdy		: out std_logic;
		anfis_valid		: out std_logic;
		vkp_data		: out std_logic_vector(31 downto 0);
		vki_data		: out std_logic_vector(31 downto 0);
		ikp_data		: out std_logic_vector(31 downto 0);
		iki_data		: out std_logic_vector(31 downto 0);
		-- External ROM LUT connections
		vkp_addr		: out std_logic_vector(5 downto 0);
		vkp_cons		: in std_logic_vector(95 downto 0);
		vki_addr		: out std_logic_vector(5 downto 0);
		vki_cons		: in std_logic_vector(95 downto 0);
		ikp_addr		: out std_logic_vector(5 downto 0);
		ikp_cons		: in std_logic_vector(95 downto 0);
		iki_addr		: out std_logic_vector(5 downto 0);
		iki_cons		: in std_logic_vector(95 downto 0)
	);
end entity anfis_ctrl;

architecture rtl of anfis_ctrl is

	-- CDC signals
	signal axi_en_sync		: std_logic;
	signal dma_en_sync		: std_logic;
	signal fis_start		: std_logic;
	signal fis_start_sync		: std_logic;
	signal fis_start_valid		: std_logic;
	signal fis_rdy			: std_logic;
	signal fis_ready		: std_logic;
	signal vkp_rdy			: std_logic;
	signal vki_rdy			: std_logic;
	signal ikp_rdy			: std_logic;
	signal iki_rdy			: std_logic;
	signal vkp_valid		: std_logic;
	signal vki_valid		: std_logic;
	signal ikp_valid		: std_logic;
	signal iki_valid		: std_logic;
	signal fis_valid_reg		: std_logic_vector(3 downto 0);
	signal anfis_valid_buf : std_logic;

	-- Membership data registers
	signal vkp_stdy_mems		: member_array(3 downto 0);
	signal vkp_delt_mems		: member_array(3 downto 0);
	signal vki_stdy_mems		: member_array(3 downto 0);
	signal vki_delt_mems		: member_array(3 downto 0);
	signal ikp_stdy_mems		: member_array(3 downto 0);
```

```
        signal ikp_delt_mems            : member_array(3 downto 0);
        signal iki_stdy_mems            : member_array(3 downto 0);
        signal iki_delt_mems            : member_array(3 downto 0);
        signal vkp_stdy_map             : std_logic_vector(3 downto 0);
        signal vkp_delt_map             : std_logic_vector(3 downto 0);
        signal vki_stdy_map             : std_logic_vector(3 downto 0);
        signal vki_delt_map             : std_logic_vector(3 downto 0);
        signal ikp_stdy_map             : std_logic_vector(3 downto 0);
        signal ikp_delt_map             : std_logic_vector(3 downto 0);
        signal iki_stdy_map             : std_logic_vector(3 downto 0);
        signal iki_delt_map             : std_logic_vector(3 downto 0);

        -- Raw data registers
        signal cur_reg                  : std_logic_vector(31 downto 0);
        signal di_reg                   : std_logic_vector(31 downto 0);
        signal ind_reg                  : std_logic_vector(31 downto 0);
        signal dind_reg                 : std_logic_vector(31 downto 0);

        -- temp output registers
        signal vkp_value                : std_logic_vector(31 downto 0);
        signal vki_value                : std_logic_vector(31 downto 0);
        signal ikp_value                : std_logic_vector(31 downto 0);
        signal iki_value                : std_logic_vector(31 downto 0);

begin

        --------------------------------------------------------------------------
        -- Synchronise the control signals to the ANFIS clock
        --------------------------------------------------------------------------

        sync_ready: component single_bit_cdc
                port map (
                        clock   => aclk,
                        resetn  => aresetn,
                        d_in    => fis_ready,
                        sync_d  => anfis_rdy
                );

        sync_axi_start: component single_bit_cdc
                port map (
                        clock   => fis_clk,
                        resetn  => resetn,
                        d_in    => fis_start,
                        sync_d  => fis_start_sync
                );

        fis_start <= dma_start and axi_start;

        sync_axi_wr: component pulse_cdc
                port map (
                        clock_a         => aclk,
                        reseta          => aresetn,
                        clock_b         => fis_clk,
                        resetb          => resetn,
                        d_in            => axi_wr_enable,
		d_out		=> axi_en_sync
	);

sync_dma_wr: component pulse_cdc
	port map (
		clock_a		=> aclk,
		reseta		=> aresetn,
		clock_b		=> fis_clk,
		resetb		=> resetn,
		d_in		=> mem_wr_enable,
		d_out		=> dma_en_sync
	);

---------------------------------------------------------------------------
-- Register the inputs from the AXI bus interfaces
---------------------------------------------------------------------------

vkp_register_members:
process (fis_clk) is
begin
if rising_edge(fis_clk) then
	if (resetn = '0') then
		vkp_stdy_map<= (others => '0');
		vkp_delt_map <= (others => '0');
		vkp_stdy_mems <= (others => (others => '0'));
		vkp_delt_mems <= (others => (others => '0'));
	elsif (dma_en_sync = '1') then
		-- Register the kp current values
		vkp_stdy_mems(3)	<= vkp_stdy_mem4;
		vkp_stdy_map(3)		<= or_reduce(vkp_stdy_mem4);
		vkp_stdy_mems(2)	<= vkp_stdy_mem3;
		vkp_stdy_map(2)		<= or_reduce(vkp_stdy_mem3);
		vkp_stdy_mems(1)	<= vkp_stdy_mem2;
		vkp_stdy_map(1)		<= or_reduce(vkp_stdy_mem2);
		vkp_stdy_mems(0)	<= vkp_stdy_mem1;
		vkp_stdy_map(0)		<= or_reduce(vkp_stdy_mem1);
		-- Register the kp delt current values
		vkp_delt_mems(3)	<= vkp_delt_mem4;
		vkp_delt_map(3)		<= or_reduce(vkp_delt_mem4);
		vkp_delt_mems(2)	<= vkp_delt_mem3;
		vkp_delt_map(2)		<= or_reduce(vkp_delt_mem3);
		vkp_delt_mems(1)	<= vkp_delt_mem2;
		vkp_delt_map(1)		<= or_reduce(vkp_delt_mem2);
		vkp_delt_mems(0)	<= vkp_delt_mem1;
		vkp_delt_map(0)		<= or_reduce(vkp_delt_mem1);
	end if;
end if;
end process vkp_register_members;

vki_register_members:
process (fis_clk) is
begin
if rising_edge(fis_clk) then
	if (resetn = '0') then
		vki_stdy_map		<= (others => '0');
```

```
                vki_delt_map            <= (others => '0');
                vki_stdy_mems           <= (others => (others => '0'));
                vki_delt_mems           <= (others => (others => '0'));
        elsif (dma_en_sync = '1') then
                -- Register the ki current values
                vki_stdy_mems(3)        <= vki_stdy_mem4;
                vki_stdy_map(3)         <= or_reduce(vki_stdy_mem4);
                vki_stdy_mems(2)        <= vki_stdy_mem3;
                vki_stdy_map(2)         <= or_reduce(vki_stdy_mem3);
                vki_stdy_mems(1)        <= vki_stdy_mem2;
                vki_stdy_map(1)         <= or_reduce(vki_stdy_mem2);
                vki_stdy_mems(0)        <= vki_stdy_mem1;
                vki_stdy_map(0)         <= or_reduce(vki_stdy_mem1);
                -- Register the ki delt current values
                vki_delt_mems(3)        <= vki_delt_mem4;
                vki_delt_map(3)         <= or_reduce(vki_delt_mem4);
                vki_delt_mems(2)        <= vki_delt_mem3;
                vki_delt_map(2)         <= or_reduce(vki_delt_mem3);
                vki_delt_mems(1)        <= vki_delt_mem2;
                vki_delt_map(1)         <= or_reduce(vki_delt_mem2);
                vki_delt_mems(0)        <= vki_delt_mem1;
                vki_delt_map(0)         <= or_reduce(vki_delt_mem1);
        end if;
end if;
end process vki_register_members;

ikp_register_members:
process (fis_clk) is
begin
if rising_edge(fis_clk) then
        if (resetn = '0') then
                ikp_stdy_map <= (others => '0');
                ikp_delt_map  <= (others => '0');
                ikp_stdy_mems <= (others => (others => '0'));
                ikp_delt_mems <= (others => (others => '0'));
        elsif (dma_en_sync = '1') then
                -- Register the kp inductor values
                ikp_stdy_mems(3)        <= ikp_stdy_mem4;
                ikp_stdy_map(3)         <= or_reduce(ikp_stdy_mem4);
                ikp_stdy_mems(2)        <= ikp_stdy_mem3;
                ikp_stdy_map(2)         <= or_reduce(ikp_stdy_mem3);
                ikp_stdy_mems(1)        <= ikp_stdy_mem2;
                ikp_stdy_map(1)         <= or_reduce(ikp_stdy_mem2);
                ikp_stdy_mems(0)        <= ikp_stdy_mem1;
                ikp_stdy_map(0)         <= or_reduce(ikp_stdy_mem1);
                -- Register the kp delt inductor values
                ikp_delt_mems(3)        <= ikp_delt_mem4;
                ikp_delt_map(3)         <= or_reduce(ikp_delt_mem4);
                ikp_delt_mems(2)        <= ikp_delt_mem3;
                ikp_delt_map(2)         <= or_reduce(ikp_delt_mem3);
                ikp_delt_mems(1)        <= ikp_delt_mem2;
                ikp_delt_map(1)         <= or_reduce(ikp_delt_mem2);
                ikp_delt_mems(0)        <= ikp_delt_mem1;
                ikp_delt_map(0)         <= or_reduce(ikp_delt_mem1);
        end if;
end if;
end process ikp_register_members;

iki_register_members:
process (fis_clk) is
begin
if rising_edge(fis_clk) then
        if (resetn = '0') then
                iki_stdy_map  <= (others => '0');
                iki_delt_map  <= (others => '0');
                iki_stdy_mems <= (others => (others => '0'));
                iki_delt_mems <= (others => (others => '0'));
`       elsif (dma_en_sync = '1') then
                -- Register the ki inductor values
                iki_stdy_mems(3)      <= iki_stdy_mem4;
                iki_stdy_map(3)       <= or_reduce(iki_stdy_mem4);
                iki_stdy_mems(2)      <= iki_stdy_mem3;
                iki_stdy_map(2)       <= or_reduce(iki_stdy_mem3);
                iki_stdy_mems(1)      <= iki_stdy_mem2;
                iki_stdy_map(1)       <= or_reduce(iki_stdy_mem2);
                iki_stdy_mems(0)      <= iki_stdy_mem1;
                iki_stdy_map(0)       <= or_reduce(iki_stdy_mem1);
                -- Register the ki delt inductor values
                iki_delt_mems(3)      <= iki_delt_mem4;
                iki_delt_map(3)       <= or_reduce(iki_delt_mem4);
                iki_delt_mems(2)      <= iki_delt_mem3;
                iki_delt_map(2)       <= or_reduce(iki_delt_mem3);
                iki_delt_mems(1)      <= iki_delt_mem2;
                iki_delt_map(1)       <= or_reduce(iki_delt_mem2);
                iki_delt_mems(0)      <= iki_delt_mem1;
                iki_delt_map(0)       <= or_reduce(iki_delt_mem1);
        end if;
end if;
end process iki_register_members;

register_data:
process (fis_clk) is
begin
if rising_edge(fis_clk) then
        if (resetn = '0') then
                cur_reg         <= (others => '0');
                di_reg          <= (others => '0');
                ind_reg         <= (others => '0');
                dind_reg        <= (others => '0');
        elsif (axi_en_sync = '1') then
                -- Register the outputs from the AXI bus
                cur_reg         <= cur;
                di_reg          <= cur_delta;
                ind_reg         <= ind;
                dind_reg        <= ind_delta;
        end if;
end if;
end process register_data;

---------------------------------------------------------------------------
```

```
-- The start signal needs to be delayed by a clock to align the data
-- correctly, use an SR flip flop for this
---------------------------------------------------------------------------
fis_start_reg:
process (fis_clk) is
begin
        if rising_edge(fis_clk) then
                if (resetn = '0') then
                        fis_start_valid <= '0';
                elsif (fis_start_sync = '1') then
                        fis_start_valid <= '1';
                elsif (fis_ready = '0') then
                        fis_start_valid <= '0';
                end if;
        end if;
end process fis_start_reg;

---------------------------------------------------------------------------
-- Anfis engine
---------------------------------------------------------------------------

vkp_anfis: component anfis_engine
        port map (
                clock                   => fis_clk,
                resetn                  => resetn,
                steady                  => cur_reg,
                delta                   => di_reg,
                steady_member           => vkp_stdy_mems,
                steady_mem_map          => vkp_stdy_map,
                delta_member            => vkp_delt_mems,
                delta_mem_map           => vkp_delt_map,
                in_valid                => fis_start_valid,
                seq                     => vkp_addr,
                cons_data               => vkp_cons,
                fis_data                => vkp_value,
                fis_ready               => vkp_rdy,
                fis_valid               => vkp_valid
        );

vki_anfis: component anfis_engine
        port map (
                clock                   => fis_clk,
                resetn                  => resetn,
                steady                  => cur_reg,
                delta                   => di_reg,
                steady_member           => vki_stdy_mems,
                steady_mem_map          => vki_stdy_map,
                delta_member            => vki_delt_mems,
                delta_mem_map           => vki_delt_map,
                in_valid                => fis_start_valid,
                seq                     => vki_addr,
                cons_data               => vki_cons,
                fis_data                => vki_value,
                fis_ready               => vki_rdy,
                fis_valid               => vki_valid
```

```
	);
ikp_anfis: component anfis_engine
	port map (
		clock			=> fis_clk,
		resetn			=> resetn,
		steady			=> ind_reg,
		delta			=> dind_reg,
		steady_member		=> ikp_stdy_mems,
		steady_mem_map		=> ikp_stdy_map,
		delta_member		=> ikp_delt_mems,
		delta_mem_map		=> ikp_delt_map,
		in_valid		=> fis_start_valid,
		seq			=> ikp_addr,
		cons_data		=> ikp_cons,
		fis_data		=> ikp_value,
		fis_ready		=> ikp_rdy,
		fis_valid		=> ikp_valid
	);

iki_anfis: component anfis_engine
	port map (
		clock			=> fis_clk,
		resetn			=> resetn,
		steady			=> ind_reg,
		delta			=> dind_reg,
		steady_member		=> iki_stdy_mems,
		steady_mem_map		=> iki_stdy_map,
		delta_member		=> iki_delt_mems,
		delta_mem_map		=> iki_delt_map,
		in_valid		=> fis_start_valid,
		seq			=> iki_addr,
		cons_data		=> iki_cons,
		fis_data		=> iki_value,
		fis_ready		=> iki_rdy,
		fis_valid		=> iki_valid
	);

fis_ready <= vkp_rdy and iki_rdy and ikp_rdy and iki_rdy and fis_start_sync;

---------------------------------------------------------------------------
-- As the valid signals are set at different times,
-- SR flip flop must be used to produce the module output
---------------------------------------------------------------------------
fis_valid_register:
process (fis_clk) is
begin
if rising_edge(fis_clk) then
	if (resetn = '0') then
		vkp_data	<= (others => '0');
		vki_data	<= (others => '0');
		ikp_data	<= (others => '0');
		iki_data	<= (others => '0');
		fis_valid_reg	<= (others => '0');
	elsif (anfis_valid_buf = '1') then
		fis_valid_reg <= (others => '0');
```

```
                        else
                                if (vkp_valid = '1') then
                                        vkp_data                <= vkp_value;
                                        fis_valid_reg(0)        <= '1';
                                end if;
                                if (vki_valid = '1') then
                                        vki_data                <= vki_value;
                                        fis_valid_reg(1)        <= '1';
                                end if;
                                if (ikp_valid = '1') then
                                        ikp_data                <= ikp_value;
                                        fis_valid_reg(2)        <= '1';
                                end if;
                                if (iki_valid = '1') then
                                        iki_data                <= iki_value;
                                        fis_valid_reg(3)        <= '1';
                                end if;
                        end if;
                end if;
                end process fis_valid_register;

                anfis_valid_buf <= and_reduce(fis_valid_reg);
                anfis_valid       <= anfis_valid_buf;

end architecture rtl;
```

Appendix D.8 - single_bit_cdc.vhd

```
library ieee;
   use ieee.std_logic_1164.all;

entity single_bit_cdc is
   port (
      clock   : in std_logic;
      resetn  : in std_logic;
      d_in    : in std_logic;
      sync_d  : out std_logic
   );
end entity single_bit_cdc;

architecture rtl of single_bit_cdc is

   signal d_in_bis : std_logic;

begin

   double_sync:
   process (clock) is
   begin
      if rising_edge(clock) then
         if (resetn = '0') then
            d_in_bis    <= '0';
            sync_d      <= '0';
            else
                d_in_bis    <= d_in;
                sync_d      <= d_in_bis;
            end if;
        end if;
    end process double_sync;

end architecture rtl;
```

Appendix D.9 – pulse_cdc.vhd

```
library ieee;
    use ieee.std_logic_1164.all;

entity pulse_cdc is
    port (
        clock_a : in std_logic;
        reseta  : in std_logic;
        clock_b : in std_logic;
        resetb  : in std_logic;
        d_in    : in std_logic;
        d_out   : out std_logic
    );
end entity pulse_cdc;

architecture rtl of pulse_cdc is

    signal clka_reg : std_logic;
    signal clkb_reg : std_logic_vector(2 downto 0);

begin

    -- Clock domain A toggle flip flop
    domain_a_reg:
    process (clock_a) is
    begin
        if rising_edge(clock_a) then
            if (reseta = '0') then
                clka_reg  <= '0';
            elsif (d_in = '1') then
                clka_reg  <= not clka_reg;
            end if;
        end if;
    end process domain_a_reg;

    -- cdc synchronisation
    domain_b_reg:
    process (clock_b) is
    begin
        if rising_edge(clock_b) then
            if (resetb = '0') then
                clkb_reg  <= (others => '0');
            else
                clkb_reg(0) <= clka_reg;
            clkb_reg(2 downto 1)  <= clkb_reg(1 downto 0);
         end if;
      end if;
   end process domain_b_reg;

   -- Simple logic to generate the output pulse
   d_out <= clkb_reg(2) xor clkb_reg(1);

end architecture rtl;
```

Appendix D.10 – anfis_ctrl.vhd

```
library ieee;
        use ieee.std_logic_1164.all;
library der_controller_lib;
        use der_controller_lib.der_controller_utils.all;
        use der_controller_lib.components.all;

entity anfis_engine is
        port (
                clock                   : in std_logic;
                resetn                  : in std_logic;
                steady                  : in std_logic_vector(31 downto 0);
                delta                   : in std_logic_vector(31 downto 0);
                steady_member           : in member_array(3 downto 0);
                steady_mem_map          : in std_logic_vector(3 downto 0);
                delta_member            : in member_array(3 downto 0);
                delta_mem_map           : in std_logic_vector(3 downto 0);
                in_valid                : in std_logic;
                seq                     : out std_logic_vector(5 downto 0);
                cons_data               : in std_logic_vector(95 downto 0);
                fis_data                : out std_logic_vector (31 downto 0);
                fis_ready               : out std_logic;
                fis_valid               : out std_logic
        );
end entity anfis_engine;

architecture struct of anfis_engine is

        -- Permutator control and output signals
        signal steady_mem     : std_logic_vector(31 downto 0);
        signal delta_mem      : std_logic_vector(31 downto 0);
        signal perm_valid     : std_logic;
        signal perm_last      : std_logic;

        -- Inference control and output signals
        signal dividend_tdata : std_logic_vector(31 downto 0);
        signal divisor_tdata  : std_logic_vector(31 downto 0);
        signal acc_valid      : std_logic;
        signal acc_last       : std_logic;

begin

        perm: component permutator
```

```
		port map (
			clk			=> clock,
			resetn			=> resetn,
			steady_member		=> steady_member,
			steady_mem_map	=> steady_mem_map,
			delta_member		=> delta_member,
			delta_mem_map		=> delta_mem_map,
			in_valid		=> in_valid,
			perm_ready		=> fis_ready,
			steady_out		=> steady_mem,
			delta_out		=> delta_mem,
			seq			=> seq,
			out_valid		=> perm_valid,
			out_last		=> perm_last
		);

	inf: component inference_wrapper
		port map (
			clock			=> clock,
			resetn			=> resetn,
			steady_data		=> steady,
			delta_data		=> delta,
			rom_data		=> cons_data,
			mem1			=> steady_mem,
			mem2			=> delta_mem,
			in_valid		=> perm_valid,
			in_last			=> perm_last,
			dividend_tdata		=> dividend_tdata,
			divisor_tdata		=> divisor_tdata,
			acc_valid		=> acc_valid,
			acc_last		=> acc_last
		);

	defuzz: component defuzzifier
		port map (
			clock			=> clock,
			dividend_tdata		=> dividend_tdata,
			divisor_tdata		=> divisor_tdata,
			acc_valid		=> acc_valid,
			acc_last		=> acc_last,
			m_axis_result_tdata	=> fis_data,
			m_axis_result_tvalid	=> fis_valid
		);

end architecture struct;
```

Appendix D.11 – Consequence Parameter LUT Example (pv_vkp_rom.vhd)

```
library ieee;
	use ieee.std_logic_1164.all;
library der_controller_lib;
	use der_controller_lib.components.all;
```

```
entity pv_vkp_rom is
        port (
                clock   : in std_logic;
                addr    : in std_logic_vector(5 downto 0);
                data    : out std_logic_vector(95 downto 0)
        );
end entity pv_vkp_rom;

architecture rtl of pv_vkp_rom is

        -- Constants for the ROM decoders
        constant SEQ_ADDR1          : std_logic_vector(5 downto 0)          := "001001";
        constant SEQ_ADDR2          : std_logic_vector(5 downto 0)          := "001010";
        constant SEQ_ADDR3          : std_logic_vector(5 downto 0)          := "001011";
        constant SEQ_ADDR4          : std_logic_vector(5 downto 0)          := "001100";
        constant SEQ_ADDR5          : std_logic_vector(5 downto 0)          := "010001";
        constant SEQ_ADDR6          : std_logic_vector(5 downto 0)          := "010010";
        constant SEQ_ADDR7          : std_logic_vector(5 downto 0)          := "010011";
        constant SEQ_ADDR8          : std_logic_vector(5 downto 0)          := "010100";
        constant SEQ_ADDR9          : std_logic_vector(5 downto 0)          := "011001";
        constant SEQ_ADDR10         : std_logic_vector(5 downto 0)          := "011010";
        constant SEQ_ADDR11         : std_logic_vector(5 downto 0)          := "011011";
        constant SEQ_ADDR12         : std_logic_vector(5 downto 0)          := "011100";
        constant SEQ_ADDR13         : std_logic_vector(5 downto 0)          := "100001";
        constant SEQ_ADDR14         : std_logic_vector(5 downto 0)          := "100010";
        constant SEQ_ADDR15         : std_logic_vector(5 downto 0)          := "100011";
        constant SEQ_ADDR16         : std_logic_vector(5 downto 0)          := "100100";

        constant ROM_ADDR1          : std_logic_vector(3 downto 0)          := "0000";
        constant ROM_ADDR2          : std_logic_vector(3 downto 0)          := "0001";
        constant ROM_ADDR3          : std_logic_vector(3 downto 0)          := "0010";
        constant ROM_ADDR4          : std_logic_vector(3 downto 0)          := "0011";
        constant ROM_ADDR5          : std_logic_vector(3 downto 0)          := "0100";
        constant ROM_ADDR6          : std_logic_vector(3 downto 0)          := "0101";
        constant ROM_ADDR7          : std_logic_vector(3 downto 0)          := "0110";
        constant ROM_ADDR8          : std_logic_vector(3 downto 0)          := "0111";
        constant ROM_ADDR9          : std_logic_vector(3 downto 0)          := "1000";
        constant ROM_ADDR10         : std_logic_vector(3 downto 0)          := "1001";
        constant ROM_ADDR11         : std_logic_vector(3 downto 0)          := "1010";
        constant ROM_ADDR12         : std_logic_vector(3 downto 0)          := "1011";
        constant ROM_ADDR13         : std_logic_vector(3 downto 0)          := "1100";
        constant ROM_ADDR14         : std_logic_vector(3 downto 0)          := "1101";
        constant ROM_ADDR15         : std_logic_vector(3 downto 0)          := "1110";
        constant ROM_ADDR16         : std_logic_vector(3 downto 0)          := "1111";

        signal rom_address : std_logic_vector(3 downto 0);

begin

        with addr select rom_address <=
                ROM_ADDR1 when SEQ_ADDR1,
                ROM_ADDR2 when SEQ_ADDR2,
                ROM_ADDR3 when SEQ_ADDR3,
                ROM_ADDR4 when SEQ_ADDR4,
                ROM_ADDR5 when SEQ_ADDR5,
		ROM_ADDR6 when SEQ_ADDR6,
		ROM_ADDR7 when SEQ_ADDR7,
		ROM_ADDR8 when SEQ_ADDR8,
		ROM_ADDR9 when SEQ_ADDR9,
		ROM_ADDR10 when SEQ_ADDR10,
		ROM_ADDR11 when SEQ_ADDR11,
		ROM_ADDR12 when SEQ_ADDR12,
		ROM_ADDR13 when SEQ_ADDR13,
		ROM_ADDR14 when SEQ_ADDR14,
		ROM_ADDR15 when SEQ_ADDR15,
		(others => '1') when others;

	-- Parameter selection ROM, IP core generated in vivado
	pv_kp_rom: component pv_vkp_lut
		port map(
			clka		=> clock,
			addra		=> rom_address,
			douta		=> data
		);

end architecture rtl;
```

Appendix D.12 – hw_ctrl.vhd

```
library ieee;
	use ieee.std_logic_1164.all;
library der_controller_lib;
	use der_controller_lib.components.all;

entity hw_ctrl is
	port (
		clk		: in std_logic;
		resetn		: in std_logic;
		aclk		: in std_logic;
		aresetn		: in std_logic;
		mode		: in std_logic_vector(1 downto 0);
		gain_valid	: in std_logic;
		vkp		: in std_logic_vector(31 downto 0);
		vki		: in std_logic_vector(31 downto 0);
		ikp		: in std_logic_vector(31 downto 0);
		iki		: in std_logic_vector(31 downto 0);
		meas_valid	: in std_logic;
		volt		: in std_logic_vector(31 downto 0);
		cur		: in std_logic_vector(31 downto 0);
		iref_en		 out std_logic;
		iref_fpga	: out std_logic_vector(31 downto 0);
		pwm_out		: out std_logic
	);
end entity hw_ctrl;

architecture struct of hw_ctrl is

	constant VOLT_VALUE		: std_logic_vector(31 downto 0) := x"42400000";
	constant VOLT_NOISE		: std_logic_vector(31 downto 0) := x"3A9D4952";
```

```
        constant NVOLT_NOISE      : std_logic_vector(31 downto 0) := x"BA9D4952";
        constant CUR_NOISE        : std_logic_vector(31 downto 0) := x"3C23D70A";
        constant NCUR_NOISE       : std_logic_vector(31 downto 0) := x"BC23D70A";

        signal op_mode            : std_logic_vector(1 downto 0);
        signal pi_clear           : std_logic;

        signal meas_valid_sync    : std_logic;
        signal meas_valid_shift   : std_logic_vector(6 downto 0);
        signal pi_valid           : std_logic;

        signal vkp_reg            : std_logic_vector(31 downto 0);
        signal vki_reg            : std_logic_vector(31 downto 0);
        signal ikp_reg            : std_logic_vector(31 downto 0);
        signal iki_reg            : std_logic_vector(31 downto 0);

        signal volt_err           : std_logic_vector(31 downto 0);
        signal v_err_valid        : std_logic;
        signal volt_hi_valid      : std_logic;
        signal comp_volt_hi       : std_logic_vector(7 downto 0);
        alias volt_hi             : std_logic is comp_volt_hi(0);
        signal volt_lo_valid      : std_logic;
        signal comp_volt_lo       : std_logic_vector(7 downto 0);
        alias volt_lo             : std_logic is comp_volt_lo(0);
        signal volt_valid         : std_logic;
        signal volt_err_reg       : std_logic_vector(31 downto 0);
        signal volt_err_lim       : std_logic_vector(31 downto 0);

        signal cur_ref            : std_logic_vector(31 downto 0);
        signal cur_ref_valid      : std_logic;
        signal cur_err            : std_logic_vector(31 downto 0);
        signal c_err_valid        : std_logic;
        signal cur_hi_valid       : std_logic;
        signal comp_cur_hi        : std_logic_vector(7 downto 0);
        alias cur_hi              : std_logic is comp_cur_hi(0);
        signal cur_lo_valid       : std_logic;
        signal comp_cur_lo        : std_logic_vector(7 downto 0);
        alias cur_lo              : std_logic is comp_cur_lo(0);
        signal cur_valid          : std_logic;
        signal cur_err_reg        : std_logic_vector(31 downto 0);
        signal cur_err_lim        : std_logic_vector(31 downto 0);

        signal ctrl_word          : std_logic_vector(31 downto 0);
        signal ctrl_valid         : std_logic;
        signal ctrl_word_norm     : std_logic_vector(15 downto 0);
        signal ctrl_word_valid    : std_logic;

begin

        -- Retime the mode bits
        mode_0_sync: component single_bit_cdc
                port map (
                        clock         => clk,
                        resetn        => resetn,
                        d_in          => mode(0),
		sync_d		=> op_mode(0)
	);

mode_1_sync: component single_bit_cdc
	port map (
		clock		=> clk,
		resetn		=> resetn,
		d_in		=> mode(1),
		sync_d		=> op_mode(1)
	);

pi_clear <= not op_mode(1);

-- Retime the voltage and inductor currents start signal

meas_sync: component pulse_cdc
	port map (
		clock_a		=> aclk,
		reseta		=> aresetn,
		clock_b		=> clk,
		resetb		=> resetn,
		d_in		=> meas_valid,
		d_out		=> meas_valid_sync
	);

-- Register the PI controller gain values
gain_register:
process (clk) is
begin
	if rising_edge(clk) then
		if (resetn = '0') then
			vkp_reg		<= (others => '0');
			vki_reg		<= (others => '0');
			ikp_reg		<= (others => '0');
			iki_reg		<= (others => '0');
		elsif (gain_valid = '1') then
			vkp_reg		 <= vkp;
			vki_reg		 <= vki;
			ikp_reg		<= ikp;
			iki_reg		<= iki;
		end if;
	end if;
end process gain_register;

-- float_sub is a floating point subtractor IP core  which is generated within vivado
volt_error: component float_sub
	port map (
		aclk			=> clk,
		s_axis_a_tvalid		=> meas_valid_sync,
		s_axis_a_tdata		=> VOLT_VALUE,
		s_axis_b_tvalid		=> meas_valid_sync,
		s_axis_b_tdata		=> volt,
		m_axis_result_tvalid	=> v_err_valid,
		m_axis_result_tdata	=> volt_err
	);
```

```
-- Comparators to limit errors caused by noise (improves stability)
-- float_compare component is an IP block generated within vivado
-- which performs a comparison on floating point numbers
volt_hi_comp: component float_compare
        port map (
                aclk                    => clk,
                s_axis_a_tvalid         => v_err_valid,
                s_axis_a_tdata          => volt_err,
                s_axis_b_tvalid         => v_err_valid,
                s_axis_b_tdata          => VOLT_NOISE,
                m_axis_result_tvalid    => volt_hi_valid,
                m_axis_result_tdata     => comp_volt_hi
        );

volt_lo_comp: component float_compare
        port map (
                aclk                    => clk,
                s_axis_a_tvalid         => v_err_valid,
                s_axis_a_tdata          => NVOLT_NOISE,
                s_axis_b_tvalid         => v_err_valid,
                s_axis_b_tdata          => volt_err,
                m_axis_result_tvalid    => volt_lo_valid,
                m_axis_result_tdata     => comp_volt_lo
        );

limit_volt_err:
process (clk) is
begin
        if rising_edge(clk) then
                if (resetn = '0') then
                        volt_valid      <= '0';
                        volt_err_reg    <= (others => '0');
                        volt_err_lim    <= (others => '0');
                else
                        -- Register the volt err from the subtractor
                        if (v_err_valid = '1') then
                                volt_err_reg <= volt_err;
                        end if;
                        -- Ignore the error if it is just noise
                        if (volt_hi_valid = '1') then
                                volt_valid      <= '1';
                                if (volt_hi = '1' or volt_lo = '1') then
                                        volt_err_lim    <= volt_err_reg;
                                else
                                        volt_err_lim    <= (others => '0');
                                end if;
                        else
                                volt_valid      <= '0';
                        end if;
                end if;
        end if;
end process limit_volt_err;

-- Voltage PI Controller
```

```
voltage_pi: component pi_ctrl
	port map (
		clk		=> clk,
		resetn		=> resetn,
		clear		=> pi_clear,
		kp		=> vkp_reg,
		kp_valid	=> volt_valid,
		ki		=> vki_reg,
		ki_valid	=> volt_valid,
		err		=> volt_err_lim,
		err_valid	=> volt_valid,
		ctrl_out	=> cur_ref,
		ctrl_valid	=> cur_ref_valid
	);

-- Current Error
cur_error: component float_sub
	port map (
		aclk			=> clk,
		s_axis_a_tvalid		=> cur_ref_valid,
		s_axis_a_tdata		=> cur_ref,
		s_axis_b_tvalid		=> cur_ref_valid,
		s_axis_b_tdata		=> cur,
		m_axis_result_tvalid	=> c_err_valid,
		m_axis_result_tdata	=> cur_err
	);

-- Comparators to limit errors caused by noise (improves stability)
cur_hi_comp: component float_compare
	port map (
		aclk			=> clk,
		s_axis_a_tvalid		=> c_err_valid,
		s_axis_a_tdata		=> cur_err,
		s_axis_b_tvalid		=> c_err_valid,
		s_axis_b_tdata		=> CUR_NOISE,
		m_axis_result_tvalid	=> cur_hi_valid,
		m_axis_result_tdata	=> comp_cur_hi
	);

cur_lo_comp: component float_compare
	port map (
		aclk			=> clk,
		s_axis_a_tvalid		=> c_err_valid,
		s_axis_a_tdata		=> NCUR_NOISE,
		s_axis_b_tvalid		=> c_err_valid,
		s_axis_b_tdata		=> cur_err,
		m_axis_result_tvalid	=> cur_lo_valid,
		m_axis_result_tdata	=> comp_cur_lo
	);

limit_cur_err:
process (clk) is
begin
	if rising_edge(clk) then
		if (resetn = '0') then
```

```
                cur_valid       <= '0';
                cur_err_reg     <= (others => '0');
                cur_err_lim     <= (others => '0');
            else
                -- Register the volt err from the subtractor
                if (c_err_valid = '1') then
                    cur_err_reg <= cur_err;
                end if;
                -- Ignore the error if it is just noise
                if (cur_hi_valid = '1') then
                    cur_valid <= '1';
                    if (cur_hi = '1' or cur_lo = '1') then
                        cur_err_lim <= cur_err_reg;
                    else
                        cur_err_lim <= (others => '0');
                    end if;
                else
                    cur_valid <= '0';
                end if;
            end if;
        end if;
end process limit_cur_err;

-- Current PI Controller
current_pi: component pi_ctrl
        port map (
            clk             => clk,
            resetn          => resetn,
            clear           => pi_clear,
            kp              => ikp_reg,
            kp_valid        => cur_valid,
            ki              => iki_reg,
            ki_valid        => cur_valid,
            err             => cur_err_lim,
            err_valid       => cur_valid,
            ctrl_out        => ctrl_word,
            ctrl_valid      => ctrl_valid
        );

pwm_norm: component normaliser_wrapper
        port map (
            aclk                    => clk,
            pi_output               => ctrl_word,
            pi_valid                => ctrl_valid,
            normalised_out_tdata    => ctrl_word_norm,
            normalised_out_tvalid   => ctrl_word_valid
        );

pwm: component pwm_generator
        port map (
            clk             => clk,
            resetn          => resetn,
            mode            => op_mode,
            pwm_ctrl        => ctrl_word_norm(11 downto 0),
            pwm_wr          => ctrl_word_valid,
                    pwm_out         => pwm_out
                );

        -- Set the iref output
        iref_fpga       <= cur_ref;
        iref_en         <= cur_ref_valid;

end architecture struct;
```

Appendix D.13 – pi_ctrl.vhd

```
library ieee;
        use ieee.std_logic_1164.all;
library der_controller_lib;
        use der_controller_lib.components.all;
        use der_controller_lib.float_support.all;

entity pi_ctrl is
        generic (
                PI_MAX          : real    := 10.0
        );
        port (
                clk             : in std_logic;
                resetn          : in std_logic;
                clear           : in std_logic;
                kp              : in std_logic_vector(31 downto 0);
                kp_valid        : in std_logic;
                ki              : in std_logic_vector(31 downto 0);
                ki_valid        : in std_logic;
                err             : in std_logic_vector(31 downto 0);
                err_valid       : in std_logic;
                ctrl_out        : out std_logic_vector(31 downto 0);
                ctrl_valid      : out std_logic
        );
end entity pi_ctrl;

architecture struct of pi_ctrl is

constant MAX_OUT   : std_logic_vector(31 downto 0) := std_logic_vector(to_float(PI_MAX));
constant ZERO_FL   : std_logic_vector(31 downto 0) := std_logic_vector(to_float(0.0));

function and_reduce(a : std_logic_vector) return std_logic is
        variable res            : std_logic := '1';
begin
        for i in a'range loop
                res := res and a(i);
        end loop;
        return res;
end function and_reduce;

signal ki_res           : std_logic_vector(31 downto 0);
signal ki_res_valid     : std_logic;
signal kp_res           : std_logic_vector(31 downto 0);
alias kp_sign           : std_logic is kp_res(31);
```

```
signal kp_res_valid     : std_logic;
signal is_nan           : std_logic;

signal comp_long        : std_logic_vector(7 downto 0);
alias comp              : std_logic is comp_long(0);
signal comp_valid       : std_logic;

signal int_data         : std_logic_vector(31 downto 0);
signal int_valid        : std_logic;

signal err_valid_reg    : std_logic;
signal kp_valid_reg     : std_logic;
signal ki_res_reg       : std_logic_vector(31 downto 0);
alias ki_sign           : std_logic is ki_res_reg(31);
signal err_reg          : std_logic_vector(31 downto 0);
signal kp_reg           : std_logic_vector(31 downto 0);

begin

        -- Instantiate the multiply-add IP core as generated in vivado
        ki_fma: component fma
                port map (
                        aclk                    => clk,
                        s_axis_a_tvalid         => ki_valid,
                        s_axis_a_tdata          => ki,
                        s_axis_b_tvalid         => err_valid,
                        s_axis_b_tdata          => err,
                        s_axis_c_tvalid         => ki_valid,
                        s_axis_c_tdata          => int_data,
                        m_axis_result_tvalid    => ki_res_valid,
                        m_axis_result_tdata     => ki_res
                );

        -- KI FMA Output registering
        ki_fma_register:
        process (clk) is
        begin
                if rising_edge(clk) then
                        if (resetn = '0') then
                                ki_res_reg <= (others => '0');
                        else
                                if (ki_res_valid = '1') then
                                        ki_res_reg <= ki_res;
                                end if;
                        end if;
                end if;
        end process ki_fma_register;

        -- Instantiate the multiply-add IP core as generated in vivado
        kp_fma : component fma
                port map (
                        aclk                    => clk,
                        s_axis_a_tvalid         => kp_valid_reg,
                        s_axis_a_tdata          => kp_reg,
                        s_axis_b_tvalid         => err_valid_reg,
			s_axis_b_tdata		=> err_reg,
			s_axis_c_tvalid		=> ki_res_valid,
			s_axis_c_tdata		=> ki_res,
			m_axis_result_tvalid	=> kp_res_valid,
			m_axis_result_tdata	=> kp_res
		);

	-- Process to align the valid signals
	valid_ctrl:
	process (clk) is
	begin
		if rising_edge(clk) then
			if (resetn = '0' or clear = '1') then
				err_reg		<= (others => '0');
				kp_reg		<= (others => '0');
				err_valid_reg	<= '0';
				kp_valid_reg	<= '0';
			else
				if (err_valid = '1') then
					err_reg		<= err;
					err_valid_reg	<= '1';
				elsif (ki_res_valid = '1') then
					err_valid_reg	<= '0';
				end if;
				if (kp_valid = '1') then
					kp_reg		<= kp;
					kp_valid_reg	<= '1';
				elsif (ki_res_valid = '1') then
					kp_valid_reg	 <= '0';
				end if;
			end if;
		end if;
	end process valid_ctrl;

	-- Instantiate the floating point compaire IP core  as generated in vivado
	comparator: component float_compare
		port map (
			aclk			=> clk,
			s_axis_a_tvalid		=> kp_res_valid,
			s_axis_a_tdata		=> kp_res,
			s_axis_b_tvalid		=> kp_res_valid,
			s_axis_b_tdata		=> MAX_OUT,
			m_axis_result_tvalid	=> comp_valid,
			m_axis_result_tdata	=> comp_long
		);

	is_nan <= and_reduce(kp_res(30 downto 23));

	-- Select the value for the conditional integration
	cond_int:
	process (clk) is
	begin
		if rising_edge(clk) then
			if (resetn = '0' or clear = '1') then
				int_data <= ZERO_FL;
```

```
                elsif (comp_valid = '1') then
                        if (comp = '0' and is_nan = '0' and ki_sign = '0') then
                                int_data <= ki_res_reg;
                        elsif (ki_sign = '1') then
                                int_data <= ZERO_FL;
                        end if;
                end if;
        end if;
    end process cond_int;
    -- Set the output, latching it if neccessary
    output_reg:
    process (clk) is
    begin
        if rising_edge(clk) then
                if (resetn = '0' or clear = '1') then
                        ctrl_valid      <= '0';
                        ctrl_out        <= (others => '0');
                elsif (comp_valid = '1') then
                        ctrl_valid <= '1';
                        if (comp = '1' or is_nan = '1') then
                                ctrl_out <= MAX_OUT;
                        elsif (kp_sign = '1') then
                                ctrl_out <= ZERO_FL;
                        else
                                ctrl_out <= kp_res;
                        end if;
                else
                        ctrl_valid <= '0';
                end if;
        end if;
    end process output_reg;

end architecture struct;
```

Appdenx D.14 – pwm_generator.vhd

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;
library der_controller_lib;
    use der_controller_lib.components.all;
    use der_controller_lib.der_controller_utils.all;

entity pwm_generator is
    port (
        clk             : in std_logic;
        resetn          : in std_logic;
        mode            : in std_logic_vector(1 downto 0);
        pwm_ctrl        : in std_logic_vector(11 downto 0);
        pwm_wr          : in std_logic;
        pwm_out         : out std_logic
    );
end entity pwm_generator;
```

```
architecture rtl of pwm_generator is

        signal pwm_wr_sync  : std_logic;
        signal pwm_ctrl_sync : std_logic_vector(pwm_ctrl'range);
        signal pwm_ctrl_reg  : std_logic_vector(pwm_ctrl'range);
        signal not_zero      : std_logic;
        signal ctrl_high     : std_logic;

        signal op_mode_bis   : std_logic_vector(mode'range);
        signal op_mode       : std_logic_vector(mode'range);

        signal adder1        : std_logic_vector(11 downto 0) := (others => '0');
        signal add1_reg      : std_logic_vector(4 downto 0) := (others => '0');
        signal add1_msb      : std_logic_vector(6 downto 0) := (others => '0');
        signal adder2        : std_logic_vector(4 downto 0) := (others => '0');
        signal add2_lsb      : std_logic_vector(2 downto 0) := (others => '0');
        signal add2_msb      : std_logic_vector(1 downto 0) := (others => '0');
        signal adder3        : std_logic_vector(1 downto 0) := (others => '0');
        signal duty_ref      : std_logic_vector(8 downto 0) := (others => '0');

        alias count_ref is duty_ref(8 downto 4);
        alias sclk_ref is duty_ref(3 downto 2);
        alias fclk_ref is duty_ref(1 downto 0);

        signal d1_clk0       : std_logic;
        signal d1_clk90      : std_logic;
        signal d1_clk180     : std_logic;
        signal d1_clk270     : std_logic;
        signal d1_clk2       : std_logic;
        signal d1_clk4       : std_logic;

        signal dll_1_0       : std_logic;
        signal dll_1_90      : std_logic;
        signal dll_1_180     : std_logic;
        signal dll_1_270     : std_logic;
        signal dll_1_clk     : std_logic;

        signal dll_1_0_n     : std_logic;
        signal dll_1_90_n    : std_logic;
        signal dll_1_180_n   : std_logic;
        signal dll_1_270_n   : std_logic;
        signal dll_1_clk_n   : std_logic;

        signal d2_clk0       : std_logic;
        signal d2_clk90      : std_logic;
        signal d2_clk180     : std_logic;
        signal d2_clk270     : std_logic;
        signal d2_clk2       : std_logic;

        signal dll_2_0       : std_logic;
        signal dll_2_90      : std_logic;
        signal dll_2_180     : std_logic;
        signal dll_2_270     : std_logic;
```

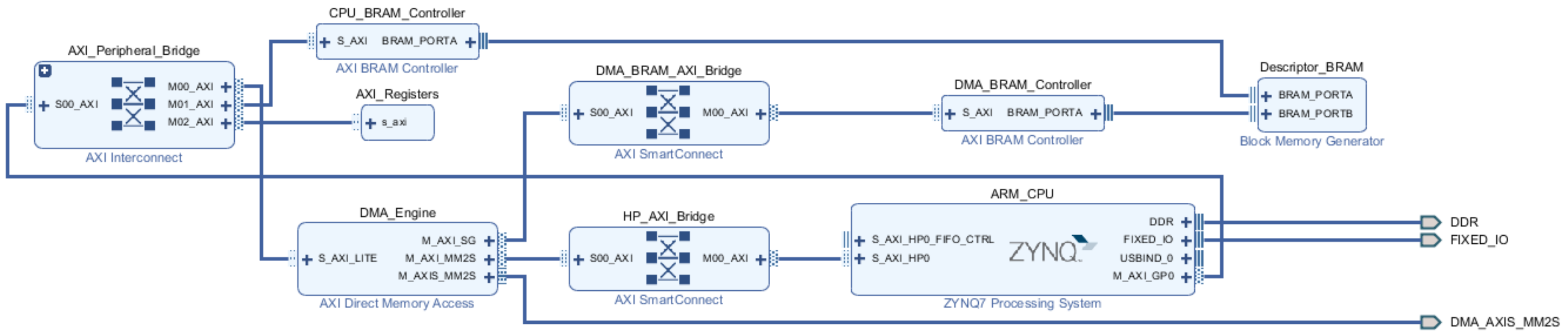
```
	signal cnt_comp		: std_logic;
	signal dcm_pulse	: std_logic;
	signal cnt_zero		: std_logic;
	signal pwm_cnt		: unsigned(4 downto 0) := (others => '0');

	signal rst_q		: std_logic := '0';
	signal rst_qbar		: std_logic := '1';
	signal q		: std_logic := '0';
	signal qbar		: std_logic := '1';

	signal reset_bis	: std_logic;
	signal resetn_sync	: std_logic;

begin

	------------------------------------------------------------------------------
	-- Synchronise the reset
	------------------------------------------------------------------------------

	synchronise_reset:
	process(d1_clk0) is
	begin
		if rising_edge(d1_clk0) then
			reset_bis		<= resetn;
			resetn_sync <= reset_bis;
		end if;
	end process synchronise_reset;

	------------------------------------------------------------------------------
	-- Retime the inputs
	------------------------------------------------------------------------------

	sync_wr: component pulse_cdc
		port map (
			clock_a		=> clk,
			reseta		=> resetn,
			clock_b		=> d1_clk0,
			resetb		=> resetn_sync,
			d_in		=> pwm_wr,
			d_out		=> pwm_wr_sync
		);

	ctrl_word_reg:
	process (clk) is
	begin
		if rising_edge(clk) then
			if (pwm_wr = '1') then
				pwm_ctrl_reg <= pwm_ctrl;
			end if;
		end if;
	end process ctrl_word_reg;

	ctrl_word_sync:
	process (d1_clk0) is
	begin
```

```
		if rising_edge(d1_clk0) then
			if (resetn_sync = '0' or op_mode = "00") then
				not_zero		<= '0';
				ctrl_high		<= '0';
				pwm_ctrl_sync		<= (others => '0');
			elsif (op_mode = "01") then
				not_zero		<= '1';
				ctrl_high		<= '0';
				pwm_ctrl_sync		<= x"800";
			elsif (pwm_wr_sync = '1') then
				not_zero		<= or_reduce(pwm_ctrl);
				ctrl_high		<= and_reduce(pwm_ctrl);
				pwm_ctrl_sync		<= pwm_ctrl_reg;
			end if;
		end if;
	end process ctrl_word_sync;

	mode_sync:
	process (d1_clk0) is
	begin
		if rising_edge(d1_clk0) then
			if (resetn_sync = '0') then
				op_mode_bis	<= (others => '0');
				op_mode		<= (others => '0');
			else
				op_mode_bis	<= mode;
				op_mode		<= op_mode_bis;
			end if;
		end if;
	end process mode_sync;

	-------------------------------------------------------------------------------
	-- Sigma Delta MASH modulator circuit
	-------------------------------------------------------------------------------

	mash_modulator:
	process (d1_clk0) is
		variable add_u		: unsigned(12 downto 0);
		variable add2_u		: unsigned(5 downto 0);
	begin
	if rising_edge(d1_clk0) then
		if (resetn_sync = '0' or op_mode = "00") then
			adder1		<= (others => '0');
			add1_reg	<= (others => '0');
			add1_msb	<= (others => '0');
			adder2		<= (others => '0');
			add2_lsb	<= (others => '0');
			add2_msb	<= (others => '0');
			adder3		<= (others => '0');
		elsif (cnt_zero = '1') then
			-- stage one adder
			add_u := to_unsigned(to_integer(unsigned(pwm_ctrl_sync)) +
					to_integer(unsigned(add1_reg)), 13);
			if (add_u(add_u'high) = '1') then
				adder1 <= (others => '1');
```

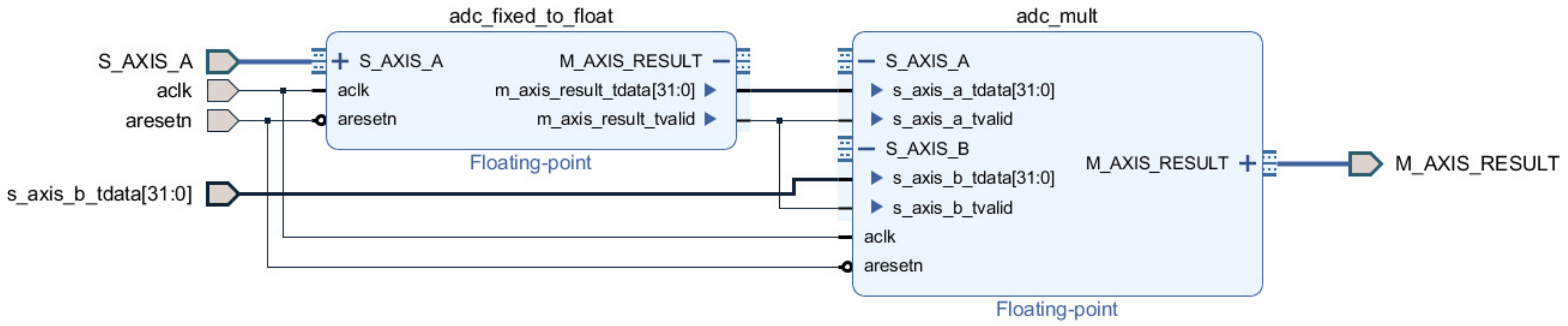
```
                else
                        adder1 <= std_logic_vector(add_u(11 downto 0));
                end if;
                add1_reg        <= adder1(4 downto 0);
                add1_msb        <= adder1(11 downto 5);
                -- stage two adder
                add2_u          := to_unsigned(to_integer(unsigned(adder1(4 downto
                                        0))) + to_integer(unsigned(add2_lsb)), 6);
                if (add2_u(add2_u'high) = '1') then
                        adder2 <= (others => '1');
                else
                        adder2 <= std_logic_vector(add2_u(4 downto 0));
                end if;
                add2_lsb        <= adder2(2 downto 0);
                add2_msb        <= adder2(4 downto 3);
                -- stage three adder
                adder3          <= std_logic_vector(unsigned(adder2(4 downto 3)) –
                                        unsigned(add2_msb));
        end if;
end if;
end process mash_modulator;

duty_ref <= add1_msb & adder3;

--------------------------------------------------------------------------------
-- Clock management circuit
--------------------------------------------------------------------------------

-- Instantiate the MMCM
i_dcm_14 : component dcm_14
        port map (
                clk_in1         => clk,
                resetn          => resetn,
                clk_0           => d1_clk0,
                clk_90          => d1_clk90,
                clk_180         => d1_clk180,
                clk_270         => d1_clk270,
                clk_2x          => d1_clk2,
                clk_4x          => d1_clk4
        );

dll_1_0         <= d1_clk2 and d1_clk4 and d1_clk0;
dll_1_90        <= not d1_clk2 and d1_clk4 and d1_clk90;
dll_1_180       <= d1_clk2 and d1_clk4 and d1_clk180;
dll_1_270       <= not d1_clk2 and d1_clk4 and d1_clk270;

dll_1_0_n       <= d1_clk2 and not d1_clk4 and d1_clk0;
dll_1_90_n      <= not d1_clk2 and not d1_clk4 and d1_clk90;
dll_1_180_n     <= d1_clk2 and not d1_clk4 and d1_clk180;
dll_1_270_n     <= not d1_clk2 and not d1_clk4 and d1_clk270;

-- Mux select for the slow clock
with sclk_ref select dll_1_clk <=
        dll_1_0 when "00",
        dll_1_90 when "01",
        dll_1_180 when "10",
        dll_1_270 when others;

-- Mux select for the slow clock
with sclk_ref select dll_1_clk_n <=
        dll_1_0_n when "00",
        dll_1_90_n when "01",
        dll_1_180_n when "10",
        dll_1_270_n when others;

-- Instantiate the MMCM
i_dcm_12 : component dcm_12
        port map (
                clk_in1         => d1_clk2,
                resetn          => resetn,
                clk_0           => d2_clk0,
                clk_90          => d2_clk90,
                clk_180         => d2_clk180,
                clk_270         => d2_clk270,
                clk_2x          => d2_clk2
        );

dll_2_0         <= d2_clk2 and dll_1_clk and d2_clk0;
dll_2_90        <= not d2_clk2 and dll_1_clk and d2_clk90;
dll_2_180       <= d2_clk2 and dll_1_clk_n and d2_clk180;
dll_2_270       <= not d2_clk2 and dll_1_clk_n and d2_clk270;

-- Mux select for the slow clock
with fclk_ref select dcm_pulse <=
        dll_2_0 when "00",
        dll_2_90 when "01",
        dll_2_180 when "10",
        dll_2_270 when others;

--------------------------------------------------------------------------------
-- Counter comparator
--------------------------------------------------------------------------------

counter_comparator:
process (d1_clk0) is
begin
if rising_edge(d1_clk0) then
        if (resetn_sync = '0') then
                pwm_cnt         <= (others => '0');
                cnt_zero        <= '0';
                cnt_comp        <= '0';
        else
                -- Increment the counter and determine when it reaches zero
                pwm_cnt         <= pwm_cnt + 1;
                cnt_zero        <= and_reduce(pwm_cnt);
                -- Determine when the counter and the reference are about to
                -- become equal
                if (op_mode /= "00" and (pwm_cnt + 1) = unsigned(count_ref)) then
                        cnt_comp        <= '1';
                else
```

```
                    cnt_comp        <= '0';
                end if;
            end if;
        end if;
        end process counter_comparator;

        --------------------------------------------------------------------------------
        -- Output latching circuit
        --------------------------------------------------------------------------------

        rst_qbar        <= (cnt_comp and dcm_pulse) nor rst_q;
        rst_q           <= not cnt_comp nor rst_qbar;
        q               <= (cnt_comp and dcm_pulse) nor qbar;
        qbar            <= (cnt_zero and rst_qbar) nor q;
        pwm_out         <= (q and not_zero) or ctrl_high;

end architecture rtl;
```

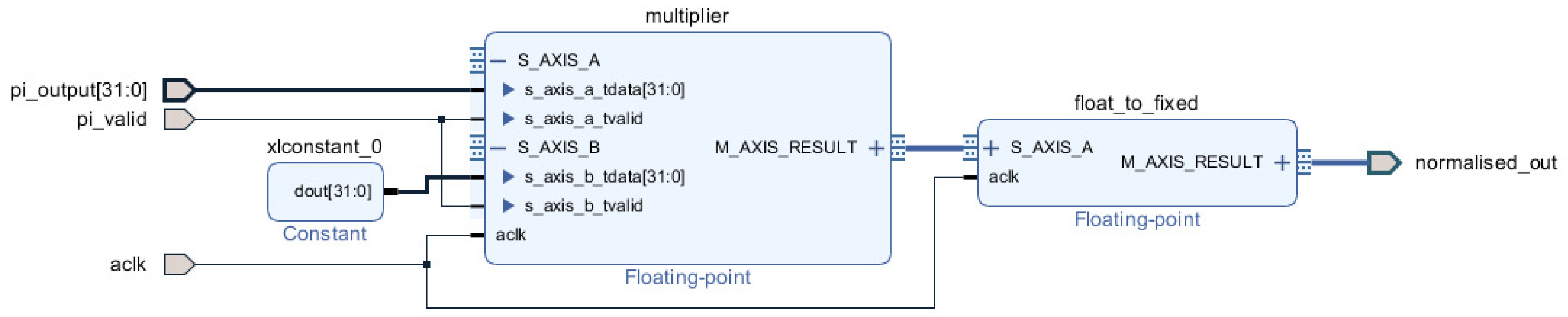
Appendix D.15 – cpu_block.bd



Appendix D.16 – adc_to_float.bd



Appendix D.17 – normaliser.bd



Appendix E – C Code Used in the CPU Core

Append E.1 – Main.c

```
/* Xilinix includes */
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xil_io.h"

/* My includes */
#include "hw_drivers.h"
#include "hw_test.h"
#include "member_lut.h"

/*---------------------------------------------------------*/

/* setup as either test SW or full versions */

#define TEST

#ifdef TEST
int test_mode = 1;
#else
int test_mode = 0;
#endif

/*---------------------------------------------------------*/

/* Consts for the main control loop */
static const float MinDelta = 0.5;
static const float MinDeltaNeg = -0.5;
static const float ConstCurEnd = 24;
static const float SoftStartEnd = 48;
static const float RestartVolt = 40;

/*---------------------------------------------------------*/

/* Interrupt handlers */
void vRunTests(void *CallBackRef, int Bank, u32 Status);
void vUpdateANFIS(void);

/* Test functions for HW debug */
static void prvTestRAM(void);

/*---------------------------------------------------------*/

int main()
{
   init_platform();

   xil_printf("Booting the DER Controller SW...\n\r");
```

```
  /* Initialise the HW and FPGA core*/
  vInitHW();
  vInitDMA();
  prvTestRAM();
  vInitLUT();

  /* Test the AXI Based HW works correctly */
  vSelfTestHW();

  /* Initialise the intterupt routine */
  if (test_mode > 0) {
    /* Setup the HW to allow test to run on switch presses */
    vInitSWInterrupt();
  }
  else {
    /* Setup the HW for normal operation */
    vInitPLInterrupt();
  }

  /* Start the main loop running */
    for (;;);

    cleanup_platform();
    return 0;
}
/*-----------------------------------------------------------*/

static void prvTestRAM(void)
{
  /* Write some data into the RAM buffer for DMA testing */
  u32 base = 0x00100000;

  for (int i = 7; i--; ) {
    Xil_Out32(base, base);
    Xil_Out32(base + 0x4, base + 0x4);
    Xil_Out32(base + 0x8, base + 0x8);
    Xil_Out32(base + 0xC, base + 0xC);

    xil_printf("Performing RAM read back...\n\r");

    xil_printf("Addr = 0x%x, Data =  0x%x\n\r", base, Xil_In32(base));
    xil_printf("Addr = 0x%x, Data =  0x%x\n\r", base + 0x4, Xil_In32(base + 0x4));
    xil_printf("Addr = 0x%x, Data =  0x%x\n\r", base + 0x8, Xil_In32(base + 0x8));
    xil_printf("Addr = 0x%x, Data =  0x%x\n\r", base + 0xC, Xil_In32(base + 0xC));

    base += 0x0010000;
  }

}
/*-----------------------------------------------------------*/

void vRunTests(void *CallBackRef, int Bank, u32 Status)
{
  /* Run a single test on button press in test mode */
  xil_printf("HW Btn Interrupt...\n\r");

  static u8 test_number = 0;
  axi_data_t fis_pattern;

  switch (test_number) {
  case 0 :
    /* Test 1 - Generate a test pattern for the AXI bus */
    vTestAXIRegisters();
    break;

  case 1 :
    /* Test 2 - Read and Write a bigger block of the BRAM */
    vTestDMABRAM();
    break;

  default:
    /* Test 3 - Generate the ANFIS test pattern */
    fis_pattern = AnfisTestPattern[test_number - 2];
    vSetLUTADDR(fis_pattern);
    vStartDMATransfer();
    vWriteFPGAMeasurements(fis_pattern);
    break;
  }

  /* Increment for the next test */
  test_number = test_number > NUM_FIS_PATTERNS + 1 ? 0 : test_number + 1;
}
/*----------------------------------------------------------*/

void vUpdateANFIS(void)
{
  /* FPGA Interrupt handler, triggers an update of the FPGA */

  static float ld_old = 0;
  static float iref_old = 0;
  static u32 delt_cnt = 0;
  static u32 mode = 0;
  axi_data_t adc_dat;

  /* Read the current ADC values */
  vReadFpgaMeasurements(&adc_dat);

  /* Determine the delta values */
  float ld_delt = adc_dat.load - ld_old;
  ld_old = adc_dat.load;

  if (ld_delt > MinDelta || ld_delt < MinDeltaNeg) {
    /* Set the delta value */
    adc_dat.dload = ld_delt;
    adc_dat.diref = adc_dat.iref - iref_old;
    delt_cnt = 12;
  } else if (delt_cnt == 0) {
    /* Clear the delta values after 50 us */
    adc_dat.dload = 0;
    adc_dat.diref = 0;
  }
  else {
    delt_cnt--;
  }

  /* Update the DMA data */
  vSetLUTADDR(adc_dat);
  vStartDMATransfer();

  /* Determine the ctrl mode */
  switch(mode) {
  case 0 :
    /* First stage of the soft-start move to next state when 24V */
    if (adc_dat.volt > ConstCurEnd) {
      mode = 0x1;
    }
    break;

  case 1 :
    /* First stage of the soft-start move to next state when 24V */
    if (adc_dat.volt > SoftStartEnd){
      mode = 0x2;
      adc_dat.ctrl = 0x2;
    }
    break;

  default:
    /* Normal operation - return to start when less than 40V  */
    if (adc_dat.volt < RestartVolt) {
      mode = 0x0;
    }
    break;
  }

  adc_dat.ctrl = mode;

  /* And finally update the AXI registers */
  vWriteFPGAMeasurements(adc_dat);
}
```

Appendix E.2 – AXI_control.h

```
#ifndef SRC_AXI_CONTROL_H_
#define SRC_AXI_CONTROL_H_

#include "xil_io.h"
#include "xparameters.h"

/* Define the register addresses */
#define DMA_BRAM_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_BRAM_CTRL_0_S_AXI_BASEADDR
#define DMA_CTRL_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_DMA_0_BASEADDR
```

```
#define AXI_BASE_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_PSPL_AXI_INTERFACE_0_BASEADDR

#define VOLT_ADDR AXI_BASE_ADDR + 0x0
#define RVOLT_ADDR AXI_BASE_ADDR + 0x4
#define IND_ADDR AXI_BASE_ADDR + 0x8
#define RIND_ADDR AXI_BASE_ADDR + 0xC
#define CUR_ADDR AXI_BASE_ADDR + 0x10
#define RCUR_ADDR AXI_BASE_ADDR + 0x14
#define DCUR_ADDR AXI_BASE_ADDR + 0x18
#define IREF_ADDR AXI_BASE_ADDR + 0x1C
#define DIREF_ADDR AXI_BASE_ADDR + 0x20
#define CTRL_ADDR AXI_BASE_ADDR + 0x24
#define TEST1_ADDR AXI_BASE_ADDR + 0x30
#define TEST2_ADDR AXI_BASE_ADDR + 0x40

/* type to store all data to transfer */
typedef struct {
  float volt;
  float iref;
  float diref;
  float load;
  u32 r_load;
  float dload;
  int ctrl;
} axi_data_t;

/* Main read/write functions for control */
void vWriteFPGAMeasurements(axi_data_t data);
void vReadFpgaMeasurements(axi_data_t * data);

/* Self testing functions */
void vTestWriteAXI(u32 data1, u32 data2);
void vCheckAXI(u32 data1, u32 data2);
void vTestWriteBRAM(u32 data1, u32 data2);
void vCheckBRAM(u32 data1, u32 data2);

#endif /* SRC_AXI_CONTROL_H_ */
```

Appendix E.3 – AXI_control.c

```
#include "axi_control.h"

union u_type {
  u32   i;
    float f;
};

/*---------------------------------------------------------*/

static u32 prvFloatToU32(float fl)
{
  /* Convert a float into a u32 */
  union u_type cnvrt;
  cnvrt.f = fl;
  return cnvrt.i;
}
/*----------------------------------------------------------*/

static float prvU32ToFloat(u32 val)
{
  /* Convert a float into a u32 */
  union u_type cnvrt;
  cnvrt.i = val;
  return cnvrt.f;
}
/*----------------------------------------------------------*/

void vWriteFPGAMeasurements(axi_data_t data)
{
  /* Write the current measurements to the AXI engine */
  Xil_Out32(IREF_ADDR, prvFloatToU32(data.iref));
  Xil_Out32(DIREF_ADDR, prvFloatToU32(data.diref));
  Xil_Out32(CUR_ADDR, prvFloatToU32(data.load));
  Xil_Out32(DCUR_ADDR, prvFloatToU32(data.dload));

  /* Write the FPGA control data */
  Xil_Out32(CTRL_ADDR, data.ctrl);
}
/*----------------------------------------------------------*/

void vReadFpgaMeasurements(axi_data_t * data)
{
  /* Read back the current measurements for the ANFIS engine */
  data->volt = prvU32ToFloat(Xil_In32(VOLT_ADDR));
  data->iref = prvU32ToFloat(Xil_In32(IREF_ADDR));
  data->load = prvU32ToFloat(Xil_In32(CUR_ADDR));
  data->r_load = Xil_In32(RCUR_ADDR);
}
/*----------------------------------------------------------*/

void vTestWriteAXI(u32 data1, u32 data2)
{
  /* Write data into two test locations */
  Xil_Out32(TEST1_ADDR, data1);
  Xil_Out32(TEST2_ADDR, data2);
}
/*----------------------------------------------------------*/

void vCheckAXI(u32 data1, u32 data2)
{
  /* Read the test space and check the values are correct */
  u32 test = Xil_In32(TEST1_ADDR);

  if (test /= data1) {
    xil_printf("AXI self test failed\n\r");
    return;
  }
  test = Xil_In32(AXI_BASE_ADDR);

  if (test /= data2) {
    xil_printf("AXI self test failed\n\r");
  }
  else {
    xil_printf("AXI self test passed\n\r");
  }
}
/*-----------------------------------------------------------*/

void vTestWriteBRAM(u32 data1, u32 data2)
{
  /* Write data into two test locations */
  Xil_Out32(DMA_BRAM_ADDR + 0x200, data1);
  Xil_Out32(DMA_BRAM_ADDR + 0x204, data2);
}
/*-----------------------------------------------------------*/

void vCheckBRAM(u32 data1, u32 data2)
{
  /* Read the test space and check the values are correct */
  u32 test = Xil_In32(DMA_BRAM_ADDR + 0x200);

  if (test /= data1) {
    xil_printf("BRAM self test failed\n\r");
    return;
  }

  test = Xil_In32(DMA_BRAM_ADDR + 0x4);

  if (test /= data2) {
    xil_printf("BRAM self test failed\n\r");
  }
  else {
    xil_printf("BRAM self test passed\n\r");
  }
}
/*-----------------------------------------------------------*/
```

Appendix E.4 – hw_drivers.h

```
#include "xparameters.h"

#ifndef SRC_HW_DRIVERS_H_
#define SRC_HW_DRIVERS_H_

/* Define the GPIO pins */
#define LOAD_SW 50  // This is the switch BTN8
#define PS_LED 7    // This is the LED attached to the PS side
#define FET_SW 13   // JE1

/* Define the IDs in a more sensible fashion */
#define GPIO_DEVICE_ID XPAR_PS7_GPIO_0_DEVICE_ID
```

```
#define SCUGIC_DEVICE_ID XPAR_SCUGIC_SINGLE_DEVICE_ID
#define GPIO_INTERRUPT_ID XPS_GPIO_INT_ID
#define FPGA_INTERRUPT_ID XPS_FIQ_INT_ID
#define GIC_INTR_RISING_EDGE 0x3
#define DMA_DEVICE_ID
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_DMA_0_DEVICE_ID

/* DMA descriptor addresses */
#define DMA_BASE_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_DMA_0_BASEADDR
#define DMA_DESC_START_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_DMA_0_BASEADDR + 0x8
#define DMA_DESC_END_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_DMA_0_BASEADDR + 0x10

#define BRAM_BASE_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_BRAM_CTRL_0_S_AXI_BASEADDR
#define BRAM_STATUS_FIRST BRAM_BASE_ADDR + 0x1C

#define DMA_FIRST_BUF_ADDR 0xC0000000
#define DMA_LAST_BUF_ADDR 0xC00001C0

/* Store for the DMA buffer addresses  */
typedef struct {
  u32 vkp_addr;
  u32 vki_addr;
  u32 ikp_addr;
  u32 iki_addr;
} dma_buf_t;

/* Wrapper functions to drive the hardware */
void vInitHW(void);
void vInitDMA(void);
void vInitPLInterrupt(void);
void vInitSWInterrupt(void);
void vStartDMATransfer(void);

#endif /* SRC_HW_DRIVERS_H_ */
```

Appendix E.5 – hw_drivers.c

```
/*
 * hw_drivers.c
 *
 *  Created on: 15 Sep 2017
 *      Author: John
 */

/* Xilinx includes. */
#include "xgpiops.h"
#include "xil_printf.h"
#include "xscugic.h"
#include "xil_exception.h"
#include "xaxidma.h"
```

```
/* My includes */
#include "hw_drivers.h"

/* DMA control parameters */
#define DMA_ALIGNMENT 0x40

/* GPIO Configuration variables */
static XGpioPs Gpio;

/* Interrupt controller variables */
static int IntCreated = 0;
static XScuGic Intc;
XScuGic * GicInstPtr = &Intc;

/*-----------------------------------------------------------*/

/* Interrupt handlers declared in main.c */
extern void vRunTests(void *CallBackRef, int Bank, u32 Status);
extern void vUpdateANFIS(void);

/*-----------------------------------------------------------*/

static void prvXGpioPs_IntrHandler(XGpioPs *InstancePtr)
{
  /* Fixed version of the Xilinx driver to stop double firing */
  u8 Bank;
  u32 IntrStatus;
  u32 IntrEnabled;

  Xil_AssertVoid(InstancePtr != NULL);
  Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);

  for (Bank = 0U; Bank < InstancePtr->MaxBanks; Bank++) {
    IntrStatus = XGpioPs_IntrGetStatus(InstancePtr, Bank);
    IntrEnabled = XGpioPs_IntrGetEnabled(InstancePtr, Bank);

    if ((IntrStatus & IntrEnabled) != (u32)0) {
      XGpioPs_IntrClear((XGpioPs *)InstancePtr, Bank,
            (IntrStatus & IntrEnabled));
      InstancePtr->Handler(InstancePtr->
             CallBackRef, Bank,
             (IntrStatus & IntrEnabled));
    }
  }
}
/*-----------------------------------------------------------*/

static void prvFPGAInterrupt (void * CallBackRef)
{
  Xil_AssertVoid(CallBackRef != NULL);
  static int ticks = 0;

  /* Invoke the FPGA update task */
  XScuGic_Disable(GicInstPtr, FPGA_INTERRUPT_ID);
  vUpdateANFIS();
  XScuGic_Enable(GicInstPtr, FPGA_INTERRUPT_ID);

  ticks++;
  if (ticks > 4) {
    XScuGic_Disable(GicInstPtr, FPGA_INTERRUPT_ID);
  }
  else {
    xil_printf("Hi PL interrupt\n\r");
  }
}
/*-----------------------------------------------------------*/

void vInitHW(void)
{
  /* Initialise the GPIO */
  XGpioPs_Config * GPIOConfigPtr = XGpioPs_LookupConfig(GPIO_DEVICE_ID);
  int Status = XGpioPs_CfgInitialize(&Gpio, GPIOConfigPtr,
    GPIOConfigPtr->BaseAddr);
  if (Status != XST_SUCCESS) {
    xil_printf("GPIO initialisation failed\n\r");
    return;
  }
  else {
    xil_printf("GPIO initialised successfully\n\r");
  }

  /* Set GPIO directions */
  XGpioPs_SetDirectionPin(&Gpio, LOAD_SW, 0x0);
  XGpioPs_SetDirectionPin(&Gpio, PS_LED, 0x1);
  XGpioPs_SetDirectionPin(&Gpio, FET_SW, 0x1);

  /* Enable GPIO output pins */
  XGpioPs_SetOutputEnablePin(&Gpio, PS_LED, 1);
  XGpioPs_SetOutputEnablePin(&Gpio, FET_SW, 1);
}
/*-----------------------------------------------------------*/

void vInitDMA(void)
{

  /* Reset the DMA engine */
  Xil_Out32(DMA_BASE_ADDR , 0x00015016);
  Xil_Out32(DMA_BASE_ADDR , 0x00015012);

  /* Initialise the descriptor string pointers */
  u32 desc_addr = BRAM_BASE_ADDR;
  u32 desc_data = DMA_FIRST_BUF_ADDR + DMA_ALIGNMENT;

  while (desc_addr < BRAM_BASE_ADDR + 0x1C0) {
    /* Set the next descriptor pointer & clear MSB */
    Xil_Out32(desc_addr, desc_data);
    Xil_Out32(desc_addr + 0x4, 0x0);
    /* Set the data for the next installment */
    desc_addr += DMA_ALIGNMENT;
    desc_data += DMA_ALIGNMENT;
  }

  Xil_Out32(BRAM_BASE_ADDR + 0x1C0, DMA_FIRST_BUF_ADDR);

  /* Write the packet control words (16 bytes for all) */
  desc_addr = BRAM_BASE_ADDR + 0x58;

  while(desc_addr < BRAM_BASE_ADDR + 0x1C0) {
    Xil_Out32(desc_addr, 0x00000010);
    desc_addr += 0x40;
  }

  /* Set the SOF/EOF flags in the first and last descriptor */
  Xil_Out32(BRAM_BASE_ADDR + 0x18, 0x08000010);
  Xil_Out32(BRAM_BASE_ADDR + 0x1D8, 0x04000010);
}
/*---------------------------------------------------------*/

void vInitPLInterrupt(void)
{

  xil_printf("Setting up the FPGA interrupt...\n\r");

  /* Lookup & intitalise the GIC config data */
  XScuGic_Config * IntcConfig = XScuGic_LookupConfig(SCUGIC_DEVICE_ID);

  if (IntcConfig == NULL) {
    xil_printf("Couldn't find the GIC config\n\r");
    return;
  }

  int status = XScuGic_CfgInitialize( GicInstPtr,
                                      IntcConfig,
                                      IntcConfig->CpuBaseAddress);

  if (status != XST_SUCCESS) {
    xil_printf("Couldn't initialise the GIC cfg\n\r");
    return;
  }

  /* Perform the HW self test function */
  status = XScuGic_SelfTest(GicInstPtr);

  if (status == XST_SUCCESS) {
    xil_printf("GIC Self Test Passed\n\r");
  }
  else {
    xil_printf("GIC Self Test Failed\n\r");
  }

  xil_printf("Finished setting up the FPGA interrupt...\n\r");

  /*Setup the exception handler */
  Xil_ExceptionInit();

  Xil_ExceptionRegisterHandler( XIL_EXCEPTION_ID_FIQ_INT,
                         (Xil_ExceptionHandler) vUpdateANFIS, //prvFPGAInterrupt,
                    GicInstPtr);

  Xil_ExceptionEnableMask(XIL_EXCEPTION_FIQ);
}
/*-----------------------------------------------------------*/

void vInitSWInterrupt(void)
{

  xil_printf("Setting up the hardware switch interrupt...\n\r");

  /* Initialise the interrupt controller */
  Xil_ExceptionInit();
  XScuGic_Config * IntcConfig = XScuGic_LookupConfig(SCUGIC_DEVICE_ID);
  int status = XScuGic_CfgInitialize( GicInstPtr,
                                IntcConfig,
                                IntcConfig->CpuBaseAddress);

  if (status != XST_SUCCESS) {
    xil_printf("Failed to initialise the interrupt engine\n\r");
    return;
  }
  else {
    xil_printf("Finished initialising the interrupt engine\n\r");
    IntCreated = 1;
  }

  /* connect to the hardware */
  Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
              (Xil_ExceptionHandler) XScuGic_InterruptHandler,
              GicInstPtr);

  /* Enable interrupts in the Processor. */
  Xil_ExceptionEnableMask(XIL_EXCEPTION_IRQ);


  /* Connect to the interrupt hardware */
  XScuGic_Connect(GicInstPtr,
                GPIO_INTERRUPT_ID,
                (Xil_ExceptionHandler) prvXGpioPs_IntrHandler,
                (void *) &Gpio);

  /* Setup the handler for all pins in bank 0  */
  XGpioPs_SetIntrTypePin(&Gpio, LOAD_SW, XGPIOPS_IRQ_TYPE_EDGE_RISING);

  /* Configure the handler for gpio interrupts */
  XGpioPs_SetCallbackHandler(&Gpio, (void *) &Gpio, vRunTests);

  /* Enable the interrupt for the GPIO device. */
  XScuGic_Enable(GicInstPtr, GPIO_INTERRUPT_ID);
  /* Enable the interrupt pins */
  XGpioPs_IntrEnablePin(&Gpio, LOAD_SW);

  xil_printf("Finished setting up the interrupt engine\n\r");
}
/*---------------------------------------------------------*/

void vStartDMATransfer(void)
{
  /* Reset the DMA engine and initialise a transfer */
  Xil_Out32(DMA_BASE_ADDR , 0x00015016);
  Xil_Out32(DMA_BASE_ADDR , 0x00015012);
  Xil_Out32(DMA_DESC_START_ADDR , DMA_FIRST_BUF_ADDR);
  Xil_Out32(DMA_BASE_ADDR , 0x00015013);
  Xil_Out32(DMA_DESC_END_ADDR , DMA_LAST_BUF_ADDR);
}
```

Appendix E.6 – hw_test.h

```
#ifndef SRC_HW_TEST_H_
#define SRC_HW_TEST_H_

#include "axi_control.h"

/* ANFIS engine test patterns */
#define NUM_FIS_PATTERNS 16

const axi_data_t AnfisTestPattern [NUM_FIS_PATTERNS] = {
  /* Steady state tests */
  {.iref = 1, .diref = 0, .load = 0.5, .r_load = 0x3FF, .dload = 0, .ctrl = 0},
  {.iref = 2, .diref = 0, .load = 1,   .r_load = 0x800, .dload = 0, .ctrl = 0},
  {.iref = 3, .diref = 0, .load = 1.5, .r_load = 0xBFF, .dload = 0, .ctrl = 0},
  {.iref = 4, .diref = 0, .load = 2,   .r_load = 0xFFF, .dload = 0, .ctrl = 0},
  /* Half an amp step up */
  {.iref = 2, .diref = 1, .load = 1,   .r_load = 0x800, .dload = 0.5, .ctrl = 0},
  {.iref = 3, .diref = 1, .load = 1.5, .r_load = 0xBFF, .dload = 0.5, .ctrl = 0},
  {.iref = 4, .diref = 1, .load = 2,   .r_load = 0xFFF, .dload = 0.5, .ctrl = 0},
  /* Half an amp step down */
  {.iref = 1, .diref = -1, .load = 0.5, .r_load = 0x3FF, .dload = -0.5, .ctrl = 0},
  {.iref = 2, .diref = -1, .load = 1,   .r_load = 0x800, .dload = -0.5, .ctrl = 0},
  {.iref = 3, .diref = -1, .load = 1.5, .r_load = 0xBFF, .dload = -0.5, .ctrl = 0},
  /* 1A step up */
  {.iref = 3, .diref = 2, .load = 1.5, .r_load = 0xBFF, .dload = 1, .ctrl = 0},
  {.iref = 4, .diref = 2, .load = 2,   .r_load = 0xFFF, .dload = 1, .ctrl = 0},
  /* 1A step down */
  {.iref = 1, .diref = -2, .load = 0.5, .r_load = 0x3FF, .dload = -1, .ctrl = 0},
  {.iref = 2, .diref = -2, .load = 1,   .r_load = 0x800, .dload = -1, .ctrl = 0},
  /* Full range tests */
  {.iref = 4, .diref = 3,  .load = 2,   .r_load = 0xFFF, .dload = 1.5, .ctrl = 0},
  {.iref = 1, .diref = -3, .load = 0.5, .r_load = 0x3FF, .dload = -1.5, .ctrl = 0}
};

/* Hardware test functions */
void vSelfTestHW(void);
void vTestAXIRegisters(void);
void vTestDMABRAM(void);

#endif /* SRC_HW_TEST_H_ */
```

Appendix E.7 – hw_test.c

```
#include "axi_control.h"
#include "xil_printf.h"

static const u32 TEST_DATA1 = 0x55555555;
static const u32 TEST_DATA2 = 0xAAAAAAAA;

/*---------------------------------------------------------*/
void vSelfTestHW(void)
{
  /* Perform the PL/PS interface tests */
  vTestWriteAXI(TEST_DATA1, TEST_DATA2);
  vCheckAXI(TEST_DATA1, TEST_DATA2);
  vTestWriteBRAM(TEST_DATA1, TEST_DATA2);
  vCheckBRAM(TEST_DATA1, TEST_DATA2);
}
/*---------------------------------------------------------*/

void vTestAXIRegisters(void)
{

  xil_printf("Writing AXI Test data...\n\r");

  /* Generate some test stimulus for the AXI interface */
  axi_data_t axi_dat;
  axi_dat.iref = 0.5;
  axi_dat.diref = 0.75;
  axi_dat.load = 1.0;
  axi_dat.dload = 1.25;
  axi_dat.ctrl = 3;
  vWriteFPGAMeasurements(axi_dat);

  /* Write/Readback test for the test registers */
  Xil_Out32(TEST1_ADDR, TEST_DATA2);
  Xil_Out32(TEST2_ADDR, TEST_DATA1);
  xil_printf("Addr = 0x400000030, Data =  %x\n\r", Xil_In32(TEST1_ADDR));
  xil_printf("Addr = 0x400000040, Data =  %x\n\r", Xil_In32(TEST2_ADDR));

  xil_printf("AXI bus test complete\n\r");
}
/*---------------------------------------------------------*/

void vTestDMABRAM(void)
{
  /* More extensive test of the BRAM interface */
  xil_printf("Writing test data into the DMA BRAM\n\r");

  Xil_Out32(DMA_BRAM_ADDR + 0x200, DMA_BRAM_ADDR + 0x200);
  Xil_Out32(DMA_BRAM_ADDR + 0x240, DMA_BRAM_ADDR + 0x240);
  Xil_Out32(DMA_BRAM_ADDR + 0x280, DMA_BRAM_ADDR + 0x280);
  Xil_Out32(DMA_BRAM_ADDR + 0x2C0, DMA_BRAM_ADDR + 0x2C0);

  /* Read the test data back over the BRAM */
  xil_printf("Starting DMA BRAM read back tests...\n\r");

  xil_printf("Addr = 0x800000200, Data =  %x\n\r", Xil_In32(DMA_BRAM_ADDR + 0x200));
  xil_printf("Addr = 0x800000240, Data =  %x\n\r", Xil_In32(DMA_BRAM_ADDR + 0x240));
  xil_printf("Addr = 0x800000280, Data =  %x\n\r", Xil_In32(DMA_BRAM_ADDR + 0x280));
  xil_printf("Addr = 0x8000002C0, Data =  %x\n\r", Xil_In32(DMA_BRAM_ADDR + 0x2C0));

  xil_printf("DMA BRAM testing completed...\n\r");
}
/*------------------------------------------------------------*/
```

Appendix E.8 – member_lut.h

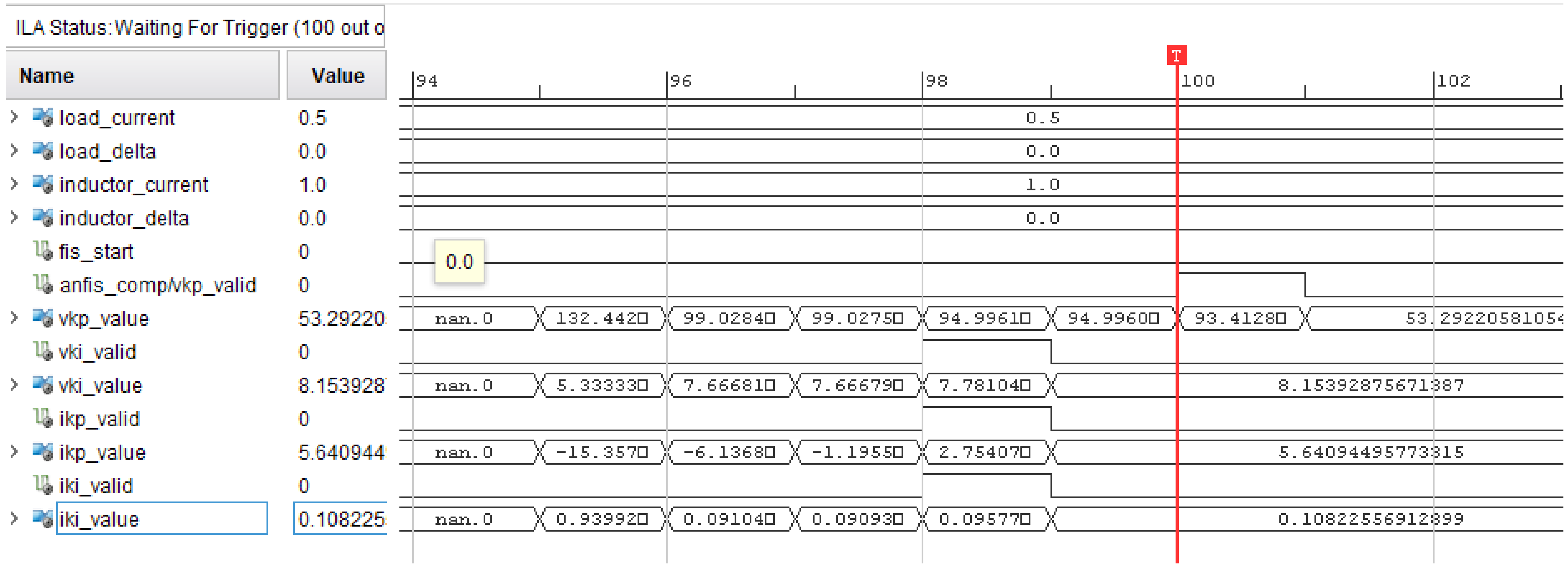
```
#ifndef SRC_MEMBER_LUT_H_
#define SRC_MEMBER_LUT_H_

#include "xil_io.h"
#include "axi_control.h"

/* Define the buffer base address */
#define VKP_STDY_BASE_ADDR 0x120000
#define VKP_DELT_BASE_ADRR 0x130000
#define VKI_STDY_BASE_ADDR 0x150000
#define VKI_DELT_BASE_ADDR 0x160000
#define IKP_STDY_BASE_ADDR 0x180000
#define IKP_DELT_BASE_ADDR 0x190000
#define IKI_STDY_BASE_ADDR 0x1B0000
#define IKI_DELT_BASE_ADDR 0x1C0000

/* Define the BRAM base addresses */
#define BRAM_VKP_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 0x8
#define BRAM_VKP_DELT_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 0x48
#define BRAM_VKI_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 0x88
#define BRAM_VKI_DELT_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + 0xC8
#define BRAM_IKP_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_BRAM_CTRL_0_S_AXI_BASEADDR +
0x108
#define BRAM_IKP_DELT_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_BRAM_CTRL_0_S_AXI_BASEADDR +
0x148
#define BRAM_IKI_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_BRAM_CTRL_0_S_AXI_BASEADDR +
0x188
#define BRAM_IKI_DELT_ADDR
XPAR_CPU_SUB_SYS_CPU_BLOCK_I_AXI_BRAM_CTRL_0_S_AXI_BASEADDR +
0x1C8
```

```
/* Struct which contains the info about the ANFIS */
typedef struct {
  float steady_mems[4][2];
  float delta_mems[4][2];
} anfis_data_t;

/* Function declarations */
void vInitLUT(void);
void vSetLUTADDR(axi_data_t adc_val);

#endif /* SRC_MEMBER_LUT_H_ */
```

Appendix E.9 – member_lut.c

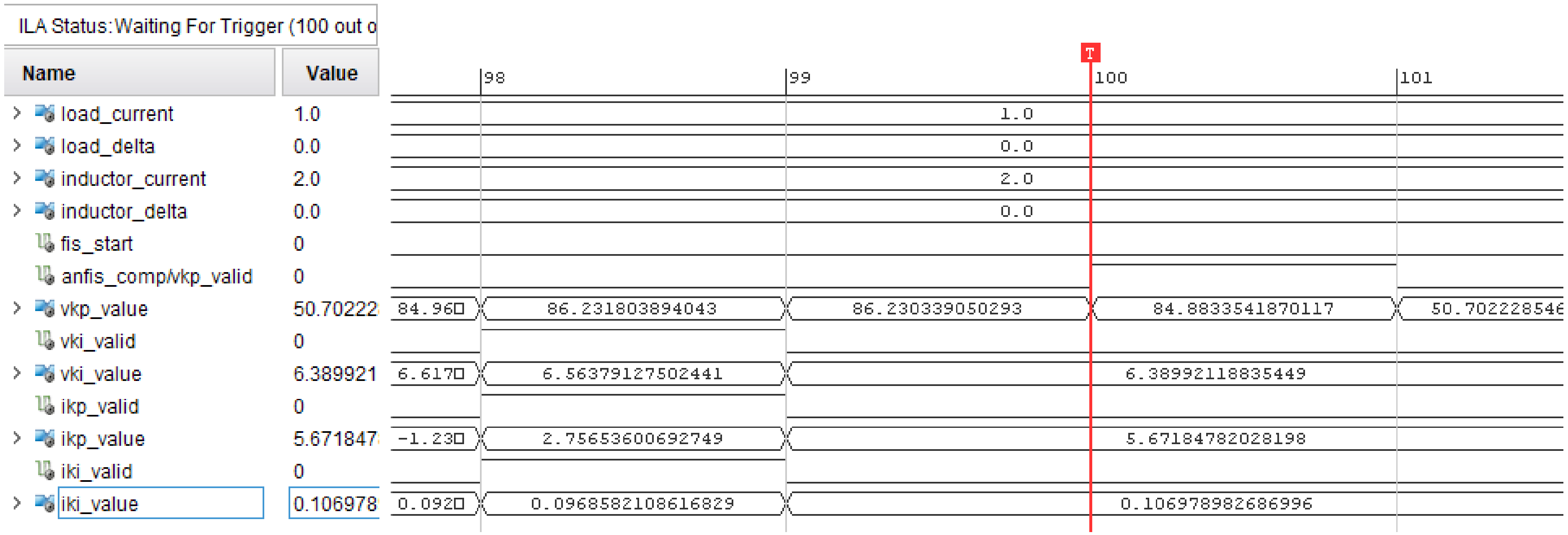
```
/*
 * member_lut.c
 *
 *  Created on: 8 Oct 2017
 *      Author: John
 */

#include <math.h>
#include "member_lut.h"

#define SQUARE(x) (x) * (x)

/* Define the ADC scaling factors */
static const float LoadScale = 2047.5;
static const float IndScale = 1023.75;
static const float LoadBitValue = 2.0 / 4095.0;
static const float IndBitValue = 4.0 / 4095.0;

/* Union for double to u32 conversion */
union float_u_t {
  u32   i;
    float f;
};

/*-----------------------------------------------------------*/

/* Define the parameters of the ANFIS functions*/
static const anfis_data_t VKPFis = {
    {
      {1.27896196298006, 1.13358964698942},
      {0.851997240311144, 4.08298944792725},
      {0.929419778411536, 6.65072719355158},
      {1.07266973068116, 9.97940925638625}
    },
    {
      {1.9923076026989, -9.29603945640981},
      {0.695976416320164, -4.15262498686497},
      {1.04434340337271, 4.256160234167},
      {2.29389783460814, 9.3966886863281}
    }
};

static const anfis_data_t VKIFis = {
    {
      {0.690804881076229, 0.850254019465858},
      {0.812244279289328, 4.07655286277922},
      {0.533525681475427, 6.84268924973098},
      {1.07255693238821, 9.97943395965198}
    },
    {
      {1.91263958719969, -9.26991945412401},
      {0.722677630318396, -3.87515175559317},
      {0.793390097461039, 4.36230489790685},
      {2.06733381183133, 9.50178750098178}
    }
};

static const anfis_data_t IKPFis = {
    {
      {0.887691041101543, 0.949001329589385},
      {0.381077200846656, 4.25006526674098},
      {0.440252803991601, 6.86576891306185},
      {1.07218470564311, 9.97951179796942}
    },
    {
      {2.0125354055693, -9.24966709699988},
      {0.790702021548468, -3.82455853601802},
      {0.883372811779322, 4.37459398788082},
      {1.93996374398332, 9.52108969869462}
    }
};

static const anfis_data_t IKIFis = {
    {
      {0.890525253161075, 0.951264469464089},
      {0.370136819026424, 4.27784670828263},
      {0.296659508423983, 7.03583368452245},
      {0.891182734116097, 10.0618941720149}
    },
    {
      {2.37302962647052, -9.03553182420158},
      {0.850787403981641, -3.94944188693516},
      {1.0677294127161, 3.93459245764684},
      {2.39997690253802, 9.16626611584046}
    }
};

/*-----------------------------------------------------------*/
static u32 prvCalcGaus(float x, float sig, float c)
{
    /* Function to calculate a Gaussian member */
  union float_u_t cnvrt;

    float num = -1 * (SQUARE(x - c));
    float denom = 2 * SQUARE(sig);
    cnvrt.f = expf(num / denom);

    /* Ignore outputs which are below a certain value */
    if (cnvrt.f < 1e-6) {
      return 0;
    }
    else {
      return cnvrt.i;
    }
}

/*------------------------------------------------------------*/
void vInitLUT(void)
{
  xil_printf("Initialising the input LUT...\n\r");

  u32 vkp_stdy;
  u32 vkp_delt;
  u32 vkp_delt_n;
  u32 vki_stdy;
  u32 vki_delt;
  u32 vki_delt_n;
  u32 ikp_stdy;
  u32 ikp_delt;
  u32 ikp_delt_n;
  u32 iki_stdy;
  u32 iki_delt;
  u32 iki_delt_n;
  u32 offs;

  /* Initialise the Gaussian LUT values */
  for (u32 j = 0; j < 0x1000; j++) {
    for (u32 i = 0; i < 4; i++) {
      /* Convert the  */
      float floatLoad = (j * LoadBitValue);
      float nFloatLoad = floatLoad * -1.0;
      float floatInd = (j * IndBitValue);
      float nFloatInd = floatInd * -1.0;

      /* Calcualte the gaus values */
      vkp_stdy = prvCalcGaus(floatLoad, VKPFis.steady_mems[i][0],
        VKPFis.steady_mems[i][1]);
      vkp_delt = prvCalcGaus(floatLoad, VKPFis.delta_mems[i][0],
        VKPFis.delta_mems[i][1]);
      vkp_delt_n = prvCalcGaus(nFloatLoad, VKPFis.delta_mems[i][0],
        VKPFis.delta_mems[i][1]);

      vki_stdy = prvCalcGaus(floatLoad, VKIFis.steady_mems[i][0],
        VKIFis.steady_mems[i][1]);
      vki_delt = prvCalcGaus(floatLoad, VKIFis.delta_mems[i][0],
        VKIFis.delta_mems[i][1]);
      vki_delt_n = prvCalcGaus(nFloatLoad, VKIFis.delta_mems[i][0],
          VKIFis.delta_mems[i][1]);
      ikp_stdy = prvCalcGaus(floatInd, IKPFis.steady_mems[i][0],
        IKPFis.steady_mems[i][1]);
      ikp_delt = prvCalcGaus(floatInd, IKPFis.delta_mems[i][0],
        IKPFis.delta_mems[i][1]);
      ikp_delt_n = prvCalcGaus(nFloatInd, IKPFis.delta_mems[i][0],
        IKPFis.delta_mems[i][1]);

      iki_stdy = prvCalcGaus(floatInd, IKIFis.steady_mems[i][0],
        IKIFis.steady_mems[i][1]);
      iki_delt = prvCalcGaus(floatInd, IKIFis.delta_mems[i][0],
        IKIFis.delta_mems[i][1]);
      iki_delt_n = prvCalcGaus(nFloatInd, IKIFis.delta_mems[i][0],
        IKIFis.delta_mems[i][1]);

      /* Write the values into the DDR */
      offs = (j << 4) + ((3 - i) * 4);

      Xil_Out32(VKP_STDY_BASE_ADDR + offs, vkp_stdy);
      Xil_Out32(VKP_DELT_BASE_ADRR + offs, vkp_delt);
      Xil_Out32(VKP_DELT_BASE_ADRR + offs + 0x10000, vkp_delt_n);

      Xil_Out32(VKI_STDY_BASE_ADDR + offs, vki_stdy);
      Xil_Out32(VKI_DELT_BASE_ADDR + offs, vki_delt);
      Xil_Out32(VKI_DELT_BASE_ADDR + offs + 0x10000, vki_delt_n);

      Xil_Out32(IKP_STDY_BASE_ADDR + offs, ikp_stdy);
      Xil_Out32(IKP_DELT_BASE_ADDR + offs, ikp_delt);
      Xil_Out32(IKP_DELT_BASE_ADDR + offs + 0x10000, ikp_delt_n);

      Xil_Out32(IKI_STDY_BASE_ADDR + offs, iki_stdy);
      Xil_Out32(IKI_DELT_BASE_ADDR + offs, iki_delt);
      Xil_Out32(IKI_DELT_BASE_ADDR + offs + 0x10000, iki_delt_n);
    }
  }

  xil_printf("Finished initialising the input LUT\n\r");
}

/*--------------------------------------------------------*/
void vSetLUTADDR(axi_data_t adc_val)
{

  /* Convert the steady floats into DDR base */
  u32 cur_int = adc_val.r_load << 4;
  u32 iref_int = ((u32) (adc_val.iref * IndScale)) << 4;

  /* Convert the delta floats into DDR base*/
  u32 dcur_int;
  u32 diref_int;

  if (adc_val.dload < 0) {
    dcur_int = (((u32) ((-1 * adc_val.dload) * LoadScale)) + 0x1000) << 4;
  } else {
    dcur_int = ((u32) (adc_val.dload * LoadScale)) << 4;
  }
  if (adc_val.diref < 0) {
    diref_int = (((u32) ((-1 * adc_val.diref) * IndScale)) + 0x1000) << 4;
  } else {
    diref_int = ((u32) (adc_val.diref * IndScale)) << 4;
  }

  /* Write the addresses into the BRAM buffer */
  Xil_Out32(BRAM_VKP_ADDR, cur_int + VKP_STDY_BASE_ADDR);
  Xil_Out32(BRAM_VKP_DELT_ADDR, dcur_int + VKP_DELT_BASE_ADRR);
  Xil_Out32(BRAM_VKI_ADDR, cur_int + VKI_STDY_BASE_ADDR);
  Xil_Out32(BRAM_VKI_DELT_ADDR, dcur_int + VKI_DELT_BASE_ADDR);
  Xil_Out32(BRAM_IKP_ADDR, iref_int + IKP_STDY_BASE_ADDR);
  Xil_Out32(BRAM_IKP_DELT_ADDR, diref_int + IKP_DELT_BASE_ADDR);
  Xil_Out32(BRAM_IKI_ADDR, iref_int + IKI_STDY_BASE_ADDR);
  Xil_Out32(BRAM_IKI_DELT_ADDR, diref_int + IKI_DELT_BASE_ADDR);
}
```

Appendix F – APSOC ANFIS Algorithm Test Results


ILA Status: Waiting For Trigger (100 out o

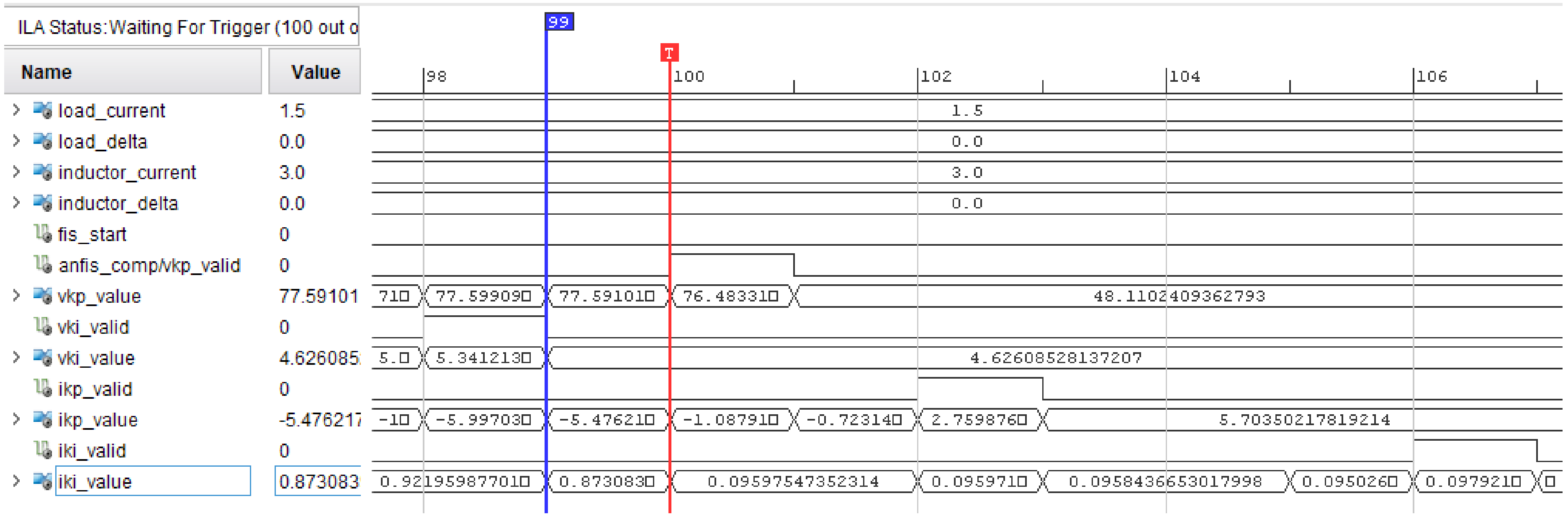
Name	Value
load_current	0.5
load_delta	0.0
inductor_current	1.0
inductor_delta	0.0
fis_start	0
anfis_comp/vkp_valid	0
vkp_value	53.29220
vki_valid	0
vki_value	8.153928
ikp_valid	0
ikp_value	5.640944
iki_valid	0
iki_value	0.108225

```
94            96            98            100           102
load_current:      0.5
load_delta:        0.0
inductor_current:  1.0
inductor_delta:    0.0
vkp_value:  nan.0 | 132.442□ | 99.0284□ | 99.0275□ | 94.9961□ | 94.9960□ | 93.4128□ | 53.2922058105
vki_value:  nan.0 | 5.33333□ | 7.66681□ | 7.66679□ | 7.78104□ | 8.15392875671387
ikp_value:  nan.0 | -15.357□ | -6.1368□ | -1.1955□ | 2.75407□ | 5.64094495773315
iki_value:  nan.0 | 0.93992□ | 0.09104□ | 0.09093□ | 0.09577□ | 0.10822556912899
```

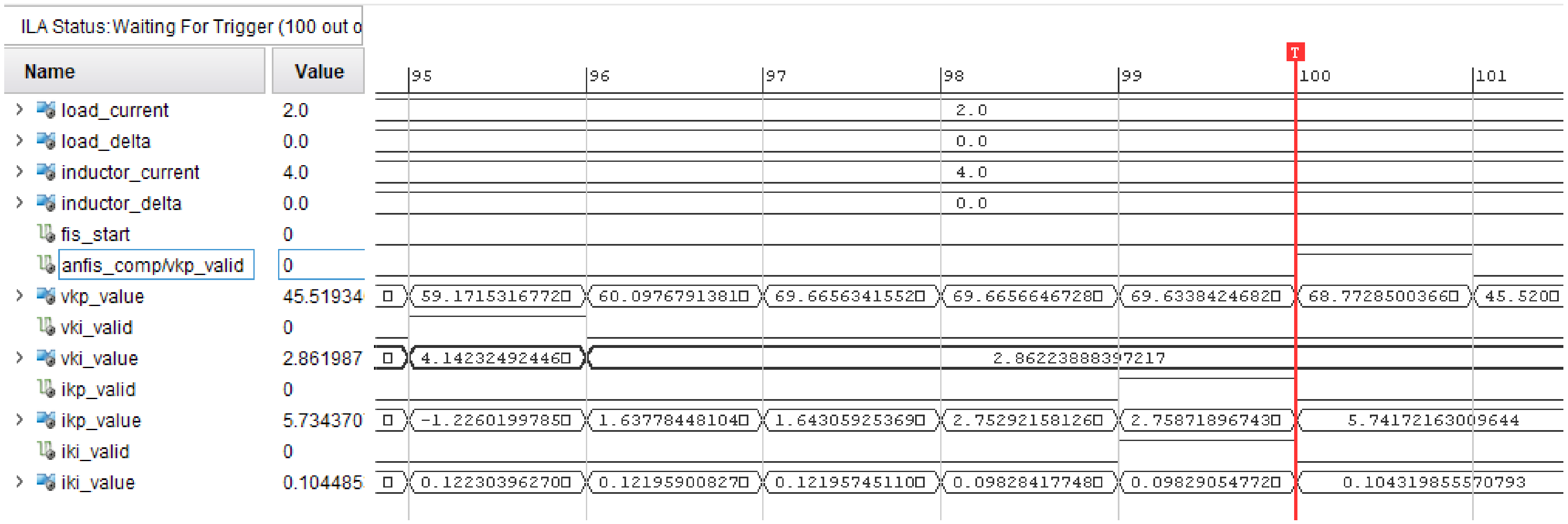


ILA Status:Waiting For Trigger (100 out o

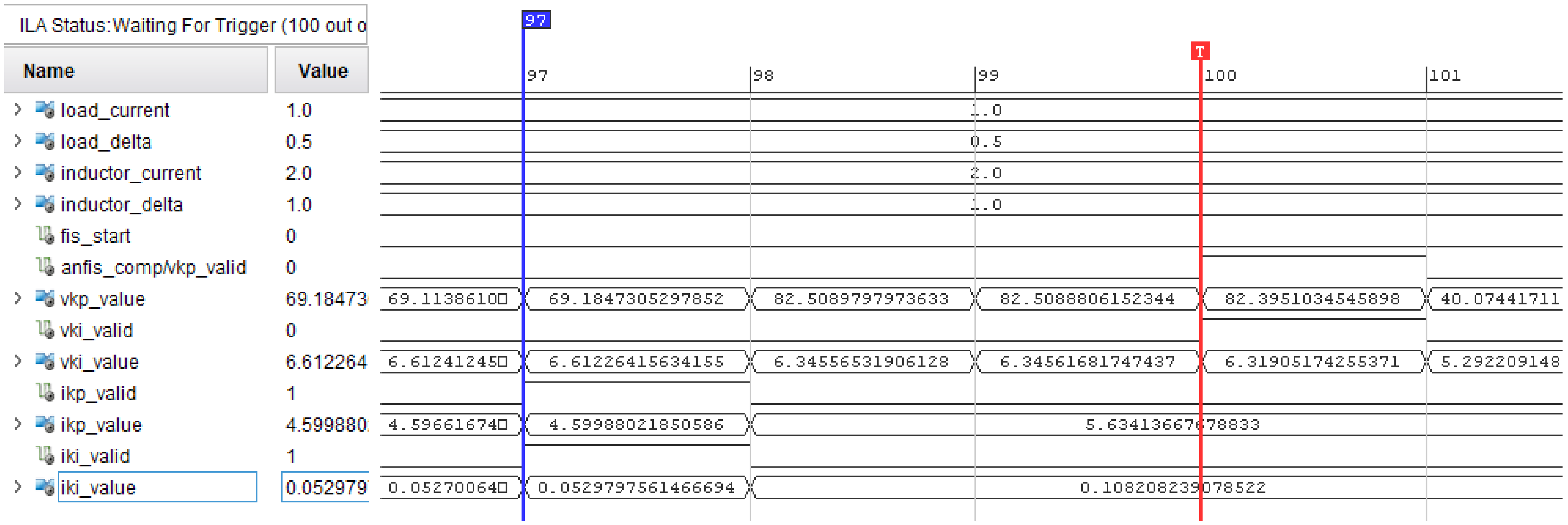
Name	Value					
			98	99	T 100	101
load_current	1.0	1.0				
load_delta	0.0	0.0				
inductor_current	2.0	2.0				
inductor_delta	0.0	0.0				
fis_start	0					
anfis_comp/vkp_valid	0					
vkp_value	50.70222	84.96	86.231803894043	86.230339050293	84.8833541870117	50.70222854€
vki_valid	0					
vki_value	6.389921	6.617	6.56379127502441	6.38992118835449		
ikp_valid	0					
ikp_value	5.671847	-1.23	2.75653600692749	5.67184782028198		
iki_valid	0					
iki_value	0.106978	0.092	0.0968582108616829	0.106978982686996		



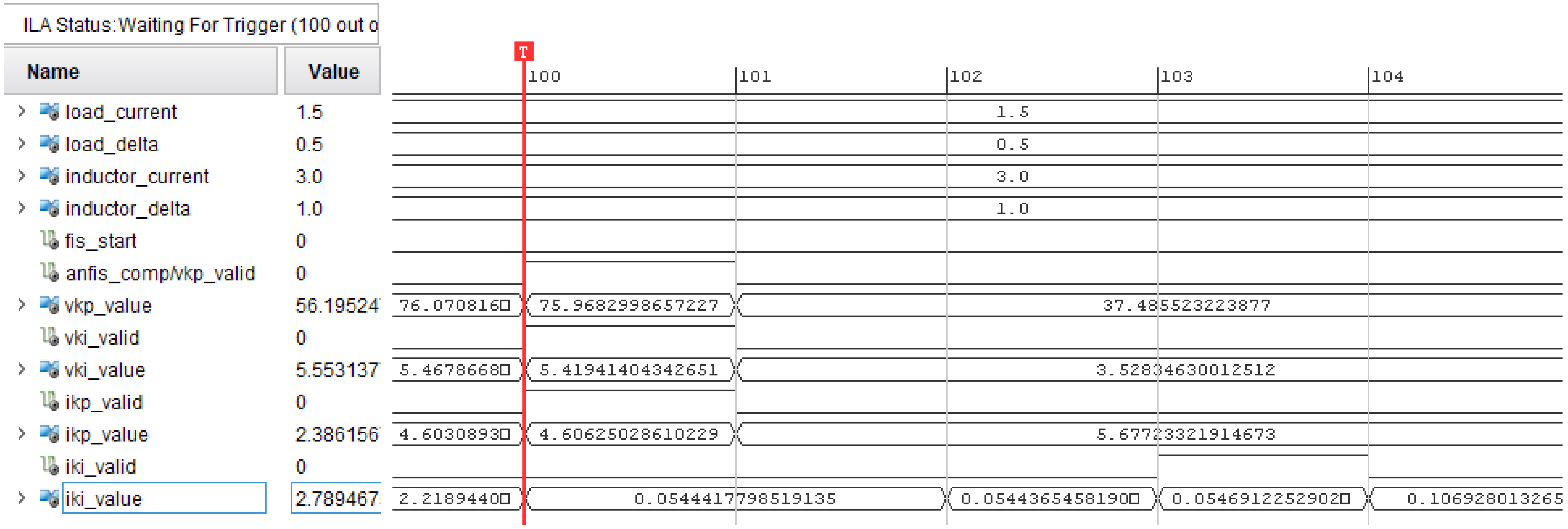
```
ILA Status:Waiting For Trigger (100 out o

Name                      Value
load_current              1.5
load_delta                0.0
inductor_current          3.0
inductor_delta            0.0
fis_start                 0
anfis_comp/vkp_valid      0
vkp_value                 77.59101
vki_valid                 0
vki_value                 4.626085
ikp_valid                 0
ikp_value                 -5.476217
iki_valid                 0
iki_value                 0.873083
```



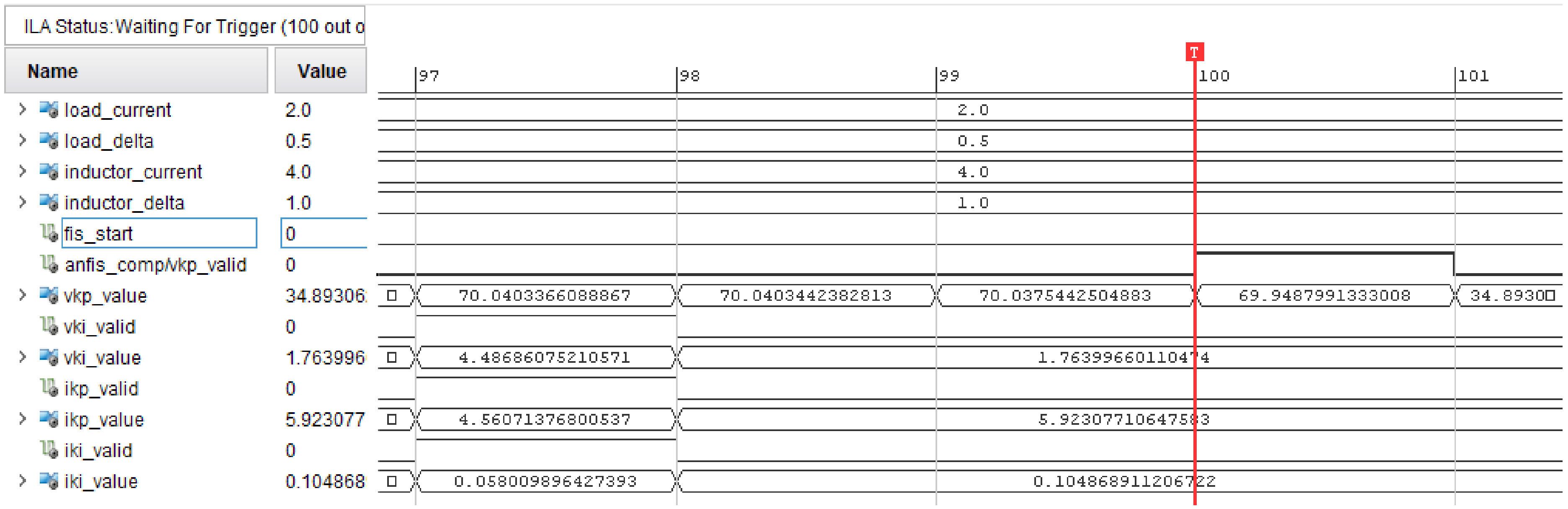
```
ILA Status: Waiting For Trigger (100 out o

Name                      Value       95                  96                  97                  98                  99                  T 100               101
load_current              2.0                                                                        2.0
load_delta                0.0                                                                        0.0
inductor_current          4.0                                                                        4.0
inductor_delta            0.0                                                                        0.0
fis_start                 0
anfis_comp/vkp_valid      0
vkp_value                 45.51934    □  59.1715316772□  60.0976791381□  69.6656341552□  69.6656646728□  69.6338424682□  68.7728500366□  45.520□
vki_valid                 0
vki_value                 2.861987    □  4.14232492446□                          2.86223888397217
ikp_valid                 0
ikp_value                 5.734370    □  -1.22601997850□  1.63778448104□  1.64305925369□  2.75292158126□  2.75871896743□  5.74172163009644
iki_valid                 0
iki_value                 0.104485    □  0.12230396270□  0.12195900827□  0.12195745110□  0.09828417748□  0.09829054772□  0.10431985570793
```


ILA Status: Waiting For Trigger (100 out o

Name	Value		97	98	99	100	101
load_current	1.0	1.0					
load_delta	0.5	0.5					
inductor_current	2.0	2.0					
inductor_delta	1.0	1.0					
fis_start	0						
anfis_comp/vkp_valid	0						
vkp_value	69.18473	69.1138610	69.1847305297852	82.5089797973633	82.5088806152344	82.3951034545898	40.07441711
vki_valid	0						
vki_value	6.612264	6.61241245	6.61226415634155	6.34556531906128	6.34561681747437	6.31905174255371	5.292209148
ikp_valid	1						
ikp_value	4.599880	4.59661674	4.59988021850586	5.63413667678833			
iki_valid	1						
iki_value	0.052979	0.05270064	0.0529797561466694	0.108208239078522			



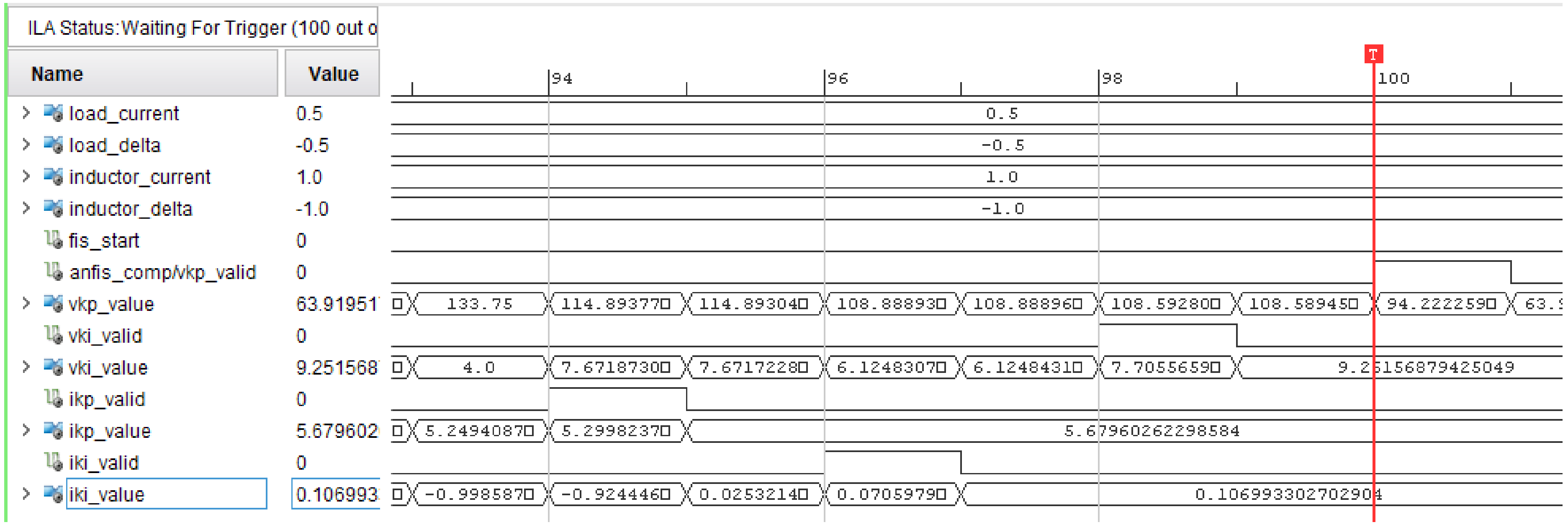
```
ILA Status: Waiting For Trigger (100 out o

Name                     Value        T 100                 101        102                    103                   104
load_current             1.5                                                1.5
load_delta               0.5                                                0.5
inductor_current         3.0                                                3.0
inductor_delta           1.0                                                1.0
fis_start                0
anfis_comp/vkp_valid     0
vkp_value                56.19524     76.070816□  75.9682998657227          37.485523223877
vki_valid                0
vki_value                5.553137     5.4678668□  5.41941404342651          3.52834630012512
ikp_valid                0
ikp_value                2.386156     4.6030893□  4.60625028610229          5.67723321914673
iki_valid                0
iki_value                2.789467     2.2189440□  0.0544417798519135  0.0544365458190□  0.0546912252902□  0.106928013265
```



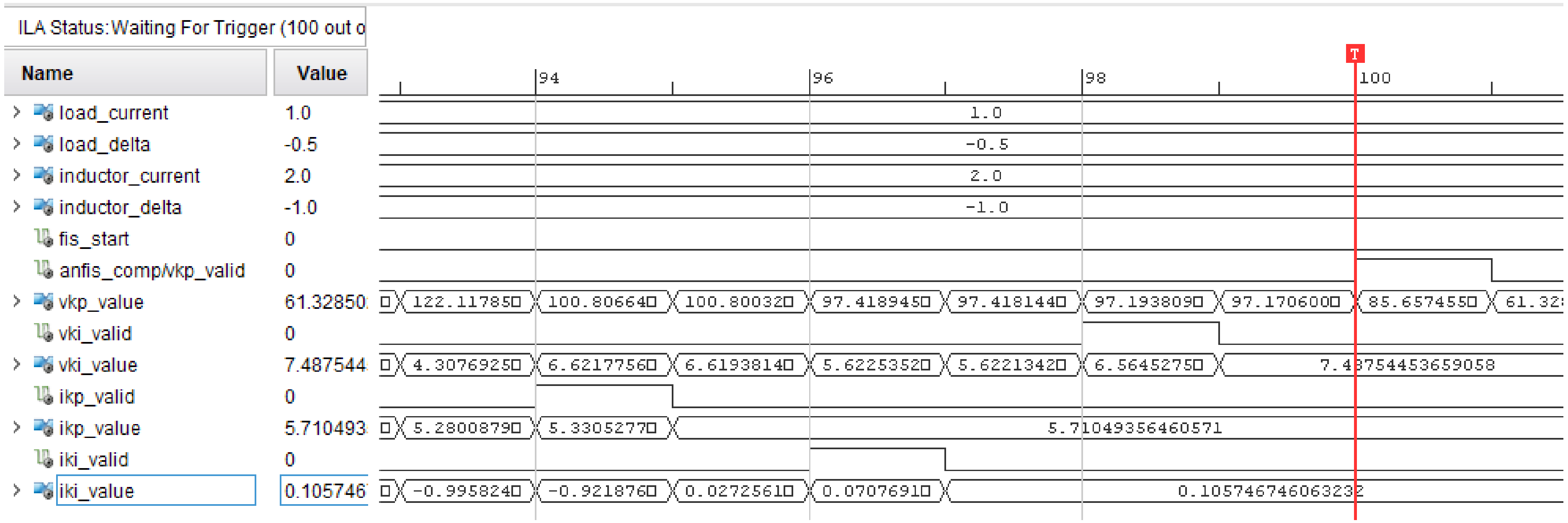
ILA Status: Waiting For Trigger (100 out o

Name	Value
load_current	2.0
load_delta	0.5
inductor_current	4.0
inductor_delta	1.0
fis_start	0
anfis_comp/vkp_valid	0
vkp_value	34.89306
vki_valid	0
vki_value	1.763996
ikp_valid	0
ikp_value	5.923077
iki_valid	0
iki_value	0.104868

```
97                     98                     99                     T 100                  101
                                              2.0
                                              0.5
                                              4.0
                                              1.0
  70.0403366088867      70.0403442382813      70.0375442504883      69.9487991333008      34.8930
  4.48686075210571                            1.76399660110474
  4.56071376800537                            5.92307710647583
  0.058009896427393                           0.104868911206722
```

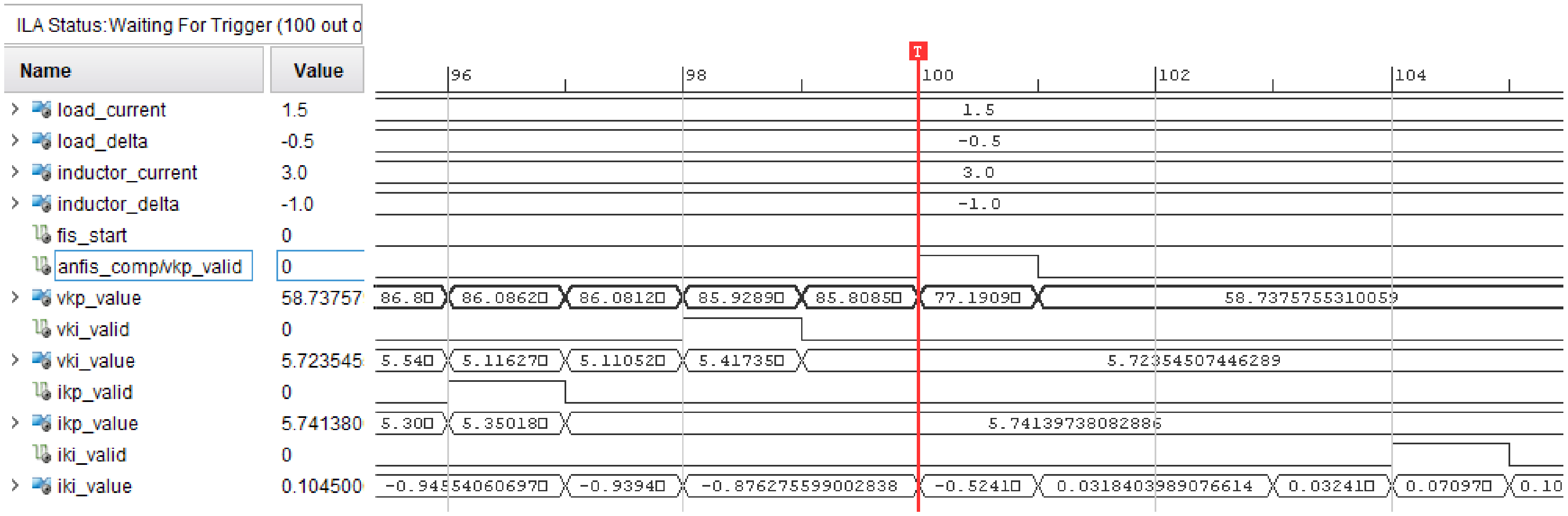

ILA Status:Waiting For Trigger (100 out o

Name	Value
load_current	0.5
load_delta	-0.5
inductor_current	1.0
inductor_delta	-1.0
fis_start	0
anfis_comp/vkp_valid	0
vkp_value	63.91951
vki_valid	0
vki_value	9.251568
ikp_valid	0
ikp_value	5.679602
iki_valid	0
iki_value	0.106993

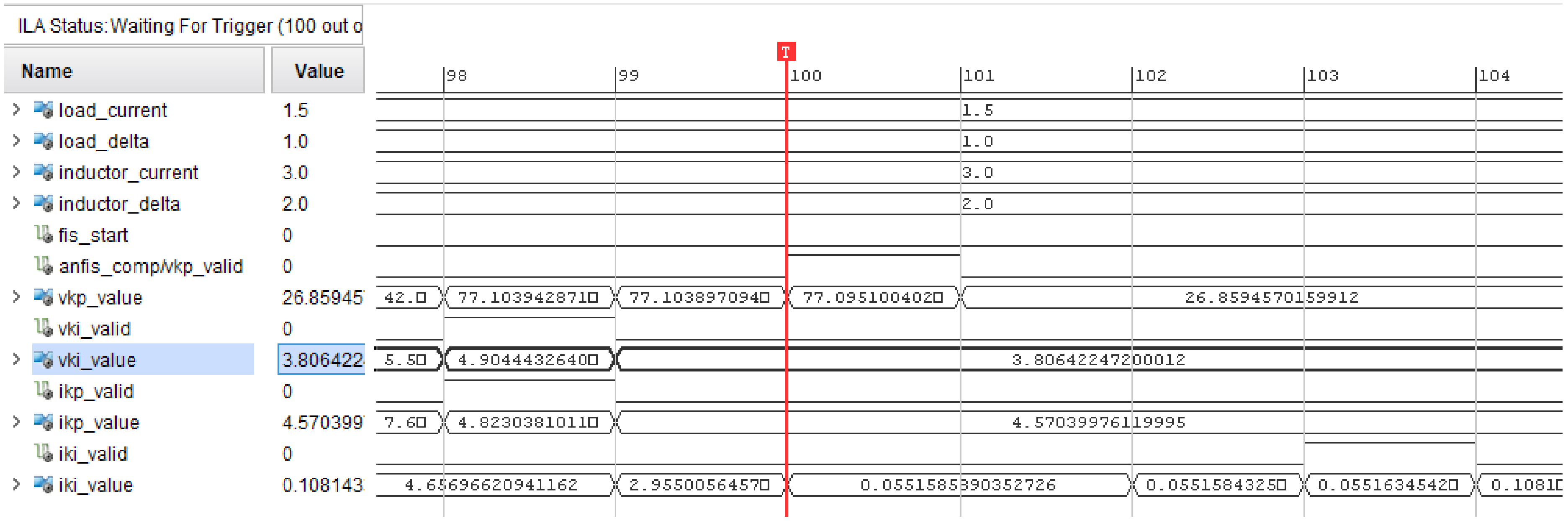

ILA Status: Waiting For Trigger (100 out o

Name	Value
load_current	1.0
load_delta	-0.5
inductor_current	2.0
inductor_delta	-1.0
fis_start	0
anfis_comp/vkp_valid	0
vkp_value	61.32850
vki_valid	0
vki_value	7.487544
ikp_valid	0
ikp_value	5.710493
iki_valid	0
iki_value	0.105746

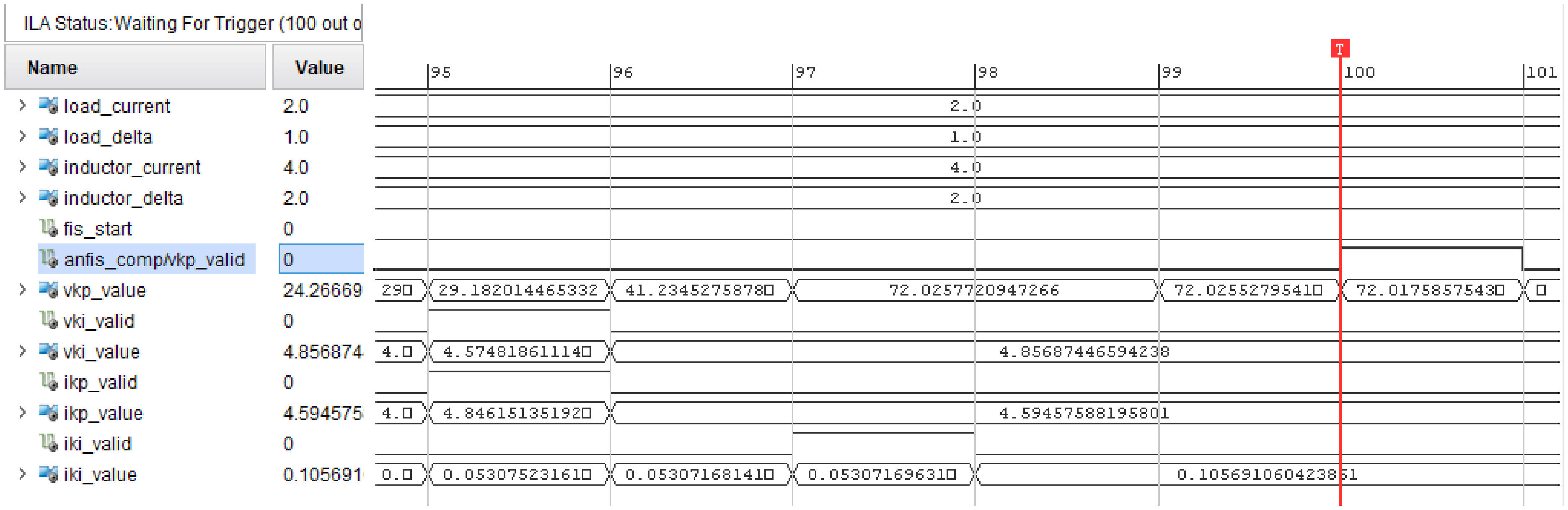
```
94        96        98        T 100
1.0
-0.5
2.0
-1.0
122.11785 | 100.80664 | 100.80032 | 97.418945 | 97.418144 | 97.193809 | 97.170600 | 85.657455 | 61.32
4.3076925 | 6.6217756 | 6.6193814 | 5.6225352 | 5.6221342 | 6.5645275 | 7.48754453659058
5.2800879 | 5.3305277 | 5.71049356460571
-0.995824 | -0.921876 | 0.0272561 | 0.0707691 | 0.105746746063232
```


ILA Status:Waiting For Trigger (100 out o

Name	Value
load_current	1.5
load_delta	-0.5
inductor_current	3.0
inductor_delta	-1.0
fis_start	0
anfis_comp/vkp_valid	0
vkp_value	58.73757
vki_valid	0
vki_value	5.723545
ikp_valid	0
ikp_value	5.741380
iki_valid	0
iki_value	0.104500

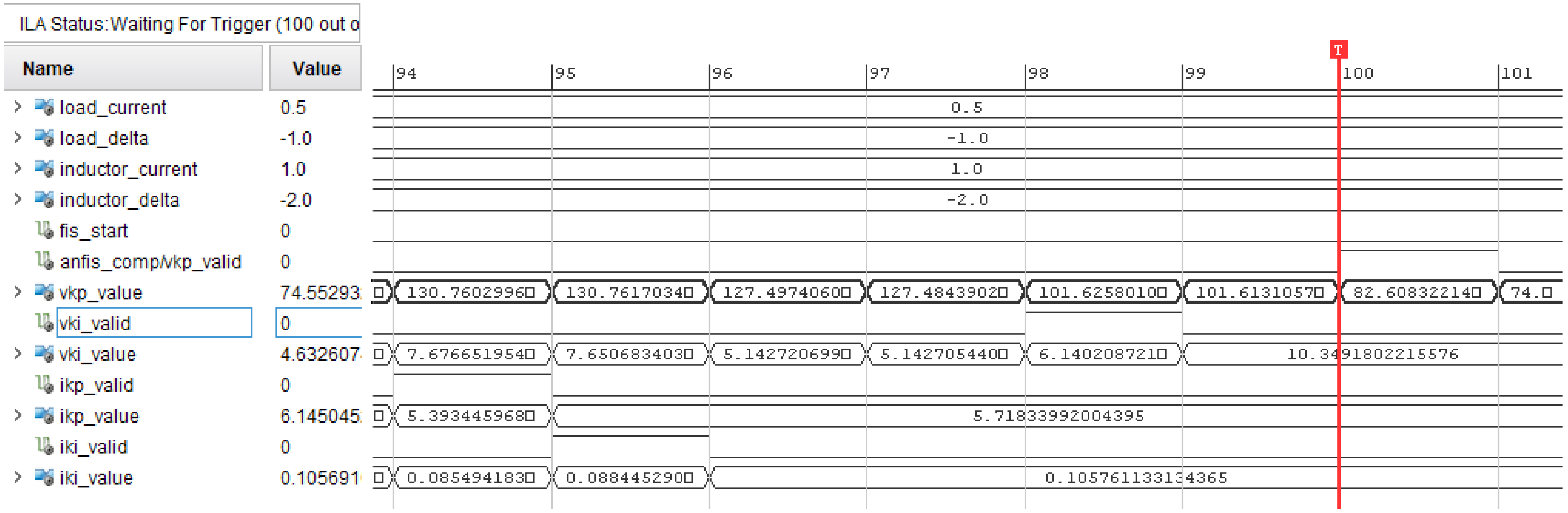


```
ILA Status: Waiting For Trigger (100 out o

Name                      Value       98            99            T 100          101     102     103     104
load_current              1.5         1.5
load_delta                1.0         1.0
inductor_current          3.0         3.0
inductor_delta            2.0         2.0
fis_start                 0
anfis_comp/vkp_valid      0
vkp_value                 26.85945    42.0 | 77.1039428710 | 77.1038970940 | 77.0951004020 | 26.8594570159912
vki_valid                 0
vki_value                 3.806422    5.50 | 4.9044432640 | 3.80642247200012
ikp_valid                 0
ikp_value                 4.570399    7.60 | 4.8230381011 | 4.57039976119995
iki_valid                 0
iki_value                 0.108143    4.65696620941162 | 2.9550056457 | 0.0551585390352726 | 0.0551584325 | 0.0551634542 | 0.1081
```



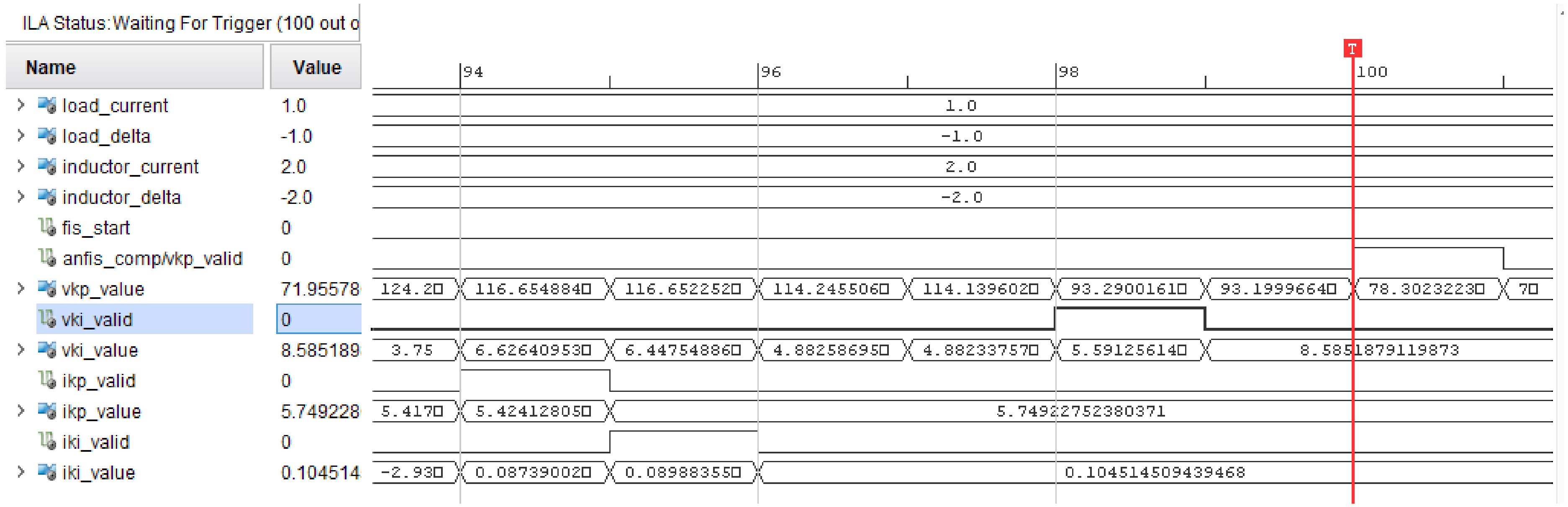
ILA Status: Waiting For Trigger (100 out o

Name	Value
load_current	2.0
load_delta	1.0
inductor_current	4.0
inductor_delta	2.0
fis_start	0
anfis_comp/vkp_valid	0
vkp_value	24.26669
vki_valid	0
vki_value	4.856874
ikp_valid	0
ikp_value	4.594575
iki_valid	0
iki_value	0.105691


ILA Status: Waiting For Trigger (100 out o

Name	Value
load_current	0.5
load_delta	-1.0
inductor_current	1.0
inductor_delta	-2.0
fis_start	0
anfis_comp/vkp_valid	0
vkp_value	74.55293
vki_valid	0
vki_value	4.632607
ikp_valid	0
ikp_value	6.145045
iki_valid	0
iki_value	0.105691

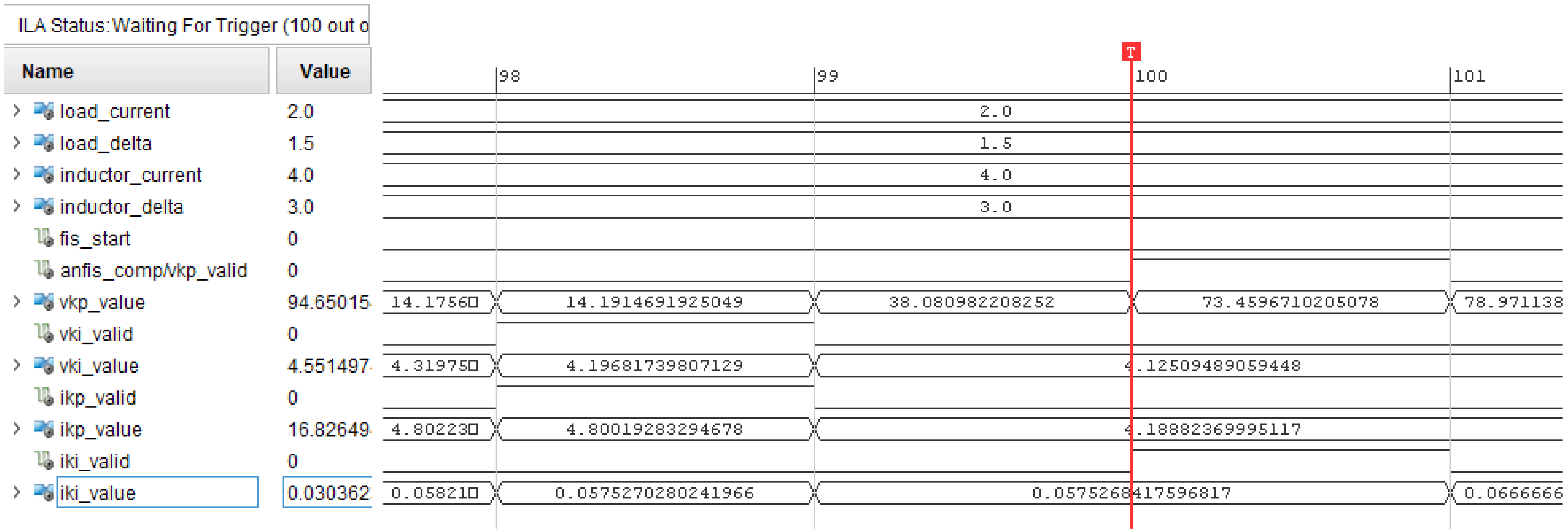
```
94  95  96  97  98  99  T 100  101
load_current      0.5
load_delta        -1.0
inductor_current  1.0
inductor_delta    -2.0
vkp_value   130.7602996  130.7617034  127.4974060  127.4843902  101.6258010  101.6131057  82.60832214  74.0
vki_value   7.6766519540  7.6506834030  5.1427206990  5.1427054400  6.1402087210  10.3491802215576
ikp_value   5.3934459680  5.71833992004395
iki_value   0.0854941830  0.0884452900  0.10576113314365
```



ILA Status: Waiting For Trigger (100 out o

Name	Value
load_current	1.0
load_delta	-1.0
inductor_current	2.0
inductor_delta	-2.0
fis_start	0
anfis_comp/vkp_valid	0
vkp_value	71.95578
vki_valid	0
vki_value	8.585189
ikp_valid	0
ikp_value	5.749228
iki_valid	0
iki_value	0.104514

```
94        96        98        T 100
1.0
-1.0
2.0
-2.0
124.2□  116.654884□  116.652252□  114.245506□  114.139602□  93.290161□  93.199664□  78.302322□  70□
3.75  6.626409530□  6.44754886□  4.88258695□  4.88233757□  5.59125614□  8.5851879119873
5.417□  5.42412805□  5.74922752380371
-2.93□  0.08739002□  0.08988355□  0.104514509439468
```


ILA Status:Waiting For Trigger (100 out o

Name	Value
load_current	2.0
load_delta	1.5
inductor_current	4.0
inductor_delta	3.0
fis_start	0
anfis_comp/vkp_valid	0
vkp_value	94.65015
vki_valid	0
vki_value	4.551497
ikp_valid	0
ikp_value	16.82649
iki_valid	0
iki_value	0.030362

Appendix G – Published Materials

Darvil, J., Tisan, A. and Cirstea, M.N., 2014. *A Novel PSIM and Matlab Co-Simulation Approach to Particle Swarm Optimization Tuning of PID Controllers. IEEE* International Conference on Optimization of Electrical and Electronic Equipment, pp. 784-789.

Darvil, J., Tisan, A. and Cirstea, M.N., 2015. *An ANFIS-PI Based Boost Converter Control Scheme.* IEEE Conference on Industrial Informatics, pp. 632-639

Darvil, J., Tisan, A. and Cirstea, M.N., 2017. *A Novel ANFIS Algorithm Architecture for FPGA Implementation*, IEEE International Symposium on Industrial Electronics, pp. 1243-148.

A Novel PSIM and Matlab Co-Simulation Approach to Particle Swarm Optimization Tuning of PID Controllers

John Darvill, Alin Tisan, Marcian Cirstea
Anglia Ruskin University, Cambridge, UK
john.darvill@student.anglia.ac.uk, alin.tisan@anglia.ac.uk, marcian.cirstea@anglia.ac.uk

***Abstract*-Particle swarm optimization (PSO) has proven to be a particularly popular evolutionary algorithm for the tuning of PID controllers, due to its relative simplicity and suitability as a function approximator. This paper presents a new modeling technique, featuring co-simulation between Matlab and PSIM, a specialist control and power electronics simulation environment. The tuned PID is applied to a buck converter, driven by a PV cell. The novel solution is compared with and evaluated against a system which is tuned using a traditional method. The simulation results indicate that the controller developed using the PSO algorithm achieves a faster response time without adversely affecting peak errors. There are added benefits of this new approach, in terms of a shorter time scale for conceptual model development.**

I. INTRODUCTION

The classical PID controller was first presented in 1922 by Minorsky [1], although its first use is widely attributed to Elmer Sperry as early as 1911. The classical PID controller has proven to be consistently popular across a range of different control applications and remains one of the most widely used control solutions even today. Much of this popularity comes from the algorithm offering both flexibility and robustness, whilst being relatively simple to implement, especially in the modern era where numerous tools are available for quick and easy modeling.

Although the initial publications on the PID controller established a strong theoretical basis for numerical analysis of the algorithm, the issue of selecting appropriate branch weightings wasn't addressed. The issue of parameter selection was first addressed by Ziegler and Nichols in two important papers, [2] and [3], which established how both the open loop and closed loop response of the system could be used to achieve optimization. Although these methods are highly manual and involve placing the system in an unstable state, they are nonetheless highly effective and the Ziegler-Nichol method remains one of the most widely used approaches. There have been several other tuning processes which have been proposed and whilst some, such as the Cohen-Coon method [4], have gained some popularity, none are as popular as the Ziegler-Nichols approach.

A more recent development in the field of PID tuning stems from the advent of the evolutionary algorithm, starting with the genetic algorithm (GA) in the 1960s. This algorithm, coupled with the increasing availability of affordable fast processing power, has seen a rise in a new approach to the tuning of the PID controller. The GA is intended to mimic the behavior of genomes, evolving and mutating potential solutions within a problem search space until an optimal solution is found. This makes the algorithm an ideal fit for optimization problems such as PID tuning and subsequently the GA has found use in this application as shown in [5] and [6]. Unlike the Ziegler-Nichols method, the GA approach is a highly automated process and results show favorable performance in comparison to the traditional Ziegler-Nichols method [7].

In 1995, Kennedy and Eberhart introduced a new evolutionary algorithm, known as Particle Swarm Optimization (PSO) [8], which models the behaviour of animal flocking patterns, such as those seen in migrating birds. This algorithm is similar to the GA as it evolves a population of potential solutions in a search space until an optimal solution to a given problem is found, making it a powerful tool for optimization problems such as PID tuning. Despite being a reasonably new development, this method has been quick to gain popularity as it has comparable performance to the GA approach, whilst being simpler to implement. Another major advantage of the PSO is the ability to be simply used on numeric values rather than binary strings, meaning popular tools such as Matlab can be simply used to develop the algorithm. As with the GA, this algorithm has found much use in PID tuning, as shown in [9] and [10]. The work presented in [10] shows that the results offer improved performance over the Ziegler-Nichols method.

In this paper, a new modeling solution is presented, using co-simulation between a specialist modeling tool called PSIM and the popular Matlab software. This approach takes advantage of the simple mathematical syntax used in Matlab, whilst the schematic capture nature of the PSIM environment allows for a rapid creation of the circuit model. The tuned PID algorithm is applied to a buck converter, which is used to regulate the output of a solar panel. The rest of this paper is organized as follows: firstly the control problem is presented, then the concept of PID tuning using the PSO is introduced, next the system model is discussed before simulations are presented and finally conclusions are drawn to end the paper.

II. Control Problem

The main goal of the controller discussed in this paper is to provide a robust DC-DC buck converter. This power converter will take an unstable, unregulated input, which is provided by the BP3230N solar cell and converts this down to a stable 24V DC output. The solar panel is rated at 230W, with an 8A current at maximum power. A diagram of the full circuit is given in Fig 1, showing the buck converter, a feedback path and the PID controller. The output is controlled by the switching action of the transistor, S, which is driven using pulse width modulation (PWM). In order for the circuit to maintain a well regulated output, the duty cycle of the switch must be varied in accordance with the input voltage and the load. The error from the converter is fed back to the PID controller, which then controls the duty cycle from the PWM module to give a 24V output. Once tuning is complete, the PSO block in the controller will no longer be part of the system and the PID controller parameters will be hard coded.

A. *Mathematical Model of the Buck Converter*

The performance of the buck converter circuit, shown in Fig 1, is influenced heavily by the current through the inductor, L. When the inductor isn't allowed to fully discharge, known as continuous conduction mode, the equations modeling the behavior of the circuit are simple to understand. The other mode of operation occurs when the inductor is allowed to fully discharge for some period, also known as discontinuous conduction. In this mode of operation, the equations governing the behavior of the circuit become much more complex, meaning it is preferential to keep the converter running in continuous conduction mode. The average output of the converter is most heavily defined by the duty cycle of the switch, which can be defined by the ratio of the input voltage to the output voltage or of the period that the switch is closed to the total switching period as shown in Eq (1). This equation shows how the output can be easily controlled by adjusting the duty ratio of the switch when in continuous conduction mode.

$$D = \frac{V_{out}}{V_{in}} = \frac{T_{on}}{T_s} \tag{1}$$

There are two equations which are needed to understand the behaviour of the inductor in the converter: one for the voltage and one for the ripple current. The equation used to model the voltage across the inductor is a simple function of inductance and change in current, as shown in Eq. (2). The ripple current of the inductor is important in the modeling of the buck converter, as keeping this value to a minimum helps to prevent the converter from entering the discontinuous conduction mode of operation. The ripple current of the inductor can be modelled by the equation given in Eq. (3), where f_s is the transistor switching frequency.

$$V_L(t) = L\frac{di}{dt} \tag{2}$$

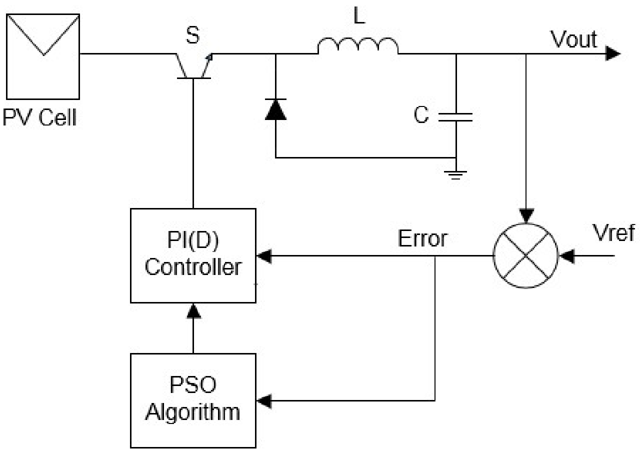


Fig 1 Diagram of the system being developed

$$\Delta I_L = \frac{(\mathrm{Vin} - \mathrm{Vout}).\mathrm{D}}{f_s.L} \tag{3}$$

Another defining feature of the buck converter circuit is the low pass filter, formed by the capacitor and the inductor at the output. This filter is an important feature of the converter as it serves to remove the fluctuations which are introduced by the action of the switch. Therefore, the smoothing action of this filter has a large bearing on the quality of the power produced, as it controls the amount of ripple seen on the output. The overall ripple of the circuit can be modeled by the equation shown in Eq. (4), with f_c being the cut-off value of the filter and f_s being the switching frequency. In order to keep the output ripple to a minimum, it is essential then that the cut-off of the low pass filter is significantly less than the switching frequency of the converter.

$$Ripple = \frac{\pi}{2}(1-D)(\frac{f_c}{f_s})^2 \tag{4}$$

B. *PID Controller*

The PID controller is a widely used controller, firstly introduced in the early parts of the 20th century. This controller consists of three separate branches - a proportional branch, an integral branch and a derivative branch. The three terms of the controller serve to respond to different elements of the error signal, with the proportional part acting immediately on the current error, the integral adding a contribution equivalent to the history of all errors and the derivative action predicting the future contribution of the errors. The addition of the integral and derivative action to the basic proportional enhance the performance of the controller by reducing steady state errors and the rise/fall times respectively. Eq. (5) shows the basic PID algorithm, which is used in the controller, with e being the output error. In order to achieve optimal performance in any application, there are three parameters which must be manually tuned: K_p, T_r and T_d. K_p is the proportional gain of the system, T_r, known as reset time, controls the amount of integral action which is applied to the system, whilst T_d controls the amount of derivative action applied by the controller. By tuning these parameters, it is clear that the influence of each of the three branches can be altered to give the best performance for any given plant.

$$K_p.e + \frac{1}{T_r}.\int e(t).dt + T_d.\frac{de(t)}{dt} \qquad (5)$$

III. PSO ALGORITHM

Originally introduced in 1995 by Eberhart and Kenedy [8], the PSO algorithm is a popular tool for solving optimization problems, making it a good fit for solving the PID tuning problem. The main flow of the algorithm is straight forward, consisting of just three stages, as shown by the flowchart presented in Fig 2.

At the heart of the PSO algorithm is the particle which consists of three parts – the current potential solution, the velocity and the personal best fit. The current potential solution stores the data which is to be applied to the problem, whilst the personal best fit stores the value of the particle which has yielded the best result so far. The velocity is a key parameter in controlling how the problem space is searched as it controls how fast and in which direction the particle is accelerated. This value is set to a random value during the initialisation of the algorithm but is iteratively updated to pull the particle towards either a global or personal best fit. The equation for updating the velocity of the particle is shown in Eq. (6), with V_n being current velocity, X_n being the current solution, GX_n the global best fit and PX_n the personal best fit.

$$\begin{aligned} V_{N+1} = \omega.V_N &+ (C_1.Rand.(GX_N - X_N)) \\ &+ (C_2.Rand.(PX_N - X_N)) \end{aligned} \qquad (6)$$

The equation shown in Eq. (6) highlights the fact there are two constants which are important in controlling how the particle moves through the search space. The social and cognitive coefficients, shown by C_1 and C_2 respectively, have the effect of pulling the particle towards either its personal best fit (cognitive) or the global best fit (social). These coefficients therefore control how much the algorithm converges towards a global maximum or searches around promising potential solutions. A good balance between the two is crucial to the optimal performance of the algorithm. A number of different methods have been used and proposed, from statically setting the variables to using complex algorithms to dynamically alter them. One popular method is discussed in [12], with the cognitive value linearly decreasing from 2.5 down to 0.5, whilst the social value is linearly increased from 0.5 up to 2.5. This has been demonstrated to give good results and is the method which will be used for this tuning procedure. The two variables will be set according to the equations shown in Eq. (7) and Eq. (8), with C_{min} and C_{max} representing the minimum and maximum values of the coefficients.

$$C_1 = c_{\max} - (\frac{c_{\max} - c_{\min}}{iter_{\max}}) \times iteration \qquad (7)$$

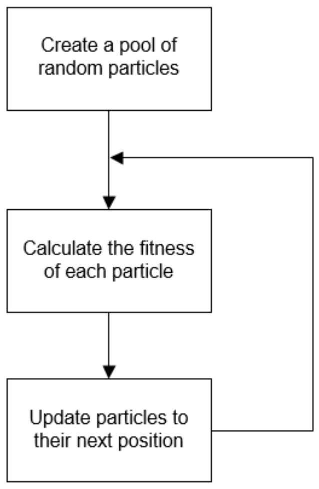

Create a pool of random particles
Calculate the fitness of each particle
Update particles to their next position

Fig 2 Flow Chart for the PSO algorithm

$$C_2 = c_{\min} + (\frac{c_{\max} - c_{\min}}{iter_{\max}}) \times iteration \qquad (8)$$

One more important parameter in Eq. (6) is the inertia weight, ω. Inertia weight was a later introduction to the algorithm [13] and serves the purpose of reducing stagnant particles and promoting convergence to the best solution. Setting the inertia weight to a high value promotes a wider search space, giving the algorithm the opportunity to look at numerous potential solutions. Setting the value to a smaller value, on the other hand, forces the algorithm to search around a smaller solution area in effect giving an element of fine tuning to the algorithm. As it is beneficial to initially have a large search space before reducing the search area to help find a true optima, a popular strategy is to begin with a large value of inertia weight and then reduce this as the algorithm nears completion. According to[14] and [15], the best solution is to linearly decrease the inertia weight between the values of 0.9 and 0.4, according to the equation shown in Eq. (9) and this is the method which will be used with this implementation.

$$\omega = \omega_{\max} - \frac{\omega_{\max} - \omega_{\min}}{iter_{\max}} \times iteration \qquad (9)$$

The PSO algorithm also requires an additional function to evaluate how well the potential solution solves the problem being explored. This measure is referred to as fitness and there are a number of different methods used to measure this. One of the most popular approaches is to use the mean square error (MSE), which involves taking the mean of all errors squared. This method is straight forward to implement whilst also offering a good way of measuring the quality of the fit and for this reason this is the method which will be used here. The MSE algorithm is shown in Eq. (10).

$$MSE = \frac{1}{n}\sum_{x=1}^{n} E_x^{\,2} \qquad (10)$$

IV. SYSTEM MODELING

A. *Modeling Environment*

There are a number of different tools which can be used for the modeling of the PSO tuned PID controller but the most commonly used solution for power and control engineers is Matlab. There are many benefits to this approach for the implementation of the PSO algorithm, such as the ease of implementation and the computational power. This environment also has the ability to create models graphically, through the use of the Simulink plug-in, allowing for rapid creation of a system model.

Despite the strengths of Matlab, there are some limitations in comparison to other tools for this application. One big drawback is the need to create a mathematical model for the solar cell. Although the performance and mathematical equations of the solar cell are well known, the development of an accurate model is still a time consuming process. This is in contrast to the PSIM software suite, where a generic model of a solar cell is included. This can be quickly customized to fit any given solar cell using simply a few parameters from the data sheet, which can significantly reduce development time. PSIM also has the ability to quickly present IV graphs of the cell, so that a comparison can be made with the model and the data presented in the data sheet. In addition to this, the Spice like nature of PSIM allows for a quick and simple approach to modeling electronics components, which is in contrast to the tools offered in Matlab. The use of the PSIM environment also enables the future integration of VHDL code, as co-simulation with the popular Modelsim simulation tool is supported. With FPGAs becoming increasingly popular in industrial electronics applications [16], this takes into consideration a final hardware target.

In this paper, the Simcoupler module, which is a plug in for the PSIM environment, will be used, allowing for co-simulation between PSIM and Matlab. This module automatically creates an “s-function” block from the PSIM schematic for use within a Simulink system model. This means that a drop in block, which contains the power electronic and solar model, can be quickly developed and added to any Matlab model. This enables the exploitation of the rapid prototyping elements of the PSIM tool, whilst the superior algorithm development capabilities of Matlab are retained. Once the PID has been tuned, the system can then be fully simulated to validate the performance of the PSO in comparison to the Ziegler-Nichols method.

B. *Circuit Model*

The main buck converter circuit model is created using the PSIM software, with the capacitor selected as 47uF and the inductor selected as 120uH, resulting in a 2kHz cut off for the output filter. The switching frequency of the transistor is selected as 100 kHz, a frequency which allows for the selection of a smaller inductor, whilst not placing undue stress on the transistor. The switching frequency also benefits from being significantly larger than the cut off of the output filter, reducing the output ripple voltage.

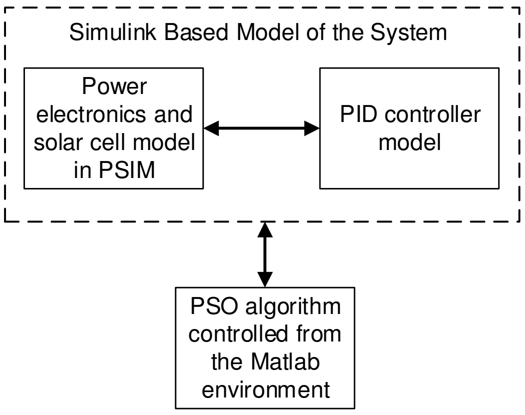

Simulink Based Model of the System
Power electronics and solar cell model in PSIM
PID controller model
PSO algorithm controlled from the Matlab environment

Fig 3 The tool flow used for modeling of the system

C. *Model of the Solar Panel*

As has already been discussed, one of the great benefits of PSIM is the ability to quickly model the solar panel which is used in this control problem. The generic solar cell model can be customized to the individual application using a few parameters from the data sheet, such as rated power, series resistance, shunt resistance and short circuit current. The solar cell model used in this paper was quickly developed and the IV curves where obtained using the verification tool in PSIM, as shown in Fig 4. These indicate that the model of the PV cell closely matches the behavior defined in the data sheet for the BP3230N solar cell.

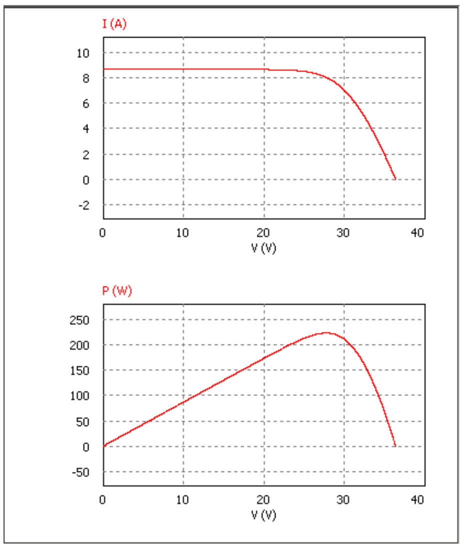


Fig 4 BP3230N IV Curve from PSIM

V. SIMULATION RESULTS

The simulations are preceded by the tuning of the two controllers using both the PSO algorithm and the Ziegler Nichols method. In both instances the tuning of the circuit is conducted with an 8A load, corresponding to the maximum current the solar cell can deliver. The results of both of these operations are presented in Table I, and these parameters represent the ones which are used for the experiments presented in this part of the paper.

TABLE I

PARAMETERS OBTAINED FROM THE TUNING METHODS

Tuned Parameters			
Tuning Method	**Kp**	**Ti**	**Td**
Ziegler-Nichols	7.5	500E-6	125E-6
PSO	10	8.9E-4	9.4E-5

In order to establish the performance of the two PID controllers, there are four tests which are carried out. Firstly, the static characteristics of the two systems are measured after the system has reached a steady state following the initial ramp up, as shown by the waveforms in Fig 5. The next simulation examines the dynamic characteristics of the system by measuring the rise time, settling time and overshoot of the system in response to a transient as shown in Fig 6. The final sets of simulations measure how well the system responds to disturbances in the operating conditions by applying a one amp load disturbance followed by a 10% input voltage disturbance. The results of these two disturbance simulations are shown in Fig 7 and Fig 8. The key performance indicators of these simulations are tabulated and presented in Table II.

The steady state characteristics of both systems are examined in the simulation waveform shown in Fig 5 and in this instance both offer identical, excellent performance. The output ripple voltage of both converters is measured at just 8mV and likewise the steady state error of both is 0%. This shows that in both instances the tuning has been able to achieve an excellent level of fitness for the steady state operation of the system.

The initial transient response of the system demonstrates that the PSO tuned controller exhibits stronger stability than the Ziegler Nichols tuned controller. This is shown by a

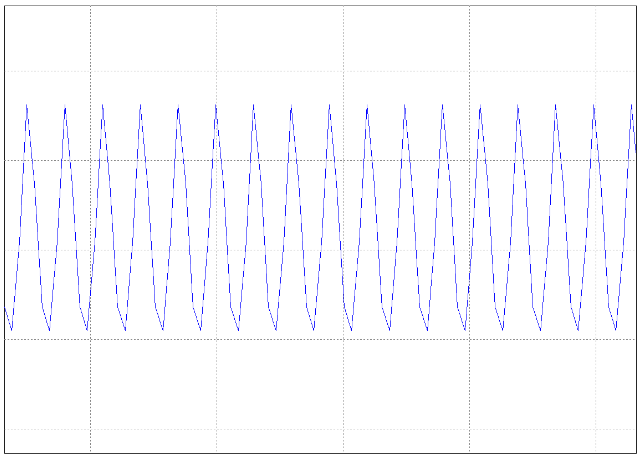


Fig 5 Ripple voltage on the output of the converter (same for both tuning methods)

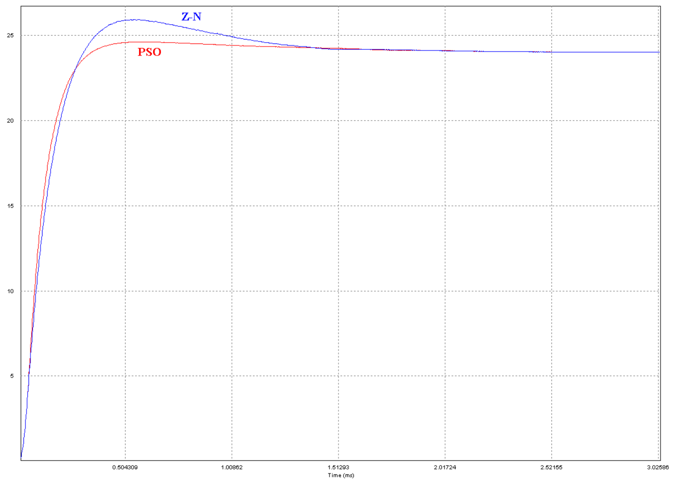


Fig 6 Output voltage of the two converter during start up

TABLE II

SIMULATION RESULTS

Initial Ramp Up Performance		
Tuning Method	**Z-N**	**PSO**
Rise Time (us)	230	210
Settling Time (ms)	1.42	1.5
Overshoot (%)	8	2.3
Steady State Performance		
Tuning Method	**Z-N**	**PSO**
Ripple (mV)	8	8
Steady State error (%)	0	0
Input Voltage Disturbance Performance		
Tuning Method	**Z-N**	**PSO**
Max output error (mV)	305	313
Peak error time (us)	247	188
Setling time (us)	470	440
Load Disturbance Performance		
Tuning Method	**Z-N**	**PSO**
Peak error (mV)	289	289
Peak error time (us)	24	24
Settling time (us)	46	36

reduction in the overshoot of the output voltage from 8% to just 2.6%. The PSO tuned algorithm also benefits from a quicker rise time by around 20µs. Despite this, the settling time of the Ziegler Nichols method, to within 1% error, is slightly faster, which is down to the increased derivative action of the PSO controller. This has the effect of slowing the rate of change during moments when the output is decreasing from a maximum error. Despite this, both converters reach a state of 0% error at the same time.

The final set of simulations to consider is the disturbance tests for both the load and input voltage. In the case of the load disturbance test, both controllers exhibit excellent performance, with a peak error of just over 1% in both instances. The PSO tuned controller again has the more favorable performance of the two controllers, with a settling time of just 36us, which is quicker than the Ziegler Nichols controller time of 46us. In the case of the input voltage disturbance test, the PSO algorithm is again the faster of the two converters, with a faster time to peak error and a quicker settling time. This again demonstrates the speed of the PSO controller compared to the Ziegler Nichols method, which can be attributed to the increased proportional action. By allowing an increased proportional gain, the controller

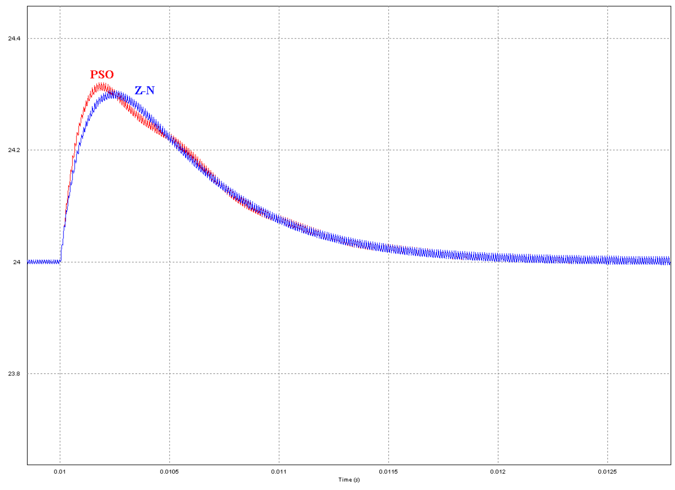


Fig 7 Output voltage of the converters with an input voltage disturbance

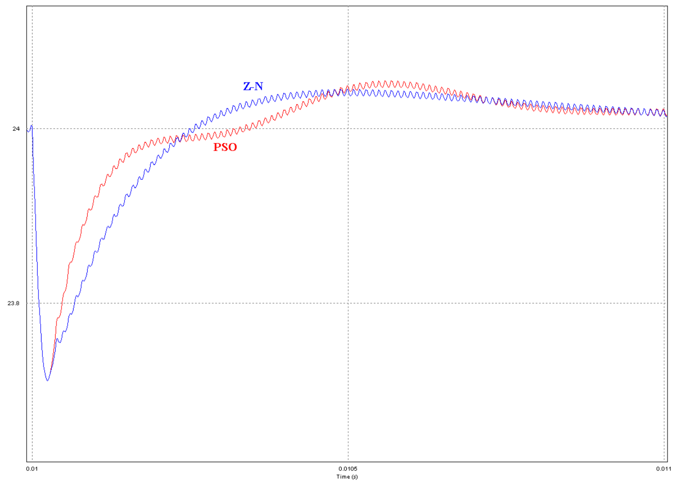


Fig 8 Output voltage of the converters during a load disturbance

becomes more sensitive to fluctuations and is therefore faster to respond to them. One downside to this increased gain is the Ziegler Nichols method, offering a marginally smaller peak error in this simulation.

VI. CONSLUSIONS

This paper demonstrates how co-simulation between the PSIM and Matlab tools can be used to tune a PID controller using the PSO algorithm. The simulation results indicate that the use of this method achieves an improved stability and a faster response in comparison to traditional methods. Further to this, the paper also demonstrates how the use of PSIM rather than the Simulink tool for the models of the power converter and solar panel can result in a reduced development cycle. The PSIM software also allows for the final hardware target to be considered within the system model, using simulations with either VHDL or C, giving further confidence in the design. This shows that the use of co-simulation between the PSIM and Matlab environments is a convenient approach to rapid modelling and prototyping of power electronics systems, particularly related to the field of renewable energy.

REFERENCES

[1] N. Minorsky, “Directional stability of auto-matically steered bodies.” *J. Amer. SOC .of Naval Engineers,* Vol. 34, pp. 280-309, May 1922
[2] J.G. Ziegler and N.B. Nichols, “Optimum settings for automatic controllers.” *Trans. ASME*, Vol. 64, pp. 759-768, 1942
[3] J.G. Ziegler and N.B. Nichols , “Process lags in automatic control circuits.” *Trans. ASME*, Vol. 65, pp. 433-444, 1943
[4] G.H. Cohen and G.A. Coon, “Theoretical consideration of retarded control.” *Trans. ASME*, Vol. 75, pp. 827-834, 1953
[5] M.J.Neath, A.K.Swain, U.K.Madawala and D.J.Thrimawithana, “An Optimal PID Controller for a Bidirectional Inductive Power Transfer System Using Multiobjective Genetic Algorithm.” *IEEE Transactions on Power Electronics*, Vol. 29, pp. 1523-1531, March 2013
[6] H.M. Hasanien , “Design Optimization of PID Controller in Automatic Voltage Regulator System Using Taguchi Combined Genetic Algorithm Method.” *IEEE Systems Journal*, Vol. 7, pp. 825-831, Dec 2013
[7] M.Korkmaz, O.Aydogdu and H.Dogan, “Design and performance comparison of variable parameter nonlinear PID controller and genetic algorithm based PID controller.” *International Symposium on Innovations in Intelligent Systems and Applications*, pp. 1 -5, July 2012
[8] J. Kennedy and R.Eberhart, “Particle Swarm Optimization.” *International Conference on Neural Networks*, Vol 4, pp.1942, December 1995
[9] X. Li, M.Chen and Y.Tsutomu, “A method of searching PID controller's optimized coefficients for Buck converter using particle swarm optimization.” *IEEE Electric Power and Energy Conference*, pp.238, April 2013
[10] H.I. Jaafar, Z. Mohamed, A.F.Z Abidin and Z.A Ghani, “PSO-tuned PID controller for a nonlinear gantry crane system.” *IEEE International Conference on Control System, Computing and Engineering*, pp 515-519, November 2012
[11] J. Femmy Nirmal and D.Jeraldin Auxillia, “Adaptive PSO based tuning of PID controller for an Automatic Voltage Regulator system.” *International Conference on Circuits, Power and Computing Technologies*, pp.661, March 2013
[12] A. Ratnaweera, S.Halgamuge. and H.C.Watson, “Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients.” *IEEE Transactions on Evolutionary Computation*, Vol 8, pp 240, June 2004
[13] R.Eberhart and Y.Shi, “A Modified Particle Swarm Optimizer” *IEEE International Conference on Evolutionary Computation Proceedings*, pp. 67, May 1998
[14] J.C.Bansal, P.K. Singh, M.Saraswat, A. Verma, S.S Jadon and A.Abraham, “Inertia Weight strategies in Particle Swarm Optimization*.” Third World Congress on Nature and Biologically Inspired Computing*, pp. 67, October 2011
[15] M.A.A. Aziz, M.N. Taib and R. Adnan, “The effect of varying inertia weight on Particle Swarm Optimization (PSO) algorithm in optimizing PID controller of temperature control system”. *IEEE 8th International Colloquium on Signal Processing and its Applications*, pp. 545, March 2012
[16] E. Monmasson and M.N. Cirstea, “FPGA Design Methodology for Industrial Control Systems – A Review*” IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, August 2007
.

An ANFIS-PI Based Boost Converter Control Scheme

John Darvill, Alin Tisan, Marcian Cirstea
Anglia Ruskin University, Cambridge, UK
john.darvill@student.anglia.ac.uk , alin.tisan@anglia.ac.uk, marcian.cirstea@anglia.ac.uk

Abstract- **The PI algorithm has proven to be a popular and widely used control method, due to its relative simplicity and robustness. Despite this, the linear nature of the algorithm means it doesn't provide optimal control to non-linear systems. This paper presents a novel method of improving the performance of the PI controller using an ANFIS network to provide gain scheduling. This control scheme is applied to a Boost Converter circuit and simulated within the PSIM modelling environment. The simulation results indicate that using the ANFIS controller provides a fast system response with minimal errors even under dynamic operating conditions. The ANFIS controller is also shown to simplify the design flow in comparison to the popular Fuzzy-PI gain scheduling method.**

VII. Introduction

The first use of the classical PID controller is widely attributed to Elmer Sperry in 1911 in his work on automating ships steering. The first known work on the topic was published eleven years later by Minorksy [1]. Despite being a century old, the classical PID controller remains a popular control solution due to the algorithms flexibility and robustness. In the modern era, a number of modelling packages are available which offer quick and easy design and tuning of the PID controller helping to sustain its use.

Despite its popularity, the PID algorithm is not always the ideal control solution. As the algorithm provides a control signal which is essentially proportional to the error input, the PID is fundamentally linear. This means that sub optimal control is provided for non-linear systems, leading to the need for improvements in control systems to enable control of non-linear plants. This is particularly prevalent in the more complex plants which are often present in modern systems.

A number of different approaches have been taken in attempting to develop control systems which are more suitable for non-linear plants. There are now two distinct sets of controller designs for non-linear systems – adaptive control and gain scheduling. In an adaptive controller, the control law is adaptive to the altering dynamics of the plant. As the plant transitions into different operating conditions, the controller responds by updating its control parameters for that state. One of the most commonly used types of adaptive control is the sliding mode control. In this scheme, the controller provides a discontinuous control signal which forces the system to "slide" along a section of its normal operating range. As a popular method of non-linear control, sliding mode control has been shown to be a good solution for the control of power converters [2][3]. In addition to this, sliding mode control has been shown to be a robust method of control for non-linear systems. However, its implementation is a complex task which requires detailed knowledge of the underlying plant.

Gain scheduling is another popular method employed for non-linear control. In a gain scheduling controller, a number of linear controllers are employed, depending on the state of the system. Each of the separate linear controllers is able to give optimal control for a different operating state of the system. In a real system which employs a gain scheduling controller there will be just one linear controller, commonly a PID controller, which is adapted to the systems operating characteristics by altering the gain of the controller. This approach is an attractive option for power converters where the PID parameters can be altered according to the load, as well as for disturbances. An approach which has proven to be popular is the use of Fuzzy logic control (FLC) to provide the gain-scheduling for the PID, such as in [4] and [5]. This approach has also been successfully adopted in power converter systems, such as in [6] and [7]. These papers demonstrate the improved performance which can be achieved in non-linear systems when this type of controller is introduced.

The main attraction of the fuzzy logic controller in the gain scheduling scheme is its ability to make decisions. However, this is offset by the need for expert level of knowledge in the system to enable the design of a controller with suitable performance. In contrast, Neural Networks have a well defined training methodology based on their ability to learn but are limited by the inability to make decisions. In order to best exploit the capabilities of both types of controller, it is possible to utilize a neural network which is trained to perform as a fuzzy logic controller. One system which has been shown to combine the positive aspects of both types of AI controller is the Adaptive Neuro-Fuzzy Inference System (ANFIS) [8]. The ANFIS algorithm is an adaptive network which has a similar training scheme to the neural network whilst offering equivalent performance to a fuzzy logic inference system. Although the ANFIS algorithm is a computationally complex algorithm to implement, the advancement of fast and affordable processing, especially FPGAs, has seen the network employed in a number of applications such as in [9], [10] and [11]. The characteristics of the ANFIS network make it an ideal tool for use in a gain scheduling PID based control system for power converter systems.

In this paper a novel control scheme is presented for a boost type power converter which is considered to be a second order, state dependent plant. The boost converter will be fed by a solar photovoltaic (PV) cell and will give a stable output of 48 VDC. The new control solution presented here features the use of the ANFIS network to provide gain scheduling for a PI controller instead of the more commonly used FLC. The novel ANFIS-PI boost controller benefits from a simplified, well defined training methodology, which in turn makes the optimal performance of the controller easier to achieve. The ANFIS-PI controller is an example of demand side management [12], which is concerned with intelligent monitoring of the load and control which is subsequently influenced by this monitoring.

The rest of this paper is organized as follows: firstly the system being considered is presented, then the ANFIS network is introduced, next the system model is dealt with in greater detail before simulations are presented and discussed and finally conclusions end the paper.

VIII. System Description

The main goal of the system presented in this paper is to create a stable 48VDC power supply from a solar PV cell. The PV cell selected for this system is the BP3230, which is capable of producing 230W at a current of 8A. The output of the solar cell is applied firstly to a pre-regulator, which is used to apply a MPPT algorithm, and finally to a boost converter to provide the stable output. The main interest of this paper is the control of the boost converter, which features the novel ANFIS-PI control solution. A diagram of the full circuit is given in Fig 195, showing the PV cell, the boost converter and a pair of ANFIS-PI controllers. As shown in Fig 195, there are two PI controllers in the system - the inner PI controls the inductor current whilst the other controls the output voltage. The inner PI controller is shown by the bounded area labeled inner control loop and the outer loop is labeled outer control loop.

C. *Mathematical Model of the Boost Converter*

The performance of the boost converter circuit, shown in Fig 195, is influenced heavily by the current through the inductor. The converter behaviour is considerably simpler to model when the inductor isn't allowed to fully discharge during a switching period. It is therefore preferential for the converter to maintain this operating condition, which is known as continuous conduction mode. In this mode of operation the output of the boost converter can be obtained from the equation shown in Eq (1) where T_s denotes the switching period, T_{off} is the period that the switch is open during the switching period and D represents the switch duty cycle.

$$\frac{V_{out}}{V_{in}} = \frac{T_s}{T_{off}} = \frac{1}{1-D} \qquad D = 1 - \frac{V_{in}}{V_{out}} \tag{1}$$

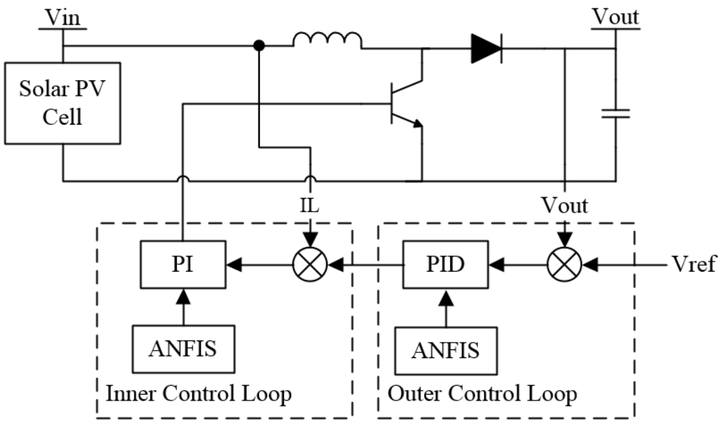


Fig 195 Diagram of the System

In order to maintain the boost converter in a continuous conduction mode of operation it is important that the inductor ripple current is considered. By controlling the amount of ripple current in the inductor and keeping it to a minimum the operating range of the converter can be extended. The equation modeling the inductor ripple current, ΔI_L, is shown in Eq. (2) where f_s is the switching frequency of the transistor.

$$\Delta I_L = \frac{V_{in\,\min} D}{f_s L} \tag{2}$$

Another important defining feature of the boost converter is the output capacitor as this has a big effect on the quality of the DC output. The output capacitor smoothes out the fluctuations which are introduced into the circuit through the transistor switching action, meaning it has a large influence on the output voltage ripple. The circuits overall output voltage ripple, ΔV_{out}, can be modeled by the equation shown in Eq. (3) where I_{out} is the output current.

$$\Delta V_{out} = \frac{I_{out\,\max} D}{f_s C} \tag{3}$$

D. *PI Controller*

The PID controller consists of three separate branches - proportional, integral and derivative. The three terms of the controller serve to respond to different elements of the error signal, with the proportional part acting immediately on the current error, the integral adding a contribution equivalent to the history of all errors and the derivative action predicting the future contribution of the errors. The addition of the integral and derivative actions to the basic proportional element enhances the performance of the controller by reducing steady state errors and the rise/fall times respectively. A popular variation of the PID algorithm is the PI algorithm, which features only the proportional and derivative terms. Elimination of the derivative term in the algorithm results in increased noise immunity, making it a popular and widely used variant of the classic PID. Eq. (4) shows the basic PI algorithm, with e being the output error and Kp and Tr being tunable gain coefficients.

$$K_p.e + \frac{1}{T_r}.\int e \qquad (4)$$

Whilst the simple PI algorithm offers robust control from a relatively simple algorithm, it is not ideally suited to the control of complex non-linear systems. In order to provide improved performance in such systems, the gain parameters K_p and T_r can be adapted according to the operating conditions of the plant. This popular approach allows for the PI to achieve a more optimal level of control for non-linear systems by deconstructing the system into a range of distinct linear operating points.

Whilst the ANFIS –PI is designed to counteract the non-linearity of the system, it is possible to further improve the boost circuit's response. If the current through the inductor is more closely controlled, then this has the effect of reducing the system response from a second order to a first order type. In doing this, the system becomes easier to control and a better response can be achieved. In the system shown in Fig 195, the inner PI controller is used to more closely control the current though the inductor, thus reducing the control complexity of the output and helping to further improve the system output.

E. *Maximum Power Point Tracking*

The operating characteristics of solar PV cells are influenced by both solar irradiation and temperature. In order to utilize the maximum available power from the PV cell, a Maximum Power Point Tracking (MPPT) algorithm is included in the system. There are a number of different algorithms available for MPPT with two of the most widely used being the Perturb and Observe (P&O) and Incremental Conductance (INC). In both cases, the algorithm works by slightly increasing or decreasing the operating point of the solar PV cell. By observing whether this causes an increase or decrease in the power, it is possible to track the MPPT. These techniques are popular as they are relatively simple to implement, exhibit good efficiency and are technology independent. A review of these algorithms applied to grid-tied PV systems is conducted in [13], showing that the INC algorithm yields the best results. A variation of the INC algorithm introduced in [14], which features adaptive step sizes, has been used in this application. Using this variation of the INC algorithm is shown to give reductions in oscillations in steady state operation, whilst also benefitting from improved dynamic response. The flow for the variable step size INC algorithm used is shown in Fig 196, where S(k) is the variable step size which is calculated as in Eq. (5) with N being a scaling coefficient which is tuned for the device.

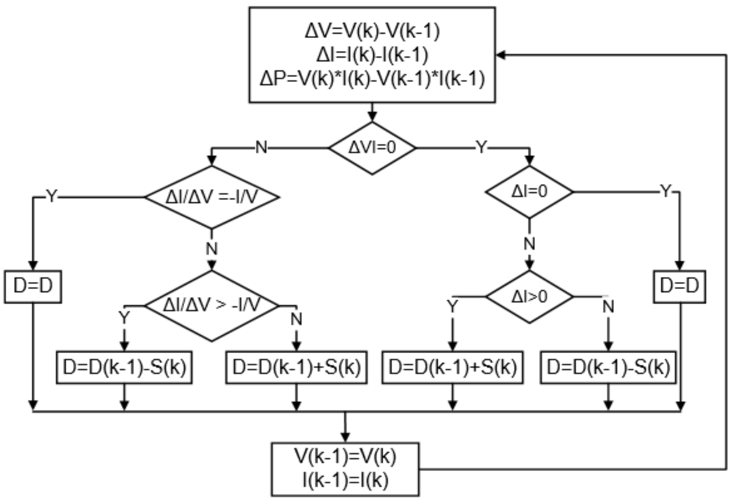


Fig 196 Incremental Conductance Algorithm with Variable Step Size

$$S(k) = N * \left| \frac{\delta P}{\delta V} \right| \qquad (5)$$

IX. ANFIS ALGORITHM

Whilst Fuzzy logic and Neural Networks are both well established AI techniques which have been widely applied to the field of control, both have drawbacks. In the case of fuzzy logic, whilst it is excellent at making decisions, the major drawback is the inability to learn. Conversely, neural networks don't share this ability to make decisions but do possess the ability to learn. Combining the benefits of both types of system, new hybrid Neuro-Fuzzy systems have emerged. One popular example of such a system is the ANFIS network. The ANFIS network was originally introduced in 1993 by J-S Jang [8] and has since been used in a variety of different systems.

The ANFIS consists of a five layer feed forward neural network, as shown in Fig 197. This algorithm is designed to ensure that the learning capabilities of the Neural Network are preserved with the addition of the decision making abilities of the fuzzy inference. When fully trained, this network exhibits behaviour which is analogous to a Sugeno type fuzzy inference system. The Sugeno inference engine is a universal approximator which is capable of approximating non-linear functions. As the ANFIS network is analogous to the Sugeno system, this non-linear function approximation capability is inherited. When coupled with the relative ease of training, thanks to its learning ability, this makes the ANFIS an attractive option for the control of non-linear plant such as switch mode DC-DC converters. In the rest of this section, the layers of the ANFIS network are discussed in more detail.

A. *Layer 1 - Fuzzification*

The first stage of the ANFIS converts the crisp input values into fuzzy number sets in much the same way that a fuzzy logic system would. The output of the membership functions is given in Eq (6) where μ_A and μ_B represents each of the membership groups and has a value between one and zero.

$$O_{1,i} = \mu_{Ai}(x_1)$$
$$O_{1,i} = \mu_{Bi}(x_2) \qquad (6)$$

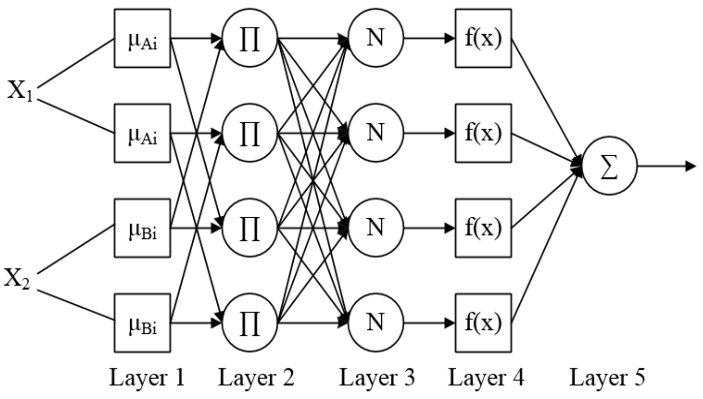


Fig 197 ANFIS Network Architecture

B. *Layer 2 and 3 – Firing Strength*

Once the fuzzification of the inputs has been performed, the next two layers calculate the rules firing strength. This is done in two stages - firstly the values of μ_A and μ_B are multiplied together as shown in Eq 0. This is performed in layer two; layer three then performs a normalization of the multiplied values as shown in Eq. Eq. 28 .

$$O_{2,i} = w_i = \mu_{Ai}(x_1)\mu_{Bi}(x_2) \qquad (7)$$

$$O_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2} \qquad (8)$$

C. *Layer 4 – Consequence Parameters*

The next layer in the ANFIS calculates the consequence parameters. In a traditional fuzzy system, consisting of IF THEN rules, this part of the algorithm is equivalent to the THEN portion of rule. The output of this layer is shown in Eq. 0. The output of this stage is effectively the same as the output rule for Takagi-Sugeno type fuzzy inference engine.

$$O_{4,i} = \overline{w}_i f_i = \overline{w}_i(px_1 + qx_2 + r) \qquad (9)$$

D. *Layer 5 – De-fuzzification*

The final layer of the ANIFS algorithm converts the fuzzy logic sets into a crisp output which can be used by the external PI controller. This stage takes the simple form of a summation of all the rule outputs, meaning the output of this stage, and the whole ANFIS algorithm, is as given in Eq. 0.

$$O_5 = \frac{w_i f_i}{w_i} \qquad (10)$$

E. *Training the ANFIS*

The ANFIS network features two layers which must be trained - the fuzzy logic sets in layer one and the consequent parameters in the fourth layer. The training process is much the same as in a neural network, whereby a training set is provided which consists of a number of inputs and the expected outputs. The goal of the training is then to minimize the error between the networks actual outputs and the expected outputs. This process is typically supervised by a hybrid training algorithm, which features a forward pass and a backward pass phase. During the forward pass phase, the node outputs are fed forward until layer four. At this stage, the consequent parameters in layer four are then tuned using the least squares estimation algorithm. This method is designed to minimize the sum of the squared error of the system output. In the back pass phase of the hybrid algorithm, the membership sets in layer one are tuned. In this part of the training the error signals are propagated backwards from the output and the membership parameters are optimized using the gradient descent algorithm. This algorithm finds the minimum error by moving the membership parameters a distance which is proportional to the functions gradient at the given point. This movement is performed in each of the training iterations until the output error is minimized as much as possible.

The ANFIS utilized in this system is designed to have two inputs – with one being the load current and one being changes in load current. There will be seven trapezoidal membership functions for the inner control loop and five in the outer control loop as this was shown to give a good approximation of the training data. The training process is automated in Matlab using the Neuro-Fuzzy plug-in, which uses the hybrid learning algorithm previously discussed to tune the performance of the ANFIS controller. Once the training stage has been completed, it is also easily possible to validate the data and import it into the Simulink model using this tool.

X. SYSTEM MODELING

F. *Boost Converter Circuit Model*

In order to optimize the performance of the boost circuit, the values of the output capacitor and the inductor must be carefully selected as previously discussed. The inductor used in this circuit is chosen as 22μH, which gives a ripple current of 550mA according to Eq. (2). The output capacitor is selected as 100μF, which gives a maximum ripple voltage of 25 mV according to Eq.(3). The boost converter circuit will be modeled using the PSim simulation environment. PSim is a Spice based simulation tool which is targeted at the design of power electronics and control systems.

G. *Solar PV Cell and MPPT Tracking*

The modelling of solar PV cells can prove to be a complex task. However the use of the PSim software suite allows for this task to be simplified greatly. PSim supplies a generic solar cell modelling tool which can be used to create a custom model of any given cell using a few parameters from the data sheet, such as rated power, series resistance, shunt resistance and short circuit current. This tool also allows for the generation of power curves at any operating point to allow for the model to be validated before it is imported into the PSim model for use in simulations. The solar PV cell model developed for this system and the obtained IV curves at maximum power are shown in Fig 198.

The implementation of the variable step size INC algorithm

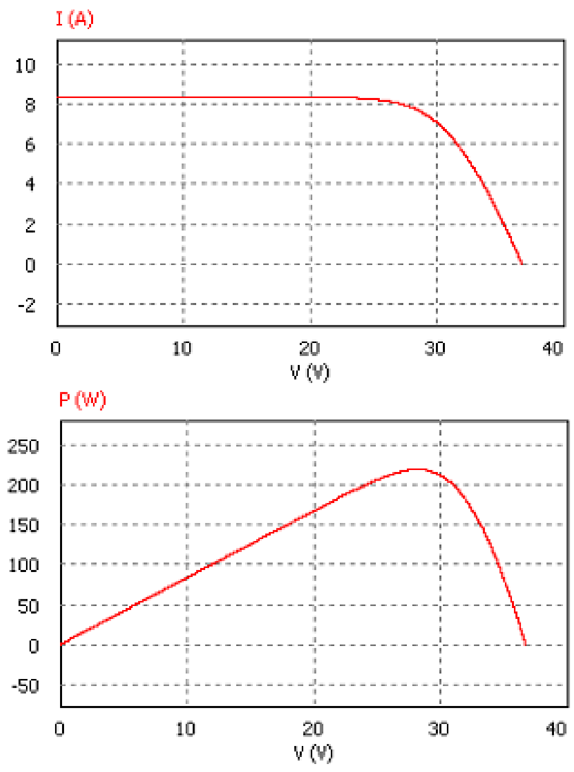


Fig 198 BP3230N IV Curve obtained in PSim

is also achieved using the PSim software suite. Whilst this task could have been achieved using Matlab, the use of PSim at this stage allows for closer consideration of a potential final hardware target. The PSim tool offers compilation and simulation of functions written using C code. This means that the code can be developed and simulated at the system modelling stage and easily imported to a microcontroller hardware target, reducing development time.

H. Tuning the PI Controllers

Prior to the ANFIS being trained, it is first necessary to create the training data. In order to gather this data, the PI controllers need to be tuned under a number of operating scenarios. There are various different methods available for the tuning of PI controllers, from traditional methods like Ziegler-Nichols, to modern approaches using artificial intelligence (AI) algorithms and online training solutions. The training algorithm selected here is Particle Swarm Optimization (PSO) which is an AI based approach designed to mimic the behaviour of flocking birds. The main flow of this algorithm is shown in Fig 199. The PSO algorithm has proven to be a popular solution to the tuning of PI controllers [9][16] and benefits from being effective and relatively simple to implement.

A crucial component of the PSO algorithm is the particle, which consists of three separate parts – the current solution, personal best fit and a velocity. The current solution stores the data which is to be applied to the problem, whilst the personal best fit stores the value which has yielded the best result for the particle. The velocity is a key parameter in controlling how the problem space is searched, dictating how fast and in which direction the particle is moved. This parameter is initially set to a random value but is iteratively updated. Each update has the effect of pulling the particle towards either a global or personal best fit. The equation for updating the velocity of the particle is shown in Eq. (11) with V_n being current velocity, X_n being the current solution, GX_n the global best fit, PX_n the personal best fit and ω is a constant known as inertia weight.

$$V_{N+1} = \omega V_N + (C_1 Rand(GX_N - X_N)) \\ + (C_2 Rand(PX_N - X_N)) \qquad (11)$$

The social and cognitive coefficients, shown by C_1 and C_2 respectively, have the effect of pulling the particle towards either its personal best fit (cognitive) or the global best fit (social). The values of these two therefore have the effect of promoting convergence to either the personal or global best fit. To achieve optimal performance of the algorithm, the cognitive value is linearly decreased, whilst the social value is linearly increased.

Using this algorithm, the PI controllers are tuned, firstly the inner current PI controller and then the voltage PI. The two PI controllers are both trained under a number of different steady state loads and with a number of load transients to give a more complete set of training data. The optimized gain values found during tuning are then used to form the training data of the gain scheduling ANFIS network. The PSO algorithm is modeled in the popular Matlab simulation tool using co-simulation with the boost converter developed in PSim through the Sim-Coupler plug-in.

This simplified design methodology is one of the main advantages of its application. A similar control strategy can be employed through the use of the Fuzzy logic controller. The use of Fuzzy logic control for the control of non-linear plants is relatively well established as shown in [17] and [18]. In the instances presented in these papers, the authors rely on a combination of both user knowledge and trial and error to achieve optimal tuning. This is in contrast to the methodology presented here, where both the training and the gathering of

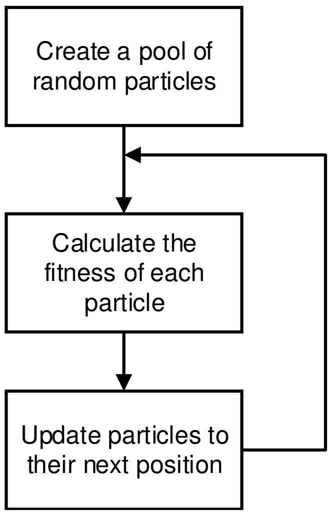

Create a pool of random particles
Calculate the fitness of each particle
Update particles to their next position

Fig 199 PSO Algorithm

the data are automated processes. This means that the need for detailed knowledge of the underlying plant is not necessary and there isn't a reliance on a trial and error approach for the controller training. Whilst other training methods exist for Fuzzy-PI based control which doesn't rely on trial and error, such as [19], these still tend to be complicated in comparison to the method in this paper.

XI. SIMULATION RESULTS

In order to demonstrate the performance of the ANFIS-PI controlled boost converter PV system, a number of simulations have been carried out on the model. Simulations are also carried out on the same system using a FLC-PI based controller, allowing for performance characteristics to be compared between the two methodologies. A total of five of simulations have been carried out to ascertain the systems performance under both steady state and dynamic conditions. The results of all the simulations carried out are discussed in further detail in this section and are also summarized in TABLE III below. These simulation results indicate the novel ANFIS-PI approach exhibits a faster system response and a generally smaller absolute error under dynamic conditions. The steady state of operation for both of the controller types exhibit similar behaviour.

TABLE III

SUMMARY OF SIMULATION RESULTS

Steady State Tests		
4A Load		
	ANFIS-PI	**FUZZY-PI**
Steady State Error (%)	0	0
Peak to Peak Ripple (mV)	4	4
1A Load		
Steady State Error (%)	0	0
Peak to Peak Ripple (mV)	16	16
Dynamic Test		
1A Step Decrease in Load		
Max. Error (mV)	137	220
Max. Error (%)	0.28	0.45
Settling Time (us)	37	126
1 A Step Increase in Load		
Max. Error (mV)	147	187
Max. Error (%)	0.3	0.39
Settling Time (us)	55	130
Input Disturbance		
Max. Error (mV)	69	66
Max. Error (%)	0.14	0.13
Settling Time (us)	30	78

The first set of simulations carried out is intended to measure the steady state performance of the system under both a heavy load (4A) and a light load (1A). The results of these tests are illustrated in Fig 200 and Fig 201 respectively. In both of the systems, the output ripple voltage for the light load is measured as just 4mV and at 16mV for the heavy load test. In addition to this, both of these systems also exhibit no steady state error.

The next simulation is carried out to show the dynamic performance of the system when there is a step decrease in output load demand. The system performance with a one amp decrease in current is shown in Fig 202. In this simulation the ANFIS based system shows improved performance compared to the FLC system, with the maximum absolute error being just 137mV, or 0.28%, compared to 220mV (0.45%) in the FLC system. Additionally the settling time to within 0.1% of the reference output is also reduced in the ANFIS system from 126us to 37us.

A similar dynamic simulation is carried out next, this time with an increase in output load demand. Fig 203 shows the output of the two systems with a one amp step increase in output current. In this instance the ANFIS controlled system again exhibits improved behaviour with an absolute maximum error of 147mV (0.3%) compared to 187mV (0.39%) in the FLC based system. Similarly the settling time is also greatly reduced, with the ANFIS settling to within 0.1% of the reference in just 55us compared to 130us.

The final simulation is carried out with a disturbance of 10% in the input voltage, with the results being shown in Fig 204. In this instance the absolute maximum error of the FLC based system is marginally less than the ANFIS system at 66mV (0.13%) compared to 69mV (0.14%). However the ANFIS based system still exhibits the faster response with a settling time of 30us compared to 78us in the FLC system.

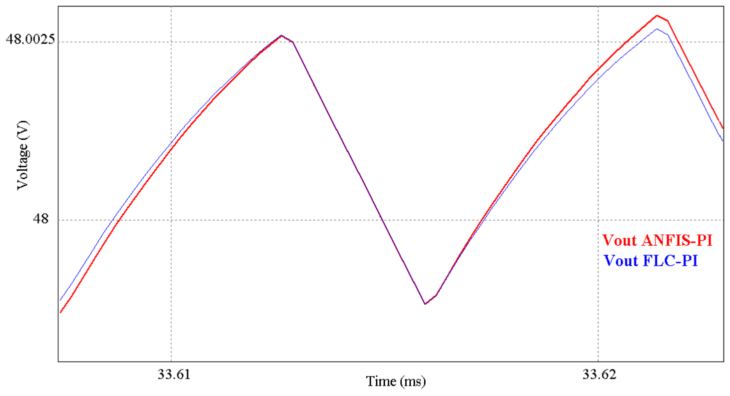


Fig 200 Ripple Voltage with a One Amp Load

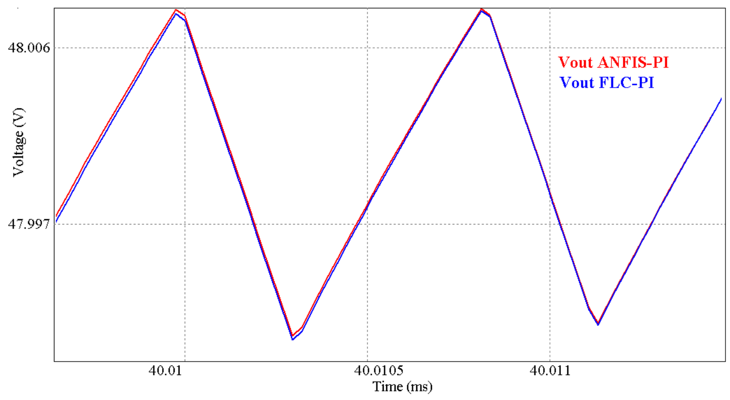


Fig 201 Ripple Voltage with a Four Amp Load

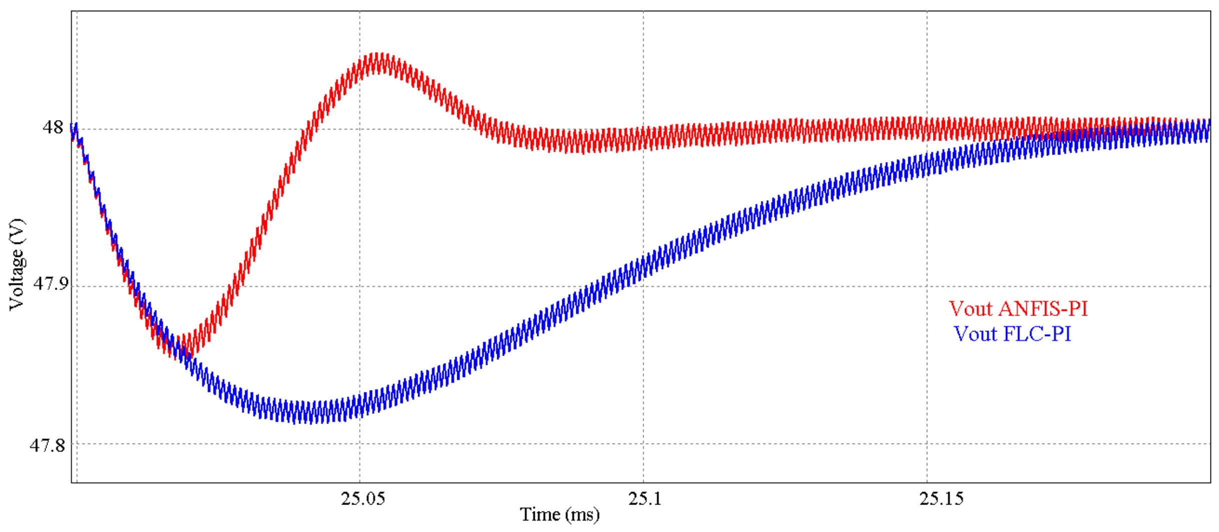


Fig 202 Boost converter output with a 1A step decrease in current

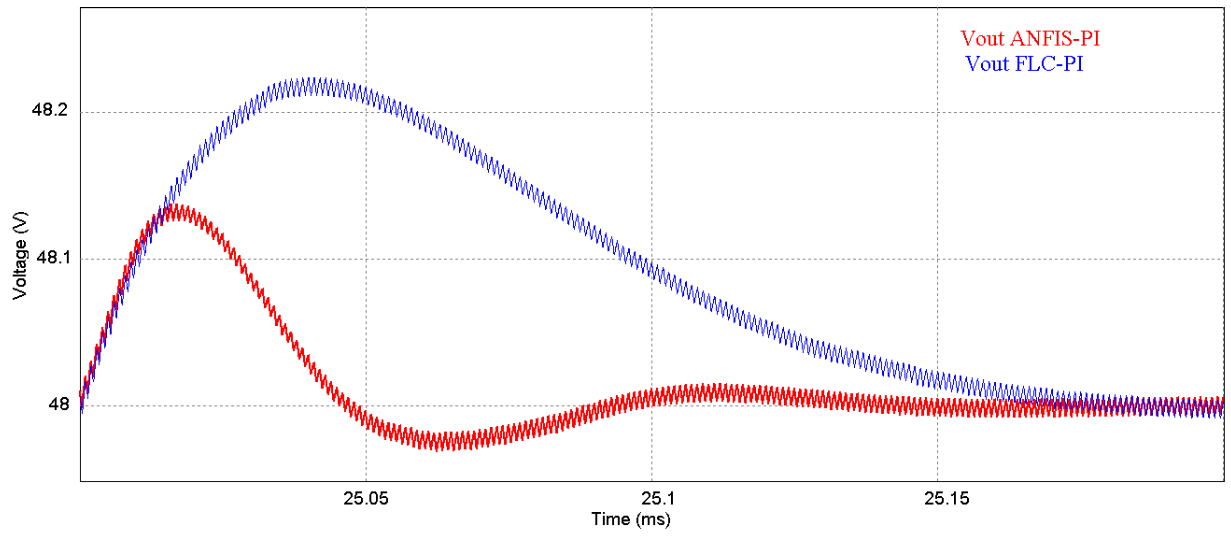


Fig 203 Boost converter output with a 1A step increase in current

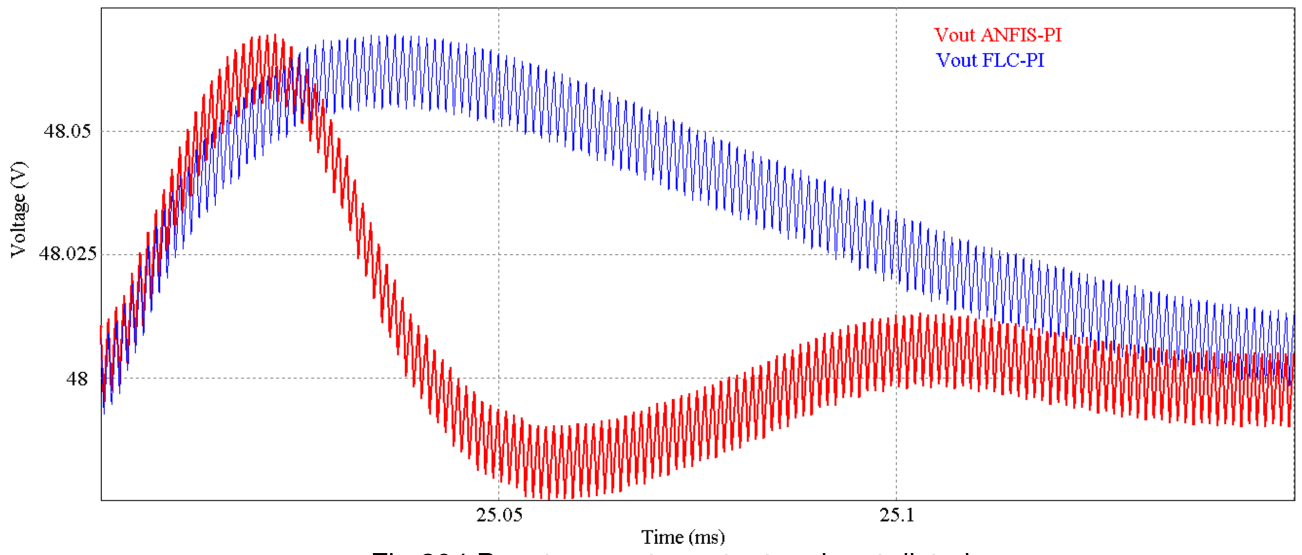


Fig 204 Boost converter output an input disturbance

XII. CONCLUSIONS

The simulation results of the novel gain scheduling ANFIS network in the developed solar PV application indicate that this methodology can achieve accurate control even for a non-linear system. The boost converter to which the ANFIS-PI controller is applied exhibits excellent behaviour under both steady state and dynamic operating conditions. In the steady state operation, the power converter exhibits no steady state error, whilst under heavy loads the peak to peak ripple voltage is just 16mV. Under dynamic operating conditions the maximum absolute error observed is just 0.3 %. This performance characteristic can be especially important in the PV cell fed boost converter application, to which this is applied. This level of voltage deviation allows for the system to be fed into an inverter without any instability issues being introduced, meaning it can be tied to the grid.

As well as being capable of delivering reductions in absolute error and a faster response time on a non-linear plant, the presented methodology is also seen to be simpler to implement in comparison with a similar fuzzy logic based system. The design flow for the controller is simplified further by implementing the PSO algorithm in the training phase. The PSO algorithm is both efficient and simple to implement, meaning that the training data can be gathered easily. Leveraging this algorithm in this stage also means that the tuning of the PI becomes an automated process. The availability of the Matlab Neuro-Fuzzy tool makes the design flow more desirable still, as this means that ANFIS training can be achieved quickly and without the need for further algorithm development. This shows that this methodology can offer a fast modelling and prototyping solution.

REFERENCES

[1] N. Minorsky, "Directional stability of auto-matically steered bodies." *J. Amer. SOC .of Naval Engineers,* Vol. 34, pp. 280-309, May 1922

[2] L. Martinez-Salamero, G. Garcia, M.Orellana, C.Lahore and B.Estibals, "Start-Up Control and Voltage Regulation in a Boost Converter Under Sliding-Mode Operation." *IEEE Transactions on Industrial Electronics*, Vol. 60, Iss. 10, pp. 4637-4649, 2012

[3] S. Oucheriah and L. Guo , "PWM-Based Adaptive Sliding-Mode Control for Boost DC-DC Converters." *IEEE Transactions on Industrial Electronics*, Vol. 60, Iss. 8, pp. 3291-3294, 2012

[4] A. Rubaai, M.J. Castro-Sitiriche and A.R Ofoli, "DSP-Based Laboratory Implementation of Hybrid Fuzzy-PID Controller Using Genetic Optimization for High-Performance Motor Drives." *IEEE Transactions on Industry Applications*, Vol. 60, Iss. 6, pp. 1977 – 1986, 2008

[5] H. Yu, T. Chen and C. Liu, "Adaptive Fuzzy Logic Proportional-Integral-Derivative Control for a Miniature Autofocus Voice Coil Motor Actuator with Retaining Force" *IEEE Transactions on Magnetics*, Vol. 50, Iss. 11, 2014

[6] M. Elshaer, A.Mohamed and O.A. Mohammed, "Smart Optimal Control of DC-DC Boost Converter for Intelligent PV Systems" *16th Interntaional Conference on Intelligent Sytem Application to Power Systems*, pp. 1-6, 2011

[7] V.Vindhya and V. Reddy, "PID-Fuzzy Logic Hybrid Controller for a Digitally Controlled DC-DC Converter." *International Conference on Green Computing, Communication and Conservation of Energy*, pp. 362-366, 2013

[8] J-S, Jang , "ANFIS: Adaptive-Networtk-Based Fuzzy Inference System." *IEEE Transacrtions on Systems, Man and Cybernetics*, Vol. 23, Iss 3, pp. 665-685, 1993

[9] P. Garcia. C.A. Garcia, L.M. Fernandez, F. Lorens and F. Jurado, "ANFIS Based Control of Grid Connected Hybrid System Integrating Renewable Energies, Hydrogen and Batteries." *IEEE Transactions on Industrial Informatics*, Vol. 10, Iss. 2, pp 1107 – 1117, 2013

[10] M. Singh and A.Chandra, "Real Time Implementation of ANFIS Control for Renewable Interfacing Inverter in 3P4W Distribution Network." *IEEE Transactions on Industrial Electronics*, Vol. 60, Iss. 1, pp. 121-128, 2012

[11] H. Abu-Rub, A. Iqbal, S. Mon Ahmed, F.Z. Peng and G. Baoming, "Quasi-Z-Source Inverter Based Photovoltaic Generation System with Maximum Power Tracking Control using ANFIS." *IEEE Transactions on Sustainable Energy*, Vol 4, Iss. 1, pp 11-20, 2012

[12] P. Palensky and D. Dietrich, "Demand Side Management: Demand Response, Intelligent Energy Systems and Smart Loads." *IEEE Transactions on Industrial Informatics*, Vol. 7, Iss. 3, pp 381 – 388, 2011

[13] R.A. Mastromauro, M. liserre and A. Dell'Aquila, "Control Issues in Single-Stage Photovoltaic Systems: MPPT, Current and Voltage Control", *IEEE Transactions on Industrial Informatics*, Vol. 8, Iss. 2, pp. 241 – 254, 2012

[14] F. Liu, S. Duan, F. Liu, B. Liu and Y. Kang, "A Variable Step-Size INC MPPT Method for PV Systems", *IEEE Transactions on Industrial Electronics*, Vol. 55, Iss. 7, pp 2622 – 2628, 2008

[15] X. Li, M.Chen and Y.Tsutomu, "A method of searching PID controller's optimized coefficients for Buck converter using particle swarm optimization." *IEEE Electric Power and Energy Conference*, pp.238, 2013

[16] J. Femmy Nirmal and D.Jeraldin Auxillia, "Adaptive PSO based tuning of PID controller for an Automatic Voltage Regulator system." *International Conference on Circuits, Power and Computing Technologies*, pp.661, March 2013

[17] L. Guo, J.Y. Hung and R.M. Nelms, "Evaluation of DSP-Based PID and Fuzzy Controllers for DC-DC Converters". *IEEE Transactions on Industrial Electronics*, vol. 56, Iss 6, pp. 2237, June 2008.

[18] R.F. Bastos, C.R. Aguiar, A.F.Q. Gonclaves and R.Q. Machaldo. "An Intelligent Control System Used to Improve Energy Production from Alternative Sources with DC/DC Integration." *IEEE Transactions on Smart Grid*, Vol 5, Iss 5, pp. 2486, June 2014.

[19] X. Duan, H. Deng and H. Li, "A Saturation Based Tuning Method for Fuzzy PID Controller". *IEEE Transactions on Industrial Electronics*, vol. 60, Iss. 11, pp. 5177, June 2012.

A Novel ANFIS Algorithm Architecture for FPGA Implementation

John Darvill, Alin Tisan, Marcian Cirstea
Anglia Ruskin University, Cambridge, UK
john.darvill@student.anglia.ac.uk, alin.tisan@anglia.ac.uk, marcian.cirstea@anglia.ac.uk

Abstract— This paper presents a new architecture for the Adaptive Neuro-Fuzzy Inference System (ANFIS) algorithm targeting FPGA implementation. This new architecture offers higher efficiency and scalability in comparison to the existing methods. The proposed architecture is modeled and simulated using VHDL and is targeted at a Xilinx FPGA. Existing implementation architectures are also modeled and comparisons are drawn between them in terms of both performance and logic utilization. The results show that the new architecture offers a reduction in calculation cycles of around 50% in comparison to the architecture from which it's derived. This increase in calculation speed comes with only a modest increase in logic utilization, specifically a 2.5% increase in look-up table (LUT) usage and a 1.5% increase in flip-flop usage. The new architecture also eliminates scalability issues which can arise in the existing architectures when extra input members are required.

I. INTRODUCTION

The field of Artificial Intelligence (AI) has become a popular research topic in recent years due to the potential for application in a range of real world problems, such as voice recognition, data mining, image processing and control systems. Two of the most popular AI based algorithms are Artificial Neural Networks (ANN) and fuzzy logic (FL). The ANN is designed to mimic the neural networks of the brain and as such has an inherent capability to learn, making it a powerful tool for function approximation. On the other hand, FL based algorithms are good at decision making, which means they are popular tool for intelligent control solutions. However the FL algorithm lacks a defined training method, meaning it can be difficult to ensure optimal performance, whilst ANN based systems don't benefit from the ability to make decisions. In order to best exploit the capabilities of both types of algorithm, it is possible to utilize a neural network which is trained to perform as a fuzzy logic controller. Using this approach, it is possible to leverage both the learning capabilities of the ANN and the decision making ability of the FL algorithm. One system which has been shown to achieve the best of best types of AI controller is the Adaptive Neuro-Fuzzy Inference System (ANFIS) [1]. The ANFIS algorithm is an adaptive network which has a similar training scheme to the neural network whilst offering equivalent performance to a fuzzy logic inference system. Although the ANFIS algorithm is a computationally complex algorithm to implement, the advancement of fast and affordable processing has seen the network employed in a wide range of applications such as in [2], [3] and [4].

The ANFIS algorithm has been successfully implemented using both DSPs ([5], [6] and [7]) and digital hardware techniques ([8], [9] and [10]). However, the nature of the algorithm means it is ideally suited to a parallel processing implementation. Whilst modern DSPs typically feature multi core devices, offering a degree of parallel processing, true parallel processing can only be achieved using a hardware approach. For this reason, FPGAs are a popular choice for the implementation of such neural based algorithms ([11]) as they allow for improved throughput in comparison to DSPs.

In this paper, a new digital architecture for the implementation of the ANFIS algorithm is proposed. This new architecture improves scalability, allowing for more membership groups to be utilized without adversely affecting the latency. Additionally, the architecture is designed to utilize as few logical resources as possible. The digital architecture presented in this paper is targeted at a Xilinx Zynq device and is primarily implemented using VHDL, although it is portable to other devices and languages. To show the effectiveness of this new approach, it is compared with some existing digital ANFIS architectures.

The rest of this paper is organized as follows: firstly the ANFIS algorithm is introduced, then a review is undertaken of the existing architecture options in digital hardware, after this the novel architecture is discussed, next the implementation is presented before simulations and hardware tests are presented. Finally conclusions are drawn at the end of the paper.

II. OVERVIEW OF THE ANFIS ALGORITHM

The ANFIS algorithm was originally introduced in 1993 by J-S Jang [1]. This algorithm is comprised of a five layer feed forward neural network, as shown in Fig. 1. When fully trained, this network exhibits behaviour which is analogous to a Sugeno type fuzzy inference system. The Sugeno inference engine is a universal approximator which is capable of approximating even the most complex of non-linear functions. This non-linear function approximation capability is therefore inherited by the analogous ANFIS algorithm. When coupled

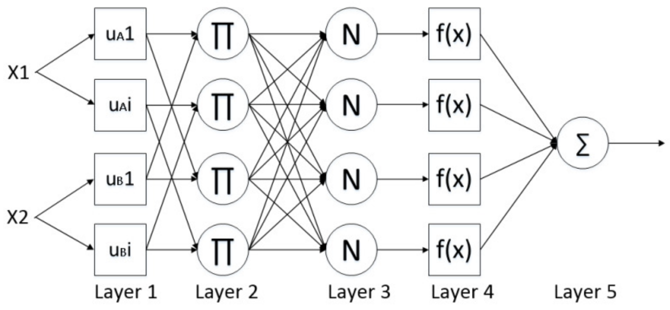


Fig. 1. ANFIS Architecture

with the relative ease of training, thanks to its learning capability, this makes the ANFIS an attractive option for a number of applications. The ANFIS algorithm has proven particularly popular in the field of intelligent control, such as in [12] and [13]. In these applications the novel architecture discussed in this paper could be utilized. In the rest of this section the layers of the ANFIS network are discussed in detail.

A. *Layer 1 - Fuzzification*

The first stage of the ANFIS converts the crisp input values into fuzzy number sets in much the same way that a fuzzy logic system would. The output of this layer denotes the degree to which the inputs satisfy the membership groups (given as μ_A and μ_B) and has a value of between 0 and 1.

B. *Layer 2 and 3 – Firing Strength*

The next stage in the system calculates the firing strength of the rules. This is done in two stages - firstly the values of μ_A and μ_B are multiplied together in the second layer. The third layer then performs normalization of the multiplied values, crunching them into a predefined range.

C. *Layer 4 – Consequence Parameters*

The next stage in the ANFIS calculates the consequence parameters. In a traditional fuzzy system, consisting of IF THEN rules, this part of the algorithm is equivalent to the THEN part of rule. The output of this layer is shown in Eq. (1) with $\overline{W_i}$ denoting he output from the third layer. The output of this stage is effectively the same as the output rule for Takagi-Sugeno type fuzzy inference engine.

$$O_{4,i} = \overline{w}_i f_i$$
$$\text{where } f_i = z = (px_1 + qx_2 + r) \tag{1}$$

D. *Layer 5 – De-fuzzification*

The final stage of the ANIFS algorithm is to convert the fuzzy logic sets into a crisp output. This stage takes the simple form of a summation of all the rule outputs, meaning the output is as given in Eq. (2).

$$O_5 = \frac{\sum w_i f_i}{\sum w_i} \tag{2}$$

E. *Training the ANFIS*

Before the ANFIS can be used in any given application it is firstly necessary to train the underlying neural network. There are two adaptive nodes which must be trained – the input members in layer one and the consequent parameters in the fourth layer. The training method is similar to the normal neural network approach, with the ANFIS being presented with a set of training data. This training set consists of a number of inputs and the expected outputs, with an algorithm being utilized to minimize the error between the expected and actual outputs. The training algorithm is typically a hybrid learning algorithm, featuring forward and backward pass phases. During the forward pass phase of this training the node outputs are fed forward until layer four. At this stage the consequent parameters are tuned using the least squares estimation method. The least squares method is designed to minimize the sum of the squared error of the system output. In the backward pass phase, the membership sets in layer one are tuned. In this phase the error signals are propagated backwards from the output and the membership parameters are optimized using the gradient descent algorithm. This gradient descent algorithm finds the minimum error by moving the membership parameters a distance which is proportional to the functions gradient at a given point. This algorithm is performed in each of the training iterations until the output error is sufficiently minimized.

III. Current State of Art

There are two different types of ANFIS architectures which are presented in research papers [8], [9] and [10] – one using a parallel approach and the other a serial approach. In the parallel architecture, each of the output members has a dedicated logic circuit allowing for all the values to be calculated in a single clock cycle. In comparison, the serial approach has one logic circuit which is used to calculate each of the output members, with the outputs being calculated cyclically. Whilst the parallel architecture offers advantages in the speed of calculation, the logic utilization is considerably greater. The limitations of both of these architectures become exacerbated as more input members are added and the implementation becomes more complex. In the rest of this section both of these architectures are discussed.

A. *Serial Architecture*

The serial architecture splits the algorithm into four subsystems – a fuzzifier, a permutator, the inference engine and finally a de-fuzzifier as is shown in Fig. 2.

Fuzzifier

This block of the architecture relates to layer one in the ANFIS block diagram as shown in Fig. 1 and is responsible for calculating the fuzzy input member values. This block is typically a ROM based look up table (LUT).

Permutator

The permutator outputs every possible permutation of input members. This block takes the form of circular shift registers which are shifted in each clock cycle until all permutations have been output. The number of clock cycles required to permutate all data is therefore equal to the total number of permutations. This block represents the main speed bottleneck for this implementation.

Inference Engine

The inference engine calculates the values of the numerator and denominator as shown in Eq. (2). This block has three functions which are required in order to calculate these values. The first function is a multiplier which is used to determine the value of the denominator. The second function calculates the value of the consequence parameters, as given by the polynomial equation in Eq. (1). The values of p,q and r are loaded from a ROM, whilst basic adders and multipliers are used to form the rest of the equation. The final function calculates the value of the numerator by multiplying the results of the first two functions.

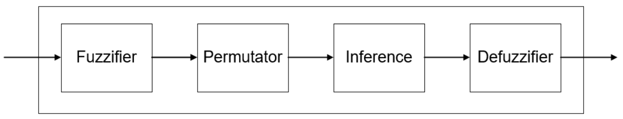


Fig. 2. erial ANFIS FPGA Implementation

Fig. 3. Parallel ANFIS Implementation

De-Fuzzifier

This final block accumulates the numerator and denominator values output by the inference engine over a full calculation cycle. A divider is then utilized to perform the final operation required to generate the output of the ANFIS as given in Eq. (2).

B. Parallel Architecture

The parallel architecture is utilized in [8] and is shown in Fig. 3. This approach is discussed further in this section.

1. Fuzzifier

This block uses a ROM based LUT to perform the fuzzification of the system inputs as in the serial approach.

Rule Weight

This block is formed of a single multiplier which performs the calculation shown in Eq. (1). There is one multiplier for each possible permutation of consequence parameters, meaning the amount required is equal to the number of permutations.

Consequence Parameters

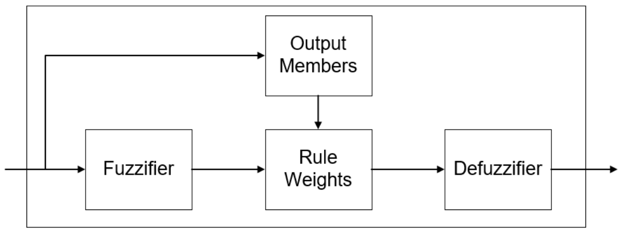
This subsystem calculates the values for each of the consequence parameters as required for layer 4 in Fig. 1. The method for this is the same as that utilized in the inference engine for the serial approach. However whilst there is one of these circuits in the serial method, in the parallel method there is one used for the calculation of each of the output functions. The total number of circuits required is therefore equal to the total number of possible permutations of the input members. This means that the logic required for this function increases in proportion to the number of input members used which hampers the scalability of this architecture.

De-Fuzzifier

The final subsystem in the parallel architecture is responsible for calculating the output and performs three tasks. The first task is to calculate the value of the numerator in Eq. (2) by multiplying each of the rule weights with the related consequence parameter value. The values of all of these multiplications are then added together giving the numerator value. The second task of the de-fuzzifier is to calculate the value of the denominator which is performed through the addition of all the rule weights. The final task of this block is to calculate the algorithm output by dividing these values.

IV. PROPOSED FPGA IMPLEMENTATION

In the previous section, the two most prevalent ANFIS architectures which are deployed in FPGAs were discussed. Whilst both of these implementations have been designed and implemented for cases with three members, both of these approaches suffer from scalability issues. In the case of the parallel architecture the amount of logic increases in proportion to the number of input members required, meaning excessive logic utilization can become a problem. For the serial approach, the limitation is evident in the number of clock cycles required in the permutator module, which is equal to the number of members squared. In this paper, a novel approach is proposed, which improves the efficiency of the permutator sub system within the serial architecture. This improvement removes redundant permutations, thus reducing the number of clock cycles required by the permutator. This modification improves both the speed and the scalability of the architecture. In the rest of this section, the features of the new architecture are discussed in more detail.



As was mentioned in the previous discussion of the implementation techniques, the permutator sub system represents something of a speed bottleneck in the serial implementation approach. This bottle neck exists as there is a need to iterate over each permutation of the input members. This means that the total number of iterations required by the permutator is equal to the number of input members squared. This is unnecessary as it is unlikely that any input will ever occupy more than two or three of the possible input members. By way of an example, the application which is simulated in the next section has five input members for each of the two inputs. This means that in the existing serial implementation there would need to be a total of twenty five iterations for the calculation of the permutator. However, in reality one of the inputs can only ever exist in two input members at once, whilst the other can only exist in three members at any one time. This means that the maximum number of iterations required would actually be six. This shows that there are nineteen superfluous operations in this instance when using the current serial method.

Given the inefficiency in the existing implementation, the novel approach seeks to improve the efficiency of the permutator by removing the need to iterate over redundant combinations. In order to facilitate this objective, the shift registers in the original permutator design are replaced with multiplexors. There is then additional logic which is used to drive the address bits of the multiplexor so that each valid combination of outputs is achieved. The full circuit diagram for the new permutator circuit is given in Fig. 4.

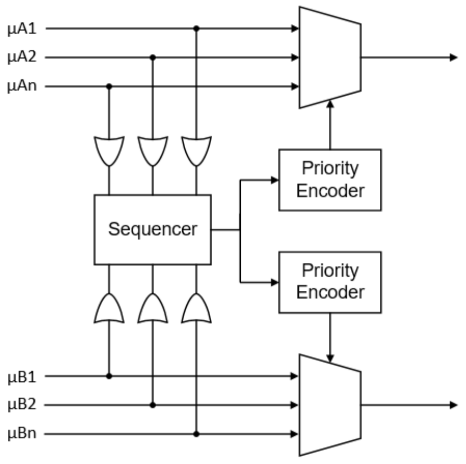


Fig. 4. Novel Permutator Solution

The first stage of the novel permutator is to identify which of the input members have a valid, non-zero value. In order for this to be achieved, a member map is created which shows which of the members has a non-zero value. The member map has one bit per input which is set to 1b when the value is non-zero. It is simple to create the logic to drive this map by taking an OR of all the bits in each of the membership values.

Sequencing logic drives the multiplexor address bits, iterating over each of the valid member combinations. The sequencer clears one of the set bits in the member map in each clock cycle. Once all bits are cleared in the first member map, it is restored to the original value and one of the bits is cleared in the second member map. This process is repeated until each of the valid combinations has been iterated over. An example of the sequencer operation for a five input ANFIS is given in TABLE I. The sequencer drives a priority encoder which converts the output into a valid multiplexor address. The output of the priority encoder is a binary representation of the least significant bit which is set in the input. TABLE II. shows the priority encoder process for a 5 bit input.

V. FPGA IMPLEMENTATION

In order to verify the performance of the new architecture for the ANFIS algorithm, a VHDL based model has been developed. This model is derived from the ANFIS-PI based power converter controller solution which was previously presented in [14]. However, this methodology would be equally suited to other ANIFS based industrial electronics applications, such as those presented in [6] and [12]. In order to compare the new method with the existing architectures, models are also developed which utilize those methodologies. All three models have two inputs, both of which utilize five Gaussian input members. This is in comparison to the implementations presented in [8], [9] and [10] which use only three input members. The system has a single 8-bit input, with the delta of this input with respect to time being the second ANFIS input. The models convert the 8 bit inputs to single-precision floating point number and all data is subsequently processed using this precision.

The target device for the implementation will be the Xilinx Zynq, part number XC7Z020, which features an Artix-7 based FPGA core and a dual core ARM processor. The Artix-7 core is a low-end member of the Xilinx family of FPGAs and is chosen to illustrate how the novel architecture can be implemented using simple logic blocks. However, the novel architecture could just as easily be incorporated into more high-performance FPGAs such as the Virtex-7 series. The architecture of the Artix-7 family uses customizable logic blocks (CLB) organized in slices (2 per CLB), each containing 4 lookup tables (LUT) and 8 Flip Flops (FF) that can be used to implement combinatorial and sequential logic circuits. The LUTs can also implement ROM or RAM memories, 6 input functions or shift registers. There are also a number of DSP cells which are optimized for fast mathematic operations.

TABLE I. SEQUENCER EXAMPLE

Clock Cycles	x1 Sequence	x2 Sequence
1	01110	01100
2	01100	01100
3	01000	01100
4	01110	01000
5	01100	01000
6	01000	01000

TABLE II. PRIORITY ENCODER LOGIC

In4	In3	In2	In1	In0	Out2	Out1	Out0
0	0	0	0	0	0	0	0
x	x	x	x	1	0	0	1
x	x	x	1	0	0	1	0
x	x	1	0	0	0	1	1
x	1	0	0	0	1	0	0
1	0	0	0	0	1	0	1

The VHDL models have been synthesized and net lists have been generated for each of the architectures. The logic utilization of the various FPGA cells is shown in TABLE III. These results show that, whilst offering the greatest speed, the parallel implementation requires the greatest amount of logic by far. In addition to this, the long combinatorial chains mean that a large amount of pipelining would needed for the design to operate at higher frequencies. This fact would negate many of the speed gains that are generated from the architecture. This shows that the parallel architecture is only generally suitable for applications which require a greater amount of through put than is achievable with the other methods. The novel approach also requires more logic than the serial implementation although this increase is only modest. There is a 2.5% increase in LUT utilization and 1.5% increase in the FF utilization compared to the serial approach.

VI. SIMULATION AND IMPLEMENTATION RESULTS

The three models are simulated under a number of different scenarios to prove their operation. The output of the models are verified against a simulation model of the algorithm which was created using MATLAB. The ANFIS algorithm developed in MATLAB was created using automated training tools and scripts were then written to export this into a VHDL model. All three of the models have been successfully verified against the MATLAB simulation model. The full sets of simulations are shown in Fig 7 to Fig. 10. The VHDL models have one single bit input to indicate that a new piece of data is available to process, called in_valid in the waveforms. Similarly there is a single bit output which is set high whenever a new output is available, called fis_ready. The 8-bit input to the model is contained in the signal data_in and the output is presented on the signal fis_out.

When carrying out the simulations of the three VHDL models, the main criteria against which the performance is judged is the response time. The metric used for assessing this is the number of clock cycles required per calculation. These results show that the parallel implementation offers the fastest

TABLE III. LOGIC UTILIZATION

Architecture	Logic Utilization			
	LUT	FF	DSP	RAM
Parallel	56072	373	303	5
Serial	3524	1295	26	6
Novel	3615	1314	26	6

response time, taking a total of 3 clock cycles to generate an output. This speed does come at the expense of a large increase in the amount of logic required for implementation, as previously discussed. This means that this architecture is only suitable for cases which require very high throughput. The novel approach has the next fastest response time with the number of clocks required varying from a minimum of 15 to a maximum of 21, depending on the exact input values. Finally the serial approach is the slowest of the architectures, taking 36 clock cycles to complete a calculation cycle. All of the simulations carried out in this paper have been done using the Xilinx Vivado software suite, version 2016.3.

In order to further verify the novel ANFIS architecture, the VHDL has been committed to silicon and tested. As previously mentioned the hardware target for the model is the FPGA core in a Xilinx Zynq XC7Z020 device. The model has been tested by creating an on-board stimulus generator which emulates the stimulus generated in the VHDL test bench. In order to test the hardware target an on-board scope was added using a Xilinx IP core. The on-board scope is capable of capturing signals on the FPGA and outputting this to a PC using the JTAG interface. The waveforms, which are captured using the Xilinx Vivado software tool, are shown in Fig 11 and Fig 12. The hardware test results show that the hardware implementation matches with the previously verified VHDL model as shown in Fig 9

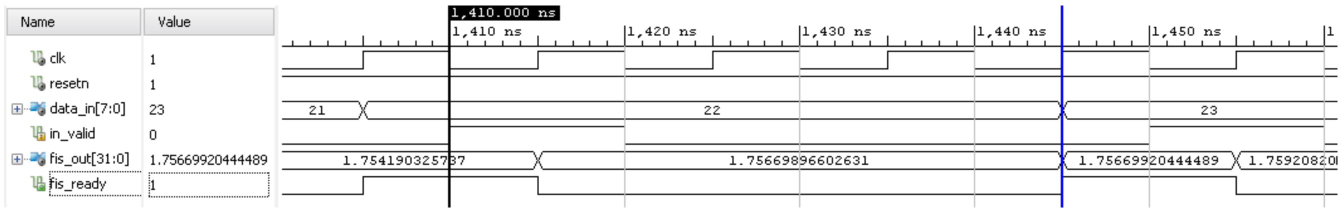


Fig.7. Parallel Implementation Simulation Result

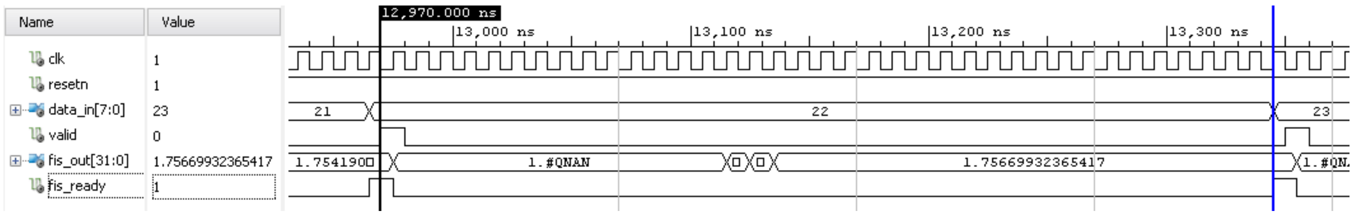


Fig.8. Serial Implementation Simulation Result

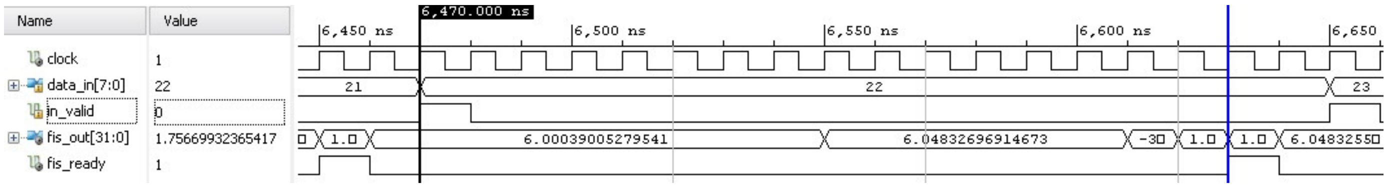


Fig. 9. Novel Implementation Simulation Result with 14 Clock Cycles

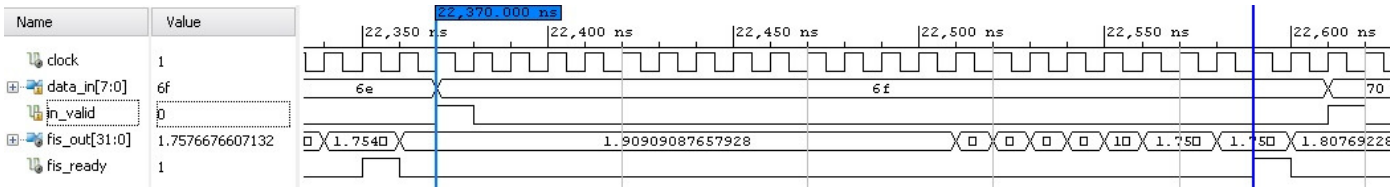


Fig. 10. Novel Implementation Simulation Result with 20 Clock Cycles

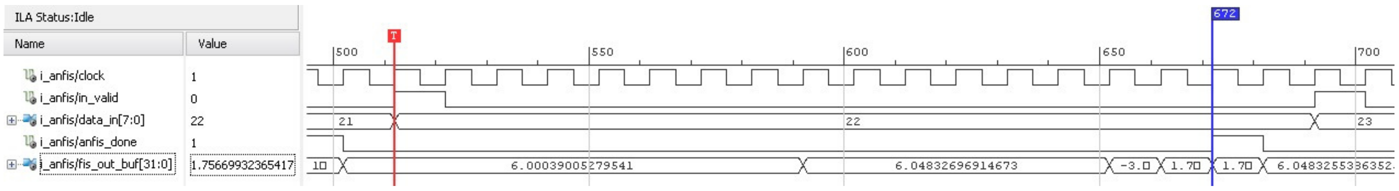


Fig. 11. Novel Architecture Hardware Test Result with 15 Clock Cycles

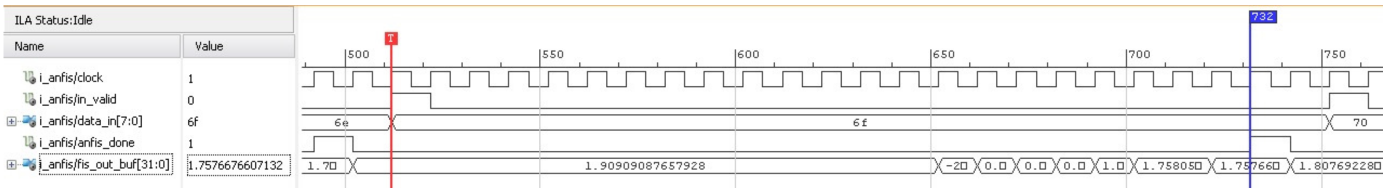


Fig. 12. Novel Architecture Hardware Test Result with 21 Clock Cycles

and Fig 10. Note that the fis_ready signal in the simulations has been replaced by the anfis_done signal in hardware.

Another important consideration which is not illustrated in the simulations is the scalability of the different architectures. If the implementation of the ANFIS algorithm was changed to include more than five members then this would have an adverse affect on the existing architectures. In the case of the parallel architecture, this would result in a large increase in the logic utilization. As an example, consider a new implementation using seven input members. This would mean that there would be a total of 49 consequence parameters as opposed to the current 25. This would result in a requirement for nearly double the amount of logic.

Whilst the serial architecture doesn't suffer with excessive increases in logic, calculation speed is affected by the functionality of the permutator circuit. In the five input member implementation a total of 25 permutations are possible. In comparison the seven member implementation would have a total of 49 possible permutations – an increase of 24. As the permutator must output every valid permutation, this would require an increase of 24 clocks for the calculation time. This would take the total number of clocks required per calculation from 37 up to 61. This shows that as the ANFIS implementation is made more complex by adding input members, the speed of the serial implementation decreases.

The novel architecture, on the other hand, suffers from neither of these limiting factors when the number of input members is increased. As the novel implementation is based upon the serial implementation, there would be a similarly modest increase in logic utilization. Due to the efficiency improvements in the permutator, the number of extra clock cycles required would also be significantly less than the serial architecture would require. It is difficult to put an exact figure on this for the novel architecture as it is very much implementation specific. In order to give an illustration, the algorithm presented in this paper can be retrained in MATLAB to use seven input members. In this case the distribution of the input members is similar to the five input implementation. As a result, the novel approach would still only require between 15 and 21 clock cycles for this implementation. This example is a good illustration of the improved scalability.

VII. CONCLUSION

A novel digital system architecture for the ANFIS algorithm is presented in this paper. This method improves upon the performance of the existing serial architecture by removing redundant calculation cycles. Models of the new architecture, as well as two existing architectures, were then created using VHDL. By simulating the models it was observed that the novel methodology offers improvements in calculation speed compared to the existing serial methodology on which it is based. The novel architecture is shown to require around half the number of clock cycles to perform the calculation. In addition to this, only a modest increase is required in the logic utilization – just a 2.5% increase in the number of LUTs and 1% more FFs. Whilst a parallel architecture was also simulated and shown to offer a faster response time, this came at the expense of using more than ten times as much logic in the target FPGA. This means that this parallel approach is only suitable for the most time sensitive applications. As an additional benefit, the novel approach also allows for greater scalability than the serial approach due to the removal of redundant calculation cycles. A case study was considered with two extra input members and through analysis it can be illustrated that the novel approach would not require any extra calculation time for this. In comparison, the serial approach would require an additional 24 clock cycles.

REFERENCES

[1] J-S, Jang , "ANFIS: Adaptive-Networtk-Based Fuzzy Inference System." *IEEE Transacrtions on Systems, Man and Cybernetics*, Vol. 23, Iss 3, pp. 665-685, 1993

[2] C.N. Pitas, D.E. Charilas, A.D. Panagopoulos and P. Constantinou, "Adaptive Neuro-Fuzzy Model for Speech and Voice Quality Prediction in Real-World Communication Networks." *IEEE Wireless Communications*, Vol. 20, Iss. 3, pp 1536 – 1284, 2013

[3] C. Chen, B. Zhang, G. Vachtsevanos and M.Orchard, "Machine Condition Predicition Based on Adaptive Neuro-Fuzzy and high-Oder Particle Filtering. "*IEEE Transactions on Industrial Electronics* Vol. 58, Iss. 9, pp. 4353-4364, 2011

[4] J.P.S. Catalao, H.M.I Pousinho and V.M.F Mendes, "Hybrid Wavelet-PSO-ANFIS Approach for Short Term Electricity Prices Forecasting." *IEEE Transactions on Power Systems*, Vol 26, Iss. 1, pp 137-144, 2011

[5] H. Abu-Rub, A. Iqbal, S. Mon Ahmed, F.Z. Peng and G. Baoming, "Quasi-Z-Source Inverter Based Photovoltaic Generation System with Maximum Power Tracking Control using ANFIS." *IEEE Transactions on Sustainable Energy*, Vol 4, Iss. 1, pp 11-20, 2012

[6] M. Singh and A.Chandra, "Real Time Implementation of ANFIS Control for Renewable Interfacing Inverter in 3P4W Distribution Network." *IEEE Transactions on Industrial Electronics*, Vol. 60, Iss. 1, pp. 121-128, 2012

[7] A. Chikh and A. Chandra, "An Optimal Maximum Power Point Tracking Algorithm for PV Systems with Climatic Parameters Estimation." *IEEE Transactions on Sustainable Energy*, Vol. 6, Iss.2, pp. 664-652,2015

[8] F. Gomez-Castaneda, G.M. Tornez-Xavier, L.M. Flores-Nava, O. Arellano-Cardenas and J.A. Moreno-Cadenas, "Photovoltaic Panel Emulator in FPGA Technology Using ANFIS Approach." *International Conference on Electrical Engineering, Computing Science and Automatic Control*, pp 1-6, 2014

[9] H.J.S Saldana and C. Silva-Cardenas, "A Digital Hardware Architecture for a Three-Input One-Output Zero-Order ANFIS." *Third Latin American Symposium on Circuits and Systems*, pp. 1-4, 2012

[10] P.R. Pande, P.L. Paikrao and D.S. Chaudhari, "Digital ANFIS Model Design." *International Journal of Soft Computing (IJSCE)*, vol 3, iss 1, pp 314 - 318, 2013

[11] A.R.Omondi and J.C. Rajapakse, "FPGA Implementations of Neural Networks." Dordrecht, The Netherlands: Springer, 2006

[12] P. Garcia. C.A. Garcia, L.M. Fernandez, F. Lorens and F. Jurado, "ANFIS Based Control of Grid Connected Hybrid System Integrating Renewable Energies, Hydrogen and Batteries." *IEEE Transactions on Industrial Informatics*, Vol. 10, Iss. 2, pp 1107 – 1117, 2013

[13] L. Wang and D.N. Truong, "Stability Enhancement of a Power System with PMSG-Based and a DFIG-Based Offshore Wind Farm Using a SVC With an Adaptive Network Based Fuzzy Inference System." *IEEE Transactions on Industrial Electronics*, Vol. 60, Iss. 7, pp 2799 – 2807, 2013.

[14] J. Darvill, A. Tisan and M. Cirstea, "An ANFIS-PI Based Boost Converter Control Scheme." *IEEE International Conference on Industrial Informations*, pp 632 – 639, 2015.