Design Exploration of EMBRACE Hardware Spiking Neural Network Architecture and Applications

by

Sandeep Pande

A thesis submitted for the degree of
Doctor of Philosophy in Electronic Engineering

Supervisor
Dr. Fearghal Morgan

Bio-Inspired Electronics and
Reconfigurable Computing Research Group

Electrical & Electronic Engineering
College of Engineering and Informatics

National University of Ireland, Galway
IRELAND

February, 2014

Abstract

The operation and structure of the human brain has inspired the development of next generation smart embedded computing systems. The cognitive abilities of the human brain have been partially explained by the dense and complex interconnection of neurons and synapses, where each neuron connects to thousands of other neurons and communicates through short transient pulses (*spikes*) along synaptic links. Brain-inspired computing paradigms such as Spiking Neural Networks (SNNs) mimic the key functions of the human brain and have the potential to offer smart and adaptive solutions for complex real world problems.

The main design challenges for the realisation of practical hardware SNN systems are large scale simulation and performance measurement, compact hardware implementation, architectural scalability, application reliability, low power consumption, efficient and accurate SNN learning/training algorithms, compact implementation of complex neural models, fault tolerance and application design methodologies.

This thesis contributes to the development of EMBRACE, a compact, scalable, modular, hardware SNN architecture, as an embedded computing platform. The thesis presents a prototype implementation of the EMBRACE architecture on a Xilinx Virtex-6 FPGA and demonstrates reliable, practical embedded classifier and control applications. The thesis contributes to a number of hardware SNN system design challenges such as hardware SNN simulations and performance measurement, compact hardware implementation, architectural scalability, application reliability and efficient practical application design. The research is organised in four distinct phases as follows:

Simulation and Performance Measurement of Hardware SNN Systems: The thesis presents EMBRACE-SysC, a SystemC simulation-based design exploration framework for Network on Chip (NoC) based hardware SNN architectures. EMBRACE-SysC incorporates performance measurement and reporting capabilities including spike communication infrastructure, neuron model validation, hardware architecture design exploration and SNN application evolution, used in later phases of this research.

Architectural Techniques for Scalability: The storage of large synaptic connectivity information in hardware SNNs translates to poorly scalable, large distributed on-chip memory in hardware SNN architectures. Inspired by the modular organisation of the human brain, this thesis presents a hardware Modular Neural Tile (MNT) architecture that reduces the memory requirement of the architecture using a combination of fixed and configurable synaptic connections. The silicon footprint of the architecture is reduced by an average of 66% for practical SNN application topologies, as compared to the previously reported EMBRACE architecture.

Interconnect Architecture for SNN Application Reliability: Distortion in spike timings impacts the accuracy of SNN operation by modifying the precise firing time of neurons within the SNN. The thesis presents an in-depth, simulation-based analysis of the synaptic information jitter in NoC based hardware SNNs. The thesis presents a ring topology NoC architecture using a timestamped, spike broadcast flow control technique that offers fixed spike transfer latency under various network traffic conditions, to provide reliable SNN application behaviour.

Modular Application Design: Efficient implementation and training of large scale embedded applications on hardware SNN architectures poses a serious challenge due to the lack of suitable application design methodologies. The thesis presents the modular application design of a robotic navigational controller application implemented on the EMBRACE FPGA prototype. Results indicate faster application evolution as compared to monolithic application SNNs. The stepwise knowledge integration and simplified SNN training facilitate rapid application prototyping.

॥ ॐ ॥

श्री भगवान ऊवाच

बीजं मां सर्वभूतानां विद्धि पार्थ सनातनम् ।
बुद्धिर्बुद्धिमतामस्मि तेजस्तेजस्विनामहम् ॥ ७-१० ॥

śrī-bhagavān uvāca

bījaṁ māṁ sarva-bhūtānāṁ, viddhi pārtha sanātanam
buddhir buddhimatām asmi, tejas tejasvinām aham

Shrimad Bhagavad-Gītā 7.10

O son of Pṛthā, know that I am the eternal seed of all existences, the intelligence of the intelligent, and the prowess of all powerful men.

O Supreme Personality of Godhead, Lord Śrī Kṛṣṇa please accept my humble obeisances. This PhD research and thesis is dedicated onto your lotus feet.

Sandeep Pande

Acknowledgments

This PhD has been an important phase of my life. Overall, my PhD journey was a wonderful mix of experiences and I learnt very important lessons that will help me lead this life in a meaningful way.

The success of this PhD research is due to an enormous support from my teachers, family and friends. My heartfelt thanks to all of you for your backing, encouragement and love.

I sincerely thank my supervisor, Dr. Fearghal Morgan. His guidance and prudent supervision has helped in getting significant results and made this research a success.

I wish to thank our collaborators from University of Ulster, Dr. Jim Harkin and Prof. Liam McDaid for the constructive discussions and review of the key manuscripts.

My special thanks go to Prof. Gerard Smit for hosting me at University of Twente during the most important phase of this research and for productive discussions on the novel Network on Chip ideas.

I thank to the research team Dr. Brian McGinley, Dr. Seamus Cawley, Finn Krewer, Dr. Snaider Carrillo, Tom Bruintjes and Jochem Rutgers for interesting and stimulating discussions. This research was possible with important contributions from all of you. I also thank the staff members of Electrical & Electronic Engineering, in NUI Galway.

I am grateful to all my family members for their support and encouragement. My biggest thanks to my wife, Vrushali for her support in all the phases of this PhD.

This research is supported by the International Centre for Graduate Education in Micro and Nano-Engineering (ICGEE), Irish Research Council for Science, Engineering and Technology (IRCSET) and Xilinx University Programme.

Sandeep Pande

Declaration of Authorship

I, **Sandeep Pande**, declare that this thesis titled, **"Design Exploration of EMBRACE Hardware Spiking Neural Network Architecture and Applications"** 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: _____

Contents

List of Publications

Thesis Publications

[1] **Sandeep Pande**, Fearghal Morgan, Brian McGinley, Jim Harkin, Liam McDaid, "Application prototyping for embrace hardware modular spiking neural network architecture," *Neural Compting and Applications Journal (To be submitted)*, 2013.

[2] **Sandeep Pande**, Fearghal Morgan, Gerard Smit, Tom Bruintjes, Jochem Rutgers, Brian McGinley, Seamus Cawley, Jim Harkin, Liam McDaid, "Fixed latency on-chip interconnect for hardware spiking neural network architectures," *Parallel Computing*, vol. 39, pp. 357 – 371, Sep. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2013.04.010

[3] **Sandeep Pande**, Fearghal Morgan, Seamus Cawley, Tom Bruintjes, Gerard Smit, Brian McGinley, Snaider Carrillo, Jim Harkin, Liam McDaid, "Modular neural tile architecture for compact embedded hardware spiking neural network," *Neural Processing Letters*, vol. 38, pp. 131–153, Oct. 2013. [Online]. Available: http://dx.doi.org/10.1007/s11063-012-9274-5

[4] **Sandeep Pande**, Fearghal Morgan, Seamus Cawley, Brian Mc Ginley, Jim Harkin, Snaider Carrillo, Liam Mc Daid, "Addressing the hardware resource requirements of Network-on-Chip based neural architectures," in *NCTA, International Conference on Neural Computation Theory and Applications, Paris, France*, Oct. 2011. [Online]. Available: http://dx.doi.org/10.5220/0003676601280137

[5] **Sandeep Pande**, Fearghal Morgan, Seamus Cawley, Brian McGinley, Snaider Carrillo, Jim Harkin, Liam McDaid, "EMBRACE-SysC for analysis of NoC-based spiking neural network architectures," in *System on Chip (SoC), 2010 International Symposium on*, Sep. 2010, pp. 139–145. [Online]. Available: http://dx.doi.org/10.1109/ISSOC.2010.5625566

Contributed Publications

[1] Snaider Carrillo, Jim Harkin, Liam McDaid, **Sandeep Pande**, Seamus Cawley, Brian McGinley, Fearghal Morgan, "Hierarchical network-on-chip and traffic compression for spiking neural network implementations," in *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, May 2012, pp. 83–90. [Online]. Available: http://dx.doi.org/10.1109/NOCS.2012.17

[2] Snaider Carrillo, Jim Harkin, Liam McDaid, Fearghal Morgan, **Sandeep Pande**, Seamus Cawley, Brian McGinley, "Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations," *Parallel and Distributed Systems, IEEE Transactions on*, no. 99, Oct. 2012. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2012.289

[3] Snaider Carrillo, Jim Harkin, Liam McDaid, **Sandeep Pande**, Seamus Cawley, Brian McGinley, Fearghal Morgan, "Advancing interconnect density for spiking neural network hardware implementations using traffic-aware adaptive network-on-chip routers," *Neural Networks*, vol. 33, pp. 42–57, Sep. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2012.04.004

[4] Snaider Carrillo, Jim Harkin, Liam McDaid, **Sandeep Pande**, Seamus Cawley, Fearghal Morgan, "Adaptive routing strategies for large scale spiking neural network hardware implementations," in *Artificial Neural Networks and Machine Learning. ICANN 2011*, vol. 6791. Springer Berlin Heidelberg, Jun. 2011, pp. 77–84. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-21735-7_10

[5] Seamus Cawley, Fearghal Morgan, Brian McGinley, **Sandeep Pande**, Liam McDaid, Snaider Carrillo, Jim Harkin, "Hardware spiking neural network prototyping and application," *Genetic Programming and Evolvable Machines*, vol. 12, pp. 257–280, Sep. 2011. [Online]. Available: http://dx.doi.org/10.1007/s10710-011-9130-9

[6] Snaider Carrillo, Jim Harkin, Liam McDaid, **Sandeep Pande**, Fearghal Morgan, "An efficient, high-throughput adaptive NoC router for large scale spiking neural network hardware implementations," in *Evolvable Systems: From Biology to Hardware*, vol. 6274. Springer Berlin Heidelberg, Sep. 2010, pp. 133–144. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15323-5_12

[7] Seamus Cawley, Fearghal Morgan, Brian McGinley, **Sandeep Pande**, Liam McDaid, Jim Harkin, "The impact of neural model resolution on hardware spiking neural network behaviour," in *Signals and Systems Conference (ISSC 2010), IET Irish*, Jun. 2010, pp. 216–221. [Online]. Available: http://dx.doi.org/10.1049/cp.2010.0515

[8] Fearghal Morgan, Seamus Cawley, Brian McGinley, **Sandeep Pande**, Liam McDaid, Brendan Glackin, John Maher, Jim Harkin, "Exploring the evolution

of NoC-based spiking neural networks on FPGAs," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, Dec. 2009, pp. 300–303. [Online]. Available: http://dx.doi.org/10.1109/FPT.2009.5377663

[9] Fearghal Morgan, Seamus Cawley, Jim Harkin, Brian McGinley, Liam McDaid, **Sandeep Pande**, "An evolvable NoC-based spiking neural network architecture," in *Signals and Systems Conference (ISSC 2009), IET Irish*, Jun. 2009, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1049/cp.2009.1732

CHAPTER 1

Introduction

1.1 Introduction

Conventional computing paradigms are suitable for applications comprising a series of well-defined calculations. Next generation complex embedded applications are characterised by contradicting functional requirements in unexplored data and application scenarios resulting in the lack of an exact computational algorithm. This situation is pushing the limits of system complexity and size, and makes it even harder to assure system performance and reliability due to hardware faults and software complexity.

Biological and nature inspired computing paradigms such as neural networks and evolutionary computation can provide promising solutions for designing complex and intelligent systems [1–4]. The organic central nervous system includes a dense and complex interconnection of neurons and synapses, where each neuron connects to thousands of other neurons through synaptic connections. The brain-inspired Spiking Neural Network (SNN) computing paradigm offers the potential for elegant, low-power and scalable embedded computing, with rich nonlinear dynamics, ideally suited to applications including classification, estimation, prediction, dynamic control and signal processing [5, 6].

The main design challenges for realisation of practical hardware SNN systems are large scale neural simulations and performance measurement, compact hardware implementation, architectural scalability, application reliability, low power consumption, efficient learning/training algorithms, implementation of complex neural models, fault tolerance and efficient application design.

This research has contributed to the development of EMBRACE[1], a compact, scalable, modular, hardware SNN architecture as an embedded computing platform. The prototype implementation of the EMBRACE architecture on Xilinx Virtex-6 XC6VLX240T FPGA comprises 448 neurons, 32K synapses and demonstrates reliable, practical embedded classifier and control applications. The thesis contributes to a number of hardware SNN system design challenges [7] such as

- Hardware SNN simulations and performance measurement
- Compact hardware implementation
- Architectural scalability
- Application reliability
- Efficient practical application design

This research has developed a structured system level design methodology for EMBRACE, a compact scalable hardware modular SNN architecture comprising a hierarchical, mesh and ring topology Network on Chip (NoC) using novel architectural techniques and demonstrated modular application design on the EMBRACE-FPGA prototype [8–12].

1.1.1 Previously Reported EMBRACE Architecture and Applications Research

This section summarises previously reported research on the development of the EMBRACE hardware SNN system and provides the foundation for the research contributions presented in the thesis. The collaborative research has been accomplished with contributions from Dr. Fearghal Morgan, Dr. Brian McGinley, and Dr. Seamus Cawley (Bio-Inspired Electronics and Reconfigurable Computing (BIRC),National University of Ireland, Galway, Ireland), and Dr. Jim Harkin and Dr. Liam McDaid (Intelligent Systems Research Centre (ISRC),University of Ulster, Derry, Northern Ireland, UK).

The EMBRACE architecture has been initially conceptualised and reported as a mixed-signal hardware SNN array [13]. EMBRACE 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 switched Network on Chip (NoC) spike communication architecture[13, 14]. A number of benchmark SNN applications such as XOR data classifier and inverted pendulum controller have been successfully evolved on

[1] **EM**ulating **B**iologically-inspi**R**ed **A**r**C**hitectures in hardwar**E**

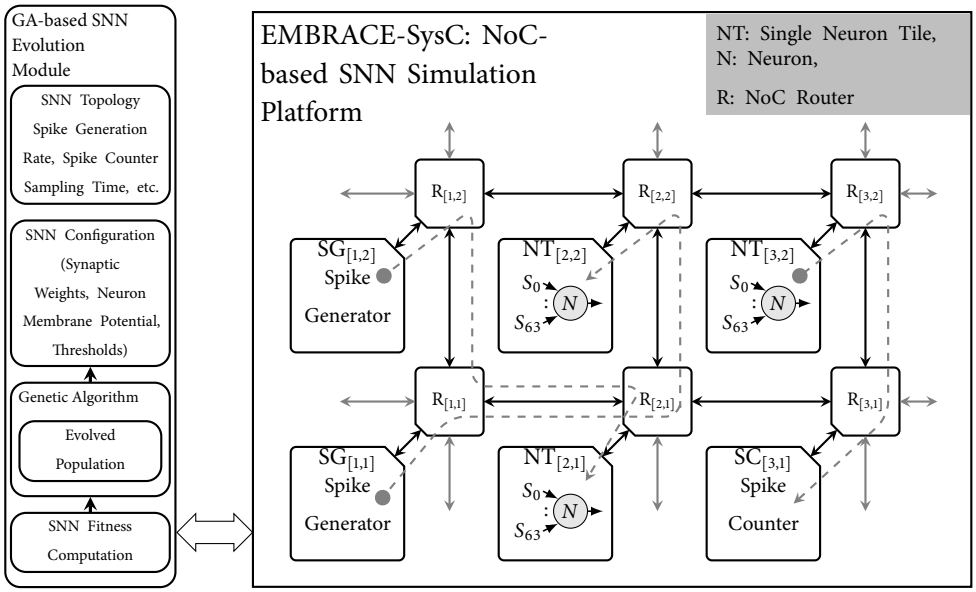


Figure 1.1: XOR Benchmark SNN Application Simulated with the EMBRACE-SysC Architecture in a 3×2 Mesh Configuration and GA-based SNN Evolution Module

the EMBRACE-FPGA prototype [14]. Also, real-world embedded applications including Wisconsin breast cancer dataset classifier and robotic controller have been successfully implemented on the first EMBRACE-FPGA prototype [14–16].

1.2 Thesis Contributions and Novelty Claims

This research has systematically developed the EMBRACE, compact, scalable, modular, hardware SNN architecture. The research follows four distinct phases, where each phase contributes to a hardware SNN system design challenge outlined in section 1.1.

1.2.1 Simulation and Performance Measurement

System design for embedded hardware SNN architecture offers a number of architectural choices which have significant impact on the area, power and performance of the system. The thesis presents the EMBRACE-SysC, a SystemC simulation-based design exploration framework for the EMBRACE hardware SNN architecture. EMBRACE-SysC models each of the EMBRACE architecture elements in a modular fashion at a Timed Functional (TF) abstraction level using SystemC [17]. EMBRACE functionality and interfaces are modelled with clock cycle accuracy and mimic the RTL functionality and pin interface behaviour (using standard Transac-

tion Level Modelling (TLM) interface methods). Figure 1.1 illustrates modelling of XOR benchmark SNN application on the EMBRACE-SysC simulation platform[2]. During simulation, all EMBRACE-SysC modules send important system activity information to the performance measurement modules, which collate and store this information in a database for performance reporting. EMBRACE-SysC can simulate various SNN topologies, SNN application configurations and EMBRACE architectural configurations. In summary, EMBRACE-SysC enables:

- Accurate modelling of spike data traffic through clock cycle accurate modelling of all EMBRACE digital components and timed modelling of the analogue CMOS neuron cell
- Systematic investigation of performance bottlenecks (e.g. traffic hotspots) in the architecture
- Performance comparison and analysis of architectural design choices
- Faster architecture design, prototyping and simulation (compared to RTL simulation)
- Verification of specifications for EMBRACE design enhancements, down to clock cycle accuracy
- Studying of SNN application feasibility prior to hardware prototyping

This thesis demonstrates the performance measurement capabilities of the EMBRACE-SysC framework including measurement of spike injection rate for error-free spike communication and NoC hotspot detection due to NoC router bandwidth exhaustion [8]. EMBRACE-SysC offers a fast executable specification of the EMBRACE architecture and performance measurement capabilities including spike communication infrastructure, neuron model validation, hardware architecture design exploration and SNN application evolution, which are extensively used in later phases of this research.

1.2.2 Architectural Techniques for Scalability

The Network on Chip (NoC) design paradigm provides a promising solution for the flexible interconnection of large SNNs [18]. Storage of large synaptic connectivity (SNN topology) information in SNNs requires large distributed on-chip memory, which poses serious challenges for compact hardware implementation of NoC

[2] The array indexing in EMBRACE architecture figures in chapter 1, 2 and 3 start at [1,1] and the rest of the thesis uses [0,0] as the start array index to maintain consistency with the published manuscripts.

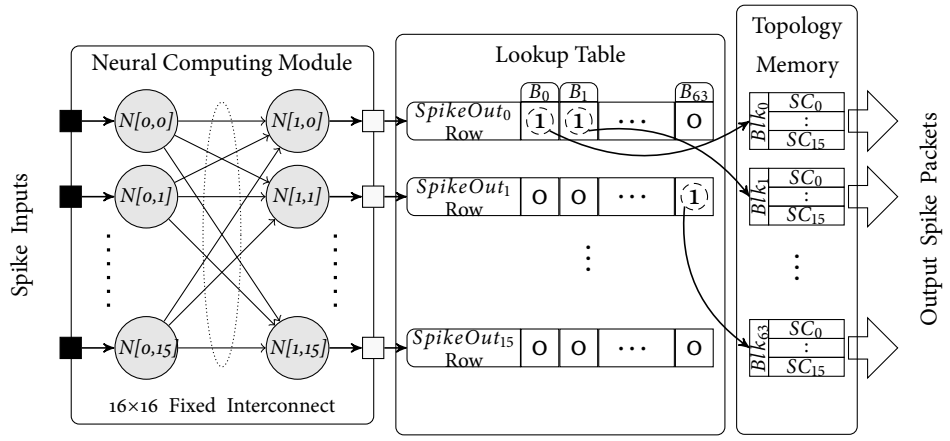


Figure 1.2: Two layered 16:16 Fully Connected SNN Structure as the Neural Computing Module and the Lookup Table-based Shared Topology Memory Organisation within the Proposed Modular Neural Tile

based hardware SNN architectures. This thesis presents an elaborate analysis of the SNN topology memory requirements and its impact on the area and scalability of NoC based hardware SNN architectures. The thesis presents the Modular Neural Network (MNN) design approach to reduce SNN topology memory and increase scalability of the EMBRACE architecture.

The thesis presents a hardware Modular Neural Tile (MNT) tile architecture (figure 1.2), that reduces the SNN topology memory requirement of NoC-based hardware SNNs by using a combination of fixed and configurable synaptic connections. A two-layered 16:16 fully connected feed-forward SNN structure (figure 1.2) as the Neural Computing Module (NCM) inside each MNT[9, 10]. Fixed interconnection between neurons within the NCM removes the need for storage of synaptic connectivity information, reducing the SNN topology memory requirement of the architecture to 50% compared to the reported monolithic EMBRACE NoC based hardware SNN implementation [13, 14]. A novel lookup table based SNN topology memory sharing technique further increases the memory utilisation efficiency. Results demonstrate an average 66% reduction in the overall area requirement of the architecture for practical SNN application topologies compared to the previously reported EMBRACE architecture [14].

The thesis presents the micro-architecture details of the proposed MNT and the digital neuron circuit used for system validation. The architectural components are synthesised using 65nm low-power CMOS technology and silicon area results are presented. The proposed MNT architecture has been validated on a Xilinx Virtex-6 FPGA and resource usage results are reported. The evolvable capability

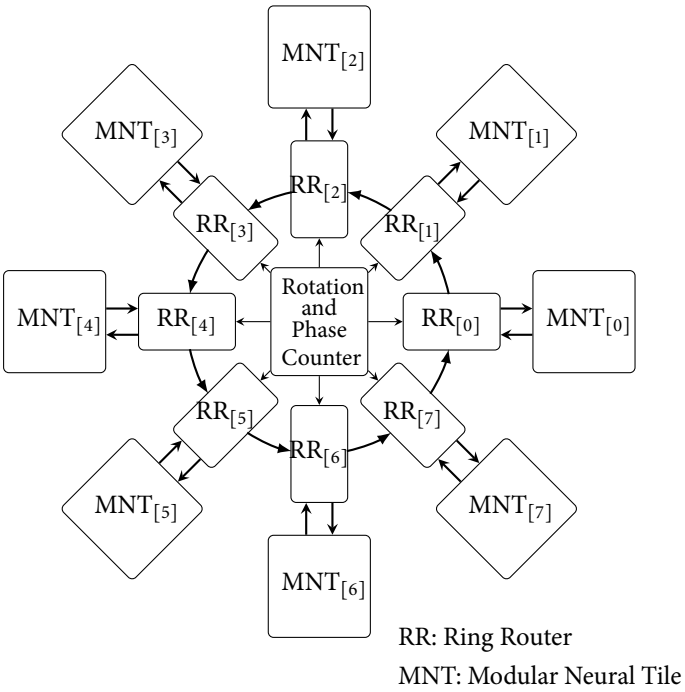


Figure 1.3: Ring Topology Interconnect Architecture

of the proposed MNT and its suitability for executing application subtasks in an MNN execution architectures is demonstrated by successfully evolving the XOR benchmark SNN application and a robotics obstacle avoidance controller using the player-stage robotics simulator and previously reported Genetic Algorithm based hardware SNN evolution platform [14]. These benchmark SNN applications represent data classification and non-linear control functions.

1.2.3 Interconnect Architecture for SNN Application Reliability

Shared resources in packet-switched NoC architectures can result in unwanted variation in packet transfer latency. This packet latency jitter alters spike timings and distorts the information conveyed on the synaptic connections in the SNN, resulting in unreliable application behaviour. The thesis presents an elaborate analysis of the spike transfer latency variations in mesh topology, packet switched NoC architectures and it's impact on the information flowing in the SNN.

The thesis presents a novel ring topology interconnect illustrated in figure 1.3, for spike communication between neural tiles and a novel timestamped spike

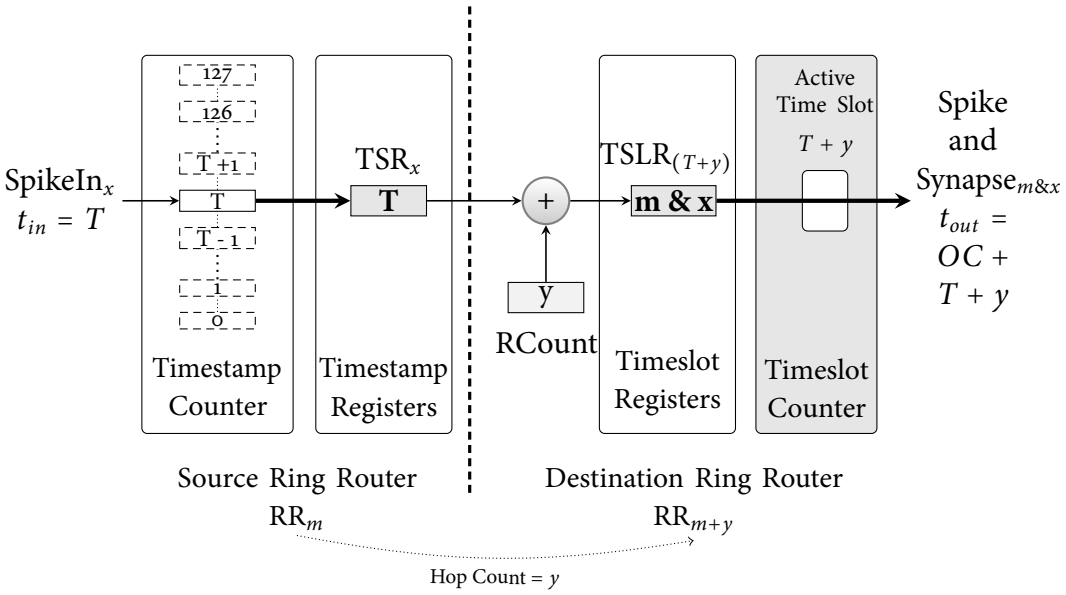


Figure 1.4: Fixed Latency Spike Packet Flow Control

broadcast flow control scheme that offers fixed spike transfer latency for all the synaptic connections within the ring [11]. Figure 1.4 illustrates the flow control of the proposed ring topology interconnect which treats each spike event timing separately, maintaining the clock cycle count of the spike event occurrence (in timestamp registers TSR$_x$). The recorded spike event clock cycle count is preserved throughout the spike packet flow control. At the destination ring router, the spike event occurrence clock cycle count (stored in timeslot registers TSLR$_{T+y}$) is used to send out the spike at the precise clock cycle (with respect to spike event occurrence). This ensures fixed spike transfer latency and maintains the integrity of the SNN information flow.

The thesis presents the micro-architecture details of the ring router and its ASIC and FPGA synthesis results. Spike transfer latency results are presented for the ring interconnect under various SNN spike traffic conditions. The ring interconnect offers fixed spike transfer latency under various spike traffic densities making it suitable as localised spike communication architecture for hardware SNN architectures and ensures reliable SNN application behaviour [11]. The hierarchical NoC design comprising ring topology interconnect within a packet switched mesh topology network of routers is described.

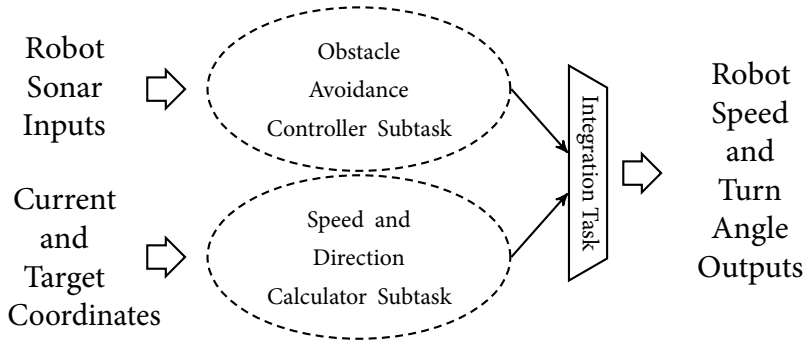


Figure 1.5: Robotic Navigational Controller Application Organisation

1.2.4 Modular Application Design

Solutions for real-world applications often lack deterministic algorithms. The inherent fuzzy nature of these applications pose a serious challenge for their implementation on hardware SNN architectures. Although, the Genetic Algorithm (GA) based search technique offers a practical way of prototyping applications on hardware SNN platforms, the technique has a number of limitations including poor scalability and search space explosion which restricts its use for evolution of complex SNN applications. These problems can be mitigated by the classic *divide and conquer* technique, by defining the overall behaviour as simple orthogonal functions. Structured partitioning and modular evolution of the overall application functionality has potential in the design of complex, multifunctional real-world embedded SNN applications.

The thesis demonstrates the modular robotic navigational controller application evolved on the EMBRACE-FPGA prototype [12]. Figure 1.5 illustrates the complex multifunctional robotic navigational controller application decomposed into the Obstacle Avoidance Controller (OAC) and the Speed and Direction Manager (SDM) application subtasks. The OAC subtask controls the robot movement for avoiding obstacles and the SDM subtask steers the robot towards target location. Figure 1.6 illustrates the robotic controller application evolution setup comprising the EMBRACE hardware SNN architecture FPGA prototype interfaced with player-stage robotics simulator [19] and GA based hardware SNN evolution platform. The individual application subtasks (OAC and SDM) are evolved separately and integrated using an integration module.

The modular application design for the EMBRACE hardware modular SNN architecture results in faster application evolution (compared to monolithic SNN

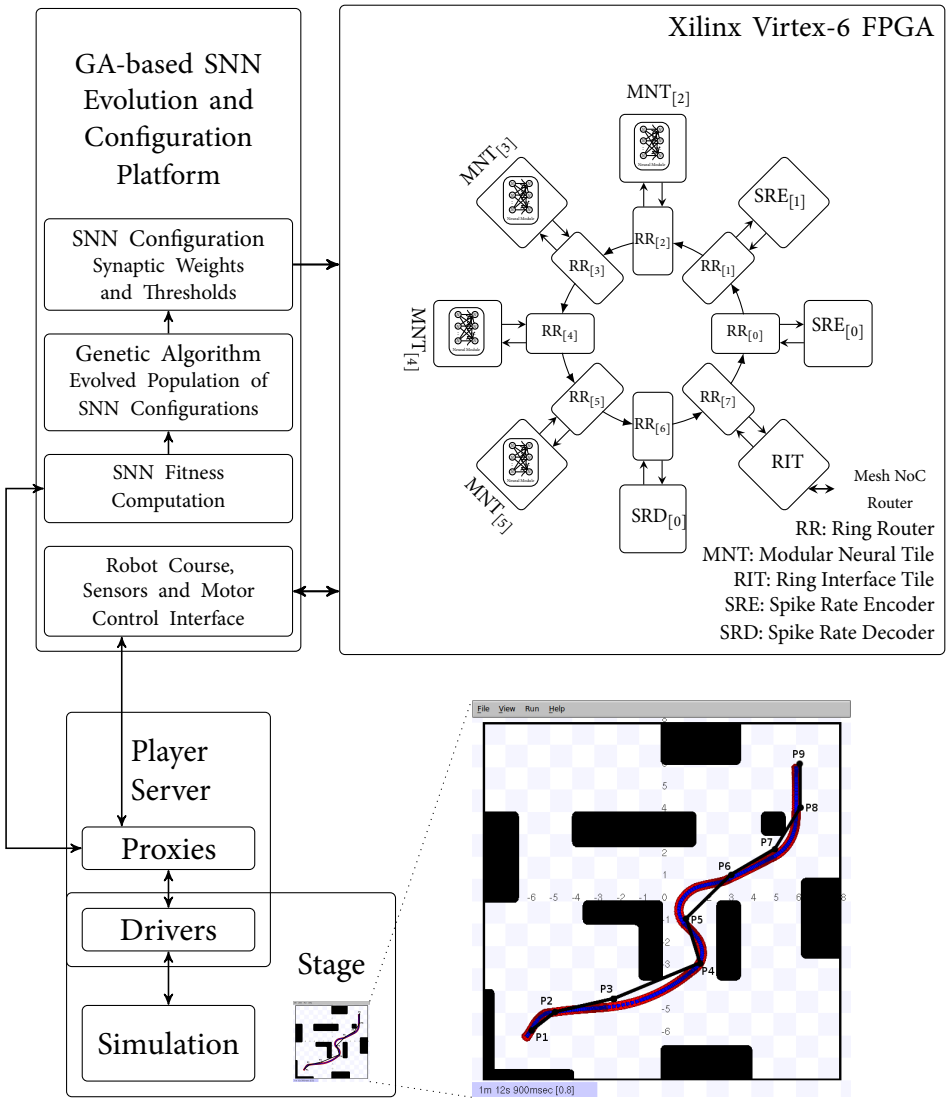


Figure 1.6: Robotic Controller Evolution Setup

evolution) through stepwise knowledge integration and simplified SNN training. This thesis presents a rapid and effective application prototyping technique for the EMBRACE hardware SNN architecture using the modular application partitioning and evolution technique.

1.3 Thesis Structure

The thesis follows an article based presentation style, where many of the chapters are peer reviewed publications. All articles are preceded by a preamble outlining the research challenge and the main contributions of the article. References for

each article are listed at the end of each chapter.

Chapter 2 presents the Spiking Neural Network (SNN) computing paradigm and reviews related research work. The chapter also presents the previously reported EMBRACE architecture concept and application prototyping.

Chapter 3 presents EMBRACE-SysC, a simulation-based design exploration framework for the EMBRACE Network on Chip (NoC)-based hardware SNN architecture [8].

Chapter 4 presents a hardware Modular Neural Tile (MNT) architecture that reduces the SNN topology memory requirement of NoC-based hardware SNNs [9, 10].

Chapter 5 presents the novel ring topology interconnect for spike communication between neural tiles, and a novel timestamped spike broadcast flow control scheme that offers fixed spike transfer latency [11].

Chapter 6 presents the modular SNN application prototyping technique for the EMBRACE modular hardware SNN architecture [12].

Chapter 7 concludes the thesis and proposes future work. The chapter summarises the contribution of the research towards building a compact, scalable, modular hardware SNN system and applications.

References

[1] J. Liu and K. Tsui, "Toward nature-inspired computing," *Commun. ACM*, vol. 49, no. 10, pp. 59–64, Oct. 2006.

[2] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999, vol. 13.

[3] S. Bohte and J. Kok, "Applications of spiking neural networks," *Information Processing Letters*, vol. 95, no. 6, pp. 519–520, 2005.

[4] M. Pearson, A. Pipe, B. Mitchinson, K. Gurney, C. Melhuish, I. Gilhespy, and M. Nibouche, "Implementing spiking neural networks for real-time signal-processing and control applications: A model-validated fpga approach," *Neural Networks, IEEE Transactions on*, vol. 18, no. 5, pp. 1472 –1487, sept 2007.

[5] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[6] W. Gerstner and W. Kistler, *Spiking neuron models*. Cambridge University Press, 2002.

[7] L. Maguire, T. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for large-scale implementations of spiking neural networks on FPGAs," *Neurocomputing*, vol. 71, no. 1-3, pp. 13–29, Dec. 2007.

[8] S. Pande, F. Morgan, S. Cawley, B. McGinley, S. Carrillo, J. Harkin, and L. McDaid, "EMBRACE-SysC for analysis of NoC-based spiking neural network architectures," in *System on Chip (SoC), 2010 International Symposium on*, Sept. 2010, pp. 139–145.

[9] S. Pande, F. Morgan, S. Cawley, B. McGinley, J. Harkin, S. Carrillo, and McDaid, "Addressing the hardware resource requirements of Network-on-Chip based neural architectures," in *NCTA, International Conference on Neural Computation Theory and Applications, Paris, France*, Oct. 2011.

[10] S. Pande, F. Morgan, S. Cawley, T. Bruintjes, G. Smit, B. McGinley, S. Carrillo, J. Harkin, and L. McDaid, "Modular neural tile architecture for compact embedded hardware spiking neural network," *Neural Processing Letters*, vol. 38, pp. 131–153, Oct. 2013.

[11] S. Pande, F. Morgan, G. Smit, T. Bruintjes, J. Rutgers, B. McGinley, S. Cawley, J. Harkin, and L. McDaid, "Fixed latency on-chip interconnect for hardware spiking neural network architectures," *Parallel Computing*, vol. 39, pp. 357 – 371, Sep. 2013.

[12] S. Pande, F. Morgan, B. McGinley, J. Harkin, and M. L., "Application prototyping for embrace hardware modular spiking neural network architecture," *Neural Compting and Applications Journal (To be submitted)*, 2013.

[13] J. Harkin, F. Morgan, L. McDaid, S. Hall, B. McGinley, and S. Cawley, "A reconfigurable and biologically inspired paradigm for computation using network-on-chip and spiking neural networks," *Int. J. Reconfig. Comput.*, vol. 2009, pp. 1–13, 2009.

[14] 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*, vol. 12, pp. 257–280, Apr. 2011.

[15] F. Morgan, S. Cawley, B. McGinley, S. Pande, L. McDaid, B. Glackin, J. Maher, and J. Harkin, "Exploring the evolution of NoC-based spiking neural networks on FPGAs," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, 2009, pp. 300–303.

[16] F. Morgan, S. Cawley, J. Harkin, B. McGinley, L. McDaid, and S. Pande, "An evolvable noc-based spiking neural network architecture," in *Signals and Systems Conference (ISSC 2009), IET Irish*. IET, 2009, pp. 1–6.

[17] "IEEE standard for standard systemc language reference manual," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pp. 1–638, 2012.

[18] D. Vainbrand and R. Ginosar, "Scalable network-on-chip architecture for configurable neural networks," *Microprocessors and Microsystems*, vol. 35, no. 2, pp. 152–166, 2011, special issue on Network-on-Chip Architectures and Design Methodologies.

[19] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm Intelligence*, vol. 2, pp. 189–208, Aug. 2008.

CHAPTER 2

Background and Related Work

2.1 Introduction

This chapter presents the Spiking Neural Network (SNN) computing paradigm and reviews related research work. Section 2.2 introduces the SNN computing paradigm. Section 2.3 discusses the design challenges for the realisation of hardware SNN architectures. Section 2.4 reviews the current research in hardware SNN architectures. Section 2.5.1 summarises the previously reported EMBRACE architecture concept and section 2.5.2 presents EMBRACE application prototyping.

2.2 Spiking Neural Network Computing Paradigm

The organic nervous system has been extensively studied to understand the cognitive abilities of human brain. SNN based computing systems emulate real biological neural networks to realise arbitrary functions for real-world applications. This section discusses spiking neuron models, information coding, network connection topology and training/learning algorithms for SNN computing paradigm.

2.2.1 Biological Neuron Structure and Operation

Biological neurons (also known as *neurones* or *nerve cells*) process information through a collection of electro-chemical processes [1]. Figure 2.1 outlines the physiology of neural cell. The electrically excitable neuron cell receives current pulses on its dendrites, which are summed in the cell body (or *Soma*). The electrical charge accumulates in the soma and if it exceeds a certain threshold value, the

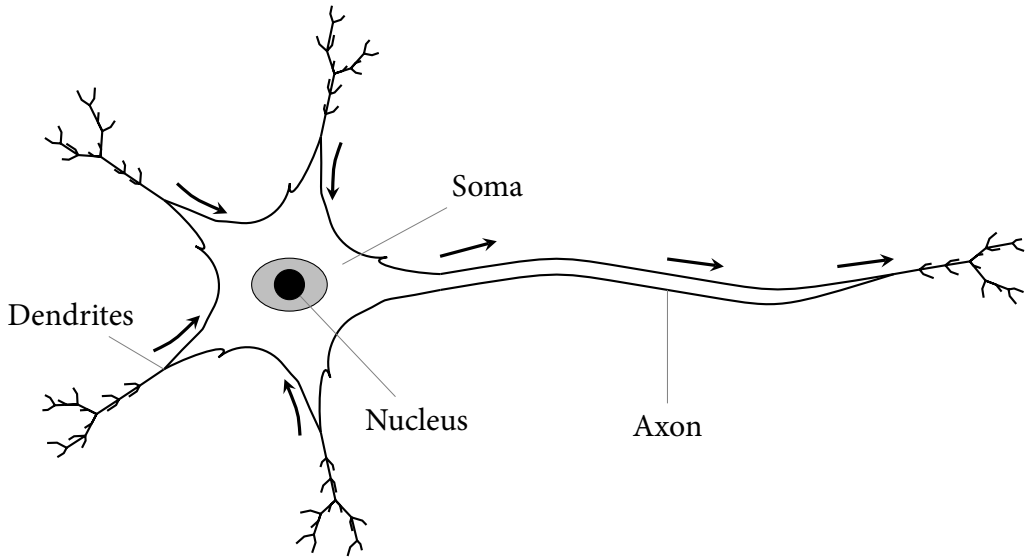


Figure 2.1: Physiology of a Biological Neuron

cell emits an electrical pulse through the axon. The generated electrical pulse propagates along the axon which connects to other thousands of neurons/synapses through dendrites. The cognitive abilities of the human brain have been partially explained by the massive number of densely interconnected small processing elements (neurons), working in parallel to solve a specific problem.

2.2.2 Spiking Neuron Models

Biological neurons exhibit complex behaviour comprising a series of electrochemical processes which are difficult to capture mathematically. The electrical pulses received on dendrites are summed in the cell body. The cell emits an electrical pulse through the axon after the potential reaches the threshold level. For an artificial SNN system the neuron can be modelled as a multi-input single output unit with a non-linear transfer function.

Artificial SNN systems are realised using a variety of neuron models with various degrees of abstraction. Neuron models with high biological plausibility have higher computational requirements compared to abstract models with relatively low biological resemblance. The choice of neuron model for an SNN system implementation is mainly influenced by parameters such as application response time, accuracy requirements and computational resources in the platform.

Hodgkin-Huxley (HH) model mimics the behaviour of a biological neuron by modelling different ionic channels of the cell membrane [2, 3]. The HH model

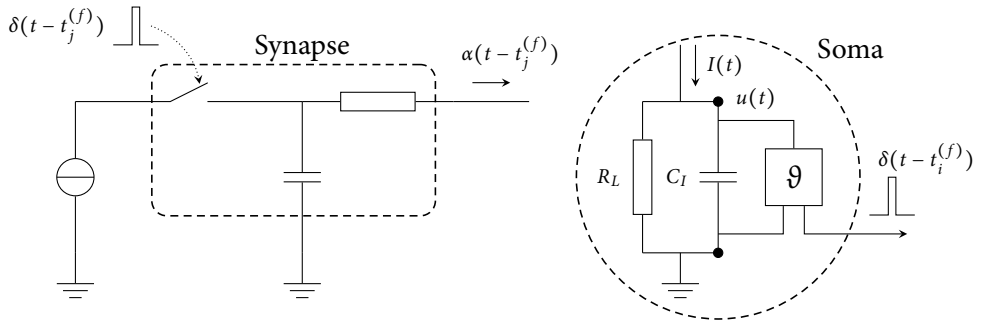
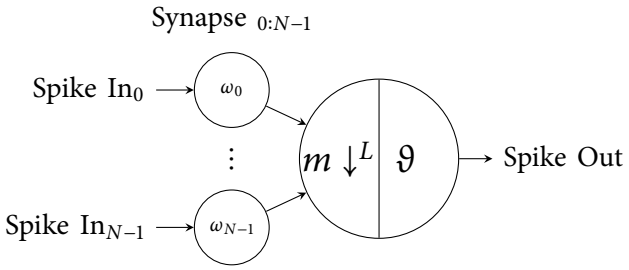


Figure 2.2: Leaky Integrate and Fire Model Schematic



w_i = Synaptic Weight
m = Membrane Potential
L = Leakage Coefficient
ϑ = Firing Threshold

Figure 2.3: Simplified Leaky-Integrate-Fire Neuron Model

which has high biological plausibility, offers a high level of accuracy. However, high computational complexity due to a large number of differential equations in the HH mathematical model limits its use in large sized SNN systems.

Based on the dynamics of the biological neuron, the Integrate-and-Fire (IF) and Leaky-Integrate-and-Fire (LIF) neuron models have been proposed [3]. These simplified neuron models abstract out the properties of the biological neuron related to its spatial structure, and model the charge build-up behaviour of the neuron as an integrator.

Figure 2.2 illustrates the LIF neuron model as a simplified schematic. The *Synapse* (or the synaptic junction) is modelled as a RC filter, where a pre-synaptic spike ($\delta(t-t_j^{(f)})$) is low-pass filtered and a proportional current pulse ($\alpha(t-t_j^{(f)}) = w_j\delta(t-t_j^{(f)})$) is generated (where w_j is the synaptic weight). The leaky integration behaviour of the *Soma* is modelled as an RC circuit. The integrating capacitor (C_I) charges through summing of all synaptic currents ($I(t) = \Sigma\alpha(t-t_j^{(f)})$) and also

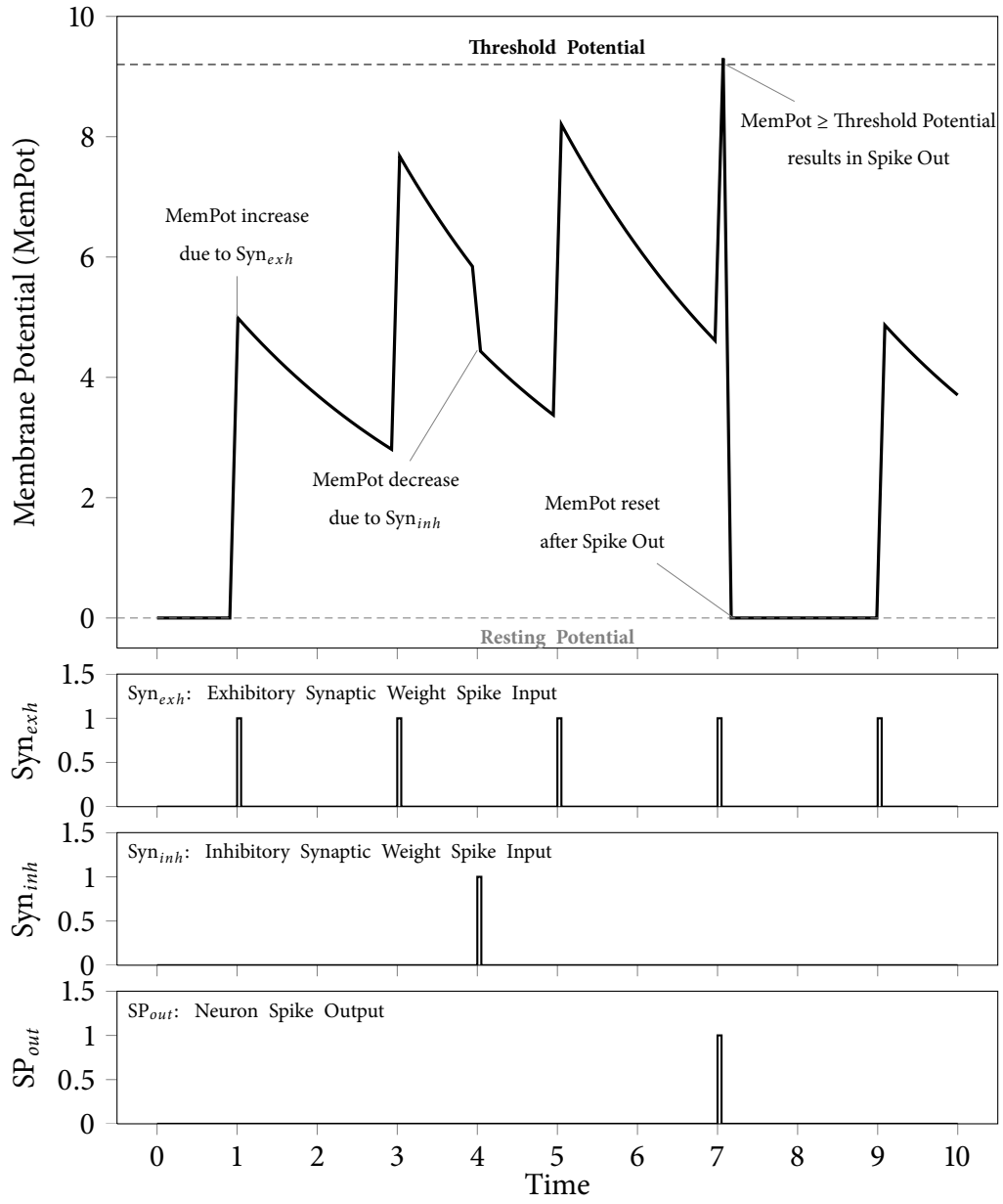


Figure 2.4: Leaky-Integrate-Fire Neuron Characteristic

leaks through the resistor (R_L) constantly. The voltage ($u(t)$) across the capacitor is compared to the firing threshold (ϑ). If $u(t) \geq \vartheta$ at time $t_i^{(f)}$, an output pulse $\delta(t - t_i^{(f)})$ is generated and the capacitor voltage $u(t)$ resets to its resting value. Figure 2.3 shows a simplified LIF neuron symbol with spike inputs, synaptic weights and firing threshold, suitable for SNN implementation.

Figure 2.4 illustrates the detailed characteristic of a practical LIF neuron implementation depicting membrane potential variations due to input spikes on

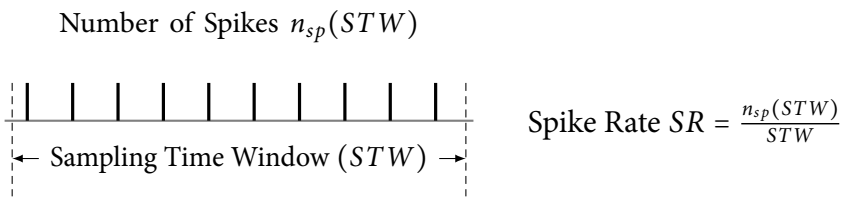


Figure 2.5: Spike Rate Coding

inhibitory ($w_i < 0$) and excitatory ($w_i > 0$) synapses and output spike generation. Spike input on an input synapse causes the membrane potential to change instantaneously based on the associated synaptic weight. As the membrane potential reaches the firing threshold, the neuron emits an output spike and the membrane potential resets to its resting value.[1]

The Izhikevich spiking neuron model mimics the spiking and bursting behavior of cortical neurons [4]. The model combines the biological plausibility of the Hodgkin-Huxley-type dynamics and the computational efficiency of integrate-and-fire neurons.

A detailed survey and analysis of spiking neuron models is presented in [3].

2.2.3 Spiking Neural Network Information Coding

Understanding biological neuronal coding is one of the fundamental issues in neuroscience [1]. Traditional hypothesis suggest that the information is encoded in the mean firing rate of neurons, whereas experimental results on response time of humans (for a number of senses) suggest that the neuronal information is encoded in the relative timing between spikes [3, 5–7]. Based on these findings, information encoding in artificial SNN systems can be broadly categorised as:

1. **Rate Coding:** This coding technique is based on encoding the SNN information in 'mean firing rate' of neurons. Based on different notions of the mean firing rate, this category is further subdivided in the three averaging procedures, namely *Rate as a Spike Count*, *Rate as a Spike Density* and *Rate as a Population Activity* [8].

   For practical SNN systems, this coding technique encodes the spike count within a fixed time interval (sampling time window) as depicted in figure 2.5.

[1] This research employs digital LIF neuron model circuit for FPGA prototyping and application validation.

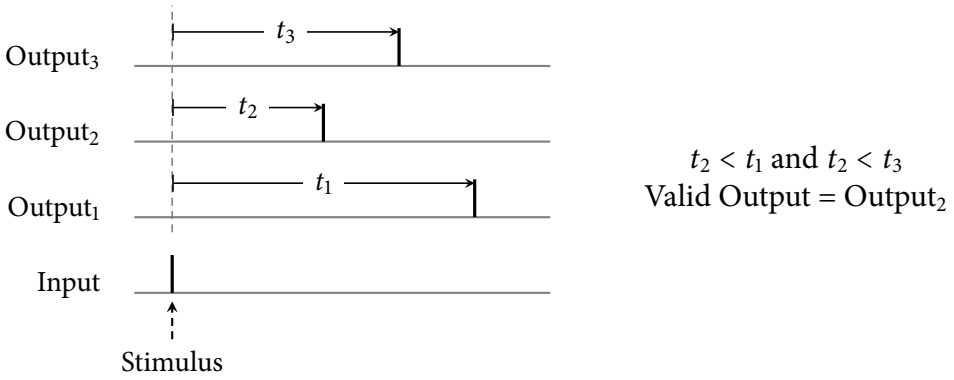


Figure 2.6: Temporal Coding (Time to First Spike Coding Scheme)

The time interval or sampling time of the SNN is often determined by the application response time and stimulus encoding resolution requirements. Due to its simplicity, the spike rate coding technique is a popular choice for embedded applications.

2. **Temporal Coding:** This coding technique encodes information based on the 'precise spike timings'. For practical SNN systems, this coding technique can be implemented as *Time to First Spike*, *Spike Phase Encoding* and *Spike Correlation Encoding* [7]. Figure 2.6 depicts the time to first spike coding scheme as applied to decide the valid SNN output.

   The absence of sampling time reduces the response time of this coding technique and speeds-up the application, but the implementation complexity limits its use in practical SNN systems.

2.2.4 Neural Network Topologies

The biological nervous system exhibits clustered random neural connection topologies [9]. SNN topologies for practical implementations are generally organised in ordered fashion. Based on the network organisation, the SNN topologies can be broadly categorised as:

- **Single/Multi-Layer Feedforward Networks:** This neural network topology illustrated in figure 2.7 is characterised by a simple organisation and unidirectional information flow. The neurons are organised in the form of single or multiple layers, where each layer receives information from the previous layer and feeds to the next layer, and the network does not contain any loops.

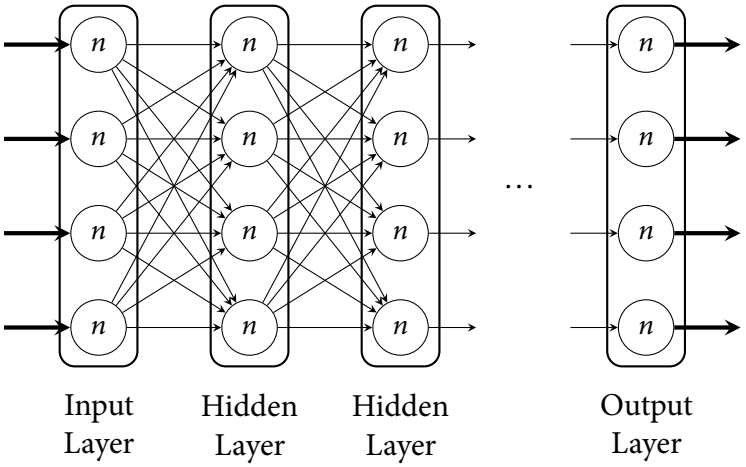


Figure 2.7: An Example Feedforward Neural Network Topology

Feedforward networks process the information in a pipelined fashion, without maintaining the state information. These network topologies are ideally suited for applications where the output is only dependent on the present input state [10, 11].

- **Recurrent Networks:** This neural network topology exhibits neural connections in the form of a directed cycle, where neurons connect to one or more neurons from previous layers as depicted in figure 2.8. This information feedback mechanism creates an internal state of the network, allowing it to exhibit dynamic temporal behaviour [12, 13].

  Recurrent networks can use the internal state information along with input state to produce accurate results. This property makes them suitable for pattern recognition and prediction applications [14].

- **Hybrid Networks:** Recently, the hybrid network topology, which comprises groups of interconnected subnetworks has been proposed [15]. The individual subnetworks follow either feedforward or recurrent topology and the subgroup interaction is either directional or reciprocal as depicted in figure 2.9[2].

[2]These topologies closely resemble Modular Neural Network (MNN) topologies (discussed in section 2.4.2).

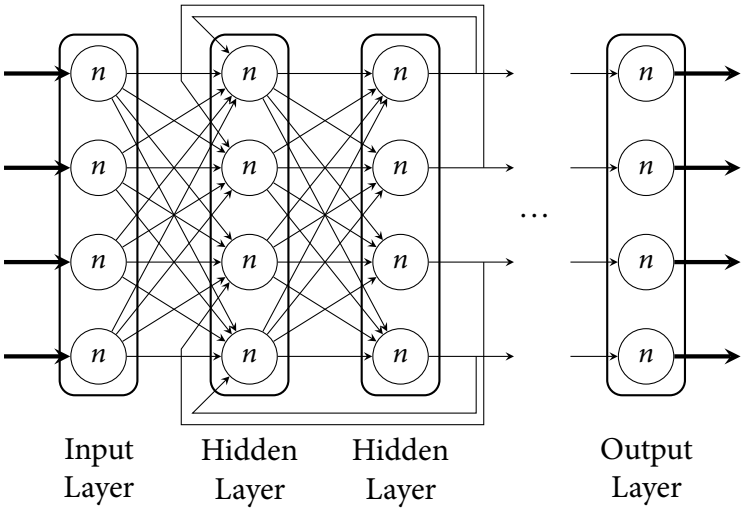


Figure 2.8: An Example Recurrent Artificial Neural Network Topology

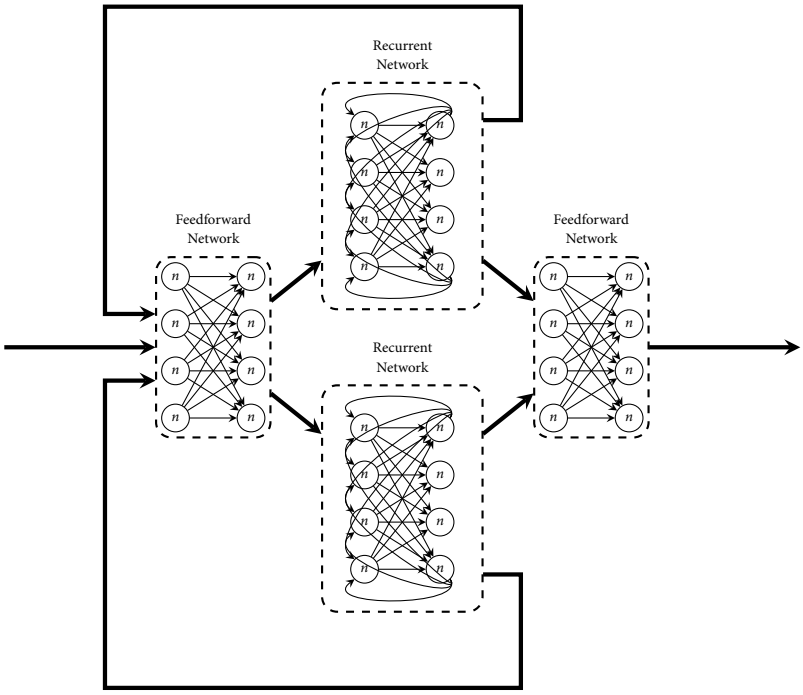


Figure 2.9: An Example Hybrid Artificial Neural Network Topology

2.2.5 Neural Network Training

Neural networks are characterised by their ability to learn and provide a generalised solution in unexplored application scenarios. The SNN training process involves iterative adjustment of the SNN configuration (synaptic weights, threshold potential and/or network topology) to achieve the desired application behaviour.

Based on the configuration process, the neural network training algorithms can be broadly classified as:

- **Unsupervised Training Algorithms:** These training algorithms attempt to find the hidden structures within input data patterns. The technique relies on tuning the neural network to statistical regularities of the input data, such that the neural network correctly responds to these data patterns.

  Hebbian learning suggests strengthening of the synaptic weight w_{ij}, whenever neurons i and j fire simultaneously [16]. Hebbian learning leads to reconfiguration of SNN and results in emergence of new functions, such as pattern recognition and associative memories [17]. The probability of a neuron firing based on the pre-synaptic spike timing is termed as Spike Timing Dependent Plasticity (STDP). STDP based unsupervised SNN training techniques have been effective for data/pattern classification, pattern recognition and associative memories [12, 18–22].

- **Supervised Training Algorithms:** These algorithms are primarily characterised by the iterative adjustment of the neural network configuration based on feedback. Back-Propagation and Spike-Prop SNN training algorithms are the most popular supervised neural network training algorithms [23, 24].

  Supervised Hebbian Learning (SHL) offers a biologically realistic implementation of Hebbian learning rules. SHL employs an additional 'teaching' signal that reinforces the post-synaptic neuron to fire at the target times and to remain inactive otherwise. Detailed analysis of SHL for spiking neurons is presented in [25, 26].

- **Reinforcement Training Algorithms:** These algorithms tune the neural network configurations to achieve a predetermined goal, while interacting with the environment. The online neural network configuration exploration technique relies on rewarding performance improving configurations and avoiding performance deteriorating configurations [27–32]. These algorithms have significant potential due to their high biological plausibility.

- **Evolutionary Training Algorithms:** Evolutionary methods have been successfully applied for SNN training and evolution of robotic applications [33, 34]. GA based SNN evolution is a supervised SNN training method that maintains a population of SNN configurations and uses nature inspired evolutionary techniques (comprising selection, crossover and mutation) to

find a correct set of SNN configuration by evaluating the desired behaviour or 'fitness' of the individual SNN configurations. The technique neither requires any information about the application domain, nor imposes architectural constraints or any additional training specific elements in the neural computation components resulting in compact architecture [35]. The technique offers an easy and fast way to prototype applications on hardware SNNs.

However, the GA based SNN evolution do not scale well for large multifunctional applications having complex fitness landscape. The search space for the binary coded, integer valued, n bit SNN configuration is 2^n. Increase in the SNN size (number of neurons and synapses) increases the GA search space and complexity by the order of $O(2^n)$. The technique also requires large amount of memory for storage of SNN configurations and results in slower operation due to the iterative nature of SNN configuration evaluation. These shortcomings limit its use for evolving large real-world complex SNN applications.

2.3 SNN System Design Challenges

Research in artificial neural networks has lead to the development of numerous neuron models, network topologies, training algorithms and application scenarios. However, the realisation of SNN based embedded computing systems faces a number of challenges. This section reviews the key research and design challenges for the realisation of hardware SNN systems suitable for embedded computing applications.

2.3.1 Large Scale SNN System Simulation

Structured system level design comprising simulation based design space exploration and synthesis has been used for the design of large scale System on Chip (SoC) architectures [36]. This approach offers performance estimates of the architecture for various application scenarios at an early stage of system development. Various simulators have been proposed for the design of Network on Chip (NoC) based SoC architectures [37–39]. System design architectural choices have significant impact on the area, power and performance of the hardware system. A high

level simulation based design space exploration framework can be of great benefit for architectural design choices early in the system design phase.

This thesis presents EMBRACE-SysC a powerful SystemC simulation based design exploration framework for NoC-based hardware SNN architectures which enables performance analysis of architectural choices at an early system design stage [40].

2.3.2 Interconnect Architecture for Hardware SNNs

Practical SNN systems are characterised by a large numbers of neurons and high interconnectivity through inter-neuron synaptic connections. The aim of SNN architecture research is to create powerful neural computing platforms incorporating thousands of neurons and millions of synapses [41–48].

For large scale hardware implementation of SNNs, the neuron interconnect imposes problems due to the high levels of inter-neuron connectivity. Often the number of neurons that can be realised in hardware is limited by high fan in/out requirements [49]. Direct neuron-to-neuron interconnection exhibits switching requirements that grow exponentially with the network size. Efficient, low area and low power implementations of neuron interconnect and synaptic junctions are therefore key to achieving scalable hardware SNN implementations [49].

The NoC design paradigm provides a promising solution for the flexible interconnection of large SNNs [50]. A packet switched NoC architecture can provide a suitable communication infrastructure for hardware SNNs, offering scalable, parallel, distributed communication channels with flexible connection reconfigurability [50–52].

The main design challenges for a NoC architecture suitable for hardware SNNs are:

- **Scalable Synaptic Connectivity:** Practical SNN application topologies are characterised by a large number of synaptic connections. Storage of this large synaptic connectivity (or SNN topology) information translates to a large distributed on-chip memory in a packet switched NoC based hardware SNN architecture [53]. As the hardware SNN size scales, the SNN topology memory requirement grows quadratically and poses a serious challenge for scalability of hardware SNN architectures. Architectural techniques to reduce SNN topology memory are crucial for the efficient and compact hardware implementation.

- **Distortion-free Synaptic Connectivity:** SNNs employ spike rate and time based coding techniques, where information is primarily encoded as the relative timing between spikes. Shared resources in NoC architectures results in unwanted variation in spike packet transfer latency. This spike latency jitter distorts the SNN information, eventually leading to unreliable application behaviour. A NoC architecture offering fixed spike communication latency is a key to reliable SNN application behaviour in a NoC based hardware SNN.
- **Hierarchical Synaptic Connectivity:** SNN topologies for practical applications have been observed to exhibit clustered organisation [9]. The sparsely connected neuron clusters exhibit high density localised connections within neuron groups. Localised multicast communication schemes closely resemble the connectivity patterns typically observed in SNN application topologies and NoC architectures supporting such connectivity patterns are especially suitable for hardware SNN architectures [50].

Hence, a NoC architecture suitable for hardware SNNs should:

- support a large number of virtual synaptic communication channels
- maintain the integrity of information encoded within spike timings
- support connection topologies that closely resemble neural connectivity patterns

In summary, the NoC architectures for hardware SNN architectures should provide a large number of virtual synaptic communication channels and localised multicast connectivity patterns, while offering low spike transfer latency jitter.

This thesis presents a scalable and noise-free hierarchical NoC architecture comprising fixed spike transfer latency ring interconnect integrated within a mesh topology network of routers [54].

2.3.3 Application design for Hardware SNNs

Solutions for real-world applications often lack deterministic algorithms. Also, training of large scale complex embedded applications poses a serious challenge for their implementation on hardware SNN architectures due to lack of appropriate application design methodologies.

The SNN application design problem can be subdivided into following main categories:

- Application specific SNN topology design
- SNN information coding to insure appropriate application response time
- Dynamic application behaviour for unaccounted input scenarios
- SNN training algorithm for a range of real-world application classes

This thesis presents modular SNN application design technique that offers a rapid, simple and effective application prototyping for large, practical and complex applications on the hardware SNN architectures.

2.4 Related Work in Hardware SNN Systems

Inspired by the computational abilities of biological nervous system, there is a significant research to implement reconfigurable and highly interconnected arrays of neural network elements in hardware to produce powerful signal processing units. This section reviews the prominent SNN architectures and their suitability as embedded hardware SNNs.

2.4.1 Hardware Spiking Neural Network Systems

A hybrid SNN computing platform including a neuron model implemented in hardware and the network model and learning implemented in software has been reported in [44]. A time multiplexed FPGA embedded processor SNN implementation reports 4.2K neurons and >1.9M synapses [47]. The neuron model and partial SNN elements are implemented on an embedded FPGA processor, where speed-acceleration is the key motivation. The system relies on external memory for spike transfer management.

FACETS, a configurable wafer-scale mixed-signal neural ASIC system, proposes a hierarchical neural network and the use of analogue floating gate memory for storage of synaptic weights [46, 48]. A mixed-signal SNN architecture of 2,400 analogue neurons, implemented using switched capacitor technology and communicating via an asynchronous event-driven bus has been reported in [45].

SpiNNaker project aims to develop a massively parallel computer capable of simulating SNNs of various sizes, topology and with programmable neuron models [41]. The SpiNNaker architecture uses ARM-968 processor-based nodes for computation and an off-chip NoC communication infrastructure. Each NoC tile in the SpiNNaker system models 1000 Leaky-Integrate-and-Fire (LIF) neurons, each having 1000 synapse inputs.

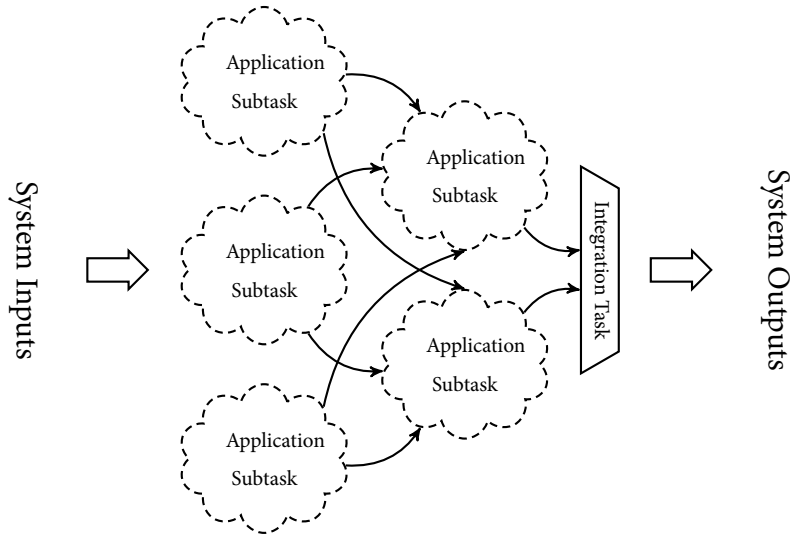


Figure 2.10: Modular Neural Network Application Organisation

A clustered embedded hardware SNN has been proposed along with a process for mapping SNNs to hardware [55]. The variable sized neuron clusters support flexible synaptic connections even from and to within the clusters.

2.4.2 Modular Neural Network Computing Paradigm

The biological brain is composed of several anatomically and functionally discrete areas [56]. Various brain areas and neuron groups are dedicated to various sensory and motor tasks. For example, the visual cortex processes different aspects of the visual information (such as form, colour and motion) in separate anatomically distinct regions. These different regions exchange information but remain functionally discrete [57, 58]. Damage or deterioration of part of the visual cortex can result in a partial loss of colour identification, pattern or motion detection, etc., without considerably affecting other senses. Inspired by this modular organisation in brain, the Modular Neural Network (MNN) design strategy of partitioning application tasks into a number of subtasks has been developed [59–61].

The MNN computing paradigm is primarily based on the *divide-and-conquer* strategy. The MNN design methodology primarily consists of *task decomposition* i.e. breaking down a high level application into smaller, less complex, manageable subtasks. Figure 2.10 illustrates a typical MNN application organisation. Figure 2.11 illustrates MNN organisation depicting individual and distinct neural modules that solve the small sized subtasks. The intermediate outputs from these neural

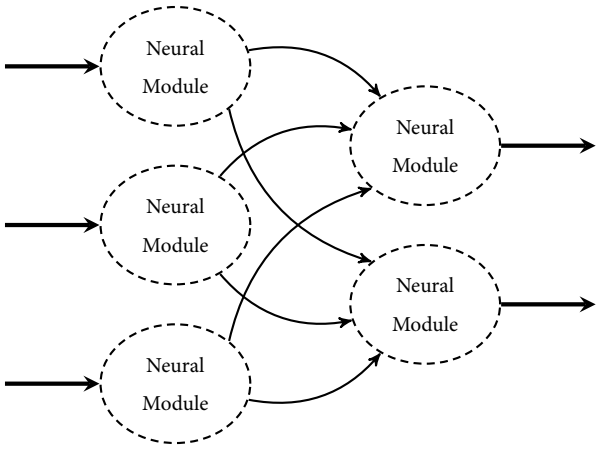


Figure 2.11: An Example Modular Neural Network Organisation

modules are combined to solve the high level task or the whole application [62, 63].

MNN Application Design

Application design for MNN systems primarily involves partitioning the application to find the correct set of application subtasks in order to achieve the overall application functionality. Various task decomposition algorithms have been proposed for the MNN design strategy. These algorithms are based on different techniques such as output vector partitioning, class relationships, neuro-evolutionary approach and co-evolutionary methods [64–68]. Subtasks or Subnetworks obtained after task decomposition are executed as neural modules in an MNN computing architecture. Similarly, genetic algorithm based technique to find subnetworks from large sized complex neural networks has been proposed in [69]. Networks for practical applications are observed to have clustered organisation and can map efficiently on MNN execution architectures [9]. MNNs have been shown to outperform monolithic neural networks in terms of training time and data classification accuracy [70].

The MNN approach offers improved training time, functional accuracy, structured implementation, functional partition, functional mapping and re-mapping, competitive and co-operative mode of operation and fault tolerance [60]. The *Subsumption Architecture*, a widely influential computing paradigm for reactive, behaviour-based autonomous robotic control applications has been developed based on the MNN design concepts [71]. This technique suggests partitioning the robotic behaviour into layers of basic and abstract functions.

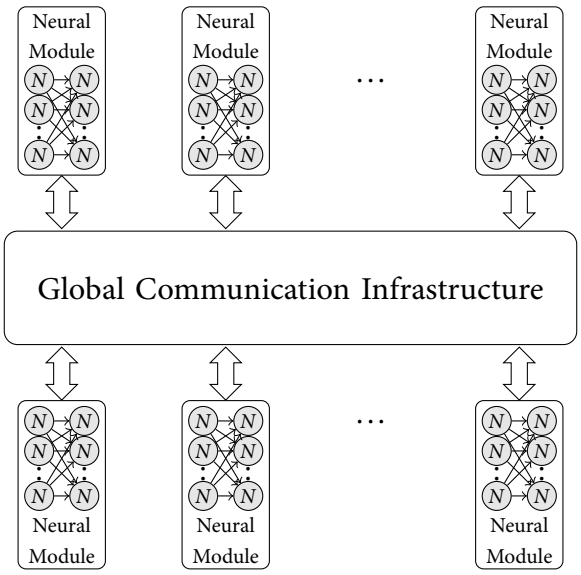


Figure 2.12: Modular Neural Network Execution Architecture

MNN System Architecture

Task decomposition of an application leads to two types of subtasks; namely the *Application Subtasks* (discrete functional subtasks) and the *Integration Subtasks* as depicted in figure 2.10. The application subtasks operate on individual and distinct system inputs to provide intermediate outputs, which are integrated in the integration subtasks to generate the overall system output. Both the application and integration subtasks are identical in terms of input/output interface and computational requirements. Figure 2.12 illustrates a typical MNN execution architecture comprising individual neural modules interconnected by a global communication infrastructure. Each individual neural module in the MNN execution architecture includes a group of neurons interconnected using an internal communication infrastructure and has multiple inputs and/or multiple outputs. The global (external inter-module) communication infrastructure provides connectivity between the individual neural modules [62, 63].

2.5 Previously Reported EMBRACE Hardware SNN Architecture and Applications

This collaborative research aims to develop EMBRACE, a compact, scalable, modular, low-power hardware SNN architecture as an embedded computing platform

ideally suited for real-world data/pattern classification, estimation, prediction, dynamic control and signal processing applications. This section summarises the previously reported research for the development of EMBRACE hardware SNN system which provides the foundation for the research challenges addressed by this thesis.

The EMBRACE architecture has been initially conceptualised and reported as a mixed-signal hardware SNN array [52]. EMBRACE architecture 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 switched Network on Chip (NoC) spike communication architecture.

2.5.1 EMBRACE Hardware SNN Architecture

Efficient implementation of hardware SNN architectures is primarily influenced by neuron design, scalable spike communication infrastructure and SNN training/learning algorithms [49]. The EMBRACE mixed-signal implementation plans to incorporate a compact, low power, high-resolution CMOS-compatible analogue neuron cell [72]. The use of digital spike voltages in both analogue and digital domains removes the necessity for Digital-to-Analogue and Analogue-to-Digital converters. This architectural scheme has the potential to offer synaptic densities significantly in excess of that currently achievable in other hardware SNNs [73]. The EMBRACE NoC communication infrastructure provides flexible, packet-switched inter-neuron communication channels, scalable interconnect and connection reconfigurability. The NoC approach addresses the interconnect resources challenge of a large scale SNN, while meeting biological-scale spike transmission timings.

The first EMBRACE NoC-based SNN architecture illustrated in figure 2.13, was proposed as a two-dimensional mesh topology array of neural tiles, where each neural tile connects in North, East, South and West directions, forming a nearest neighbour connection scheme [52]. An application specific SNN can be realised on the proposed EMBRACE architecture by programming neuron configuration parameters (SNN synaptic weights and neuron firing threshold) and SNN connection topology [74]. Spike communication within the SNN has been achieved by routing spike information packed in spike data packets over the network of neural tiles.

The EMBRACE neural tile illustrated in figure 2.14 comprises a NoC router and a neural cell. The NoC router interfaces with neighbouring NoC routers and

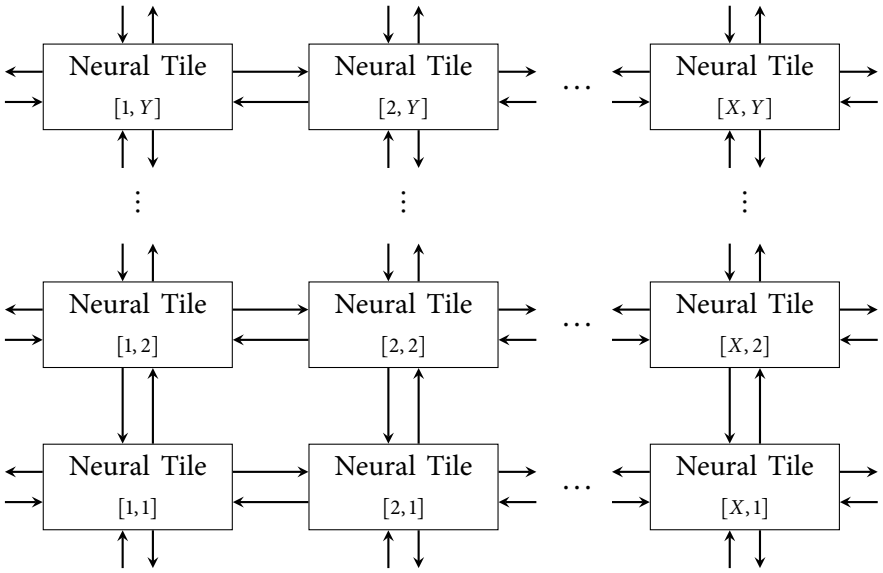


Figure 2.13: EMBRACE Hardware SNN Architecture

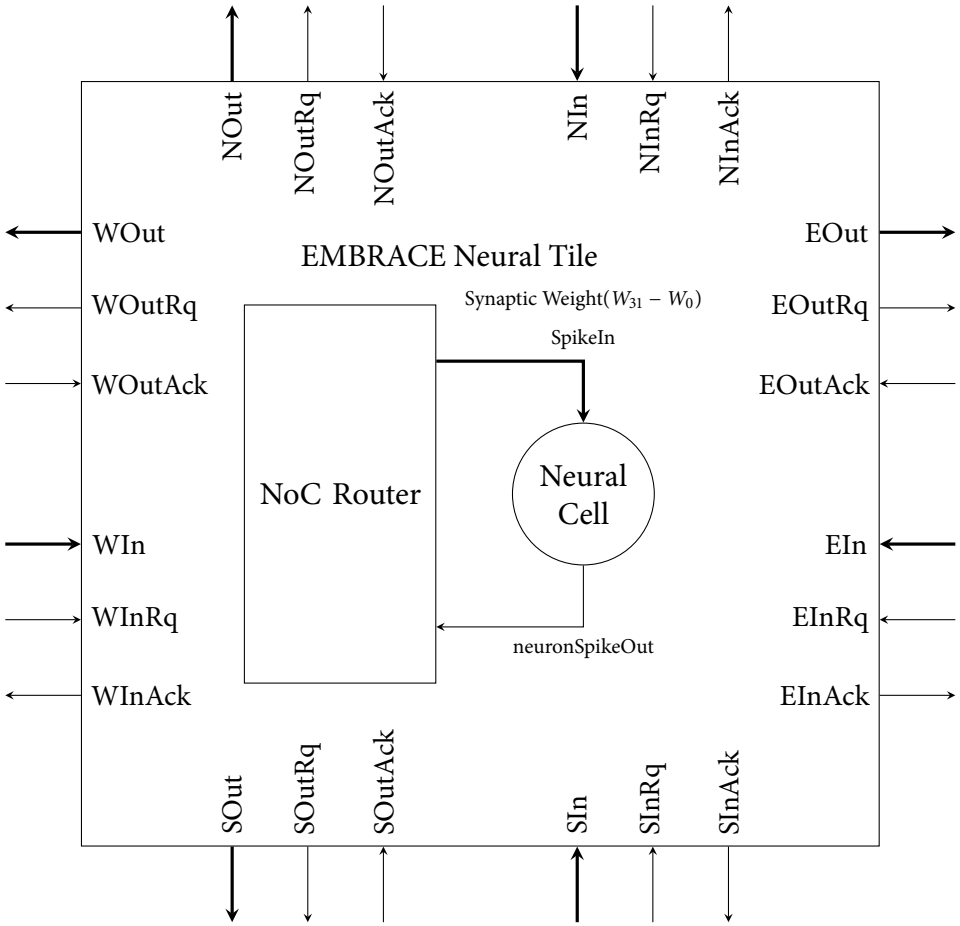


Figure 2.14: EMBRACE Neural Tile

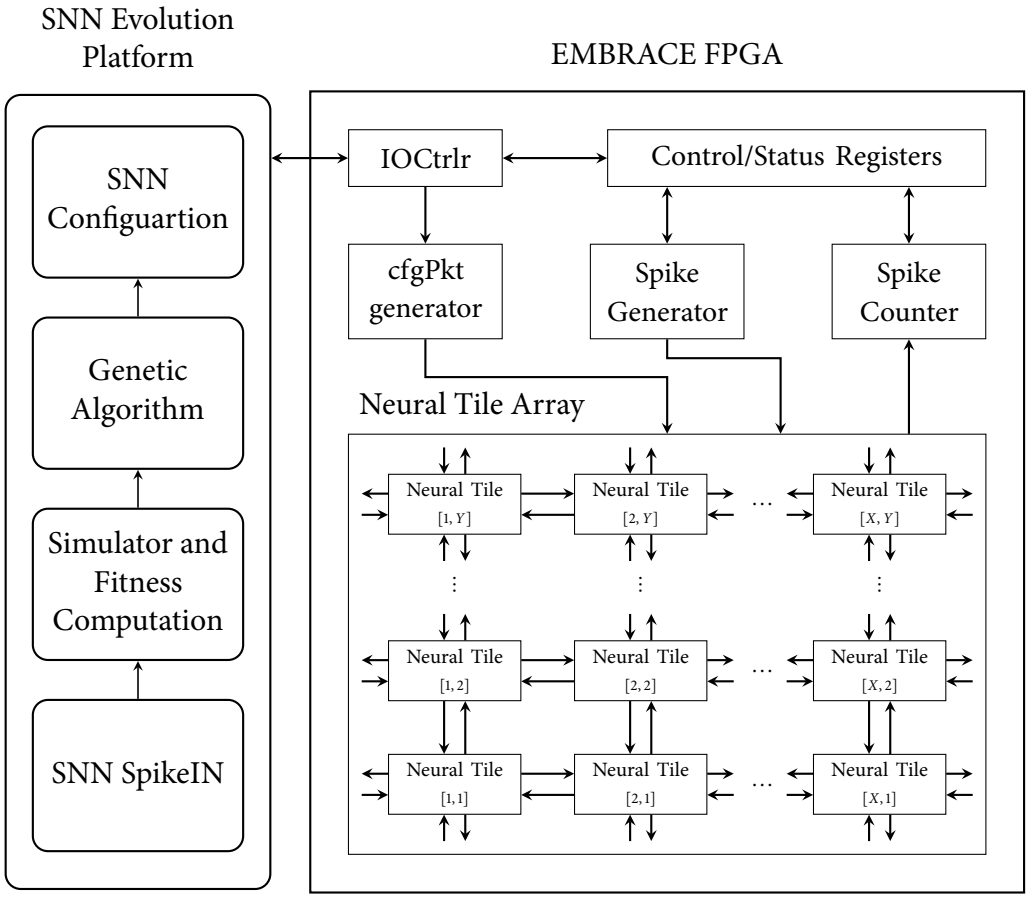


Figure 2.15: EMBRACE-FPGA Application Evolution Setup

the attached neural cell, and manges packet routing. The NoC router decodes the packets destined for the neural tile and applies spike input to the attached neural cell. The NoC router forwards the spike information generated by the attached neural cell and the packets received for other tiles. The EMBRACE FPGA implementation employs a neural cell realised as microcode, executing on PicoBlaze 8-bit RISC microcontroller. The neural cell models LIF neuron behaviour and contains 32 synapses. The membrane potential resolution is 16-bits and each synaptic weight is a signed 5-bit number (to support inhibitory and excitatory synaptic weight). The neural tile also stores SNN configuration parameters and topology information.

2.5.2 EMBRACE SNN Application Prototyping

The proposed initial EMBRACE architecture had been successfully validated on FPGA for faster application prototyping. Figure 2.15 illustrates SNN application

evolution setup on the EMBRACE-FPGA prototype implementation. The GA based intrinsic evolution setup is a supervised SNN training method that maintains a population of SNN configurations and uses nature inspired evolutionary techniques (comprising selection, crossover and mutation) to find a correct set of SNN configuration. The SNN evolution platform (running on a host computer) initialises a random SNN configuration (comprising synaptic weights and firing thresholds) and configures spike generation rates on the EMBRACE-FPGA hardware SNN. The SNN evolution platform reads back the SNN output value from the spike counter, evaluates the accuracy of the SNN configuration and evolves the SNN configuration. This process is repeated until the desired SNN application behaviour is achieved.

A number of benchmark SNN applications such as XOR data classifier and inverted pendulum controller have been successfully evolved on the EMBRACE-FPGA prototype [74]. Also, real-world embedded applications including Wisconsin breast cancer dataset classifier and robotic controller have been successfully implemented on the first EMBRACE-FPGA prototype [74–76].

2.6 Summary

This chapter summarised the SNN computing paradigm and related research work. The biologically inspired SNN computing paradigm is presented with emphasis on spiking neuron models, neuronal information coding, neural network topologies and SNN training algorithms.

Key SNN system design challenges have been presented. This thesis contributes to a number of hardware SNN system design challenges such as hardware SNN simulations and performance measurement, compact hardware implementation, architectural scalability, application reliability and efficient SNN application design.

A review of current research in hardware SNN architectures and MNN computing paradigm has been presented. This research employs MNN design principles to design compact, scalable hardware SNN architecture and efficient modular SNN applications.

This chapter presents the initial EMBRACE architecture design and application prototyping used as a foundation for this research. The following chapters present the research phases and contributions for designing compact, scalable and reliable hardware SNN architecture and efficient and rapid application prototyping.

References

[1] M. Arbib, *The handbook of brain theory and neural networks*. Bradford Book, 2003.

[2] A. Hodgkin and A. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *Bulletin of Mathematical Biology*, vol. 52, no. 1-2, pp. 25–71, 1990.

[3] W. Gerstner and W. Kistler, *Spiking neuron models*. Cambridge University Press, 2002.

[4] E. Izhikevich, "Simple model of spiking neurons," *Neural Networks, IEEE Transactions on*, vol. 14, no. 6, pp. 1569–1572, Nov.

[5] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[6] W. Gerstner, A. Kreiter, H. Markram, and A. Herz, "Neural codes: Firing rates and beyond," *Proceedings of the National Academy of Sciences*, vol. 94, no. 24, pp. 12 740–12 741, 1997.

[7] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *Nature*, vol. 381, pp. 520 – 522, 1996.

[8] F. Rieke, D. Warland, R. Deruytervansteveninck, and W. Bialek, *Spikes: Exploring the Neural Code (Computational Neuroscience)*. The MIT Press, Jun. 1999.

[9] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, jun 1998.

[10] L. Perrinet, M. Samuelides, and S. Thorpe, "Sparse spike coding in an asynchronous feed-forward multi-layer neural network using matching pursuit," *Neurocomputing*, vol. 57, pp. 125–134, 2004.

[11] M. Escobar, G. Masson, T. Vieville, and P. Kornprobst, "Action recognition using a bio-inspired feedforward spiking network," *International Journal of Computer Vision*, vol. 82, no. 3, pp. 284–301, 2009.

[12] W. Gerstner and J. Van Hemmen, "Associative memory in a network of spiking neurons," *Network*, vol. 3, no. 2, pp. 139–164, 1992.

[13] F. Sommer and T. Wennekers, "Associative memory in networks of spiking neurons," *Neural Networks*, vol. 14, no. 6-7, pp. 825–834, 2001.

[14] A. Robinson, "An application of recurrent nets to phone probability estimation," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 298–305, 1994.

[15] W. Maass, "Paradigms for computing with spiking neurons," *Models of Neural Networks IV*, pp. 373–402, 2002.

[16] D. Hebb, "The organization of behavior; a neuropsychological theory." 1949.

[17] G. Hinton and T. Sejnowski, *Unsupervised learning: foundations of neural computation*. The MIT press, 1999.

[18] T. Natschlaeger and B. Ruf, "Online clustering with spiking neurons using temporal coding," *Progress in Neural Processing*, pp. 33–42, 1998.

[19] S. Bohte, H. La Poutre, and J. Kok, "Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer rbf networks," *Neural Networks, IEEE Transactions on*, vol. 13, no. 2, pp. 426–435, 2002.

[20] F. Landis, T. Ott, and R. Stoop, "Hebbian self-organizing integrate-and-fire networks for data clustering," *Neural computation*, vol. 22, no. 1, pp. 273–288, 2010.

[21] J. Hopfield, "Pattern recognition computation using action potential timing for stimulus representation," *Nature*, vol. 376, no. 6535, pp. 33–36, 1995.

[22] M. Zamani, A. Sadeghian, and S. Chartier, "A bidirectional associative memory based on cortical spiking neurons using temporal coding," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*, 2010, pp. 1–8.

[23] E. Alpaydin, *Introduction to machine learning*. The MIT Press, 2010.

[24] S. Bohte, J. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

[25] R. Legenstein, C. Naeger, and W. Maass, "What can a neuron learn with spike-timing-dependent plasticity?" *Neural Computation*, vol. 17, no. 11, pp. 2337–2382, 2005.

[26] R. Legenstein, D. Pecevski, and W. Maass, "A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback," *PLoS Computational Biology*, vol. 4, no. 10, p. e1000180, 2008.

[27] D. Baras and R. Meir, "Reinforcement learning, spike-time-dependent plasticity, and the bcm rule," *Neural Computation*, vol. 19, no. 8, pp. 2245–2279, 2007.

[28] M. Farries and A. Fairhall, "Reinforcement learning with modulated spike timing–dependent synaptic plasticity," *Journal of neurophysiology*, vol. 98, no. 6, pp. 3648–3665, 2007.

[29] R. Florian, "A reinforcement learning algorithm for spiking neural networks," in *Symbolic and Numeric Algorithms for Scientific Computing, 2005. SYNASC 2005. Seventh International Symposium on*, 2005.

[30] R. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural Computation*, vol. 19, no. 6, pp. 1468–1502, 2007.

[31] E. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral Cortex*, vol. 17, no. 10, pp. 2443–2452, 2007.

[32] E. Vasilaki, N. Frémaux, R. Urbanczik, W. Senn, and W. Gerstner, "Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail," *PLoS computational biology*, vol. 5, no. 12, p. e1000586, 2009.

[33] E. Di Paolo, "Spike-timing dependent plasticity for evolved robots," *Adaptive Behavior*, vol. 10, no. 3-4, pp. 243–263, 2002.

[34] H. Hagras, A. Pounds-Cornish, M. Colley, V. Callaghan, and G. Clarke, "Evolving spiking neural network controllers for autonomous robots," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 5, 2004, pp. 4620–4626.

[35] M. Melanie, "An introduction to genetic algorithms," *Cambridge, Massachusetts London, England, Fifth printing*, vol. 3, 1999.

[36] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 12, pp. 1523–1543, 2000.

[37] A. Jantsch, "Nocsim: A NoC simulator," 2006.

[38] Z. Lu, R. Thid, M. Millberg, E. Nilsson, and A. Jantsch, "NNSE: Nostrum network-on-chip simulation environment," *Proc. of SSoCC*, 2005.

[39] M. Lis, K. Shim, M. Cho, P. Ren, O. Khan, and S. Devadas, "DARSIM: a parallel cycle-level noc simulator," in *MoBS 2010-Sixth Annual Workshop on Modeling, Benchmarking and Simulation*, 2010.

[40] S. Pande, F. Morgan, S. Cawley, B. McGinley, S. Carrillo, J. Harkin, and L. McDaid, "EMBRACE-SysC for analysis of NoC-based spiking neural network architectures," in *System on Chip (SoC), 2010 International Symposium on*, Sept. 2010, pp. 139–145.

[41] S. Furber and A. Brown, "Biologically-Inspired Massively-Parallel architectures - computing beyond a million processors," in *Application of Concurrency to System Design, 2009. ACSD '09. Ninth International Conference on*, 2009, pp. 3–12.

[42] A. Upegui, C. Peña-Reyes, and E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks," *Microprocessors and Microsystems*, vol. 29, no. 5, pp. 211–223, Jun. 2005.

[43] M. Pearson, A. Pipe, B. Mitchinson, K. Gurney, C. Melhuish, I. Gilhespy, and M. Nibouche, "Implementing spiking neural networks for Real-Time Signal-Processing and control applications: A Model-Validated FPGA approach," *Neural Networks, IEEE Transactions on*, vol. 18, no. 5, pp. 1472–1487, 2007.

[44] E. Ros, E. Ortigosa, R. Agis, R. Carrillo, and M. Arnold, "Real-time computing platform for spiking neurons (RT-spike)," *Neural Networks, IEEE Transactions on*, vol. 17, no. 4, pp. 1050–1063, 2006.

[45] R. Vogelstein, U. Mallik, J. Vogelstein, and G. Cauwenberghs, "Dynamically reconfigurable silicon array of spiking neurons with Conductance-Based synapses," *Neural Networks, IEEE Transactions on*, vol. 18, no. 1, pp. 253–265, 2007.

[46] M. Ehrlich, C. Mayr, H. Eisenreich, S. Henker, A. Srowig, A. Grubl, J. Schemmel, and R. Schuffny, "Wafer-scale VLSI implementations of pulse coupled neural networks," in *Proceedings of the International Conference on Sensors, Circuits and Instrumentation Systems*, 2007.

[47] B. Glackin, T. McGinnity, L. Maguire, Q. Wu, and A. Belatreche, "A novel approach for the implementation of large scale spiking neural networks on FPGA hardware," in *Computational Intelligence and Bioinspired Systems*, 2005, pp. 552–563.

[48] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, june 2008, pp. 431–438.

[49] L. Maguire, T. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for large-scale implementations of spiking neural networks on FPGAs," *Neurocomputing*, vol. 71, no. 1-3, pp. 13–29, Dec. 2007.

[50] D. Vainbrand and R. Ginosar, "Scalable network-on-chip architecture for configurable neural networks," *Microprocessors and Microsystems*, vol. 35, no. 2, pp. 152–166, 2011, special issue on Network-on-Chip Architectures and Design Methodologies.

[51] T. Theocharides, G. Link, N. Vijaykrishnan, M. Invin, and V. Srikantam, "A generic reconfigurable neural network architecture as a network on chip," in *SOC Conference, 2004. Proceedings. IEEE International*, sept. 2004, pp. 191 – 194.

[52] J. Harkin, F. Morgan, L. McDaid, S. Hall, B. McGinley, and S. Cawley, "A reconfigurable and biologically inspired paradigm for computation using network-on-chip and spiking neural networks," *Int. J. Reconfig. Comput.*, vol. 2009, pp. 1–13, 2009.

[53] S. Pande, F. Morgan, S. Cawley, B. McGinley, J. Harkin, S. Carrillo, and L. McDaid, "Addressing the hardware resource requirements of Network-on-Chip based neural architectures," in *NCTA, International Conference on Neural Computation Theory and Applications, Paris, France*, Oct. 2011.

[54] S. Pande, F. Morgan, G. Smit, T. Bruintjes, J. Rutgers, B. McGinley, S. Cawley, J. Harkin, and L. McDaid, "Fixed latency on-chip interconnect for hardware spiking neural network architectures," *Parallel Computing*, vol. 39, pp. 357 – 371, Sep. 2013.

[55] R. Emery, A. Yakovlev, and G. Chester, "Connection-centric network for spiking neural networks," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, 2009, pp. 144–152.

[56] S. Johannes, B. Wieringa, M. Matzke, and T. Münte, "Hierarchical visual stimuli: electrophysiological evidence for separate left hemispheric global and local processing mechanisms in humans," *Neuroscience letters*, vol. 210, no. 2, pp. 111–114, May 1996.

[57] D. Van Essen, C. Anderson, and D. Felleman, "Information processing in the primate visual system: an integrated systems perspective," *Science*, vol. 255, no. 5043, pp. 419–423, Jan. 1992.

[58] T. Binzegger, R. Douglas, and K. Martin, "Stereotypical bouton clustering of individual neurons in cat primary visual cortex," *The Journal of Neuroscience*, vol. 27, no. 45, pp. 12 242–12 254, Nov. 2007.

[59] B. Happel and J. Murre, "Design and evolution of modular neural network architectures," *Neural Networks*, vol. 7, no. 6-7, pp. 985–1004, 1994.

[60] G. Auda and M. Kamel, "Modular neural networks a survey," *International Journal of Neural Systems*, vol. 9, no. 2, pp. 129–151, 1999.

[61] Ronco and P. Gawthrop, "Modular neural networks: a state of the art," *Rapport Technique CSC95026 Center of System and Control University of Glasgow*, vol. 1, pp. 1–22, 1995.

[62] D. Osherson, S. Weinstein, and M. Stob, "Modular learning." Cambridge, MA, USA: MIT Press, 1993, pp. 369–377.

[63] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999, vol. 13.

[64] S. Guan, S. Li, and S. Tan, "Neural network task decomposition based on output partitioning," *Journal of the Institution of Engineers Singapore*, vol. 44, pp. 78–89, 2004.

[65] B. Lu and M. Ito, "Task decomposition and module combination based on class relations: a modular neural network for pattern classification," *Neural Networks, IEEE Transactions on*, vol. 10, no. 5, pp. 1244–1256, sep 1999.

[66] J. Thangavelautham and G. D.eleuterio, "A neuroevolutionary approach to emergent task decomposition," in *Proc. of 8th Parallel Problem Solving from Nature*. Springer, 2004, pp. 991–1000.

[67] V. Khare, Y. Xin, B. Sendhoff, Y. J., and H. Wersing, "Co-evolutionary modular neural networks for automatic problem decomposition," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3, sept 2005, pp. 2691–2698.

[68] J. Santos, L. Alexandre, and J. de Sa, "Modular neural network task decomposition via entropic clustering," in *Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on*, vol. 1, oct. 2006, pp. 62–67.

[69] C. Pizzuti, "A multiobjective genetic algorithm to find communities in complex networks," *Evolutionary Computation, IEEE Transactions on*, vol. 16, no. 3, pp. 418–430, june 2012.

[70] B. Gour, T. Bandopadhyaya, and R. Patel, "Art and modular neural network architecture for multilevel categorization and recognition of fingerprints," in *Knowledge Discovery and Data Mining, 2010. WKDD '10. Third International Conference on*, jan. 2010, pp. 536–539.

[71] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, mar 1986.

[72] Y. Chen, S. Hall, L. McDaid, O. Buiu, and P. Kelly, "A solid state neuron for the realisation of highly scaleable third generation neural networks," in *Solid-State and Integrated Circuit Technology, 2006. ICSICT '06. 8th International Conference on*, 2006, pp. 1071–1073.

[73] B. Roche, T. McGinnity, L. Maguire, and L. McDaid, "Signalling techniques and their effect on neural network implementation sizes," *Information Sciences*, vol. 132, no. 1, pp. 67–82, 2001.

[74] 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*, vol. 12, pp. 257–280, Apr. 2011.

[75] F. Morgan, S. Cawley, B. McGinley, S. Pande, L. McDaid, B. Glackin, J. Maher, and J. Harkin, "Exploring the evolution of NoC-based spiking neural networks

on FPGAs," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, 2009, pp. 300–303.

[76] F. Morgan, S. Cawley, J. Harkin, B. McGinley, L. McDaid, and S. Pande, "An evolvable noc-based spiking neural network architecture," in *Signals and Systems Conference (ISSC 2009), IET Irish*. IET, 2009, pp. 1–6.

CHAPTER 3

EMBRACE-SysC for Analysis of NoC-based Spiking Neural Network Architectures

This work has been published and presented at International Symposium on System on Chip (SoC), Tampere, Finland, Sept. 2010. DOI: 10.1109/ISSOC.2010.5625566

Preamble

This chapter details the EMBRACE-SysC design space exploration framework developed for the systematic design of EMBRACE hardware SNN architecture.

System design for embedded hardware SNN architecture offers a number of architectural choices which have significant impact on the area, power and performance of the hardware system. EMBRACE-SysC provides high level simulation and performance measurement capabilities suitable for making architectural design choices early in the system design phase.

This chapter presents structured modelling of the EMBRACE hardware architecture in SystemC, including the detailed design of the architectural entities such as neural tile, spike generators/counters and NoC router. The chapter presents EMBRACE-SysC components illustrating the various model, performance measurement, simulation support and genetic algorithm libraries. The chapter also presents the EMBRACE-SysC simulation flow illustrating generated performance reporting, output and debug logs.

The performance measurement capabilities of EMBRACE-SysC framework are presented. The measurement of spike injection rate for error-free spike communication through the packet switched NoC architecture is presented. An analysis of the NoC router micro-architecture and its impact on the spike transfer capabilities of NoC router is provided. The chapter presents experiment depicting NoC hotspot detection due to NoC router bandwidth exhaustion. A sample SNN topology, the mapping of the SNN to the EMBRACE architecture and NoC router bandwidth utilisation results are provided, and an analysis of the NoC traffic is presented.

EMBRACE-SysC comprises GA-based SNN evolution modules that can be seamlessly integrated within the EMBRACE architecture simulations. This integration allows evolution of the SNN application, easy validation of neuron models, and confirms suitability of the overall architecture as an SNN computing platform. The chapter presents NoC traffic measurements for various application states through an evolved XOR SNN benchmark application.

This research addresses the large scale accurate simulation challenge for the design of hardware SNN systems. The development of EMBRACE-SysC introduces a powerful design exploration framework for EMBRACE architecture design. Future phases of this research extensively employs modeling and performance measurement capabilities of EMBRACE-SysC to make architectural choices based on performance analysis.

EMBRACE-SysC for Analysis of NoC-based Spiking Neural Network Architectures

Sandeep Pande[a], Fearghal Morgan[a], Seamus Cawley[a], Brian Mc Ginley[a], Snaider Carrillo[b], Jim Harkin[b], Liam Mc Daid[b]

[a] Bio-Inspired Electronics and Reconfigurable Computing Research Group (BIRC), National University of Ireland, Galway, Ireland.

[b] Intelligent Systems Research Centre, University of Ulster, Magee Campus, Derry, Northern Ireland.

Abstract: *This paper presents EMBRACE-SysC, a simulation-based design exploration framework for the EMBRACE mixed signal Network on Chip (NoC)-based hardware Spiking Neural Network (SNN) architecture. EMBRACE-SysC incorporates Genetic Algorithm-based training of SNN applications. Results illustrate the application of EMBRACE-SysC for performance analysis of a NoC-based SNN architecture. The development of EMBRACE-SysC introduces a powerful design exploration framework for EMBRACE architecture development.*

3.1 Introduction

Nature-inspired computing paradigms such as neural networks and evolutionary computation provide promising solutions for designing complex and intelligent systems [1]. The organic central nervous system includes a dense and complex interconnection of neurons and synapses. Computing systems based on Spiking Neural Networks (SNNs) emulate real biological neural networks, conveying information through the communication of short transient pulses between neurons via their synaptic connections. Each neuron maintains a *membrane potential*, which is a function of incoming spikes, synaptic weights, leakage coefficient, and current membrane potential [2, 3]. A neuron fires when the sum of its weighted input spikes exceeds a firing threshold value. Brain-inspired computing paradigms

such as SNNs offer the potential for elegant, low-power and scalable methods of computing, with rich non-linear dynamics ideally suited to applications including data/pattern classification, dynamic control and signal processing [2].

Traditional System on Chip (SoC) designs commonly use a shared-bus topology, which offers simple and inexpensive interconnection between various on-chip elements. However, this approach cannot scale to support large hardware SNN implementations and cannot guarantee real-time SNN behaviour since the response time of the SNN increases with the number of neurons and synapses connected to the shared-bus [4]. Key to the development of low-power, high-performance, reconfigurable hardware SNN-based embedded systems is the efficient and scalable implementation of synaptic junctions and neuron interconnect [5]. A Network on Chip (NoC) approach can provide a scalable and efficient on-chip communication infrastructure for large SoC designs, and also for hardware SNNs [6].

The authors have proposed and investigated EMBRACE [7], a mixed-signal, low power, scalable, NoC-based hardware architecture as an embedded computing element that supports implementation of large scale SNNs [4, 8, 9]. The EMBRACE architecture incorporates a compact, low power, high-resolution CMOS-compatible analogue neuron cell. The use of digital spike voltages in both analogue and digital domains takes away the necessity for Digital-to-Analogue and Analogue-to-Digital converters. This architectural scheme offers synaptic densities significantly in excess of that currently achievable in other hardware SNNs [4]. The EMBRACE NoC communication infrastructure provides flexible, packet-switched inter-neuron communication channels, scalable interconnect and connection reconfigurability. The NoC approach addresses the interconnect resources challenge of a large scale SNN, while meeting biological-scale spike transmission timings (of the order of 10 ms) [3].

The EMBRACE NoC-based SNN architecture (figure 3.1) is a two-dimensional mesh topology array comprising of Routers (R), Neurons (N), Spike Generators (SGs), and Spike Counters (SCs). Each router is connected in North (N), East (E), South (S) and West (W) directions, forming a nearest neighbour connection scheme. An application specific SNN is realised on the EMBRACE architecture by programming neuron configuration parameters (SNN synaptic weights and neuron firing potential) and SNN connection topology. Spike communication within the SNN is achieved by routing spike information within spike data packets through N, S, E and W router ports. A GA-based search method is used to evolve a correct set of synaptic weights and neuron threshold potentials for the specific

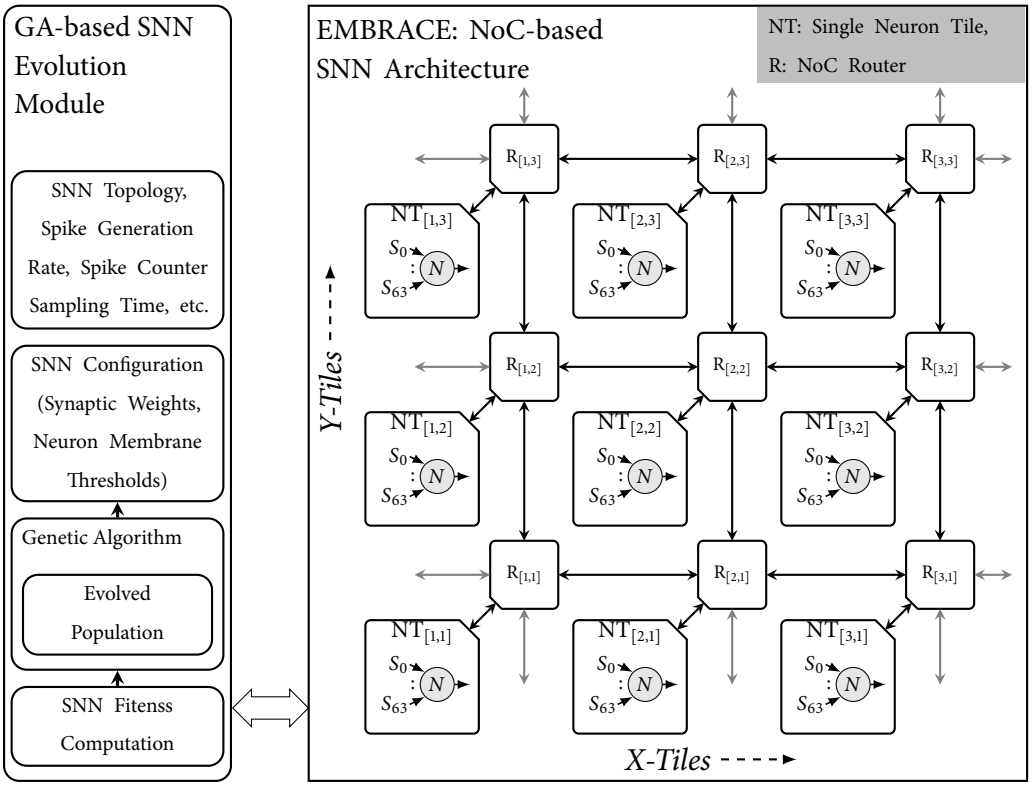


Figure 3.1: EMBRACE NoC-based SNN Architecture illustrating two-dimensional mesh topology array of Routers (R), Neurons (N), Spike Generators (SGs), and Spike Counters (SCs) and the GA-based SNN Evolution Module

SNN application.

The authors have implemented and reported EMBRACE-FPGA [7, 10–12], a hardware NoC based SNN prototype, applied to control and classification applications.

This paper presents EMBRACE-SysC, a SystemC based [13], clock cycle accurate simulation and performance measurement platform for simulation and analysis of EMBRACE architectures. EMBRACE-SysC enables analysis of NoC router architectures, packet flow control and performance bottlenecks, and SNN topologies and is a central component in the EMBRACE and EMBRACE-FPGA hardware development process. The paper describes EMBRACE-SysC and presents NoC simulation results for various spike data traffic conditions. EMBRACE-SysC incorporates a Genetic Algorithm (GA)-based SNN evolution module.

Information exchange in SNNs is in the form of continuous spike streams, where information is coded in the relative timing of spikes. Router design and time multiplexing of router interconnections results in unwanted variations in

packet latency. The resulting packet latency jitter distorts spike timing information in SNN applications and can affect the accuracy of the SNN output. While the presence of packet latency jitter increases SNN evolution time, it can lead to robust SNN individuals capable of adapting to changing environments [14]. A goal of this research is to design the EMBRACE architecture with minimal packet latency jitter. The paper presents and discusses EMBRACE-SysC simulation of spike packet latency for an evolved XOR benchmark SNN application.

The structure of the paper is as follows: Related work on the simulation and performance analysis of NoC and SNN architectures is reviewed in Section 3.2. The EMBRACE-SysC platform is described in Section 3.3. Section 3.4 presents the results and analysis of EMBRACE NoC and XOR benchmark SNN application for various spike data traffic conditions. Section 3.5 concludes the paper and proposes future work.

3.2 Related Work

This section reviews related work on the simulation and performance analysis of NoC and SNN architectures, and proposes the requirements for the EMBRACE-SysC platform.

Several simulation-based NoC design exploration approaches have been reported, including Nostrum [15], Nirgam [16], Noxim [17], CHAIN/Silistix [18] and Orion [19]. These simulators mainly focus on embedded multimedia and multiprocessor applications and represent digital NoC architectures. The EMBRACE mixed signal architecture uses analogue components for neural computations and digital NoC for transfer of spike information. Existing NoC simulators allow the use of traffic patterns such as random, bursty, permutation and Poisson distribution. For correct representation of spike data traffic, accurately timed model of the analogue neuron cell is required. A SystemC-based simulation platform has been developed for the SpiNNaker [20, 21] NoC-based multi-processor SNN. This simulator is used for SNN functional validation, software development and debugging.

A simulator for analysis of a System-on-Chip, NoC based hardware SNN has not been reported. SystemC offers rich features and capabilities needed for modelling and simulation of mixed signal NoC-based SNN architecture. EMBRACE-SysC has been developed using SystemC and is a central component in the EMBRACE SoC NoC-based hardware SNN development process. EMBRACE-SysC enables:

- Accurate modelling of spike data traffic through clock cycle accurate modelling of all EMBRACE digital components and timed modelling of the analogue CMOS neuron cell
- Systematic investigation of performance bottlenecks (e.g. traffic hotspots) in the architecture
- Performance comparison and analysis of architectural design choices
- Faster architecture design, prototyping and simulation (compared to RTL simulation)
- Verification of specifications for EMBRACE design enhancements, down to clock cycle accuracy
- Studying of SNN application feasibility prior to hardware prototyping

3.3 EMBRACE-SysC: NoC-based SNN Simulation and Performance Measurement Platform

This section describes the EMBRACE-SysC simulation platform, the modelling of the architectural components, and the simulation flow. Figure 3.2 illustrates the EMBRACE-SysC modelling and simulation flow, which includes simulation platform creation, configuration, and performance data collection.

EMBRACE-SysC models each of the EMBRACE architecture elements in a modular fashion at a Timed Functional (TF) abstraction level using SystemC [13]. EMBRACE-SysC offers a fast and clock cycle accurate executable specification of the EMBRACE architecture. EMBRACE-SysC can simulate various SNN topologies, SNN application configurations and EMBRACE architectural configurations. The configuration information like NoC dimension, SNN topology, NoC location of various Neurons, SGs and SCs, etc is supplied to the simulator through a platform configuration file. EMBRACE functionality and interfaces are modelled with clock cycle accuracy and mimic the RTL functionality and pin interface behaviour (using standard Transaction Level Modelling (TLM) interface methods). During simulation, all EMBRACE-SysC modules send important system activity information to the performance measurement modules, which collate this information and store it in the performance database.

Spike Generator (SG) modules generate spikes based on the configured spike generation rate. Spike Counter (SC) modules measure the spike rate at SNN outputs. The Neuron (N) module models Leaky-Integrate-and-Fire behaviour [12].

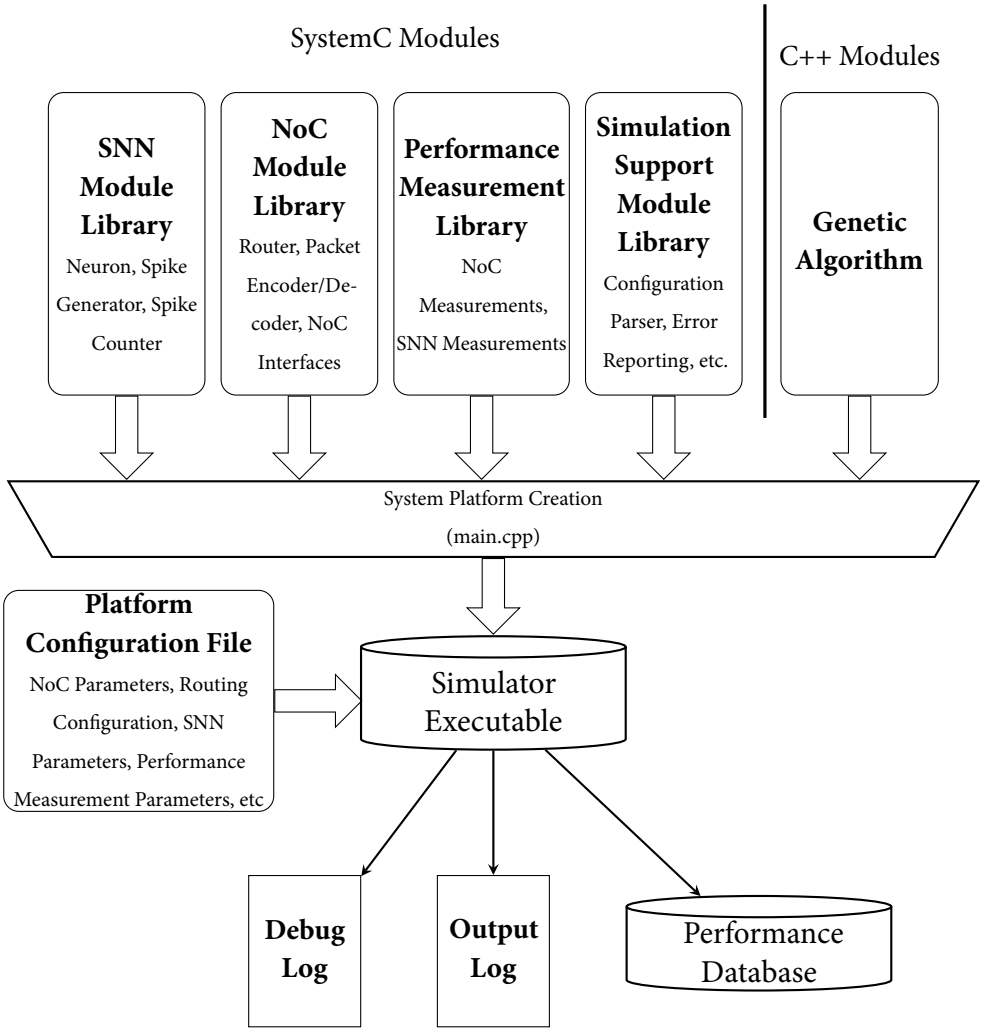


Figure 3.2: EMBRACE-SysC Modelling and Simulation Flow

A configuration interface provides read/write access to SG spike generation rates, SC measured spike rates, and neuron synaptic weights and threshold potential.

Figure 3.3 illustrates the internal architecture of the neuron, NoC router, their connectivity and configuration interface. Figure 3.4 illustrates the internal architecture of the spike generator, spike counter, NoC router connection ports, and configuration interface.

The NoC module library contains architectural components including router, packet decoder and encoder, and NoC communication interfaces. A NoC router communicates with Neuron, SG and SC via the Router Interface (RI). The NoC router polls all input ports for incoming spike packets (using a round robin arbitration scheme), acknowledges receipt of an incoming spike packet, and buffers the packet. The NoC router decodes an incoming packet's destination address and

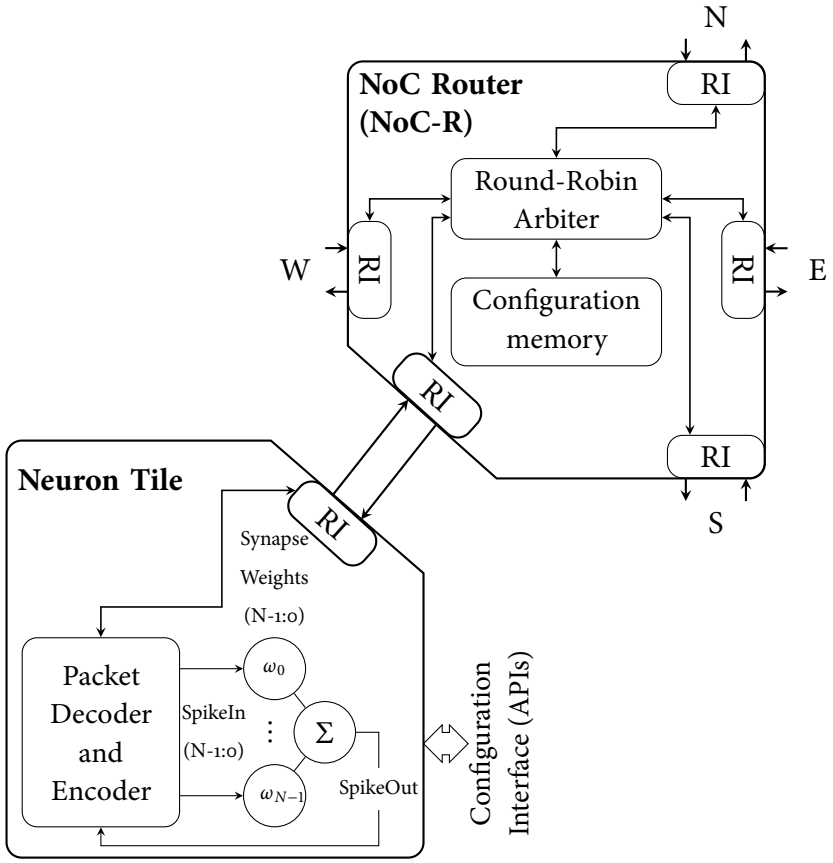


Figure 3.3: Neuron and NoC Router Internal Architecture, Connectivity and Configuration Interface

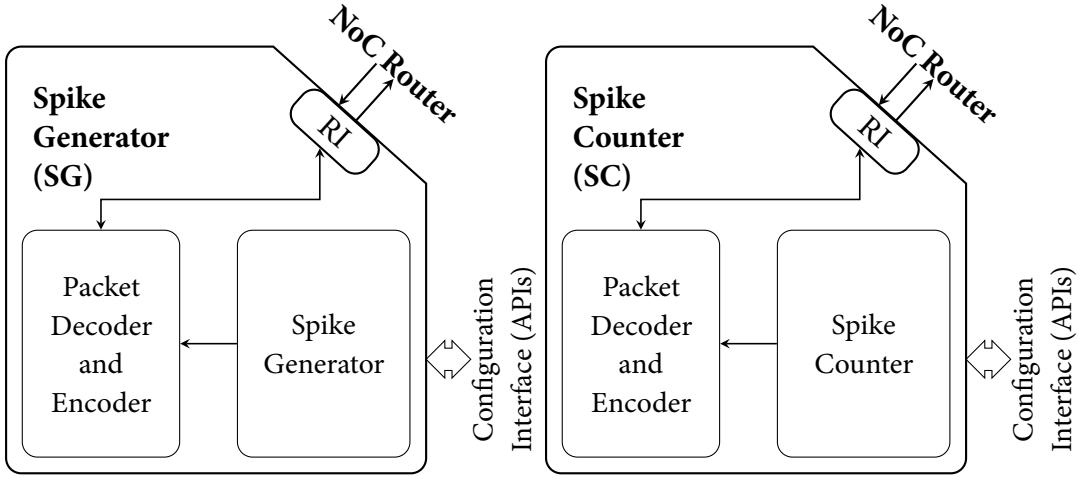


Figure 3.4: Spike Generator and Spike Counter Internal Architecture, NoC Router Connection Ports and Configuration Interface

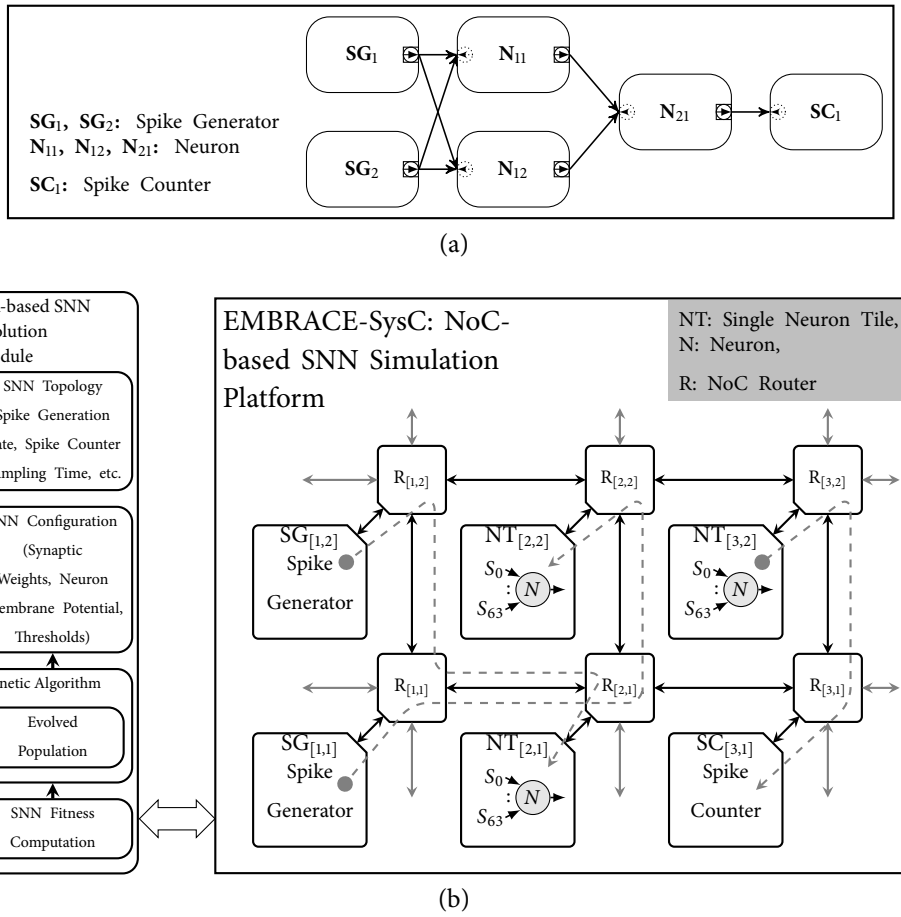


Figure 3.5: (a)XOR benchmark SNN Connection Topology, (b) Mapping of XOR SNN Modules (Spike Generator, Neuron and Spike Counter) to the EMBRACE-SysC Architecture in a 3×2 Mesh Configuration and GA-based SNN Evolution Module

forwards the packet to the connected Neuron, SG or SC, or to a neighbouring router based on routing table stored in router configuration memory.

Different SNN topologies are created by configuring traffic connections between SNN elements, to define the synaptic connections between Neurons, SGs and SCs. Spike communication within the SNN is achieved by routing packet-based spike information through the network of NoC routers [22].

A three neuron, fully connected SNN structure illustrated in figure 3.5a is used to evolve the XOR benchmark SNN application [11]. Figure 3.5b illustrates the mapping of the XOR SNN modules and synaptic connections to the EMBRACE-SysC platform (incorporating a 3×2 mesh configuration).

The GA-based SNN evolution module finds a correct configuration for the SNN application. This module configures SG spike generation rates. SGs gener-

ate spikes which are connected to SNN input layer neurons. SNN output layer neurons are connected to SCs, which count the number of spikes received within a fixed time window, and convert the SNN output spike rate to integer output values. The SNN output value is read by the SNN evolution module through the configuration interface. An SNN individual configuration is described by neuron synaptic weights and firing thresholds. The GA-based SNN evolution module evolves an SNN population (a set of SNN individuals) through a process of selection, crossover and mutation [23]. For every SNN generation, the fitness (or correctness) of each SNN individual configuration is evaluated. Fitness evaluation is followed by the generation of a new SNN population. This process is repeated for a predefined number of generations, or until a satisfactory SNN fitness has been achieved.

3.4 EMBRACE Architecture Analysis Results

This section presents the results of EMBRACE-SysC simulations. The EMBRACE NoC is simulated under various spike data traffic conditions to determine the maximum spike packet injection rate for NoC paths and to identify NoC router traffic hotspots. This section also presents the results of EMBRACE-SysC analysis of NoC latency for an evolved XOR benchmark SNN application and analyses the suitability of the EMBRACE architecture for practical and biologically plausible SNN applications.

3.4.1 NoC Spike Packet Injection Rate

This experiment illustrates the use of EMBRACE-SysC to determine the maximum supported spike packet injection rate of a number of EMBRACE NoC configurations. The EMBRACE-SysC model is configured as a 2, 3, and 4 router NoC path. A 2-router NoC path is illustrated in figure 3.6 Spike packets propagate from the SG through two consecutive routers to the SC. SNN neuron functionality is not active in this example.

As the spike generation rate is varied, EMBRACE-SysC monitors spike packet traffic and level of spike packet loss. Figure 3.7 illustrates the worst case relationship between the spike packet injection rate and spike packet loss for the 2, 3 and 4 router NoC paths. Spike packet loss is mainly governed by the NoC router design and packet flow control. Each NoC router polls and services its ports using an eight

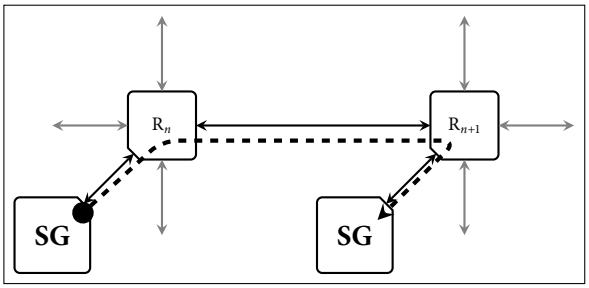


Figure 3.6: EMBRACE-SysC 2-Router NoC Path Configuration used to determine the Maximum Spike Injection Rate for a selected NoC Configuration

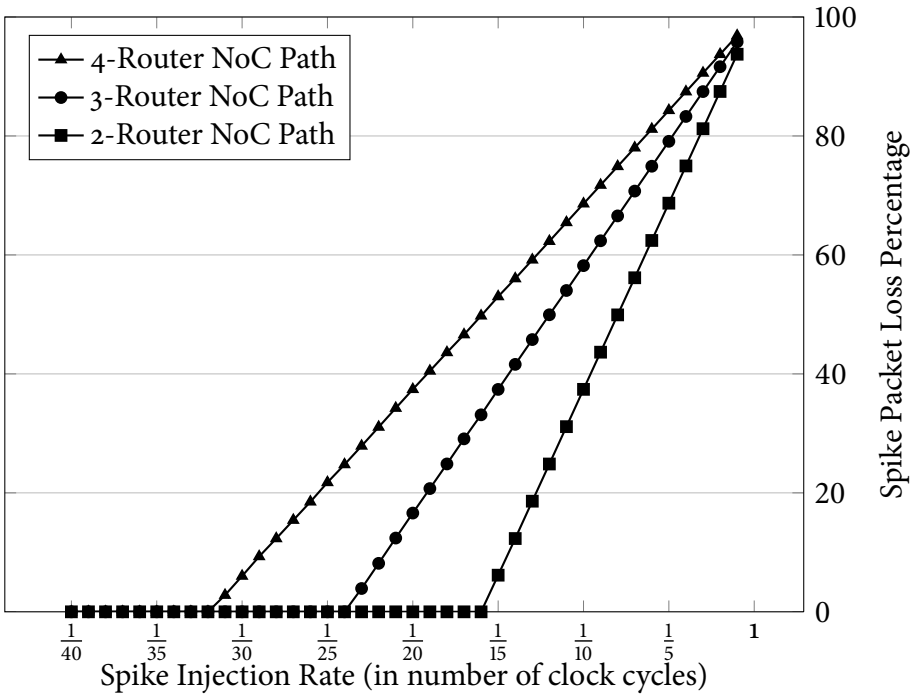


Figure 3.7: Worst Case Spike Packet Injection Rate vs Spike packet Loss for Two, Three and Four Router NoC Paths

state round-robin scheme (N, E, S, W, attached SG/SC/N ports and three extra states currently used for housekeeping tasks). The NoC router introduces a single clock cycle delay in routing the packet to a neighbouring router. The maximum router processing time is therefore 9 clock cycles.

As the spike injection rate increases, NoC routers can no longer route the packet before another incoming packet is received. Each SNN element stores outgoing spike packets in a buffer, which overflows if packets cannot be forwarded.

Bandwidth exhaustion in a NoC router can cause blocking of neighbouring

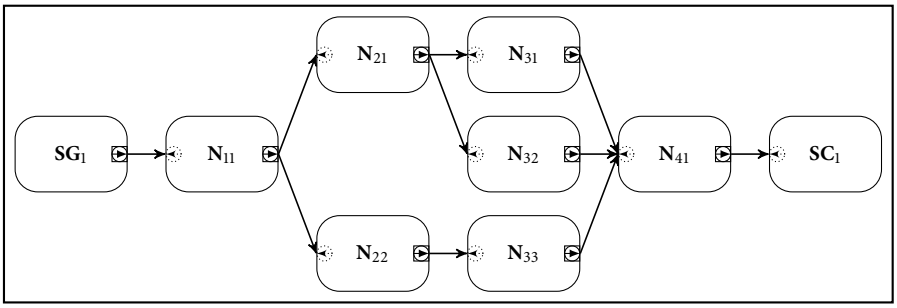


Figure 3.8: SNN Structure for Traffic Hotspot Detection Experiment

routers and loss of spike packets. To avoid blocking of routers and deadlock, the EMBRACE router drops packets if the packet is not received and acknowledged within a fixed (TimeOut) period by a neighbouring router. For a range of SNN applications and configurations, EMBRACE-SysC can be used to verify occurrence of spike packet loss. For future architecture revisions, EMBRACE-SysC will be used to design the NoC architecture with minimum or no packet loss.

The maximum packet acceptance rate of each NoC path is dependent on the number of cascaded routers and the traffic load on each router in the NoC path. This measurement technique can be extended for determining spike injection rates for complex SNN topologies mapped onto the EMBRACE architecture.

3.4.2 NoC Traffic HotSpot Detection

This section demonstrates the use of EMBRACE-SysC for NoC router bandwidth measurement and NoC traffic hotspot detection. The bandwidth of a NoC router is shared between spike packets generated/consumed by the attached SNN element, and spike packets forwarded by the NoC router. Figure 3.8 illustrates an example SNN topology, in which neurons fire and generate an output spike on receipt of an input spike. Spike traffic into the input layer neuron results in spike input to multiple hidden layer neurons. The spike traffic from all of the hidden layer neurons converges at the output layer neuron, which fires at the maximum possible rate and feeds the spikes to the output spike counter. The selected SNN topology and configuration offers a variety of spike rates within the SNN. Figure 3.9 illustrates the mapping of the SNN and synaptic connections onto the EMBRACE-SysC platform incorporating a 4×4 NoC configuration. The mapping ensures that the NoC routers are loaded with distinct packet rates. The SNN topology shown in figure 3.8 results in spikes being routed to multiple hidden layer neurons. Spikes

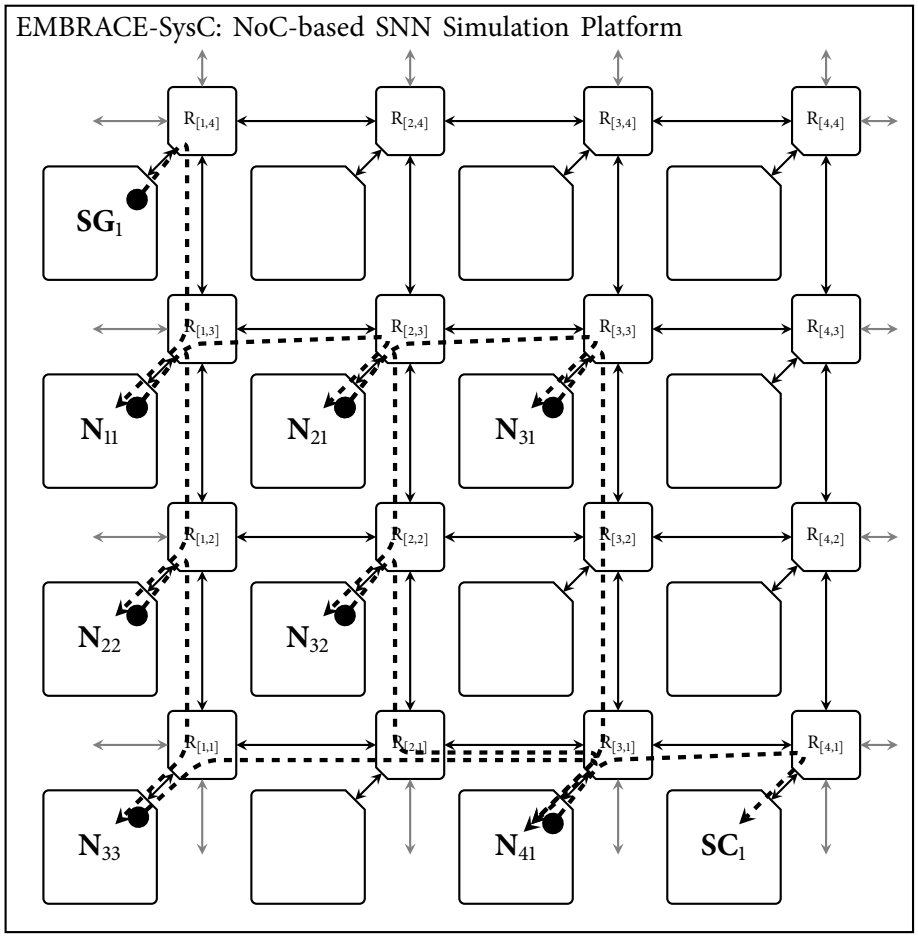


Figure 3.9: Mapping of the SNN Structure onto the 4 × 4 NoC configuration of the EMBRACE architecture

converge at the output layer neuron (N41), which is connected to router R[3,1]. As a result, each router in the network has a different workload, e.g. Router R[3,3] services two NoC paths. R[3,1] services four NoC paths and consumes 100% of the available bandwidth in the router. Any further increase in packet rate on NoC paths going through router R[3,1] causes packet loss.

Figure 3.10 illustrates bandwidth utilisation in each router at the maximum SG1 spike generation rate. This is obtained by gradually increasing the SG1 spike generation rate to a level where spike packet loss in the NoC is about to occur. The level of shade reflects the percentage bandwidth utilisation in each NoC router. Detection of traffic hotspots using EMBRACE-SysC and remapping of the NoC paths can provide improved distribution of traffic across the NoC.

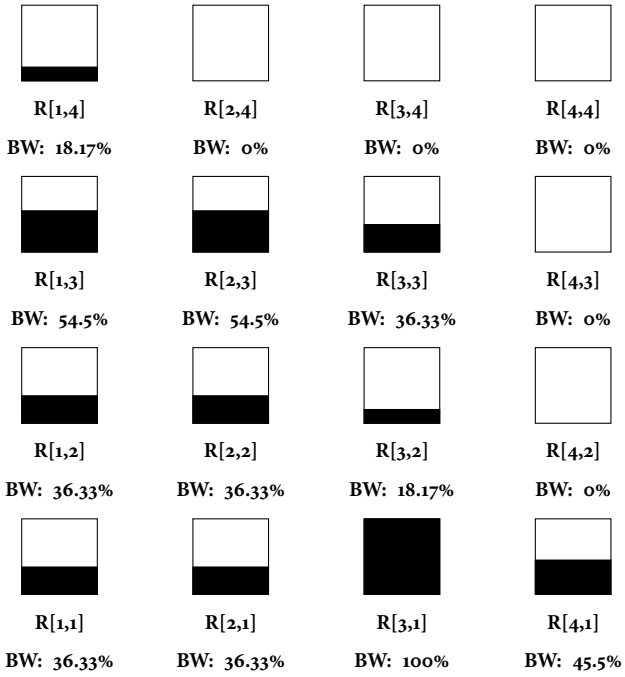


Figure 3.10: Percentage Bandwidth Utilisation in each NoC router

3.4.3 NoC Latency Analysis for an Evolved XOR SNN

This section presents the results of EMBRACE-SysC analysis of NoC latency for an evolved XOR benchmark SNN application and analyses the suitability of the EMBRACE architecture for practical and biologically plausible SNN applications.

The SNN structure shown in figure 3.5a is used to successfully evolve the XOR benchmark SNN application on the EMBRACE-SysC simulation platform. Figure 3.5b illustrates the mapping of the XOR SNN structure and synaptic connections onto EMBRACE-SysC platform incorporating a 3 × 2 NoC configuration. The mapping of synaptic connection SG1 → N11, SG2 → N12 and N21 → SC1 are also shown in figure 3.5b.

Figure 3.11 illustrates the simulated packet latency on the synaptic connections and NoC paths within the evolved XOR benchmark SNN for each of the four XOR input states (logic inputs **00**, **01**, **10** and **11**). Table 3.1 illustrates the selected SG spike rates for the input logic levels.

Figure 3.12 illustrates a detailed view of spike packet latency variation for XOR input state 01. The following illustrates the figure 3.10 behaviour.

For logic input 01: (SG1 ⇒ 0, SG2 ⇒ 1)

a. **NoC Path (a):** R[1,2] receives a high spike packet rate (corresponding to

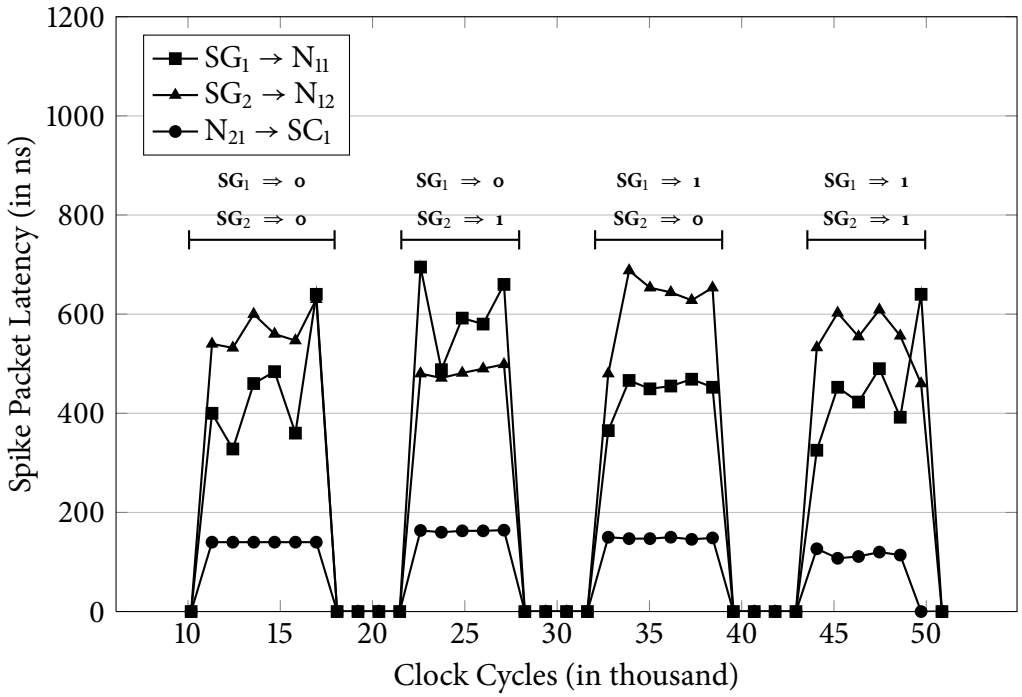


Figure 3.11: Spike Packet Latency for a Synaptic Connections in Evolved XOR Benchmark SNN Application Simulated on EMBRACE-SysC Simulation Platform shown in figure 3.5b

Table 3.1: SG Spike Generation Rates

Logic Inputs	**SG Rate Generation Rate**
0	One spike every 216 clock cycles
1	One spike every 72 clock cycles

logic 1) from SG2. R[1,2] routes packets to N12 via R[1,1] and R[2,1]

b. **NoC Path (b):** R[1,1] receives a low spike packet rate (corresponding to logic 0) from SG1. R[1,1] routes packets to N11 via R[2,1] and R[2,2]

c. **NoC Path (c):** R[3,2] routes packet from N21 to SC1 via R[3,1]

NoC path (b) exhibits the maximum spike packet latency. A goal of this research is to design EMBRACE architecture with minimal packet latency jitter. These results illustrate the use of EMBRACE-SysC to determine packet latency within SNNs.

Detection of high latency NoC paths using EMBRACE-SysC and remapping of these NoC paths can reduce latency for real-time hardware SNN applications.

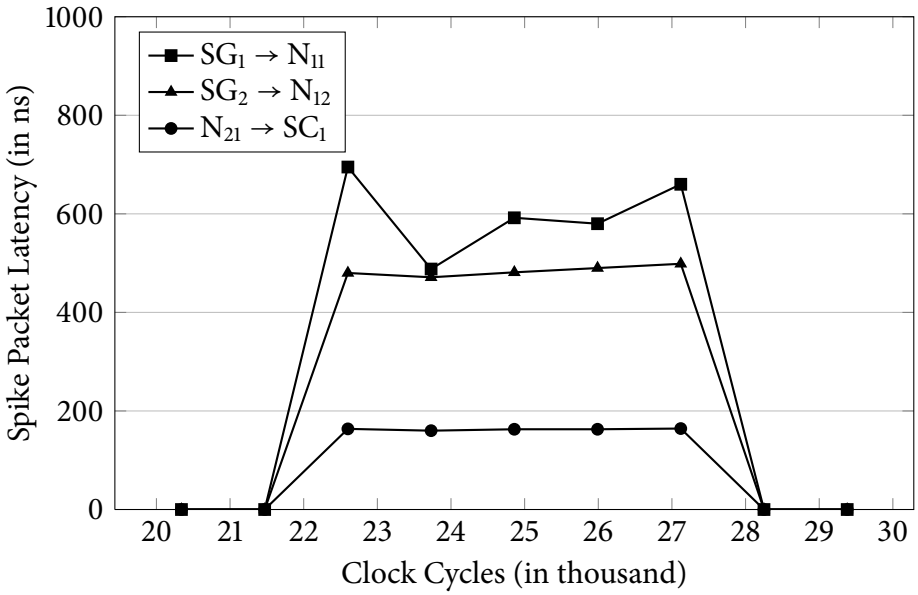


Figure 3.12: Spike Packet Latency for a Selected Synaptic Connections in Evolved XOR Benchmark SNN Application Simulated on EMBRACE-SysC

3.5 Conclusions and Future Work

This paper presents EMBRACE-SysC, a SystemC-based simulation and performance measurement platform for the analysis of the EMBRACE NoC-based SNN architecture. EMBRACE-SysC incorporates a Genetic Algorithm (GA)-based SNN evolution module. Results illustrate the application of EMBRACE-SysC to determine the maximum spike injection rate, to detect traffic hotspots within the NoC, and to analyse NoC packet latency for an evolved XOR SNN application. EMBRACE-SysC enables exploration of design trade-offs including network topology, routing scheme, latency, throughput and synapse/neuron ratio.

The performance results presented will be used as a reference for future EMBRACE designs to reduce spike packet latency, latency jitter and NoC traffic congestion avoidance. Future work will include the development of a performance visualisation tool for the EMBRACE-SysC platform. EMBRACE-SysC will also be used for SNN topology evolution research. The development of EMBRACE-SysC introduces a powerful NoC-based SNN design exploration framework into the EMBRACE research project.

Acknowledgment

This research is supported by International Centre for Graduate Education in Micro and Nano-Engineering (ICGEE), Irish Research Council for Science, Engineering and Technology (IRCSET) and Science Foundation Ireland (Grant No. 07/SRC/I1169).

References

[1] J. Liu and K. Tsui, "Toward nature-inspired computing," *Commun. ACM*, vol. 49, no. 10, pp. 59–64, Oct. 2006.

[2] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[3] W. Gerstner and W. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[4] B. Roche, T. McGinnity, L. Maguire, and L. McDaid, "Signalling techniques and their effect on neural network implementation sizes," *Information Sciences*, vol. 132, no. 1, pp. 67–82, 2001.

[5] L. Maguire, T. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for large-scale implementations of spiking neural networks on FPGAs," *Neurocomputing*, vol. 71, no. 1, pp. 13–29, 2007.

[6] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[7] J. Harkin, F. Morgan, L. McDaid, S. Hall, B. McGinley, and S. Cawley, "A reconfigurable and biologically inspired paradigm for computation using Network-on-Chip and spiking neural networks," *International Journal of Reconfigurable Computing*, p. 2, 2009.

[8] H. Paugam-Moisy, "Spiking neuron networks: a survey," *Rapport Technique RR-11, IDIAP, Martigny, Switzerland*, 2006.

[9] R. VanRullen, R. Guyonneau, and S. Thorpe, "Spike times make sense," *Trends in Neurosciences*, vol. 28, no. 1, pp. 1 – 4, 2005.

[10] Y. Chen, L. McDaid, S. Hall, and P. Kelly, "A programmable facilitating synapse device," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2008, pp. 1615–1620.

[11] F. Morgan, S. Cawley, B. McGinley, S. Pande, L. McDaid, B. Glackin, J. Maher, and J. Harkin, "Exploring the evolution of NoC-based spiking neural networks on fpgas," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, 2009, pp. 300–303.

[12] 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," in *Signals and Systems Conference (ISSC 2010), IET Irish*, 2010, pp. 216–221.

[13] Open SystemC Initiative. Systemc initiative and transaction-level modeling standard. [Online]. Available: http://www.accellera.org/downloads/standards/systemc

[14] P. Rocke, B. McGinley, J. Maher, F. Morgan, and J. Harkin, "Investigating the suitability of FPAAs for evolved hardware spiking neural networks," *Evolvable Systems: From Biology to Hardware*, pp. 118–129, 2008.

[15] S. Kumar, A. Jantsch, J. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*, 2002, pp. 105–112.

[16] L. Jain, B. Al-Hashimi, M. Gaur, V. Laxmi, and A. Narayanan. NIRGAM: a simulator for NoC interconnect routing and application modeling. [Online]. Available: http://nirgam.ecs.soton.ac.uk/

[17] F. Fazzino, M. Palesi, and D. Patti, "Noxim: Network-on-chip simulator," *URL: http://sourceforge. net/projects/noxim [24.06. 2008]*, 2008.

[18] J. Bainbridge and S. Furber, "CHAIN: A delay-insensitive chip area interconnect," *IEEE Micro*, vol. 22, no. 5, pp. 16–23, 2002.

[19] A. Kahng, B. Li, L. Peh, and K. Samadi, "Orion 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proceedings of the conference on Design, Automation and Test in Europe*, 2009, pp. 423–428.

[20] M. Khan, E. Painkras, X. Jin, L. Plana, J. Woods, and S. Furber, "System level modelling for spinnaker cmp system," in *Proc. 1st International Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO.09)*, 2009.

[21] M. Khan, D. Lester, L. Plana, A. Rast, X. Jin, E. Painkras, and S. Furber, "SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2008, pp. 2849–2856.

[22] W. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann, 2003.

[23] A. Eiben and J. Smith, *Introduction to evolutionary computing*. springer, 2008.

CHAPTER 4

Modular Neural Tile Architecture for Compact Embedded Hardware Spiking Neural Network

This work has been published in the Neural Processing Letters Journal (ISSN: 1370-4621 / Springer), Oct 2013. DOI: 10.1007/s11063-012-9274-5

Part of this work has been published and presented at International Conference on Neural Computation Theory and Applications, Paris, France, Oct 2011. DOI: 10.5220/0003676601280137

Preamble

This chapter details the Modular Neural Tile (MNT) architecture that reduces the area and increases the scalability of the EMBRACE hardware SNN architecture.

The computational power of SNNs is often attributed to massive synaptic connectivity between a large number of neurons. Practical SNN application topologies are similarly characterised by a large number of synaptic connections that translate to large distributed on-chip memory in packet switched NoC-based hardware SNN architectures. This large storage requirement grows quadratically and poses a serious challenge for scalability of hardware SNN architectures.

This chapter presents an analysis of the SNN topology memory requirements and its impact on the area and scalability of the NoC based hardware SNN architectures. The chapter suggest the use of Modular Neural Network (MNN) computing

paradigm for mitigating the synaptic connectivity information storage problem and design of practical embedded SNN applications. The chapter presents the MNN execution architecture organisation for hardware SNNs.

The chapter presents detailed design of the Modular Neural Tile (MNT) for EMBRACE hardware SNN architecture and presents the reduction in SNN topology memory requirements by designing the MNT with configurable synaptic connections. A novel topology memory organisation and NoC spike packet flow control for the MNT are presented. The chapter details increased utilisation of the SNN topology memory due to the proposed look-up table and fragmented memory organisation.

The micro-architecture details of the digital neuron model and spike transfer interconnect within the MNT are described, with detailed ASIC and FPGA synthesis results elaborating the compact architecture of the MNT. The chapter presents compact silicon footprint of the EMBRACE hardware architecture using the MNT and demonstrates suitability of the MNT architecture for executing practical application subtasks through successful evolution of benchmark SNN applications.

The research addresses the area and scalability challenge for the design of hardware SNN systems. Development of EMBRACE modular hardware SNN architecture contributes to the design of compact and scalable practical SNN based embedded systems. Future phases of this research extensively use the presented MNT architecture.

Modular Neural Tile Architecture for Compact Embedded Hardware Spiking Neural Network

Sandeep Pande[a], Fearghal Morgan[a], Seamus Cawley[a],
Tom Bruintjes[b], Gerard Smit[b], Brian McGinley[a],
Snaider Carrillo[c], Jim Harkin[c], Liam Mc Daid[c]

[a] Bio-Inspired Electronics and Reconfigurable Computing Research Group (BIRC),
National University of Ireland, Galway, Ireland.
[b] Computer Architecture for Embedded Systems,
University of Twente, Enschede, The Netherlands.
[c] Intelligent Systems Research Centre, University of Ulster,
Magee Campus, Derry, Northern Ireland.

Abstract: *Biologically-inspired packet switched Network on Chip (NoC) based hardware Spiking Neural Network (SNN) architectures have been proposed as an embedded computing platform for classification, estimation and control applications. Storage of large synaptic connectivity (SNN topology) information in SNNs require large distributed on-chip memory, which poses serious challenges for compact hardware implementation of such architectures. Based on the structured neural organisation observed in human brain, a Modular Neural Networks (MNN) design strategy partitions complex application tasks into smaller subtasks executing on distinct neural network modules, and integrates intermediate outputs in higher level functions.*

This paper proposes a hardware Modular Neural Tile (MNT) architecture that reduces the SNN topology memory requirement of NoC-based hardware SNNs by using a combination of fixed and configurable synaptic connections. The proposed MNT contains a 16:16 fully-connected feed-forward SNN structure and integrates in a mesh topology NoC communication infrastructure. The SNN topology memory requirement is 50% of the monolithic NoC-based hardware SNN implementation. The paper also presents a lookup table based SNN topology memory allocation technique, which further increases the memory utilisation efficiency. Overall the area requirement of the architecture is reduced by an average of 66% for practical SNN application topologies.

The paper presents micro-architecture details of the proposed MNT and digital neuron circuit. The proposed architecture has been validated on a Xilinx Virtex-6 FPGA and synthesised using 65nm low-power CMOS technology. The evolvable capability of the proposed MNT and its suitability for executing subtasks within a MNN execution architecture is demonstrated by successfully evolving benchmark SNN application tasks representing classification and non-linear control functions. The paper addresses hardware modular SNN design and implementation challenges and contributes to the development of a compact hardware modular SNN architecture suitable for embedded applications.

4.1 Introduction

Artificial Neural Network (ANN) computing techniques, which are primarily inspired by the functioning of human brain, can provide promising solutions for designing complex and intelligent systems [1]. The organic central nervous system includes a dense and complex interconnection of neurons and synapses, where each neuron connects to thousands of other neurons through synaptic connections. Computing systems based on Spiking Neural Networks (SNNs) emulate real biological neural networks, conveying information through the communication of short transient pulses (*spikes*) between neurons via their synaptic connections. Each neuron maintains an internal *membrane potential*, which is a function of input spikes, associated synaptic weights, current membrane potential, and a constant membrane potential *leakage coefficient* [2, 3]. A neuron *fires* (emits a spike to all connected synapses/neurons) when its membrane potential exceeds the neuron's firing threshold value.

Brain-inspired computing paradigms such as SNNs offer the potential for elegant, low-power and scalable methods of embedded computing, with rich non-linear dynamics, ideally suited to applications including classification, estimation, prediction, dynamic control and signal processing. The efficient implementation of SNN-based hardware architectures for real-time embedded systems is primarily influenced by neuron design, scalable on-chip interconnect architecture and SNN training/learning algorithms [4]. The authors have investigated and proposed EMBRACE[2], as an embedded hardware neural network architecture [5, 6]. The EMBRACE NoC-based SNN architecture is a two-dimensional mesh topology array of neural tiles each comprising a single neuron and NoC router. The NoC-based synaptic connectivity approach employed in the EMBRACE architec-

[2] **EM**ulating **B**iologically-inspi**R**ed **A**r**C**hitectures in hardwar**E**

ture provides a flexible, packet-switched inter-neuron communication channels, scalable interconnect and connection reconfigurability [7, 8].

SNN application topologies are characterised by a large number of synaptic connections that translate to large distributed on-chip memory in packet switched NoC-based hardware SNN architectures. The storage requirement for large SNNs poses a serious challenge for their compact hardware implementation. The Modular Neural Networks (MNN) design strategy partitions complex application tasks into smaller subtasks executing on distinct neural network modules and integrates outputs in higher level functions [1, 9]. The neural network modules in MNN execution architectures maintain internal communication, which is isolated from other modules and the global communication infrastructure. The orthogonalisation of synaptic connectivity suggested in the MNN design paradigm can help tackle the large connectivity problem of SNN architectures and offer practical system implementation for embedded applications.

This paper proposes a hardware Modular Neural Tile (MNT) tile architecture, that reduces the SNN topology memory requirement of NoC-based hardware SNNs by using a combination of fixed and configurable synaptic connections [10]. The proposed MNT comprises a 16:16 fully connected feed-forward SNN structure and supports execution of application subtasks for MNN-based application designs. Fixed connections between the neurons within the MNT remove the requirement for storage of connectivity information. The MNTs integrate in a two-dimensional mesh topology NoC communication infrastructure to form an MNN execution architecture, where the overall SNN topology memory requirement is 50% of the previously reported monolithic NoC-based hardware SNN implementation [5, 6]. The paper also proposes a further architectural enhancement, which involves sharing the SNN topology memory (within each MNT) between the SNN structure outputs. The proposed lookup table based memory allocation scheme increases memory utilisation by offering flexible synaptic connectivity suitable for practical SNN application topologies, which are characterised by irregular connectivity patterns [11]. In total, the area requirements of the architecture is reduced by 66%.

The paper presents micro-architecture details of the proposed MNT and the digital neuron circuit used for system validation. The architectural components are synthesised using 65nm low-power CMOS technology and silicon area results are presented. The proposed MNT architecture has been validated on a Xilinx Virtex-6 FPGA and resource usage results are reported. The evolvable capability of the proposed MNT and its suitability for executing application subtasks in a

MNN execution architectures is demonstrated by successfully evolving the XOR benchmark SNN application and a robotics obstacle avoidance controller using the player-stage robotics simulator and previously reported Genetic Algorithm based hardware SNN evolution platform. (These benchmark SNN applications represent data classification and non-linear control functions.)

The structure of the paper is as follows: Section 4.2 summarises reported hardware SNN architectures and their suitability as embedded hardware SNNs. Section 4.3 summarises the reported EMBRACE NoC-based hardware monolithic SNN architecture and highlights the impact of SNN topology memory on the overall device size and the challenge of reducing the SNN topology memory. Section 4.4 introduces the modular neural network computing paradigm, its benefits and its application for reducing topology memory within the EMBRACE hardware SNN. Section 4.5 describes the proposed MNT hardware design including neural computing module, packet encoder/decoder, digital neuron design, topology memory and SNN configuration memory. This section also reports ASIC and FPGA synthesis results. Section 4.6 presents classification and non-linear control benchmark functions implemented on the EMBRACE MNT FPGA prototype. Section 4.7 concludes the paper.

4.2 Hardware SNN Architectures

Inspired by biology, researchers aim to implement reconfigurable and highly interconnected arrays of neural network elements in hardware to produce computationally powerful and cognitive signal processing units [12–21]. This section summarises reported hardware and hybrid SNN architectures and their suitability as embedded hardware SNNs.

A hybrid SNN computing platform is reported in [17]. This platform includes a neuron model implemented in hardware and the network model and learning implemented in software. A time multiplexed FPGA embedded processor SNN implementation [20] reports 4.2K neurons and >1.9M synapses. The neuron model and partial SNN elements are implemented on an embedded FPGA processor, where speed-acceleration is the key motivation. The system relies on external memory for spike transfer management. Analogue spiking neuron design approaches can benefit from a compact area implementation due to their ability to model electrical charge flow in the brain [13, 18, 18, 22, 23]. These architectures rely on digital components for a flexible communication infrastructure. FACETS,

a configurable wafer-scale mixed-signal neural ASIC system, proposes a hierarchical neural network and the use of analogue floating gate memory for storage of synaptic weights [19, 21]. A mixed-signal SNN architecture of 2,400 analogue neurons, implemented using switched capacitor technology and communicating via an asynchronous event-driven bus has been reported in [18]. The chip area is reported to be $3mm \times 3mm$ using $0.5\mu m$ CMOS VLSI technology.

Practical SNN systems are characterised by a large numbers of neurons and high interconnectivity through inter-neuron synaptic connections. Each of the SNN execution architectures presented in [14–21] aim for thousands of neurons and millions of synapses, which is essential for a powerful neural computing platform. For large scale hardware implementation of SNNs, the neuron interconnect imposes problems due to high levels of inter-neuron connectivity and often the number of neurons that can be realised in hardware is limited by high fan in/out requirements [4]. Direct neuron-to-neuron interconnection exhibits switching requirements that grow exponentially with the network size. Efficient, low area and low power implementations of neuron interconnect and synaptic junctions are therefore key to scalable hardware SNN implementations [4].

The NoC design paradigm provides a promising solution for the flexible interconnection of large SNNs [7]. The SpiNNaker project [14] aims to develop a massively parallel computer capable of simulating SNNs of various sizes, topology and with programmable neuron models. The SpiNNaker architecture uses ARM-968 processor-based nodes for computation and an off-chip NoC communication infrastructure. Each NoC tile in the SpiNNaker system models 1000 Leaky-Integrate-and-Fire (LIF) neurons, each having 1000 synapse inputs. Each SpiNNaker node requires approximately 4MBytes of memory for storing synaptic connectivity information [24]. Hence, the SpiNNaker architecture stores the synaptic connection data in off-chip SDRAM. The SNN implementations described above are not targeted at compact embedded hardware applications. A clustered embedded hardware SNN is proposed in [25] along with a process for mapping SNNs to hardware. The variable sized neuron clusters support flexible synaptic connections even from and to within the clusters. The authors have reported the scalable NoC-based EMBRACE [5] hardware SNN architecture, targeting compact embedded hardware applications. The architecture supports large monolithic SNN topologies. In a monolithic SNN, the neuron connectivity is global (i.e. any neuron can connect to any other neuron). The EMBRACE topology memory requirement increases exponentially with network size [10]. Modular neural network

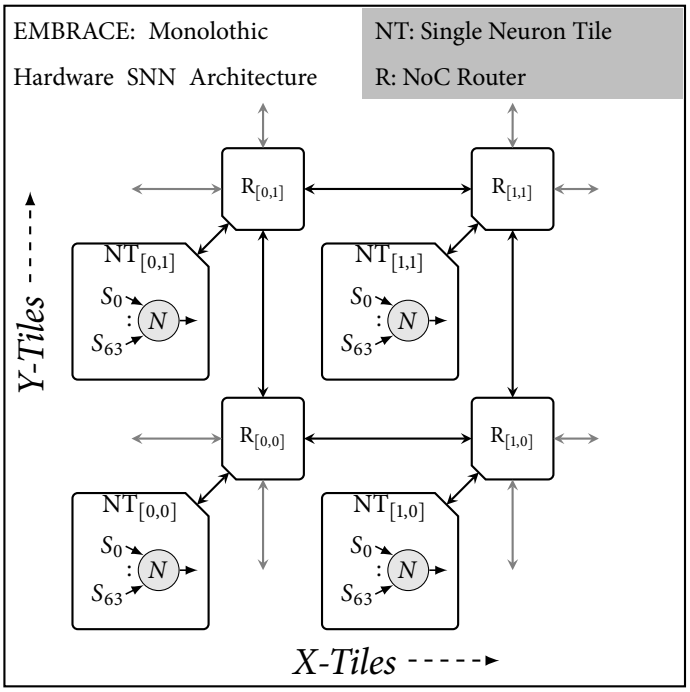


Figure 4.1: EMBRACE NoC-based Embedded Hardware Monolithic SNN Architecture (The Neural Tile (NT) comprises a single neuron, packet encoder/decoder, SNN configuration and topology memory)

techniques investigated in this paper, can enable trade-offs between the topology memory size and synaptic connectivity, leading to scalable, more compact, practical embedded hardware SNN structures.

4.3 EMBRACE: Hardware SNN Architecture

This section summarises the reported EMBRACE NoC-based hardware monolithic SNN architecture and analyses the impact of SNN topology memory on the overall EMBRACE device size and the challenge of reducing the SNN topology memory.

4.3.1 EMBRACE: NoC-based Embedded Hardware Monolithic SNN Architecture

The EMBRACE (figure 4.1) [5] architecture incorporates neural circuits within a digital NoC-based packet switching interconnect to realise a scalable hardware monolithic SNN architecture suitable for embedded systems. This architecture offers high synaptic densities while maintaining a small silicon footprint and low

power consumption. EMBRACE is a two-dimensional mesh topology array of Neural Tiles (NT) and NoC Routers (R). Each neural tile comprises a single neuron supporting up to 64 input and 64 output synaptic connections. The embedded hardware architecture supports the implementation of monolithic SNNs and provides a benchmark for comparison of the work of this paper. The SNN topology memory defines each inter-neuron synaptic connection. NoC routers are connected in North (N), East (E), South (S) and West (W) directions, forming a Manhattan-style, two-dimensional mesh topology NoC architecture. An application specific SNN is realised on the EMBRACE architecture by programming neuron configuration parameters (SNN synaptic weights and neuron firing threshold potential) and SNN connection topology. Spike communication within the SNN is achieved by routing spike information within spike data packets over the network of routers. The reported architecture (figure 4.1) requires 11MBytes of SNN topology memory to support a 64K neuron/4M synapse hardware SNN.

The authors have implemented and reported EMBRACE-FPGA [8], an FPGA prototype implementation of the EMBRACE architecture. The EMBRACE-FPGA prototype has been successfully applied to benchmark SNN control and classifier applications (such as pole balancer, two-input XOR and Wisconsin cancer dataset classifier). EMBRACE-SysC, a SystemC-based clock cycle accurate simulation and performance measurement platform for simulation and analysis of EMBRACE architecture has been reported in [26]. EMBRACE-SysC enables rapid architectural exploration and performance analysis of the EMBRACE architecture.

4.3.2 EMBRACE Hardware Resource Analysis

The transistor count and chip area has been estimated for the EMBRACE architecture to understand the practicality of realising the EMBRACE architecture in silicon. The silicon area estimation technique takes into account the transistor count for storage and control logic of digital components (based on standard SRAM cell design) and actual silicon area for neurons [23].

Figure 4.2 presents the silicon area proportions (in mm^2) by scaling the EMBRACE hardware monolithic SNN architecture in $32nm$ CMOS VLSI technology. The x-axis indicates the number of neurons and synapses (Neuron/Synapse) and the stacked columns in the histogram denote the silicon area for NoC and SNN architectural entities described below:

- **NoC infrastructure:** The EMBRACE NoC infrastructure comprises NoC

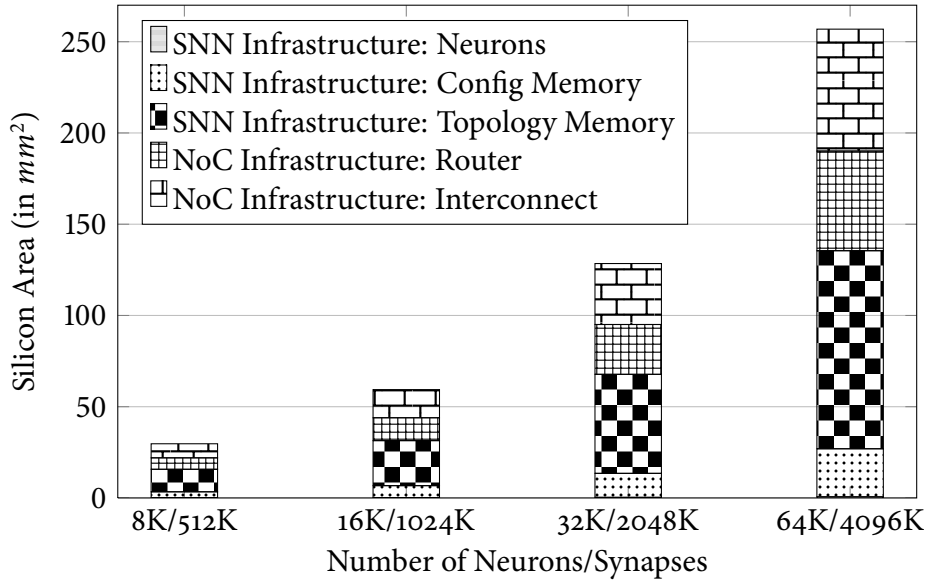


Figure 4.2: Silicon Area Proportion (for 32*nm* CMOS technology) for the EMBRACE Hardware Monolithic SNN Architecture

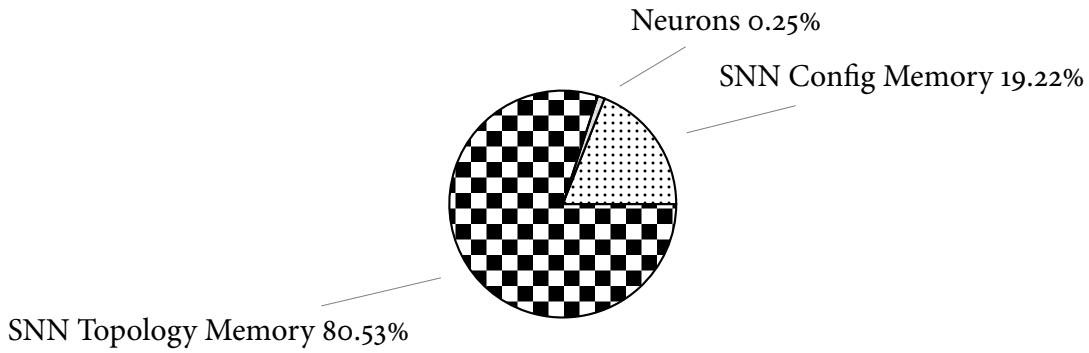


Figure 4.3: Silicon Area Proportion for the SNN Infrastructure Entities within EMBRACE Hardware Monolithic SNN Architecture

routers, packet buffers, NoC interconnect channels and associated control circuits. The analytical estimates indicate that the complete NoC infrastructure will occupies 47.27% of the total chip area.

- **SNN infrastructure:** The SNN support infrastructure is made-up of SNN configuration memory (for storing synaptic weights of 5 bits each and threshold values of 16 bits each) and the SNN topology memory (for storing synaptic connectivity information) [6]. The SNN infrastructure also contains the neural elements (which includes synapses, synaptic weight summing and membrane potential thresholding circuits) [13, 22, 23]. Due to its compact implementation, the silicon area for the neural elements is negligible com-

pared to the rest of the SNN support infrastructure, which occupies 52.74% of the total chip area. (Figure. 4.3 further enumerates the silicon area of the SNN components within the EMBRACE architecture.)

Architectural techniques to reduce SNN topology memory are crucial for their compact hardware implementations. This paper presents a novel modular neural tile architecture that reduces the SNN topology memory area by 50%.

4.4 Modular Neural Networks

This section introduces the Modular Neural Network (MNN) computing paradigm and execution architecture. The section highlights the benefits and application of the MNN design approach to reduce topology memory within the reported EMBRACE hardware monolithic SNN architecture.

4.4.1 Biological Motivations for the MNN Computing Paradigm

The biological brain is composed of several anatomically and functionally discrete areas [27]. Various brain areas and neuron groups are dedicated to various sensory and motor tasks. For example, the visual cortex processes different aspects of the visual information (such as form, colour and motion) in separate anatomically distinct regions. These different regions exchange information but remain functionally discrete [28, 29]. Damage or deterioration of part of the visual cortex can result in a partial loss of colour identification, pattern or motion detection, etc. without considerably affecting other senses. Inspired by this modular organisation in brain, the MNN design strategy of partitioning application tasks into a number of subtasks has been developed [30–32].

4.4.2 Modular Neural Network Computing Paradigm

The MNN computing paradigm is primarily based on the *divide-and-conquer* strategy. Figure. 4.4 illustrates the MNN design methodology, which primarily consists of *task decomposition* i.e. breaking down a high level application into smaller, less complex, manageable subtasks. These small subtasks are then solved by individual and distinct neural modules. The intermediate outputs from these

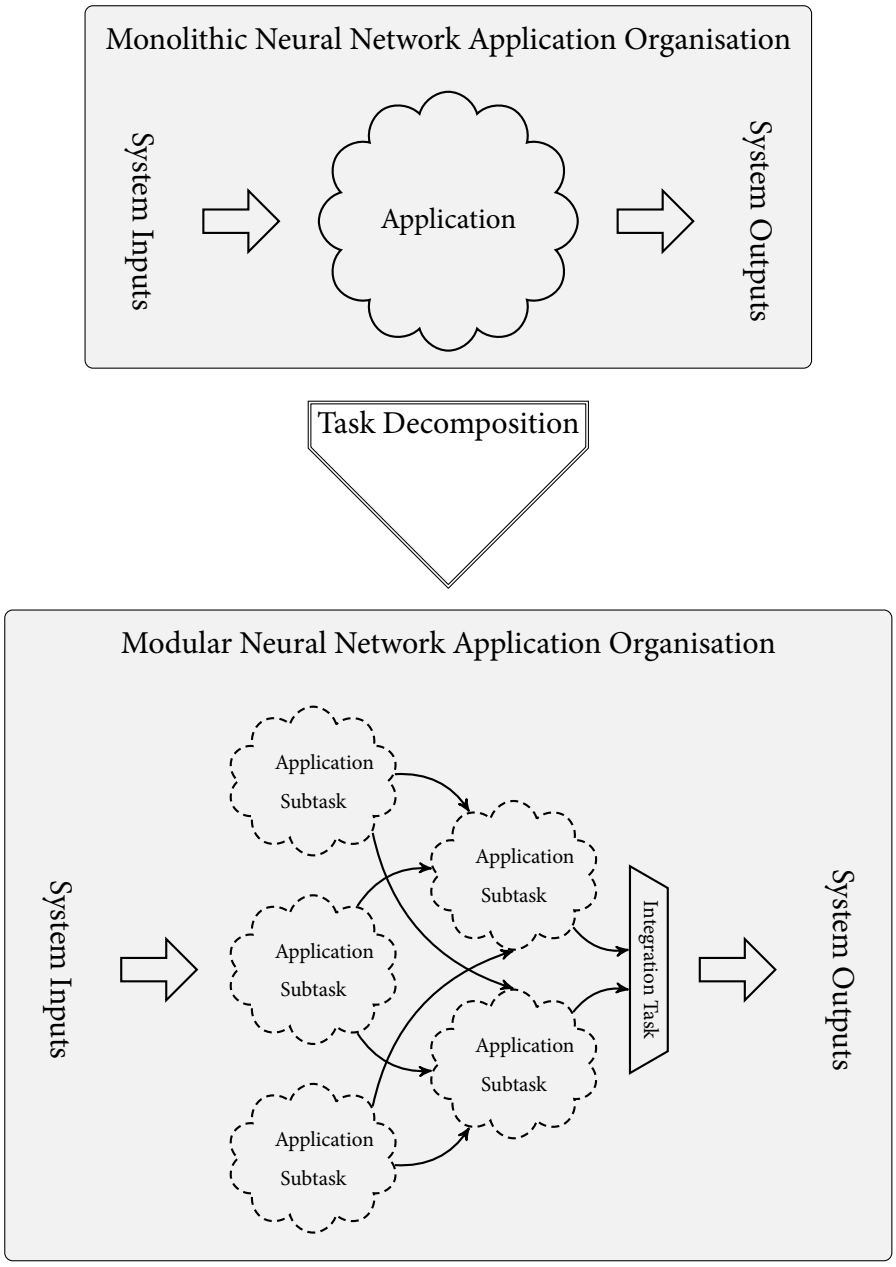


Figure 4.4: Modular Neural Network Design Methodology

neural modules are combined to solve the high level task or the whole application [1, 9]. Various task decomposition algorithms have been proposed for the MNN design strategy. These algorithms are based on different techniques such as output vector partitioning, class relationships, neuro-evolutionary approach and co-evolutionary methods [33–37]. Similarly, genetic algorithm based technique to find subnetworks from large sized complex neural networks has been proposed in [38]. Subtasks/Subnetworks obtained after task decomposition are executed as neural modules in MNN computing architecture.

The MNN approach offers structured implementation, functional partition, functional mapping and re-mapping, competitive and co-operative mode of operation and fault tolerance [31]. The *Subsumption Architecture*, a widely influential computing paradigm for reactive, behaviour-based autonomous robotic control applications has been developed based on the MNN design concepts [39].

4.4.3 Modular Neural Network Execution Architectures

In the MNN design methodology, the task decomposition or partitioning of the overall application leads to two types of subtasks; namely the application subtasks (discrete functional subtasks) and the integration tasks. The application subtasks operate on individual and distinct system inputs to provide intermediate outputs. The integration subtasks integrate the intermediate outputs from the application subtasks to generate the overall system output. Both the application and integration subtasks are similar in terms of input/output interface and computation requirements. Fig. 4.5 illustrates a typical MNN execution architecture comprising individual neural modules interconnected by a global communication infrastructure. Based on the definition of modularity in neural networks [1, 9], the individual neural modules in the MNN execution architecture should:

- Include a group of neurons interconnected using an internal communication infrastructure
- Support multiple inputs and/or multiple outputs
- Include an internal communication infrastructure that is isolated from the other modular neural elements and the global (external inter-module) communication infrastructure

Since the synaptic communication between neurons in a neural module is isolated from the rest of the architecture, the requirement for the storage of associated

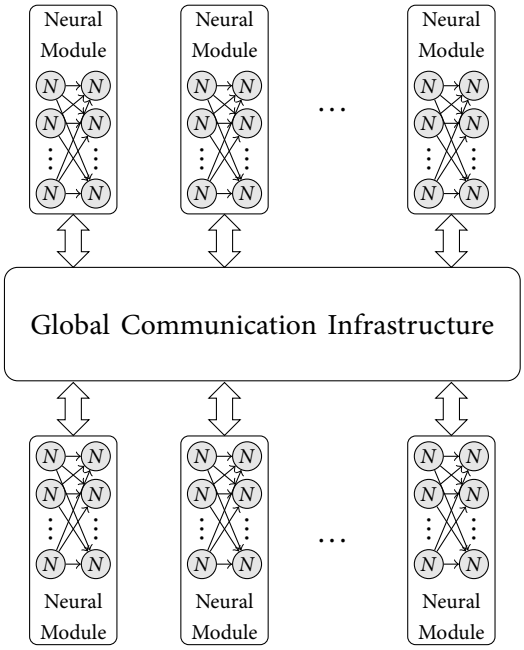


Figure 4.5: Modular Neural Network Execution Architecture

connectivity information is eliminated. The overall MNN architecture has a considerably lower synaptic connectivity storage requirement compared to monolithic neural architectures. The SNN topologies for applications that are inherently non-modularizable exhibit uniform connectivity patterns. The neural modules in the MNN execution architecture can be cascaded using global communication infrastructure to realise such large monolithic SNN structures. However, the resource utilisation of the MNN execution architecture will be higher as compared to hardware SNN arrays supporting uniform synaptic connectivity.

4.5 Modular Neural Tile Architecture

This paper presents a novel Modular Neural Tile (MNT) architecture, its digital prototype and evolvable capabilities. The proposed MNT forms the basic neural module for the MNN execution architecture (proposed NoC-based modular hardware SNN execution architecture).

Architectural techniques for reducing the SNN topology and configuration memory are vital for compact silicon implementation of hardware SNN architectures suitable for embedded computing. This section presents the MNT architecture comprising a 16:16 fully connected feed-forward topology SNN structure as the Neural Computing Module (NCM) and the lookup table-based SNN topology

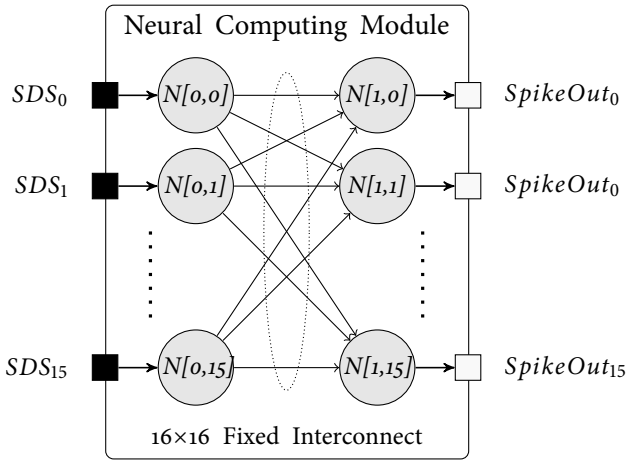


Figure 4.6: Two layered 16:16 Fully Connected SNN Structure as the Neural Computing Module within the Proposed MNT

memory sharing scheme. The micro-architecture of an efficient digital neuron model is also described. Detailed ASIC and FPGA synthesis results are presented. Silicon area requirement of the NoC architecture employing the proposed MNT is compared with the previously reported EMBRACE hardware monolithic SNN architecture.

4.5.1 Neural Computing Module

The Neural Computing Module (NCM) forms the basic computing entity within the MNN execution architecture. The NCM micro-architecture design is primarily influenced by the following:

- **VLSI Technology Limitations:** Fixed interconnections between neurons within the SNN structure removes the need for synaptic connectivity information storage within the MNT. The size of the fully-connected hardware SNN structure is limited by the permitted fan-out of individual neuron circuits. Also, metal layer routing limitations are imposed due to interconnect crossbar capacitance and crosstalk.
- **MNN Subtask Granularity:** The NCM should support sufficient computing power for MNN application subtasks and integration subtasks. Large NCM designs, can accommodate a variety of MNN application subtasks, but would lead to unused neurons in the case of fine granularity MNN application subtasks. NCM designs with smaller SNN structures can be cascaded based

on the computing requirements of the particular MNN application and integration subtasks.

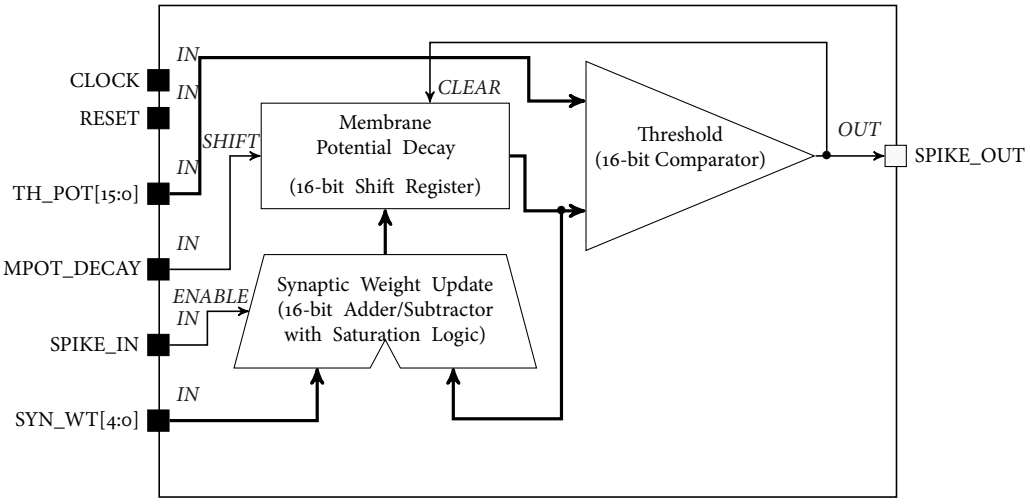
Considering the above, a two-layered 16:16 fully connected feed-forward SNN structure (figure 4.6) has been proposed as the Neural Computing Module (NCM) inside each MNT [10]. The input layer ($N[0, n]$) and output layer ($N[1, n]$) of the NCM comprises 16 Leaky-Integrate-and-Fire (LIF) neurons. The neurons support a *Single Dynamic Synapse* (SDS_n) approach (figure 4.7a), where the synaptic weight is supplied along with spike input. This shared synapse approach removes the need for internal multiplexers for selection of synaptic weight and results in compact hardware implementation [40]. Each of the 16 input layer neurons connects directly to each of the 16 output layer neurons to form a fully connected feed-forward SNN structure. Each output layer neuron has 16 input synapses, which individually receive spikes from the corresponding input layer neurons. The NCM has 16 spike outputs ($SpikeOut_n$) each corresponding to the 16 output layer neurons.

The NCM in a MNN execution architecture should be capable of solving application subtasks and integration tasks. The proposed modular neural computing module made-up of 32 LIF neurons, supports multiple synaptic inputs and provides 16 spike outputs that can connect to multiple synaptic inputs in the architecture. This paper demonstrates the suitability of the proposed NCM by evolving SNN benchmark functions.

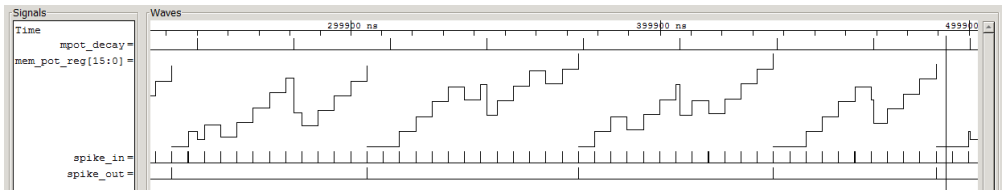
Digital Neuron Model

This section describes the multiplier-less, compact hardware digital neuron design.

Based on biological plausibility, computational power and implementation complexity, various mathematical models representing the spiking behaviour of biological neurons have been proposed [2, 3]. Amongst them the Leaky-Integrate-and-Fire (LIF) model is a popular choice for hardware SNN architectures due to its simplicity. The proposed EMBRACE research project ultimately aims to develop a mixed-signal VLSI architecture with compact, low power, high-resolution CMOS-compatible analogue synapse and LIF neuron cells [13, 23] to achieve very high synaptic density. The LIF neuron model has multiple synaptic inputs, a single spike output and maintains an internal *membrane potential*, which constantly decays (to its resting value) based on a constant *leakage coefficient*. The synaptic weight value associated with the input synapse is summed with the current membrane potential value on receipt of an input spike. The exhibitory/inhibitory synaptic



(a) Micro-Architecture



(b) Simulation Waveform

Figure 4.7: Digital Neuron Model

weights increase/decrease the membrane potential by the weight value. The neuron *fires* (emits a spike) when the membrane potential reaches the threshold potential value. Firing of the LIF neuron causes the membrane potential to reset to the resting value.

For validation of the proposed MNT architecture on FPGA, a digital neuron circuit exhibiting LIF behaviour with sufficient resolution has been developed (figure 4.7a). The digital neuron circuit uses a 16-bit shift register for storage of the *Membrane Potential* value. Stepwise exponential decay of the membrane potential value is achieved by periodic right shift (divide by 2) controlled by the MPOT_DECAY input pulse. The desired decay rate or *Leakage Coefficient* can be controlled by programming the membrane potential decay strobe generator (figure 4.7b). The membrane potential value is updated by the input *Synaptic Weight* (SYN_WT) value using a 16-bit adder/subtractor enabled by the incoming spike pulse (SPIKE_IN). The overflow/underflow bit of the adder/subtractor is used to determine the upper/lower limit of the membrane potential value, and activate the saturation logic. A 16-bit comparator generates the spike output (SPIKE_OUT) if the internal $MembranePotential > Threshold$ (TH_POT) input. The generated

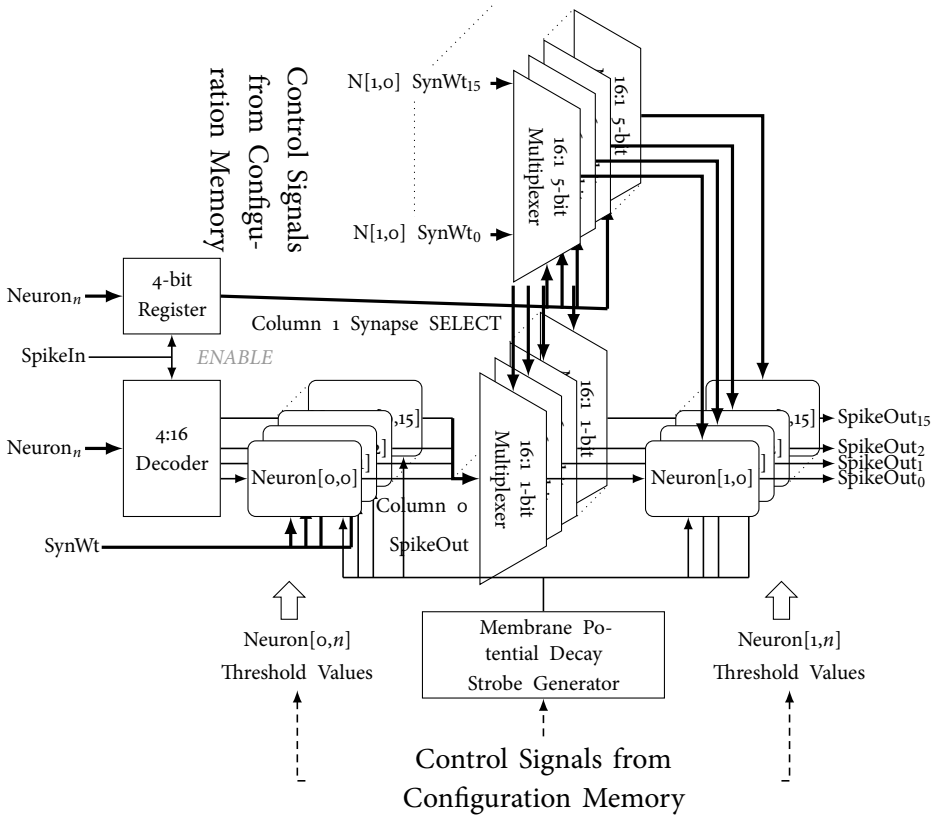


Figure 4.8: Neural Computing Module Micro-Architecture

spike pulse (SPIKE_OUT) is also used to clear the 16-bit shift register, bringing the membrane potential to the resting value.

Figure 4.7b shows a simulation waveform for the digital neuron circuit including membrane potential decay due to MPOT_DECAY pulses, membrane potential variation due to input spikes, and output spike generation. The synthesizable digital neuron circuit provides the required resolution for practical SNN applications. The neuron occupies only 12 slices in the Xilinx Virtex-6 FPGA and occupies $608.4\mu m^2$ in 65nm CMOS VLSI technology. This compact neuron model contributes to the goal of portable embedded hardware SNNs.

Neural Computing Module Architecture

Fixed interconnection between neurons within the neural computing module removes the need to store synaptic connectivity information. Figure 4.8 shows the fixed connection micro-architecture of the NCM. The interconnect architecture transfers the generated spikes from the input layer neurons to output layer neurons. The interconnect architecture enables selection of synaptic weights from

configuration memory for the output layer neurons while applying spikes.

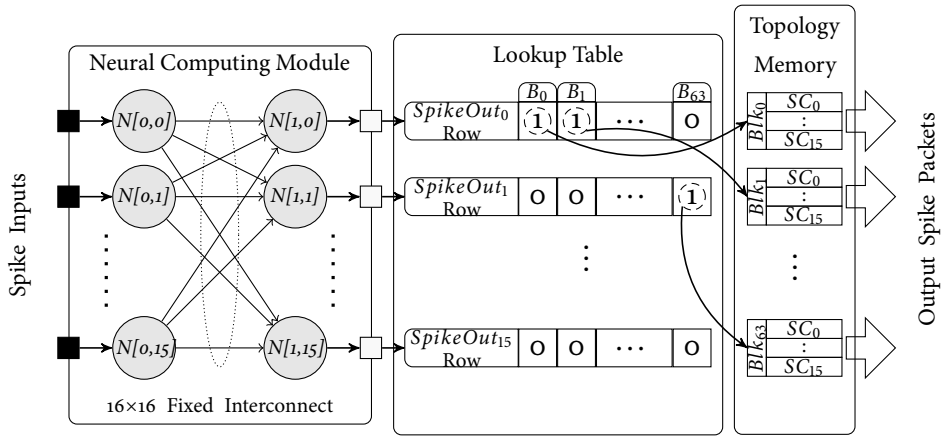
The input interface to the NCM consists of neuron number (Neuron_n), synaptic weight (SynWt) and input spike pulse (SpikeIn). The NCM layers operate in a pipelined fashion, where the input spike is applied to the selected input layer neuron during the first clock cycle and any generated spike is transferred to output layer neurons in the next clock cycle. The input neuron number (Neuron_n) is stored in a 4-bit register for use in the next clock cycle. The input synaptic weight is directly connected to all of the input layer neurons. The 4:16 decoder enables the selected input neuron number. The decoder is enabled by the input spike pulse, which causes the selected neuron to activate and update its internal membrane potential.

According to the LIF neuron behaviour [3], neurons can generate a spike only on the occurrence of an input spike. Hence, the stored neuron number in n^{th} clock cycle is used for transfer of spike and selection of synaptic weight for output layer neurons in $n + 1^{th}$ clock cycle within the NCM. The NCM output interface comprises spike outputs ($\text{SpikeOut}_{0:15}$) from all output layer neurons. Spike pulses generated by the NCM are further processed within the MNT to generate output spike packets for synaptic connections outside the MNT.

4.5.2 Topology Memory Sharing and Spike Packet Output Flow Control

Spike communication between MNTs is achieved by routing spike information within spike data packets over the network of routers. Within each MNT a lookup table based SNN topology memory allocation technique is implemented, which enables variable synaptic connection densities for efficient topology memory resource usage. This section describes spike packet generation based on the proposed lookup table-based topology memory sharing scheme, which offers a flexible number of synaptic connections for NCM outputs.

Figure 4.9a illustrates the lookup table-based shared topology memory organisation. The synaptic connection information for the NCM outputs is stored in this topology memory. Fields include destination MNT address (*[X,Y]* mesh topology NoC tile address), destination neuron ($Neuron_n$) and synaptic weight ($SynWt$) (see figure 4.9b). The topology memory is partitioned into 64 blocks (B_0 to B_{63}), where each block is made-up of 16 synaptic connection information entries (B_xSC_0 to B_xSC_{15}), giving a total of 1024 neuron/synapse destinations. The lookup table



(a) The Lookup Table-based Shared Topology Memory Organisation

$X_{Address}$	$Y_{Address}$	$Neuron_n$	$SynWt$
NoC Tile X Address	NoC Tile Y Address	MNT Input Neuron Number	Synaptic Weight
(4-bit)	(4-bit)	(4-bit)	(5-bit)

(b) Synaptic Connection (SC) Information Entry Stored in the Topology Memory

Figure 4.9: The MNT Spike Packet Output Flow Control

maintains the topology memory block allocation information in designated rows for each NCM output. Each bit in the 64-bit lookup table row allocates the corresponding topology memory block to the NCM output. For example, bit number B_x of the row number *N[1,0]* allocates topology memory block number X to the NCM output *N[1,0]*. Also, for the row number *N[1,0]*, asserting the bit value B_x, allocates the topology memory block X to NCM output *N[1,0]*; deasserting B_x disassociates the topology memory block X from the NCM output *N[1,0]*. The packet encoder generates spike packets for NCM outputs based on the allocated topology memory blocks.

The process of mapping the MNN application topology onto the proposed MNT involves populating the lookup table and MNT topology memory entries, such that the correct synaptic connections are established between the neural computing modules in the architecture. If the required number of synaptic connections for a particular NCM cannot be accommodated in the available topology memory, additional MNTs can be used as spike repeaters. The NCM can be configured as spike repeater by configuring synaptic weights and threshold to generate a spike for each input spike.

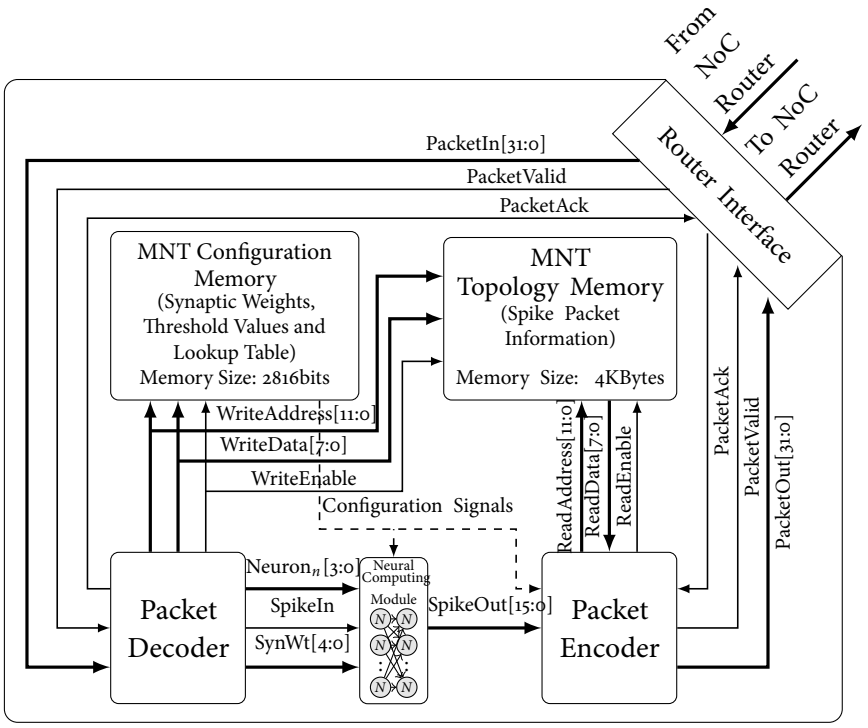


Figure 4.10: Modular Neural Tile Internal Architecture

4.5.3 Modular Neural Tile Hardware Design

The modular neural tile (figure 4.10) comprises a Neural Computing Module (NCM), configuration memory (for storing neuron synaptic weights and threshold values), topology memory (for storing synaptic connectivity information) and packet decoder and encoder. This section describes the interfaces and functionality of the hardware entities within the proposed MNT.

Modular Neural Tile Internal Architecture

The MNT connects to the NoC router through the router interface, which comprises 32-bit PacketIn/PacketOut, valid (PacketValid) and acknowledgment (PacketAck) handshake signals. The MNT supports the following packet types (figure 4.11):

- **Configuration Packet:** used for configuration of neurons (synaptic weights and threshold values), the lookup table and output synaptic connectivity. The configuration packet consists of destination tile address (XY NoC tile address), configuration memory address (13-bit) and data (8-bit).

Configuration Packet Format

B_{31}-B_{28}	B_{27}-B_{24}	B_{23}-B_{21}	B_{20}-B_8	B_7-B_0
$X_{Address}$ (4-bit) NoC Tile X Address	$Y_{Address}$ (4-bit) NoC Tile Y Address	$PacketType$ (3-bit) 001 ⇒ Spike Packet; 010 ⇒ Configuration Packet	$ConfigAddress$ (13-bit) Configuration and Topology Memory Address	$ConfigData$ (8-bit) Configuration and Topology Memory Data

Spike Packet Format

B_{31}-B_{28}	B_{27}-B_{24}	B_{23}-B_{21}	B_{20}-B_{12}	B_{11}-B_8	B_7-B_5	B_4-B_0
$X_{Address}$ (4-bit) NoC Tile X Address	$Y_{Address}$ (4-bit) NoC Tile Y Address	$PacketType$ (3-bit) 001 ⇒ $SpikePacket$; 010 ⇒ $ConfigurationPacket$	$RESERVED$ (9-bit)	$Neuron_n$ (4-bit) MNT Input Layer Neuron Number	$RESERVED$ (3-bit)	$SynWt$ (5-bit) MNT Input Layer Neuron Synaptic Weight

Figure 4.11: MNT Configuration and Spike Packet Format

- **Spike Packet:** used for transferring spikes from source MNT to destination MNT. It consists of the destination tile address 2-D array format (XY NoC tile address), neuron number ($Neuron_n$) and synaptic weight. Inclusion of synaptic weight in the spike packet reduces the overall storage requirement of the architecture, and removes the necessity for synaptic weight selection multiplexers for the NCM inputs, resulting in compact hardware implementation [40].

The packet decoder interfaces with the NCM, configuration and topology memory by receiving and decoding the input packets. The input packet type (spike or configuration packet) is decoded from the $PacketType$ bits (B_{23}-B_{21}). For a configuration packet, the configuration memory address and data are retrieved from the packet and written to configuration or topology memory. For a spike packet, neuron number ($Neuron_n$) and synaptic weight ($SynWt$) are retrieved from the packet and forwarded to the NCM along with an internally generated spike pulse. The configuration memory supplies synaptic weight and threshold values to the NCM and lookup table bits to the packet encoder through dedicated control signals. The topology memory is a dual-ported RAM module as it interfaces with the packet decoder for memory write operation and with the packet encoder for memory read operation. The packet encoder generates individual spike packets based on spike inputs from the NCM, synaptic connectivity information in the topology memory and memory block allocation information from the lookup table. Refer to section 4.5.2 for the detailed output spike packet flow control.

Table 4.1 shows the detailed hardware synthesis results for the proposed MNT. ASIC synthesis has been performed using Synopsys Design Compiler 2009-sp5

Table 4.1: Modular Neural Tile Synthesis Results (Clock Frequency: 200Mhz)

	ASIC Synthesis (65nm Low-Power CMOS)		FPGA Synthesis (Xilinx Virtex-6 FPGA)	
RTL Entity	Area (in μm^2)	Percentage	Slices	BRAMs
Modular Neural Tile	110806.63	100.0%	1204	0
Neural Computing Module	20505.24	18.7%	458	0
Digital Neuron Circuit	608.40	0.5%	12	0
Packet Decoder	408.71	0.4%	7	0
Packet Encoder	2368.60	2.1%	66	0
NCM Configuration Memory	31444.47	28.4%	662	0
NCM Topology Memory	55800.00	50.4%	0	1

and TSMC $65nm$ low-power CMOS libraries. FPGA synthesis has been carried out using Xilinx XST 13.2. Since the configuration memory supplies all the MNT control signals (2816 bits, synaptic weights, threshold values and lookup table) to the NCM and packet encoder, the silicon footprint is considerably higher compared to memory size. Due to massive interconnectivity in neural architectures, the majority of the area is occupied by topology memory (50% in the proposed MNT). The topology memory is realized using STMicroelectronics low-power CMOS dual-ported memory IP in ASIC implementation.

Figure 4.12 compares the silicon area of various sized SNNs MNT NoC architecture with that of the reported EMBRACE hardware monolithic SNN architecture. The reduced number of NoC routers, resulting from the 16:16 fully connected SNN (NCM), decreases the area occupied by the NoC infrastructure in the proposed MNT NoC architecture by 89% as compared to the reported EMBRACE SNN architecture. The fixed interconnection within the NCM removes the need for storing the output synaptic connectivity information for the input layer neurons. The regularly structured interconnect requires much less silicon area than the SRAM-based synaptic connectivity storage and the associated control circuitry. Consequently, the topology memory for the proposed MNT NoC architecture is reduced by 54.05%. The size of the entire MNT NoC-based hardware SNN is approximately 33% of that of the previously reported EMBRACE chip area estimation.

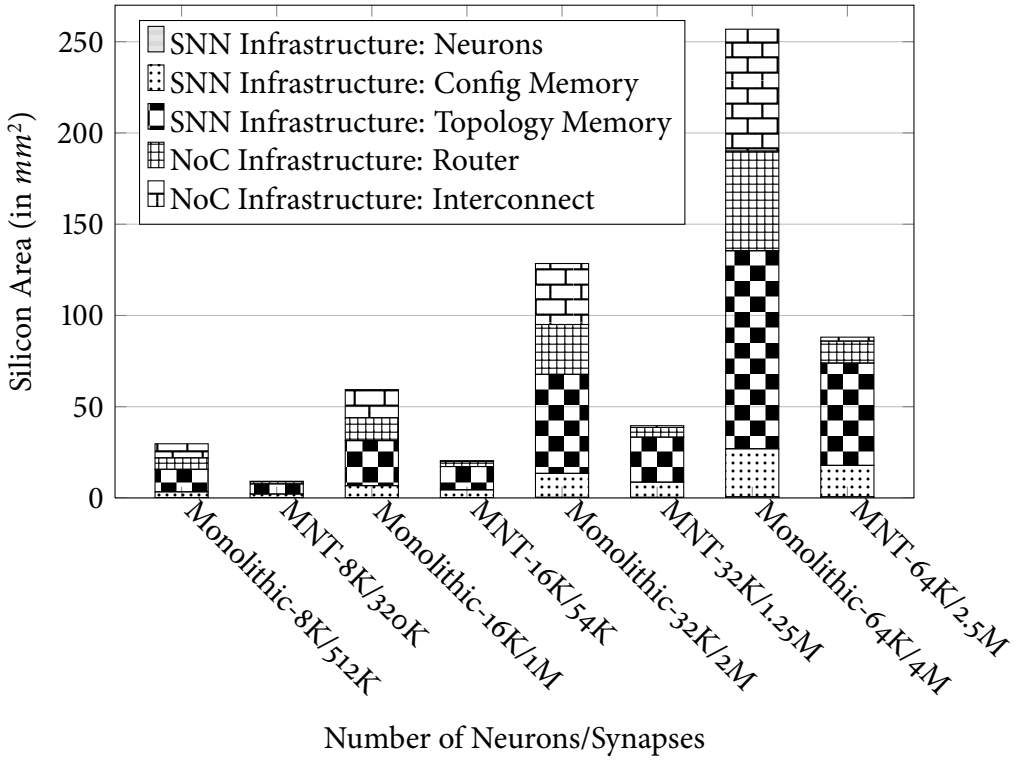
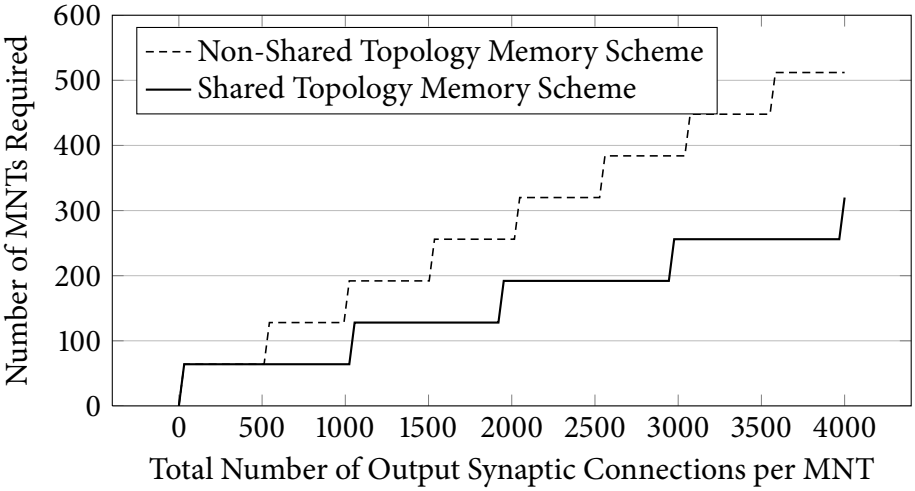


Figure 4.12: Silicon Area Estimate (for 32nm CMOS technology) Comparison for the EMBRACE Monolithic and the Proposed MNT Hardware SNN Architecture

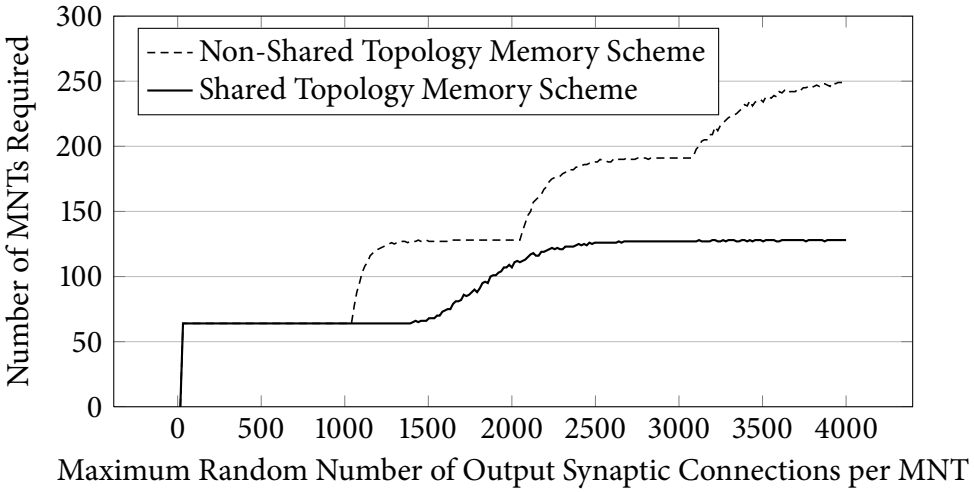
Hardware Resource Requirements of Practical SNN Topologies

Practical SNN application topologies exhibit a variety of connectivity patterns. Flexible sharing of the topology memory within the MNT addresses diverse connectivity requirements of the practical modular SNN application topologies. This section presents and compares hardware resource requirements for the proposed MNT architecture with shared and non-shared topology memory schemes for SNN application topologies with irregular and random connectivity patterns [11]. Additional MNTs are used for relaying spikes, if the synaptic connectivity requirement of the NCM cannot be accommodated in the topology memory within the MNT.

A large modular neural network application, made-up of 64 individual NCMs, has been mapped to the proposed MNT-based NoC architecture. The application implementations using non-shared and shared topology memory configurations have been compared. The non-shared topology memory scheme uses a fixed allocation of 4 topology memory blocks to each NCM output. The NCMs in the shared topology format are configured such that 8 outputs from each NCM remain



(a) Irregular Connections



(b) Random Connections

Figure 4.13: NoC Tile Requirements for Non-Shared and Shared Topology Memory Schemes for the (a) Irregularly and (b) Randomly Connected Example Modular Topology

inactive (by configuring zero synaptic connections). The number of required MNTs and the size of the NoC are calculated for various synaptic connection densities in the remaining 8 active NCM outputs. Figure 4.13a compares the number of MNTs required using non-shared and shared topology memory approach, executing the MNN application topology with irregular synaptic connectivity pattern. The topology memory in the MNT can hold 1K (i.e. 1024) synaptic connection entries. When the synaptic connectivity requirement of each NCM increases by 1K steps, additional set of MNTs are used for relaying spike packets. This can be seen in the step wise ascending graph in figure 4.13a.

The SNN topologies evolved using Genetic Algorithm (GA) based search methods often exhibit random connectivity patterns [11]. The application topology described above has been configured for a random number of output synaptic connections from each of the 64 individual NCMs. This MNN application representing a random synaptic connectivity pattern has been mapped to the proposed MNT NoC architecture and tested under non-shared and shared topology memory configuration. Figure 4.13b illustrates the MNT requirement for the proposed MNT NoC architecture under non-shared and shared topology memory scheme, executing the application topology with a random synaptic connectivity pattern.

The proposed shared topology memory architecture facilitates the allocation of topology memory blocks to the NCM outputs based on the synaptic connectivity requirement. The lookup table based shared topology memory architecture offers a flexible number of synaptic connections from the NCM outputs resulting in efficient usage of each MNT. Figure 4.13 illustrates that the shared topology memory scheme requires a smaller number of MNTs for MNN application topologies with irregular and random synaptic connectivity patterns (observed in practical SNN application topologies). This enables implementation of larger application topologies within the given architectural configuration.

4.6 Modular Neural Tile Applications

This section presents classification and controller benchmark functions implemented on the EMBRACE modular neural tile FPGA prototype. The benchmark XOR function (a basic data classifier) and robotics controller (closed loop nonlinear control system) have been used to test the evolvable capability of the proposed MNT. Accuracy (fitness) of the SNN configuration and training time results (in terms of GA generation count) are presented for successfully evolved XOR

Table 4.2: Two-Input XOR Function Fitness Score Assignment

Number of Correct Outputs	0	1	2	3	4
Fitness Score Assignment Value	0	1	4	9	16

and robotics controller application on the proposed MNT prototype. This demonstrates the capability of the proposed MNT to evolve small-sized subtasks within a larger MNN applications. The intrinsic evolution setup comprising the proposed MNT prototype on FPGA and GA-based SNN configuration platform running on host computer [8], highlight the hardware validation aspects of the modular SNN architecture.

4.6.1 Classification Subtasks

The XOR function is a basic benchmark data classification problem for neural networks and is a sub-problem of more complicated classifiers [41]. A two-input XOR function is implemented on the proposed MNT. The XOR function uses two spike rate encoders feeding distinct spike rates for logic '0' and logic '1' and an output spike rate decoder. The GA-based SNN evolution and configuration platform configures the SNN configuration comprising synaptic weights and threshold potential for all the neurons in the SNN, check the accuracy (fitness) of the configuration and evolves the desired functionality by searching the correct configuration. The evolved XOR function on the proposed MNT outputs a high spike rate when the two inputs differ in spike rate, and a low spike rate if the inputs are equal. Table 4.2 illustrates the fitness score assignment used for the two input binary XOR function. The three neuron XOR SNN partially uses the modular neural tile. Figure 4.14 illustrates the average and best fitness of the evolved SNN XOR function on the proposed MNT.

4.6.2 Non-Linear Control Subtasks

The robotics controller application is a classical example of a non-linear closed loop control system. Neural network systems are widely used as controllers for practical robotics applications. A robotics obstacle avoidance controller, using the player-stage robotics simulator [42] and the previously reported Genetic Algorithm (GA)-based hardware SNN evolution and implementation platform (figure 4.15). The simulated robot is equipped with 16 sonar sensors. The average values of front,

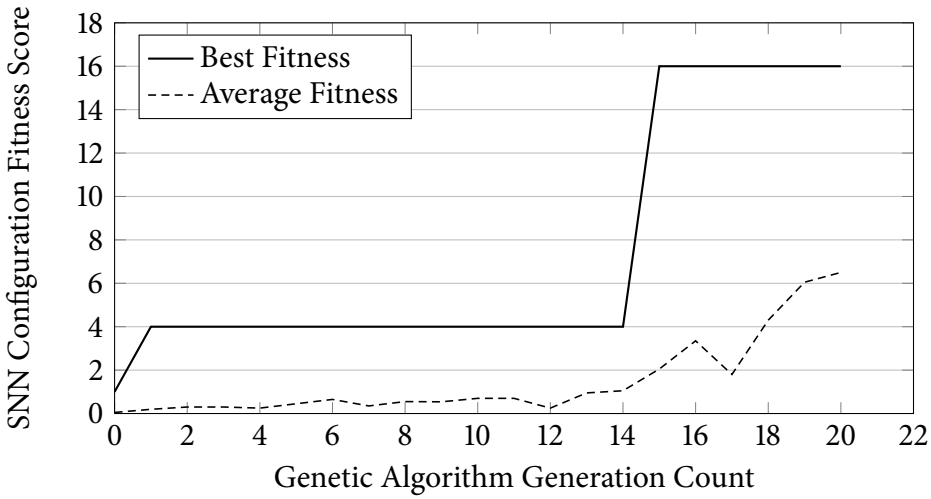


Figure 4.14: XOR Benchmark SNN Application on the Proposed MNT

rear and side sonars are used as inputs to the SNN. The sonar values are converted to spike rates by the host application and are passed to spike rate encoders, which continuously generate and feed spikes to the Neural Computing Module (NCM) within the MNT. Spikes from two neural computing module outputs are monitored by spike rate decoders and are converted to analogue values as robot acceleration and turning angle inputs to the simulator.

Fitness criteria for the robotic obstacle avoidance controller application is defined as travelling finite distance and avoiding obstacles for ≥ 120 seconds, and the fitness of the SNN configuration is calculated as:

$$F = \alpha T + \beta D + \gamma S \tag{4.1}$$

Where:

F = Fitness of the individual
T = Robot travel time (in $seconds$)
D = Robot travel distance (in cms)
S = Robot travel speed (in cm/sec)

Given:

Number of crashes = 0
SNN outputs are within the operating range

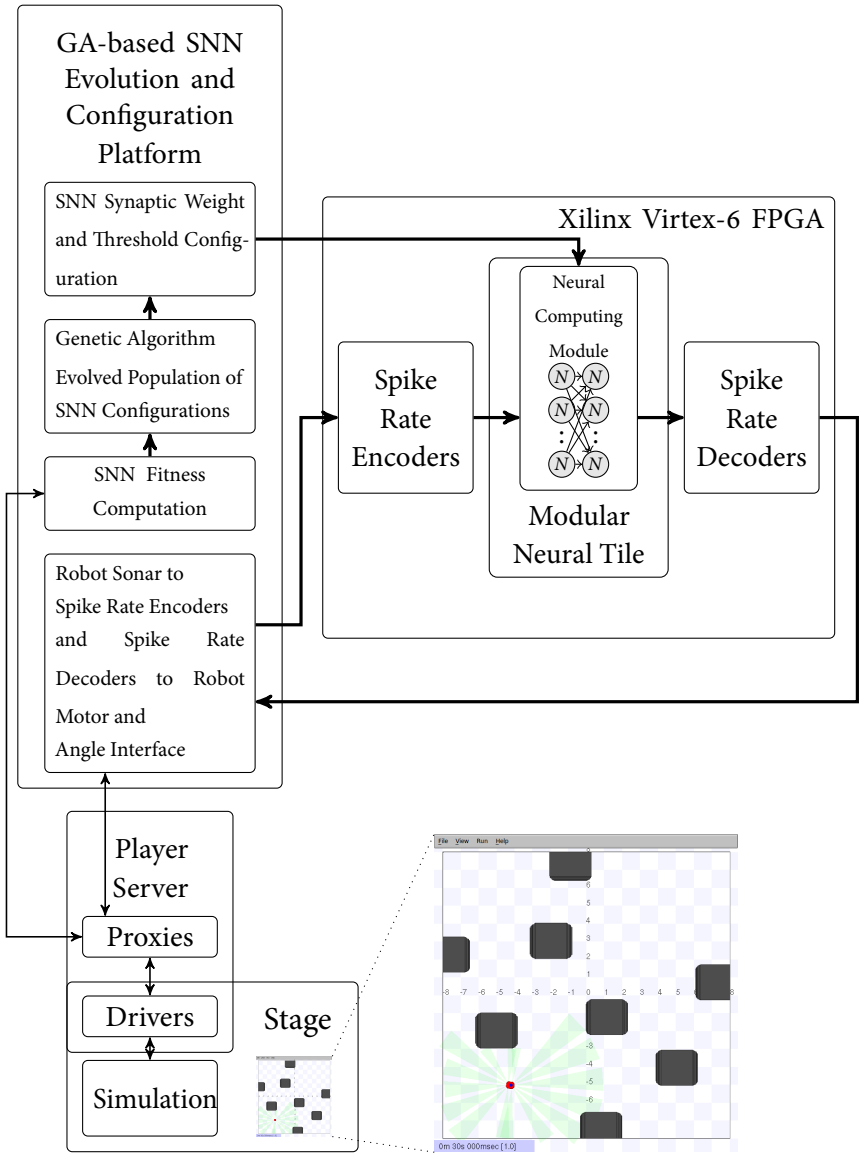


Figure 4.15: Robotic Obstacle Avoidance Controller Setup

The fitness evaluation constants α, β and γ are chosen to prioritise the robot motion behaviour.

Evaluation of the individual configurations has been accomplished with the robot roaming within the simulated environment for $300 \geq t > 120$. On timeout, or if the robot has crashed, the GA-based evolution and configuration platform processes the recorded robot behaviour and assigns a fitness score to the individual SNN configuration. Fitness scores are then used by the GA to determine the probability of an individual configuration progressing to later evolved generations. Figure 4.16 illustrates the average and best fitness of the evolved robotic obstacle

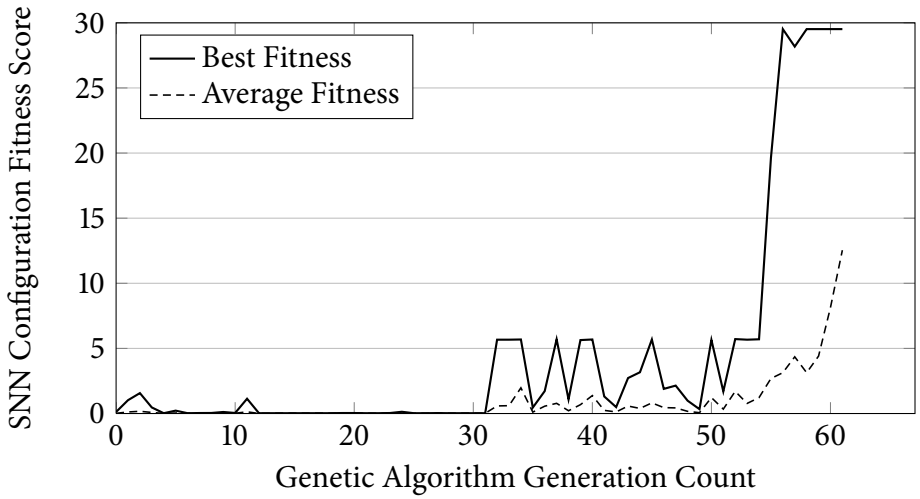


Figure 4.16: Robotic Obstacle Avoidance Controller Benchmark SNN Application on the Proposed MNT (Fitness Evaluation Constants: α = 2, β = 1 and γ = 0.5)

avoidance controller application on the proposed MNT.

Successful evolution of classification and non-linear control functions with constant and time varying input patterns on the proposed MNT demonstrates its suitability for a variety of MNN application designs.

4.7 Conclusions

Neural architectures are typically characterised by thousands of neurons and millions of synapses which are essential for a powerful neural computing platform. Storage of the large synaptic connectivity information in packet switched Network on Chip (NoC)-based hardware Spiking Neural Network (SNN) architectures translates to large distributed on-chip memory and poses a serious challenge for compact hardware implementations. Discrete synaptic connectivity observed in MNN execution architectures help in reducing the storage requirement of NoC-based hardware neural architectures.

This paper presented a novel Modular Neural Tile (MNT) architecture comprising a 16:16 fully connected feed-forward topology SNN structure as neural computing module. The topology memory of the architecture is 50% of the previously reported NoC-based hardware monolithic SNN implementation. Furthermore, a lookup table-based topology memory sharing scheme is presented that provides a flexible number of synaptic connections from Neural Computing Module (NCM) outputs. The proposed shared topology memory scheme requires less number

of MNTs for practical MNN application topologies with irregular and random synaptic connectivity patterns. Overall the area requirement of the architecture is reduced by an average 66% for practical SNN application topologies. This facilitates accommodation of larger application topologies in the given architectural configuration.

The paper presented micro-architecture details of the proposed MNT and the digital neuron circuit used for system validation. The architectural components are synthesised using $65nm$ low-power CMOS technology and silicon area results are presented. The proposed MNT architecture has been validated on Xilinx Virtex-6 FPGA and resource utilisation is presented. The evolvable capability of the proposed MNT, and its suitability for executing application subtasks, in a modular hardware SNN architecture is demonstrated by successfully evolving the XOR benchmark SNN function and a robotics obstacle avoidance controller, using the player-stage robotics simulator and the previously reported Genetic Algorithm (GA)-based hardware SNN evolution and configuration platform. Successful evolution of classification and non-linear control functions with constant and time varying input patterns on the proposed MNT demonstrates its suitability for variety of MNN application designs.

The architectural enhancements proposed and validated in this paper helps to achieve a compact neural modular hardware implementation and demonstrate the ability of the proposed MNT to successfully evolve benchmark SNN functions. This work contributes to the development of the EMBRACE NoC-based embedded hardware SNN device [12].

Acknowledgment

This research is supported by International Centre for Graduate Education in Micro and Nano-Engineering (ICGEE), Irish Research Council for Science, Engineering and Technology (IRCSET), Science Foundation Ireland (Grant No. 07/SRC/I1169) and Xilinx University Programme.

References

[1] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999, vol. 13.

[2] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[3] W. Gerstner and W. Kistler, *Spiking neuron models*. Cambridge University Press, 2002.

[4] L. Maguire, T. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for large-scale implementations of spiking neural networks on FPGAs," *Neurocomputing*, vol. 71, no. 1-3, pp. 13–29, Dec. 2007.

[5] J. Harkin, F. Morgan, L. McDaid, S. Hall, B. McGinley, and S. Cawley, "A reconfigurable and biologically inspired paradigm for computation using network-on-chip and spiking neural networks," *Int. J. Reconfig. Comput.*, vol. 2009, pp. 1–13, 2009.

[6] 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*, vol. 12, pp. 257–280, Apr. 2011.

[7] D. Vainbrand and R. Ginosar, "Scalable network-on-chip architecture for configurable neural networks," *Microprocessors and Microsystems*, vol. 35, no. 2, pp. 152–166, 2011, special issue on Network-on-Chip Architectures and Design Methodologies.

[8] F. Morgan, S. Cawley, B. McGinley, S. Pande, L. McDaid, B. Glackin, J. Maher, and J. Harkin, "Exploring the evolution of NoC-based spiking neural networks on FPGAs," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, 2009, pp. 300–303.

[9] D. Osherson, S. Weinstein, and M. Stob, "Modular learning." Cambridge, MA, USA: MIT Press, 1993, pp. 369–377.

[10] S. Pande, F. Morgan, S. Cawley, B. McGinley, J. Harkin, S. Carrillo, and McDaid, "Addressing the hardware resource requirements of Network-on-Chip based neural architectures," in *NCTA, International Conference on Neural Computation Theory and Applications, Paris, France*, Oct. 2011.

[11] N. Kohl and R. Miikkulainen, "Evolving neural networks for fractured domains," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ser. GECCO '08. New York, NY, USA: ACM, 2008, pp. 1405–1412.

[12] J. Harkin, F. Morgan, S. Hall, P. Dudek, T. Dowrick, and L. McDaid, "Reconfigurable platforms and the challenges for large-scale implementations of spiking neural networks," in *International Conference on Field Programmable Logic and Applications, 2008. FPL 2008*. IEEE, Sep. 2008, pp. 483–486.

[13] Y. Chen, S. Hall, L. McDaid, O. Buiu, and P. Kelly, "A solid state neuron for the realisation of highly scaleable third generation neural networks," in *Solid-State and Integrated Circuit Technology, 2006. ICSICT '06. 8th International Conference on*, 2006, pp. 1071–1073.

[14] S. Furber and A. Brown, "Biologically-Inspired Massively-Parallel architectures - computing beyond a million processors," in *Application of Concurrency to System Design, 2009. ACSD '09. Ninth International Conference on*, 2009, pp. 3–12.

[15] A. Upegui, C. Peña-Reyes, and E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks," *Microprocessors and Microsystems*, vol. 29, no. 5, pp. 211–223, Jun. 2005.

[16] M. Pearson, A. Pipe, B. Mitchinson, K. Gurney, C. Melhuish, I. Gilhespy, and M. Nibouche, "Implementing spiking neural networks for Real-Time Signal-Processing and control applications: A Model-Validated FPGA approach," *Neural Networks, IEEE Transactions on*, vol. 18, no. 5, pp. 1472–1487, 2007.

[17] E. Ros, E. Ortigosa, R. Agis, R. Carrillo, and M. Arnold, "Real-time computing platform for spiking neurons (RT-spike)," *Neural Networks, IEEE Transactions on*, vol. 17, no. 4, pp. 1050–1063, 2006.

[18] R. Vogelstein, U. Mallik, J. Vogelstein, and G. Cauwenberghs, "Dynamically reconfigurable silicon array of spiking neurons with Conductance-Based synapses," *Neural Networks, IEEE Transactions on*, vol. 18, no. 1, pp. 253–265, 2007.

[19] M. Ehrlich, C. Mayr, H. Eisenreich, S. Henker, A. Srowig, A. Grubl, J. Schemmel, and R. Schuffny, "Wafer-scale VLSI implementations of pulse coupled neural networks," in *Proceedings of the International Conference on Sensors, Circuits and Instrumentation Systems*, 2007.

[20] B. Glackin, T. McGinnity, L. Maguire, Q. Wu, and A. Belatreche, "A novel approach for the implementation of large scale spiking neural networks on FPGA hardware," in *Computational Intelligence and Bioinspired Systems*, 2005, pp. 552–563.

[21] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, june 2008, pp. 431–438.

[22] Y. Chen, S. Hall, L. McDaid, O. Buiu, and P. Kelly, "On the design of a low power compact spiking neuron cell based on Charge-Coupled synapses," in *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, 2006, pp. 1511–1517.

[23] Y. Chen, L. McDaid, S. Hall, and P. Kelly, "A programmable facilitating synapse device," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, june 2008, pp. 1615–1620.

[24] S. Furber, S. Temple, and A. Brown, "On-chip and inter-chip networks for modeling large-scale neural systems," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, 2006, p. 4 pp.

[25] R. Emery, A. Yakovlev, and G. Chester, "Connection-centric network for spiking neural networks," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, 2009, pp. 144–152.

[26] S. Pande, F. Morgan, S. Cawley, B. McGinley, S. Carrillo, J. Harkin, and L. McDaid, "EMBRACE-SysC for analysis of NoC-based spiking neural network architectures," in *System on Chip (SoC), 2010 International Symposium on*, Sept 2010, pp. 139–145.

[27] S. Johannes, B. Wieringa, M. Matzke, and T. Münte, "Hierarchical visual stimuli: electrophysiological evidence for separate left hemispheric global and local processing mechanisms in humans," *Neuroscience letters*, vol. 210, no. 2, pp. 111–114, May 1996.

[28] D. Van Essen, C. Anderson, and D. Felleman, "Information processing in the primate visual system: an integrated systems perspective," *Science*, vol. 255, no. 5043, pp. 419–423, Jan. 1992.

[29] T. Binzegger, R. Douglas, and K. Martin, "Stereotypical bouton clustering of individual neurons in cat primary visual cortex," *The Journal of Neuroscience*, vol. 27, no. 45, pp. 12 242–12 254, Nov. 2007.

[30] B. Happel and J. Murre, "Design and evolution of modular neural network architectures," *Neural Networks*, vol. 7, no. 6-7, pp. 985–1004, 1994.

[31] G. Auda and M. Kamel, "Modular neural networks a survey," *International Journal of Neural Systems*, vol. 9, no. 2, pp. 129–151, 1999.

[32] Ronco and P. Gawthrop, "Modular neural networks: a state of the art," *Rapport Technique CSC95026 Center of System and Control University of Glasgow*, vol. 1, pp. 1–22, 1995.

[33] S. Guan, S. Li, and S. Tan, "Neural network task decomposition based on output partitioning," *Journal of the Institution of Engineers Singapore*, vol. 44, pp. 78–89, 2004.

[34] B. Lu and M. Ito, "Task decomposition and module combination based on class relations: a modular neural network for pattern classification," *Neural Networks, IEEE Transactions on*, vol. 10, no. 5, pp. 1244–1256, sep 1999.

[35] J. Thangavelautham and G. D.eleuterio, "A neuroevolutionary approach to emergent task decomposition," in *Proc. of 8th Parallel Problem Solving from Nature*. Springer, 2004, pp. 991–1000.

[36] V. Khare, Y. Xin, B. Sendhoff, Y. J., and H. Wersing, "Co-evolutionary modular neural networks for automatic problem decomposition," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3, sept 2005, pp. 2691–2698.

[37] J. Santos, L. Alexandre, and J. de Sa, "Modular neural network task decomposition via entropic clustering," in *Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on*, vol. 1, oct. 2006, pp. 62–67.

[38] C. Pizzuti, "A multiobjective genetic algorithm to find communities in complex networks," *Evolutionary Computation, IEEE Transactions on*, vol. 16, no. 3, pp. 418–430, june 2012.

[39] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, mar 1986.

[40] S. Cawley, S. Pande, L. McDaid, B. McGinley, and F. Morgan, "Memory efficient storage of reconfigurable topology information in network-on-chip based spiking neural networks," *Bio-Inspired Electronics and Reconfigurable Systems, National University of Ireland, Galway. Internal Report*.

[41] J. Maher, B. McGinley, P. Rocke, and F. Morgan, "Intrinsic hardware evolution of neural networks in reconfigurable analogue and digital devices," in *Field-Programmable Custom Computing Machines, 2006. FCCM '06. 14th Annual IEEE Symposium on*, 2006, pp. 321–322.

[42] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm Intelligence*, vol. 2, pp. 189–208, Aug. 2008.

CHAPTER 5

Fixed Latency On-Chip Interconnect for Hardware Spiking Neural Network Architectures

This work has been published in the Parallel Computing journal (Elsevier), Sep 2013. DOI: 10.1016/j.parco.2013.04.010

Preamble

This chapter details the ring topology interconnect that provides fixed spike transfer latency for EMBRACE hardware SNN architecture.

SNNs employ spike rate and time based coding techniques, where information is primarily encoded as the relative timing between spikes. Shared resources in NoC architectures result in unwanted variation in spike packet transfer latency. This spike latency jitter distorts the SNN information eventually leading to unreliable application behaviour.

This chapter presents an analysis of the spike transfer latency variations in mesh topology, packet switched NoC architectures. The chapter further presents detailed analysis of the impact of this spike latency jitter on the information flowing in SNNs under various SNN configurations.

A novel ring topology interconnect for spike communication between neural tiles and the timestamped spike broadcast flow control scheme is described, which offers fixed spike transfer latency for all the synaptic connections within the ring.

The chapter presents the micro-architecture details of the ring router and its ASIC and FPGA synthesis results.

Spike transfer latency results for the ring interconnect under various SNN spike traffic conditions are presented. The ring interconnect offers fixed spike transfer latency under various spike traffic densities making it suitable as localised spike communication architecture for hardware SNN architectures and ensuring reliable SNN application behaviour.

The scalability of the NoC architecture for EMBRACE hardware SNN is evaluated. The chapter presents a hierarchical NoC design comprising a ring topology interconnect within a packet switched mesh topology network of routers and presents FPGA synthesis results.

The research addresses the challenge of spike communication infrastructure for practical embedded hardware SNN architectures. Future phase of this research employs the presented EMBRACE architecture FPGA prototype using the hierarchical NoC architecture.

Fixed Latency On-Chip Interconnect for Hardware Spiking Neural Network Architectures

Sandeep Pande[a], Fearghal Morgan[a], Gerard Smit[b],
Tom Bruintjes[b], Jochem Rutgers[b], Brian McGinley[a],
Seamus Cawley[a], Jim Harkin[c], Liam McDaid[c]

[a] Bio-Inspired Electronics and Reconfigurable Computing Research Group (BIRC),
National University of Ireland, Galway, Ireland.
[b] Computer Architecture for Embedded Systems,
University of Twente, Enschede, The Netherlands.
[c] Intelligent Systems Research Centre, University of Ulster,
Magee Campus, Derry, Northern Ireland.

Abstract: *Information in a Spiking Neural Network (SNN) is encoded as the relative timing between spikes. Distortion in spike timings can impact the accuracy of SNN operation by modifying the precise firing time of neurons within the SNN. Maintaining the integrity of spike timings is crucial for reliable operation of SNN applications. A packet switched Network on Chip (NoC) infrastructure offers scalable connectivity for spike communication in hardware SNN architectures. However, shared resources in NoC architectures can result in unwanted variation in spike packet transfer latency. This packet latency jitter distorts the timing information conveyed on the synaptic connections in the SNN, resulting in unreliable application behaviour.*

This paper presents a SystemC simulation based analysis of the synaptic information distortion in NoC based hardware SNNs. The paper proposes a fixed spike transfer latency ring topology interconnect for spike communication between neural tiles, using a novel timestamped spike broadcast flow control scheme. The proposed architectural technique is evaluated using spike rates employed in previously reported mesh topology NoC based hardware SNN applications, which exhibited spike latency jitter over NoC paths. Results indicate that the proposed interconnect offers fixed spike transfer latency and eliminates the associated information distortion.

The paper presents the micro-architecture of the proposed ring router. The FPGA validated ring interconnect architecture has been synthesised using 65nm low-power CMOS technology.

Silicon area comparisons for various ring sizes are presented. Scalability of the proposed architecture has been addressed by employing a hierarchical NoC architecture.

5.1 Introduction

Biologically inspired artificial neural network computing techniques mimic key functions of the human brain and have the potential to offer smart and adaptive solutions for complex real world problems [1]. The organic central nervous system includes a dense and complex interconnection of neurons and synapses, where each neuron links to thousands of other neurons through synaptic connections. Computing systems based on Spiking Neural Networks (SNNs) emulate real biological neural networks, conveying information through the communication of short transient pulses (*spikes*) via synaptic connections between neurons. Each neuron maintains a *membrane potential*, which is a function of incoming spikes, associated synaptic weights, current membrane potential, and the membrane potential *leakage coefficient* [2, 3]. A neuron *fires* (emits a spike to all connected synapses/neurons) when its membrane potential exceeds the neuron's firing threshold value. Hardware SNN systems offer the potential for elegant, low-power and scalable methods of embedded computing, with rich non-linear dynamics, ideally suited to applications including classification, estimation, prediction, dynamic control and signal processing [4, 5]. Efficient implementation of hardware SNN architectures for real-time embedded systems is primarily influenced by neuron design, scalable on-chip interconnect architecture and SNN training/learning algorithms [6].

Packet switched Network on Chip (NoC) architectures have recently been proposed as the spike communication infrastructure for hardware SNNs, where data packets containing spike information are routed over a network of routers. The NoC based synaptic connectivity approach provides flexible inter-neuron communication channels, scalable interconnect and connection reconfigurability [7] [8]. The authors have investigated and proposed EMBRACE[2] (illustrated in figure 5.1),as an embedded computing element for the implementation of large scale SNNs [9–11]. A Modular Neural Tile (MNT) architecture has been proposed for reducing the silicon area, by using a combination of fixed and configurable synaptic connections [11]. The proposed MNT comprises a 16:16 neuron fully connected hardware SNN structure. MNTs are integrated in a two dimensional mesh topology

[2] **EM**ulating **B**iologically-inspi**R**ed **A**r**C**hitectures in hardwar**E**

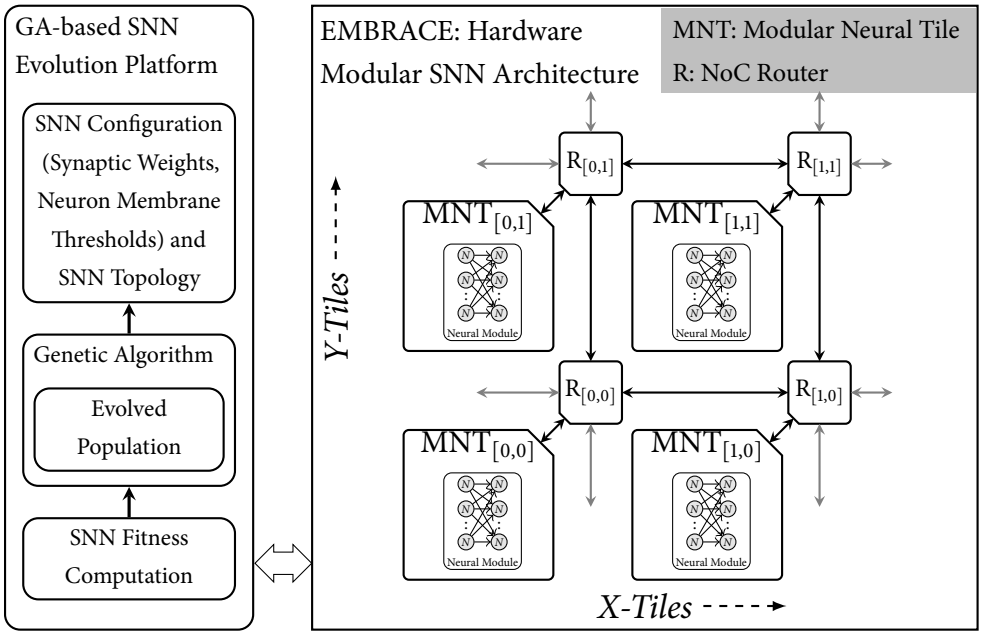


Figure 5.1: EMBRACE Hardware Modular SNN Architecture and GA-based SNN Evolution Platform (The Modular Neural Tile (MNT) comprises a two layered 16:16 fully connected SNN structure, packet encoder/decoder, SNN configuration and topology memory)

packet switched NoC interconnect to form a hardware modular SNN architecture. Spike communication between the MNTs is achieved by routing spike information within spike data packets over the network of routers. The NoC architecture uses XY routing scheme and supports unicast packet flow control, where each spike packet contains destination synapse information for a single spike and is routed independently. A GA-based SNN evolution platform evolves SNN applications by finding a correct set of synaptic weights and neuron threshold potentials. A number of benchmark SNN applications such as XOR data classifier, inverted pendulum controller, Wisconsin breast cancer dataset classifier and robotic controllers have been successfully implemented on the EMBRACE-FPGA prototype [8, 10, 11].

Packet transfer via shared on-chip resources within a NoC based hardware SNN results in different latency values for each NoC packet. The NoC packet latency variation (jitter) alters the arrival timing of spike packets at the destination neurons, distorting the encoded information within the SNN. While a NoC based hardware SNN can be trained to produce correct application behaviour, NoC traffic dependent packet latency jitter can impact the accuracy of the SNN applications and potentially cause an application to fail. Providing guaranteed packet transfer latency is difficult in NoC based SNN architectures due to the large number of

virtual synaptic connections. In past few years, research in NoC architectures have been primarily focused on providing statistical guarantees (or soft bound QoS) for large streaming data traffic [12, 13]. However NoC architectures that can provide fixed packet transfer latency for a large number of virtual connections have not been explored.

This paper simulates (using clock cycle accurate SystemC models) and analyses the synaptic information distortion in NoC based hardware SNNs due to packet transfer latency jitter caused by the NoC communication infrastructure. The paper proposes a fixed spike transfer latency ring topology interconnect for spike communication between neural tiles using a novel timestamped spike broadcast flow control scheme. The proposed architectural technique is evaluated using spike rates employed in previously reported hardware SNN applications. Results indicate that the proposed interconnect offers fixed spike transfer latency and eliminates the associated information distortion. The paper presents the micro-architecture of the proposed ring router. The ring interconnect architecture has been validated on a Xilinx Virtex-6 FPGA and synthesized using 65nm low-power CMOS technology. Silicon area comparisons for various ring sizes are presented. Limitations on the scalability of the proposed ring architecture and selection of the optimal ring size based on spike rate resolution and hardware resources are discussed. Scalability of the proposed architecture has been addressed by employing a hierarchical NoC architecture.

The structure of the paper is as follows: Section 5.2 reviews the NoC architectures suitable for hardware SNNs. Section 5.3 presents simulation based analysis of the effect of latency jitter in NoC based hardware SNN architectures, and discusses the impact of the noise introduced by latency jitter on SNN applications. Section 5.4 presents the spike flow control of the proposed interconnect architecture and the detailed micro-architecture of the proposed ring router. Section 5.5 presents ring interconnect spike transfer latency results and discusses the limitations on the scalability of the proposed ring architecture. The section analyses the optimal ring size for practical spike rate encoding resolution and hardware implementation, and presents hardware resource utilisation results. Section 5.6 concludes the paper.

5.2 NoC Architectures for Hardware SNNs

The biological nervous system exhibits direct interconnection between neurons, where spikes stream through fixed length axons ensuing deterministic spike trans-

fer latency and integrity of synaptic information. For biologically inspired SNN systems, the spike communication infrastructure should provide fixed spike transfer latency to ensure integrity of information. Network on Chip (NoC) architectures have been proposed as a promising solution for multi-core System on Chip (SoC) systems [14] [15]. This section reviews related work on NoC based communication architectures for hardware SNN systems. This review focuses on the NoC architectures which can provide QoS for latency management and are suitable as hardware SNN interconnect.

Information in SNNs is contained within the timing of spikes. Hence the NoC spike communication infrastructure should ideally support fixed latency spike transfer in order to maintain the integrity of the encoded information. Computationally powerful hardware SNN architectures are characterised by thousands of neurons and millions of synapses [16–23]. Hence, the NoC spike communication infrastructure should support millions of virtual synaptic communication channels. Localised multicast communication schemes closely resemble the connectivity patterns typically observed in SNN application topologies and NoC architectures supporting such connectivity patterns are especially suitable for hardware SNN architectures [7]. Information distortion in SNN structures is directly proportional to spike transfer latency jitter. For practical hardware SNN implementations, the spike transfer latency can be longer than the minimum (or best case) latency, though the latency jitter should be as low as possible.

A packet switched NoC architecture can provide suitable communication infrastructure for hardware SNNs, offering scalable, parallel, distributed communication channels with flexible connection reconfigurability [7, 9, 24]. A NoC architecture suitable for hardware SNNs should:

- maintain the integrity of information encoded within spike timings
- support a large number of virtual synaptic communication channels
- support connection topologies that closely resemble neural connectivity patterns

In summary, the NoC architectures for hardware SNN architectures should provide a large number of virtual synaptic communication channels and localised multicast connectivity patterns, while offering low spike transfer latency jitter.

The Æthereal NoC architecture [25], developed specifically for streaming media applications, combines both Guaranteed Services (GS) and Best Effort (BE) services in a single router design. The architecture aims to offer guaranteed ser-

vices such as uncorrupted, lossless, ordered data delivery, guaranteed throughput and bounded latency essential for real-time embedded multi-media applications. ×*pipes* [26], a scalable and high performance NoC architecture, consists of a library of synthesisable soft macros to realise latency insensitive on-chip communication. The router design is deeply pipelined and the NoC architecture is optimised for high throughput and low latency. The QNoC architecture [27] comprises a two-dimensional planar mesh topology NoC with fixed shortest path multi-class wormhole routing and provides statistical guarantees for packet communication. The QNoC architecture defines various service levels such as *Real-Time Service Level* which guarantees bandwidth and latency essential for real-time applications, such as streamed audio and video processing. SoCBUS [28], a circuit switched NoC architecture uses the initial phase of resource reservation using setup and acknowledge packets. The system can provide throughput and latency guarantees after successful channel setup.

Recent research indicates an increasing use of NoC architectures for hardware SNN systems [7, 9, 24]. A generic reconfigurable neural network architecture using a packet switched NoC communication infrastructure is reported in [24]. The architecture consists of a 2-D torus topology NoC, where each router serves four neurons and offers monolithic connectivity. An FPGA-based mesh topology NoC, using XY unicast routing for a clustered neural network, has been proposed in [29]. A unicast communication scheme is a limiting factor in large hardware neural network implementations because of the large number of synaptic connections. A theoretical and comparative analysis of interconnect architectures for hardware SNN systems is reported in [7]. The paper argues the use of packet switched mesh topologies over fat tree, point-to-point and shared bus topologies. Due to the connectivity patterns observed in neural topologies, multicast communication schemes outperform unicast and broadcast. A bufferless ring topology NoC architecture has been proposed, which aims to achieve low packet transfer latency while maintaining a small silicon footprint [30]. Although, the ring topologies offer deterministic hop count between nodes, the inherent connectivity pattern limits its scalability, affecting the network performance [31]. However, ring topology offers a simple and compact architecture suitable for hardware implementation. This paper presents an eight node ring topology interconnect with a fixed spike transfer latency flow control. The proposed eight neural tile ring interconnect integrates as an element within a mesh topology NoC forming a scalable, modular hardware SNN architecture.

5.3 Information Distortion in NoC-based Hardware SNN Architectures

This section analyses the synaptic information distortion in SNN structures due to packet transfer latency jitter caused by the NoC communication infrastructure, using clock cycle accurate simulation models and discusses impact of the noise introduced by latency jitter on SNN applications.

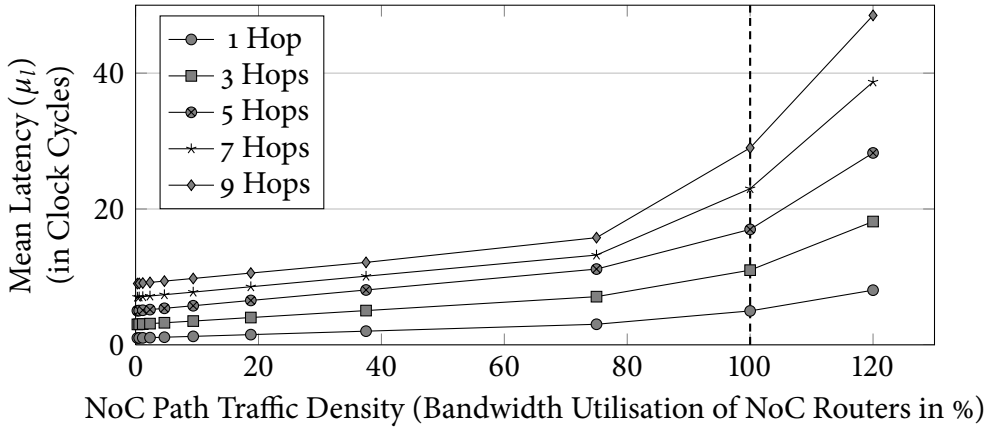
5.3.1 Spiking Neural Network Information Coding

Understanding biological neuronal coding is one of the fundamental issues in neuroscience. Traditional hypothesis suggests that the neural information is encoded in the mean firing rate of neurons, whereas experimental results on response time of humans (for a number of senses) suggest that the neuronal information is encoded in the relative timing between the spikes [2, 3, 32, 33]. Based on these findings, information encoding in artificial SNN systems can be broadly categorised in the following two categories:

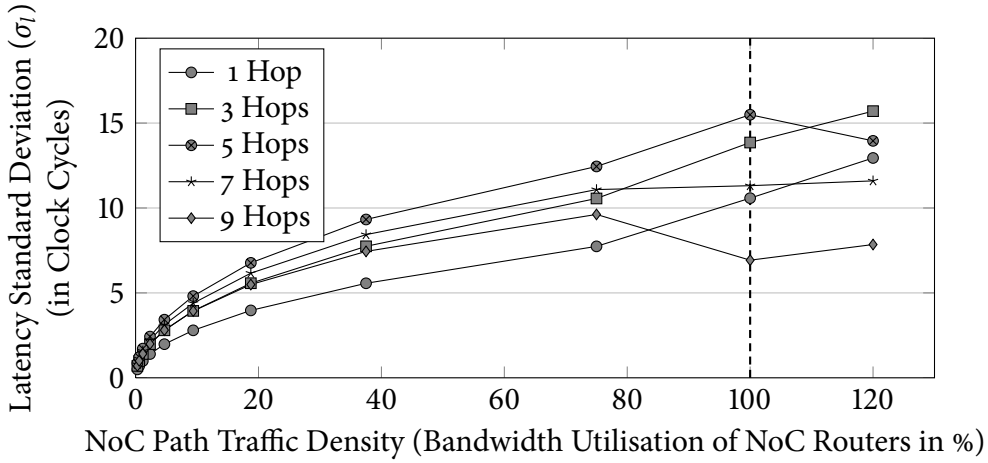
1. **Rate Coding** technique is based on encoding the SNN information as the 'mean firing rate' of neurons. Based on different notions of the mean firing rate, this category is further subdivided in three averaging procedures, namely *Rate as a Spike Count*, *Rate as a Spike Density* and *Rate as a Population Activity* [34]. For practical SNN systems, rate coding encodes spike count within a fixed time interval (or sampling time window as depicted in figure 5.3). The time interval or sampling time of the SNN is often determined by the application response time and stimulus encoding resolution requirements. Spike rate coding technique is a popular choice for embedded applications due to its simplicity [8, 10, 11].

2. **Temporal Coding** technique encodes information based on the 'precise spike timings'. For practical SNN systems, this coding technique can be implemented as *Time to First Spike*, *Spike Phase Encoding* and *Spike Correlation Encoding* [33]. The absence of a fixed sampling time reduces the response time of this coding technique and speeds-up the application, but the implementation complexity limits its use in practical SNN systems.

Variable spike transfer time between neurons distorts the information flowing in SNNs employing either rate coding or temporal coding. This paper presents an



(a) Mean Latency



(b) Latency Standard Deviation

Figure 5.2: Mesh Topology NoC Packet Transfer Latency Variations (Mean and standard deviation of latency is measured in terms of clock cycles)

elaborate analysis of spike latency variation on the information flowing in SNNs employing rate coding. Due to the dependency on exact spike timings, temporal coding is highly susceptible to spike latency variations.

5.3.2 Latency Jitter in The EMBRACE Mesh Topology NoC Architecture

Variation in packet transfer latency resulting from the spike communication interference introduced by the reported EMBRACE NoC architecture has been modelled using EMBRACE-SysC clock cycle accurate SystemC simulation models

[35]. The mean (μ_l) and standard deviation (σ_l) values for packet transfer latency (l) are used to analyse the noise introduced in SNN structures by the latency jitter. NoC paths with multiple hops (through a number of routers) are exercised with a fixed rate spike packet stream. The intermediate NoC routers are loaded with additional spike packet traffic to mimic traffic conditions in hardware SNNs. Figure 5.2 illustrates the packet transfer latency variations (μ_l and σ_l) simulated for the multi-hop NoC paths under various traffic density conditions (i.e. bandwidth utilisation values of intermediate NoC routers). The mean latency plot (figure 5.2a) illustrates slow variations in the packet latency, while the latency standard deviation plot (figure 5.2b) illustrates dispersion of the spike packets received at destination neural tiles due to NoC traffic conditions. After 100% bandwidth utilisation, the intermediate NoC routers are unable to handle the traffic and therefore drop spike packets. The rate of change of mean packet latency (μ_l) increases significantly after 100% NoC traffic density. The latency standard deviation (σ_l) increases and saturates at 100% traffic density conditions in the network.

5.3.3 Impact of Spike Latency Jitter on SNN Applications

The packet switched NoC spike communication infrastructure cannot guarantee a precise spike transfer time for a hardware SNN system comprising a large number of virtual synaptic connections. This makes it very difficult to support temporal coding. The EMBRACE hardware SNN employs spike rate coding scheme, where information is encoded as the number of spikes in a fixed Sampling Time Window (STW = 1ms). This technique offers practical implementation by allowing a sampling time window of a few milliseconds, required by real-life embedded applications. Benchmark SNN control and classifier applications (such as inverted pendulum controller, two-input XOR and Wisconsin breast cancer dataset classifier) have been successfully evolved on EMBRACE FPGA prototype using spike rate coding [8, 10, 11]. The remainder of this section analyses the impact of spike packet latency variations (due to the packet switched NoC) on information processed in SNN structures using simulations. The mean (μ_l) and standard deviation (σ_l) values of spike packet transfer latency (l) shown in figure 5.2 are used for the analysis.

Figure 5.3 illustrates the distortion of information in a Leaky-Integrate-Fire (LIF) neuron [3] due to NoC path spike transfer latency variations. The spike rate encoder generates a fixed Spike Rate (SR). Spikes traverse through the packet

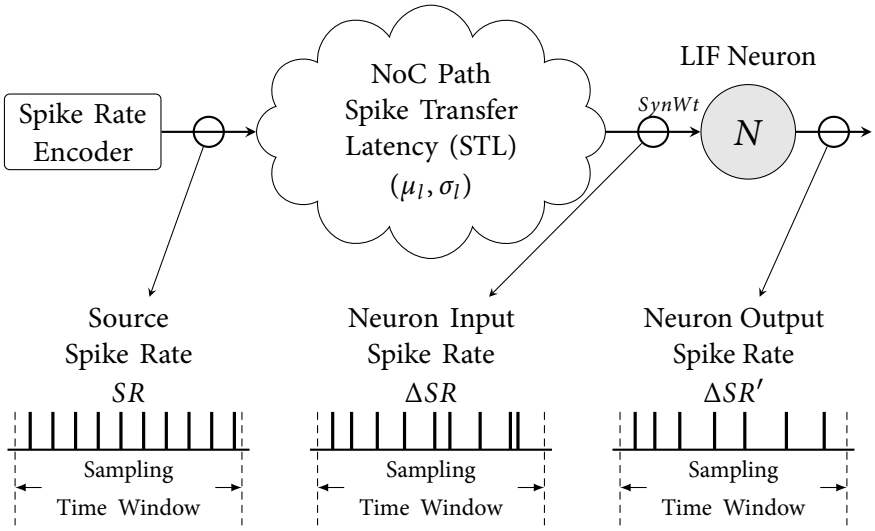


Figure 5.3: Information Distortion in LIF Neuron Due to Spike Transfer Latency Variations

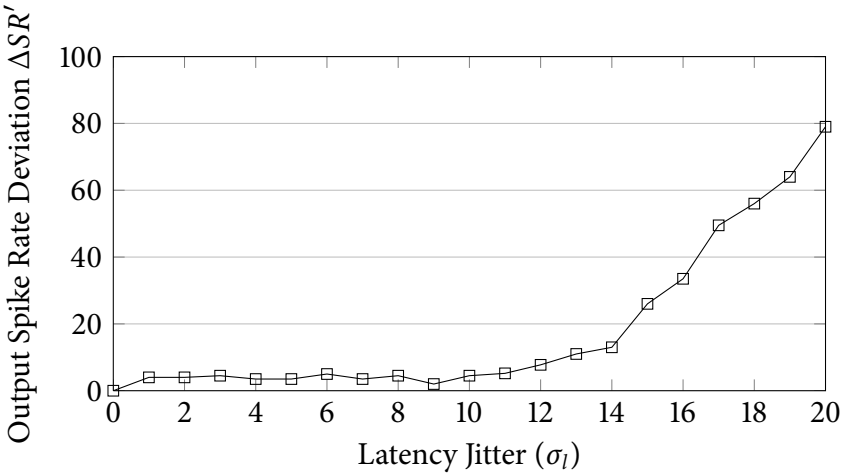


Figure 5.4: LIF Neuron Output Spike Rate Deviation due to Latency Jitter (Spike rate deviation ΔSR is measured as the difference in number of spikes within a STW of 1ms. Constant Synaptic Weight $SynWt$ = 3.5)

switched NoC before reaching the neuron synaptic input. Due to the NoC path Spike packet Transfer Latency (STL) variation (μ_l, σ_l), the source spike rate SR deviates to ΔSR at the input of the LIF neuron. The input spikes update the neuron membrane potential by the associated synaptic weight ($SynWt$). Thus, the altered input spike rate (ΔSR) translates into the neuron membrane potential variation, resulting in additional error in the neuron output spike rate ($\Delta SR'$). Figure 5.4 illustrates the LIF neuron output spike rate deviation due to input NoC path spike

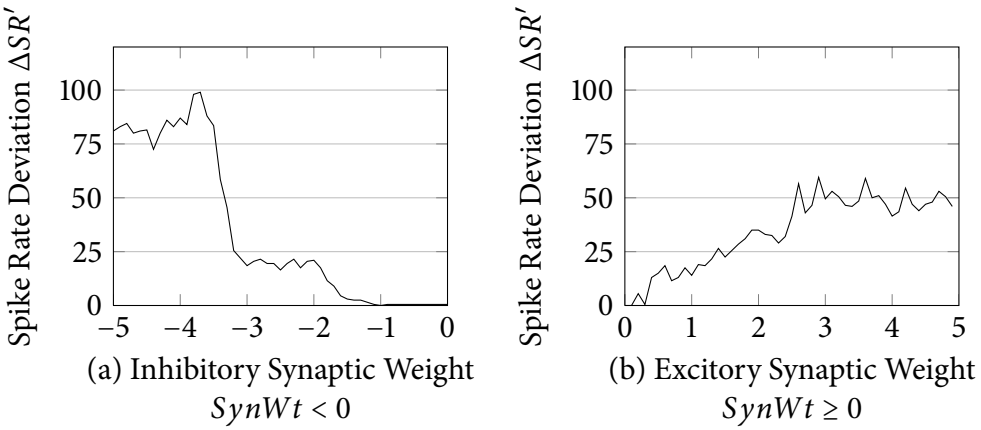
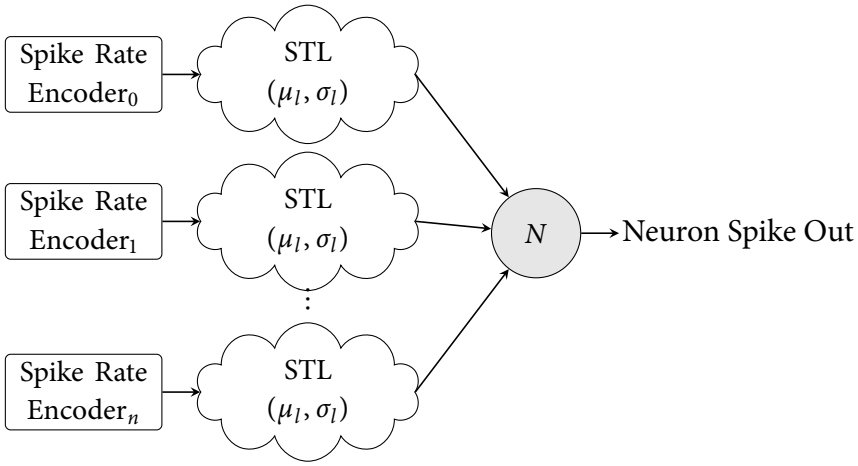


Figure 5.5: Spike Rate Deviation (ΔSR) due to Latency Jitter and Synaptic Weight (Constant Latency Jitter $\mu = 15$)(Spike rate deviation ΔSR is measured as the difference in number of spikes within a STW of 1ms)

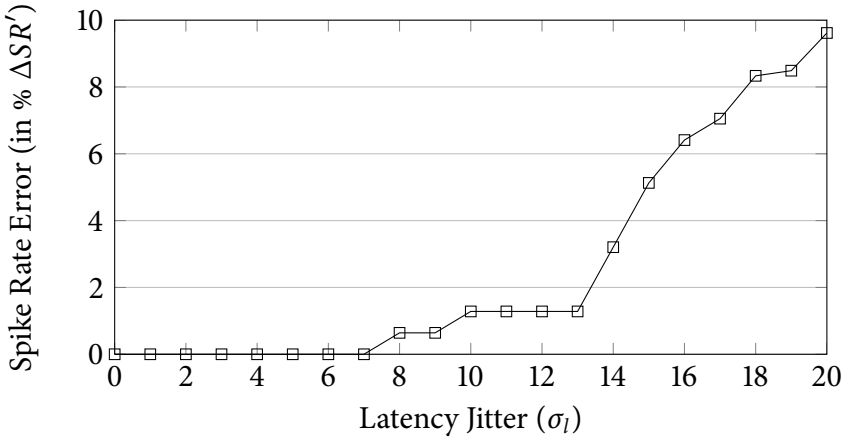
transfer latency jitter. An input spike event on a large synaptic weight alters the neuron membrane potential significantly and thus has a larger effect on the output spike rate changes due to jitter. The experiment highlights effect of NoC packet latency jitter on the SNN information distortion. A similar setup is used for analysis of the impact of latency jitter on SNN information in next set of experiments.

Both excitatory (positive) and inhibitory (negative) synaptic weight values affect the output spike rate of a LIF neuron due to input spike rate variations. Figure 5.5 shows the comparison of LIF neuron output spike rate deviation (ΔSR) due to the input spike rate jitter and the excitatory/inhibitory synaptic weight values. Since the threshold potential of a LIF neuron can only be positive, large inhibitory synaptic weight values dampen the output spike rate significantly, resulting in a large variation in the output spike rate due to input spike rate jitter. Similarly, large excitatory synaptic weight values cause immediate firing of the neuron resulting in significant spike rate deviation.

Each synaptic input experiencing spike rate jitter contributes to output spike rate deviation (proportional to the associated synaptic weight). Multiple synaptic inputs of a LIF neuron have a combined effect on the output spike rate. Figure 5.6a shows the SystemC simulation setup used to measure the output spike rate error of a LIF neuron due to multiple synaptic inputs with associated spike transfer latency jitter. Figure 5.6b shows the output spike rate error of a LIF neuron for a range of spike latency jitter values on multiple synaptic inputs. The output spike rate error of a LIF neuron is directly proportional to latency jitter (σ_l). Low jitter values have



(a) LIF Neuron Multiple Synaptic Inputs Jitter Measurement Setup

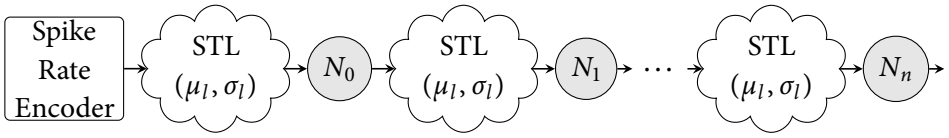


(b) Spike Error Rate for LIF Neuron Multiple Synaptic Inputs (μ_l = 30)

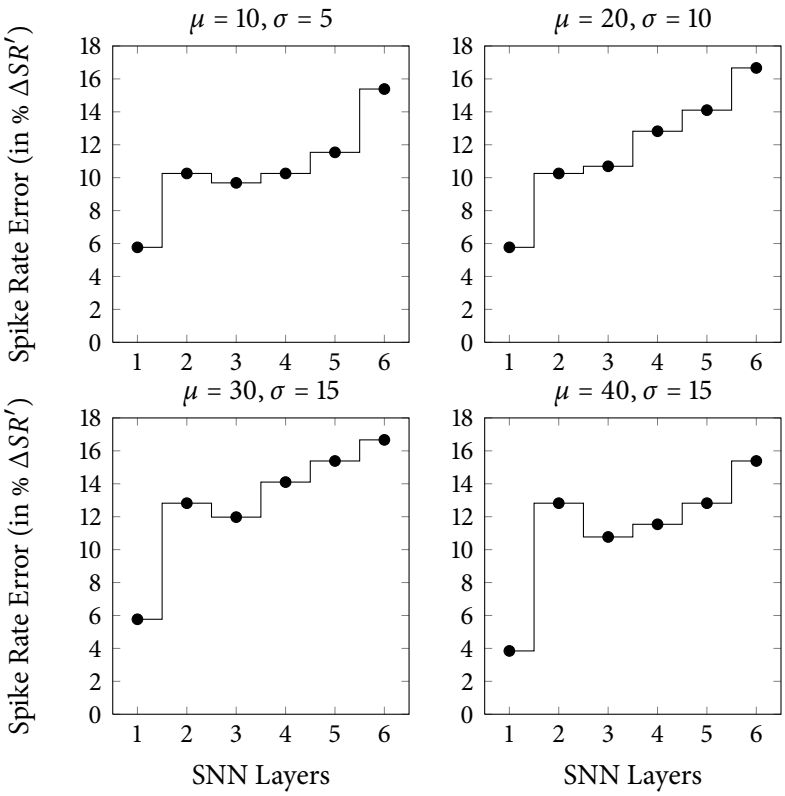
Figure 5.6: LIF Neuron Multiple Synaptic Paths Jitter (Spike rate SR is measured as number of spike in a STW of 1ms)

negligible effect on the output spike rate, since the multiple input spike events cancel the minor variations in membrane potential. As the jitter increases, the changes in membrane potential are significant and have a large impact on output spike rate.

SNN application topologies consists of multiple feed-forward and feed-back (recurrent) neuron layers. Figure 5.7a shows the SystemC simulation setup used to measure the spike rate error due to cascaded layers in a feed-forward SNN topology. Figure 5.7b illustrates the spike rate error in a multi-layer feed-forward SNN topology. Latency jitter has a cascading effect on the spike error rate in multi-layered SNN structures for various jitter values (μ_l, σ_l). As the information



(a) Spike Rate Error Propagation in SNN Layers Measurement Setup



(b) Spike Error Rate in SNN Layers

Figure 5.7: Spike Rate Error Propagation in SNN Layers (Spike rate SR is measured as number of spike in a STW of 1ms)

is processed in each SNN layer, the corresponding spike rate is altered due to spike transfer latency jitter. In some cases, the effect of large spike transfer latency is marginally cancelled by low spike transfer latency in the next layer. This can be seen as a slight reduction in the spike rate error in the third layer in figure 5.7.

Adaptive properties of SNNs are evident from the negligible spike rate error observed at low jitter values. However, large latency jitter observed at high network traffic in the NoC distorts the SNN information considerably. This behaviour, illustrated in figures 5.4, 5.6 and 5.7, shows the traffic dependent behaviour of the SNN structure within the NoC based hardware SNN architectures and can result in erroneous application behaviour. Practical SNN systems use spike rates

that are much lower than the system clock frequency. Assuming that the SNN application spike rate encoding requirement is 1024 spikes in a STW of 1ms, the maximum spike injection rate is one spike in 196 clock cycles for the system clock frequency of 200Mhz [8] [11]. Also, increasing the system clock frequency results in slower spike rates for the given encoding resolution. These properties of the SNN application spike traffic can be used to design a fixed latency spike communication infrastructure.

5.4 Ring Topology Interconnect Architecture

Fixed spike transfer latency interconnect supporting a large number of virtual synaptic connections is the key for accurate and reliable operation of applications executing on packet switched NoC-based hardware SNN architectures. This section presents the proposed ring topology interconnect architecture for hardware SNNs. Spike packet flow control of the proposed NoC architecture, supporting fixed spike transfer latency is discussed. The section also presents the detailed micro-architecture of the proposed ring router.

5.4.1 Fixed Latency Spike Flow-Control

The essential elements for designing a fixed latency, packet switched spike communication interconnect are:

- connection topology offering fixed packet transfer latency between the source-destination nodes [31]
- deterministic packet transmission and reception scheduling in the source and destination routers respectively

Network topologies with appropriate flow control schemes exhibiting these properties, can offer fixed packet transfer latency. Extending this flow control to support broadcasting can efficiently support the bandwidth requirements of localised, high density synaptic connections observed in modular hardware SNNs. This section describes a ring topology with unidirectional, broadcast packet flow control to achieve fixed spike transfer latency. The unidirectional ring topology offers simple and compact router design and allows simplified placement on the silicon die. Other network topologies such as star, mesh and torus (with minimal routing) can also be used with suitable modifications in the proposed fixed latency flow control.

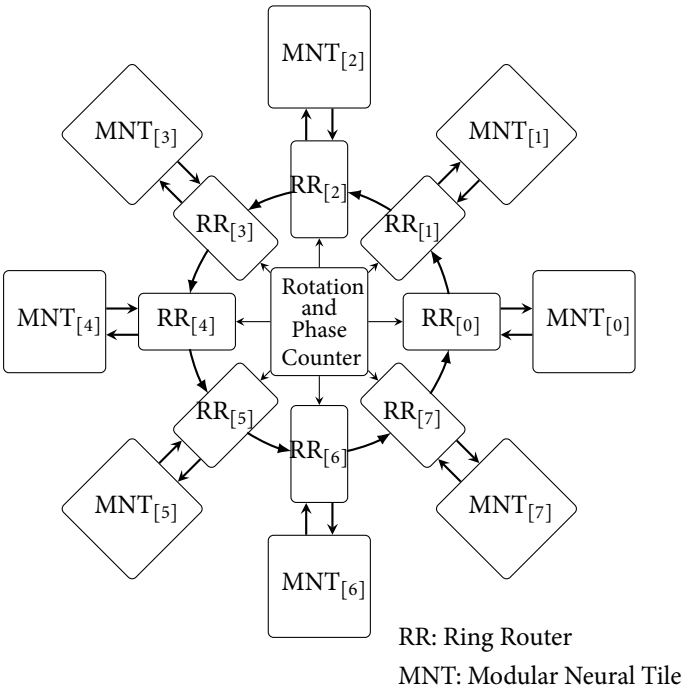


Figure 5.8: Ring Topology Interconnect Architecture (Number of Ring Nodes R = 8)

Rotation Count	Rotation Count	Rotation Count	Rotation Count	Rotation Count
$RCount = 6$	$RCount = 7$	$RCount = 0$	$RCount = 1$	$RCount = 2$
Ring Phase	Ring Phase	Ring Phase	Ring Phase	Ring Phase
$RPhase = FP$	$RPhase = FP$	$RPhase = IP$	$RPhase = FP$	$RPhase = FP$
CLOCK_{n-2}	CLOCK_{n-1}	CLOCK_{n}	CLOCK_{n+1}	CLOCK_{n+2}
$n \neq \mathbb{N}R$	$n \neq \mathbb{N}R$	$n = \mathbb{N}R$	$n \neq \mathbb{N}R$	$n \neq \mathbb{N}R$

Figure 5.9: Rotation Count and Insert (*IP*)/Forward (*FP*) Phases of the Ring (Number of Ring Nodes R = 8)

The flow control of the proposed ring topology interconnect treats each spike event timing separately, maintaining the clock cycle count of the spike event occurrence. The recorded spike event clock cycle count is preserved throughout the spike packet flow control. At the destination ring router, the spike event occurrence clock cycle count is used to send out the spike at the precise clock cycle (with respect to spike event occurrence) offering fixed spike transfer latency and

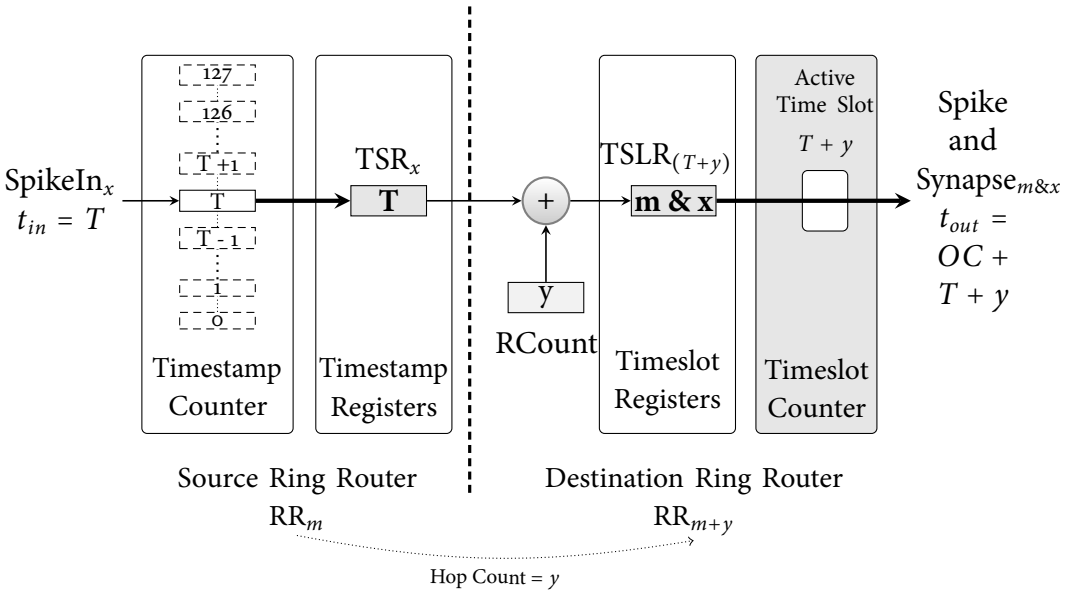


Figure 5.10: Fixed Latency Spike Packet Flow Control

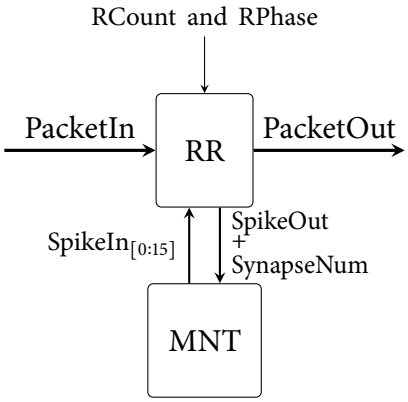
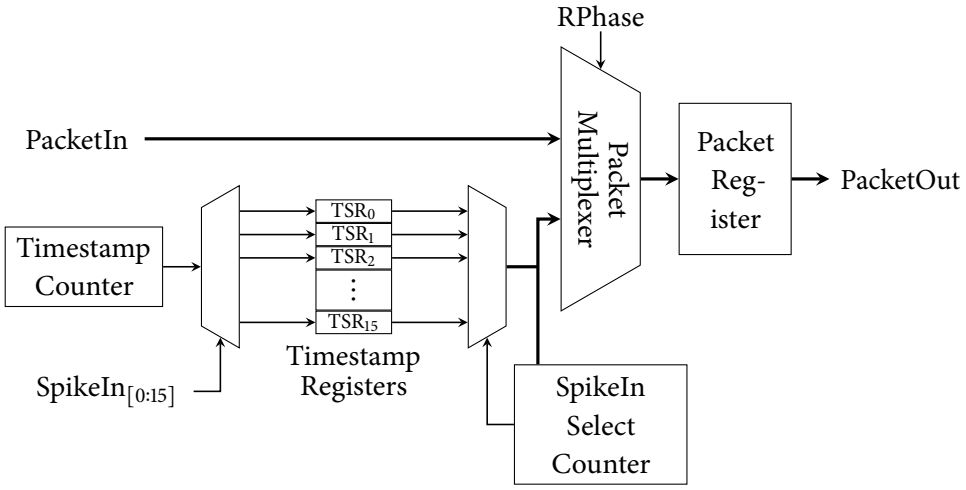


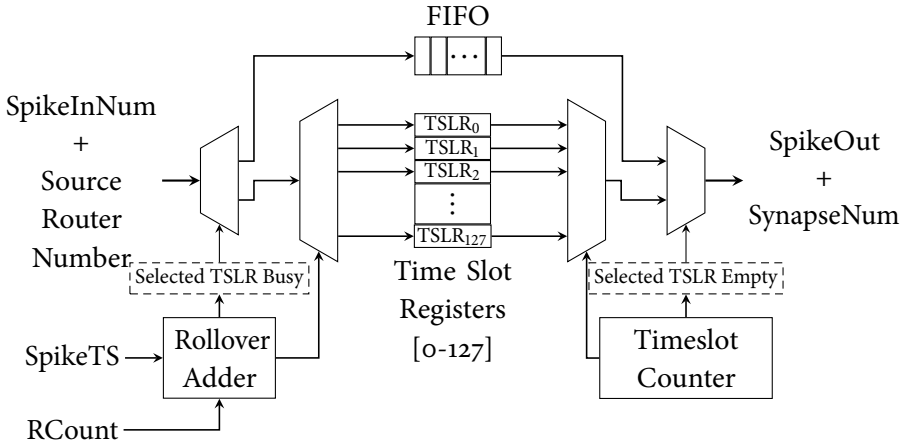
Figure 5.11: Ring Router and MNT Interface

maintaining the integrity of the SNN information flow.

Figure 5.8 illustrates the proposed ring topology interconnect (in an eight node configuration) with unidirectional packet flow control for hardware modular SNN architectures. The proposed interconnect comprises routers connected in the unidirectional ring topology, where each router receives a spike packet from the previous router and sends a spike packet to the next router. The MNTs interface with ring routers for spike communication. Each router buffers the spike events generated by the attached MNT and encodes them into spike packets. Generated spike packets are processed and forwarded to the next router in a single clock cycle. Full rotation of the spike packet on the ring ensures broadcast packet flow control.



(a) Packet Sender



(b) Packet Receiver

Figure 5.12: Ring Router Micro-Architecture Organisation

B_{11}	B_{10}-B_4	B_3-B_0
PacketVal Packet Valid (1-bit)	***SpikeTS*** Spike Timestamp (7-bits)	***SpikeInNum*** Spike Input Number (4-bits)

Figure 5.13: Ring Interconnect Packet Structure

The full *Rotation Cycle (RC)* of the ring (i.e. the number of clock cycles required for a generated packet to traverse all the ring routers and arrive at the source router) is equal to the number of nodes in the ring ($RC = R$).

The ring interconnect operates in two phases; namely the *Insert Phase (IP)* and *Forward Phase (FP)*. Figure 5.9 illustrates the sequencing of insert and forward

phases of the ring, based on the clock cycle count. The ring operates in the *insert phase* for every natural number multiple of the number of ring nodes ($n = \mathbb{N}R$). For all remaining clock cycles ($n \neq \mathbb{N}R$), the ring operates in the *forward phase*. Thus, the rotation cycle consists of one *insert phase* and $R-1$ *forward phases*. Phase synchronisation of the ring is maintained by the global *Rotation and Phase Counter*. The phase counter maintains the clock cycle count and controls the ring operation phase. For both phases, the packet received by the router from the previous router is processed to retrieve the spike information. During an *insert phase*, the router outputs a new spike packet to the next router in the ring. During a *forward phase*, the received packet is forwarded to the next router in the ring.

Figure 5.10 illustrates the spike flow control implemented on the proposed ring topology interconnect. The ring router uses a round-robin scheduling policy for transmitting spike packets corresponding to spike inputs ($\text{SpikeIn}_{[0:15]}$). A new spike packet is generated during each *insert phase*. To serve the 16 spike inputs (from the attached MNT), the ring router requires $16RC$ clock cycles (i.e. 128 clock cycles for an eight node ring configuration). This is defined as the full *Operating Cycle (OC)* of the ring. Each router maintains a timestamp counter which counts the clock cycles within an operating cycle. A valid spike event on a spike input (SpikeIn_x) is timestamped and buffered in the corresponding Timestamp Register (TSR_x). During an *insert phase*, the TSR_x value (selected by the round-robin sequence) is encoded in a new spike packet for the spike input x, and is forwarded to the next router in the ring.

Each ring router decodes the input spike packets and buffers the spike events for processing at precise clock cycle based on the timestamp value. On receipt, the timestamp value in the spike packet is incremented by the rotation count to account for the packet transfer latency between the corresponding source-destination router pair. The spike event is then buffered in the Time Slot Registers (TSLR) on the resulting timestamp value. A timeslot counter counting the clock cycles within the operating cycle is used to select and process spike events from the TSLRs. The TSLRs contain 128 registers for the eight node ring configuration ($16RC$ registers), each corresponding to a time slot in the ring operating cycle. The spike events buffered in TSLRs are delivered during the next operating cycle. Thus the spike generated at $t_{in} = T$ in the source router, is delivered at $t_{out} = OC + T + y$. Thus the flow control offers fixed spike transfer latency of one operating cycle and the source-destination hop count ($OC + y$) for input spikes rates of $ISI \geq OC$ (where ISI is inter spike interval in number of clock cycles.

5.4.2 Ring Router Micro-Architecture Organisation

The ring routers facilitate spike communication between the MNTs in the ring. Figure 5.11 illustrates the ring router and MNT interface. The ring router packet sender subsystem buffers the spikes generated by the attached MNT and inserts spike packets on the ring. The ring router packet receiver subsystem decodes and buffers the received spike information and sends the spikes to the attached MNT. Figure 5.12 illustrates the micro-architecture organisation of the proposed ring router RTL design, comprising (a) *Packet Sender* and (b) *Packet Receiver* subsystems.

a) **Packet Sender:** The phase input (RPhase), determines the operating phase of the ring router. During a forward phase, the packet multiplexer selects the input packet from the previous router. During an insert phase, a new spike packet generated by the router is selected. The selected spike packet is buffered in the packet register and passed to the next router in the ring. On receiving a valid spike on the spike input (SpikeIn_x), the current timestamp counter value is stored in the corresponding Timestamp Register (TSR_x) along with a valid bit. The packet sender uses a round-robin policy for transmitting the output spike packet for spike inputs ($\text{SpikeIn}_{[0:15]}$). The SpikeIn Select Counter is incremented during each insert phase and a new spike packet comprising the Packet Valid, Spike Timestamp and Source Spike Number is generated. Figure 5.13 illustrates the packet structure for the proposed ring interconnect.

b) **Packet Receiver:** The packet receiver subsystem decodes and buffers valid input spike packets (*PacketVal* = '1') and forwards the output spike pulse along with synapse number to the MNT. The Spike Timestamp (*SpikeTS*), the Rotation Count (*RCount*) and the router number (RR_n) present in each router are used to calculate the resulting timeslot for the output spike pulse. The *rollover adder* increments the timestamp value by the rotation count value. If the result is larger than the maximum number of timeslots in the operating cycle, the sum is rolled over to the next operating cycle. The rollover adder output selects the Timeslot Register (TSLR) for storing the SpikeIn Number (*SpikeInNum*) from the input spike packet, source router number (calculated from destination router number and RCount), and asserts the *Busy* bit. If the selected TSLR is already occupied (*Busy* = '1'), the *SpikeInNum* and source router number values are pushed into the FIFO.

The Timeslot Counter continuously counts clock cycles within the operating cycle and keeps track of the current time slot. The timeslot count is used to select the TSLR and retrieve the *SpikeInNum* and source router number. If the selected TSLR is empty (*Busy* = 'o'), the next available entry from the FIFO is retrieved. The *SynapseNum* is computed by concatenating the source router number and *SpikeInNum*, and is sent to the attached MNT along with a generated spike pulse (*SpikeOut*).

Multiple spike events that are destined for the same TSLR are buffered in the FIFO for delivery during vacant time slots, which are indicated by an empty TSLR. This untimed delivery of spikes results in variable spike transfer latency. As the spike rates for practical SNN applications is considerably lower than the worst case spiking rates, the probability of the TSLR collisions is low.

5.5 Results and Discussion

The proposed ring topology interconnect has been modelled as a clock-cycle accurate model in SystemC. Performance of the ring architecture has been measured using EMBRACE-SysC design exploration framework for hardware SNN architectures [35]. This section presents spike transfer latency and hardware implementation results for the proposed ring interconnect. Scalability of the proposed ring interconnect has been addressed by employing a hierarchical NoC architecture.

5.5.1 Ring Topology Interconnect Spike Transfer Latency Performance

Performance of the proposed ring topology interconnect has been evaluated using worst case spike traffic observed in SNN applications [8] [11]. The measurement setup uses spike rate encoders (instead of MNTs as shown in figure 5.8) feeding constant spike streams to each router in the ring. Each spike rate encoder has 16 spike outputs connected to the spike inputs ($\text{SpikeIn}_{[0:15]}$) of the corresponding ring router. The frequency of spikes on each of the spike inputs (SpikeIn_x) is $\frac{1}{ISI}$; where ISI is *Inter Spike Interval* in clock cycles. Each ring router can receive a spike every $ISI_{min}/16$ clock cycles from the attached MNT, resulting in a maximum spike rate of $16F/ISI_{min} = 25 \times 10^6$ spikes per second (where system clock frequency F = 200Mhz and ISI_{min} = 128 clock cycles). Overall the ring generates, transfers and consumes $(16F\mathbb{N}R)/ISI_{min} = 200 \times 10^6$ spikes per second. The proposed ring

Table 5.1: Mean Spike Transfer Latency (μ_l) for Multiple Hop Spike Traffic on the Proposed Ring Topology Interconnect

Inter Spike Interval	1-Hop	2-Hop	3-Hop	4-Hop	5-Hop	6-Hop	7-Hop	Full Ring Rotation 8-Hop
2048	129	130	131	132	133	134	135	128
1024	129	130	131	132	133	134	135	128
512	129	130	131	132	133	134	135	128
256	129	130	131	132	133	134	135	128
128	129	130	131	132	133	134	135	128
96	204.87	205.87	206.87	207.87	208.87	209.87	210.87	203.87
64	269.80	270.80	271.80	272.80	273.80	274.80	275.80	268.80
32	327.45	328.45	329.45	330.45	331.45	332.45	333.45	326.45

Inter Spike Interval (ISI) is specified in clock cycles between consecutive spikes and the spike transfer latency is measured in clock cycles.

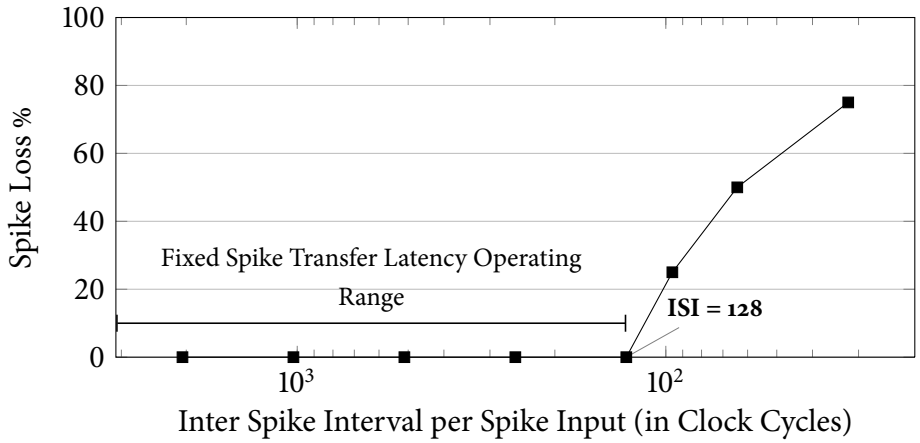


Figure 5.14: Ring Topology Interconnect Spike Loss Rate

interconnect supports high spike density communication, with each ring router able to deliver a spike on every clock cycle. The previously reported EMBRACE mesh topology NoC employing unicast flow control scheme cannot offer such performance.

Table 5.1 illustrates the mean spike transfer latency (μ_l) for multi-hop spike traffic on the proposed ring topology interconnect in an eight node configuration. The interconnect offers fixed latency for $ISI \geq OC$. As the spike rate reaches its upper limit ($ISI < OC$), the generated spike events are overwritten on TSRs (in the packet sender subsystem within the source ring router), resulting in loss of spikes.

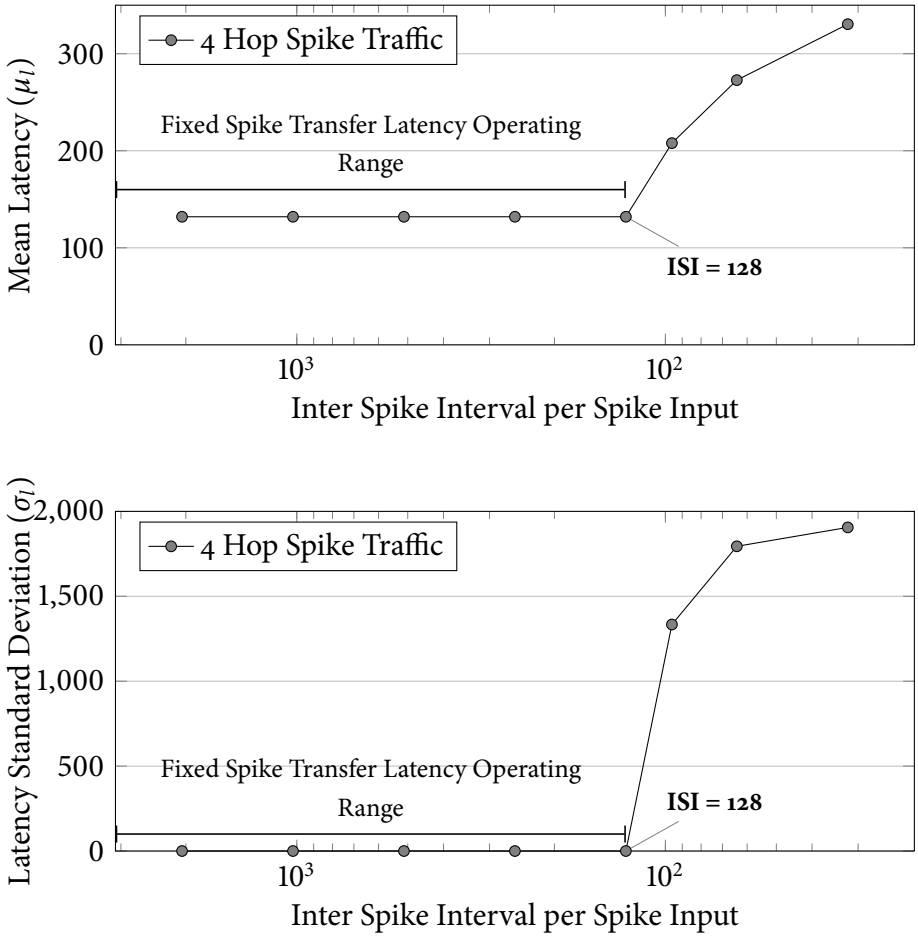


Figure 5.15: Ring Topology Interconnect Spike Transfer Latency Variations (Inter spike interval is specified in clock cycles between consecutive spikes. Mean and standard deviation of latency is measured in terms of clock cycles)

Table 5.2: Neuron Density, Spike Injection Rate and Maximum Number of Spikes in a Sampling Time Window (STW) of 1ms, for Various Ring Sizes

Ring Size	Neurons	Synapses	Minimum ISI (in Clock Cycles)	Maximum Spikes (in a STW)
4	128	5K	64	3125
8	**256**	**18K**	**128**	**1562**
16	512	68K	256	781
32	1024	264K	512	390
64	2048	1040K	1024	195
128	4096	4128K	2048	97
256	8192	16448K	4096	48

Table 5.3: Ring Topology Interconnect Synthesis Results (Clock Frequency: 200Mhz)

	ASIC Synthesis (65nm Low-Power CMOS)		FPGA Synthesis (Xilinx Virtex-6 FPGA)		
Ring Size	Ring NoC Area (in μm^2)	Ring Router Area (in μm^2)	Ring NoC Slices	Ring Router Slices	Ring Router Slice Registers
4	40580	10136	1060	265	581
8	**157700**	**19707**	**5202**	**650**	**1175**
16	642555	40145	24010	1212	2486
32	2723136	85099	NA	2970	5322

Figure 5.14 illustrates the increase in spike loss due to the increase in input spike rate. Also, as the spike rate exceeds the upper limit ($ISI < OC$), the probability of received spike events destined for the same time slot increases, resulting in collisions on the destination TSLRs. The packet receiver subsystem within the destination ring router buffers these spike events (causing collisions on the TSLR) in a FIFO. Due to the unavailability of time slots and untimed delivery of spike events from the FIFO, the spike transfer latency variations increase considerably. Figure 5.15 illustrates the spike transfer latency variations (μ_l and σ_l) measured for the 4-hop spike traffic on the proposed ring topology interconnect. For $ISI < 128$ (i.e. the operating cycle of the eight node ring), the spike transfer latency mean and standard deviation (μ_l and σ_l) increase significantly.

The operating cycle ($OC = RSpikeIn_n$) of the ring imposes a limit on the maximum spike rate supported (for spike inputs), for the fixed spike transfer latency operation of the proposed ring interconnect. The maximum number of spikes for a sampling time window[3] of 1ms for various ring sizes is depicted in table 5.2 (Highlighted values represent the selected ring size of eight nodes). A large number of spikes within a STW allows higher resolution for spike rate encoding in a SNN application setup (figure 5.3).

5.5.2 Hardware Implementation

The proposed ring interconnect has been implemented as a configurable design in VHDL. A number of ring configurations have been synthesised to Xilinx Virtex

[3]Choice of sampling window time is primarily influenced by the response time requirements of the application.

6 FPGA (to validate the RTL design) and 65nm low-power CMOS technology from STMicroelectronics for silicon area results. Table 5.3 shows the hardware synthesis results for the proposed ring topology interconnect. ASIC synthesis has been performed using Synopsys Design Compiler 2009-sp5 and FPGA synthesis using Xilinx XST 14.4.

The ring router architecture is primarily organised around timestamp and time slot registers, which are integral parts of the packet sender and receiver subsystems. The number of timestamp registers is determined by the number of spike inputs, and the register width is defined by the operating cycle. The number of time slot registers is determined by the operating cycle, and the register width is defined by the operating cycle and the number of synapses in the ring. As the ring dimension scales, the number of synapses increases quadratically and similar change is observed in the silicon footprint of the ring router and the complete interconnect. In FPGA synthesis, the TSR and TSLR registers are mapped to slice registers resulting in a proportional increase in slice register count and associated control logic mapped onto the slices.

The previously reported EMBRACE mesh topology NoC uses unicast packet flow control for the synaptic connections between MNTs. The architecture employed SNN topology memory for storing the spike packet information for the large number of synaptic connections. The topology memory within each MNT occupies 50% of the silicon area [11]. The proposed ring interconnect employs spike broadcast flow control, where the weight values for unwanted synaptic connections are set to zero in the neuron configuration. This results in a significant reduction in the silicon footprint of the architecture.

The rotation and phase counter is replicated in each ring router to facilitate scalable hardware implementation of the interconnect. For simplicity, the rotation and phase counter is shown as a central unit in figure 5.8.

5.5.3 Scalability for Large SNN Arrays

Practical SNN systems are characterised by a large numbers of neurons and high interconnectivity through inter-neuron synaptic connections. Each of the SNN execution architectures presented in [16–23] aims for thousands of neurons and millions of synapses, which is essential for a powerful neural computing platform. Hence, scalability is an important aspect of interconnect design for hardware SNN architectures.

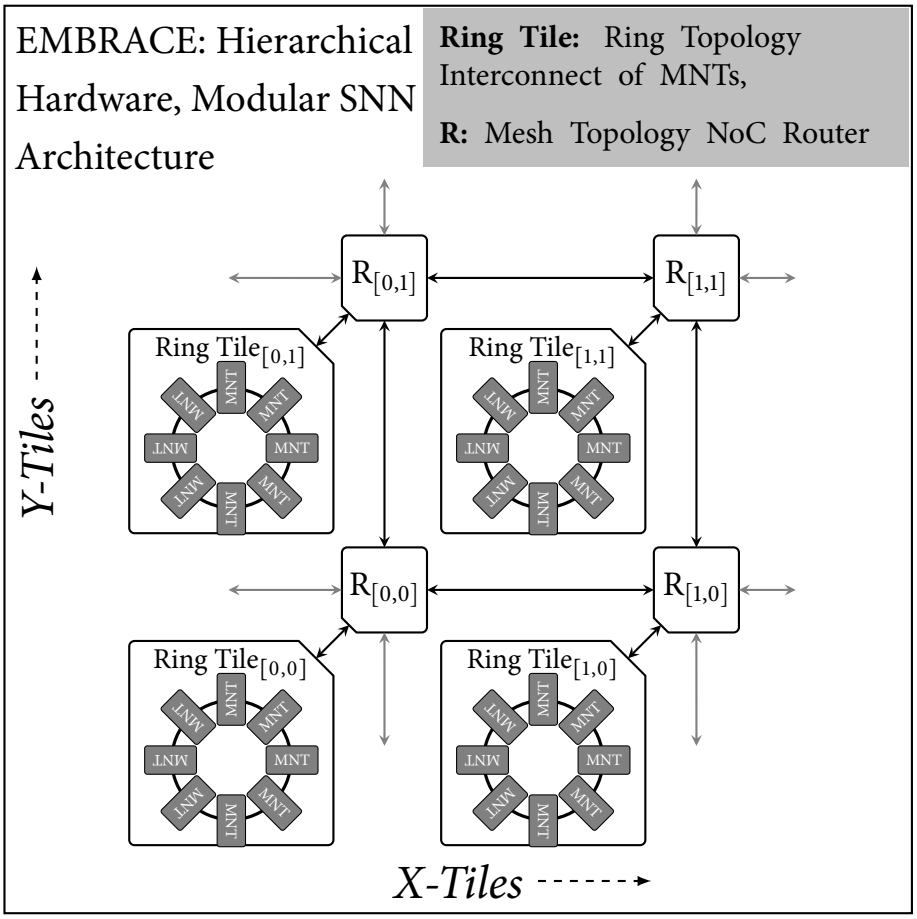


Figure 5.16: Hierarchical Modular SNN Architecture Comprising Ring Topology Interconnects within Mesh Topology NoC

The spike flow control of the proposed ring topology interconnect relies on the ring size and number of spike inputs. Scaling the ring interconnect results in a quadratic increase in the synaptic density, but also increases the spike transfer latency. Scaling also adversely affects the spike rate encoding resolution (Table 5.2). Based on the supported synaptic density, spike rate resolution (which is suitable for practical SNN applications) and silicon area requirements, an optimal sized eight node ring interconnect with MNTs has been designed as the *Ring Tile* (Table 5.2).

Research shows that real world biological networks exhibit high communication locality [36, 37]. The modular organisation in the human brain has been the motivation for the MNN design strategy, which suggests partitioning of application tasks into a number of subtasks [38–40]. Based on these factors, the proposed interconnect architecture can be scaled by replicating individual ring

Table 5.4: Hierarchical Modular SNN Architecture FPGA Synthesis Results (Xilinx Virtex-6)

RTL Entity	Slices	Slice Registers	DSP48E1	BRAM
Neuron	8	16	1	0
Modular Neural Tile	313	1705	32	0
Interface Tile	490	656	0	16
Ring Router	617	1175	0	1
Ring Tile	7693	22004	224	24
Mesh NoC Router	133	35	0	0

tiles connected in a mesh topology NoC architecture. The individual ring tile is made up of the ring topology interconnect of seven MNTs and one interface tile. The mesh topology NoC router bandwidth utilisation drops considerably as most of the localised, high density spike traffic is contained within the ring. This results in low spike packet latency jitter for inter ring tile spike communication.

Figure 5.16 illustrates the architecture of the proposed hierarchical modular hardware SNN. The proposed hierarchical, modular hardware SNN in a two ring tile configuration comprises 448 neurons, 32K synapses and has been synthesised on Xilinx Virtex-6 XC6VLX240T FPGA device and the FPGA synthesis results are presented in table 5.4.

5.6 Conclusions

Spike transfer latency jitter in previously reported EMBRACE mesh topology NoC architecture for hardware SNNs has been quantified and presented as mean and standard deviation of latency over synaptic paths. The mean latency changes represent slow variations in latency, whereas the standard deviation represents latency jitter. The latency jitter in XY routing based mesh topology NoC increases with the network traffic and can affect the SNN application behaviour.

This paper analysed the effects of NoC latency jitter on the information in SNN structures. The LIF neuron output spike rate is dependent on the input spike rate, synaptic weight and associated latency jitter on the input synaptic path. The analysis shows that the LIF neuron output spike rate error is directly proportional to input spike latency jitter. Large inhibitory synaptic weight values have a bigger impact on the LIF neuron output spike rate, due to the associated jitter as compared to excitatory synaptic weight values. The synaptic weight integration within an

LIF neuron cancels the minor variations (low jitter) on the multiple synaptic inputs, but the high jitter results in a steep increase in the neuron output spike rate error. The SNN information error increases due to the latency jitter in NoC, as the information is processed in cascaded SNN layers connected via NoC.

A ring topology interconnect offering a fixed spike transfer latency flow control has been presented. The proposed ring interconnect supports high spike density communication. The spike transfer latency is proportional to the ring size and number of spike inputs to each ring router. The proposed eight node ring configuration offers a fixed latency of 128-135 clock cycles between the ring nodes for $ISI \geq 128$ clock cycles. The fixed latency offered by the proposed ring interconnect makes the architecture suitable for employing temporal coding in SNN applications. Scaling the ring interconnect increases the number of synaptic connections quadratically, but also results in proportional increase in spike event buffer size. The ring sizes with more than eight nodes have been found to be unsuitable for applications with response time requirement of few milliseconds, due to spike rate coding constraints and limited resolution. A hierarchical, modular hardware SNN architecture comprising ring tiles integrated in a mesh topology NoC has been presented as a scalable SNN computing platform.

Future work includes validation of the proposed hierarchical hardware modular SNN architecture using real-life modular SNN applications. A modular robotic navigational controller application comprising multiple application subtasks running on MNTs within the ring topology interconnect is in development. The suitability of the proposed architecture as a hardware platform for real-life cognitive embedded applications will be analysed by measuring the application accuracy and reliability.

Acknowledgment

This research is supported by the International Centre for Graduate Education in Micro and Nano-Engineering (ICGEE), Irish Research Council for Science, Engineering and Technology (IRCSET) and Xilinx University Programme.

References

[1] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999, vol. 13.

[2] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659 – 1671, 1997.

[3] W. Gerstner and W. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, Aug. 2002.

[4] S. Bohte and J. Kok, "Applications of spiking neural networks," *Information Processing Letters*, vol. 95, no. 6, pp. 519–520, 2005.

[5] M. Pearson, A. Pipe, B. Mitchinson, K. Gurney, C. Melhuish, I. Gilhespy, and M. Nibouche, "Implementing spiking neural networks for real-time signal-processing and control applications: A model-validated fpga approach," *Neural Networks, IEEE Transactions on*, vol. 18, no. 5, pp. 1472 –1487, sept 2007.

[6] L. Maguire, T. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for large-scale implementations of spiking neural networks on FPGAs," *Neurocomputing*, vol. 71, no. 1-3, pp. 13 – 29, Dec. 2007.

[7] D. Vainbrand and R. Ginosar, "Scalable network-on-chip architecture for configurable neural networks," *Microprocessors and Microsystems*, vol. 35, no. 2, pp. 152 – 166, 2011.

[8] F. Morgan, S. Cawley, B. McGinley, S. Pande, L. McDaid, B. Glackin, J. Maher, and J. Harkin, "Exploring the evolution of noc-based spiking neural networks on fpgas," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, Dec. 2009, pp. 300 –303.

[9] J. Harkin, F. Morgan, L. McDaid, S. Hall, B. McGinley, and S. Cawley, "A reconfigurable and biologically inspired paradigm for computation using network-on-chip and spiking neural networks," *Int. J. Reconfig. Comput.*, vol. 2009, pp. 2:1–2:13, Jan. 2009.

[10] 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*, vol. 12, pp. 257–280, 2011.

[11] S. Pande, F. Morgan, S. Cawley, T. Bruintjes, G. Smit, B. McGinley, S. Carrillo, J. Harkin, and L. McDaid, "Modular neural tile architecture for compact embedded hardware spiking neural network," *Neural Processing Letters*, vol. 38, pp. 131–153, Oct. 2013.

[12] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surv.*, vol. 38, no. 1, 2006.

[13] E. Salminen, A. Kulmala, and T. Hamalainen, "On network-on-chip comparison," in *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, aug. 2007, pp. 503 –510.

[14] L. Benini and G. De Micheli, "Powering networks on chips," in *System Synthesis, 2001. Proceedings. The 14th International Symposium on*, 2001, pp. 33 – 38.

[15] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70 –78, jan 2002.

[16] M. Ehrlich, C. Mayr, H. Eisenreich, S. Henker, A. Srowig, A. Grubl, J. Schemmel, and R. Schuffny, "Wafer-scale VLSI implementations of pulse coupled neural networks," in *Proceedings of the International Conference on Sensors, Circuits and Instrumentation Systems*, 2007.

[17] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, june 2008, pp. 431 –438.

[18] S. Furber and A. Brown, "Biologically-inspired massively-parallel architectures - computing beyond a million processors," in *Application of Concurrency to System Design, 2009. ACSD '09. Ninth International Conference on*, july 2009, pp. 3 –12.

[19] E. Ros, E. Ortigosa, R. Agis, R. Carrillo, and M. Arnold, "Real-time computing platform for spiking neurons (rt-spike)," *Neural Networks, IEEE Transactions on*, vol. 17, no. 4, pp. 1050 – 1063, july 2006.

[20] A. Upegui and E. PeÃśa-Reyes, CC.and Sanchez, "An fpga platform for online topology exploration of spiking neural networks," *Microprocessors and Microsystems*, vol. 29, no. 5, pp. 211 – 223, 2005.

[21] M. Pearson, A. Pipe, B. Mitchinson, K. Gurney, C. Melhuish, I. Gilhespy, and M. Nibouche, "Implementing spiking neural networks for real-time signal-processing and control applications: A model-validated fpga approach," *Neural Networks, IEEE Transactions on*, vol. 18, no. 5, pp. 1472 –1487, sept. 2007.

[22] B. Glackin, T. McGinnity, L. Maguire, Q. Wu, and A. Belatreche, "A novel approach for the implementation of large scale spiking neural networks on fpga hardware," in *Computational Intelligence and Bioinspired Systems*, ser. Lecture Notes in Computer Science, J. Cabestany, A. Prieto, and F. Sandoval, Eds. Springer Berlin / Heidelberg, 2005, vol. 3512, pp. 1–24.

[23] R. Vogelstein, U. Mallik, J. Vogelstein, and G. Cauwenberghs, "Dynamically reconfigurable silicon array of spiking neurons with conductance-based synapses," *Neural Networks, IEEE Transactions on*, vol. 18, no. 1, pp. 253 –265, jan. 2007.

[24] T. Theocharides, G. Link, N. Vijaykrishnan, M. Invin, and V. Srikantam, "A generic reconfigurable neural network architecture as a network on chip," in *SOC Conference, 2004. Proceedings. IEEE International*, sept 2004, pp. 191 – 194.

[25] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal Network on Chip: concepts, architectures, and implementations," *Design Test of Computers, IEEE*, vol. 22, no. 5, pp. 414 – 421, sept-oct 2005.

[26] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "Xpipes: a latency insensitive parameterized network-on-chip architecture for multiprocessor socs," in *Computer Design, 2003. Proceedings. 21st International Conference on*, oct. 2003, pp. 536 – 539.

[27] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Qnoc: Qos architecture and design process for network on chip," *Journal of Systems Architecture*, vol. 50, no. 2-3, pp. 105 – 128, 2004.

[28] D. Wiklund and D. Liu, "Socbus: switched network on chip for hard real time embedded systems," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, april 2003, p. 8 pp.

[29] R. Emery, A. Yakovlev, and G. Chester, "Connection-centric network for spiking neural networks," in *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '09. IEEE Computer Society, 2009, pp. 144–152.

[30] J. Kim and H. Kim, "Router microarchitecture and scalability of ring topology in on-chip networks," in *Proceedings of the 2nd International Workshop on Network on Chip Architectures*, ser. NoCArc '09. ACM, 2009, pp. 5–10.

[31] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[32] W. Gerstner, A. Kreiter, H. Markram, and A. Herz, "Neural codes: Firing rates and beyond," *Proceedings of the National Academy of Sciences*, vol. 94, no. 24, pp. 12 740–12 741, 1997.

[33] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *Nature*, vol. 381, pp. 520 – 522, 1996.

[34] F. Rieke, D. Warland, R. Deruytervansteveninck, and W. Bialek, *Spikes: Exploring the Neural Code (Computational Neuroscience)*. The MIT Press, Jun. 1999.

[35] S. Pande, F. Morgan, S. Cawley, B. McGinley, S. Carrillo, J. Harkin, and L. McDaid, "Embrace-sysc for analysis of noc-based spiking neural network architectures," in *System on Chip (SoC), 2010 International Symposium on*, sept. 2010, pp. 139 –145.

[36] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.

[37] S. Strogatz, "Exploring complex networks," *Nature*, vol. 410, no. 6825, pp. 268–276, Mar. 2001.

[38] B. Happel and J. Murre, "Design and evolution of modular neural network architectures," *Neural Networks*, vol. 7, no. 6-7, pp. 985 – 1004, 1994.

[39] G. Auda and M. Kamel, "Modular neural networks a survey," *International Journal of Neural Systems*, vol. 9, no. 2, pp. 129–151, 1999.

[40] Ronco and P. Gawthrop, "Modular neural networks: a state of the art," *Rapport Technique CSC95026 Center of System and Control University of Glasgow*, vol. 1, pp. 1–22, 1995.

CHAPTER 6

Application Prototyping for EMBRACE Hardware Modular Spiking Neural Network Architecture

This work is under review for publication in the Neural Computing and Applications journal.

Preamble

This chapter details application prototyping for the EMBRACE hardware modular Spiking Neural Network (SNN) FPGA prototype implementation.

Efficient implementation and training of large scale embedded applications on hardware SNN architectures poses a serious challenge due to the lack of appropriate application design methodologies. The SNN application design problem can be subdivided into the following categories:

- Application specific SNN topology design
- SNN information coding to insure appropriate application response time
- Dynamic application behaviour for unaccounted input scenarios
- SNN training algorithm for a range of real-world application classes

The chapter summarises reported EMBRACE FPGA prototype comprising Modular Neural Tiles (MNTs) and hierarchical Network on Chip (NoC) architec-

ture. The chapter presents GA-based SNN application evolution platform including tightly integrated robotics simulator.

This chapter addresses the application design problem by partitioning the robotic navigational controller application based on the functional aspects and sensory inputs. The detailed design of the application subtasks as medium grained SNN application functions is presented.

The chapter addresses the large scale SNN application training through successful evolution of the application subtasks and the complete robotic navigational controller application on the EMBRACE FPGA prototype. The proposed modular SNN evolution is compared with monolithic SNN evolution for a range of application complexities. The chapter highlights the reduction in evolution complexity and the SNN training speedup using the presented modular application design methodology.

Application Prototyping for EMBRACE Hardware Modular Spiking Neural Network Architecture

Sandeep Pande[a], Fearghal Morgan[a], Brian McGinley[a],
Jim Harkin[b], Liam Mc Daid[b]

[a] Bio-Inspired Electronics and Reconfigurable Computing Research Group (BIRC),
National University of Ireland, Galway, Ireland.
[c] Intelligent Systems Research Centre, University of Ulster,
Magee Campus, Derry, Northern Ireland.

Abstract: *Hardware modular Spiking Neural Network (SNN) architectures have been recently proposed as embedded computing platforms. In addition to real-life embedded applications such as classification, estimation, prediction and signal processing, these architectures are well suited for robotic control applications. The inherent non-algorithmic nature of these applications poses a challenge for their implementation on hardware SNN architectures. The Subsumption architecture, a layered application organisation partitions the overall behavioral robotic application into layers of control modules, where higher level layers subsume the roles of lower layer functions. A Modular Neural Network (MNN) execution architecture comprises interconnected neural computation modules that are capable of executing various control layers of the behavioural application.*

This paper presents a modular SNN application prototyping technique for the reported EMBRACE modular hardware SNN architecture comprisning neural tiles interconnected using ring topology Network on Chip interconnect. The paper presents a robotic navigational controller SNN application decomposed into obstacle avoidance controller and speed and direction manager application subtasks that are evolved separately and integrated using an integration module. Results indicate fast application evolution through stepwise knowledge integration and simplified SNN training. The paper presents validation results of the application on the EMBRACE FPGA architecture prototype.

6.1 Introduction

Conventional computing paradigms are suitable for applications comprising a series of well-defined calculations. Traditional computing techniques have a number of limitations when applied to real-world embedded applications, which are often characterised by the contradicting functional requirements in unexplored data and application scenarios resulting in the lack of an exact computational algorithm. Biologically inspired artificial neural network computing techniques mimic the key functions of the human brain and have the potential to offer smart and adaptive solutions for complex real world problems [1]. The organic nervous system includes a dense and complex interconnection of neurons and synapses, where each neuron connects to thousands of other neurons through synaptic connections. Computing systems based on Spiking Neural Networks (SNNs) emulate real biological neural networks, conveying information through the communication of short transient pulses (*spikes*) via synaptic connections between neurons [2, 3].

Embedded systems based on hardware SNNs offer elegant solutions as low-power and scalable embedded computing elements, characterised with rich non-linear dynamics. Hardware SNNs are suited to real-life applications including data/pattern classification, estimation, prediction, dynamic/non-linear control and signal processing [4, 5]. The authors have investigated and proposed EMBRACE[1] as an embedded computing architecture for the implementation of large scale SNNs [6–8]. EMBRACE comprises Modular Neural Tiles (MNTs) interconnected using a hierarchical Network on Chip (NoC) communication infrastructure [9]. EMBRACE uses a Genetic Algorithm (GA) based SNN evolution and configuration platform that configures the SNN synaptic weights and neuron threshold potentials and evolves the desired functionality by searching the correct SNN configuration. The authors have successfully applied the EMBRACE hardware SNN system to a number of benchmark data classification and control applications such as XOR data classifier, inverted pendulum controller, Wisconsin breast cancer dataset classifier and robotic controllers [7, 8, 10].

Solutions for real-world applications often lack deterministic algorithms. The inherent fuzzy nature of these applications poses a serious challenge for their implementation on hardware SNN architectures. Although, the GA based search technique offers a simple way of prototyping applications on hardware SNN platforms, the technique has a number of limitations including poor scalability and

[1] **EM**ulating **B**iologically-inspi**R**ed **Ar**C**h**itectures in hardwar**E**

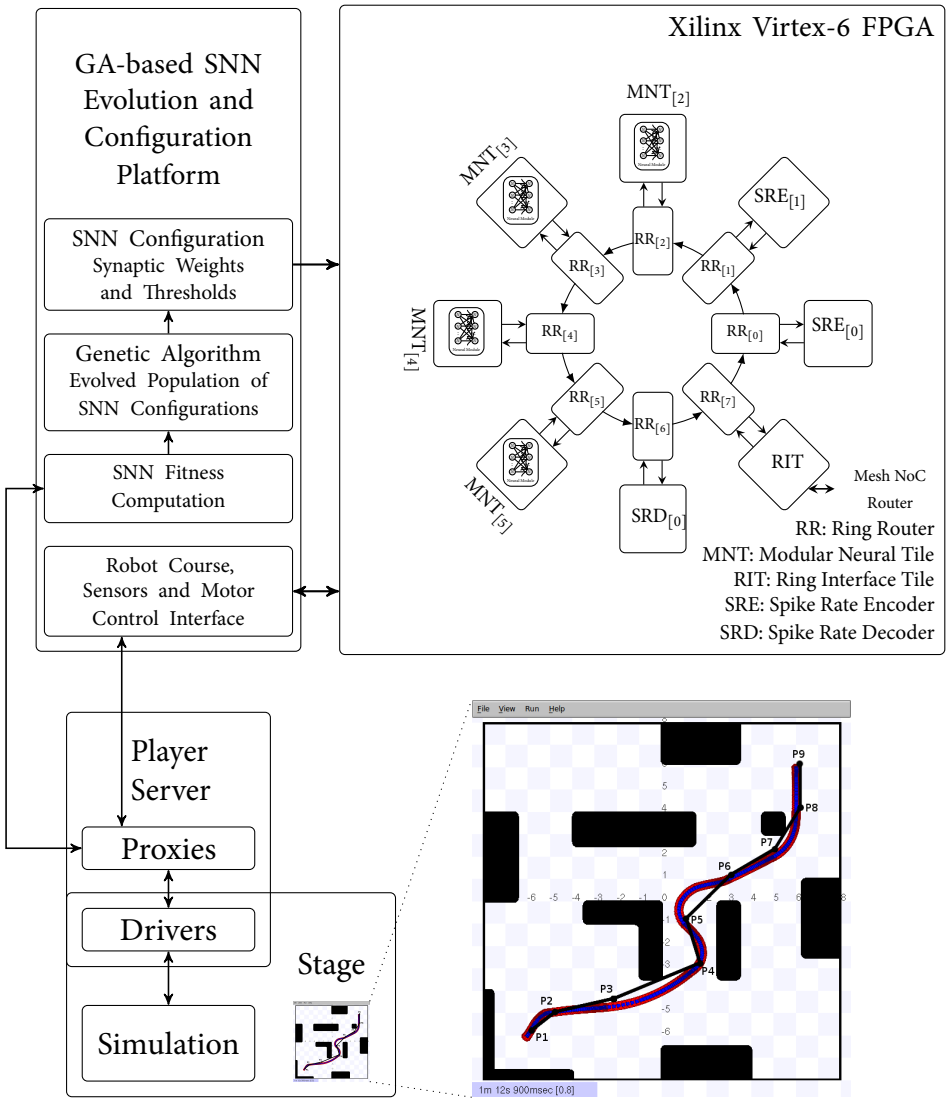


Figure 6.1: Robotic Controller Evolution Setup

search space explosion which restricts its use for evolution of complex SNN applications. These problems can be mitigated using the classic *divide and conquer* technique, by defining the overall behaviour as less complex orthogonal functions. A widely popular, layered application organisation for behavioral robots has been proposed in the *Subsumption Architecture* [11], which partitions overall functionality into layers of control modules, where higher level layers can subsume the roles of lower layer functions.

This paper presents a rapid application prototyping technique for the EMBRACE hardware SNN architecture using the subsumption architecture based modular application partitioning technique. This mitigates the problems associ-

ated with GA based SNN evolution for complex real-world embedded applications. The technique has been applied to a robotic navigational controller application which directs the robot on a pre-determined route in a two-dimensional environment containing obstacles. The multifunctional robotic navigational controller application has been decomposed into the Obstacle Avoidance Controller (OAC) and the Speed and Direction Manager (SDM) application subtasks. The OAC subtask uses sensory inputs for controlling the robot movement for obstacle avoidance while the SDM subtask manoeuvres the robot towards the target location. Figure 6.1 illustrates the robotic controller application evolution setup comprising the reported EMBRACE hardware SNN architecture FPGA prototype [8] [9] interfaced with the player-stage robotics simulator [12] and GA based hardware SNN evolution platform. The individual application subtasks (OAC and SDM) are evolved separately. The overall robotic behaviour is elaborated by combining the OAC and SDM output in the Integration Module (INT).

The paper presents the robotic navigational controller modular application design including SNN application topology, GA fitness criteria and application evolution. Results illustrate that the modular evolution of the application reduces the SNN configuration search space and complexity (for GA based SNN evolution), offering rapid prototyping of complex applications on the hardware SNN platform. The modular application design approach offers simplified SNN training and faster application evolution compared to the monolithic application evolution. The monolithic application evolution uses the overall application functionality as 'fitness criteria' during the GA based SNN evolution.

The structure of the paper is as follows: Section 6.2 reviews the MNN computing paradigm, hardware neural network execution architectures, modular application design and SNN training challenges. Section 6.3 presents the EMBRACE hardware SNN architecture (comprising MNTs interconnected using ring topology NoC) and FPGA prototype implementation. Section 6.4 presents the modular robotic navigational controller application design and compares it with the monolithic application evolution. Section 6.5 concludes the paper.

6.2 Related Work

This section reviews the Modular Neural Network (MNN) computing paradigm and discusses the MNN application design and execution architectures. SNN training techniques are reviewed to highlight their suitability for rapid SNN application

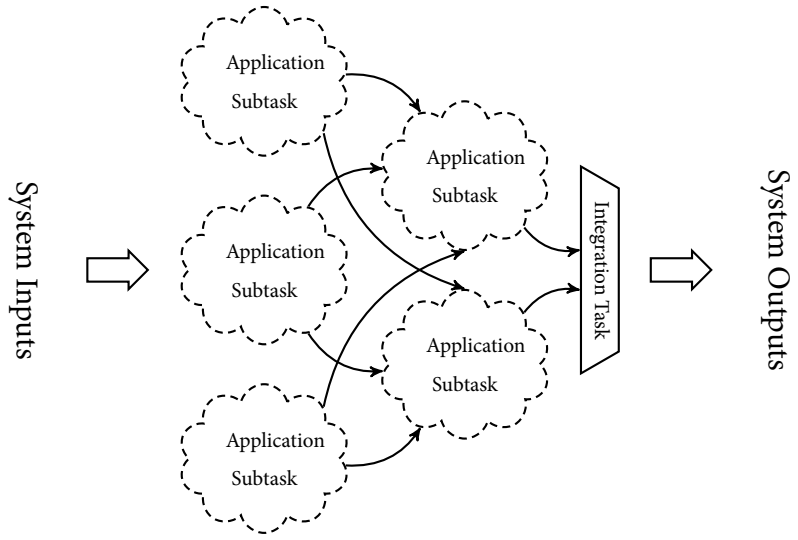


Figure 6.2: Modular Neural Network Application Organisation

prototyping on hardware SNN architectures.

6.2.1 Modular Neural Network Computing Paradigm

The biological brain is composed of several anatomically and functionally discrete areas which process various sensory and motor tasks, and different aspects of information [13–15]. This modular organisation observed in the brain has been the inspiration behind the MNN computing paradigm [16]. The MNN computing paradigm is primarily based on the *divide-and-conquer* strategy. The MNN design strategy proposes partitioning of application into a number of subtasks executing distinctly on neural modules [16–18]. Figure 6.2 illustrates MNN application organisation, where *task decomposition* of the high level application leads to smaller, less complex and manageable subtasks which are solved by individual and distinct neural modules. The intermediate outputs from these neural modules are combined to solve the high level task or the complete application [1, 19].

MNN Application Design

Application design for MNN systems primarily involves *Task Decomposition* to find the correct set of application subtasks in order to achieve the overall application behaviour. Various task decomposition algorithms based on different techniques such as output vector partitioning, class relationships, neuro-evolutionary and

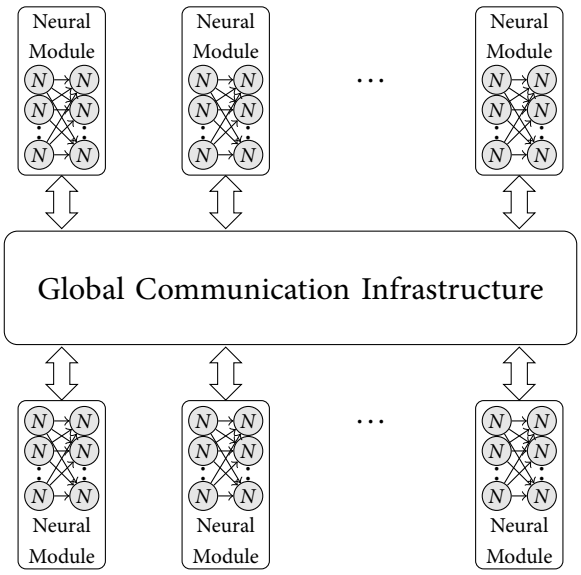


Figure 6.3: Modular Neural Network Execution Architecture

co-evolutionary methods have been proposed for MNN design strategy [20–24]. Subtasks or subnetworks obtained after task decomposition are executed in neural modules within an MNN computing architecture. Genetic algorithm based techniques have been applied to find subnetworks from large sized complex neural networks [25]. Networks for practical applications are observed to have clustered organisation and can map efficiently onto MNN execution architectures [26]. MNNs have been shown to outperform monolithic neural networks in terms of training time and data classification accuracy [27].

The MNN approach offers reduced training time, improved operation accuracy, structured implementation, functional partition, functional mapping and re-mapping, potentially competitive and co-operative mode of operation and fault tolerance [17]. A survey of MNN application designs is presented in [17, 18]. The *Subsumption Architecture* is a widely influential computing paradigm for reactive, behaviour-based autonomous robotic control applications. The subsumption architecture has been developed based on the MNN design concepts [11].

MNN Execution Architectures

Task decomposition of an application leads to two types of subtasks; namely the *Application Subtasks* (discrete functional subtasks) and the *Integration Subtasks*. The application subtasks operate on individual and distinct system inputs to provide intermediate outputs. The integration subtasks integrate the intermediate outputs

from the application subtasks to generate the overall system output. Both the application and integration subtasks are similar in terms of input/output interface and computational requirements. Figure 6.3 illustrates a typical MNN execution architecture comprising individual neural modules interconnected by a global communication infrastructure. Each neural module includes a group of neurons interconnected using an internal communication infrastructure and has multiple inputs and/or multiple outputs. The global (external inter-module) communication infrastructure provides connectivity between the individual neural modules [1, 19].

6.2.2 Spiking Neural Network Training Challenges

SNN training process involves iterative adjustment of the SNN configuration (synaptic weights, threshold potential and/or network topology) to achieve the desired application behaviour. The neural network training algorithms can be broadly classified as:

- **Unsupervised Training Algorithms:** These training algorithms attempt to find the hidden structures within the input data patterns. The technique relies on tuning the neural network to statistical regularities of the input data, such that the neural network correctly responds to these data patterns.

  Hebbian learning suggests strengthening of the synaptic weight w_{ij} between neurons i and j, whenever both neurons fire simultaneously [28]. Hebbian learning leads to reconfiguration of SNN and results in emergence of new functions, such as pattern recognition and associative memories [29]. The probability of a neuron firing based on the pre-synaptic spike timing is termed as Spike Timing Dependent Plasticity (STDP). STDP based unsupervised SNN training techniques have been effective for data/pattern classification, pattern recognition and associative memories [30–35].

- **Supervised Training Algorithms:** These algorithms are primarily characterised by the iterative adjustment of the neural network configuration based on feedback from a training data set. Back-Propagation and Spike-Prop SNN training algorithms are the most popular neural network supervised training algorithms [36, 37].

  Supervised Hebbian Learning (SHL) offers a biologically realistic implementation of Hebbian learning rules. SHL employs an additional 'teaching' signal that reinforces the post-synaptic neuron to fire at the target times and

to remain inactive otherwise. Detailed analysis of SHL for spiking neurons is presented in [38] [39].

- **Reinforcement Training Algorithms:** These algorithms tune the neural network configurations to achieve a predetermined goal, while interacting with the environment. The online neural network configuration exploration technique relies on rewarding the performance improving configurations and avoiding the performance deteriorating configuration [40–45]. These algorithms have enormous potential due to their high biological plausibility.

- **Evolutionary Training Algorithms:** Evolutionary methods have been successfully applied for SNN training and evolution of robotic applications [46, 47]. GA based SNN evolution is a supervised SNN training method that maintains a population of SNN configurations and uses nature inspired evolutionary techniques (comprising selection, crossover and mutation) to find a correct set of SNN configuration by evaluating the desired behaviour or 'fitness' of the individual SNN configurations. The technique neither requires any information about the application domain, nor imposes architectural constraints or any additional training specific elements in the neural computation components resulting in compact architecture [48]. The technique offers an easy and fast way to prototype applications on hardware SNNs.

  However, the GA based SNN evolution do not scale well for large multifunctional applications having complex fitness landscape. The search space for the binary coded, integer valued, n bit SNN configuration is 2^n. Increase in the SNN size (number of neurons and synapses) increases the GA search space and complexity by the order of $O(2^n)$. The technique also requires large amount of memory for storage of SNN configurations and results in slower operation due to iterative nature of SNN configuration evaluation. These shortcomings limit its use for evolving large real-world complex SNN applications.

The modular application prototyping presented in this paper reduces the search space and complexity for the GA based SNN evolution offering fast evolution of complex, multifunctional applications. The proposed technique helps rapid prototyping of real-world complex embedded applications on EMBRACE hardware SNN platform and mitigates the weaknesses of the GA based SNN evolution.

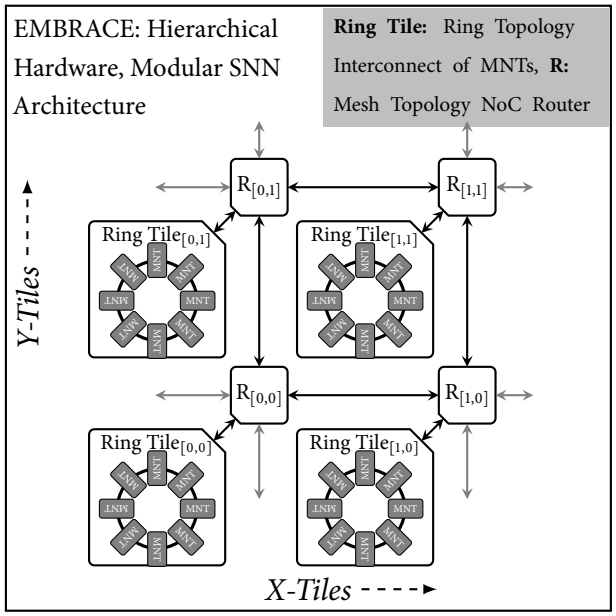


Figure 6.4: Hierarchical Modular SNN Architecture Comprising Ring Topology Interconnects within Mesh Topology NoC

6.3 EMBRACE Modular Neural Network Execution Architecture

This section presents EMBRACE architectural components.

Inspired by the computational abilities of biological nervous system, there is significant research in implementing reconfigurable and highly interconnected arrays of neural network elements in hardware to produce powerful cognitive signal processing units [49–58]. The efficient implementation of hardware SNN architectures for real-time embedded systems is primarily influenced by neuron design, scalable on-chip interconnect architecture and SNN training/learning algorithms [59].

The authors have reported the EMBRACE hardware modular SNN architecture as an embedded computing platform for real-life applications. The EMBRACE architecture depicted in figure 6.4 comprises Modular Neural Tiles (MNTs) interconnected using a scalable, hierarchical NoC [8, 9, 60].

6.3.1 Modular Neural Tile Architecture

The EMBRACE MNT architecture is a two-layered 16:16 fully connected feed-forward SNN structure (figure 6.5) [8, 60]. The input layer ($N[0, n]$) and output

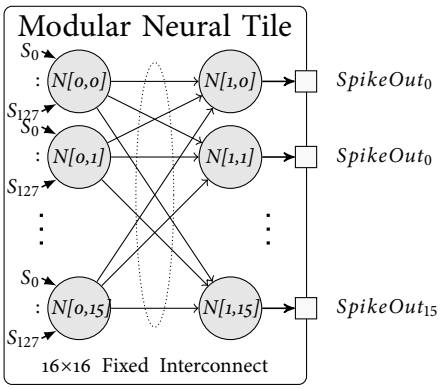


Figure 6.5: Two layered 16:16 Fully Connected SNN Structure as Modular Neural Tile

layer ($N[1, n]$) of the MNT comprises 16 Leaky-Integrate-and-Fire (LIF) neurons. Each neuron maintains a *membrane potential*, which is a function of incoming spikes, associated synaptic weights, current membrane potential, and the membrane potential *leakage coefficient* [2, 3]. A neuron *fires* (emits a spike to all connected synapses/neurons), when its membrane potential exceeds the neuron's firing threshold value. Each input layer neuron has 128 synapses corresponding to each output layer neuron within the ring. The MNT has 16 spike outputs ($SpikeOut_n$) each corresponding to the 16 output layer neurons.

The EMBRACE MNT architecture prototype on Xilinx Virtex-6 FPGA and micro-architectural details of the digital neuron model have been presented in [8]. The digital neuron circuit uses a 16-bit shift register for storage of the membrane potential value. Stepwise exponential decay of the membrane potential value is achieved by periodic right shifting (divide by 2). Analytical and simulation results indicate that the maximum spike error probability (noise) of the neuron model is approximately 0.03. The EMBRACE MNT FPGA prototype has been successfully applied to benchmark SNN application tasks representing classification and non-linear control functions [8].

6.3.2 EMBRACE Hierarchical NoC Architecture

The NoC based synaptic connectivity approach provides flexible inter-neuron communication channels, scalable interconnect and connection reconfigurability [10, 61]. The EMBRACE spike communication infrastructure is a bi-level, hierarchical NoC architecture. The MNTs are grouped and interconnected through a synchronous ring topology interconnect. The MNT groups (organised as ring

tiles) are interconnected to an upper level mesh topology network of routers. The following subsections describe these interconnects and the spike flow between MNTs.

Ring Interconnect Architecture

Modular SNNs exhibit a high density of synaptic connections within localised groups. The ring topology interconnect offers fixed spike transfer latency for inter-MNT spike communication within the ring [9]. The interconnect comprises routers connected in a unidirectional ring topology, where each router receives a spike packet from the previous router and sends a spike packet to the next router. The MNTs interface with ring routers for spike communication. Each router buffers the spike events generated by the attached MNT and encodes them in spike packets. Generated spike packets are processed and forwarded in a single clock cycle. Full rotation of the spike packet on the ring ensures broadcast packet flow control.

Fixed latency flow control of the ring topology interconnect uses timestamping of the input spikes at the source router, and broadcasting of spike packets to all routers within the ring. Each destination router sorts and buffers spike events based on the received spike packet timestamp value. Buffered spike events are converted to output spikes at the precise clock cycle (corresponding to the source timestamp) determined by the timeslot counter. The ring NoC interconnect offers fixed latency spike communication which eliminates information distortion in SNNs and results in stable and reliable application behaviour.

Mesh NoC Architecture

Each ring contains seven MNTs and a Ring Interface Tile (RIT) (instantiated as tile number $n = 7$ figure 6.1). The RIT acts as the bridge between the ring interconnect and mesh NoC router, facilitating the synaptic connectivity between MNTs within the ring and the rest of the chip [9]. An incoming spike on an input synapse is converted to a spike packet by the packet encoder and forwarded to the attached NoC router. The NoC architecture supports unicast packet flow control, where each spike packet contains destination synapse information for a single spike and is routed independently. Different SNN topologies are created by configuring traffic connections between SNN elements (neurons, spike rate encoders and decoders).

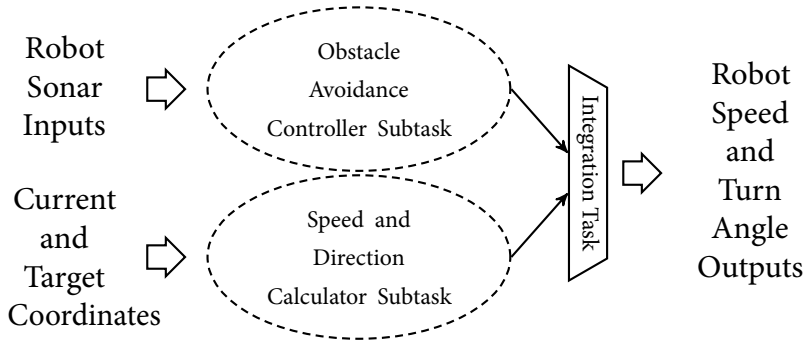


Figure 6.6: Robotic Navigational Controller Application Organisation

6.4 Modular Application Design

Real-world applications comprise many functional and behavioural elements. Although these applications lack algorithmic solutions, a solution can be built using a number of application modules solving distinct functional elements. This section highlights modular application design on hardware SNN architectures based on the subsumption architecture design style. The section presents the modular robotic navigational controller application design including task decomposition, subtask design and evolution on EMBRACE-FPGA.

The subsumption architecture proposes decomposition of the overall application in small sized subtasks arranged in a layered fashion, where each subtask addresses a specific application functionality [11]. Based on the subtask functionality, individual subtasks can be supplied with all or a subset of the system inputs. Subtask outputs are combined into higher level tasks or integration tasks. The final system output is always produced by the integration tasks based on various intermediate output patterns.

In summary, an application can be primarily decomposed based on:

- sensory inputs
- application goals or functionalities
- functional redundancy (multiple application modules assigned for same functionality to increase robustness)
- structured application layers (functional levels during integration phase)
- application extensibility

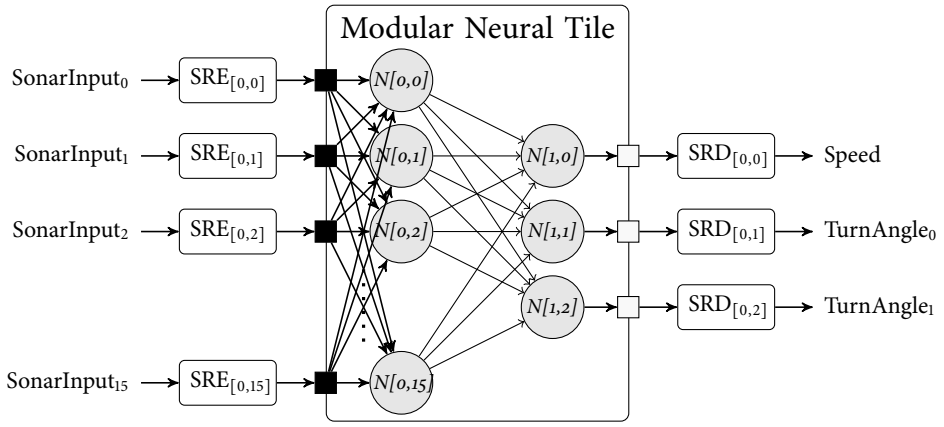


Figure 6.7: Obstacle Avoidance Controller Subtask SNN Topology

6.4.1 Robotic Navigational Controller Application Design

The primary goal of the robotic navigational controller application is to direct the robot on a pre-determined route in a two-dimensional simulated environment containing obstacles. The route is marked with fixed points (or markers) and the two-dimensional simulated world contains obstacles that must be avoided by the robot (depicted in figure 6.1). The two main functions of the applications are

- obstacle avoidance in the two-dimensional environment and
- speed and turn angle calculation to reach the next marker.

The obstacle avoidance function requires information about the robot surrounding, obtained through the sonar proximity sensors on the robot. The speed and turn angle calculation function requires information about the current robot position and orientation, and the next marker coordinate. Based on these functional aspects and sensory input requirements, the robotic navigational controller can be partitioned into **Obstacle Avoidance Controller** (OAC) and **Speed and Direction Manager** (SDM) subtasks. Figure 6.6 illustrates the robotic navigational controller application organisation. Both the OAC and SDM subtasks produce speed and turn angle output suitable for manoeuvring the robot for obstacle avoidance and advancing towards target respectively. These subtask outputs can be integrated to produce actuation signals (speed and turn angle) for the robot.

6.4.2 Obstacle Avoidance Controller Subtask Design

The Obstacle Avoidance Controller (OAC) controls the speed and turn angle of the robot to avoid collisions with the obstacles within the two-dimensional simulated

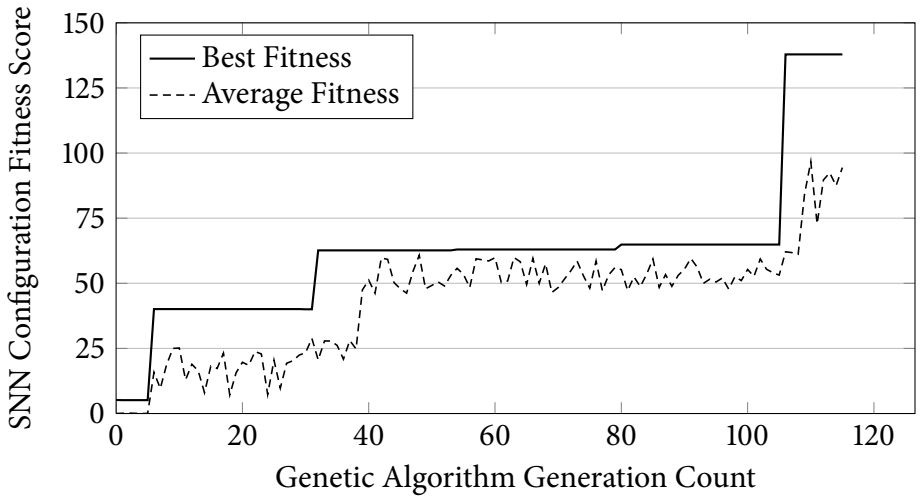


Figure 6.8: Obstacle Avoidance Controller Subtask Evolution (Fitness Evaluation Constants: α = 0.25 and β = 0.125)

robotic environment[2]. The robotic application evolution setup illustrated in figure 6.1 is used for OAC subtask evolution. The simulated robot is equipped with 16 sonar proximity sensors which act as input to the OAC subtask SNN. The OAC subtask SNN topology illustrated in figure 6.7 is mapped to $MNT_{[2]}$ in figure 6.1. Robot sonar values are converted to spike rates by the host application and are passed to spike rate encoders ($SRE_{[0]}$), which continuously generate spikes into the MNT. Spikes from three MNT outputs (corresponding to speed and differential turnangle) are monitored by spike rate decoders ($SRD_{[0]}$) and are converted to analogue values as robot speed and turning angle inputs to the simulator (where, Turning angle = $TurnAngle_0$ - $TurnAngle_1$). The use of differential outputs for the calculation of turnangle eliminates the directional bias by individual outputs.

The fitness criteria for the OAC application is defined as robot travelling a finite distance and avoiding obstacles for $t \geq 120$ seconds. The fitness of the SNN configuration is calculated as:

$$F_{OAC} = \alpha T + \beta D \tag{6.1}$$

Where:

F_{OAC} = Fitness of the individual (for OAC subtask)
T = Robot travel time (in $seconds$)

[2]The authors have previously reported the evolution capabilities of the MNT and the robotic obstacle avoidance controller benchmark SNN application [8].

D = Robot travel distance (in cms)

Given:

Number of collisions = 0
SNN outputs are within the operating range ($0 \leq Speed \leq 5$ and $-3.14 \leq TurnAngle \leq 3.14$)
Robot does not spin indefinitely
The fitness evaluation constants α and β are chosen to scale the outputs to match the time and distance measurement units.

Evaluation of the individual configuration has been accomplished with the robot roaming within the simulated environment for $120s \leq t \leq 300s$. On timeout ($t = 300s$), or if the robot has collided with an obstacle, the GA-based evolution and configuration platform processes the recorded robot behaviour and assigns a fitness score to the individual SNN configuration. Fitness scores are used by the GA to determine the probability of an individual SNN configuration progressing to the subsequent evolved generation of individual SNN configurations. Figure 6.8 illustrates the average and best fitness of the evolved OAC application subtask on the MNT.

6.4.3 Speed and Direction Manager Subtask Design

The Speed and Direction Manager (SDM) subtask computes required speed and turn angle to advance the robot to the next marker. The robot is navigated by presenting a series of markers to the SDM subtask. The SDM subtask takes the current robot coordinates and orientation (from positioning sensors on the robot), the target marker coordinates as input and generates speed, and the turn angle. The ideal speed and turn angle can be calculated using trigonometric equations (6.2) (6.3) (6.4) (6.5). The fitness function for the evolution of the SDM application subtask is given by equations (6.3) (6.5).

$$D = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} \tag{6.2}$$

$$S = \gamma D \tag{6.3}$$

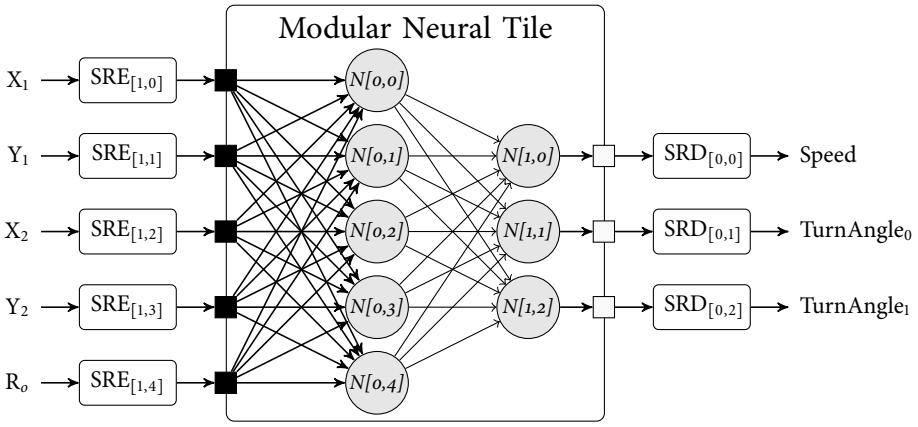


Figure 6.9: Speed and Direction Manager Subtask SNN Topology

$$\theta = tan^{-1}\left(\frac{Y_2 - Y_1}{X_2 - X_1}\right) \tag{6.4}$$

$$T_a = \theta - R_o \tag{6.5}$$

Where:

X_1, Y_1 = Current Coordinate
X_2, Y_2 = Target marker Coordinate
θ = Angle for (X_1, Y_1) to (X_2, Y_2)
R_o = Current robot orientation
T_a = Robot Turn Angle
D = Distance between (X_1, Y_1) and (X_2, Y_2)
γ = 0.475
S = Robot speed

The SDM subtask SNN topology illustrated in figure 6.9 is mapped to $MNT_{[3]}$ in figure 6.1. The current and target marker coordinates are converted to spike rates by the host application and are passed to spike rate encoders ($SRE_{[1]}$), which continuously generate and feed spikes to the MNT. Spikes from three MNT outputs are monitored by spike rate decoders ($SRD_{[0]}$) and are converted to analogue values as robot speed and turning angle (Turning angle = $TurnAngle_0$ - $TurnAngle_1$). The evolution process uses a number of combinations of current and target marker coordinates to ensure minimum calculation error. Figure 6.10 illustrates average and best error values for the evolution of the SDM application subtask on the MNT.

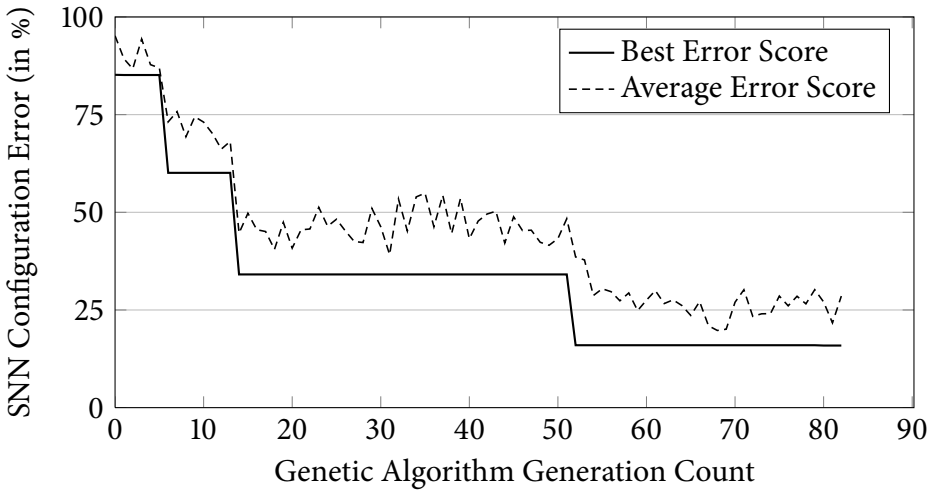


Figure 6.10: Speed and Direction Manager Subtask Evolution (Fitness Evaluation Constants: $\gamma = 0.1$)

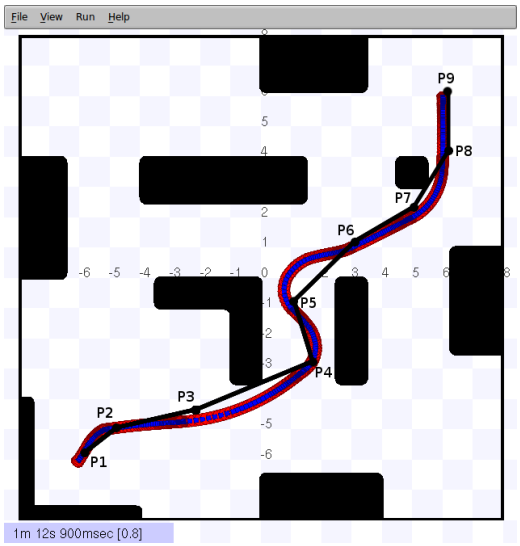


Figure 6.11: Robotic Navigational Controller Application Demonstration

The error value represents the deviation of the calculated speed and turn angle by the SDM subtask from the ideal values calculated using equation 6.2 and 6.4.

6.4.4 Integration Subtask Design

The obstacle avoidance controller and speed and direction manager subtask individually solve a functional aspect of the overall application. The integration task combines the intermediate outputs from OAC and SDM application subtasks to generate the overall output.

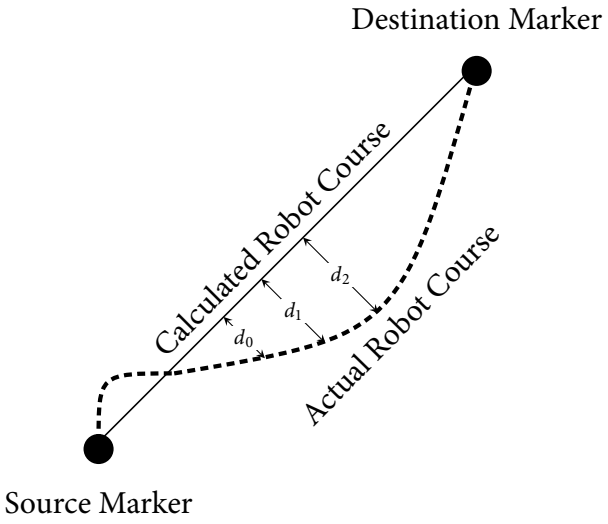


Figure 6.12: Robot Course Deviation Calculation

$$F_{INT} = \frac{1}{\Sigma d_i} \tag{6.6}$$

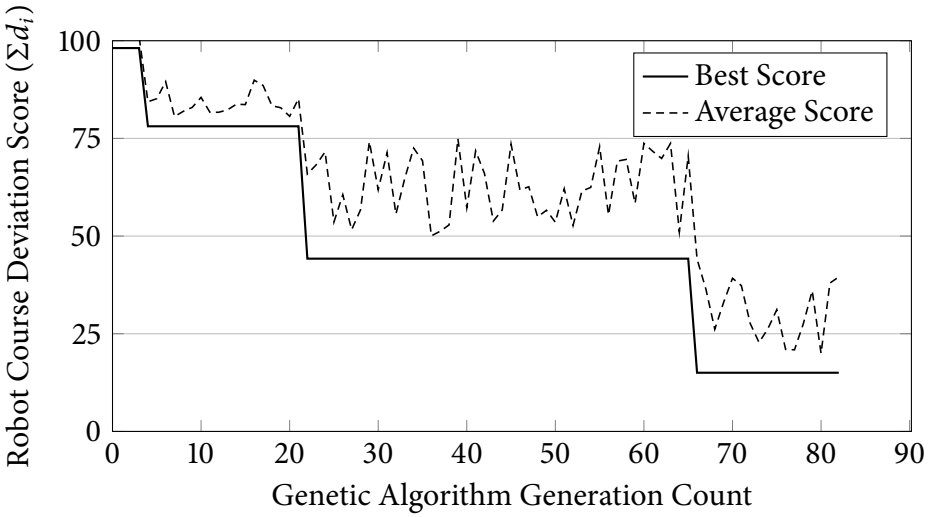


Figure 6.13: Integration Subtask Evolution

The modular robotic navigational controller application (figure 6.6) is mapped onto three MNTs executing the OAC, SDM and integration subtasks in the robotic application evolution setup illustrated in figure 6.1. The application subtask MNTs are configured with the synaptic configuration obtained through SNN evolution. The integration subtask has six individual inputs (from the two application sub-

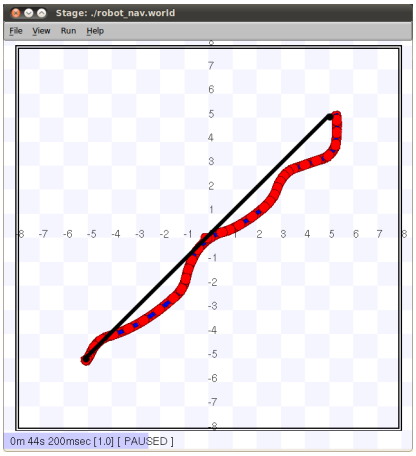
tasks) and three outputs namely Speed, TurnAngle_0 and TurnAngle_1. Figure 6.11 illustrates the robotic navigational controller application on the EMBRACE-FPGA platform. The robot route (shown as the thinner line) is marked by the series of markers P_1, P_2, ... P_9. The actual robot course is shown as a trail (thick line). The accuracy of the integration subtask and the overall application is evaluated based on the deviation of the robot course from the marked route figure 6.12. Equation 6.6 illustrates the fitness function for the overall application based on the robot course deviation from the calculated route. Figure 6.13 illustrates the average and best robot course deviation score for the evolution of integration subtask on the MNT.

6.4.5 Application Knowledge Integration

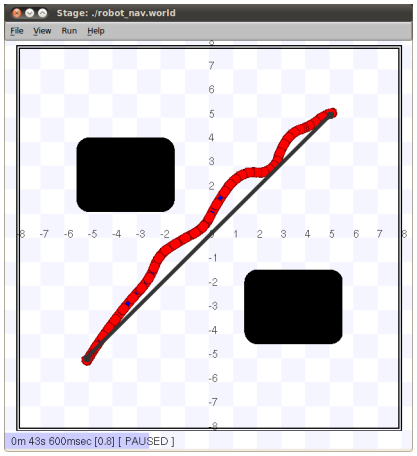
Real-world applications often encounter scenarios where no unique solution can satisfy all the functional aspects of the application. Different application behaviours have contradicting requirements for certain application scenarios which makes it impossible to have a correct solution for the whole application. Often a balanced solution satisfying all the functional aspects to a certain extent helps to solve the overall application and overcome the application scenario gracefully. One such scenario in the robotic navigational controller application is manoeuvring the robot around obstacle corners. The ideal solution to direction calculation functionality requires the robot to move towards the (obstructed) target, whereas the ideal solution to obstacle avoidance functionality requires the robot to steer away from the obstacle.

The proposed modular application evolution is compared with the monolithic SNN evolution, where all the three subtasks are evolved as a single SNN to highlight the problem of complex SNN application evolution. The monolithic SNN evolution has been performed for a range of application scenarios with increasing complexity level to understand the application evolution for complex scenarios. Figure 6.14 illustrates the monolithic SNN evolution for a robotic world with increasing complexity level[3]. Figure 6.15 illustrates average robot course deviation score for the evolution of the robotic navigational controller monolithic SNN application for worlds 1-5 in figure 6.14. The monolithic SNN successfully evolves for application scenarios with low complexity levels, where a number of possible solutions satisfy all of the application requirements (world no. 1-5). However, the

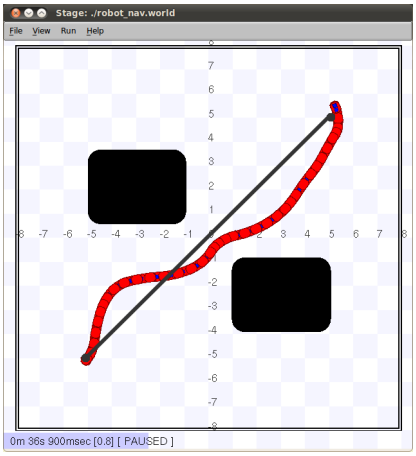
[3]The complexity level in the worlds is controlled by arranging the obstacles alongside the ideal robot course.



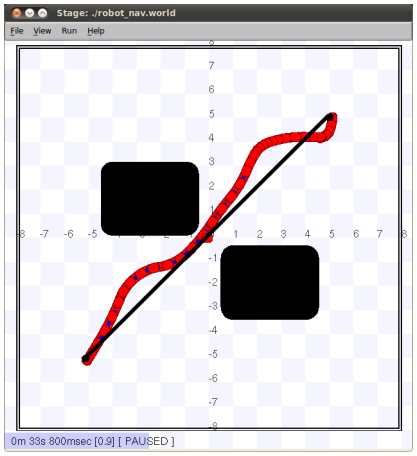
(a) World No. 1 (Lowest Complexity)



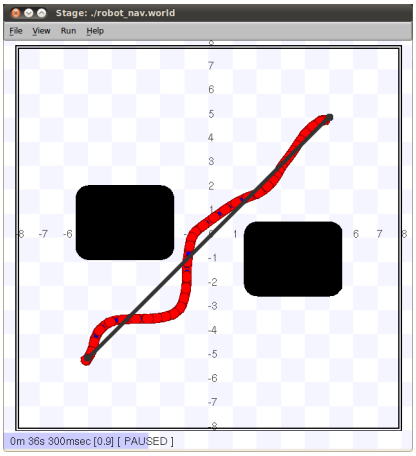
(b) World No. 2



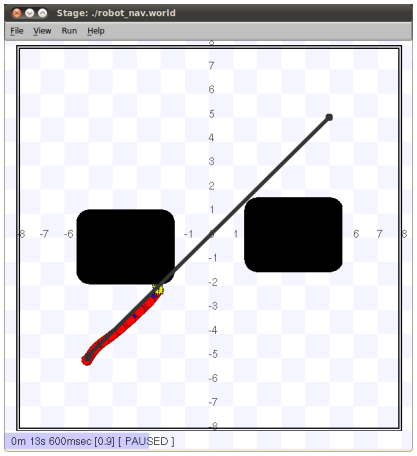
(c) World No. 3



(d) World No. 4



(e) World No. 5 (Highest Complexity with Successful Evolution)



(f) World No. 6

Figure 6.14: Robotic Navigational Controller Application Demonstration on Simulated Worlds with Increasing Complexity on Monolithic SNN

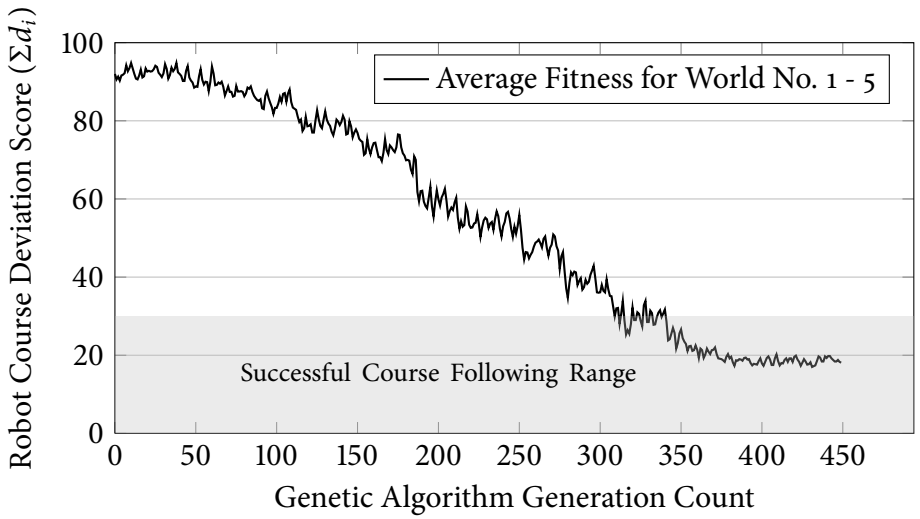


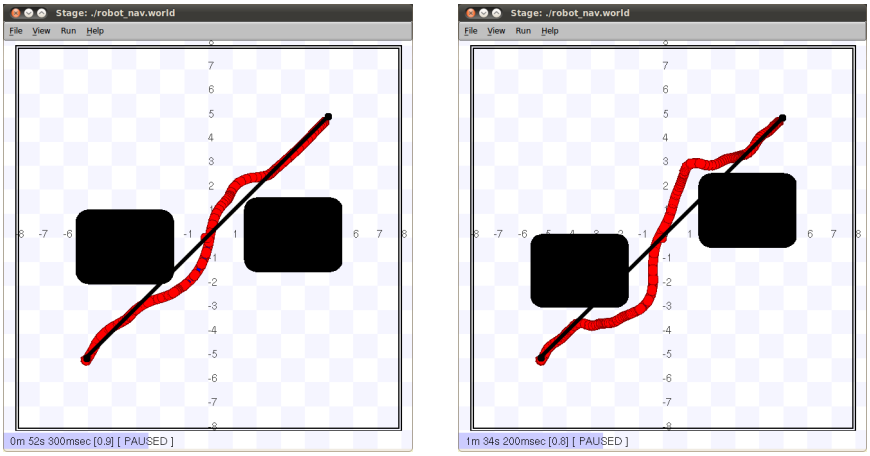
Figure 6.15: Monolithic Application Evolution

monolithic SNN evolution fails for more complex application scenario (world no. 6) where the possible solution is a balance between contradicting application behaviours (advancing on the designated course and avoiding obstacles). The monolithic SNN evolves for robot movement on the designated course but fails to avoid obstacles.

The modular application evolution presented in this paper evolves each functional element of the application separately and integrates the solution to generate the system output. The approach generates a balanced solution in various application scenarios (e.g. robot path from marker P_3 to P_4 and P_7 to P_8) and efficiently navigates the robot on the marked course. Figure 6.16 illustrates the robotic navigational controller modular SNN application for worlds 6-7. The proposed modular application evolution offers an effective and generalised solution by balancing the contradicting application behaviours.

6.4.6 Application Evolution Complexity

The individual SNN configuration in GA based search comprises synaptic weight (5-bit signed value) and threshold potential (16-bit unsigned value) of all the neurons within the SNN. The binary coded SNN configuration has the search space of 2^n for n bits of configuration information. Increasing the SNN size (number of neurons and synapses) increases the GA search space and complexity by the order of $O(2^n)$. The proposed modular SNN evolution technique is compared with the earlier reported monolithic SNN evolution (as depicted in figure 6.14). Table 6.1



(a) World No. 6 (b) World No. 7

Figure 6.16: Robotic Navigational Controller Application Demonstration on Simulated Worlds with Increasing Complexity on Modular SNN

Table 6.1: Modular Robotic Navigational Controller Application SNN Configuration Complexity

Application Organisation	Modular				Monolithic
	OAC	SDM	INT	Modular Total	
Number of neurons	19	8	9	36	36
Number of synapses	304	40	54	398	398
Bits in the GA gene	6384	328	414	7126	7126
GA search space	2^{6384}	2^{328}	2^{414}		2^{7126}
Total GA search space	$(2^{6384} + 2^{328} + 2^{414})$				2^{7126}

illustrates the SNN configuration of the robotic navigational controller application subtasks and overall application evolution complexity for monolithic and modular evolution approaches. In the monolithic SNN evolution approach, all the three subtasks are evolved as a single SNN, which results in a massive solution search space of 2^{7126} (table 6.1. Due to massive search space (2^{7126}) the GA-based search requires hundreds of generations in order to effectively search for a correct SNN configuration (even for simple application scenarios). The modular SNN evolution speed-up is evident from the low number of GA generation count compared to monolithic SNN evolution (Refer figure 6.8, figure 6.10, figure 6.13 and figure 6.15).

The modular approach evolves each subtask is separately resulting in a considerably smaller solution search space of $2^{6384} + 2^{616} + 2^{414}$ (Table 6.1). The modular

SNN implementation results in smaller solution search space, mitigating the SNN evolution problem many folds and speeding-up the SNN evolution process.

6.5 Conclusions

This paper presents reliable SNN application behaviour using the EMBRACE modular hardware SNN architecture comprising ring topology NoC interconnect and modular application prototyping technique. The EMBRACE architecture and its suitability as an embedded computing platform has been presented. The limitations of GA based search algorithms (such as poor scalability and search space explosion) and their unsuitability for evolving large, monolithic and complex SNN applications have been discussed. The modular neural network computing paradigm has been discussed with emphasis on real-life complex application design. The paper presents task decomposition for practical embedded applications based on application functionality, robustness, extensibility and various sensory inputs.

The paper presents the design of the robotic navigational controller application. The gradual SNN evolution of the application subtasks and integration tasks results in a successful application behaviour on the EMBRACE-FPGA prototype. The modular application integration mitigates the search space and evolution complexity for GA based SNN evolution. The modular application design for the EMBRACE hardware modular SNN architecture results in faster application evolution compared to monolithic SNN evolution, through stepwise knowledge integration and simplified SNN training. The proposed technique offers a rapid, simple and effective application prototyping for hardware SNN architectures.

Acknowledgment

This research is supported by International Centre for Graduate Education in Micro and Nano-Engineering (ICGEE), Irish Research Council for Science, Engineering and Technology (IRCSET) and Xilinx University Programme.

References

[1] Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999, vol. 13.

[2] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659 – 1671, 1997.

[3] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, Aug. 2002.

[4] S. Bohte and J. Kok, "Applications of spiking neural networks," *Information Processing Letters*, vol. 95, no. 6, pp. 519–520, 2005.

[5] M. Pearson, A. Pipe, B. Mitchinson, K. Gurney, C. Melhuish, I. Gilhespy, and M. Nibouche, "Implementing spiking neural networks for real-time signal-processing and control applications: A model-validated fpga approach," *Neural Networks, IEEE Transactions on*, vol. 18, no. 5, pp. 1472 –1487, sept 2007.

[6] J. Harkin, F. Morgan, L. McDaid, S. Hall, B. McGinley, and S. Cawley, "A reconfigurable and biologically inspired paradigm for computation using network-on-chip and spiking neural networks," *Int. J. Reconfig. Comput.*, vol. 2009, pp. 2:1–2:13, Jan. 2009.

[7] 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*, vol. 12, pp. 257–280, 2011.

[8] S. Pande, F. Morgan, S. Cawley, T. Bruintjes, G. Smit, B. McGinley, S. Carrillo, J. Harkin, and L. McDaid, "Modular neural tile architecture for compact embedded hardware spiking neural network," *Neural Processing Letters*, pp. 131–153, Oct. 2013. [Online]. Available: http://dx.doi.org/10.1007/s11063-012-9274-5

[9] S. Pande, F. Morgan, G. Smit, T. Bruintjes, J. Rutgers, B. McGinley, S. Cawley, J. Harkin, and L. McDaid, "Fixed latency on-chip interconnect for hardware spiking neural network architectures," *Parallel Computing*, vol. 39, pp. 357 – 371, Sep. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2013.04.010

[10] F. Morgan, S. Cawley, B. McGinley, S. Pande, L. McDaid, B. Glackin, J. Maher, and J. Harkin, "Exploring the evolution of noc-based spiking neural networks on fpgas," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, dec. 2009, pp. 300 –303.

[11] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, mar 1986.

[12] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm Intelligence*, vol. 2, pp. 189–208, Aug. 2008.

[13] S. Johannes, B. M. Wieringa, M. Matzke, and T. F. MÃijnte, "Hierarchical visual stimuli: electrophysiological evidence for separate left hemispheric global and local processing mechanisms in humans," *Neuroscience Letters*, vol. 210, no. 2, pp. 111–114, May 1996.

[14] D. Van Essen, C. Anderson, and D. Felleman, "Information processing in the primate visual system: an integrated systems perspective," *Science*, vol. 255, no. 5043, pp. 419–423, Jan. 1992.

[15] T. Binzegger, R. J. Douglas, and K. A. C. Martin, "Stereotypical bouton clustering of individual neurons in cat primary visual cortex," *The Journal of Neuroscience*, vol. 27, no. 45, pp. 12 242–12 254, Nov. 2007.

[16] B. Happel and J. Murre, "Design and evolution of modular neural network architectures," *Neural Networks*, vol. 7, no. 6-7, pp. 985 – 1004, 1994.

[17] G. Auda and M. Kamel, "Modular neural networks a survey," *International Journal of Neural Systems*, vol. 9, no. 2, pp. 129–151, 1999.

[18] Ronco and P. Gawthrop, "Modular neural networks: a state of the art," *Rapport Technique CSC95026 Center of System and Control University of Glasgow*, vol. 1, pp. 1–22, 1995.

[19] D. N. Osherson, S. Weinstein, and M. Stob, "Modular learning." Cambridge, MA, USA: MIT Press, 1993, pp. 369–377.

[20] S. Guan, S. Li, and S. K. Tan, "Neural network task decomposition based on output partitioning," *Journal of the Institution of Engineers Singapore*, vol. 44, pp. 78–89, 2004.

[21] B.-L. Lu and M. Ito, "Task decomposition and module combination based on class relations: a modular neural network for pattern classification," *Neural Networks, IEEE Transactions on*, vol. 10, no. 5, pp. 1244–1256, sep 1999.

[22] J. Thangavelautham and G. M. T. D.eleuterio, "A neuroevolutionary approach to emergent task decomposition," in *Proc. of 8th Parallel Problem Solving from Nature*. Springer, 2004, pp. 991–1000.

[23] V. Khare, X. Yao, B. Sendhoff, Y. Jin, and H. Wersing, "Co-evolutionary modular neural networks for automatic problem decomposition," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3, sept 2005, pp. 2691–2698.

[24] J. Santos, L. Alexandre, and J. de Sa, "Modular neural network task decomposition via entropic clustering," in *Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on*, vol. 1, oct. 2006, pp. 62–67.

[25] C. Pizzuti, "A multiobjective genetic algorithm to find communities in complex networks," *Evolutionary Computation, IEEE Transactions on*, vol. 16, no. 3, pp. 418–430, june 2012.

[26] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.

[27] B. Gour, T. Bandopadhyaya, and R. Patel, "Art and modular neural network architecture for multilevel categorization and recognition of fingerprints," in *Knowledge Discovery and Data Mining, 2010. WKDD '10. Third International Conference on*, jan. 2010, pp. 536–539.

[28] D. Hebb, "The organization of behavior; a neuropsychological theory." 1949.

[29] G. E. Hinton and T. J. Sejnowski, *Unsupervised learning: foundations of neural computation*. The MIT press, 1999.

[30] T. Natschlaeger and B. Ruf, "Online clustering with spiking neurons using temporal coding," *Progress in Neural Processing*, pp. 33–42, 1998.

[31] S. Bohte, H. La Poutre, and J. Kok, "Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer rbf networks," *Neural Networks, IEEE Transactions on*, vol. 13, no. 2, pp. 426–435, 2002.

[32] F. Landis, T. Ott, and R. Stoop, "Hebbian self-organizing integrate-and-fire networks for data clustering," *Neural computation*, vol. 22, no. 1, pp. 273–288, 2010.

[33] J. Hopfield *et al.*, "Pattern recognition computation using action potential timing for stimulus representation," *Nature*, vol. 376, no. 6535, pp. 33–36, 1995.

[34] W. Gerstner and J. Van Hemmen, "Associative memory in a network of spiking neurons," *Network*, vol. 3, no. 2, pp. 139–164, 1992.

[35] M. Zamani, A. Sadeghian, and S. Chartier, "A bidirectional associative memory based on cortical spiking neurons using temporal coding," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*, 2010, pp. 1–8.

[36] E. Alpaydin, *Introduction to machine learning*. The MIT Press, 2010.

[37] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

[38] R. Legenstein, C. Naeger, and W. Maass, "What can a neuron learn with spike-timing-dependent plasticity?" *Neural Computation*, vol. 17, no. 11, pp. 2337–2382, 2005.

[39] R. Legenstein, D. Pecevski, and W. Maass, "A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback," *PLoS Computational Biology*, vol. 4, no. 10, p. e1000180, 2008.

[40] D. Baras and R. Meir, "Reinforcement learning, spike-time-dependent plasticity, and the bcm rule," *Neural Computation*, vol. 19, no. 8, pp. 2245–2279, 2007.

[41] M. A. Farries and A. L. Fairhall, "Reinforcement learning with modulated spike timing–dependent synaptic plasticity," *Journal of neurophysiology*, vol. 98, no. 6, pp. 3648–3665, 2007.

[42] R. Florian, "A reinforcement learning algorithm for spiking neural networks," in *Symbolic and Numeric Algorithms for Scientific Computing, 2005. SYNASC 2005. Seventh International Symposium on*, 2005.

[43] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural Computation*, vol. 19, no. 6, pp. 1468–1502, 2007.

[44] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral Cortex*, vol. 17, no. 10, pp. 2443–2452, 2007.

[45] E. Vasilaki, N. Frémaux, R. Urbanczik, W. Senn, and W. Gerstner, "Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail," *PLoS computational biology*, vol. 5, no. 12, p. e1000586, 2009.

[46] E. Di Paolo, "Spike-timing dependent plasticity for evolved robots," *Adaptive Behavior*, vol. 10, no. 3-4, pp. 243–263, 2002.

[47] H. Hagras, A. Pounds-Cornish, M. Colley, V. Callaghan, and G. Clarke, "Evolving spiking neural network controllers for autonomous robots," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 5, 2004, pp. 4620–4626.

[48] M. Melanie, "An introduction to genetic algorithms," *Cambridge, Massachusetts London, England, Fifth printing*, vol. 3, 1999.

[49] J. Harkin, F. Morgan, S. Hall, P. Dudek, T. Dowrick, and L. McDaid, "Reconfigurable platforms and the challenges for large-scale implementations of spiking neural networks," in *International Conference on Field Programmable Logic and Applications, 2008. FPL 2008.* IEEE, Sep. 2008, pp. 483–486.

[50] Y. Chen, S. Hall, L. McDaid, O. Buiu, and P. Kelly, "A solid state neuron for the realisation of highly scaleable third generation neural networks," in *Solid-State and Integrated Circuit Technology, 2006. ICSICT '06. 8th International Conference on*, 2006, pp. 1071–1073.

[51] S. Furber and A. Brown, "Biologically-inspired massively-parallel architectures - computing beyond a million processors," in *Application of Concurrency to System Design, 2009. ACSD '09. Ninth International Conference on*, july 2009, pp. 3 –12.

[52] A. Upegui and E. PeÃśa-Reyes, CC.and Sanchez, "An fpga platform for online topology exploration of spiking neural networks," *Microprocessors and Microsystems*, vol. 29, no. 5, pp. 211 – 223, 2005.

[53] M. Pearson, A. Pipe, B. Mitchinson, K. Gurney, C. Melhuish, I. Gilhespy, and M. Nibouche, "Implementing spiking neural networks for real-time signal-processing and control applications: A model-validated fpga approach," *Neural Networks, IEEE Transactions on*, vol. 18, no. 5, p. 1472, 2007.

[54] E. Ros, E. Ortigosa, R. Agis, R. Carrillo, and M. Arnold, "Real-time computing platform for spiking neurons (rt-spike)," *Neural Networks, IEEE Transactions on*, vol. 17, no. 4, pp. 1050 – 1063, july 2006.

[55] R. Vogelstein, U. Mallik, J. Vogelstein, and G. Cauwenberghs, "Dynamically reconfigurable silicon array of spiking neurons with conductance-based synapses," *Neural Networks, IEEE Transactions on*, vol. 18, no. 1, pp. 253 –265, jan. 2007.

[56] M. Ehrlich, C. Mayr, H. Eisenreich, S. Henker, A. Srowig, A. Grubl, J. Schemmel, and R. Schuffny, "Wafer-scale VLSI implementations of pulse coupled neural networks," in *Proceedings of the International Conference on Sensors, Circuits and Instrumentation Systems*, 2007.

[57] B. Glackin, T. McGinnity, L. Maguire, Q. Wu, and A. Belatreche, "A novel approach for the implementation of large scale spiking neural networks on fpga hardware," in *Computational Intelligence and Bioinspired Systems*, ser. Lecture Notes in Computer Science, J. Cabestany, A. Prieto, and F. Sandoval, Eds. Springer Berlin / Heidelberg, 2005, vol. 3512, pp. 1–24.

[58] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, june 2008, pp. 431 –438.

[59] L. Maguire, T. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for large-scale implementations of spiking neural networks on fpgas," *Neurocomputing*, vol. 71, no. 1-3, pp. 13 – 29, Dec. 2007.

[60] S. Pande, F. Morgan, S. Cawley, B. McGinley, J. Harkin, S. Carrillo, and L. McDaid, "Addressing the hardware resource requirements of Network-on-Chip based neural architectures."

[61] D. Vainbrand and R. Ginosar, "Scalable network-on-chip architecture for configurable neural networks," *Microprocessors and Microsystems*, vol. 35, no. 2, pp. 152 – 166, 2011.

CHAPTER 7

Conclusions and Future Work

7.1 Introduction

This research has developed EMBRACE, a compact, scalable, modular, hardware SNN architecture as an embedded computing platform. The prototype implementation of the EMBRACE architecture on Xilinx Virtex-6 XC6VLX240T FPGA comprises 448 neurons, 32K synapses and demonstrates reliable, practical embedded classifier and control applications.

7.2 Contributions to SNN Design Challenges

This research has contributed to the development of hardware SNN based embedded computing platform through a structured design methodology comprising high-level simulation, architecture design, FPGA validation and application prototyping. The salient contribution of the research are hardware SNN system design exploration framework, compact and scalable hardware SNN design, NoC architecture for hardware SNNs and modular SNN application prototyping.

7.2.1 Hardware SNN System Design Exploration Framework

This research has developed EMBRACE-SysC, a SystemC-based simulation and performance measurement platform for the analysis of the EMBRACE NoC-based SNN architecture. EMBRACE-SysC incorporates SNN application aware, NoC traffic performance measurement capabilities. EMBRACE-SysC enables exploration of design trade-offs including network topology, routing scheme, latency,

throughput and synapse/neuron ratio. EMBRACE-SysC also comprises GA-based SNN evolution modules that seamlessly integrates with the EMBRACE architecture simulations and allows evolution of the SNN application, easy validation of neuron models and confirms suitability of the overall architecture as an SNN computing platform.

EMBRACE-SysC is a powerful design exploration framework for NoC-based hardware SNN architectures which enables performance analysis of architectural choices at an early system design stage.

7.2.2 Compact and Scalable Hardware SNN Design

This research employs the Modular Neural Network (MNN) computing paradigm for mitigating the synaptic connectivity information storage problem in NoC-based hardware SNN architectures. The novel approach and improved architecture also paves way for tailoring the EMBRACE architecture for embedded modular SNN applications. This thesis presents a novel Modular Neural Tile (MNT) architecture comprising a 16:16 fully connected feed-forward topology SNN structure as neural computing module. The size of the topology memory of the architecture is 50% of that of the previously reported NoC-based hardware monolithic SNN implementation. Successful evolution of benchmark SNN applications confirms the suitability of the proposed MNT for executing application subtasks within a large MNN.

SNN topologies and MNN organisation for practical embedded applications exhibit irregular and random synaptic connectivity patterns. This thesis presents a novel lookup table-based topology memory sharing scheme that provides a flexible number of synaptic connections from MNTs. Overall the area requirement of the architecture is reduced by an average 66% for practical SNN application topologies compared to previously reported EMBRACE architecture. This facilitates accommodation of larger application topologies in the given architectural configuration.

7.2.3 Network on Chip Architecture for Hardware SNNs

This research has analysed the impact of spike transfer latency jitter on SNN information flow in NoC-based hardware SNN architectures. The elaborate analysis helps to understand the effects of spike transfer latency jitter on SNN information distortion under various configurations.

This thesis presents a novel ring topology interconnect for spike communication between neural tiles. The proposed ring interconnect employs a novel timestamped spike broadcast flow control scheme, which offers fixed spike transfer latency for all the synaptic connections within ring. The ring interconnect has been simulated using EMBRACE-SysC. The architecture has been synthesised for 65nm low-power CMOS technology and Xilinx Virtex-6 FPGA. The ring interconnect offers fixed spike transfer latency under various spike traffic densities making it suitable as localised spike communication architecture for hardware SNN architectures and ensures reliable SNN application behaviour.

This thesis further presents a scalable, modular hardware SNN architecture comprising a hierarchical NoC architecture where the ring interconnect is integrated within a mesh topology network of routers.

7.2.4 Modular SNN Application Prototyping

This research presents a systematic modular application design technique for the EMBRACE hardware SNN architecture. Task decomposition for practical embedded applications is presented based on application functionality, robustness, extensibility and various sensory inputs.

This thesis presents the design of a robotic navigational controller partitioned into obstacle avoidance controller and speed and direction manager subtasks. The gradual SNN evolution of the application subtasks and integration tasks results in a successful application behaviour on the EMBRACE-FPGA prototype. The modular application integration mitigates the search space and evolution complexity for the GA based SNN evolution.

The modular application design facilitates implementation of large SNN applications of the EMBRACE hardware SNN architecture and results in faster application evolution through stepwise knowledge integration and simplified SNN training. The application design technique offers a rapid, simple and effective application prototyping for the hardware SNN architectures.

7.3 Future Work

This research contributes to the development of scalable, hardware SNN architecture as an embedded computing platform.

There is potential for further advancements in this research field. The following

potential avenues could further enhance and follow the work presented in this thesis.

7.3.1 Neural Network Topology Evolution

Neural network configuration for an application can be defined as a correct set of synaptic weights, threshold potential and network topology. Traditionally, hardware SNN systems employ fixed topology SNNs, where the network structure and topology is chosen based on the application properties and where the SNN training only evolves the partial SNN configuration comprising synaptic weights and threshold potential.

Evolving the network topology along with other SNN configuration parameters (synaptic weights and threshold potential) can lead to compact, efficient and accurate SNNs [1]. GA based evolutionary search methods have been explored to generate efficient application specific neural network topologies [2]. Similarly, architecture aware evolution of neural network topologies can exploit the architecture specific services of the hardware SNN architecture, such as neuron grouping, availability of localised high density bandwidth, synaptic delays, etc., and can produce efficient SNN topologies and configurations resulting in accurate applications.

7.3.2 Adaptive and Active SNN Training/Learning

Real-life embedded applications often encounter temporary or permanent changes in the operating conditions, and the need to handle unexplored data. Currently mission critical embedded systems are monitored and adjusted for an uninterrupted operation. SNN based computing systems can be potentially employed for such critical applications due to their ability to 'learn' arbitrary functions and provide generalised solution for unexplored input conditions. SNN training/learning algorithms employ known input-output data patterns and/or data correlations to systematically adjust the SNN configuration.

The research in SNN training/learning algorithms for hardware SNN systems is still in its infancy and fails to gracefully handle major changes in application scenarios. This poses a serious limitation on the deployment of hardware SNN based embedded systems for mission critical applications (deep sea exploration, satellite, aviation, automotive, etc.).

Understanding the learning mechanism in human brain biological is one of the fundamental issues in neuroscience. Further research can help in designing biologically plausible, adaptive and active SNN learning algorithms. The key to designing autonomous and adaptive hardware SNN systems is:

- Knowledge integration through SNN system output errors
- Continuous SNN learning through SNN system runtime faults
- Tight integration of SNN system and learning algorithms
- Fault tolerant and distributed SNN learning algorithms

7.3.3 SNN Application Design

Solutions for real-world applications often lack deterministic algorithms. Implementation of such applications on hardware SNN systems pose a serious challenge, since the inherent fuzzy nature of these applications often makes it difficult to define application components and organisation. Although the problem can be mitigated by dividing the application goals into manageable functions, the technique is time consuming and requires human intervention. Currently the application design for hardware SNN systems is predominantly domain specific and lack definitive methodologies.

Automatic design of real-life SNN applications suitable for execution on hardware SNN embedded computing platforms is required for the SNN computing paradigm to succeed in a longer run.

7.3.4 Fault Tolerant Hardware SNN Architectures

As predicted by the Moore's law, the International Technology Roadmap for Semiconductors (ITRS) forecasts shrinking feature sizes for future VLSI implementation technologies [3]. The ITRS roadmap estimates the MOS transistor size to be 6nm and metal line widths between 6nm to 140nm by year 2026. Shrinking silicon geometries will allow billions of transistors to be fabricated on a single die. However, at such small geometries the device, fabrication and aging faults can results in low manufacturing yield and failure of the overall chip. The major faults envisaged are:

- Open metal lines due to manufacturing (lithography) faults
- Open metal lines due to electro-migration (aging) faults
- MOS transistor short circuit due to tunneling faults
- Signal distortion due to crosstalk

ITRS encourages research towards the design of fault tolerant SoC architectures to mitigate the effects of these faults, increase usability of the chip and graceful degradation of the circuit operation. Due to their distributed architecture and inherent adaptiveness, SNNs have potential to overcome minor system faults. However, faults on critical neural circuits and synaptic junctions can result in failure of hardware SNNs. Architectural techniques for identification of faulty circuits and reallocation of the computing logic to redundant components can gracefully handle these faults and result in fault-tolerant hardware SNN architectures in silicon.

References

[1] K. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[2] M. O'Halloran, S. Cawley, B. McGinley, R. C. Conceicao, F. Morgan, E. Jones, and M. Glavin, "Evolving spiking neural network topologies for breast cancer classification in a dielectrically heterogeneous breast," *Progress In Electromagnetics Research Letters*, vol. 25, pp. 153–162, 2011.

[3] International technology roadmap for semiconductors, 2012. [Online]. Available: http://www.itrs.net/

श्री कृष्णार्पणमस्तु