Direct Neural Networks Hardware Implementation Algorithm

Andrei Dinu, *Member, IEEE,* Marcian N. Cirstea, *Senior Member, IEEE,* Silvia E. Cirstea

A. Dinu is with Goodrich Corporation, Birmingham, UK (e-mail: andrei.dinu@goodrich.com).
M. N. Cirstea, S. E. Cirstea are with Anglia Ruskin University, UK. (marcian@ieee.org, silvia.cirstea@anglia.ac.uk).

***Abstract*— An algorithm for compact neural network hardware implementation is presented, which exploits special properties of the Boolean functions describing the operation of artificial neurones with step activation function. The algorithm contains three steps: ANN mathematical model digitisation, conversion of the digitised model into a logic gate structure, and hardware optimisation by elimination of redundant logic gates. A set of C++ programs automates algorithm implementation, generating optimised VHDL code. This strategy bridges the gap between ANN design software and hardware design packages (Xilinx). Although the method is directly applicable only to neurones with step activation functions, it can be extended to sigmoidal functions.**

***Index Terms*— Neural Networks, Hardware Implementation, FPGA.**

I. Introduction

ACCORDING to an European Network of Excellence report [1], the future implementation of hardware neural networks is shaped in 3 ways: i) by developing advanced techniques for mapping neural networks onto FPGA, ii) by developing innovative learning algorithms which are hardware-realizable [2], iii) by defining high-level descriptions of the neural algorithms in an industry standard to allow full simulations and fabrication and to produce demonstrators of the technology for industry. Such new designs will be of use to industry if the cost of adopting them is sufficiently low. Hardware-based neural networks are important to industry as they offer low power consumption and small size compared to PC software and they can be embedded in a wide range of systems. Software libraries exist for traditional Artificial Neural Network (ANN) models (Matlab). The industry-standard form is however VHDL or C++ parameterized modular code, allowing customization.

A range of research papers on ANN based controllers were published over the last decade ([3], [4]). Some recent publications ([5], [6], [7], [8]) consider the FPGA as an effective implementation solution of control algorithms for industrial applications. Hardware implemented ANNs have an important advantage over computer simulated ones by fully exploiting the parallel operation of the neurones, thereby achieving high speed of information processing [9]. Some VLSI algorithms achieve efficient implementation by using a combination of AND gates, OR gates and Threshold Gates (TG) [10]. This method leads to compact hardware structures but it cannot be used for FPGA implementation because TGs are not available in FPGA's Configurable Logic Blocks.

The algorithm presented in this letter is applicable to both ASIC and FPGA implementation of ANNs composed of neurones with step activation functions [10]. Each neurone is treated as a Boolean function and it is implemented separately, thus minimising implementation complexity. The most useful property of such a Boolean function is that if its truth table is constructed as a matrix with as many dimensions as neurone inputs, then the truth table has only one large group of '1' and one large group of '0'. The solid group of '1' is not visible when the Gray codification is used and thus classical Quine-McClusky algorithms or Karnaugh maps cannot efficiently be used. Our algorithm uses a different approach and generates a multilayer pyramidal hardware structure, where layers of AND gates alternate with layers of OR gates. The bottom layer consists of incomplete NOT gates, a structure to be optimised later by eliminating redundant logic gates groups. However, the method is effective only when the numbers of inputs and bits on each input are low, otherwise a classical circuit may be more efficient.

II. The Implementation Algorithm

Each neurone of the ANN is first converted into a binary equivalent neurone whose inputs are only '1' and '0', in a two-step process. Subsequently, the binary neurone model is iteratively transformed into a logic gate structure.

A. *Digitisation of One Neurone Mathematical Model*

The binary codification used for neurone inputs is the "two's complement", generally used to represent integers but it can be adapted for real values in the interval [-1, +1). Thus, considering a n-bit representation $b_{n-1}b_{n-2}b_{n-3}....b_1b_0$, the corresponding integer value (I_n) is given by:

$$I_n = -2^{n-1} \cdot b_{n-1} + \sum_{i=0}^{n-2} 2^i \cdot b_i \qquad (1)$$

The largest positive number, which can be represented on 'n' bits, is $2^{n-1}-1$, and -2^{n-1} the smallest. Real values between -1.0 and +1.0 can be represented by dividing the corresponding integer value I_n by 2^{n-1}. Thus, equation (2) illustrates the complementary code for real numbers:

$$R_n = \frac{I_n}{2^{n-1}} = -b_{n-1} + \sum_{i=0}^{n-2} 2^{-n+1+i} \cdot b_i \qquad (2)$$

The analogue neurone model is transformed, in two steps, into an appropriate digital model. At each stage, the input weights and the threshold levels of the initial NN are altered carefully, keeping the neurone functionality. This can be achieved by keeping constant the sign of the argument of the activation function:

$$\mathrm{sign}\left(\sum_{i=1}^{m} w_i \cdot x_i - t\right) = \mathrm{sign}(net - t) = constant \qquad (3)$$

However, for mathematical simplicity, a more restrictive condition is used instead: argument "net-t" of the activation function is kept itself constant rather than only its sign:

$$\sum_{i=1}^{m} w_i \cdot x_i - t = net - t = constant \qquad (4)$$

➢ *Conversion Stage One*

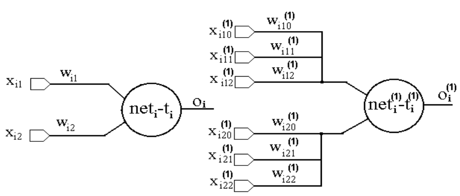


Fig. 1 Neurone model before / after stage one conversion

The first step transforms the analogue inputs of the neurones into digital inputs expressed as groups of n_b bits. This process is associated with transforming each analogue neurone input into an equivalent group of n_b binary inputs. The task is achieved by splitting each input defined by its initial weight w_{ij} into n_b subinputs, whose weights w_{ijp} (p=0,1, ..., n_b-1) are calculated as follows [11]:

$$\begin{cases} w_{ijp}^{(1)} = \dfrac{2^{p+1}}{2^{n_b}} \cdot w_{ij} \;\forall p < n_b - 1 \\ w_{ij(n_b-1)}^{(1)} = -w_{ij} \\ t_i^{(1)} = t_i \end{cases} \qquad (5)$$

The superscripts '(1)' and '(2)' refer to the respective conversion stage. The initial 'm' inputs are turned into 'm' input clusters, each containing 'n_b' subinputs (Fig. 1). The symbol 'w_{ij}' stands for the weight number 'j' of the neurone 'i' in the network, while '$w_{ijp}^{(1)}$' represents the weight of subinput 'p' in cluster 'j' pertaining to neurone 'i'. According to the previous considerations, only those neurone parameter changes that maintain the argument "net_i-t_i" of the activation function constant are allowed. The argument after the first conversion stage is calculated as:

$$net_i^{(1)} - t_i^{(1)} = \sum_{j=1}^{m}\sum_{p=0}^{n_b-1} w_{ijp}^{(1)} \cdot x_{jp}^{(1)} - t_i^{(1)} = \sum_{j=1}^{m}\left(-w_{ij} \cdot x_{jp}^{(1)} + \sum_{p=0}^{n_b-2} w_{ij} \cdot \frac{2^{p+1}}{2^{n_b}} \cdot x_{jp}^{(1)}\right) - t_i^{(1)} \qquad (6)$$

where $x_{jp}^{(1)}$ (p=0,1,2,...n_b-1) are bits of the complementary code received by each new neurone input. The equation is:

$$net_i^{(1)} - t_i^{(1)} = \sum_{j=1}^{m} w_{ij} \cdot \left(-x_{j(n_b-1)}^{(1)} + \sum_{p=0}^{n_b-2} 2^{-n_b+p+1} \cdot x_{jp}^{(1)}\right) - t_i^{(1)} \qquad (7)$$

The expression in brackets relates to the complementary code definition given in equation (2). Then (6) becomes:

$$net_i^{(1)} - t_i^{(1)} = \sum_{j=1}^{m} w_{ij} \cdot x_j - t_i^{(1)} = \sum_{j=1}^{m} w_{ij} \cdot x_j - t_i = net_i - t_i \qquad (8)$$

where x_j is an analogue input value of the initial neurone. This meets the condition expressed by (4). Thus, the codification style based on complementary code has been introduced and the required parameter modifications have been performed, without changing the neurone's behaviour.

➢ *Conversion Stage Two*

The second conversion stage aims to replace the neurones with negative weights resulting from the first stage, with equivalent ones, having only positive weights, by using only the module of their values: $w_{ijp}^{(2)} = \left|w_{ijp}^{(1)}\right|$. This means that supplementary parameter alterations are required in order to counteract the neurone behaviour alteration caused by changing the sign of some input weights. A simple solution is to reverse the value of the affected input bits. The modification can be implemented into hardware with NOT logic gates. The relationship between the input bits $x_{ijp}^{(2)}$ and those at stage-one ($x_{ijp}^{(1)}$) is:

$$x_{ijp}^{(2)} = \begin{cases} x_{ijp}^{(1)} \text{ if } w_{ijp}^{(1)} > 0 \\ 1 - x_{ijp}^{(1)} \text{ if } w_{ijp}^{(1)} < 0 \end{cases} \qquad (9)$$

These two alternatives can be compressed into:

$$x_{ijp}^{(2)} = \frac{1 - \mathrm{sign}\left(w_{ijp}^{(1)}\right)}{2} + \mathrm{sign}\left(w_{ijp}^{(1)}\right) \cdot x_{ijp}^{(1)} \qquad (10)$$

The transfer function argument is calculated as [11]:

$$net_i^{(2)} - t_i^{(2)} = \sum_{j=1}^{m}\sum_{p=0}^{n_b-1} w_{ijp}^{(1)} \cdot x_{ijp}^{(1)} + \sum_{j=1}^{m}\sum_{p=0}^{n_b-1}\left(\frac{\left|w_{ijp}^{(1)}\right| - w_{ijp}^{(1)}}{2}\right) - t_i^{(2)} \qquad (11)$$

The arguments of the activation function before and after the second conversion stage have to be equal:

$$\sum_{j=1}^{m}\sum_{p=0}^{n_b-1} w_{ijp}^{(1)} \cdot x_{ijp}^{(1)} + \sum_{j=1}^{m}\sum_{p=0}^{n_b-1}\left(\frac{\left|w_{ijp}^{(1)}\right| - w_{ijp}^{(1)}}{2}\right) - t_i^{(2)} = \sum_{j=1}^{m}\sum_{p=0}^{n_b-1} w_{ijp}^{(1)} \cdot x_{ijp}^{(1)} - t_i^{(1)} \qquad (12)$$

Therefore, the threshold level of the stage-two neurones is:

$$t_i^{(2)} = t_i^{(1)} + \sum_{j=1}^{m}\sum_{p=0}^{n_b-1} \frac{\left|w_{ijp}^{(1)}\right| - w_{ijp}^{(1)}}{2} \qquad (13)$$

The stage-one neurone parameters in equation (13) depend on the initial parameters of the analogue neurone as described by (5). Consequently, substituting (5) in (13):

$$t_i^{(2)} = t_i + \sum_{j=1}^{m} \frac{\left|w_{ij}\right| + w_{ij}}{2} + \sum_{j=1}^{m}\sum_{p=0}^{n_b-2} \frac{\dfrac{2^{p+1}}{2^{n_b}} \cdot \left|w_{ij}\right| - \dfrac{2^{p+1}}{2^{n_b}} \cdot w_{ij}}{2} \qquad (14)$$

This expression can be successively transformed [11] into:

$$t_i^{(2)} = t_i + \left(1 - 2^{-n_b}\right) \cdot \sum_{j=1}^{m}\left|w_{ij}\right| + 2^{-n_b} \cdot \sum_{j=1}^{m} w_{ij} \qquad (15)$$

So, the neurone parameters after stage 2 can be calculated as a function of initial analogue neurone parameters:

$$\begin{cases} w_{ijp}^{(2)} = \dfrac{2^{p+1}}{2^{n_b}} \cdot \left|w_{ij}\right| \quad p = 0,1,2,...n_b - 1 \\ t_i^{(2)} = t_i + \left(1 - 2^{-n_b}\right) \cdot \sum_{j=1}^{m}\left|w_{ij}\right| + 2^{-n_b} \cdot \sum_{j=1}^{m} w_{ij} \end{cases} \qquad (16)$$

B. *The Binary Neurone Implementation and Optimization*

The ANN implementation into a hardware structure is done separately for each neurone and requires at first that the input weights $w^{(2)}_{ijp}$ are sorted in descending order, in an array with A=m · n_b elements: $w_1^s, w_2^s, w_3^s, ..., w_A^s$, where 'm' is the number of initial analogue neurone inputs and 'n_b' the number of bits for each input binary code. The weights correspond to the input binary signals: $x_1^s, x_2^s, \ldots x_A^s$. An iterative conversion procedure is used to analyse the input weights and to generate the logic gate implementation netlist description. At each step, a larger neurone is split into subneurones. Some of them can be implemented with only a few AND and OR logic gates, while the rest are further decomposed into simpler subneurones, until all have been implemented. Several important concepts and definitions are presented in [12], along with the step by step iterative implementation procedure, which ends by adding inverters to those inputs corresponding to the initial negative weights at stage one of neural model digitisation. The hardware implementation netlist obtained has redundancies both inside each neurone and across different neurones. Most are eliminated using a simple procedure: the file is repeatedly analysed and when same type logic gates are found, of same input signals, all but one are removed from the netlist and interconnections are updated; the cycle ends when no gates can be removed.

C. Neurone Implementation Example

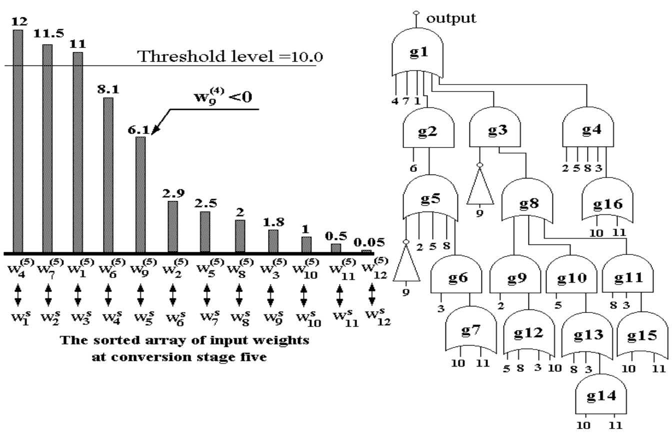


Fig. 2 Digital mathematical model to gate conversion

The sample in Fig. 2 shows a neurone with 12 input weights and positive threshold level. The weights are sorted in descending order and a recursive implementation starts. The first three weights are larger than the threshold, so inputs 4, 7, 1 will drive an OR gate along with the subneurones built using the other subgroups [11].

D. Automated Implementation Method

The algorithm was automated using C++ programs that generate a netlist description of the circuit, optimize it and then generate the VHDL code. In terms of the software, there is no limitation of the ANN size. The characteristics of the ANN are introduced in the C++ program as a matrix text file (.csv format). A feed forward ANN with 3 subnetworks generating the PWM switching pattern for an inverter, was designed [9] using this method (Fig. 3):

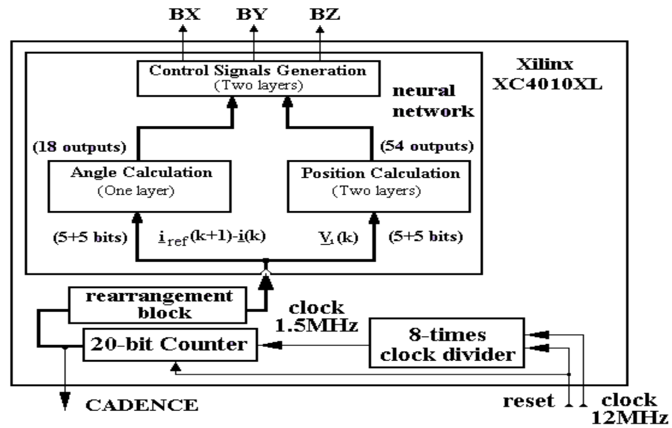


Fig. 3 ANN structure and testbench for operation speed testing

○ ***Angle*** analyses the argument of current difference vector.
○ ***Position*** analyses the argument and value of the voltage.
○ ***Control Signals*** generates three PWM binary outputs.

In contrast with training algorithms, constructive ones determine both the network architecture and the neurone weights and are guaranteed to converge in finite time. The numerical values of all neurone weights and thresholds were calculated [11] using a geometric constructive solution known as Voronoi diagrams [7]. For this work, the complex plane is divided into triangular Voronoi cells. The master program allows user control over main parameters: i) Number of Voronoi cells, ii) Number of sectors dividing the 360 degrees interval for argument analysis, iii) Number of bits used to code the components of the two complex inputs iv) Maximum fan-in for the VHDL logic gate model. The desired performance / complexity ratio is adopted. In this case, 5 bits to code each component of the two complex inputs gives enough precision (delays less than 100 ns), resulting in a total number of logic gates of 1329 on 14+6=20 layers [10], which fits Xilinx XC4010XL FPGA.

When the number of inputs and bits on each input is low (precision appropriate for drives), this method is more effective than a classical digital circuit design implemented in FPGA. For a high number of bits/controller inputs, the NN approach can be less effective than a classical circuit. The explanation is that in the NN approach the complexity of the resulting circuit raises exponentially with these numbers, whereas in a traditional approach, the complexity increases quadratically. The case study presented in this paper was implemented as part of an induction motor controller in a 10,000 gates equivalent FPGA, as opposed to a classical digital vector control circuit, for controlling the same motor, which was commissioned in our research group, using 99% of a 40,000 gate equivalent FPGA [12].

III. SIMULATION AND VERIFICATION

The ANN operation speed was tested by designing a VHDL testbench (Fig. 3). Input patterns are generated by a 20-bit counter and a pseudo-random sequence block. A simulation waveform is shown in Fig. 4, illustrating delay readings of 39.5 ns and 80.5 ns.

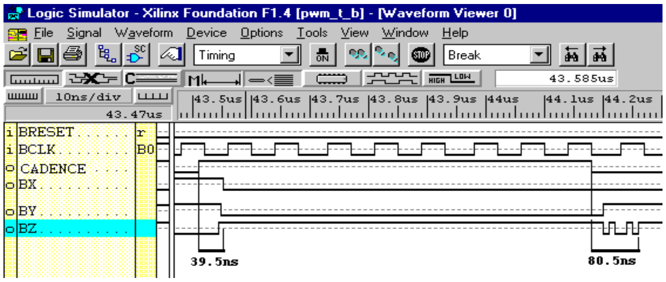


Fig. 4 Timing simulation using Xilinx software

Generally, oscilloscope measurements taken on the XS40 board, containing a Xilinx XC4010XL FPGA, indicate delays not exceeding 100 ns. Thus, the propagation time is less than 1.5 clock cycles, which demonstrates the advantage of higher operating speeds comparing with other digital circuits [13].

IV. Conclusions

A new digital hardware implementation strategy for feed-forward ANNs with step activation functions is reported. The novel algorithm treats each neurone as a special case of Boolean function with properties that can be exploited to achieve compact implementation. This is accomplished by means of reusable VHDL code that can be easily translated into an FPGA implementation, using suitable EDA software.

The VHDL programs bridge the gap between the facilities offered by simulation software and software packages specialised in hardware design. This method is most efficient for a low number of inputs/bits on each input, otherwise a classical circuit may be preferred.

V. References

[1] NEuroNet Roadmap, "Future Prospects for Neural Networks", European Network of Excellence in Neural Networks, 2001. http://cordis.europa.eu/ist/ka3/iaf/links.htm

[2] B. M. Wilamowski, N. J. Cotton, O. Kaynak, G. Dundar, "Computing Gradient Vector and Jacobian Matrix in Arbitrarily Connected Neural Networks," *IEEE Trans. on Ind. Electronics*, vol. 55, no. 10, pp. 3784-3790, Oct. 2008.

[3] Special Section: "Neural Networks for Robotics", *IEEE Trans. on Ind. Electronics*, vol. 44, no. 6, Dec. 1997.

[4] Special Section: "Fusion of Neural Nets, Fuzzy Systems and Genetic Algorithms in Industrial Applications", *IEEE Trans. on Ind. Electronics*, vol.46, no.6, 1999.

[5] J. J. Rodriguez-Andina, M. J. Moure, M. D. Valdes: "Features, Design Tools and Application Domains of FPGAs", *IEEE Trans. on Ind. Electronics*, vol.54, no.4, pp.1810-1823. Aug. 2007.

[6] E. Monmasson, M.N. Cirstea: "FPGA Design Methodology for Industrial Control Systems – a Review", *IEEE Transactions on Ind. Electronics*, vol. 54, no.4, pp. 1824-1842, Aug. 2007.

[7] L. Vachhani, K. Sridharan: "Hardware efficient Prediction-Correction-Based Generalised Vornonoi Diagram Construction and FPGA Implementation", *IEEE Tran. on Ind. Electronics*, vol. 55, no.4, pp. 1558-1569. April 2008.

[8] Da Zhang, Hui Li, "A Stochastic-Based FPGA Controller for an Induction Motor Drive With Integrated Neural Network Algorithms," *IEEE Trans. on Ind. Electronics*, vol. 55, no. 2, pp. 551-561, Feb. 2008.

[9] M. N. Cirstea, A. Dinu, J. Khor and M. McCormick: "Neural and Fuzzy Logic Control of Drives and Power Systems", Elsevier Science Ltd., Oxford, UK, 2002.

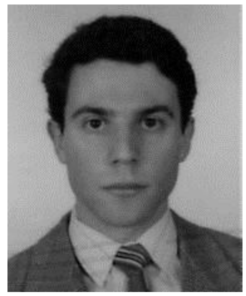
[10] A. Dinu, M. N. Cirstea: "A Digital Neural Network FPGA Direct Hardware Implementation Algorithm", Proc. of ISIE 2007, Vigo, Spain, pp.2307-2312.

[11] A. Dinu, "FPGA Neural Controller for Three Phase Sensorless Induction Motor Drive Systems", PhD Thesis, De Montfort University, 2000.

[12] A. Aounis, M. McCormick, M.N. Cirstea: "A Novel Approach to Induction Motor Controller Design and Implementation", Proc. of IEEE Power Conversion Conference (PCC), Osaka, pp.993-998, April 2002.

[13] B. K. Bose, "Neural Network Applications in Power Electronics and Motor Drives - An Introduction and Perspective", *IEEE Transactions on Ind. Electronics*, vol. 54, no. 1, pp. 14-33, Feb. 2007.

Author Biographies



Andrei Dinu (M'05) received the BEng and MSc degrees in electrical engineering from Transilvania University of Brasov, Romania, in 1995 and 1996, respectively. He completed his PhD in electrical / electronic engineering at De Montfort University, UK, in 2000, with a thesis concerning the sensorless control of induction motors using hardware implemented neural networks. He was then appointed lecturer at the same university, where he conducted research in electrical drives control until 2003, when he moved to industry.
He was design engineer at Datalink Electronics (Loughborough, UK), and in 2004 he joined Goodrich Corporation as control systems engineer. He is co-author of two books and over 20 refereed papers, one of which has received an ABB award. He currently works on R&D projects carried out by the Electromagnetic Systems Technical Centre of Goodrich Corporation (Birmingham, UK). Dr. Dinu is also Member of IET and Member of the IEEE Industrial Electronics Society (IES).

Marcian N. Cirstea (M'97-SM'04) completed a PhD at Nottingham Trent University, UK, in 1996, after obtaining a degree in electrical engineering from Transilvania University of Brasov, Romania. He is currently Professor of Industrial Electronics and Head of Computing and Technology Department at Anglia Ruskin University in Cambridge, UK, after previously working for De Montfort University, UK.
Prof. Cirstea has co-authored several technical books and over 100 peer reviewed papers, three of which have received awards. His research is focused on digital / FPGA controller design for power electronics, with recent interests in renewable energy. He is founder and past Chairman of the 'Electronic Systems on Chip' Technical Committee of the IEEE Industrial Electronics Society, Member of IET and Chartered Engineer (CEng). Prof. Cirstea is also Associate Editor for IEEE Transactions on Industrial Electronics and was General Co-Chair of ISIE conference (Cambridge, 2008). He currently coordinates an European renewable energy project consortium.

Silvia E. Cirstea received a BSc and an MSc in mathematics from the University of Bucharest, Romania, and a PhD in electronics from De Montfort University, UK, with a thesis on depth extraction from 3D integral images. She then worked at the Central Laboratory of the Research Councils in the field of theoretical modeling of radiowave propagation, and at the Medical Research Council's Institute of Hearing Research, UK, being involved in acoustic modeling and sound synthesis.
Since 2005, Dr. Cirstea is a lecturer in Applied Mathematics and Media Technology at Anglia Ruskin University, Cambridge, UK. Her current research interests are in mathematical modeling, artificial intelligence algorithms and simulation for engineering and physical applications.