Artificial Neural Network Implementation on a Fine-Grained FPGA

P. Lysaght, J. Stockwood, J. Law and D. Girma

Communications Division,
Department of Electronic and Electrical Engineering,
University of Strathclyde,
Glasgow G1 1XW

Abstract This paper reports on the implementation of an Artificial Neural Network (ANN) on an Atmel AT6005 Field Programmable Gate Array (FPGA). The work was carried out as an experiment in mapping a bit-level, logically intensive application onto the specific logic resources of a fine-grained FPGA. By exploiting the reconfiguration capabilities of the Atmel FPGA, individual layers of the network are time multiplexed onto the logic array. This allows a larger ANN to be implemented on a single FPGA at the expense of slower overall system operation.

1. Introduction

Artificial neural networks, or *connectionist classifiers*, are massively parallel computation systems that are based on simplified models of the human brain. Their complex classification capabilities, combined with properties such as generalisation, fault–tolerance and learning make them attractive for a range of applications that conventional computers find difficult. Examples of these include video motion detection, hand-written character recognition and complex control tasks.

Traditionally, ANNs have been simulated in software or implemented directly in special-purpose digital and analogue hardware. More recently, ANNs have been implemented with reconfigurable FPGAs. These devices combine programmability with the increased speed of operation associated with parallel hardware solutions. One of the principal restrictions of this approach, however, is the limited logic density of FPGAs resulting from the intrinsic overhead of device programmability.

This paper presents an alternative approach to previously reported neural network implementations on FPGAs [2][3][7]. The novelty of the design is achieved by exploiting several design ideas which have been reported previously in different designs and by combining them to form a new implementation. The design is based on a fine–grained FPGA implementation of an ANN in contrast to most of the FPGA implementations reported to date. It emphasises careful selection of network topology and methods of realisation to produce a circuit which maps well to the special requirements of fine-grained architectures. These include the realisation of the ANN using digital pulse–stream techniques and the choice of a feedforward network topology. It further exploits the use of run–time device reconfiguration to time–

multiplex network layers to offset the logic density limitations of current devices. These topics are introduced in section two, where a reconfigurable pulse-stream ANN architecture [5][6] that is well suited for implementation on fine–grained FPGAs is described. Section three reviews some of the physical design issues that arose when mapping the ANN onto the AT6005 architecture, and in section four the performance of the network is appraised.

2. Reconfigurable ANN based on Pulse–Stream Arithmetic

2.1 Overview

ANNs employ large numbers of highly interconnected processing nodes, or *neurons*. Each neuron contains a number of synapses, which multiply each neuron input by a weight value. The weighted inputs are accumulated and passed through a non–linear *activation function* as illustrated in Fig. 1. These arithmetic–intensive operations and numerous interconnections are expensive in terms of logic and routing resources when implemented on an FPGA. Typically, as a result of these restrictions, expensive arrays of FPGAs have to be employed to implement "useful" networks [2], or alternatively, a single neuron is placed on the FPGA and used to emulate a network serially [3].

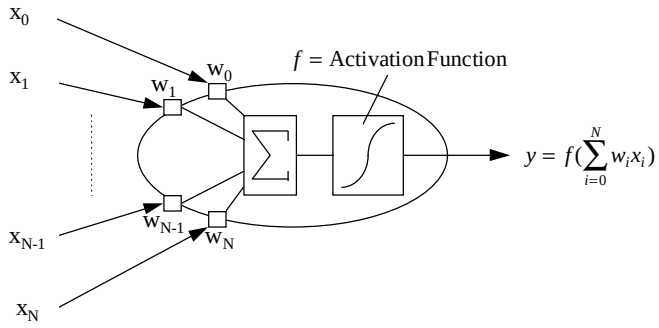

Fig. 1. The components of a simple artificial neuron.

The reported ANN incorporates three approaches to overcoming logic density limitations. First, pulse-stream arithmetic is used to provide an efficient mapping of the network onto a fine-grained FPGA. This technique is discussed in more detail in section 2.2.

Second, a reduction in the number of inter–neuron connections, which consume valuable routing resources, is made by adopting a layered, feed–forward network topology. As Fig. 2 shows, in contrast to the fully–interconnected network, the layered topology has connections only between nodes in adjacent layers. Further, supervised training is used to eliminate the need for feedback connections. This makes for easier partitioning of the network, since data flow through the network is uni–directional, from the input layer to the output layer.

Finally, by exploiting the reconfigurability of static memory–based FPGAs, the ANN can be *time-multiplexed* so that one physical layer is reconfigured to perform the function of all the other network layers. This makes it possible to implement a much larger design than would otherwise be possible on a single device. However, for this strategy to be successful it is important that the time spent reconfiguring the FPGA is relatively short, otherwise the speed of the overall network is severely degraded.

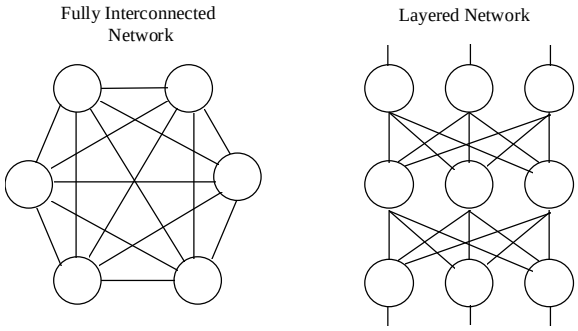

Fig. 2. Fully interconnected and layered ANN topologies

The ANN is implemented on an Atmel AT6005 (formerly Concurrent Cli6005). This is a fine-grained FPGA which is presently the only commercially available device capable of being *dynamically reconfigured*, i.e. selectively reconfigured while the logic array is active [4]. It will be shown that by exploiting this capability, and only reconfiguring those parts of the array which differ between network layers, it is possible to dramatically reduce the amount of system processing time that is lost during reconfiguration.

2.2 Pulse Stream Arithmetic

Pulse Frequency Modulation (PFM) is a coding scheme where circuit state values are represented by the frequency of narrow constant–width pulses. Fig. 3 shows an example of PFM, where the fractional value 7/16 is represented by the presence of 7 pulses in a 16–pulse window. Signals encoded in this manner can be summed and multiplied using simple logic gates. This technique, known as *pulse–stream arithmetic* [5], maps well onto fine–grained FPGAs such as the Atmel AT6005 which contain a large number of low fan–in gates.

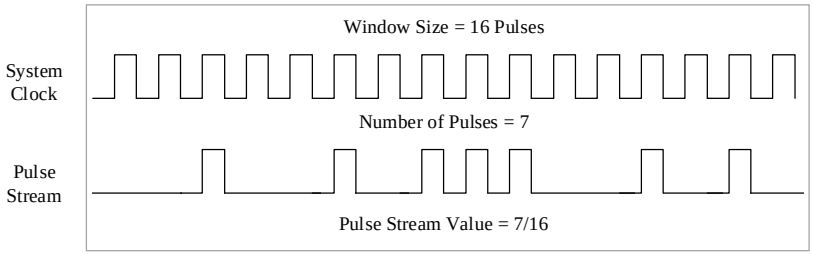

Fig. 3. Example of a Pulse Frequency Modulated signal.

2.3 Pulse Stream Neuron

In Section 1 the principal components of an artificial neuron were introduced. Here, the digital pulse–stream implementation of synaptic weight multiplication, post–synaptic summation and non–linear activation are described.

The inputs to the ANN are encoded as a constant stream of narrow pulses. Within each synapse, this pulse stream must be gated so that only a certain proportion of the pulses are allowed to pass through to the summation stage of the neuron. This proportion represents the value of the synaptic weight. A suitable gating function can be constructed by selectively ORing together a series of *chopping clocks* [5]. These are synchronous, non–overlapping binary clocks with duty cycles of 1/2, 1/4, 1/8 and so on. Fig. 4 shows a 4–bit chopping clock generator which can be used to construct weights in the range 0 to 15/16. Multiplication of the input pulse–stream by the weight value can be achieved by simply ANDing the input and the gating function, as shown in the diagram.

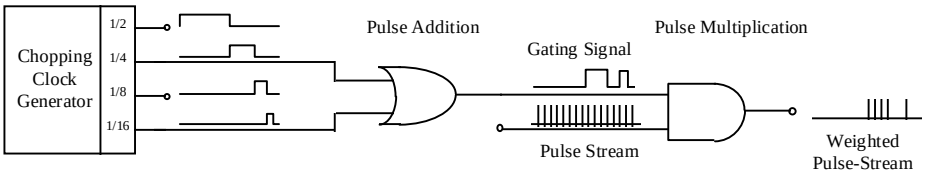

Fig. 4. Pulse arithmetic using simple logic gates

A synapse output is either *excitatory*, i.e. it increases the chance of the neuron firing, or *inhibitory*. In the pulse–stream neuron, positive and negative synaptic weights are accomplished by feeding excitatory and inhibitory synapse outputs to separate *up* and *down* inputs of a binary counter.

The neuron activation function is a simple binary step function, rather than the sigmoid function that is often used. There are two principal reasons behind this choice:

1. The sigmoid function is considerably more complex to implement, and requires neuron outputs to have a range of values rather than a simple binary output.

2. The binary step function's primary limitation applies to networks which employ back–propagation learning, which are less likely to converge on a correct solution without the smoothing effect of the sigmoid. The ANN reported here uses supervised learning, so this restriction is less relevant.

The output of the neuron is therefore calculated using a simple thresholding operation based on the most significant bit of the counter. Fig. 5 shows a block diagram of the complete digital pulse–stream neuron.

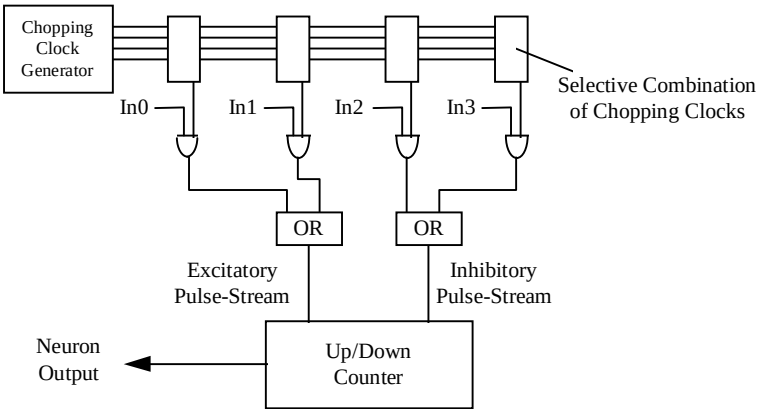

Fig. 5. Pulse–Stream Neuron

2.4 Pulse Stream Artificial Neural Network

Fig. 6 shows pulse–stream neurons connected together to form a single layer of the ANN. A layer consists of a maximum of four neurons, each with four synapses to allow full connectivity between successive network layers. The restriction to four neurons is imposed by the need to lay out the design on a single AT6005 device, and is discussed more fully in the following sections.

Inputs to the circuit are latched and encoded into non–overlapping pulse–streams so that on any given system clock cycle a pulse appears on only one input line. This ensures that pulses are processed one at a time by the neural counter. The chopping clocks are distributed to every synapse, where they are selectively combined to represent the weight value. Synaptic weights have a resolution of four bits. Higher resolution weights require more chopping clocks to be distributed to the synapses. Moreover, each additional weight bit doubles the number of pulses needed to represent circuit values and hence halves the processing speed of the network. Four bit weights were therefore chosen as a compromise between speed of operation and accuracy.

After processing of a network layer is complete, the neuron outputs are latched, and the FPGA is reconfigured to load the next layer. Any unused neurons in a layer can effectively be "switched off" by assigning them zero–valued weights. This means that the only parts of the circuit to be reconfigured are the OR gates in each synapse which are used to combine chopping clocks.

After reconfiguration, the previous layer's outputs are fed to the input latches and the next layer processed. When the final layer is completed the network outputs can be sampled.

To implement the complete circuit within the FPGA, it is important that both the input and output latches, and the FSM which controls reconfiguration, retain their state during device reprogramming. This requires dynamic reconfiguration, i.e. partial reconfiguration while the logic array of the FPGA remains active [4]. Note that in this particular system no datapath processing takes place on the logic array during reconfiguration. This limited form of dynamic reprogramming, where the logic array remains active only to maintain storage values, constitutes a sub–class of the wider class of dynamic reconfiguration. Currently, the only commercially available FPGAs capable of dynamic reconfiguration are the Atmel AT6000 series [1].

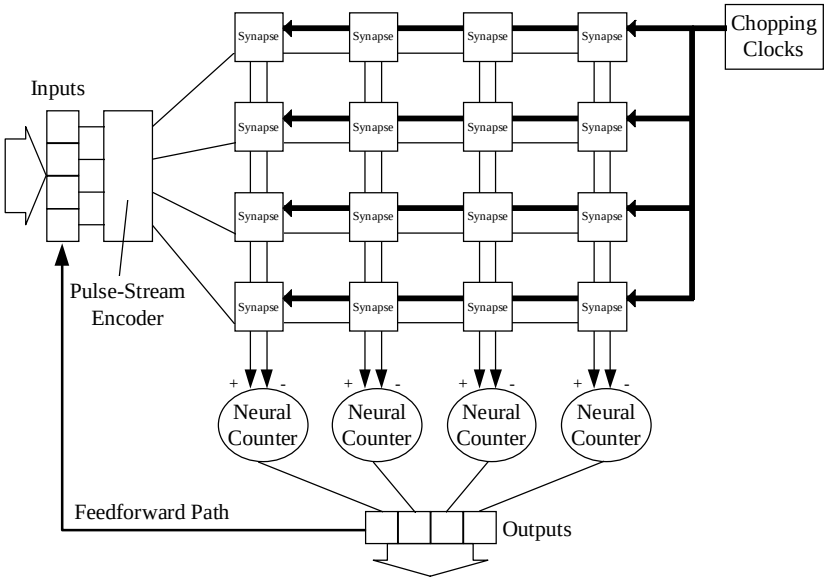

Fig. 6. Single Layer of Pulse–Stream ANN

3. Implementation on the Atmel AT6005

3.1 AT 6000 Series Architecture

The Atmel AT6005 FPGA comprises an array of 54 × 54 fine–grained cells, each of which can implement all common 2–input functions, or certain functions of 3 inputs

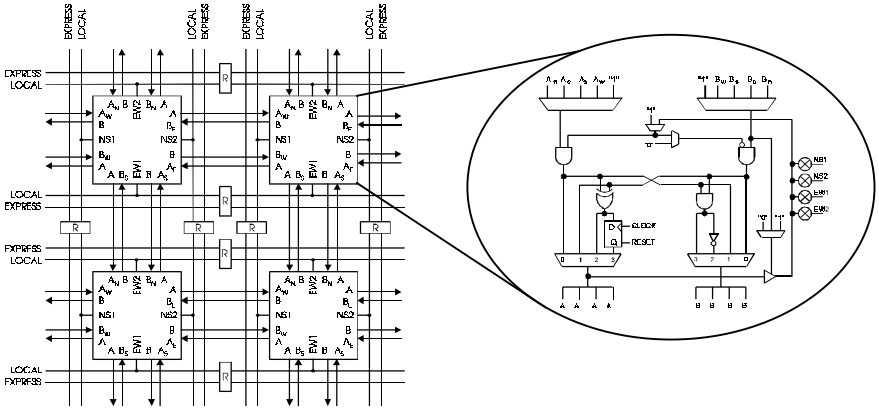

Fig. 7. Atmel AT6000 Series Architecture

along with a single storage register (see Fig. 7). Routing resources are split between slower through–cell connections and fast long–range busses, a limited number of

which are available to each block of 8 × 8 cells. The equivalent gate capacity of this architecture is quoted by the manufacturer as 5000 gates.

3.2 Circuit Layout on the AT6005

The pulse–stream ANN circuitry was manually placed and routed on the AT6005 FPGA for the following reasons:

- The Atmel APR tool employs a generous placement algorithm with respect to inter–component spacing. This appears to be optimised for maximum routing flexibility, but makes it difficult to achieve the degree of macro clustering required for this design.
- Timing is critical when implementing pulse–stream circuitry, as excessive signal skew can result in errors due to two or more pulses overlapping. Sub–circuits therefore have to be placed symmetrically such that delays on the signal lines which distribute pulse–streams and chopping clocks are well balanced. These special timing requirements are difficult to achieve with the current Atmel tools, which use a simple ordered list of nets to enable the designer to prioritise routing. A more advanced timing–driven layout tool such as that supplied by Xilinx would be needed to provide the necessary flexibility [8].
- The layout of the synapse circuits has to be optimised to minimise the time needed to reconfigure the device between layers of the network.

An Interactive Layout Editor is shipped with the development system, and this was used for manual design layout. Fig. 8 shows the floorplan of the FPGA with the first layer of the ANN after placement and routing.

The diagram does not fully indicate the extensive amount of long–range routing consumed by the design. Considerable areas of the logic array had cells which could not be used for logic because the adjacent routing busses were already heavily committed. The only way into and out of such areas is via through–cell routing, which is in general inappropriate for anything but short nets.

A potential shortcoming of the Atmel architecture was encountered during circuit layout. As with most designs, the ANN requires a large number of OR–gates, including a number with wide inputs. Unfortunately, both the 2–input OR–gate macros have limitations – one is slow and takes up three cells, while the single cell version is fast but has inflexible connections. Furthermore, the Atmel literature indicates that it is not possible to implement a totally glitch–free single cell OR function, due to the nature of the internal cell structure. The provision of a wired–OR capability would have been a considerable advantage for this design.

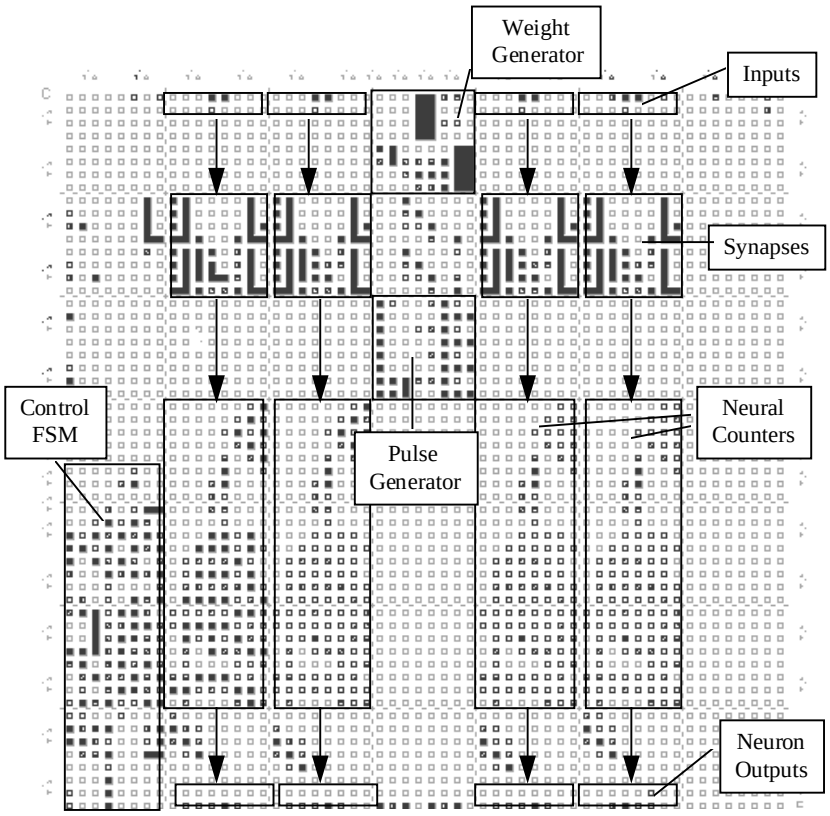

Fig. 8. Layout of a single network layer on the AT6005

The equivalent gate count calculated by the Atmel software for a single network layer (see Fig. 9) equates to a utilisation figure of 24.3%. It is clear from the table that routing forms a very significant proportion of the layout – more cells are used for routing than for logic (462 routing cells versus 476 logic cells), while 529 local and 68 express buses are also used.

Utilization Summary	Utilized
Number of Macros	273
Number of Flip-Flops	59
Number of Gates	417
Number of Turns	65
Number of Buses:	597
Local Buses:	529
Express Buses:	68
Number of IO's:	13 of 64
Number of Cells:	938 of 3136
Number of Equivalent	1216.5

Fig. 9. Design utilisation summary produced by Atmel software

3.3 Reconfiguration of the AT6005

To fully reconfigure the Atmel chip takes a minimum of 808µs, although this is only necessary when loading the first network layer. For subsequent layers partial reconfiguration can be used since the only changes to be made are to the synaptic weights.

Partial configurations are loaded into the device as a series of one or more *windows*, each of which contains programming data for a single contiguous block of cells along a row of the device. Every reconfiguration has an overhead of 10 bytes for preamble and control information, and each separate window within the bitstream carries an additional overhead of 5 bytes. In order to maximise reconfiguration speed, therefore, the following rules apply:

- The number of cells reconfigured should be the absolute minimum necessary to effect the required circuit changes.
- Configurations should be loaded at the maximum permissible rate, which in the case of the AT6005 is 10MHz. This typically requires some form of direct memory access.
- The number of configuration windows should be kept to a minimum. This has implications for circuit layout, since the reconfiguration of contiguous blocks of cells is faster than a "fragmented" reconfiguration.

It is worth noting that if any two windows are separated by less than three cells (i.e. 6 bytes of configuration data) it is faster to merge the two windows and overlay the intervening cells with an identical configuration. Experiments suggest that any stored results in these cells are unaffected by the reconfiguration operation, although this is not specified in the Atmel documentation.

4. Results and Performance

To date, the ANN has only been tested with the binary XOR function. This simple problem is non–linearly separable, which means that it requires a network with at least one hidden layer. The appropriate synaptic weights were calculated manually and subsequently incorporated into the FPGA configurations as detailed in section 2.4.

Testing of the ANN took place with the aid of an FPGA prototyping system which was developed in–house. This is based around a pair of Inmos Transputers which handle communications between the FPGA and a host computer, and also provide control over reconfiguration of the AT6005. Whilst this system is highly flexible, it is currently unable to match the maximum configuration loading rate of the AT6005, which would require a write cycle of 100ns. A mechanism to allow the FPGA to directly access fast memory to achieve full reconfiguration speed is under development at the time of writing.

With a 20MHz system clock, each layer of the ANN takes 6.5µs to produce an output. Reconfiguration between network layers for the XOR problem takes 17.6µs

when a 10MHz configuration loading clock is applied. This is faster than the general case, however, since for this specific problem some weights are the same in successive network layers. The initial full configuration, for the first network layer, takes 808µs. This only takes place when the network is first initialised and so has not been included in the performance calculations.

The three–layer ANN can produce results for the XOR problem at a rate of 24kHz, when reconfiguration overhead is taken into account. This corresponds to a network performance of 0.77M CPS (Connections Per Second). In comparison, the same network implemented using full static reconfiguration, again at the maximum configuration loading rate, would produce results at a rate of only 625Hz, or 20k CPS. Thus, for this network, partial reconfiguration gives a speedup of 38 over full reconfiguration, as well as a reduction in the amount of external configuration storage needed.

The reported network is considerably slower than "static" FPGA–based ANNs such as the GANGLION, which is reported to operate at 4.48G CPS [2]. It should be borne in mind, however, that this impressive performance is achieved at considerable expense, using an array of more than 30 large Xilinx devices in a fixed configuration. Where the technique of time–multiplexing offers benefit is as a cost–effective solution to ANN implementation which uses limited logic resources.

5. Conclusions

The authors have a particular interest in investigating potential application areas for dynamically reconfigurable FPGAs. Since the only FPGAs capable of dynamic reconfiguration to date are fine–grained devices, the technology mapping of reconfigurable designs onto fine–grained FPGAs is a valuable experiment. The ANN implementation reported here has provided useful information about mapping this type of circuit onto the particular resource set of fine–grained FPGA architectures such as the Atmel AT6005. Further, the use of reconfiguration, and in particular dynamic reconfiguration, has led to the implementation of a considerably larger ANN than would otherwise be possible on a single FPGA. Whilst the current system is limited to the time–multiplexing of whole network layers, the extension of the technique to allow individual layers to be partitioned for time–sharing would offer the potential of larger networks and is currently under consideration.

The work done in developing the pulse–stream ANN has highlighted certain restrictions in both the reconfiguration mechanism of the AT6005 and the CAD tools used to produce designs on it. When compared to the system speeds possible on the logic array, reconfiguration is currently very slow. If the advantages of device reconfiguration are to be exploited in real–time applications, it is important that this situation is improved. In addition, no vendor yet provides software for the simulation of reconfigurable designs, or floorplanning tools to optimise design layouts for fast reconfiguration.

Architectural changes to the AT6005 have been identified which would increase the density and performance of the pulse–stream ANN. These include the provision

of wired–OR capability, dedicated fast carry logic for counters, increased bussing resources and a faster reconfiguration mechanism.

These observations point to a possible future direction for the development of new FPGA architectures. Most design classes implemented on FPGAs would benefit in some way from having the logic and routing resources available on the device tailored to the particular application. Moreover, many designs which exploit reconfiguration contain a proportion of logic that is always static. Performance and integration levels could be further increased by providing dedicated resources to perform some of these static functions. This approach is a natural extension of the special purpose "hard macros" used in the Xilinx 4000 series devices for wide decoding functions. In the case of the pulse–stream ANN, dedicated pulse–stream and chopping clock generation could be combined with the architectural changes outlined previously to produce a Field Programmable Artificial Neural Network (FPANN). Such a device would lose the capability to implement large amounts of general–purpose logic, but would be particularly well suited to the efficient implementation of ANNs.

In general, it is conceivable that the optimisation of logic and routing resources to specific application classes could help to bridge the performance gap between FPGAs and ASICs, whilst retaining the benefits of reconfigurability.

Acknowledgements

The authors would like to thank the Nuffield Foundation, SERC and the Defence Research Agency for their support.

References

1. Atmel Corporation: Configurable Logic Design and Application Book, Atmel Corporation, San Jose, California, USA, 1994.
2. C.E. Cox, W.E. Blanz: GANGLION – A Fast Field-Programmable Gate Array Implementation of a Connectionist Classifier, IEEE Journal of Solid–State Circuits, Vol. 27, No. 3, March 1992, pp. 288-299
3. S.A. Guccione, M.J. Gonzalez: A Neural Network Implementation using Reconfigurable Architectures, In: W. Moore, W. Luk (eds.): More FPGAs, Abington, Oxford, UK, 1994, pp. 443–451
4. P. Lysaght, J. Dunlop: Dynamic Reconfiguration of FPGAs, In: W. Moore, W. Luk (eds.): More FPGAs, Abington, Oxford, UK, 1994, pp. 82–94
5. A.F. Murray: Pulse Arithmetic in VLSI Neural Networks, IEEE Micro, December 1989, pp. 64–74
6. J.E. Tomberg, K.K.K. Kaski: Pulse–Density Modulation Technique in VLSI Implementation of Neural Network Algorithms, IEEE Journal of Solid–State Circuits, Vol. 25, No. 5, October 1990, pp. 1277–1286
7. M. van Daalen, P. Jeavons, J. Shaw–Taylor: A Stochastic Neural Architecture that Exploits Dynamically Reconfigurable FPGAs, Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Cali-

fornia, USA, April 1993, pp. 202–211

8. Xilinx Inc.: The Programmable Logic Data Book, Xilinx Inc., San Jose, California, USA, 1993.