AstroByte:
Programmable and Scalable Multi-FPGA Interconnect infrastructure for Accelerated Simulations of Self-Repairing Spiking Astrocyte Neural Networks

Shvan Haji Karim

Faculty of Computing, Engineering and the Built Environment

A thesis submitted for the degree of Doctor of Philosophy (PhD)

April 2020

"I confirm that the word count of this thesis is less than 100,000 word”

Before fire-worshiping, before being Muslims,

Whether being captives or free,

We have always been ***Kurds*** and we will forever remain so

Ibrahim Ahmad

Table of contents

Acknowledgement

I would like to genuinely thank Ulster University and the EPSRC funding council grants (EP/N00714X/1 & EP/N007050/1) for providing me with this opportunity to study a PhD degree through the Vice Chancellor's Research Scholarship (VCRS).

My special thanks go to my PhD supervisors, Dr Jim Harkin, Prof. Liam McDaid and Dr Bryan Gardener, for their advice and guidance during the period of this doctoral research.

I also thank Dr John Wade for his help with the original astrocyte model code and his guidance at the start of the PhD program. Likewise, I thank the other participants of the SPANNER project. Dr Junxiu Lui (Ulster University), Dr Andrew Tyrrell, Dr David Halliday, Dr Jon Timmis, Dr Anju Johnson and Dr Allan Millard (University of York).

Similarly, I thank all the staff at ISRC, Ulster University Doctoral College, Ulster University Research Office and throughout Ulster University generally. Their efforts helped the PhD program proceed smoothly.

My deepest gratitude goes to my family, my mother Dlxwaz, my sisters Shaee and Shang, and my father Ali, whose soul will always be with us. It was because of him I learnt dedication in life, love for science and love for humanity.

Finally, I truly thank the President of Komar University Dr Salahalddin Saeed Ali, the head of Ulster University Doctoral College Professor Alison Gallagher, Senior lecturer at Lewis School of English Mr Andrew Bourner and Engineer Musaddaq Saeed for their kindness, encouragement and support whenever was necessary.

ABSTRACT

The human brain is one of the most complex systems known to mankind with an estimated 100 billion neurons, 0.15 quadrillion (150 thousand billion) synapses and around a trillion glial cells. Understanding this intricate organism has long been the objective of many scientists and researchers. The motivations for comprehending the human brain can be out of sheer curiosity, to cure diseases (e.g. Alzheimer's and depression) or to build efficient brain-inspired computers. More recently, progress has been achieved in modelling the role of glia cells, a cell that was not previously recognised as a key player in brain repair. To this end, mathematical models have been developed for predicting the behaviour of cells (neuronal and glia) under stimuli. The models range in complexity from highly rich in detail such as Hodgkin-Huxley to simpler models like Leaky-integrate and Fire (LIF). An artificial neural network is a grouping of neurons in a certain structure such as feed forward. Spiking Neural Networks (SNNs) are the third generation of neural networks and employ the temporal computing abilities of the human brain. Programming languages such as Matlab or Pynn are used to simulate SNNs however, simulations exhibit signification execution times for large networks. The inclusion of astrocytes, a type of glia cell, - in an SNN provides Spiking Astrocyte Neural Network (SANN). Due to the high computational and complexity of astrocytes, SANNs suffer from even further increased execution times.

This PhD research addresses the issue of prolonged simulation times by utilizing dedicated hardware on FPGAs for accelerating SANN simulations. Astrocytes are incorporated into the simulated models to give the SANN the ability to self-repair. Since FPGAs have limited resources, a NoC-based multi-FPGA infrastructure is developed to accommodate SANNs with resource demands exceeding a single FPGA. On top of that, a monitoring and configuration platform is implemented to configure various aspects of the network at the start of operation and to take data off-chip for storage and analysis on a PC during simulations.

The following points summarize the major contributions of this study;

1- Developing a new 32-bit fixed-point hardware prototype for biologically faithful astrocyte model.
2- Incorporating the astrocyte prototype with hardware models of neurons and synapses to facilitate a self-repairing FPGA-based SANN Accelerator (FSA).
3- Designing a novel NoC router, NoC infrastructure and data format to facilitate scalable a multi-FPGA NoC AstroByte platform.
4- A novel FPGA Configuration and Monitoring Platform (FCMP) was utilized for injecting faults, configuring the AstroByte platform and capturing real time simulation data for monitoring and analysis on a PC.

List of Figures

List of Tables

List of acronyms and abbreviations

2-AG	2-Arachidonoylglycerol
AER	Address Event Representation
AG	Address Generators
ANN	Artificial Neural Network
ASIC	Application Specific Integrated Circuits
ATT	Arbiter Track Table
ATT FIFO	Arbiter Track Table FIFO
BCM	Bienenstock, Cooper, and Munro
BFT	Butterfly Fat Tree
BSTDP	BCM-STDP
CB1Rs	Type 1 Cannabinoid Receptors
CDR	Clock Data Recovery
CE	Computing Element
CEI	Computing Element Interface
CICR	Calcium Induced Calcium Release
CNN	Convolutional Neural Network

CPU	Central Processing Unit
DCFIFO	Dual Clock First In First Out
DDR DRAM	Double Data Rate DRAM
DOR	Dimension Order Routing
DRAM	Dynamic RAM
DSE	Depolarization Induced Suppression of Excitation
DSP	Digital Signal Processor
e-SP	Synaptic Potentiation
EDA	Electronics Design Automation
ER	Endoplasmic Reticulum
Fault CNTRL	Fault controller
FC	Frequency Calculators
FCMP	FPGA Configuration and Monitoring Platform
FI	FCMP Interface
Flit	Flow Control Unit
FPGA	Field Programmable Gate Array
FSA	FPGA SANN Accelerator
GA	Genetic Algorithm
Gbps	Giga bits per second
GPU	Graphics Processing Unit
GXB	Gigabit Transceiver Block
H-NoC	Hierarchical NoC
HANA	Hierarchical Astrocyte Network
HDL	Hardware Description Language
HH	Hodgkin-Huxley
HSMC	High Speed Mezzanine Card
HMI	Human Machine Interface
HLS	High Level Synthesis
IC	Integrated Circuits
ICR	Input Controller

ICTT	Input Channel Track Table
ICTT FIFO	Input Channel Track Table FIFO
InCh FIFO	Input channel FIFO
IP	Intellectual Property
IP3	Inositol trisphosphate
ISA	Instruction Set Architecture
ISC	Input Stream Controller
ISRC	Intelligence Systems Research Centre
L-Ca_v	L-type Voltage Gated Calcium
LAM	Loihi Astrocyte Module
LFSR	Linear Feedback Shift Register
LIF	Leaky Integrate and Fire
LPDDR	Low Power Double Data Rate DRAM
LUT	Look Up Table
mGlu	Metabotropic Glutamate
mGluR	Metabotropic Glutamate Receptor
NGN	Neuron-Glial Network
NMDA	N-methyl-D-aspartate
NoC	Networks-on-Chip
OC	Output Controller
OSC	Output stream controller
OutCh FIFO	Output channel FIFO
PAXY	Pseudo Adaptive XY Routing
PC	Personal Computer
PR	Probability of Release
PRNG	Pseudo Random Number Generator
RAC	Reward-Attention Coupled
RAM	Random Access Memory
RdCNTRL	Read Controller
RDC	Router Data Controller

RE	Routing Engine
RTL	Register Transfer Level
SANN	Spiking Astrocyte Neural Networks
SATA	Serial Advanced Technology Attachment
SCM	Single Constant Multiply
SDK	Software Development Kit
SDRAM	Synchronous DRAM
SERCA	Sarco-Endoplasmic-Reticulum
SI	SANN Interface
SM	State Machine
SNN	Spiking Neural Networks
SoC	System on Chip
SRAM	Static RAM
STDP	Spike Timing Dependent Plasticity
SXYR	Surrounding XY Routing
TDM	Time Division Multiplexing
TRNG	True Random Number Generator
UART	Universal Asynchronous Receiver/Transmitte
VHDL	Very High-Speed Integrated Circuit HDL
WrCNTRL	Write Controller
Xbar	Crossbar

Chapter 1: Introduction

In this chapter, a brief background will be provided regarding the major scientific and engineering concepts that have been utilized in the PhD research. The key objectives, motivation and challenges of the research will also be presented.

1.1 Background

This research project is multi-disciplinary in nature incorporating the fields of brain-inspired computing, FPGA hardware and Networks-on-Chip (NoC) strategies. The major motivation behind this study is leveraging the flexibility provided by FPGA devices to reduce the simulation times of large SANNs to enable carrying out these simulations in a time-efficient manner. The PhD addresses the challenge of scalable acceleration of SANN applications on FPGA hardware. Achieving large-scale parallel simulations of such systems introduces two major challenges. First, establishing dedicated hardware that faithfully mimic complex biological processes, such as astrocytes, requires a significant amount of hardware resources [1], meaning that the design has to incorporate multiple FPGAs. Second, the challenge of communicating both spike event (neuron data) and numeric (astrocyte data) across significant interconnect pathways between astrocytes and neurons, requires an appropriate hardware communication mechanism. The NoC interconnection paradigm is a communication mechanism suitable for tackling the interconnect challenge. NoCs allows for massive parallel structures in hardware that facilitate a great number of computing nodes, while maintaining throughput performance under node scaling conditions [2], [3].

The following sub-sections provide a brief introduction for explaining Brain-inspired Computing, NoC paradigm and FPGAs.

- **Brain-inspired Computing**

In traditional Von Neumann computer architectures, programs are stored in a memory and are fetched and executed sequentially by a central processing unit This type of computers have functioned well so far but are facing major challenge in terms of power consumption and scalability. Brain-inspired Computing is seen as a potential replacement for traditional Von Neumann architectures[4]. This new trend in research is motivated by the fact that biology, in particular the human brain, is able to perform computations more efficiently in terms of power consumption, more robust to failures, and in a massively parallel manner. lately, SNNs have been seen as a modern way to create such brain-like computing systems [5].

The ability of the human brain to perform self-repair is another significant feature that electronic systems designers are keen to mimic in the next generation of computer systems. It has been put forward lately that a specific

type of glial cell, namely the Astrocyte, is a key player in self-repair as it is now believed they have an important role in modulating synaptic activity by mediating direct and indirect feedback to presynaptic and postsynaptic neurons [6], [7]. The interactions between astrocyte and neuron cells facilitates a distributed self-repair capability.

Astrocytes enwrap groups of tripartite synapses and therefore are able to communicate with presynaptic and postsynaptic neurons at synaptic junctions [6], [8]–[10]. The SANN [11] is the addition of astrocyte cells in the current SNN paradigm. Because of the astrocyte inclusion, SANNs introduce significant computational complexity along with higher exchanges of data. Previous work at Ulster has developed a small hardware demonstrator of a spiking astrocyte-neuron network [12], [13], however, progress needs to be made in developing a method of accelerating the simulation of large-scale SANNs in hardware.

- **Networks-on-Chip**

Networks-on-Chip is an interconnection centric paradigm with a scalable infrastructure that can host a wide variety of subsystems and Intellectual properties (IPs). NoC is the answer for current bus-based architecture bottlenecks in terms of scalability, performance and efficiency. As the number of Computing Elements (CE)s are ever increasing in modern computing systems, sharing a single bus introduces many problems [2], [14]–[17]. NoC replaces highly dense point to point connections with an internet like structure.

- **FPGAs**

Field Programmable Gate Arrays (FPGAs) are reconfigurable hardware that through a programmable interconnection structure and Look Up tables (LUTs) give the designers the ability of changing the implemented logical function when necessary [18]–[20]. This flexibility has made FPGA extremely convenient for use in research environments and in applications where hardware reconfigurability is an advantage. Modern FPGAs can work at frequencies of over 500MHz with complete digital systems now able to reside on the largest, high-end devices. Furthermore, beside the reconfigurable elements, modern FPGAs contain specialized elements such as Digital Signal Processors (DSPs), embedded processors and high-speed transceivers. Therefore, FPGAs have found their way into many industrial applications such as aerospace, ASIC prototyping and automotive [21].

1.2 Motivation

Prior work within the research team at Ulster has demonstrated the astrocyte-enabled brain-inspired self-repair principle in hardware using floating-point implementations [12]. Implementing such SANNs in hardware allows for future computing tasks to self-repair in the presence of hardware failure; thereby providing high degrees of reliability. One key focus and contribution of this work is the feasibility of implementing such mechanisms on FPGAs using reduced

fixed-point representation, where biologically realistic accuracy can still be maintained. In this context, current research in self-repair has focused on astrocytes, which is the mechanism responsible for facilitating fine-grained self-repair. These SANNs modulate the synaptic activities between neurons via distributed astrocytes in the network. This concept was proven in previous work when an astrocyte was integrated with an SNN [6]. Due to the complex mathematical nature of the astrocyte model in previous work [6], simulating the SANN using tools such as Matlab required long execution times, in particular as the astrocyte is not event based like traditional SNNs and operates over longer biological timescales of hundreds of seconds. This motivated the need for designing an FPGA-based SANN Accelerator (FSA) platform for accelerating simulations of SANNs through implementing dedicated astrocyte, neuron and synapse hardware models on FPGAs [1].

To facilitate experimentation, a framework was required to enable fault injection, spike/astrocyte data recording and visualization of FPGA-based SANNs. Current on-chip mechanisms such as Intel SignalTap II Logic Analyzer are not adequate as their capacity is limited by the amount of on-chip memory afforded by the FPGA.

This is the motivation for designing a platform that enables configuration of the accelerator SANN- the FSA, injecting faults and sending monitoring data to a PC for analysis and storage. The **F**PGA – based **C**onfiguration and **M**onitoring **P**latform (FCMP) forms the second contribution of this research and is used for injecting configuration packets at the start and then collecting monitoring packets during the FSA running time.

Accelerating simulations of massive SNNs necessitates dedicating significant FPGA resources for their implementations. The inclusion of astrocytes, especially biologically faithful ones as in this work, into spiking neural networks, dramatically increases FPGA hardware usage. Furthermore, the communication demands increase as two types of data, namely spikes (from neurons) and astrocyte data, have to be exchanged. Thus, accelerating large-scale SANNs calls for novel approaches beyond single-FPGA shared bus utilizations. This interesting challenge has motivated the third contribution of this work, which is designing a programmable multi-FPGA NoC-based infrastructure that is capable of accommodating large numbers of astrocytes, neurons and tripartite synapses and the two-fold communications requirements.

AstroByte, illustrated in Figure 1:1 is a single platform that integrates all the above-mentioned contributions. It uses the FSA as its computing element, FCMP as its core component for injecting configuration packets and collecting data for analysis, and the multi-FPGA NoC structure for providing data communication infrastructure for massive parallel SANNs. AstroByte can have any number of nodes, Figure 1:1 shows an example 3x3 AstroByte platform.

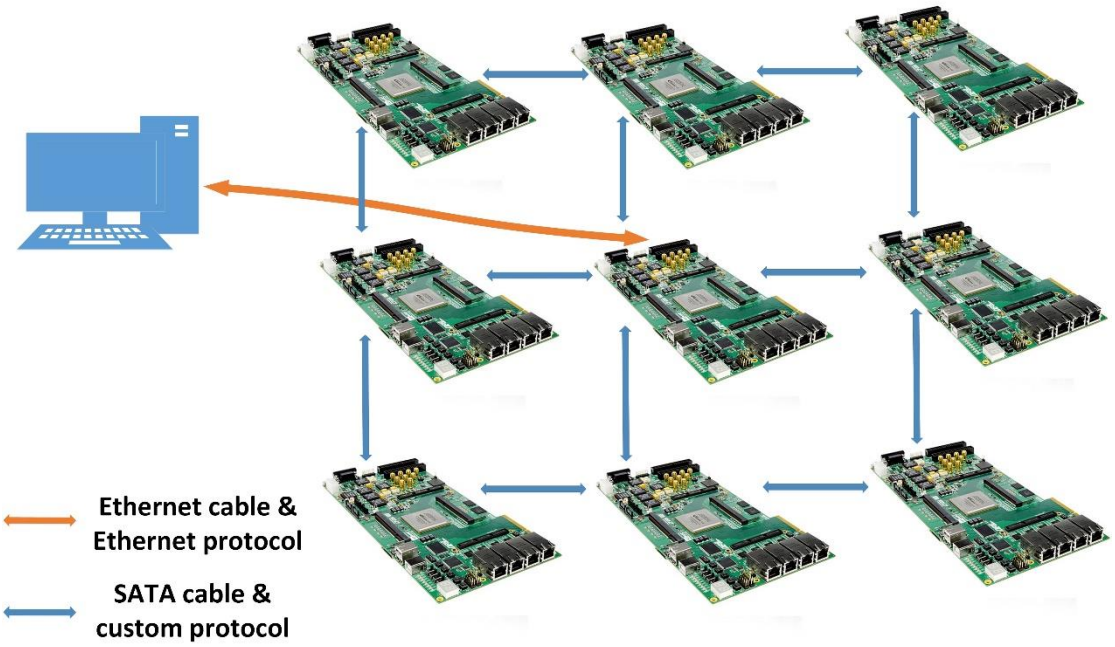


Figure 1:1 AstroByte platform

1.3 Thesis Hypothesis and Contributions

The main hypothesis of this research is that dedicated FPGA hardware can accelerate SANN simulation. The main task of this research is to prove that both spikes, which are discrete in nature, and astrocyte data, which is continues, can be generated concurrently in the same system. Furthermore, this work proves that efficient interconnection mechanism and data format can make a NoC to support both continues and discrete data exchange. Additionally, this study acts as evidence that despite NoC interconnection and data monitoring overheads, a multi-FPGA SANN platform can retain functional accuracy while providing up to x188 speedup when compared to an equivalent software model of the SANN. As will be become clearer in chapter 2, there is a gap in the literature regarding large-scale implementation of self-repairing SANNs that incorporate both continues and discrete computations for self-repair.

Below is the list of the major scientific and technical contributions springing from this research.

1- A novel fixed-point hardware prototype for a biologically faithful astrocyte model [6] was developed with an appropriate numerical representation type and bit resolution. Results demonstrate that the astrocyte accuracy is comparable to equivalent software implementations. This contribution allows for building SANN FPGA Accelerators with high accuracy while reducing hardware footprint when compared to floating point implementations.

2- The new astrocyte hardware was Integrated with neurons and synapses to facilitate an FPGA-based SANN Accelerator (FSA). Results indicate that accelerations of more than x1000 has been achieved and high accuracy was maintained in terms of neurons firing frequency when compared to equivalent Matlab models. This proves that FPGA accelerations can save researchers in the field of computational brain modelling a great deal of time.
3- A novel FPGA Configuration and Monitoring Platform (FCMP) was utilized for injecting faults, configuring the AstroByte platform and capturing real time simulation data for monitoring and analysis on a PC. The FCMP helps bypassing the limitations of data collection of tools like SignalTap II in terms of the amount of data that can be monitored. Additionally, through the FCMP platform the user can change some attributes of the FPGA design without re-synthesizing the design, allowing for more time-efficient simulations.
4- New controllers for high-speed serial transceivers to allow transmission of data between FPGA boards were developed, making way for large scale, highly parallel multi-FPGA SANN implementations.
5- A new NoC router was developing that can function with the transceiver controllers to realize scalable multi-FPGA NoC infrastructure.
6- Novel protocols were designed for ensuring integrity of data transformation between FPGA boards, as well as the NoC router and AstroByte components. This allows for maintaining the accuracy and integrity of simulations while large scale SANNs are mapped to the multi-FPGA NoC structure.

1.4 Thesis Outline

Chapter 2 provides a literature review and starts by discussing neuron models and then moves onto investigating astrocyte and tri-partite synapse models. The chapter summarizes the significant works regarding astrocyte hardware utilizations, both in-house models and others, together with the interconnection methods for astrocytic signalling in hardware. In addition, the main literature regarding SNNs are investigated and the most state of the art in the domain of ASIC implementations of SNNs are reviewed.

Chapter 3 starts with introducing a novel FPGA SANN Accelerator (FSA) that has a biologically faithful astrocyte at its core, forming the first contribution of this work. A reduced fixed-point representation of the astrocyte process is implemented and discussed in detail. To strike a balance between FPGA resource usage and accuracy, different bit resolutions are tried and the appropriate one is chosen. In addition to that, the ability of FSA to perform self-repair when exposed to various levels of faults is proved. Furthermore, high accuracy is maintained when weighed up against an identical Matlab model. In addition, the chapter provide comparisons between acceleration due to utilizing a SANN model on dedicated FPGA hardware and a similar Matlab implementation.

The second contribution of this research is a configuration and monitoring platform, called FCMP. The chapter evaluates the architecture and operation of FCMP in detail by providing thorough descriptions regarding each of FCMP components. The operation of the platform and the interactions between a Nios II embedded processor system and custom hardware are analysed. Data integrity is assessed by comparing data collected using Intel SignalTap II and the FCMP. The new acceleration data from the result of integrating FCMP with the SANN hardware are presented as well. Chapter 3 also examines methods to enhance FCMP performance and overall acceleration along with showing new acceleration figures as the result of these improvements.

Chapter 4 presents the AstroByte platform's multi-FPGA NoC infrastructure by means of detailed explanation of the AstroByte NoC data format, router architecture, operation and the logic blocks forming the router. Additionally, AstroByte programmability and the design choices that allow for this flexibility are investigated. Furthermore, the FCMP Interface (FI) and the SANN Interface (SI) blocks that allow smooth communication between the NoC router and the FSA or the FCMP are discussed. FCMP from *Chapter* 3 is adapted for proper interfacing with the router through FI.

Chapter 5 presents results and the experiment setup that was used throughout. The chapter introduces the example 9-FPGA AstroByte platform configuration used for collecting results on throughput and latency. Additionally, a prototype AstroByte platform is used to test the features in terms of simulation, under-sampling and data acquisition. A subsequent section deals with integrating the SANN accelerator (FSA) on a 2x2 FPGA AstroByte configuration. Figures regarding speedup factors, at different under-sampling rates, and accuracy are showcased. In addition, AstroByte is compared against two other (relatively) similar platforms, SNAVA and Bluehive, in regard to features and capabilities.

Chapter 6 is the thesis conclusion which includes a section discussing future works and another for self-critic.

1.5 Publications

The list of the outcome publications from this work are given below:

Papers I have published as the first author:

1- S. Karim et al., "FPGA-based Fault-injection and Data Acquisition of Self-repairing Spiking Neural Network Hardware," in 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018, pp. 1–5.

(The work originates from Chapter 3)

2- S. Karim et al., "Assessing Self-Repair on FPGAs with Biologically Realistic Astrocyte-Neuron Networks," in Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI, 2017, vol. 2017-July, pp. 421–426.

(The work originates from Chapter 3)

3- S. Karim et al., “AstroByte: A multi-FPGA Architecture for Accelerated Simulations of Fault-tolerant Spiking Astrocyte-Neuron Networks,” in Proceedings of IEEE Design, Automation & Test in Europe, DATE, 2020 (Accepted- a virtual conference is planed pending confirmation of the date).

(The work originates from Chapters 4 and 5)

Other papers in which I contributed to include:

4- A. P. Johnson et al., “Fault-Tolerant Learning in Spiking Astrocyte-Neural Networks on FPGAs,” in 2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID), 2018, pp. 49–54.

5- J. Liu et al., “Self-repairing Learning Rule for Spiking Astrocyte-Neuron Networks,” Lecture Notes in Computer Science, Springer 2017, pp. 384–392.

6- A. P. Johnson et al., “Homeostatic Fault Tolerance in Spiking Neural Networks: A Dynamic Hardware Perspective,” IEEE Trans. Circuits Systems I: Regular. Paper, vol. 65, no. 2, pp. 687–699, Feb. 2018.

7- A. P. Johnson et al., “Homeostatic fault tolerance in spiking neural networks utilizing dynamic partial reconfiguration of FPGAs,” in 2017 International Conference on Field Programmable Technology (ICFPT), 2017, pp. 195–198.

8- J. Liu et al., “Self-repairing Learning Rule for Spiking Astrocyte-Neuron Networks,” Springer, Cham, 2017, pp. 384–392.

9- A. P. Johnson et al., “An FPGA-based hardware-efficient fault-tolerant astrocyte-neuron network,” in 2016 IEEE Symposium Series on Computational Intelligence (SSCI), 2016, pp. 1–8

Chapter 2: Literature Review

This chapter outlines the most important and relevant literature in the domain of this work. The main models available for both neurons and astrocytes will be discussed. The review will highlight the dynamics and equations of the astrocyte model that enables SNNs to perform self-repair. Current research progress in FPGA implementations of SNNs will be reviewed along with the state of the art in astrocytes hardware implementations. In addition, the chapter reviews efforts in establishing astrocyte communications using the NoC paradigm. Particular attention will be given to the SNN hardware architectures that are comparable to the multi-FPGA architecture implemented in this work.

2.1 Neuron Models

As a vital component for both understanding and modelling the human brain along with construction of neural networks, quantifying neurons behaviour has been the subject of intensive research during the last six decades. Rosenblatt's perceptron model [22] is considered to be the first neuron model that has the ability of forming a neural networks to perform semi-autonomous tasks through supervised learning. Since then, the field of neuron modelling and neural networks has gained much attention in the scientific community for various purposes e.g. machine learning, computational modelling of the brain and power efficient computing.

As neurons are the fundamental elements in neural networks, this section discusses the most widely used models available in the literature.

2.1.1 Hodgkin-Huxley (HH) Model

This is one of the most famous models as regards quantitatively reproducing neural action potentials [23]. This model takes into consideration the dependency of sodium, potassium and leak current on the membrane potential. Results gained from experiments show that this model can accurately represent action potentials together with the membrane voltage threshold and refractory periods in biology cells. The current through the cell membrane is given by

$$I = Cm\frac{dVm}{dt} + \bar{g}_K n^4(V_m - V_K) + \bar{g}_{Na}\, m^3 h(V_m - V_{Na}) + \bar{g}_l(V_m - V_l) \quad (1)$$

where

$$\frac{dn}{dt} = \propto_n (V_m)(1-n) - \beta_n(V_m)n \quad (2)$$

$$\frac{dm}{dt} = \propto_m (V_m)(1-m) - \beta_m(V_m)m \quad (3)$$

$$\frac{dh}{dt} = \propto_h (V_m)(1-h) - \beta_h(V_m)h \quad (4)$$

Where *I* is the total current of the membrane, C_m represents the membrane capacitance, $\bar{g}_k$, $\bar{g}_{Na}$ and $\bar{g}_L$ are potassium, sodium and leak maximum conductance, respectively. All the above-mentioned variables are per unit area. V_l, V_K *and* V_{Na} are leak, potassium and sodium reversal potentials respectively. V_m is the membrane potential while $\propto_x$ *and* β_x are rate constants belonging to ion channel *x*. *n and m* are dimensionless quantities with values of either 1 or 0 relating to activation of potassium and sodium channels correspondingly and *h* is of a similar nature as *n and m* but associated with inactivation of sodium channels. The HH model reproduces biological properties of membrane cells faithfully. However, it requires a great deal of computational power and for this reason simpler neuron models are generally used for modelling SNNs.

2.1.2 Pinsky-Rinzel Model

This is a non-linear two compartment neuron model in which the soma and dendrite are modelled in two separate compartments [24]. The equations of this model are as follows: -

$$Cm\frac{dV_s}{dt} = -\bar{g}_L(V_s - E_L) - g_{Na}(V_s - E_{Na}) - g_{DR}(V_s - E_K) + \frac{g_c}{P}(V_d - V_s) + \frac{Is}{P} \quad (5)$$

$$Cm\frac{dV_d}{dt} = -\bar{g}_L(V_d - E_L) - g_{Ca}(V_d - E_{Ca}) - g_{AHP}(V_d - E_K) - g_C(V_d - E_K) + \frac{g_c}{1-P}(V_s - V_d) + \frac{I_{Syn}}{1-p} \quad (6)$$

Where I_s is the current applied to soma and *p* is the percentage area occupied by the soma. E_L, E_{Ca}, E_{Na} and E_K are respective leakage, Calcium, Sodium and Potassium equilibrium (reverse) potentials. $\bar{g}_L$, g_{Na}, g_{DR} g_{Ca} , g_{AHP} and g_c are leakage, Sodium, Potassium delayed rectifier, Calcium, Potassium (afterhyperpolarization) and coupling conductance, respectively. I_{Syn} is the synaptic current. Although the model uses only two compartments, it can replicate a wide range of realistic activity patterns that are created as a result of accurate injection of current through the soma or dendrite compartments [25].

2.1.3 FitzHugh-Nagumo Model

This model is a simplified form of the HH model. Richard Fitzhugh reduced the four equations describing the HH model to a two-dimensional form by making several observations and reductions regarding the gating variables *n, m and h.* The final equations adopted by this model are given below:

$$\frac{dv}{dt} = v(v - \propto)(1 - v) - w + I \quad (7)$$

$$\frac{dw}{dt} = \varepsilon(v - \gamma w) \quad (8)$$

Where *v* represented the potential (fast variable) and *w* is the slow variable (gating variable of sodium). Other symbols, namely $\propto, \varepsilon$ and γ are constants

[26]. The FitzHugh-Nagumo model is also referred to as a single compartment model [25].

2.1.4 Leaky Integrate and Fire Neurons

The Leaky Integrate-and-Fire (LIF) neuron is capable of reproducing firing of spikes (action potentials) when the membrane potentials level passes a certain threshold. This model can be represented by means of an RC circuit as shown in Figure 2:1.

$$C_m \frac{dV}{dt} = -\frac{Vi - E_m}{R_m} + I \tag{9}$$

Where R_m represents the membrane resistance, C_m is the membrane capacitance and I is the cellular current. Other variants of integrate-and-fire neurons also exist, such as Quadratic integrate and fire model [25].

This study uses this model for accelerating SANN simulations with FPGAs for two reasons. Mainly, despite its simplicity, LIF model can reproduce the basic characteristics of neurons that are necessary for SNN implementation in an efficient manner. In fact, LIF model is used widely in SNN literature because the model does not require a lot of computational resources. For this work, the characteristics that LIF can reproduce is enough for assessing the AstroByte platform and its capabilities in terms of acceleration, accuracy etc, nullifying the need of using more area and power-hungry models. Additionally, the hardware implementation of SANN in this study will be compared against an equivalent in-house built Matlab model for assessing accuracy and speed of Astrobyte. The Matlab model implements LIF model, necessitating choosing LIF for the equivalent hardware model to permit comparisons between the two models. An alternative approach would be changing the neuron model both in the Matlab model and AstroByte, however one can't justify spending such extra effort giving that the LIF model can adequately simulate self-repairing SANNs.

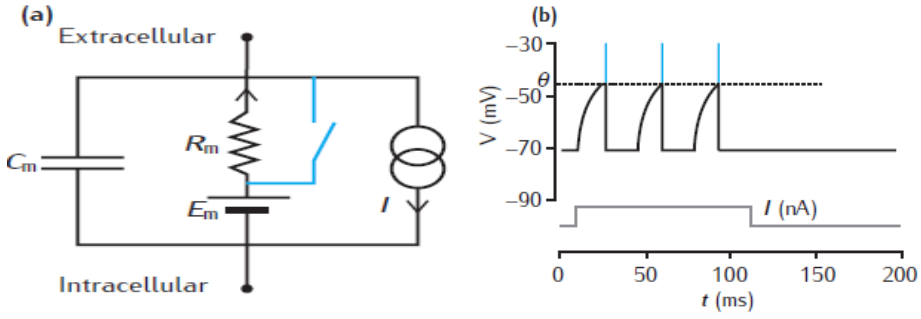


Figure 2:1: LIF Circuit model (a) and response (b) [25]

.

2.2 Astrocyte and Tripartite Synapse

Astrocytes are a type of glial cell found in the human brain in large quantities. It is estimated that their numbers outweigh that of neurons by 10:1 and amass around 25%-50% of the total brain volume. Contrary to neurons, astrocytes do not have axons and are unable to output action potentials. Despite the fact that the existence of astrocytes has been known for a long time, their functionality was not understood as they were thought to only provide structural support to neurons. Recent findings, however, indicate that astrocytes contribute to synaptic activity modelling and metabolic interactions with neurons [6], [27]. Astrocytes are able to interact with neurons via the tripartite synapse. At a tripartite synapse junction, an astrocyte process surrounds both presynaptic and postsynaptic terminals (Figure 2:2) enabling the astrocytes to monitor and regulate synaptic activity between the presynaptic and postsynaptic neurons. Moreover, astrocytes are able to communicate to each other through propagation of IP3 (Inositol trisphosphate) and calcium waves [6], [9], [28], [29]. Although there is a form of consensus in the literature regarding the fact that astrocytes observe and modify synaptic activity, the exact mechanisms of this process are interpreted differently in the available publications. In the following sections, we study some of these models.

2.2.1 Astrocyte Models

The Gatekeeper model [30] views the astrocyte as a gatekeeper that controls the spiking rate of the presynaptic terminals that fire constantly and frequently. Astrocyte Intracellular calcium levels are affected by synaptic activity since Metabotropic Glutamate (mGlu) receptors located on astrocytes can sense such activities and lead to an increase in the concentrations of IP3. The Gatekeeper model uses the Li-Rinzel model in which IP3 levels increase upon release of neurotransmitters and causes IP3 mediated calcium release in the astrocyte cell [31].

In addition to the Gatekeeper model, other astrocyte formulae regarding their impacts on both presynaptic and postsynaptic neurons can be found in the literature. A model developed by Nadkarni and Jung [32] has expressed astrocytes as synaptic activity enhancers, in contrast to the Gatekeeper model discussed above. In this model astrocyte activity results in synaptic potentiation. Here, glutamate emitted by astrocytes balances calcium in the presynaptic axon. The source of this calcium is different from that released as a result of the application of a voltage to the presynaptic terminal since it correlates with the astrocyte in terms of timescale. The overall effect is improving the probability of neurotransmitter release following an action potential.

Other astrocyte models have focussed on the function of astrocytes in neural networks. Guimarães et al [33] has investigated using astrocyte for stimulating the impact of nicotine on attentional focus. The upshots of tripartite synapses absorbing glutamate imitated from astrocyte is investigated. Also, the research

addresses the significant capacity of astrocytes to regulate neural response throughout a Reward-Attention Coupled (RAC) circuit [34]. The RAC combines thalamocortical and reward circuitry to look into how nicotine affects attention focus. To this effect, a mathematical model has been developed to represent the mechanism of interactions between the reward and thalamocortical circuits. Simulations of the mathematical model have been carried out that show the interactions between an astrocyte and the RAC circuit and vice versa. The authors have come to conclusion that astrocytes enhance neural network transmissions. Deemyad et al [35] has investigated the relationship of astrocytic calcium activity with neuronal barrage firing [36], [37]. The authors have reported two main evidences for this phenomenon. Firstly, they have reported that barrage firing can still be observed in mice despite lacking necessary units for inducing electrical signals between neurons. The authors see this as evidence that astrocytes, through numerous connections at synaptic gap, could be the initiators of barrage firing. Secondly, inhibiting of barrage firing can be observed by deconcentrating extracellular calcium or closing L-Ca_v (L-type voltage gated calcium) channels. In contrary, buffering calcium in interneurons does not inhibit barrage firing. The authors have summarized that mechanisms of neural networks, both special and temporal, are being influenced by astrocytes through direct integration of neuronal activity and driving barrage firing some populations of inhibitory interneurons.

Astrocytes have also been used to increase firing rate at tripartite synapses. In Abed et al [38], the internal calcium pool of the astrocyte is being excited by two mechanisms. The first one is a fast mechanism that is initiated by depolarization in postsynaptic neuron. The second one is a slow mechanism which is initiated by presynaptic diffusion. Both these values add to calcium volumes inside the astrocyte. Once a threshold is reached, an astrocyte mediator is emitted that affects the synaptic terminal and increases the possibility of firing in the postsynaptic terminal. Improved firing rate has also been investigated in a theoretical study of the reasons behind unusual (extremely high or low) levels of gliotransmitters at synaptic connections exists [39]. The aim is to remotely keep gliotransmitters concentration levels by means of improving performance of the molecular communications that facilitate Ca^{2+} signalling.

Other publications have integrated astrocytes into neural networks for improving the networks ability to solve classification problems. In one of such study, an artificial neuron-glial network (NGN) has been developed and used for solving classification problems [40]. The astrocyte in this model is stimulated when the associated synaptic junction experiences neural activity for a minimum of *x* out of *y* iterations (4 out of 6, for example). When the astrocyte is active the synaptic weights increase by 25%, otherwise they decrease by 50%. The authors report that the NGN performance is significantly higher than that of traditional neural networks. Furthermore, astrocytes have been used in artificial NGNs to automate the generation of the values that are usually considered as constant parameters in neuron-glial models. Each neuron is connected to an

astrocyte in this model and improvements have been reported over traditional Artificial Neural Networks (ANNs) and other NGNs in terms of solving classification problems [41].

This section discussed the most prominent astrocyte models (e.g. The Gatekeeper and Nadkarni and Jung models). Further models of astrocytes were discussed to cite some of the usages of astrocytes in the literature. Finally, two review articles will be cited that contain more models of astrocytes, their calcium dynamics and their interactions with neurons [42], [43].

- **SPANNER & In-house Models**

The SPANNER project explored the association between astrocytes and neurons to develop the next generation of algorithms and hardware with the capability of fine-grain fault-tolerant systems [44].

This section discusses the state-of-the-art research in astrocyte modelling and their internal dynamics that have been developed at *Intelligent Systems Research Centre* (ISRC).

Liu et al [45] proposes a biophysical model for reproducing presynaptic and postsynaptic neurons, A Gamma AminoButyric Acid (GABA) interneuron and an astrocyte. According to this model, astrocytic release of calcium from IP3 pathways is triggered by GABA produced by GABA interneurons. In turn, the presynaptic transmission probability rate increases which leads to weight potentiation as well as an incremental uptake in postsynaptic spiking activity before stabilising. The authors have proven that IP3-probability rate interactions cause burst-firing at postsynaptic neurons before the initial phase of weight potentiation.

Moreover, an astrocyte model with BSTDP learning rule has been reported [46]. It combines Spike Timing Dependent Plasticity (STDP) [47] and Bienenstock, Cooper, and Munro (BCM) [48] rules (hence the name BSTDP). In this model, postsynaptic neuron activity participates in potentiation of synaptic weight along with temporal dissimilarity between presynaptic and postsynaptic neurons firing times. It is proven that the height of plasticity window is governed by BSTDP that results in the formation of mapping of inputs to outputs during the learning phase as well as keeping the mapping through self-repair in case synaptic pathways become defective. The real-time self-repairing capabilities of the proposed SANN is affected by the influence of the plasticity window height on the firing activity of postsynaptic neurons. The self-repairing attribute of the SANN has been demonstrated through an obstacle avoidance application on a robotic platform. Furthermore, the reported simulation results reveal that despite fault ratios of up to 80% (not affected by fault location) the SANN maintains its learning manoeuvres.

Modelling glutamate transporters of astrocytes has been explored in a different study [49]. Experimental data and thermodynamics fundamentals have been used to derive an explicit model for glutamate transporters of astrocytes,

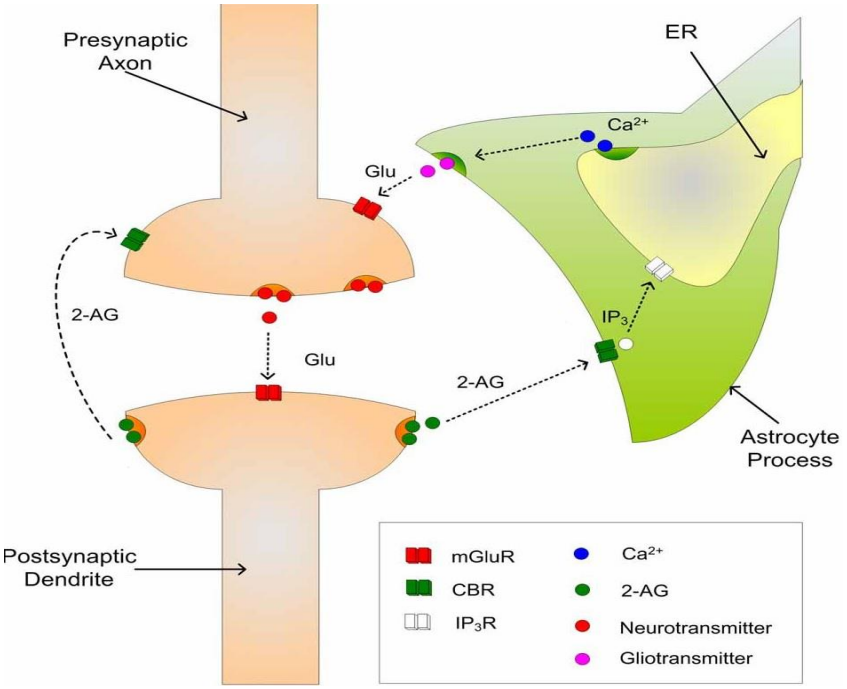


Figure 2:2 Tripartite synapse [6]

focussing on glutamate-induced chemical potential on astrocyte membranes. The model shows that uptake in glutamate as a result of synthetize downregulation of glutamine causes postsynaptic quantal size because of gliotransmission. Also, the proposed model indicates that higher astrocytic glutamate levels could increase the time course of glutamate in synaptic cleft and modifies the slow currents produced by the astrocyte. This leads to uptake in frequency of postsynaptic junction and disturbing synaptic signalling.

2.2.2 An Astrocyte Model for Self-repairing Hardware

Research at the *Intelligent Systems Research Centre* (ISRC) has led to an astrocyte model specific to performing self-repair in SANNs. This model will be investigated in more detail than the others since it is used in this work for realizing an astrocyte module in hardware.

The self-repair mechanism focuses on the interactions between astrocyte and neurons. The Ulster model for astrocytes operates within a feedback loop where the activity of neurons is relayed back to their associated synapses via astrocytes [3], [5], [19],[50].

This self-repair mechanism is illustrated in Figure 2:2. When a synapse fails to release neurotransmitter, the associated neural activity falls and consequently the level of 2-AG decreases. The absence of the 2-AG signal, which is a retrograde feedback messenger from active postsynaptic neurons, causes an overall increase in PR at all tripartite synapses. This is because the direct feedback of 2-AG to the presynaptic terminal DSE has diminished leaving the indirect feedback signal from the astrocyte e-SP to cause a sudden increase in PR. It is important to note that one or more nearby active neurons will be sufficient to maintain the astrocyte in an excited state. Therefore, the repair-

capability can be viewed as distributed whereby the state of neurons is continually monitored locally by the nearby astrocyte. Also because the astrocyte coverage is up to about 10^5 synapses then local in biological network terms can be anything between 4 to 12 neurons [51]. The equations representing the self-repairing mechanism in the astrocyte and neuron models are given in equations (10)-(27).

Whenever the postsynaptic neuron fires, an amount of 2-AG is released and can be represented using the following equation:

$$\frac{d(AG)}{dt} = \frac{-AG}{\tau_{AG}} + r_{AG}\,\delta(t - t_p) \tag{10}$$

Here, *AG* is the volume of available 2-AG, τ_{AG} and r_{AG} are the decay rate and production rate of 2-AG, respectively. IP3 can be modelled by:

$$\frac{d(IP3)}{dt} = \frac{IP3^* - IP3}{\tau_{ip3}} + r_{ip3}\,AG \tag{11}$$

where τ_{ip3} and r_{ip3} are IP3 decay and production rates, consecutively and $IP3^*$ is the initial level of IP3. This change in IP3 quantity can trigger calcium release which is given by [31]:

$$\frac{d(Ca^{2+})}{dt} = J_{chan} - J_{pump} + J_{leak} \tag{12}$$

Here, J_{chan} is the *IP3* and Ca^{2+} dependent calcium release, J_{pump} is the quantity of Ca^{2+} pumped into the Endoplasmic Reticulum (ER) and J_{leak} is the total Ca^{2+} leaked from the ER. The fraction of *IP3* receptors that is activated (*h*) is given by:

$$\frac{dh}{dt} = \frac{h_\infty - h}{\tau_h} \tag{13}$$

where h_∞ and τ_h are given by:

$$h_\infty = \frac{Q_2}{Q_2 + Ca^{2+}} \tag{14}$$

$$\tau_h = \frac{1}{a_2(Q_2 + Ca^{2+})} \tag{15}$$

and

$$Q_2 = d_2\,\frac{IP3 + d1}{IP3 + d_3} \tag{16}$$

J_{chan} is represented as follows:

$$J_{chan} = r_C m_\infty^3 n_\infty^3 h^3 (C_0 - (1 + C_1)Ca^{2+}) \tag{17}$$

where, r_C is the maximum Calcium Induced Calcium Release (CICR) rate, C_1 is the volume ratio of ER to cytoplasm and C_0 is the amount of free calcium in the Cytoplasm, d_1, d_2 and d_3 are constants. Moreover, m_∞ and n_∞ are given by:

$$m_\infty = \frac{IP3}{IP3+d_1} \quad (18)$$

and

$$n_\infty = \frac{Ca^{2+}}{Ca^{2+}+d_5} \quad (19)$$

m_∞ is the *IP3* Induced Calcium Release channel, d_5 is a constant and n_∞ is the CICR channel.

Equations (20) and (21) quantify the leak channel as:

$$J_{leak} = r_L(C_0 - (1 + c_1)Ca^{2+}) \quad (20)$$

and the pump channel as:

$$J_{Pump} = \frac{v_{ER}\,(Ca^{2+})^2}{k_{ER}^2 + (Ca^{2+})^2} \quad (21)$$

Where V_{ER} is the maximum rate of Calcium pumped into the ER, K_{ER} is a SERCA pump activation constant and r_L is the leakage rate of Calcium in the opposite direction.

Depolarization induced Suppression of Excitation (DSE) is assumed to have the following relation with *AG*:

$$\text{DSE} = \text{AG} * K_{AG} \quad (22)$$

The reason for including the scaling factor K_{AG} (=-4000) is to change the 2-AG level to a favourable negative range.

The internal calcium dynamics control glutamate release in this astrocyte model which is given as:

$$\frac{d(Glu)}{dt} = \frac{-Glu}{\tau_{Glu}} + r_{Glu}\,\delta(t - t_{Ca}) \quad (23)$$

τ_{Glu} and r_{Glu} are the (respective) decay and production rates of glutamate, and t_{Ca} is the time the calcium passes a certain threshold. Lastly, *eSP*, increase of synaptic transmission Probability of Release (PR), is given by:

$$\tau_{eSP}\frac{d(eSP)}{dt} = -eSP + m_{eSP}\,Glu(t) \quad (24)$$

Where τ_{eSP} is the rate of eSP decay, m_{eSP} is a constant (fixed at $55*10^3$).

The passive LIF neuron model [52] that is used in the self-repairing SANN can be given as:

$$\tau_m\frac{dv}{dt} = -v(t) + R_m\sum_{i=1}^{n} I_{syn}^i(t) \quad (25)$$

Where τ_m,v and R_m are the membrane constant, potential and resistance, respectively, I_{syn}^i is the current injected to the neuron through synapse i.

A probabilistic based tripartite synapse model is used in this model where a random number between 0 and 1 is generated by a uniformly distributed pseudorandom number generator. The injected current is given by:

$$I_{syn}^{i}(t) = \begin{cases} Iinj & P_{rand} \leq PR_{(t)} \\ 0 & P_{rand} > PR_{(t)} \end{cases} \quad (26)$$

Where I_{inj}=6650pA. In the case the network suffers no faults, as seen in Figure 2:3.a, the PR relating to each synapse is presented as:

$$PR_{(t)} = \frac{PR_{(t0)}}{100} * DSE_{(t)} + \frac{PR_{(t0)}}{100} * eSP_{(t)} \quad (27)$$

Here, $PR_{(t0)}$ is the initial PR of each corresponding synapse. In this case each synapse activity is depressed by ~50% overall. When a major fault is present the equation can be rewritten as:

$$PR \rightarrow (\frac{PR_{(t0)}}{100} * eSP_{(t)}) \quad (28)$$

With the absence of *DSE*, the PR sees an increase of 200% at the associated healthy synapse site. In short, it means that even if N2 is silent because of one or more faulty synapses, other healthy synapses or the recoverable faulty synapses still can bring N2 back to activity due to the indirect signalling from N1 via the astrocyte pathway.

This study implements the astrocyte model discussed in this section for several reasons. Firstly, as it is an in-house model the minute details of it is implementation is clear. These details are highly important when transferring such a complex model to hardware. Secondly, this model provides SNNs with self-repairing ability, making it extremely interesting for hardware implementation. It would be interesting to compare this model's hardware implementation with those of other astrocyte models in the literature in terms of power consumption and area footprint, however this falls beyond the scope of this work.

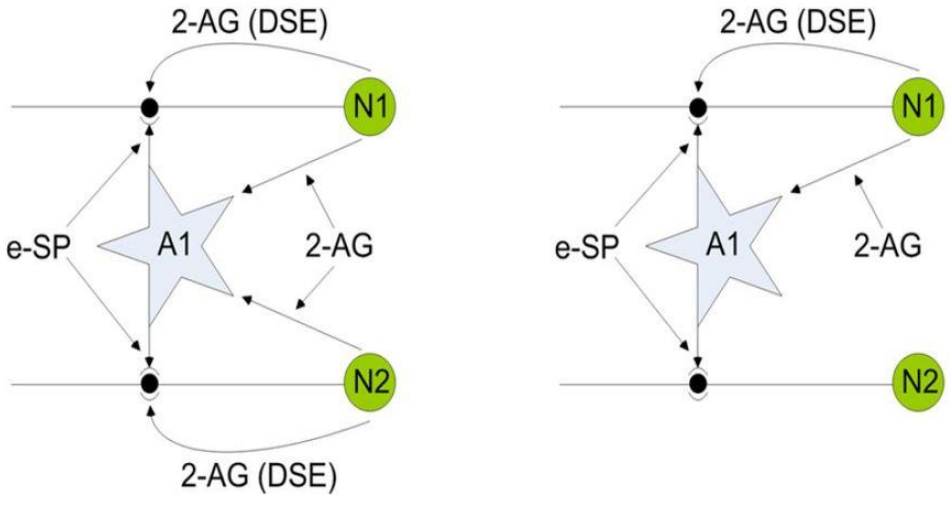


Figure 2:3 Interactions between the astrocyte and the two neurons between and after faults [6]

2.2.3 Astrocyte Hardware Implementations

In this section, the efforts to realize astrocytes in hardware are reviewed.

Research has focused on hardware implementation of astro-neuron interactions using resource-efficient FPGA astrocyte hardware [53] , [54]. The authors report that their model proves that firing rates of neurons can be modified by astrocytes and that astrocytes heavily regulate exchange of information at synaptic junctions. This is physically done my means of bi-directional channels between neurons and astrocytes. Single Constant Multiply (SCM) and linear approximation techniques have been used that permit deploying shift registers and adders instead of more resource-heavy components such as multipliers.

Linearization of the non-linear functions that represent biologically faithful astrocyte models has also been explored in the literature [55]. The linearization is done by using search algorithms for finding parameters that allow for lower cost hardware implementation while retaining reasonable accuracy. This technique has enabled the authors to implement an astrocyte hardware model without using DSP multipliers. Another similar work uses piecewise-linear approximation to eliminate the need for using multipliers while realizing astrocytes in FPGA hardware [56].

In other works [57], [58], an astrocyte model is put forward that breaks the synchronisation of two “Hopf” oscillators. The circuit is realized on FPGAs and the authors report a successful asynchronous operation of the “Hopf” oscillators.

Astrocytes have been incorporated into Intel’s Loihi [59] many core neuromorphic processor [60] through connecting Loihi compartments and tuning the compartments’ internal dynamics to mimic astrocytes behaviour. Loihi’s original Software Development Kit (SDK) has been modified to allow for astrocyte implementation in the neuromorphic chip. The proposed Loihi Astrocyte Module (LAM) is able to reproduce the fundamental astro-neuron communication channels at the tripartite junctions as well as astrocytic intra-cellular calcium changes in response to synaptic activities. The three cases in which the effect of the proposed LAM is demonstrated are 1) How astrocytes modulate neuronal activity through their calcium dynamics; 2) How STDP mediated by astrocytes introduce single-shot pattern memorization; 3) How the transition between order and chaos can be sensed by astrocytes.

In addition to the digital implementations discussed so far, analogue realization of astrocytes has also been explored [8], [61],[62], [63]. Generally, analogue circuits are more compact and energy efficient than digital implementation. However, analogue circuits are more prone to noise and require more memory. Moreover, the existing Electronics Design Automation (EDA) tools cannot infer analogue circuit from HDL codes.

SPANNER (a hardware architecture, not to be confused with SPANNER project) is a hardware implementation of the astrocyte model (Section 2.2.1) that is used to perform self-repair in astrocyte-Neural networks [12]. In this work, an astrocyte monitors synaptic activities that happen in two adjacent neurons as shown in Figure 2:4. Encouraging results have been reported that demonstrates the ability of the astrocyte to recover the firing rate when faulty occurs on synapses. If the postsynaptic terminal is damaged beyond repair, the astrocyte will try to make up for the lost neural activity by increasing the rate at which the remaining healthy synapses belonging to the same neuron can fire.

The SPANNER architecture has been used to implement a simple controller for a robotic car [64] to demonstrate the principle.

A modified SANN has also been developed in hardware that uses STDP and BCM rules to facilitate learning and self-repair [65]. The astrocyte in this model regulates spike transmission between the two layers of the network, effectively acting as a frequency filter. Whenever a fault occurs (manifested by lowering of a target neuron's firing rate) the learning window will be re-opened, and the target neuron's frequency reaches its previous value again after the end of training window. The proposed SANN has been made the bases of a controller that allows a robotic car to avoid obstacles and follow a particular colour. This work has been realized using time domain multiplexing for reducing interneuron hardware overhead in a different publication [66].

Optimizing resource consumption of a hardware implementation of the astrocyte model [6] has been explored in SPANNER project [67]. Some of the biologically detailed features of the astrocyte process have been removed. The result is a resource-optimized astrocyte model that is less faithful in terms of biological dynamics but can still perform self-repair in SANNs. Further optimizations have been carried out through modifying equation parameters that allow for simpler and less resource-hungry implementation.

Finally, research undertaken as part of SPANNER project has focussed on using homeostatic principle in neurons for self-repair [68], [69]. No astrocyte is implemented in this work as, instead, homeostatic has been achieved by; 1) Designing neurons with variable firing voltage threshold; and 2) Using FPGAs' Partial Dynamics Feature for adjusting the operating frequency of the neurons.

- **The limitations of the available approaches**

Apart from SPANNER [67], [70], none of the other approaches aimed at providing SNNs with fine-grained self-repairing ability. Several of the astrocyte hardware implementations do not report any applications [55], [56]. Other works apply astrocytes for regulating and improving neuronal activity [53], [54]. However, on top of inability to self-repair, no solid infrastructure has been provided in these works for scalability beyond one FPGA devices. Furthermore, the reported approaches lack re-configurability and ability of data acquisition, the features that are incorporated in the AstroByte platform. Other published applications of astrocytes are not related to SNNs [57], [58].

2.2.4 Hardware Interconnection Paradigms for Astrocytic Signalling

The focus of this research is designing a multi-FPGA, scalable and programmable platform for SANN acceleration called AstroByte. Among its many attributes, AstroByte can send astrocytic data, namely eSP, to nearby neurons. This section will outline similar work that deal with the challenge of communicating astrocytic data through scalable structures.

The first interconnection architecture to be referred to is H-NoC [71]. In fact, H-NoC, in its original form, is designed for SNNs as opposed to SANN. However,

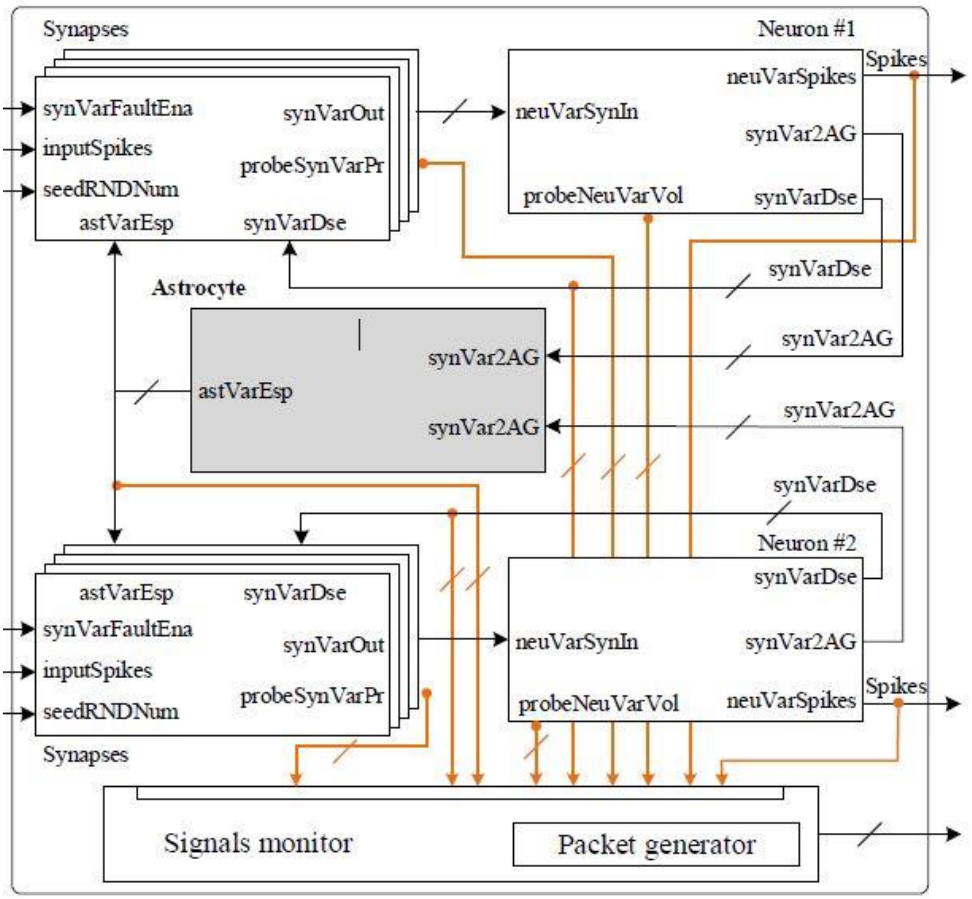


Figure 2:4 SPANNER hardware architecture [12]

since H-NoC will be improved on by several works that will be discussed in this section, it will be studied here.

H-NoC is a hierarchical and scalable NoC architecture for large scale SNN implementation as can be seen in Figure 2:5. At the bottom of the hierarchy exist the neuron facilities, each composing of a node router and four neuron cells. Ten such neuron facilities are interconnected using a tile router which, as well its ability to route spikes between neuron facilities, can also route data to the next layer in the hierarchy. Cluster routers are placed at the top of the hierarchy to create cluster facilities which route SNN packets between four tile facilities and other cluster facilities. Another feature of H-NoC is its utilization of a compression technique that enhances throughput and decreases traffic overhead.

H-NoC has been modified to accommodate for astrocytic communications with neurons as well with each other, giving rise to HANA - Hierarchical Astrocyte Network Architecture [72], [73]. HANA main components are astrocyte cells and astrocyte tile facilities. Large scale astrocytic networks can be realized by using a number of astrocyte tile facilities as shown in Figure 2:6. A number of communication approaches have been applied which are broadcast in one tile, broadcast among a different tile's astrocyte cells, point-to-point with in one tile and point-to-point between astrocyte cells in different tiles. This will manage traffic both globally and locally and achieves traffic balancing.

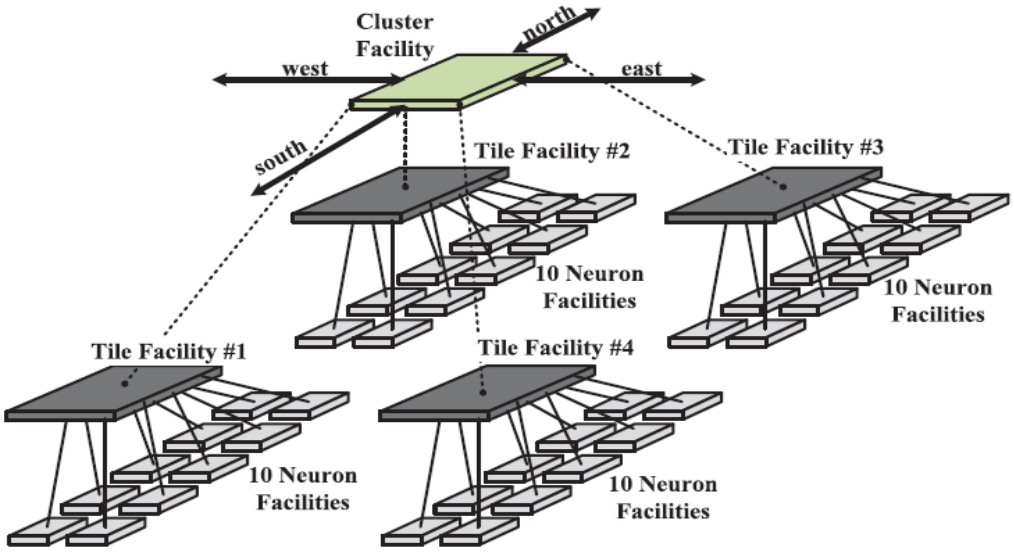


Figure 2:5 H-NoC overall architecture [71]

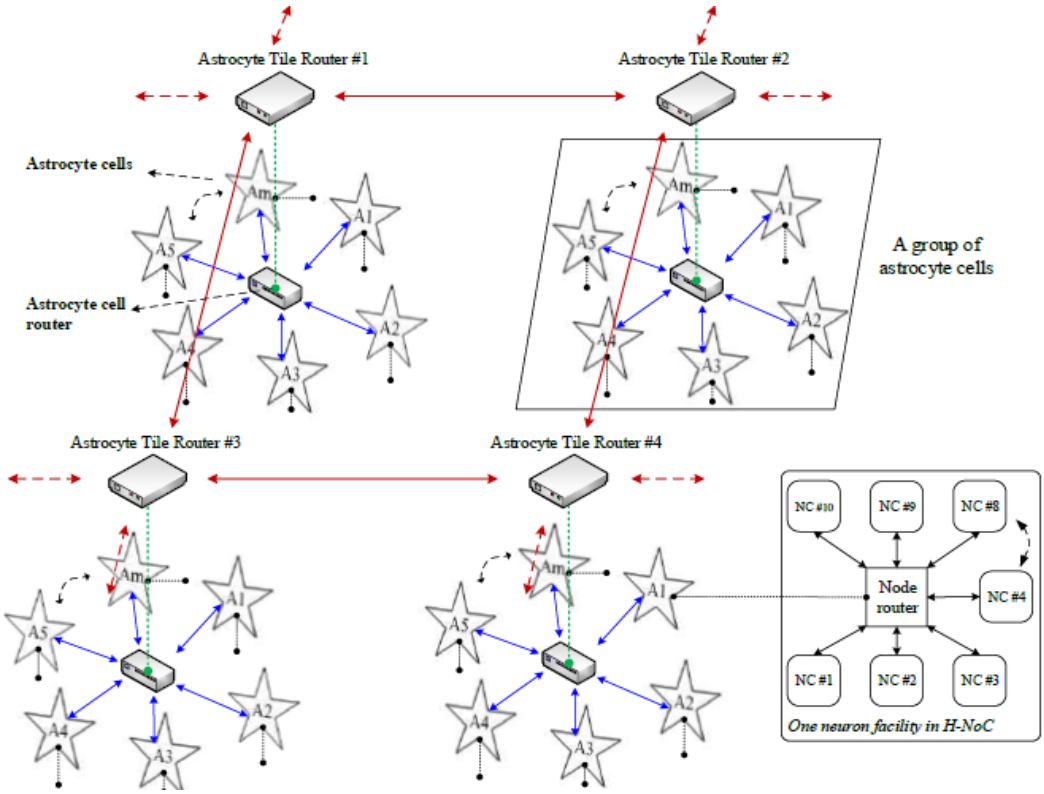


Figure 2:6 HANA overall architecture [61]

Further research has been carried out for establishing astro-neuron interconnections [74], [75]. The overall architecture is similar to the works mentioned above [72], [73] but differ in implementation details. Here, an astrocyte tile router has been used for regulating an IP3 pool among the astrocytes. Inter-routers have been implemented that serializes 64-bit IP3 values from the astrocyte for internal operations and parallelizes astrocyte-bound serial data that represents updated IP3 value. The overall architecture of the astrocyte tile router is shown Figure 2:7. The IP3 value from the astrocyte tile router is sent to all the astrocytes through an update manager that grants interneurons tokens for updating the IP3 pool value and then schedules updates to the astrocytic IP3 levels.

2.3 SNN FPGA Implementations

Simulating SNNs has traditionally been performed using a programming language such as MATLAB or PyNN [76] that run on single or multicore general-purpose processors [77]–[80]. In addition, some researchers have focussed on implementing SNN simulations on GPUs [81]–[86].

FPGAs are another attractive option for simulating SNNs due to their on-field re-configurability and ability to realize dedicated hardware in a parallel manner [10], [87]. This has made FPGAs an interesting alternative to general purpose CPUs and GPUs for SNN realizations, whether it is for purely simulation, acceleration or application purposes.

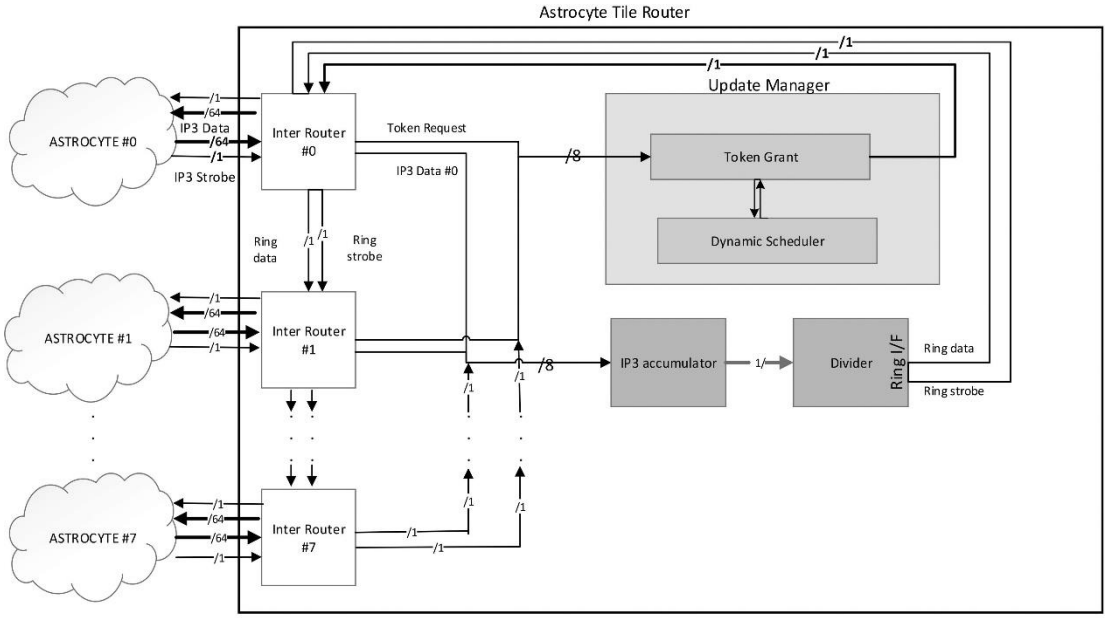


Figure 2:7 Astrocyte tile router [75]

To this effect, the EMBRACE architecture, an in-house custom SNN structure that leverages NoC paradigm for scalable SNN FPGA simulations using a power and area efficient neuron model has been developed [10]. Embrace deploys a 2D mesh NoC structure that has a programmable neural tile router as its building block. The neural tiles can be reconfigured via packets that updates an address table to store the addresses of target presynaptic connections, defines the synaptic weights, and choses the model implemented for synaptic connections. An XOR problem was successfully mapped into an SNN created on EMBRACE architecture to verify its functionality. An overall architecture of Embrace platform is shown in Figure 2:8.

Subsequent work has realized an EMBRACE-based SNN that facilitates 32 neurons with 32 synapses each on Xilinx Virtex II-Pro FPGA device [88]. The XOR problem has been solved on the SNN, which has been incorporated into

Evo Platform – a Genetic Algorithm (GA) platform that aids in evaluating SNN features for various applications. Further research has been carried out to re-route traffic in the mesh interconnect in case of congestion, thanks to incorporating a router that is able to adapt to traffic patterns, avoiding dropped router packets when congestion happens [89]. A 4 x 2 EMBRACE architecture model was tested on an Intel Stratix II FPGA to demonstrate the new traffic-adaptability feature.

A different approach has been adopted in NeuroFlow [87], a scalable platform for SNN simulation using FPGAs. This is a specialized processor with flexible architecture that can be modified to suit different configurations. It uses PyNN, an open source, neural network simulation language that is based on Python for modifying the processor architecture. Necessary compilation tools ensure translating high-level specifications of networks to FPGA implementations. Different models for neurons (e.g. LIF and Izhikevich) can be chosen and using

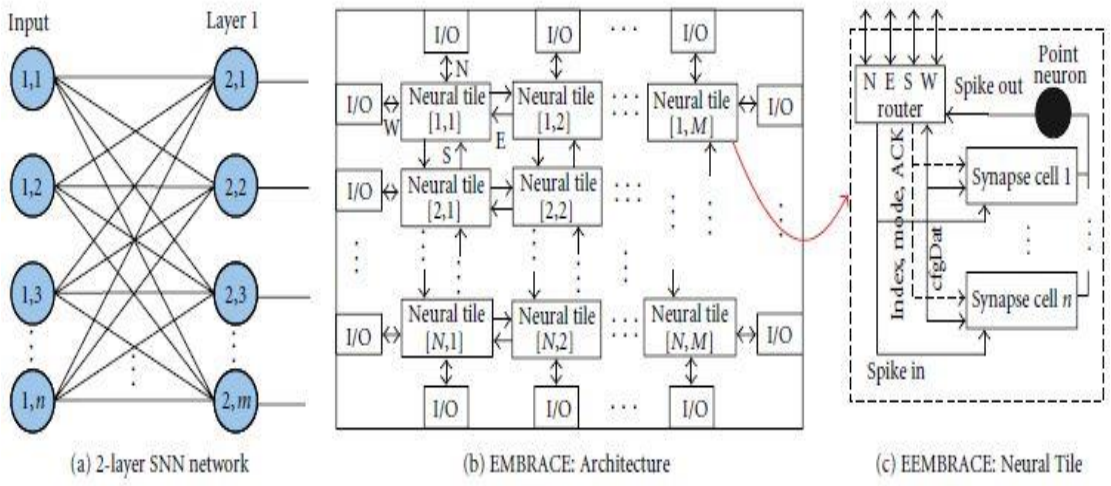


Figure 2:8 Embrace Architecture [10]

STDP as a learning rule is optional. A platform consisting of 6 FPGAs can perform real-time simulations of up to 400,000 neurons. A single FPGA NeuroFlow system can run simulations x33.6 faster than and equivalent software implementation on an 8-core processor while performing x2.83 better than a GPU model. NeuroFlow does not create a fully specialized accelerator for the network to be simulated as it uses kernels for simulating networks of large dimensions. NeuroFlow builds on two previous works by the authors for leveraging FPGA hardware for parallel, large-scale SNN simulations [90], [91]

A separate study has employed a fully connected SNN of 1024 spiking neurons on FPGAs [92]. Synaptic weights have been stored in block RAMs and the Izhikevich Neuron model [93] has been used. The authors have reported 2.2 GFlops in double precision as a performance measure. A different research has also focussed on SNN simulations in real-time [94]. An FPGA platform has been designed on Xilinx Virtex 6 FPGA device to emulate a fully connected network of 256 Izhikevich neurons. The work has explored fixed-point arithmetic for fast and hardware-efficient employment. The hardware prototype a sampling rate of 10 KHz to achieve “real-time” simulations in timesteps of 0.1ms.

FPGA utilizations of polychromous SNNs has also been investigated in the literature [95]. In contrast to using synaptic weights for information encoding, polychromous [96] method depends on delays caused by axons for evaluating and encoding information, making polychromous SNNs “ideal” for simulating short-term memory for patterns that are spatio-temporal in nature. The hardware realization is achieved by grouping the neurons together – creating neuron array- and grouping axons together - axon array. Instead of modifying synaptic weights, axonal delays are tuned during the training phase by a delay adaptation algorithm depending on pre-synaptic and post-synaptic firing time, effectively utilizing STDP rule. Two buses that operate using a modified version of Address Event Representation (AER) connect the two arrays together. One takes AER post-synaptic spikes to the axon arrays form the neuron arrays and

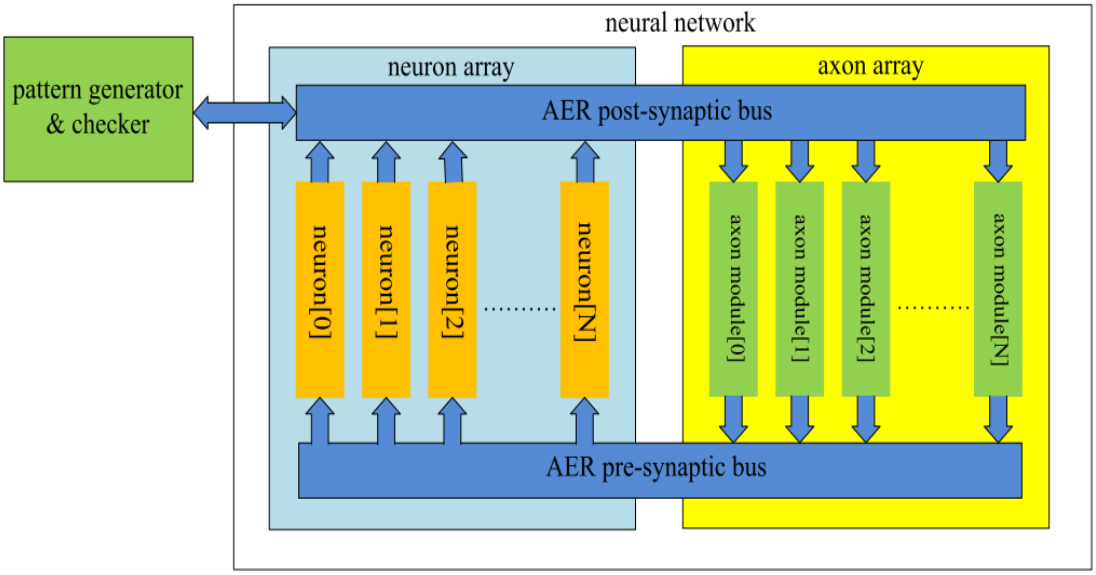


Figure 2:9: Architecture of the proposed polychromous SNN [95]

the other bringing pre-synaptic spikes from the axon arrays to the neuron array, creating a loop-like structure as shown in Figure 2:9.

The authors reveal that 1.15 million axons (with programmable delay capability) and 4096 neurons can be accommodated on a Xilinx Virtex 6 FPGA through leveraging time multiplexing. Finally, it is indicated that >95% of spikes for 96% of the stored patterns can be reproduced.

Bluehive [97] and SNAVA [98] are two multi-FPGA platforms that share several attributes with AstroByte platform, the contribution of this research. In subsequent chapters both SNAVA and Bluehive will be re-visited for comparison with AstroByte platform in terms of features, capabilities and performance. Bluehive [97] sees SNN emulation as a communication centric issue and tries tackling it from this perspective. It is a multi-FPGA platform for large-scale SNN real-time simulations. The FPGA boards used are DE4 boards are supplemented with Intel Stratix IV 230 FPGAs. The mapping of an application is done by using external DDR2 RAMs as neural netlists. The external DRAMs also contain parameters to be streamed to the processing units (called equation processor in the paper) along with the addresses of the destination neurons. The platform focusses on massive real time simulations as opposed to accelerating these simulations. The configuration given in the paper is 3D torus with each link having 18gb/s bandwidth. Simulation results are stored off chip and then read out post-simulation. Each FPGA supports real-time simulation of 64,000 neurons and 64M synapses through time-multiplexing hardware resources. Bluehive has several limitations which can be summarized as following. Firstly, it has limited reconfigurability as only the SNN structure can be modified. Secondly, simulations are limited to SNNs as opposed to SANNs, the focus of the AstroByte platform. Furthermore, no acceleration is possible as Bluehive has been exclusively designed for real-time simulations.

In addition, simulations have to be stopped completely before analysis data can be off-loaded from the platform to a PC.

SNAVA [98] allows parallel real-time simulation of SNNs by means of a scalable and programmable architecture that can scale to incorporate a number of FPGAs. It uses AER on a shared bus to communicate spikes between different Processing Elements (PEs). This shared bus extends to multiple FPGA boards in a ring topology. A special Instruction Set Architecture (ISA) is provided that can; 1) Configure the programmable Central Processing Element (CPE) to a desired neuron and synapse model utilizing time-multiplexing technique for taking full advantage of the available hardware; 2) Loading configuration data from a host PC to SNAVA that defines the SNN architecture; 3) Storing data in BRAMs before sending them back to the host PC for monitoring through an Ethernet cable. SNAVA creates configuration files to implement the desired network model. Figure 2:10 presents a 2-FPGA SNAVA platform. The limitations of SNAVA platform are as the following. Firstly, simulations are confined to SNNs and astrocytes haven't been incorporated as with the Astrobyte platform. Second, SNAVA focusses on real-time simulations instead of acceleration, which is a key feature of the Astrobyte platform. Additionally, only the SNN structure and neuron models are reconfigurable after synthesis. Finally, each FPGA board in the SNAVA platform requires a separate Ethernet connection for acquiring monitoring data, an approach that is less efficient than the one used in the AstroByte platform, requiring only one cable for acquiring simulation data.

FPGAs have also been used extensively in the literature for accelerating Convolutional Neural Networks (CNNs) application. CNNs are widely used, both in academia and industry, for AI applications in the domain of feature recognition and object detection, to name but a few [99]–[103]. Similar to SNNs, CNNs are also based on neural networks however their relative literature will not be mentioned as the subject falls outside the scope of this study.

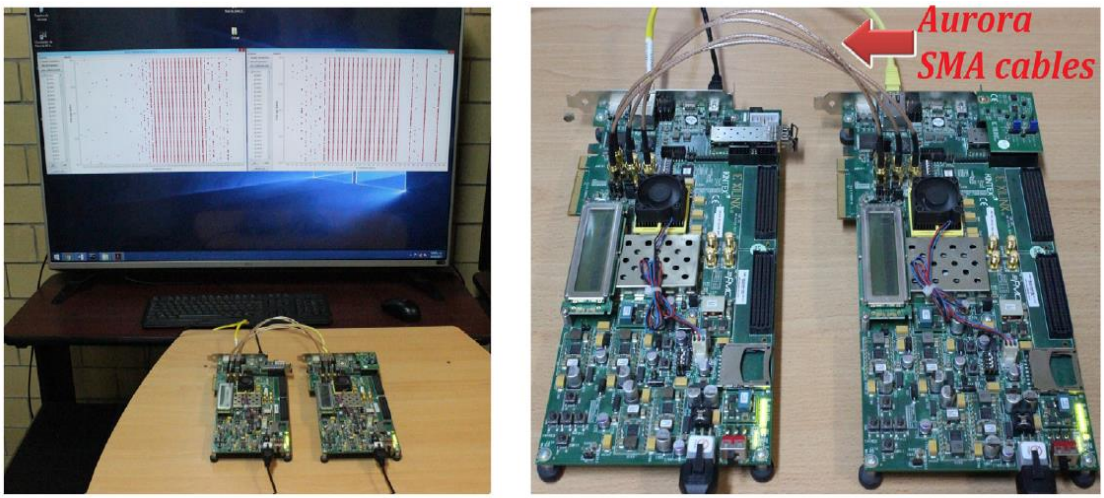


Figure 2:10 A 2-FPGA SNAVA platform [98]

2.4 SpiNNaker

Despite the flexibility provided by FPGAs, researchers in both academia and industry are still interested in ASIC implementations of SNNs [59], [104]–[107]. This is due to the fact that ASICs provide superior performance compared to FPGAs, in spite of the fact that they lack flexibility.

SpiNNaker [108]–[110] is the most prominent SNN ASIC simulator in the literature and it will be described here. It is a million-core massive parallel computer with an interconnection architecture that is inspired by the mammalian brain through which SpiNNaker sends AER based packets in large numbers. These features pave way to simulating large scale SNNs (hundreds of thousands of neurons and millions of synapses) in real time. The logical topology is a 2-D triangular mesh with each node being an ASIC chip. In its first iteration, the chip contains ARM processors with local memory each and a shared SDRAM. A router and peripheral IPs also included in each node. There are more than one iteration of SpiNNaker and each provide different amount of processing power, cache and RAM. Several software packages have been developed for SpiNNaker that allow for programming, configuring and running applications on the platform. SpiNNaker focusses on real-time simulations while AstroByte focuses on accelerated simulations. SpiNNaker architecture is designed to support event driven simulations and the inclusion of astrocytes in AstroByte introduces non-event-based data (i.e. actual numeric data) which would require a new packet format for the SpiNNaker NoC (synchronous and asynchronous). In addition, the SpiNNaker CPUs are not designed for the highly complex operations required within the dynamics of astrocytes and therefore can significantly slow down SANN simulation on SpiNNaker as many clock cycles will be required to calculate the complex equations representing the astrocyte process. Further research is required to quantitively compare utilizing SANNs on SpiNNaker and AstroByte.

2.5 FPGA Data Acquisition Platforms

One of the contributions of this work is designing a data acquisition platform for acquiring real-time data from FPGAs and sending the data to a PC. This subsection mentions the relevant literature in this domain. In one paper an FPGA has been used as a bridge between an Analogue to Digital Converter (ADC) and an off-chip DDR3 SDRAM [111]. However, it is not clearly stated how the data is stored in the DDR3 SDRAM will be analysed as no method of transferring this data to a PC is mentioned. A different publication uses a Xilinx Spartan 6 FPGA for acquiring data from an ADC and sending it to a PC by using Ethernet for monitoring [112]. This is similar in concept to the work in this study, but the implementation used Xilinx FPGAs and tools. Also the application proposed in [112] focusses on dedicated imagers in nuclear medicine as opposed to SANNs, the focus of this research. Another FPGA-based monitoring platform has included an FPGA for the purpose of signal processing and

controlling multiple sensor channels in a machine condition monitoring system [113]. Here the FPGA doesn't include any form of soft processor, but instead controls a number of monitoring channels while processing data at the same time. The processed data is then sent to a PowerPC based control system which in turn sends the data to a monitor.

2.6 NoC Background

This work utilizes NoCs for implementing a scalable and programmable SANN accelerator, called AstroByte. This section is dedicated to the important aspects that should be considered while designing NoCs. These attributes are NoC topologies, flow controls, routing algorithms and issues that arise as the result of routing.

2.6.1 NoC Terminology

The most common NoC-specific terms used in this work are clarified below.

1- *Upstream router* (or node): Is the router that sends packet i.e. the source of the packets.
2- *Downstream router* (or node): The router that receives the packets sent by the upstream router, i.e the receiving end of the communication link.
3- *Computing core, Internal core, Computing element*: The element connected to port 0 where the actual computation/processing takes place. SANN in this research.
4- *Packetization*: The process of converting data from the computing core to a data packet (will be studied in Section 4.4). Taking place in source nodes, a data word (flit) will be preceded by a header flit allowing the routers direct the packet to the destination node through the NoC structure.
5- *De-packetization*: The reverse process which happens at the destination nodes. Packets will be analyzed, and data flits will be forwarded to the computing core while the header flit will be discarded.
6- *SANN elements/components*: Astrocytes, neurons, and tripartite synapses are SANN elements.
7- Neuron entity: Is a neuron along with a tripartite synapse, spike and probability generators.

The following section discusses some aspects of NoC design such as NoC topologies and routing algorithms mainly to justify the NoC design choices made in this work. More detailed information about NoC design is found in the literature [114]–[120].

2.6.2 Networks-on-Chip Topologies

NoCs are in essence a group of routing nodes that communicate packets from and to a whole host of computing cores, attached to the NoC routers. The routers can be interconnected in many different configurations, called

"*Topology*". Examples of NoC topologies are *Mesh*, *Torus* and *Butterfly*, to name but a few.

- **Mesh**

Nodes forming the network are arranged into an *X-Y* dimension lattice. This network permits communication between one node and its neighboring nodes. This suggests that the nodes that are not on the edges of the network can transfer data with 4 different routers as shown Figure 2:11 [114].

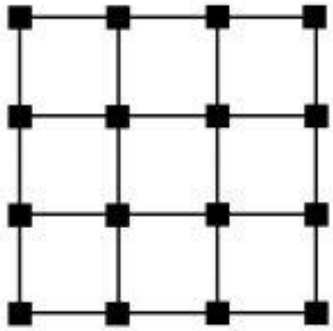


Figure 2:11 Mesh network

Mesh topology requires relatively simple routing engine, is easy to implement and scalable. Additionally, each node in the network contains a computing node, this means the distance between neighboring nodes is always 1 in any direction. These attributes along with other practical reasons(discussed in chapter 4) make mesh the topology of choice for the AstroByte platform.

2.6.3 Networks-on-Chip Flow Controls

Flow control mechanisms decide the way the network's resources - channel bandwidth and buffers - are allocated to data packets transmitted across the network. If the flow-control mechanism is a good one, the network distributes the resources in an effective way, allowing it to achieve a high fraction of the network's maximum bandwidth and packets traverse the network with predictable and low latency [3].

- **Buffer-less Flow Control**

The simplest mechanisms to control the data flow inside a network do not have any buffers. By using some arbitration technique, buffer-less flow control techniques give the network resources access to one of the competing channels while misrouting or dropping the packets which have lost the competition [3]. *Dropping Flow Control* and *Circuit Switching* are the main types of *Buffer-less Flow Control*.

Packets will be directly dropped should it loses competition for an output channel in case of *Dropping Flow Control*. A high-level error control protocol should be used to order the transmitter to resend the packet. This is the most

basic flow control approach. *Circuit Switching*, on the other hand, creates a circuit from source to destination and a number of data packets are sent along the dedicated path which is usually dissolved by means of a tail packet. This method retains the packets that lose the channel contention and retransmits them regularly until access to the channel is granted.

- **Buffered Flow Control**

Buffered Flow Control utilizes buffers for storing packets to decouple the allocation of neighbouring channels. The buffers used can be either packet-sized or flit (Flow control unit)-sized. Four types of buffered flow control will be examined below.

- **Store and forward flow control**

In this flow control mechanism, routers wait for the entire packet to be received before forwarding it to the next stage in the network, meaning each router should have a packet-sized buffer. Networks use this type of flow control can suffer from high latencies [3].

- **Cut-through flow control**

Improves on store-and-forward flow control by sending the packet to the next router as soon as the header flit is received. The implication is that waiting for the entire packet to arrive is no longer a necessity. This is perceived as a step forward in terms of the latency exists in store-and forward flow control.

- **Wormhole flow control**

By allocating the network resources (such as channels and buffers) to flits as opposite to packets, *Wormhole Flow Control* increases the efficiency of buffer usage. However some throughput degradation should be expected when compared with other types of flow controls [3].

- **Virtual Channel (VC) flow control**

By sharing the physical resources of a channel among a number of logical channels, VC flow control allows for more efficient usage of the bandwidth available by constantly using the physical links among the routers [3].

The AstroByte platform utilizes store and forward flow control mechanism because AstroByte depends on quick dispatching of short packets rather than waiting for long data streams to be generated, in which case other buffered mechanisms would be more attractive. The AstroByte throughput could be more improved using virtual channelling which will be discussed in the future works.

2.6.4 Routing Algorithms

Routing data from a transmitter to a receiver over a network requires a routing algorithm that governs how packets move between source and destination. [117].

Examples of routing algorithms are *XY* routing and *Destination-tag* routing.

- ***XY Routing**:*

This is a sort of dimension order routing in which, firstly, packets of data will be moved forward horizontally before changing to vertical movement when the designated column is reached. XY can absolutely fit torus or mesh topology [117], [121], thus is the routing mechanism of choice in this work.

2.7 Main Challenges

This section discusses the major challenges that are left unaddressed in the literature that will be tackled in this work.

2.7.1 Incorporating Astrocytes in SNNs

The research approaches reviewed in this chapter that utilize FPGAs for SNN simulations (e.g. [10], [66], [92], [98], [122]) do not utilize astrocytes. The current consensus is that astrocytes contribute to neural activity [30], [32], [123]. This implies that without including astrocytes, SNN simulations will not be an accurate reflection of its biological counterpart. Integrating astrocytes into SNNs to create SANN, however, gives rise to two main challenges:

1- The communication demands of SANNs is higher than that of SNNs due the additional signalling pathways introduced by addition of astrocytes. Addressing this challenge requiring novel solutions to accommodate astrocyte communications efficiently.
2- The astrocyte process, especially a biologically accurate one, requires a lot of FPGA hardware resources.

2.7.2 Astrocyte Process Optimization

The astrocyte process has to be optimized to reduce its hardware footprint. A high-level optimization technique would be using fixed-point number representation instead of floating point. The fixed-point resolution has to be decided upon through research to find a balance between accuracy loss and hardware consumption.

2.7.3 Networks-on-Chip Interconnection Paradigm for SANNs

NoC paradigms implemented in the literature [97], [98], [124] are used for SNNs and do not accommodate the communication requirements of SANNs. HANA

[72] creates a separate network for astrocyte data communication in addition to the one that already exists for the SNN. A more efficient approach would be using one NoC interconnection structure for routing both astrocyte and spike data packets, an approach that has been taken in this research. Using one NoC means using half the number of routers and interconnections will be used, causing significant improvements as far as resource usage, area footprint and communication requirements are concerned.

2.7.4 Methodology for Configuration and Monitoring of FPGA-based Hardware Accelerators

Simulating SNNs or SANNs requires capturing simulation data off-chip for monitoring and analysis purposes by researchers. Additionally, to avoid continues modifications to FPGA hardware designs, modern approaches introduce some degree of configurability in the Register Transfer Level (RTL) code before FPGA realization. The techniques reported in the literature limit configurability to a few attributes [97], [98]. Furthermore taking data off-chip either requires stopping simulations completely [97] or dedicated point-to-point connection from the host PC to all the FPGA boards present [98]. This study tackles this challenge by means of the FCMP which acts as a node on the NoC interconnect, removing the necessity for dedicated connections or stopping simulation before capturing data. FCMP allows for easy C-language programming of configurations along with real-time data acquisition.

2.8 Chapter Conclusion

By way of conclusion, this chapter covered the major publications in the domain of this work. As implementing FPGA-dedicated hardware for SANNs is one of the main contributions of this study, the major computational models of neurons, astrocytes and tripartite synapses were discussed. This was followed by describing the published hardware models for SANNs, both in-house and those reported in the literature. Another contribution of this study is designing a multi-FPGA interconnection NoC-based infrastructure for SANNs. Therefore, NoC interconnect realizations for SANNs and intra-astrocyte communication were examined. Furthermore, main work available in the literature were cited that deal with SNN simulations on FPGAs. A brief summary of SpiNNaker is given, citing a key research project in academia that implements ASICs. To cover basic principles in NoC design (will be explored in Chapter 4), background regarding NoC topologies, flow controls and routing algorithms were presented. Finally, the major challenges that remain in the literature have been designated since these challenges are addressed in this research.

The next chapter focuses on the hardware implementation of an SANN which incorporates a biologically faithful astrocyte into a neural network for accelerating simulations.

Chapter 3: A Biologically Faithful SANN Model for FPGAs

A novel FPGA hardware implementation of a biologically faithful astrocyte-based self-repairing mechanism for SNNs is presented in this chapter. This chapter looks at Astrocyte fixed-point implementation and demonstrating with reduced precision. Accuracy is maintained with overall reduced area of FPGAs. SANNs are a new computing paradigm which capture the key mechanisms of how the human brain performs repairs [6]. Using SANN in hardware affords the potential for realizing computing architecture that can self-repair. This chapter demonstrates that SANN in hardware have a resilience to significant levels of faults. The key novelty of this contribution resides in implementing a FPGA SANN Accelerator (FSA) using fixed-point representation and demonstrating graceful performance degradation to different levels of injected faults via its self-repair capability. A fixed-point implementation of each of the astrocyte, neurons and tripartite synapses are presented and compared against previous hardware floating-point and Matlab software implementations of SANNs. Results show how the reduced fixed-point representation in hardware can maintain the biologically realistic repair capability. The reduced precision FSA will be used in the chapter 5 as the bases of a multi-FPGA accelerator platform, called AstroByte. The FSA implementation and results are the first contribution of this chapter and are published in the paper titled "*Assessing Self-Repair on FPGAs with Biologically Realistic Astrocyte-Neuron Networks* " at the IEEE Computer Society Annual Symposium on VLSI (ISVLSI) conference in 2017 [1].

To support the operation of FSA, the capability of fault injection to synapses and monitoring significant levels of neuron and astrocyte data for off-chip transmission to PC-based analysis, are required. The second contribution of this chapter is designing an **F**PGA-based **C**onfiguration and **M**onitoring **P**latform (FCMP) for injecting faults and capturing and analyzing data acquired from the FSA. The FCMP uses custom logic and a NIOS II based system to control fault injection and data monitoring on the FPGA. Results show accurate accelerated simulations of fault injection scenarios using FCMP with speedups up to 249 times greater compared with equivalent Matlab implementations. The FCMP is the second contribution of this chapter, published under the title "*FPGA-based Fault-injection and Data Acquisition of Self-repairing Spiking Neural Network Hardware*" in 2018 IEEE International Symposium on Circuits and System (ISCAS) conference that took place in Florence, Italy [11].

3.1 AstroByte Platform Overview

AstroByte platform, shown in Figure 3:1, is a multi-FPGA NoC based platform for accelerating simulations of SANN. Each node on AstroByte platform is a

separate FPGA board that is composed of NoC router. Each router (e.g. 1,1) is connected to other routers as well as a computing core. The FCMP and FSA elements (the astrocyte, neuron and tripartite synapses) are utilized as computing cores in AstroByte. Each AstroByte platform utilizes one FCMP core that is connected to a PC (node 0,0 in Figure 3:1 above) and the rest of the nodes have FSA components or counters (for testing and verification purposes) as their computing cores. The focus of this chapter is on developing the computing cores. The NoC infrastructure will be detailed in the next chapter.

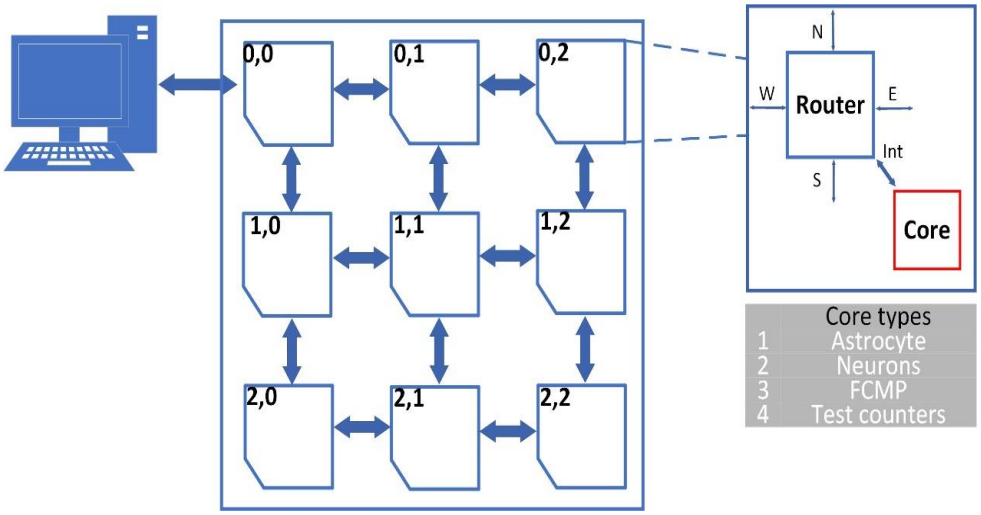


Figure 3:1 an overview of AstroByte architecture

3.2 Astrocyte Hardware Architecture

In this work, the astrocyte block diagram in Figure 3:2 has been implemented with a fixed-point representation on a Stratix IV FPGA and verified using Intel SignalTap II. This astrocyte model is biologically faithful and gives accurate results when compared to a dual floating point MATLAB software version that uses the same equations as the hardware version. The PC used for all the experiments in this study has the following specifications; 64-bit Windows 10 Enterprise, Intel Core i7-2600 3.4 GHz processor with 16GB RAM. Intel Quartus Prime 18.0 SE was used for VHDL coding, analysis, synthesis and FPGA programming. MATLAB R2015b was used for capturing simulation data and analysis. An Intel build for Eclipse Mars 2 was used with Nios II embedded processor.

The mathematical equations, parameters of the astrocyte, input values and expected output results for both the software and the HDL versions of the astrocyte are obtained from the original self-repair model; i.e. this is a block based design with each block representing a parameter of the astrocyte mentioned in the original model [6].

In comparison to the existing literature regarding astrocyte hardware implementation (section 2.2.3), this work's novelty is using a biologically faithful astrocyte for providing SNNs with self-repairing capability. Additionally, several

resolutions of fixed-point astrocyte are considered and one (32-bit) is chosen provides best balance between accuracy and FPGA resource usage.

In software, using a large number of bits to represent the values of the parameters and the variables of the astrocyte model presents little performance issues since modern computers can compute using double floating precision without much effort. In hardware, however, this can prove challenging for the designer because using larger resolutions can significantly increase the design resource consumption and physical blueprint, while using a small resolution can affect the accuracy of the results significantly. For efficient implementation, fixed point arithmetic is used. We start from a resolution of 24 bits, 16 bits for the fraction part and the remaining 8 bits for the real part. This means all the sub-blocks in Figure 3:2 were reduced to fixed-point 24-bit precision. These values were chosen because the astrocyte values include significant fraction arithmetic elements. Several sample values of different parameters are as following; IP3 → 0.373345, m_∞ → 0.7417, ΔGlu→-0.00000135079. The precision of the results is important since inaccurate results from any of the astrocyte sub-blocks in Figure 3:2 will propagate through the chain and affect the output result of the complete block. Figure 3:3(a) shows a 24-bit implementation compared against the floating-point MATLAB and fixed-point VHDL astrocyte models. The x-axis represents the number of iterations (the cycles of simulation) while the y-axis represents the calcium value which is obtained from the "Ca2+ generator" block in Figure 3:2. The calcium value has been chosen for accuracy comparison because the blocks that follow are threshold based. Each 1,000 iterations represent one second of biological time, as the Euler method is used for solving the differential equations representing various components of the SANN with a time constant of 1ms as per the original model [6]. Intel Quartus Prime software is used for synthesizing the VHDL code onto a Stratix IV FPGA and Intel SignalTap II is used for verifying the results.

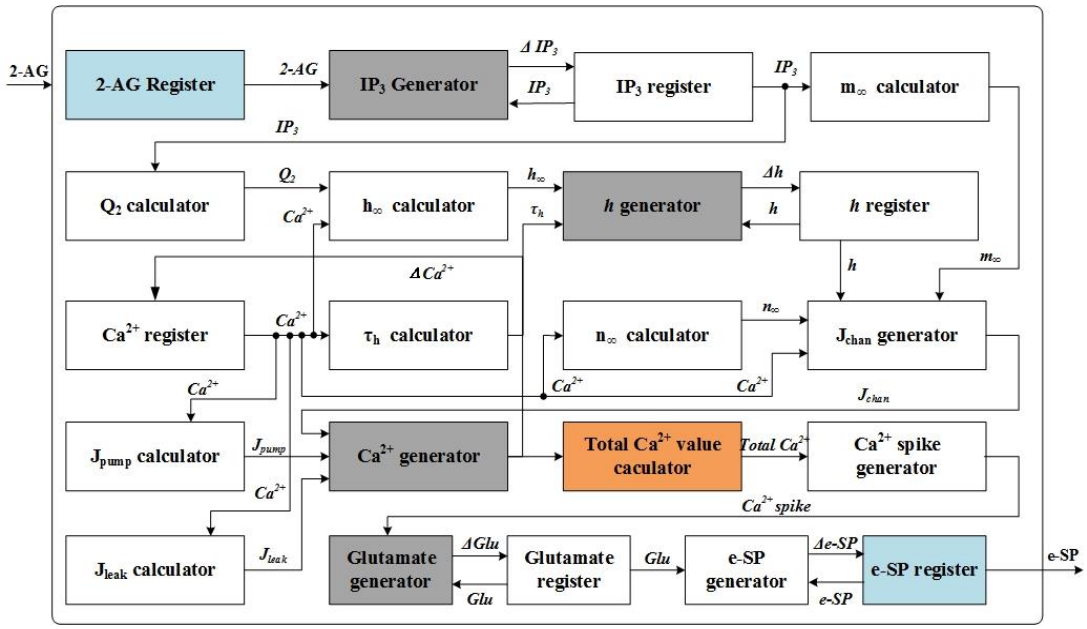
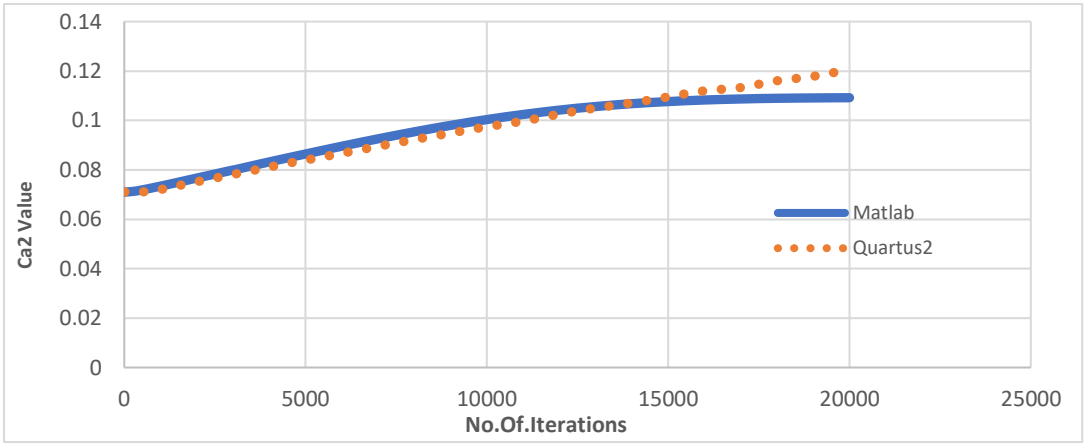
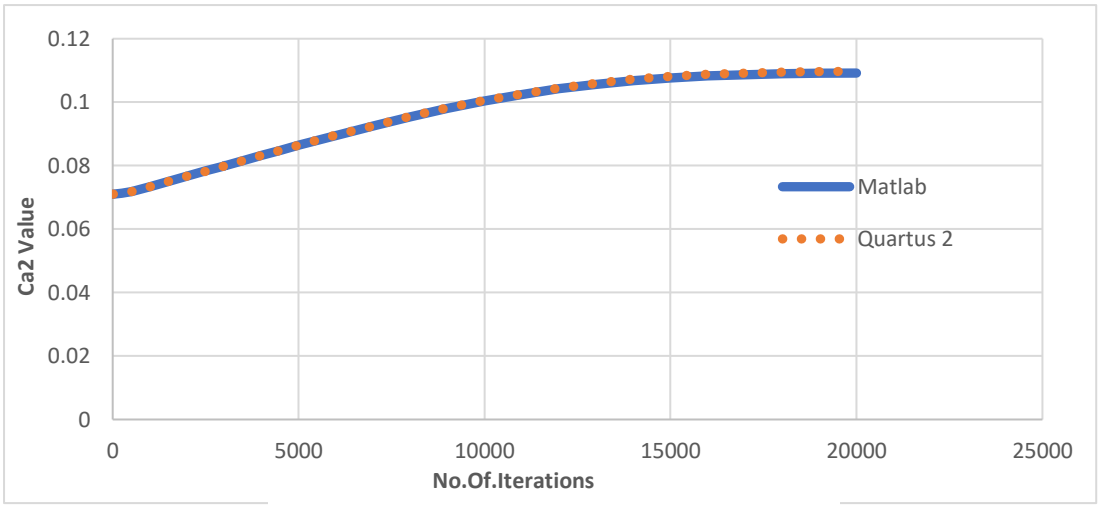


Figure 3:2 Astrocyte hardware block diagram [13]



(a) FPGA- Matlab- 24-bit resolution comparison



(b) Matlab - 32-bit resolution comparison

Figure 3:3 Comparison between the double-float Matlab astrocyte model and its fixed-point VHDL hardware implementation

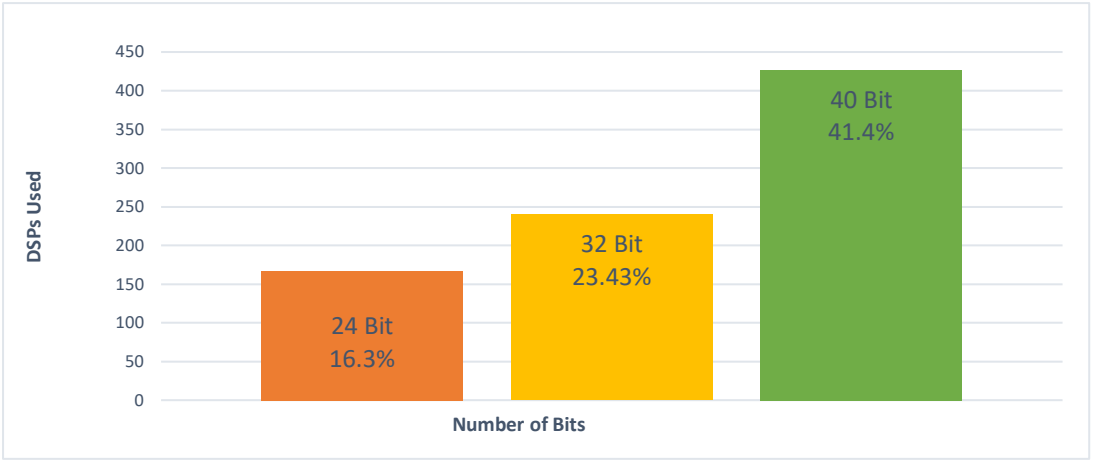
As can be seen from Figure 3:3 (a), for the 24-bit precision, the difference between the two trajectories is non-linear and fluctuates between iterations 10,000 and 15,000 before picking up steadily until it reaches a difference of just over 10% at 20,000. It is believed that for a biologically faithful astrocyte, loss in accuracy of 10% is non-trivial. This led to increasing the resolution from 24 bits to 32 bits with the additional 8 bits assigned to the fraction part since that is where the error occurs. Figure 3:3 (b) illustrates enhancement in accuracy of the hardware model after the resolution was increased as the trajectory of the hardware model is closely aligned with the Matlab software model; this indicates a better fit of the 32-bit resolution with the Matlab. For the 32-bit resolution, the difference between the two plots in Figure 3:3(b) is 0.04% after 20,000 iterations. This error is small compared to over 10% for the 24-bit model. On average, for over 20,000 iterations, the percentage difference is 2.73% for the 24-bit implementation while it is 0.1667% for the 32-bit design. Figure 3:4 shows hardware resource usage for 24-bit, 32-bit and 40-bit resolutions.

Therefore, given the area and accuracy, the 32-bit resolution was chosen. In comparison with the Matlab software model, the error rate of the 24-bit

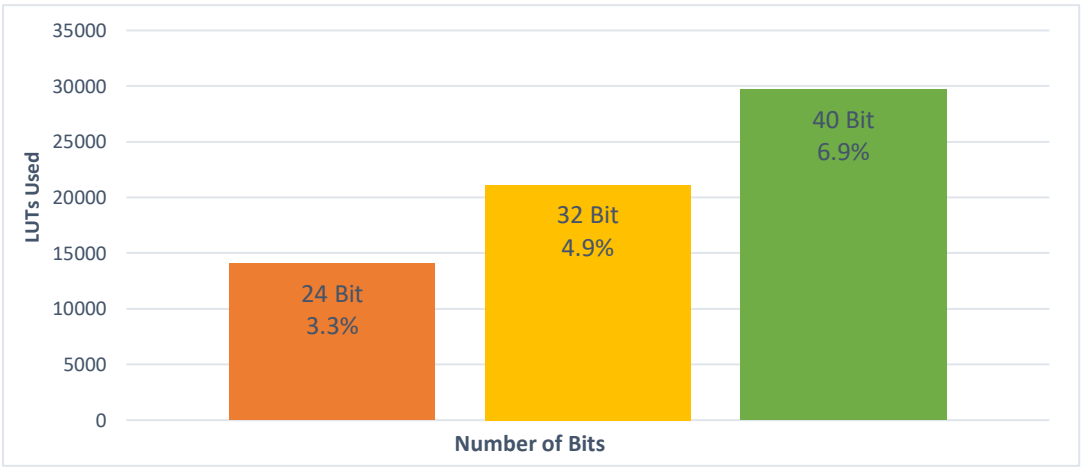
hardware design is unacceptable. A 40-bit resolution was also explored to examine area overhead compared to the 32-bit resolution. The 40-bit resolution doesn't significantly enhance the accuracy over the 32-bit design while significantly increasing resource usage. A neuron in 32-bit resolution requires 1840 LUTs (< 1% of device resources) and 60 DSP blocks (6% of device resources). It is worth noting that the resource utilization results are obtained using Intel Quartus tools. It is possible to command the synthesis tool to further optimize the resource usage or to use less DSP at the cost of using more LUTs. However, we show these figures for the fair comparison between the designs of different bit resolutions. Tradeoffs in terms of energy usage has not been explored in this work.

3.3 FSA Evaluation and Hardware Implementation

In this section, an FSA containing one astrocyte and two surrounding neurons with ten synapses each was implemented on an Intel DE4 development board with the Intel EP4SGX530KH40C2 device. The purpose of developing an FSA was to examine the effect of the reduced bit resolution on the self-repairing capacity of the SANN. Figure 3:5 shows the topology of this network. The astrocyte (A1) monitors the activity at two group of synaptic junctions, each



(a) DSP count



(b) LUT count

Figure 3:4 Comparison of resource utilization between the different astrocyte implementations.

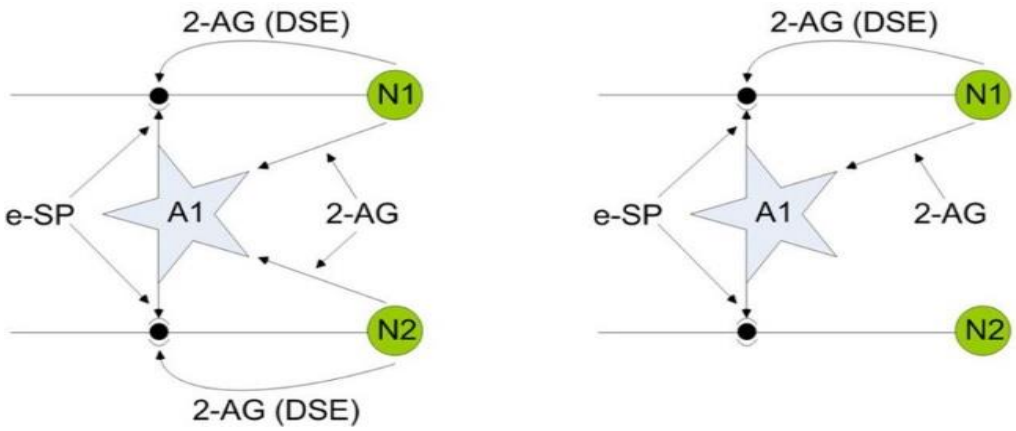


Figure 3:5 Interactions between the astrocyte and the two neurons between and after the fault is injected [6].

group containing 10 synapses that are associated with the two neurons (N1 and N2). The LIF model was used for describing the neurons and the tripartite synapses are probabilistic and based on the model developed in [6]. The inputs to the tripartite synapses are stimulated by means of simple shift registers acting like spike generators. The FSA was developed to examine the effect of reduced resolution in a typical larger configuration. The advantage of FSA is highly tolerant information processing that is resilience to high fault ratios with reduced fixed-point implementation.

3.3.1 Fault modelling in SANNs

For neurons, a fault means that the neuron either stops issuing spikes completely or its firing rate has dropped to levels below normal. Figure 3:6 shows fault progression at the synaptic junctions. A fault for a synapse means that the value of $PR_{(to)}$ in equation (27), calculated in the Probability Calculator block in the figure, will be modified. $PR_{(to)}$ has the value of 0.5 under normal circumstances, however, when the synapses suffers a fault, the $PR_{(to)}$ changes to 0 in case of a *severe* fault and 0.1 in case of a *partial* fault. It worth mentioning that, in this study the faults, whether severe or partial, are all user-induced to monitor the response of the system. Additionally, the terms *severe* and *partial* are specific to the domain of the SANN that has been realized in this work, these are not standard terms that are used for hardware fault description in the literature. This research deals with faults that are specific to the domain of this work-SANNs- as opposite to standard hardware fault models.

$$PR_{(t)} = \frac{PR_{(to)}}{100} * DSE_{(t)} + \frac{PR_{(to)}}{100} * eSP_{(t)} \quad (27)$$

Referring to Figure 3:6, the fault affects the SANN in the following sequence. The user defines the faults, which will turn into hardware signals that act as the select signal for a multiplexer (Mux in the figure). The inputs to the Mux are hardwired values corresponding to no-fault (0.5), partial fault (0.1) and severe fault (0). The Mux output feeds into the probability calculator which calculates the instant probability at the tripartite synapse $PR_{(t)}$ according to equation (27) from chapter 2. The generated probability then goes into the Spike generation block which works according to equation (26) that was discussed in chapter 2.

$$I_{syn}^{i}(t) = \begin{cases} Iinj & P_{rand} \leq PR_{(t)} \\ 0 & P_{rand} > PR_{(t)} \end{cases} \quad (26)$$

A Pseudo Random Number Generator (PRNG) generates a random probability $P_{rand.}$ If there is a spike from the SR (Shit Register), the spike generator block compares the two probabilities. If $PR_{(t)}$ was larger than $P_{rand,}$ the synapse passes the spike at the input to the neuron, otherwise the synapse generates no spike. The effect of this process is a probability-based spike generation that depends on both randomness and the instant probability in equation (27).

For instance, when the observer injects a partial fault, $PR_{(o)}$ will be 0.1. This causes a reduction in $PR_{(t)}$ value, causing the spike generation block to pass less spikes as $PR_{(t)}$ will be less than P_{rand} more often than not. The effect is reducing the spikes that are passed to the neuron and eventually the firing rate of the neurons. In case of a severe fault this effect is even more extreme as the value of $PR_{(t)}$ will be zero and no spike passes from the synapse to the neuron. At this stage the self-repairing mechanism starts to take effect through the astrocyte process as discussed in section 2.2.2.

In neural networks, each neuron normally has more than one synapse inputs depending on the structure of the SNN/SANN. In this study each neuron is connected to 10 synapses at its input. The fault percentage that will be referred to in the following sections refer to the 10 input synapses. When a fault of 20% is injected, for example, it means that 2 of the 10 input synapses are faulty.

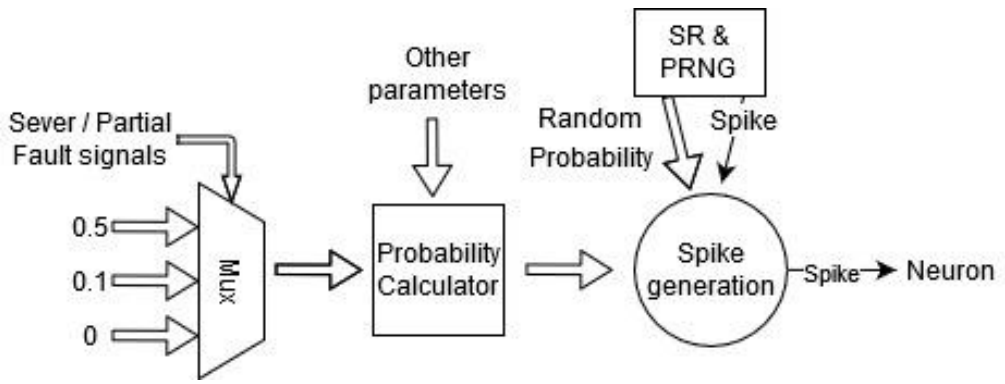


Figure 3:6: Fault modelling architecture in FPGA hardware

- **Spike and probability generation**

To generate spikes at 10Hz, the design implements 100-bit round shift registers for each synapse (10/neuron). Only one bit is asserted (its value set to '1') per shift register and the other 99 places are all zero. It worth mentioning that this is not an LFSR (Linear Feedback Shit Register), the ratio of 1s to zeros always remain the same. A 100-bit wide register has been chosen because of the Euler constant which is set to 1ms. This means to achieve a frequency of 10Hz 10 spikes should be generated for each 1000 cycles- or spike 1 per 100 cycles.

Two LFSRs generate a 64-bit pseudo random number (32 each). A sliding window that shift through this 64-bit input to generate 20 unique random probabilities for all the synapses (2 neurons- 10 synapses each). The synapses take in the random 32-bit numbers from the sliding windows and convert it to fixed-point numbers between 0-1 to act as the pseudo-random probability at the tripartite synapses.

The effect of the spike and probability generation schemes discussed above is pseudo-random spike generation in hardware. The study also implements the same methods in Matlab software to ensure fair comparison between the two models.

3.3.2 Self-repair - Reduced Bit Precision

To demonstrate the self-repair characteristics of the SANN in hardware, an experiment was conducted where the network suffers a catastrophic failure, with 80% of synapses no longer functioning. Despite this, FSA is still able to maintain more than 55% of its firing rate. This is the worst-case scenario; the network will retain a higher amount of its frequency when the fault rate is less significant as is shown in Section 3.3.2. Faults were injected into hardware by means of physically reducing the PR value of synapses to zero (classed as "severe" fault), or to a value of 0.1 ("partial" fault). That means the PR values of the synapses were physically changed on the FPGA.

The SANN in Figure 3:5 was implemented on an FPGA with the 32-bit astrocyte model. The outputs of the spike generators were fed into the presynaptic connections at the tripartite junctions. Average frequencies that were read from neuron outputs over a window of 1,000 iterations were used to assess the network. Average frequencies were used rather than instant frequencies because the output frequencies in SNNs oscillate over time. The average output frequencies of the network were observed from the two neurons, N1 and N2, as shown in Figure 3:5. Under normal circumstances, both N1 and N2 produce the same average output frequency of 7.57 Hz. Initially, the synaptic junctions were all operating normally, then a severe fault was injected simultaneously into (8 out of the 10) synapses connected to N2. The average output frequency of N1 and N2 were then observed on the FPGA. Figure 3:7 shows the average output frequency of the two neurons before and after the faults were injected. The values present on the *x*-axis represent the number of

iterations. The fault is injected after 200K iterations which is indicated by a dotted black line in Figure 3:7. As shown in the figure, the average output frequency of N2 is reduced to zero steeply at first, before increasing again due to the activation of the astrocyte self-repair mechanism. As the number of synapses damaged is (8 out 10) for N2, e.g. 80% failure, and the severity of each fault (PR forced to zero) the N2 average frequency does not fully recover to its previous values again. Since no fault is introduced to the synapses of N1, the average frequency graph fluctuates as normal.

3.3.3 Self-repair over Varied Fault Rates

In this section, tests are repeated with varied fault rates in the synapses, e.g. 10%, 20%…etc. If the fault rate was 10% it would mean that one of the 10 synapses is damaged. This is the first study in assessing the graceful degradation of SANNs in hardware. In essence, this enables a threshold to be identified where resilience of the SANN network starts to diminish as it deviates significantly from its original target average output frequency. 'Severe' and 'partial' types of faults will be evaluated.

A severe fault in the following experiments means a synapse's probability of release, $PR_{(t0)}$ in equation (2), is reduced from an initial value of 0.5 to zero, while partial fault means the probability of release is reduced to 0.1 (a non-zero value). Faults of different percentages were injected to the SANN running on the FPGA and the average output frequencies were recorded. Figure 3:8 illustrates the average output frequency of the faulty neuron (N2 of Figure 3:5) in response to different fault ratios and compares these values to a situation when no fault exists. The average frequency in this experiment is the average of the entire simulation duration in Figure 3:7. In Figure 3:8, The *x*-axis represents the rate of faults and the *y*-axis is the main average frequency of N2. Average frequencies of N2 for both FSA and the equivalent Matlab model are shown for comparison. Furthermore, the frequency of N2 is represented when no fault is present to demonstrate deviation of the average frequency

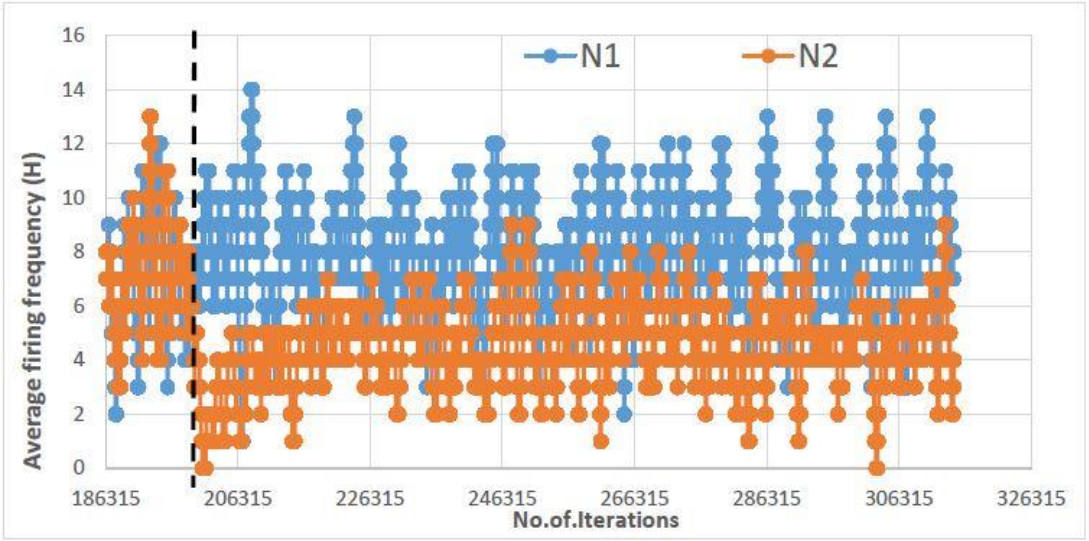
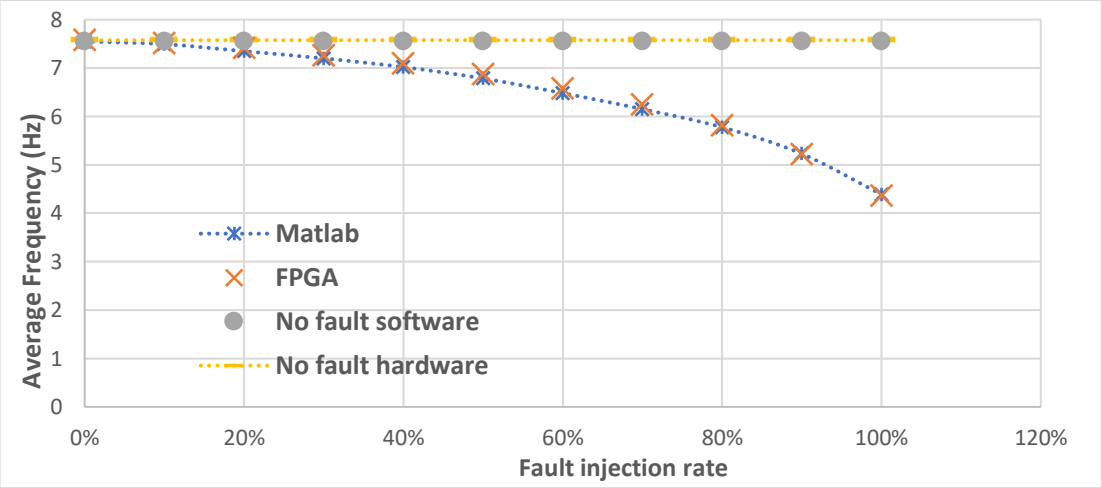


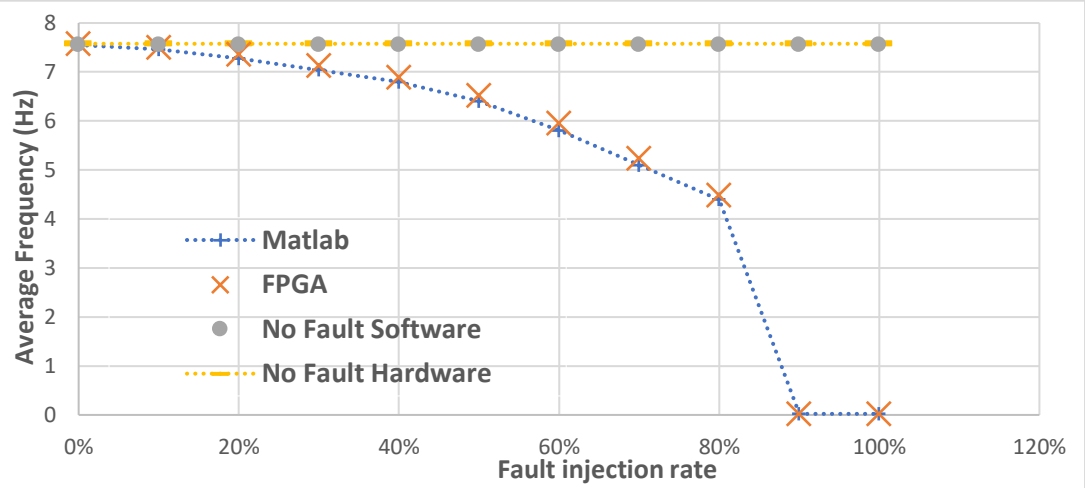
Figure 3:7 Average output frequencies of N1 and N2

trajectory when various rates of faults are inserted from the no-fault value. Figure 3:8(a) demonstrates the response of the neuron (N2) with faulty synapses (probability of release set to 0.1). According to equation (3), the tripartite synapse is still able to generate $PR_{(t)}$ in the subsequent iterations. Although this value is less when compared to a situation when no fault exists, it still can cause the synaptic junction to pass spikes to its associated neuron when the value of the random probability generator is less (see equation (3)).

It is evident from Figure 3:8(a) that even with 40% damage the output of N2 can still reach a value of over 7Hz which is close to the target 7.55 Hz (7.57 Hz for the FPGA implementation). This degradation is minimized by the self-repair mechanism; e.g. the astrocyte re-strengthens the remaining healthy and partially damaged synapses to encourage the neuron N2 to maintain its firing frequency. Note: neural networks are insensitive to slight variation in output frequencies. When all 10 synapses (100%) are deemed 'partial' faulty (100%), the neuron degrades more than 55% from its target firing frequency. However, this is a graceful degradation as shown in Figure 3:8(a). In Figure 3:8(b), severe



(a) Average neuron frequency in response to different rates of a partial fault



(b) Average neuron frequency in response to different rates of a severe fault

Figure 3:8 Average neuron frequencies in response to different rates of injected faults.

faults are injected to the synapses and the neuron output frequency degradation is more significant since a severe fault reduces PR(t) to zero, where the faulty synapse will not propagate any spikes regardless of the value of the random probability generator. In this case, at 30% damage the output of N2 can still reach a value of over 7Hz which is acceptable to the target frequency of 7.57Hz. This is clearly evident in Figure 3:8(b). In addition, the average frequency projection falls more steeply and N2 becomes inactive when the fault rate is more than 80%. Figure 3:8(b) also shows that the self-repair mechanism helps the neuron retain more than 50% of its original average frequency even at 80% fault rate. Figure 3:8 also illustrates the Matlab model results for the same network under the same testing condition where it can be seen that the fixed-point hardware model provides significant accuracy when benchmarked against the double-float Matlab implementation [6]. The key point to note is the capability of the SANN hardware to still maintain operation when over 30% of faults in the synapses are experienced. This provides a high degree of tolerance to failure via self-repair, even when the core astrocyte model's precision is reduced to minimize hardware resources.

3.3.4 FSA Acceleration Performance

The SANN model presented in this chapter is used as the basis for developing an FSA for various parameters and dimensions. Since the envisioned accelerator platform is expected to span multiple FPGAs, appropriate communication strategies will be required to ensure integral data exchange between the FPGAs. Table 3:1 shows the early results of the improved performance achieved as a result of implementing the SANN model on a dedicated FPGA at a frequency of 10 MHz. This is the maximum frequency at which the FSA can operate currently. The long combinational paths formed due to complex mathematical operations (multiplications etc.) in the astrocyte process seriously limit the operation frequency. This is due to the fact many DSPs are implemented along one combinational path to implement the astrocyte's equations. One way to mitigate this limitation is implementing frequency optimization techniques such as pipelining. Optimizing the astrocyte process will be explored in the future work. The acceleration factor in Table 3:1 shows the advantage the FSA has over previous SANN implementations in terms of simulation speedup. As can be seen, the FSA is 1,067 times faster than an equivalent software implementation. This is a significant speedup since

Table 3:1 Speedup results using different astrocyte implementations

Platform implementation	Run time	Acceleration factor
Biological neural network	600	N/A
Software simulation	64	1,067
SPANNER	7	116.7
FPGA SANN Accelerator (FSA)	0.06	1

the fixed-point, area-reduced astrocyte enables larger networks to be implemented in a smaller area.

- **FPGA Performance Trade-off**

This chapter has so far made the case of using dedicated FPGA hardware for accelerating SANN simulations in light of accuracy and speedup gained on Matlab simulations. On one hand FPGAs can accurately simulate SANNs with high accuracy and accelerated performance. On the other hand, coding for FPGAs using HDLs requires more niche skills and is much more time consuming than programming for software such as Matlab, Python etc. On top of that FPGA, at least for now, is not standard in PCs. It is extra hardware which means extra cost. Despite these hurdles, acceleration using FPGAs can prove vital in reducing simulations times specifically for systems – such as SNNs- which are used over long periods of times and don't require major modifications to the modelling beyond the initial investment. In addition, High Level Synthesis (HLS) tools, which depend on extracting synthesisable hardware from programming languages such as C and C++, can reduce the skillset required for FPGA development while speeding up the development process.

3.4 FPGA-based Configuration and Monitoring Platform for Self-repairing SANN Hardware

Spiking Astrocyte-neuron Networks (SANNs) model the adaptive/repair feature of the human brain. They integrate astrocyte cells with spiking neurons to facilitate a distributed and fine-grained self-repair capability at the synaptic level. SANNs are more complex with the addition of astrocyte cells and require longer simulation times, as they are dynamic over much longer time-scales than traditional neural networks. Therefore, there is a need for dedicated FPGA accelerators that offer reductions in simulation times. To support the acceleration of SANNs, the capability to inject faults in synaptic pathways and monitoring significant levels of neuron and astrocyte data for off-chip transmission to PC-based analysis, are required. This chapter contribution is in designing an **F**PGA-based **C**onfiguration and **M**onitoring **P**latform (FCMP) for injecting faults and capturing and analyzing data acquired from the FSA. The FCMP uses custom logic and a NIOS II based system to control fault injection and data monitoring on the FPGA. Results show accurate accelerated simulations of fault injection scenarios using FCMP with speedups of up to 65 times greater compared with equivalent Matlab implementations.

In comparison to the works mentioned in Section 2.5, the novelty of the FCMP proposed here resides in the provision of a framework that is able to inject faults into SANNs on FPGA hardware and acquire real-time network data from an FSA [1] by means of the FCMP [11].

The advantages of this work can be summarized in the following.

1- The ability to inject faults by means of software manipulation leads to significantly more time-efficient manipulation as opposed to re-synthesising the FPGA design when experimentation with a different fault rate is required.
2- Being capable of collecting on-chip FPGA data from the SANN accelerator and sending the data to PC, circumvents the limitation set by relatively small on-chip FPGA memory for data collection.

The rest of the chapter is structured as following. First, the background for this contribution through evaluating several related publications in the literature. In the subsequent section, the architecture and operation of the FCMP platform will be discussed. This will be followed by a section dedicated to experiments and results. Further improvements regarding the FCMP will be investigated in the section that follows before concluding the chapter.

3.4.1 Architecture and Operation

- **Components**

Figure 3:10 shows the overall architecture of the platform. Except for the FSA block, all other components are part of the FCMP (FCMP itself contains two main blocks, the FSA Interface and the Nios II system). Besides the main blocks, the FCMP also contains several off-chip components that are located outside the FPGA. The description of FSA and the FCMP blocks are given below.

The FSA [1] is a custom FPGA design that accelerates the simulation of a self-repairing SANN based on neuron, astrocyte and synapse models reported in [6]. In [1], [6], an astrocyte is included in an SNN for regulating the synaptic activity of neurons at tripartite synapses. Each neuron is fed from ten synapses. Faults are injected at a specific time to one or more synapses and the average output frequencies of the two neurons, N1 and N2 in Figure 3:10, are monitored. It was verified that the network can recover from faults due to the astrocyte since it compensates for lack of activity at the synapses that suffered the fault.

The FSA Interface contains several hardware sub-blocks. The two Frequency Calculators (FC) calculate the average spiking frequencies of the neurons over a time interval. The two Address Generators (AG) are essentially counters with an enable signal and a parameterizable start value. This is necessary because the Nios II based system operates as a memory mapped system thus, at least one of the AGs has to start from a base value other than zero. The State Machine (SM) is responsible for controlling the operation of the FSA and the FSA Interface(FI). The SM exchanges control and status signals with the Nios II based system. The SM operation will be discussed in more detail later in this section. The DRAM BUS blocks are combinational blocks that combine addresses generated from AGs and data from FCs along with control signals from the SM to form a complete bus that writes into the DRAM controllers.

The Nios II based system is a heavily modified version of the simple socket server design example provided by Terasic for use with DE4 boards. The Nios

II based system has an Intel Nios II soft-core processor at its heart. The system also includes a Data and Program Memory which is an SRAM, an Intel Ethernet IP which is responsible for sending Ethernet packets to an off-chip PHY chip and DRAM Controllers for communicating with off-chip DDR2 SDRAMs. Additionally, a JTAG IP that allows the system to be programmed from a PC using Eclipse environment, and a number of command and interfacing blocks are also included in the Nios II system. The command and interfacing blocks allow the Nios II processor to communicate with the FSA Interface components for passing data, status, address and control signals. The command and interfacing blocks are Reset Controller, Nios Starter, Address range Interface, Fault Injection Interface, DDR Bus Interface and Operation Complete. The purpose of these blocks will become clear in later subsections.

The architecture shown in Figure 3:10 allows for control over the start and end time of FSA simulations besides fault injections and capturing simulation data - average output frequencies - and sending this information to a PC.

- **Datapath**

As shown in Figure 3:10, the two outputs from the FSA represent the synapses of the two neurons, N1 and N2, which feed into the two FC blocks. Outputs from FCs will be combined with addresses generated by AGs and control signals from the SM at the DRAM BUS blocks. Next, the DRAM BUS output has to go through the DDR BUS Interface block since an interface is required to transfer data to the Nios II system. Also, the outputs of DDR BUS Interfaces will become part of a unified Avalon (Intel standard interface) bus before entering the DRAM Controller. The DRAM Controller takes address, data and control signals and generates appropriate outputs so that data is transferred to and from the external DDR2 SDRAM.

- **Operation**

The flow chart given in Figure 3:9 explains the operation of the FCMP. The Nios II processor controls the operation of the custom hardware circuit by means of a set of parameters, control and reset signals which are passed during the initial state. One of these parameters is the fault ratio, which is the number of damaged synapses to the total number of synapses in the FSA. Another parameter is the time at which the faults occur (through the Fault Injection Interface in Figure 3:10).

Moreover, the amount of data to be written into the DDR2 SDRAMs can also be passed to the FSA Interface through the Address Range Interface. This value equals the number of cycles the FSA runs for. Furthermore, through the Nios starter block in Figure 3:10, the Nios II processor sends a start signal to the SM. Once this command is received, the SM enables the FCs and AGs along with issuing write commands to the DRAM controllers. This commences writing of the average frequency data from the FCs into the external DDR2 SDRAMs at addresses generated by the AGs. This process continues until data

is written into a range of addresses defined by the parameter passed from the Address Range Interface at the reset state. After this process, the SM sends Nios II a signal through the Operation Complete block and the Nios II processor starts to send data from the external DDR2 SDRAMs to a PC through Ethernet. Figure 3:11 shows a simplified sequence diagram for passing of commands and data between Matlab, Nios II and the FSA hardware.

The operation of FCMP platform in Figure 3:9 is separated into 3 parts; handshaking, computation and data movement. The next sections reveal that most of the time is consumed in data movement between the Nios II system and the PC. The working algorithm and the architecture in Figure 3:10 indicate that the, despite the addition of an FPGA accelerator – the FSA – the working principles are still bound by Von Neuman architecture. Studies like this, however, can help to further understand the brain functionality with the aim of using the brain's working principles to create novel efficient neuromorphic computers that break the bounds of Von Neuman architecture.

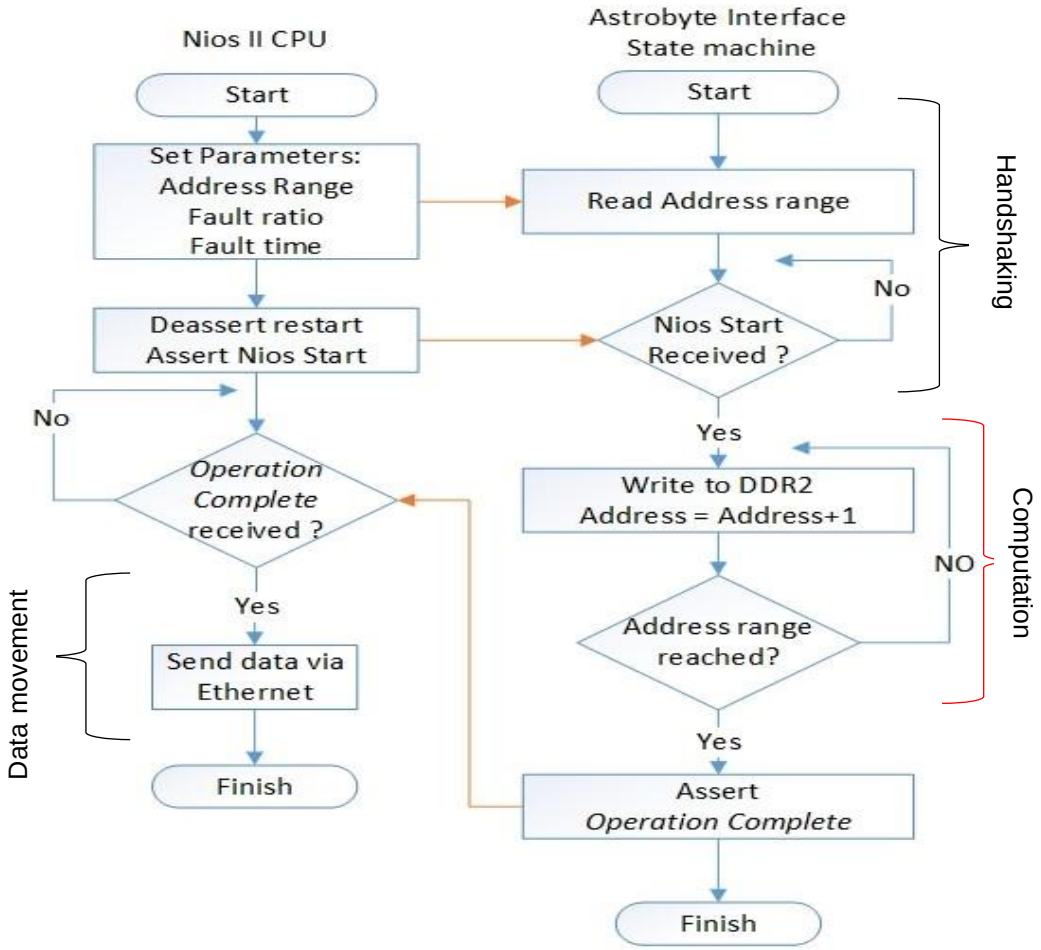


Figure 3:9 Algorithm outlining the FMP operation

3.4.2 Experimentations and Results

- **Assessing Data Integrity**

Several experiments were performed to ensure that data captured using the FCMP is accurate and remains undistorted while transferred from the FSA to external memory, and then onto the PC via Ethernet. Matlab software was used to communicate with the Nios II soft processor which managed the fault injection and recording the data communication between the SANN and SDRAM. To validate the data management, the same experiment that was carried out in [1] and recorded using SignalTap II was repeated using the FCMP. Figure 3:12(a) shows data collected from Intel SignalTap II and the FCMP.

The *x*-axis is the number of clock cycles, or iterations, and the y-axis represents the average output frequency of the neurons. The simulations were initially run without injecting faults. Subsequently, faults were inserted, damaging 80% of synapses connected to N2. As the FCMP had not been developed in previous work, Intel In-System Sources and Probes Editor had to be used to inject faults. The timing of these faults was decided before synthesizing the design from the HDL code. In the current work, the FCMP allows for inserting faults at different ratios at a time chosen by the user. The faults are indicated by a black vertical line in Figure 3:12(a) and Figure 3:12(b).

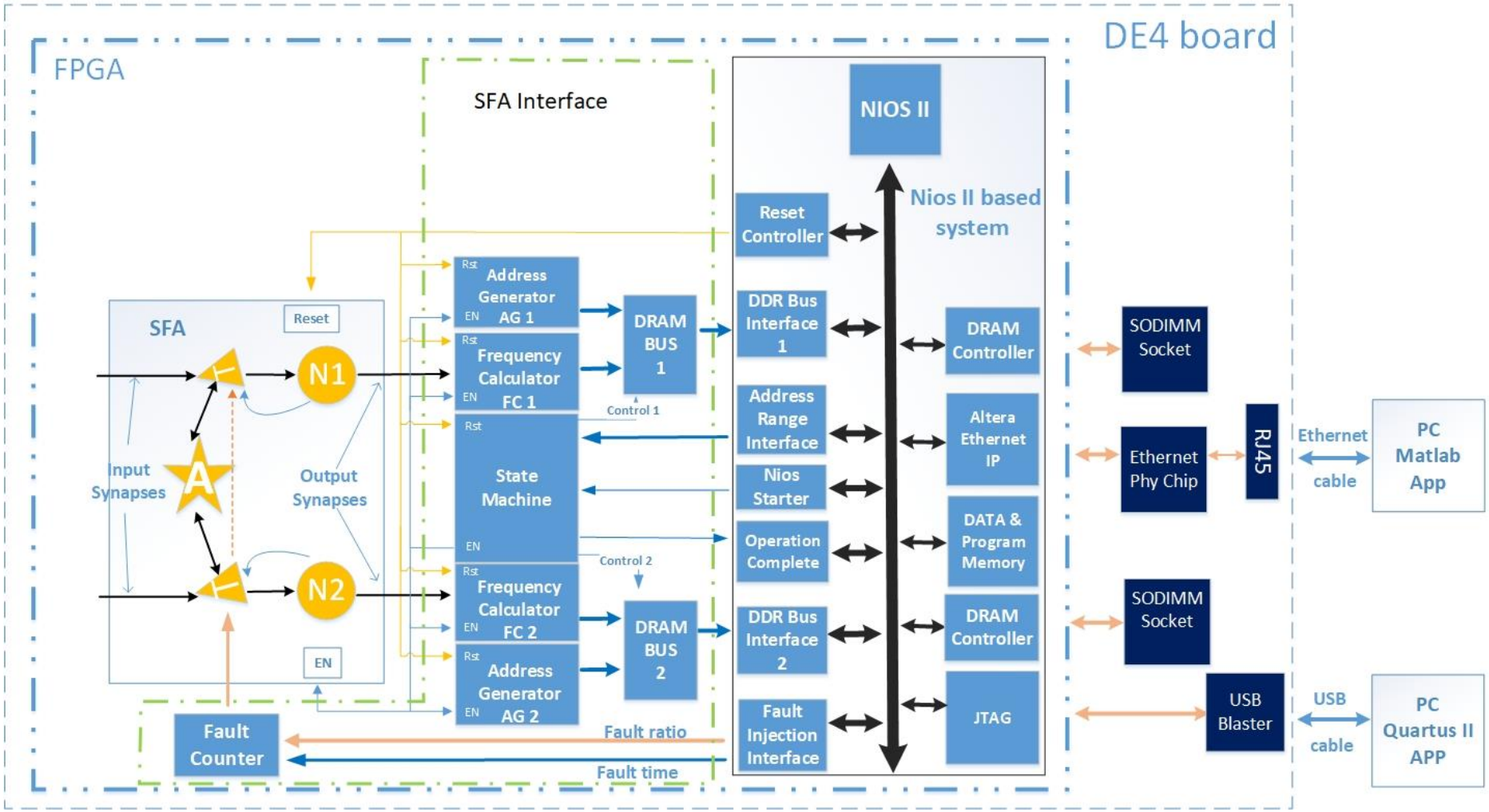


Figure 3:10: FMCP Architecture and FSA

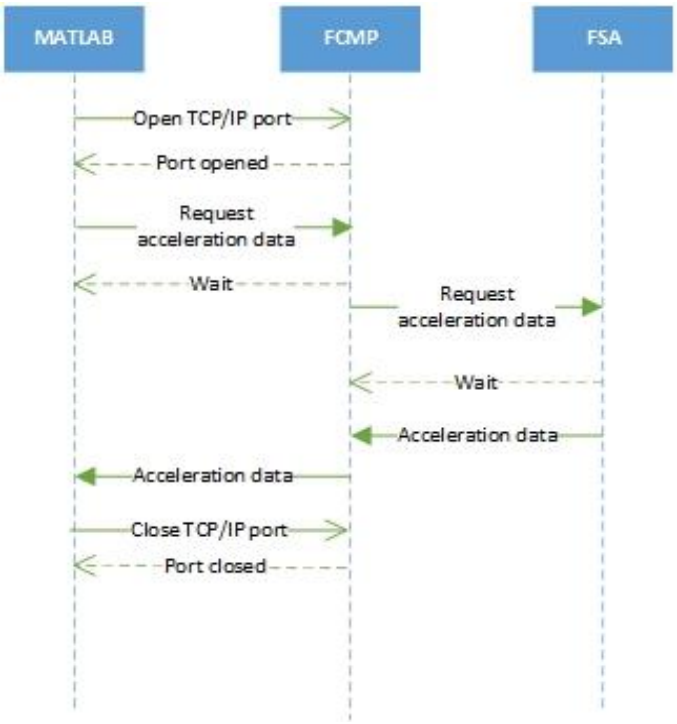


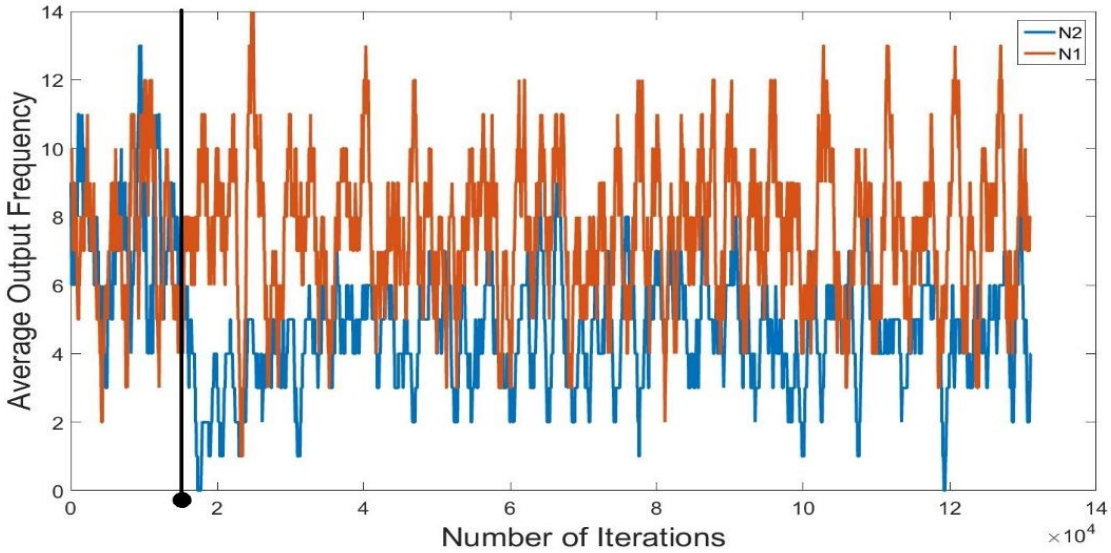
Figure 3:11 Interactions between Matlab, Nios II and FSA

At first, neuron N2 will see a sharp decrease in average output frequency but then recovers as a result of the self-repair mechanism that is regulated by the astrocyte. Since the fault ratio is very high, i.e. 80%, the average output frequency does not go back to its pre-fault level. Data from this experiment was analysed after it was transferred to the PC. Then the data was plotted against data recorded by using SignalTap II in the original SANN accelerator paper [1]. Figure 3:12(a) shows data acquired using the FCMP. As no loss of data happens when using the FCMP, acquiring data using Intel SignalTap II yields a similar response.

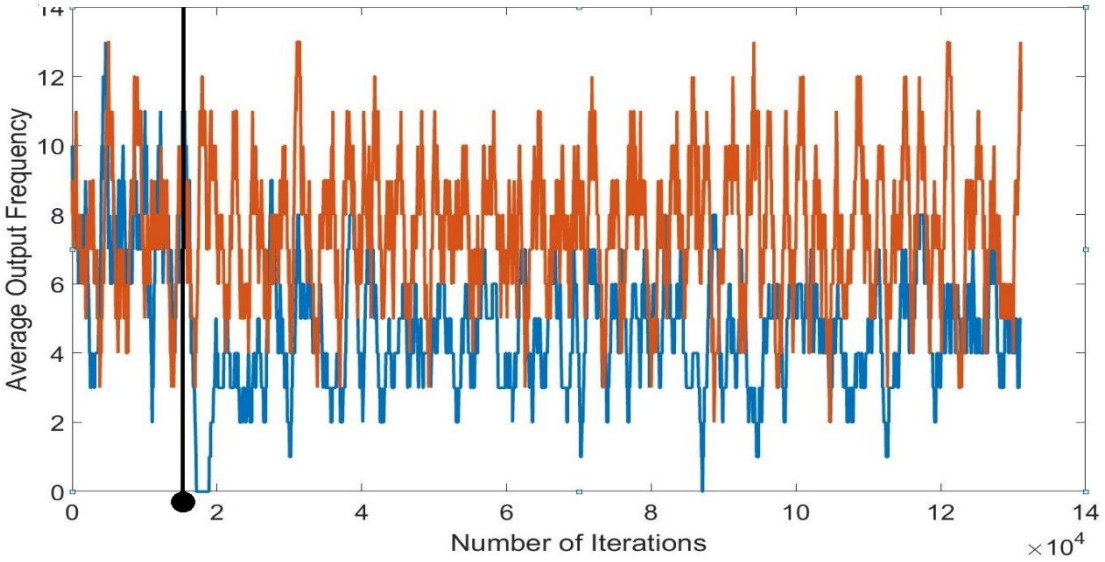
Figure 3:12(b) shows results from the Matlab implementation of SANN. Clearly, Figure 3:12(a) and Figure 3:12(b) show the same astrocyte behaviour, with the marginal difference in the trajectories of data presented in the figures due to the difference in implementations, i.e. the Matlab model uses double floating-point precision and the FSA uses an area optimized 32-bit fixed-point hardware implementation [1].

- **Acceleration**

Table 3:2 presents a comparison between simulating a SANN using Matlab (software) and the FSA platform (dedicated FPGA hardware). The column *Biology* represents the actual biological timescale of the simulations; *Iterations* is the number of times the equations representing the SANN must be calculated or the number of cycles FSA needs to run to meet the set biological time-scale. This value is x1000 the biological time since a time step of 10^{-3} is selected [6] for the Euler method. Both *Matlab* and *FSA* columns show how much time a Matlab software model [6] and an equivalent FPGA accelerator [1] respectively,



a) Matlab implementation results



b) Data collected using FCMP and SignalTap II

Figure 3:12 Comparison between data collected using Matlab and FCMP

take to execute the simulation for the corresponding biological time-scale. The Matlab software runs on Windows 10 PC with a 3.4 GHz Intel Core i7-2600 CPU (Octa-Core), SSD drive, and 16 GB RAM. Also, the table shows the time it takes the FCMP to transfer data generated by the FSA to a PC, shown under the FCMP column. The column FSA + FCMP provides the total time from the start of running the design in FSA to collecting the data on the PC side. The Speedup column shows the speedup gained by using FSA and the FCMP, in comparison to the Matlab software. The overall speedup is in the order of between x50-x65 depending on the number of iterations the designs are run for. It is worth mentioning that the values under Matlab, FCMP, FSA + FCMP and Speedup can vary from one PC to another and also from time to time as they are PC and Windows OS dependent.

Studying Table 3:2 shows that using the FCMP to transfer simulation data from the FPGA to a PC will introduce a significant overhead to the FSA time. However, up to x65 speedup rate is possible which is significant for a wide range of applications including the study of how astrocytes impact on other neurological conditions such as Alzheimer's [123].

Table 3:2 Acceleration gain by using DRAM

Biology (Sec)	***Iterations (Cycles)***	***Matlab (Sec)***	***FSA (Sec)***	***FCMP (Sec)***	***FSA + FCMP (Sec)***	***Speedup***
400	400 K	153.7	0.04	~3	~3.04	50.5
1,000	1 M	381.5	0.1	~6.7	~6.8	56.0
5,000	5 M	1960	0.5	~30	~30	65.2
10,000	10 M	4095	1	~62	~63	65.0

- **Reducing Sampling Rate (Under-sampling)**

When simulating biology, it is possible to drop the sampling rate to one sample per ten computations or one sample per hundred computations as the rate of change in biology is slow, e.g. astrocyte dynamics operate in hundreds of seconds. That would allow simulating the SANN for longer periods while maintaining significant speedup since we would need to transfer less data to the PC side. Table 3:3 shows the effect of under-sampling for biology time of 100,000 seconds. As an instance, if every one-in-ten samples are recorded, the overall simulation and acquisition time will be reduced from around 724 seconds to ~63 seconds. This provides a significant reduction in the time communicating data off-chip with the FPGA. This feature allows the FSA to vary the accuracy of simulations as required. Exploring the self-repair aspect for fault tolerant networks can require less samples discussed above, however, for simulating neural experiment to study astrocyte dynamics between neurons for example, can require the higher data sampling rate. The under-sampling provides a trade-off between simulation accuracy and speedup capability.

Table 3:3 Under-sampling evaluation

Sampling	**Iterations (Cycles)**	**FSA (Sec)**	**FCMP (Sec)**	**FSA +FCMP (Sec)**
1/1	100 M	10	~714	~724
1/10	10 M	1	~62	~63
1/100	1M	0.1	~6.7	~6.8

- **Enhancing FCMP performance**

As can be seen from Figure 3:10, two external DRAM memories have been used for storing SANN simulation data before moving the data to a PC by Ethernet protocol implemented as part of the FCMP. In subsequent work, the external DRAMs were replaced with internal SRAMs similar to the program memory in Figure 3:10. The FPGA SRAMs have less storage capacity than the external DRAMs. The implication is that instead of saving the complete simulation data before sending it to a PC, the required amount of data has to be partitioned over a number of times, depending on the size of the SRAMs. For example, should simulation data for 10M iterations be stored and then sent to a PC, using the current configuration with external DRAMs (1GB each) as per Figure 3:9. The simulation data can be completely saved to the external DRAMs and then be sent to a PC via Ethernet as one patch. This is because storing 10M iterations with 32-bit data width requires 40MB of memory, which can be easily accommodated by the external DRAMs (1GB each). However, as an instance, when 200KB SRAMs are used, only 50K iterations can be stored each time, which signifies that the complete simulation data has to be sent in 200 parts. Despite this, replacing DRAMs with SRAM significantly increased acceleration since the advantage gained by removing the DRAM bottleneck and using fast SRAMs far outweighs the time overhead introduced as the result of partitioning the simulation data and sending it parts. Table 3:4 shows that the using SRAMs result in x3.1 – x3.6 faster operation than DRAMs results reported in Table 3:3, both designs operating at 100 MHz.

Table 3:4 Acceleration gain by using SRAM at 100 MHz

Biology (Sec)	**Iterations (Cycles)**	**Matlab (Sec)**	**AstroByte (Sec)**	**FCMP (Sec)**	**FSA +FMP (Sec)**	**Speedup**
400	400 K	53.72	0.04	~0.81	~0.85	180.8
1,000	1 M	381.5	0.1	~1.9	~2	190.75
5,000	5 M	1960	0.5	9.2	~9.7	202.67
10,000	10 M	4095	1	~18.4	~19.4	211

Another opportunity for further speeding up the simulation and data acquisition process was observed when the external DRAMs were replaced with SRAMs. This is obtained by increasing the operation frequency of the Nios II system from 100 MHz to 200 MHz, as shown in Table 3:5, in which speedups of between x3.8 – x4.9 can be observed in comparison to the DRAM-based Nios II system. Comparing the acceleration of FSA without FCMP (Section 3.3.3) and with FCMP reveals that even in the case of using SRAM, the Nios II soft processor still cases significant overhead in terms of speedup gained. Finding

solutions to mitigate the overhead caused by FCMP will be discussed in the future works in Chapter 6.

Table 3:5 Acceleration gain by using SRAM at 200 MHz

Biology (Sec)	**Iterations (Cycles)**	**Matlab (Sec)**	**AstroByte (Sec)**	**FCMP (Sec)**	**FSA + FCMP (Sec)**	**Speedup**
400	400 K	53.72	0.04	~0.58	~0.62	247.9
1,000	1 M	381.5	0.1	~1.43	~1.53	249.3
5,000	5 M	1960	0.5	~7.96	~8.21	246.23
10,000	10 M	4095	1	~15.67	~16.67	245.6

3.5 Chapter Conclusion

In summary, a fixed-point hardware model of an astrocyte has been implemented on FPGA hardware, and results comparing the astrocyte hardware with the software model. Based on the FPGA resource utilization and the accuracy of the hardware astrocyte model, a resolution of 32-bits was identified. This model was used in the FSA to evaluate the self-repairing capability in FPGAs. Results obtained directly from the FPGA using Intel SignalTap II demonstrates this self-repair capability together with the accuracy of the fixed-point hardware implementation when benchmarked against the double floating-point software model [6]. This provides scope for implementing much larger SANNs on FPGAs and integrating with existing Networks-on-Chip.

A novel FPGA based Configuration and Monitoring Platform (FCMP) was also presented in this chapter which configures and captures data from FSA. Comparisons were made between the FSA and Matlab software, to assess the overall hardware speedup with FCMP inserted into the system. Results demonstrated over x65 speedup using hardware accelerated SANN with the FCMP. Additionally, using FCMP, an analysis from the effects of under-sampling were discussed which highlighted possible trade-offs between reduced simulation accuracy and increased speedup. Further speedup was gained by replacing slow external DRAM with fast internal SRAMs. As indicated in Table 3:5, the accelerated SANN with FCMP can achieve speedups of up to x249.3 when compared to the equivalent Matlab implementations.

In Chapter 4, a novel NoC-based multi-FPGA platform will be implemented which incorporates both the FSA and FCMP developed in this chapter to create AstroByte. The FCMP will be used for injecting configuration packets into the platform and to collect monitoring data through the mechanisms explored in this chapter.

Chapter 4: A Networks-on-Chip based Multi-FPGA Infrastructure

The Networks-on-Chip (NoC) paradigm is one popular solution to the performance limitations of computing architectures that utilize a shared bus. As the number of IP cores has increased over the last decade due to soaring computational demands and the variety of IP cores used, the shared bus has become the bottleneck of many computing systems in terms of performance, scalability and power consumption. On the other hand, NoCs use an internet-like communication-centric structure that out-performs a shared bus as far as scalability, efficiency and re-usability are concerned [17], [125], [126].

This research developed a NoC architecture as part of a scalable, programmable and re-usable multi-FPGA platform, called AstroByte. The platform focusses on accelerating simulations of self-repairing SANNs by using off-the-shelf FPGA boards that are interconnected by means of a custom protocol. AstroByte is the third contribution of this work. Some of the results and overall architecture design are published in the paper *"AstroByte: A multi-FPGA Architecture for Accelerated Simulations of Fault-tolerant Spiking Astrocyte-Neuron Networks*" at the Design, Automation & Test in Europe (DATE) conference in 2020.

This chapter presents the AstroByte platform is and the rationale for the use of various clock frequencies, the NoC data format, NoC router microarchitecture and its various components are presented. Both SANN hardware and FCMP interface architectures are also reported.

The contributions presented in this chapter can be summarized as follows:

1- A new fully scalable NoC-based multi-FPGA architecture for accommodating SANN hardware acceleration.
2- A novel router for facilitating NoC data routing across multiple-FPGAs and clock domains.
3- Novel interfaces for integrating FSA, FCMP and the multi-FPGA NoC architecture into one platform, AstroByte.

4.1 AstroByte Platform Overview

Figure 4:1 shows an overview of the AstroByte platform. It contains a NoC infrastructure and computing cores. Each node in Figure 4:1 is located on a separate FPGA board and contains a NoC router, a computing node and an interface between them. While chapter 3 discusses the design of the computing cores, this chapter details the NoC infrastructure. The IPs developed in this

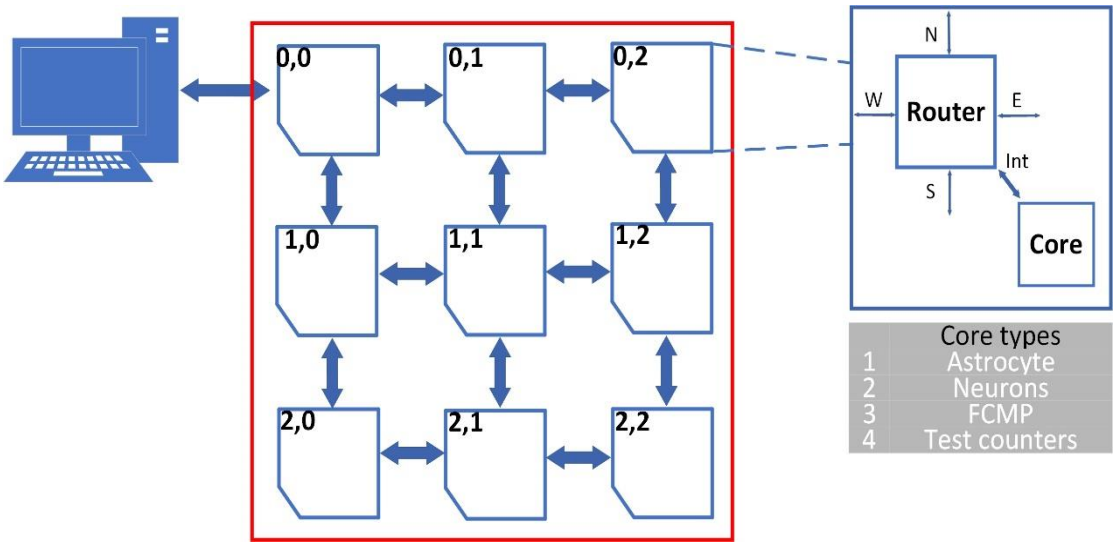


Figure 4:1 An overview of AstroByte platform

chapter are the NoC router and the two types of interfaces that exist between the router in the computing core. One of the interfaces is for use with the FSA elements (or test counters) and the other is for use with the FCMP.

Several practical and architectural reasons motivated the configuration in Figure 4:1. The AstroByte platform deploys mesh topology because of relatively simple routing engine and scalability. Additionally, each node in the network contains a computing node, this means the distance between neighbouring nodes is always 1 hop in any direction, reducing node-node latency. In terms of practicality, the DE4 boards (Figure 4:2) that this study uses have 4 SATA ports, which is the number of connections that is required for mesh structure per node. There are two HSMC (High Speed Mezzanine Card), 4 Ethernet ports and a PCI (Peripheral Component Interconnect) slot per board. One could utilize adaptor cards to convert these connections to extra connections to increase the channel bandwidth. Another use of extra ports would be dedicated connection for transferring the credit packets (discussed later) which will positively impact the throughput of each channel. However, this was not explored since for the

Figure 4:2 Terasic DE4 board

prototype SANN realized in this study the available bandwidth is sufficient and would make justifying the additional cost and effort spend difficult. It worth mentioning that HSMC is Terasic's proprietary port and purchasing any form of adapter cards or even just cables can be costly. Furthermore, the GPIO pins available on the boards are too slow for giga bit transfers. In summary, the structure of Figure 4:1 was the most suitable considering the advantages provided by mesh structure, development time and funding available.

4.2 AstroByte frequencies

The AstroByte platform consists of three main blocks, the FSA, the NoC routers, and the FCMP, operating at respective frequencies of 10 MHz, 200 MHz and 150 MHz. The FSA frequency is 10 MHz because of long combinational paths that contain several DSPs. The NoC routers operate at a frequency of 150 MHz, limited by the GBX blocks at input and output ports of the NoC router.

The GBX blocks frequency are in turn limited by the SATA physical connections. More discussions about GBX blocks, their operation and frequencies are given in the appendix.

4.3 Intel GXB transceiver IP

Intel Gigabit Transceiver Block (GXB) is a highly reconfigurable IP for utilizing multi Gigabit serial communications between Intel FPGAs and external devices [127]. The components can be different depending on the mode and protocol of the transceiver. For a bespoke and low overhead utilization, AstroByte uses *Basic* mode and a custom synchronization protocol, as opposed to standard protocols (UART etc.). In *Basic* mode, the transceiver doesn't is extremely flexible in terms of the features and blocks that can be included or omitted.

The transmitter has two separate paths, one used for transmission (GXB_Tx) and the other for receiving data from the upstream node to achieve a full duplex transmission.

The transmitter data path is responsible accepting FPGA fabric user data and serializing it before transmitting it off-FPGA to the downstream node. The receiver data path supervises converting the incoming serial data to a 32-bit word before forwarding to the FPGA fabric. Similar to the transmitter data path, the receiver data path contains several blocks.

The appendix provides detailed information about the architecture of the transceivers and in-house controller modules that are necessary for the GXB IPs operation.

4.4 AstroByte NoC Data Format

An appropriate data format ensures that packets, whether data or control, traverse from the source, go through the NoC structure and arrive at the destination within a timeframe.

Figure 4:3 presents the data format in AstroByte platform which consists of two types of packets, namely data and credit packets.

The Data packet consists of two flits, a header flit and a data flit. The two flits always move through the NoC one after each other. Apart from at the destination node, when the header flit is finally disposed of, at no point in the NoC are the two separated. This means the AstroByte platform utilizes store and forward flow control mechanism. It has already been discussed in section 2.6.3 that wormhole flow control is more efficient than store and forward flow control. However, this applies more to cases when packets contain a header flit along with more than several data flits, and in most cases a tail flit. On the other hand, each data packet only contains two flits in AstroByte, which eliminates the difference to a large degree.

A two-flit data packet was implemented since the hardware implementation of the FSA works at 10MHz while the NoC router operation frequency is 150MHz. The outputs of the FSA elements (astrocytes and neurons) are what constitute a data flit. If the data format was, for example, five data flits, no data could have been transferred until the slow FSA frequency completed five cycles. That would mean losing a part of the NoC bandwidth while waiting for the FSA to generate the necessary data. This combined with the fact that AstroByte is able to send spikes and 2AG from neurons to separate (multiple) destinations makes it more efficient to send what is calculated in cycle n to their separate destinations via the NoC infrastructure while the FSA calculates the iteration n+1. This ensures quick dispatch of data flits and packets and reduces the latency between source and destination nodes considerably.

- **Header flit**

Figure 4:3 illustrates the data packet structure. HF [1:0] (Header Flit bit positions 1 and 0) are set to “11”. The following two bits, HF [3:2] are reserved bits and their values are ‘0’.

HF [7:4] contain the control bits. This field will be decoded at the destination nodes to determine what information is carried by the data flit that follows. In case of Start and Stop commands the data flit is all zeros because it is not needed.

The control field is used by the FCMP platform to configure the FSA to designate the data packets it sends as *2AG* or spikes in case of neurons or *eSP* when the computing core contains an astrocyte.

HF [13:8], HF [19:14], HF [25:20] and HF [31:26] are source address X, source address Y, destination address X and destination address Y, respectively.

HF [32] designates whether this word is a header word. This is used in conjunction with a fault recovery mechanism that will be discussed in later sections.

➢ **Credit packet**

The credit packet carries information about the free buffer space at the downstream router. The credit packet consists of one flit, the Credit Flit (CF). CF [1:0] hold 01 and CF [31:26] contain the free buffer value the downstream router sends. CF [25:2] and CF [32] are all zeros.

4.5 AstroByte Router Architecture

This section introduces the NoC router microarchitecture. Figure 4:4 illustrates the router consisting of 5 ports, North, South, East, West and an internal port for communicating with the computing element. The computing element can be either an FCMP, neuron element, or an astrocyte core. There are differences between the components in router port 0, the internal computing port, and ports 1-4. The justification for the difference will be discussed later in section 4.4.

The router is comprised of several key blocks. The Input Stream Controller (ISC) is the first block after the transceiver's receiver part and is responsible for de-multiplexing the incoming data stream, i.e. separating data and credit packets. Arbiter Track Table FIFO (ATT FIFO) stores the credit values while Input Channel FIFO (InCh FIFO) stores the data packets. The Arbiter Track Table (ATT) implements a specific scheme to keep track of the amount of free space in the downstream router's InCh FIFO. The Arbiter utilizes a fair Round-Robin arbitration when giving access of an output channel to the computing requests from the input channels. The Routing Engine (RE) block is responsible for decoding the address of the header packet and issues a request to the appropriate output port. The Crossbar switch (Xbar) implements multiplexers to allow the input channel to access the output channels. The Output Channel FIFO (OutCh FIFO) stores data packets to be forwarded to the downstream router. The RdCNTRL and WrCNTRL blocks perform reading from and writing to Inch FIFOs and OutCh FIFOs respectively. The Fault CNTRL blocks monitor OutCh FIFOs and InCh FIFOs to detect a particular fault that may occur if AstroByte platform is reset while simulations are running. The Input Channel

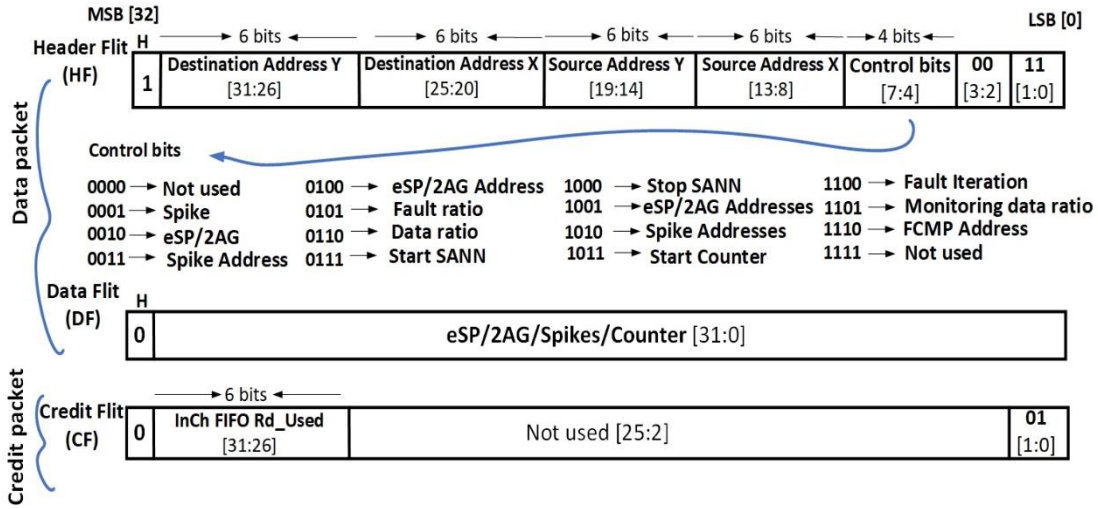


Figure 4:3 an overview of AstroByte data format

Track Table (ICTT) block monitors the InCh FIFO usage and creates credit packets, then stores them in Input Channel Track Table FIFO (ICTT FIFO). Output Stream Controller (OSC) multiplexes the physical output channel to accommodate both credit packets and data packets before forwarding them to the transmitter part of the transceiver. Port 0 does not include ATT FIFO, ICTT and ICTT FIFO while ports 1-4 contain all the sub-blocks.

The following subsections will describe each of the above-mentioned components in detail.

- **Different clock domains**

This work implements Intel's GXB to facilitate multi-FPGA communications between different DE4 boards. The GXB contains two main blocks, a transmitter, denoted by GXB_Tx and a receiver (GXB_Rx). The NoC router has to receive data using the clock provided by GXB_Rx at first before crossing to its clock domain using clock domain crossing techniques. Similarly, in the last stage, the data has to cross to the clock provided by GXB_Tx before the transceiver can carries out transmission.

Overall, there are 3 main clock domains per router. The GXB_Rx, GXB_Tx, and the router clock domain as in Figure 4:4. In actual fact, all the clock frequencies are the same, 150 MHz, however, there are phase differences between them due to the phase variations GXB_Rx and GXB_Tx recovered clocks experience, necessitating dealing with them as different clock domains.

In Figure 4:4, GXB_Tx and GXB_Rx are placed at the two ends of the router ports 1-4 for better visualization although they are the same GXB working in duplex mode.

Additionally, the figure indicates that port 0 does not make use of GXBs insofar it communicates with FSA via the FSA Interface (SI), both located on the same FPGA as the router, and both falling into the same clock domain.

In this research, the function of DCFIFOs is decoupling different clock domains whenever necessary. Furthermore, the DCFIFOs have two-stage synchronization registers as it is the minimum requirement for Intel FIFO IP [128] for avoiding metastability [127] when synchronising between two asynchronous clocks (clocks from different sources and/or having different phases). The appendix provides more information about using DCFIFOs in clock domain crossing.

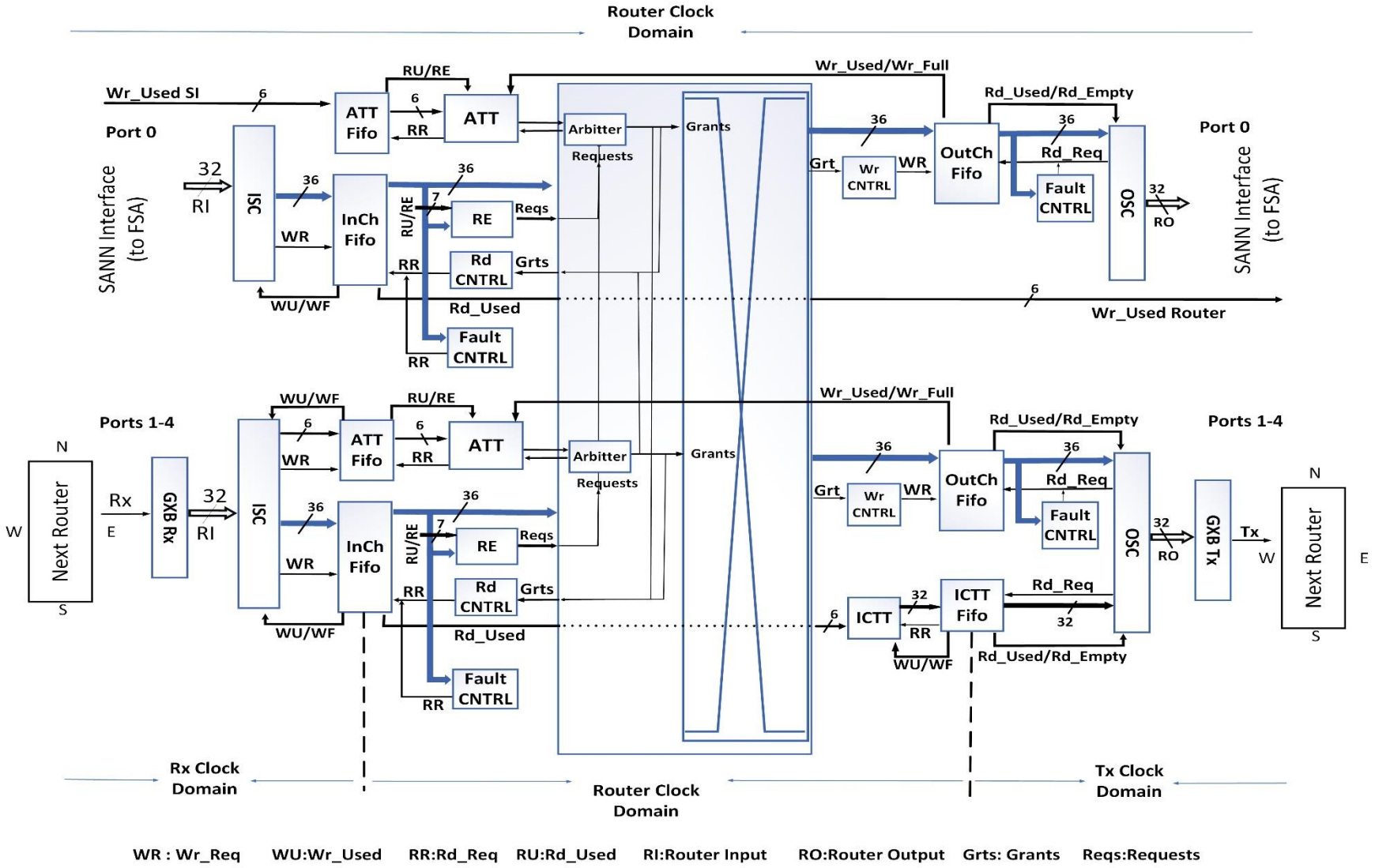


Figure 4:4 NoC router overall architecture

- **Input Stream Controller (ISC)**

The ISC is responsible for differentiating between data packets and credit packets (refer to section 4.3), effectively de-multiplexing the incoming data stream.

The main input to ISC is the GXB receiver channel parallel output for ports 1-4 and SI output for port 0, both are 32 bits wide. The ISC block checks the input stream data and designates them as either credit or data packet by checking Router Input (RI). As per the AstroByte data format, studied in section 4.3, the word is a credit packet If RI [1:0] is 01 while it is a data packet when RI [1:0] is 11. ISC discards other words with RI [1:0] holding a different value, except for the second word of a data packet, the data flit.

When the ISC detects a credit packet, (always one flit), it will de-packetize it and stores the credit value in the ATT FIFO. This is done by separating the credit value from the credit packet and forwarding it on the ATT FIFO bus along with pulling the associated wr_req to logic '1'.

If there is a data packet, the ISC performs no de-packetization. The current flit and the next flit will be forwarded to the InCh FIFO since the header flit and data flit are always transposing together as discussed in section 4.3. This calls for asserting wr_req signal controlling the write operation of the InCh FIFO for two clock cycles.

The ISC attaches an additional bit to the MSB in case of a data packet detection, denoted by H in Figure 4:3, converting both the header flit and data flit to 33 bits. The value for the H is 1 in case of a header flit and is 0 for data flit. This approach serves as a fault detection and avoidance measure that will be investigated in a following section dedicated to the Fault CNTRL block.

Referring to Figure 4:4, the data path for data packets are 36 bits wide. This is because Intel DCFIFO IPs don't support 33-bit words. The closest supported width is 36. The rest three bits are always 0.

The ISCs belonging to ports 1-4 are clocked by the transceivers recovered clock belonging to this particular channel receiver (GXB_Rx). Port 0 ISC is clocked by the router clock.

- **Arbiter Track Table FIFO (ATT FIFO)**

This is a Dual Clock FIFO (DCFIFO) with depth of four words for decoupling the GXB_Rx clock domain, the write side, and the router frequency domain, the read side. It contains the credit packets sent by the downstream router and retrieved by ISC. The choice of four words as the depth is because the credit word does not have to be held for many clock cycles and will be dispatched to the ATT module as soon as it is available on the read side. The ISC controls the write side of the FIFO while the ATT controls the read side. This block is absent from port 0 because the SI that communicates with port 0 works on the

same clock domain of the router, thus cancelling the requirement of a DCFIFO for decoupling the two clock domains.

- **Input channel FIFO (InCh FIFO)**

Similar to ATT FIFO, Inch FIFO operates in dual clock mode and is used for clock domain crossing between the channel's GXB_Rx clock domain and the router's clock domain. Unlike ATT FIFO, however, clock domain crossing is not the sole purpose of Inch FIFO. As explained in section 4.3, a type of buffered flow control, specifically store and forward flow control, has been used, indicating that buffers are required in any case. Instead of decoupling the different clock domains using a four-word DCFIFO and then storing the data packets in a separate buffer, it was decided to merge the two processes into a 64-word deep DCFIFO, acting both as decoupling mechanism and also buffer for packets. This approach reduces the number of accesses required to read data from the FIFOs and use of FIFO control blocks.

The depth of Inch FIFO is set to 64 words deep to accommodate the 32 data packets, each composing of a header flit and a data flit. Selecting a depth of 64 was carried out through experimentation as it was perceived that a depth of 32 words (16 packets) wouldn't fully exploit the available bandwidth while a 128 deep FIFO would not add any improvement to the throughput. The read side of the InCh FIFO is controlled by RdCNTRL while the write side is controlled by ISC.

The *Rd_used* and *Wr_used* indicate the number of words used available on the read and write sides respectively.

- **Read Controller (RdCNTRL)**

RdCNTRL is a relatively simple controller that accepts the Grant signal from one of the Arbiters and extends it for two clock cycles to fulfil the data format requirement of sending the stored two-flit packet.

- **Arbiter track table (ATT)**

Figure 4:5 shows a block diagram of the ATT. The architecture is composed of a Time stamp counter, a 32-word deep FIFO, a block to calculate the value of *x* (a variable that will be discussed later), a track table resister and several arithmetic and comparison blocks. The depth of the FIFO is 32 which is equal to number of packets the InCh FIFO of the downstream router can accommodate.

Figure 4:6 illustrates the pseudo code representing the mechanism that ensures the ATT precisely tracks the number of free spaces in the downstream Inch FIFO. ATT accounts for the latency of the credit packet and credit value introduced as it goes through many components on its way to ATT input. In the downstream router, where the credit value is initiated and then packetized, it has to go through (ICTT (packetized) → ICTT FIFO → OSC → GXB_Tx). In the

current router-upstream it goes through (GXB_Rx → ISC (de- packetized) → ATT FIFO) before entering the ATT port, as per Figure 4:4.

The operation of ATT is as follows. The timestamp counter increments each clock cycle: step (1) of the pseudo code. Whenever the *Sent* is asserted (2) the timestamp is added by 75 (will be explained later) and stored in the SCFIFO (3). If sent is not asserted, then no value is stored in the FIFO (4). As long as the FIFO is not empty (5), the value at its output port will be compared against the timestamp counter. If the time is equal or greater than this value (6), meaning that 75 clock cycles has passed since the data packet has been forwarded to the next router, a signal (*Equal)* is issued (7), otherwise no change occurs (8). Step (9) subtracts *x,* which is calculated in the *x* calculator block of Figure 4:5 from the credit value – it is stored in the ATT FIFO. X is a variable increased by 2 when a *Sent* signal is detected, decreased by 2 when Equal is logic high, and is unchanged otherwise as the net effect is zero when they are both '1' or '0' (steps 10-13). The *Track table* value equals the credit value from ATT FIFO, subtracted by X, as shown in the pseudo code.

This arrangement ensures that when the current router forwards a data packet to the next router, the ATT module subtracts the *Track table* by 2 to account for the space the data packet will take in the downstream InCh FIFO. When the data packet reaches its destination and a credit packets returns reflecting the status of the InCh FIFO with the sent data packet considered (decided to take 75 cycles), the ATT stops subtracting the *Track table* by 2.

In order to allow the Arbiter to grant a request, there should be enough space in both the downstream router's InCh FIFO (at least 4 words) and current router's OutCh FIFO – at least 2 words. The values 4 and 2 are chosen conservatively to protect against writing into a full FIFO (see appendix).

The output from ATT is a single bit control signal that the Arbiter receives. The ATT sets this signal to '0' if it concludes that the Arbiter can answer to requests and to '1' otherwise.

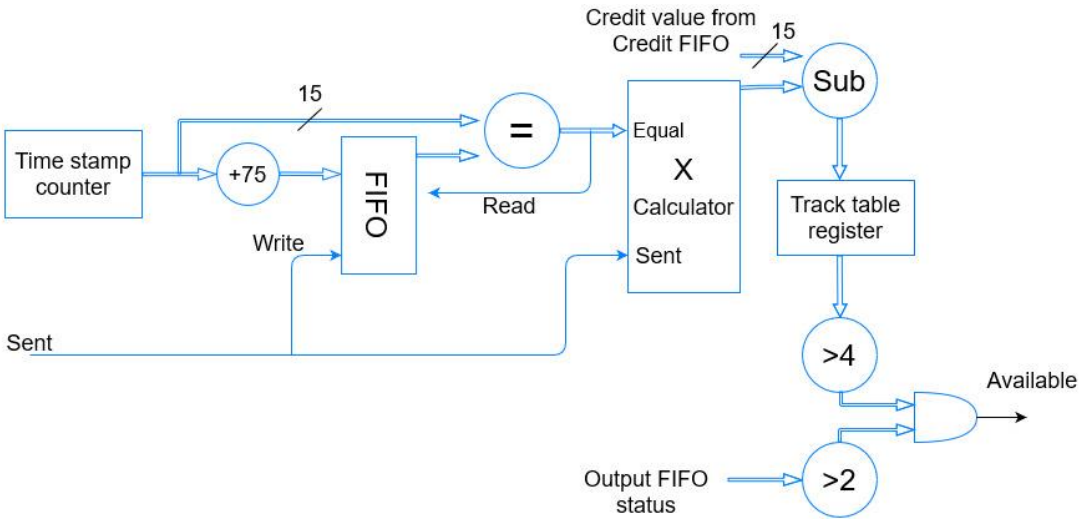


Figure 4:5 ATT hardware block diagram

As an example, if two packets are sent in clock cycles 100 and 110, then ($timestamp_{stored\_1}$ = 175) and ($timestamp_{stored\_2}$ = 185). Also (*Track table* = credit value -2) after the first packet and (Track table = credit value -4) after the second packet. The table updates its value in cycles 175 (*Track table = credit value -2)* and 185, (*Track table = credit value)*. The value 75 is derived by accounting for the worst-case scenario delay through the routers. This includes the FIFOs and control blocks along the data path. For port 0, this value is 5, as its components are close to the interface and both are located on the same FPGA.

```
1  Timestamp = Timestamp + 1;
   ----------------------------------------
2  If Sent='1'
3  SCFIFO_input = Timestamp + 75;
   SCFIFO_Wr_Req = '1';
   Else
4  SCFIFO_input = 0;
   SCFIFO_Wr_Req = '0';
   End if;
   ----------------------------------------
5  If SCFIFO_Empty = '1'
6  Equal ='0';
   SCFIFO_Rd_Req = '0';
   Elseif  Timestamp >= SCFIFO_Output
7  Equal ='1';
   SCFIFO_Rd_Req = '1';
   Else
8  Equal ='0';
   SCFIFO_Rd_Req = '0';
   End if;

9  Track table = credit value – X;
   ----------------------------------------
   X=0;
10 If Sent='1' and Equal='0'
   X= X + 2;
11 Elseif Sent='0' and Equal='1'
   X= X - 2;
12 Elseif Sent='1' and Equal='1' then
   X = X;
13 Else
   X = X;
```

Figure 4:6 ATT pseudo code

- **Routing Engine (RE)**

The RE module determines whether a word is a header flit, decodes the address portion of the header flit and issues a request signal accordingly to XY routing algorithm examined in 2.6. The RE carries out comparison between the address of the router and the address portion of the header flit.

Figure 4:7 gives the pseudo code for the routing engine utilized in RE. If both X and Y addresses are equal, the RE asserts the Request (0) signal to ask for port 0, which is the internal IP port (either FCMP, an astrocyte, a neuron). In the case when the two addresses are not equal, firstly X-axis addresses are checked. In the case when the X part of the router address is smaller than that of the header address, Request (3) -East port- will be asserted. The RE Asserts

the Request (4) signal -West port- if the address belonging to the X-axis of the router is larger than that of the header flit.

When the packet finally arrives at the same column as the destination node, meaning that X-axis addresses are the same, the RE assesses the Y-axis part. In the event the Y-axis of the router is larger than the header address, Request (1) – North – will be asserted and Request (2) – South - will be asserted otherwise.

The RE does not process its input from the InCh FIFO if there are less than four packets in InCh FIFO, indicated by *Rd-used* input from the InCh FIFO. This mechanism is in place to prevent a read from an empty DCFIFO since there are delays on *Rd-used (See Appendix)*.

```
The routing engine pseudo code

Request (0) <= '1' when (Destination _Y = Router_Y) and (Destination _X = Router_X) else '0';

Request (3) <= '1' when (Destination _X > Router_X) else '0';

Request (4) <= '1' when (Destination _X < Router_X) else '0';

Request (2) <= '1' when (Destination _Y > Router_Y) and (Destination _X = Router_X) else '0';

Request (1) <= '1' when (Destination _Y < Router_Y) and (Destination _X = Router_X) else '0';
```

Figure 4:7 The routing engine pseudo code

- **Arbiter**

The Arbiter's main duty is to control access to the output channel. There is an Arbiter per output port, and they work in parallel. The arbiter changes its priority in a Round-Robin fashion. The most recent port the Arbiter has served will have the lowest priority for the next request, which facilitates a fair arbitration [3].

The hardware implementation of the Arbiter is memory-based. The Arbiter utilizes a five-word SRAM memory to store the port numbers from highest to lowest in terms of priority. If, as an example, address 0 in the memory contains the decimal number 2, it means port number 2 has the highest priority. Once it is serviced, the memory shifts its content by one position in a Round-Robin manner in counter-clockwise direction. In this case, number 2 is stored in address 5 (the lowest priority) and the contents of the other address locations move to address+1 each, increasing their priority by one. The Arbiter carries out competition only among the input ports that are actively requesting access to the output port and disregards the ports that have not asserted their request.

As illustrated in Figure 4:4, each Arbiter has a 5-bit wide Requests signal as its input since every RE sends each Arbiter a request signal. Additionally, there is a single bit signal from ATT that indicates if the arbiter is able to process requests. The Arbiter ignores all the requests if this signal is at logic '1'.

A 5-bit wide out bus from the Arbiter feeds into the Xbar to decide which input can have access to the OutCh FIFO for the duration of two clock cycles.

- **Crossbar (Xbar)**

The Xbar switch contains 5 smaller switches, each controlling the access to a particular output channel as shown in Figure 4:8.

Switches 0-4 are basically one-hot multiplexers with the Grants 0-4 signals acting as the select signals. The Grants signals are 5-bit each (one for each RE) and are supplied by the Arbiter module.

Each output switch works independently, implying that all the output channels can be accessed simultaneously without limiting the overall throughput of the router.

- **Write Controller (WrCNTRL)**

WrCNTRL is not as complex as the other controllers in the NoC router. When the Arbiter controlling a specific output port issues logic '1' on one of its Grant lines, a delay of one cycle will be introduced and the Grant input to WrCNTRL will be asserted. The WrCNTRL will extend this Grant signal for two clock cycles to write a data packet, composed of a header and a data flit, to OutCh FIFO.

- **Input Channel Track Table (ICTT)**

As in Figure 4:4, the ICTT module receives *Rd_Used* from InCh FIFO to determine how many more data packets the InCh FIFO can accommodate as per Eq.30.

$$\text{Credit value} = FIFO_{max\_used} - (Rd\_Used + 2) \quad (30)$$

For instance, if there are 20 words (10 packets) in the InCh FIFO, using equation (30) this sets Rd_Used to 20. $FIFO_{max\_used}$ is the maximum available space, which is 64. This implies;

Credit value = 64 – (20 +2) = 64-22 =42.

Ideally, using *Wr_used* would give a better indication of the status of the FIFO from the perspective of the upstream router. However, *Wr_used* operates in the GXB_Rx clock domain while ICTT uses the router clock domain. Of course, one could use a DCFIFO for crossing the two clock domains but that would introduce further delay. Due to the fact *Rd_Used* is the delayed version of *Wr_used* [128] and it is in the same clock domain as the router. A conservative approach has been taken which assumes that there is always two words more in the write-side as expressed in Eq.30 Although, depending on the instant *Rd_Used* is read this might not be the case.

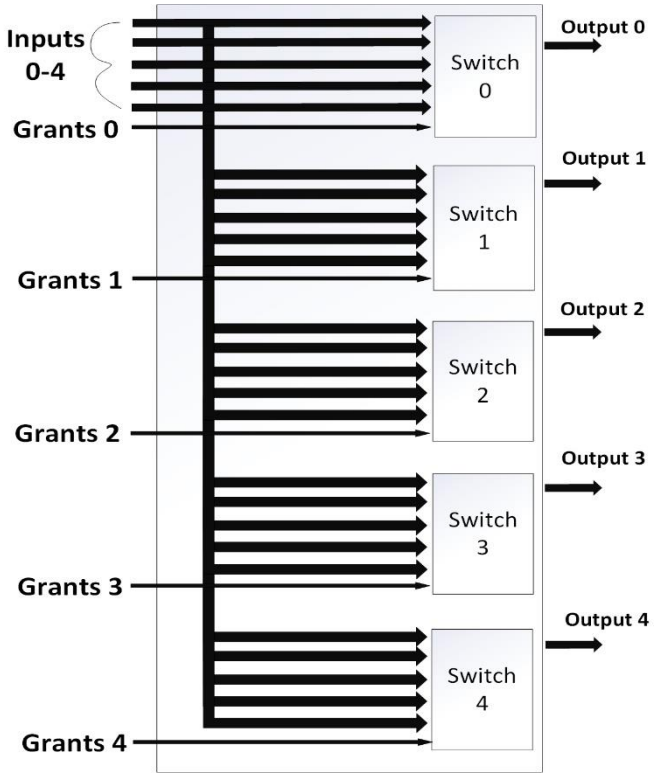


Figure 4:8 Router Xbar architecture

Credit value will be packetized into a credit flit/packet and be passed down to ICTT FIFO. The credit packet will be eventually used by the upstream router ATT to determine whether the Arbiter can issue grants on this port should requests made to it.

This block does not exist in port 0 due to the fact that, since SI is on the same FPGA, InCh FIFO *Wr_used* will be directly forwarded to SI.

- **Input Channel Track Table FIFO (ICTT FIFO)**

This block is identical to ATT FIFO. It is a 4-word-deep DCFIFO that is used for clock domain crossing. The write side works in the router clock domain while the transmitter channel clock domain (GXB_Tx) clocks its read side ICTT controls this block for write operations while Output Stream Controller (OSC) manages its read side, as will be discussed later.

Port 0, the internal computing node port, does not makes use of ICTT FIFO because a DCFIFO is not required when the blocks exist in the same clock domain.

- **Output channel FIFO (OutCh FIFO)**

The OutCh FIFO is used for managing clock domain crossing between the router clock domain and GXB_Tx. WrCNTRL controls the write side of the DCFIFO and OSC manages the read side. OutCh FIFO is 8 words deep to accommodate up to 4 packets as experimentation proved that a depth of 16 words wouldn't help throughput.

- **Output stream controller (OSC)**

The OSC multiplexes the output channel to accommodate both the data packet and credit packets. OSC decided when to read packets from OutCh FIFO and ICTT FIFO. It reads data from OutCh FIFO only when there is more than three words present to avoid reading from an empty DCFIFO.

- **Fault controller (Fault CNTRL)**

As discussed before, the NoC data packet is 36-bit wide. The MSB bit indicates if the data stored in the FIFO is a header flit.

Figure 4:4 indicates the FAULT CNTRL modules located in front of InCh FIFO and OutCh FIFO blocks. Fault CNTRL monitors the output of the DCFIFO in search for a particular fault pattern that occurred, at times, as the result of resetting the AstroByte platform mid-operation. This is due to the unexpected behaviour of the DCFIFOs implemented in the NoC router architecture when reset while the AstroByte performs simulations. The user usually resets the platform - e.g. before reprogramming the platform by the FCMP to implement a different NoC mapping.

This might cause a data flit, instead of a header flit, to be located at the top of the FIFO stack, recognized by H bit. If this occurs, the packet won't be processed as it is not a header flit, causing the closure of the path and hampering the *AstroByte* operation.

Fault CNTRL observes the top of the FIFO stack. If it detected a data flit that is not following a header flit, the data flit will be discarded The Fault CNTRL block does not forward data to any other block.

4.6 Router Design and Operation

- **Design Choices**

Several factors affected the design choices made regarding the router architecture design.

Firstly, the fact that data has to be communicated between different FPGA boards dictated using transceivers -the GXB IPs. Using transceivers mean having to use DCFIFOs for clock domain crossing, which in turn means that writing and reading data from the DCFIFOs have to be considered when designing many of the blocks in the router. One could implement a multi-FPGA system without using transceivers however this would require feeding the clock to all the boards from a single source. This facility wasn't available at the time of conducting this research, making using of GXB IPs necessary.

Secondly, reducing the clock cycles per packet processed (hence increasing throughput) was another contributor to the router microarchitecture design. One can notice the absence of Input/output channel controllers and a main control block in the router design. This is due to the optimization efforts made to

increase throughput. Instead the functionality of these blocks are distributed among other smaller blocks that work in parallel.

Optimizing the router microarchitecture for low-power consumption was not explored in this study.

- **Router Operation**

After explaining the router microarchitecture and the NoC data format, this section provides an overview regarding the way packets progress through the router.

Data stream arrives serially at GXB_Rx. The GXB parallelizes the data and forwards it to the FPGA fabric using GXB_Rx clock domain. The next block is ISC, which separates the data and credit packets and forwards them to InCh FIFO and ATT FIFO, respectively. ISC also assigns each data packet flit with another bit to designate the word as data or header flit.

ISC de-packetizes the credit packet to credit packets before storing it in ATT FIFO. Consequently, the ATT uses the stored packet to determine whether the Arbiter can grant access to a particular output channel.

In terms of InCh FIFO, Rd_Used value should be at least 3 (starting from 0, meaning four words exist in the FIFO) in order for RE to process the header flit and issues a request. This approach aims at preventing reading from an empty FIFO which would stall the operation of the NoC.

When enough data appears on the road side, the RE asserts an appropriate request signal using the routing engine. The Arbiter, with help of ATT, next decide when to grant the request. RdCNTRL takes the grant signal from the arbiter and asserts read to InCh FIFO for two clock cycles. Meanwhile the grant bus from Arbiter act as the select signal to the Xbar block switches.

The grant signal out of a particular Arbiter is sent to WrCNTRL, which in turn, allows the data packet that has progressed through the Xbar from the Inch FIFO to be written to the OutCh FIFO.

The OSC block then forwards the OutCh FIFO content to the transceiver input after removing the 33rd bit of the data packet, assigned by ISC at the start of the data path. The data words have to be changed back to 32 bits because the transceivers accept 32 bits wide words, before serializing the data stream and sending it to the downstream router.

4.7 FSA & FCMP Interfaces

The previous section investigated the router microarchitecture and its operation in detail. The router will be integrated into a mesh NoC structure as shown in Figure 4:9.

The purpose of the router is to facilitate communication between the computing elements, (FSA or FCMP), through the network. In order to establish coherent

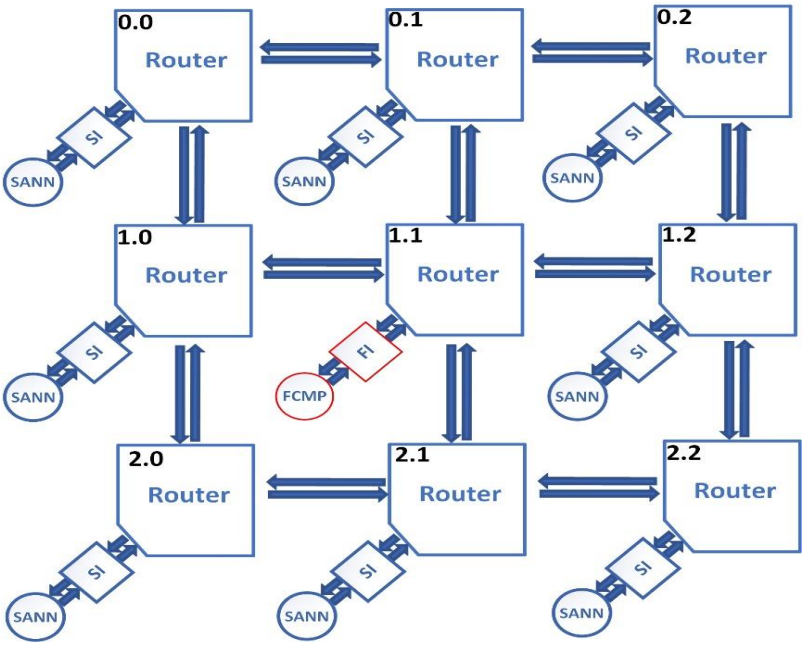


Figure 4:9 A 3x3 mesh NoC architecture

communication, both FSA and FCMP are interfaced with the NoC by means of the FSA Interface (SI) and FCMP Interface (FI).

4.7.1 FCMP

In Chapter 3, the FCMP architecture was discussed in detail. The architecture shown in Figure 3:10 had to be re-purposed for AstroByte NoC structure, depicted in Figure 4:10. Because the width of the data stream arriving at the FCMP Interface is one 32-bit bus, only one data memory is required in the modified FCMP version. The architecture in Figure 4:10 does not have controllers as the one in Figure 3:10 because SRAMs can be interfaced directly to the FPGA fabric as they are located on-chip. Other differences can be seen in re-purposing usage of parameter and interface blocks. Generally, however, the basic working principles are the same.

An Intel Nios II embedded processor is used for storing data in SRAM and for sending simulation data to a PC. Packetizing and storing configuration data will be discussed in the following section.

- **Configuration Data Packetization**

Nios II processor is programmed using C language for packetizing configuration data and storing it in an SRAM memory.

Figure 4:11 illustrates how C language pointers is used for storing configuration data before transferring to the destination (reconfigured node). Firstly, equivalent hexadecimal 32-bit data for the header flit is constructed and then stored, followed by the data (configuration) flit in the next memory location.

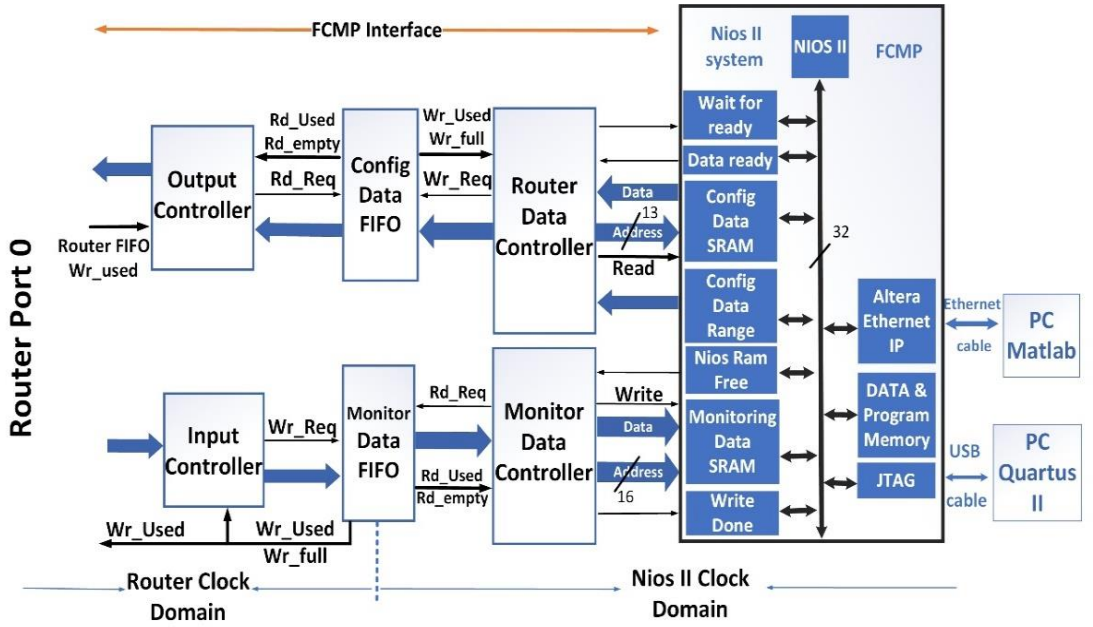


Figure 4:10: FCMP Interface architecture

Overall, there are 14 programmable attributes for the FSA elements that can be modified. In Figure 4:11, node (2,1) is programmed to send spike data to node (2,2). *SenderAdd* is the address of the reconfigured node (2,1) and *SpikeAdd* is the destination address for spike packets sent by *SenderAdd.*

In the next sections it will be seen that all the attributes for all the NoC nodes will be packetized similarly and stored, before been sent to the NoC fabric by the FCMP Interface (FI).

4.7.2 FCMP Interface (FI)

The FI architecture is shown in Figure 4:10 along with FCMP platform. It is function is transfer data between FCMP and router port 0, and vice-versa. Like SI, proper clock domain crossing techniques have been implemented to decouple the router and Nios II system domains. Unlike SI, however, no packetization or de-packetization takes place in FI. This owes to the fact that complete data packets -both header and data flits, will be sent for monitoring. Likewise, FCMP sends whole packets to port 0.

The next sections examine the operation of FCMP sub-modules.

- **The Output Controller**

It Controls data flow from the FI to router port 0. It takes into consideration the number of words in the Configuration Data FIFO and the router port 0 InCh FIFO. If there is less than 4 words in the Configuration data FIFO or more than 60 words in the router port 0 InCh FIFO, data forwarding will be stalled. In all other cases, data will be continuously sent to the InCh FIFO 0.

➢ **The Input Controller**

This module is a relatively simple one since it forwards data form router port 0 to Monitor Data FIFO.

4.7.3 Router Data Controller (RDC)

The Router Data Controller has necessary data, address and control signals to facilitate handshaking with the FCMP. At first, the RDC block asserts the *Wait for ready* signal, informing the Nios II based system that it has come out of reset. This arrangement is necessitated by the fact that FCMP and FI do not come out of reset at the same time, a constraint which is imposed by Multi-FPGA implementation.

Data ready will be asserted by FCMP after storing reconfigurable packets in its Configuration Data SRAM. RDC then starts sending the already packetized data to Configuration Data FIFO until it sends a number of packets equal to FCMP Configuration Data Register. The algorithm for the operation is shown in Figure 4:12.

4.7.4 FSA Interface (SI)

The SI module is composed of a number of blocks. It is bi-directionally connected to the router port 0 on one end and FSA on the other. The first responsibility of SI is decoupling the frequencies between the router clock domain and FSA clock domain by means of DCFIFOs as shown in Figure 4:13. Additionally, the SI module packetizes FSA data (spikes, astrocyte, or counter output) and de-packetizes incoming data packets. It then forwards the data flit either to the configuration register files or FSA.

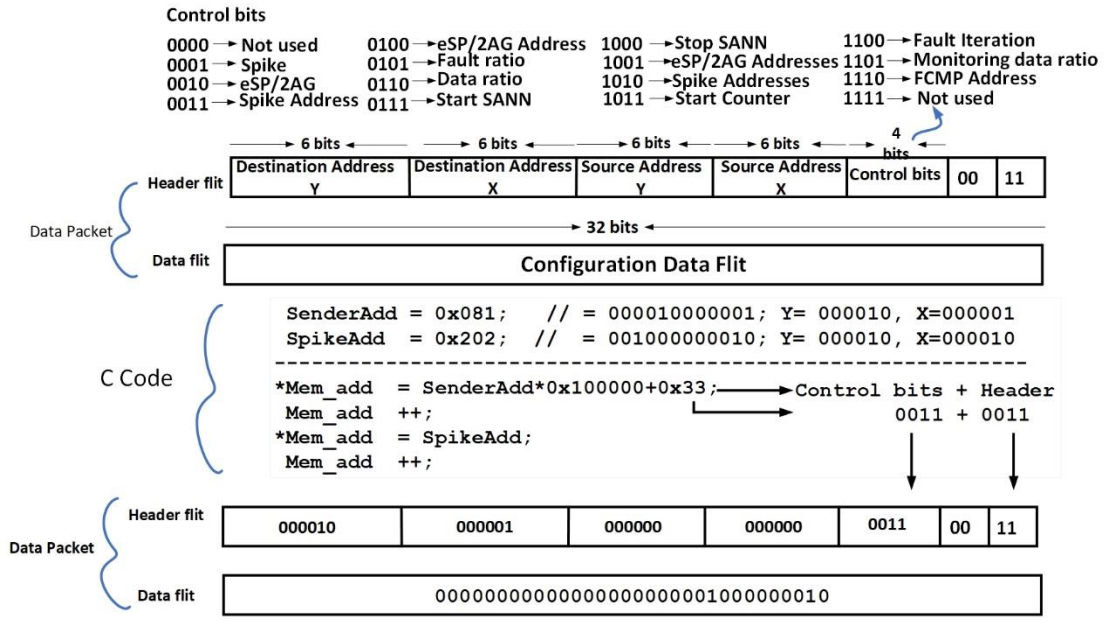


Figure 4:11 Nios ii Configuration data packetization

- **FSA Programmability**

AstroByte provides users with the ability to reconfigure various aspects of FSA components. A mixture of data format and architectural design approaches facilitate this programmability.

Another important function of SI is decoding and storing the configuration attributes specified by the user. Whereas Nios II processor creates and stores the configuration packets by using C language as explored in section 4.6.2, Input controller block decodes the configuration attributes from the header flit and issues store commands to the configuration registers and FIFOs. Next, the stored attributes will be used by Output controller and FSA respectively.

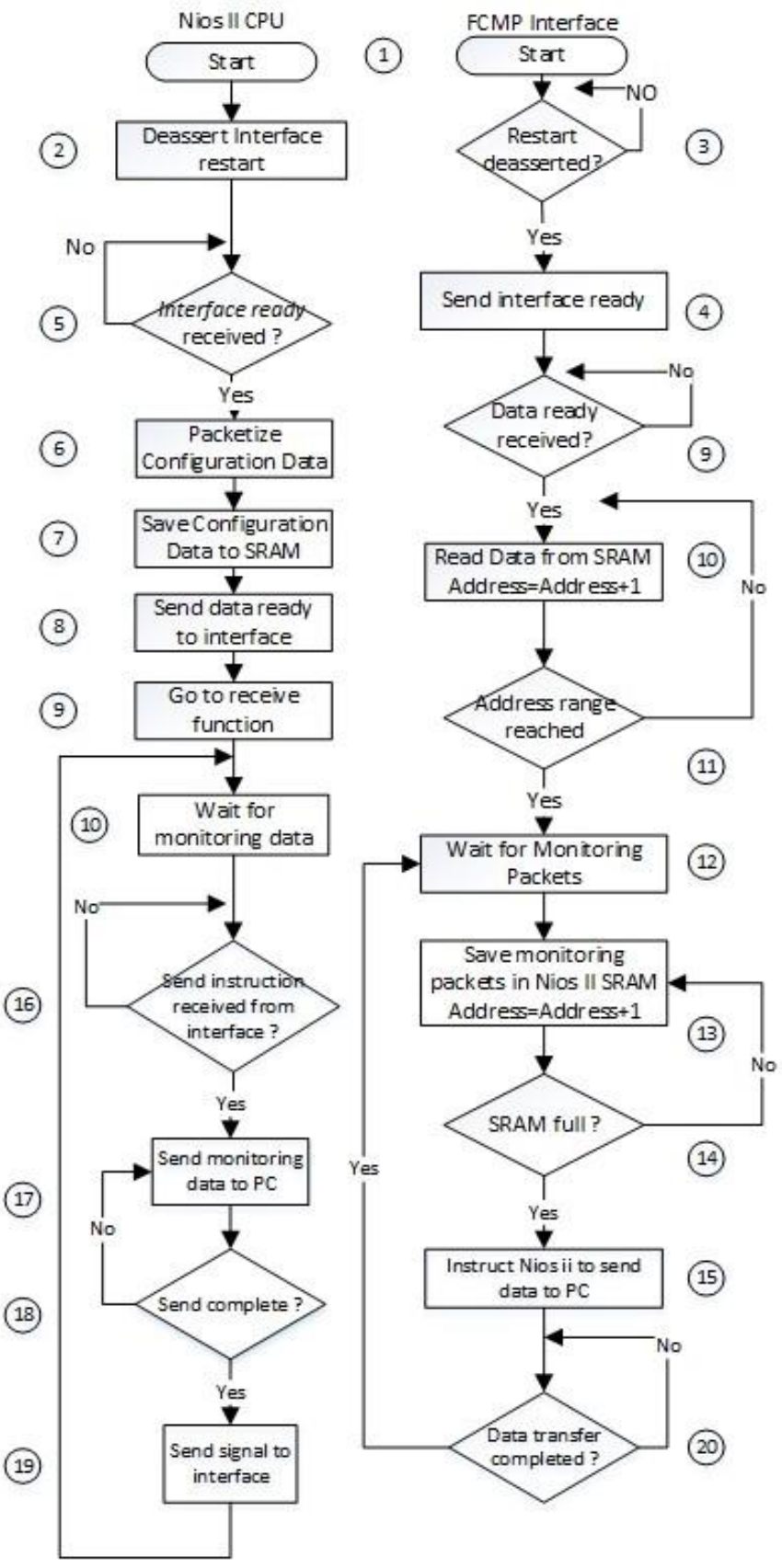


Figure 4:12 Algorithm outlining handshaking and data transfer between FCMP and FI

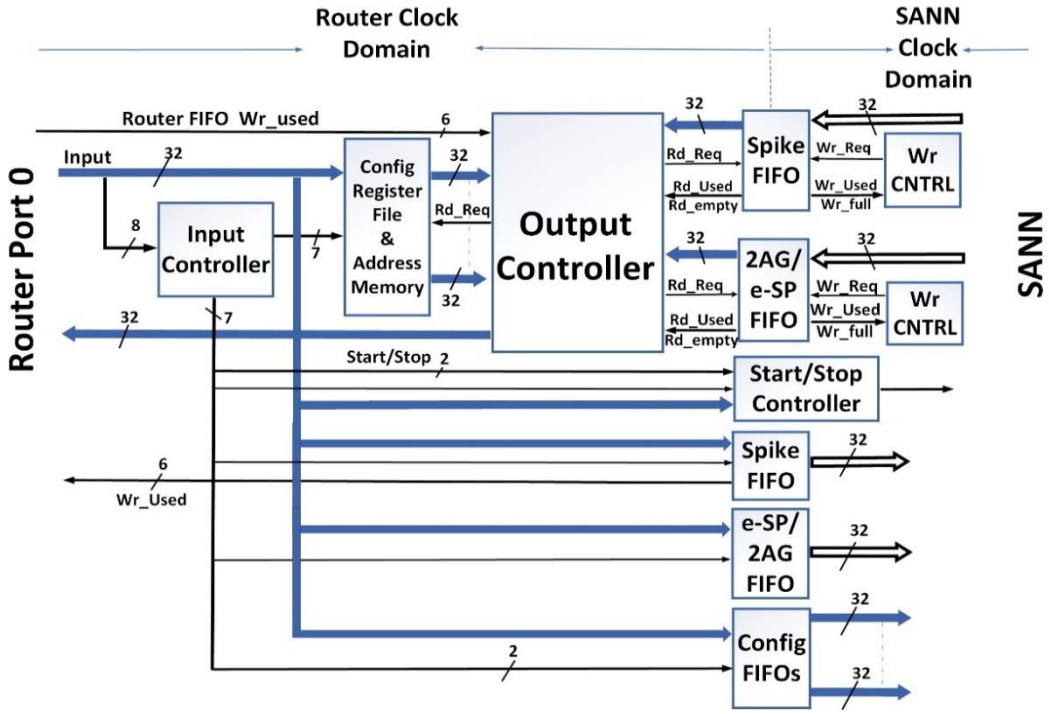


Figure 4:13 SANN Interface (SI) microarchitecture

The following table illustrates the configurable attributes, their control bits representation and a short description of each attribute.

Table 4:1 Programmable attributes of FSA elements

Control code	Data flit	Description
0000	Dummy/Not used	Not used/Used for filler packets
0001	Spike data	Spike information from source neurons
0010	eSP/2AG data	eSP/2AG from astrocyte/neurons
0011	Spike address	Address of destination neuron
0100	eSP/2AG address	Address of destination astrocyte/neuron
0101	Fault ratio	Ratio of faults to be introduced
0110	Data ratio	Ratio of simulation to be performed
0111	Start	Control start of astrocyte/neuron simulation
1000	Stop	Control stop of astrocyte/neuron simulation
1001	eSP/2AG addresses	Number of eSP/2AG destination nodes
1010	Spike addresses	Number of spike destination nodes
1011	Start counter	Number of cycles after Start received
1100	Fault iteration	Number of cycles before injecting faults
1101	Monitoring ratio	Ratio of simulation data to be sent to FCMP
1110	FCMP address	Address of FCMP
1111	Dummy /Not used	Not used/Used for filler packets

Next, the operation of various blocks of SI will examined.

➢ Input Controller (ICR)

The ICR block acts as a decoder with registered outputs. The input to the decoder is the first byte of the 32-bit wide data bus (Input in the figure) from port 0. Input [1:0] bits are the header flit designator, followed by two reserved bits (Input [3:2] = “00”) as per the NoC data format shown in Figure 4:14 .

The control bits, Input [7:4], will be decoded to determine the purpose of the packet. The packet can hold SANN data - containing either spikes or 2AG/eSP - configuration data or Start/Stop control commands.

All the ICR outputs are registered, meaning the control signals will be asserted in the next clock cycle when data flit arrives at register inputs, in line with the NoC data format.

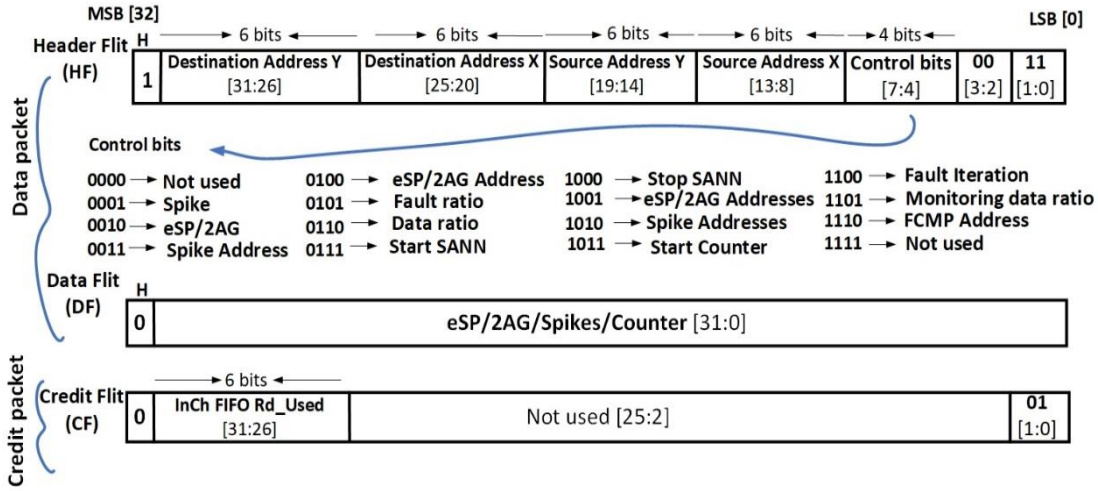


Figure 4:14 AstroByte data format

➢ Configuration Register File

Figure 4:13 indicates the location of the Configuration Register File, composed of 7 registers, located between ICR and the Output controller block. The configuration registers hold the value for the configuration parameters which sent by FCMP and recognized and decoded by ICR. From Table 4:1, the parameters saved in the Configuration Register File include; Spike address, eSP/2AG address, Data ratio, eSP/2AG addresses, Spike addresses, Monitoring ratio, and FCMP address.

➢ Address memory

The Address memory block can store up to 8 destination addresses for both spike data and eSP/2AG data. Each time the write signal on the memory goes high, a custom memory controller (not shown) will save the address in the next memory location. The parameters Spike addresses and eSP/2AG addresses are used in conjunction with the Address memory. The OC implements a down counter, starting from the value stored in Spike addresses register (for sending spikes). Once the down counter reaches 0, the OC starts again for sending the

next spike word. Similar to the write side, each time the OC asks for an address, the Address memory provides a different address. The operation of Address memory is similar to FIFO, with the difference that their entries won't be discarded when read.

- **Configuration FIFOs**

The Configuration FIFOs contain two DCFIFOs for storing Fault ratio and Fault iteration parameters. The values stored in these FIFOs are used by SI to insert various fault ratios at specific cycle according to Fault iteration value.

- **Start/Stop controller**

This inputs to this module are Start/Stop control signals out of the ICR, the main input bus from router port 0 (Input) and the Start counter control signal for saving the attribute. When a start signal is received, the module will wait for a number of clock cycles specified by the Start counter value before sending Start command to FSA. The Stop command will be passed in the next clock cycle.

- **Spike & 2AG/eSP FIFOs**

The SANN component either contains an astrocyte or a neuron element. In the case of a neuron element, the inputs will be spikes and eSP from an astrocyte. In the case of an astrocyte, 2AG from the neurons will be its inputs (Figure 2:3).

The two sets of Spike and 2SP/2AG DCFIFOs in Figure 4:13 are implemented to clock crossing from FSA to router and vice-versa.

- **Write Controller (Wr CNTRL)**

Two WrCNTRL blocks control the data flow from SANN components to Spike and 2AG/eSP DCFIFOs. In case there isn't enough space in the FIFOs the operation of the SANN elements will be suspended by means of enable signals from WrCNTRL to SANN.

- **Output Controller (OC)**

Packetizing FSA data from the DCFIFOs will be carried out by OC block. The data packet are then forwarded to ISC of the router port 0. OC is regularly updated on the status of router InCh FIFO 0 by the Router FIFO Wr_used, shown in Figure 4:13.

OC takes into account the user defined attributes stored in Configuration Register Files while packetizing and forwarding FSA data.

The spike address attribute is assigned to the address part of the header flit when Spike data is packetized. The attribute eSP/2AG address is included in the header flit when sending eSP/2AG data. Data ratio decides how much of the available channel bandwidth is used by the core. If 50 was stored in Data ratio register, OC exploits half of the available bandwidth.

FCMP address will be used in the destination field of header flits so the packet is routed to FCMP. Monitoring ratio attribute will be used to decide how much of simulation data to be forwarded to FCMP. If this value was 100, the implication is all the simulation data forwarded to destination nodes is forwarded to FCMP as well. To this effect, the OC module does not get rid of data in the DCFIFOs after sending them to the destination SANN elements. It reuses them and packetize them for a second time with FCMP address attached. If monitoring ratio value is 50, every other DCFIFO entry will be sent twice, one for destination FSA and one for FCMP.

Spike and eSP/2AG addresses point to the number of destination addresses that spike and eSP/2AG are sent to.

4.8 Chapter conclusion

This chapter was dedicated to design and implementation of AstroByte platform. Operation, architecture and implementation of Intel DCFIFOs and transceivers were examined as the IPs are used for realizing multi-FPGA communication infrastructure. The AstroByte data format and the NoC router architecture were studied. Additionally, implementation and working of the individual components facilitating NoC router architecture were presented. Lastly, architectures and operation of SI, FCMP, and FI were investigated in detail. The next chapter is allocated to analysis and experimentation on AstroByte platform. Results regarding performance parameters like throughput, latency, acceleration and accuracy will be presented.

Chapter 5: Experimentation and Results

This chapter will present experimentation results from a fully connected, scalable and programmable AstroByte platform. The experiments include throughput, latency and acceleration performances. In addition, under-sampling and accuracy performances will also be assessed.

The structure of the chapter is as follows. Firstly, the overall setup of experimentation will be discussed in Section 5.1 and section 5.2. Section 5.3 presents results on evaluating the throughput of a 3x3-FPGA system under different traffic rates. Examining the latency experienced by packets in the network will follow in section 5.4. Acceleration gained from AstroByte over an equivalent Matlab SANN implementation is presented in section 5.5. Section 5.6 provides information as regards mapping the FSA to the AstroByte NoC nodes. Section 5.7 investigates the AstroByte platform in comparison to Matlab (software) in terms of accuracy and execution time. Additionally, comparison with two other multi-FPGA platforms will be presented in section 5.7 before concluding the chapter in section 5.8.

AstroByte is the third contribution of this study. Some of the results and overall architecture design are published in the paper *"AstroByte: Multi-FPGA Architecture for Accelerated Simulations of Spiking Astrocyte Neural Networks"* at the Design, Automation & Test in Europe (DATE) conference in 2020.

5.1 Experiment setup

The PC used for all the experiments in this chapter has the following specifications; 64-bit Windows 10 Enterprise, Intel Core i7-2600 3.4 GHz processor with 16GB RAM.

Intel Quartus Prime 18.0 SE was used for VHDL coding, analysis, synthesis and FPGA programming. SignalTap II and the in-house developed FCMP IP core/software were used for design verification through analysing data acquired from the FPGAs. MATLAB R2015b was used for capturing simulation data and analysis. An Intel build for Eclipse Mars 2 was used with Nios II embedded processor.

The Terasic DE4 FPGA development and educational board was used which utilizes an Intel Stratix IV GX EP4SGX530 FPGA. For the AstroByte implementation, Ethernet, SATA, and General Input Output Pin (GPIO) interfaces were also used.

5.2 Multi-FPGA Setup

A 3x3-FPGA AstroByte platform was setup for experimentation as shown in Figure 5:1. The FPGAs are interconnected in a mesh structure using SATA

cables as the physical medium. Intel GXB transceivers are responsible for establishing integral cross-FPGA links (see the Appendix).

General Input Output Pin (GPIO) wires serve the function of restarting the entire system at the start, as this is the requirement for synchronization between the GXB transceivers. The reset GPIO originates from the FCMP board and makes its way to all the nodes forming the AstroByte platform.

The sliding switches on the DE4 boards are for choosing between different available configurations. The user can select between FSA and test counters as the computing elements. Additionally, the user can choose between 10 MHz or 150 MHz operation frequency should counters be used.

The FPGA IPs support programmable address feature, using pushbuttons located on DE4 boards. The user can program all the FPGAs with the same IP (except for the FCMP node) and change the addresses using the pushbuttons. This saves the users a lot of time since a separate IP is not required for each address. The two 7 segments show the address entered by the user at a particular instant.

Figure 5:2 shows the platform working sequence. Firstly, all the FPGA boards will be programmed with their relative bitstream files. Then the user pulls the global reset signal low by a sliding switch on the FCMP board. Led signals on the DE4 boards indicate whether all links are synchronized. If one or more links had failed to synchronize, the user can simply pull the reset low again.

Now the AstroByte platform is ready to be used. By means of Nios II programming, the FCMP node starts sending configuration data packets to the other nodes to facilitate a user defined multi-FPGA platform.

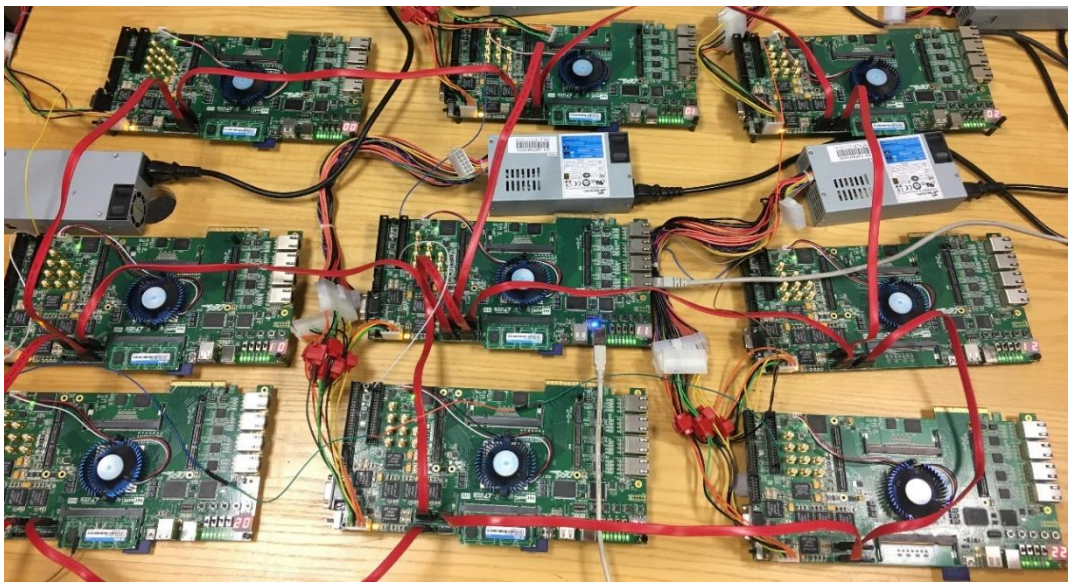


Figure 5:1: A 3x3-FPGA AstroByte platform

Next, the computing elements, FSA or test counters, start sending data through the NoC infrastructure to its destinations. This may include FCMP as specified by the configuration packets.

Figure 5:3 presents an example of the work sequence above for a prototype application. Firstly, configuration data packets make their way from FCMP at

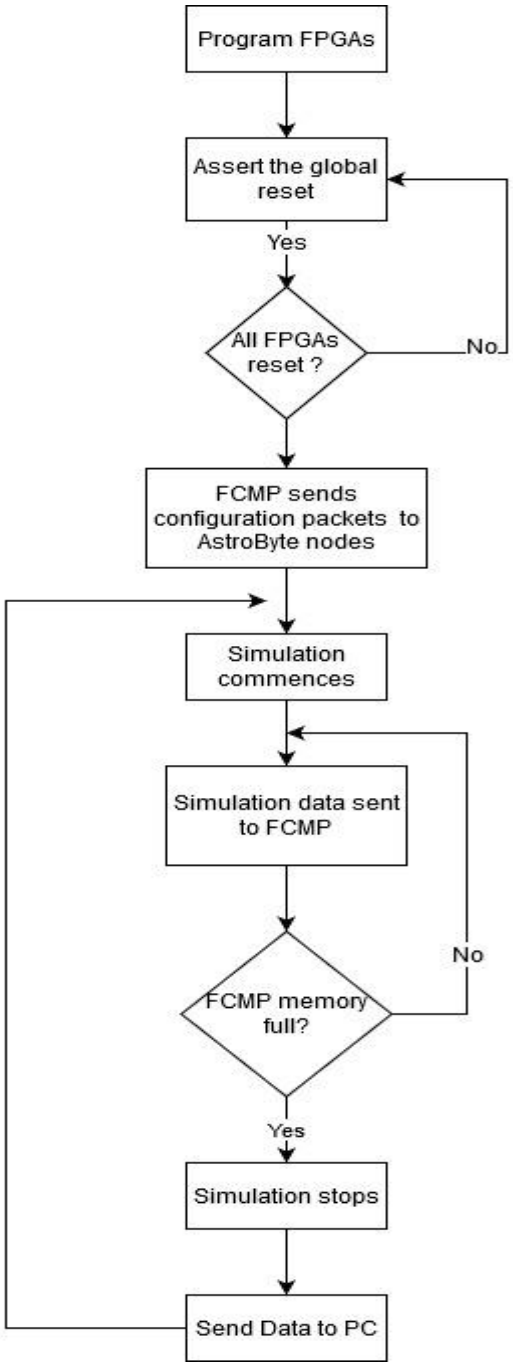


Figure 5:2: AstroByte operation protocol

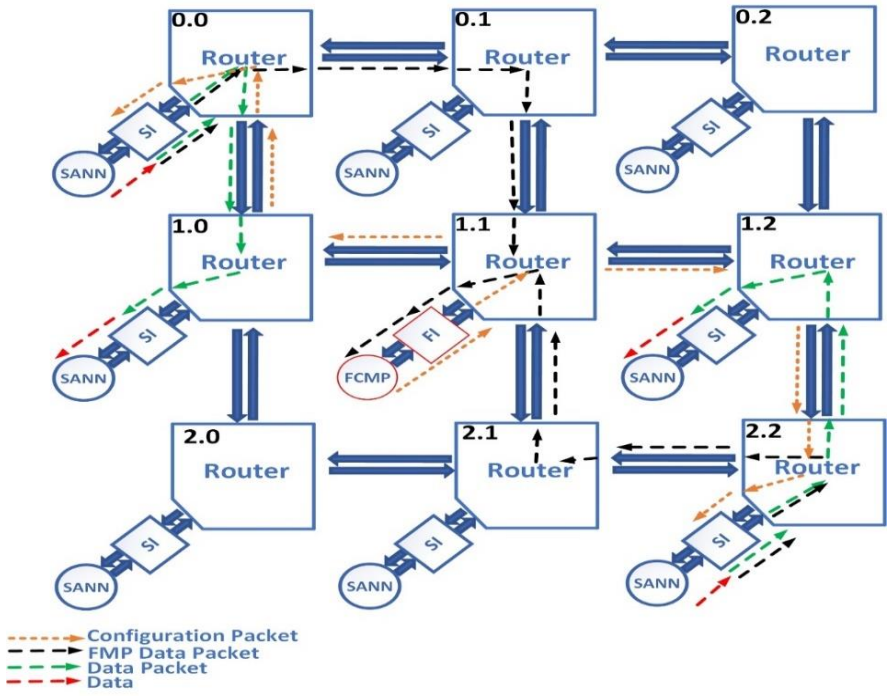


Figure 5:3 AstroByte setup for configuration, communication and acquisition phases

node (1,1) to nodes (0,0) and (2,2), setting destinations addresses to (1,0) and (1,2) respectively- all the nodes are chosen arbitrary. Both nodes also programmed to send monitoring data back to FCMP. When the configuration phase ends the computing elements start sending data packets to destination nodes and FCMP as per their configuration.

5.3 Throughput calculations

Throughput calculations are carried out by inserting special hardware blocks at the input and output ports of the router. The blocks include two counters. One counts until 150,000,000 cycles (one second), starting from the first data packet passed through the port, and the other counts the number of data packets received in that duration. For example, if 15,625,000 packets pass through calculator block, which means 31,250,000 flits (each packet is two flit) has passed through the router. To get throughput this has to be multiplied by 32, the width of GXB parallel inputs/outputs. The results is 1000,000,000, which is 1Gbps.

$$\text{Throughput (Gbps)} = \frac{\text{No of Packets} * 2 * 32}{10^9} \quad \text{.....(31)}$$

Figure 5:4 shows the AstroByte setup for this experiment. The FCMP platform is located at the router (2,2) and is used for configuring the network only, monitoring data is not fed back to it in this experiment. Using AstroByte programmability features by means of the Nios II programming, configuration packets carrying the destination node address and the traffic rates are sent to each node at start of simulation. Counters (denoted by C) were used as

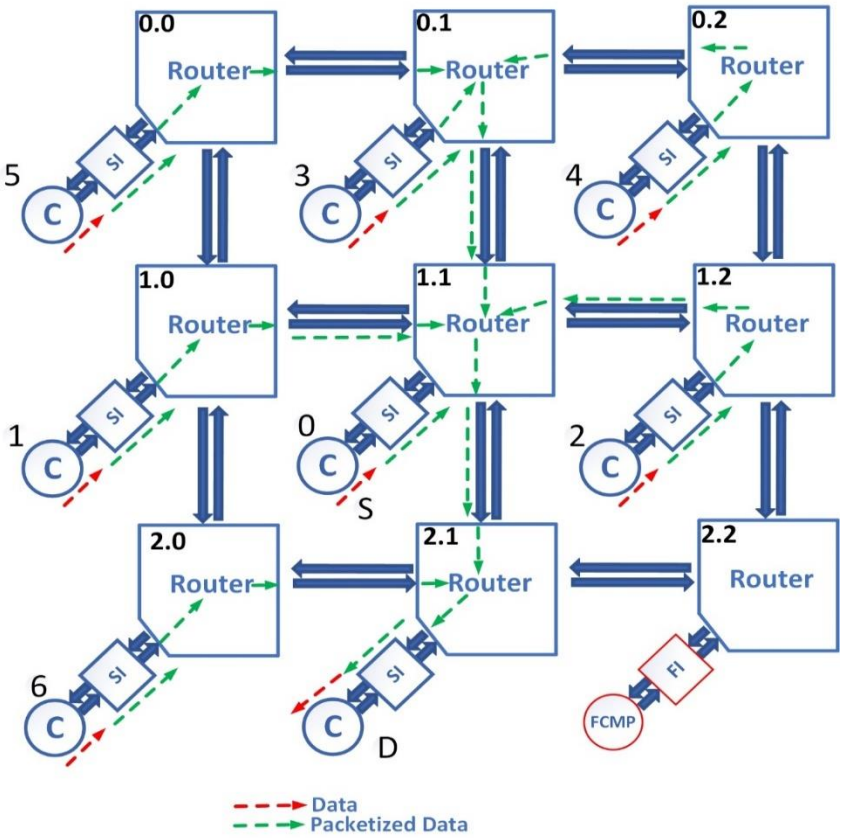


Figure 5:4 AstroByte setup for throughput

computing cores instead of FSA elements as using counters make debugging and observing the sequence of packets easier.

Packets were routed from node (1,1) – source, denoted by S- to node (2,1)-destination, denoted by D. The numbers on the upper left side of counters in the figure show the sequence at which the counters started inserting packets into the network. As an example, node (0,1) is the third node, meaning results for 0,1, and 2 were collected before the node starts sending traffic. The throughput is measured at input port 0 of the source router, at (node 1.1).

The following figures show throughput of the AstroByte platform in various scenarios. In the figures, the *x*-axis represents the rate of traffic (the percentage of the maximum theoretical available bandwidth) inserted by various nodes and the *y*-axis is throughput in Gbps. One could calculate the total throughput by accounting for all the packets that pass through a certain node, i.e. both data and credit packets. Alternatively, only the credit packets can be calculated for capturing the throughput figures of the useful data that will be eventually used in SANN simulations. In this study, firstly, the figures for data packets throughput were calculated then the total throughput figures were obtained by adding 1.44Gbps.

The results display throughput for cases when the computing cores operate at either 10MHz or 150MHz. The SANN in this study operates at a frequency of

10MHz however the infrastructure supports computing core frequencies of up to 150MHz-the operating frequency of the router.

Figure 5:5 show results for a case when only node (1,1) injects traffic into the network while all the other nodes are silent in various cases. The maximum total throughputs are 3.36Gbps and 2.72Gbps for 150Mhz and 10 MHz respectively.

For Figure 5:6 and Figure 5:7 the source node (1,1) is always injecting data packets at 100% rate while other nodes insert data packets at an incremental rate of 10%. As an example, the “3 ports” trajectory in Figure 5:6 is obtained when node (1,1) traffic rate is fixed at 100%, while nodes (1, 0), (1, 2) and (0,1) insert traffic at incremental steps of 10%. The trajectory starts at the maximum of 1.28Gbps which occurs when only node (1,1) sends traffic at 100%. The projectile then starts a linear decline when the other nodes start inserting traffic at 10% rate simultaneously. Finally, at 30%, the throughput stabilizes at 0.32 Gbps.

Figure 5:6a shows the throughput results for FSA operating frequency of 10 MHz. Here the throughput figures only account for data packets, meaning credit packets bit transfers are not accounted for.

Figure 5:6b displays the router throughput under the assumption that the computing core could be run at 150MHz, the frequency of the router. The results fluctuate between 1.92Gbps and 0.3031Gbps, depending on the number and traffic rate injected and the number of the computing nodes. Throughput results for 150MHz shows that the 10MHz does not take full advantage of the bandwidth available. As discussed in chapter 3, the rationale for using the 10MHz frequency is due to the performance of the astrocyte process in terms of operation frequency. The throughput figures show that optimizing the astrocyte for higher frequencies will result in better throughput. The astrocyte

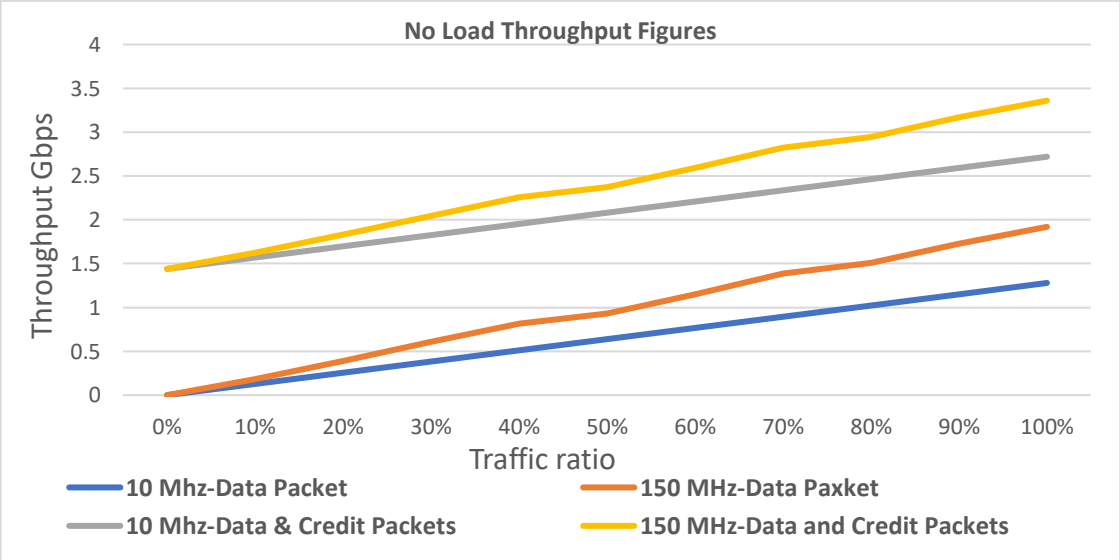
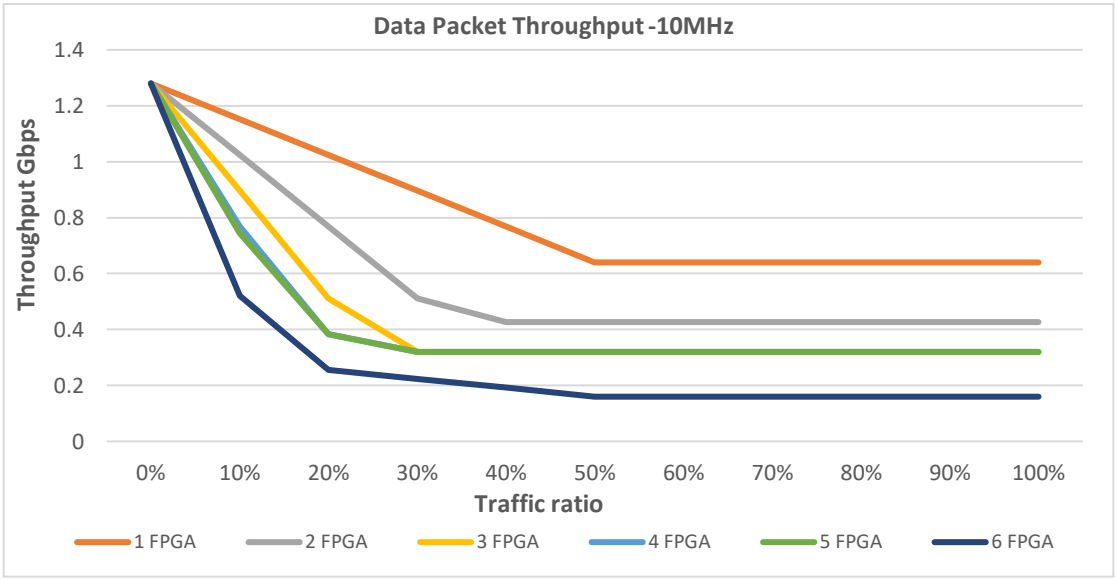
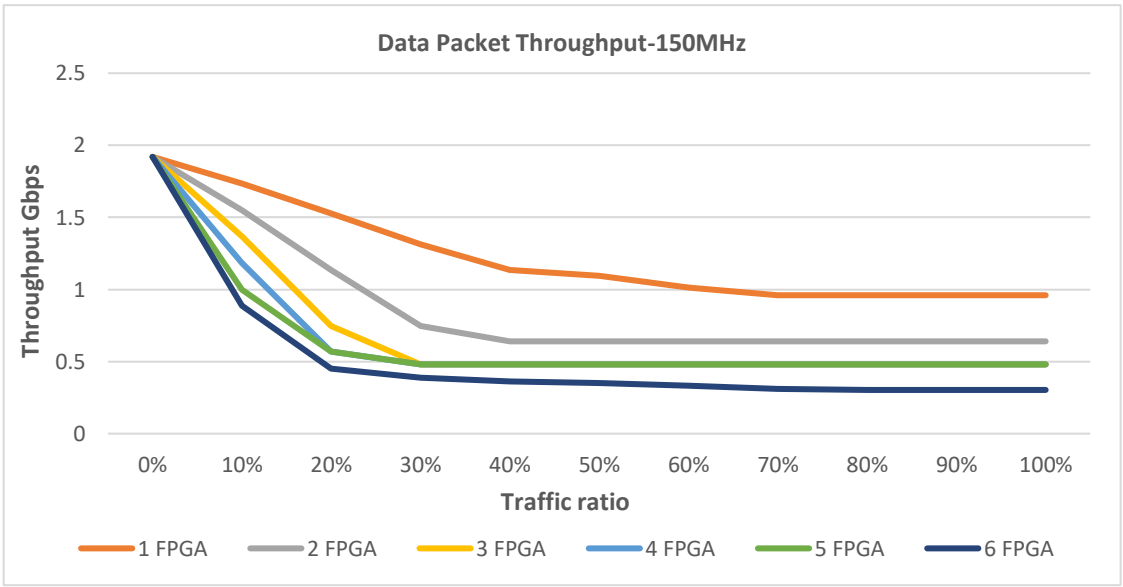


Figure 5:5: Various Throughput results under no load condition



(b) Data Packet throughput – 10MHz

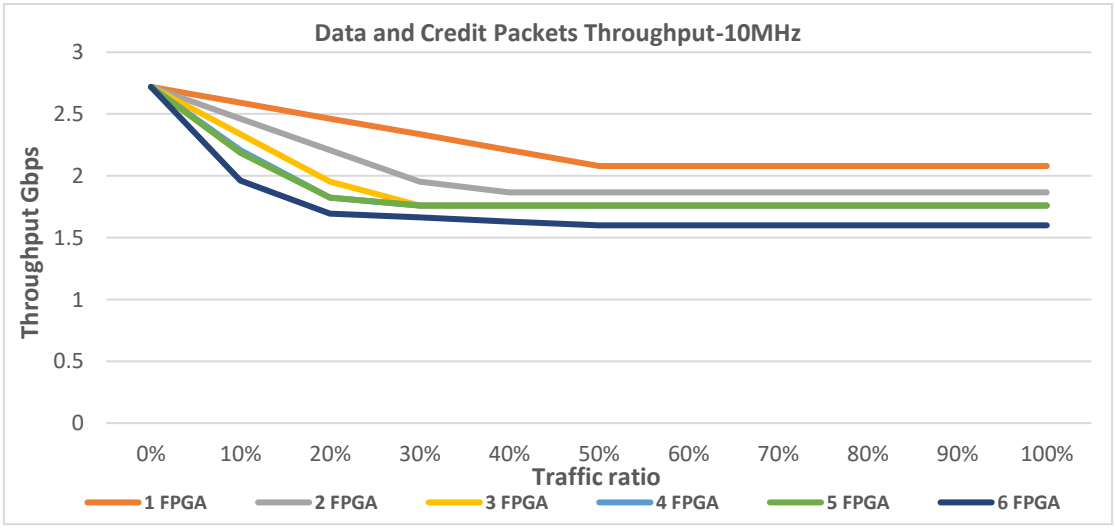


(a) Data Packet throughput – 150MHz

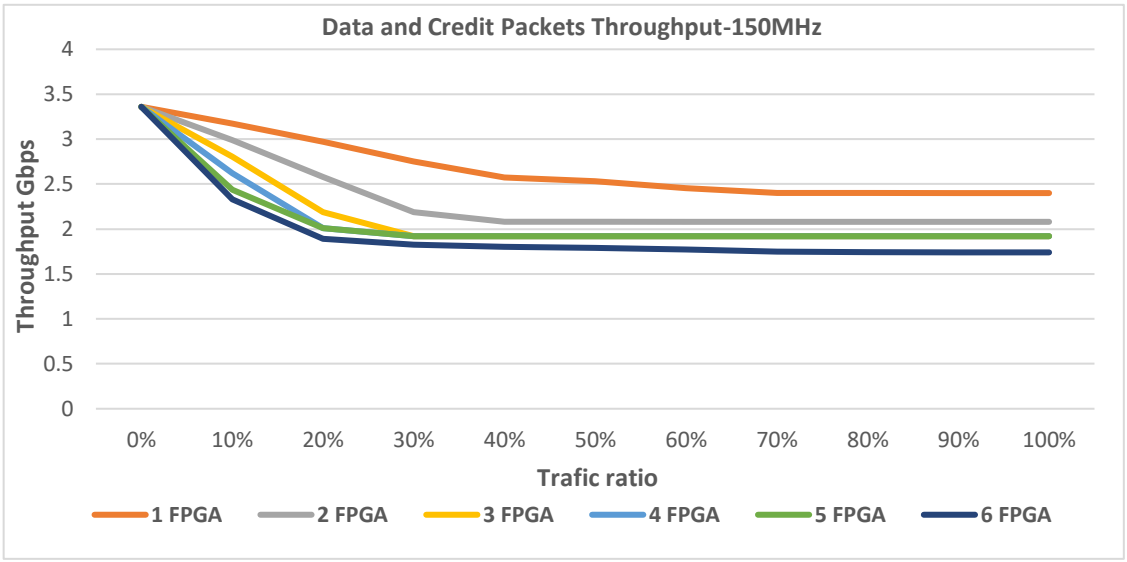
Figure 5:6 Data packet throughput for 10MHz and 150MHz

process is formed of computationally intensive hardware blocks that have several DSPs in their data path.

Figure 5:7 illustrates data for the same experiment as performed above with throughput for credit packets accounted for alongside the data packets. Figure 5:7 measurements are obtained by adding 1.44 to the readings in Figure 5:6. This is useful for calculating the transceiver link total throughput as approximately 1.44 Gbps of the throughput is dedicated to the credit packet per link. For the computing cores operating frequency of 10MHz and 150MHz, the maximum throughput is 2.72Gbps and 3.36Gbps respectively.



(a) Data and Credit Packet Throughput-10MHz



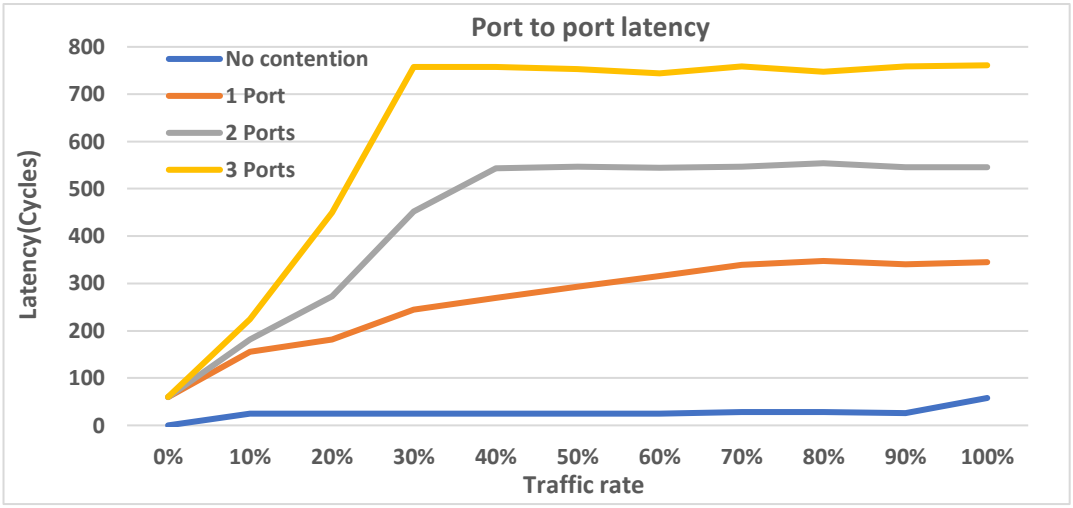
(b) Data and Credit Packet Throughput-150MHz

Figure 5:7 Data and credit packet throughput for 10MHz and 150MHz

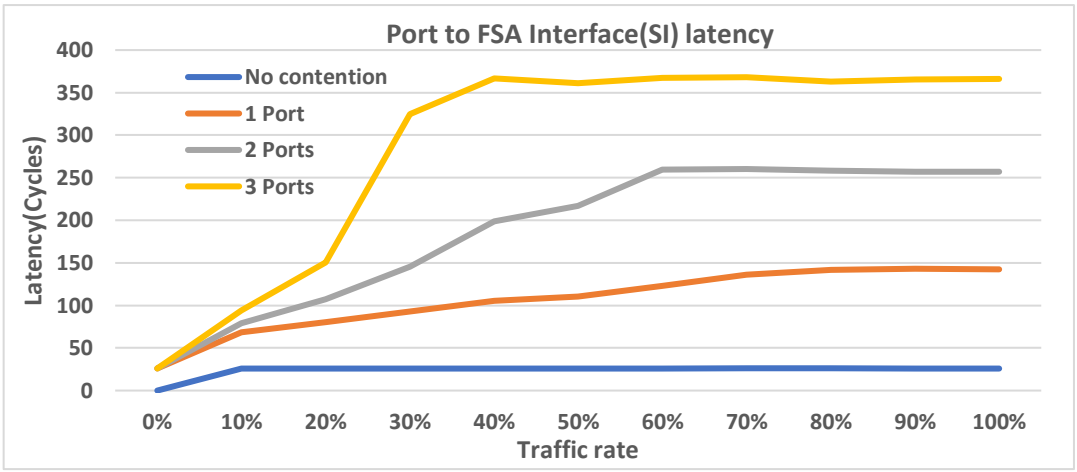
5.4 Latency

Figure 5:8 illustrates the latency experienced by packets while passing through the NoC router under various traffic ratios. The latency numbers were obtained using SignalTap II to record packets on input and output ports. The latency values are reported as the average latency.

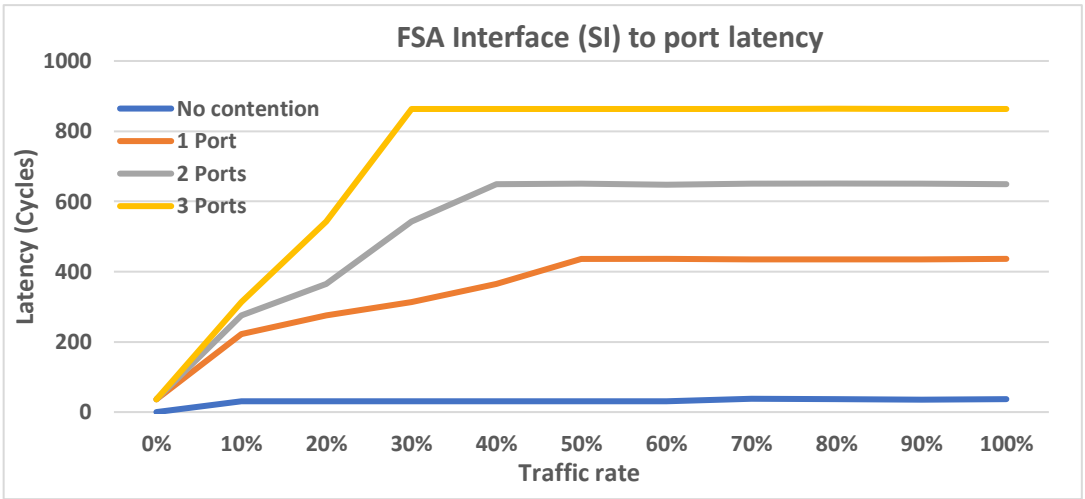
The experiment setup is similar to that for the throughput. The AstroByte platform in Figure 5:1 was used and latency was calculated by measuring the number of cycles packets take from an input port to and output port of node (1, 1). The length of each simulation was 131072 cycles, which is the maximum capacity for SignalTap. All the packets in this time were captured at an input port and an output port and the latency was observed. The average latency of all the packets that made their way through the router was recorded.



(a) Port to port latency



(b) Port to FSA Interface(SI) latency



(c) FSA Interface (SI) to port latency

Figure 5:8 Latency figures for AstroByte platform

Firstly, packets were routed from node (1,0) to node (1,2) at incremental rate without contention and the latency of packets was observed. Later, traffic generated by node (1,0) was fixed at 100% while nodes (0,1), (1,1) and (1,2) sent incremental traffic rates to node (2,1) through node (1,1). This causes

traffic at the south port of node (1,1) which is required by the experiment. The latency figures provided are represent the average number of clock cycles from an input to an output of one FPGA. If cross-FPGA latencies is required, the figures below have to be added by 13, the latency introduced by Intel transceivers [129].

Figure 5:8(a) shows latency of packets from an external port to another external port (N, S, E, W). To explain the latency calculation method, latency measurement of "2 ports" trajectory will be described next. Ports (1,0), (1,2) and (0,1) were configured by FCMP to send packets to port (2,1). The latency figures were calculated at the router node (1,1), from input west port (receives the traffic of node (1,0)) to its output south port (towards the destination node (2,1)). The reference node was node (1,0), hence its traffic was fixed at 100%. The traffic rate of nodes (0,1) and (1,2) were increased at 10% rate and the latency was calculated. As per Figure 5:8, the latency starts from 58 (router cycles) and increases in a linear fashion, before stabilizing at a range of 543-547 cycles from 40% onwards.

Figure 5:8(b) shows latencies in the cases when packets are routed from the input buffers (InCh FIFOs) of the external ports to SI. The figures are significantly lower than the latency values reported in Figure 5:8(a). This is due to the difference in the architecture for port 0 as shown in Figure 4:4. The difference in ATT latency waiting value, which is 75 for external ports and 5 for port 0, affects the latency of processed packets. This because packets have to wait less in the InCh FIFOs before getting correct information about free spaces in the SI buffers by the tracking mechanism discussed in section 4.4. Figure 5:8(c) shows the latency figures for moving packets from port 0 to the external ports1-4. The latency figures see a slight increase when compared to external port to external port latency. This is because of the proximity of InCH FIFO of port 0 to the FSA interface (SI). The effect is packets move to port 0 FIFO sooner and stay for longer before being forwarding to the destination router via one of the external ports.

5.5 Simulations and Under-sampling

To evaluate simulation times the 3x3-FPGA platform was exercised with data. Counters were used instead of FSA elements. Nodes (0,1), (1,0), (1,2), (1,2) send data packets to their two neighbours while sending monitoring data to the FCMP at node (1,1) at the same time. Results are shown in Table 5:1.

Table 5:1 Under-sampling by running simulation for 20,480,000 cycles

Granularity	Elapsed time (Sec)	Iteration/Node	Biological time scale (Sec)
1	84.79	20,480,000	20,480
10	87	204,800,000	204,800
100	147.7	2,048,000,000	2,048,000
1000	762.3	20,480,000,000	20,480,000

Total data collected is 163,840,000 words, including header and data flits from four source nodes. This means the total simulation data collected, represented by the number of data flits (50% in total) is 81,920,000. Dividing this figure by four (the number of FPGAs that produce traffic) yields the iterations carried out per node, amounting for 20,480,000 cycles in the 150MHz domain/per node. In practice there is a slight difference between data collected from various nodes (< 0.05 %) because of slight variations of instant traffic experienced by the FSA cores at different nodes. This is caused by different FPGA clock domains and the status of priority at the shared destination FPGA (the FCMP core).

Under-sampling means collecting data at a lower level of granularity (e.g. 1 in 10 packets). Granularity of one means every data sent from source to destination routes has also been forwarded to the FCMP, i.e. no under-sampling. A granularity of 1,000 means 1/1,000 of the data routed to destination node has been received by FCMP for monitoring. The granularity column in Table 5:1 represents the rate of under-sampling in this experiment.

The under-sampling feature allows for faster simulations of large biological timescales however at the cost of data granularity. Using full granularity, 20,480 seconds (~5 hrs 40 min) of biological time can be simulated in 84.79 seconds. However, if granularity is reduced, simulation of under 57 hours is possible in 87 seconds (1 min 27 sec). Down sampling further would enable users to simulate close to 569 hours in just 147.7 seconds (2 min 27.7 sec), or approximately 5,689 hours in 762.3 seconds (approximately 13 minutes).

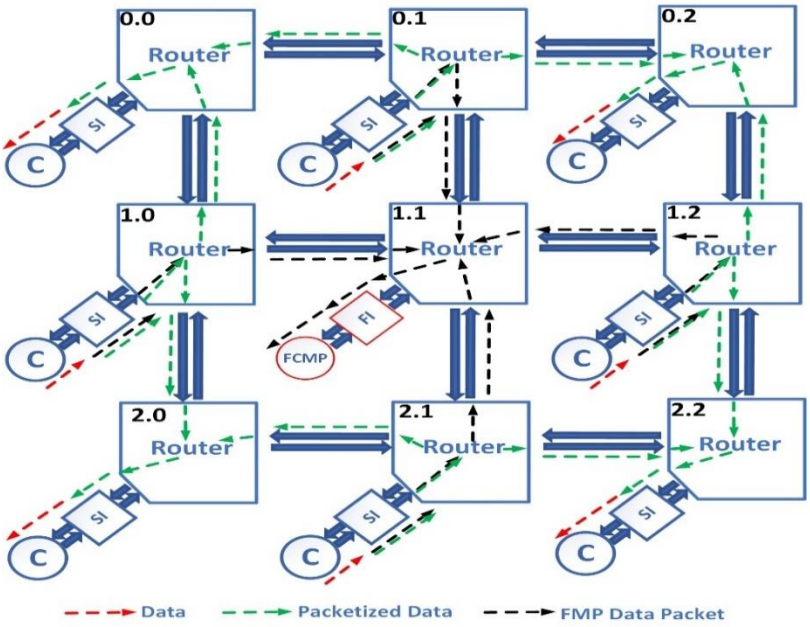


Figure 5:9 Prototype AstroByte configuration

5.6 Multi-FPGA SANN implementation

The next set of experiments aim at evaluating performance of the AstroByte platform by integrating the FSA, reported in Chapter 3, into the multi-FPGA NoC architecture.

Figure 5:10 illustrates a four FPGA platform with FSA elements (neurons, an astrocyte and synapses) mapped into separate FPGA cores (C). The first neuron entity (N1 entity), composed of N1 and the associated tripartite synapses, spike and probability generators, is mapped into node (0,1) in the multi-FPGA NoC mesh structure. N2 entity is mapped to node (1,0) while the astrocyte process is mapped to node (0,0) and FCMP is located at node (1,1).

The FCMP sends configuration packets at the start, routing information from SANN components to their destination. The Astrocyte process sends eSP signals to both neuron entities, which in turn, send the astrocyte 2AG. The neurons also send their average frequencies to the FCMP platform, enabling users to monitor the rate of firing. The FSA, as discussed in the previous chapters utilizes astrocyte for giving a SANN self-repairing capability.

5.7 SANN multi-FPGA simulation

Simulation data of the example SANN executing on the AstroByte platform in Figure 5:10 are shown in

Unlike Table 5:1, under-sampling is carried out by collecting the same amount of data with reduced granularity and less time rather than collecting larger amount of data with reduced granularity that takes longer.

For this experiment, 401,408,000 iterations of data are collected per FPGA. For under-sampling, the experiment was run for the same number of iterations to

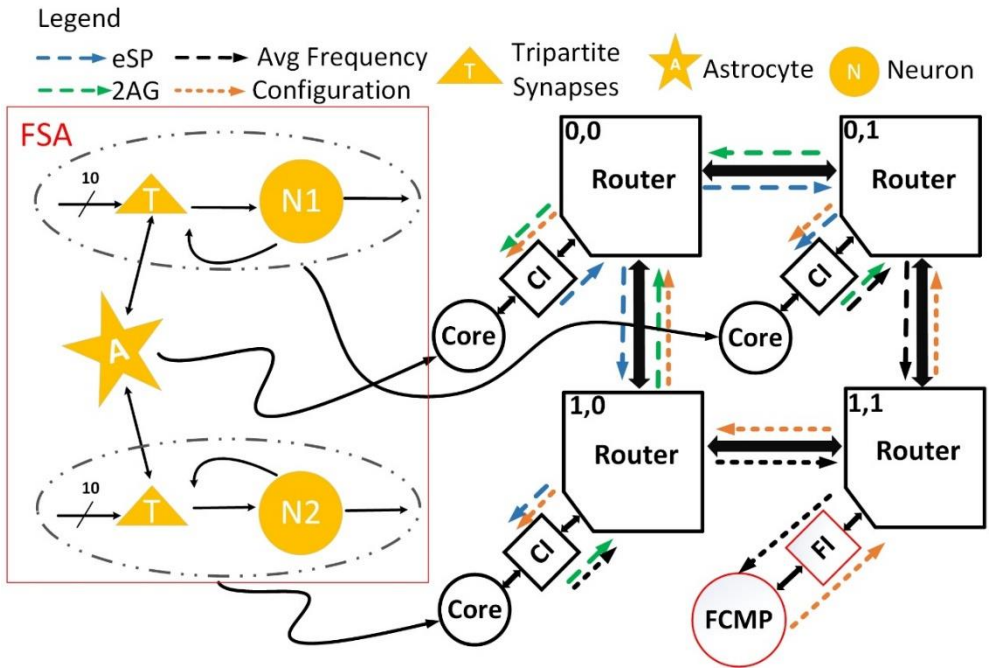


Figure 5:10 Prototype SANN mapping on AstroByte

see the impact of under-sampling on simulation time. As per Table 5:2, simulating 401,408,000 iterations/node takes 1044.8 seconds with no under-sampling. With an under-sampling rate of 10, the simulation is completed in 158.14 seconds (x6.6 speedup). Down sampling at the rate of 100 results in reducing the experiment time to 88.06 second (x1.8 speedup) while further under-sampling results in a dismal speed up of x1.064. This is because at higher under-sampling rate the platform comes closer to its maximum performance with the FCMP platform being the system bottleneck as discussed in chapter 3.

Table 5:2 Under-sampling for 2x2 FPGA AstroByte by decreasing simulation cycles

Granularity	Elapsed time (Sec)	Iteration/Node	Biological time scale (Sec)
1	1044.8	401,408,000	401,408
10	158.14	401,408,000	401,408
100	88.06	401,408,000	401,408
1,000	82.76	401,408,000	401,408

5.7.1 Acceleration

The Acceleration gained from using the prototype AstroByte platform of Figure 5:10, will be assessed in this section.

The same SANN structure developed in chapter 3 was used but was implemented on a 2x2 FPGA AstroByte structure as opposed to a single FPGA as in in chapter 3. Table 5:3 shows that speedup factors of between x162-x188 can be gained by simulating FSA on AstroByte platform, compared to an equivalent software Matlab implementation [6].

The acceleration obtained from implementing SANN on a single FPGA FSA (Table 3:5) and that a 2x2 AstroByte platform (Table 5:3) can be compared. Despite the NoC interconnection overhead, the 2x2 AstroByte platform can maintain over 75% of the speedup factor that is possible on a single FPGA.

Table 5:3 Comparison between AstroByte and software implementations

Biological Time (S)	Iterations (Cycles)	Matlab (Seconds)	AstroByte (Seconds)	Acceleration Factor
400	400,000	153.72	~0.938	163.9
600	600,000	242	~1.4	172.9
800	800,000	327	~1.9196	170.3
1,000	1,000,000	381	~2.3448	162.5
1,200	1,200,000	506	~2.71	186.7
3,600	3,600,000	1,500	~8.1618	183.8
5,000	5,000,000	1,960	~10.7723	182
10,000	10,000,000	4,095	~21.7448	188.3

Further mitigation of NoC interconnection overhead will be explored in future work.

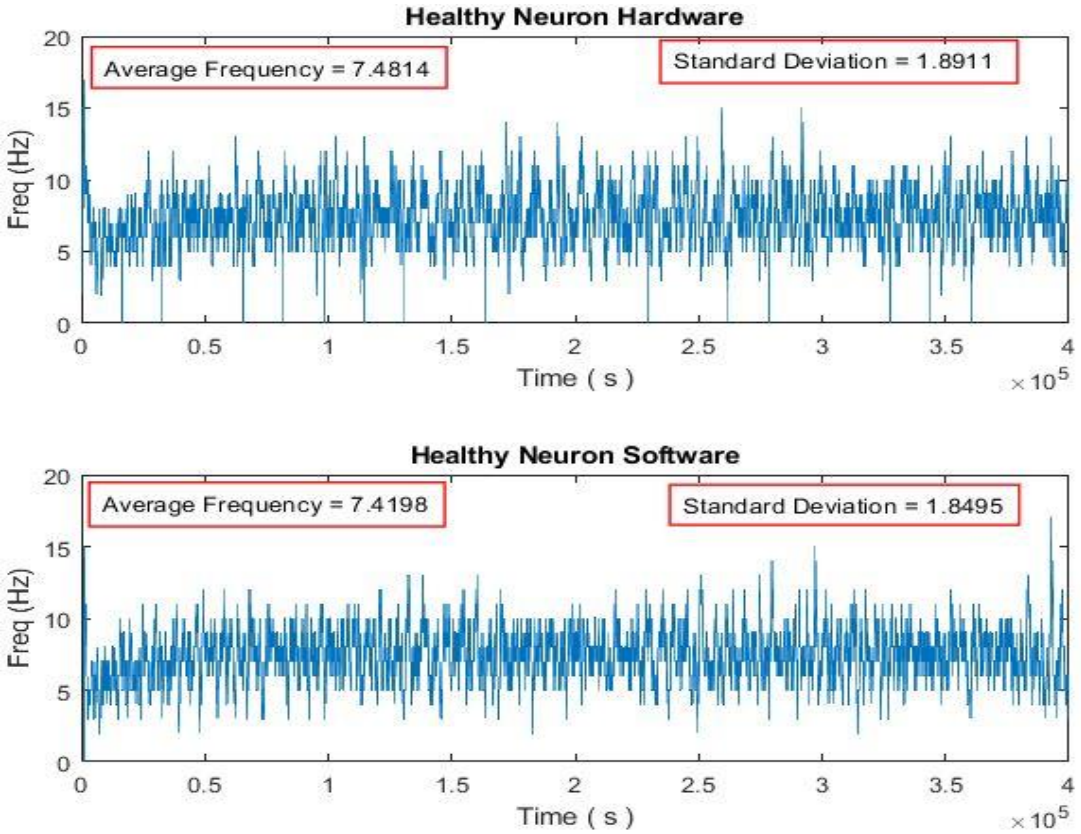
From Table 5:3 last entry, 2hrs:46min of biological time can be simulated in 1hr: 8mins (4095 second) using Matlab. The same simulation takes 21.74 seconds

using the proposed AstroByte platform, which means a speedup factor of x188.3 over the equivalent Matlab implementation.

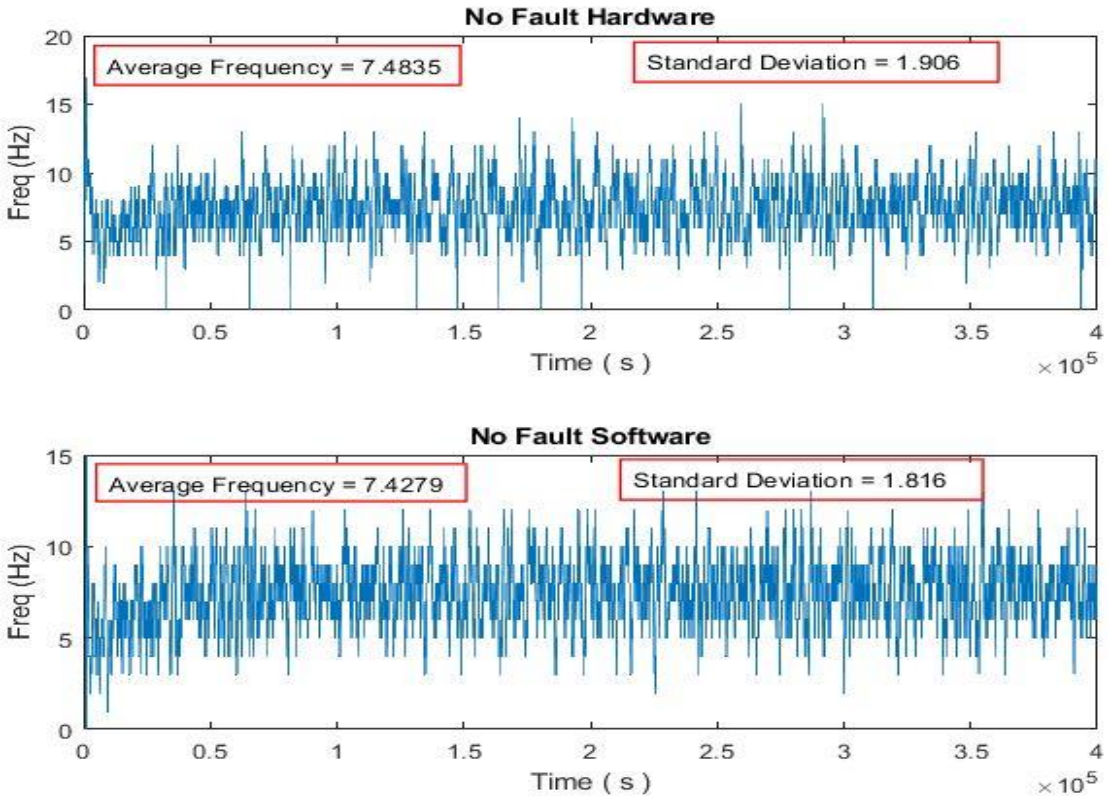
The main reason for performing comparisons with Matlab is due to the availability of an in-house model for the SANN. If this work dealt with SNNs, other software/programming languages such as C, C++ and Python could have been used for comparison. However, unlike neurons and synapses, standard models/libraries for astrocyte don't exist for self-repairing SANNs. Overall, for low-level programming languages (e.g. C and C++) one would expect, purely based on logical reasoning, that the FPGA advantage in terms of acceleration would shrink. However, to which degree depends of a great many factors, such as the specifications for the FPGA and PC used, how optimized the algorithms are, how many cores are in the processor and the level of parallelism in the software code. Another factor is that Matlab execution times are been improved constantly with each new release of software and hardware, making a reasoning-based judgement even more difficult. Only practical research in the area of comparing different software can answer these questions which fall beyond the scope of this work.

5.7.2 Accuracy

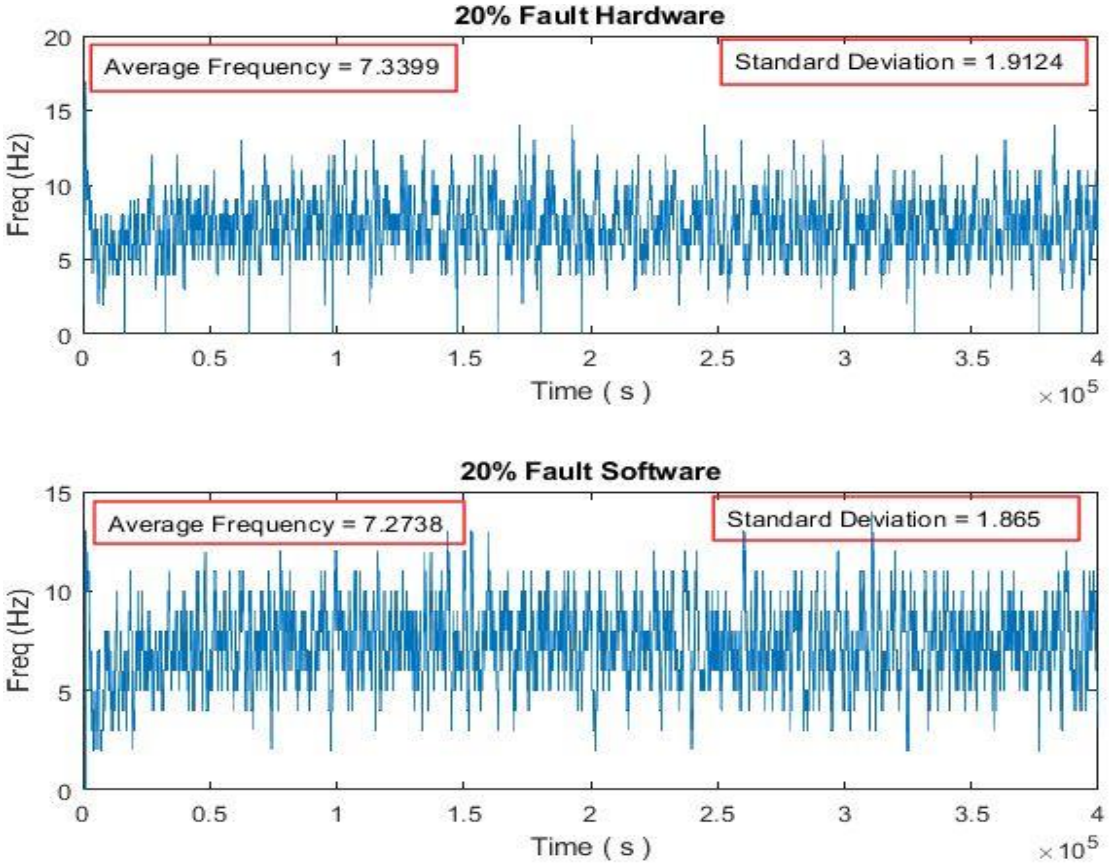
Figure 5:11 illustrates comparison between AstroByte FSA implementation and equivalent Matlab software model in terms of average frequency and standard deviation. Frequency response trajectories for the healthy neurons and various ratios of faults, including no fault situation, are presented. The average frequency and standard deviation of the trajectories are also shown in the figure. It is evidence that a multi-FPGA AstroByte platform can simulate SANNs with high accuracy. The maximum difference between the average two frequencies can be seen in Figure 5:11-f, with a difference of 0.0939 Hz (~0.016%).



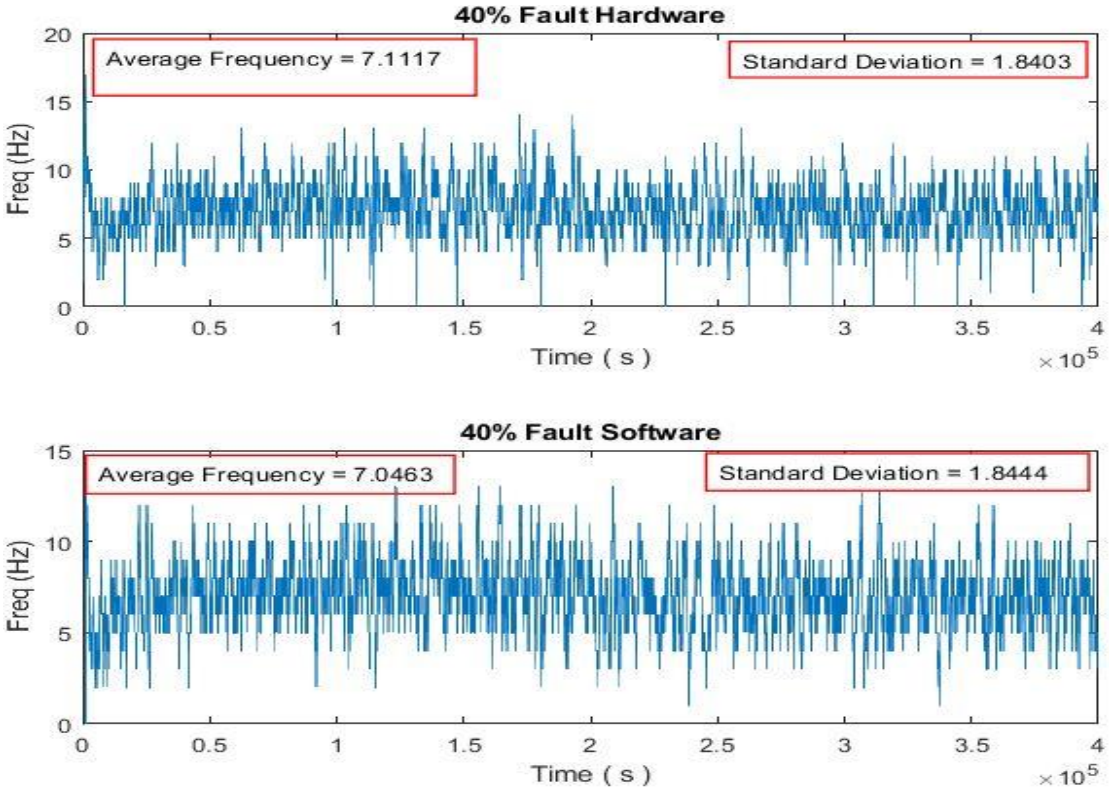
(a) Healthy neuron average frequency



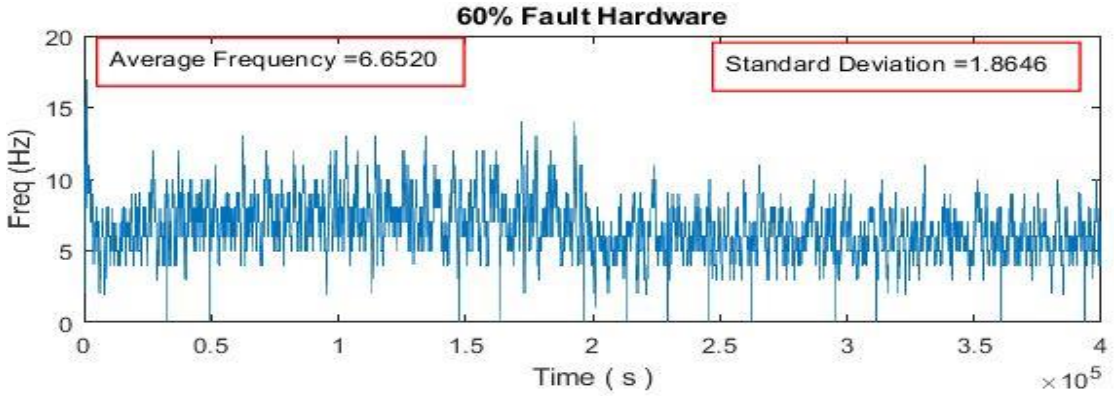
(b) Faulty neuron average frequency at 0% fault rate (no fault)

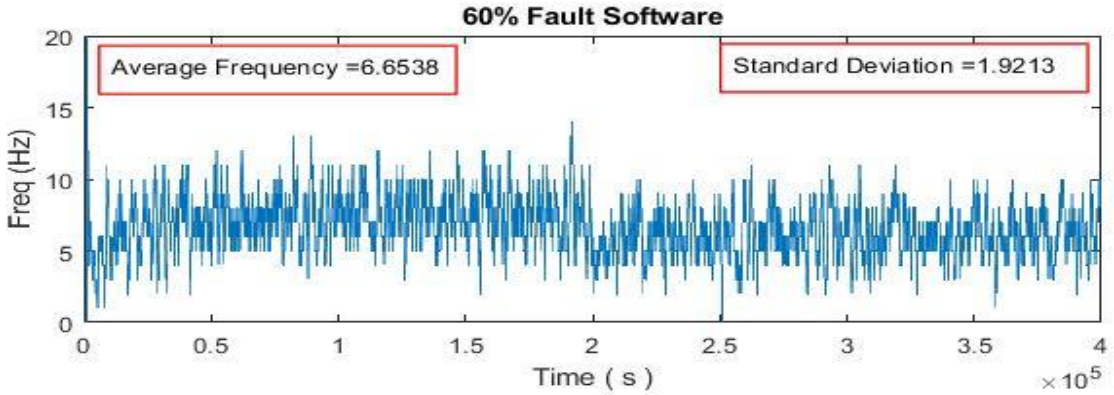


(c) Faulty neuron average frequency at 20% fault rate

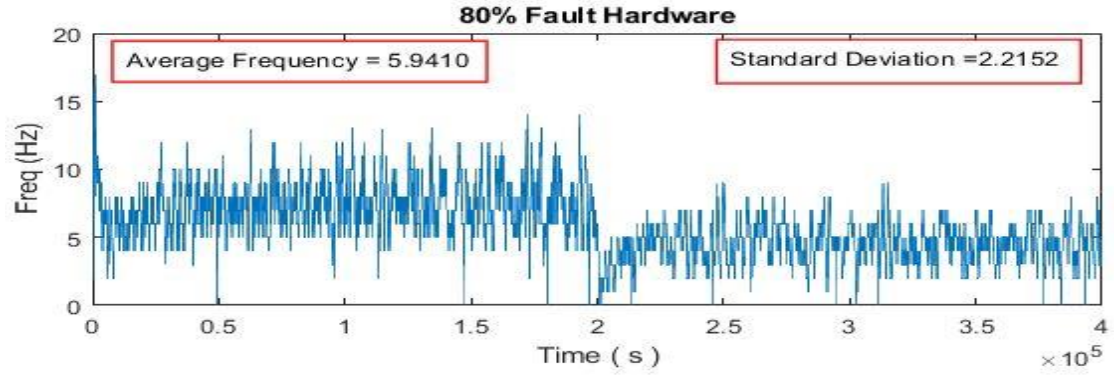


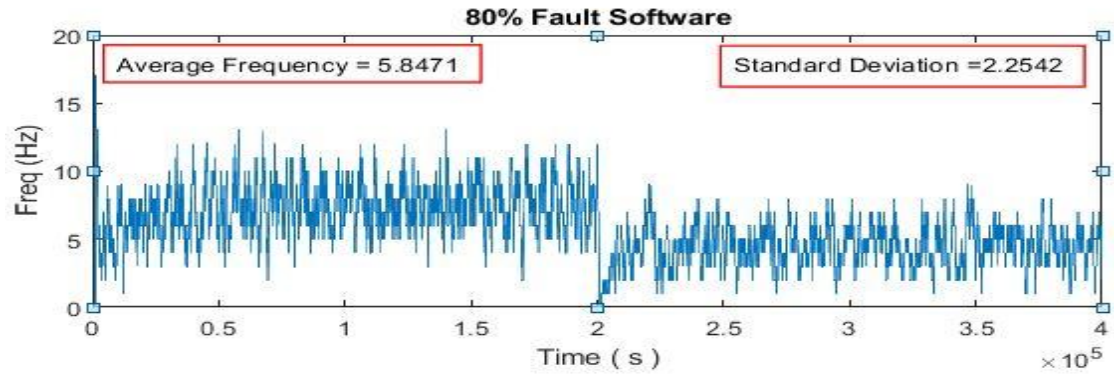
(d) Faulty neuron average frequency at 40% fault rate





(e) Faulty neuron average frequency at 60% fault rate





(f) Faulty neuron average frequency at 80% fault rate

Figure 5:11 Accuracy comparison between AstroByte and Matlab

5.7.3 Comparison with SNAVA and Bluehive

This section provides a comparison of AstroByte with Bluehive [97] and SNAVA [98] as both share several characteristics with AstroByte. They are both multi-FPGA platforms aimed at SNN simulations, offer some degree of programmability, and have mechanisms for monitoring simulation data. Despite these shared features, the architecture and implementation vary widely among the three platforms. An overview of the three platforms features is given in Table 5:4.

Both Bluehive and SNAVA simulate SNNs in real-time meaning that simulations take as long as an equivalent biological model while AstroByte achieves approximately x460 times speedup over its biological counterpart. In terms of software, all the three platforms have reported speedup gains with the highest being Astrobyte with X188 speedup.

Bluehive uses off-chip DDR RAM to store a netlist for the SNN mapping, allowing users to implement various SNN architectures. AstroByte uses on-chip SRAM for storing configuration packets. SNAVA offers programmability by means of a user interface and a programming language. In the reported SNAVA realization, each FPGA has to be connected to a dedicated Ethernet cable for reconfiguring purposes, putting a limitation on its scalability as will be dependent on the number of Ethernet ports the host PC can provide. AstroByte, on the other hand, uses FCMP to send configuration packets on the NoC interconnection, requiring only the FCMP node to be connected to a PC.

The only reported programmable feature of Bluehive is network structure. SNAVA allows the user to implement different neuron models and SNN architectures. AstroByte offers 14 configurable attributes shown in Table 4:1, however the neuron model is not reconfigurable.

Bluehive and AstroByte use SATA cables for facilitating communication between different boards. AstroByte implements Intel GXB transceivers and custom designed controllers. Bluehive, interestingly doesn't use any transceivers. An approach utilizing CRC protection and replay functionality has been implemented. No details have been provided as how frequency mismatch between different boards is dealt with. SNAVA deploys SMA cables and Xilinx Aurora protocol for this purpose.

Among the three platforms, only AstroByte has self-repairing capability, gained through incorporating an astrocyte process however at the cost of extra hardware. Figure 3:4 shows the resource usage for the astrocyte process in terms of FPGA DSPs and LUTs. These figures can be reduced by optimizing the astrocyte process, for example, through implementing less complex equations -less faithful to the biological model-that still maintains the basic functional behaviour of the astrocyte. Optimizing the astrocyte process would be an iterative process. One would have to implement an optimization

technique, simulate the SANN with the optimized astrocyte and observe the response of the system. This process would continue until finding a satisfactory balance between behavioural accuracy and hardware resource usage. Optimizing the astrocyte process falls beyond the scope of this work.

In summary, the following distinguishes AstroByte from SANAVA and Bluehive work:

1. Incorporation of astrocytes to provide SANNs with self-repairing abilities.

2. Ability to efficiently reconfigure the SANN and capture real-time simulation data using the same NoC infrastructure used for data communication.

3. Able to achieve accelerations of up to x188 while capturing full-scale real-time simulation data.

5.8 Chapter conclusion

This chapter was focussed on the evaluation of the multi-FPGA AstroByte platform. Data from experiments on DE4 FPGA boards were used for evaluating performance matrixes of the multi-FPGA NoC. The experiment setup for the performance calculation was laid out. Figures representing throughput and latency were then presented. An example of sample AstroByte platform was used to test data acquisition and under-sampling characteristics. The FSA developed previously was integrated mapped to AstroByte to test for acceleration and accuracy. Results indicate that acceleration factors of x162-x188 were possible with minimal accuracy degradation. Lastly, a comparison between AstroByte and each of SNAVA and Bluehive was provided and demonstrated that AstroByte 1) features more programmable attributes; 2) is the only platform to incorporate astrocytes to facilitate self-repair; 3) provides a less-intrusive data acquisition platform (no need for separate connections or stopping simulations); 4) is the only multi-FPGA platform that is dedicated for accelerating simulations as opposed real-time simulations.

Table 5:4 Comparison between AstroByte, SNAVA and Bluehive

	AstroByte	Bluehive [97]	SNAVA [98]
Topology	Mesh	3D Torus	Ring
HDL language	VHDL	BSV- Bluespec System Verilog	N/A
Acceleration/Real-time	Acceleration (x426-x460)	Real-time	Real-time
Acceleration/Software	Acceleration (x162-x188)	X162	X129
Self-repair/ Astrocyte	Yes	No	No
Configuration method	By sending configuration packets from FCMP through NoC infrastructure	By altering address DRAM content	By using Ethernet from a host PC to program the configurable components
Configurability	14 configurable attributes Table 4:1, Figure 4:3	Only neural mapping is reconfigurable	Neural model and Neural network mapping
Result monitoring	Real time data acquisition using NoC infrastructure	Simulation has to stop completely to readout simulation results from off-chip memories.	Real-time simulation by Ethernet cable. Each board needs an Ethernet connection
Resource TDM	No	Yes	Yes
Board to board bandwidth	4.8 Gbps	18 Gbps	N/A

Chapter 6: Conclusion and future work

6.1 Concluding summary

It is evident that the current Von Neuman computing architecture is facing obstacles regarding scalability and power efficiency. On top of that, detecting and recovering from faults in traditional Von Neuman machines and others are not intrinsic and require meticulous architectural planning that often adds to the complexity of these systems. Brain-inspired computing has the potential to address these challenges by means of massively parallel structures that perform computations in an extremely power efficient manner. In particular, self-repairing is embedded in the brain's functionality that allows for problem-free operation despite existence of faulty neurons in the SANN.

Researchers have explored SNNs with the goal to firstly simulate the brain's functionality and, secondly, to capture the power efficient and massively parallel nature of the brain in a novel, next-generation computing systems. The SANN implemented in this work model the self-repairing characteristics of the brain by incorporating astrocytes into SNNs. However, simulating SANNs in software is extremely time consuming due to the sequential nature of software execution and the complexity of astrocytes. Therefore, this work focusses on accelerating SANN simulations by designing dedicated FPGA Hardware and interconnect strategies for neurons, astrocytes and tripartite synapses. To address the challenge of simulating large-scale SANNs on FPGA hardware, a novel multi-FPGA infrastructure for accommodating large-scale SANN simulations was developed. Additionally, hardware and software tools were developed for capturing simulation data for analysis.

6.2 Main contributions

1- A dedicated FPGA hardware for accelerating SANN simulations, the FSA, was realized. Hardware blocks were designed for neurons, astrocytes and the tri-partite synapses using VHDL and verified on FPGA hardware. The FSA has a self-repairing capability and is able to accelerate SANN simulations up to 1,067 times faster compared to an equivalent Matlab model. Fixed-point arithmetic was used as opposed to floating point for efficient hardware implementations. Bit resolutions of 24, 32 and 40 were explored and comparisons were carried out as regards the different implementations accuracy and hardware consumption. The 32-bit realization was chosen as it provides the best balance between accuracy (compared to an equivalent double floating-point arithmetic Matlab model) and FPGA resource usage. It is proven that the FSA has good resiliency to both moderate and severe faults.

2- A novel FPGA Configuration and Monitoring Platform (FCMP) for configuring and monitor FSA simulations was developed. Nios II soft

processor was used to acquire data from SFA and save the data in memory (DDR2 SDRAMs or SRAMs). The acquired data was then transferred to a PC through Ethernet cable via socket programming. Results showed that, if DRAMs are utilized for storing acquired data, speedup of up to x65 is possible when compared to Matlab. The acceleration gained can be improved significantly by replacing the DRAMs with SRAMs, in which case simulations can be run up to x249 times faster than Matlab software. The FCMP was later adapted for integration with the AstroByte NoC infrastructure. The FCMP is able to store configuration packets in an internal memory and issue commands to inform the associated router to send the configuration packets to the other nodes on the NoC.

3- A NoC router was developed to enable the implementation of a multi-FPGA NoC platform. The router utilises four external ports (N, S, E, W) and one internal port for communicating with the computing core. Intel GBX IPs (transceivers) were integrated within the router's architecture to facilitate cross-FPGA communications. Suitable techniques were utilized for clock domain crossing in the router. An appropriate mechanism was realized for updating information regarding the free buffer spaces in the downstream routers. A fair arbitration scheme arbitrates requests and controls access to the output ports.

4- Two interface blocks were provided for connecting the NoC routers to the various computing blocks. The first interface is FI, which connects the internal computing core of the router to the FCMP. The second interface is SI and is located between the router and FSA elements (or test counters).

5- The above-mentioned contributions were integrated in a single platform called the AstroByte that is able to carry out large scale SANN simulations. The AstroByte is able to efficiently reconfigure the FSA and capture real-time simulation data using the same NoC infrastructure used for data communication. Using a 2x2 FPGA AstroByte platform, a speedup of x188.3 times is possible when compared to an equivalent Matlab model, while capturing 100% of the data produced by FSA elements. AstroByte can also support computing cores that operate at a frequency of 150 MHz. Throughput tests proved that the AstroByte can communicate data at 2.72Gbps and 3.36Gbps for computing cores that run at 10 MHz and 150 MHz respectively.

6.3 Future work

In this section the proposed future works to improve this PhD research will be discussed.

- Optimizing the astrocyte process with a smaller footprint: Currently a reduced fixed-point implementation is used for the astrocyte process to reduce FPGA footprint. Yet there are still other opportunities to make the astrocyte process more hardware efficient. One such opportunity is implementing a less biologically faithful astrocyte model. This would allow for a reduction in the number of equations in hardware which leads to utilizing less DSPs and other FPGA resources. This has been left to future work as striking a balance between biological faithfulness and functional accuracy needs further research.

- Optimizing astrocyte process for speed: The astrocyte process puts a limit of 10 MHz on the operation frequency of the FSA. The operating frequency can be further optimized by utilizing techniques such as pipelining. Reducing the accuracy of the biologically faithful astrocyte, as discussed above, will also help with increasing the operating frequency. This is because a less biologically faithful astrocyte model requires shorter combinational paths and allows for a higher clocking frequency.

- Researching new FCMP architectures for gaining further acceleration performances: The effects of the FCMP platform on acceleration was discussed in Section 3.4.2. It was mentioned that using FCMP for data monitoring reduces the speedup gained. Several approaches can be utilized to reduce the overhead imposed by the FCMP. Firstly, instead of transferring data using software that runs on Nios IIII soft processor, data can be transferred at PHY level directly to the PC. This calls for implementing PHY IP instead of the Ethernet IP. Secondly, more than one connection between the PC and AstroByte can be utilized to parallelize the transfer of the monitoring data to speed-up the data transfer process, thus getting around a major bottleneck in the system.

- Virtual channels for the routers: The current iteration of the NoC router design does not implement virtual channels. Virtual channelling would utilize a higher percentage of the available bandwidth when the NoC channels are congested. Virtual channelling will be useful for AstroByte platform if the FSA core is optimized for speed, as this would mean more data transfer and more congestion in the NoC.

- Adaptive routing scheme: The current routing technique is X-Y routing. Packets will always follow the same path from a source to a destination regardless of the congestion in the network. An adaptive routing scheme would allow packets to choose a less congested path if there were a high congestion at a particular node. Again, as is the case with implementing virtual channelling, AstroByte would benefit from such a scheme after the FSA is optimized for speed.

- Modify SI (FSA Interface) to support full scale simulations: - The current SI has been designed to support a prototype multi-FPGA FSA implementation for which an equivalent Matlab model is available. However, with relatively mild effort, the SI can be configured to support many neurons and an astrocyte on each FPGA. To achieve this, the AstroByte data format needs to be modified to enable users to choose, for example, whether a particular node should be a neuron (or a multiple neuron) node or an astrocyte node. Additionally, the data format and SI design should give the capability of mapping large-scale SANNs by dedicating a specific field in the data format that allows addressing a particular set of synapses for a neuron or group of neurons.

6.4 Self-critic

While the previous section offered additional ways for improving this research, this section focusses more on the improvements that always seemed within reach and the author wished he could achieve before the project's end.

Firstly, the affair of improving the astrocyte process for speed always seemed doable within the timeframe dedicated for the PhD project. However, due to the time required for architectural exploration of the AstroByte platform, not enough time could be dedicated for optimizing the astrocyte process for speed.

Additionally, the author observed, during designing the FCMP platform, that the size of buffers in the Ethernet IP is customizable. The virtual buffers that has to be dedicated in Matlab software for socket programming is also user-defined. There seem to be a potential for optimization here, e.g. finding appropriate values for both buffer sizes for optimizing the speed of data movement from FPGA hardware to the PC. Unfortunately, this also had to be sacrificed in favour of more crucial tasks.

Finally, the author would like tp implement a more complex SANN than the prototype one for which the results has been reported. The road map was to test the prototype SANN available in Matlab to be able to carry out meaningful comparisons between the hardware and software models. Therefore, the data format was designed with the prototype SANN in mind however it didn't turn out to be fit for larger scale implementations. This is due to the fact that the data format doesn't specify the address of target synapses connected to the target neurons. Although this requires minor effort compared to what has already been achieved, the project was closing to an end and this task had to be left out for future works.

References

[1] S. Karim *et al.*, “Assessing Self-Repair on FPGAs with Biologically Realistic Astrocyte-Neuron Networks,” in *Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, 2017, vol. 2017-July, pp. 421–426, doi: 10.1109/ISVLSI.2017.80.

[2] J. M. Rabaey, “System-on-Chip-Challenges in the Deep-Sub-Micron Era,” in *Interconnect-Centric Design for Advanced SoC and NoC*, Boston: Kluwer Academic Publishers, pp. 3–24.

[3] W. J. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann Publishers, 2004.

[4] J. Grollier, D. Querlioz, and M. D. Stiles, “Spintronic Nanodevices for Bioinspired Computing,” *Proc. IEEE*, vol. 104, no. 10, pp. 2024–2039, 2016, doi: 10.1109/JPROC.2016.2597152.

[5] S. R. Kulkarni, A. V Babu, and B. Rajendran, “Spiking Neural Networks – Algorithms , Hardware Implementations and Applications,” no. 1, pp. 426–431, 2017.

[6] J. Wade, L. McDaid, J. Harkin, V. Crunelli, and S. Kelso, “Self-repair in a bidirectionally coupled astrocyte-neuron (AN) system based on retrograde signaling.,” *Front. Comput. Neurosci.*, vol. 6, no. September, p. 76, 2012, doi: 10.3389/fncom.2012.00076.

[7] S. Y. Gordleeva, S. V Stasenko, A. V Semyanov, A. E. Dityatev, and V. B. Kazantsev, “Bi-directional astrocytic regulation of neuronal activity within a network.,” *Front. Comput. Neurosci.*, vol. 6, no. November, p. 92, 2012, doi: 10.3389/fncom.2012.00092.

[8] M. Ranjbar and M. Amiri, “On the role of astrocyte analog circuit in neural frequency adaptation,” *Neural Comput. Appl.*, pp. 1–13, 2015, doi: 10.1007/s00521-015-2112-8.

[9] A. Araque, V. Parpura, R. P. Sanzgiri, and P. G. Haydon, “Tripartite synapses: Glia, the unacknowledged partner,” *Trends in Neurosciences*, vol. 22, no. 5. pp. 208–215, 1999, doi: 10.1016/S0166-2236(98)01349-6.

[10] 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. Reconfigurable Comput.*, vol. 2009, pp. 1–13, 2009, doi: 10.1155/2009/908740.

[11] S. Karim *et al.*, “FPGA-based Fault-injection and Data Acquisition of Self-repairing Spiking Neural Network Hardware,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5, doi: 10.1109/ISCAS.2018.8351512.

[12] J. Liu, J. Harkin, L. Maguire, L. McDaid, J. Wade, and M. McElholm, “Self-

Repairing Hardware with Astrocyte-Neuron Networks," *IEEE Int. Symp. Circuits Syst.*, pp. 1–4, 2016.

[13] J. Liu, J. Harkin, L. P. Maguire, L. J. Mcdaid, and J. J. Wade, "SPANNER : A Self - Repairing Spiking Neural Network Hardware Architecture."

[14] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks.* Elsevier, 2003.

[15] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer (Long. Beach. Calif).*, vol. 35, no. 1, pp. 70–78, 2002, doi: 10.1109/2.976921.

[16] A. Hemani *et al.*, "Network on Chip : An architecture for billion transistor era." 2000, Accessed: Aug. 13, 2019. [Online]. Available: https://www.semanticscholar.org/paper/Network-on-Chip-%3A-An-architecture-for-billion-era-Hemani-Jantsch/decde560acf91a4e952280947016d4cac8e66b00.

[17] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," pp. 684–689, 2002, doi: 10.1109/dac.2001.935594.

[18] S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design THIRD EDITION*, vol. 12, no. 1. 2008.

[19] T. L. Floyd and T. L., *Digital fundamentals*. Pearson Prentice Hall, 2006.

[20] M. M. Mano and C. R. Kime, *Logic and computer design fundamentals*. Prentice Hall, 1997.

[21] "What is an FPGA?" https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html (accessed Jun. 26, 2019).

[22] F. Rosenblatt, "The Perceptron- a perceiving and recognizing automation," 1957.

[23] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *Bull. Math. Biol.*, vol. 52, no. 1–2, pp. 25–71, 1990, doi: 10.1007/BF02459568.

[24] P. F. Pinsky and J. Rinzel, "Intrinsic and network rhythmogenesis in a reduced Traub model for CA3 neurons.," *J. Comput. Neurosci.*, vol. 1, no. 1–2, pp. 39–60, Jun. 1994, Accessed: Aug. 12, 2016. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/8792224.

[25] D. Sterratt, B. Graham, A. Gillies, and D. Willshaw, *Principles of computational modelling in neuroscience*. 2011.

[26] R. FitzHugh, "Impulses and Physiological States in Theoretical Models of Nerve Membrane," *Biophys. J.*, vol. 1, no. 6, pp. 445–466, 1961, doi: 10.1016/S0006-3495(61)86902-6.

[27] P. J. Magistretti and B. R. Ransom, "Astrocytes," *Neuropsychopharmacol. Fifth Gener. Prog.*, no. 15, pp. 133–145, 2002.

[28] D. E. Postnov, R. N. Koreshkov, N. A. Brazhe, A. R. Brazhe, and O. V. Sosnovtseva, “Dynamical patterns of calcium signaling in a functional model of neuron-astrocyte networks,” *J. Biol. Phys.*, vol. 35, no. 4, pp. 425–445, 2009, doi: 10.1007/s10867-009-9156-x.

[29] D. E. Postnov, L. S. Ryazanova, and O. V. Sosnovtseva, “Functional modeling of neural–glial interaction,” *Biosystems*, vol. 89, no. 1–3, pp. 84–91, 2007, doi: 10.1016/j.biosystems.2006.04.012.

[30] V. Volman, E. Ben-Jacob, and H. Levine, “The astrocyte as a gatekeeper of synaptic information transfer.,” *Neural Comput.*, vol. 19, no. 2, pp. 303–326, 2007, doi: 10.1162/neco.2007.19.2.303.

[31] Y. X. Li and J. Rinzel, “Equations for InsP3 receptor-mediated [Ca2+]i oscillations derived from a detailed kinetic model: a Hodgkin-Huxley like formalism.,” *Journal of theoretical biology*, vol. 166, no. 4. pp. 461–473, 1994, doi: 10.1006/jtbi.1994.1041.

[32] S. Nadkarni and P. Jung, “Modeling synaptic transmission of the tripartite synapse.,” *Phys. Biol.*, vol. 4, no. 1, pp. 1–9, 2007, doi: 10.1088/1478-3975/4/1/001.

[33] K. Guimarães, D. Q. M. Madureira, and A. L. Madureira, “Interactions between astrocytes and the reward-attention circuit: A model for attention focusing in the presence of nicotine,” *Cogn. Syst. Res.*, vol. 50, pp. 15–28, 2018, doi: 10.1016/j.cogsys.2018.03.001.

[34] K. Guimarães, D. Q. M. Madureira, and A. L. Madureira, “The reward-attention circuit model: Nicotine’s influence on attentional focus and consequences on attention deficit hyperactivity disorder,” *Neurocomputing*, vol. 242, pp. 140–149, Jun. 2017, doi: 10.1016/J.NEUCOM.2017.02.072.

[35] T. Deemyad, J. Lüthi, and N. Spruston, “Astrocytes integrate and drive action potential firing in inhibitory subnetworks.,” *Nat. Commun.*, vol. 9, no. 1, p. 4336, 2018, doi: 10.1038/s41467-018-06338-3.

[36] M. E. J. Sheffield, G. B. Edgerton, R. J. Heuermann, T. Deemyad, B. D. Mensh, and N. Spruston, “Mechanisms of retroaxonal barrage firing in hippocampal interneurons,” *J. Physiol.*, vol. 591, no. 19, pp. 4793–4805, Oct. 2013, doi: 10.1113/jphysiol.2013.258418.

[37] M. E. J. Sheffield, T. K. Best, B. D. Mensh, W. L. Kath, and N. Spruston, “Slow integration leads to persistent action potential firing in distal axons of coupled interneurons,” *Nat. Neurosci.*, vol. 14, no. 2, pp. 200–207, Feb. 2011, doi: 10.1038/nn.2728.

[38] B. A. Abed, A. Ismail, and N. A. Aziz, “Real time astrocyte in spiking neural network,” *SAI Intell. Syst. Conf. 2015*, pp. 565–570, 2015, doi: 10.1109/IntelliSys.2015.7361196.

[39] M. Barros and S. Dey, “Feed-forward and Feedback Control in Astrocytes for Ca2+-based Molecular Communications Nanonetworks,” *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, vol. PP, no. c, p. 1, 2018, doi:

10.1109/TCBB.2018.2887222.

[40] A. B. Porto-Pazos *et al.*, “Artificial astrocytes improve neural network performance,” *PLoS One*, vol. 6, no. 4, pp. 1–8, 2011, doi: 10.1371/journal.pone.0019109.

[41] P. Mesejo, O. Ibáñez, E. Fernández-Blanco, F. Cedrón, A. Pazos, and A. B. Porto-Pazos, “Artificial Neuron–Glia Networks Learning Approach Based on Cooperative Coevolution,” *Int. J. Neural Syst.*, vol. 25, no. 04, p. 1550012, 2015, doi: 10.1142/S0129065715500124.

[42] T. Manninen, R. Havela, and M.-L. Linne, “Computational Models for Calcium-Mediated Astrocyte Functions,” *Front. Comput. Neurosci.*, vol. 12, no. April, 2018, doi: 10.3389/fncom.2018.00014.

[43] F. Oschmann, H. Berry, K. Obermayer, and K. Lenk, “From in silico astrocyte cell models to neuron-astrocyte network models: A review,” *Brain Res. Bull.*, vol. 136, pp. 76–84, Jan. 2018, doi: 10.1016/j.brainresbull.2017.01.027.

[44] “SPANNER Workshop,” 2018. .

[45] J. Liu *et al.*, “GABA Regulation of Burst Firing in Hippocampal Astrocyte Neural Circuit: A Biophysical Model,” *Front. Cell. Neurosci.*, vol. 13, no. July, 2019, doi: 10.3389/fncel.2019.00335.

[46] J. Liu *et al.*, “Exploring Self-Repair in a Coupled Spiking Astrocyte Neural Network,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 3, pp. 865–875, 2019, doi: 10.1109/TNNLS.2018.2854291.

[47] P. D. Roberts and C. C. Bell, “Spike timing dependent synaptic plasticity in biological systems,” *Biol. Cybern.*, vol. 87, no. 5–6, pp. 392–403, Dec. 2002, doi: 10.1007/s00422-002-0361-y.

[48] E. L. Bienenstock, L. N. Cooper, and P. W. Munro, “Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex.,” *J. Neurosci.*, vol. 2, no. 1, pp. 32–48, Jan. 1982, Accessed: Jun. 30, 2019. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/7054394.

[49] B. Flanagan, L. McDaid, J. Wade, K. F. Wong-Lin, and J. Harkin, “A computational study of astrocytic glutamate influence on post-synaptic neuronal excitability,” *PLoS Comput. Biol.*, vol. 14, no. 4, pp. 1–25, 2018, doi: 10.1371/journal.pcbi.1006040.

[50] M. Naeem, L. J. McDaid, J. Harkin, J. J. Wade, and J. Marsland, “On the Role of Astroglial Syncytia in Self-Repairing Spiking Neural Networks,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 26, no. 10, pp. 2370–2380, 2015, doi: 10.1109/TNNLS.2014.2382334.

[51] M. M. Halassa, T. Fellin, H. Takano, J.-H. Dong, and P. G. Haydon, “Synaptic islands defined by the territory of a single astrocyte.,” *J. Neurosci.*, vol. 27, no. 24, pp. 6473–7, Jun. 2007, doi: 10.1523/JNEUROSCI.1419-07.2007.

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

[53] S. Nazari, K. Faez, M. Amiri, and E. Karami, “A digital implementation of neuron-astrocyte interaction for neuromorphic applications,” *Neural Networks*, vol. 66, pp. 79–90, 2015, doi: 10.1016/j.neunet.2015.01.005.

[54] M. Hayati, M. Nouri, S. Haghiri, and D. Abbott, “A Digital Realization of Astrocyte and Neural Glial Interactions,” *IEEE Trans. Biomed. Circuits Syst.*, pp. 1–12, 2015, doi: 10.1109/TBCAS.2015.2450837.

[55] M. Heidrapur, A. Ahmadi, and M. Ahmadi, “Digital Implementation of a Biological-Plausible Model for Astrocyte Ca $$^{2+}$$ Oscillations,” Springer, Cham, 2019, pp. 857–868.

[56] S. Nazari, M. Amiri, K. Faez, and M. Amiri, “Multiplier-less digital implementation of neuron–astrocyte signalling on FPGA,” *Neurocomputing*, vol. 164, pp. 281–292, Sep. 2015, doi: 10.1016/J.NEUCOM.2015.02.041.

[57] S. Nazari, M. Amiri, K. Faez, and E. Karami, “A novel digital circuit for astrocyte-inspired stimulator to desynchronize two coupled oscillators,” in *2014 21st Iranian Conference on Biomedical Engineering, ICBME 2014*, 2015, no. Icbme, pp. 80–85, doi: 10.1109/ICBME.2014.7043898.

[58] H. Soleimani, M. Bavandpour, A. Ahmadi, and D. Abbott, “Digital implementation of a biological astrocyte model and its application,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 26, no. 1, pp. 127–139, 2015, doi: 10.1109/TNNLS.2014.2311839.

[59] M. Davies *et al.*, “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018, doi: 10.1109/MM.2018.112130359.

[60] G. Tang, I. E. Polykretis, V. A. Ivanov, A. Shah, and K. P. Michmizos, “Introducing Astrocytes on a Neuromorphic Processor: Synchronization, Local Plasticity and Edge of Chaos,” 2019, [Online]. Available: http://arxiv.org/abs/1907.01620.

[61] M. Ranjbar and M. Amiri, “An analog astrocyte–neuron interaction circuit for neuromorphic applications,” *J. Comput. Electron.*, vol. 14, no. 3, pp. 694–706, 2015, doi: 10.1007/s10825-015-0703-3.

[62] S. Barzegarjalali and A. C. Parker, “A neuromorphic circuit mimicking biological short-term memory,” in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Aug. 2016, pp. 1401–1404, doi: 10.1109/EMBC.2016.7590970.

[63] Y. Irizarry-Valle and A. C. Parker, “An Astrocyte Neuromorphic Circuit That Influences Neuronal Phase Synchrony,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 9, no. 2, pp. 175–187, Apr. 2015, doi: 10.1109/TBCAS.2015.2417580.

[64] J. Liu, J. Harkin, L. Mcdaid, D. M. Halliday, A. M. Tyrrell, and J. Timmis,

"Self-Repairing Mobile Robotic Car using Astrocyte-Neuron Networks," *Int. Jt. Conf. Neural Networks*, pp. 1–8, 2016.

[65] A. P. Johnson *et al.*, "Fault-Tolerant Learning in Spiking Astrocyte-Neural Networks on FPGAs," in *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)*, Jan. 2018, pp. 49–54, doi: 10.1109/VLSID.2018.36.

[66] A. P. Johnson *et al.*, "Time-multiplexed System-on-Chip using Fault-tolerant Astrocyte-Neuron Networks," *Proc. 2018 IEEE Symp. Ser. Comput. Intell. SSCI 2018*, pp. 1076–1083, 2019, doi: 10.1109/SSCI.2018.8628710.

[67] A. P. Johnson *et al.*, "An FPGA-based hardware-efficient fault-tolerant astrocyte-neuron network," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec. 2016, pp. 1–8, doi: 10.1109/SSCI.2016.7850175.

[68] A. P. Johnson *et al.*, "Homeostatic Fault Tolerance in Spiking Neural Networks: A Dynamic Hardware Perspective," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 65, no. 2, pp. 687–699, Feb. 2018, doi: 10.1109/TCSI.2017.2726763.

[69] A. P. Johnson *et al.*, "Homeostatic fault tolerance in spiking neural networks utilizing dynamic partial reconfiguration of FPGAs," in *2017 International Conference on Field Programmable Technology (ICFPT)*, Dec. 2017, pp. 195–198, doi: 10.1109/FPT.2017.8280139.

[70] A. P. Johnson *et al.*, "Time-multiplexed System-on-Chip using Fault-tolerant Astrocyte-Neuron Networks," in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, Nov. 2018, pp. 1076–1083, doi: 10.1109/SSCI.2018.8628710.

[71] S. Carrillo *et al.*, "Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 12, pp. 2451–2461, 2013, doi: 10.1109/TPDS.2012.289.

[72] J. Liu, J. Harkin, L. Mcdaid, and G. Martin, "Hierarchical Networks-on-Chip Interconnect for Astrocyte-Neuron Network Hardware."

[73] J. Liu, J. Harkin, L. P. Maguire, L. J. McDaid, J. J. Wade, and G. Martin, "Scalable Networks-on-Chip Interconnected Architecture for Astrocyte-Neuron Networks," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 63, no. 12, pp. 2290–2303, Dec. 2016, doi: 10.1109/TCSI.2016.2615051.

[74] G. Martin, J. Harkin, L. J. McDaid, J. J. Wade, J. Liu, and F. Morgan, "Astrocyte to spiking neuron communication using Networks-on-Chip ring topology," *2016 IEEE Symp. Ser. Comput. Intell. SSCI 2016*, pp. 1–8, 2017, doi: 10.1109/SSCI.2016.7850172.

[75] G. Martin, J. Harkin, L. J. McDaid, J. J. Wade, and J. Liu, "On-chip communication for neuro-glia networks," *IET Comput. Digit. Tech.*, vol. 12, no. 4, pp. 130–138, 2018, doi: 10.1049/iet-cdt.2017.0187.

[76] A. P. Davison, “PyNN: a common interface for neuronal network simulators,” *Front. Neuroinform.*, vol. 2, 2008, doi: 10.3389/neuro.11.011.2008.

[77] I. Bogdanov, R. Mirsu, and V. Tiponut, “MATLAB model for spiking neural networks,” *Proc. 13th WSEAS Int. Conf. Syst.*, no. July 2009, pp. 533–537, 2009, [Online]. Available: http://www.researchgate.net/publication/228755091_MATLAB_model_for_spiking_neural_networks/file/60b7d52848bc2c9f14.pdf.

[78] D. F. M. Goodman and R. Brette, “The Brian simulator,” *Front. Neurosci.*, vol. 3, no. 2, pp. 192–197, Sep. 2009, doi: 10.3389/neuro.01.026.2009.

[79] P. Richmond, A. Cope, K. Gurney, and D. J. Allerton, “From Model Specification to Simulation of Biologically Constrained Networks of Spiking Neurons,” *Neuroinformatics*, vol. 12, no. 2, pp. 307–323, Apr. 2014, doi: 10.1007/s12021-013-9208-z.

[80] R. Mirsu and V. Tiponut, “Parallel model for Spiking Neural Networks using MATLAB,” in *2010 9th International Symposium on Electronics and Telecommunications*, Nov. 2010, pp. 369–372, doi: 10.1109/ISETC.2010.5679345.

[81] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. V. Veidenbaum, “A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors,” *Neural Networks*, vol. 22, no. 5–6, pp. 791–800, Jul. 2009, doi: 10.1016/J.NEUNET.2009.06.028.

[82] A. K. Fidjeland and M. P. Shanahan, “Accelerated simulation of spiking neural networks using GPUs,” *Int. Jt. Conf. Neural Networks*, pp. 1–8, 2010, doi: 10.1109/IJCNN.2010.5596678.

[83] B. Kasap and A. J. van Opstal, “Dynamic parallelism for synaptic updating in GPU-accelerated spiking neural network simulations,” *Neurocomputing*, vol. 302, pp. 55–65, 2018, doi: 10.1016/j.neucom.2018.04.007.

[84] R. Brette and D. F. M. Goodman, “Simulating spiking neural networks on GPU,” *Netw. Comput. Neural Syst.*, vol. 23, no. 4, pp. 167–182, Dec. 2012, doi: 10.3109/0954898X.2012.730170.

[85] D. Yudanov, M. Shaaban, R. Melton, and L. Reznik, “GPUBased Simulation of Spiking Neural Networks with RealTime Performance and Accuracy,” pp. 18–23, 2010.

[86] P. Krishnamani and V. Venkittaraman, “Acceleration of spiking neural networks on single-GPU and multi-GPU systems,” no. May, p. 81, 2010, [Online]. Available: http://gradworks.umi.com/14/75/1475559.html.

[87] K. Cheung, S. R. Schultz, and W. Luk, “NeuroFlow: A general purpose spiking neural network simulation platform using customizable processors,” *Front. Neurosci.*, vol. 9, no. JAN, pp. 1–15, 2016, doi: 10.3389/fnins.2015.00516.

[88] F. Morgan *et al.*, “Exploring the evolution of NoC-based spiking neural networks on FPGAs,” *Proc. 2009 Int. Conf. Field-Programmable Technol. FPT’09*, pp. 300–303, 2009, doi: 10.1109/FPT.2009.5377663.

[89] S. Carrillo *et al.*, “Advancing interconnect density for spiking neural network hardware implementations using traffic-aware adaptive network-on-chip routers,” *Neural Networks*, vol. 33, pp. 42–57, 2012, doi: 10.1016/j.neunet.2012.04.004.

[90] K. Cheung, S. R. Schultz, and W. Luk, “A LARGE-SCALE SPIKING NEURAL NETWORK ACCELERATOR FOR FPGA SYSTEMS,” pp. 2–5.

[91] K. Cheung, S. R. Schultz, and P. H. W. Leong, “A Parallel Spiking Neural Network Simulator,” pp. 247–254, 2009.

[92] D. B. Thomas and W. Luk, “FPGA accelerated simulation of biologically plausible spiking neural networks,” in *Proceedings - IEEE Symposium on Field Programmable Custom Computing Machines, FCCM 2009*, 2009, pp. 45–52, doi: 10.1109/FCCM.2009.46.

[93] E. M. Izhikevich, “Simple Model of Spiking Neurons,” *IEEE Trans. NEURAL NETWORKS*, vol. 14, no. 6, 2003, doi: 10.1109/TNN.2003.820440.

[94] D. Pani, P. Meloni, G. Tuveri, F. Palumbo, P. Massobrio, and L. Raffo, “An FPGA platform for real-time simulation of spiking neuronal networks,” *Front. Neurosci.*, vol. 11, no. February, 2017, doi: 10.3389/fnins.2017.00090.

[95] R. M. Wang, T. J. Hamilton, J. C. Tapson, and A. van Schaik, “A mixed-signal implementation of a polychronous spiking neural network with delay adaptation,” *Front. Neurosci.*, vol. 7, no. 8 MAR, pp. 1–14, 2014, doi: 10.3389/fnins.2014.00051.

[96] E. M. Izhikevich, “Polychronization: Computation with Spikes,” *Neural Comput.*, vol. 18, no. 2, pp. 245–282, Feb. 2006, doi: 10.1162/089976606775093882.

[97] S. W. Moore, P. J. Fox, S. J. T. Marsh, A. T. Markettos, and A. Mujumdar, “Bluehive - A field-programable custom computing machine for extreme-scale real-time neural network simulation,” in *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, FCCM 2012*, 2012, pp. 133–140, doi: 10.1109/FCCM.2012.32.

[98] A. Sripad *et al.*, “SNAVA—A real-time multi-FPGA multi-model spiking neural network simulation architecture,” *Neural Networks*, vol. 97, pp. 28–45, 2018, doi: 10.1016/j.neunet.2017.09.011.

[99] K. Abdelouahab *et al.*, “Tactics to Directly Map CNN graphs on Embedded FPGAs To cite this version : HAL Id : hal-01626462,” 2017.

[100] T. Geng *et al.*, “FPDeep: Acceleration and Load Balancing of CNN Training on FPGA Clusters,” *Proc. - 26th IEEE Int. Symp. Field-*

Programmable Cust. Comput. Mach. FCCM 2018, pp. 81–84, 2018, doi: 10.1109/FCCM.2018.00021.

[101] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, “Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster,” pp. 326–331, 2016, doi: 10.1145/2934583.2934644.

[102] E. Chung *et al.*, “Serving DNNs in Real Time at Datacenter Scale with Project Brainwave,” *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018, doi: 10.1109/MM.2018.022071131.

[103] S. I. Venieris and C. S. Bouganis, “F-CNNx: A toolflow for mapping multiple convolutional neural networks on FPGAs,” *Proc. - 2018 Int. Conf. Field-Programmable Log. Appl. FPL 2018*, pp. 381–388, 2018, doi: 10.1109/FPL.2018.00072.

[104] A. Lines *et al.*, “Loihi Asynchronous Neuromorphic Research Chip,” in *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, May 2018, pp. 32–33, doi: 10.1109/ASYNC.2018.00018.

[105] N. Qiao *et al.*, “A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses,” *Front. Neurosci.*, vol. 9, no. APR, pp. 1–17, 2015, doi: 10.3389/fnins.2015.00141.

[106] J. S. Seo and M. Seok, “Digital CMOS neuromorphic processor design featuring unsupervised online learning,” *IEEE/IFIP Int. Conf. VLSI Syst. VLSI-SoC*, vol. 2015-Octob, pp. 49–51, 2015, doi: 10.1109/VLSI-SoC.2015.7314390.

[107] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, “A wafer-scale neuromorphic hardware system for large-scale neural modeling,” *ISCAS 2010 - 2010 IEEE Int. Symp. Circuits Syst. Nano-Bio Circuit Fabr. Syst.*, pp. 1947–1950, 2010, doi: 10.1109/ISCAS.2010.5536970.

[108] J. Navaridas, M. Luján, J. Miguel-Alonso, L. A. Plana, and S. Furber, “Understanding the interconnection network of SpiNNaker,” p. 286, 2009, doi: 10.1145/1542275.1542317.

[109] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, “The SpiNNaker project,” *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, 2014, doi: 10.1109/JPROC.2014.2304638.

[110] M. Khan, X. Jin, S. B. Furber, and L. a Plana, “System-level model for a gals massively parallel multiprocessor,” *Proc. 19th UK Asynchronous Forum*, vol. 24, no. 5, pp. 454–463, 2007.

[111] A. A. Khedkar and R. H. Khade, “High speed FPGA-based data acquisition system,” *Microprocess. Microsyst.*, vol. 49, pp. 87–94, 2017, doi: 10.1016/j.micpro.2016.11.006.

[112] E. Fysikopoulos, G. Loudos, M. Georgiou, S. David, and G. Matsopoulos,

"A Spartan 6 FPGA-based data acquisition system for dedicated imagers in nuclear medicine," *Meas. Sci. Technol.*, vol. 23, no. 12, p. 125403, 2012, doi: 10.1088/0957-0233/23/12/125403.

[113] I. Humphreys, G. Eisenblätter, and G. E. O'Donnell, "FPGA based monitoring platform for condition monitoring in cylindrical grinding," *Procedia CIRP*, vol. 14, pp. 448–453, 2014, doi: 10.1016/j.procir.2014.03.022.

[114] M. J. (Michael J. Quinn and M. J. (Michael J. Quinn, *Parallel computing : theory and practice*. McGraw-Hill, 1994.

[115] L. Benini and G. De Micheli, *Networks on chips : technology and tools*. Elsevier Morgan Kaufmann Publishers, 2006.

[116] F. T. Leighton, *Introduction to parallel algorithms and architectures : arrays, trees, hypercubes*. M. Kaufmann Publishers, 1992.

[117] V. Rantala, V. Rantala, T. Lehtonen, and J. Plosila, "Network on Chip Routing Algorithms," 2006, Accessed: Sep. 07, 2019. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.120.8910.

[118] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. C–34, no. 10, pp. 892–901, Oct. 1985, doi: 10.1109/TC.1985.6312192.

[119] P. K. Sahu, N. Shah, K. Manna, and S. Chattopadhyay, "An Application Mapping Technique for Butterfly-Fat-Tree Network-on-Chip," in *2011 Second International Conference on Emerging Applications of Information Technology*, Feb. 2011, pp. 383–386, doi: 10.1109/EAIT.2011.37.

[120] A. Bouhraoua, O. Diraneyya, and M. E. Elrabaa, "A simplified router architecture for the modified Fat Tree Network-on-Chip topology," in *2009 NORCHIP*, Nov. 2009, pp. 1–4, doi: 10.1109/NORCHP.2009.5397806.

[121] M. Dehyadgari, M. Nickray, A. Afzali-kusha, and Z. Navabi, "Evaluation of Pseudo Adaptive XY Routing Using an Object Oriented Model for NOC," in *2005 International Conference on Microelectronics*, pp. 204–208, doi: 10.1109/ICM.2005.1590068.

[122] S. W. Moore, P. J. Fox, S. J. T. Marsh, A. T. Markettos, and A. Mujumdar, "Bluehive - A field-programable custom computing machine for extreme-scale real-time neural network simulation," *Proc. 2012 IEEE 20th Int. Symp. Field-Programmable Cust. Comput. Mach. FCCM 2012*, pp. 133–140, 2012, doi: 10.1109/FCCM.2012.32.

[123] J. J. Wade, L. J. McDaid, J. Harkin, V. Crunelli, and J. A. S. Kelso, "Bidirectional coupling between astrocytes and neurons mediates learning and dynamic coordination in the brain: A multiple modeling approach," *PLoS One*, vol. 6, no. 12, pp. 1–24, 2011, doi: 10.1371/journal.pone.0029445.

[124] L. McDaid, J. Harkin, S. Hall, and T. Dowrick, "EMBRACE: emulating

biologically-inspired architectures on hardware," *NN'08 Proc. …*, 2008.

[125] V. Lahtinen, E. Salminen, K. Kuusilinna, and T. D. Hämäläinen, "Bus Structures in Network-on-Chips," in *Interconnect-Centric Design for Advanced SoC and NoC*, Boston: Kluwer Academic Publishers, pp. 207–230.

[126] J. Nurmi, *Interconnect-centric design for advanced SoC and NoC*. Kluwer Academic Publishers, 2004.

[127] Intel Corporation, "Transceiver Architecture in Stratix IV Devices," *Stratix IV Device Handbook*, vol. 2, no. September. pp. 1–228, 2012, [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.188.3802&rep=rep1&type=pdf.

[128] Intel Corporation, "FIFO Intel FPGA IP User Guide," 2018. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_fifo.pdf.

[129] Intel, "Transceiver Architecture in Stratix IV Devices," *Stratix IV Device Handbook*, vol. 2, no. September. pp. 1–228, 2012.

[130] Intel Corporation, "Transceiver Clocking in Stratix IV," vol. 2, no. September. 2012.

[131] A. X. Widmer and P. A. Franaszek, "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code," *IBM J. Res. Dev.*, vol. 27, no. 5, pp. 440–451, Sep. 1983, doi: 10.1147/rd.275.0440.

[132] I. Corporation, "8B / 10B Code," no. May. pp. 1–12, 2007.

[133] Intel Corporation, "Using High-Speed Transceiver Blocks in Stratix GX Devices Transceiver I / O Banks," no. November, pp. 1–58, 2002.

[134] C. Methods, "SCFIFO and DCFIFO IP Cores User Guide," 2015.

Appendix

This appendix is dedicated to describing the functionality and architecture of two Intel IPs, namely Gigabit Transceiver Block (GXB) and Dual Clock First in First out (DCFIFO) memory. The reason for focussing on these two IPs is they have been used extensively in the design of the multi-FPGA NoC router described in Chapter 4. Additionally, the controllers that are designed during this work to synchronous between the GXB blocks will be discussed.

Intel transceiver operation and architecture

Intel Gigabit Transceiver Block (GXB) is a highly reconfigurable IP for utilizing multi Gigabit serial communications between Intel FPGAs and external devices [127].

Figure 1 presents an overall overview of the transceiver architecture used in this project. The components can be different depending on the mode and protocol of the transceiver.

Basic mode and a custom synchronization protocol have been used in AstroByte. In *Basic* mode, the transceiver is extremely flexible in terms of the features and blocks that can be included or omitted.

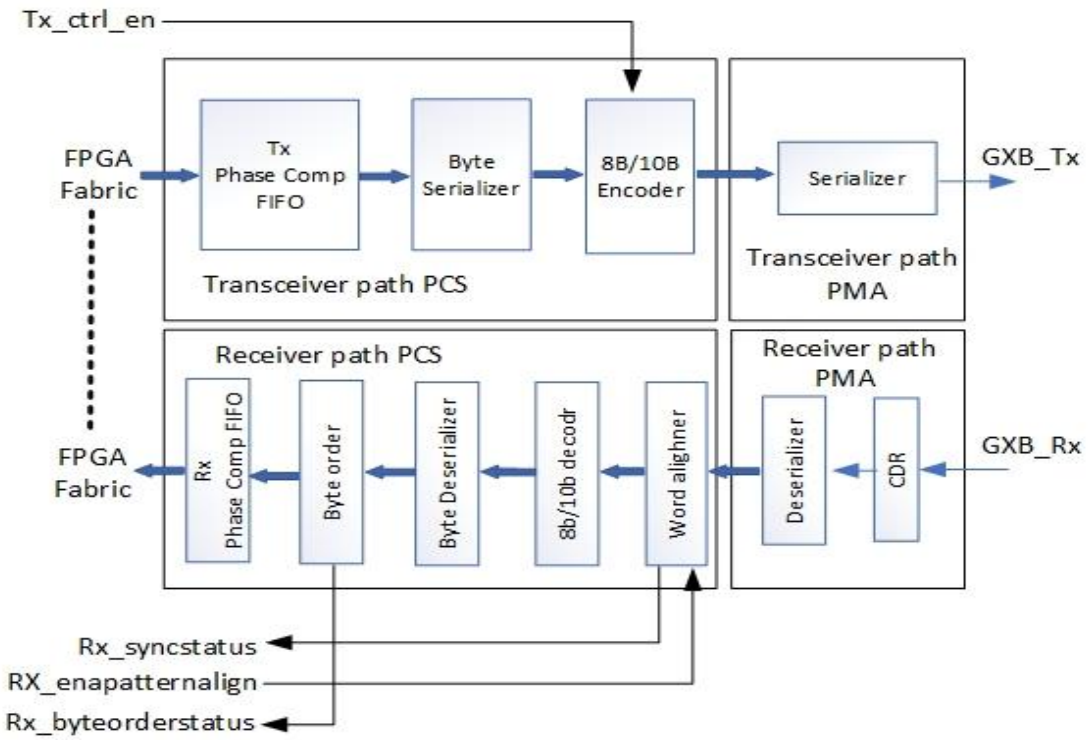


Figure 1 Intel Gigabit Transceiver architecture [131]

The transmitter has two separate paths, one used for transmission (GXB_Tx) and the other for receiving data from the upstream node.

➢ **Transmitter data path**

The transmitter data path, shown in Figure 1, is responsible accepting FPGA fabric user data and serializing it before transmitting it off-FPGA to the downstream node.

Intel GXB transceiver data path is composed of the following blocks.

- **Tx phase compensation First In first Out (FIFO) memory**

It buffers FPGA data stream at the transmitter input as well as compensating for the phase difference between an external clock and the transmitters recovered clock [127]. In AstroByte implementation no external clock is provided and both sides of the block are clocked by the transmitter internal clock. The FIFO is 8 words deep, introducing 4-5 cycle latency [130] .

- **Byte serializer**

The byte serializer converts the 32-bit (4 bytes) data bus from the Tx phase compensation FIFO to 16-bit (2 bytes). The purpose is to avoid exceeding the maximum allowed frequency at the transceiver-FPGA fabric interface meanwhile operating the transceiver at high data rates [127]. The Byte serializer input frequency is the same as that of Tx compensation FIFO frequency while its output operates on 2x of the value of input frequency [130].

- **B/10B Encoder**

The 8B/10B encoder converts each of the 2-byte wide data out of the Byte serializer block to a 10-bits wide character code according to 8B/10B transmission coding paradigm [127]. The purpose is to introduce sufficient 0-1 value change before transmitting a byte to enable the receiver to recover the clock embedded in the serial data stream. Additionally, DC balancing can be achieved using this encoding scheme [131]. Since the output of the Byte serializer is 16 bits, two such encoders are required to provide a 20-bit output. The 8B/10B encoder is supplied with the same clock as the Byte serializer output [130].

- **Serializer**

The Serializer is the last block on the transmitter path. It converts the 20-bit parallel data input to 1-bit serial data (20 serialization factor) operating at 6Gbps data rate in the AstroByte implementation. The parallel input is clocked through the Byte serializ9er clock frequency and the serial output is clocked by a multiplied version of its input clock [127]. The output of the serializer is forwarded to FPGA transceiver pins.

➢ **Receiver data path**

The receiver supervises converting the incoming serial data to a 32-bit word before forwarding to the FPGA fabric. Similar to the transmitter data path, the receiver data path contains several blocks as illustrated in Figure 1.

- **Clock Data Recovery (CDR)**

The CDR main task is recovering the clock of the serial data stream coming from the upstream transmitter node. The recovered clock (fast) along with its slower derivative will be used by CDR to clock the next blocks on the receiver data path [127].

- **Deserializer**

Deserializer transfers the serial data stream to a 20-bit wide word using the fast and slow recovered clocks from CDR [127]. The serial input will be clocked by the fast clock while the parallel 20-bit output will be synchronised to the slow clock [130].

- **Word aligner**

As it has been established, data is transmitted serially by the upstream transmitter and then parallelized by the Deserializer block inside the downstream receiver. In this process, the original transmitted word boundary becomes lost. It is the Word aligner's duty to recover the correct boundary of the data stream [127].

- **8B/10B decoder**

Each byte is converted to a 10-bit wide word in accordance with 8B/10B encoding scheme at the transmitter. The 8B/10B decoder, reverses this process to extract the original data [127]. Hence, two decoders is convert the 20-bit wide input data to 16-bit output.

- **Byte deserializer**

As it is the case with transmitter data path, the receiver deserializes the Byte numbers to avoid violation of the GXB – FPGA interface clock rating [127]. The output stream is synchronized to a divided-by-2 version of its input clock [130]. The input is 16-bits wide while the output is a 32-bit wide bus.

- **Byte ordering**

There is a possibility that the order of the bytes gets distorted at the Byte deserializer output. The Byte ordering block ensures that the correct order of bytes are restored before forwarding the data to the next block and then to the FPGA fabric [127].

- **Rx phase compensation FIFO**

This is the last block before forwarding the retrieved 32-bit data to the FPGA fabric. The Rx phase compensation FIFO is used for buffering of data and

compensating for phase differences between its read and write clocks. The depth of the FIFO is 8 words and it has a latency of 4-5 clock cycles [127].

- **Transceiver synchronization**

Figure 2 illustrates two GBX blocks communicating with each other. Each transceiver is located on a separate FPGA. Both are controlled by a GXB controller, implemented in custom FPGA hardware, which assures synchronization with the other GXB for integral data transfer between the two FPGA devices. The GXB controllers are designed during the course of this project while the GBX blocks are Intel IPs. More specifically, the GXB controller blocks control the GXB transceivers by means of a set of control and status signals. The functionality of these signals will be given next.

- **Tx_ctrl_en**

Tx_ctrl_en is a control signal issued by the controller blocks to the 8b/10b encoders on the transmitter paths as per Figure 2. When this signal is high the decoder deals with the 8-bit data as a control byte. If it is low the encoder deals with the 8-bit data as a data byte. Sending a particular control byte (K28.5 control code) orders the downstream receiver to look for an alignment pattern [132], [133]. This is performed by sending the hexadecimal value BC1C and asserting *Tx_ctrl_en* in the same clock cycle by the GXB Controller blocks.

- **Rx_enapatternalign**

Another control signal that is supplied by the GXB control blocks. *Rx_enapatternalign* controls whether the Word aligner block on the receiver path should look for the alignment pattern embedded in the data stream. The pattern has to be programmed into the Word aligner block through the GXB interface in Intel Quartus software.

- **Rx_syncstatus**

The Word aligner outputs a status signal by block on the receiver path, indicating the successful retrieval of the word boundary. This signal will show the status of the receiver channel to the GXB controller.

- **Rx_byteorderstatus**

The Byte ordering block is the source of this signal that, alongside Rx*_syncstatus,* will be used by the controllers to be informed about the status of the receiver path.

Similar to the Word alignment block, the Byte ordering block depends on pattern recognition to recover the order of the bytes that form the recovered data stream. This is due to the fact that the byte order can be distorted by the Byte deserializer block. The pattern can be defined by user and be sent using the GXB controllers to be retrieved by the receiving end of the transmission. In this case, the exact same value that is sent from the transmitter has to be

programmed into the receiver as the Byte order block comes after 8B/10B decoder.

- **GXB Controller**

The controller issues control signals and a 32-bit wide data bus to its associated transceiver while reading status signals and a 32-bit data bus from the receiver. It is function is synchronizing with the transceiver at the other end of the link, also controlled by a similar controller. A detailed algorithm representing the working of the GXB controller blocks is demonstrated in Figure 3.

The controller is partitioned to three state machines operating in conjunction. This approach was taken as it became clear that implementing the entire controller using one state machine will be much more challenging and prone to faults.

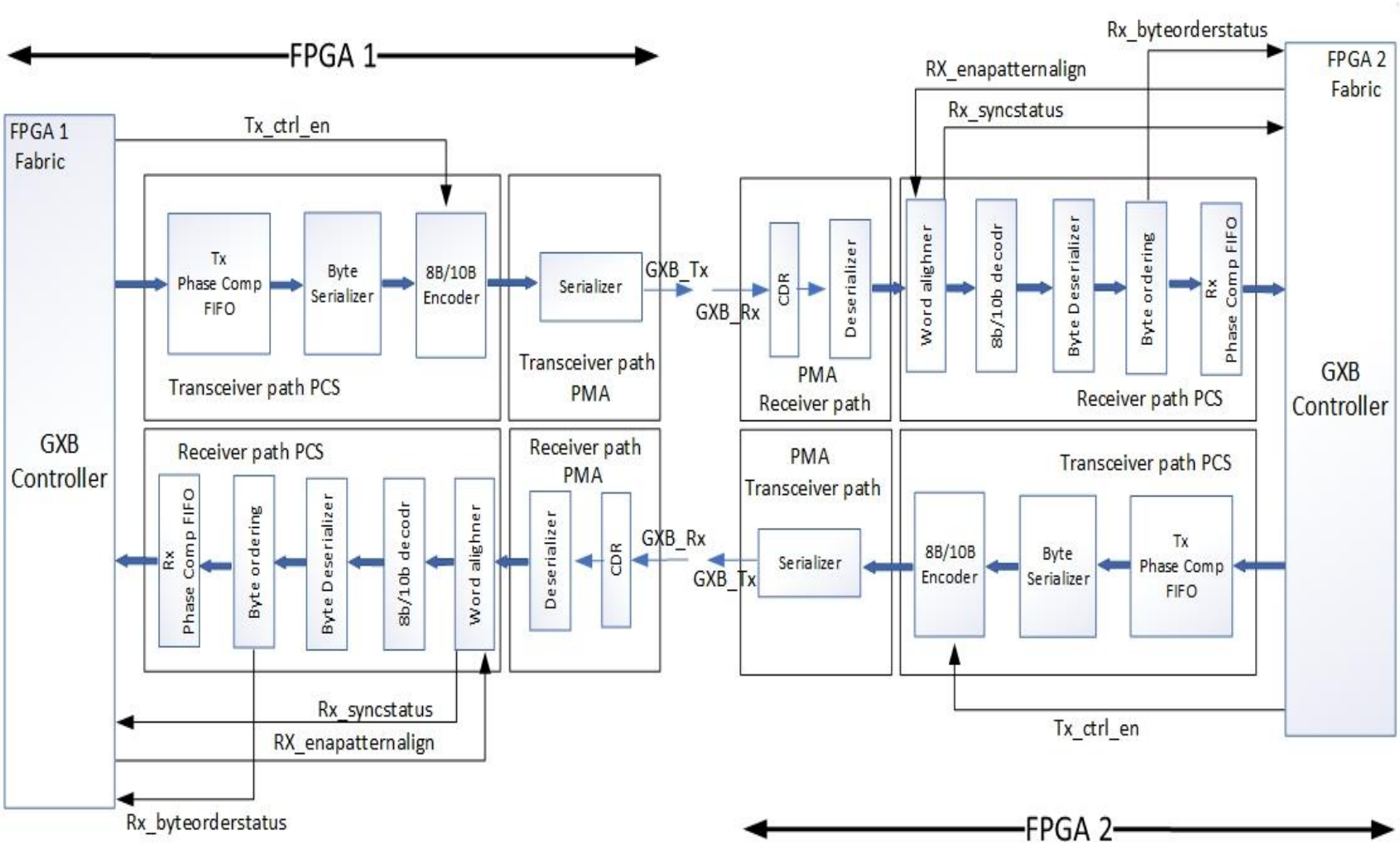


Figure 2 Synchronization between two transceivers

The output of the first state machine is *Self_sync*. This signal only goes high when Word align block has recovered the word boundary (Rx_*syncstatus* is at logic '1') and the Byte ordering block has retrieved the original byte order (*Rx_byteorderstatus is at logic '1').* The *Self_sync* is an indication that the receiver path belonging to the transceiver at this end has been synchronized with the one at the other end.

The second state machine provides *Other_sync* signal, putting it at logic high when a particular user-defined pattern –synchronization pattern-was received from the other (hence the name) transceiver. Unlike other patterns used before, namely the word align pattern and the byte order pattern, this pattern is not meant to be used by the GXB block. It is for exchanging status information between the two GXB controllers at each end of the link.

The third state machine accepts *Other_sync* and *Self_sync* signals and provides the custom FPGA fabric hardware (NoC router in AstroByte) *GXB_synced* control signal.

GXB_synced will be asserted only when the third state machine has sensed logic '1' on both *Other_sync* and *Self_sync,* confirming that both sides of the link have been synchronized*.*

The third state machine starts by constantly sending the alignment pattern and the byte order pattern to the other transceiver. Meanwhile, it examines the *Self_sync* control signal to check if its receiver has recovered the word boundary and byte order, meaning that the data on the receiver channel is the original data sent from the GXB on the other end.

Once logic '1' is sensed on *Self_sync* control line, the third state machine checks for the synchronization pattern from the other end GXB block while sending the same pattern. At this stage the third state machine "assumes" or "hopes" that the other node is at the same stage. If its assumption was right, proved by receiving the synchronization pattern from the other controller, it means the link has successfully been established and the state machine raises its GXB_synced signal high, transferring the control of the link to the NoC router. The NoC router only comes out of its reset state after this signal is received, ensuring that a valid link established before starting to operate.

As evident from Figure 3, if the synchronization pattern is not received, showing that the end point GXB has not recovered word boundary, byte order, or both, the state machine loops back to the first state to send the alignment and byte order patterns. It is important to point out that the two transceivers and their relevant controller blocks operate separately from each other in separate clock domains on different FPGAs.

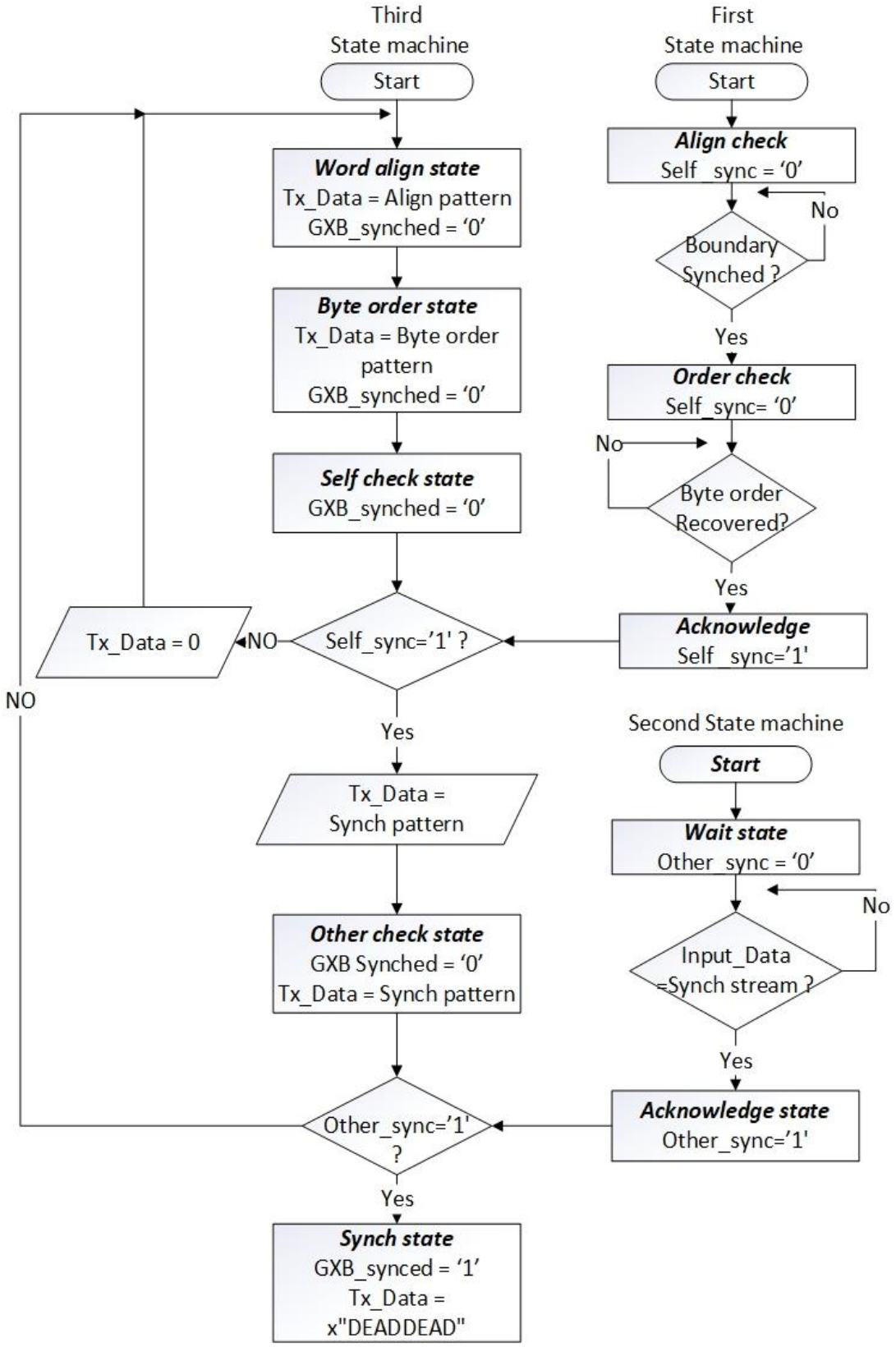


Figure 3 Custom GXB controller block

- **Transceiver overall latency**

The latency introduced by using GBX blocks to transmitter packets from one FPGA to another is between 10.5 cycles – 13 cycles. The transmitter part of the GBX blocks introduces a latency of 4-5.5 cycles while the receiver causes a delay of between 6.5-8.5 cycles [128],[129]. Here "cycles" mean the FPGA fabric's frequency cycles as opposed to the frequencies operating internal to the transceivers. Experimental tests were constantly recording a delay of 13 cycles. That is the reason behind choosing the value of 13 when accounting for cross-FPGA latencies in Chapter 5.

Dual Clock FIFO (DCFIFO) memories

Beside transceivers, explored in previous section, DCFIFOs are another vital IP for realizing coherent multi-FPGA communications, allowing for avoiding a phenomenon called *metastability* [128].

Depicted in Figure 4, DCFIFOs are used for moving data from a clock domain A to clock domain B to avoid *metastability* [128]. The input to the DCFIFO is called write side while the output is read side. Controllers are necessary to govern read and write operation on either side of the FIFO.

The write controller and write side of the FIFO use the same clock (A in the figure), while the read controller shares the same clock with the read side of the FIFO (B in the figure).

A number of status signals are issued by the DCFIFO that can be used by the two controllers before performing read/write operations. From Figure 4, *Wr_full* and *Wr_empty* indicate whether the write side is full or empty, respectively. *Wr_used* indicates the number of words sored in the write side of the FIFO at a particular clock cycle. It's a bus which has width of $\log_2$ (D), where D is the depth of the FIFO – the maximum number of words that can be accommodated by the DCFIFO. The same set of status signals are provided by the read side.

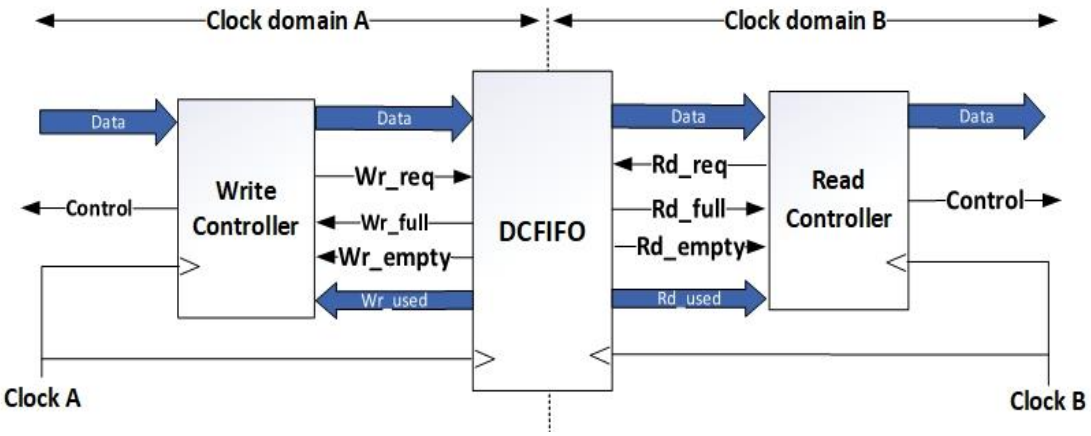


Figure 4 DCFIFO with associated controllers

The read and write status signals are synchronized to their respective clock domains.

Data streams moving through a FIFO experience some latency. Additionally, the status signals give the status of the DCFIFO after some clock cycles. Both of these latencies should be given serious consideration while using the FIFO. The most import latencies are given in the following table [128].

In Table 1 *n* denotes the synchronization stages that exist between the write and the read side. It is the number of registers that data has to go through when crossing the clock boundary. Synchronisation registers are crucial for ensuring data integrity but also introduce latency [134].

Table 1: Latency experienced by Intel DCFIFO IP status signals

Wr_req to Wr_full: 1 Wr_clk
Wr_req to Rd_full: 2 Wr_clk cycles + n Rd_clk
Wr_req to Wr_empty: 1 Wr_clk
Wr_req to Rd_empty: 2 Wr_clk + n Rd_clk
Wr_req to Wr_usedw: 2 Wr_clk
Wr_req to Rd_usedw: 2 Wr_clk + (n + 1) Rd_clk
Wr_req to Output: 1 Wr_clk + 1 Rd_clk
Rd_req to Rd_empty: 1 Rd_clk
Rd_req to Wr_empty: 1 Rd_clk + n Wr_clk
Rd_req to Rd_full: 1 Rd_clk
Rd_req to Wr_full: 1 Rd_clk + n Wr_clk
Rd_req to Rd_usedw: 2 Rd_clk
Rd_req to Wr_usedw: 1 Rd_clk + (n + 1) Wr_clk
Rd_req to Output: 1 Rd_clk

Controllers for DCFIFOs

While using Intel DCFIFO IP for FPGAs, the write controller must assure that no write operation is carried out on a full FIFO. Likewise, the read controller should not read from an empty FIFO[128].

Performing write while the DCFIFO is full does not change its status at best and can force it to an undefined state, hampering the operation of the entire circuit. The same is true for reading from an empty FIFO. Correspondingly, reading from an empty FIFO should be avoided as it will read a previously read signal at best or introduces unexpected behaviour, distorting the functionality of the circuit.