Hardware Spiking Neural Network and Remote FPGA Lab

Seamus Cawley B. Eng

A Thesis Submitted for
Doctor of Philosophy Degree

Work Carried Out at
Bio-Inspired Electronics and Reconfigurable Computing Research Group,
National University of Ireland Galway,
Ireland

And

Electrical & Electronic Engineering,
College of Engineering and Informatics,
National University of Ireland Galway,
Ireland

Supervisors
Dr. Fearghal Morgan
Professor Gearóid Ó Laighin

Submitted to the National University of Ireland Galway, September 2012
Acknowledgements

I wish to sincerely thank my supervisor, Dr. Fearghal Morgan. He has been a constant source of guidance and his tireless support and dedication have been invaluable throughout the course of this project.

I am deeply grateful to my parents, brother and sister for their constant encouragement and support, without which this would not have been possible.

A heartfelt thanks is extended to Dr. Brian McGinley for the knowledge, guidance and inspiration he provided and Sandeep Pande for the many stimulating discussions in the lab. My thanks to Dr. Liam McDaid, Dr. Jim Harkin and Snaider Carrillo of the Intelligent Systems Research Centre, in the University of Ulster Magee.

My thanks to everyone in the lab who helped make the last few years an enjoyable experience, especially Damien Dooley, Shane Tuohy, Anthony Winterlich, Pat Hurney, Dr. Martin O’Halloran, Garry Higgins, Martin Gallagher and Dr. Dallan Byrne.

I also thank the staff members of Electrical & Electronic Engineering, in NUI Galway, particularly Myles Meehan and Martin Burke who always provided a helping hand. My gratitude is extended to Professor Gearóid Ó Laighin, Dr. Martin Glavin and Dr. Edward Jones.

This research is supported by Science Foundation Ireland under Grant No. 07/SRC/I1169 and the Irish Research Council for Science, Engineering and Technology (IRCSET), International Centre for Graduate Education in Micro and Nano Engineering (ICGEE), NUI Galway Millennium Research Fund. Equipment was provided by Xilinx under the Xilinx University Programme.
Declaration of Authorship

I, Seamus Cawley, declare that this thesis titled, ‘Hardware Spiking Neural Network and Remote FPGA Lab’ and the work presented in it are my own. I confirm that:

■ This work was done wholly or mainly while in candidature for a research degree at this University.

■ Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.

■ Where I have consulted the published work of others, this is always clearly attributed.

■ Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

■ I have acknowledged all main sources of help.

■ Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:
Abstract

College of Engineering and Informatics
Electrical & Electronic Engineering

Doctor of Philosophy

Hardware Spiking Neural Network and Remote FPGA Lab

by Seamus Cawley

The automatic design of intelligent systems has been inspired by biology, specifically the operation of the human brain. Researchers hope to exploit and replicate the brain’s ability to adapt and self repair in order to develop robust and error tolerant embedded hardware computational devices. Spiking Neural Networks (SNNs) emulate neural behaviour observed in biology. This thesis describes the successful development of a Network-on-Chip based hardware SNN(EMBRACE-FPGA) and the supporting GA-based SNN training and it’s application implementation tools (SNNDevSys). Hardware SNNs can be configured for multiple applications through programming of neuron spike firing threshold potential, synaptic weights and the hardware SNN interconnection topology.

This thesis describes the hardware SNN architecture and prototype and its application to a range of benchmark control and classification problems. This work has contributed to ongoing EMBRACE-FPGA architecture development within the Bio-Inspired and Reconfigurable Computing research group to improve practical hardware SNN scalability.

Phase II of this thesis describes the development of the Remote FPGA Laboratory (RFL). In recent years there has been a growing interest in the development of web-based e-learning systems. The RFL is a web-based distance learning application which enables the teaching of digital systems design using real FPGA devices through a standard web browser. The RFL allows users to configure and interact with FPGA devices via the Internet as part of a combined training and evaluation framework. The system has been designed as an interactive learning tool, which aims to increase student interaction and understanding through a learn-by-doing approach. The system enables better understanding of the operation of digital systems through animation of internal signals, real-time timing diagrams and single-stepping of hardware circuits.
# Contents

Acknowledgements

Declaration of Authorship

Abstract

List of Figures

List of Tables

1 Introduction 1

1.1 Summary ......................................................... 1
1.2 Hardware SNN Development and Training .......................... 2
  1.2.1 Introduction .................................................. 2
  1.2.2 Spiking Neural Networks ..................................... 3
  1.2.3 Embedded Hardware SNN Architecture: EMBRACE-FPGA ... 4
  1.2.4 Impact of Neural Model Resolution on Hardware SNN Behaviour  7
  1.2.5 Optimising Hardware SNN Topology Memory Requirements .... 7
  1.2.6 EMBRACE-FPGA Benchmark Applications ...................... 8
    1.2.6.1 Introduction ............................................... 8
    1.2.6.2 XOR SNN Benchmark ...................................... 8
    1.2.6.3 Wisconsin Breast Cancer Dataset .......................... 9
    1.2.6.4 Inverted Pendulum Controller ............................ 9
    1.2.6.5 Robotics Obstacle Avoidance Controller ................. 10
  1.2.7 Hardware SNN Training and Configuration ..................... 11
    1.2.7.1 Introduction .............................................. 11
    1.2.7.2 Established ANN Training Algorithms .................... 11
    1.2.7.3 EMBRACE-FPGA Training System .......................... 13
    1.2.7.4 Evolving Neural Network Topologies ........................ 14
  1.2.8 Related Publications ......................................... 15
    1.2.8.1 Publications Included In This Thesis ..................... 15
    1.2.8.2 Other Publications ....................................... 15
1.3 Remote FPGA Laboratory (RFL) ................................... 17
  1.3.1 Introduction ................................................ 17
## Contents

1.3.2 RFL System Architecture ........................................ 18

1.3.2.1 FPGA Array .................................................. 19

1.3.2.2 RFL Server Side Architecture ................................ 19

1.3.2.3 RFL User Interface ......................................... 22

1.3.3 Related Publications ............................................ 22

1.5 Thesis Contributions And Novelty Claims ........................... 24

1.4 Thesis Contributions And Novelty Claims............................ 24

1.5 Structure of Thesis .............................................. 26

References ................................................................. 27

2 Hardware Spiking Neural Network Prototyping and Application 33

2.1 Introduction ....................................................... 34

2.2 Related Work ..................................................... 37

2.3 EMBRACE-FPGA Architecture and Operation ....................... 39

2.3.1 SNN Neural Tile Array and NoC Router Architecture and Operation 40

2.3.2 NoC Packet Latency Jitter ..................................... 42

2.3.3 EMBRACE-FPGA SNN Input-Output Interface .................. 43

2.3.4 EMBRACE-FPGA Processor-based Neuron/Synapse Model ....... 44

2.4 Intrinsic Hardware SNN Evolution, Training and Configuration Platform 46

2.4.1 GA Based Training ............................................. 46

2.4.2 Adaptability and Self Repair ................................... 47

2.4.3 On Chip Fitness Function ....................................... 48

2.5 EMBRACE-FPGA SNN Benchmark Applications .................... 48

2.5.1 EMBRACE-FPGA SNN XOR Implementation .................... 49

2.5.2 Robot Controller ............................................... 51

2.5.3 Inverted Pendulum Controller Application ..................... 53

2.5.4 Wisconsin Breast Cancer (WBC) Dataset Classifier ............ 55

2.5.5 Noise ............................................................. 57

2.6 Sources and Impact of Noise ...................................... 57

2.7 Conclusions .......................................................... 59

References ................................................................. 60

3 The Impact of Neural Model Resolution on Hardware Spiking Neural Network Behaviour 64

3.1 Introduction ....................................................... 65

3.2 Related Work ..................................................... 66

3.3 EMBRACE-FPGA Architecture .................................. 67

3.4 GA-Based Hardware SNN Training And Configuration Platform 70

3.5 Evolved Hardware SNN Robotics Controller ....................... 70

3.6 Impact Of Neuron Model Resolution ................................ 73

3.6.1 Low Resolution Neuron Model Noise ........................... 73

3.6.2 High Resolution Neuron Model .................................. 74

3.7 Conclusions .......................................................... 75

References ................................................................. 76

4 Memory Efficient Storage of Reconfigurable Topology Information in Network-on-Chip Based Spiking Neural Networks 79

4.1 Introduction ....................................................... 80

4.2 Large Scale Hardware SNN Architectures .......................... 82
4.3 SDS and MS Model Description and Memory Dependencies ........................................ 83
   4.3.1 Multi Synapse (MS) Architecture Topology Memory Dependencies .................. 84
   4.3.2 Shared Dynamic Synapse (SDS) Architecture Topology Memory Dependencies .... 86
4.4 Dynamically Reconfigurable Synapse ........................................................................ 87
4.5 Conclusions and Future Work .................................................................................. 88
References ..................................................................................................................... 89

5 Evolving SNN Topologies for UWB based Breast Cancer Detection .............................. 91
5.1 Introduction .............................................................................................................. 92
5.2 Spiking Neural Networks and Evolving Topologies .................................................. 93
   5.2.1 Spiking Neural Network .................................................................................. 93
   5.2.2 Preprocessing and Fitness Function ................................................................ 94
5.3 Breast and Tumour Modeling .................................................................................. 96
5.4 Results ................................................................................................................... 97
   5.4.1 Effects of Dielectric Heterogeneity ................................................................ 97
5.5 Conclusions ........................................................................................................... 98
5.6 Acknowledgments ................................................................................................... 98
References ..................................................................................................................... 99

6 Enhancing Learning of Digital Systems using a Remote FPGA Lab ............................... 102
6.1 Introduction ............................................................................................................. 103
6.2 Related Work ......................................................................................................... 105
6.3 Remote FPGA Lab Architecture ............................................................................. 107
6.4 Remote FPGA Web Application Server Architecture ............................................. 107
6.5 Demonstrators ....................................................................................................... 110
6.6 Conclusions and Future Work .............................................................................. 115
References ..................................................................................................................... 117

7 Remote FPGA Lab With Interactive Control and Visualisation Interface ...................... 119
7.1 Introduction ............................................................................................................. 120
7.2 Related Work ......................................................................................................... 122
7.3 FPGA Architecture ............................................................................................... 123
7.4 Remote FPGA Web Application ............................................................................. 123
7.5 Remote FPGA Design Demonstrator .................................................................... 126
7.6 Case Study: Application of the Remote FPGA in Undergraduate Teaching ............. 126
7.7 Conclusions and Future Work .............................................................................. 128
References ..................................................................................................................... 129

8 Remote FPGA Lab for Enhancing Learning of Digital Systems ................................... 130
8.1 Introduction ............................................................................................................. 131
8.2 Related Work ......................................................................................................... 132
8.3 Remote FPGA Lab Functionality ......................................................................... 134
8.4 Remote FPGA Lab Architecture ........................................................................... 136
8.5 Automatic Integration Of User Design Into The RFL .......................................... 139
8.6 Improved Learning And Achievement: RFL Usage And Student Survey ............. 142
8.7 Conclusions And Future Work ............................................................................. 143
9 Digital Systems Pedagogy using the Remote FPGA Lab

9.1 Introduction ................................................. 148
9.2 Remote FPGA Labs: Background ............................ 150
9.3 RFL Teaching Strategy ...................................... 152
  9.3.1 Introduction ............................................. 152
  9.3.2 Lock Controller: AND-based Illustrative RFL example ... 152
  9.3.3 Counter Illustrative RFL example ...................... 154
9.4 Interactive Timing Diagram Creation ....................... 158
9.5 Automatic Assessment using the RFL ...................... 158
9.6 Conclusion and Future Work ............................... 160
References ...................................................... 162

10 Conclusions and Future Work

10.1 Introduction .............................................. 163
10.2 Network-on-Chip Based Hardware SNN .................... 163
10.3 Web Based FPGA Educational Laboratory ................ 164
10.4 Future Work ............................................. 165
References ...................................................... 167
List of Figures

1.1 Two-Layer, Fully-Connected Feed-Forward SNN Topology and Spike Trains 4
1.2 Change in neuron potential on receipt of a series of spikes . . . . . . . . 4
1.3 EMBRACE-FPGA hardware SNN neural tile block diagram. . . . . . . 6
1.4 SNNDevSys Block Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
1.5 Example of (a) a linearly separable and (b) non-linearly separable task . 8
1.6 WBCD classifier SNN structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
1.7 SNN topology of inverted pendulum controller . . . . . . . . . . . . . . . . 10
1.8 MobileSim robotics simulator console including Activmedia Pioneer robot 11
1.9 RFL System Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
1.10 RFL FPGA Array . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
1.11 RFL database schema . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.1 EMBRACE-FPGA NoC-based SNN hardware neural tile array . . . . . 40
2.2 SNN neural tile block diagram. $N$ is the maximum number of synapses per neuron . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
2.3 EMBRACE NoC packet routing example (neuron connectivity inset) . . . 43
2.4 EMBRACE-FPGA Evolvable Hardware SNN block diagram . . . . . . . 44
2.5 Simulated EMBRACE-FPGA neuron membrane potential behaviour result- ing from an incoming spike train on two synapses (0 and 1). Waveform has been obtained from simulation of the EMBRACE-FPGA RTL design 45
2.6 Variation in neuron membrane potential in one neuron . . . . . . . . . . 45
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>SNN neural tiles interconnection for implementation of a two-layer feed-forward SNN (used for evolution of the XOR benchmark function)</td>
</tr>
<tr>
<td>2.8</td>
<td>Average and best fitness of the evolved EMBRACE-FPGA hardware SNN XOR function. Target fitness is 16</td>
</tr>
<tr>
<td>2.9</td>
<td>SNN topology used by EMBRACE-FPGA for robot controller</td>
</tr>
<tr>
<td>2.10</td>
<td>Best And Average Fitness Of The Evolved Robotics Controller SNN</td>
</tr>
<tr>
<td>2.11</td>
<td>EMBRACE-FPGA and SNNDevSys configuration platform used for evolution of inverted pendulum controller function</td>
</tr>
<tr>
<td>2.12</td>
<td>Average and best fitness of the evolved inverted pendulum EMBRACE-FPGA hardware SNN controller</td>
</tr>
<tr>
<td>2.13</td>
<td>WBCD classifier hardware SNN topology (nine hidden layer neurons and two output neurons)</td>
</tr>
<tr>
<td>2.14</td>
<td>Average and Best Fitness obtained using the EMBRACE-FPGA WBCD Classifier Hardware SNN</td>
</tr>
<tr>
<td>2.15</td>
<td>EMBRACE-FPGA neuron membrane potential trace</td>
</tr>
<tr>
<td>2.16</td>
<td>EMBRACE-FPGA input spike train, neuron membrane potential variation and neuron output due to NoC latency variation</td>
</tr>
<tr>
<td>3.1</td>
<td>EMBRACE-FPGA and SNNDevSys (a GA-based hardware SNN training and configuration platform)</td>
</tr>
<tr>
<td>3.2</td>
<td>EMBRACE-FPGA SNN Neural Tile (*:N,S,E,W)</td>
</tr>
<tr>
<td>3.3</td>
<td>SNN topology used EMBRACE-FPGA for robot controller</td>
</tr>
<tr>
<td>3.4</td>
<td>Best and average fitness of the evolved robotics controller SNN</td>
</tr>
<tr>
<td>3.5</td>
<td>EMBRACE-FPGA neuron potential trace</td>
</tr>
<tr>
<td>3.6</td>
<td>EMBRACE-FPGA input spike train and neuron potential variation due to NoC latency variation</td>
</tr>
<tr>
<td>3.7</td>
<td>Simulated analogue neuron potential trace</td>
</tr>
<tr>
<td>3.8</td>
<td>Simulated analogue neuron input spike train and neuron potential variation due to NoC latency jitter</td>
</tr>
</tbody>
</table>
4.1 NoC-based SNN Memory Layout ............................................. 82
4.2 Memory Usage ................................................................. 84
5.1 Example Fixed-Topology Spiking Neural Network ...................... 94
6.1 RemoteFPGA lab array of FPGA development systems and webcams . 104
6.2 displayCtrlr top level component, and the switch/LED/7-segment display connectivity on the Digilent Nexys2 board .................. 104
6.3 RemoteFPGA lab visualisation of a 6-ball (red-pnk) snooker scoring system (snkrScoreSys) with overlayed interactive switch inputs (spring loaded buttons) and outputs (LEDs and seven segment display) ............ 105
6.4 RemoteFPGA top level block diagram ....................................... 107
6.5 RemoteFPGA lab system level architecture ................................ 109
6.6 RemoteFPGA Lab streamed video of an active Nexys2 FPGA board and default interactive switch/display mirror interface ................ 109
6.7 displayCtrlr Demonstrator ...................................................... 111
6.8 displayCtrlr Data Flow Diagram (DFD) illustrating and describing the main elements within the design, and internal signals seg7Sel(1:0) and segData(3:0) ....................................................... 112
6.9 displayCtrlr Register Transfer Level schematic view with overlayed RemoteFPGA lab inputs (toggle switches), and outputs (LEDs and 7 segment displays) .................................................. 113
6.10 snkrScoreSys Demonstrator ................................................... 114
7.1 Two FPGA development systems and webcams, included within the Remote FPGA system ............................................. 120
7.2 Remote FPGA Mirrored Nexys2 User Interface .......................... 121
7.3 Top level block FPGA block diagram ....................................... 121
7.4 Top level block diagram FPGA diagram ................................... 124
7.5 Console interface demonstration of a Finite State Machine .......... 127
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6</td>
<td>Participant activity. Number of logins, and number of participant-specific FPGA configurations as course progresses (note: a demonstration FPGA design is automatically configured on login)</td>
<td>128</td>
</tr>
<tr>
<td>8.1</td>
<td>RemoteFPGA lab array of FPGA development systems and webcams</td>
<td>131</td>
</tr>
<tr>
<td>8.2</td>
<td>RFL architecture and User Design Wrapper</td>
<td>135</td>
</tr>
<tr>
<td>8.3</td>
<td>Control and visualisation interface for the 2-to-1 multiplexer design (M2_1)</td>
<td>136</td>
</tr>
<tr>
<td>8.4</td>
<td>RFL Control and visualisation interface (web-based user console) for the two digit cascadable BCD counter</td>
<td>137</td>
</tr>
<tr>
<td>8.5</td>
<td>Remote FPGA Lab System Level Architecture</td>
<td>138</td>
</tr>
<tr>
<td>8.6</td>
<td>RFLGen workflow</td>
<td>140</td>
</tr>
<tr>
<td>8.7</td>
<td>RFL Survey Results</td>
<td>142</td>
</tr>
<tr>
<td>9.1</td>
<td>Remote FPGA lab array of FPGA development systems and webcams</td>
<td>149</td>
</tr>
<tr>
<td>9.2</td>
<td>Lock controller demonstrator</td>
<td>155</td>
</tr>
<tr>
<td>9.3</td>
<td>CB2CED Demonstrator</td>
<td>157</td>
</tr>
<tr>
<td>9.4</td>
<td>CB2CED Timing Diagrams</td>
<td>159</td>
</tr>
<tr>
<td>9.5</td>
<td>User Assessment GUI</td>
<td>160</td>
</tr>
<tr>
<td>9.6</td>
<td>RFL Survey Results</td>
<td>160</td>
</tr>
</tbody>
</table>
# List of Tables

2.1 Comparison of different SNN architectures ........................................ 38
2.2 Comparison of hardware SNN technologies ........................................ 38
2.3 Summary of FPGA resource usage for an EMBRACE-FPGA Neural Tile .... 40
2.4 SNNDevSys GA Parameters and SNN topology for each benchmark applic-  
ation ......................................................................................................... 49
2.5 SNNDevSys GA Parameters for the Robotics Controller ....................... 52
3.1 GA parameters used by SNNDevSys ................................................... 72
4.1 Example Memory Requirements for EMBRACE-FPGA & SDS ............... 87
5.1 Comparison of Fixed and Evolved-Topology SNN classifiers ................. 97
5.2 Effects of dielectric heterogeneity on performance of SNN classifiers. Het-  
ero I refers to models containing one piece of fibroglanular tissue, while  
Hetero II refers to models with two pieces of fibroglanular tissue. .......... 98
9.1 Lock Controller Data Dictionary ......................................................... 153
Chapter 1

Introduction

1.1 Summary

This thesis details two research phases. Phase I concerns the development and application of a scalable, Network-on-Chip (NoC) based hardware Spiking Neural Network (SNN) (EMBRACE-FPGA), and supporting training and configuration tools (SNNDevSys). Phase II details the development of the Remote FPGA Laboratory (RFL), a web-based digital systems educational tool which enables real-time remote access to an array of FPGA devices and integrated course material. This chapter lists the novel contributions of this research and related published work.

Phase I

The automatic design of intelligent systems has been inspired by biology, specifically the operation of the human brain. Researchers hope to exploit and replicate the brain’s ability to adapt and self repair in order to develop robust and error tolerant embedded hardware computational devices. SNNs emulate the behaviour of biological neural networks. This thesis describes the successful development of a Network-on-Chip (NoC) based hardware SNN (EMBRACE-FPGA) and the supporting SNN training and application implementation tools (SNNDevSys).

Biological neurons exhibit dense interconnectivity, in the order of 2000 synaptic connections per neuron. The high level of inter-neuron connectivity in a hardware SNN results in integrated circuit interconnect area which grows non linearly with increasing SNN size [1]. This is termed the scalability problem. This thesis describes a scalable hardware SNN architecture and prototype (EMBRACE-FPGA) and application to a range of control and classification problems.
Chapter 1. Introduction

Hardware SNNs can be configured for multiple applications through programming of neuron spike firing threshold potential, synaptic weights and the hardware SNN inter-connection topology. A hardware SNN is created by connecting electronic models of neurons and synapses into a large neuron array. SNNDevSys is a GA based training and configuration platform previously developed as an evolutionary platform for hardware SNN controllers [2]. Genetic Algorithms (GAs) [3] are used to evolve solutions to problems using mechanisms observed in nature, i.e. selection, mutation and crossover of intermediate solutions. This thesis details the further development of SNNDevSys to support the EMBRACE-FPGA configuration NoC packet structure and hardware testing and debug functionality.

This work has led to ongoing EMBRACE architecture development within the Bio-Inspired and Reconfigurable Computing (BIRC) research group to improve scalability and practical hardware SNN array size. The development of EMBRACE-FPGA and extensions to SNNDevSys to support EMBRACE-FPGA training, testing and debug is considered Phase I of this thesis.

Phase II

Phase II of this thesis describes the development of the Remote FPGA Laboratory (RFL). In recent years there has been a growing interest in the development of web-based e-learning systems. The RFL is a web-based distance learning application which enables the teaching of digital systems design using real FPGA devices through a standard web browser. The RFL allows users to configure and interact with FPGA devices via the Internet as part of a combined training and evaluation framework. The system has been designed as an interactive learning tool, which aims to increase student interaction and understanding through a learn-by-doing approach. The system enables better understanding of the operation of digital systems through animation of internal signals, real-time timing diagrams and single-stepping of hardware circuits.

1.2 Hardware SNN Development and Training

1.2.1 Introduction

This section introduces Spiking Neural Networks (SNNs), the motivation for developing embedded hardware SNNs, the EMBRACE-FPGA NoC-based hardware SNN architecture and the training and configuration tools utilised by EMBRACE-FPGA. The
implementation of EMBRACE-FPGA and its application to a range of control and classification problems is summarised. This section also lists the extensions to SNNDevSys (a previously developed GA-based intrinsic hardware SNN training and configuration tool [2]), which include support for the EMBRACE-FPGA NoC packet-based configuration, packet-based testing and debug of EMBRACE-FPGA and the evolution of SNN topology. The digital hardware neuron model, NoC latency jitter issues and techniques to reduce SNN topology memory are also introduced.

1.2.2 Spiking Neural Networks

The basic processing units in organic central nervous systems are neurons. The large degree of inter-neuron connectivity and parallel processing enables the brain to perform complex learning and fuzzy-decision making capabilities [4], and to exhibit inherent adaptability and redundancy. Artificial Neural Networks (ANNs) attempt to replicate the behaviour of biological neurons by implementing neuron models interconnected via synaptic connections.

The current understanding of biological neurons is that they communicate through pulses and employ the relative timing of the pulses to transmit information and perform computations. SNNs [5], which are the third generation of ANNs, emulate real biological neurons of the brain more closely than traditional ANN models. SNNs are complex, non-linear computation systems, which are computationally more powerful than traditional ANNs [5]. Spiking neurons communicate by transmitting short transient pulses or spikes between neurons, via weighted synaptic connections. Figure 1.1 illustrates a two-layer, fully-connected, feed-forward SNN topology and a typical spike train between two neurons. Spiking neurons maintain an internal membrane potential which is modified on receipt of a spike via a connected synapse. Synapses can have an inhibitory or excitatory effect on neuron membrane potential. Over time a neuron’s membrane potential decays towards its resting potential. A spiking neuron fires when the sum of the weighted input spikes, or membrane potential, exceeds the neuron’s firing threshold. On generation of a spike the membrane potential will be reset to the neuron’s refractory potential from which it will slowly return to the resting potential. Outputs from neurons consist of a series of spike pulses which may, in turn, be interpreted by other neurons or be employed as outputs of the SNN.

Figure 1.2 illustrates the change in neuron potential on receipt of a series of spikes. Figure 1.2(a) illustrates a typical SNN spike train between two neurons. Spikes emitted by Neuron A arrive at Neuron B via an excitatory synapse, resulting in an increase in the Neuron B membrane potential (Figure 1.2(b)). The neuron membrane potential
1.2.3 Embedded Hardware SNN Architecture: EMBRACE-FPGA

This section describes the motivations for developing an embedded hardware SNN architecture. The EMBRACE-FPGA prototype NoC-based hardware SNN architecture...
Chapter 1. Introduction

and its components are detailed.

Researchers aim to implement reconfigurable and highly interconnected arrays of neural network elements in hardware to produce powerful signal processing units [6–14]. SNNs have a number of features which make them suitable for implementation in digital hardware. By representing spikes as either ‘high’ or ‘low’ values, i.e. as single bits, hardware and memory requirements are reduced dramatically compared to those of traditional multi-layer perceptron neural networks [15, 16]. However, the complex non-linear processing required by the decay of neuron membrane potential, even in simplified hardware models, requires a relatively complex digital neuron model.

The major limiting factor in scalable hardware SNN implementation is the area required by directly wired inter-neuron connectivity. This is termed the scalability problem. Often the number of neurons that can be realised in hardware is limited by high fan in/out requirements [17]. Direct neuron-to-neuron interconnection exhibits switching requirements that grow non-linearly with the network mesh size [17], e.g. a 2-layer, fully connected, feed-forward network with \( m \) neurons exhibits an interconnect density of \( m^2 \). Various methods have been investigated to address the hardware SNN interconnect scalability issue including wafer-scale integration [18, 19] and Network-On-Chip (NoC) based [12, 20–24] inter-neuron communication.

The EMBRACE project, involving collaboration with the Bio-Inspired and Reconfigurable Computing research group, NUI Galway and the Intelligent Systems Research Centre, University of Ulster Magee, aims to develop a large scale, embedded hardware based SNN through the use of Network-on-Chip (NoC) concepts. Each neuron is connected to a NoC router. Routers are connected together in a large scale NoC array. Additionally, due to the inefficiency of performing neural processing in hardware, the EMBRACE project is developing a low area, low power analogue, CMOS compatible neuron model component [11], to perform the complex, non-linear internal neuron membrane potential decay operations in analogue hardware, which is ideally suited to this kind of non-linear decay. This analogue EMBRACE neuron offers very small area and power requirements.

This research has focused on the design, development and testing of the NoC communication infrastructure using a digital neuron model and the implementation and application of a hardware SNN FPGA prototype (EMBRACE-FPGA). Each neuron is implemented using an embedded, soft-core microprocessor. Figure 1.3 illustrates the combination of embedded processor-based neuron, NoC router and associated network topology (neuron connectivity information), termed a Neural Tile (NT). Figure 1.4 illustrates the EMBRACE-FPGA NTs, which are arranged in a grid pattern, with each NT connected to the North, South, East and West. EMBRACE-FPGA utilises an array
Chapter 1. Introduction

Figure 1.3: EMBRACE-FPGA hardware SNN neural tile block diagram.

Figure 1.4: SNNDevSys Block Diagram
Chapter 1. Introduction

of Spike Generators (SGs) to convert real-world SNN input values to input neuron spike frequencies. Spike Counters (SCs) are used to convert output spike train frequencies to SNN output values by counting the number of spikes received over a predefined time period. Figure 1.3 illustrates the components of the EMBRACE-FPGA NT where \( N \) is the maximum number of synapses per neuron.

1.2.4 Impact of Neural Model Resolution on Hardware SNN Behaviour

This section introduces the impact of NoC latency variations (jitter) on SNN behaviour. Variations in jitter, amplified by the low resolution (8-bit) digital neural model processing, results in distortion of SNN inter-spike intervals (ISI). As SNNs perform computation based on the timing of inter-neuron spikes this jitter distorts ideal spike timing information and hence has an adverse effect on the behaviour of the hardware SNN. Due to the fact that the picoBlaze microprocessor chosen to model the EMBRACE-FPGA neural cell performs calculations using 8-bit values, the neuron membrane potential decay is stepwise in nature instead of an ideal, infinite resolution, smooth decay. This stepwise decay can result in an amplification of the effect of spike timing distortion caused by NoC jitter.

The thesis provides an analysis of the issue of NoC noise, including a comparison between actual neural membrane potential retrieved from the picoBlaze (via Neural Tile read-back functionality) and the simulated EMBRACE analogue neural cell membrane potential. This analysis concludes that NoC latency jitter does have an effect on hardware SNN behaviour with the effect amplified by low resolution digital neuron modelling. This effect, however, has minimal impact on practical EMBRACE-FPGA performance. The robustness of the system, in conjunction with GA-based training, results in no adverse effects on application performance. In some scenarios this noise introduced into the network can guide the GA towards more robust and error tolerant solutions.

1.2.5 Optimising Hardware SNN Topology Memory Requirements

The area required by EMBRACE-FPGA to store SNN neuron configuration information (synaptic weights and neuron firing thresholds) and network topology information is a large proportion of the total area consumed by each NT. As EMBRACE-FPGA is fundamentally an array of NTs, any improvement in NT area can have a significant effect on the area required by the hardware SNN. Many strategies have been explored to reduce the area of the EMBRACE-FPGA NT including clustered SNNs [25], NoC router optimisations [26] and alternative NoC topologies [27]. This thesis proposes and describes optimised synaptic weight and neuron synapse address storage technique. The
Figure 1.5: Example of (a) a linearly separable and (b) non-linearly separable task

approach replaces the array of synapses with a Single Dynamic Synapse (SDS) which receives its weight value from the spike packet. Each SDS is dynamically configured with this weight before the incoming spike is applied to the neuron. This approach results in a 35% saving in EMBRACE-FPGA topology memory for a network with 1024 neurons and 256 synaptic connections per neuron.

1.2.6 EMBRACE-FPGA Benchmark Applications

1.2.6.1 Introduction

This section summarises the evaluation of EMBRACE-FPGA through a range of control and classification problems. Classification applications used include the XOR problem [28] and the Wisconsin Breast Cancer Dataset classifier [29]. Control applications include an inverted-pendulum controller and a robotics obstacle avoidance problem.

1.2.6.2 XOR SNN Benchmark

Much interest in the area of ANNs was stimulated in 1958 by Rosenblatt’s proposal of the Perceptron neuron model [30]. In 1969 Minsky and Papert illustrated the computational limitations of perceptrons [31], demonstrating that a perceptron may only solve tasks which are linearly separable. The XOR problem is a task which is not linearly separable, and hence is a subset of more complex problems. As such, implementing the XOR function is a classic neural network benchmark application. Figure 1.5 illustrates examples of linearly (a) separable and linearly (b) inseparable tasks.

Using Multiple Layers of Perceptrons (MLPs) it is possible to distinguish data that is not linearly separable and hence solve the XOR problem. MLP based ANNs are considered the second generation of ANNs. SNNs have their basis in the method of inter-neuron
communication used by biological neurons and have been shown to be computationally more powerful than MLPs [5]. This thesis presents the implementation of the XOR classifier to verify the basic operation of the EMBRACE-FPGA SNN and NoC based inter-neuron communication.

### 1.2.6.3 Wisconsin Breast Cancer Dataset

The Wisconsin Breast Cancer Dataset (WBCD) [29] is a benchmark application which is part of Proben [32], a collection of benchmark problems for neural network learning. This data-set consists of 699 test vectors, which have 9 inputs and 2 outputs each, of which 65.5% are classified as benign. The EMBRACE-FPGA SNN structure chosen to implement a solution to this task is illustrated in Figure 1.6 and consists of 9 SGs, 11 NTs and 2 SCs. This thesis presents an EMBRACE-FPGA WBCD classifier with an accuracy of 96.8%. This exceeds the accuracy of reported hardware SNNs [33].

### 1.2.6.4 Inverted Pendulum Controller

The inverted pendulum problem is a standard benchmark for control methodologies due to its instability and highly non-linear behaviour. The experiment described in this thesis enables a hardware SNN to control a cart (which can move along the x-axis) and a hinged pole connected to the cart. The objective is to maintain the pole standing for as long as possible, within 30 degrees of the vertical position, by controlling the horizontal
movement of the cart. The pendulum position and velocity information (horizontal $x$ position, $x$ velocity, pole angle and velocity) are provided as hardware SNN controller inputs. The SNN controller outputs horizontal right and left cart motor forces. Figure 1.7 illustrates a minimal SNN topology which has been used to implement the hardware SNN inverted pendulum controller. EMBRACE-FPGA implements a controller which successfully balances the pole upright for 100 seconds (the length of the simulation).

1.2.6.5 Robotics Obstacle Avoidance Controller

The EMBRACE-FPGA hardware SNN is used to successfully evolve a robotics obstacle avoidance controller. The application uses the MobileSim [34] simulation framework to simulate an Activmedia Pioneer robot [35] in a closed course. The robot is equipped with eight forward-facing sonar sensors, four of which are used as inputs to the hardware SNN. The SNN is configured with two outputs, which control the two robot wheel motors. Figure 1.8 illustrates the MobileSim robot in a virtual environment representing the location and behaviour of the SNN controlled robot and associated sonar sensors used as inputs to EMBRACE-FPGA. The goal of this control task is to maximise both the time taken and distance travelled by the robot during the simulation time (or until the robot crashes). The SNN topology illustrated in Figure 1.7 has also been used to implement the hardware SNN robotics controller. The EMBRACE-FPGA controller evolved successfully navigates the simulated environment for 100 seconds.
1.2.7 Hardware SNN Training and Configuration

1.2.7.1 Introduction

The complex behaviour of spiking neurons and the fact that computation is performed in the time domain means that existing ANN training algorithms may not be used to train SNNs. This section describes established SNN training algorithms. This section also introduces the GA-based approach to training taken by the EMBRACE-FPGA project. SNNDevSys is a GA-based training and configuration platform previously developed as an evolutionary platform for hardware SNN controllers [2]. This section details extensions to SNNDevSys to support the EMBRACE-FPGA NoC packet configuration structure and the evolution of SNN topology.

SNNs have been shown to be computationally more powerful than traditional ANN models [5]. However, in order to perform a useful function, neurons must be connected together in a manner which maps the network inputs to the desired output. While the structure or topology of a neural network is an important consideration when designing a SNN to solve a particular task, much research has focused on modifying synaptic weights and neuron firing thresholds. Generally, a neural network topology which is fully connected, feed forward and has one hidden layer is used as the typical SNN structure. A supervised learning algorithm modifies weights and thresholds in order to map the SNN inputs to the desired outputs.

1.2.7.2 Established ANN Training Algorithms

This section describes a number of existing algorithms used to train ANNs. ANN training involves mapping network inputs to the desired network outputs. In order to create
this mapping neural network topology and synaptic weights are modified using, in general, a global optimisation or search algorithm.

ANN training is generally performed using supervised learning, i.e. where the training algorithm applies previously generated and annotated training input data and compares the networks output to the desired output value (which has been determined in advance), to guide modification of SNN parameters.

Backpropagation is a supervised learning algorithm, widely used to train ANNs [36]. The algorithm compares the network output values with the correct answer to compute the value of some predefined error-function. Synaptic weights are modified to reduce the error, typically by a small amount, using a gradient descent algorithm. Modifying the weights requires the calculation of the derivative of the error function with respect to the network weights. Hence, the activation function of the neuron model used must be differentiable. Traditional backpropagation algorithms typically require a large number of iterations in order to minimise the error function. Additionally, the possibility of the algorithm converging to a local minimum of the error function requires consideration in the design of the algorithm. Backpropagation, however, is not suitable for training of SNNs due to the temporal nature of the computation performed by spiking neurons, and the fact that standard spiking neuron activation functions are not easily differentiable. SpikeProp [37] utilises the concept of backpropagation and applies this to spiking neurons. This algorithm requires that neurons fire only once. SpikeProp also requires that additional logic be included in each neuron to determine the timing of input spikes and neuron output spikes. This impacts the area of the EMBRACE neural cell.

Spike-Timing-Dependent Plasticity (STDP) is a SNN training algorithm based on biological processes which adjust the strength of connections between neurons (synaptic weights). STDP aims to align the timing of input spikes to a neuron with the output spikes generated by that neuron; i.e. if an input spike to a neuron occurs on average, before that neuron fires, the strength of the connection (synaptic weight) is increased. If the input spike tends to occur after the neuron fires, the strength of the connection is reduced. STDP is thought to be the process by which the brain learns. As with SpikeProp, STDP requires logic within each neuron to determine the timing of input spikes and neuron output spikes. This additional logic impacts the area of the EMBRACE neural cell.


1.2.7.3 EMBRACE-FPGA Training System

GAs [3] are a type of Evolutionary Algorithm based on the same mechanisms of adaptation observed in nature, including natural selection and genetic diversity. Guided by the “survival of the fittest” principle, GAs are effective global search algorithms that require points in the solution space to be encoded into a linear data string (genome). Apart from implementing the encoding mechanism and fitness function, GAs also perform the genetic operations.

A combined GA-based training and configuration tool (SNNDevSys), illustrated in Figure 1.4 has been developed for use with hardware SNN controllers. The components and operation of SNNDevSys are described in detail in this thesis. A combination of crossover and mutation operators are used to evolve a solution to the benchmark problems implemented on EMBRACE-FPGA. GAs maintain a “population” of “individuals”, each of which contains a number of “genes”, each representing the configuration values for a single neuron. The GA randomly generates the initial SNN population parameters for each NT (synaptic weights and neuron firing threshold), encodes this information as an integer-valued genome, and decodes this into corresponding binary-valued hardware SNN configuration values and NoC configuration packet sequence. The evolution process involves selecting a hardware SNN configuration (SNN genes comprising neuron synaptic weights and neuron firing thresholds), configuring the hardware platform with the selected SNN individual parameters, applying the application training dataset or control inputs and evaluating the individual’s fitness. Once each individual in a population has been evaluated, the GA operations of selection, crossover and mutation are applied to create a new generation of SNN solutions. This evolutionary process continues until high fitness convergence is achieved.

This thesis has extended SNNDevSys [2] to support EMBRACE-FPGA specific features, distributed evaluation of GA individuals and evolution of SNN topology. For a specific application SNNDevSys loads a module defining the task, i.e. its fitness function, GA parameters and required network topology. On creation of a new population of individuals by the GA, SNNDevSys packages the genes of each individual into a series of NoC configuration packets. Configuration packets are transmitted to EMBRACE-FPGA and routed to the appropriate NT where synaptic weights and neuron firing thresholds are configured. For each input vector of the training data, SNNDevSys configures EMBRACE-FPGA SGs (Figure 1.4) with appropriate values and determines the hardware SNN response using on-chip SCs configured as the neural network outputs. These outputs are used by the fitness function to assign a fitness value to the individual. Extensions to the existing SNNDevSys [2], implemented in this thesis include:
• support for the EMBRACE-FPGA NoC configuration packet format
• support for NT and neuron membrane potential readback
• support for injection of raw packets into the NoC
• evolution of SNN topology

Intrinsic training refers to the process of implementing and evaluating a neural network configuration in the targeted hardware device. This differs from extrinsic training which evaluates a neural network configuration by simulating the circuit in software until training is complete, followed by implementation in hardware. Intrinsic training offers improved accuracy, faster training (assuming neural network execution is faster in hardware than in software) and requires no reliance on the accuracy of circuit simulation tools. Extrinsic training allows neural networks to be trained without the use of hardware devices, potentially not easily accessible, already in use, difficult to communicate with, or non-reconfigurable. SNNDevSys evolves SNN solutions intrinsically on EMBRACE-FPGA since the hardware architecture supports on-line reconfiguration and performs individual evaluation faster than software simulations.

SNNDevSys is implemented as a workstation-based system and communicates with EMBRACE-FPGA through a register based communication protocol. The platform has been extended to support the transmission of raw NoC packets to EMBRACE-FPGA, which facilitates testing and debug of the hardware system. SNNDevSys separates transmission of individual configuration to EMBRACE-FPGA and fitness function evaluation from the rest of the system, enabling multiple individuals to be evaluated in parallel on multiple FPGA devices. The core of the system and the “evaluation” module communicate using network socket connections (by serialising individual configuration information) allowing use of FPGA devices connected to different workstations.

1.2.7.4 Evolving Neural Network Topologies

There has been growing interest in exploiting the adaptability provided by GAs to modify the interconnect structure of hardware SNNs in order to create a topology which is both more efficient and more suitable for the task at hand. A common issue with this concept is designing an appropriate encoding mechanism for the SNN topology such that it may be mutated and combined with structurally different networks in a computationally efficient manner. Additionally, network structures evolved by a GA have a tendency to grow as the GA progresses. This topology growth results in a more complex search space, partly negating the advantage of evolving a task specific network.
NeuroEvolution through Augmenting Topologies [38] (NEAT) is an algorithm which addresses these issues through the use of historical markers in the SNN topology genes. The NEAT algorithm has been incorporated into SNNDevSys and its operation verified through the successful evolution of an Ultra Wideband Radar (UWB) based breast cancer classifier using a software-based SNN. This classifier increases accuracy by 10% when compared with a fixed topology SNN evolved by SNNDevSys.

1.2.8 Related Publications

1.2.8.1 Publications Included In This Thesis

- **Chapter 2:** S. Cawley, F. Morgan, B. McGinley, S. Pande, L. McDaid, S. Carrillo, and J. Harkin. “Hardware Spiking Neural Network Prototyping and Application”, *Genetic Programming and Evolvable Machines, Special Issue on Evolvable Hardware Challenges*, 2011 [39]

- **Chapter 3:** S. Cawley, F. Morgan, B. McGinley, S. Pande, L. McDaid and J. Harkin. “The impact of neural model resolution on hardware spiking neural network behaviour”, *Irish Signals and Systems Conference (ISSC)*, 2010 [40]

- **Chapter 4:** “Memory Efficient Storage of Reconfigurable Topology Information in Network-on-Chip Based Spiking Neural Networks”, *in preparation*

- **Chapter 5:** O’Halloran, M., Cawley, S., McGinley, B., Conceicao, R.C., Morgan, F., Jones, E. and Glavin, M. “Evolving Spiking Neural Network Topologies for Breast Cancer Classification in a Dielectrically Heterogeneous Breast”, *Progress In Electromagnetics Research Letters*, 2011 [41]

1.2.8.2 Other Publications

- Fearghal Morgan, Seamus Cawley, Brian Mc Ginley, Sandeep Pande, Liam McDaid, Brendan Glackin, John Maher, Jim Harkin “Exploring the Evolution of NoC-Based Spiking Neural Networks on FPGAs”, *Field-Programmable Technology (FPT)*, 2009 [42]

- Fearghal Morgan, Seamus Cawley, Jim Harkin, Brian McGinley, Liam Mc Daid, Sandeep Pande “An Evolvable NoC-Based Spiking Neural Network Architecture”, *Irish Signals and Systems Conference (ISSC)*, 2009 [43]


• Sandeep Pande, Fearghal Morgan, **Seamus Cawley**, Brian McGinley, Snaider Carrillo, Liam McDaid and Jim Harkin, “EMBRACE-SysC for Analysis of NoC-based Spiking Neural Network Architecture”, *IEEE System on a Chip Symposium (SOC 2010)*, 2010 [45]
1.3 Remote FPGA Laboratory (RFL)

1.3.1 Introduction

There is growing interest in distance and electronic learning due to the widespread availability of broadband speed Internet. Recent work [46–56] in the area of web-based digital systems remote learning environments has demonstrated their efficacy. Such systems provide an efficient method to teach digital systems design remotely.

Phase II of this thesis has developed the Remote FPGA Laboratory (RFL), a web-based, interactive digital design e-learning system, incorporating an always-on remote FPGA hardware prototyping platform. The work has evolved from technologies developed to enable communication between EMBRACE-FPGA and SNNDevSys, and from a goal of the Bio-Inspired and Reconfigurable Computing (BIRC) research group to enhance learning in digital systems.

The RFL provides a real-time control and visualisation interface to an array of FPGA devices housed in the cloud. This work has involved the development of FPGA design applications and an RFL FPGA IP block implementing USB and register-based FPGA-server communication. The system consists of an array of FPGA devices connected to a web application server, which manages user interaction with an FPGA through a web browser based interface. Figure 1.9 illustrates the RFL system architecture.

The RFL helps students to understand and reason about digital systems operation, using interactive animation of signal behaviour in an executing digital logic system, at any level of the design hierarchy. Traditional digital systems design education revolves around static examples of simple circuits and laboratory sessions where students implement digital logic using either breadboards or FPGA hardware. The RFL aims to change this learning paradigm through the use of a web-based, always-on array of FPGA devices.

The availability of devices on demand enables live demonstration of the traditionally static textbook examples, resulting in a greater understanding of the topic and a learn-by-example approach. Users’ ability to configure and control FPGA devices on-demand encourages a learn-by-doing approach. Usage statistics generated by the RFL have illustrated significant student usage of the system outside formal laboratory hours. Students ability to quickly experiment with demonstration hardware designs or to develop, configure and test their own designs further encourages this approach and has shown to increase students time interacting with hardware.

The system supports a number of technology training needs, including industry, third level and second level. The system enables mastery of basic to complex digital logic and
computer architecture design techniques, supported by a planned series of interactive online books/tutorials, structured as formal course modules, with single-click access to a library of interactive demos.

Remote FPGA Lab has been developed primarily as a teaching tool and includes the ability for course tutors to provide course material, documentation and demonstration designs to students. Interactive and animated diagrams with icons and switches reflecting the internal state of the FPGA can be created by users or can be provided by tutors and associated with existing hardware designs. These interactive diagrams provide an informative and dynamic alternative to traditional circuit diagrams and truth tables. A suite of demonstration bitstreams and documentation is also provided.

1.3.2 RFL System Architecture

The RFL system comprises three main components, an array of FPGA devices, a web application server and a web based user interface (UI). Currently the RFL supports Digilent Nexys 2 boards [57] which incorporate a Xilinx Spartan-3E FPGA, USB controller and SDRAM. The RFL server is built using a number of open source technologies including the Nginx [58] webserver, the Django [59] web application framework and the MySQL [60] database server. Django generates web pages (which are served by Nginx) to create the UI, facilitating interaction with the system using a standard web browser.
1.3.2.1 FPGA Array

The FPGA array (Figure 1.10) houses a number of FPGA devices, each connected to the server via USB and paired with a web-cam, which provides a live view of the operating FPGA device, its 7-Segment displays and LEDs. Each FPGA is configured with a bitstream containing two components, a core IP module and the “user design” module. The core module facilitates communication with the server, interfacing with the onboard USB controller, and provides an interface from the user design to the Control and Status Registers (CSRs) and onboard memory. The RFL system supports user designs of any complexity, limited only by the resources of the available FPGA devices. User designs can interface with onboard memory, LEDs, 7-Segment displays and peripherals. The core IP block enables visualisation and control of user signals at any level of the design hierarchy through the user’s web browser.

1.3.2.2 RFL Server Side Architecture

The web application server incorporates the logic of the system, provides user authentication, stores course content and manages interaction between users and FPGA devices.

The RFL server provides the ability for users to upload and manage their own designs, configure FPGA devices, control and visualise internal signals and allows administrative users to monitor usage of the system. The system is built using Django [59], an open source, Model View Controller (MVC) web application framework built with Python [61] which aims to adhere to the DRY (Don’t Repeat Yourself) principle. Django was chosen since it is a high level, lightweight framework which includes a number of features, including user authentication, administrative interface and Object Relational Model (ORM) as standard. The framework has also been shown to be lightweight, to reduce hardware costs, and to be scalable. All of these offer advantages to the future development of the RFL system.

Django provides a complete user authentication framework, enabling each user to maintain a personal profile, and also provides a comprehensive group based permissions facility. The RFL utilises a MySQL database as the systems persistent data-store. The
system maintains a number of Django “models”, which are similar in concept to an object within an object orientated language. A model may have a number of related methods, enabling complex administrative functions to be performed automatically when model instances are created, modified or deleted, e.g. creating UI control icons in System Diagrams for commonly used signals (clk, rst, etc).

The system maintains a number of inter-related models including:

- **FPGA**, a model instance for each FPGA device containing information including whether the device is in use and if so by which user, and for how long, the devices physical location and a reference to the *bitFile* object configured on the device.

- **bitFile**, a model instance storing the location on disk of each FPGA configuration bitfile, the user who ‘owns’ the bitfile, and a list of related diagrams.

- **fpgaUsage**, a model type which stores information related to usage of the system, including a link to the *UserProfile* object, the time spent interacting with an FPGA, and the configured bitfile

- **UserProfile**, an extension to the default Django User object allowing additional information to be stored for each user of the system, e.g. institution, preferred FPGA type, etc.

- **FunctionalPartition**, a model type storing information related to system diagrams, i.e. diagrams related to a *bitFile*, and related visualisation icon types and placement.

- **FPGAType**, a model which links an FPGA instance to a related device specific configuration method, enabling transparent use of different FPGA device families.

Figure 1.11 illustrates a portion of the RFL database schema, including a number of SQL tables, and the inter-relationship between these tables, created by Django to store information directly related to interacting and controlling FPGA devices. These SQL tables are used as the data-store for the models described above. FPGA communication is managed by server daemons, one for each device, which integrate with Django via a Python module. This module provides a high level, device agnostic API to FPGA devices, supported by a model instance managed by the Django ORM for each device. By isolating the interface to hardware devices, the system allows additional device types to be added without modifying the core system software. This improves the robustness and scalability of the system. The system separates components enabling resource intensive system components, such as the database or application logic, to be distributed across multiple machines and to be managed by a load-balancer, supporting future large scale expansion.
Chapter 1. Introduction
1.3.2.3 RFL User Interface

The RFL user interface is generated by Django using a built-in templating language. Django typically generates a web page based on a common base (which includes standard elements such as navigation, styling, etc.) via an MVC View. The main user interface to FPGA devices is based on a single web-page, which is dynamically updated using Asynchronous Javascript and XML (AJAX). The RFL interface makes significant use of Javascript through the jQuery library [62]. In general jQuery uses AJAX requests to retrieve additional information from the server upon user interaction without requiring a page refresh. In order to provide real-time updates of FPGA state to the user, HTTP streaming is used to transfer data from the server to the client (web browser), which improves the speed at which FPGA state can be transmitted to the client since connections are not continuously being created and destroyed, a process which involves significant latency.

HTTP streaming involves an AJAX request which does not strictly follow the standard HTTP request-response model. When Django receives a request for FPGA state information it retrieves this from the FPGA and sends this state to the client. However, it does not close the request but instead continues to poll the FPGA and transmit updated information to the client until the client forcibly closes the connection due to a time-out limit being reached. jQuery then initiates a new connection and the process begins again. HTTP streaming does not rely on third party software, such as Flash or Java, which may not always be available (particularly on mobile devices).

Django performs a number of permissions checks for each new client request, including verifying the user is authenticated and is currently using the FPGA from which state was requested. A large amount of content is cached by both the web server and the client browser, including the response from views which do not produce rapidly changing content and all static resources (images, Javascript files and CSS style sheets, etc) in order to improve both client side loading times and server load.

1.3.3 Related Publications

- **Chapter 7**: Fearghal Morgan, Seamus Cawley, Frank Callaly, Shane Agnew, Patrick Rocke, Martin O’Halloran, Nina Drozd, Krzysztof Kepa, Brian Mc Ginley,
“Remote FPGA Lab with Interactive Control and Visualisation Interface”, *21st International Conference on Field Programmable Logic and Applications*, FPL2011, pp. 496-499 [64]

- **Chapter 8**: Fearghal Morgan, Seamus Cawley, David Newell, “Remote FPGA Lab for Enhancing Learning of Digital Systems”, *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2012, accepted for publication [65]

- **Chapter 9**: “Digital Systems Pedagogy using the Remote FPGA Lab” (under review)
1.4 Thesis Contributions And Novelty Claims

- **Reconfigurable NoC-based hardware SNN architecture**
  This thesis describes a scalable NoC based hardware SNN architecture implemented on FPGA hardware (EMBRACE-FPGA). The architecture supports dynamic configuration of SNN parameters through NoC configuration packets. SNN inputs are managed by an array of Spike Generators (SGs) which convert integer valued inputs to spike train frequencies. SNN outputs are connected to Spike Counters (SCs) which convert spike train frequencies to integer values. The architectural concept has been verified through the use of benchmark SNN applications. EMBRACE-FPGA has demonstrated its ability to control real-time, sensitive and dynamically unstable systems. This thesis describes extensions to the SNNDevSys training and configuration platform to perform on-line testing and debug of the EMBRACE-FPGA system, injection of raw NoC packets into EMBRACE-FPGA (used to verify SNN configuration and NoC routing tables) and to support evolution of SNN topology. Distributed SNN training across multiple computers or FPGA devices has also been implemented, significantly improving SNN training times.

- **Single Dynamic Synapse (SDS)**
  This thesis has proposed the Single Dynamic Synapse (SDS) a novel approach to hardware SNN topology and weight storage which offers reduced memory requirements and an infinite number of incoming ‘virtual’ neuron synaptic connections. The SDS utilises a single incoming synapse for each neuron which is dynamically configured with the appropriate synaptic weight on receipt of a spike packet. The approach removes the requirement that NTs store the destination synapse address for each spike. Instead the synaptic weight is transmitted in the spike packet along with the destination neuron address. Biological synaptic weight resolution is thought to be no greater than 9 bits [66] and in practice is limited to a smaller range than this [67]. For networks with more incoming synaptic connections than distinct weight values the SDS offers improved memory usage and scalability. since the SDS is configured on receipt of a spike packet then neuron has an unlimited number of incoming connections, allowing greater flexibility in SNN topology design. For a network with 1024 neurons, supporting 256 output synaptic connections per neuron and 5 bit synaptic weights the SDS offers a 35% reduction in EMBRACE-FPGA memory usage.

- **Remote FPGA Laboratory**
  This thesis proposes, implements and demonstrates a novel Remote FPGA Lab
(RFL), a web-based system enabling remote interaction, visualisation and configuration of FPGA devices. The system supports web-based access to an array of FPGA devices via a standard web browser. Users can upload and configure their own designs as part of a test driven development strategy. This remote access in conjunction with real-time visualisation and control of internal FPGA signals provides a novel system which enables users to remotely test and verify FPGA designs without additional hardware or software. The RFL also incorporates a course builder, enabling tutors to add course content with embedded live demonstrations of digital design concepts. The system also supports online Continuous Assessment (CA).
1.5 Structure of Thesis

Note, this thesis includes references at the end of each chapter.

Chapter 2 discusses hardware SNNs, describes the EMBRACE-FPGA prototype system and provides an introduction to SNNDevSys: “Hardware Spiking Neural Network Prototyping and Application”, Genetic Programming and Evolvable Machines, Special Issue on Evolvable Hardware Challenges, 2011

Chapter 3 discusses the impact of NoC latency jitter and low resolution digital hardware neural model processing on the behaviour of hardware SNNs: “The Impact Of Neural Model Resolution On Hardware Spiking Neural Network Behaviour”, Signals and Systems Conference (ISSC), 2010

Chapter 4 proposes an alternative, memory efficient, approach to storing hardware SNN topology and synaptic weight information: “Memory Efficient Storage of Reconfigurable Topology Information in Network-on-Chip Based Spiking Neural Networks”, in preparation

Chapter 5 describes the use of evolved topology SNNs for the classification of breast cancer using Ultra Wideband Radar: “Evolving Spiking Neural Network Topologies for Breast Cancer Classification in a Dielectrically Heterogeneous Breast”, Progress In Electromagnetics Research Letters, 2011


Chapter 7: “Remote FPGA Lab with Interactive Control and Visualisation Interface”, 21st International Conference on Field Programmable Logic and Applications, FPL2011, pp. 496-499

Chapter 8: “Remote FPGA Lab for Enhancing Learning of Digital Systems”, ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2012, accepted for publication

Chapter 9: “Digital Systems Pedagogy using the Remote FPGA Lab” (under review)
Chapter 1. Introduction

References


Chapter 1. Introduction


Chapter 1. Introduction


Chapter 1. Introduction


Chapter 2

Hardware Spiking Neural Network Prototyping and Application

Abstract

EMBRACE has been proposed as a scalable, reconfigurable, mixed signal, embedded hardware Spiking Neural Network (SNN) device. EMBRACE, which is yet to be realised, targets the issues of area, power and scalability through the use of a low area, low power analogue neuron/synapse cell, and a digital packet-based Network on Chip (NoC) communication architecture. The paper describes the implementation and testing of EMBRACE-FPGA, an FPGA-based hardware SNN prototype. The operation of the NoC inter-neuron communication approach and its ability to support large scale, reconfigurable, highly interconnected SNNs is illustrated. The paper describes an integrated training and configuration platform and an on-chip fitness function, which supports GA-based evolution of SNN parameters. The practicalities of using the SNN development platform and SNN configuration tool-set are described. The paper considers the impact of latency jitter noise introduced by the NoC router and the EMBRACE-FPGA processor-based neuron/synapse model on SNN accuracy and evolution time. Benchmark SNN applications are described and results demonstrate the evolution of high quality and robust solutions in the presence of noise. The reconfigurable EMBRACE architecture enables future investigation of adaptive hardware applications and self repair in evolvable hardware. Keywords: EMBRACE, Evolvable Hardware, Spiking Neural Networks, Network on Chip, Intrinsic Evolution, FPGA
2.1 Introduction

The basic processing units in organic central nervous systems are neurons which are interconnected in a complex pattern using synapses [1]. The current understanding of biological neurons is that they communicate through pulses and employ the relative timing of the pulses to transmit information and perform computations. Spiking Neural Networks (SNNs) [1–3], which are the third generation of Artificial Neural Networks (ANNs), emulate real biological neurons more closely than traditional ANN models and therefore have the potential to be computationally more powerful [2]. SNNs offer the potential of an elegant, low-power and robust method of performing computation. SNNs are suitable for implementing control applications, e.g. robotics controllers, and classifiers, due to their ability to provide a good solution in the presence of imprecise or unseen data.

SNNs communicate by transmitting short transient pulses or spikes between neurons, via weighted synaptic connections. Synapses can have either an excitatory or inhibitory effect on a neuron’s internal membrane potential. A spiking neuron will emit a spike when its potential exceeds a neuron-specific membrane potential threshold value. The computational power of SNNs is realised by the synaptic connections between neurons, the weights on these connections and the internal membrane threshold potential of each neuron.

Researchers aim to implement reconfigurable and highly interconnected arrays of neural network elements in hardware [4–12]. Research in the area of hardware SNN implementations is driven by a number of factors, including: (1) the acceleration of large SNN simulations to further neuroscience research and (2) an interest in replicating the parallel computational power, efficiency, adaptability and self-repair ability of the brain.

The scalability of synaptic connection between neurons [13] is the major limiting factor in hardware SNN architectures. For large scale hardware SNNs, hardware interconnect introduces problems due to high levels of inter-neuron connectivity. Often the numbers of neurons that can be realised on hardware are limited by high fan in/out requirements [13]. Direct neuron-to-neuron interconnection exhibits switching requirements that grow non-linearly with the network mesh size [13], e.g. a 2-layered fully interconnected network with m neurons per layer exhibits an interconnect density of $m^2$. A number of different approaches have been investigated as a solution to this problem including multicast Network on Chip (NoC) routing [14] and wafer-scale integration [15].

The authors have investigated and proposed EMBRACE [9, 16], a scalable, reconfigurable, low area, low power, evolvable embedded hardware SNN device, offering efficient cost/performance compared to software-based SNNs. The EMBRACE reconfigurable
mixed-signal hardware SNN, which is still to be realised, aims to support the implementa-
tion of large scale SNNs. The EMBRACE architecture incorporates a low area, low
power, CMOS-compatible analogue neuron/synapse cell architecture [9] offering synaptic
densities significantly in excess of that currently achievable in other hardware SNNs [4–8].
EMBRACE implements inter-neuron connectivity through the use of a NoC communica-
tion architecture [10, 17–21], which provides flexible, time-multiplexed communication
channels, scalable interconnect and reconfigurability.

This paper details the issues of inter-neuron connectivity in a scalable hardware SNN
and implements a flexible NoC-based hardware SNN solution. The NoC approach offers
a potential generic solution to the scalability challenges of SNN-based evolvable hard-
ware systems. NoC routers are used to communicate packets of SNN spike and SNN
configuration information across shared buses between routers. The EMBRACE hard-
ware SNN architecture integrates synapse and neuron resources with a NoC router to
form a neural tile. The architecture supports the evolution of SNN topologies (i.e. num-
ber of neurons and/or inter-neuron connections). SNN configuration packets containing
synapse weights, neuron firing thresholds, and network topology (neuron-synapse con-
nection information) are routed on the same shared NoC buses, to individual routers.
This approach enables the efficient re-organisation of hardware SNN resources without
the design complexity of partial reconfiguration.

The requirement for configuration and debugging of hardware SNNs is highlighted and
illustrated through demonstration. The paper highlights issues of interfacing real world
inputs and outputs to hardware SNN platforms, and also proposes and demonstrates
solutions using feasible spike generators and spike counters. System level techniques
such as these can find application across the evolvable hardware field.

EMBRACE research incorporates (1) SystemC modelling and SNN NoC traffic analy-
sis [22], (2) mixed-signal hardware architectural design, (3) SNN training platform and
SNN configuration tool-set, and (4) hardware SNN prototyping and application using
the EMBRACE-FPGA platform.

This paper describes the design and application of EMBRACE-FPGA, a digital-based
prototype of EMBRACE, implemented on FPGA hardware. This platform has been
developed to practically verify the suitability of the NoC inter-neuron communication
approach, to explore SNN design issues and to support the development of hardware
SNN training and configuration tools, for future application with the EMBRACE mixed
signal hardware SNN. EMBRACE-FPGA models the EMBRACE analogue neural cell
using a digital soft-core processor. The practicalities of using the SNN development
platform and SNN configuration tool-set are described.
The paper describes the EMBRACE-FPGA architecture, including the digital neuron/synaptic processing elements and the NoC router, and presents resource utilisation results.

The paper considers the impact on SNN accuracy and evolution time, of the EMBRACE-FPGA NoC latency jitter noise, introduced by the round robin NoC router arbitration used, and the EMBRACE-FPGA processor-based neuron/synapse model. Results presented demonstrate the ability of EMBRACE-FPGA to successfully evolve high quality SNN solutions which are robust to noise, introduced as a result of the NoC routing scheme and the low resolution embedded processor-based neuron model.

In addition, the paper presents SNNDevSys, a Genetic Algorithm (GA) based training and configuration platform, and outlines the practicalities of intrinsic evolution of NoC-based SNNs. SNNDevSys supports the evolution of SNN connection topologies, synaptic weights and neuron firing thresholds.

SNN accuracy and training time results are provided for a range of control and classifier applications, including the benchmark XOR function, an inverted pendulum controller and a Wisconsin cancer dataset classifier. EMBRACE-FPGA includes an on-chip fitness function which accelerates the intrinsic hardware evolution time to compare well with that of software SNNs. SystemC-based EMBRACE architectural simulation and NoC traffic analysis [22] have been used to characterise EMBRACE NoC behaviour and to verify the practicalities of using a NoC based inter-neuron communication strategy to support SNN operation.

This work contributes to the development of the EMBRACE mixed-signal NoC-based embedded hardware SNN device and tools. The techniques and tools described in the paper can be applied to EMBRACE when implemented. The EMBRACE reconfigurable architecture enables future investigation of adaptive hardware applications and self repair in evolvable hardware.

The structure of this paper is as follows: Section 2.2 summarises related work in the area of hardware SNN implementations. The EMBRACE-FPGA NoC, neuron/synapse architecture and their operation are described in section 2.3. Section 2.4 details the intrinsic hardware SNN evolution, training and configuration platform. Section 2.5 presents results of the application of EMBRACE-FPGA to intrinsically evolve a range of benchmark SNN applications. Section 2.6 highlights the occurrence and sources of noise within EMBRACE-FPGA, considers the impact of noise on SNN training time and accuracy, and describes the expected benefits arising from using an analogue neural cell in the proposed EMBRACE architecture. Section 2.7 concludes the paper.
2.2 Related Work

This section summarises related work in the area of hardware SNN implementations. Inspired by biology, researchers aim to implement reconfigurable and highly interconnected arrays of neural network elements in hardware to produce powerful signal processing units [5–11, 23, 24]. Execution architectures for SNN neural computing platforms can be broadly categorised as software-based (multi-processor) [10, 12], FPGA [5, 6, 8, 25] or analogue/mixed-signal [7, 9, 11, 15, 23].

The SpiNNaker project [10] aims to develop a massively parallel computer capable of simulating SNNs of various sizes and topology and with programmable neuron models. The SpiNNaker architecture uses ARM-968 processor-based nodes for computation and an off-chip NoC communication infrastructure. Software implementation of the neuron model provides flexibility for the SNN computation model. SpiNNaker is aimed at exploring the potential of the spiking neuron as a component from which useful systems may be engineered. The SpiNNaker architecture does not target embedded systems applications due to its size, cost and power requirements.

FPGA-based architectures offer high flexibility for system design. Pearson et al. [5] describe an FPGA-based (Single Instruction Multiple Data) array processor architecture, which can simulate networks of over 1,000 neurons. The architecture uses a bus-based communication protocol which limits scalability of the architecture. Ros et al. [6] present an FPGA-based hybrid computing platform, where the neuron model is implemented in hardware and the network model and learning are implemented in software. Glackin et al. [25] use a time multiplexing technique to implement large SNN models (with \( \approx 1.9 \)M synapses and 4.2K neurons), implemented in software on four soft IP processors cores. Speed-acceleration is the key motivation rather than the parallel capability of SNNs as an embedded device.

Efficient, low area and low power implementations of neuron interconnect and synaptic junctions are key to scalable hardware SNN implementations [11]. Analogue design approaches enjoy the benefit of compact area implementations due to the inherent similarity with the way electrical charge flows in the brain [7, 11]. Analogue neural network architectures have been integrated with digital components for a flexible communication infrastructure. Ehrlich [26] and Schemmel [15] present FACETS, a configurable wafer-scale, mixed-signal neural ASIC system, targeting neuron densities of 105 neurons. The work proposes a hierarchical neural network and the use of analogue floating gate memory for synaptic weights. The synaptic ion channel circuit comprises op-amp and operational transconductance amplifiers along with passive RC components. Similarly, Vogelstein at al. [7] present a mixed-signal SNN architecture of 2,400 analogue
SNN Architecture | Area | Power | Biological Realism
--- | --- | --- | ---
SpiNNaker | High | High | High
FACETS | High | High | High
EMBRACE-FPGA | Low | Low | Low
EMBRACE ASIC (Proposed) | Low | Low | Medium-High

Table 2.1: Comparison of different SNN architectures

<table>
<thead>
<tr>
<th>Type of SNN architecture</th>
<th>Area</th>
<th>Power</th>
<th>Level of architectural reconfigurability</th>
<th>SNN execution speed</th>
<th>Neuron model accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>ASIC</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of hardware SNN technologies

neurons, implemented using switched capacitor technology and communicating via an asynchronous event-driven bus.

A scalable, mixed signal, reconfigurable hardware SNN neural tile array (EMBRACE) has been reported in [9, 20, 27]. This incorporates a compact reconfigurable CMOS-compatible analogue SNN neuron cell architecture. EMBRACE uses a digital NoC-based packet switching interconnect, integrated with low-power analogue neural circuitry, to realise a scalable SNN architecture. In particular, EMBRACE exploits the NoC to allow full or partial reconfiguration of the network weights, neuron potential firing threshold and connection topology. For example, configuration packets are injected into the NoC array to reconfigure selected regions allowing fine-grained changes in the interconnectivity and synaptic strength between neurons. This approach minimises the required configuration circuitry area and reduces the traditional power overheads associated with such circuitry.

Table 2.1 presents a comparison between the biologically faithful SpiNNaker and FACETS platforms and EMBRACE. SpiNNaker and FACETS both provide large-scale, complex and biologically-realistic hardware SNN systems to ultimately assist in the exploration and understanding of neuroscience. In Table I biological realism refers to both the realism of the neuron model [28] and the simulation of other biological factors such as synaptic delays. EMBRACE is targeted as a low-power, embedded computation device (ultimately to be in used future embedded intelligent electronic devices). Hence the issues of area and power consumption are important. Since the area required for complex neuron models on FPGA hardware is substantial, the complexity of the neuron model implemented in EMBRACE-FPGA is limited.
Table 2.2 presents the relative merits of implementing SNN architectures on three main technologies, namely software, FPGA and ASIC. For complex (large) SNNs, hardware implementations offer superior speed of execution compared to sequential software architectures due to the inherent parallelism of hardware. A benefit of parallel operation is that the execution times of hardware SNN architectures scale significantly better than software based implementations. A floating point software implementation can accurately model neuron behaviour such as a Leaky Integrate and Fire (LIF) model. The reported EMBRACE-FPGA neuron model accuracy is low, though provides reasonable quality results for the applications reported in this paper. FPGAs are unsuitable if higher accuracy neuron models are required. Mixed signal ASIC implementations however, have the ability to implement neuron behaviour in custom hardware. Results of the EMBRACE low area analogue neuron cell implementation [11] indicate that accurate, complex functions such as Leaky Integrate and Fire (LIF) can be realised in a small area.

The EMBRACE research project activity includes SystemC-based design exploration, mixed-signal hardware architectural design, creation of an SNN development platform and SNN configuration tool-set, and hardware SNN prototyping and application using the EMBRACE-FPGA platform. EMBRACE-FPGA models the EMBRACE analogue neural cell using a digital soft-core processor. This platform has been developed to practically verify the suitability of the NoC inter-neuron communication approach, to explore SNN design issues and to support the development of hardware SNN training and configuration tools, aimed for application with the EMBRACE mixed signal hardware SNN. The EMBRACE reconfigurable architecture also enables future investigation of adaptive hardware applications and self repair in evolvable hardware.

2.3 EMBRACE-FPGA Architecture and Operation

This section describes the EMBRACE-FPGA SNN neural tile array (Figure 2.1), the NoC router architecture, its operation and resource usage. The routing of spike and configuration packets is illustrated (Figure 3). NoC packet latency jitter is highlighted as an inherent consequence of the NoC round robin arbitration policy. The EMBRACE-FPGA SNN input-output interface and the processor-based neuron/synapse model used in EMBRACE-FPGA are also described. Table 2.3 lists FPGA resource usage for the major neural tile components of EMBRACE-FPGA, implemented on a Xilinx Virtex II Pro FPGA.
Chapter 2. Hardware SNN Prototyping and Application

2.3.1 SNN Neural Tile Array and NoC Router Architecture and Operation

EMBRACE-FPGA is a 2-dimensional array ($N \times M$) of interconnected SNN neural tiles (Figure 2.1), where each tile is connected in North, East, South and West directions, forming a nearest neighbour connection scheme. Figure 2.2 illustrates the EMBRACE-FPGA neural tile block diagram which details the main elements in the NoC router and neural cell (neuron/synapse elements).

Communication between neural tiles is achieved by routing data packets through North, South, East and West neural tile ports. NoC router configuration memory stores SNN topology information, synaptic weights and the neuron firing threshold. Each EMBRACE-FPGA neural cell contains up to 32 synapses and a neuron computing component, implemented using a picoBlaze processor.

The SNN NoC router Finite State Machine (FSM) (Figure 2.2) manages router activity. A round robin arbitration policy is used to handle spike packet generation, on the occurrence of a neuron spike (assertion of neuronSpikeOut signal) from within the attached neural tile, or when an incoming NoC spike packet request is received by the router. Subsystem decodePkt decodes an incoming NoC packet to determine if the packet is route-through (destined for another tile) or has reached its destination neural tile. Subsystem buildPkt assembles the NoC output data packet for transmission to a neighbouring neural tile. Neural tile configuration data is stored in the neural tile configuration memory ($cfgMem$) block. NoC configuration packets contain the...

---

**Figure 2.1:** EMBRACE-FPGA NoC-based SNN hardware neural tile array

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of Slices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Tile</td>
<td>641</td>
</tr>
<tr>
<td>Router</td>
<td>271</td>
</tr>
<tr>
<td>Configuration Memory</td>
<td>172</td>
</tr>
<tr>
<td>Digital Neural Cell</td>
<td>237</td>
</tr>
</tbody>
</table>

**Table 2.3:** Summary of FPGA resource usage for an EMBRACE-FPGA Neural Tile
destination neural tile address along with configuration parameter address and data. The NoC supports configuration of the following parameters:

1. Number of active synaptic connections

2. Address of each neuron/synapse pair to which the current neuron connects and corresponding output port selection; an SNN connection topology routing table determines the most appropriate N, S, E, W output port for forwarding packets to the destination neuron/synapse.

3. Synaptic weights and neuron threshold potential

The following router packet formats are supported by the NoC router:

1. Spike or configuration packet route-through: packets received by any neural tile, other than the intended destination tile, are passed directly through the router.

2. Neural tile configuration request: configure NoC and neuron parameters, including connection topology, synaptic weights, neuron firing threshold.
3. Spike packet received for current tile (reached destination): generates a spike to the addressed synapse within the current neural tile.

4. Spike test packet: used to route single spike packets from specified source tile to destination neuron/synapse for fault detection.

5. Neural tile configuration and status read-back of a) neural tile configuration values, and status, e.g. occurrence of packet transmission time-out. b) picoBlaze neuron membrane potential.

An SNN is realised on the EMBRACE-FPGA architecture by programming neuron parameters (synaptic weights and neuron threshold) in each neural tile and NoC connection topology. The network of routers allows spike signals to be propagated from source to destination using time-multiplexing of the connecting lines between NoC routers. A neuron spike firing event triggers the forwarding of spikes to each associated synapse. Spike routing uses one or many NoC router transmissions. Within each neural tile, neuron potential is modified as a step function on receipt of a spike. The neuron fires if the neuron membrane potential rises above the neuron firing threshold. Spike output events are buffered in a FIFO allowing spike packets to be queued for transmission to the connected synapses. Figure 2.3 illustrates the EMBRACE-FPGA NoC packet routing behaviour, following the firing of a spike by neuron N0. Neuron N0 is connected to both synapse 1 of N1 and synapse 0 of N3. The router of neural tile NT 0 generates a spike packet for each destination neuron/synapse pair. The spike packet destined for synapse 1 of N1 travels via the East output port of N0. The packet is decoded in neural tile N1 and a spike is applied to synapse 1. The spike packet destined for Synapse 0 of N3 is transmitted via the South port of NT 0. Neural tile NT2 forwards the packet to N3. The packet is decoded in NT3 and a spike is applied to synapse 0 of N3.

2.3.2 NoC Packet Latency Jitter

EMBRACE-FPGA uses a simple round robin packet arbitration scheme which ensures that each NoC router port is serviced fairly under spike traffic loads. This approach introduces variations or jitter in the latency of packets as they are transmitted between routers. Latency jitter has the effect of distorting spike train frequencies which can affect the intended operation of a configured SNN. Section 2.6 describes the effect of this ‘noise’ (undesired effect) on the accuracy of evolved applications and section 6 discusses the causes of, and solutions to, noise within the EMBRACE architecture.
2.3.3 EMBRACE-FPGA SNN Input-Output Interface

Figure 2.4 represents the EMBRACE FPGA SNN system illustrating the SNN neuron structure and interconnect, along with the input encoders and output decoders. EMBRACE-FPGA utilises a series of Spike Generators (SGs), illustrated on the left side of Figure 2.4, that receive integer valued network inputs from the host and generate a continuous train of spike packets at the configured frequency, which are then transmitted to neurons within the network. SNN outputs are decoded by a series of Spike Counters (SC). SCs implement the SNN output layer neurons, counting the number of SNN output spikes received within a fixed time window, and converting the output spike frequencies to integer values which are read by the host for SNN performance (fitness) evaluation. SCs (Figure 2.4) and debug memory are included in a monitor tile.
2.3.4 EMBRACE-FPGA Processor-based Neuron/Synapse Model

Each EMBRACE-FPGA prototype hardware SNN neuron model is implemented using a PicoBlaze 8-bit RISC microcontroller. The neuron model could also be implemented as a dedicated HDL element. A 5 bit synapse weight resolution and 16 bit neuron firing threshold resolution have been used. The neuron processing element receives and processes neuron configuration information via the NoC router. Synaptic weight and neuron membrane potential threshold values are stored in PicoBlaze scratchpad RAM. A spike packet received by the NoC router triggers a PicoBlaze Interrupt Service Routine (ISR) which retrieves the appropriate weight value for the addressed synapse (inhibitory or excitatory), and modifies the current neuron membrane potential value. The neuron fires when the neuron membrane potential exceeds the configured threshold value. The NoC router then generates a spike packet to each connected synapse. Firing of the neuron also triggers a refractory period timer and resets the neuron potential to zero. During normal operation the PicoBlaze neuron model implements neuron potential decay. Figure 2.5 illustrates the neuron membrane potential behaviour obtained from a simulation of the EMBRACE-FPGA RTL design.
The membrane potential variation (Figure 2.5) results from incoming spike trains on two synapses (synapse 0 and synapse 1) arriving non-periodically. Neuron potential increases with incoming spikes on synapse 0. On receipt of a spike, the neuron potential increases by the synapse 0 weight value. The plot illustrates the periodic membrane potential decay, which reduces the neuron potential by a fixed factor every time period. The neuron generates an output spike when its potential reaches its firing threshold. After firing, the neuron potential returns to its resting value and is unaffected by incoming spikes for the duration of the neuron refractory period. The single spike on synapse 1 results in a large increase in neuron potential due to the large synapse 1 weight (5 times greater than the weight of synapse 0 in the example shown in Figure 2.5).

The EMBRACE-FPGA neuron model can be configured to store membrane potential in the FPGA’s BRAM (for read-back and debug) each time the neuron potential changes due to periodic decay or receipt of a spike. Figure 2.6 illustrates the variation in neuron membrane potential in one neuron when a continuous series of spikes are applied to it at a fixed frequency.

The increase in membrane potential (with incoming spikes), and decrease in membrane potential (with periodic potential decay) results in a stair-step increase in potential up to
the neuron firing potential. The neuron fires at the point where the potential decreases sharply. The irregular step pattern of Figure 2.6 illustrates the low resolution of neuron potential calculations performed by the model.

2.4 Intrinsic Hardware SNN Evolution, Training and Configuration Platform

This section presents SNNDevSys, the GA-based intrinsic hardware SNN evolution, training and configuration platform, used with EMBRACE-FPGA. This section also summarises the hardware SNN development methodology used in this work. Figure 2.4 illustrates the EMBRACE-FPGA and SNNDevSys training and configuration platform system elements. SNNDevSys is currently implemented as a workstation based system which communicates with EMBRACE-FPGA through a register-based communication protocol. The platform also has the capability to send and receive raw NoC packets. SNNDevSys provides a platform for future EMBRACE mixed-signal hardware SNNs.

2.4.1 GA Based Training

Intrinsic evolution involves implementing and evaluating each individual in a GA population in hardware. This differs from extrinsic evolution which evaluates individuals by simulating the circuit in software, followed by selection of an individual for implementation in hardware. The use of intrinsic evolution offers improved accuracy, fast evolution and requires no reliance on the accuracy of circuit simulation tools.

GAs [29] are a type of Evolutionary Algorithm based on the same mechanisms of adaptation observed in nature, including natural selection and genetic diversity. Guided by the “survival of the fittest” principle, GAs are effective global search algorithms that require points in the solution space to be encoded into a linear data string (genome). Apart from implementing the encoding mechanism and fitness function, GAs also perform the genetic operations. A combination of crossover and mutation operators are used to evolve solutions to the benchmark problems described in this paper. The GA randomly generates the initial SNN population parameters for each neural tile (synaptic weights and neuron firing threshold), encodes this information as an integer-valued genome, and decodes this into corresponding binary-valued SNN configuration values and the corresponding NoC configuration packet sequence. The evolution process involves selecting an SNN configuration, applying each individual training dataset to the SNN, and evaluating the fitness of the evolved individual SNN configuration. Once each individual in a population has been evaluated, the GA operation of selection, crossover
and mutation are applied to create a new generation of SNN solutions. This evolutionary process continues until high fitness convergence is achieved. The host uses a register-based communication protocol to transfer configuration packets to EMBRACE-FPGA, and to control the generation of spike packets into the EMBRACE-FPGA SNN.

The hardware SNN development methodology used to successfully evolve SNN solutions using the SNNDevSys training platform and the EMBRACE-FPGA hardware SNN is as follows: SNNDevSys sequentially configures each active neural tile within the available EMBRACE-FPGA SNN, by sending topology memory configuration packets. Careful encoding of input spike train frequencies is required to aid successful evolution of SNN behaviour. For each neuron, the synaptic weights, neuron firing threshold, SNN input spike train frequency and the neuron potential decay rate collectively govern neuron spike firing rates.

The user configures the host to scale each real-valued EMBRACE-FPGA input within a range of 1 to 255 (8 bit). SGs convert SNN inputs to spike train frequencies. The scaling of SNN inputs is controlled to guarantee that a wide range of possible SNN output spike rates can be produced by the network so that a good SNN output spike rate resolution can be provided to the host. This scaling factor is also chosen to minimise neuron firing rate saturation which would cause loss of information.

To meet target firing rates, an iterative approach is followed whereby the user scales the SNN input data and modifies the neuron potential decay rate while monitoring the SNN output spike frequency. Poor parameter selection can result in saturation of neuron membrane potential and hence loss of information within the SNN. Once this tuning has been completed, SNNDevSys automates the evolution process.

2.4.2 Adaptability and Self Repair

A fault tolerant evolvable hardware SNN system, capable of self repair requires

(A) A robust and non-disruptive method of identifying and locating faults in SNN inputs (e.g. sensors in an SNN controller application) and SNN internal elements. Ideally, the fault detection method would not disrupt the operation of the SNN

(B) A method of determining a solution to the fault

(C) The ability to restore full operation of the SNN after any noise introduced through the repair process (e.g. increased latency jitter due to longer NoC connections).

(D) A full or partial reconfiguration architecture.
The use of a NoC to communicate configuration and spike packets through the SNN allows reconfiguration of system parameters during runtime. The EMBRACE architecture provides the ability to adapt to changes in the SNN, or its environment. Examples could include restoring functionality by adapting the SNN through reconfiguration where (a) noise is introduced by a defective sensor, (b) a defect occurs in a System on Chip. Investigation of the use of EMBRACE for fault tolerant applications will be the subject of future research.

2.4.3 On Chip Fitness Function

The on-chip fitness function described in this subsection moves much of the logic required to evaluate the fitness of an individual onto the FPGA. An application SNN training set comprises a series of training vectors and expected outputs. For each set of vectors and expected outputs, the host loads the Spike Generators (SGs) with the appropriate training vector. This ensures that each SG applies spikes at a rate proportional to the training vector value. The fitness function reads the SNN output spike rate from each output SC after a defined time interval. The SNN output spike rates are used to evaluate the individual’s fitness. For a large training set, communicating many training vector values to on-chip SGs and reading SC values can take a significant amount of time. The use of an on-chip approach reduces the information communicated to the FPGA to a small fixed number of bytes and avoids the high latency and bandwidth limitations of the host-FPGA communication protocol. The on-chip fitness function within EMBRACE-FPGA incorporates the entire training dataset in on-chip RAM and executes the fitness function on chip. The on-chip fitness calculation reduces the information exchange between SNNDevSys and the hardware SNN, and consequently achieves a significantly improvement in individual evaluation time. SNNDevSys configures each SNN individual, triggers the hardware SNN evaluation, reads back the fitness for each individual, and uses this fitness score to evolve the next generation of SNN configurations.

2.5 EMBRACE-FPGA SNN Benchmark Applications

This section presents results of the application of EMBRACE-FPGA to intrinsically evolve a range of benchmark SNN applications. A series of benchmark applications have been employed progressively to demonstrate increasingly complex hardware SNN solutions, and to highlight challenges in developing a hardware SNN architecture. SNN accuracy (fitness) and training time results are provided for successfully evolved hardware SNNs, implementing the benchmark XOR function, an inverted pendulum controller
and a Wisconsin cancer dataset classifier. The various applications verify the ability of the EMBRACE and SNNDevSys systems to solve both non-linear and unstable tasks.

The XOR benchmark has been a significant benchmark with which the authors have tested the system. This simplicity of the XOR application has facilitated a greater understanding of the neuron model and NoC dynamics.

Table 2.4 presents details of the GA parameters used for each application. Pm and Pc represent the probability of mutation and crossover in the GA population respectively. N is the number of individuals in each population and Mg is the number of generations evolved by the Genetic Algorithm. SNNDevSys uses a single elite individual in each GA population. Table 2.4 also presents details of the SNN topology, including Spike Generators and Spike Counters (Figure 2.4).

### 2.5.1 EMBRACE-FPGA SNN XOR Implementation

Implementing the XOR function is a classic problem for neural networks, as it is a sub-problem of more complicated ones [30]. The XOR problem has been used as a standard benchmark to verify the operation or evaluate the performance of ANN [23, 31] and SNN [32] frameworks. The application of EMBRACE-FPGA to solve the XOR benchmark problem has been reported in [33]. Figure 2.7 illustrates the XOR SNN arrangement and interconnection of three neural tiles which implement a two-layer, fully interconnected, feed-forward SNN. The inset at the bottom left of Figure 2.7 illustrates the three-neuron SNN topology. EMBRACE-FPGA implements an SNN XOR solution using two SG input layer neurons, two hidden layer neurons and one output layer neuron. Each XOR SNN individual consists of three genes, one describing three configuration values for each neuron, i.e., two synaptic weights and one neuron firing threshold. SNN topology configuration is not currently evolved in EMBRACE-FPGA applications.

Figure 2.8 illustrates the average and best fitness of the evolved SNN XOR function (the target fitness value is 16). Lower average fitness can be attributed to (1) lower fitness individuals introduced by the GA mutation operator and to (2) ‘noise’ caused by the round robin induced spike packet latency in the hardware platform.
Figure 2.7: SNN neural tiles interconnection for implementation of a two-layer feed-forward SNN (used for evolution of the XOR benchmark function)

Figure 2.8: Average and best fitness of the evolved EMBRACE-FPGA hardware SNN XOR function. Target fitness is 16
2.5.2 Robot Controller

A robotics simulator [34] is included within the SNNDevSys platform. The robot is equipped with four forward facing sonar sensors, used as inputs to the network. The robot includes two motors which are controlled by the outputs of the evolved SNN. EMBRACE-FPGA is configured to implement the neural network illustrated in Figure 2.9. SNNDevSys controls the interaction between EMBRACE-FPGA and the robot simulation environment. The GA randomly generates an initial population and each individual is configured in turn to the SNN.

Fitness assessment of the robotics controller solutions (for obstacle avoidance) is achieved using a fitness function that rewards individuals based on the following behavioural qualities:

1. Obstacle avoidance: survival time ($T$) within the environment (when an individual crashes into an obstacle it ‘dies’)
2. High speed: the distance ($D$) traversed by the robot in the allotted time
3. Effective exploration of the environment: i.e. the aggregate ($AWS$) of left ($aws_1$) and right ($aws_2$) average wheel speeds. This aggregate (minimum 1, a straight line) is inversely proportional to fitness.

An individual evaluation time of 100 seconds has been arbitrarily selected. Testing terminates if the individual crashes (fails) during the testing period.

$$F = \frac{T + D_\alpha}{AWS^\beta}$$  (2.1)
Two constants \((\alpha, \beta)\) are introduced to the fitness assessment. By varying the relative values of each of the constants, different solutions and behaviours are evolved by the GA. These include wall hugging or high speed control strategies within the evolved robots. Equation 2.1 describes the robotics fitness function formula.

The GA used to evolve these behaviours employs tournament selection and elitism as part of the evolution process. Incorporation of elitism in a GA retains the best individual from a population into the next generation. This is aimed at ensuring that the best fitness score of subsequent generations does not decrease. GA parameters are listed in Table 2.5. Figure 2.10 illustrates the best and average intrinsically evolved hardware SNN robotics controller fitness.

The noisy nature of the embedded neuron model is visible as a reduction in best fitness achieved between some generations. This can only occur when the best individual from a previous population, which is retained due to elitism, does not achieve the same fitness when re-evaluated. This reduction in fitness reduces the probability of retention of
Figure 2.11: EMBRACE-FPGA and SNNDevSys configuration platform used for evolution of inverted pendulum controller function.

This individual between generations. As such individuals will be removed due to their sensitivity to noise the GA will evolve solutions which are more robust to noise inherent in the system. However, this behaviour increases the time required to successfully evolve a highly fit robotics controller. An individual which achieves a fitness of greater than 7,000 (Figure 2.10) is capable of successfully navigating the simulation world for 100 seconds.

2.5.3 Inverted Pendulum Controller Application

Figure 2.11 illustrates the EMBRACE-FPGA and SNNDevSys intrinsic hardware SNN evolution, training and configuration platform, used for evolution of the SNN-based inverted pendulum controller function. EMBRACE-FPGA implements the inverted pendulum controller using four SG input layer neurons, four hidden layer neurons and two output layer neurons. The inverted pendulum SNN genome is comprised of five values for each neuron, i.e., four synaptic weights and one neuron firing threshold.

The inverted pendulum experiment incorporates a cart (which can move along the x-axis), and a hinged pole connected to the cart. The objective is to maintain the pole standing for as long as possible within 30 degrees of the vertical position, by controlling the movement of the cart. This problem is a standard benchmark for control methodologies due to its instability and highly non-linear behaviour. Pasero et al. [35] and Uribe et al. [36] have successfully implemented an FPGA-based inverted pendulum hardware
SNN controller using back-propagation and reinforcement-learning ANNs respectively. Similarly, several researchers have demonstrated the feasibility of applying evolutionary techniques to locate an inverted pendulum controller solution [24, 37]. The simulated pendulum position and velocity information (horizontal $x$ position, $x$ velocity, pole angle $\Theta$ position and $\dot{\Theta}$ velocity) are provided as inputs to the hardware SNN (translated into a related spike rate in the $\text{spikeInPkt}$ generator (Figure 2.4). The SNN $\text{spikeOut}$ monitor (Figure 2.4) provides two output spike count frequency values as absolute left and right motor forces to be applied to the simulated cart. The dynamics of the simulated inverted pendulum are governed by angular acceleration (Equation 2.2), where $\theta$ is the angle of the pole from the vertical, $x$ is the position along the $x$ axis, $g$ is gravity, $F$ is the correcting force applied to the cart, $m_p$ is the mass of the pole (0.1Kg), $m_c$ is the mass of the cart (1Kg) and $l$ is the length from the cart to the pole’s centre of gravity (0.5m). Note: In these equations $\Theta$ is measured in radians.
Figure 2.13: WBCD classifier hardware SNN topology (nine hidden layer neurons and two output neurons)

\[
\ddot{\Theta} = \frac{g \sin \Theta + \cos \Theta \left( -\frac{F - m_p l \dot{\Theta}^2 \sin \Theta}{m_c + m_p} \right)}{l \left( \frac{4}{3} - \frac{m_p \cos^2 \Theta}{m_c + m_p} \right)} \tag{2.2}
\]

\[
\ddot{x} = \frac{F + m_p l \left( \dot{\Theta}^2 \sin \Theta - \ddot{\Theta} \cos \Theta \right)}{m_c + m_p} \tag{2.3}
\]

Figure 2.12 illustrates the average and best fitness of the evolved inverted pendulum EMBRACE-FPGA hardware SNN controller (averaged over 300 generations). The evolved SNN successfully balances the pole upright for 100 seconds (tested using an inverted pendulum simulator). Results illustrate a high quality solution in the presence of NoC packet latency jitter noise, and using a relatively low resolution 8-bit PicoBlaze neuron model. The effect of noise is apparent in Figure 2.12, since noise may cause the pendulum to fall over, thereby drastically affecting the entire fitness evaluation.

### 2.5.4 Wisconsin Breast Cancer (WBC) Dataset Classifier

This section illustrates the EMBRACE-FPGA hardware SNN accuracy (fitness) and training time results for an evolved Wisconsin Breast Cancer (WBC) dataset classifier [38] hardware SNN. Figure 2.13 illustrates the EMBRACE-FPGA and intrinsic hardware SNN evolution, training and configuration platform (SNNDevSys) used for
evolution of the SNN-based WBC dataset classifier. The WBC dataset consists of 683 sets of 9 inputs (in the range [0, 1] with resolution 0.1) along with corresponding true/false binary classification decision. EMBRACE-FPGA implements an SNN solution to the WBC dataset classifier using nine SG input layer neurons, nine hidden layer neurons and two output layer neurons.

The SNN includes SGs, nine hidden layer neurons, two output layer neurons, and two SCs. GA parameters are listed in Table IV. Each GA individual comprises eleven genes, corresponding to the eleven SNN neurons. The GA evolves synaptic weights and neuron potential thresholds over 300 generations. Figure 2.14 illustrates the average and best fitness of the evolved EMBRACE-FPGA hardware SNN WBC dataset classifier (averaged over 20 GA runs). An accuracy of 96.8% for WBC dataset classifier has been achieved by the EMBRACE-FPGA SNN. The hardware SNN WBCD classifier [39] reports a 90.2% precision accuracy for a training set comprising 65% of the WBC dataset.

SNNDevSys evolutionary training time is dictated by the time taken to evaluate the fitness of an individual. For a large dataset classification such as the WBC dataset, which contains 683 x 9 test vectors, fitness evaluation is limited by the communication latency between the host and SNN. The generic on-chip hardware fitness function, implemented
in EMBRACE-FPGA, reduces the fitness evaluation time and speeds up SNN training
time. The use of this on-chip fitness function improves SNN training time by a factor of 18, compared to a solution implementing the fitness function on the host.

2.5.5 Noise

Noise has been observed in the best fitness plots for a range of evolved EMBRACE-FPGA based SNN applications [16, 27] This noise is introduced primarily through (1) the use of a low resolution neuron model and (2) packet latency jitter inherent in the NoC. While this noisy operation is a disadvantage in SNN operation, it is mitigated by the ability of SNNs to generalise. The use of a GA to evolve SNN configurations also means that individuals which are overly sensitive to this noise are discarded in favour of more robust individuals. Section 6 discusses in detail the sources of noise within a NoC based SNN.

2.6 Sources and Impact of Noise

This section highlights the occurrence and sources of noise within EMBRACE-FPGA and considers the impact, on SNN training time and accuracy, of (1) the EMBRACE-FPGA NoC latency jitter noise, introduced by the round robin NoC router arbitration used, and (2) the EMBRACE-FPGA 8-bit PicoBlaze processor-based neuron/synapse model. Results presented in Section 2.5 demonstrate the ability of EMBRACE-FPGA to successfully evolve high quality SNN solutions which are robust to noise. This section examines the sources of noise within the system and describes the expected benefits arising from using an analogue neural cell in the proposed EMBRACE architecture.

The neuron model used in EMBRACE-FPGA is implemented using an 8-bit PicoBlaze processor. Figure 2.15 illustrates the low resolution of the EMBRACE-FPGA PicoBlaze model neuron potential decay. Results have been obtained through simulation of an RTL model of the EMBRACE-FPGA hardware neuron.
In this example, for a constant input spike frequency, the internal neuron potential reaches a steady state. This is achieved when neuron potential decay rate equals the rate of neuron potential increase due to incoming spike packets. The neuron potential does not reach the neuron firing threshold and hence no output spike is generated. The PicoBlaze-based neuron model implementation processes neuron input spikes through the use of interrupts. The processing of the ISR, and associated context switching performed by the processor, has the effect of distorting normal neuron decay operation. This is apparent in the stepwise nature of the plot and has the effect of reducing the resolution of possible neuron output spike train frequencies, resulting in a low output resolution of the evolved SNN. Figure 2.16 illustrates the impact of NoC spike packet latency jitter (for the same input spike train as that of Figure 2.15) on the neuron membrane potential plot.

The impact of NoC latency jitter is a small distortion of neuron input spike train frequency. In Figure 2.16, this distortion causes the neuron potential to reach the neuron firing threshold and hence to generate unintended output spikes, which in turn can cause incorrect operation of a configured SNN. The use of a GA to evolve SNN configurations means that individuals which are overly sensitive to this noise are discarded in favour of more robust individuals. This feature ensures the ability of EMBRACE-FPGA to implement tasks at the expense of increased evolution time.

The neuron model proposed for the EMBRACE SNN architecture is implemented as a CMOS compatible analogue neural cell. Due to the analogue nature of the device, the neural cell may be considered to perform calculations with infinite resolution. This neural cell does not suffer from the limitations of the EMBRACE-FPGA digital neuron model related to neuron output spike train frequency resolution and neuron potential decay rate distortion. The low resolution of neuron potential processing performed by the PicoBlaze-based EMBRACE-FPGA model amplifies the effect of latency jitter. Simulation of the EMBRACE analogue neuron cell results in a greatly reduced variation of neuron potential, and intended neuron output, caused by NoC latency jitter due to the smoothing effect of the continuous neuron potential decay.


2.7 Conclusions

This paper describes EMBRACE, a proposed scalable, reconfigurable, mixed signal, embedded hardware SNN architecture. The paper details the implementation and testing of EMBRACE-FPGA, an FPGA-based hardware SNN prototype, and its application to explore issues and verify concepts surrounding the implementation of a NoC based hardware SNN, and the development of hardware SNN training and configuration tools. The paper describes an integrated training and configuration platform which supports GA-based evolution of SNN parameters. The practicalities of using the SNN development platform and SNN configuration tool-set are introduced. EMBRACE-FPGA validates the EMBRACE architectural concepts and demonstrates the ability of a NoC based SNN platform to successfully evolve solutions to a range of non-linear and unstable tasks, benchmark applications. The paper considers the impact of (1) latency jitter noise introduced by the NoC router and (2) the low resolution EMBRACE-FPGA processor-based neuron/synapse model on SNN accuracy and evolution time. An on-chip fitness function which improves intrinsic hardware evolution time is described. The paper discusses how a NoC based SNN supports the ability to repair faults in neuron elements or NoC paths through on-line dynamic reconfiguration via NoC configuration packets. The reconfigurable EMBRACE architecture enables future investigation of adaptive hardware applications, fault detection and reconfiguration-based self repair in evolvable hardware.

This work contributes to the development of the EMBRACE mixed-signal NoC-based embedded hardware SNN device and supporting tool suite. EMBRACE-FPGA validates the EMBRACE architectural concepts and demonstrates the ability of the platform to successfully evolve solutions to a range of benchmark applications.

Acknowledgements

This research is supported by Science Foundation Ireland under Grant No. 07/SRCI1169 and the Irish Research Council for Science, Engineering and Technology (IRCSET), International Centre for Graduate Education in Micro and Nano Engineering (ICGEE). The authors also thank the Xilinx University Programme.
References


Chapter 3

The Impact of Neural Model Resolution on Hardware Spiking Neural Network Behaviour

Abstract

This paper contributes to the development of the proposed EMBRACE mixed-signal, reconfigurable, Network-on-Chip based hardware Spiking Neural Network. EMBRACE-FPGA is an FPGA-based prototype of the proposed EMBRACE architecture. Results from successful evolution of an EMBRACE-FPGA SNN robotics controller are presented. Noise in best fitness plots for a range of evolved EMBRACE-FPGA based SNN applications, including the robotics controller, have been observed. This paper investigates the sources of neural noise, and considers their impact in evolving digital-based hardware SNNs. The paper considers the expected performance benefits of the EMBRACE analogue neural cell.
3.1 Introduction

The basic processing units in organic central nervous systems are neurons which are highly interconnected in a complex pattern. Spiking neurons [1–3] emulate real biological neurons of the brain more closely than traditional artificial neural network models and therefore have the potential to be computationally more powerful [4]. Spiking neurons communicate by transmitting short transient spikes between neurons, via their synapses [1]. Neuron potential is a function of incoming spike trains, synaptic weights and time. A neuron generates an output spike when its potential exceeds its neuron firing threshold. Outputs from neurons consist of a series of spike pulses which may, in turn, be interpreted by other neurons or employed as outputs of the network. Readers are referred to [5] for a comprehensive review of Spiking Neural Network (SNN) operation.

The authors have proposed a reconfigurable, scalable, Network on Chip (NoC)-based, mixed-signal hardware SNN architecture (EMBRACE) [6, 7] which supports the implementation of large scale SNNs. NoC-based inter-neuron connectivity addresses the hardware interconnect resource challenge of large scale SNNs [7–10]. The NoC topology provides flexible, time multiplexed, packet-based, inter-neuron communication channels, scalable interconnect and reconfigurability.

The proposed EMBRACE architecture incorporates a compact, low power, CMOS-compatible analogue SNN neuron cell [6] which offers synaptic densities significantly in excess of that currently achievable in other SNN hardware approaches [11–14]. The EMBRACE architecture also includes a novel weight storage mechanism [6] to allow the digital configuration of analogue synaptic weights.

EMBRACE-FPGA [5] is an FPGA prototype of the proposed NoC-based mixed-signal EMBRACE architecture. EMBRACE-FPGA hardware SNN has been employed as part of a Genetic Algorithm (GA)-based intrinsic hardware SNN training and configuration platform (SNNDevSys) (Figure 3.1) to solve a benchmark XOR function, an inverted pendulum controller and a Breast Cancer (Wisconsin) dataset classifier [5, 15]. SNNDevSys (Figure 3.1) incorporates a GA for SNN evolution and an IO interface for SNN configuration and neural network data IO.

This paper presents an effective, robust, high fitness evolved hardware SNN robotics controller implemented on EMBRACE-FPGA, using a low resolution digital hardware neuron model. Noise has been observed in the best fitness plots for a range of evolved EMBRACE-FPGA based SNNs [15, 16] and in the robotics controller.
This paper investigates the sources of neural noise, and considers their impact in evolving digital-based hardware SNNs. The paper considers the expected performance benefits of the proposed EMBRACE analogue neural cell which implements a continuous neuron potential decay. The structure of the paper is as follows: Section 3.2 highlights related work in the hardware implementations of SNNs. Section 3.3 describes the EMBRACE-FPGA NoC based hardware SNN architecture. Section 3.4 describes SNNDevSys, the GA based training and configuration platform. Section 3.5 presents and discusses the evolution of a hardware SNN robotics controller. Section 3.6 discusses the impact of noise on the behaviour of the digital hardware neuron within EMBRACE-FPGA and presents a case for reduced noise using the proposed EMBRACE analogue neural cell. Section 3.7 concludes the paper.

3.2 Related Work

Related work on the implementation of hardware SNNs is summarised in [5]. This section reports more recent work and summarises EMBRACE related research. Hardware SNNs fall into three broad categories, namely software, analogue and FPGA, as follows:

Embedded software-based (multi-processor) [8, 17] SNN platforms feature high performance, pipelined processors and large memory resources which enable the simulation of large numbers of neurons. The embedded software-based SNN approach supports flexible neuron model design and the simulation of high accuracy neuron models. These systems are relatively expensive. The use of GPUs to simulate large SNNs has been reported in [18]. The parallel nature of GPUs allows many neurons to be simulated concurrently and with a high degree of accuracy. GPUs have been shown to be capable of simulating a large number of neurons at a lower cost than large multiprocessor based implementations.

Analogue SNNs [6, 14, 19–21] model the electrical properties of biological neurons to allow efficient, low area and low power, hardware neuron model implementation. Analogue design approaches enjoy the benefit of compact area implementations due to the inherent similarity with the way electrical charge flows in the brain. Inter-neuron connectivity and reconfigurability limits the number of neurons which can be realised in analogue hardware.

FPGA-based SNN systems [11–13, 22] offer the advantage of flexible design and parallel execution. FPGAs are also well suited for prototype implementation and testing of new architectures. However, with scaling SNN size, FPGAs suffer from a large connectivity overhead and limited FPGA resources. A recent FPGA-based embedded processor
SNN implementation [23] demonstrates that spike-time coding can perform classification of real world data (for the WBCD application) through the use of supervised and unsupervised learning.

A scalable, mixed signal, reconfigurable hardware SNN (EMBRACE) has been proposed in [6]. EMBRACE offers the potential of a small, low-power, scalable embedded system. The EMBRACE-FPGA prototyping platform contributes to the development of EMBRACE and enables investigation and validation of various NoC router designs. EMBRACE-FPGA is also used in the development of SNNDevSys as a platform for training and configuration for EMBRACE.

## 3.3 EMBRACE-FPGA Architecture

This section describes the EMBRACE-FPGA SNN NoC-based architecture and neural tile operation. Further details are included in [5].

The EMBRACE-FPGA architecture consists of a 2-dimensional array (Figure 3.1) of interconnected SNN neural tiles (Figure 3.2). Each tile contains a NoC router and a neural cell. The NoC router has 4 IO ports (North, South, East, West) which connect tiles together to form a nearest neighbour connection scheme. NoC-based inter-neuron connectivity addresses the interconnect resource challenge of large scale SNNs while meeting biological-scale spike transmission timings (in the order of 10 ms) [24]. The prototype EMBRACE-FPGA SNN, synthesised on Virtex II-Pro, supports 32 neurons and 32 synapses per neuron [5].

The NoC allows full or partial reconfiguration of the synaptic weights, neuron firing threshold and connection topology. Configuration memory, which can be updated during SNN runtime, stores network topology, synaptic weights and neuron potential firing threshold. The SNN neural cell behaviour is implemented using a PicoBlaze [25] and approximates the Leaky-Integrate-and-Fire (LIF) neuron behaviour. The NoC router supports three different packet types, namely spike, configuration and debug. Spike packets contain the destination address of the synapse/neuron pair. Configuration packets contain the destination neural tile address, configuration memory identifier and configuration data. Debug packets allow interrogation of configuration memory, neuron potential and NoC router status.

SNN input Spike Packet Generators (SGs) (Figure 3.1) within the hardware SNN encode neural network input values (received from the host) into spike trains (based on the configured spike generation rate). The SGs convert analogue values into NoC spike packets, applied as inputs to the SNN input layer neurons. The SNN monitor tile
contains Spike Counters (SC) and debug memory. SCs implement the SNN output layer neurons, counting the number of spikes received within a fixed time window and converting the output spike frequencies to integer values (to be read by the host as SNN output values).

Each EMBRACE-FPGA neural tile incorporates a PicoBlaze microcontroller to implement neuron behaviour. The neuron model could also be implemented as a dedicated HDL element, though PicoBlaze implementation has been selected to facilitate neural model programming. The PicoBlaze is a soft-core (described as a synthesisable VHDL
Chapter 3. The Impact of Neural Model Resolution on HW SNN Behaviour

Figure 3.2: EMBRACE-FPGA SNN Neural Tile (*:N,S,E,W)

model) 8-bit RISC microcontroller. The PicoBlaze supports 256 IO ports, 16 byte-wide registers, 64 bytes of scratchpad RAM and memory for 1024 instructions. The device is programmed in assembly and contains an Arithmetic Logic Unit (ALU) which performs basic maths operations. The PicoBlaze program receives and processes neuron configuration information via the NoC router. Synaptic weight and neuron potential threshold values are stored in scratchpad RAM for efficient access.

Spike packets received by the NoC router trigger an interrupt in the PicoBlaze. An Interrupt Service Routine (ISR) reads the synapse address from the spike packet and retrieves the appropriate weight value from RAM. This value is applied to the neuron potential. If the applied weight is positive the ISR determines if the potential has exceeded the neuron firing threshold. On firing the neuron triggers the NoC router to generate a spike packet to each connected synapse. Firing of the neuron also starts a refractory period timer and resets the neuron potential to zero. Normal operation of the PicoBlaze program implements neuron potential decay. Each neuron may also be configured to inform the monitor tile (Figure 3.1) of its current potential, which aids in debugging.
3.4 GA-Based Hardware SNN Training And Configuration Platform

This section describes SNNDevSys, the GA-based intrinsic hardware SNN training and configuration platform.

GAs are a type of Evolutionary Algorithm based on the mechanisms of adaptation observed in nature. Points in the solution space (synaptic weights and neuron firing thresholds) are encoded into a linear data string (genome). The real-valued genome is decoded into its corresponding binary-valued SNN configuration packet format. EMBRACE-FPGA supports 6-bit signed synaptic weight values and 16-bit neuron firing threshold values [6]. Intrinsic evolution involves implementing and evaluating in hardware each individual from an evolved population. This differs from extrinsic evolution which evaluates individuals by simulating each individual in software prior to selection of a fit individual for hardware configuration. During intrinsic SNN evolution, training inputs are applied to the hardware SNN, spike train output frequencies are recorded and a fitness evaluation of each individual is performed. The GA applies selection, crossover and mutation to create a new generation of possible SNN solutions. This evolutionary process continues until high fitness convergence is achieved.

3.5 Evolved Hardware SNN Robotics Controller

This section presents an effective, robust, high fitness evolved hardware SNN robotics controller implemented on EMBRACE-FPGA, using the low resolution digital hardware neuron model implemented on PicoBlaze.

A robotics simulator [26] is included within the SNNDevSys platform. The robot is equipped with four forward facing sonar sensors, used as inputs to the network. The robot includes two motors which are controlled by the outputs of the evolved SNN. EMBRACE-FPGA is configured to implement the neural network illustrated in Figure 3.3. SNNDevSys controls the interaction between EMBRACE-FPGA and the robot simulation environment. The GA randomly generates an initial population and each individual is configured in turn to the SNN.

Fitness assessment of the robotics controller solutions (for obstacle avoidance) is achieved using a fitness function that rewards individuals based on the following behavioural qualities:
1. Obstacle avoidance: survival time ($T$) within the environment (when an individual crashes into an obstacle it “dies”)

2. High speed; the distance ($D$) traversed by the robot in the allotted time

3. Effective exploration of the environment; i.e. the aggregate ($AWS$) of left ($aws_{1}$) and right ($aws_{2}$) average wheel speeds. This aggregate (minimum 1, a straight line) is inversely proportional to fitness

An individual evaluation time of 100 seconds has been arbitrarily selected. Testing terminates if the individual crashes (fails) during the testing period.

$$F = \frac{T + D\alpha}{AWS^\beta}$$  \hspace{1cm} (3.1)

Two constants ($\alpha$, $\beta$) are introduced to the fitness assessment. By varying the relative values of each of the constants, different solutions and behaviours are evolved by the GA. These include wall hugging or high speed control strategies within the evolved robots. Equation 3.1 describes the robotics fitness function formula.

The GA used to evolve these behaviours employs tournament selection and elitism as part of the evolution process. Incorporation of elitism in a GA retains the best individual from a population into the next generation. This is aimed at ensuring that the best fitness score of subsequent generations does not decrease. GA parameters are listed in Table 3.1. Figure 3.4 illustrates the best and average intrinsically evolved hardware SNN robotics controller fitness.

The noisy nature of the embedded neuron model is visible as a reduction in best fitness achieved between some generations. This can only occur when the best individual from a previous population, which is retained due to elitism, does not achieve the same fitness.
when re-evaluated. This reduction in fitness reduces the probability of retention of this individual between generations. As such individuals will be removed due to their sensitivity to noise the GA will evolve solutions which are more robust to noise inherent in the system. However, this behaviour increases the time required to successfully evolve a highly fit robotics controller. An individual which achieves a fitness of greater than 7,000 (Figure 3.4) is capable of successfully navigating the simulation world for 100 seconds.

The previously reported EMBRACE-FPGA SNN benchmark applications [15, 16] also exhibit an occasional reduction in best fitness scores between successive generations. While elitism is also employed in these applications the ability of SNNDevSys to evolve robust solution is demonstrated. The following section investigates the individual sources of noise and their impact on the EMBRACE-FPGA SNN.
3.6 Impact Of Neuron Model Resolution

Sources of noise in the EMBRACE-FPGA architecture include NoC latency jitter and neuron model noise. NoC latency jitter is inherent in the EMBRACE-FPGA architecture due to the NoC router arbitration scheme. The impact of NoC latency jitter is a small distortion of neuron input spike train frequency.

The neuron model used in EMBRACE-FPGA is implemented in an 8-bit PicoBlaze processor. The noise introduced by this coarse resolution neuron model is examined in this section. Results have been obtained through simulation of an RTL model of the EMBRACE-FPGA hardware neuron. The behaviour of this model is compared with that of an ideal high resolution software simulation model. The proposed EMBRACE architecture will incorporate an analogue neural cell with behaviour similar to the software model described in this section.

3.6.1 Low Resolution Neuron Model Noise

Noise in the operation of the EMBRACE-FPGA SNN due to the neuron model is caused by the:

- Low resolution of neuron potential decay
- Latency of ISR driven input spike processing
- Amplification of the effect of NoC jitter variation
The low resolution of the neuron potential decay calculations is illustrated in Figure 3.5. In this example, for a constant input spike frequency, the internal neuron potential reaches a steady state. This is achieved when neuron potential decay rate equals the rate of neuron potential increase due to incoming spike packets. The low resolution of the neuron potential calculations is apparent in the stepwise nature of the plot. This has the effect of reducing the resolution of possible neuron output spike train frequencies. This results in a low output resolution of the evolved SNN. Note that the neuron potential illustrated in the example of Figure 3.6 does not reach the neuron firing threshold and hence no output spike is generated.

The PicoBlaze-based neuron model implementation processes neuron input spikes through the use of interrupts. The processing of the ISR, and associated context switching performed by the processor, have the effect of distorting normal neuron decay operations and have an impact on the output spike frequency.

Variations in NoC spike packet latency, caused by the NoC router arbitration scheme, distorts input spike train frequencies. However, due to the coarse resolution of the neuron potential decay calculations this small variation in spike packet arrival time can have a large impact in neuron potential change. The Inter Spike Interval (ISI) or time between each successive input spike packet (Figure 3.5) is 283 clock cycles. The NoC latency jitter introduced by the EMBRACE-FPGA NoC router ranges between 0 and 8 clock cycles (a maximum distortion of 2.8% of the original frequency). Figure 3.6 illustrates neuron model behaviour for the same input spike train as Figure 3.5 though with the addition of NoC latency jitter. The neuron potential is noisier and can induce variation from the intended neuron output spike frequency.

3.6.2 High Resolution Neuron Model

The neuron model used by the proposed EMBRACE SNN architecture is implemented as a CMOS compatible analogue neural cell. Due to the analogue nature of the device, the neural cell may be considered to perform calculations with infinite resolution. This neural cell does not suffer from the limitations of the EMBRACE-FPGA digital neuron model related to neuron output spike train frequency resolution and neuron potential decay rate distortion.

Figure 3.7 replicates Figure 3.5 for a software simulation of an analogue neuron model. Figure 3.8 replicates Figure 3.6 to illustrate the effect of NoC latency jitter on the internal neuron potential. The continuous nature of the potential decay performed by an analogue neuron model results in a greatly reduced variation of neuron potential caused by NoC latency jitter.
Noise in best fitness plots for a range of evolved EMBRACE-FPGA based SNNs [15, 16], including the robotics controller has been observed. This is due to the noise introduced by a low resolution neuron model. While this noisy operation is a disadvantage in SNN operation it is mitigated by the ability of SNNs to generalise. The use of a GA to evolve SNN configurations also means that individuals which are overly sensitive to this noise are discarded over generations in favour of more robust individuals.

3.7 Conclusions

The paper presented an effective, robust, high fitness evolved hardware SNN robotics controller implemented on EMBRACE-FPGA, using a low resolution digital hardware neuron model. Analysis of noise in best GA fitness plots for a range of evolved EMBRACE-FPGA based SNNs, including the robotics controller has been presented.

This paper investigated the sources of neural noise, and considered its impact in evolving digital-based hardware SNNs. Moreover, the paper demonstrated the improved behaviour presented by the EMBRACE analogue neural cell which implements a continuous neuron potential decay.
Chapter 3. The Impact of Neural Model Resolution on HW SNN Behaviour

References


Chapter 3. *The Impact of Neural Model Resolution on HW SNN Behaviour*


Chapter 4

Memory Efficient Storage of Reconfigurable Topology Information in Network-on-Chip Based Spiking Neural Networks

Abstract

Interconnect scalability for embedded hardware Spiking Neural Network (SNN) implementations remains an obstacle to their large scale adoption. Recently Network-on-Chip (NoC) concepts have been explored as an approach to provide high density inter-neuron connectivity in a scalable and reconfigurable manner. While NoC interconnect removes the area cost associated with directly wired inter-neuron synaptic connections, the memory associated with storing inter-neuron connections in the network (topology) becomes the limiting factor in hardware SNN size.

This paper presents a novel Single Dynamic Synapse (SDS) method and source-based neural network topology and synaptic weight storage, to substantially reduce memory requirements for a large scale NoC-based embedded hardware SNN architecture. Rather than maintaining multiple individually addressable and configurable synapses, each SNN neuron implements a single dynamically reconfigurable synapse, where run-time weight configuration information is transmitted within the spike packet. The SDS architecture supports large synaptic connection density per neuron, limited only by the topology and synaptic weight memory size. The SDS architecture has been implemented both in a SystemC based modelling environment and in FPGA hardware. Application testing
confirms projected memory reduction and verifies that the approach has no impact on SNN performance.

4.1 Introduction

Artificial Neural Networks (ANNs) have attracted much attention recently due to a growing interest in the automatic design of complex non-linear systems. The basic processing units in organic central nervous systems are neurons, interconnected in a complex pattern using synapses [1]. ANNs attempt to model highly interconnected neural systems to replicate the brains ability, in an area and power efficient manner, to perform complex computation. The current understanding of biological neurons is that they communicate through pulses and employ the relative timing of the pulses to transmit information and perform computations.

Spiking Neural Networks (SNNs) [1–3], which are third generation ANNs, emulate real biological neurons more closely than traditional ANN models and therefore have the potential to be computationally more powerful [2]. SNNs offer the potential of an efficient, low-power and robust method of performing computing. SNNs are suitable for implementing control and classification applications due to their ability to provide a good solution in the presence of imprecise or unseen data.

SNNs communicate by transmitting short transient pulses or spikes between neurons, via weighted synaptic connections. Synapses can have either an excitatory or inhibitory effect on a neuron’s internal membrane potential. A spiking neuron emits a spike when its potential exceeds a neuron-specific membrane potential threshold value. The computational power of SNNs is realised by the topology of synaptic connections between neurons, the weights on these connections and the internal membrane potential threshold of each neuron.

Researchers aim to implement reconfigurable and highly interconnected arrays of neural network elements in hardware to produce powerful signal processing units [4–12]. High levels of inter-neuron connectivity and the associated area of hardware interconnect [13] is the current major limiting factor in hardware SNN architectures. Often the number of neurons that can be realised in hardware is limited by high fan-in and fan-out requirements [13]. Direct neuron-to-neuron interconnection exhibits switching requirements that grow non-linearly with the network mesh size [13], e.g. a 2-layered fully interconnected, network with $m$ neurons per layer exhibits a interconnect density of $m^2$.

A number of different approaches have been investigated as solutions to the hardware interconnect scalability problem including multicast Network-on-Chip (NoC) routing [14]
and wafer-scale integration \cite{15}. The use of NoC concepts has proven to be an effective approach to implementing highly interconnected neurons in an area and power efficient manner. Additionally, the use of a NoC provides a mechanism to configure neurons, enabling run-time reconfigurable systems. However, while a NoC allows efficient transmission of spike information and reduces hardware interconnect requirements, these advantages are offset by the significant hardware resources required to store the synaptic address connections (topology) between neurons. For large scale, highly interconnected networks this neuron and synapse address memory requirement quickly dominates overall hardware area, far exceeding the hardware requirements of the neuron computation engines and NoC interconnect. Biological neurons maintain an average of 1000 synaptic connections \cite{16}. The paper derives a topology memory requirements model for the traditional Multi-Synapse (MS) NoC-based hardware SNN architecture, used as reference in this paper.

The aim of this research is the development of a large scale NoC-based embedded hardware SNN architecture. This paper presents a novel Single Dynamic Synapse (SDS) inter-neuron connection architecture and source-based neural network topology and synaptic weight storage, to substantially reduce the memory requirements in a large scale NoC-based embedded hardware SNN. Each SNN neuron implements a single dynamically reconfigurable synapse, where run-time weight configuration information is transmitted within each spike packet generated by the NoC. This single shared dynamic synapse removes the requirement to include many individually addressable and configurable physical synaptic connections in the hardware SNN. Traditional multi-synapse based approaches require additional logic for every additional synapse and additional memory to support the synapse.

The paper derives a topology memory requirements model for the proposed SDS architecture which highlights an improved relationship between SNN topology memory requirements and SNN size. The SDS architecture supports a large synaptic connection density per neuron, limited only by the topology and synaptic weight memory size. Results illustrate that use of the SDS approach reduces the topology and synaptic weight memory, for hardware SNNs having more connections per neuron than synapses per neuron.

The SDS architecture has been implemented in a SystemC based modelling environment and in FPGA hardware. Application testing confirms projected memory reduction and verifies that the approach has no impact on SNN performance.

The structure of the paper is as follows. Section 4.2 summarises related work in the area of large scale SNN architectures. Section 4.3 describes the proposed SDS and MS approaches to SNN topology and synaptic weight storage and derives memory requirements
dependencies. Section 4.4 presents and discusses results which highlight the benefits of the SDS approach over the reference MS SNN architecture. Section 4.5 concludes the paper and proposes future work.

4.2 Large Scale Hardware SNN Architectures

The aim of this research is the development of a large scale NoC-based embedded hardware SNN architecture. This section highlights existing research in the area of SNN architectures, particularly scalable, NoC-based hardware SNN architectures. Inspired by biology, researchers aim to implement reconfigurable and highly interconnected arrays of neural network elements in hardware to produce powerful signal processing units [4–11]. A typical NoC-based SNN architecture consists of an array of NoC routers, each of which incorporates a spiking neuron, associated synapses and connectivity or topology memory. Execution architectures for SNN neural computing platforms can be broadly categorised as processor-based [10, 12], FPGA [5, 6, 8, 17] or analogue/mixed-signal [7, 9, 11, 15, 18].
FPGA-based architectures offer high flexibility for system design. Pearson et al. [5] describe an FPGA-based (SIMD) array processor architecture, which can simulate networks containing over 1,000 neurons. However the architecture can be classified as a single instruction path, multiple data path (SIMD) array processor. The architecture uses a bus-based communication protocol which limits the scalability of the architecture. Ros et al. [6] present an FPGA-based hybrid computing platform, where the neuron model is implemented in hardware and the network model and learning are implemented in software.

The SpiNNaker project [10] aims to develop a massively parallel computer capable of simulating SNNs of various sizes and topologies. The architecture uses ARM-968 processor-based nodes for neural computation and an off-chip network communication infrastructure. Software implementation of the neuron model provides flexibility for the SNN computation model. SpiNNaker aims to explore the potential of the spiking neuron as a component from which useful systems may be engineered. The SpiNNaker SNN architecture does not target embedded systems applications due to its size, cost and power requirements.

The authors have previously reported EMBRACE-FPGA [19], a Multi-Synapse NoC-based SNN architecture implemented on FPGA hardware. EMBRACE-FPGA consists of an array of NoC routers each combined with SNN topology memory, a spiking neuron and weighted synaptic connections. EMBRACE-FPGA has been developed as a prototype of EMBRACE [9], a proposed mixed signal, NoC-based embedded SNN. This paper uses EMBRACE-FPGA as an archetypal NoC-based SNN, although the proposed approach can be applied in other NoC-based SNN architectures.

4.3 SDS and MS Model Description and Memory Dependencies

This section describes the proposed SDS and MS approaches to SNN topology and synaptic weight storage and derives memory requirements dependencies. Section 4.3.1 describes the functionality and memory structure of the the MS SNN architecture. A memory requirements model is derived which highlights the non-linear growth in memory size with respect to SNN network size. Section 4.3.2 describes the proposed SDS architecture, and derives an associated memory requirements model which highlights the slower growth in memory size with increasing SNN network size.
Chapter 4. Memory Efficient Storage of Reconfigurable Topology Information in NoC Based SNNs

4.3.1 Multi Synapse (MS) Architecture Topology Memory Dependencies

The MS SNN architecture includes an array of NoC routers connected in a 2-dimensional grid. Each NoC router is paired with a spiking neuron, a number of physical synapses (each of which stores its weight locally) and neuron connectivity memory (topology memory) to create a Neural Tile (NT). Each NT maintains a list of neuron/synapse pairs to which a spike is transmitted when a neuron fires.

Using a NoC communication architecture, physical synaptic connections are replaced
with ‘virtual’ connections, supported by the NoC, reducing wiring density dramatically [9, 19]. However the memory required to store these connections (SNN topology) is significant. Topology memory accounts for approx 60% of the EMBRACE-FPGA architecture area [19] and severely limits scalability.

The size of the topology memory of a NoC-based hardware SNN is a function of the number of synaptic connections. As the number of synaptic connections increases so too does the memory requirement. Traditionally a unicast NoC-based SNN stores destination neuron and synapse addresses in the source NT topology memory and includes this information in each spike packet. On receipt of the packet the destination router applies a spike to the addressed synapse, modifying the neuron potential by the locally stored synaptic weight value.

Figure 4.1(a) illustrates the various memory blocks present in an MS-type NoC-based SNN architecture. The sequence of events from the firing of a source neuron to the application of a spike on a destination synapse is illustrated in steps a-d (Figure 4.1(a)).

(a) Source neuron fires
(b) First destination neuron/synapse address is retrieved from memory
(c) Spike packet is assembled and transmitted by the NoC to the destination router
(d) Spike is applied to the addressed synapse (the synapse weight is configured during the SNN configuration process)

Steps b-d are repeated for each synaptic connection. Equation 4.1 describes the MS architecture memory size ($M_{total}$) dependency on the SNN parameters, where:

$N$ is the number of neurons in the network

$C$ is the maximum number of synaptic connections which each neuron is permitted to make

$W$ is the number of bits used to store each synaptic weight value

$S$ is the number of physical synapses per neuron

Equation 4.1 ignores the smaller contribution of neuron firing threshold values memory.

(Note: ⌈$X$⌉ denotes the ceiling of $X$, i.e. next integer number larger than $X$).

$$M_{total} = (\lceil \log_2(N) \rceil + \lceil \log_2(S) \rceil) \times C + W \times S \times N \quad (4.1)$$

As the total memory needed for the network includes a reliance on $S$, the total number of synapses in the network, $M_{total}$ increases significantly as the network size grows.
4.3.2 Shared Dynamic Synapse (SDS) Architecture Topology Memory Dependencies

This section describes the proposed SDS architecture, and derives an associated memory requirements model which highlights the reduced growth in memory size with increasing SNN network size. The SDS architecture supports an unlimited number of incoming time multiplexed ‘virtual’ synaptic connections per neuron, the limit being determined by the size of the topology memory. The source neuron stores a neuron address and weight pair for each output synaptic connection instead of a neuron and synapse address pair. The proposed alternative approach exploits the fact that the frequency of spikes in an SNN is relatively low, in the order of 10Hz [3].

Figure 4.1(b) illustrates the proposed memory layout used in the SDS approach. The source neuron stores the weight of each connected synapse, rather than the address of the physical synapse. The destination neuron implements a single, dynamically reconfigurable synapse which takes its weight information from incoming spike packets. The sequence of events from the firing of a source neuron to the application of a spike on a destination synapse is illustrated in steps a-e (also illustrated in Figure 4.1(b)):

(a) Source neuron fires
(b) First destination neuron address and synaptic weight is retrieved from memory
(c) Spike packet is assembled and transmitted by the NoC to the destination router
(d) Synaptic weight is extracted from the packet and applied to the dynamically configurable synapse
(e) Spike is applied to the synapse

Equation 4.2 describes the SDS architecture memory size ($M_{total}$) dependency on the SNN parameters.
(Note: $\lceil X \rceil$ denotes the ceiling of $X$, i.e. next integer number larger than $X$).

$$M_{total} = (\lceil \log_2(N) \rceil + W) * C + W) * N \quad (4.2)$$

The SDS approach removes the dependency on $S$, the number of physical synapses (reduced to one physical synapse per neuron) and results in memory growth that scales significantly better than the MS approach. This results in dramatically reduced memory requirements for a large neural network which includes many synaptic connections.
Chapter 4. Memory Efficient Storage of Reconfigurable Topology Information in NoC Based SNNs

EMBRACE-FPGA | SDS
---|---
Neurons ($N$) | 1024 | 1024
Connections ($C$) | 256 | 256
Synapses ($S$) | 256 | -
Weight Bits ($W$) | 5 | 5
Total ($M_{total}$) | 5.89 Mbits | 3.84 Mbits

Table 4.1: Example Memory Requirements for EMBRACE-FPGA & SDS

4.4 Dynamically Reconfigurable Synapse

This section presents and discusses results which highlight the benefits of the SDS approach over the reference MS SNN architecture. Figure 4.2(a) compares the memory requirement with varying input synaptic connection density for the proposed SDS approach and the MS approach. As SDS memory requirements do not have a dependency on the number of available input synaptic connections to a neuron memory remains constant. This scenario allows flexible connectivity between neurons at no additional memory cost. Figure 4.2(b) illustrates memory requirements as a function of network size with a fixed level of inter-neuron connectivity for both methods. Due to efficiencies introduced through the use of SDS memory usage increases slower with growing network size compared to a MS approach.

For example, for a sample 1024 neuron SNN, supporting 256 input and output synaptic connections per neuron, and 5-bit synaptic weight, results illustrate (Table 4.1) a 35% saving in SNN topology memory using the SDS approach, compared to the topology memory incorporated in the traditional multi-synapse architecture. The relative memory savings increase as SNN size increases.

The SDS architecture has been implemented both in an existing SystemC based modelling environment and FPGA hardware. Application testing has confirmed projected memory reduction and verified the approach has no impact on SNN performance.

In the proposed SDS approach, information relating to a neuron and its synaptic weights is distributed between source and destination neural tiles. Hence, during online SNN learning and online topology reconfiguration, the process of modifying connections and synaptic weights requires additional processing. Also configuration of the synaptic weight is required for each incoming spike. This introduces a low level of additional overhead logic within the hardware neuron.
4.5 Conclusions and Future Work

This paper presents a novel Single Dynamic Synapse (SDS) method and source-based neural network topology and synaptic weight storage, to substantially reduce memory requirements for a large scale NoC-based embedded hardware SNN architecture. Rather than maintaining multiple individually addressable and configurable synapses, each SNN neuron implements a single dynamically reconfigurable synapse, where run-time weight configuration information is transmitted within the spike packet.

The proposed method is compared with the traditional approach, represented here by the prototype NoC-based EMBRACE-FPGA hardware SNN architecture. The ability of the proposed method to support SNNs with many (>1000) neurons is investigated, with memory requirements for a large scale network calculated. The paper describes the relative saving in memory and identifies cases in which the proposed approach may not be beneficial. The SDS architecture has been implemented both in an existing SystemC based modelling environment and FPGA hardware. Application testing has confirmed projected memory reduction and verified the approach has no impact on SNN performance.

Future work will include a large scale implementation of the proposed method in FPGA hardware, both to further confirm the memory savings and perform analysis using benchmark applications to characterise any performance changes the additional logic may have on large SNN performance.

Acknowledgments

This research is supported by Science Foundation Ireland under Grant No. 07/SRC/C/11169 and the Irish Research Council for Science, Engineering and Technology (IRC-SET), International Centre for Graduate Education in Micro and Nano Engineering (ICGEE). The authors also thank the Xilinx University Programme.
Chapter 4. Memory Efficient Storage of Reconfigurable Topology Information in NoC Based SNNs

References


Chapter 5

Evolving SNN Topologies for UWB based Breast Cancer Detection

Abstract

Several studies have investigated the possibility of using the Radar Target Signature (RTS) of a tumour to classify the tumour as either benign or malignant, since the RTS has been shown to be influenced by the size, shape and surface texture of tumours. The Evolved-Topology SNN presented here extends the use of evolutionary algorithms to determine an optimal number of neurons and interneuron connections, forming a robust and accurate Ultra Wideband Radar (UWB) breast cancer classifier. The classifier is examined using dielectrically realistic numerical breast models, and the performance of the classifier is compared to an existing Fixed-Topology SNN cancer classifier.
5.1 Introduction

In the United States alone, breast cancer accounts for 31% of new cancer cases, and is second only to lung cancer as the leading cause of deaths in American women [1]. The current standard screening method for detecting non-palpable early stage breast cancer is X-ray mammography. Despite the fact that X-ray mammography provides high resolution images using relatively low radiation doses, its limitations are well documented [2]. The search for new imaging techniques is motivated by the need for increased specificity and sensitivity, especially in the case of radiographically dense tissue.

One of the most promising emerging breast imaging modalities is UWB Radar imaging. The physical basis of UWB Radar imaging is the dielectric contrast between normal and malignant breast tissue that exists at microwave frequencies[3–8].

This dielectric contrast is due to the increased water content present in the cancerous tissue, and this contrast ensures that when the breast is illuminated by an Ultra Wideband (UWB) pulse, cancerous tissue in the breast tissue will provide backscattered energy, which may be used to detect, localise and classify tumours. UWB Radar imaging is non-ionising, non-invasive, does not require uncomfortable breast compression, and is potentially lower cost.

Several studies have also examined the use of UWB Radar to classify breast cancer. This classification approach is based on the Radar Target Signature, which reflects the size, shape and surface texture of the tumour. Benign tumours typically have smooth surfaces and have spherical, oval or at least well-circumscribed contours. Conversely, malignant tumours usually present rough and complex surfaces with spicules or microlobules, and their shapes are typically irregular, ill-defined and asymmetric [9]. These tumour characteristics are generally reflected in the details of the RTS and can be used in classifiers. Several classifiers and classification architectures have been investigated [10–15].

In this paper, a novel Spiking Neural Network (SNN) classifier is presented and examined. SNNs [16] emulate biological neurons and aim to replicate the brains ability to function well when presented with noisy or incomplete data. SNNs are typically trained for a specific task using a Genetic Algorithm (GA), a type of evolutionary algorithm. This training process involves modifying neuron firing thresholds and synaptic weights. This paper extends the use of evolutionary algorithms to determine an optimal number of neurons and interneuron connections to form a UWB breast cancer classifier. This approach results in networks which are more compact than traditional fixed topology networks and simplifies the search space resulting in faster training time and increased accuracy.
Chapter 5. *Evolving SNN Topologies for UWB based Breast Cancer Detection* 

The structure of the paper is as follows: Section 5.2 describes the SNNs and the NEAT Genetic Algorithm used for SNN training; Section 5.3 describes the generation of realistic tumour models, including dielectric heterogeneity and corresponding FDTD simulations; while Section 5.4 describes the results and Section 5.5 draws the corresponding conclusions.

### 5.2 Spiking Neural Networks and Evolving Topologies

#### 5.2.1 Spiking Neural Network

Spiking Neural Networks (SNNs) are more closely related to their biological counterparts than previous Artificial Neural Networks (ANNs) generations, such as multi-layer perceptrons. SNNs, in contrast to previous models, employ transient pulses for communication and computation. Maass has demonstrated that spiking neurons are more computationally powerful than threshold-based neuron models [16] and that SNNs possess similar and often more computation ability compared to multi-layer perceptrons [17].

Inspired by nature, a Genetic Algorithm (GA) [18] models natural evolution through a set of computational operators. A GA is a parallel, population-based search strategy that encodes individual solutions into a data-structure known as a genome. A population of such genomes is maintained by the GA and mechanisms analogous to evolution are employed to evolve high-fitness solutions. Exploration of the search space is performed using a diversity introducing mutation operator while crossover (mating of two parent solutions) is employed to exploit good solution building blocks (known as genes) already in the population. Selection pressure is added through a tournament selection operator to incorporate “survival of the fittest”. Traditionally SNN simulations involve constructing a fixed, regular structure of neurons arranged in layers where all neurons are fully-forward connected (Figure 5.1). This approach simplifies the design of the network and provides a structure whose neuron firing thresholds and synaptic interconnect weights may be evolved, with a fixed-length genome, to form a solution. Recently, there has been growing interest in exploiting the adaptability provided by GAs to modify the interconnect structure of an SNN in order to create a topology which is both simpler and more suitable for the task at hand. A common issue with this concept is designing an appropriate encoding mechanism for the structure of the network such that it may be mutated and combined with other networks in a feasible manner. Additionally network structures evolved by a GA have a tendency to grow as the GA progresses. This topology growth results in a more complex search space partly negating the advantage of evolving a task specific network.
The authors have chosen to incorporate the NeuroEvolution through Augmenting Topologies (NEAT) [19] algorithm which is tailored to address these particular concerns. The NEAT algorithm incorporates historical markers in the SNN gene which allows genes with common ancestors to be combined as part of the GA’s crossover mechanism. NEAT also uses this historical information to group individuals into species based on common ancestors [19]. When the GA creates a new generation, selection of individuals (i.e. choosing which individuals will be combined together to form an individual for the new population) is traditionally based on each individual’s fitness. NEAT implements explicit fitness sharing [20] within species, where individuals in a species must share their combined fitness (i.e. the fitness of an individual is modified to be the average fitness of all individuals in that species). This deters species from growing too large, as each individual must contribute to the species fitness, hence allowing many diverse species (i.e. many unique approaches to solving the problem) to co-exist. Species whose fitness does not increase over a number of generations become extinct (i.e. the individuals are deleted from the population and replaced with new, randomly-initialised, individuals) ensuring individuals continue to improve as the network complexity grows. Traditional fully connected SNNs will contain many neurons and connections which do not contribute to classifier accuracy. The NEAT operators create networks which are of optimal size and only contain neurons and connections which aid the function of the classifier.

5.2.2 Preprocessing and Fitness Function

The classifier considered here is a two-class problem (i.e. malignant vs. benign). Principal Component Analysis is applied to extract the most significant classification features of the RTS. In this study, high PCA values are mapped to high spike frequencies while low PCA values are mapped to low frequencies. Since PCA values are scaled between [-1, +1], it is necessary to decouple the positive and negative ranges of each PCA component (P(n)) into two spike generating inputs (P(n)⁺ and P(n)⁻). This decoupling
ensures that a +1 PCA input generates the same number of spikes (and influence) on the SNN as a -1 PCA input, thus removing any bias from the encoding process. The SNN processes the first 15 PCA components (P(1)-P(15)). Thirty spike generators are used to map real-valued PCA data into spike trains using a linear magnitude to (spike train) frequency conversion [21].

Two output layer spiking neurons generate two spike trains, which are processed by two spike counters to count the number of output spikes within a given update interval [21]. These counter values are used to determine classifier behavior. The counter with the largest spike count value designates the selected class. The neuron model chosen for these experiments is based on the leaky integrate and fire model [16]. Each SNN individual is initially composed of thirty input neurons and two output neurons. The NEAT GA progressively adds neurons and connections and hence each individual has a variable number of genes. The GA also modifies the weights on the synaptic connections and neuron firing threshold.

Synaptic weights range from [-1, 1] while thresholds vary between [0, 5.0] [22]. Fitness assessment of the SNN-based breast cancer classifier is achieved using a fitness function, which rewards individuals based on the number of correct classifications made. $C_m$ refers to the number of correct malignant classifications made by the SNN. $C_b$ refers to the number of correct benign classifications. $C_{max}$ and $C_{min}$ are defined in Equations (5.2) and (5.3). The fitness function, $f$, of the SNN is defined as follows:

$$f = C_{min}^\beta + C_{max} \quad (5.1)$$

where

$$C_{max} = \max(C_m, C_b) \quad (5.2)$$

$$C_{min} = \min(C_m, C_b) \quad (5.3)$$

A $\beta$ value of 1.6, chosen through empirical analysis, is employed in this research to reward the correct classification of both tumour classes. Without this fitness bias, fitness can be accumulated by classifying a single tumour class repeatedly. By including a $\beta$ value greater than one, networks that select correctly from both classes are rewarded above networks that correctly select from just one class.
5.3 Breast and Tumour Modeling

Shape and texture of the surface of a tumour are two of the most important characteristics used to differentiate between a benign and a malignant tumour. The tumour models used in this paper are based on the Gaussian Random Spheres (GRSs) method [23, 24]. Three different tumour models at two different sizes are considered in this paper. Malignant tumours are represented by spiculated and microlobulated GRSs, whereas benign tumours are modelled by smooth GRSs. Microlobulated and smooth GRSs are obtained by varying the correlation angle from low to high. Spiculated GRSs are obtained by adding 3, 5 or 10 spicules to smooth GRSs. The average radius of all types of spheres are 2.5 and 7.5 mm. Between all sizes and shapes, the number of tumour models developed was 160 (80 malignant and 80 benign). Two sets of simulations were performed to examine performance in heterogenous tissue. For the first set (Hetero I), a single piece of fibroglandular tissue is added to the FDTD models, positioned at one of ten random locations surrounding the tumour. For the second set of simulations (Hetero II), two independent portions of fibroglandular tissue are positioned at two of ten random locations around the tumour. Portions of fibroglandular tissue were extracted from the UCWEM Breast Phantom Repository (phantom ID 071904). The background material is assumed to be homogeneous adipose tissue. In order to model loss and dispersion, the dielectric properties of adipose, fibroglandular and cancerous tissue are modelled using Debye parameters based on the dielectric data as published by Lazebnik et al. [7, 8].

The tumours (80 of size 2.5mm and 80 of size 7.5mm) are placed in a 3D Finite-Difference Time-Domain (FDTD) model. The FDTD model has a 0.5 mm cubic grid resolution and the backscattered signals were generated through a Total-Field/Scattered-Field (TF/SF) structure, in which the tumours and fibroglandular tissue are completely embedded in the Total Field (TF) [13, 15]. A pulsed plane wave is transmitted towards the target from four different equidistant angles (0°, 90°, 180°, 270°) and the resulting cross-polarized backscatter is recorded and analysed from four observation points located at: (0,0,-74), (-74,0,0), (0,0,74) and (74,0,0) mm in (x,y,z) axes. The incident pulse is a modulated Gaussian pulse with center frequency at 6 GHz where the 1/e full temporal width of the Gaussian envelope was 160 ps. A more detailed description of both the tumours and the model is presented in [25].
Table 5.1: Comparison of Fixed and Evolved-Topology SNN classifiers

<table>
<thead>
<tr>
<th>Classifier</th>
<th>One-Stage Type (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-Topology SNN</td>
<td>73</td>
</tr>
<tr>
<td>Evolved-Topology SNN</td>
<td>83</td>
</tr>
</tbody>
</table>

5.4 Results

In this study, a direct “type” classifier that simply classifies each tumour as either benign or malignant is considered. The tumour backscatter is classified using the Evolved-Topology SNN presented here, but also using a Fixed-Topology SNN previously presented by the authors [25], providing a useful baseline when examining the performance and robustness of the Evolved-Topology SNN classifier. In order to evaluate both classification methods, the entire data-set is randomly shuffled and divided into 75% (120 Tumours) and 25% (40 Tumours) training and test groups respectively. The classification process is repeated 10 times and the average performance of each classifier is calculated. The results are presented in Table 1 and illustrate a 10% increase in accuracy for the Evolved-Topology SNN compared to the traditional fixed topology SNN.

5.4.1 Effects of Dielectric Heterogeneity

In order to examine the effect of increasing dielectric heterogeneity on the performance of the SNN classifier, two specific scenarios are considered. In the first instance, a single piece of fibroglandular tissue surrounds the tumour, while in the second more difficult scenario two separate pieces of fibroglandular tissue are located around the tumour. The performance of the classifier in an increasingly heterogeneous environment is shown in Table 2. The performance of the classifiers drops by 10% and 7.4% for one-stage type for the traditional and Evolved-Topology SNN respectively as heterogeneity increases. Overall, the Evolved-Topology SNN classifier is shown to be relatively robust to significant increases in dielectric heterogeneity. In fact, in the most dielectrically heterogeneous models (Hetero II), the average performance (across large and small tumours) of the Evolved-Topology SNN classifier was almost 80%.
Table 5.2: Effects of dielectric heterogeneity on performance of SNN classifiers. Hetero I refers to models containing one piece of fibroglandular tissue, while Hetero II refers to models with two pieces of fibroglandular tissue.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Hetero I (%)</th>
<th>Hetero II (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-Topology SNN</td>
<td>78</td>
<td>68</td>
</tr>
<tr>
<td>Evolved-Topology SNN</td>
<td>86.75</td>
<td>79.4</td>
</tr>
</tbody>
</table>

5.5 Conclusions

The performance of an Evolved-Topology SNN based classifier (based on the NEAT algorithm) in a dielectrically heterogeneous breast was examined in this paper and compared to the performance of a traditional fixed topology SNN. Results demonstrate the ability of Evolved-Topology SNN classifiers to outperform a traditional SNN classifier. The improved classification performance can be attributed to the following:

- Specialisation within the GA population allows differing approaches to solving the task to evolve in parallel, effectively protecting potential innovative network structures and forcing search within many solution spaces to proceed in parallel.

- Historical markers inserted into each individual’s genes by NEAT allows divergent networks to be combined in an intelligent manner. This feature allows different networks which have, for example, evolved to perform well on a particular size of tumor to be combined in a manner which preserves and combines each networks specialisation.

- Finally, as NEAT promotes minimal networks, the search space remains small allowing for a greater exploration of the search space.

5.6 Acknowledgments

This work is supported by Science Foundation Ireland (SFI) under grant numbers 07/RF-P/ENEF420 and 07/SRC/I1169 and the Irish Research Council for Science, Engineering and Technology (IRCSET).
References


Chapter 6

Enhancing Learning of Digital Systems using a Remote FPGA Lab

Abstract

Learning in digital systems design and reconfigurable computing can be enhanced through applying a learn-by-doing approach on practical hardware systems. This paper presents the web-based RemoteFPGA lab which enables users to interact with a range of demonstrator digital hardware systems, operating in real time on an FPGA. The RemoteFPGA lab provides interactive control of system inputs, and monitoring of signals at any level of the design hierarchy. Users can also integrate their own HDL design descriptions within a RemoteFPGA HDL-based project template, for synthesis and implementation on the RemoteFPGA. Users can create a system block diagram for upload to the RemoteFPGA server. Interactive control and monitor signal icons can be overlayed on the block diagram to provide real-time demonstrations of the user designs. The RemoteFPGA lab provides enhanced visualisation and interaction with FPGA hardware compared to other reported remote FPGA laboratory systems. The paper describes the RemoteFPGA lab elements and demonstrates its use to support learning using two application case studies for illustration.
6.1 Introduction

Learning in digital systems design and reconfigurable computing can be enhanced through applying a learn-by-doing approach on practical hardware systems. Previous research has highlighted many benefits of remote FPGA laboratories including improved learning outcomes, reduced costs, facilitating inter-institutional resource sharing, improved reliability, flexibility and convenience of laboratories, increased laboratory usage, enhanced student interaction, facilitating new teaching pedagogies, etc. Recent advances in web technology and development tools make the remote lab concept more achievable and effective. This paper describes the web-based RemoteFPGA lab [1] which offers enhanced real-time visualisation and interaction with the remote FPGA, compared to other reported remote FPGA laboratory systems. Figure 6.1 illustrates the RemoteFPGA array of FPGA development systems. Each FPGA module is paired to an associated webcam for streaming of real-time images for each active FPGA. A USB server attaches FPGA resources to the host server. The RemoteFPGA lab server allocates access of an FPGA resource to a single authorised user.

The system currently supports Digilent Nexys 2 (-1200 and -500) FPGA modules, and can be scaled and extended to support a range of FPGA technologies. The architecture allows straightforward addition or removal of supported FPGA modules and individual FPGA webcams. The RemoteFPGA lab provides a range of demonstrator digital hardware component building blocks and complete digital systems, with which users can interact to enhance their understanding of digital systems operation. The RemoteFPGA lab also provides a facility for the user to integrate their HDL-based digital designs within a RemoteFPGA HDL-based project template, and to define their application input and monitor signals. Users can overlay visually interactive FPGA input control icons and signal monitors on a system block diagram uploaded to the RemoteFPGA server. In this way, users can develop interactive real time FPGA hardware applications, illustrating system behaviour at any level of design abstraction and design hierarchy. The RemoteFPGA lab provides the facility to remotely configure an FPGA, to transfer data to and from an FPGA, via a USB/Control and Status Register (CSR) command/data interface. The system allows a user to control and monitor up to 64 byte-wide CSRs within a user’s digital design. The RemoteFPGA lab allows always-on implementation and real-time interactive testing of a wide range of user designs, following design verification. The RemoteFPGA lab enables the creation of real-time FPGA digital design demonstrators to aid students to learn by doing. Two RemoteFPGA application case studies are presented in this paper for illustration, executing on a remote Xilinx Spartan-3E FPGA (Digilent Nexys2 board [2]).
Demonstrator 1 is a display controller (displayCtrlr) component (Figure 6.2) which can be used within an FPGA design to connect signals (eight toggle switches and four spring-loaded push button switches in this case) to eight LEDs and four multiplexed 7-segment display devices. The paper presents interactive views of the displayCtrlr RemoteFPGA lab implementation [1], at three levels of the design hierarchy. The user can select and interact in real time with any of these hierarchical views to control and observe the operation of the displayCtrlr design. The authors have described a number of other RemoteFPGA lab applications in [3] and have reported a case study of the RemoteFPGA lab usage in undergraduate teaching.

Demonstrator 2 is a 6-ball snooker scoring system (snkrScoreSys). Figure 6.3 illustrates the high level view of the snkrScoreSys, providing interactive control and visualisation of the real time RemoteFPGA implementation. Scoring follows a two-ball sequence, where the first ball potted must be a red ball. Ball scores range from red (1) to pink (6). The scoring system maintains a current (break score) and accumulated scores for
Chapter 6. Enhancing Learning of Digital Systems using a Remote FPGA Lab

6.2 Related Work

This section reviews reported work in the development of remote FPGA laboratories. Lowe [4] highlights the incentives for developing remote laboratories including reducing cost and facilitating inter-institutional resource sharing. While there is not any significant research data on remote laboratory cost comparisons, anecdotal evidence indicates that operating costs can be significantly reduced. Security, reliability, flexibility and convenience are also listed as motivating factors. Lowe considers the architectures of remote laboratories, with recent trends focusing on enriching the student interaction, and pedagogic aspects of remote laboratories, since video streaming and bandwidth issues...
have been overcome. The use of technologies such as Asynchronous Javascript and XML (AJAX) simplify remote lab architectures, and can provide a more integrated and responsive environment to enhance the student experience. With remote labs, more rather than less experimentation by students becomes possible, and remote labs make possible student exposure to systems which might not have otherwise been afforded them. Trevelyan [5] reports that students using remote access laboratories operated equipment for much more time than in conventional laboratory classes, and learning outcomes seemed to be significantly improved as a result. Remote laboratories are becoming a common educational environment. A number of remote FPGA laboratories have been reported [6–17]. Many remote laboratories comprise analogue expansion modules and support a range of laboratories including analogue and digital components. Many use a LabVIEW interface and Virtual Network Computing (VNC) to a pool of remote test and measurement equipment (function generators, oscilloscope etc), to control and monitor the remote PC/FPGA. Rajasekhar [12] evaluates the effectiveness of remote labs for FPGA education and describes a remote laboratory system which includes a custom hardware interface to an array of FPGA boards. Drutarovsky [6] describes a remote FPGA which supports remote configuration of an Altera Cyclone II EP2C35F672. The system uses a complex series of test and measurement equipment (logic analyser, digital oscilloscope, signal generator or low cost DAQ IO module), and powerful server to control and monitor the behaviour of the FPGA. The FPGA e-Lab [7] provides remote access to a Xilinx Spartan-3E FPGA, using Windows XP Remote Desktop, an interactive Labview-based GUI, and acquisition hardware via serial and USB ports. The Remote Monitored Controlled Laboratory (RMCLab) [15] enables sharing of hardware and instrumentation resources, and dedicated PLD for addressing individual FPGAs in an FPGA array. The UTS remote laboratory [13] and MIT iLabs [14] remote laboratories are complex systems which support a wide and complex range of experiments, including a number of FPGA experiments. While effective, the requirement of third party software such as LabVIEW for data acquisition and instrument control, increases system cost and complexity. Soares [10] reports a remote Altera FPGA lab for introductory digital systems courses, which uses the Altera Quartus in-system memory content editor to access and control/display the state of FPGA module switches, without requiring additional hardware or third party software. The RemoteFPGA lab described in this paper provides enhanced real-time visualisation and interaction with the FPGA, compared to other reported remote FPGA laboratory systems.
6.3 Remote FPGA Lab Architecture

This section describes the FPGA architecture developed to support the RemoteFPGA lab. Figure 6.4 illustrates the FPGA system block diagram. The RemoteFPGA IP core provides the interface to the USB peripheral, and manages the core and design CSRs. The RemoteFPGA supports 16 core CSRs and 64 design CSRs. The core CSRs mirror the console switch and display settings for the specific FPGA. Core CSRs also handle user clock selection and clock generation logic. The allocation of CSRs to user design signals is defined by the user in the template ISE project HDL, to allow signals to be controlled and monitored using the interactive console. The RFL uses a byte-wide host command interface, and byte-wide transfer between core and user design. A command/data specification defines read or write access from/to CSRs or SDRAM. Complex RFL applications can be implemented using RFL command and data sequences.

6.4 Remote FPGA Web Application Server Architecture

This section describes the RemoteFPGA web application server functionality. The operation of the RemoteFPGA interactive console is also outlined. Figure 6.5 illustrates the RemoteFPGA lab system architecture. This includes the web application server and one USB server, FPGA and webcam for each active session. The web application associates an FPGA/webcam pair with a user session and creates and manages associated information in a database. The database provides persistent storage and state information for user accounts, uploaded bitstreams, system diagrams and FPGAs. A USB server is activated for each FPGA/webcam session. Each USB server executes on the PC to which the FPGAs and webcams are connected, and provides the link between the webserver.
and the Nexys2 Cypress EZ-USB FX USB 2.0 peripheral. The web application configures the FPGA with the requested configuration bitstream and the USB server manages data reads/writes from/to the USB interface module implemented on the FPGA. The RemoteFPGA webpage communicates with the FPGA via the web application and USB server and uses HTTP streaming to provide a real-time interface. Webpage javascript supports the addition, placement and update of items on a predefined system block diagram to provide the RemoteFPGA console interface. The UI update rate for core and user CSRs is 1-10 Hz depending on the network speed and latency. As with a local FPGA hardware development system, it is not practical to closely monitor fast changing display values. The streamed RemoteFPGA image updates at a default rate of once per second. Even with normal network latencies, it is still adequate for practical visualisation of one-second rate changes in the FPGA display devices. Access to the RemoteFPGA is provided using a secure account system. Once logged in, a user requests access to an FPGA from a list of available devices, e.g., Xilinx Spartan-3E XC3S1200E or XC3S500E device. The RemoteFPGA automatically allocates an FPGA/webcam pair to the user, and removes this resource from the available list. FPGA configurations are performed sequentially to balance the CPU load. Configuration typically takes several seconds and queued users are automatically advised. The RemoteFPGA system includes a library of reference design bitstreams with which all users can configure an FPGA. Selection of a reference demo automatically configures the FPGA and opens the related graphical system diagram and user interface. The current RemoteFPGA lab implementation provides up to seven FPGAs and seven webcams running concurrently, supported by a dual core Intel PC. The RemoteFPGA lab uses Xilinx IMPACT for the translation of the configuration bitstream file to svf format.

Students learning digital systems design and reconfigurable computing can benefit from the use of an FPGA hardware development system switch-based interface and standard display devices (LEDs and 7 segment displays). Figure 6.6 illustrates the default RemoteFPGA lab interactive console display for the Digilent Xilinx Nexys2 development system [2]. The streamed video illustrates the selected FPGA module and the operation of its display devices, which provide a visual though limited guide to dynamic system behaviour. The streamed webcam image updates at a rate of once per second. Even with normal network latencies, this is adequate for practical visualisation of one-second rate changes in the FPGA display devices. The number of manual switch and on-board display devices on FPGA modules is typically quite limited. However, providing extended I/O using the RemoteFPGA CSRs and associating interactive user interface icons for overlaying on top of system block diagrams, can greatly enhance user control and monitoring of digital systems. Controls and monitors can be paired with specific
signals within the remotely executing FPGA, at any level of the design hierarchy. The user interface CSR update rate is 1-10 Hz depending on the network speed and latency.

User-entered changes such as input signal switch settings are mirrored in the FPGA input CSRs, and affect the behaviour of the FPGA hardware. Changes to the monitored FPGA signals are reflected on the system block diagram as changes in the display devices (e.g., turning on a red LED). The RemoteFPGA lab offers a diagrammatic view of a digital system with a large array of overlayed remote interactive switch and display icons, to produce an informative hardware demonstration to aid learning.
6.5 Demonstrators

This section describes and demonstrates the displayCtrlr and snkrScoreSys RemoteFPGA lab demonstrators [1], and illustrates the use of the RemoteFPGA lab to enhance learning of the operation of these systems. The examples illustrate the interactive visualisation of digital systems applications at various levels of hierarchy. These applications and views can be used to enhance user understanding of the behaviour of the digital systems. Course participants can benefit both from an interactive learn-by-doing approach using the RemoteFPGA lab console on demonstrator designs, and by creating demonstrators of their own project assignments.

- displayCtrlr demonstrator

Figure 6.7(a) illustrates the RemoteFPGA lab console display of the displayCtrlr top level component and connectivity diagram of Figure 6.2 [1], which provides interactive controls and monitors.

The paper presents three interactive demonstrator views of the displayCtrlr design, which follow a top-down structured design and documentation approach. This approach aids the user’s understanding of the design hierarchy, the behaviour of each digital component within the specific design solution, and the behaviour of the complete digital system.

The RemoteFPGA lab also provides control of the clock input to facilitate a step-by-step control of the assertion and deassertion of input signals (including clock), and allowing the user to perform single step operation of the displayCtrlr design. Figure 6.7(b) illustrates the RemoteFPGA lab FPGA core element which enables interactive user control of the clock signal input. The availability of this type of clock-level design monitoring is typically provided using a logic simulator, providing either timing waveform or text-based visibility of the system signal behaviour. The RemoteFPGA lab facilitates a high level of control and visualisation of a digital system, running on a real FPGA, to aid and enhance learning in digital systems education.

The displayCtrlr implements a multiplexed 7-segment display, using sequenced low assertion of the common anode control signals (anL(3:0), low asserted) to enable each PNP transistor in turn, and hence the current flow through each 7-segment common anode and illuminated 7-segment LEDs. Figure 6.8 illustrates the displayCtrlr Data Flow Diagram (DFD) which describes the main elements within the design, and the internal signals seg7Sel(1:0) and segData(3:0). Figure 6.9 illustrates the displayCtrlr Register Transfer Level (RTL) schematic view with
(a) displayCtrlr top level diagram (of Figure 6.2) with overlayed RemoteFPGA lab inputs (toggle switches), and outputs (LEDs and 7 segment displays)

(b) zoomed view of the RemoteFPGA lab input switches and monitor LEDs, and the RemoteFPGA lab FPGA core element which enables interactive user control of the clock signal input

Figure 6.7: displayCtrlr Demonstrator
Figure 6.8: displayCtrlr Data Flow Diagram (DFD) illustrating and describing the main elements within the design, and internal signals seg7Sel(1:0) and segData(3:0)

overlaid RemoteFPGA lab inputs (toggle switches), and outputs (LEDs and 7 segment displays are visible).

A standard design component HDL interface enables straightforward integration of the user’s HDL designs into the RemoteFPGA lab HDL project hierarchy. All signals to be viewed on the RemoteFPGA lab monitor (as inputs or outputs) must be routed to the top level of the design HDL model, for allocation to CSRs and to be accessible to the user. The RemoteFPGA lab core IP includes a USB interface, and a core and design CSR interface.

Following creation of the FPGA configuration bitstream, the user uploads the bitstream file to their RemoteFPGA lab account and associates the bitstream file with one or more block diagram views of the system. A dedicated FPGA is allocated to each user during their application session. Interactive element symbols (switch and display devices) can be overlayed on each of the diagrams, and associated with the relevant CSRs. Interactive design diagrams can be saved or modified.

A video demonstrating the control and operation of the RemoteFPGA lab for the displayCtrlr is available to registered users [1]. The RemoteFPGA lab facilitates a high level of control and visualisation of a digital system, to aid and enhance learning in digital systems education. For example, the RemoteFPGA lab provides control of the clock input to facilitate a slow step-by-step control of the assertion and deassertion of input signals, and allow the user to perform single step operation
Figure 6.9: displayCtrlr Register Transfer Level schematic view with overlayed RemoteFPGA lab inputs (toggle switches), and outputs (LEDs and 7 segment displays)

of the displayCtrlr design. In this way, a number of operation modes are possible for the displayCtrlr design as follows:

1. Generation of one second update display pulse:

The displayCtrlr displays four digit values on the four 7-segment displays. The rate of display change (controlled by the application of a pulse rate to input signal pulse) can be selected as slow (pulse assertion for one clock period every 1 second) or fast (pulse assertion for one clock period every 81.92us) (Figure 6.7(b)). The former provides the user with a practical visualisation of the multiplexing behaviour of the displayCtrlr, with the 7-segment display changing at a one second rate.

Selection of the (typical) 81.92us rate on input signal pulse provides a steady display to the user, where the fast changing blank display states cannot be detected by the human eye.

2. Generation of user clock:

The displayCtrlr application allows selection of either usrClk or the (default) 50MHz system clock (clk). For the former, the effect of assertion and de-assertion of signal userClk (using the interactive userClk icon), enables the user to observe the clock-by-clock activity at all nodes of the displayCtrlr, to enhance understanding of the system operation.
Chapter 6. Enhancing Learning of Digital Systems using a Remote FPGA Lab

(a) Top level component view

(b) RTL level view for the RemoteFPGA lab 6-ball snooker scoring system with overlayed interactive switch inputs (spring loaded buttons) and outputs (LEDs and seven segment display)
Chapter 6. Enhancing Learning of Digital Systems using a Remote FPGA Lab

- SnkrScoreSys demonstrator

Figure 6.10 illustrates the (a) top level component view and (b) the Register Transfer Level (RTL) view for the snkrScoreSys. Along with the snkrScoreSys top level application view (Figure 6.3), the interactive visualisations of various views of the design enhance the learning of the operation of digital systems. Figure 6.3 illustrates the interactive spring loaded and toggle switch inputs to the snkrScoreSys, with LED and 7-segment display outputs. The RemoteFPGA easily extends the 7-segment display count to six for this application (the Nexys2 FPGA provides four 7-segment display devices). This illustrates the use of the RemoteFPGA lab to develop a complex interactive visual user interface to a remote FPGA-based digital system. Figure 6.10(a) illustrates a current break of 1 for player 1, who has an accumulated score of 85. Figure 6.10(b) illustrates the state of the SnkrScoreSys prior to the addition of the 1 point score for player 1. Player 2 is not active, and has an accumulated score of 73. Using the usrClk control and stepping through a sequence of inputs, the user can observe internal snkrScoreSys signal behaviour, including: synchronisation delays, accumulator chip enable assertion (for one clock period), followed by the accumulation of the ball score value to the selected player’s score, and breac score. The action on assertion of the asynchronous reset assertion (clearing of all registers) can also be observed, along with the behaviour of the various elements which make up the snkrScoreSys digital system.

6.6 Conclusions and Future Work

This paper demonstrates the use of the web-based RemoteFPGA lab to support learning in digital systems design and reconfigurable computing. The RemoteFPGA lab provides enhanced visualisation and interaction with FPGA hardware compared to other reported remote FPGA laboratory systems. The paper describes the RemoteFPGA lab elements and demonstrates its use to support learning using two application case studies for illustration. Users can create visually interactive in real-time with remote FPGA controls and monitors, overlayed on and interacting with signals on system block diagrams at various levels of design abstraction and design hierarchy. Two demonstrator case studies have been described for illustration. The extension of the Remote FPGA lab to support further Xilinx and Altera FPGA modules is in progress.
Acknowledgments

This research is supported by the Irish Research Council for Science, Engineering and Technology (IRCSET), International Centre for Graduate Education in Micro and Nano Engineering (ICGEE), the NUI Galway Millennium Research Fund, NUI Galway Centre for Excellence in Teaching and Learning, and the Xilinx University Programme.
Chapter 6. Enhancing Learning of Digital Systems using a Remote FPGA Lab

References


Chapter 7

Remote FPGA Lab With Interactive Control and Visualisation Interface

Abstract

This paper describes a scalable and extendable Remote Field Programmable Gate Array Laboratory (Remote FPGA) which can be used to enhance the learning of digital systems and FPGA applications. The web-based console provides an always-on, real-time, interactive control and visualisation interface to/from a bank of remote FPGAs. A Xilinx ISE project template enables integration of user HDL-based designs to execute on the Remote FPGA. Host-FPGA communication is supported using a register-based interface. Users can create real-time, interactive and visual demonstrators of digital systems components. The paper presents a demonstrator for a Finite State Machine (FSM) application, and illustrates the use of web-based control and visualisation for enhanced learning of FSM behaviour. The paper also presents a case study of the use of Remote FPGA in undergraduate teaching.
Chapter 7. *Remote FPGA Lab With Interactive Control and Visualisation Interface*

7.1 Introduction

The goal of this research is to enhance the learning of digital systems and FPGA applications. The paper describes an always-on, scalable and extendable Remote Field Programmable Gate Array Laboratory (Remote FPGA) [1]. Remote FPGA includes a novel web-based, real-time, interactive control and visualisation interface to the FPGA hardware. Fig. 7.1 illustrates two FPGA development systems, included within the Remote FPGA array. Each FPGA module is paired to an associated webcam for streaming real-time images of each active FPGA. FPGA resources are attached via a USB hub to the host server. The server allocates FPGA resource access to a single authorised requester. The system currently supports remote configuration and data transfers to/from Digilent Nexys 2 (-1200 and -500) FPGA modules, and can be scaled and extended to support a range of FPGA technologies. The architecture allows straightforward addition or removal of supported FPGA modules and individual FPGA webcams. The paper describes the internal USB and register-based FPGA interface architecture, and the architecture of the web application server.

The novel Remote FPGA console enables users to upload a diagrammatic view of their design (at any level of the design hierarchy) and to overlay a range of interactive input-output icons as a User Interface (UI) on top of the graphic. This facilitates real-time interaction with signals within the user’s FPGA design. UI switch setting changes on the remote monitor are mirrored within the FPGA input Control and Status Register (CSR), to affect the behaviour of the FPGA hardware. Changes to selected monitored FPGA signals are reflected on the UI graphic through changing UI icon states. The paper presents a demonstrator for a Finite State Machine (FSM) application, and illustrates the use of web-based control and visualisation for enhanced learning of FSM behaviour. The Remote FPGA has supported a digital design workshop course. Individual user access and activity is monitored using secure account-based access, to provide information on the learning process and the student activity. The paper presents user activity statistics.

A Xilinx ISE project template is provided to enable the integration of user HDL-based designs to execute on the Remote FPGA. This includes a coreCSR module to manage and monitor the FPGA module-specific interface such as switch and button inputs, LEDs
and seven segment display status. Students can create interactive demonstrators of their own project assignments, to better illustrate their understanding and achievement.

Fig. 7.2 illustrates the top level Remote FPGA interactive monitor display for the Digi- lent Xilinx Nexys2. The streamed video window provides a real-time view of the selected FPGA module, and the operation of its display devices, which provide a visual, though limited, guide to the FPGA dynamic system behaviour. The coreCSRs mirror the UI switch settings and display interface. The UI update rate for core and user CSRs is 1-10 Hz depending on the network speed and latency. As with a local FPGA hardware development system, it is not practical to closely monitor fast changing display values. The streamed FPGA image updates at a rate of once per second. Even with normal network latencies, this is adequate for practical visualisation of one-second rate changes in the FPGA display devices.

Fig. 7.3 illustrates the Remote FPGA system architecture. This includes the FPGA, webcam and webpage, application server, web server, and USB server for each active session. The USB server provides the link between the web application and the USB
Chapter 7. Remote FPGA Lab With Interactive Control and Visualisation Interface

The system executes on a dual core Intel system, supporting seven FPGAs and associated webcam, concurrently.

The structure of the paper is as follows: Section 7.2 reviews previous work in the development of remote FPGA laboratories. Section 7.3 describes the FPGA architecture developed to support the Remote FPGA. Section 7.4 describes the application server and web server architectures, and the process of FPGA configuration and data transfer. Section 7.5 presents a Remote FPGA demonstrator. Section 7.6 presents a case study of the application of the Remote FPGA in undergraduate teaching. Section 7.7 concludes the paper and proposes future work.

7.2 Related Work

This section reviews previous work in the development of remote FPGA laboratories. Lowe [2] highlights the incentives for developing remote laboratories including reducing cost, facilitating inter-institutional resource sharing, security, reliability, flexibility and convenience, enriching the student interaction, and pedagogic aspects of remote laboratories. The use of technologies such as Asynchronous Javascript and XML (AJAX) simplify remote lab architectures. With remote labs, more rather than less experimentation by students becomes possible, and remote labs make possible student exposure to systems which might not have otherwise been afforded them. Trevelyan [3] reports that students using remote access laboratories operated equipment for much more time than in conventional laboratory classes, and learning outcomes seemed to be significantly improved as a result.

The authors have provided a summary review [4] of reported work and reported remote FPGA laboratories. Many remote laboratories comprise analogue expansion modules and support a range of laboratories including analogue and digital components. Many use a LabVIEW interface and Virtual Network Computing (VNC) to a pool of remote test and measurement equipment (function generators, oscilloscope etc), to control and monitor the remote PC/FPGA. A number of reported remote labs support a wide and complex range of experiments, including a number of FPGA experiments. While effective, the requirement of third party software such as LabVIEW for data acquisition and instrument control, increases system cost and complexity. Soares [5] reports a remote Altera FPGA lab for introductory digital systems courses, which uses the Altera Quartus in-system memory content editor to access and control/display the state of FPGA module switches, without requiring additional hardware or third party software. The
Remote FPGA novel web-based, real-time, interactive control and visualisation interface to the FPGA hardware provides a unique platform for interactive digital systems training. Remote FPGA does not require additional third party software or hardware.

### 7.3 FPGA Architecture

This section describes the FPGA architecture developed to support the Remote FPGA. Fig. 7.4 illustrates the FPGA top level system block diagram. The USB&CSR_IF subsystem includes the host-FPGA (USB) interface, 16 core CSRs and 64 design CSRs. Users can therefore select up to 64-bytes of design data for control or monitoring of the FPGA behaviour. The RFLDesignTop subsystem includes the user design, along with the interfaces to/from USB&CSR_IF, SDRAM and displays/switches. RFLDesignTop selects between external or CSR-based switch inputs, and stores FPGA display outputs to coreCSRs. User designs can be integrated into the Remote FPGA HDL hierarchy using the RFLDesignTop VHDL template. RFLDesignTop also defines the signals selected as user interactive input controls or monitors to be overlaid on the interactive monitor diagram. The user is required to connect these signals to their FPGA design top level HDL before integrating their HDL within the Remote FPGA top level HDL model.

Complex applications can be implemented using host-based command and data sequences. For example, a read/write SDRAM memory console interface is included in the Remote FPGA to support block SDRAM data transfers.

### 7.4 Remote FPGA Web Application

This section describes the Remote FPGA web application server architecture (Fig. 7.3) and functionality, and the process of FPGA configuration and data transfer. The operation of the interactive monitor is also outlined. Fig. 7.3 illustrates the web application server and one USB server, FPGA and webcam for each active session. The web application associates an FPGA/webcam pair with a user session, and creates and manages associated information within a database. The database provides persistent storage and state information for user accounts, uploaded bitstreams, system block diagrams and FPGAs. A USB server is activated for each FPGA/webcam session. Each USB server executes on the PC to which the FPGAs and webcams are connected, and provides the link between the web server and the specific Cypress EZ-USB FX USB 2.0 peripheral. The web application configures the FPGA with the requested configuration bitstream.
Chapter 7. Remote FPGA Lab With Interactive Control and Visualisation Interface

Figure 7.4: Top level block diagram FPGA diagram
Chapter 7. Remote FPGA Lab With Interactive Control and Visualisation Interface

and the USB server manages data reads/writes from/to the USB interface module implemented on the FPGA.

A Model View Controller (MVC) framework supports the GUI application, FPGA/webcam resource management, and configuration and data transfers. The MVC model element provides the persistent data store (state management). The Remote FPGA webpage requests data to/from the USB server and uses http streaming to send and receive real-time data to/from the FPGA. Webpage javascript supports the addition, placement and update of items on a predefined system block diagram to provide the console interface. The FPGA configuration steps, executed by the web application, are as follows:

- EZ-USB firmware is loaded to enable USB JTAG configuration
- Xilinx ISE IMPACT is used to convert the FPGA configuration bitstream to SVF format suitable for JTAG configuration.
- SVF configuration data is programmed to the FPGA over JTAG via EZ-USB.
- EZ-USB firmware is loaded to enable USB communication to the FPGA. A USB server is activated, connects to the EZ-ESB device and listens for a connection from the web application.

Subsequent data transfers between the Remote FPGA web page and the FPGA use the USB server via a socket connection from the web application. Access to the Remote FPGA is provided using a secure account system. One logged in, a user requests access to an FPGA from a list of available devices, e.g, Xilinx Spartan-3E XC3S1200E or XC3S500E device. The system automatically allocates an FPGA/webcam pair to the user, and removes this resource from the available list. FPGA configurations are performed sequentially to balance the CPU load. Configuration typically takes several seconds and queued users are automatically advised.

The Remote FPGA system includes a library of reference design bitstreams which all users can use to configure an FPGA. Selection of a reference demonstration automatically configures the FPGA and opens the associated interactive system diagram and user interface to the user.
7.5 Remote FPGA Design Demonstrator

This section presents a Remote FPGA demonstrator for a single shot Finite State Machine (FSM) (Fig. 7.5) and illustrates the novel use of web-based control and visualisation for enhanced learning of FSM behaviour. The singleShot asserts signal aShot for a maximum of one clock period on assertion of input signal sw. aShot is an unregistered output (Mealy state machine). The interactive graphic (Fig. 7.5) includes a range of elements typically used to describe FSM behaviour. These include the singleShot FSM component, flowchart, state transition diagram, system RTL block diagram and truth table. Each element is animated during FSM operation on the remoteFPGA hardware. The user can interact in real-time with the FSM, implemented FPGA hardware design, while observing (in context) the behaviour of signals (at any level of the design hierarchy). The Remote FPGA system also includes a clock gating function within the CoreCSR subsystem which enables interactive user control of clock activity to aid interaction with the design implemented on the FPGA.

Feedback from students has indicated improved understanding of FSM behaviour and the relationship between the various digital system behaviour description formats, compared to that obtained from static text-book-based descriptions.

7.6 Case Study: Application of the Remote FPGA in Undergraduate Teaching

This section illustrates how the Remote FPGA can provide information on the learning process and student activity. Fig. 7.6 illustrates course participant activity on the Remote FPGA during three weeks of usage on a digital design and implementation workshop course. Fig. 7.6 displays the number of logins (which automatically configure a demonstration FPGA design and related interactive console), and the number of participant-specific FPGA configurations.

The workshop requires the implementation of several FPGA hardware subsystems. Participants capture an integrated series of VHDL modules [6], verify design behaviour using the Xilinx ISim simulator and synthesise the design, before configuring the Xilinx Spartan-3E FPGA on the remote Digilent Nexys 2 FPGA development system. Since FPGA configuration and testing occupies a relatively small amount of the VHDL capture-to-FPGA implementation process, remote sharing of five FPGA modules has proven effective for a group of 20 participants, working in pairs. No manual FPGA
Figure 7.5: Console interface demonstration of a Finite State Machine

hardware setup is required. The course assignment extended over a period of approximately three weeks. Fig. 7.6 illustrates the increasing usage of the Remote FPGA system as the assignment progresses. Fig. 7.6 highlights the increasing configuration of user’s own designs as the course progresses. Results indicate more frequent access to the FPGA than in previous workshops which used bench-based FPGA development systems. Participants have benefitted in particular from the interactive control and visualisation of demonstrators and user designs. Feedback from students has indicated enhanced understanding of digital systems behaviour, compared to that obtained from static textbook-based descriptions. The Remote FPGA system could also be used to monitor both under-activity, and over-activity of course participants, where over-activity may suggest an over-reliance on an inefficient design-by-implementation methodology, rather than a comprehensive simulation-based test strategy prior to FPGA implementation.
Chapter 7. Remote FPGA Lab With Interactive Control and Visualisation Interface

7.7 Conclusions and Future Work

This paper describes a scalable and extendable Remote FPGA system, used to enhance the learning of digital systems and FPGA applications. The Remote FPGA web-based remote console provides a novel real-time, interactive control and visualisation interface to/from a bank of remote FPGAs. A Xilinx ISE project template enables integration of HDL-based user designs. The application of Remote FPGA on a digital design workshop course has verified its efficacy in supporting an interactive learn-by-doing methodology. Extending support to other Xilinx FPGA modules, and Altera FPGA modules is in progress.

Acknowledgment

Supported by the Irish Research Council for Science, Engineering and Technology (IRC-SET), International Centre for Graduate Education in Micro and Nano Engineering (ICGEE), NUI Galway Millennium Research Fund, NUI Galway Centre for Excellence in Teaching & Learning, and Xilinx University Programme.
Chapter 7. Remote FPGA Lab With Interactive Control and Visualisation Interface

References


Chapter 8

Remote FPGA Lab for Enhancing Learning of Digital Systems

Abstract

Learning in digital systems can be enhanced through applying a learn-by-doing approach on practical hardware systems and by using web-based technology to visualise and animate hardware behaviour. The authors have reported the web-based Remote FPGA Lab (RFL) which provides a novel, real-time control and visualisation interface to a remote, always-on FPGA hardware implementation. The RFL helps students to understand and reason about digital systems operation, using interactive animation of signal behaviour in an executing digital logic system, at any level of the design hierarchy. The RFL supports the creation of real-time interactive digital systems teaching demos. The paper presents student RFL usage data and survey data which highlight improved student engagement, learning and achievement. The paper describes the RFL architecture, communication interface, web page functionality, user access administration and database management. The paper also describes the RFLGen program, developed to automate user design integration into the Xilinx ISE VHDL-based RFL project wrapper for creation of FPGA configuration bitstreams and RFL animations.
8.1 Introduction

Learning in digital systems design can be enhanced through applying a learn-by-doing approach on practical hardware systems and by using web-based technology to visualise and animate hardware behaviour. Previous research has highlighted many benefits of remote FPGA laboratories including improved learning outcomes, reduced costs, facilitating inter-institutional resource sharing, improved reliability, flexibility and convenience of laboratories, increased laboratory usage, enhanced student interaction, facilitating new teaching pedagogies, etc.

The authors have reported the web-based Remote FPGA Lab (RFL) [1, 2] which provides a novel, real-time control and visualisation interface to a remote always-on FPGA hardware implementation, and animation of digital logic system behaviour at any level of the design hierarchy, to aid student learning and to demonstrate user designs. Real-time interactive digital systems teaching demos are integrated with the RFL, each selectable for execution on the remote FPGA hardware. Animated interactive behaviour descriptions include hierarchical logic circuit signal state, truth tables and karnaugh maps, linked to the remotely executing FPGA. The paper presents two demonstrator design animation tutorials for illustration (multiplexer and cascaddable BCD counter).

The paper reviews a range of reported remote FPGA laboratories, and highlights the novelty of the RFL. The paper also presents student RFL usage data and survey data which highlight improved student engagement, learning and achievement.

Figure 8.1 illustrates the array of FPGA development systems currently included within the RFL. Each FPGA module is paired with a webcam to enable user-selected streaming of real-time images of each active FPGA. Web-based, real-time user interaction with the remotely operating FPGA is provided through an extendable USB-based register interface. The paper describes the RFL architecture, communication interface, web page functionality, user access administration and database management.

RFL digital systems are captured, verified and implemented using industry-standard design, verification and implementation tools and processes, and execute in real-time on the FPGA hardware.
Xilinx ISE VHDL project templates allow users to modify existing tutorial HDL models, to create their own interactive RFL demos. A Xilinx ISE RFL User Design Wrapper (UDW) provides the USB register interface to enable integration of user designs into the RFL. The paper describes the RFLGen program, developed to automate the integration of user VHDL-based designs into the UDW. RFLGen also modifies the user design VHDL description to connect user-selected internal signals to the wrapper HDL level for connection with the register interface, and generates the RFL configuration bitstream.

The RFL website supports uploading of one or more user-generated illustrative system diagrams to the RFL console, uploading of the user design configuration bitstream, interactive placement of control and visualisation icons, and saving of the user design animation description. This enables the creation of interactive animations of designs to support teaching and interactive user project demonstrations.

The structure of the paper is as follows: Section 8.2 reviews previous work in the development of remote FPGA laboratories and highlights functionality provided by the Remote FPGA Lab. Section 8.3 summarises the RFL functionality. Section 8.4 describes the RFL web application server architecture and functionality, the operation of the interactive console, the RFL FPGA architecture and USB register interface. Section 8.5 describes the RFLGen program. Section 8.6 presents RFL usage data and student survey data. Section 8.7 concludes the paper and proposes future work.

8.2 Related Work

This section reviews reported work in the development of remote FPGA laboratories and highlights the functionality provided by the proposed Remote FPGA Lab. Lowe [3] highlights the incentives for developing remote laboratories including reducing cost and facilitating inter-institutional resource sharing. While there is not any significant research data on remote laboratory cost comparisons, anecdotal evidence indicates that operating costs can be significantly reduced. Security, reliability, flexibility and convenience are also listed as motivating factors. Lowe considers the architectures of remote laboratories, with recent trends focusing on enriching the student interaction, and pedagogic aspects of remote laboratories, since video streaming and bandwidth issues have been overcome. The use of technologies such as Asynchronous Javascript and XML (AJAX) simplify remote lab architectures, and can provide a more integrated and responsive environment to enhance the student experience. With remote labs, more rather than less experimentation by students becomes possible, and remote labs make possible student exposure to systems which might not have otherwise been afforded them. Trevelyan [4] reports that students using remote access laboratories operated equipment
for much more time than in conventional laboratory classes, and learning outcomes seemed to be significantly improved as a result. Remote laboratories are becoming a common educational environment. A number of remote FPGA laboratories have been reported [5–16]. Many remote laboratories comprise expansion modules and support a range of laboratories which include analogue and digital components. Many use a LabVIEW interface and Virtual Network Computing (VNC) to a pool of remote test and measurement equipment (function generators, oscilloscope etc), to control and monitor the remote PC/FPGA. Rajasekhar [11] evaluates the effectiveness of remote labs for FPGA education and describes a remote laboratory system which includes a custom hardware interface to an array of FPGA boards. Drutarovsk et al, 2009] describes a remote FPGA which supports remote configuration of an Altera Cyclone II EP2C35F672. The system uses a number of test and measurement devices (logic analyser, digital oscilloscope, signal generator or low cost DAQ IO module), and a powerful server to control and monitor the behaviour of the FPGA. The FPGA e-Lab [6] provides remote access to a Xilinx Spartan-3E FPGA, using Windows XP Remote Desktop, an interactive Labview-based GUI, and acquisition hardware via serial and USB ports. The Remote Monitored Controlled Laboratory (RMCLab) [14] enables sharing of hardware and instrumentation resources, and dedicated PLD for addressing individual FPGAs in an FPGA array. The UTS remote laboratory [12] and the MIT iLabs [13] remote laboratory are complex systems which support a wide and complex range of experiments, including a number of FPGA experiments. While effective, the requirement of third party software such as LabVIEW for data acquisition and instrument control increases system cost and complexity. Olivares et al [16] present a remote FPGA lab and related learning methodology highlighting the benefits of the FPGA project-based methodology.

Embedded FPGA logic analysers such as Xilinx Chipscope [17] and Altera SignalTap II [18] have been developed to support hardware FPGA debug. IP elements are typically included within the FPGA following user design completion, to enable selection and capture of selected signal state sequences for read back and analysis, typically using timing diagrams. To the authors’ knowledge, embedded logic analyser tools have not been proposed as formal digital logic education tools, offering graphical design control and animation, or multi-user access and logging.

Soares and Lobo [9] report a remote Altera FPGA lab for introductory digital systems courses, which uses the Altera Quartus in-system memory content editor to access, control and display the state of FPGA module switches, without requiring additional hardware or third party software. The in-system probe uses JTAG resources to both read and write to a user design. A mega function instantiated into the user’s HDL code contains source ports and probe ports for driving input signals and sampling signals that are connected to the ports. During runtime, a GUI displays each source and probe port
by instance and allows the user to read from each probe port and to drive each source port. The GUI enables the toggling of control signals during the debugging process. Soares and Lobo [9] use the in-system sources and probe GUI as a replacement for the push buttons and LEDs used during the development phase of an FPGA project. The authors indicate that Altera Quartus scripting commands can be used to build a virtual front panel during a design prototyping phase. Details of the JTAG interface are abstracted away by the megafunction.

The Remote FPGA Lab presented in this paper [1, 2] provides a novel, real-time control and visualisation interface to a remote always-on FPGA hardware implementation, and animation of digital logic system behaviour at any level of the design hierarchy. The RFL aids student learning through interactive demonstration of user design operation. A series of real-time interactive digital systems teaching demos are provided within the RFL, each configured and animated using a single-click access method. Animated interactive behaviour descriptions include hierarchical logic circuit signal state, truth tables and karnaugh maps, linked to the remotely executing FPGA. For complex digital systems, the RFL is also an enhancement over abstract and slower simulation-based animated teaching tools. The RFL helps students to understand and reason about digital system operation, using real-time animation. The system supports concurrent, multi-user access to an array of FPGAs, with user registration, secure user access and usage logging. Third party software such as Labview is not required. To the author’s knowledge, no such visually interactive multi-user remote FPGA array has been reported.

### 8.3 Remote FPGA Lab Functionality

This section describes the functionality of the Remote FPGA Lab. The RFL currently supports an array of Digilent Nexys 2 Xilinx XC3S1200E-5FG320 FPGA module [19, 20]. The architecture is scalable and extendable to support a range of FPGA technologies. Figure 8.2 illustrates (a) the RFL FPGA architecture and interfaces, and (b) the User Design Wrapper (UDW), which enables the integration of the user design into the RFL.

The M2\_1 demonstrator (Figure 8.3) provides animation of a 2-to-1 multiplexer design implementation [21]. User control of the multiplexer input signals (sel, muxIn1, muxIn0) is provided by interactive switch icons linked to the control register interface (controlCSR, Figure 8.2(b)). The RFL provides animation of the multiplexer symbol, logic circuit, truth table, lookup table and karnaugh map. The star icon on the animation (Figure 8.3) represents the active state of the M2\_1 inputs to aid student understanding of the logic behaviour. All signals, including the multiplexer output signal (muxOut), are
monitored remotely via the status register interface (statusCSR, Figure 8.2(b)). LED icon states (Figure 8.3) reflect the state of the physical signals.

The cascadable BCD counter demonstrator control and visualisation interface (web-based user console, Figure 8.4) illustrates a two digit loadable, up/down BCD counter [22], with two count enable input signals (low asserted ceL input and user-selectable one clock duration pulse input). LED and 7 segment display icon states reflect the state of the physical signals. Interactive hierarchical counter animation diagrams (Top level, Figure 8.4(a)) and RTL, Figure 8.4(c)) are selectable without requiring reconfiguration of the remotely executing FPGA. The RFL supports clock gating (Figure 8.4(b)) enabled by assertion of signals selUserClk and userClk (Figure 8.4(b)). The counter demonstrator also provides (Figure 8.4(b)) user control of counter enable input pulse signal frequency. These elements enable user control and viewing of the behaviour of sequential systems on a clock cycle level. For example, the counter state can be controlled by the user to views the assertion of ceo, typically not visible since it is asserted for one 20 ns (50MHz) clock period. This enhances student understanding of sequential design operation.
8.4 Remote FPGA Lab Architecture

This section describes the RFL web application server architecture and functionality, the operation of the interactive console, the RFL FPGA architecture and the register-based USB interface. Figure 8.5 illustrates the top level RFL architecture, including the web application server, USB servers, FPGAs and webcams. The web application is built using Django [23], a Python web application framework. This framework enables rapid application development and implements a robust and secure user account system. Django is a Model View Controller (MVC) framework allowing separation of application logic, persistent data store (database) and presentation code (web pages).

In a MVC framework, data structures are defined as models. User interaction is performed through a view. In the case of the RFL, user interaction is provided through a web page. Application logic is termed the controller, and interacts both with models and views. Such separation simplifies development and promotes a robust code base. The RFL application maintains a number of models, as follows:

- User Accounts
- User Groups (to which a user can be allocated)
Figure 8.4: RFL Control and visualisation interface (web-based user console) for the two digit cascadable BCD counter
Django manages user authentication, account and group-based permissions, and the association between model instances. For example, a bitstream instance is “owned” by a user account instance, and an FPGA instance is “loaded with” a bitstream instance and is “in use by” a user instance.

Communication with the FPGA hardware is managed by a USB-based protocol and associated USB logic interface and register-based USB interface. The interface is exposed to Django using an API which supports FPGA configuration and byte-wide data transfer. The USB interface and extendable user CSR (UICSR) IP core (Figure 8.2(a) currently supports a 64 byte interface between the RFL user console and the FPGA user design. Sixteen core CSRs are included in the UICSR IP to mirror the Nexys 2 Xilinx Spartan-3E FPGA module switch inputs, LEDs and seven segment displays. Core CSRs are used to create the RFL Nexys 2 default remote console display (Figure 8.4(d)) which mirrors the state of the hardware switches, LEDs and seven segment displays. The RFL uses a byte-wide host-FPGA command/data interface (Figure 8.2, rxDat and txDat interface). Each FPGA is associated with a webcam which provides live streaming video (Figure 8.4(c)) of the FPGA in use.

User interaction with the FPGA hardware takes place through the view (webpage). The web application framework uses an API user view for permission and session-based access. The view communicates with the controller which verifies user permission to perform the requested task, before calling the relevant API function. To provide an
application-like interface for users, all view-controller interaction related to FPGA hardware takes place using asynchronous requests implemented using JQuery [24] wrappers around the standard XML HTTP request (XHR) javascript function. FPGA status is relayed by the controller to the view through HTTP streaming (also known as HTTP server push, XHR pushlets or Comet).

The RFL web page supports the upload of one or more user-generated digital logic system descriptive drawings, and user FPGA configuration bitstream files, and their subsequent association and interaction. The web page also provides user interactive placement of the control and visualisation icons on the RFL console e.g., interface register value indicators, switches, buttons, updated in real time by the view.

Access to FPGA hardware is only required during configuration and usage of an FPGA. In this way, FPGA resources can be efficiently shared between a group of users. Users are automatically allocated an available FPGA. Users are advised if all FPGA resources are in use. A 90 second FPGA access timeout occurs when input signals have been inactive for the timeout period. This policy automatically frees up FPGA resource and provides an effective resource access management facility. On configuration of an FPGA by a user, the RFL automatically allocates an FPGA/webcam pair to the user, and removes this resource from the available list. FPGA configurations are performed sequentially to balance the CPU load. Configuration typically takes several seconds and queued configuration users are automatically advised.

The streamed video illustrates the selected FPGA module and the operation of its display devices, which provide a visual guide to dynamic system behaviour. The streamed webcam image updates at a rate of once per second. Even with normal network latencies, this is adequate for practical visualisation of one-second rate changes in the FPGA display devices.

### 8.5 Automatic Integration Of User Design Into The RFL

This section describes the RFLGen program, developed to automate the integration of the user design VHDL (User Design/UD, Figure 8.2) into the Xilinx ISE VHDL RFL project VHDL User Design Wrapper (UDW) and top level RFL VHDL design RemoteFPGALabTop (Figure 8.2 and Figure 8.5). RemoteFPGALabTop integrates the user design with the USB interface and register interface, and is used to create the RFL FPGA configuration bitstream. RFLGen also enables selection of signals within the design hierarchy and modification of a copy of the user VHDL to provide internal
Chapter 8. Remote FPGA Lab for Enhancing Learning of Digital Systems

(a) User design hierarchical block diagram, illustrating two user-selected internal signals routed to the userDesignTop level

(b) userDesignTop VHDL model, following processing using RFLGen

(c) segB VHDL model, following processing using RFLGen

**Figure 8.6:** RFLGen workflow

signal connection to the User Design level and statusCSR register array, and hence their association with interactive RFL icons on the RFL animation drawings.
All design input and output signals are available by default in the top level user design HDL entity description. This enables direct connection from the interface register control bits (controlCSR register array, Figure 8.2) and status bits (statusCSR register array Figure 8.2), for use as input/output icons in the interactive RFL animation. Signals within a hierarchical digital logic system are not routed as output signals to the top level HDL and are therefore not (by default) available via the statusCSR for inclusion in the RFL animation web page. Manual routing of selected internal signals to the UDW is time-consuming and error-prone. RFLGen has been developed to automate the process. Bringing the user-selected signals to the top level design only uses FPGA routing resources. RFLGen first replicates the source user project database, extracts the hierarchical VHDL project file list from the ISE .prj file, and extracts a complete list of internal signals from the hierarchical VHDL- based project database.

The following example illustrates the functionality of RFLGen. Figure 8.6(a) illustrates a basic user design hierarchical block diagram example with two internal signals selected by the user (highlighted) for routing as userDesignTop level outputs, for connection to the RFL status register interface in the UDW. The selected internal signals include the RFL signal name prefix in the userDesignTop HDL model. Figures 8.6(b) and 8.6(c) illustrate elements of the resulting VHDL description after processing by RFLGen. Tasks include the creation of the new routing signals, the assignment of existing selected internal signals to routing signals, and modification of (a) sub-component entities, (b) and component declarations (c).

RFLGen modifies the VHDL project files to route selected internal signals to the userDesignTop HDL, connects the RFL signals to the register interface in the UDW description and then generates the user design FPGA configuration bitstream.

If global signal synthesis is supported by the EDA tools, the VHDL RFL integration process can be simplified by connecting VHDL signals across multiple levels of hierarchy using a VHDL global signal alias assignment method. Alias signals can be declared within a VHDL package, shared across all relevant hierarchical VHDL components (which refer to the package) and the wrapper HDL file. Global signal synthesis is expected to be supported in Xilinx ISE 14.

The user can also select console icons to associate with the selected internal signals (from the available list of icons) and can save the association information for future modification. Following upload of the bitstream and associated diagram(s) to the RFL, the list of selected signals and associated icons appears on the RFL console, ready for placement by the user to create and save the RFL console configuration for the application.
8.6 Improved Learning And Achievement: RFL Usage And Student Survey

This section presents student RFL usage data and survey responses for a class of sixteen 4th year B.E (Hons) degree students, during the 12 week Digital Systems Design workshop module in the National University of Ireland Galway (Sept-Dec 2011). Thirteen survey responses were received.

Data extracted from the RFL system usage logs indicates a total of 503 remote FPGA
configurations of various course demonstration applications and student design implementations, over the 12 week period. This represents an average of 31 FPGA configurations per individual student during the course, a substantial increase in the student interaction compared with previous access to local FPGA hardware available during previous workshops prior to the development of the RFL. Configurations include tutorial demos (interactive animated hardware demos) and user designs configurations. Figure 8.7(a) illustrates the survey respondents’ overall level of RFL usage and usage outside formally scheduled classes and workshop laboratories. Figure 8.7(b) illustrates the automatically logged user RFL configuration event times. Each mark represents the configuration of an FPGA one or more times within a 30 minute period. The graph illustrates participant RFL usage, with several users active outside formal class time. The survey and automatically logged results correlate.

The module assessment is by examination and continuous assessment. Figure 8.7(c) illustrates the survey respondents’ opinion of the contribution of the RFL to their overall module achievement. The average (median) written examination mark for participants in this module in 2011 is 65% (68%). This compares with an average (median) written examination mark of 57% (58%) for the class of 2010 (which did not use the RFL).

Figure 8.7(d) illustrates the respondents’ opinion of the contribution of the RFL application demonstration to their understanding of a range of digital components and subsystems, presented through formal tutor-led and self-paced demonstration. The RFL recorded usage data and user survey responses indicate enhanced student engagement, learning and achievement through the use of the RFL.

8.7 Conclusions And Future Work

This paper describes the web-based Remote FPGA Lab which helps students to understand and reason about digital system operation, using interactive animation of signal behaviour in an executing digital logic system at any level of the design hierarchy. The paper presents student RFL usage data and survey data which highlights improved student engagement, learning and achievement. The paper illustrates the operation of the RFLGen program, developed to automate user design VHDL integration into the Xilinx ISE RFL project wrapper for creation of the FPGA configuration bitstream and RFL animations. Extension of FPGA support to additional Xilinx and Altera FPGA families is currently in progress.
Acknowledgments

This research is supported by Science Foundation Ireland under Grant No. 07/SRC/C/11169, the Irish Research Council for Science, Engineering and Technology (IRCSET), International Centre for Graduate Education in Micro and Nano Engineering (ICGEE), the NUI Galway Millennium Research Fund, NUI Galway Centre for Excellence in Teaching and Learning, and the Xilinx University Program.
References


Chapter 9

Digital Systems Pedagogy using the Remote FPGA Lab

Abstract

The reported interactive online Remote FPGA Lab (RFL) array provides real-time control and visualisation of digital logic system operation, to multiple users, at any level of the design hierarchy. Reported student RFL usage and survey data results have illustrated the benefits of a teaching and learning strategy using interaction and animation on the RFL, compared to the traditional use of static images in text books. This paper presents a digital systems pedagogy which uses the RFL to animate Truth Table, Lookup Table (LUT) and D Flip Flop (DFF)-based design descriptions, and representations of physical FPGA LUT and DFF-based implementations. The authors propose that this methodology can be effective without the need for logic minimisation using Karnaugh maps and Boolean algebra, particularly in foundation-level digital design modules. The paper also presents the interactive real-time timing diagram creation and user assessment functions of the RFL platform for digital systems education.
9.1 Introduction

The RFL [1] is a highly flexible, always-on, scalable array of remote hardware FPGAs which provides multi-user real-time control and visualisation of digital logic system descriptions, at any level of the design hierarchy. Figure 9.1 illustrates the RFL array of FPGA development systems. The RFL supports structured online digital logic systems courses through real-time interactive animation of multiple diagrammatic views of digital systems. The RFL provides a web-based graphical user interface to the remote FPGA, through which the user can control FPGA input signals and view FPGA output and selected internal signal state in real time. A webcam relays real-time behaviour of the display devices on the active FPGA module. The RFL provides an experience that comes close to that of having hardware always connected and available to the user’s workstation. This can positively impact teaching and learning of digital systems.

The paper proposes and illustrates a digital systems teaching strategy which focuses on animation of Truth Table (TT), LUT and DFF logic descriptions, and animated representations of physical FPGA LUT and DFF-based implementations. The authors propose that this methodology can be effective with a reduced emphasis on logic minimisation using Karnaugh maps and Boolean algebra, particularly in foundation-level digital design modules.

Currently the user interface supports up to 512 FPGA input/output signals and a range of visual UI widgets including switch and signal/bus text box inputs, LEDs and 7-segment display outputs. User input signals and internal FPGA registers are transmitted to/from the RFL web browser respectively via a USB-based server-FPGA register protocol. The RFL system provides an extendable control and visualisation UI widget palette. The paper also presents the recently developed interactive timing diagram RFL functionality using illustrative digital logic examples. The temporal behaviour of digital logic systems is often a difficult concept and is typically viewed and analysed using logic simulation tools or an embedded FPGA logic analyser. The RFL interactive timing diagram function provides an easy to use, intuitive interface for selection of signals, control of input signals and monitoring of signal behaviour on real FPGA hardware. Reported student feedback presented in this paper indicates that the timing diagram animation tool is an effective learning tool.

The paper also presents the recently developed real time RFL assessment functionality. Since the RFL incorporates an FPGA engine executing a real-time digital design, the RFL can be used to produce quiz-master answers on the state of the FPGA signal/bus logic for a given input signal state or signal sequence. When the user enters their prediction of signal/bus values on the browser the RFL compares the actual FPGA
signal states to assess the user’s understanding. This information can be provided as learning feedback to the user and as learning statistics to the course tutor.

The RFL has been used on eight courses to date to provide live remote FPGA lab demos as illustrative animated examples and to enable course participants to upload and animate their own designs on the RFL. The RFL has been applied in distance learning modules, in formal interactive group-based presentations and individually by students outside formal class time. The authors have reported student RFL usage and survey data [2] which highlights the effectiveness of the RFL-based learn-by-doing approach for enhanced learning, achievement and engagement. Student activity and interaction with FPGA hardware has increased significantly [2] since the introduction of the RFL into courses. Results presented in this paper illustrate strong user support for the use of the RFL to enhance digital system and computer architecture education.

RFL applications are created using industry-standard design capture, verification and implementation tools and processes. Users can control the system clock within sequential logic systems to allow viewing of user-defined signal behaviour at clock cycle resolution.

The RFL currently supports animation and visualisation of the following diagrams (several are presented in this paper), to illustrate the entire process from specification to FPGA hardware implementation:

- Hierarchical logic block diagrams.
- Truth Tables (TT) / Function Tables.
- FPGA hardware Look Up Table (LUT) and D Flip Flop
- (DFF) implementation representations.
- Finite State Machine (FSM) flow charts.
- Timing diagrams.
- Representation of physical FPGA LUT/DFF/routing implementation (provided by Xilinx FPGA Editor).
• Gate/technology level digital logic circuit schematics.

• Logic karnaugh maps.

The reader is referred to previous publications [2–4] for descriptions of the:

• RFL architecture, communication interface and web page functionality.

• Reported student RFL usage and survey data results.

• User access administration and database management.

• Client side tool which integrates existing user VHDL-based designs to enable implementation on the RFL (including automated generation and linking of UI widgets to FPGA internal signals).

• Upload of user-generated illustrative diagrams and user design configuration bitstreams to the RFL console.

• Interactive placement of control and visualisation UI widgets on the diagrams.

The structure of the paper is as follows: Section 9.2 presents a range of reported remote laboratories and embedded FPGA logic analysers and differentiates the functionality of the RFL which brings static illustrations of digital logic systems to life through interactive animation. Section 9.3 proposes and illustrates a digital systems pedagogy using the RFL and animated visual representations to support learning and understanding of digital logic system behaviour and FPGA implementation. Section 9.4 presents the RFL timing diagram animation function. Section 9.5 describes the RFL support for automatic user assessment. Section 9.6 concludes the paper and proposes future work.

9.2 Remote FPGA Labs: Background

This section provides a summary of selected remote laboratories and embedded FPGA logic analysers and differentiates the functionality of the RFL for supporting structured online digital logic system courses through interactive animation. The reader is referred to [2] for a more complete review.

Lowe [5] has highlighted many benefits of remote FPGA laboratories including improved learning outcomes, reduced costs, facilitating inter-institutional resource sharing, improved reliability, flexibility and convenience of laboratories, increased laboratory usage, enhanced student interaction, facilitating new digital systems pedagogies, etc. Lowe
considers the architectures of remote laboratories, with recent trends focusing on enriching the student interaction, and pedagogic aspects of remote laboratories, since video streaming and bandwidth issues have been overcome. The use of technologies such as Asynchronous Javascript and XML (AJAX) simplify remote lab architectures, and can provide a more integrated and responsive environment to enhance the student experience. With remote labs, more rather than less experimentation by students becomes possible, and remote labs make possible student exposure to systems which might not have otherwise been afforded them. Trevelyan [6] reports that students using remote access laboratories operated equipment for much more time than in conventional laboratory classes, and learning outcomes seemed to be significantly improved as a result. This outcome has also been reported using the RFL [2].

The MIT iLab project [7] dedicated to the proposition that online laboratories can enrich science and engineering education by greatly expanding the range of experiments that students are exposed to in the course of their education. The iLabs Logic Design iLab [8] uses an Altera DE1 FPGA development board and supports VHDL synthesis. Soares and Lobo [9] report a remote Altera FPGA lab for introductory digital systems courses, which uses the Altera Quartus in-system memory content editor to access, to control and display the state of FPGA virtual module switches.

Embedded FPGA logic analysers such as Xilinx Chipscope [10] and Altera SignalTap II [11] have been developed to support hardware FPGA debug. Embedded FPGA Logic Analysers monitor selected trigger signal patterns in an operating FPGA, and capture selected signal trace progression following the detection of a pattern signal signature. Embedded logic analysers provide a very effective signal trace and design debug facility.

The RFL extends the remote FPGA lab and embedded logic analyser concepts to bring illustrations of static digital logic systems to life through interactive animation. The RFL platform supports online learning of digital logic systems and an automatic assessment methodology. The RFL does this through novel real-time interactive control, animation and visualisation of signals at any level of the hierarchy within operating complex digital logic hardware systems.
9.3 RFL Teaching Strategy

9.3.1 Introduction

This paper proposes and illustrates an effective digital systems teaching strategy which focuses on animation of TT, LUT and DFF logic descriptions, and animated representations of physical FPGA LUT and DFF-based implementations. The paper proposes that logic minimisation using Karnaugh maps and Boolean algebra is not necessary for effective understanding of the digital logic design to FPGA implementation process, and propose a reduced focus on Karnaugh maps and Boolean algebra, particularly in foundation-level digital design modules. This section presents the animation of example combinational and sequential digital logic modules (Figures 9.2 and 9.3) in order to illustrate this concept. An industry-standard structured top-down design, bottom-up capture, simulation and implementation methodology has been applied in all RFL-based course tutorial application documentation and designs. Structured diagrammatic documentation presents designs as a series of partitioned hierarchical drawings using the following formats:

- Context Diagram (CD): top level component and interface definition.
- Data Dictionary (DD): provides a list of appropriately named system input and output signals, their assertion levels, and signal function description. The DD also provides information on the connected component interfaces.
- Data Flow Diagrams (DFDs): series of related hierarchical diagrams, including subsystem functions, related descriptions and incremental DDs.
- Functional Partition (FP), representing the complete system as a schematic block diagram.

9.3.2 Lock Controller: AND-based Illustrative RFL example

Figures 9.2(a-e) illustrate selected RFL animated views of an AND-based lock controller component. Figures 9.2(a-e) illustrate and animate steps and representations in the process from design specification to FPGA hardware implementation. This very basic animated logic example is used for illustration of aspects of the combinational logic design-to-implementation process. More complex designs can be illustrated similarly. Signal animation is used across the various types of diagrams to strengthen the user’s understanding through investigation and interaction with dynamic diagrams.

Figure 9.2 presents the following animated views of the basic lock controller design:
<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T9amTo4pm</td>
<td>asserted (H) when time is between 9am and 4pm. Signal generated by electronic clock.</td>
</tr>
<tr>
<td>cashierPresent</td>
<td>asserted (H) when cashier in attendance. Generated by electronic key lock.</td>
</tr>
<tr>
<td>openSafe</td>
<td>assertion (H) activate solenoid switch to open lock on a safe.</td>
</tr>
</tbody>
</table>

Table 9.1: Lock Controller Data Dictionary

(a) CD
(b) TT
(c) LUT
(d) LUT/DFF-based synthesised technology schematic
(e) Physical FPGA implementation

Each Figure 9.2 element is described below. LEDs illustrate the signal state. Switches enable interactive user input. The star symbols highlight the selected cell in TT and LUT descriptions. The Star UI widget and TT state is provided by the RFL application and highlights the active LUT cell.

1) Figure 9.2(a): animated context diagram and connect interfaces. Figure 9.2(a) inset illustrates the streamed video of the active FPGA.

2) Figure 9.2(b): openSafe TT representation. The star symbols, provided by a TT decoder within the RFL application, highlights the active TT cell for illustration of operation.

3) Figure 9.2(c): openSafe LUT representation. Again, the star symbols animate the active LUT cell. There is a direct translation of the openSafe signal TT description for implementation on the LUT-rich FPGA architecture.

4) Figure 9.2(d): lock controller synthesised technology schematic (for Xilinx Spartan-3E technology). The FPGA implementation schematic, produced by the Xilinx View Synthesised Technology Schematic operation, includes a single 2-i/p LUT (configuration inset) implementing the lock controller logic function (AND) and providing the openSafe signal. The I/O buffer elements have been removed in the diagram.

5) Figure 9.2(e): RFL animation of lock controller FPGA implementation openSafe signal LUT description (provided by Xilinx FPGA Editor).
Other animated diagrams such as an interactive timing diagram (proposed in this paper) could also be included to enhance understanding further. The authors consider that this sequence of representations (Figure 9.2(a-e)) can provide a good understanding of the combinational component behaviour and its implementation on an FPGA (especially when animated and controllable). Note that no reference to logic minimisation, Boolean algebra or Karnaugh maps has been required or included in this example.

The authors also propose the presentation of illustrative abstract HDL descriptions early in the digital systems education cycle in order to develop an awareness and understanding of HDL-based methodologies and the use of EDA-based logic synthesis tools.

### 9.3.3 Counter Illustrative RFL example

Figures 9.3(a-e) illustrate selected RFL animated views of a two-bit up/down synchronously reset counter (CB2CED) sequential system design. Figure 9.3 presents the following animated views of the counter design:

1. FP
2. DFD
3. LUT/DFF-based synthesised technology schematic
4. Physical FPGA implementation
5. Counter FSM flow chart

There is a direct translation of the counter next state decode TT description and counter state memory for implementation on the LUT- and DFF-rich (respectively) FPGA architecture. Other animated diagrams such as the top level context diagram, an interactive timing diagram, and a HDL model, simulation waveform etc could also be included to enhance understanding further. The authors consider that this sequence of representations (Figure 9.3(a-e)) can provide a good understanding of the CB2CED component behaviour and its implementation on an FPGA (especially when animated and controllable). Note that no reference to logic minimisation, Boolean algebra or Karnaugh maps has been included. In fact the authors suggest that this material could complicate the students understanding of the process from specification to FPGA implementation in this example. If the target technology is ASIC rather than FPGA, then gate level and Boolean description formats are relevant. These topics could be introduced at an intermediate digital systems module level when an ASIC-focus is introduced. Each Figure 9.3 element is described below.
(a) Lock controller Component/Context Diagram

(b) lock controller openSafe TT

(c) lock controller openSafe LUT implementation

(d) lock controller synthesised technology schematic (Spartan-3E technology). The FPGA includes a single 2-i/p LUT (configuration inset)

(e) RFL animation of lock controller FPGA implementation openSafe signal LUT description (provided by Xilinx FPGA Editor)

Figure 9.2: Lock controller demonstrator
1) Figure 9.3(a): animated counter FP diagram which includes illustration of animated increment and decrement component behaviour, Next State (NS(1:0)) behaviour and the effect of signals up/down (uD) and chip enable (ceL, low asserted) on the NS(1:0) logic assignment selection path. The CB2CED counter logic state of Figure 9.3(a) illustrates the assertion of terminal count (TC) and count enable output (ceo) signals for a count(1:0) value = 3, operating as an up counter. The ‘user clock and ce control’ element (Figure 9.3(a)) is provided within the RFL project wrapper [2]. These controls enable the selection of various counter operating modes to enhance the user’s understanding. The ‘ce control’ element provides the user with a range of count enable signal inputs (pulse every 1 second, 80us, 2 clock periods, or always asserted). The user can select the system clock (50MHz for the Digilent Nexys-2 Xilinx Spartan-3E FPGA module) or manual, user-controlled clock.

2) Figure 9.3(b): animated DFD-based structured design top-down design hierarchy and the relationship between the components and signals in this basic counter example.

3) Figure 9.3(c): animation of the CB2CED counter Synthesised Technology Schematic FPGA implementation LUT/FF descriptions. Clock buffer and I/O buffers have been removed in this diagram. LUTs provide signal NS(1:0) (counter Next State), TC and ceo.

4) Figure 9.3(d): animated FPGA implementation of signal CS(1). No Boolean logic or Karnaugh map-based minimisation step is required in the CB2CED counter to provide the student with an understanding of the counter function, logic relationships and its implementation on FPGA. The Xilinx ISE project describing the VHDL and simulation test bench is provided to users, who can use the Xilinx ISE webpack to implement the CB2CED or modified the counter functionality, along with their own animated illustrative diagrams.

5) Figure 9.3(e): animated flow chart and state transition diagram for the CB2CED. While the CB2CED is a FSM, flow charts are not typically presented for counters, being generally used to describe application-specific FSMs. The CB2CED and other animated FSMs provide useful practical illustration of state transition behaviour and clock control in FSMs.

The user can control the clock to monitor both the active current state (CS(1:0)) and next state (NS(1:0)), illustrated by the star UI widget in Figure 9.3(e). Next state is defined as a function of current state and input signal values, also illustrated in Figure 9.3(a).
Chapter 9. Digital Systems Pedagogy using the Remote FPGA Lab

Figure 9.3: CB2CED Demonstrator

(a) RFL animation of CB2CED counter FP

(b) RFL animation of the CB2CED DFDs

(c) RFL animation of the Xilinx ISE synthesised LUT/DFF technology schematic for the CB2CED, along with LUT configuration data.

(d) RFL animation of CB2CED counter FPGA implementation LUT/DFF signal CS(1) (provided by chart and state transition diagram. Xilinx FPGA Editor)

(e) RFL animation of CB2CED counter state flow
9.4 Interactive Timing Diagram Creation

This section presents a novel RFL real-time user-generated interactive timing diagram control and animation interface. Understanding timing diagrams often poses a difficulty for students. The RFL provides a facility to experiment with circuit state progression by varying inputs and monitoring user-selected signals over time in a waveform format, connected to the operating FPGA. Dynamic timing diagrams are created using the open source WaveDrom [12] library. Timing diagram input signal states are modified by toggling the animation controls on the standard RFL control and visualisation interface, e.g., Figures 9.2 and 9.3. Input signals/bus states modify the FPGA digital module input signals via the register input interface and are reflected in the timing diagram trace. The resulting state of the selected internal and output signals is obtained from the operating FPGA through the RFL register interface and used to modify the timing diagram output signals.

For combinational logic examples, clicking on the Step Clock button (Figure 9.4) progresses the simulation waveform by a time step. For sequential logic examples, the Step Clock button progresses the user clock by one clock period. By default, creating a timing diagram automatically enables the clock single stepping function in a sequential logic project.

Figure 9.4(a) illustrates a real-time timing diagram created using the RFL for the lock controller (Figure 2) example. The glitch in time step 5 cashierPresent and openSafe signal results from the user having toggled signal cashierPresent low and then high during time step 5.

9.5 Automatic Assessment using the RFL

This section describes the RFL support for automatic user and group assessment. The RFL FPGA provides the hardware computing engine to provide selected logic signal values as a reference for comparison with user-predicted signal values. The RFL can therefore be used for automatic assessment of a user’s understanding of logic system behaviour. Online assessment can encourage individual and class participation and provide in-class feedback to highlight topics requiring further tutor support. Input stimulus sequences are generated automatically or by a tutor. The RFL architecture supports tutor-led, class-based assessment, applying simultaneous display of the animated real-time application in each active user’s browser, recording of individual response entries and assessing their correctness.
Figure 9.5(a) illustrates a RFL learning assessment user GUI for a basic illustrative AND logic example. In this example, the user is presented with the four possible AND gate input signal states (illustrated by switch and LED states). The user enters their AND gate output logic prediction in the lower (blue box) fields. On completion of the exercise, or after a timeout period, the actual FPGA state is obtained and displayed (for each AND gate example) as the LED state in the dashed box. In this example, the user is correct in all but Case D, where the erroneous result is automatically coloured red. The RFL presents the user score which can enable automatic module progression on achieving target success levels in a self-paced learning programme using the RFL. The RFL presents selectable user scores and the overall group average.

Figure 9.5(b) illustrates a RFL learning assessment user GUI for a multiplexed 7-segment display controller example. System input signal/bus stimuli are indicated on the left hand of the module (in dashed boxes). The users enters their predicted signal/bus states in the lower user entry fields *1-*4. On completion of the exercise, the FPGA state is presented (for each of the examined logic system signals/buses), with errors highlighted and the assessment score. Figure 9.6(a) illustrates positive user feedback on the effectiveness of the RFL in helping student understanding of timing diagrams. Note that a number of the respondents did not use the timing diagram facility at all.
Chapter 9. Digital Systems Pedagogy using the Remote FPGA Lab

(a) User GUI for RFL learning assessment (Boolean logic)

(b) User GUI for RFL learning assessment (Multiplexed 7-segment display controller example)

Figure 9.5: User Assessment GUI

(a) The RFL helped my understanding of timing diagrams

(b) The RFL enhances digital system and computer architecture education

Figure 9.6: RFL Survey Results

Figure 9.6(b) illustrates user opinion on the use of the RFL to enhance digital system and computer architecture education. The opinion is very positive.

9.6 Conclusion and Future Work

The RFL offers the possibility of bringing static digital logic system illustrations to life through interactive animation. The RFL provides many different animated views of operating digital systems, from top level to FPGA implementation level. The paper proposes and illustrates a focus on truth table, look up table and D flip flop logic descriptions for digital logic training, with a reduced emphasis on logic minimisation, Karnaugh maps and Boolean algebra, particularly for foundation level courses. The authors have previously reported enhanced learning, achievement and engagement using
the RFL. The authors propose that the above pedagogy can be effective, can lead to improved focus and learning, encourage student interest through an application-based approach, and avoid perceived less interesting aspects of digital logic implementation, particularly for newcomers to the topic. The paper also presents and demonstrates extended RFL functionality including interactive timing diagram creation for enhanced learning, and an automated online assessment strategy, both using the RFL FPGA engine.

On-going and future work includes application and UI improvement to enhance the user HDL model integration tools, and to make the use of the RFL more intuitive for creating embedded RFL applications within a range of teaching modules. Timing diagram stimulus recording and playback functionality will also be added, and the assessment functionality will be further developed to include collaborative group-based assessment. Extended FPGA support for Altera FPGAs is also in progress. Further application of the RFL in courses will be used to capture student feedback and assessment of the proposed teaching methodologies.

**Acknowledgements**

Supported by the Irish Research Council for Science, Engineering and Technology (IRC-SET), International Centre for Graduate Education in Micro and Nano Engineering (ICGEE), NUI Galway Millennium Research Fund, NUI Galway Centre for Excellence in Teaching & Learning, and Xilinx University Programme.
Chapter 9. Digital Systems Pedagogy using the Remote FPGA Lab

References


Chapter 10

Conclusions and Future Work

10.1 Introduction

This thesis describes two phases of research. Phase I describes the design, implementation and application of a Network-on-Chip based hardware Spiking Neural Network architecture and the extension of SNNDevSys, a GA-based intrinsic hardware SNN training and configuration platform. Phase II describes the Remote FPGA Laboratory, a web-based digital systems educational tool.

10.2 Network-on-Chip Based Hardware SNN

This thesis details the design, implementation and application of the EMBRACE-FPGA architecture, a NoC based hardware SNN architecture. The challenges to the development of a scalable hardware SNN architecture are described and a solution in the form of NoC based inter-neuron connectivity is proposed. The EMBRACE-FPGA system utilises an array of Neural Tile (NT) elements, each of which is comprised of a NoC router, a spiking neuron model and associated configuration memory. The NoC router enables online reconfiguration of SNN connectivity (topology) and neuron model parameters.

This thesis describes extensions to SNNDevSys [1], a combined Genetic Algorithm (GA) based training and configuration platform, supporting online read-back and verification of EMBRACE-FPGA configuration, graphing of internal neuron membrane potential and evolution of SNN topology as part of the training process.

This thesis describes the application of benchmark SNN application to the EMBRACE-FPGA system. The operation of the SNN has been verified using the standard XOR
classification benchmark. System real-time performance and ability to implement complex control tasks has been demonstrated using an inverted pendulum control application. An implementation of the Wisconsin breast cancer dataset classification problem illustrated the systems ability to provide highly accurate solutions to difficult tasks. The EMBRACE-FPGA implementation of this classifier outperformed existing hardware SNN classifiers by a significant margin.

The use of NoC inter-neuron communication introduces the concept of packet latency and variations in this latency, termed jitter. This jitter has the effect of distorting spike train frequencies and hence the operation of the SNN. A robotics control application has been implemented using EMBRACE-FPGA to investigate the effects of this NoC jitter on the performance of the system. Results indicate that while NoC jitter does have an effect of spike train frequencies, this effect is minimal. The noise introduced by jitter can have a positive effect on the network through guiding GA-based training algorithm towards more robust and error tolerant solutions. The use of a low resolution neural model was found to exacerbate the effect of jitter to some degree.

As the EMBRACE-FPGA has been designed as a reconfigurable system, significant memory is required to store the system configuration. Configuration memory area comprises a large proportion of the total NT area, with most of this memory dedicated to storage of SNN topology information. The thesis has proposed the Single Dynamic Synapse (SDS), an approach that replaces the array of neuron input synapses with a single synapse which is dynamically reconfigured with a weight (received in the spike packet) before the spike is applied. The SDS approach leads to significant memory savings for large SNNs with many incoming connections to each neuron. A 35% reduction in EMBRACE-FPGA memory requirements is demonstrated for a network with 1024 neurons, 256 incoming connections per neuron and 5 synaptic weight bits. Memory saving increases as the SNN grows. Research in the area of improving EMBRACE-FPGA scalability has been conducted in [2–5].

### 10.3 Web Based FPGA Educational Laboratory

This thesis describes the Remote FPGA Laboratory (RFL), a web based digital systems design educational tool. The RFL comprises an array of FPGA devices connected to a server which provides remote access, control, visualisation and configuration to the FPGA array through a web page interface. The system enables authenticated users to access FPGA devices through a standard web browser without additional software. A course builder tool is provided to tutors which supports the creation of course modules
using original or existing course material. Live demonstrations of digital design concepts can be incorporated into courses providing an illustrated and interactive aspect to training modules. The RFL enables users to upload and configure their own designs to FPGA devices as part of a test driven development methodology.

The RFL enhances the student experience through increased student interaction with digital hardware. The system encourages a learn-by-doing approach to digital systems education through its real-time interface and visualisation to real hardware devices. Users can view and modify the state of internal FPGA signals, control the system clock and visualise truth table, Karnaugh map and Finite State Machine states. Timing diagrams of internal signals can be created in real time demonstrating and providing a deeper understanding of internal operation of FPGA operation.

The RFL supports continuous assessment of users as part of a self paced learning environment. Users will have the ability to assess their own progress on a course module and track this progress through their profile page. Course tutors can create assessments through the course builder tool and link assessments to individual course modules. The RFL enables course tutors to monitor users usage and interaction with the system to help gauge student progression.

The system is designed to allow easy addition of new FPGA technologies, facilitating access to the latest FPGA technology and high end FPGA devices, not normally accessible to students. As the system shares resources among a large number of users, most of whom require only intermittent access to an FPGA, the RFL can accommodate approximately 20 users for each physical FPGA device. This helps to reduce both hardware and management costs.

10.4 Future Work

Future work related to Phase I of this thesis could investigate further approaches towards increasing the scalability and density of NoC based SNNs.

An investigation of alternative NoC topologies may result in an interconnect and routing strategy which reduces NT router density through reduced memory usage. Each router is currently configured with a routing table indicating the port a packet should be transmitted on in order to reach its destination. An algorithm based alternative to this memory requirement would reduce EMBRACE-FPGA memory usage.

An investigation of the impact of placing multiple neurons on each NT could reveal efficiencies, particularly if such neurons are directly wired within the NT, removing
much of the memory required for storage of SNN topology. This approach, however, may have an adverse impact on the efficiency of the SNN configuration as a whole and result in reduced memory savings.

EMBRACE-FPGA currently uses a unicast NoC approach, i.e. for each spike transmitted by a neuron an individual NoC packet is created. This approach retains the full flexibility of NoC inter-neuron communication. However traffic density can become quite large as a result, exacerbating NoC latency jitter related noise and increasing power usage through increased switching required by NoC routers. Alternative strategies, such as multicast NoC packets, long range NoC connections and subnets within the NoC may alleviate this traffic density.

A reduction in NoC latency or a facility to provide fixed latency between NTs would reduce latency related noise within the SNN and improve EMBRACE-FPGA performance, particularly as the SNN size grows.

The development of the RFL (Phase II of this thesis) would benefit from research related to websocket communication with client web browsers, particularly in relation to traversing caching proxy servers, which currently disrupt the operation of both HTTP streaming and websockets. This issue will become more significant as RFL traffic grows, due to a relatively high CPU and memory costs associated with keeping many HTTP connections open to provide real-time updates of internal FPGA signal status.

Currently the RFL connects to the array of FPGA devices using USB connections. Due to practical limits of USB, notwithstanding the USB standard supporting 127 devices connected to a single controller, the number of devices which may be connected simultaneously does not scale in a manner which will support further expansion of the RFL. This is due mainly to connected USB devices reserving bandwidth with the controller, quickly exceeding the total available bandwidth. Issues related to electromagnetic interference and power limits imposed by USB hubs have also been experienced which further reduce the practical number of devices. An investigation of different connection strategies is required to ensure scalability of the RFL system.
References


