Digital Neural Circuits: From Ions to Networks

by

Junwen Luo

A thesis submitted in partial fulfilment of the degree of Doctor of Philosophy

in the

Faculty of Science, Agriculture and Engineering

School of Electrical and Electronic Engineering

November 2014

Abstract

The biological neural computational mechanism is always fascinating to human beings since it shows several state-of-the-art characteristics: strong fault tolerance, high power efficiency and self-learning capability. These behaviours lead the developing trend of designing the next-generation digital computation platform. Thus investigating and understanding how the neurons talk with each other is the key to replicating these calculation features. In this work I emphasize using tailor-designed digital circuits for exactly implementing bio-realistic neural network behaviours, which can be considered a novel approach to cognitive neural computation. The first advance is that biological real-time computing performances allow the presented circuits to be readily adapted for real-time closed-loop *in vitro* or *in vivo* experiments, and the second one is a transistor-based circuit that can be directly translated into an impalpable chip for high-level neurologic disorder rehabilitations. In terms of the methodology, first I focus on designing a heterogeneous or multiple-layer-based architecture for reproducing the finest neuron activities both in voltage-and calcium-dependent ion channels. In particular, a digital optoelectronic neuron is developed as a case study. Second, I focus on designing a network-on-chip architecture for implementing a very large-scale neural network (*e.g.* more than 100,000) with human cognitive functions (*e.g.* timing control mechanism). Finally, I present a reliable hybrid bio-silicon closed-loop system for central pattern generator prosthetics, which can be considered as a framework for digital neural circuit-based neuro-prosthesis implications. At the end, I present the general digital neural circuit design principles and the long-term social impacts of the presented work.

Acknowledgements

I sincerely thank my supervisors Patrick Degenaar and Alex Yakovlev. They have consistently supported me with my work over the last four years. Their suggestions have significantly enhanced the quality of my work and broadened my global research visions.

I will forever be thankful to my former supervisor Terrence Mak. He guided me to build a strong basis in the early stages and taught me how to be an excellent researcher.

Also, I genuinely thank my co-supervisors and research advisors Peter Andras, Chung Tin, Chi-Sang Poon and Wolfgang Stein. They gave me valuable opportunities to study and work in various places across the world such as the University of Ulm, City University of Hong Kong and Massachusetts Institute of Technology. I learned about different cultures and gained further essential knowledge from these experiences.

I am very lucky that I worked with a good support team. In particular, my collaborators Graeme Coapes, Hock Soon Law and Jannetta Stein were always helpful when I was in trouble. I greatly appreciate their assistance.

Last but not least, I dedicate this thesis to my family for their unerring support and unconditional love.

Contents

List of figures

List of tables

Acronyms

FPGA	Field-Programmable Gated Array
HH	Hodgkin-Huxley
ChR2	Channelrhodopsin2
NoC	Network-on-Chip
POT	Passenger-of-Time
STG	Stomatogastric ganglion
CPG	Central Pattern Generator
CPU	Central Processing Unit
GPU	Graphic Processing Unit
ASIC	Application-Specific Integrated Circuit
AER	Address event representation
LUT	Look-up Table
LIF	Leaky Integrate-and-Fire
STDP	Spike Time-Dependent Plasticity
LTP	Long Term Potential
LTD	Long Term Depression

Chapter 1 Overview and Rationale

This chapter generally describes the definition, history, development trends and current bottlenecks of the neuromorphic circuit. Then it gives a brief description of the contributions of the presented work and organization of the thesis.

1.1 Overview

The concept of the “neuromorphic circuit” was first proposed by Carver Mead [1] in 1989 to describe electronics that can replicate neurobiological behaviour. The key goal of this community is to understand how neural circuits process information and how biological systems adapt to different environments incorporating learning, robustness to damage and development.

This field can inspire hardware engineers and computer scientists to design and build the next-generation computational platform, which captures the major merits of the brain’s features: highly parallel computing, ultra-low power consumption, strong fault tolerance and adaptive capability [2].

There are two main streams within the neuromorphic community: bio-inspired and bio-mimicking groups, as shown in Figure 1-1. The bio-inspired group primarily investigates how to develop an electronic system that can capture concepts or features of biological processes [3]. For example, inspired by the insect fly navigation optic flow (OF) sensing system, which can easily avoid hindrances and accurately move in the most changeable environments, an FPGA-based elementary motion detector (EMD) model [4] is developed to replicate this smart navigation mechanism, which is applied on a MicroAirVehicle.

Bio-mimicking groups attempt to use electronics to exactly reproduce biological neural network behaviour in real-time computing [5]. Their purpose is to try to understand the neural mechanisms of insight.

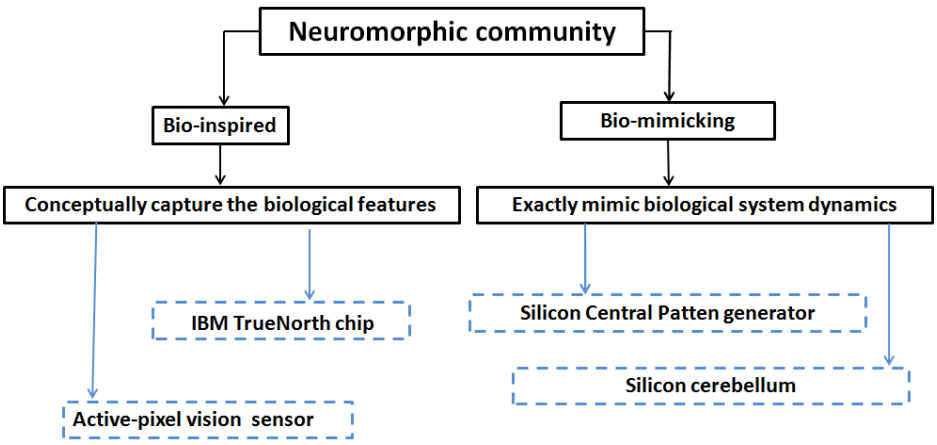


Figure 1-1: The neuromorphic community classifications: bio-inspired and bio-mimicking groups. The bio-inspired devices include IBM “TrueNorth” process chip. and dynamic vision sensors (DVSs); the bio-mimicking system contains a

silicon central pattern generator for cat movement prosthesis and silicon cerebellum for mouse fine movement control recovery

Nowadays there are several projects closely related to this field as shown in Figure 1-2. The human brain project [6] was established in 2012 by the European Union. It is a 10-year 1.19 billion euro scientific research project that aims to fully map human brain activity on specifically designed hardware. Its purpose is to provide better understanding of the mechanisms of the brain. In addition, it also plans to design and build a computational model that can be used to explore the effect of psychoactive drugs on the human brain. However, there has been some controversy in that cognitive scientists are largely excluded from the project. This indicates that this large flagship project mainly focuses on low-level bottom-up approaches [7]. The US-based BRAIN Initiative (Brain Research through Advancing Innovative Neurotechologies) [8] is another giant project related to neuromorphic computing. It was started in 2013 under the Obama administration. The total funds are $300 million per year over ten years. Itwill initially map the mouse neural network dynamics and eventually transfer these into the human brain neurons.

On the computer architectural side, a project called SpiNNaker [9] has also given strong impetus to the neuromorphic computing community. It is a highly parallel computing platform that is mainly focused on the three areas of neuroscience, robotic and computer science. The platform hopes to evolve to a million-core system to simulate the brain cortex neurons in real time. Similarly, IBM started the SyNAPSE (Systems of Neuromorphic Adaptive Plastic Scalable Electronics) project [10], which aims to design a neurosynaptic chip. The aim is to reproduce brain computing characteristics related to efficiency, size and power consumption. The main applications will be for cognitive tasks such as pattern recognition with new programming languages [11].

Meanwhile, one of the largest semiconductor companies, Qualcomm, recently developed the first commercialized neuromorphic chip, “Zeroth”, in 2014 [12]. “Zeroth” is able to observe and predict the external environment similarly to human beings. The chip has defined a new concept, the Neural Processing Unit (NPU), which is a new class of processor mimicking the cognitive functions of the brain. Compared to traditional chips, they argue that the NPU is more

suitable for detecting and recognizing visual figures and patterns in complicated data with much higher power efficiency than conventional systems.

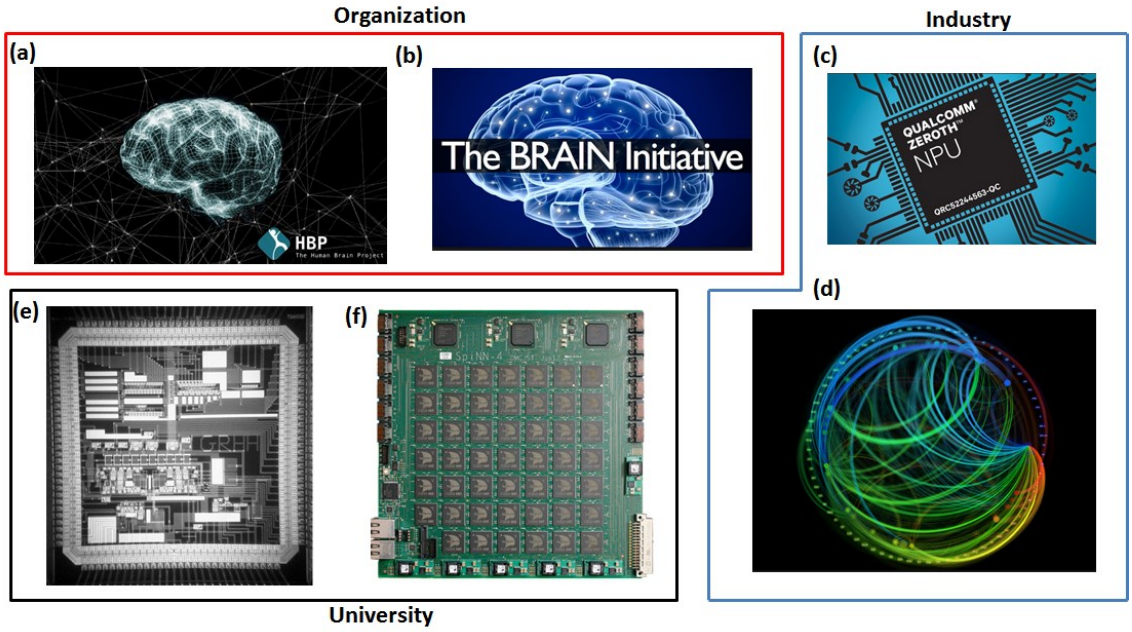


Figure 1-2: The neuromorphic communities from organizations, industries and universities. (a) is the human brain project emitted by the European Union; (b) is the BRAIN Initiative supported by the American government; (c) is the first generation of commercialized neuromorphic Zeroth chips from the Qualcomm company; (d) is the visualization of a simulated network of neurosynaptic chips from IBM research; (e) is the analogue CMOS-based chip designed for two-neuron communication (MIT); and (f) is the Spinnaker computational platform of Manchester University.

Finally, in 2011, researchers at MIT [13] designed the first analogue chip that could simulate ion-based communication between two neurons. It was fabricated by standard CMOS manufacturing techniques with 400 transistors. The MIT's chip is capable of reproducing the synaptic behaviours of spike rate-dependent plasticity and spike-timing-dependent plasticity hebbian learning rules.

1.2 History and trends

The first conceptual neuromorphic circuit was developed by Carver Mead [1] in 1990. He used analogue circuits to mimic active ion channel current-voltage behaviours in a nerve membrane. As this field develops, the neuromorphic circuit has broader scopes such as analogue, digital and mixed-model analogue/digital VLSI. Particularly in the high-level exploration of implementing neural networks (using a Field-Programmable Gated Array), in 2004 E.L. Graas was the first [15] to implement a Hodgkin-Huxley (HH) neural model in digital

circuits [15]. This specific field has rapidly developed and become an important niche in society nowadays. The main developed history is shown in Figure 1-3. E.L. Graas's research gave a basic framework for using FPGAs to implement computational neural models. It described the time multiplexing technique and speed optimization issues. Then, in 2007, Andrew Cassidy [16] used 32 digital neurons to replicate biological synaptic plasticity behaviours. This indicated that the digital neural system was capable of reproducing vital neural system performances. Meanwhile, RK Weinstein [17] contributed an auto-development tool kit for implementing neural models; this developed tool kit can not only alter model populations but also model inherent architectures such as adding/deleting ion channels. A pre-Bötzinger complex model was implemented as a case study that contains 40 HH-based neurons. After that, researchers started to investigate novel hardware architectures for large-scale neural network implementation; SW Moore [18] and Kit Cheung [19] developed a Bluehive and FPGA-based neural modelling accelerator that could implement 256,000 and 64,000 Izhikevich neurons in 2012. However, these neurons showed poor bio-plausibility. Recently, G Smaragdos [20] presented a digital network based on 96 HH neurons with compartments in 2014; it significantly

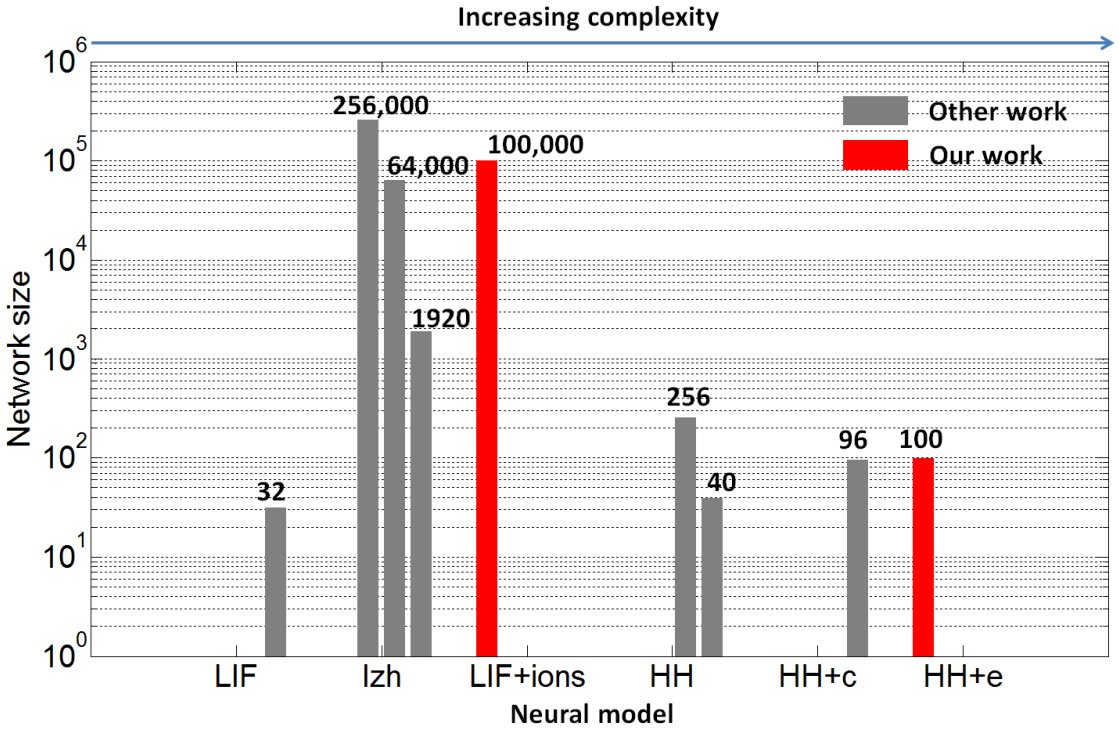


Figure 1-3: History development diagram of digital neural circuits. The x-axis is the implemented network size and y-axis is the bio-plausibility level: leakage

integrate-and-fire (LIF) model, Izhikevich model, LIF with ion expression model, Hodgkin-Huxley (HH) model, HH model with compartment parts such as soma and axon (HH-c) and HH model containing voltage & calcium ion channel and ChR2 channels (HH-e).

improved digital neural network bio-realistic characteristics but the number of neurons is limited.

It can be deduced that the aim of digital neural network implementation is to create very large-scale networks with highly bio-plausible behaviours. However, the main challenges lie in limited hardware resources and biological real-time computing requirements. Using timing multiplexing or pipelining techniques can significantly save hardware resources but affects calculation speed. Parallel implementation allows digital circuits to do biological real-time calculations but requires massive resources. Also, since large-scale neural networks have more complicated synaptic connections and neuron/ion types, the implementation requires customer-designed routing technology and heterogeneity architectures, which increases the design difficulty.

1.3 Rationale

In summary, the implementation of a current digital circuit-based high-level neural network has two limitations:

1. It still cannot reproduce multi-ion channel-type activities including both electricity- and chemistry-related behaviours.
2. When the network scale becomes very large (*e.g.* 100,000), the system has to use a simplified neuron model and shows poor bio-plausibility.

In this work I have developed two novel hardware architectures to address these issues:

1. **Pipelining-Based Multi-Loop Process Mechanism**: A Pipelining-Based Multi-Loop Process architecture is presented that can mimic different ion channel-type activities (voltage-dependent, voltage & calcium-dependent, Channelrhodopsin). This successfully reproduces the ions closed-loop process mechanisms, including both in electricity and chemistry, and fills

the gap whereby previous architectures can only implement voltage-dependent ion channel models.

2. **The Frame Based Network-on-Chip**: A frame based network-on-chip architecture is also presented to implement a cerebellum model that contains 100,000 neurons. At the same time, the implemented model still has high plausibility. It can accurately mimic biological passenger-of-time functionalities, and the network is based on a conductance-based integrate-and-fire neural model.
3. **The Hybrid Bio-silicon Network**: This network is designed for central pattern generator rehabilitation, which can be considered one of the potential important applications of digital neural circuits.

1.4 Contributions and organizations

The major contributions are as follows:

- A bio-realistic digital ion channel model for the neuron, which can incorporate 13 different types of ion channels. The advances include the implementation of a channelrhodopsin model into a digital platform, together with a multitude of calcium dependent and independent ion channels. These latter channels are derived from biological data from the ion channels of crustaceans (crab). Although the creation of a MatLab model may be interesting in its own right, I have additionally created a digital processing platform that can explore networks of these neural models in real time. Specifically I have utilized a Field-Programmable Gated Array (FPGA) to achieve the implementation. This allows scalability not only for closed-loop neuroscience experiments but also prosthetic applications.
- An efficient FPGA-based network-on-chip (NoC) hardware architecture has been developed for implementing a very large-scale neural network. This has been used to implement a 100 k granular-layer model of the cerebellum to explore passage-of-time (POT) behaviours. The computational delay has been sustainably minimized to 25.6 ms for running a 1 s real-world activity. This model may have future applications in neuro-prosthetics for ataxia.
- A reliable and capable system is presented specifically for CPG function restoration. Compared to previous systems, the work is stronger in two

aspects: silicon neuron bio-plausibility and system reliability. Firstly, digital neural circuits are designed to reproduce both real CPG control and pharmacological outputs, which particularly aim for conditions with a totally damaged and partially damaged system. Secondly, the designed system has the capability of robustly changing the computing speed to achieve the best communication performances with biology by using an adaptive control mechanism.

The selected publications are as follows:

Journals

1. J. W. Luo, G. Coapes, T. Mak, T. Yamazaki, C. Tin, and P. Degenaar, "Real-Time Reproduction of Passage-of-Time Functionality Using FPGA," in *2014 IEEE Transactions on Biomedical Circuits and Systems* (minor revision).
2. J. W. Luo, Peter Andras, Alex Yakovlev, and P. Degenaar, "Digital Implementation of Bio-realistic optogenetic neurons," in *2014 Journal of Neural Engineering* (prepared).
3. J. W. Luo, T. Mak, Peter Andras, Alex Yakovlev, and P. Degenaar, "A Reliable Central Pattern Generator Prosthesis Technique Based on Digital Neural Circuits," in 2014 *IEEE Transactions on Neural System and Rehabilitations (prepared).*

Conferences

1. J. W. Luo, T. Mak, B. Yu, P. Andras, and A. Yakovlev, "Towards neuro-silicon interface using reconfigurable dynamic clamping," in *Conference proceedings: ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 2011, vol. 2011, pp. 6389–92.
2. J. W. Luo, P. Degenaar, G. Coapes, A. Yakovlev, T. Mak, and P. Andras, "Towards reliable hybrid bio-silicon integration using novel adaptive control system," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, 2013, pp. 2311–2314.
3. J. W. Luo, P. Degenaar, A. Yakovlev, T. Mak, and P. Andras, "A novel hardware architecture for large-scale hybrid bio-silicon network," in *2012 Royal Academy of Engineering Young Researchers Futures Neural Engineering meeting, Warwick, 2012.*
4. J. W. Luo, T. Mak, P. Andras, and A. Yakovlev, "FPGA-based simulation of the pyloric circuits of the crab stomatogastric ganglion," in *2012 Neuroscience, D.11.f;G.06.a .*
5. J. W. Luo, G. Coapes, T. Yamazaki ,T. Mak, C.Tin, and P. Degenaar, "A Scalable FPGA-based Cerebellum for Passage-of-Time Representation.," in *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 2014.

The thesis structure is organized as follows:

Chapter 1: **Overviews and Purposes.** This briefly describes neuromorphic computing background, history and development trends and the work purposes.

Chapter 2: **The Fundamentals.** This reviews the basic computation architectures and circuits, the digital and neural computing mechanisms, current implementation techniques and systems, the FPGA advances and developments.

Chapter 3: **The Digital Optoelectronic Neuron.** A bio-realistic digital ion channel model, which can incorporates 13 different types of ion channels. The advances include the implementation of a channelrhodopsin model onto a digital platform, together with a multitude of calcium dependent and independent ion channels.

Chapter 4: **The Digital Cerebellum.** A frame-based network-on-chip (NoC) architecture for implementing a very large-scale neural network (100,000) with specific biological passage-of-time (POT) functionalities is presented. The design could be a potential neuro-prosthetics tool for future experimental and clinical applications owing to its high computational power, flexibility, high scalability and power efficiency.

Chapter 5: **Case study: Central Pattern Generator Prosthesis Technique.** A reliable and capable system is presented specifically for CPG function restoration. Compared to previous systems, this work is stronger in two aspects: silicon neuron bio-plausibility and system reliability. Firstly, digital neural circuits are designed to reproduce both real CPG control and pharmacological outputs, which are particularly aimed at conditions with a totally damaged and partially damaged system. Secondly, the designed system has the capability of robustly changing the computing speed to achieve the best communication performances with biology by using an adaptive control mechanism.

Chapter 6: **Conclusion.** This summarizes the main work of the thesis and briefly describes the major contributions. Also, the things that need to be improved and future work are presented as well.

Chapter 2 The Fundamentals

This chapter first describes the basic computational principles of digital systems and biological neural networks. After that, these two systems are compared. In particular, the different features are emphasized. Then it gives a brief historical review of previous digital-based biological systems. Finally, the design conclusions are also presented.

2.1 Digital computational architectures

A basic digital computational device can be defined as a system or circuit that is capable of performing information processing or specific functions that people require in their daily lives. For example, a calculator: by entering several digital numbers, the machine will automatically carry out arithmetical operations that you need such as addition, subtraction, multiplication and division. Specifically in numerical computing, it can perform such computations at much faster speeds than the human brain. Designing a digital computational device in general raises several basic questions:

1. How can the digital circuit state best represent analogue (real) world information.
2. What is the optimal architecture for processing and storing information?
3. How should computational components communicate with each other.

We are living in an analogue world. The information we sense is continuous values changing with time. A digital computational system has to represent analogue world information (*e.g.* continuous values) by using digital states such as low and high, on and off, charged and discharged. A positional number system has been developed to address this issue. By using the position of digital bits, each with different weights, numbers can be represented in a digital system. The equation is shown in Equation 2-1:

$$D = \sum_{i=-n}^{p-1} d_i \, r^i \qquad \text{Equation 2-1}$$

where r^i is the weight and d_i is the analogue values, the rightmost bit ($i = -n$) is called the least significant bit (LSB), and the leftmost bit ($i = p - 1$) is called the most significant bit (MSB).

Once the analogue value can be represented by digital circuit states, the next key consideration is how to use logic signals and gates for information processing. The gate functions AND, OR and NOT are developed as the basic logic operations; the symbols and corresponding truth table are shown in Table 2-1and Figure 2-1. The complementary Metal-Oxide Semiconductor (CMOS) is the fundamental unit for implementing these logic functions.

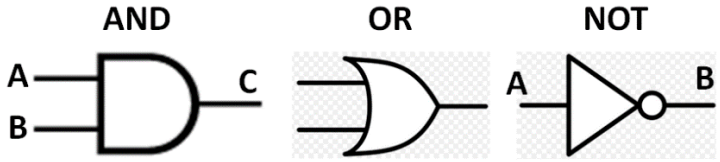


Figure 2-1: The basic gate functions: AND, OR and NOT.

Table 2-1: The truth table of logic gate functions

AND			OR			NOT	
A	B	C	A	B	C	A	B
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	1	1	0	1		
1	1	1	1	1	1		

It is a three-terminal device that can be considered as a voltage-controlled resistance or amplifier. In the digital operation principles, the MOSFET is always operated either very high (switch on) or very low (switch off). An example of using CMOS to implement a NOT function circuit is shown in Figure 2-2: A: the typical CMOS inverter architecture for NOT gate function, B: The typical input-output transfer characteristic of a CMOS inverter.

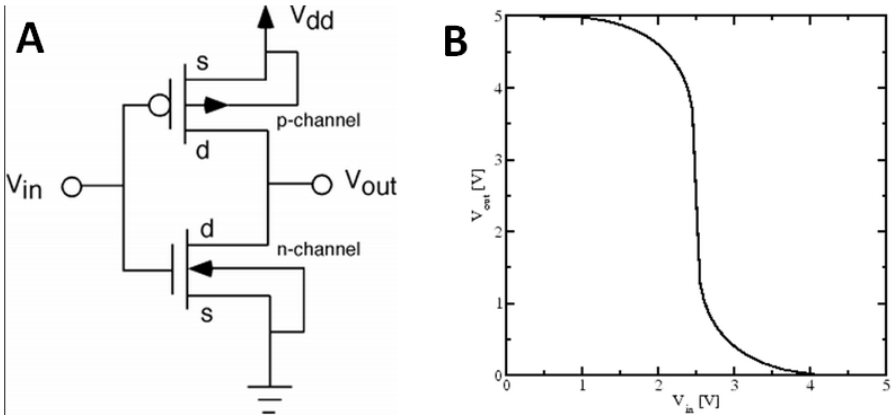


Figure 2-2: A: the typical CMOS inverter architecture for NOT gate function, B: The typical input-output transfer characteristic of a CMOS inverter.

Therefore, by combining and arranging different numbers of these gate functions as shown in Figure 2-1, from these building blocks all important circuits and memory elements can be created. When the circuit outputs are purely dependent on the input values, it is defined as a combinational circuit. When a process has to involve previous inputs or calculations, a memory

component such as flip-flops or latches is necessary. This circuit is defined as a sequential circuit.

As the required information process functions become more complex and varied, people seek to develop a general-purpose computational device with a reprogrammable function, thus the architecture requires a customer-designed controller or program register to manage operation sequences and data and become more sophisticated. The Central Processing Unit (CPU) came of age.

Very early in the 19th century, there were two basic CPU structures: the Von Neumann [21] and Harvard computing architectures. The Von Neumann architecture contains a processing unit that consists of:

1. An arithmetic logic unit.
2. Registers (accumulators).
3. A control unit that consists of an instruction register and program counter.
4. An external massive memory storage.
5. Input/output pads.

The arithmetic logic unit is responsible for calculating data such as add, multiply and subtract operations and comparisons such as “greater than” or “less than”. The control unit is for managing the process of moving data and codes in and out of memory, and also for executing program instructions. The memory is for storing both data and program instructions such as random access memory. The Harvard architecture maintains the same components but the key difference is that instruction and data memory are physically separate and have different signal pathways, as displayed in Figure 2-3.

Because the Von Neumann architecture instructions and data memory share the same communication bus, it strongly limits the effective calculating speed. The CPU speed becomes limited by the time taken for memory access. Harvard architecture, modified Harvard architecture or parallel computing can alleviate this performance problem because the data bus is separate.

The basic CPU goes through the following process sequence: first, it fetches the instruction in the memory location indicated by the Programmer Counter (PC), and loads it into the Instruction Register (IR); then the PC will be automatically updated to indicate the next instruction by increasing an

appropriate amount. At the execution phase, CPU will carry out the instruction in IR and execute it. This is the typical sequence of the fetch-execute cycle.

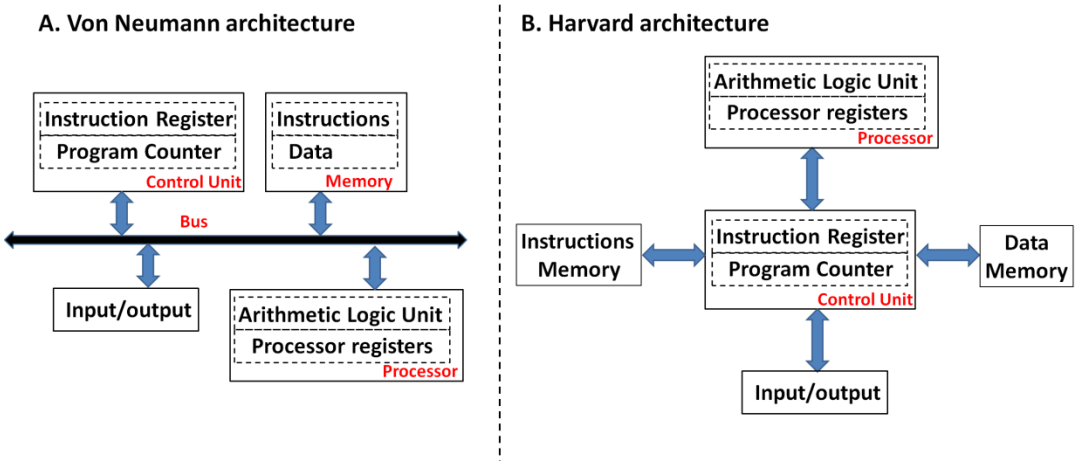


Figure 2-3: Comparison between Von Neumann and Harvard computing architecture.

As the computer technology develops, the modern CPU is capable of performing general-purpose tasks and significantly enhancing people's quality of life. However, the CPU faced limitations as graphics process requirements increased. Such processing requires massive matrix and vector operations, which take an extremely long processing time when processed sequentially. Engineers then developed a tailor-designed highly parallel computing device for graphic processing tasks, called the Graphic Processing Unit (GPU). The first consumer-level GPU card, named Nvidia GeForce 256, was released in1999.

A GPU [22] is an interesting computing architecture. It has a highly parallel structure. It is a heterogeneous chip multiprocessor. Because there are lots of matrix and vector calculations, the basic architecture is shown in Figure 2-4. The red block is the fetch/decode function unit that sends an instruction stream across many ALUs, which refers to single-instruction multi-data processing. The yellow block is the ALUs. And the blue block is execution contexts and shared memory.

The GPU [24] process mechanism is complicated and often involves many steps. The basic operation principles are as follows: first, everything is translated into triangles by using a computer graphic library. Then the lighting process will identify each triangle colour. After that, all these triangles are translated into the virtual camera's film coordinates. The rasterization step will separate all the overlapped triangles. Next each camera pixel colour will be

identified and the incorrect hidden surfaces of objects will be removed. Nowadays people increasingly use GPU for other non-graphical applications such as bitcoin mining and neural network modelling [23].

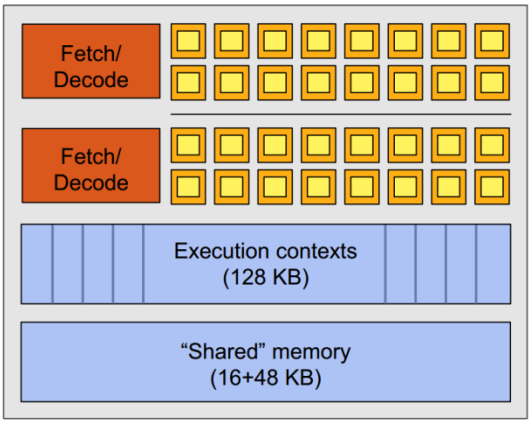


Figure 2-4: The NVIDIA GeForce GTX580 “core”. The yellow block is the SIMD (Single Instruction Multi Data) function unit. This figure comes from the Fermi Compute Architecture Whitepaper CUDA Programming Guide 3.1.

Also, as the automation, mechanical and electrical industries develop, some information process functions in commercial products have to be specifically designed to save hardware resources, increase power efficiency and enhance speed, in terms of raising net benefits. This raises people’s interests in designing an Application-Specific Integrated Circuit (ASIC).

An ASIC is a customized integrated circuit for a specific function. The first ASIC was a gate array invented in 1980 by Ferranti. The design methodology of ASICs can be roughly divided into three categories: gate-array designs, standard-cell designs and full-customer designs. The gate-array design is where transistors or other active devices are predefined and unconnected. The interconnections of the final system are decided by the engineering. Nowadays it has been almost entirely replaced by FPGAs. The standard-cell design uses manufacturer-designed standard function blocks to build circuits with high electrical performance. This design involves several stages: module specification, top-level design, system implementation, simulation, synthesis, layout and testing of silicon. The full-customer design is defined all the silicon layers of the device. The advantages of full-customer design usually include smaller areas, speed enhancement and less power consumption, and also the

ability to integrate other components. Examples of gate-array and full-customer-designed ASICs are displayed in Figure 2-5.

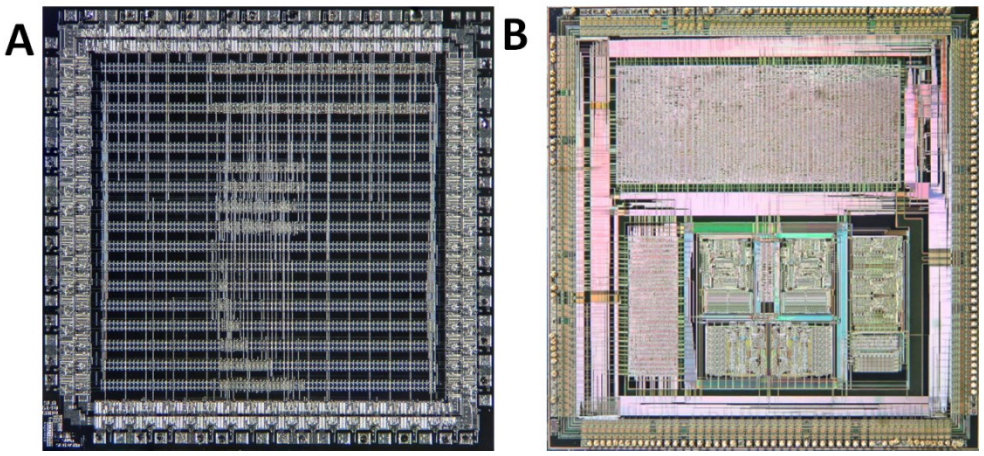


Figure 2-5: A: gate-array-designed ASIC; B: full-custom-designed ASIC.

On the other aspect, all the above three different architectures (CPUs, GPUs and ASICs) can be implemented in a digital reconfigurable tool, which is the FPGA.

Ross Freeman and Bernard Vonderschmitt released the first commercial available FPGA in 1985, named XC2064 [25], which created a new beginning market of computational architecture. The FPGA [26] is a bit different from previous computing architectures; it is defined as “a prefabricated silicon device that can be electrically programmed to become almost any kind of digital circuit or system” [26]. This is done by customized programming technology, which can change circuit performances after chip fabrication. The digital circuits are created in the “field”. The conceptual structure of an FPGA is displayed in Figure 2-6. It contains routing channel, logic block and I/O interfaces.

The routing channel design refers to programming technologies. The approaches include EPROM [27], EEPROM [28], flash [29], static memory [30] and anti-fuses [31]. Among these approaches, the flash, static memory and anti-fuse techniques are widely applied in the FPGA model. The logic block is for implementing circuit function; the design has to consider the trade-off among speed, power and areas. The I/O pad is the input and output interface.

The FPGA contains three main elements: Look-Up Table (LUT), flip-flops and routing matrix. Look-Up Tables are fundamentally how logic is actually implemented on a block of; the output is the values of the corresponding

indexed address location. Flip-flops are typically used for function reset or latching. They are usually connected to the output of LUTs, which consist of a slice. The complex logic block contains two slices in FPGAs. The routing matrix is a number of multiplexers and wires that respond to connecting CLBs and the other FPGA resources. For example, a summation function needs to be implemented that requires an adder operator. An adder can be synthesized by using several logic functions including: AND, OR and NOT. These logic functions are implemented by using LUT; the connections between them are achieved by using a routing matrix. Specifically, system reset, enable and memory functions can be realized by using flip-flops.

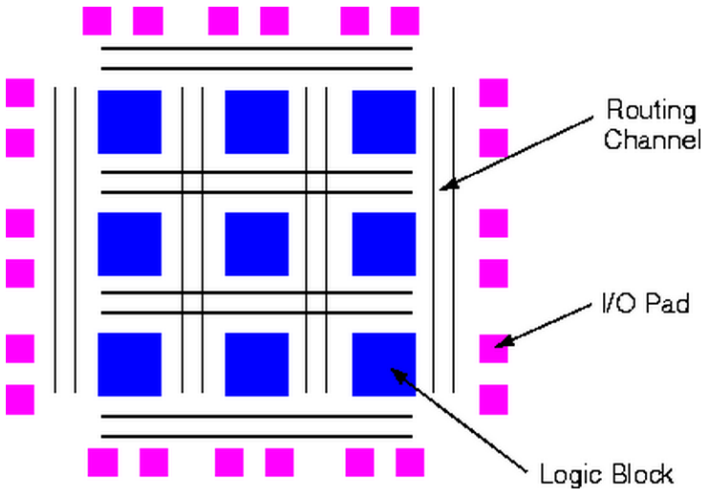


Figure 2-6: The conceptual architecture of an FPGA. The figure is cited in [26].

Overall, the characters of each platform are summarized in Table 2-2. There are two computing mechanisms for sequential and parallel approaches. CPUs follow a sequential computing approach and comprise four main steps – fetch, decode, execute and write-back – while GPUs, FPGAs and ASICs use parallel computing.

The clock signal governs all different digital platforms' computing speed; it is very important and can allow or stop a process and in general provide synchronization for the circuits. Increased clock frequency can directly make digital processors run faster, but it is limited by circuit delays. The clock period has to be longer than the total propagation delay of the circuits to avoid glitches. Generally, larger circuits have longer propagation delays. The clock cycle of CPU Intel Core i7-960 can be up to 3.2 GHz, which is much faster than a Nvidia GTX285 1.5 GHz. Specifically, since FPGAs have switch blocks in the circuits,

which have large propagation delays, the clock frequency of FPGA V6-LX670 is 0.3 GHz.

Table 2-2: Comparisons among different computing platforms

	CPUs	GPUs	FPGAs	ASICs
Mechanism	sequential	parallel		
Architecture specifications				
Clock cycle	3.2 GHz	1.5 GHz	0.3 GHz	-
Die area	263 mm^2	470 mm^2	-	-
CMOS tech	45 nm	55 nm	40 nm	-
Benchmarks (Fast Fourier Transform)				
GFLOP/s	67	250	380	952
GFLOP/J	0.71	4.2	6.5	90
Characteristics				
Flexibility	strong	strong	strong	weak
Design cycles	normal	normal	relatively long	long
Cost	cheap	cheap	normal	expensive
Implantable	no	no	no	yes

*: CPU is an Intel Core i7-960; GPU is a Nvidia GTX285, FPGA is a V6-LX760; the ASIC circuit is the same RTL in 65 nm for FFT implementation; GFLOP refers to a unit of computing capacity equal to one billion floating point operations per second. The benchmark data is cited at Computer Architecture Lab at Carnegie Mellon.

A benchmark Fast Fourier Transform (FFT) algorithm was implemented on these four platforms. The characteristics of FFT are complex dataflow and low arithmetic density. The results indicated that ASICs have the fastest computational speed of 952 GFLOP/S and CPU has the slowest speed of 67 GFLOP/S. However, the power consumption of CPU is 0.71 GFLOP/J and that of ASICs is 90 GFLOP/J.

In terms of system flexibility and feasibility, CPU- and GPU-based platforms enjoy strong flexibility and low cost, and the level of design difficulty is relatively easy. Meanwhile, FPGAs are also reconfigurable platforms with normal costs; the design cycles are generally a bit longer since hardware design requires extra time for circuits' synthesis and on-board testing. Finally, ASICs are non-reconfigurable and expensive, and the design time generally takes months, depending on the specific target. But the circuits are implantable and more efficient in terms of power consumption and computing speed.

2.2 The digital circuit design flow

The overall design flow of digital Integrated Circuit (IC) implementation is described in Figure 2-7. A digital ion channel implementation is given as a case study. First, a mathematical biological neural model/algorithm (function) is selected for implementation. By carefully considering the model parameter

range and resolution, neural network architecture and functionalities, two hardware architecture generation tools can be candidates for designing: Very high-speed integrated circuit Hardware Description Language (VHDL) and visualization software Cadence. Cadence gives more design flexibility and controllability, and VHDL is for high-level architecture (system) modelling. In this case study, a voltage-dependent ion channel model is described by using VHDL. And the next step is to employ ISE software to carry out behaviour and post-translate simulation. Behaviour simulation verifies model functionalities from the logic-design perspective, while post-translate simulation includes physical hardware constraints such as timing and layout issues, which is as close as the real hardware calculation. After the synthesis, the developed hardware architecture is represented by using the register transfer level. Finally, a Virtex-7 evaluation board is used for implementation and on-board testing.

After verification by using an FPGA, the next milestone is to transfer VHDL into ASIC circuits. At this stage the software Synopsys is applied to transfer a previous VHDL code into a netlist in terms of generic cells such as *and*, *or*, *not* and sequential elements and mapping into logic cells from the CMOS library. Specifically, timing, area and power performances of architecture should be optimized to meet requirements. A synthesized netlist result is shown in Figure 2-7 as well. Then the software Encounter is chosed to perform a digital IC place & route task, which includes floor planning, placement of cells, clock tree synthesis and optimization, routing of nets and full custom layout finishing (if required). A 90 nm CMOS library is selected for mapping the ion channel model and the final physical layout is shown in Figure 2-7. Finally, in the Signoff stage, static timing analysis, dynamic simulation, formal equivalence checking, power analysis (peak, average and time based) and transistor-level simulation should be considered.

2.3 How the neuron works

Compared to the artificial information process system based on silicon, the natural information process system of the biology shows totally different features. The basic processing unit of biology is called the neuron.

A typical biological neuron consists of a soma, dendrites and an axon (Figure 2-8B). It processes and transmits information through electrical and chemical

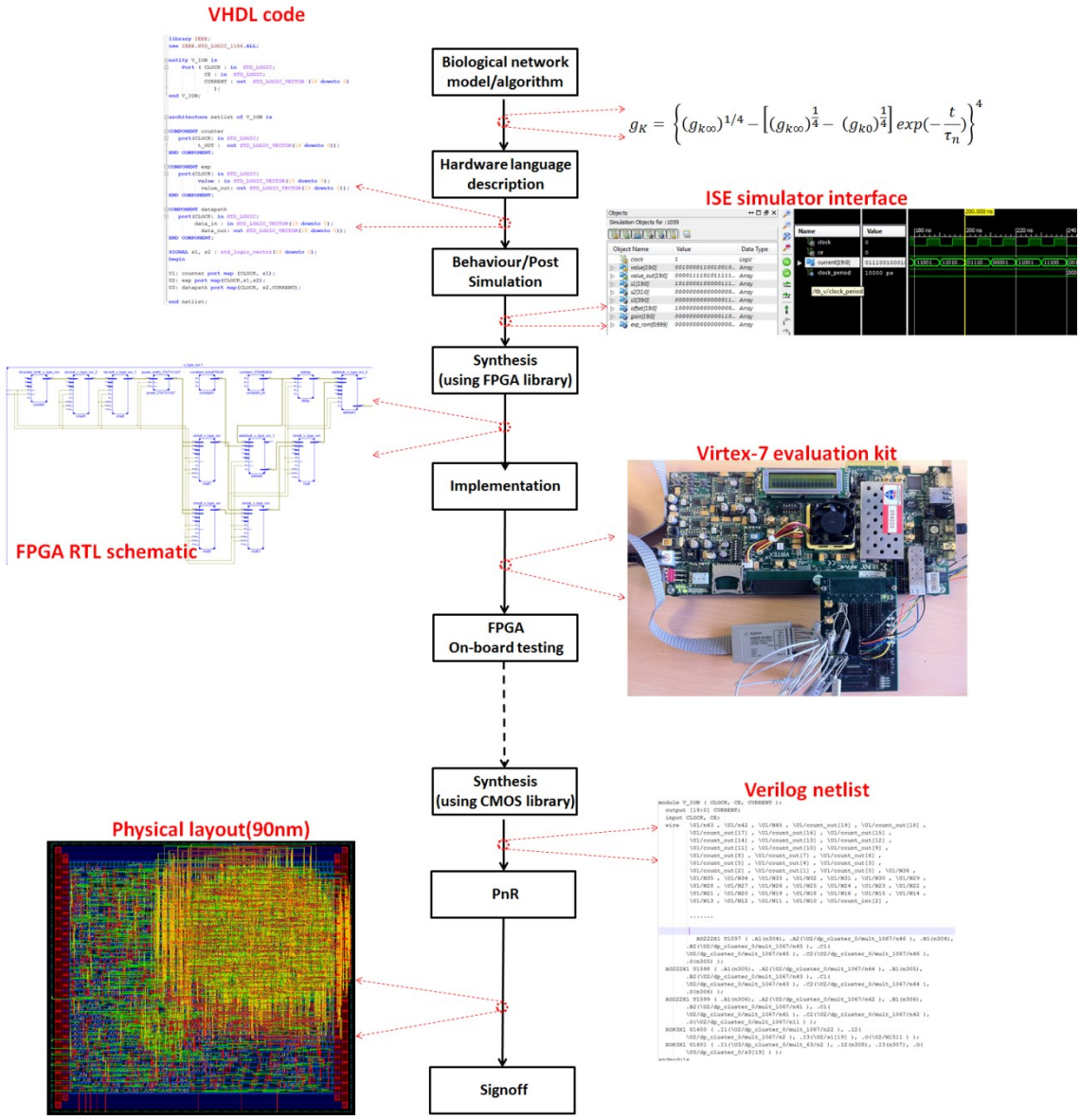


Figure 2-7: The design flow of digital Integrated Circuits (IC) implementation. A case study of implementation of an ion channel model is given as a demonstration.

signals. The basic single neuron computational mechanisms can be described as follows: first, charges from the spikes are summated at the dendritic tree, which is an information receiver. The summated output from dendrites will be sent to the soma for processing (*e.g.* integration). If a large enough amount of ions (inputs) change, an output signal will be generated. This signal can be considered a digital event that is transmitted to the other neurons by the axon. Here the axon can be seen as an information sender. Dendrite computation is analogue while axon communication is digital.

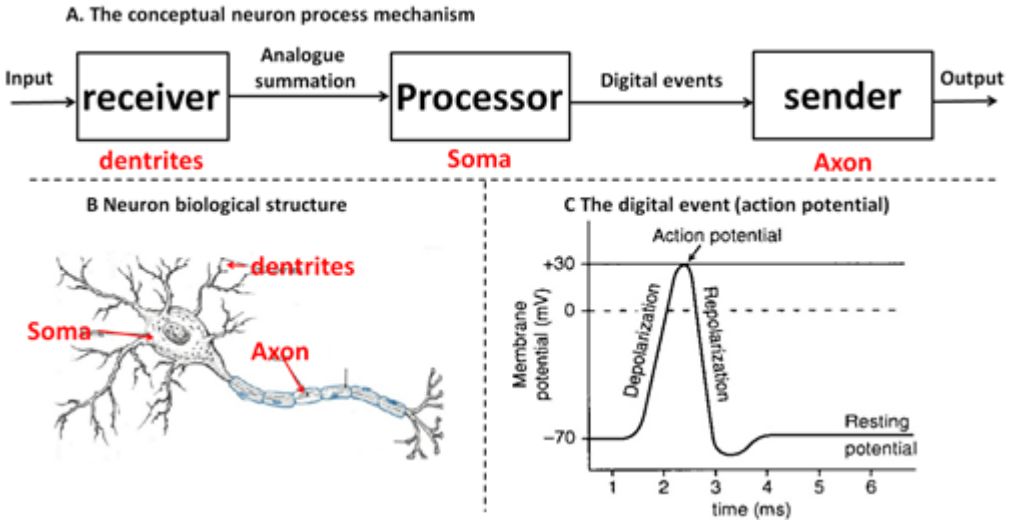


Figure 2-8: The single neuron computational mechanism. A is the conceptual neuron process mechanism; B is the neuron biological structure; and C is the digital event (action potential).

Particularly in neuron processing, the digital event is called action potential in the biological system. It mainly depends on the two components ofthe soma membrane. One is called ion pumps, which maintain the negative voltage differences across the membrane, and the other one is called ion channels, which are responsible for generating the ions (*e.g.* sodium, potassium and chloride) concentration differences in and out of the neuron. In detail, the generation process can be divided into five parts: the rising phase, the peak phase, the falling phase, the undershoot phase and the refractory period. In the rising phase, if the depolarization current is large enough, the inward sodium current overwhelms the outward potassium current This indicates that the membrane potential increases. Therefore, the more membrane potential increases, the more sodium currents come in. Eventually, the membrane potential will increase towards the sodium equilibrium voltage of around 55 mV.

The peak and falling stage refers to the sodium permeability being maximized and the membrane potential being approximately equal to the sodium equilibrium voltage. After that, the sodium ion channels are closed and become inactivated, which indicates that the sodium membrane's permeability is lower than that of potassium, driving the voltage back to the resting state. The increased voltage activates the opening of more potassium ion channels that usual, so the potassium permeability of the membrane is very high in transient periods. This drives the membrane potential towards to the potassium equilibrium voltage, which is defined as post-hyperpolarization. In the refractory period, the two ion channels are return to normal, and the membrane potential will back to resting potential values. The five stages are summarized in Figure 2-8C.

The communication element between neurons is called synapse. Its location is between the previous neuron axon terminal and the next neuron dendrite, which is shown in Figure 2-9B. It allows the digital event (action potential) to pass between two neurons. There are two different types of synapses: electrical and chemical synapses. In a chemical synapse, the action potential will be translated into chemical signals, which are used to initialize the received neuron electrical response, while in an electrical synapse, the signal transmission is achieved by the gap junction.

One thing that should be pointed out is that the synapse plays an important role in the establishment of memory. When both communicated neurons are active at the same time, the connection between the two neurons is strengthened as a result of signalling mechanisms. This process is called long-term potentiation, which is acknowledged as memory information.

2.4 Comparison of neural and digital computing

In this section, I investigate the differences between neural and digital system process mechanisms, which are shown in Figure 2-11.

As can be seen, in the digital circuits, the basic element transistors can be synthesized into logic functions AND, OR and NOT. Then these functions are built into specific function circuits such as register, ALU and multiplexer and further to higher-level CPUs etc. In contrast, in the biological system, the basic component is neurons; these neurons interact with each other by using synaptic

connections to generate spiking patterns. This series of neurons can be equally considered as a system level with some specific functions. Figure 2-11

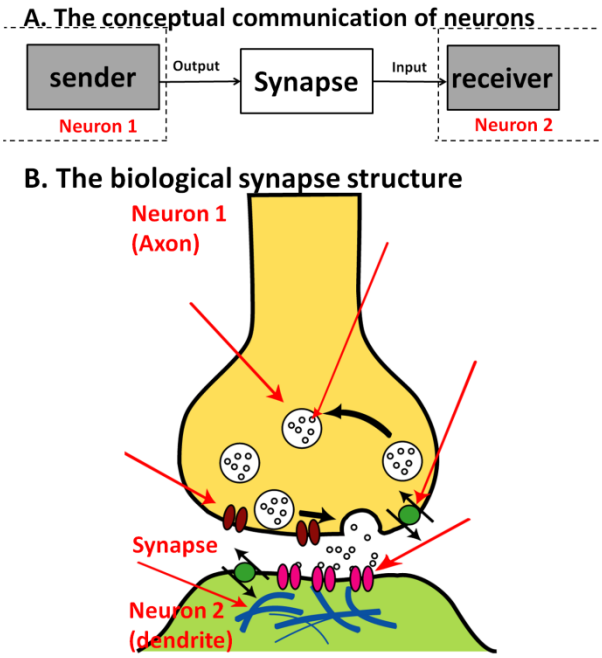


Figure 2-9: The biological synapse architecture.

shows an example of biological networks in the human brain for controlling people's daily life behaviours.

Both systems have different information encoding mechanisms. For digital circuit information encoding, it contains several important characteristics: synchronization, language, errors, copying, granularity and compressibility. Here is an example of two-wire serial control model in a portable Internet audio CODEC chip in Figure 2-10.

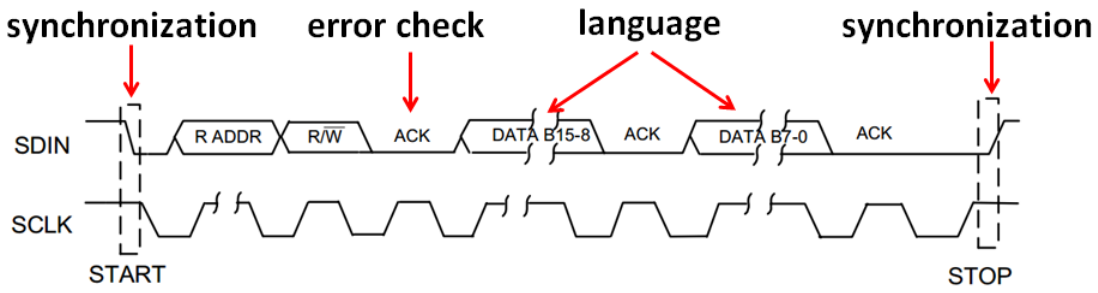


Figure 2-10: An example of digital system information coding. The figure displayed is the two-wire serial control model of a WM8731/L audio CODEC chip.

The synchronization refers to each digital signal frame; the information start and stop point are specifically designed, which can be recognized by both digital

sender and receiver. At this point when information starts to send, the clock signal is at the high level and data signal at the transition level from high to low, while when information is finishing, the clock signal is at the high level and data signal

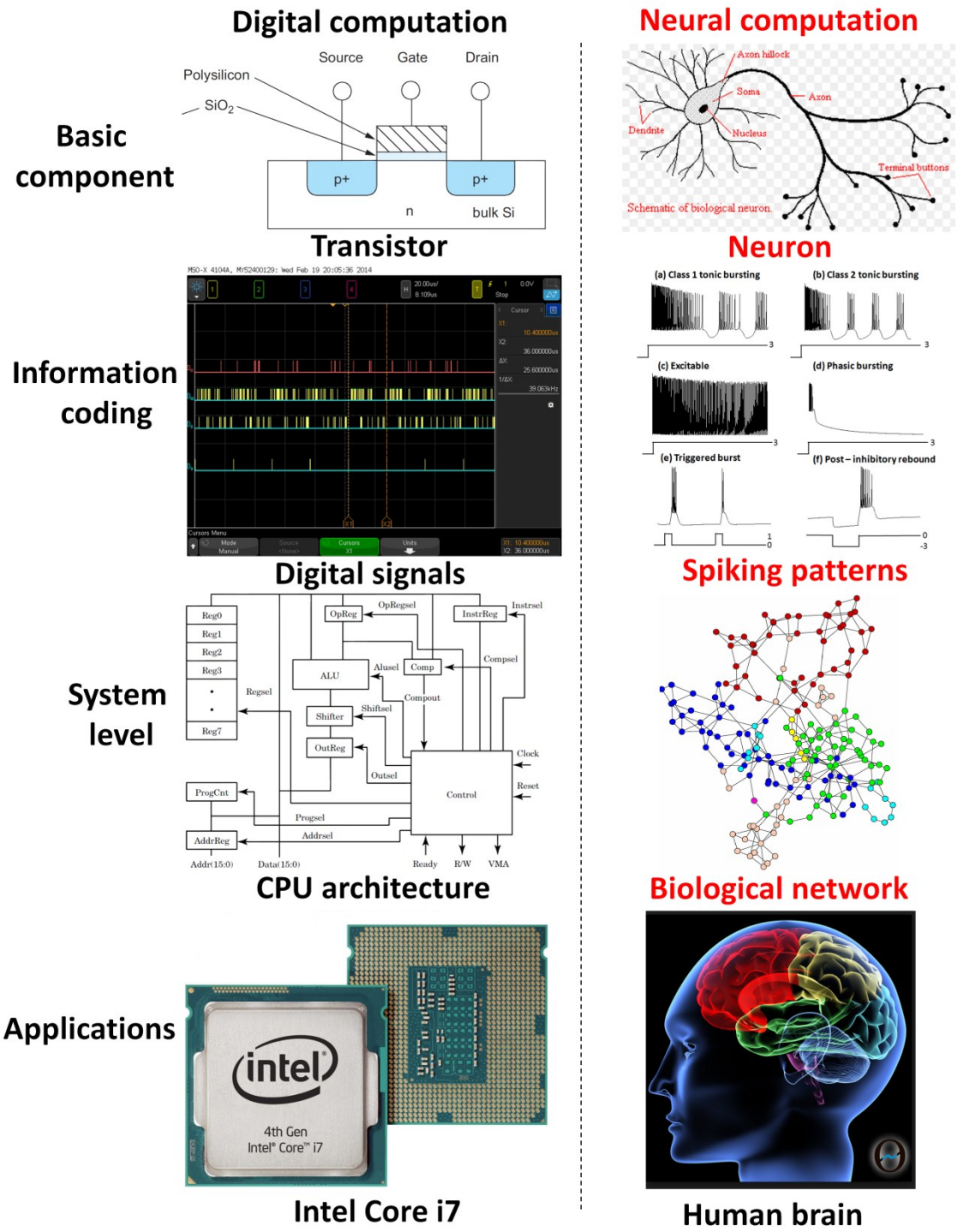


Figure 2-11: Comparisons between digital and neural system processing.

at the transition level from low to high. Once the digital communication system establishes a start condition, a eight bits(7-bit address+R/W bit) will be send out, and MSB(Most Significant Bit) is transferred first. If the correct address is received and the R/W is zero, which indicating a write function, then the WM8731/L will generate a ACK bit by pulling SDIN low on the next cycle. The WM8731/L is a write only device only respond to the R/W bit indicating a write. On the other hand, if the address is not recognised the device will return to the initialized condition and wait for a new start condition and valid address. Also, the digital information can be copied to the other system since it is noise free. And it can be compressible to save space. But there is a granularity (quantization error) in the digital information encoding, which refers to the differences between the actual analogue values and the digital representation.

Neural system coding can be classified into four schemes: rate coding [34], temporal coding [35], population coding [36] and sparse coding. Rate coding is defined as the information containing in the firing rate of the neuron. It was originally described by E. D. Adrain and Y. Zotterman in 1926. Temporal coding is defined as the information that is carried by precise spiking timing or high-frequency firing-rate fluctuations. The timescale of temporal coding is in the range of a millisecond. Both coding examples are shown in Figure 2-12. Population coding is an approach that uses correlation of a number of neuron' activities to represent sensory information. It is widely used in the sensor and motor areas of the brain. For example, the object moving direction can be retrieved from the monkey visual area medial temporal population activity. Sparse coding is information that is encoded by the significantly strong activation of a relatively small sparse set of neurons.

A comparison between digital and neural computation is shown in Table 2-3.

The digital computation system enjoys high calculation speed and limited bus connections, while the neural circuit has relatively slow firing frequency but

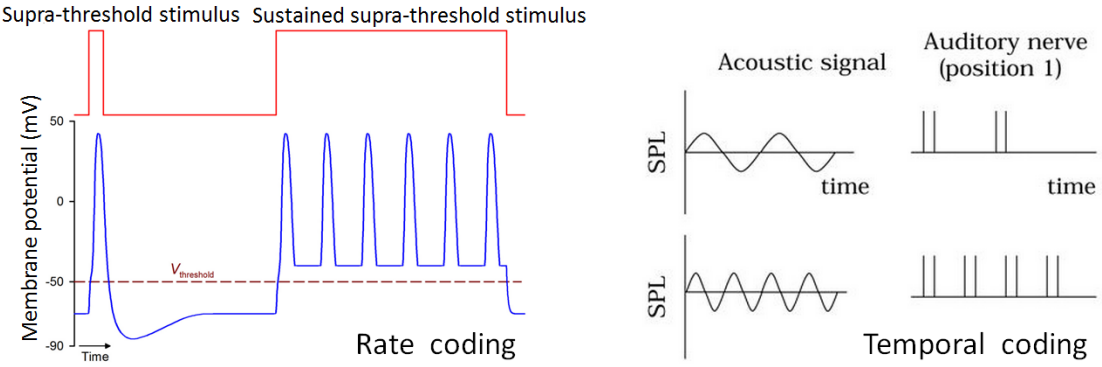


Figure 2-12: The neural coding schemes: rate coding and temporal coding.

Table 2-3: Comparison between digital and neural computation

Computation characters	Digital circuits	Neural circuits
Speed	Fast(GHz) Global clock	Slow (Hz) Event-driven
Architecture connections	Low(bus) connected	Highly connected
Information coding	Deterministic	Non-deterministic
Fault tolerant	Poor	Good
Learning	No	Yes
Applications	Numerical computing	Image processing

massive point-to-point synaptic connections. Also, the digital circuit is governed by a global clock. In each clock cycle, the circuit processes the information, while in the biological neural circuit, only when information comes in does the system have to do the processing. This can be thought as being based on the event-driven technique. The computation in the digital circuit is deterministic because the logic operation is fixed; but in the neural circuit it is stochastic. This may lead to corresponding poor and good fault tolerance characters. The digital system has fixed behaviours, while the neural circuit has a strong learning capability to be adaptive to external environments. This is due to the synaptic memory learning at the computation. For applications, the digital circuit is good at numerical computing and the neural circuit is good at image processing.

2.5 Digital based biological systems and techniques

Because the neural system has so many fascinating characters shown in Table 2-3, engineers aim to exactly reproduce these biological behaviours by using silicon to more fully understand how neural networks achieve this. The contemporary bio-mimicking society has successfully reproduced several core biological system behaviours by using digital circuits from ion to network levels.

A. The conduction and excitation of membrane current

E. L. Graas originally reproduced the conduction and excitation of membrane current biological dynamics by using hardware in 2004 [15]; the developed hardware architecture simulated 17 versions of the HH models and successfully predicts sodium, potassium and leakage ionic flow manners in a neuron under a variety of conditions. The developed system is implemented on a Xilinx Virtex-xc2v1000 and consumes 2186 slices and 12 RAM blocks of hardware resources. The on-board clock frequency is set at 40 MHz with a 0.001 ms simulation time step. The whole system used 45 ms to simulate all the models, which is 16 times faster than running on the computer.

In this work, two strategies that increase system clock frequency and integration step are suggested to increase system computational speed: clock frequency is decided by the longest critical path, and integration step will influence system reliability.

In particular, a simulation multiplexing (SM) technique [15] is presented in this work. The digital circuit computes multi-version models simultaneously by exploiting the latency in the computational architecture, and each model will be executed sequentially in lockstep.

The simulation multiplexing concept is illustrated in Figure 2-13 and the basic mechanism is as follows: each external input (1^{st}, 2^{nd}, 3^{rd},…, n^{th}) is sequentially sent into the process block (neural model) by a Time Division Multiplexer (TDM), and the neural model calculates each input signal and consecutively feeds back to the time division de-multiplexer as outputs. The TDM input channel number has to equal the neural model latency to avoid data process mismatch.

A. Synaptic ion channel behaviours

Excitation receptor-gated ion channel N-methyl-D-asparate (NMDA) and alpha-amino-3-hydroxy-5-methyl-4-isoxazole propionic acid (AMPA) receptor-gated ion channels are both implemented on digital circuits by using a component-based approach [37]. Those channels are mainly responsible for synaptic plasticity such as Long-Term Depression (LTD) and Long-Term Potential (LTP) [38][39]. These exist in the glutamatergic excitatory synapses as shown in Figure 2-14.

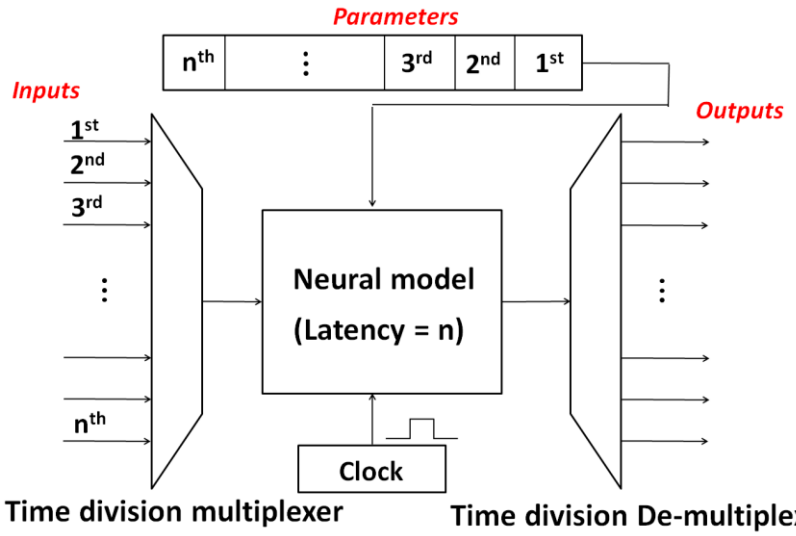


Figure 2-13: The conceptual architecture of simulation multiplexing technique.

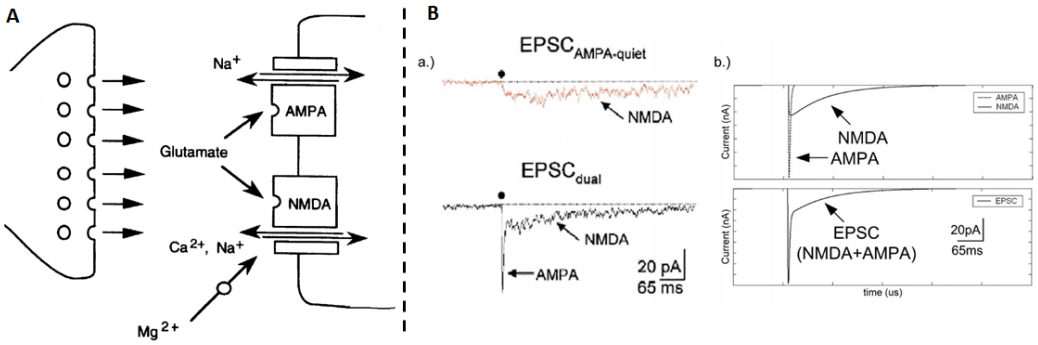


Figure 2-14: A: Both AMPA and NMDA gated ion channels are activated by excitatory neurotransmitter glutamate in a biological synapse. The figure is cited from [37]. B: (a) is the biological recordings of excitatory postsynaptic currents from NMDA & AMPA channels and individual NMDA channels. The figure is cited from [40]; (b) is the FPGA-based simulation results.

In Figure 2-14A, both AMPA and NMDA gated ion channels are activated by excitatory neurotransmitter glutamate in a biological synapse. In addition, Na^{+}, Ca^{2+} and Mg^{2+} ionic are transmitted in between as well, while in Figure 2-14B, a comparison between biological recordings and FPGA-based simulation of NMDA and AMPA gated ion channel currents is displayed. It shows that the presented silicon AMPA and NMDA gated ion channels can exactly reproduce real synaptic ion channel excitatory current behaviours.

More importantly, a component-based approach was developed to implement neural maths division and exponentiation in the architecture. This technique

utilizes digital logics to achieve factor approach iterative calculations. Hence, the system is benefits from the limited hardware resource utilization and adaptive model parameters. For example, a division is introduced in this section as a case study.

Division format (Y/X) can be rephrased as a communal calculation, and the communal element is a unique case in which Y = 1. Therefore, division functionalities can be recomputed as multiplications of different factors as shown in Equation 2-2:

$$\frac{Y}{X} = Y * \prod_{j=1}^{n} (1 + s_j 2^{-j}) \qquad \text{Equation 2-2}$$

where $p_j = \prod_{j=1}^{n} (1 + s_j 2^{-j})$; after several computing iterations, the mutual part p_j eventually converges to Y/X and $p_j * X$ converges to one. The details factoring the algorithm for division and hardware architecture are shown in Figure 2-15.

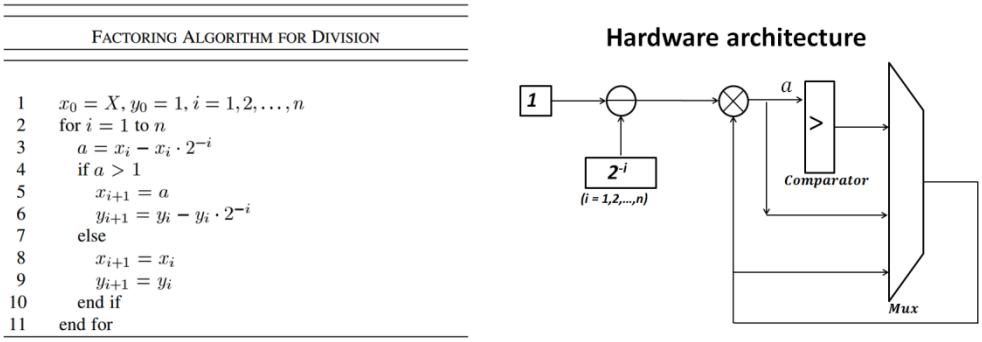


Figure 2-15: The conceptual algorithm and hardware architecture of factoring algorithm for division. The factoring algorithm for division is cited by [37].

B. *Rhythm generation in the Pre-Bötzinger Complex (PBC)*

By using an auto-generation tool kit [17], a silicon Pre-Bötzinger Complex (PBC) network is generated based on a FPGA circuit. Implemented PBC that contains 40 oscillatory bursting neurons with specific synaptic connections aims to explore the respiratory rhythm generation in mammals [41]. The hardware simulation results are shown in Figure 2-16.

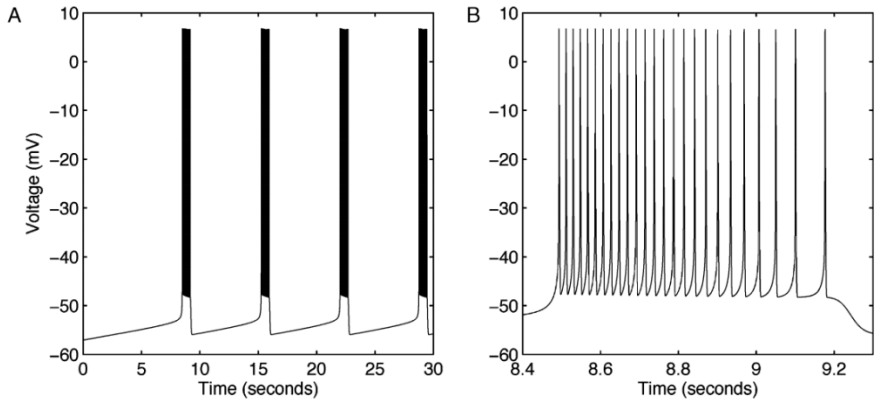


Figure 2-16: The PBC network output patterns. A displays the oscillatory burst patterns in 30s, while B shows the first burst pattern details of four bursts in A. The simulation is based on the single clock-cycled mode with 0.01 time step. The figure is cited in [17].

The auto-generation tool kit significantly reduced the system modification design periods, since the hardware architecture is explicitly classified into two parts: memory blocks and data path. Memory system is based on the shared memory technique, which can be modified in real-time simulation. Adjust data path (adding/deleting a specific ion channel) is also straightforward because the computational component is entirely distinct from memory components. The conceptual structure of the auto-generation tool kit is shown in Figure 2-17.

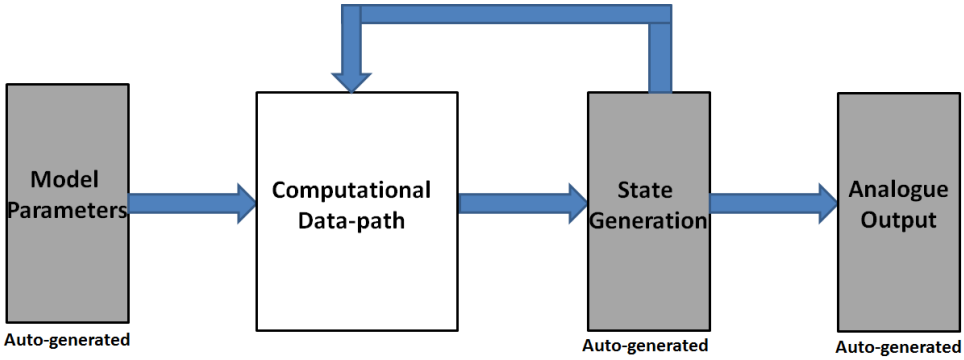


Figure 2-17: The conceptual structure of the auto-generation tool kit. Two main modules are involved in the system: memory-based component (model parameters and state generation) and computational component (data path). The figure is adapted in the work [17].

C. Spike Time-Dependent Plasticity (STDP) learning rule

Spike Time-Dependent Plasticity (STDP) [42] basically reflects the synaptic connection strength dynamic variations between two neurons in a biological

network. In summary, synapse, which contributes to an output spike event generation, will be strengthened and inversely reduced. Specifically, it refers to long-term potential (LTP) and long-term depression (LTD). The work in [16] effectively mimics such an important biological communication feature by using an FPGA. The corresponding architecture and STDP behaviour are shown in Figure 2-18:

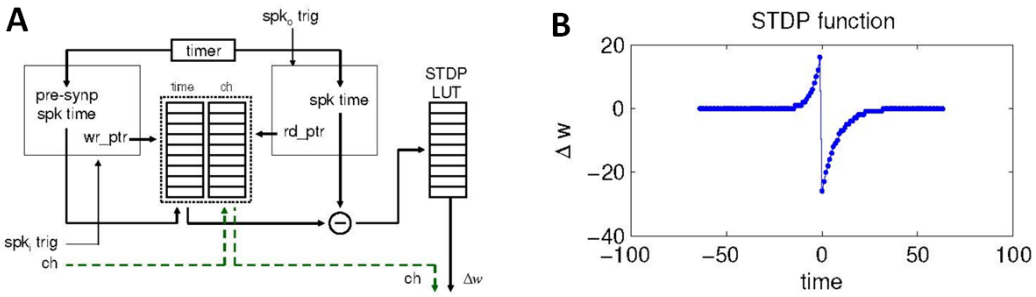


Figure 2-18: The partial hardware architecture of STDP (A) and STDP modification function (B). The figure is cited in [16].

where *spk$_i$ trig* is the pre-synaptic event and *spk$_o$ trig* is the postsynaptic event. Two buffers are employed to temporally stock pre-synaptic event time and synaptic index. The STDP functionality values are pre-stored by using LUT technology. The differences between pre-synaptic and postsynaptic event timing are the index address for value *w*, which is for updating the current synapse strength.

A developed FPGA-based silicon neural array, which contains 32 Leaky Integrate-and-Fire (LIF) neurons, is implemented on a Xilinx Spartan XC3S1500, which utilized 745 slices of FF and 4-LUTs aapproximately 11 2 KB RAMs. The system clock frequency is 50 MHz.

The conceptual mathematical equations and architecture of LUT technology are displayed in Equation 2-3 – Equation 2-4 and Figure 2-19.

$$addr(x) = \frac{(x - \min(x))}{\Delta x} \qquad \text{Equation 2-3}$$

$$ROM_{dep} = \frac{\max(x) - \min(x)}{\Delta x} \qquad \text{Equation 2-4}$$

where x is the input, and $\min(x)$ and $\max(x)$ are the minimum and maximum values of input ranges, respectively. Δx is the resolution step, ROM_{dep} is the

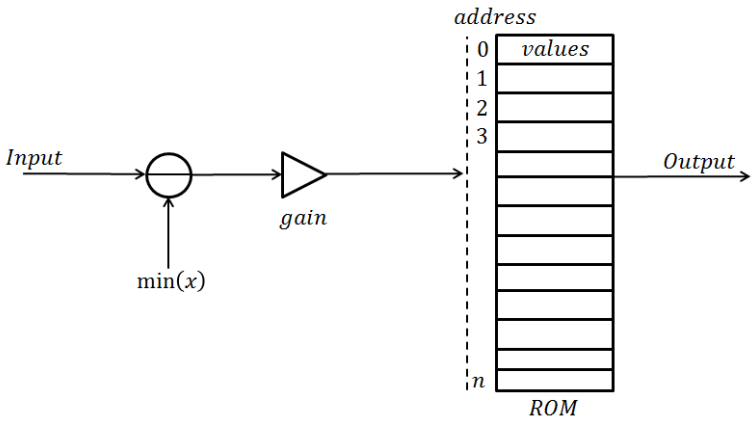


Figure 2-19: Conceptual architecture of LUT approach.

depth of ROM memory block. The basic LUT technique is applied on the system. First, an input value is compared with the base value $\min(x)$ whose ROM address is 0. Then, by multiplying the difference gain (resolution), the correct address is calculated for the output. Equation 2-3 deduces the architecture calculation clock cycles while Equation 2-4 decides the hardware memory resource utilization. Therefore, it is not adequate for models with large-range parameters and accuracy resolutions.

D. Address-event representation for mapping synaptic connection

Inspired by multiplexing methodology applied in telecommunications and computer networks [43], neuromorphic engineering has adopted the Address-Event Representation (AER) technique, which is an asynchronous handshaking protocol used to transmit signals between neuromorphic systems. The basic concept of the AER [44] is displayed in Figure 2-20. It can be simplified as a protocol for data transmissions among many simulation multiplexing-based process cores. For example, every time an event (*e.g.* spike)is generated on Chip 1, the address encoder will write a address(corresponding to address encoder index and its own type) onto a common digital transmission bus which is shared by all neuron events. Arbitration circuits ensure that the addresses are sent off sequentially. The AER handshaking protocol is responsible for the sender and receiver respectively writes and read the correct event from the bus only when they are allowed to. Based on the different system performances of throughputs, neural ensembles (described in the next paragraph) and network firing frequency, the protocol has a variety of architecture styles.

Compared to the encoding/decoding functionalities in biology, the retina codes two bits per spike take optic nerve transmits 40 Mb/s[46], which is a thousand times less than traditional imagers that require 40 Gb/s based on the Nyquist rate. In order to describe such a stimulus-driven, fine, spatiotemporal spike event architecture in biology, the concept of neural ensembles is defined as describing neural network events at statistically aspect; the corresponding equation is show in:

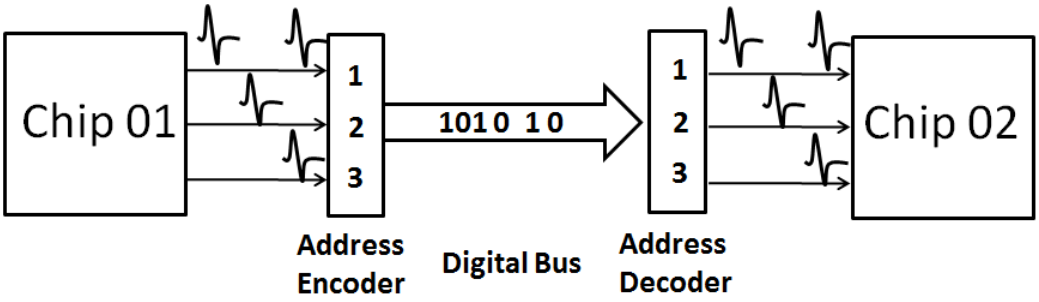


Figure 2-20: The conceptual structure of Address-Event Representation (AER) technique. Time-division multiplexing is applied on neuromorphic chip 01 and 02. The generated spikes are transmitted serially by broadcasting on a digital bus. The figure is adapted from [45]. The address encoder and decoder of 1, 2 and 3 are the timing multiplexed channel index rather than individual spike address.

$$\varepsilon = \{x_0, x_1, \dots\ x_i, \dots\}; t_0 < t_1 < \cdots t_i < \cdots \qquad \text{Equation 2-5}$$

where x_i represents an event that happens at a specific location and time t_i. The developed encoding part is described as an address-event presentation (AER).

Neural ensembles general contain two items: neural latency and neural temporal dispersion. Neural latency refers to the time interval between stimulus onset and spike appeals, while neural temporal dispersion refers to the variation and heterogeneity of individual neurons. By considering the neuromorphic chip signal features within these two indexes, the communication channel architecture has to be carefully designed from capacity, latency temporal dispersion and integrity four aspects. And the design faces several trade-offs as well. For instance, the sampling frequency can be adaptive or static depending

on the signal changing; conflict happens when two spike events attempt to access the same communication channel simultaneously. Either simply abandoning spikes or introducing an arbiter is acceptable for dissolving collision. Also, the time constraint is exposed at spike-event queuing in the arbiter channel, and the solution of arranging new data events versus giving up old data to create new spike-event timings can significantly influence all the system throughputs. In addition, the channel should have the ability to predict the maximum spiking frequency in real time to achieve adaptive communication performances.

In [44] the argued arbitration is the best optimization selection for a multi-core neuromorphic chips system as the spike activity is sparse both in time and space. However, the arbiter channel architecture design is relatively challenging due to the requirements of reliable and robust capabilities, and even asynchronous digital VLSI systems [47] are employed in this research field. An example of AER transmitter and receiver architecture [44] is given in Figure 2-21. The send neuron drives a request to the arbiter via the row-column controller, and the address encoder is also activated at the same time. At the receiver neuron stage, an address (X and Y) is read and latched by activating address decoder, and acknowledged signals are fed back immediately.

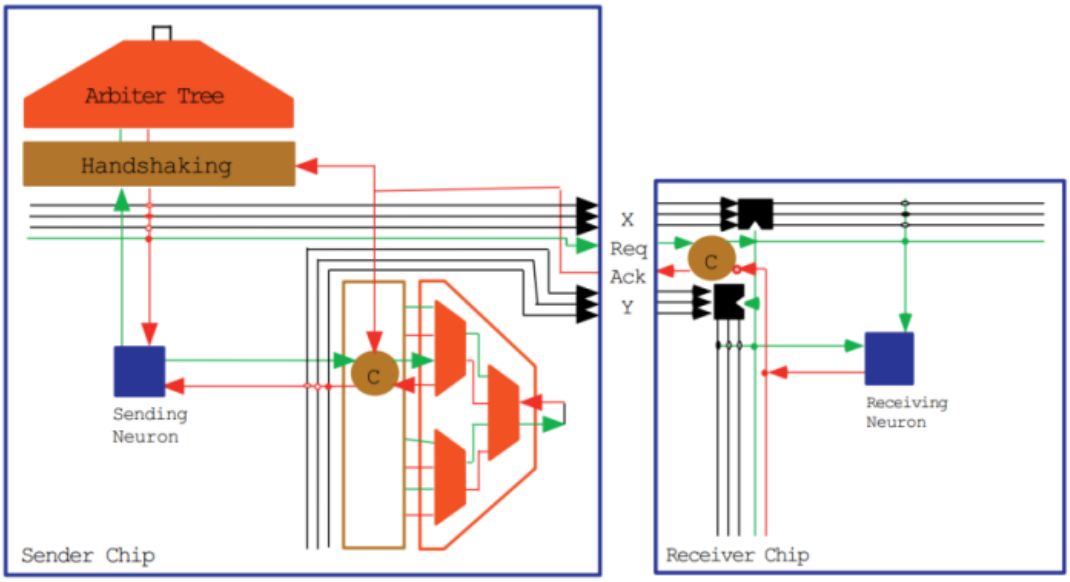


Figure 2-21: Architecture of AER transmitter and receiver. The figure is cited from [44].

2.6 Design conclusions

In this work I focus on exploring high-level hardware architecture to replicate neural network behaviours (from highly bio-plausible to large-scale). The reconfigurable capability and highly parallel computing characteristics of FPGA is a reasonable candidate for this purpose. Furthermore, the other features of precisely timing management and system scalability are capability of meeting the requirements of complex neuroscience experiments. In general, selecting FPGA is motivated by the challenges of large latency by using normal simulation software or multiple-core systems; the significant latency banned the implementation of real-time routine for simulation or Brain Machine Interface (BMI) interaction. Particularly in biological network modelling, simulation of neurons requires strong scalability, from 20 to millions for stomatogastric ganglion [32] to mammal cerebellum [33], respectively. Currently devices such as computers or many-core systems fail to provide such a vital advantage. In addition, customization of a special neural system is important for design, PCs employs standardized software that leads to poor performance and fails to mimic certain behaviour. From the other perspective, analogue-based models take advantage of compact architecture, efficient power consumption, relative cheapness and signals of no loss of information. But the subthreshold analogue CMOS circuit-based large-scale systems are extremely sensitive to divergence of transistor threshold and currents due to the working temperature variations and fabrication issue [16]. More importantly, the fundamental VLSI process variation significantly influences the scalability of the system. Meanwhile, emulating biological complexity with a multi-scale structure and spatiotemporal dynamic on a chip is still a major challenge to engineers [16][17][18].

A brief summary of FPGA features and development trends is presented in Table 2-4. It can clearly be seen that FPGA families have developed rapidly in recent years; the logic cells, block RAM and DSP slices of the Virtex-7 family have increased approximately 100, 7 and 37.5 times compared to the Virtex-4 family. The peak DPS performances and transceiver speed have increased from 48 GMAC/s to 5335 GMAC/s and from 6.5 Gb/s to 28.05 Gb/s. In addition, the package option has enjoyed a smart transition from Pb-Free style to highest performance flip-flop chip. In addition, the physical boards of FPGA Virtex-4,

Virtex-5 and Virtex-7 are displayed in Appendix B, and details of each board application can be found in the following chapters.

Table 2-4: Comparison of series families

Maximum capability	Virtex-4 family	Virtex-5 family	Virtex-7 family
Logic cells	200,448	51,840[1]	1,995,000
Block RAM	9.7 Mb	18 Mb	68 Mb
DSP slices	96	1,056	3,600
Peak DSP performances[2]	48 GMAC/s	580 GMAC/s	5,335 GMAC/s
Transceivers	24	24	96
Peak transceiver speed	6.5 Gb/s	6.6 Gb/s	28.05 Gb/s
I/O pins	960	960	1200
CMOS technology	90 nm	65 nm	28 nm
Package option	Pb-Free	High signal integrity FF	Highest performance FF

1: Virtex-5 slice contains four LUTs and flip-flops (previously it was two LUTs and flip-flops(FF)).

2: Peak DSP performance equals number of DPS slices multiple clock frequency.

2.7 Related biological principles

Three different biological mechanisms are selected for the hardware implementation in the following: the optogenetic technique, the Passager-of-Time(POT) cerebellum model and the stomatogastric ganglion Central Pattern Generators (CPG) of crabs.

First, the optogenetic technique is to use blue light to monitor and control single neuron activities. The basic mechanism is as follows: first, a light-sensitive protein such as channelrhodopsin is obtained from algae; this protein is an ion channel that opens in response to blue light. Then the gene of this protein is taken and the DNA is inserted into specific neurons. After that, the target neurons can be controlled by a flashing blue light. Since the advantages of without damage neurons, and little influence on the other neurons of optogenetic technique, it is widely accepted and acknowledged for different applications such as curing neurologic disorders and fundamental neuroscience research.

The mechanism of optogenetic technique can be described by using the Hodgkin-Huxley (HH) model combined with a four state channelrhodopsin 2 model. An HH model is to accurately describe how action potentials are

initialized and generated on the neuron membrane, which basically consists of three different ion channels: potassium, sodium and leakage. All ion channel currents are calculated by their resting potential, channel conductance and gate activities. While a four-state ChR2 model is to illustrate how photo-current is generated by using two dark and two light adapted states of a single ion channel. Hence, based on these two fundamental components, by studying the photo kinetics of hippocampal cells expressing ChR2, the dynamics of the ChR2-evoked spikes and light sensitivity and efficiency of a new ChR2 version can be achieved.

Second, people precisely timing and fine movement control are decided by biological cerebellum. A POT mechanism is to explain how cerebellum neurons represent the passage of time over a range of tens to hundreds of milliseconds, which fundamentally is for organising movements of different body parts into a coordinated action; it contains approximately 100,000 neurons with random recurrent connections. It can successfully reproduce the classic Pavlovian delay eyeblink conditioning.

In detail the POT model has two types of neuron, one is the granule and the other one is the Golgi cell. It is a virtual sheet composed of a square lattice arrangement. In this model, two requirements are necessary for representing timing information over a dynamic population of active granule cells: 1) long temporal integration of cell ion channels; 2) random recurrent connections from Golgi to granule cells.

Third, the stomatogastric ganglion (STG) system is one of the most identified neural networks since all neuron functions and their connections are well analysed. It is responsible for crab stomach activities such as digesting and transporting food. It contains gastric and pyloric CPGs. Specifically, it is suitable for neuroprosthesis experiments and neuronal-machine system investigation: 1) it still generates fictive motor patterns when removed from the animal and placed in a saline-filled dish; 2) the neurons in the CPG are motor neurons as well without interneurons as connections; 3) individual neuron signals can be well identified and recorded; 4) the CPG in vitro can active for 18-24 hours and can be sustained for weeks if required.

Chapter 3 The Digital Optoelectronic Neuron

This chapter addresses one of the most important design aspects in the digital neural circuit field: system bio-plausibility. A novel architecture is presented to implement the different ion channel-type process dynamics, including calcium feedback mechanisms and optogenetic behaviours. Compared to the previous implementations that can only mimic voltage-dependent ion channel behaviours, a developed system is capable of reproducing not only electricity-related but also chemistry-related behaviours. This significantly improves hardware bio-realistic performances. The operation per 1 ms in a neuron can be achieved up to 76618, which is roughly five times faster than the latest neural design architecture. In summary, the brief design conclusion can be drawn that the architecture should be heterogeneous or multiple-layer based with precise latency management mechanisms to capture the various ion channel-type process details.

3.1 Introduction

A key goal in the neural engineering field is to create real-time operating models of the nervous system. Such implementations may increase the understanding of future computational systems, and provide a better understanding of biological neural process systems [4][32]. These in turn may lead to advanced neuroprosthesis [48][49]. Furthermore, many major diseases of the nervous system relate to channel dysfunctions [50]. Thus advanced neural models that include ion channel functionality could potentially support drug discovery and the burgeoning field of optogenetic and chemogenetic neural systems.

Electronic neural networks come in multiple forms from simple abstract implementations that allow large-scale processing to detailed models that allow accurate representation of ionic flow within neurons. The majority of the effort, including previous work [51], has focused on scalability to large-scale networks utilizing, for example, the integrate-and-fire model [52], the Izhikevich model [53] and the Hindmarsh-Rose model[54]. These models are idealy for implementing large neural circuits, but lack fine detail. In this work, I am interested in exploring the effects of multi-ion channel types on the network. Thus, while I want to develop real-time bio-realistic neurons, I wish to do so with models that can simulate realworld effects of individual ion channels.

The ion channel model of the neuron was developed in 1952 by Hodgkin and Huxley [14], who worked on marine invertebrates. This was later updated for mammals by Traub [55] in the 1990s, which are more specific to real-time implementation. Since 2004, hardware implementation of ion channel models has attracted the attention of many researchers. Graas et al [15] developed a field-programmable gated array (FPGA) framework for implementing conductance-based neuron models.This was the first time a hardware process was used to reproduce HH-based ion channel activities . In 2007, Weinstein et al [17] demonstrated a 40-neuron Hodgkin-Huxley (HH) population model utilizing an auto-generation tool kit. Meanwhile, A. Cassidy et al [16] presented a digital spiking array (32 neurons) that can reproduce synaptic plasticity. In 2012, Coapes et al [56] also developed a scalable FPGA-based design that could simulate large-scale ion channels utilizing HH modes.

However, in a general real neuron computational process, the potential membrane variations will change the concentration of ions such as calcium in

and out of the neuron. This indicates that the corresponding calcium ion channels will alter the gate behaviours. As a result of this, the final neuron spiking patterns will also be reformed, and in turn to shape the ions concentrations again. This is a closed-loop process mechanism related to both electricity and chemistry interactions. Such a complicated state process requires a heterogeneous and multiple-layer-based architecture to reproduce the entire spiking pattern dynamics. Previous implemented digital neural systems show limitations since they can only reproduce membrane voltage-related spiking patterns and ignore ion concentration chemistry-related behaviours.

In this work I have created architecture for implementing a bio-realistic neuron that incorporates 13 different types of ion channels and a calcium feedback process mechanism as shown in Figure 3-1. The advances include the implementation of a channelrhodopsin (ChR2) model [57] onto a digital platform, together with a multitude of calcium-dependent and independent ion channels. These latter channels are derived from biological data from the ion channels of a crustacean (crab) [58]. The developed architecture has the ability to successfully replicate the entire ion channel dynamics in a neuron, including calcium ion concentration and the feedback mechanisms.

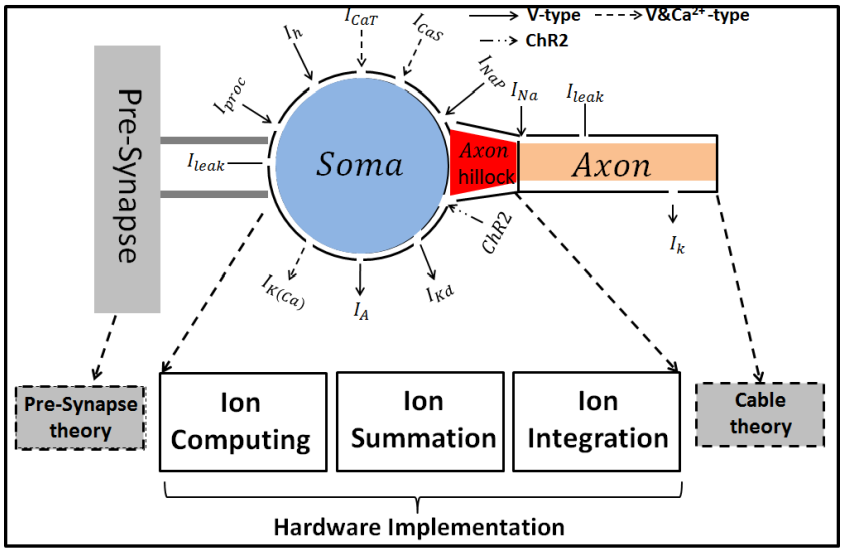


Figure 3-1: An optoelectronic neuron architecture. It contains 12 ion channels in total: a delayed-rectifier I_{Kd} [63], a transient potassium current I_A [64], a persistent sodium current I_{Nap} [65][66], a fast sodium I_{Na}, a potassium current I_K [67], a hyperpolarization-activated inward current I_h [68], a descending

modulatory input current I_{proc} [69], a calcium-dependent I_{KCa}[70], a transient I_{CaT}[71], a persistent calcium current I_{Cas}[71] and ChR2.

The implemented digital neuron has two key advances: the first is that the digital neuron contains the artificial ion channel ChR2 that is directly related to the optogenetics field. Optogenetics is an exciting technique that monitors and controls neuron activities by using light [59]. Before that, the neuron is genetically sensitized to light by using optogenetic actuators such as ChR2 [60]. Therefore, the presented digital neurons can be considered a novel tool for simulation brain network with optogenetic behaviours in biologically real time.

In particular, the biological ChR2 [61] originates from chlamydomonas reinhardtii algae but can be genetically inserted into nerve cells to allow optical control of their electrical potential. Its conductivity varies with ion size but is of the order of pS [62]. Nevertheless, with sufficient activation, it is possible to stimulate neural activity. Although in simple terms it can be considered an optical switch, an optimal biophysical strategy in terms of accuracy and complexity is to utilize a four-state model developed by Nikolic et al [57]. These consist of light and dark-adapted ON and OFF states. The light-adapted ON state is less efficient than the dark-adapted version, giving a non-linear response profile to light. Thus the discussed ChR2 ion channel model is an ideal candidate for implementation.

The second key aspect to this work is that traditional HH models [14] have looked primarily at the three key ion channels in the mammalian nervous system [55]: Cl^+, Na^+ and K^+. However, there are many processes in cells that are mediated by calcium. Furthermore, the advanced CatCh version of ChR2 uses a calcium feedback. It would therefore be useful to have an arsenal of channel variants to explore calcium feedback and the effect of pharmaceutical agents or neurotransmitters on ion channels and receptors. As such, crustaceans are very interesting. They have 12 ion channels, nine of which are voltage dependent and three of which are both voltage and calcium dependent. As I have their characteristics, I can create an implementation model that can incorporate these additional channels at will in addition to the standard Cl^+, Na^+ and K^+.

Although the creation of a MatLab model may be interesting in its own right, I have additionally created a digital processing platform that can explore networks of these neural models in real time. Specifically, I have utilized a Field-Programmable Gated Array (FPGA) to achieve the implementation. This allows scalability for not only closed-loop neuroscience experiments but also prosthetic applications.

3.2 Methods

The methods contain two sections: in the first section a bio-plausibility HH-based neuron model that contains 13 different ion channel types is presented. In the second section I developed a novel multi-loop process architecture for implementing the presented neuron model in a Field Programme Gated Array (FPGA) to achieve biologically real-time computing.

The original HH equations accurately describe three ion channel sodium, potassium and leakage dynamic activities in a neuron, and explain the process of how action potentials initialize and generate. These equations are developed based on biological experiment voltage clamp recordings and successfully predict how ion channel conductance varies. Meanwhile, the ChR2 four-state model I employed is presented by Konstantin et al [57] and can precisely mimic ChR2 current decay dynamics under voltage clamp conditions.

3.2.1 *Ion channel mathematical relations*

The implemented ion channels are listed as below:

- Voltage dependent ion channels: a delayed-rectifier I_{Kd} [63], a transient potassium current I_A [64], a persistent sodium current I_{Nap} [65][66], a fast sodium I_{Na}, a potassium current I_K [67], a hyperpolarization-activated inward current I_h [68] and a descending modulatory input current I_{proc} [69].
- Voltage & calcium-dependent ion channels: a calcium-dependent I_{KCa} [70], a transient I_{CaT} [71] and a persistent calcium current I_{CaS} [71].
- ChR2.

The mathematical equations for voltage-dependent ion channels [58] are given in Equation 3-1 – Equation 3-3:

$$I_i = g_i * m_i^p h_i^q * (v - E_i) \quad \text{Equation 3-1}$$

$$dm = ((m_\infty - m)/m_\tau \quad \text{Equation 3-2}$$

$$dh = ((h_\infty - h)/h_\tau \quad \text{Equation 3-3}$$

where I_i is the ion channel current, g_i is the ion conductance, m and h are gate variables, v is the membrane potential and E_i is the resting potential. $m(h)_\infty$ and $m(h)_\tau$ are gate variable steady-state value and time constant. The basic ion channel model circuit is shown at Figure 3-2: voltage-dependent and leakage ion channels are represented by non-linear conductance g and resting potential E; and ion pumps and exchanges are represented by current sources I_p.

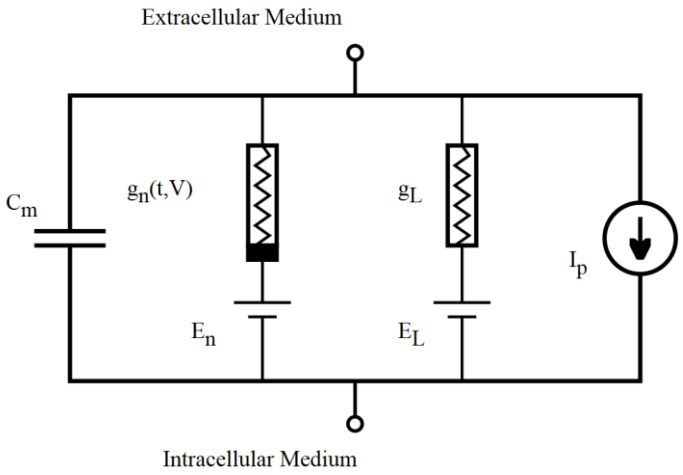


Figure 3-2: The basic circuit diagram of ion channel model.

An additional mathematical equation for calcium-dependent ion channels is given in Equation 3-4 and Equation 3-5:

$$\tau_{Ca} \frac{d[Ca^{2+}]}{dt} = -FI_{Ca} - [Ca^{2+}] + C_o \quad \text{Equation 3-4}$$

$$E_{Ca} = (\frac{R * T}{z * F} * \log(\frac{CaConcOut}{Ca^{2+}})) \quad \text{Equation 3-5}$$

where τ_{Ca} is the Ca^{2+} time constant, C_o is the resting Ca^{2+} intracellular concentration, and the parameter F responsible for translating Ca^{2+} related current into Ca^{2+} concentration. The reversal potential E_{Ca} is calculated by using Equation 3-6. It is based on the Nernst equation, where R is the ideal gas

constant, T is the temperature in kelvin, F is the Faraday constant (coulombs per mole), and z is the number of moles of electrons transferred in the cell reaction or half-reaction; here I am assuming the extracellular concentration $CaConcOut$ is 13 mM [72]. And the corresponding parameters of the above ion channels are shown in Table 3-1 to Table 3-3.

Table 3-1 Parameter values of voltage and voltage & calcium-dependent ion channels

V	I_{Na}	I_K	I_{Leak}	I_h	I_K	I_{NaP}	I_A	I_{proc}
$g\ \mu s$	300	52.5	0.0018	0.054	1890	2.7	200	570
$E\ mV$	50	-80	-60	-20	-80	-50	-80	0
V&Ca	I_{CaT}	I_{CaS}	I_{KCa}		[Ca]	τ_{Ca}	F	C_o
$g\ \mu s$	55.2	9	6		570	303ms	0.418 $\mu M/nA$	0.5 μM
$E\ mV$	0	0	-80		0	-50		

Table 3-2: Parameter values of resting potential Nernst equation

$CaConcOut$	R	T	z	F
13000	8314.47215	273.15 + 10.919	$z = 2$	96485.3399

*: $if\ Ca^{2+} < Co,\ Ca^{2+} = Co$

Table 3-3 Voltage and calcium dependency for the steady-state activation and inactivation of the currents

	m, h	x_∞			τ_x			
	$Sytles$	$\frac{1}{1+\exp(\frac{cv-d}{b})}$			$a-\frac{e}{1+\exp(\frac{-v-d}{b})}$			
		b	c	d	a	b	d	e
I_{Na}^{+}	m^3	5.29	-1	24.7	1.32	25	120	1.26

	h	5.18	1	-48.9	$\left\{\frac{0.67}{1+exp\left(\frac{-v-62.9}{10}\right)}\right\} \times \left\{1.5 + \frac{1}{1+exp(\frac{v+34.9}{3.6})}\right\}$			
I_{CaT}	m^3	7.2	-1	25	55	17	58	49.5
	h	7	1	36	87.5	16.9	50	75
I_{Nap}	m^3	8.5	-1	22	16	26.4	25.1	13.1
	h	4.8	1	48.5	666	11.7	33.6	379
I_h	m	6	1	70	272	8.74	42.2	-1499
I_K	m^4	11.8	-1	14.2	7.2	19.2	28.3	6.4
I_{KCa}	m^4	$(\frac{[Ca]}{[Ca]+30})\frac{1}{1+\exp(\frac{-v-14.2}{11.8})}$			90.3	22.7	46	75.09
I_A	m^3	8.7	-1	27	11.6	15.2	32.9	10.4
	h	4.9	1	56.9	38.6	26.5	38.9	29.2
I_{proc}	m	3.05	-1	12	0.5			
I_{CaS}	m^3	22	-1	8.5	16	26.4	25.1	13.1

Since parameters in tables are estimated based on biological experiment recordings, which are under seawater temperature of approximately 12 degrees, they have to be updated when applied to mammalian animal systems. The corresponding temperature correction equations [55] are shown in Equation 3-7 and Equation 3-8:

$$Q = 3^{\frac{temperature-36}{10}} \quad \text{Equation 3-7}$$

$$am(h) = \beta Q \quad \text{Equation 3-8}$$

where temperature is the system environment temperature, Q is the total energy number, β is an amplified constant depending on different ion channels, and am(h) is the ion channel activations (inactivation).

Also, a four-state model of channelrhodopsin, which has an optimal structure in terms of accuracy and simplicity, was previously described by Grossman et al

[73] and Nikolic et al [57]. The model describes channelrhodopsin as having four states: two dark states and two activated states. The retinal molecular core of the ChR2 ion channel absorbs a photon switching from all-trans to 13-cis-retinal. This induces the channel to switch from a dark-adapted OFF state [C1] to a dark-adapted ON state [O1]. If illuminated in the ON state there is a finite probability of further photon absorption. This would switch the ChR2 from a dark-adapted ON state to a less conductive light-adapted ON state [O2]. From there it may thermally switch back to [O1] or decay to the light-adapted OFF state [C2]. The [C2] state slowly (in the order of seconds) reverts to the [C1] state by thermal means. These relations can be described as four correlated differential equations:

$$\frac{\mathrm{dO1}}{\mathrm{dt}} = \mathrm{G_{a1}(t)C1} - (\mathrm{G_{d1}} + \mathrm{e_{ct}})\mathrm{O1} + \mathrm{e_{tc}O2} \qquad \text{Equation 3-9}$$

$$\frac{dO2}{dt} = G_{a2}(t)C2 - (G_{d2} + e_{tc})O2 + e_{ct}O1 \qquad \text{Equation 3-10}$$

$$\frac{dC2}{dt} = G_{d2}O2 - (G_{a2}(t) + G_{rd})C2 \qquad \text{Equation 3-11}$$

$$G_a(t) = \varepsilon F\left[1 - \exp\left(-\frac{t}{\tau_{ChR}}\right)\right],\ for\ t < t_{light} \qquad \text{Equation 3-12}$$

$$= \varepsilon F\left[\exp\left(\frac{t - t_{light}}{\tau_{ChR}}\right) - \exp\left(-\frac{t}{\tau_{ChR}}\right)\right], for\ t > t_{light}$$

where $\mathrm{O1}$, $\mathrm{O2}$ and $\mathrm{C2}$ are the numbers of ChR2 molecules in the open states 1 and 2, and closed state2. $\mathrm{G_{d1}}$ and $\mathrm{G_{d2}}$ are the rates of thermal conversion of C2 to C1, and $\mathrm{e_{tc}}$ and $\mathrm{e_{ct}}$ are the rates of transition between O1 and O2 and vice versa. Also, Ga1 and Ga2 are the activation rates for C1 to O1 and C2 to O2. $\mathrm{G_{rd}}$ is the rate of thermal conversion of C2 to C1, F is photons per ChR2 per millisecond. The corresponding parameters are given in Table 3-4.

Table 3-4: Parameters of the ChR2 model

Parameter	τ_{ChR}	ε	$\mathrm{e_{ct}}$	$\mathrm{e_{tc}}$	$\mathrm{G_{d1}}$	$\mathrm{G_{d2}}$	$\mathrm{I_{max}}$
Unit	ms		$\mathrm{ms^{-1}}$	$\mathrm{ms^{-1}}$	$\mathrm{ms^{-1}}$	$\mathrm{ms^{-1}}$	nA
Value	1.3	0.1	0.01	0.02	0.35	0.02	0.2

3.2.2 *Implementation*

The hardware architecture is shown in Figure 3-3; there are four main components: voltage-dependent ion channels, $\mathrm{Ca^{2+}}$ concentration, ChR2 and parameter & control. Voltage-dependent ion channel block responses for calculating ion channel activities that are only dependent on neuronal membrane potentials. $\mathrm{Ca^{2+}}$ concentration is for updating $\mathrm{Ca^{2+}}$ resting potential

based on the input calcium-related currents. The ChR2 block responses for mimicking ChR2 light-gated ion channel behaviours. The parameter & control block focuses on pre-store system parameters and configures architecture data path. Also, there are two external inputs in the system: one is the light pulse specifically for the digital ChR2 block, and the other is the pre-synaptic inputs that come from other neurons. The entire system is based on pipelining technique; in each clock cycle, new parameters and control signals are released to compute specifically the ion channel results.

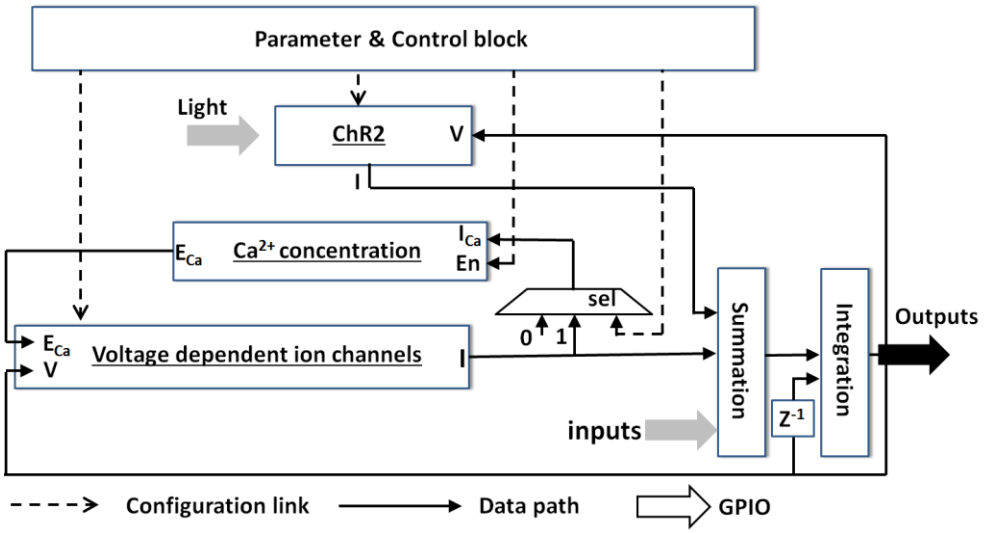


Figure 3-3: The conceptual architecture of a digital neuron. Three signal types are displayed in the system: configuration link, data path and general-purpose input/output (GPIO).

The voltage-dependent ion channel architecture is shown in Figure 3-4. It generally consists of mux, maths operators and custom-designed look-up table (LUT) blocks. The mux components (*e.g.* C1, C2) response for selecting different pre-implemented circuit blocks in specific time periods, and the configuration signals are given by the parameter & control block; maths operators such as gains and multipliers are utilized to perform equation functions, and custom-designed LUT blocks are used to implement complicated math operations such as exponential and division.

There are three stages to computing voltage-dependent ion channels. First, when inputs come in, the function C1 and C2 select the corresponding circuits (∞_v or ∞_{Ca} , τ_v or τ_{con}) to calculate the steady-state activation m and inactivation h values. After the integration process, the function C3 and C4 decidetheir power function and combination styles. Then the calculated values

is done the subtraction with resting potential e_v or e_{Ca}, which is decided by function C5. Finally, the ion channel currents are calculated by multiplying the conductance. All the configuration signals are decided based on implemented models.

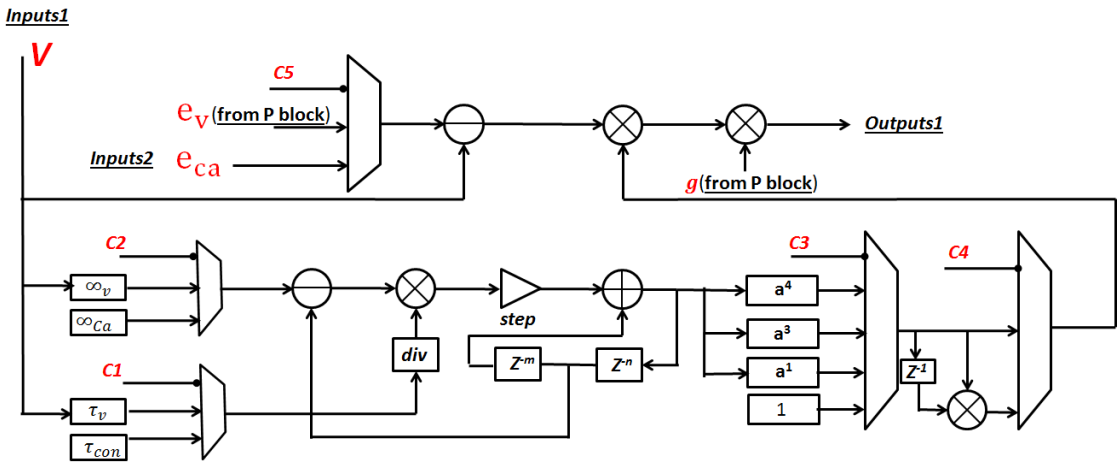


Figure 3-4: A voltage-dependent ion channel block for HH-based ion channel styles. The equations are shown in Equation 3-1 – Equation 3-3. The integration step is optimized at 0.003 ms, and the total delay m+n equals the implemented gate variable ion number.

For calculating voltage & calcium-dependent ion channels, a Ca^{2+} concentration block is added into the system as displayed in Figure 3-3. The main role of the Ca^{2+} concentration block is to update e_{Ca} resting potential values based on input currents I_{CaT} and I_{CaS}. The architecture is shown in Figure 3-5, where τ_{Ca} is the Ca^{2+} time constant, C_o is the resting Ca^{2+} intracellular concentration, and the parameter F is responsible for translating Ca^{2+} related current into Ca^{2+} concentration.

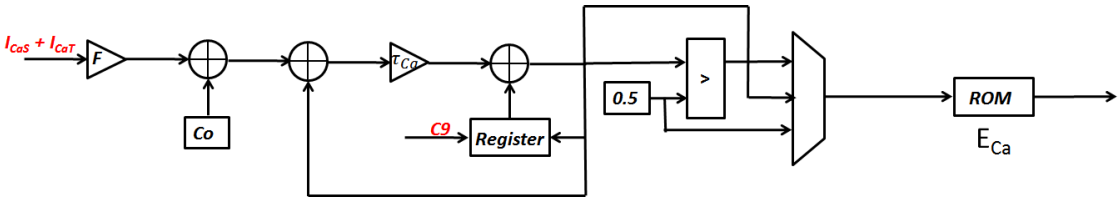


Figure 3-5: A Ca2+ concentration computing block. The mathematical equation is shownin Figure 3-5.

When the system is calculating ion channel I_{CaT} and I_{CaS}, the mux in Figure 3-3 is automatically switched from 0 to 1. This indicates that at this stage the calculated ionic currents will be sent to the Ca^{2+} concentration, and ROM is for translating Ca^{2+} concentration into Ca^{2+} resting potentials. Meanwhile, an

enable signals from control block will be and only active at Ca^{2+} computational block periods.

The data path of ChR2 is shown in Figure 3-6. The left part is for calculating the activation rates for C1 to O1 and C2 to O2. Since the activation rate is different between light on and off, the component mux decides which block is connected to the next stage of computing based on the light duration. The right part is for calculating the ChR2 molecule number and currents. And three differential equations are implemented (Equation 3-7 – Equation 3-10) to perform this task. At each iteration loop, the current ChR2 molecule number, which is stored in the register, will participate in the next stage of the process to update ChR2 outputs. The VHDL code of ChR2 is shown in Appendix C.

Since the pipeline technique is applied to the system to enhance computational speed and save hardware resources, a precise latency management and parameter storage are required, as shown in Figure 3-7. First, the ion channel parameters and configuration information are pre-stored in the different ROM components shown in Figure C; the length of ROM n has to equal the number of gate variables. Then I calculate the latency of data paths in the system, and the exact time periods (*e.g*. a, b and c) for passing data on different data paths can be obtained. In order to maintain the synchronizations between data-path computing and its corresponding parameters, registers are artificially inserted within that ROM based system to mimic data-path computing delays, which are shown in Figure A. Also, the self-counted clock constantly generates output values from 0 to n as the ROM addresses. Hence the systems can accurately compute each ion channel current without data collision.

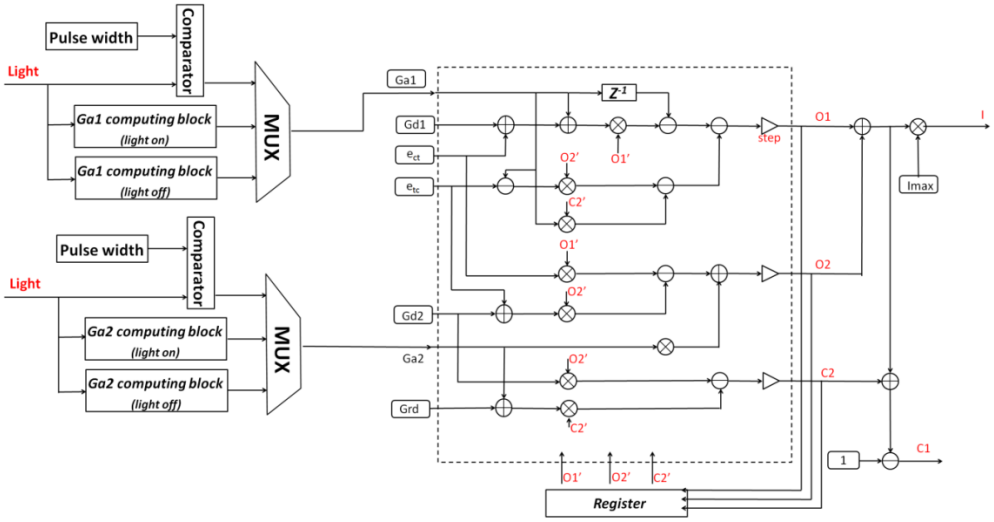


Figure 3-6: Data path of ChR2 computing block. The mathematical equation is shown in Equation 3-7-Equation 3-10.

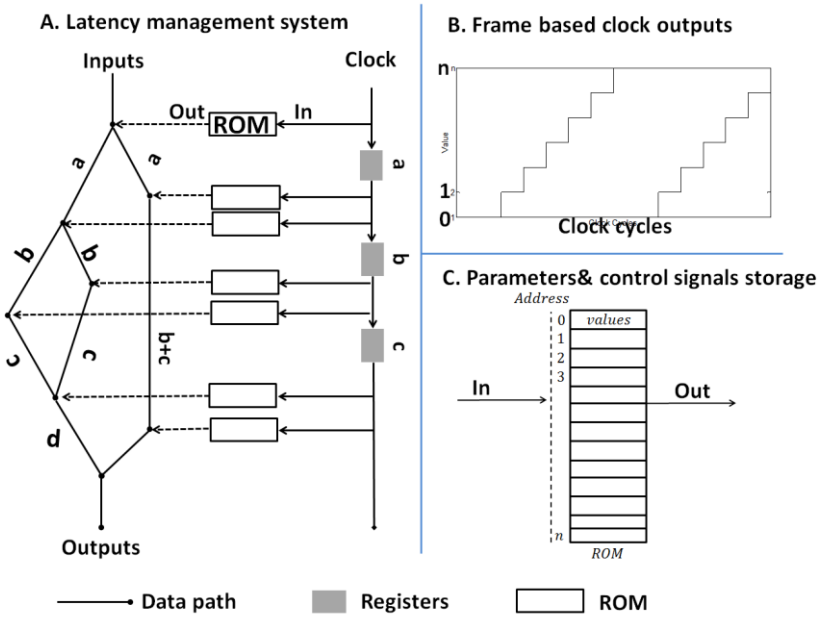


Figure 3-7: System latency management system. A is the latency management system; B is the frame-based clock outputs for addressing ROM; C is the parameters & control signals storage-based ROM.

3.3 Results

3.3.1 *Individual ion channel behaviours*

I showed 12 individual ion channel results of the voltage gated process block in Figure 3-8. The red dashed lines are the FPGA simulation results while the blue solid lines are the software references. The FPGA-based system uses the fixed integration step 0.003 ms while the software system uses variable integration steps. It can be seen that there is a small delay (around 1 us) between them. This is due to the hardware truncation errors and different integration steps in the two systems. From Figure 3-8 I can deduce that different ion channels played specific roles in generating the burst patterns. The ionic currents from axon showed very fast dynamic spikes, while the I_H ion channel from soma displayed slow wave oscillation, and the other ions (I_{CaS}, I_{CaT}, I_{NaP}) generated a slow wave but with tiny fluctuations. In terms of functionality, these ion channels decide the shape of action potential and the firing properties of neurons; H-type current I_H can be served as a function of generating leakage current; potassium currents I_K are responsible for the duration time of burst patterns; persistent sodium current I_{NaP} participates in the function of initialization of neural firing; and transient potassium current I_A maintains the inactivation state almost all the time.

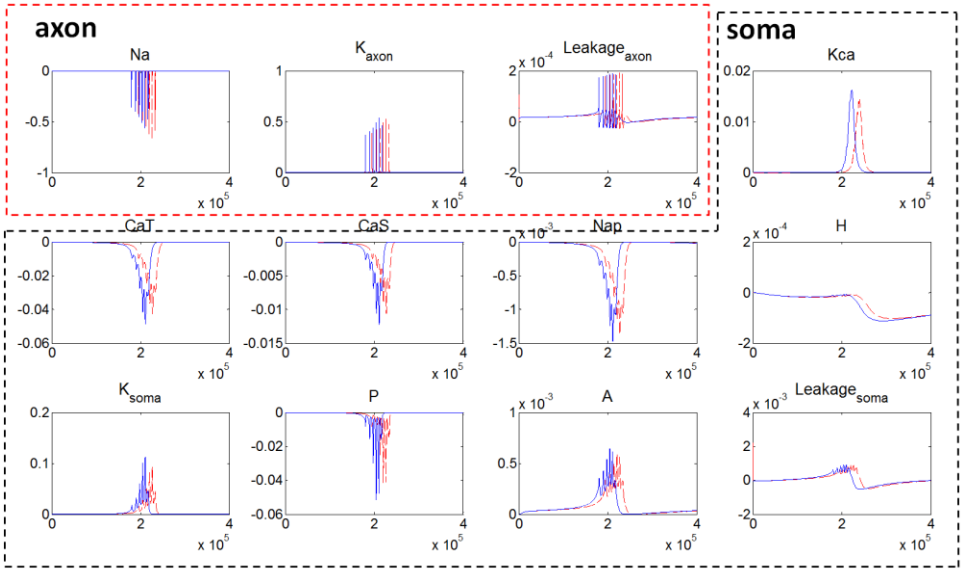


Figure 3-8: Different ion channel dynamic behaviours. The red dashed line is the FPGA simulation results while the blue solid line is the software reference. The Y-axis is the current (mA) and the X-axis the system clock cycles.

It can clearly be seen that FPGA simulation results are identified with biological experiment and software results [57]. This indicates that the developed silicon ChR2 performs the same behaviours as the biological one when the same light pulse is applied. Specifically, when the light pulse is 20 ms, the developed

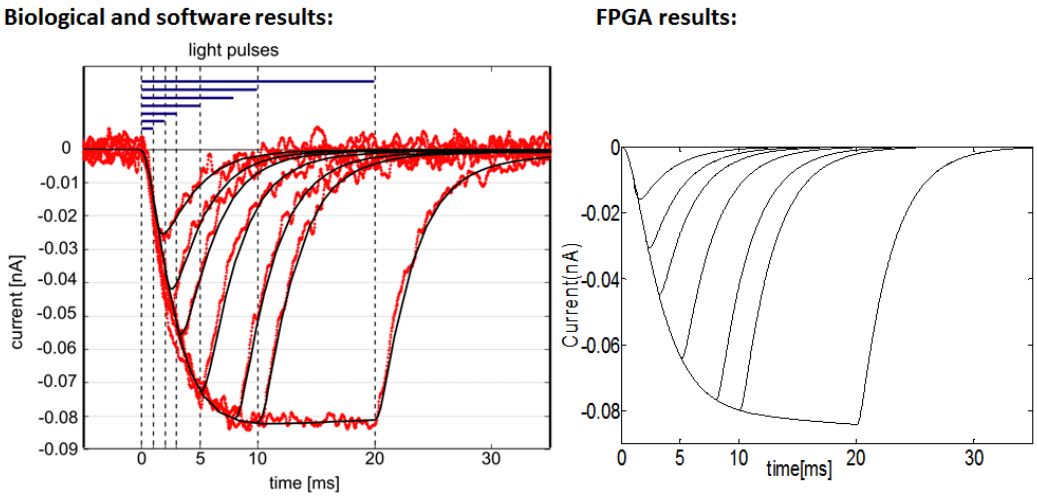


Figure 3-9: The hardware simulation results of ChR2. Comparisons between biological [57] and FPGA simulation results. The short light pulses are 1, 2, 3, 5, 8, 10 and 20 ms. The software fitting parameters are τ_{ChR} = 1.3 ms, γ = 0.1, e_{ct} = 0.01, e_{tc} = 0.02, Gd1 = 0.35 ms^{-1}, Gd2 = 0.02 ms^{-1} and I_{max} = 0.2 nA.

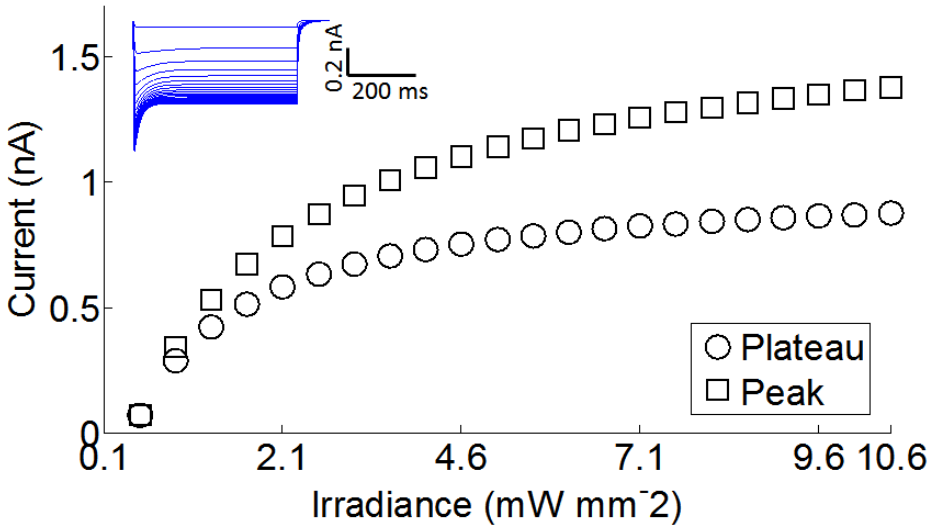


Figure 3-10: By giving different irradiances, the corresponding peak (square) and plateau (cycle) currents are displayed in the figure.

ChR2 displays a saturation situation, and the maximum current that can be generated is 0.08 nA. Furthermore, I quantitatively analyse silicon ChR2 performances with different irradiances as shown in Figure 3-10; when irradiance is larger than 7.1 mW mm^{-2} , both peak and plateau current slow their pace and increase when they approach saturation.

3.3.2 *Mimicking pharmacological performances of crustacean pacemaker*

The anterior burster (AB) of a crustacean is simulated by using voltage-dependent and voltage & calcium-dependent ion channels. The AB is a central pattern generator pacemaker that is responsible for stomach activities such as transport and digestion in crustaceans.

I artificially blocked some specific ion channels of silicon neuron AB to mimic its pharmacological performances. The results are shown in Figure 3-11 and Figure 3-12. It can clearly be seen that in the control condition, silicon neuron AB generated regular and steady burst patterns, which are identical with the software reference. However, in the $\mathrm{K}_{\mathrm{Ca}}^{+}$ ion channel blocked condition, the silicon neuron constantly generated extreme high-frequency spikes rather than burst patterns. This is because the slow wave component of the burst pattern is missed.

Also, I mimicked the other two conditions: in the Na^{+} ion channel blocked condition, the axon part became disabling so these fast spike events disappeared; in the Ca^{2+} ion channel blocked condition, neuron AB became silent and no burst pattern was generated. This indicated that V&Ca^{2+}-type ion channel can directly control pacemaker AB bursting states, which is identified with the relative biological experiment results [74][75].

3.3.3 *Hardware specification*

I show the FPGA-based hardware resource utilizations in Table 3-5**:** Hardware specifications. The system clock periods are all around 20 ns for three different types of ion channel, and the system requires 10^{7} clock cycles to calculate a burst pattern because of the extremely tiny time step 0.003 ms. Hence the presented system uses approximately 0.2 s to mimic 1 s of real-world neuron activities. By applying previous work routing techniques [51], the developed silicon neuron can be scaled up to 20 at a network level. And by applying timing

multiplexing technique, the maximum implemented virtual neuron number can be achieved at around 100 neurons with biological real-time computing

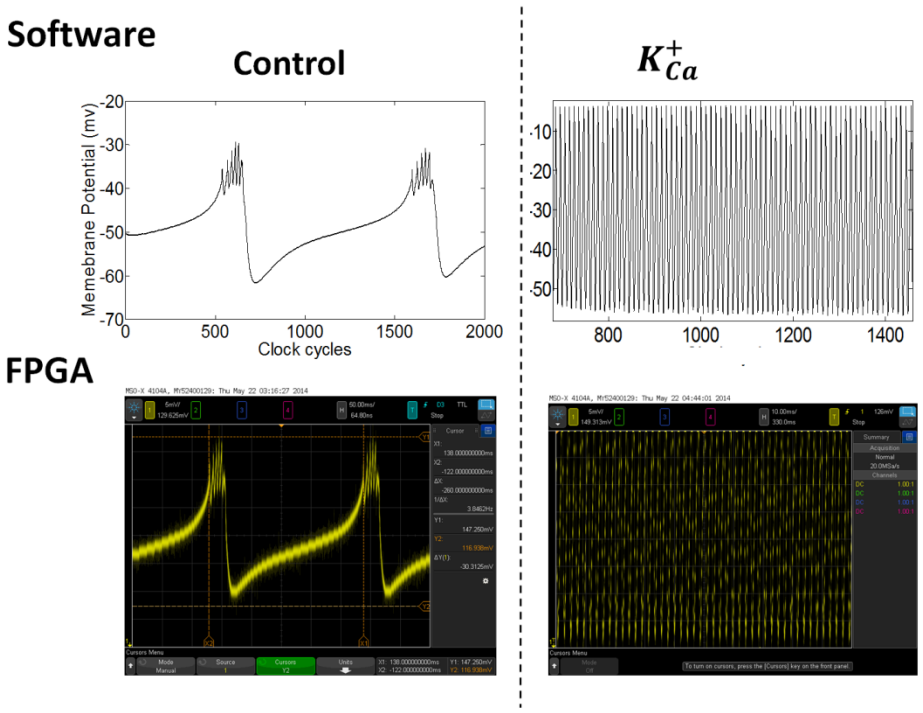


Figure 3-11: Mimicking pharmacological results of FPGA and software. The performances of K_{Ca}^{+} channel blocked and control conditions of pacemaker AB are reproduced.

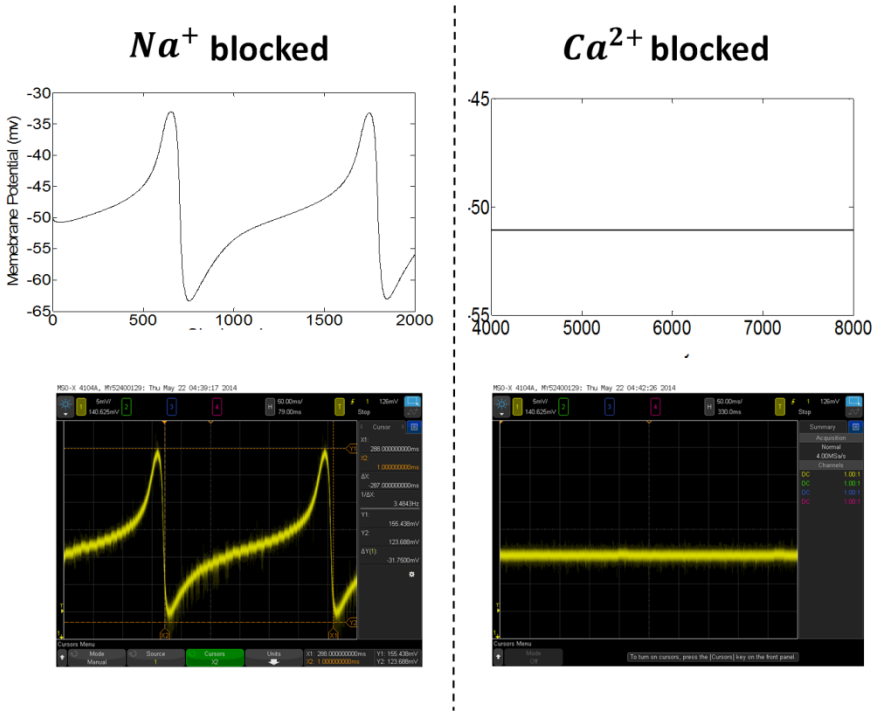


Figure 3-12: Mimicking pharmacological results of FPGA and software. The performances of Na^{+} and Ca^{2+}channel blocked conditions of pacemaker AB are reproduced.

performances. For the memory resources utilization aspect, implementation of V-type and V&Ca^{2+}-type ion channel consumed 26and 43 RAM blocks, 10210and12049 slice LUTs. This is because of the extremely large range (from 0.001 to 1700) of model parameters and many custom-defined mathematical functions; even memory optimization technique was applied to the system. Slice registers are applied to implement digital calculation and control logic. The consumption is around 1500 slices. For the power consumption aspect, V&Ca^{2+}-type, V-type and ChR2 ion used dynamic power of 0.315 w, 0.196 w and 0.203 w, respectively.

Table 3-5**:** Hardware specifications

	V	V&Ca^{2+}	ChR2	Router	Total
Clock periods (ns)	20.781	18.627	22.828	--	22.828
RAM block	26 (2%)	43 (4%)	0	0	43 (4%)
Slice register	1556 (1%)	1998 (1%)	975 (1%)	791 (1%)	3764 (1%)
Slice LUT	10210 (3%)	12049 (3%)	2231 (1%)	1213 (1%)	15493 (5%)
Dynamic power (w)	0.196	0.315	0.203	0.006	0.524
Quiescent power (w)	0.209	0.212	0.210	--	--
Max neurons (physical)	33	25	100	--	20
Max neurons (virtual)	165	125	10000	--	100

3.4 Discussion

3.4.1 *Implementation of different neural models*

In this work I implemented a strong bio-plausible digital neuron incorporating ChR2 in FPGA hardware, and this neuron can be scaled up to a small/medium-size neural network by using timing multiplexing technique. A summary of comparisons of previous FPGA-based neural network modelling is displayed in Table 3-6.

The Izhikevich model has a weak bio-plausibility since it describes the spiking patterns from the mathematical perspectives that lack sufficient ionic process details, while the HH model design is based on recordings of conductivity of membrane potential. Therefore, it has details of each individual ion channel computing dynamic performances. In between, there is a conductance-based integrated and fire (IF) model that has a mediated bio-plausibility, because it

can mimic neuronal integration-and-firing characteristics based on the ion channel computation parts [33].

Due to the different implemented neural models, the system focuses on quite diverse research areas. The simplified Izhikevich model is applied for simulating a large-scale neural network in FPGA [19]. The architecture utilized an event-driven approach with an integrated processor to perform simulationThis can achieve up to 2.48 x real times for running 64,000 neurons. Similarly, the previous work uses a conductance-based IF model to reproduce biological granular-layer (contains 100,000 neurons) passage-of-time functionalities. A frame-based network-on-chip architecture is developed and by using this computational speed, it can achieved up to 39 x real times. However, both architectures are mainly focused on the large population activities and ignore individual neuron action potential generations.

Weinstein et al [17] and G Smaragdos et al [20] utilized the HH and HH extended model to mimic the neural system in a more detailed way. The number of implemented ion channel types in a neuron is four, and this can basically reflect all ionic current dynamics. They contributed a solution for biological real-time simulation of a bio-realistic neuronal network with more than 100 neurons (with 8.7 x real time and 12.5 x C code).

In this work, I increase the ion channel types in a neuron from four up to 13, which significantly increases the FPGA-based neural model bio-plausibility. Compared to the previous work, calcium-related ion channels and ChR2 are first implemented and integrated with voltage-dependent ion channels. Therefore, developed digital neurons can not only mimic standard but also pharmacological spiking-bursting behaviours, and the speed can be up to 5 x real times.

There is an interesting discovery in the comparison table: the more neural model complexity increases, the smaller the system time step is. This is due to the fact that in some ion channel algorithms, the dynamic spike patterns change very rapidly: for example, Ip current in the soma, which generates several spikes in very short periods of 10 us. This requires the system to have a sufficiently accurate time step for simulation. Hence, the developed system time

step is 0.003 ms and can achieve approximately 76618 operations per neuron in 1 ms.

3.4.2 *Implementation tools*

In terms of programming tools, current C and Java languages [76] can be directly applied to hardware architecture design. It enjoys the advantages of easy modification and programming. However, it still lacks flexibility to some extent. The graphic tool system generator is quite popularly utilized in hardware-based neural network modelling since it is very good at digital signal processing [15][17][77][78], but it shows limitations and constraints when routing algorithms or communication protocols face implementation. In this work, I use the system generator for mathematical neural model calculation and VHDL for routing strategy implementation, which has developed an efficient approach for neural modelling in the bio-mimicking society.

3.4.3 *Neuroscience applications*

I can also use this digital neuron for neural rehabilitation. One issue is that I artificially damaged the real pacemaker AB in the pyloric network, and the implemented silicon AB was embedded in the damaged neural network by using a dynamic clamp [78] to restore the original neural network activities. Also, I aim to include sufficient details in the individual neuron models to allow the replication of circuit behaviour dynamics in a wide range of physiologically plausible situations.

In terms of optoelectronic/optogenetic areas, the developed system can be further developed into a processing platform for simulating neural network patterns with ChR2. Biological real-time simulation allows us to investigate the design of efficient optoelectronic devices for neurologic disorders [79]. More importantly, the implemented pacemaker model architecture is identified with a mammal-based HH model [55]; I can directly map a mammal's biological recording parameters on the parameter & control block to mimic the brain network with optogenetic behaviours. Therefore, it serves as a novel reliable simulation tool to verify emerging optogenetic hypotheses and systems [80][81].

3.5 Conclusion

In this section I propose novel hardware architecture for implementing multi-type ion channel models that can capture the finest things in ionic activities.

Compared to previous work, this is the first time voltage-dependent, voltage & calcium-dependent and ChR2 ion channels have been integrated into a single neuron. A silicon pacemaker neuron with ChR2 is successfully implemented on a Virtex-7 FPGA board as a case study. Based on the hardware results, it can not only reproduce normal neural burst patterns but also pharmacological burst patterns. This significantly improved hardware bio-realistic performances and is a new processing platform potentially for ion channel-related mechanism discovery and drug investigation.

Table 3-6: Comparison of other techniques

Model	Izhikevich (64,000) [19]	IF (100,000) [51]	HH (40) [17]	Extended HH (96) [20]	This work (100)
FPGA chip	Virtex6 SX475T	Virtex7 XC7VX485 T	Virtex4 XC4VSX35	Virtex7 XC7VX485T	Virtex7 XC7VX485T
Time step (ms)	1	1	0.01	0.05	0.003
Operations per neuron in 1 ms	>7	30	>1200	22,200	76,618
Real-time speed	2.48 x real time	39 x real time	8.7 x real time	12.5 x C code	5 x real time
Resource utilization	LUTs (199421) FFs (135032) BRAMs (886)	LUTs (268544) FFs (176424) BRAMs (960)	Slices (13,840) DSP (183)	LUTs (251485) FFs (162217) BRAMs (804)	LUTs (294367) FFs (75280) BRAMs (860)
Precision	Fix point	Fix point	Fix point	Floating point	Fix point
Programming tool	Java description	VHDL+ System generator	System generator	C-code	VHDL+ System generator
Novelty	Using event-driven approach and reasonable memory bandwidth	A frame-based network-on-chip architecture without data collision	Auto-generation tool kit	Real-time simulation tool for investigating ION	Real-time simulation tool for optoelectronic/ optogenetic research areas

Chapter 4 The Digital Cerebellum

This chapter, based on the previous chapter's research findings, develops the system from a single heterogeneous structure-based digital neuron to a large-scale neural network. The mouse cerebellum is selected as an ideal study, since it has a massive number of neurons (up to billions) and plays a vital role in animal motor control and balance movement mechanisms. The passage-of-time (POT) cerebellum model is implemented in the new architecture: frame-based network-on-chip. The presented digital cerebellum has approximately 100,000 neurons, and it can successfully reproduce timing memorable performances, which a function to represent the passage-of-time (POT) over a range of tens to hundreds of milliseconds. The system has 48 cores, 48 routers and one frame master. Each core implements 2000 granule cells and 20 Golgi cells with a connection ratio of 100:1. The routers are based on custom-designed address event representation techniques to map random recurrent synaptic connections. And the frame master is to maintain synchronization between cores and routers. At this stage, the design strategy can be described as a pipeline-based multi-core-based architecture with tailor-designed routing technique, which is an efficient system for implementing biological brain networks.

4.1 Introduction

Smooth and robust motor control requires precisely timed muscle activations at specific strengths. This is critically mediated by the cerebellum, which functions to represent the passage-of-time (POT) over a range of tens to hundreds of milliseconds, and is essential for organizing movements of different body parts into a coordinated action [82]. Errors in POT encoding consequent to cerebellar damages can lead to dysmetria or delays in movement onsets in these patients [83]. This condition, usually described as ataxia, cannot be cured completely at the moment, and affects millions of patients worldwide. To foster a potential cure based on neuro-prosthetic technology, an efficient computational platform that can favourably mimic the complex function of the cerebellar neural network is important. Figure 4-1 shows a conceptual closed-loop system for a cerebellar prosthesis.

POT representation in the cerebellum is clearly evident in the classical Pavlovian delayed eyeblink conditioning [84][85], where animals learn the inter-stimulus interval (ISI), or POT, between conditioned (CS) and unconditioned (US) stimulus onsets upon repetitive training. It has been suggested that this information concerning POT is encoded in the extensive cerebellar granular layer. When excited by CS through mossy fibres (MFs), the population of granule cells exhibits different bursting dynamics such that the sequence of active cells does not recur for a sufficiently long time. This forms a one-to-one correspondence between the active cell population and a time interval. Various computational models have been developed to investigate a possible mechanism in the granular layer for POT representation. Four classes of such models have been reviewed in [86], including the delay line model [87][88], spectral timing model [89], oscillator model [90] and random projection model [91][33]. Among these computational models, the random projection model is suggested to be both a robust and biologically plausible framework in the representation of POT, and can also be used to reproduce the classical Pavlovian delay eyeblink conditioning. This spiking network model makes use of two critical properties of the cerebellar granular-Golgi layers: 1) extensive random recurrent connections between granule and Golgi cells; and 2) long temporal integration of input signals by the NMDA receptors, which are both evident in the biological systems.

Thus far, this large-scale (~106 cells) spiking network cerebellum model has been investigated by software simulation using PC and GPU implementation [33][92]. However, in order to use the model in real-time biological experiments, particularly *in vivo*, some form of compact digital real-time implementation with versatile I/Os would prove valuable. A scalable hardware platform that can be tailor-designed and takes advantage of highly parallel computing capability would be greatly preferred. Such a system would be a powerful tool for helping to explore the POT mechanism and related disease mechanisms in the cerebellum. Future neuro-prosthetic developments could also benefit from an efficient hardware platform for implementing a large-scale spiking network model for real-time computation.

In general, CPU-based process platforms are limited by their sequential computing architecture. The large latency makes them difficult to use in real-time Brain Machine Interfaces (BMI). GPUs [93] are capable of parallel computing but are constrained by memory and communication bandwidth issues. Circuits can be implemented directly onto CMOS [94][95], but a single implementation can be time-consuming. Field-Programmable Gate Arrays (FPGAs) are a versatile reconfigurable digital computational platform that can be used for both direct computational implementation and as a stepping stone to compact low-power CMOS chip implementation. It contains massive flexible programmable logic with concurrent high-speed operation, allowing direct use in bench-top *in vitro* and constrained *in vivo* systems. If designs are then translated to CMOS, the subsequent chips can be applied to implantable neuro-prosthetic devices. In recent years, FPGAs have been extensively used in neural system modelling and simulation of large-scale biologically realistic neural systems [77][37][15][17].

Hardware implementations of cerebellar neural networks for neuroprosthesis have already attracted the interest of neuroscientists and engineers. Bamford et al [95] have designed a VLSI field-programmable mixed-signal array to produce eyeblink conditioning performances by modelling the cerebellum system. This has been fabricated as a core on a chip prototype intended for use in an implantable closed-loop prosthetic system aimed at rehabilitation of associated

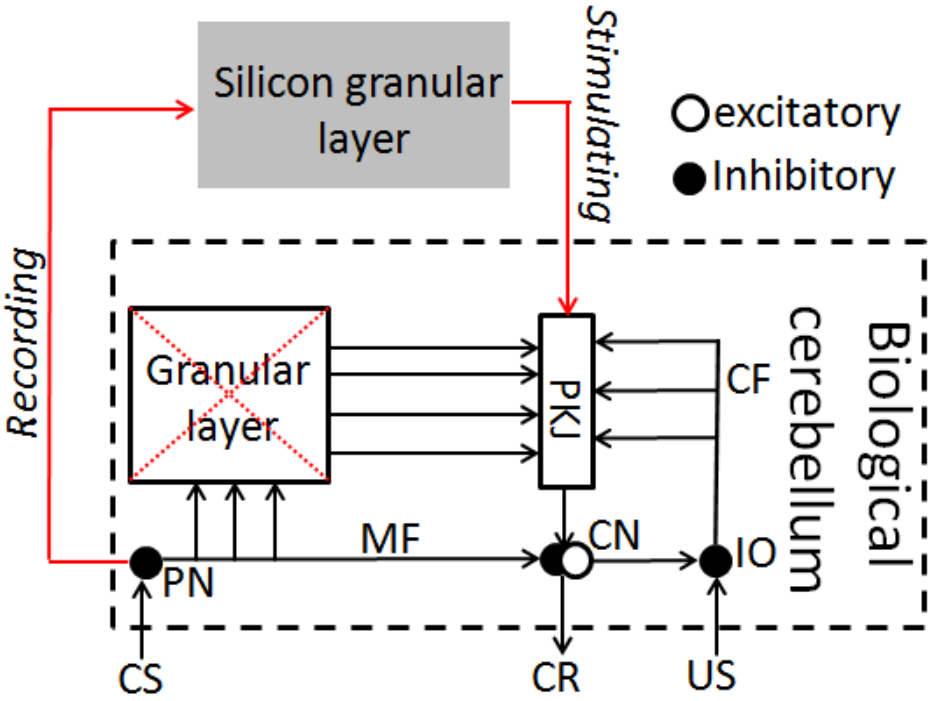


Figure 4-1: Conceptual closed-loop system cerebellum passage-of-time (POT) prosthetic. Damaged biological granular layer is replaced by FPGA-based granular-layer system. CS is a conditional stimulus while US is an unconditional stimulus. MF is the mossy fibre and CF is the climbing fibre. PKJ is the Purkinje cell. The granular layer with a red cross represents a damaged biological one.

behaviour. While they have demonstrated a proof of concept of success in their implementation, a highly simplified neural model with abstract modelling of cerebellar information processing is used in the work. Such simplification is convenient for hardware implementation, but lacks direct physiological correspondence for quantitative comparison with the biological system. In contrast, Yamazaki and Tanaka's model [33] is more biologically realistic and pays specific attention to the role of the granular-Golgi layer in timing and gain control by the cerebellar cortex to reproduce experimental results. However, this comes at the cost of a significant increase in the size and complexity of the computational model in order to produce a robust system behaviour. As such, an efficient implementation is required to overcome these computational challenges, especially when real-time application is required.

Previously [96][51] I presented the concept of an FPGA-based network-on-chip (NoC) hardware architecture for implementing the granular layer of a random projection cerebellum model. It produced a network behaviour of POT representation consistent with the simulation results presented in the original

paper by Yamazaki and Tanaka [33]. In this work I have conducted a more in-depth investigation of the details of system performance implementation and analysis. The system contains ~100,000 granule cells and ~1000 Golgi cells, using a conductance-based, leaky integrate-and-fire neuron model. The parameter values all have an experimental basis, such that the network model produces realistic firing behaviour. In particular, three accomplishments are highlighted in thisthesis: 1). I have reproduced the granular-layer firing patterns for representation of POT in real time under normal as well as pharmacologically perturbed conditions. 2) The architecture allows for efficient scalability to 100,000 neurons and beyond and can be used for more complex biological neural network applications. 3) I have eliminated multiplexing timing errors and allowed for network profiling at key time points.

4.2 The passage-of-time computational model

The cerebellar granular layer consists of two main cell types, namely granule cells and Golgi cells. The input signal from the pre-cerebellar nucleus to the granule cells is conveyed by MFs (Figure 4-1). The spiking network of the cerebellar granular layer developed in [33] is modelled as a 1 mm^2 virtual sheet composed of a square lattice arrangement of 32*32 Golgi cells and glomeruli, and 320*320 granule cells. The same network with minor changes is used in this paper. Figure 2 describes the topology between Golgi and granule cells.

Figure 4-2A illustrates the topology of the granular-layer model, which contains 1024 granule-cell clusters and a Golgi cell. The different colours represent communities of closely connected cells within the network. Each granule-cell cluster contains 100 granule cells. The size of the circles is proportional to the number of other clusters that it is connected to. Each dot represents one granule-cell cluster and one Golgi cell, as shown in Figure 2B. Every Golgi cell receives excitatory input from its nearest granule-cell cluster, while Golgi cells project randomly to the nearby granule-cell clusters such that each granule-cell cluster receives inhibitory inputs from ~8 Golgi cells on average. The probability distribution of number of synaptic connection from Golgi cell to granule-cell cluster is shown in Figure 2C.

The equations for modelling the neurons and analysis have been detailed in [11] and I briefly repeat the key ones here. The granule and Golgi cells were

modelled as conductance-based, leaky integrate-and-fire units, as described in Equation 4-1:

$$C\frac{dV(t)}{dt} = g_{leak}\big(E_{leak} - V(t)\big) + g_{ex:AMPA}(t)\big(E_{ex} - V(t)\big) + g_{ex:NMDA}(t)\big(E_{ex} - V(t)\big) + g_{inh}(t)\big(E_{inh} - V(t)\big) + g_{ahp}(t-\hat{t})\Big(E_{ahp} - V(t)\Big) \quad \text{Equation 4-1}$$

where $\mathrm{V(t)}$ and C are the membrane potential at time t and the capacitance, respectively, E_{leak} are the reversal potential and $\hat{t}$ denotes the last firing time of the neuron. The membrane potential depends on five types of currents: α-amino-3-hydroxy-5-methyl-4-isoxazolepropionic (AMPA) receptor-mediated, N-methyl-D-aspartate (NMDA) receptor-mediated, leak current, inhibition current and the post-hyperpolarization current. The conductance, $\mathrm{g(t)'s}$, is calculated by convolving the alpha function α(t) with the spike event δ_j(t) of presynaptic neuron j at time t as follows:

$$g_c(t) = \bar{g}_c \sum_j w_j \int_{-\infty}^{t} \alpha(t-s)\delta_j(s)ds \quad \text{Equation 4-2}$$

where $\bar{g}_c$ is the maximum conductance and w_j is the synaptic weight of neuron j. A neuron fires a spike (δ_j(t) = 1) when its membrane potential exceeds a threshold θ, and the post-hyperpolarization will follow. The conductance for the post-hyperpolarization was given by:

$$g_{ahp}(t-\hat{t}) = \exp(-(t-\hat{t})/\tau_{ahp} \quad \text{Equation 4-3}$$

I followed the same analysis procedures as in [33] for evaluating the POT behaviour produced by the simulation model. I first computed $z_i(t)$, which represents the average activity of a granule-cell cluster i:

$$z_i(t) = \frac{1}{\tau}\sum_{s=0}^{t} \exp\left(-\frac{t-s}{\tau}\right)\left(\frac{1}{N_{gr}}\sum_j \delta_j(s)\right) \quad \text{Equation 4-4}$$

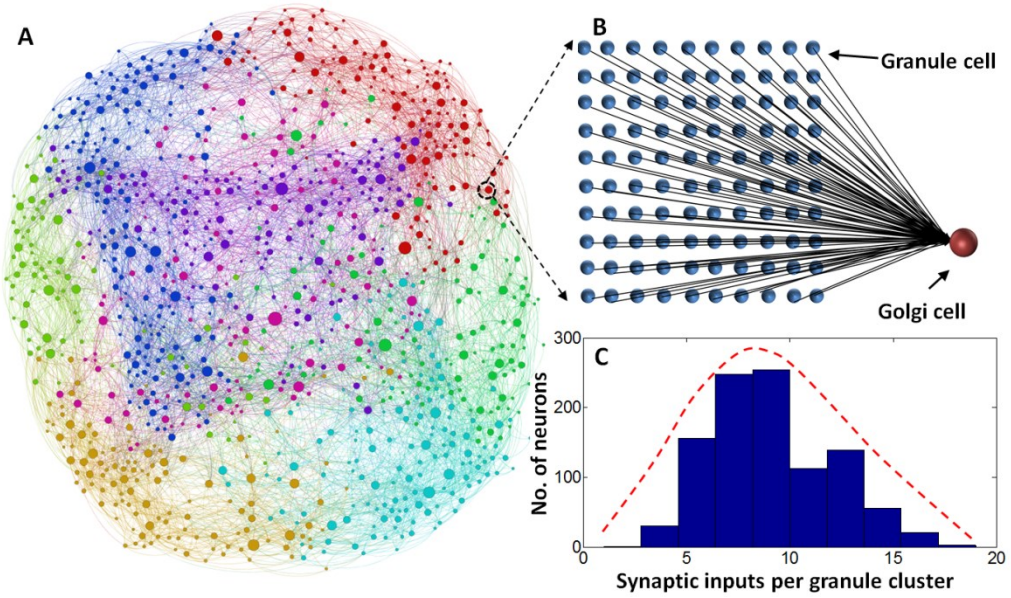


Figure 4-2: Topology of the granular-layer model. Figure A contains 1024 granule-cell clusters and a Golgi cell; the different colours represent communities of closely connected cells within the network. The size of the circles is proportional to the number of other clusters that they are connected to. Each dot represents one granule-cell cluster and one Golgi cell, as shown in Fig. B. The synaptic input number distribution is displayed in Fig. C.

where $\delta_j(s)$ is the spike event in the granule cell j in the cluster at time s, N_{gr} is the number of granule cells in a cluster (100 in this case) and τ is the decay time constant, which was set at 8.3 ms.

How the activity patterns of granule cell clusters evolved over time is evaluated based on the similarity index, $S(\Delta t)$. I first computed the autocorrelation of the activity pattern between time t and t+Δt as follows:

$$C(t, t+\Delta t) = \frac{\sum_i z_i(t) z_i(t+\Delta t)}{\sqrt{\sum_i z_i^2(t)} \sqrt{\sum_i z_i^2(t+\Delta t)}} \qquad \text{Equation 4-5}$$

$C(t, t+\Delta t)$ takes a value between 0 and 1 since $z_i(t)$ is always non-negative. It will be 1 if the activity pattern vectors $z_i(t)$ and $z_i(t+\Delta t)$ are identical, and it will be 0 when they are orthogonal, indicating that the activity patterns have no overlap. Then the similarity index is computed as the timed average of Eq. (5) over the CS duration, T, shown as follows:

$$S(\Delta t) = \frac{1}{T} \sum_{t=0}^{T} C(t, t+\Delta t) \qquad \text{Equation 4-6}$$

$S(\Delta t)$ represents how two activity patterns separated by Δt are correlated, on average. If the similarity index decreases as Δt increases, it indicates that activity patterns have evolved with time into uncorrelated patterns.

I further computed the reproducibility index *R(t)* as follows:

$$(t) = \frac{\sum_i z_i^{(1)}(t) z_i^{(2)}(t)}{\sqrt{\sum_i z_i^{(1)2}(t)} \sqrt{\sum_i z_i^{(2)2}(t)}} \qquad \text{Equation 4-7}$$

where $z_i^{(1)}(t)$ and $z_i^{(2)}(t)$ are the activity patterns of granule-cell cluster i at time t for two different input signals. The reproducibility index quantifies how activity patterns elicited by two different input signals differ from each other over time and serves as a measure for the robustness of the POT representation by the network model.

4.3 Hardware architecture design

To implement the POT model, I developed a frame-based network-on-chip (NoC) hardware architecture on a FPGA. The conceptual structure is shown in Figure 4-3.

In Figure 4-3, the left side shows the *n* by *m* frame-based NoC system, where the size can be adjusted as needed. The architecture consists of three main components: the neural processor, the router and the global controller. In this work, I implemented an NoC system containing 48 processors. The neural processor calculates the neural activates, with each processor implementing 2000 granule cells and 20 Golgi cells with a connection ratio of 100:1. The router is used for implementing the inhibitory connections from Golgi cells to granule clusters. A unicast routing strategy is applied due to the optimization of processing, power consumption and areas. It involves a direct transmission package from the source to the destination, and the package contains both source and destination identifiers. According to the destination identifier information, routers are able to decide the transmitting directions in the network. The interface modules packetize spike events received from the processor ready for transmission through the network. When the interface modules receive packets the message is decoded and transmitted to the required cells within the neural processor. Finally, a frame master is implemented to

coordinate neural and communication processing periods. The details of the hardware architecture (four processors) are displayed in Appendix D*.

4.3.1 *Neural computing*

The neural processor data path is shown in Figure 4-4. Two types of neurons are implemented in the processor: the granule cell (GR) and the Golgi cell (GO). Both models use the same hardware architecture but with different parameters. Each granule cluster, containing 100 granule cells, connects to one Golgi cell. The activities (1 or 0) of all the 100 granule cells will be calculated first, whilst an

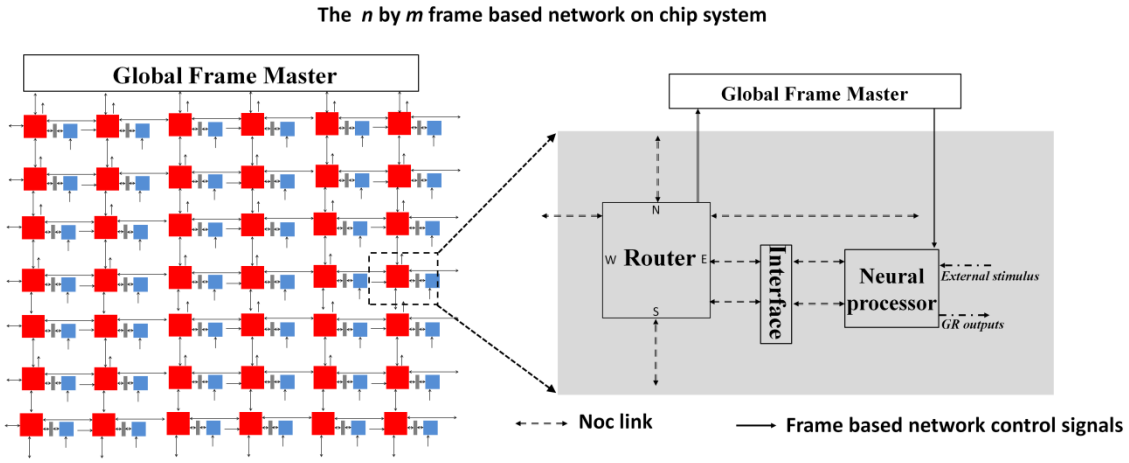


Figure 4-3: A conceptual FPGA-based network-on-chip hardware architecture. The figure on the left is the scalable n by m structure of the frame-based network-on-chip system. It contains n*m neural processors, n*m routers and one global controller. This architecture can be scaled up depending on the required model. In this paper, I implemented a network-on-chip system that contains 48 processors. On the right, there is a detailed structure of a module. The neural processor calculates the neural activity, with each processor implementing 2000 granule cells and 20 Golgi cells with a connection ratio of 100:1. The router is for implementing the connections from Golgi to granule clusters. The interface modules packetize spike events received from the processor ready for transmission through the network. When the interface modules receive packets the message is decoded and transmitted to the required cells within the neural processor. Finally, a frame master is developed to coordinate neural and communication processing periods.

accumulator will add all of them together and in the 100th clock cycle send the summated value to the Golgi cell model as an excitatory input.

Figure 4-4B details the data path inside the neural model, which takes two computing stages: ion channel activities and integration. Each stage takes four clock cycles. Because of the parallel computational architecture, the latency in

each individual path has to be consistent; therefore appropriate delay blocks (the rectangular blocks) are added as necessary.

Figure 4-4C and Figure 4-4D show the subcomponent circuits, including the inhibition and excitation circuits and FIFO-based delay circuits. Since each neural processor implements 2000 granule cells and 20 Golgi cells, a pipelining technique is applied for reducing hardware resources. A long pipelining stage is required for storing granule cells calculation intermediate values. A First-In First-Out (FIFO)-based delay circuit is designed for achieving long computational stages.

4.3.2 *Network-on-chip**

To manage the transmission of action potentials between Golgi cells and granular clusters I have developed an NoC infrastructure. This system allows for arbitrary connectivity between Golgi cells and granular clusters. Each processing element is connected to a router through which the action potentials are communicated. The routers are connected together in a mesh topology as shown in Figure 4-4.

Routing strategy is decided on the system bandwidth, the memory overheads, the power consumption and area requirements. Bandwidth and memory size can mainly determine the power and areas consumptions. Due to the relative low connections from Golgi to Granule cells, the power versus area relationships are similar for unicast, multicast and broadcast routing strategies are similar. However, multicasting and broadcasting approximately require 2x as much memories as the unicasting strategy, because it needs to store extra routing information in the routers. Therefore, a custom designed unicasting strategy is applied on the system

When a Golgi cell produces an action potential the interface fetches a list of destination granular clusters from memory, and an individual packet is generated to be sent to each of these destinations[1] within the network. The connectivity of the neural network can be updated by adjusting the contents of the memory. A user may alter the contents of the memory to adjust the

[1] *: The presented work is finished by my collaborator Graeme Coapes, a PhD student at Newcastle University.

connectivity by injecting configuration packets into the network. This can be done at start-up or part way through simulation if required by halting the system by using the global frame master.

The packet format is shown in the lower panel of Table 4-1. Packets are classified by the setting of a two-bit-type identifier. The generated spike packet contains the address of the granular cell, allowing for the routers to direct the packet to the correct processing elements. Each granular cluster summates the packets received. This value is used as an input into the granular clusters. Packets are transmitted between routers using a four-phase asynchronous protocol and a parallel data bus. The routers are output buffered using a two-deep FIFO memory element.

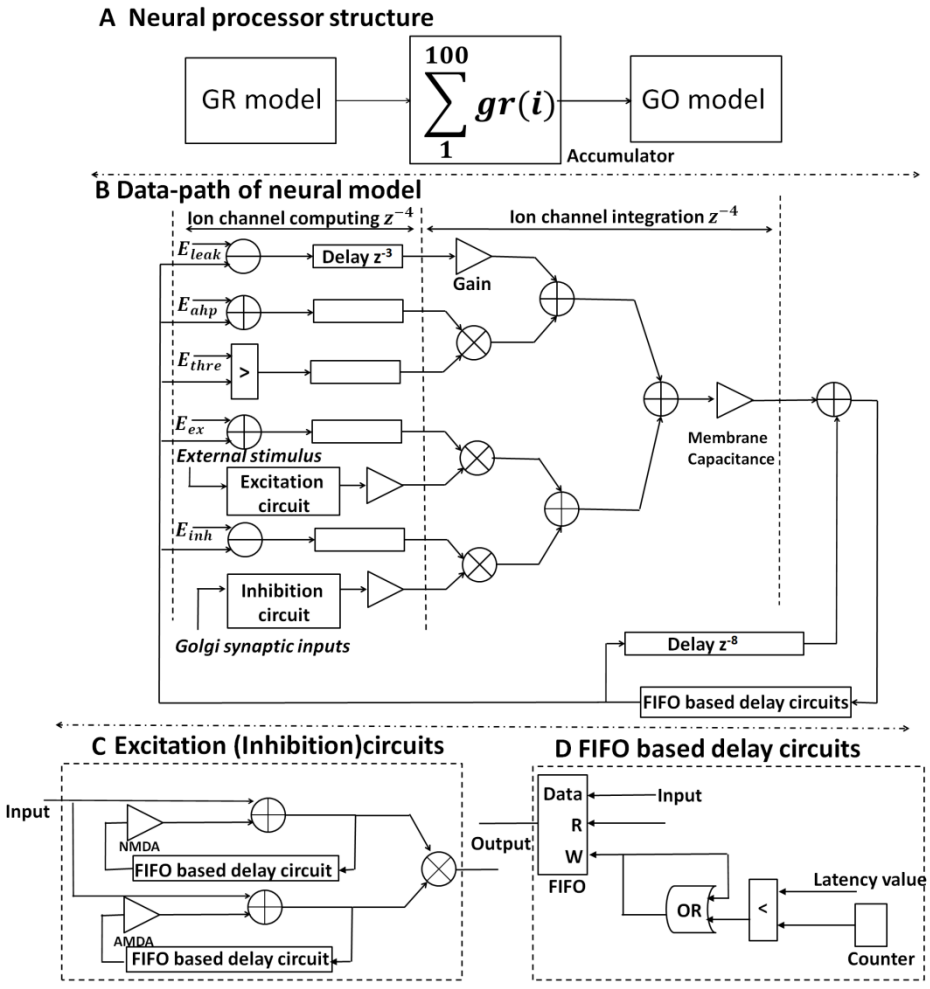


Figure 4-4: The neural processor structure and the data path of neural model. Fig. 4A shows the conceptual structure of the processor and Fig. 4B shows the data path of the neural model. Both GR and GO models use the same hardware architecture but with different parameters. The rectangular block is the delay function and the triangular block (gain) is the different ion channel conductances, which refer to Eq. (2). Fig. C and Fig. D show the subcomponent circuits:

excitation (inhibition) circuits and FIFO-based delay circuits. The triangular blocks denote the NMDA and AMPA receptor conductance.

Table 4-1: Standard spike package format

Name	Number of bits	Value
Golgi Spike Packet	2-bit	x0
Configuration Packet	2-bit	x1
Core ID	6-bit	\
Cluster ID	5-bit	\

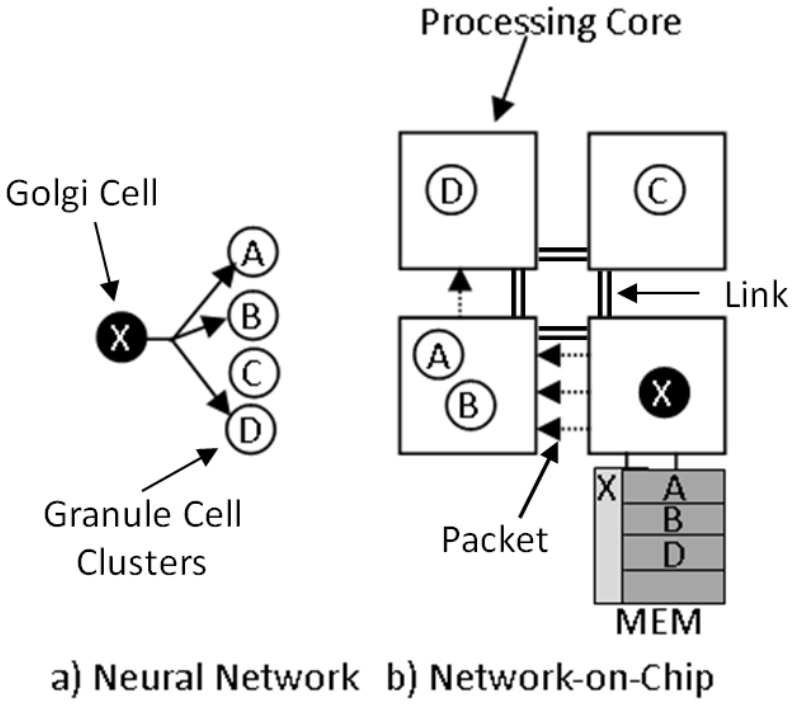


Figure 4-5: Example of mapping of neural network to a network-on-chip: a) A sample Golgi neural network with a single Golgi cell connected to three out of four granule-cell clusters. b) Four processing cores are shown. Each core may model multiple Golgi cells. When the Golgi cell X produces an action potential, individual packets are transmitted to each connected granule-cell cluster. The targeted granule-cell clusters are distributed throughout the mesh NoC.

To inspect the state of the model the network-on-chip is also responsible for transmitting information externally. When a Golgi cell produces an action potential, a 'Golgi Message' packet is also transmitted to a specialist processing element. This processing element buffers all received packets and transmits these packets to a PC. This enables a user to review the state of each Golgi cell at any time.

4.3.3 *Frame master*

In order to maintain synchronicity within the system a frame master is used. The master is responsible for ensuring that all packets are transmitted to their destination before the processing elements start to process the next time step. This ensures that the granular clusters receive all their updates within the correct time period.

For example, as shown in Figure 4-6, the duration of the network communication depends on the load of the network, which is determined by the frequency of Golgi cells spiking and the Golgi cell topologies. This varies for each frame. In each frame, once the first Golgi cell spike event is released (at time t2), the router starts to process the corresponding synaptic packages. After all 20 Golgi cell spike events are computed (at time t3), the processor's duty in frame 1 is finished. Then the neural processor needs to start computing the next 20 Golgi cell activities for frame 2. However, in frame 1 after time t3, the network is still processing the current 20 Golgi cell communication tasks. Therefore there is extra time allocated for the network to finish the first frame, before frame 2 begins. As a result of this, the frame master generates a low-level signal that disables the processor clock for the t3–t4 period until the network has completed the current frame routing task. The frame master then enables the processor to allow it to start computing again.

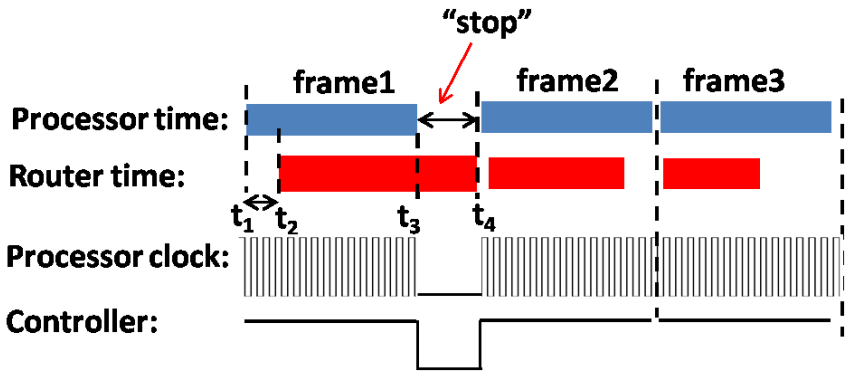


Figure 4-6: The frame master performances. In frame 1, the router processing time is longer than the processor's, so the frame master temporarily disables the neural processor at t3–t4 periods until the router finishes its current traffic loads, while in frames 2 and 3, because the routing time is shorter than the processor time, the processor clock is continuously running.

4.4 Results

4.4.1 *The hardware passage-of-time (POT) results*

Figure 4-7 shows a comparison of the membrane potential of a fundamental granule (Eq.(1)) neuron model simulated by the FPGA neural processor and by software (implemented in C). A fixed-point system with 40-bit and 22-fractional bit is employed in this system, and the selected length of bits has to guarantee each operation has sufficient precision to avoid data overflows and mismatch. The same inputs were given to both simulations.

They produce essentially identical results with very minor differences due to hardware truncation errors. Increasing the length of bits can eliminate truncation errors but introduce resource utilization waste.

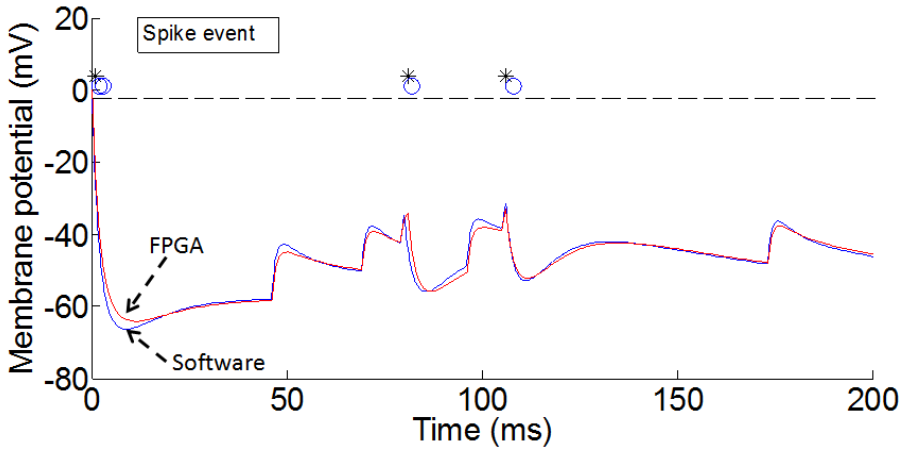


Figure 4-7: The comparison results of a fundamental granule (Eq.(1)) neuron model simulated by the FPGA neural processor and CPU. The CPU implementation is the original software described in [33], running with an Intel Quad Core™ i7 CPU with 8 GB of RAM under the Ubuntu operating system.

The hardware POT simulation results are summarized in Figure 8. Poisson spikes were fed into the simulated network to represent CS inputs through MFs. The simulated network was first fed at each MF with 5 Hz Poisson spikes for 300 ms to set the network to steady state, then 30 Hz Poisson spikes, preceded by 5 ms 200 Hz spikes, were given to excite the network.

Figure 4-8a shows the spike patterns of 40 granule cells randomly chosen from different granule-cell clusters. These granule cells show different temporal activity patterns. Specifically, they show a random repetition of transitions between bursting and silent states. These bursts are sustained for tens to hundreds of milliseconds. In contrast, the Golgi cells fire spikes rather regularly

as shown in the bottom panel of Figure 4-8 (a). Figure 4-8 shows the similarity index of the activity pattern against the time shift Δt (Equation 4-6). The gradual decrease of the similarity index with |Δt| demonstrates a smooth encoding of POT from the onset of CS, indicating that the populations of active granule cells change gradually over time such that no active granule-cell clusters appear more than once throughout the simulation. Both of the software and hardware simulation results are consistent with results shown in [11], which confirms a proper POT behaviour in the simulation, in that the sequence of active granule cell population maintains a one-to-one correspondence with the POT from the CS onset. The hardware simulation result is very comparable with software simulation, with mean error being less than 5% (Figure 4-8). The error is mainly caused by hardware truncation errors. Figure 8c shows the reproducibility index (Equation 4-7) from the hardware simulation, which compares the activity pattern generated by two different Poisson spike inputs. The reproducibility index remains high (>0.7), indicating that the POT encoding will remain robust despite in the spite of the variability of signals in the two stimulating inputs through MFs. This shows that the neuron population can maintain consistent POT representation across trials when, for instance, learning of delayed eyeblink conditioning over multiple training sessions is to be incorporated in the model [33].

4.4.2 *Effects of blocking NMDA channels on POT representation*

To further verify the hardware simulation results, I adapted the model to investigate the effect of blocking NMDA channels, which play a critical role in delayed eyeblink conditioning [97]. The hardware and software simulation results are summarized in Figure 8d–f. When NMDA channels are blocked in either granule cells or Golgi cells, granule cells lose the temporal structure in their firing; instead, they fire spikes in a rather continuous manner (Figure 4-8d). The similarity index becomes flat except for |Δt| smaller than ~30 ms. Within the time scale of 30 ms, there are a very limited number of spikes to encode a robust temporal structure for POT. On the other hand, 30 ms is too short for physiologically relevant POT in a classic firing pattern after NMDA-R blockade cannot capture a temporal structure on a timescale of physiologicalrelevance. The disruption of POT encoding consequent to NMDA channel blockade is

reflected by both software (Fig. 8e) and hardware simulation (Fig. 8f). The results (both software and hardware) are consistent with those presented in [33].

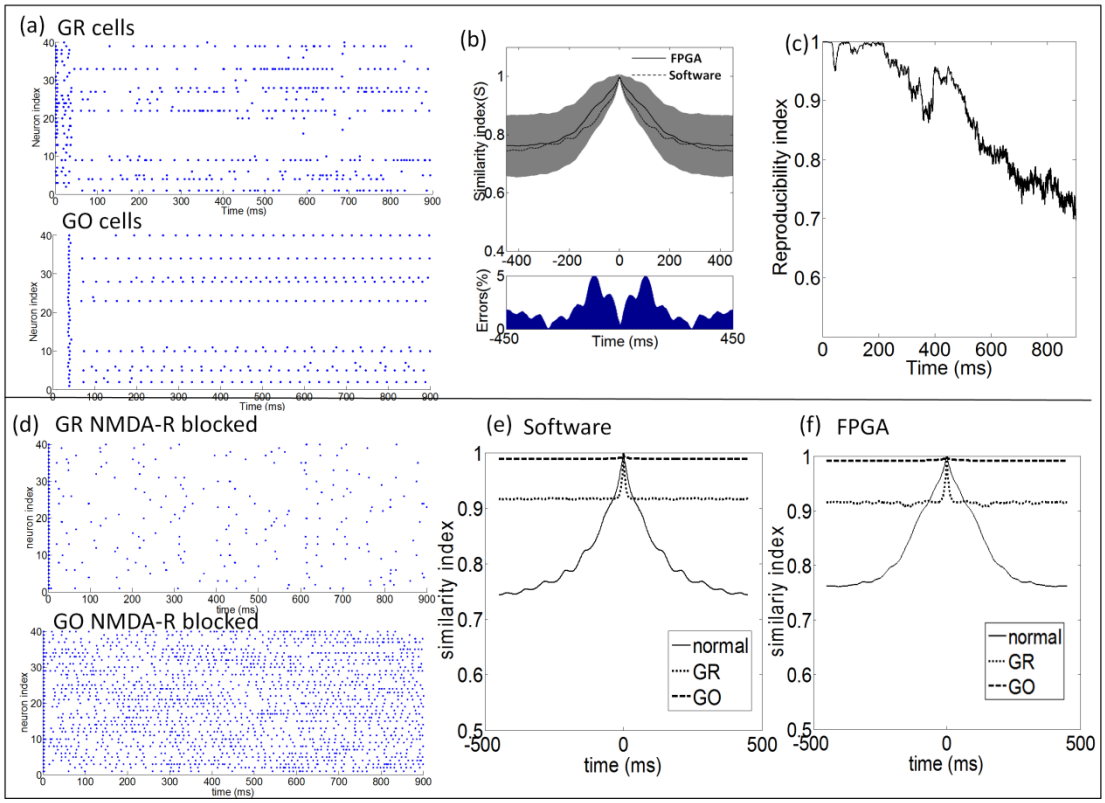


Figure 4-8: (a): Spike patterns of 40 granule cells and Golgi cells chosen randomly in an implemented granular layer. (b): Comparison of similarity index between software and FPGA simulations. The grey areas are the standard deviations of the hardware results. The errors between the two results are shown at the bottom. The maximum error is less than 5%. (c): The reproducibility index is calculated by Eq. (5). It maintains a high value, which suggested a robust POT representation despite the input variability. (d): Spike patterns of 40 granule cells when NMDA channels of granule cells (upper panel) and Golgi cells (lower panel) were blocked. Each neuron was chosen randomly from 40 different granule-cell clusters. The firing of the cells become rather regular and hence lost the ability to encode temporal information about POT. (e) and (f) : Comparison of similarity index between software and FPGA simulations when NMDA channels of granule cells (dotted line) or those of Golgi cells (dashed line) were blocked. The similarity indices become flat, indicating a loss of temporal structure in the granule cells' activity pattern.

4.4.3 *Frame master performances*

In particular, I examined the frame master performances of a two-by-two network-on-chip system as a case study. The simulation results are shown in Figure 4-9.

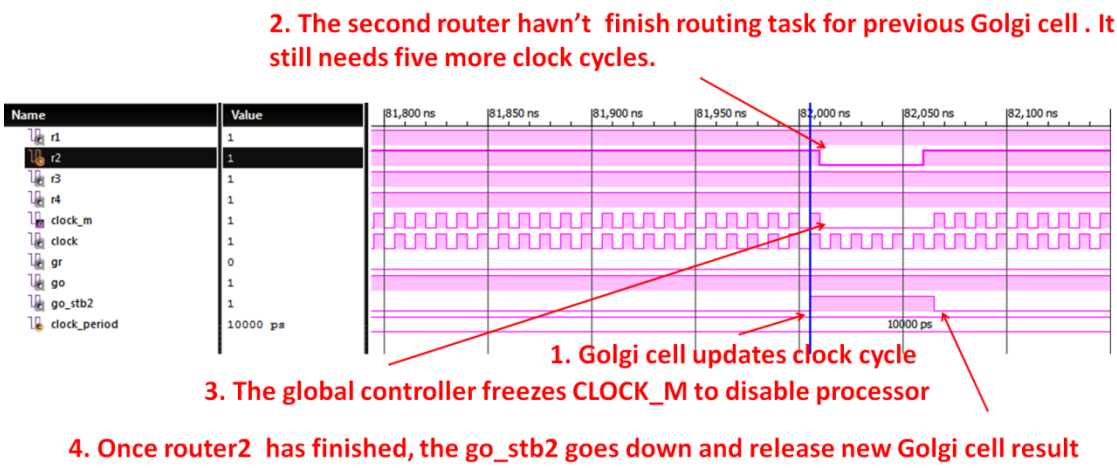


Figure 4-9: The simulation results of the two-by-two network-on-chip system.

In Figure 4-9, at the frame update clock cycle time point 82,000 ns, router 1 (r1), router 3(r3) and router 4(r4) all released control signal "1", which indicates that they have all finished their routing tasks. Only router 2 still generates the low-level signal "0". This shows that in the communication duty of the Golgi cell still being performed by router 2. As a results of this, the developed frame master immediately stops the clock from processor blocks computing clock_m until the router 2 control signal becomes a high level signal After five clock cycles, the process clock becomes enabled again since all the routers have finished their current frame routing missions. Then the entire system is ready to calculate the next frame process duties.

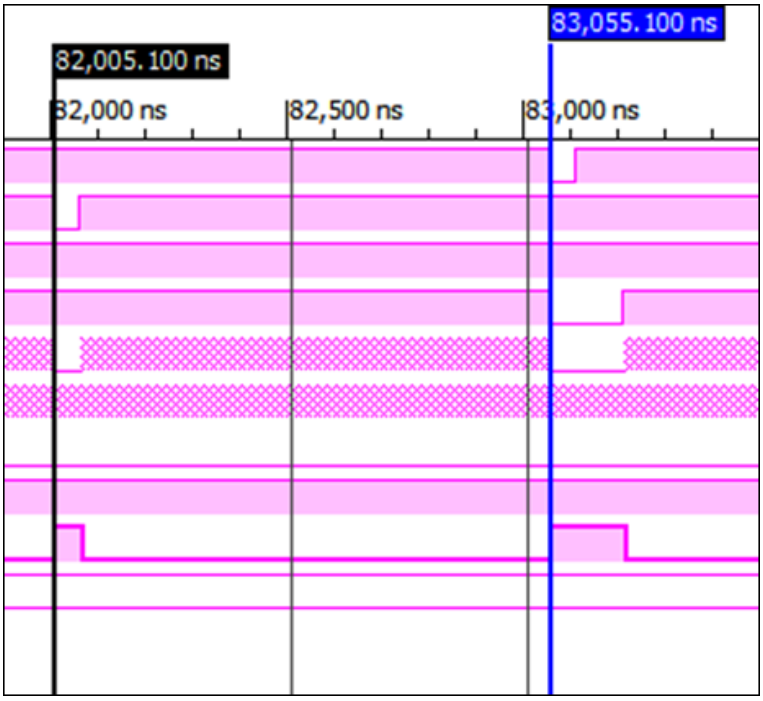


Figure 4-10: The performances of four system processors.

It can be clearly seen in Figure 4-10 that the current frame process duration is 105 clock cycles (83,055,100-82,005,100 = 1,050,000 ns), which is five clock cycles more than the standard frame process period of 100 clock cycles. This illustrates that the developed frame master successfully “freezes” the processors’ five clock cycles to avoid routing package traffic congestion.

4.4.4 *FPGA-based granular layer for neural rehabilitation*

I illustrated a hypothetical *in vivo* experimental set-up for closed-loop prosthetic application using the developed FPGA granular-layer system in Figure 4-11A. Biological neuronal spike signals will be recorded by using a multi-channel neural recording system that will then be used as inputs to the silicon granular-layer model. These neuronal spikes will be processed by the silicon-granular layer, which then generates the appropriately timed output discrete spikes to trigger the stimulation to be injected into the animal. Figure 4-11B shows an electronic system set-up to demonstrate such an experiment. A Virtex-5 board is employed to simulate the neural spike inputs conveyed by MFs, which are delivered to the FPGA cerebellum model via four-bit wires. The input discrete spikes are modelled as two 5 Hz and two 30 Hz Poisson spike trains in four-bit signals. The developed silicon granular layer is implemented on the Virtex-7 board with the I/O interface for displaying the system output on the oscilloscope in real time (Figure 4-11C). The displayed GR spikes were taken from three neural processors. The frame-based signal, which is used to monitor and verify system processing behaviours, is also shown. When each frame workload is finished, the frame-based signal is changed to a high-level value, and each frame uses 25.6 us (the distances between X1 and X2) to mimic 1 ms real-world activities. Hence, this set-up can complete 1 sec real-world activities in 25.6 ms at full speed as shown in Figure 4-12. The system specifications are summarized in Table 4-2.

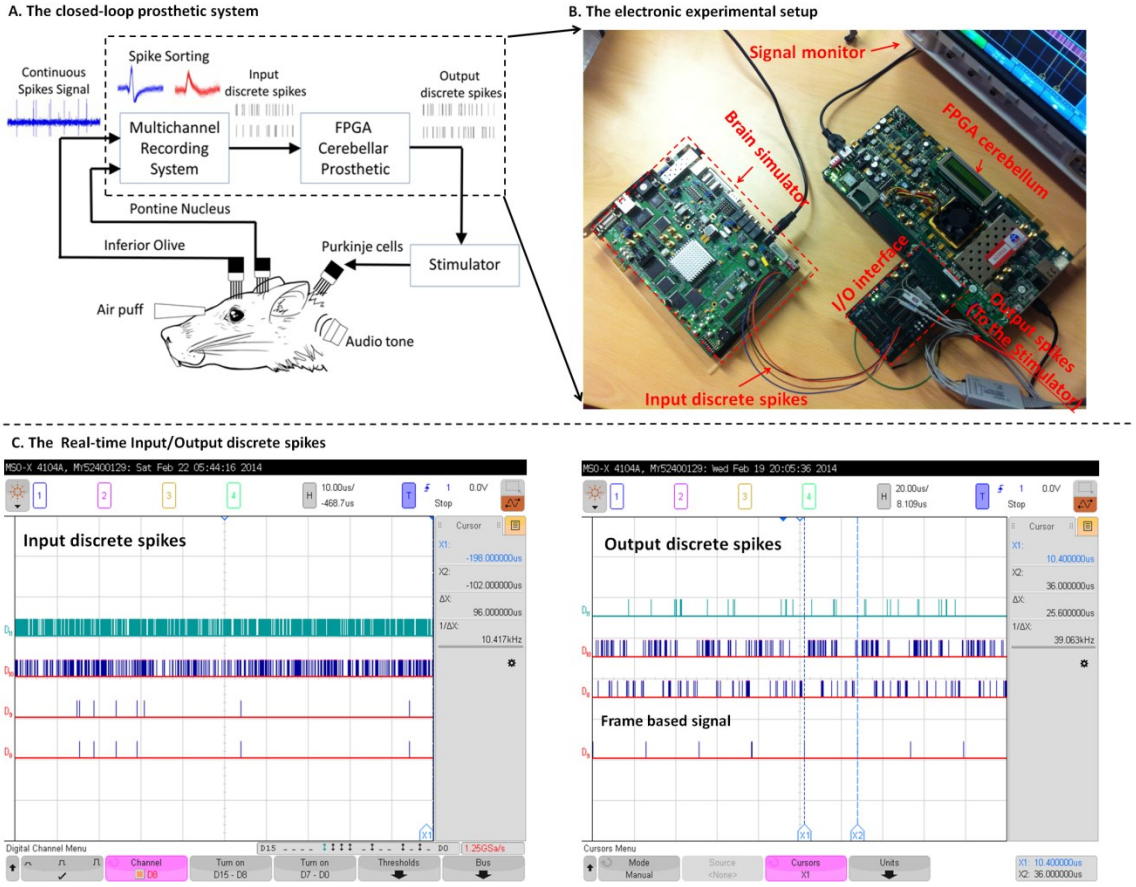


Figure 4-11: The overall system experimental set-up. A is the hypothetical *in vivo* closed-loop experimental set-up for cerebellum rehabilitation. B is an electronic set-up to demonstrate the feasibility of the *in vivo* experiment. A Virtex-5 board is employed to simulate the biological spikes conveyed by MFs, which are delivered to the FPGA cerebellum model via four-bit wires. The input discrete spikes are modelled as two 5 Hz and two 30 Hz Poisson spike trains in four-bit signals. The developed silicon granular layer is implemented on the Virtex-7 board with the I/O interface for displaying the system output on the oscilloscope in real time. C shows the real-time input/output discrete spikes and the frame-based signal.

Table 4-2: FPGA-based granular-layer specifications

Timing issues				
Maximum clock frequency				121.945 MHz
Minimum period				8.2 ns
Hardware resource utilization				
	Processor	Router	Module	Total
Slice register	2884	792	3676	176424 (29%)
Slice LUTs	4379	1213	5592	268455 (88%)
Block RAM/FIFO	20	0	20	960 (93%)
DSP48E1s	48	0	48	2304 (82%)
Power consumption				
Dynamic power	-	-	60 mW	2.88 W

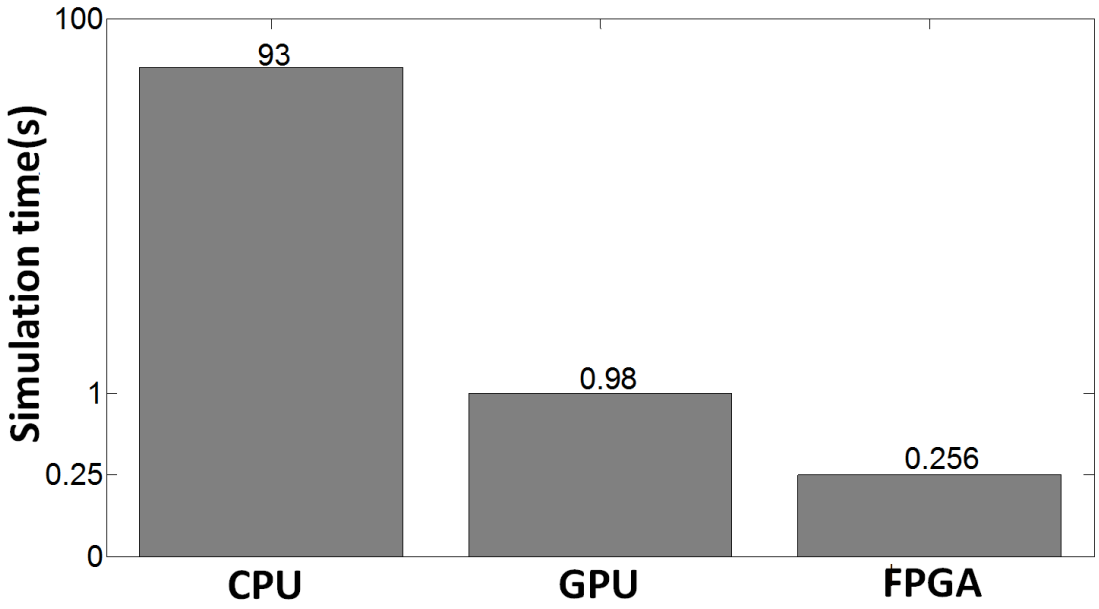


Figure 4-12: The real-time computational condition among CPU, GPU and FPGA for simulating 1 s activities. The CPU and GPU results are cited from previous work [92].

4.5 Discussion

4.5.1 *Scalability*

In Figure 4-13 I compare the performance of the presented design with three alternative approaches previously developed for implementing a spiking neural network. In addition to its higher computational speed, the FPGA-based NoC approach clearly demonstrates scalability compared with other approaches. The computation time remains almost constant even if the network size increases by an order of magnitude. (For both FPGA-based systems, there is an assumption of thay the resources corresponding increasingly as neuron number rises.)

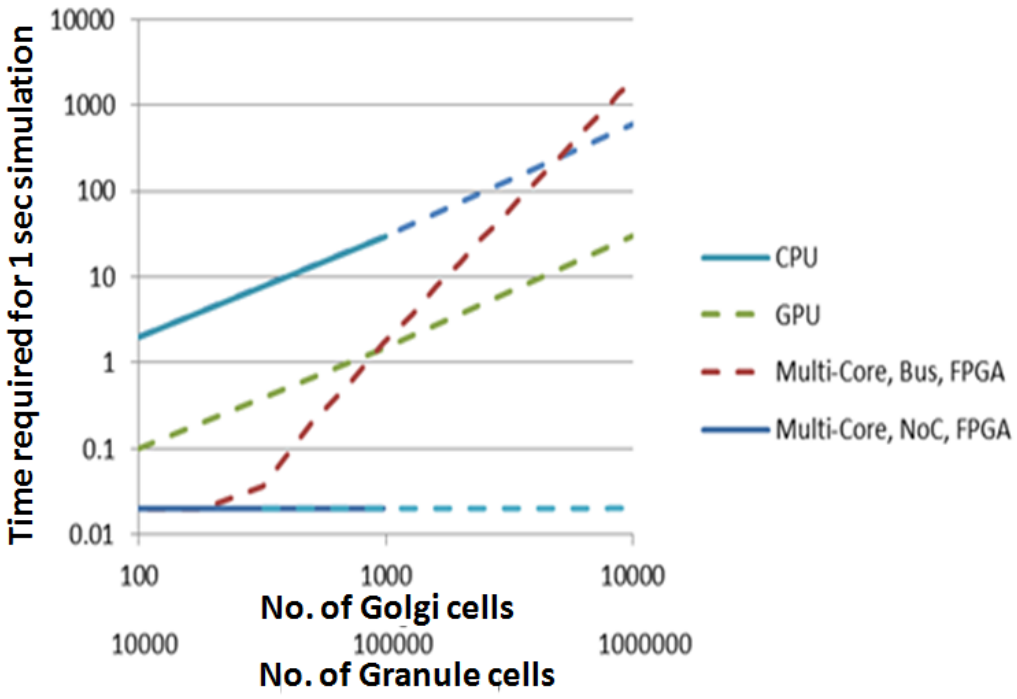


Figure 4-13: Scalability of four different approaches. The dotted lines represent the estimation of system performances, whereas solid lines represent the measurements. The FPGA-based NoC computation time remains constant due to its parallel nature and the efficient communication system.

The architecture of FPGA allows a variety of techniques to be employed to reduce the overall memory consumption. As such, memory requirement of FPGA based optimized system scales much less steeply. For instance, the major memory consumption is to store the connectivity information of the individual granule and Golgi cells. In implementing the network connectivity, the NoC architecture requires only tiny resources for storing routing information, in that each Golgi neuron package is only 15 bits. On the other hand, pipelining technique is employed to significantly save memory resources when simulating a large number of neurons.

Another alternative is to use GPU processors, which can supplement or even replace CPUs for parallelizable code. The rise of GPU languages such as CUDA and Open CL has simplified their use enormously. Modern GPUs exceed 5000 cores and can increase processing speed by orders of magnitude for parallelizable tasks [23][98][92]. Additionally, GPUs offer extremely high raw memory bandwidth, though this is difficult to achieve in practice and requires adherence to strict memory access patterns [23].

Nevertheless, with sufficient power, it is possible to implement spiking neural networks for high-speed computation on a GPU. However, this would be at the cost of relatively large power consumption, which is not scalable to prosthetic devices. I therefore chose an FPGA platform with large numbers of I/Os for potential *in vitro* and *in vivo* operation.

One key difference between the FPGA platform and processor-based implementations is that I utilize distributed, localized memory banks that avoid sharing of global memory resources. This avoids delays associated with accessing global memory and reduces power consumption by minimizing the size and operating frequency of channels between processors and memory.

A further variance of previous work is the use of frame-based encoding. One issue with real-time NoC systems is that spiking information encoded in latency or frequency can be prone to distortion due to congestion [99][100][101]. In contrast, I utilize a stop-start approach whereby all the neural spikes processed and then stopped to allow full transmission around the network whenever necessary. This is actually akin to biology, whereby synaptic transmission, dendritic signal integration and action potential initialization can take time, but the transmission speed is actually very fast [102]. In addition to low distortion, this approach also allows us to easily compare among computational models. I can simply extract a specific frame N from the simulations in all cases for detailed comparison.

An alternative digital implementation of an NoC is perhaps a bus between processing cores. This will form a limiting factor whereby increases in frequency lead to distortion of the information. Alternatively, some of these effects can be alleviated using traffic management via hierarchical AER architectures [103].

Using an NoC infrastructure as opposed to a bus also reduces power consumption within the design as it allows for much reduced clock frequency. Using a Xilinx XPower Analyzer I estimate that when implemented upon a Virtex-7 VC707 XC7VX485T-2FFG1761C Evaluation Kit, each module, containing a processor, router and interface, consumes 60 mW of dynamic power, equating to a total dynamic power consumption of 2.88 W when running at full speed, or 60 mW per processing module.

4.5.2 *Comparison of other techniques*

There are several possible alternative techniques to the frame-based network-on-chip architecture. Currently, SpiNNaker [3], NeuroGrid [104] and IBM SyNAPSE [11] are projects that build custom chips or systems for efficient large-scale simulation of general neural network models. These systems are powerful and innovative; however, they may not be optimal for the system that I am implementing in this paper. For example, SpiNNaker with multicast strategy will require the addition of extra memory resources to control the routing at some intermediate nodes. With the unicast strategy, routing is determined purely from the packets contents - reducing the memory overheads.NeuroGrid employs a smart approach to combining analogue circuits for mimicking the neural process and digital circuits for implementing routing components. It can potentially save a significant amount of energy consumption. But analogue circuit-based dimensionless models are not ideal for mapping conductance-based leaky integration-and-fire neurons in a POT model. IFAT [105] is also a well-established platform for brain network real-time operation, but the analogue-based integrate-and-fire array may not provide good scalability.

I am seeking to further optimize the system and to use it for other applications. Cassidy et al [106][107] developed a neuro-array architecture for a general large-scale neuromorphic systems with corresponding analysis. Their design principles, including external SRAM technique, can provide new insight for optimizing the system. Also, applying the developed silicon granular layer to perform pattern recognition would be another implication that is similar to the new IBM chip TrueNorth [11].

In fact, the developed frame-based network-on-chip architecture is general for spiking neural networks, although in order to implement other models, I need to

modify the components appropriately for the target model. For instance, in this work the routing components (transmitter, router and receiver) are custom designed for implementing POT recurrent random network connections; and neural processor architecture is also specifically designed for mapping the connections from granule cells to Golgi cells. Further system tweaking will be required to optimize the performance for a different target model.

4.5.3 *Neuro-prosthesis applications*

For translation into neuro-prosthesis, the architecture lends itself easily to electrical [108] or optical stimulation methodologies [109][110]. The FPGA-based granular model can correctly predict responses of POT behaviour and thus be used to interface with *in vivo* and *in vitro* experiments. Furthermore, it is straightforward to translate generated spikes directly to tissue as each will be encoded with a destination address.

For long-term neuro-prosthesis experiments this design can be translated directly to an ASIC platform in order to increase portability and to reduce power consumption. I estimate that by translating the design into CMOS, each module will consume 1.3 mW in high-speed operation and only 0.6 mW in real-time operation, giving a total power consumption of 28. 8mW for implementing a neural network containing 100,000 elements. This compares favourably to power requirements in the brain whereby exceeding 100 mW can cause thermal damage [111].

4.6 Conclusion

The goal of the work has been to implement a real-time cerebellar granular-layer model onto an FPGA hardware platform utilizing an NoC hardware architecture. The design can achieve (more than) real-time operation for a system of 1000 Golgi cells and 100,000 granule cells on a single FPGA board. This is achieved via an efficient implementation of the mathematical models of the neuron cells, and the use of a frame-based architecture that eliminates congestion distortion of spike timing in multiplexed networks. The design is also highly scalable in that computation time remains almost unchanged for a much larger network model.

The major contributions of this paper are summarized as follows: 1) An efficient FPGA-based NoC hardware architecture is developed for implementing a large-scale cerebellar granular-Golgi layer model for POT encoding; 2) The implementation is computationally efficient in that it can complete a 1 sec simulation in 25.6 ms and that FPGA provides precise timing control. Together they allow our design to be readily adapted for real-time closed-loop *in vitro* or *in vivo* experiments; 3) The NoC architecture is highly scalable and hence it is now possible to simulate the full-scale granular layer with a cell density of 1 million cells/mm^3 as in the real brain, which is 10 times the size of the current model. Such simulation power can open up new possibilities for understanding the dynamics of the cerebellar network; 4) The design can be a potential neuro-prosthetics tool for future experimental and clinical applications owing to its high computational power, flexibility, high scalability and power efficiency.

Chapter 5 Case Study: Central Pattern Generator Prosthesis

This chapter aims to explore digital neural circuit neuro-prosthesis applications. A reliable pyloric central pattern generator prosthesis technique is presented to explain the methodology in details. The biological pyloric network is an appropriate platform since individual neuron characters and synaptic connections are clearly identified. The approach steps, from software modelling, hardware implementation and system-level reliability investigation to experimental set-up, are discussed in depth. The simulation results demonstrate that the developed system can successfully restore the damaged network functionalities in different external environments. This work can be considered as a framework of digital circuits for neurorehabilitation applications.

5.1 Introduction

Neurorehabilitation is a vital technique that aims to help people disabled by injury or disease affecting the brain, spinal cord or muscles. It is a collection process that specifically focuses on a person's recovery potential and can help a patient to live a more normal, active and independent life. Traditional neurorehabilitation techniques include occupational [112], psychological [113], speech and language therapies. It shows its effects and strongly helps people to recover from diseases (e.g. stroke, Parkinson's and brain injury), both mentally and physically.

However, when these approaches confront neurologic disorders such as ataxia, epilepsy and conditions caused by damage to the nervous system, they display limitations and constraints since neural circuits are fundamentally injured. Therefore, currently there is still no effective rehabilitation approach to cure these diseases.

Nowadays, with the rapid development of neuroscience and electronic subjects, there is potentially a new way to address this dilemma: using silicon neurons to replace the damaged real neurons to restore original biological functionalities.

Achieving this technique that integrates neural network and electronic circuits into an entire system is a great challenge. The fundamental issue is that the two systems utilize totally different computing mechanisms: electronic circuits based on Metal-Oxide-Semiconductor (MOS) utilize small amounts of dopants to create conductivity and are transformed into specific circuits for daily applications, while biological neurons receive multi-synaptic inputs from synapses, and integrate this information to generate spike patterns related to movement behaviours. The challenges lie in the fact that the silicon neurons have to accurately replicate various biological system spiking patterns in real time. More importantly, since the neural network is adaptive to the external environment, it will automatically change its bursting frequency according to the sensory inputs. This indicates that electronic systems have to modify their computational speed in real time to follow the biological ones.

Previously, Berger et al [114] showed an external silicon chip connected to rat and monkey brains to achieve memory prosthesis. R.J. Vogelsteinet al [49] created a neuromorphic chip to reproduce biological spinal central pattern

generator (CPG) activities and used this chip to support a cat walking *in vivo*. These works all serve as strong evidence of the feasibility of this concept.

In this work, I propose a reliable and capable system specifically for CPG function restoration, which is shown in Figure 5-1. Compared to the previous systems, the work is stronger frow two aspects: silicon neurons' bio-plausibility and system reliability. Firstly, the digital neural circuits are designed to reproduce both real CPG control and pharmacological outputs, which are particularly aimed at conditions with totally damaged and partially damaged systems. Secondly, the developed system is capable of robustly changing the computing speed to achieve the best communication performances with biology by using an adaptive control mechanism.

The rest of the paper is organized as follows. Section 5.2 briefly describes the pyloric CPG modeling work. Section 5.3 explains the system architecture and individual component functionalities. Section 5.4 presents the hardware simulation and biological experimental results. Section 5.5 provides a comprehensive discussion, and in Section 5.6 a conclusion is presented.

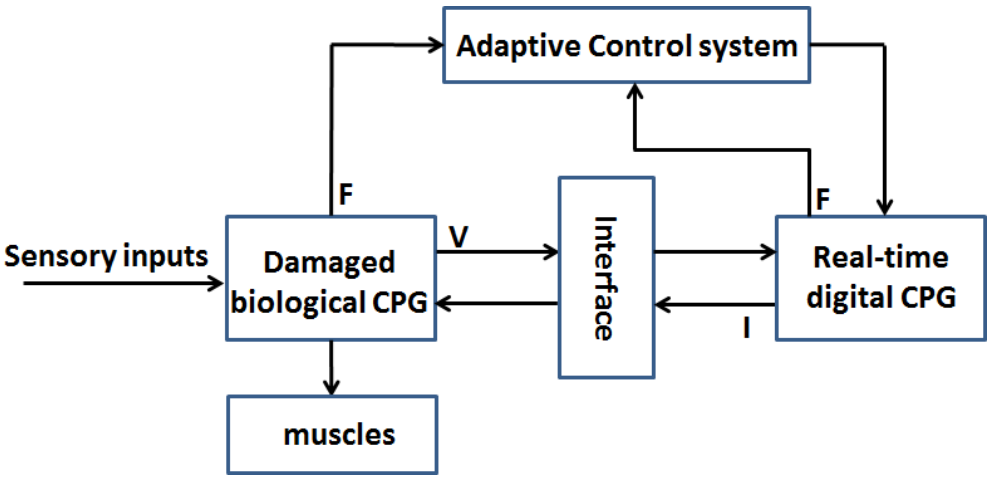


Figure 5-1: The conceptual system architecture. V is the membrane potential, I is the generated current and F is the neural bursting frequency.

5.2 Pyloric CPG modelling

5.2.1 *Pyloric behaviours*

The pyloric network is one of two CPGs in the stomatogastric ganglion (STG) of a crab. It contains around 14 neurons with complex connections [1]. The function of the pyloric network is to control striated muscles that dilate and

constrict pyloric areas in the stomach [115]. It has been investigated for almost 50 years and contributes many important neural mechanisms, such as adaption [116], compensation [117] and evolution [32], to the neuroscience society. Figure 5-2 describes network rhythms and synaptic connections.

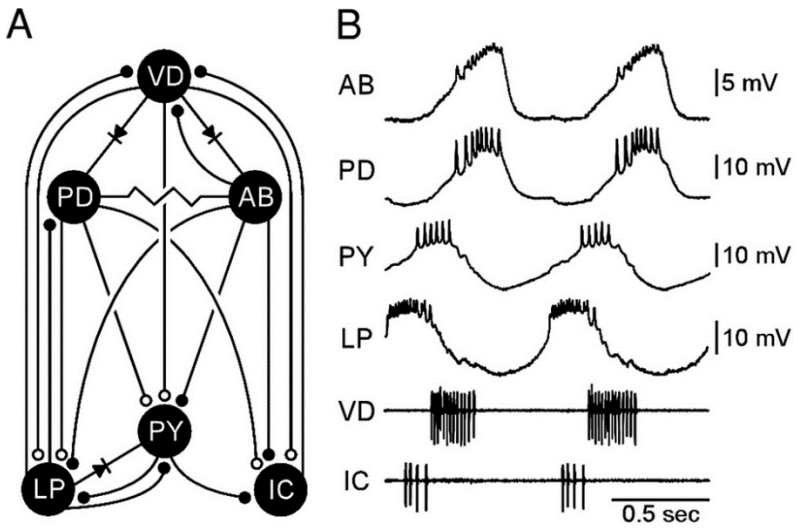


Figure 5-2: The pyloric network synaptic connectivity and output patterns. There are six neurons in the network: AB, PD, PY, LP, VD and IC. The figures are cited from [118].

In the pyloric network, neuron AB is electrically coupled with neuron PD as the pacemakers in the whole network. At first, the PD and AB neurons together inhibit the LP and PY neurons. Then the LP neuron rebounds before the PY neurons due to various factors, and in turn inhibits the PY neurons. When the PY neurons rebound from the inhibitions, they in turn terminate LP neuron bursts. The firing frequency varies from 0.5 Hz to 3 Hz [115].

The muscles active in each phase of the pyloric cycle are shown in Figure 5-3. Activity of the pyloric dilator muscles mediated by the two PD neurons appears to open a valve in the pyloric region, which is then closed in the second phase by an antagonist, the lateral pyloric muscle operated by the LP cell. In the third phase, a sheet of pyloric muscles contracts under the activation of several PY neurons (divisible into PE and PL subtypes), giving overall a peristaltic appearance to the pyloric surface. The two muscles located in the anterior (cardiac) portion of the stomach control a curious valve structure [119].

5.2.2 *Modelling*

I model the pyloric CPG in three different stages. In the first stage, I analyse and investigate which factors strongly influence the pattern generation

according to the biological experiment recordings; in the second stage, based on previous results, I model the pyloric pattern generation mechanism at qualitative level to give a fundamental architecture; in the third stage, the specific neural models will be selected and parameters will be optimized to achieve exactly the same behaviours as the biological one. The model will then be quantitatively defined.

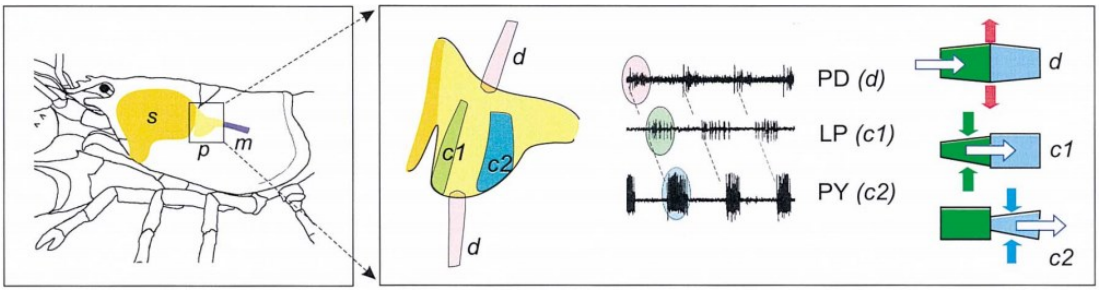


Figure 5-3: The pyloric muscle activities in a lobster stomach. Neuron PD controls muscle d; neuron LP controls muscle c1 and PY controls muscle c2. The figure is cited from [119].

Four factors will be described in the first stage: commissural inputs, synaptic connection, oscillators and non-oscillators. Commissural inputs are from inter neurons, and the inter neurons make excitatory synaptic contact with all pyloric cells except the oscillators AB and PD. Oscillators are neurons AB and PD while the other neurons are non-oscillators. Synaptic connection is the topology of this pyloric network. The main difference between oscillators and non-oscillators is that oscillators can generate bursts without inputs but non-oscillators cannot. The influences of these four factors [120][38][121][118] on pattern generation are displayed in Table 5-1. The conclusions can be summarized as below:

- With commissural inputs contacted, this pyloric circuit can still generate spiking patterns even with one or two lost oscillators.
- Without commissural inputs, this pyloric circuit can only generate spiking patterns with complete oscillators (AB and PD) and the conditional oscillator LP.
- A synaptic connection is necessary for generating rhythm patterns.

According to these important findings, I translate this information into a visualized model, which is shown in Figure 5-4. The model uses three different measurements to define network properties: neuron bursting ability, synaptic strength and resting potential values. It is shown that neurons AB and PD have the strongest bursting generation behaviours and neurons IC and PY seem to

be inactive neurons. AB and PD drive IC and PY for bursts, and for synapse properties, both AB and PD have a strong inhibition to neuron LP. Also, neuron LP has the same effects as neurons PY and VD. But the synaptic activation among neurons LP, IC and VD is quite weak and exerts a tiny influence on all the entire network pattern generations.

Table 5-1: The influences of three factors on CPG spiking pattern generation

TABLE I

THE INFLUENCES OF THREE FACTORS ON SPIKING PATTERN BEHAVIORS

Synaptic connections	Inputs	Oscillation AB	Oscillation PD	Non-Oscillation LP	Non-Oscillation VD	Spiking patterns
+	+	+	+	+	+	Strong
+	+	—	—	+	+	1. Overall frequency decreased 33%. 2. IC cell activates increase. 3. The remain cells shift phase of burst.
+	—	+	+	+	+	1. Overall frequency decreased 50%. 2. AB and PD amplitudes decreased 50%.
+	—	—	—	+	+	None
+	+	—	+	+	+	1. Overall frequency decreased 33%. 2. VD cell bursts shift.
+	—	—	+	+	+	None
+	+	+	—	+	+	1. PY cell silent 2. VD and IC spike per burst decreased.
+	—	+	—	+	+	None
+	+	+	+	—	+	1. The overall frequency increased 15%. 2. PY cell burst has a prolongation. 3. Introduction of IC cells.
+	+	—	+	—	+	None
+	+	—	—	+	—	None
—	+	+	+	+	+	None

+ : Activate — : de-activate

In the final stage, these neurons are mimicked by the Hindmarsh-Rose (HR) [54] model. The reasons for choosing this model are the relatively simple mathematical equations and strong bio-plausibility. Neuron functions such as repetitive firing, post-inhibitory rebound and plateau properties can all be

reproduced by the HR model, and the equations are shown in Equation 5-1 – Equation 5-3:

$$\frac{dx}{dt} = (-x^3 + a \times x^2 + y + I - z) \quad \text{Equation 5-1}$$

$$\frac{dy}{dt} = (1 - b \times x^2 - y) \quad \text{Equation 5-2}$$

$$\frac{dz}{dt} = (r(s(x - x_o) - z)) \quad \text{Equation 5-3}$$

Where $x(t)$ is the membrane potential, which is written in dimensionless units, and $y(t)$ and $z(t)$ can be considered as fast and slow ions variables. The model has eight parameters: $a, b, c, d, r, s, x_o \; and \; I$. The parameter I indicates the current that injected in the neuron, is taken as a control parameter. For synaptic modelling aspects, in line with [122] and [123], I employ these two models as electrical and chemical synapses. For electrical synapse, it can be modelled by resistances to capture gap junction behaviours that are both bidirectional signal transfer and synchronization. However, chemical synapses remain the key communication in this network. I employed a synaptic model in Equation 5-4 – Equation 5-6:

$$I_{syn} = g_{syn} \times S \times (E_{syn} - V_{post}) \quad \text{Equation 5-4}$$

$$(1 - S_\infty) \times \tau_{syn} \frac{dS}{dt} = (S_\infty - S) \quad \text{Equation 5-5}$$

$$S_\infty(V_{pre}) = \begin{cases} \tanh\left(\dfrac{V_{pre} - V_{1/2}}{V_{slope}}\right), & V_{pre} > V_{\frac{1}{2}} \\ 0, otherwise \end{cases} \quad \text{Equation 5-6}$$

where $\mathrm{V_{post}}$ and $\mathrm{V_{pre}}$ are the post- and pre-membrane potentials, $\mathrm{E_{syn}}$ is the resting potential and $\mathrm{g_{syn}}$ is the conductance, S is the ration between 0 and 1, and $\mathrm{V_{1/2}}$ and $\mathrm{V_{slope}}$ are synaptic half-activation voltage and slope voltage.

The parameters of each neuron and synapse are shown in Table 5-2 and Table 5-3.

Table 5-2: Pyloric neuron parameters

Neuron	AB	PD	LP	PY	VD	IC
a	2.7	2.7	2.8	2.6	2.8	2.8
b	0	0	0.2	0.2	0.2	0.2
r	0.003	0.003	0.003	0.003	0.0021	0.0021
s	4	4	4	4	4	4
x_o	-1.1	-1.1	-1.1	-1.1	-1.2	-1.2

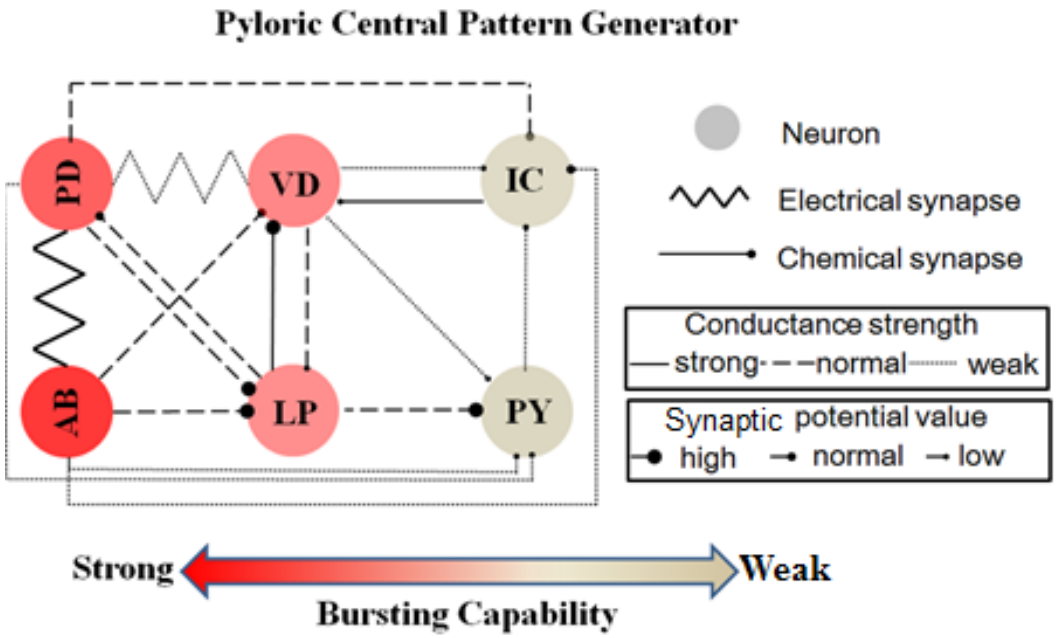


Figure 5-4: The qualitative pyloric computational model. Neuron bursting capabilities, synaptic strengths and resting potential values are fully described in this model.

Table 5-3: Pyloric synapse parameters

Synapse	g_{syn}	$V_{1/2}$	V_{slope}	E_{syn}
LP→PD	0.5	1.5	0.02	-10
AB→LP	0.5	1.5	0.02	-20
PD→LP	0.5	1.5	0.02	-20
PY→LP	0.5	1.5	0.02	-5
VD→LP	0.5	1.5	0.02	-5
LP→PY	0.5	1.5	0.02	-15
PD→PY	0.1	1.5	0.02	-5
AB→PY	0.1	1.5	0.02	-5
VD→PY	0.1	1.5	0.02	-1
LP→VD	0.7	1.5	0.02	-15
AB→VD	0.5	1.5	0.02	-10
IC→VD	1	1.5	0.02	-5
PD→IC	0.5	1.5	0.02	-10
AB→IC	0.5	1.5	0.02	-10
PY→IC	0.2	1.5	0.02	-5
VD→IC	0.2	1.5	0.02	-1

5.3 System architecture

The system mechanism is described as follows: the damaged CPG receives sensory inputs and generates motor signals to control muscles; at the same time, it sends outputs to the digital CPG via the neural interface. After the

process time, the digital CPG feeds back calculated recovery currents to restore the damaged original functionalities. Meanwhile, the output frequency of the digital CPG is adaptive in real time to biological CPG bursting frequency through adaptive control mechanisms to maintain system reliability. The individual blocks are discussed below.

5.3.1 *Digital CPG*

I implement the previously developed pyloric model on a parallel computational platform FPGA as the prosthesis processor. The hardware architecture is shown in Figure 5-5.

A classic timing multiplexing technique is employed to save hardware resources. A timing division multiplexer (TDM) has six channels; each channel is responsible for one neuron's activities. And the TDM selects input signals sequentially for calculation, while the corresponding parameters will also be sent from the block of initial states.

The key design principle for applying timing multiplexing technique to build a biological network is shown in Equation 5-7.

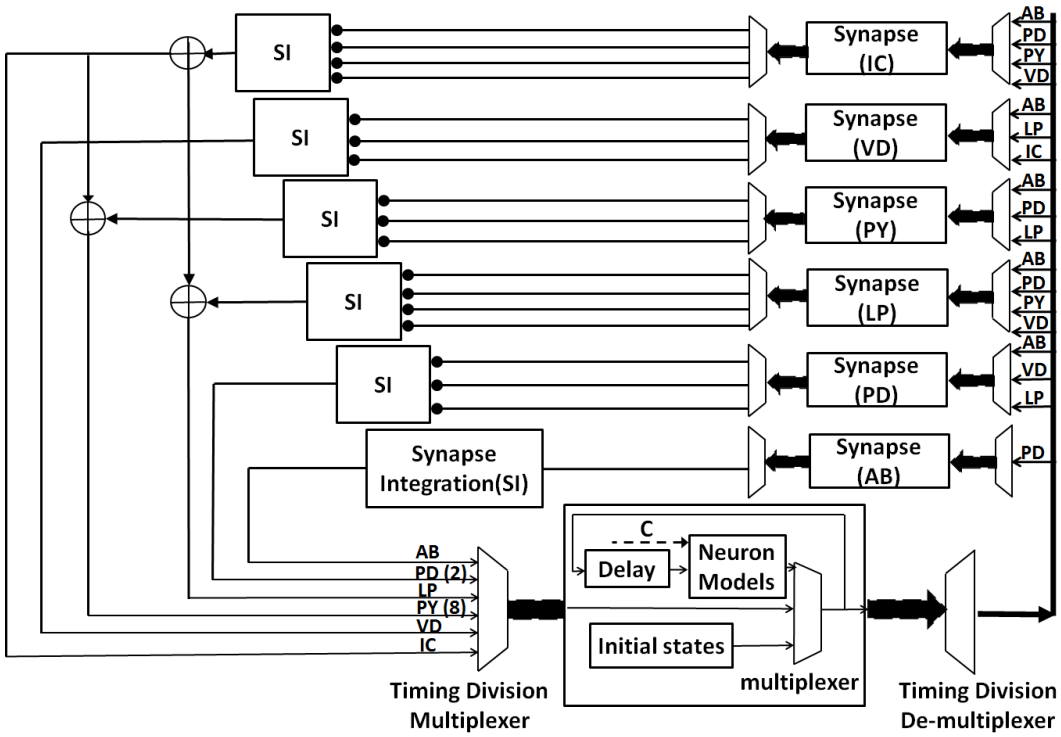


Figure 5-5: The hardware architecture of digital CPG. The SI block is the synapse integration; signal C is the control signals from the adaptive controller;

the block of initial states is used to pre-store different neuron parameters; the block of delay is applied to balance computing latency.

$$t_i = n = t_c \qquad \text{Equation 5-7}$$

Where t_i refers to integrator latency, t_c refers to the model data-path computational periods, and n is the total neural number.

With regard to latency constraints in TDM technique, the integrator latency $\mathrm{t_i}$ has to be equal to the number of TDM inputs n in order to maintain correct calculating sequences among different input channels. In general, $\mathrm{t_i}$ is set to one cycle to achieve the integration function. However, when TDM is applied in a neuronal model, the value of $\mathrm{t_i}$ has to be set to a specific value to match TDM calculating sequences. For example, a neuronal network consists of 14 neurons and the integrator latency is also one cycle. When computation of the first input channel signals for neuron A is finished, results are delayed for one cycle by $\mathrm{t_i}$. While the second input channel signals are calculating for the activities of neuron B during the second cycle, results calculated from the previous input channel (i.e. neuron A) are released. Calculated activities of neuron A are included in the calculation of neuron B. As a consequence, the activities calculated for neuron B are incorrect. Therefore, integrator latency $\mathrm{t_i}$ has to be equal to the number of TDM inputs.

The calculating latency $\mathrm{t_c}$ has to be equal to or be in a multiple relationship with $\mathrm{t_i}$ because the timing division de-multiplexer (TDD) selects calculated results at specific time points. In the previous example, suppose the implementation of the 14-neuron neuronal model has 56 clock cycles for computing a burst, so each neuron uses a total of four clock cycles for calculation. The TDD output results of the first neuron A will be released at the time period of clock cycles 2, 16, 33 and 44. However, if the calculating latency is 15 clock cycles, the second results for neuron A finish computing at the time period of clock cycle 17, which is the output time point for neuron C. This makes the system output results incorrect.

A data-path diagram of the HR model is shown in Figure 5-6. The arithmetic functions multipliers and adders have three and one latencies, respectively. So three data-path latencies are 12, 8 and 11 cycles in total. According to Equation

5-7, I have to artificially add 2, 6 and 3 delays into the corresponding places. This in turn is to shorten the critical path and bring more computational cycles into the model to match the network neuron numbers. Also, integration functions are achieved by using a register and an adder component. Register delays are artificially set to 14 to match the calculating latency as well.

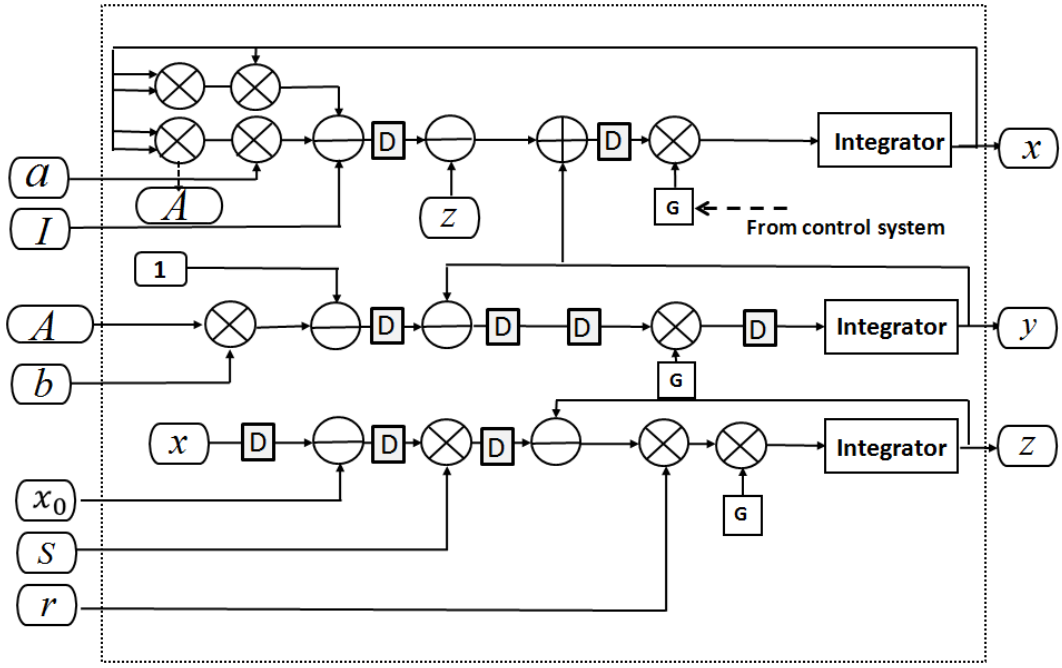


Figure 5-6: Data path of HR neural model. D is the delay register, and the integration step G is real-time updated by control system outputs. The corresponding equations are shown in Equation 5-1 – Equation 5-3.

A data-path diagram of the chemical synapse model is shown in Figure 5-7. The synaptic circuit computes synaptic currents based on pre- and post-synaptic membrane input voltages. In each step in the circuit, two state variables are stored and a single integration result is calculated. The parameters of this kinetic synapse model can fit directly into physiological measurements. In addition, the triangle and divider functions are achieved by using look-up table techniques.

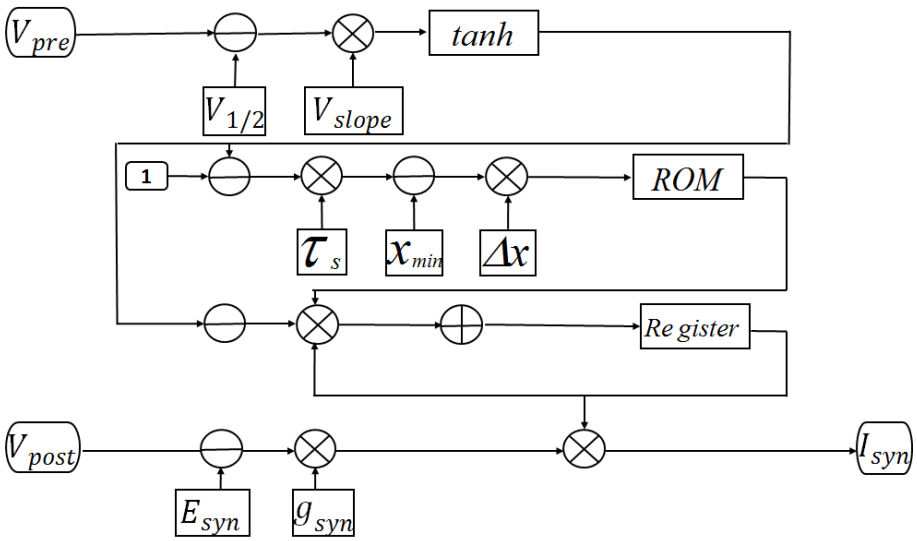


Figure 5-7: Data path of chemical synapse. The corresponding equations are shown in Equation 5-4 – Equation 5-6. The triangle and divider functions are achieved by using look-up table techniques.

5.3.2 *Adaptive control mechanism*

The entire adaptive control system is shown in Figure 5-8. The basic mechanisms are described below: first, biological neuron bursting frequency is measured in real time as shown by $r(t)$. Then it is compared to the three references in the adaptive controller: reference 1 is the slow bursting period, reference 2 is the normal bursting period and reference 3 is the fast bursting period. According to the calculated errors $e(t)$, the switches automatically choose the gain that corresponds to the smallest value $e(t)$. This is the adaptive gain that best fits the current biological neuron bursting state. After an electrical neuron receives control outputs and interacts with biological neurons, the electrical neuron can follow biological neuron bursting period characters to avoid an incorrect spiking phase relationship and irregular burst patterns.

For a real-time bursting period measurement algorithm, there are three stages in the calculation. The algorithm flow is shown in Figure 5-9.

Firstly, in order to avoid heavy computations in the algorithm, a low-pass filter is applied to focus on burst fields rather than individual spikes. The frequency pass parameter in the system is 0.01 and the frequency stop is 0.03. In the next step, every burst threshold time point is detected and recorded by using hit crossing(Matlab library) technique. The burst threshold value is set to -15 mV. Finally, the burst period is calculated by using the current burst threshold time point t_n subtracted from the previous burst threshold time point t_{n-1}.

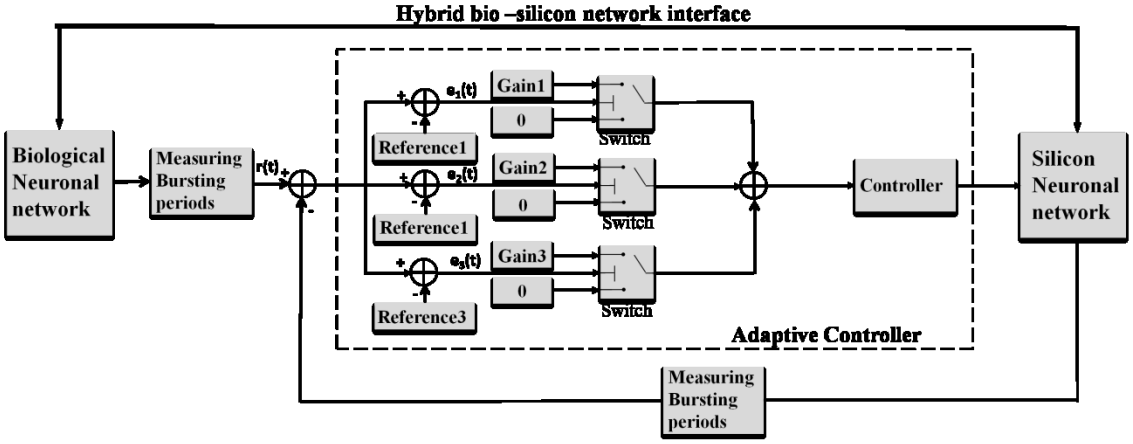


Figure 5-8: An adaptive control system for the central pattern generator prosthesis system. Blocks of measuring bursting periods are responsible for real-time sensor neuron bursting frequency; blocks of switch system are for optimizing controller gain, and the block of controller is for automatically modifying silicon neuron calculation speed. The controlled neuron is the silicon neuron LP.

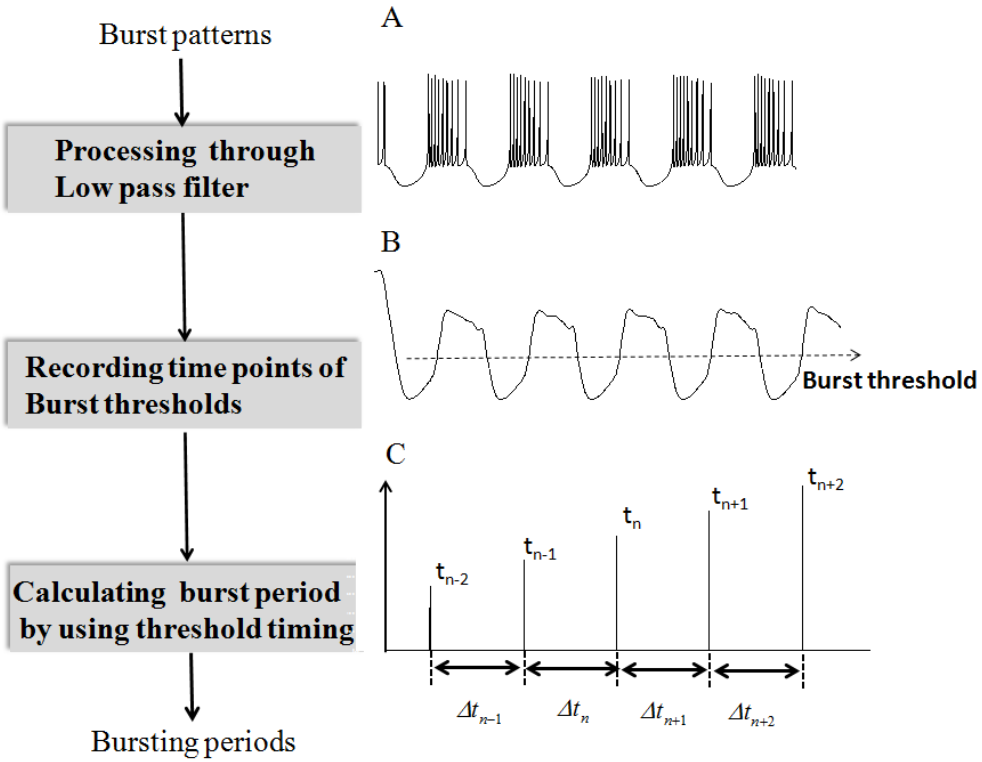


Figure 5-9: The algorithms of measuring real-time neuronal spiking period. There are three stages for computing: low-pass filter, recording and calculation.

The switch mechanism is designed as follows: by comparing with three different reference periods (slow, normal and fast), the controller can identify what bursting state the biological neuron is in. Then the switch will choose the corresponding adaptive gain for the current system. In this circuit, the three reference periods are defined as 0–0.5 seconds, 0.5–1 seconds and 1–2

seconds. And the three adaptive gain values are 1/300, 1/500 and 1/800. For example, when the biological neuron bursting period is 0.4 periods, the value belongs to reference 1 range 0–0.5 periods. This indicates that the calculated errors $e1(t)$, $e2(t)$ and $e3(t)$ are 0, 1 and 1. The gain valued 1/300 is selected as the result.

For digital controller design, three steps are considered in this system. Firstly, I employ the trigonometric function $\cos\theta$ to represent biological neuron bursting characters because of their share of the identified wave patterns. The Z-transform equation of neuron bursting behaviours is below:

$$X(z) = \frac{1 - e^{-aT}z^{-1}cos\omega T}{1 - 2e^{-aT}z^{01}cos\omega T + e^{-2aT}z^{-2}} \qquad \text{Equation 5-8}$$

Then, according to the biological recordings, parameter values a, T and ω are set as -0.7, 1 and 0.0018. The specific pyloric neuron Z-transform is below:

$$X(z) = \frac{z(z-2)}{(Z+3.414)(z-0.568)} \qquad \text{Equation 5-9}$$

After transforming biological spiking-pattern performances into digital-based Z-equations, I consider that this hybrid bio-silicon system requires stable communication performances; two control system parameters are optimized: by minimizing system setting time (T_s = 3.027), the hybrid network can achieve quick transient response while biological neuron states vary; by minimizing percentage overshoot (τ = 1%), the hybrid network can avoid overload in most cases. The controller Z-transform is below:

$$X(z) = 2.857 \times \frac{z-0.568}{z+0.149} \qquad \text{Equation 5-10}$$

5.4 Results

5.4.1 *System implementation*

The system implementation is shown in Figure 5-10. For the silicon aspect, the digital CPG and adaptive control mechanism are implemented at Xilinx Virtex-4 DSP board. The hardware architecture is designed by using software system

generator and VHDL languages, which are shown in Figure 5-10A. The details are shown in Appendix E.2.

For the biological preparations, Adult Cancer pagurus L. were obtained from local sources (Newcastle University, Dove Marine Laboratories) and kept in filtered seawater (10–12 °C). Animals were kept in ice for 20–40 minutes for anaesthetizing. The STG was pinned down in a silicone elastomer-lined (ELASTOSIL RT601, Wacker, Munich, Germany) petri dish with chilled saline (10–13 °C). The details of dissection and desheathingthe STG were performed as in [124] and [78]. The rhythmic activity patterns generated in the STG were recorded using extracellular recordings: a petroleum jelly-based cylindrical compartment was built around a section of the main motor nerve, the inferior ventricular nerve (LVN), to electrically isolate the nerve from the bath. One of two stainless steel electrode wires was placed in this compartment and the other one was placed in the bath as a reference electrode. The differential signal was recorded, filtered and amplified with an AC differential amplifier (Kaiserslautern University, Germany). The motor activities of the ganglion were monitored using an oscilloscope (DL708E; Yokogawa, Tokyo, Japan) and were recorded using a data acquisition board (CED Power, 1401) and the software Spike 2. The pyloric network image under a microscope is shown in Figure 5-10C and real-time system recording and stimulating signals are shown in Figure 5-10 D and E. The details of the network recording and mapping are shown in Appendix E.1.

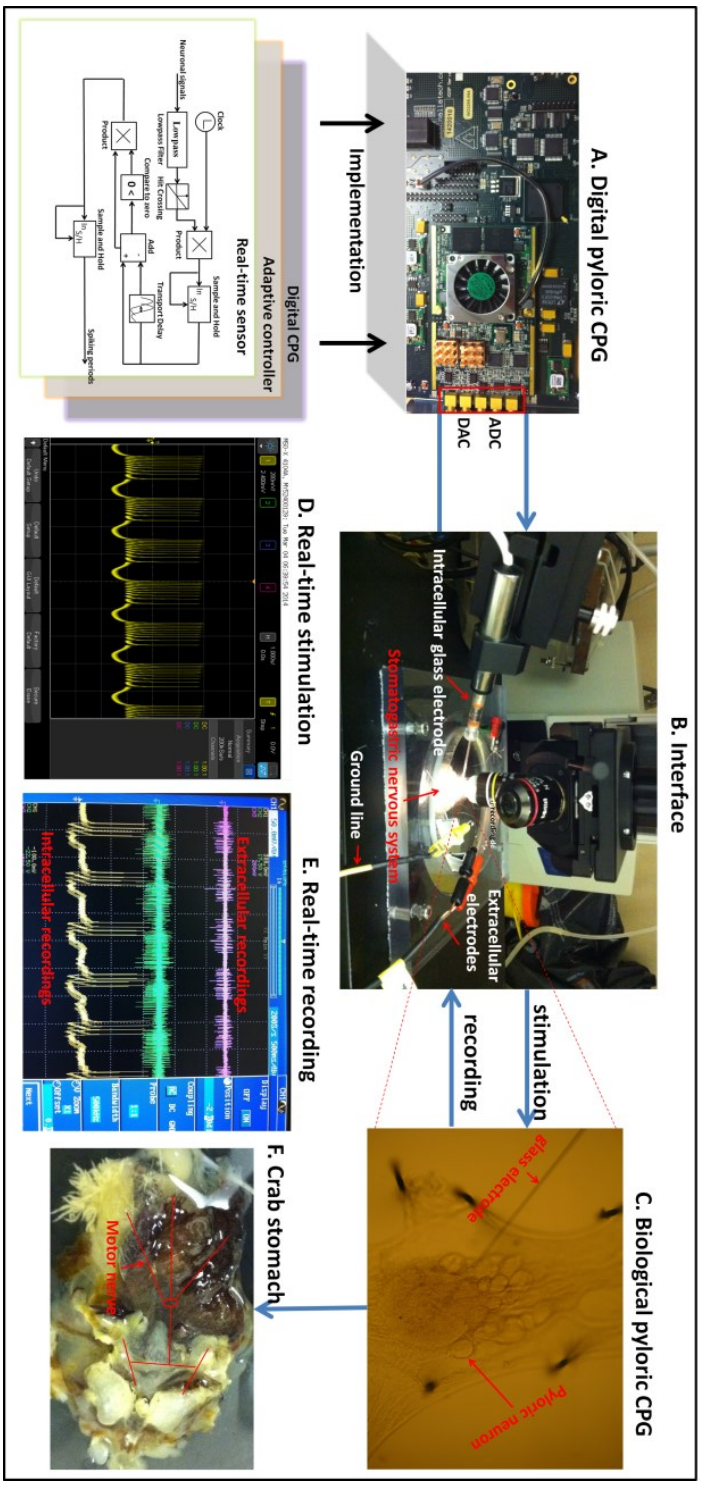


Figure 5-10: The system implementation. A is the Virtex-4 DSP platform that used to implement digital neurons and adaptive control system; B is the neural interface based on intracellular/extracellular recording techniques; C is the image of real pyloric CPG under microscope, the neurons (cycles) are clearly displayed in the picture; D and E are the real-time simulation /recording signals; D is one of the pyloric neuron outputs, E is both intracellular and extracellular recording results; F is the physical stomach muscles.

5.4.2 *Software simulation results*

I simulate biological pyloric CPG in both control and pharmacological conditions, which can correspondingly cure diseases where CPG is totally or partially damaged.

A comparison of control simulation and recording results is shown in Figure 5-11. The network rhythm phase relationship PD-LP-PY is quite similar to the biological recordings, although digital CPG displays a slightly higher spiking frequency per burst. Furthermore, the measurable values PD-onset, LP-onset and PY-offset are approximately the same values. At the bottom of Figure 5-12, the maximum errors are approximately less than 7%. However, the digital LP neuron displayed a slightly longer bursting time than the biological one, and the digital PY neuron showed a slightly later bursting start time in the rhythm. This variation can be further eliminated by optimizing the parameters. A comparison of pharmacological simulation results of LP-VD-PD and biological recordings is shown in Figure 5-13. In the biological recordings, with the commissural inputs being contacted, the subnetwork LP-PD-VD displayed a stable and regular spiking pattern. However, when the network was without commissural inputs, this system generated irregular spikes. This is because a single oscillator PD doesn't have such the ability to drive two conditional neurons bursts together. The digital subnetwork successfully replicated this behaviour in both conditions as shown in Figure 5-13B.

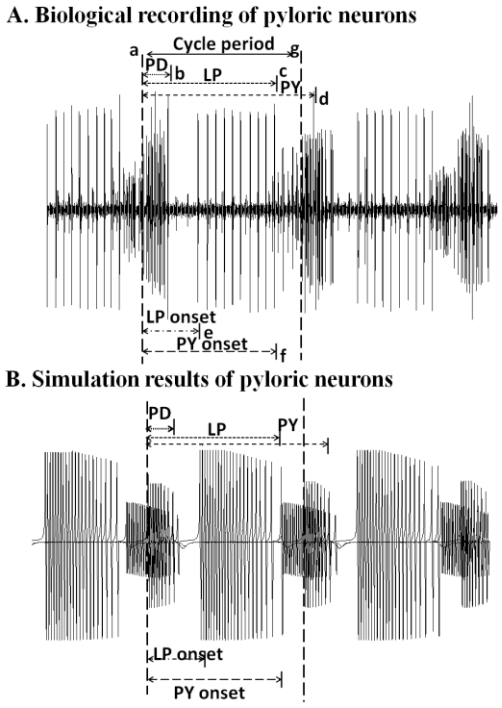


Figure 5-11: A. Biological recordings of pyloric neurons; B: simulation results of pyloric neurons. The arrow from a to g indicates pyloric period, measured as the latency from the onset of one PD neuron burst to the next. The arrow from a to e indicates the latency of PD neuron offset. The arrow from a to c indicates the latency of LP neuron offset. The arrow from a to d indicates the latency of PY neuron offset. The arrow from a to e indicates the latency of LP neuron onset. The arrow from a to f indicates the latency of PY neuron onset.

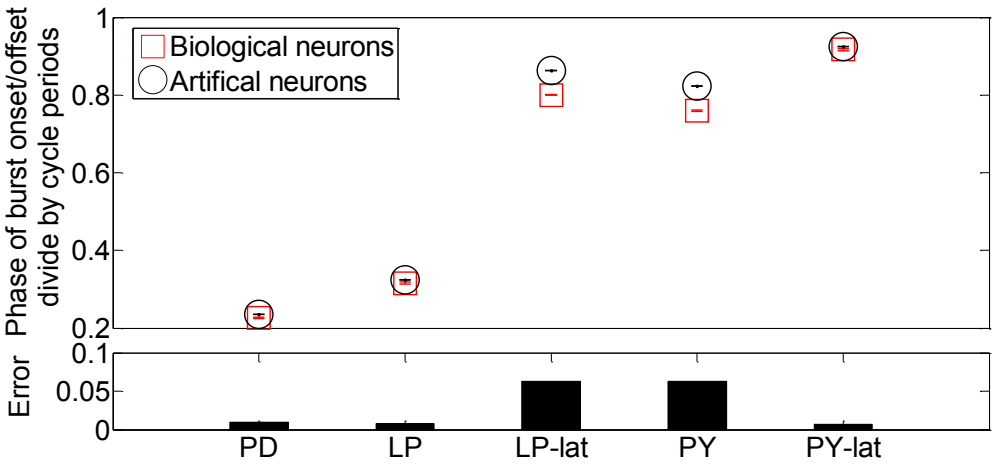


Figure 5-12: A comparison of the phase relationship between biological neurons and model neurons. The x-axis is the individual neuron name. In the top figure, the y-axis is the phase of burst onset/offset divided by cycle periods; and in the bottom figure, the y-axis is the differences between biological recordings and simulation results.

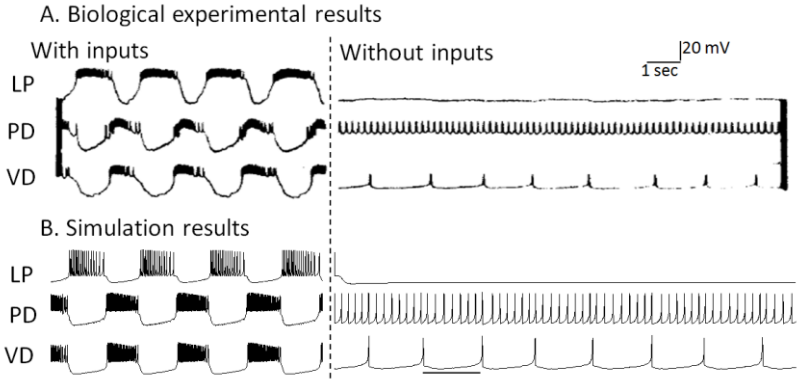


Figure 5-13: A comparison between biological recordings and simulation results of network LP-VD-PD under with and without sensory input conditions.

5.4.3 *System reliability*

The damaged biological CPG (AB, PD and LP) is simulated by using MatLab software, and the neuron LP is implemented on the FPGA as a prosthesis processor. Therefore, the neuron silicon neuron LP is the controlled target. I

artificially modify software-based neuron bursting periods to investigate control system performances.

There are two classic case studies of hybrid network reliable performances. One is the network bursting frequency, which fluctuates from standard to fast due to external stimulus, and the other one is the frequency from standard to slow due to the system becoming inactive.

The biological neurons AB, PD and LP are simulated by using MatLab software while the digital neuron LP is implemented on FPGA. The entire system is simulated in a hardware/software co-design environment.

As shown in Figure 5-14, on the left for the first case, the hybrid network with controller shows standard burst patterns while without controller it displays irregular bursts. Compared to the regular biological pyloric patterns, the network pattern with controller still maintains spiking behaviours and phase relationships. The other case is shown in Figure 5-14 on the right, where the spiking phase relationship b/a shows significant differences between the hybrid network with controller and without controller. In the with controller case, the value of b/a is approximately 0.5. However, while under without the controller state, LP maintains same bursting frequency in the network, and the value of b/a is approximately 0.8. This causes incorrect network burst pattern phase relationships and makes it less energy efficient for muscle activities. The detailed specifications of the control system are shown in Table 5-4. The closed-loop system setting time is 0.293 seconds, which is smaller than the fastest biological neuron bursting frequency of 0.5 seconds. Also, the system overshoots are approximately 1.75%. This indicates that system communications are in a reliable condition. Also, system sensitivities of input/output and noise are calculated as well. The results explain that they all show strong anti-disturbance behaviours.

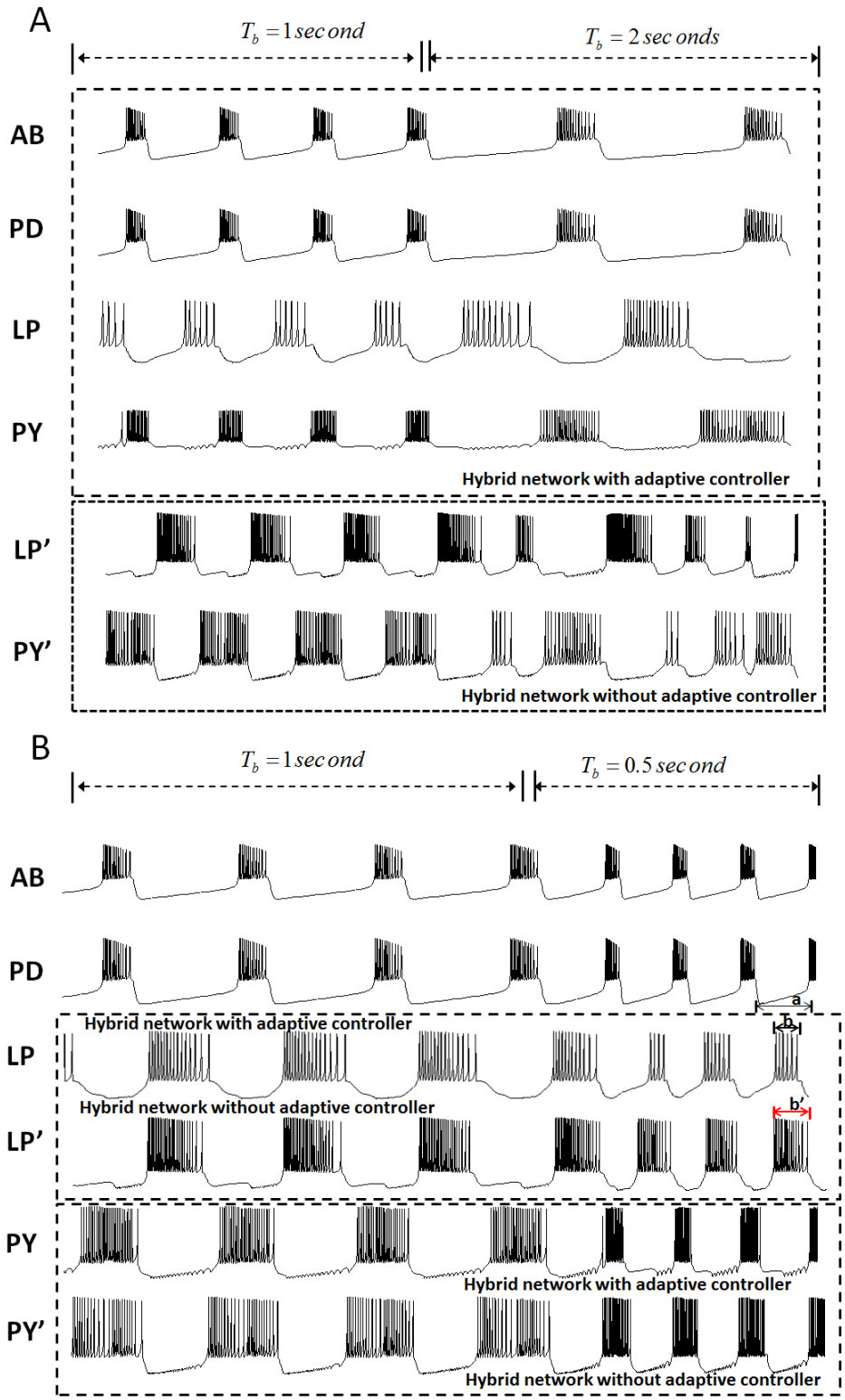


Figure 5-14: Simulation results of hybrid network. A hardware/software co-simulation to simulate system prosthesis results. The damaged CPG neurons AB, PD and PY are mimicked by using MatLab software and the prosthesis neuron LP is implemented in FPGA. In the left figure, the software-based neurons have changed their bursting periods from 1 to 2 seconds and in the right figure from 1 to 0.5 seconds. Both hybrid networks with and without controller spiking patterns are displayed.

Table 5-4: Control system specifications of step response

	System	S (input)	S (output)	S (noise)
Setting time (sec)	0.293	0.79	0.293	0.239
Overshoot (%)	1.75	0	0.987	1.75
Rise time (sec)	0.208	0.436	0.208	0.208
Steady state	-1.29	-1.2	2.29	1.29

***: S: S is the sensitivity.**

5.4.4 *Hardware implementation specifications*

I use MatLab software R2012b discrete floating point calculation as a reference to verify hardware simulation. The accuracy of the results is evaluated by using mean square error algorithms.

In each experiment, integer bits were fixed at 6, and the fraction bits were varied to explore and study the effect of truncation errors. The accuracy percentage calculation algorithm is modified to the Mean Square Error algorithm, which more precisely analyses errors quantitatively. Secondly, the reference answer is changed from hardware implementation results with a fixed 60-bit integer and 40-bit fraction system to a software discrete, floating point system. Figure 5-15 shows that there is a steady increase in the precision percentage and a gradual decrease in computational speed as the number of bits increases. Surprisingly, there is a sharp drop in computational speed between 18 and 20 bits. This is mainly due to the synthesis tool utilizing four times more embedded multipliers to calculate the algorithms when they are more than 18 bits. Here I selected a 24-bit fractional system for implementation.

The resource utilizations are shown in Table 5-5. Compared to the standard implementation technique, timing multiplexing technique only utilizes one-sixth hardware resources for implementation.

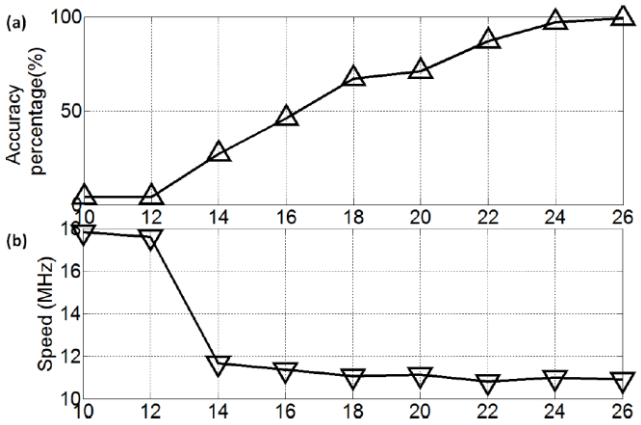


Figure 5-15: The numerical computational performances of an FPGA. (a) and (b) display system accuracy and speed performances with various fraction bits.

Table 5-5: Hardware specifications of digital CPG

Resources	TM	Standard	Improvement
Slice	6545	20417	71.4%
LUTs	8137	37960	80.8%
FIFO/RAMB	48	184	78%
DSP blocks	79	368	78.5%

5.5 Discussion

5.5.1 *Comparison of other neurorehabilitation techniques*

Traditional neurorehabilitation techniques [125] mainly refer to appropriate locomotor training, which is the facilitation and assistance of stepping-like movements with patients' legs, custom-designed reflex electrical stimulation or drug treatment. Compared to the presented technique, it is more reliable, safe and convincing. However, in most cases it cannot fundamentally solve the problem and has limited scope, because the neural circuits are still damaged. The approach aims to recover the neural circuits' behaviour essentially by using artificial neurons. Similarly, Vogelstein et al [49] demonstrated that using silicon circuits can successfully restore the damaged CPG, and hence the disabled cat can walk again. One key unique advantage of the presented system is its strong reliability. In reality, the CPG-related movements such as respiration and locomotion are always altering their speed to adapt to external environments. Therefore the prosthesis system has to be adaptive as well to solve this constraint. The adaptive mechanism in this system is capable of modifying digital processor computational speed in real time to follow biological spiking time to restore the original functionalities. The control system setting time is 0.293 seconds and the overshoots are approximately 1.75%.

5.5.2 *The advantages of the FPGA-based system*

In terms of hardware architecture design, the timing multiplexing technique successfully divides the entire neural network into several subnetworks. In a frame, each channel is responsible for the corresponding subnetwork activities. Although the entire simulation speed is relatively decreased, the hardware resources are significantly saved by up to 70%. More importantly, the information of each neuron/synapse currents can be easily fetched out in different clock cycles, which shows the advantage to network state estimation and monitoring.

Also, by taking FPGAs reconfigurable advantage [26], the implemented circuits can be modified according to the damaged neural circuit conditions, which increases the range of the presented technique applications (*e.g*. totally or partially damaged). Meanwhile, by using an auto-generation tool kit approach[17], the parameters and digital neural circuits can be easily updated in hours, such as by adding and deleting neurons/ions.

Last but not least, since the FPGA-based platform is a highly parallel computing system, it shows strong scalability when a large-scale neural model is required. Compared to the traditional CPU-based systems [126][127][128], it solves the timing constraints in the bio-silicon closed-loop system.

5.5.3 *Challenges*

The portable characters and long-term recording/stimulating (*e.g.* years) interfaces are two major issues of concern for the developed system, since a patient has to do outdoor activities, and the damage from surgery has to be maximally minimized. For the implantable aspect, the designed digital neural circuits can be transformed into ASIC directly to achieve portable device features, which may be considered in the next step of the project'. Alternatively, a custom-designed'printed circuit board integrated with FPGA-based neural circuits and micro-controller with limited sizes can also be considered as portable devices. Meanwhile, interfaces with long-term recording/stimulating performances are required at the next stage. There are some existing techniques [129][130] that can be adopted into the system in the near future.

5.6 Conclusion

In this paper, I propose a novel system for biological central pattern generator rehabilitation based on digital neural circuits. To demonstrate system feasibility and scalability, a complete biological pyloric model consisting of 14 neurons and 24 synapses is implemented on FPGA. Simulation results indicate that a silicon pyloric model can mimic real pyloric model rhythms. The mean error of five parameters between biological and silicon neurons is 7%. By applying TDM techniques, these circuits utilize one-sixth of the hardware resources of standard techniques for implementation. More importantly, the presented system shows strong reliable behaviours under different conditions; the control system setting time is 0.293 seconds and the overshoots are approximately 1.75%.

Optimization of power utilization, area and computational speed will be considered in the next step. Data reuse technique [131] can be applied for reducing the power consumption for off-chip memory, data transfer and storage, and the full pipelining method [77] can effectively save the hardware resources. Taken together, these will lead to an efficient custom-designed dynamic clamp experiment.

In the future, I am going to use this model to interact with the imperfect real pyloric network aiming to rehabilitate biological functionality via restoring biological neurons. I investigated the behaviour of the model under the impact of simulated neuromodulators (e.g. dopamine), and I also simulated the impact of losing selected neurons from the network. I expect that the FPGA model of the pyloric circuit may help to facilitate the execution of simultaneous dynamic clamp experiments with multiple STG neurons. The ultimate goal is to include sufficient details in individual neuron models to allow the replication of circuit behaviour dynamics in a wide range of physiological situations.

Chapter 6 Conclusion

This chapter gives a summary of what I have done in the digital neural circuit field. Then, based on these works, several general principles of digital neural circuit architecture design are given. Also, the potential applications and social impacts are discussed as well.

6.1 Summary

The major contribution of this work is to illustrate and investigate the profound methodologies used for designing digital neural circuits. Put simply, that is how to mimic various biological (*e.g.* ions, neurons and networks) system behaviours by using digital electronic circuits. I first introduced several classic and vital techniques such as: simulation of virtual neurons; Look-up-table (LUT) and component-based methodology; address event representation (AER); and auto generation tool kit. After that, three novel techniques (a pipeline-based multi-loop process structure, a framework-based network-on-chip structure and a reliable closed-loop system for central pattern generator rehabilitation) were developed as major contributions. At the end, I briefly explained the impacts, meanings and implications of the developed work and the issues for the next step.

Specifically, I developed the first digital optogenetic neuron using reconfigurable hardware that contains 13 different types of ion channel. A pipelining-based multi-loop process architecture is presented to implement a neural model. The results indicate that it cannot only reproduce normal neural burst patterns but also pharmacological burst patterns. The system can achieve approximately 76,618 operations per neuron in 1 ms, which is five times faster than the latest digital cerebellum neuron system [20].

Furthermore, a frame-based network-on-chip (NoC) architecture has been developed to implement a granular-layer model of the cerebellum with approximately 100,000 neurons. The system can not only meet the biological real-time computing requirement (it only takes 25.6 ms to mimic 1 s of real-world activities), but can also avoid NoC architecture package traffic congestion by using frame mastering. After verification of on-board simulation results, the design can be readily adapted for real-time closed-loop *in vitro* or *in vivo* experiments and as a potential neuro-prosthetics tool for future experimental and clinical applications owing to its high computational power, flexibility, scalability and power efficiency.

Finally, a concept of FPGA-based hybrid bio-silicon integration is developed to restore the biological pyloric central pattern generator (CPG) functionalities. Potentially this can be one of the most important neuroscience applications for

digital neural circuits. The simulation results indicate that the presented system can successfully repair damaged CPG behaviours in different situations.

6.2 Principles of designing digital neural circuits

Based on the research findings, there are several general principles for hardware architecture design of implementing a large-scale neural network with high bio-plausibility.

The first is to use multi-core architecture with pipelining technique to mimic large-scale neuron activities. Multi-core architecture has the ability to reproduce biological highly parallel computing performances; and timing multiplexing or pipelining technique utilizes the digital speed advantages (GHz) to implement a number of neurons in the same physical hardware resources. Previous computational platforms such as SpiNNaker [3], NeuroGrid [104] and IBM chip [11] are all similar to this architecture. The implementation generally requires appropriate memory space for storing calculation neuronal states and distributed memory location to increase communication bandwidth.

The second is the individual processor design. The architecture should be heterogeneous and multiple-layer based to meet the network bio-plausibility requirements. In a neuron, the membrane voltage alternation always introduces related ion concentration fluctuation such as calcium [132] or ChR2 [73]. Therefore, the calcium-dependent ion channels or other channels will update their gate behaviours to shape the final neuron spiking patterns. A digital processor should have a responsibility to mimic all these ion channel dynamics including these closed-loop process mechanisms. The multiple-layer- or heterogeneous-based cores are proper candidates but will significantly cost more in terms of hardware resources and limits system integration step. Optimization of resource utilizations and speed is necessary in the final design stage.

The third is system level optimization. Because the system itself is massive, it has to be carefully optimized to achieve the best computing performances. There are many issues involved in this topic. For example, the trade-off between the number of processors and the implemented neuron numbers per processor in the architecture: the more the number of processors implemented, the faster the computational speed and the more resources utilized, while the

more the number of implemented neurons in a processor, the slower the computational speed but the fewer the number of resources utilized. Also, the neural data-path design generally decides the critical path and power consumption, and the two implementation approaches of operation balance and latency balance will lead to totally different hardware specifications. Last but not least, the address event representation techniques should be custom designed to map the neural network connectivity. In general, the biological neurons are highly connected but the connections are quite varied in detail. Different routing strategies (*e.g.* uni-cast, multi-cast and board-cast) are correspondingly map different connection types[133].

6.3 Future work

There are three main areas for the future work: optimization of the methodology of digital neural circuit design, neuronal-machine prosthesis system experimental verification and bio-inspired device development.

First, based on the previous two architecture designs, we have drawn some conclusions for the digital neural circuit design. However, there are still some major issues that need to be further considered. For example, in terms of a single neuron design, there is a variety of different architectures for minimal power consumption, minimal areas and fastest speed. The fundamental principles of these designs should be investigated and summarized. Next, synaptic connections and network sizes decide the architecture's main features. A general and systematic approach should be developed for implementing different kinds of neural network. In the end, the high-level optimization method of the entire architecture still needs to be improved.

Second, developed silicon neural network cerebellums and optogenetic neurons will be further taken into biological experiments. By interacting with the real biology, we will study how the bio-silicon system works and how to further custom modify silicon part to achieve system adaptive performance

Finally, since developed artificial neural systems have the ability to capture major biological intelligence, there is a potential that we will transfer these presented systems into practical devices for intelligent tasks such as environment detecting and monitoring.

References

[1] C. Mead, “Neuromorphic electronic systems,” *Proc. IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[2] S. Furber and S. Temple, “Neural systems engineering.,” *J. R. Soc. Interface*, vol. 4, no. 13, pp. 193–206, Apr. 2007.

[3] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, “Overview of the SpiNNaker System Architecture,” *IEEE Trans. Comput.*, vol. 62, no. 12, pp. 2454–2467, Dec. 2013.

[4] F. Aubépart and N. Franceschini, “Bio-inspired optic flow sensors based on FPGA: Application to Micro-Air-Vehicles,” *Microprocess. Microsyst.*, vol. 31, no. 6, pp. 408–419, Sep. 2007.

[5] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. Saighi, T. Serrano-Gotarredona, J. Wijekoon, Y. Wang, and K. Boahen, “Neuromorphic silicon neuron circuits.,” *Front. Neurosci.*, vol. 5, p. 73, Jan. 2011.

[6] E. R. Kandel, H. Markram, P. M. Matthews, R. Yuste, and C. Koch, “Neuroscience thinks big (and collaboratively).,” *Nat. Rev. Neurosci.*, vol. 14, no. 9, pp. 659–64, Sep. 2013.

[7] “Scientists threaten to boycott €1.2bn Human Brain Project | Science | The Guardian.” [Online]. Available: http://www.theguardian.com/science/2014/jul/07/human-brain-project-researchers-threaten-boycott. [Accessed: 18-Jul-2014].

[8] A. P. Alivisatos, M. Chun, G. M. Church, R. J. Greenspan, M. L. Roukes, and R. Yuste, “The brain activity map project and the challenge of functional connectomics.,” *Neuron*, vol. 74, no. 6, pp. 970–4, Jun. 2012.

[9] S. B. Furber, F. Galluppi, S. Temple, and L. . Plana, “The SpiNNaker Project,” *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[10] “IBM Research: Neurosynaptic chips.” IBM Corporation, 16-Dec-2013.

[11] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science (80-.).*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

[12] “Introducing Qualcomm Zeroth Processors: Brain-Inspired Computing | Qualcomm.” [Online]. Available:

http://www.qualcomm.com/media/blog/2013/10/10/introducing-qualcomm-zeroth-processors-brain-inspired-computing. [Accessed: 11-Jul-2014].

[13] G. Rachmuth, H. Z. Shouval, M. F. Bear, and C.-S. Poon, “A biophysically-based neuromorphic model of spike rate- and timing-dependent plasticity.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 108, no. 49, pp. E1266–74, Dec. 2011.

[14] A. L. HODGKIN and A. F. HUXLEY, “A quantitative description of membrane current and its application to conduction and excitation in nerve.,” *J. Physiol.*, vol. 117, no. 4, pp. 500–44, Aug. 1952.

[15] E. L. Graas, E. A. Brown, and R. H. Lee, “An FPGA-based approach to high-speed simulation of conductance-based neuron models.,” *Neuroinformatics*, vol. 2, no. 4, pp. 417–36, Jan. 2004.

[16] A. Cassidy, S. Denham, P. Kanold, and A. Andreou, “FPGA Based Silicon Spiking Neural Array,” in *2007 IEEE Biomedical Circuits and Systems Conference*, 2007, pp. 75–78.

[17] R. K. Weinstein, M. S. Reid, and R. H. Lee, “Methodology and design flow for assisted neural-model implementations in FPGAs.,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 15, no. 1, pp. 83–93, Mar. 2007.

[18] 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 *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, 2012, pp. 133–140.

[19] K. Cheung, S. R. Schultz, and W. Luk, *Artificial Neural Networks and Machine Learning – ICANN 2012*, vol. 7552. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 113–120.

[20] G. Smaragdos, S. Isaza, M. F. van Eijk, I. Sourdis, and C. Strydis, “FPGA-based biophysically-meaningful modeling of olivocerebellar neurons,” in *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays - FPGA ’14*, 2014, pp. 89–98.

[21] M. D. Godfrey, “Introduction to‘The First Draft Report on the EDVAC’ by John von Neumann.”

[22] T. Oguchi, M. Higuchi, T. Uno, M. Kamaya, and M. Suzuki, “A single-chip graphic display controller,” in *1981 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, 1981, vol. XXIV, pp. 170–171.

[23] A. K. Fidjeland and M. P. Shanahan, “Accelerated simulation of spiking neural networks using GPUs,” in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1–8.

[24] D. Luebke and G. Humphreys, “How GPUs Work,” *Computer (Long. Beach. Calif).*, vol. 40, no. 2, pp. 96–100, Feb. 2007.

[25] “Xilinx, ASIC vendors talk licensing | EE Times.” [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1180867. [Accessed: 31-Oct-2014].

[26] I. Kuon, R. Tessier, and J. Rose, “FPGA Architecture: Survey and Challenges,” *Found. Trends® Electron. Des. Autom.*, vol. 2, no. 2, pp. 135–253, Feb. 2007.

[27] D. Frohman-Bentchkowsky, “A fully-decoded 2048-bit electrically-programmable MOS ROM,” in *1971 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, 1971, vol. XIV, pp. 80–81.

[28] R. Cuppens, C. D. Hartgring, J. F. Verwey, H. L. Peek, F. A. H. Vollebragt, E. G. M. Devens, and I. A. Sens, “An EEPROM for microprocessors and custom logic,” *IEEE J. Solid-State Circuits*, vol. 20, no. 2, pp. 603–608, Apr. 1985.

[29] D. C. Guterman, I. H. Rimawi, R. D. Halvorson, and D. J. McElroy, “An electrically alterable nonvolatile memory cell using a floating-gate structure,” *IEEE J. Solid-State Circuits*, vol. 14, no. 2, pp. 498–508, Apr. 1979.

[30] H.-C. Hsieh, K. Dong, J. Y. Ja, R. Kanazawa, L. T. Ngo, L. G. Tinkey, W. S. Carter, and R. H. Freeman, “A 9000-gate user-programmable gate array,” in *Proceedings of the IEEE 1988 Custom Integrated Circuits Conference*, 1988, pp. 15.3/1–15.3/7.

[31] E. Hamdy, J. McCollum, S.-O. Chen, S. Chiang, S. Eltoukhy, J. Chang, T. Speers, and A. Mohsen, “Dielectric based antifuse for logic and memory ICs,” in *Technical Digest., International Electron Devices Meeting*, 1988, pp. 786–789.

[32] M. P. Nusbaum and M. P. Beenhakker, “A small-systems approach to motor pattern generation.,” *Nature*, vol. 417, no. 6886, pp. 343–50, May 2002.

[33] T. Yamazaki and S. Tanaka, “A spiking network model for passage-of-time representation in the cerebellum.,” *Eur. J. Neurosci.*, vol. 26, no. 8, pp. 2279–2292, Oct. 2007.

[34] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, *Principles of neural science*. Elsevier, 1991, p. 1135.

[35] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Massachusetts Institute of Technology Press, 2001, p. 460.

[36] J. H. Maunsell and D. C. Van Essen, “Functional properties of neurons in middle temporal visual area of the macaque monkey. I. Selectivity for stimulus direction, speed, and orientation,” *J Neurophysiol*, vol. 49, no. 5, pp. 1127–1147, May 1983.

[37] T. S. T. Mak, G. Rachmuth, K.-P. Lam, and C.-S. Poon, “A component-based FPGA design framework for neuronal ion channel dynamics simulations.,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 14, no. 4, pp. 410–8, Dec. 2006.

[38] J. P. Miller and A. I. Selverston, “Mechanisms underlying pattern generation in lobster stomatogastric ganglion as determined by selective inactivation of identified neurons. II. Oscillatory properties of pyloric neurons.,” *J. Neurophysiol.*, vol. 48, no. 6, pp. 1378–91, Dec. 1982.

[39] Z. Zhou, J. Champagnat, and C. S. Poon, “Phasic and long-term depression in brainstem nucleus tractus solitarius neurons: differing roles of AMPA receptor desensitization.,” *J. Neurosci.*, vol. 17, no. 14, pp. 5349–56, Jul. 1997.

[40] J. J. Renger, C. Egles, and G. Liu, “A Developmental Switch in Neurotransmitter Flux Enhances Synaptic Efficacy by Affecting AMPA Receptor Activation,” *Neuron*, vol. 29, no. 2, pp. 469–484, Feb. 2001.

[41] R. J. Butera, J. Rinzel, and J. C. Smith, “Models of respiratory rhythm generation in the pre-Bötzinger complex. I. Bursting pacemaker neurons.,” *J. Neurophysiol.*, vol. 82, no. 1, pp. 382–97, Jul. 1999.

[42] K. D. M. L. F. A. N. G. P. Sen Song, “Competitive Hebbian Learning through Spike-Timing-Dependent Synaptic Plasticity.”

[43] M. Schwartz, “Telecommunication networks: protocols, modeling and analysis,” Jan. 1986.

[44] K. A. Boahen, “Point-to-point connectivity between neuromorphic chips using address events,” *IEEE Trans. Circuits Syst. II Analog Digit. Signal Process.*, vol. 47, no. 5, pp. 416–434, May 2000.

[45] M. Mahowald, “VLSI analogs of neuronal visual processing: a synthesis of form and function.” 12-Sep-1992.

[46] D. K. Warland, P. Reinagel, and M. Meister, “Decoding visual information from a population of retinal ganglion cells.,” *J. Neurophysiol.*, vol. 78, no. 5, pp. 2336–50, Nov. 1997.

[47] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic, and P. J. Hazewindus, “The design of an asynchronous microprocessor,” *ACM SIGARCH Comput. Archit. News*, vol. 17, no. 4, pp. 99–110, Jun. 1989.

[48] Bo Wen and K. Boahen, “A silicon cochlea with active coupling.,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 3, no. 6, pp. 444–55, Dec. 2009.

[49] R. J. Vogelstein, F. Tenore, L. Guevremont, R. Etienne-Cummings, and V. K. Mushahwar, “A silicon central pattern generator controls locomotion in vivo.,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 2, no. 3, pp. 212–22, Sep. 2008.

[50] B. Dworakowska and K. Dołowy, “Ion channels-related diseases.,” *Acta Biochim. Pol.*, vol. 47, no. 3, pp. 685–703, Jan. 2000.

[51] J. Luo, G. Coapes, T. Mak, T. Yamazaki, C. Tin, and P. Degenaar, “A Scalable FPGA-based Cerebellum for Passage-of-Time Representation,” in *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 2014.

[52] L. . Abbott, “Lapicque’s introduction of the integrate-and-fire model neuron (1907),” *Brain Res. Bull.*, vol. 50, no. 5–6, pp. 303–304, Nov. 1999.

[53] E. M. Izhikevich, “Simple model of spiking neurons.,” *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–72, Jan. 2003.

[54] J. L. Hindmarsh and R. M. Rose, “A model of neuronal bursting using three coupled first order differential equations.,” *Proc. R. Soc. Lond. B. Biol. Sci.*, vol. 221, no. 1222, pp. 87–102, Mar. 1984.

[55] R. D. Traub, R. K. Wong, R. Miles, and H. Michelson, “A model of a CA3 hippocampal pyramidal neuron incorporating voltage-clamp data on intrinsic conductances.,” *J. Neurophysiol.*, vol. 66, no. 2, pp. 635–50, Aug. 1991.

[56] G. Coapes, T. Mak, J. W. Luo, A. Yakovlev, and C.-S. Poon, “A scalable FPGA-based design for field programmable large-scale ion channel simulations,” in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 112–119.

[57] K. Nikolic, N. Grossman, M. S. Grubb, J. Burrone, C. Toumazou, and P. Degenaar, “Photocycles of channelrhodopsin-2.,” *Photochem. Photobiol.*, vol. 85, no. 1, pp. 400–11.

[58] C. Soto-Treviño, P. Rabbah, E. Marder, and F. Nadim, “Computational model of electrically coupled, intrinsically distinct pacemaker neurons.,” *J. Neurophysiol.*, vol. 94, no. 1, pp. 590–604, Jul. 2005.

[59] K. Deisseroth, G. Feng, A. K. Majewska, G. Miesenböck, A. Ting, and M. J. Schnitzer, “Next-generation optical technologies for illuminating genetically targeted brain circuits.,” *J. Neurosci.*, vol. 26, no. 41, pp. 10380–6, Oct. 2006.

[60] J. J. Mancuso, J. Kim, S. Lee, S. Tsuda, N. B. H. Chow, and G. J. Augustine, “Optogenetic probing of functional brain circuitry.,” *Exp. Physiol.*, vol. 96, no. 1, pp. 26–33, Jan. 2011.

[61] G. Nagel, T. Szellas, W. Huhn, S. Kateriya, N. Adeishvili, P. Berthold, D. Ollig, P. Hegemann, and E. Bamberg, “Channelrhodopsin-2, a directly light-gated cation-selective membrane channel.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 100, no. 24, pp. 13940–5, Nov. 2003.

[62] J. Y. Lin, “A user’s guide to channelrhodopsin variants: features, limitations and future developments.,” *Exp. Physiol.*, vol. 96, no. 1, pp. 19–25, Jan. 2011.

[63] P. Brown and N. Dale, “Spike-independent release of ATP from Xenopus spinal neurons evoked by activation of glutamate receptors.,” *J. Physiol.*, vol. 540, no. Pt 3, pp. 851–60, May 2002.

[64] A. J. Tierney and R. M. Harris-Warrick, “Physiological role of the transient potassium current in the pyloric circuit of the lobster stomatogastric ganglion.,” *J. Neurophysiol.*, vol. 67, no. 3, pp. 599–609, Mar. 1992.

[65] J. C. Smith, R. J. Butera, N. Koshiya, C. Del Negro, C. G. Wilson, and S. M. Johnson, “Respiratory rhythm generation in neonatal and adult mammals: the hybrid pacemaker–network model,” *Respir. Physiol.*, vol. 122, no. 2–3, pp. 131–147, Sep. 2000.

[66] R. H. Lee and C. J. Heckman, “Adjustable amplification of synaptic input in the dendrites of spinal motoneurons in vivo.,” *J. Neurosci.*, vol. 20, no. 17, pp. 6734–40, Sep. 2000.

[67] A. Alaburda, J.-F. Perrier, and J. Hounsgaard, “An M-like outward current regulates the excitability of spinal motoneurones in the adult turtle.,” *J. Physiol.*, vol. 540, no. Pt 3, pp. 875–81, May 2002.

[68] B. Santoro, S. Chen, A. Luthi, P. Pavlidis, G. P. Shumyatsky, G. R. Tibbs, and S. A. Siegelbaum, “Molecular and functional heterogeneity of hyperpolarization-activated pacemaker channels in the mouse CNS.,” *J. Neurosci.*, vol. 20, no. 14, pp. 5264–75, Jul. 2000.

[69] S. Hooper and E. Marder, “Modulation of the lobster pyloric rhythm by the peptide proctolin,” *J. Neurosci.*, vol. 7, no. 7, pp. 2097–2112, Jul. 1987.

[70] P. Sah and E. S. Louise Faber, “Channels underlying neuronal calcium-activated potassium currents,” *Prog. Neurobiol.*, vol. 66, no. 5, pp. 345–353, Apr. 2002.

[71] B. R. Johnson, P. Kloppenburg, and R. M. Harris-Warrick, “Dopamine modulation of calcium currents in pyloric neurons of the lobster stomatogastric ganglion.,” *J. Neurophysiol.*, vol. 90, no. 2, pp. 631–43, Aug. 2003.

[72] F. Buchholtz, J. Golowasch, I. R. Epstein, and E. Marder, “Mathematical model of an identified stomatogastric ganglion neuron.,” *J. Neurophysiol.*, vol. 67, no. 2, pp. 332–40, Feb. 1992.

[73] N. Grossman, K. Nikolic, C. Toumazou, and P. Degenaar, “Modeling study of the light stimulation of a neuron cell with channelrhodopsin-2 mutants.,” *IEEE Trans. Biomed. Eng.*, vol. 58, no. 6, pp. 1742–51, Jun. 2011.

[74] J. Golowasch and E. Marder, “Ionic currents of the lateral pyloric neuron of the stomatogastric ganglion of the crab.,” *J. Neurophysiol.*, vol. 67, no. 2, pp. 318–31, Feb. 1992.

[75] L. M. Hurley and K. Graubard, “Pharmacologically and functionally distinct calcium currents of stomatogastric neurons.,” *J. Neurophysiol.*, vol. 79, no. 4, pp. 2070–81, Apr. 1998.

[76] J. Becker, M. Platzner, and S. Vernalde, Eds., *Field Programmable Logic and Application*, vol. 3203. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[77] G. Coapes, T. Mak, J. W. Luo, A. Yakovlev, and C.-S. Poon, “A scalable FPGA-based design for field programmable large-scale ion channel simulations,” in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 112–119.

[78] J. W. Luo, T. Mak, B. Yu, P. Andras, and A. Yakovlev, “Towards neuro-silicon interface using reconfigurable dynamic clamping.,” in *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 2011, vol. 2011, pp. 6389–92.

[79] W. Al-Atabany, B. McGovern, K. Mehran, R. Berlinguer-Palmini, and P. Degenaar, “A processing platform for optoelectronic/optogenetic retinal prosthesis.,” *IEEE Trans. Biomed. Eng.*, vol. 60, no. 3, pp. 781–91, Mar. 2013.

[80] P. Degenaar, N. Grossman, M. A. Memon, J. Burrone, M. Dawson, E. Drakakis, M. Neil, and K. Nikolic, “Optobionic vision--a new genetically enhanced light on retinal prosthesis.,” *J. Neural Eng.*, vol. 6, no. 3, p. 035007, Jun. 2009.

[81] J. M. Barrett, R. Berlinguer-Palmini, and P. Degenaar, “Optogenetic approaches to retinal prosthesis.,” *Vis. Neurosci.*, vol. 31, no. 4–5, pp. 345–54, Sep. 2014.

[82] R. B. Ivry and R. M. C. Spencer, “The neural representation of time,” *Curr. Opin. Neurobiol.*, vol. 14, no. 2, pp. 225–232, Apr. 2004.

[83] J. D. Schmahmann, “Disorders of the cerebellum: ataxia, dysmetria of thought, and the cerebellar cognitive affective syndrome.,” *J. Neuropsychiatry Clin. Neurosci.*, vol. 16, no. 3, pp. 367–78, Jan. 2004.

[84] K. M. Christian and R. F. Thompson, “Neural substrates of eyeblink conditioning: acquisition and retention.,” *Learn. Mem.*, vol. 10, no. 6, pp. 427–55.

[85] M. D. Mauk and N. H. Donegan, “A model of Pavlovian eyelid conditioning based on the synaptic organization of the cerebellum.,” *Learn. Mem.*, vol. 4, no. 1, pp. 130–58.

[86] T. Yamazaki and S. Tanaka, “Computational models of timing mechanisms in the cerebellar granular layer.,” *Cerebellum*, vol. 8, no. 4, pp. 423–32, Dec. 2009.

[87] J. W. Moore, J. E. Desmond, and N. E. Berthier, “Adaptively timed conditioned responses and the cerebellum: A neural network approach,” *Biol. Cybern.*, vol. 62, no. 1, pp. 17–28, 1989.

[88] J. E. Desmond and J. W. Moore, “Biological Cybernetics Adaptive Timing in Neural Networks : The Conditioned Response,” vol. 415, pp. 405–415, 1988.

[89] D. Bullock, J. C. Fiala, and S. Grossberg, “A neural model of timed response learning in the cerebellum,” *Neural Networks*, vol. 7, no. 6–7, pp. 1101–1114, Jan. 1994.

[90] M. Fujita, “Adaptive filter model of the cerebellum,” *Biol. Cybern.*, vol. 45, no. 3, pp. 195–206, 1982.

[91] T. Yamazaki and S. Tanaka, “Neural Modeling of an Internal Clock,” *Neural Comput.*, vol. 17, no. 5, pp. 1032–1058, May 2005.

[92] T. Yamazaki and J. Igarashi, “Realtime cerebellum: a large-scale spiking network model of the cerebellum that runs in realtime using a graphics processing unit.,” *Neural Netw.*, vol. 47, pp. 103–11, Nov. 2013.

[93] M. Wang, B. Yan, J. Hu, and P. Li, “Simulation of large neuronal networks with biophysically accurate models on graphics processors,” in *The 2011 International Joint Conference on Neural Networks*, 2011, pp. 3184–3193.

[94] C. Hofstoetter, M. Gil, K. Eng, G. Indiveri, M. Mintz, J. Kramer, and P. F. Verschure, “The Cerebellum Chip: an Analog VLSI Implementation of a Cerebellar Model of Classical Conditioning,” in *Advances in Neural Information Processing Systems*, 2004, pp. 577–584.

[95] S. A. Bamford, R. Hogri, A. Giovannucci, A. H. Taub, I. Herreros, P. F. M. J. Verschure, M. Mintz, and P. Del Giudice, “A VLSI field-programmable mixed-signal array to perform neural signal processing and neural modeling in a prosthetic system.,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 20, no. 4, pp. 455–67, Jul. 2012.

[96] J. Luo, G. Coapes, P. Degenaar, T. Mak, T. Yamazaki, and C. Tin, “A Real-time Silicon Cerebellum Spiking Neural Model based on FPGA,” in *International Symposium on Integrated Circuits (ISIC)*, 2014.

[97] Y. Kishimoto, S. Kawahara, Y. Kirino, H. Kadotani, Y. Nakamura, M. Ikeda, and T. Yoshioka, “Conditioned eyeblink response is impaired in

mutant mice lacking NMDA receptor subunit NR2A.," *Neuroreport*, vol. 8, no. 17, pp. 3717–21, Dec. 1997.

[98] 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 Netw.*, vol. 22, no. 5–6, pp. 791–800.

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

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

[101] T. T. Mak, P. Sedcole, P. K. Cheung, and W. Luk, "On-FPGA Communication Architectures and Design Factors," in *2006 International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–8.

[102] B. L. Sabatini and W. G. Regehr, "Timing of synaptic transmission.," *Annu. Rev. Physiol.*, vol. 61, pp. 521–42, Jan. 1999.

[103] S. Carrillo, J. Harkin, L. J. McDaid, S. Pande, S. Cawley, B. McGinley, and F. Morgan, "Hierarchical Network-on-Chip and Traffic Compression for Spiking Neural Network Implementations," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, 2012, pp. 83–90.

[104] B. V. Benjamin, Peiran Gao, E. McQuinn, S. Choudhary, A. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. . Merolla, and K. Boahen, "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[105] R. J. Vogelstein, U. Mallik, E. Culurciello, G. Cauwenberghs, and R. Etienne-Cummings, "A multichip neuromorphic system for spike-based visual information processing.," *Neural Comput.*, vol. 19, no. 9, pp. 2281–300, Sep. 2007.

[106] A. Cassidy, A. G. Andreou, and J. Georgiou, "Design of a one million neuron single FPGA neuromorphic system for real-time multimodal scene analysis," in *2011 45th Annual Conference on Information Sciences and Systems*, 2011, pp. 1–6.

[107] A. S. Cassidy, J. Georgiou, and A. G. Andreou, "Design of silicon brains in the nano-CMOS era: spiking neurons, learning synapses and neural architecture optimization.," *Neural Netw.*, vol. 45, pp. 4–26, Sep. 2013.

[108] Y. Ugawa, K. Genba-Shimizu, J. C. Rothwell, M. Iwata, and I. Kanazawa, "Suppression of motor cortical excitability by electrical stimulation over the cerebellum in ataxia.," *Ann. Neurol.*, vol. 36, no. 1, pp. 90–6, Jul. 1994.

[109] B. McGovern, R. B. Palmini, N. Grossman, E. Drakakis, V. Poher, M. A. A. Neil, and P. Degenaar, “A New Individually Addressable Micro-LED Array for Photogenetic Neural Stimulation.,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 4, no. 6, pp. 469–76, Dec. 2010.

[110] X. Liu, S. Ramirez, P. T. Pang, C. B. Puryear, A. Govindarajan, K. Deisseroth, and S. Tonegawa, “Optogenetic stimulation of a hippocampal engram activates fear memory recall.,” *Nature*, vol. 484, no. 7394, pp. 381–5, Apr. 2012.

[111] S. Kim, P. Tathireddy, R. A. Normann, and F. Solzbacher, “Thermal impact of an active 3-D microelectrode array implanted in the brain.,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 15, no. 4, pp. 493–501, Dec. 2007.

[112] J. Elbaum and D. M. Benson, Eds., *Acquired Brain Injury*. New York, NY: Springer New York, 2007.

[113] E. Galante, L. Gazzi, and S. Caffarra, “Psychological activities in neurorehabilitation: from research to clinical practice.,” *G. Ital. Med. Lav. Ergon.*, vol. 33, no. 1 Suppl A, pp. A19–28.

[114] T. W. Berger, R. E. Hampson, D. Song, A. Goonawardena, V. Z. Marmarelis, and S. A. Deadwyler, “A cortical neural prosthesis for restoring and enhancing memory.,” *J. Neural Eng.*, vol. 8, no. 4, p. 046017, Aug. 2011.

[115] E. Marder and D. Bucher, “Understanding circuit dynamics using the stomatogastric nervous system of lobsters and crabs.,” *Annu. Rev. Physiol.*, vol. 69, pp. 291–316, Jan. 2007.

[116] E. Marder and A. L. Taylor, “Multiple models to capture the variability in biological neurons and networks.,” *Nat. Neurosci.*, vol. 14, no. 2, pp. 133–8, Feb. 2011.

[117] R. Grashow, T. Brookings, and E. Marder, “Compensation for variable intrinsic neuronal excitability by circuit-synaptic interactions.,” *J. Neurosci.*, vol. 30, no. 27, pp. 9145–56, Jul. 2010.

[118] A. L. Weaver and S. L. Hooper, “Follower neurons in lobster (Panulirus interruptus) pyloric network regulate pacemaker period in complementary ways.,” *J. Neurophysiol.*, vol. 89, no. 3, pp. 1327–38, Mar. 2003.

[119] S. Clemens, J. C. Massabuau, P. Meyrand, and J. Simmers, “A modulatory role for oxygen in shaping rhythmic motor output patterns of neuronal networks.,” *Respir. Physiol.*, vol. 128, no. 3, pp. 299–315, Nov. 2001.

[120] J. S. Eisen and E. Marder, “Mechanisms underlying pattern generation in lobster stomatogastric ganglion as determined by selective inactivation of identified neurons. III. Synaptic connections of electrically coupled pyloric neurons.,” *J. Neurophysiol.*, vol. 48, no. 6, pp. 1392–1415, Dec. 1982.

[121] E. Marder and R. L. Calabrese, “Principles of rhythmic motor pattern generation.,” *Physiol. Rev.*, vol. 76, no. 3, pp. 687–717, Jul. 1996.

[122] A. A. Sharp, L. F. Abbott, and E. Marder, “Artificial electrical synapses in oscillatory networks.,” *J. Neurophysiol.*, vol. 67, no. 6, pp. 1691–4, Jun. 1992.

[123] A. Destexhe, Z. F. Mainen, and T. J. Sejnowski, “An Efficient Method for Computing Synaptic Conductances Based on a Kinetic Model of Receptor Binding,” *Neural Comput.*, vol. 6, no. 1, pp. 14–18, Jan. 1994.

[124] G. J. Gutierrez and R. G. Grashow, “Cancer borealis stomatogastric nervous system dissection.,” *J. Vis. Exp.*, no. 25, p. e1207, Jan. 2009.

[125] M. Hubli and V. Dietz, “The physiological basis of neurorehabilitation--locomotor training after spinal cord injury.,” *J. Neuroeng. Rehabil.*, vol. 10, no. 1, p. 5, Jan. 2013.

[126] I. Raikov, A. Preyer, and R. J. Butera, “MRCI: a flexible real-time dynamic clamp system for electrophysiology experiments.,” *J. Neurosci. Methods*, vol. 132, no. 2, pp. 109–23, Jan. 2004.

[127] T. Nowotny, A. Szucs, R. D. Pinto, and A. I. Selverston, “StdpC: a modern dynamic clamp.,” *J. Neurosci. Methods*, vol. 158, no. 2, pp. 287–99, Dec. 2006.

[128] T. J. Kispersky, M. N. Economo, P. Randeria, and J. A. White, “GenNet: A Platform for Hybrid Network Experiments.,” *Front. Neuroinform.*, vol. 5, p. 11, Jan. 2011.

[129] A. Sharma, L. Rieth, P. Tathireddy, R. Harrison, H. Oppermann, M. Klein, M. Töpper, E. Jung, R. Normann, G. Clark, and F. Solzbacher, “Long term in vitro functional stability and recording longevity of fully integrated wireless neural interfaces based on the Utah Slant Electrode Array.,” *J. Neural Eng.*, vol. 8, no. 4, p. 045004, Aug. 2011.

[130] T. D. Y. Kozai, N. B. Langhals, P. R. Patel, X. Deng, H. Zhang, K. L. Smith, J. Lahann, N. A. Kotov, and D. R. Kipke, “Ultrasmall implantable composite microelectrodes with bioactive surfaces for chronic neural interfaces.,” *Nat. Mater.*, vol. 11, no. 12, pp. 1065–73, Dec. 2012.

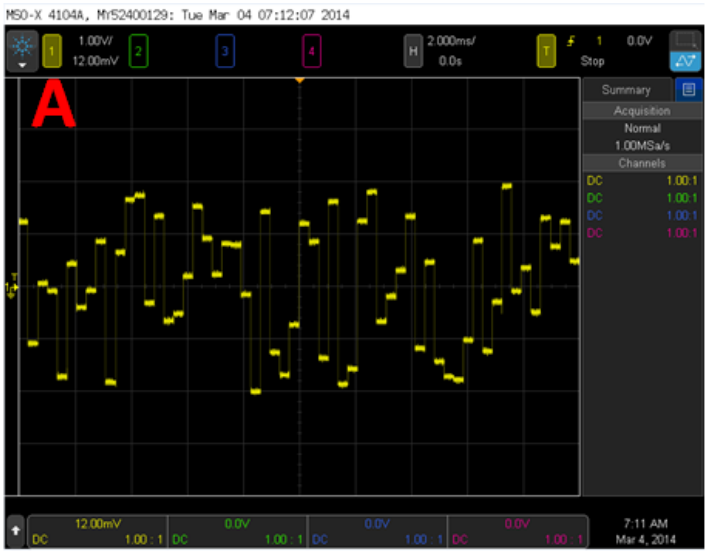
[131] Q. Liu, G. A. Constantinides, K. Masselos, and P. Y. K. Cheung, “Data-reuse exploration under an on-chip memory constraint for low-power FPGA-based systems,” *IET Comput. Digit. Tech.*, vol. 3, no. 3, p. 235, 2009.

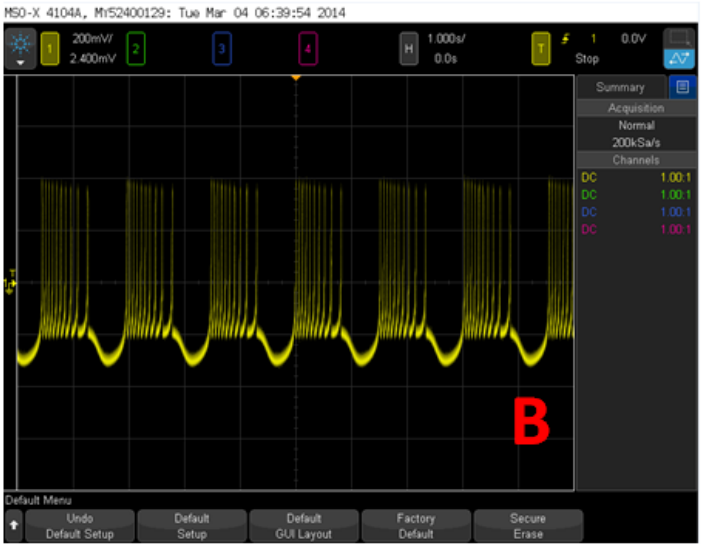
[132] C. Soto-Treviño, P. Rabbah, E. Marder, and F. Nadim, “Computational model of electrically coupled, intrinsically distinct pacemaker neurons.,” *J. Neurophysiol.*, vol. 94, no. 1, pp. 590–604, Jul. 2005.

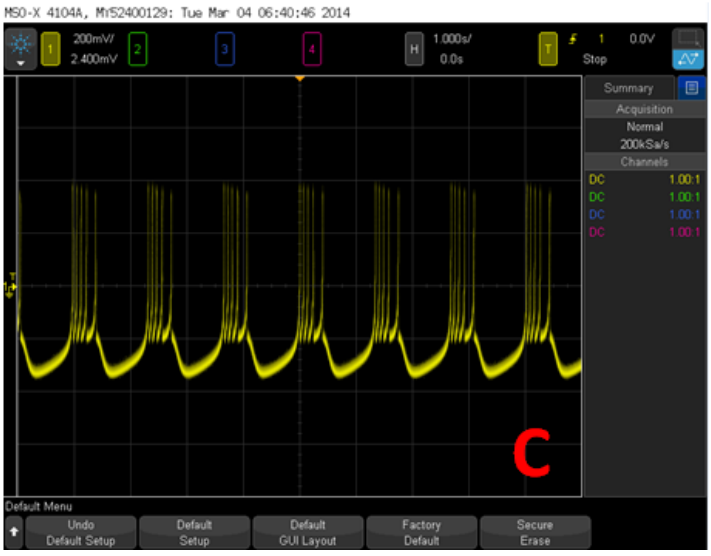
[133] K. Cheung, S. R. Schultz, and P. H. W. Leong, “A parallel spiking neural network simulator,” in *2009 International Conference on Field-Programmable Technology*, 2009, pp. 247–254.

Appendices

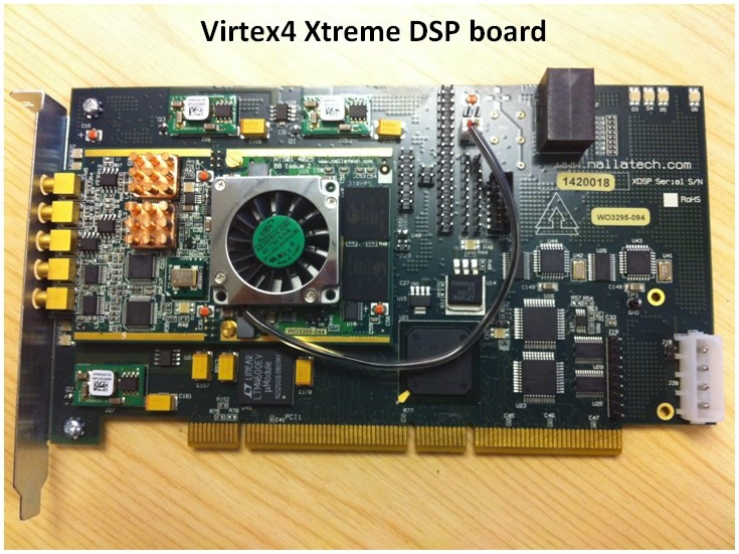
A. The FPGA on-board results of a standard HR and IF neuronal model

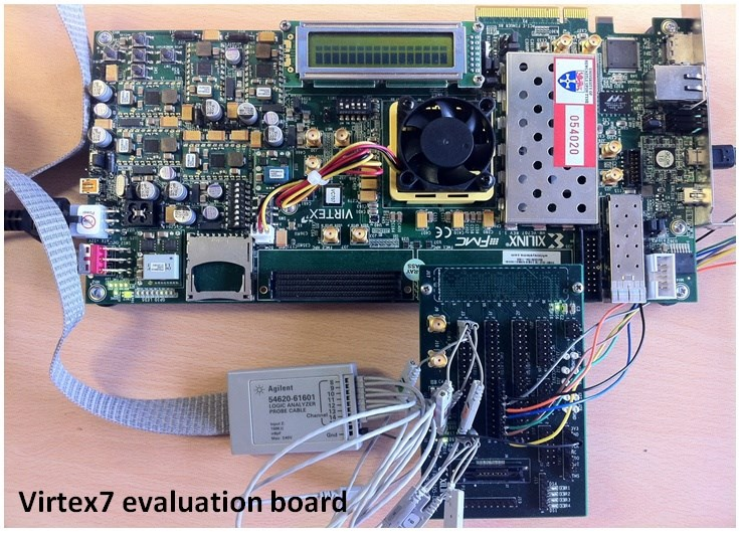






B. The physical board display of Virtex-4, 5 and 7





C. The VHDL code of ChR2

```
----------------------------------------------------------------------------------
-- Company:   Newcastle University
-- Engineer:   Junwen Luo
--
-- Create Date:    20:08:08 06/27/2014
-- Design Name:     Channelrhodopsin-2 (ChR2)
-- Module Name:    chr2 – Behavioural
-- Project Name:   Silicon ChR2
-- Target Devices:  Virtex-7 evaluation kit
-- Tool versions:
-- Description:
 --  All fixed-point values are represented by 45-bit and 30-fractional bit
 --  The architecture is to mimic ChR2 ion dynamic with single light pulse
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 – File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity chr2 is
    port ( imax : in  std_logic_vector(44 downto 0);
                CLOCK : in  std_logic;
                current_chr2 : out  std_logic_vector (44 downto 0)
);
end chr2;

architecture netlist of chr2 is

 component selection port                          -----decide ChR2 conductance between light on
and light off
      (  CLOCK :in  std_logic;
        t_out :out  std_logic_vector( 44 downto 0);
                                control_out :out std_logic;
        pulse_out   :out std_logic_vector (44 downto 0)
                                );
  end component;

  component ga1 port                                -----Calculate ChR2 ga1 conductance
       ( t  :in std_logic_vector( 44 downto 0);
                                 CLOCK :in  std_logic;
                                 control :in std_logic;
         pulse :  in std_logic_vector (44 downto 0);
         ga1_out :out  std_logic_vector( 44 downto 0)
);
```

```
 end component;

 component  ga2 port                          -----Calculate ChR2 ga2 conductance
       ( t  :in std_logic_vector( 44 downto 0);
                              CLOCK :in  std_logic;
                              control :in std_logic;
         pulse  : in std_logic_vector (44 downto 0);
         ga2_out :out  std_logic_vector( 44 downto 0)
);
 end component;

 component ode port                           -----Calculate ChR2 ionic numbers
      (  ga1_in  :in std_logic_vector( 44 downto 0);
                        CLOCK :in  std_logic;
                        ga2_in : in std_logic_vector( 44 downto 0);
                              o1      :out std_logic_vector (44 downto 0);
                              o2      :out std_logic_vector (44 downto 0);
                              o3     : out std_logic_vector (44 downto 0)
                              );
           end component;

 component current_g port                     -----Calculate ChR2 current
     (  imax    : in std_logic_vector(44 downto 0);
                        CLOCK :in  std_logic;
                              O1    :  in std_logic_vector (44 downto 0);
                              O2    :  in std_logic_vector (44 downto 0);
                              current :out std_logic_vector (44 downto 0)
                              );
 end component;

signal s1 ,s3,s4,s5,s6,s7,s8 : std_logic_vector(44 downto 0);
signal s2  : std_logic;

begin

  U1:  selection port map (CLOCK, s1,s2,s3);
  U2:  ga1       port map (s1,CLOCK,s2,s3,s4);
  U3:  ga2       port map (s1,CLOCK,s2,s3,s5);
  U4:  ode       port map (s4, CLOCK, s5, s6,s7,s8);
  U5:  current_g port map (imax, CLOCK, s6,s7, current_chr2);

end netlist;
```

U1:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity selection is
```

```
      generic ( light : std_logic_vector (44 downto 0) :=
"000000000000101"&"00000000000000000000000000000000" );
      port (  CLOCK :in  std_logic;
        t_out :out  std_logic_vector( 44 downto 0);
                              control_out :out std_logic;
        pulse_out   :out std_logic_vector (44 downto 0)
                              );

function adder(a: in std_logic_vector;
           b:in std_logic_vector;
                                        width: in integer;
                                        lowbit: in integer)

return std_logic_vector is
  variable s_p : std_logic_vector(width-1 downto 0);
          begin
            s_p := a+b;
                  return s_p;
end function;


function mult(a: in std_logic_vector;
           b:in std_logic_vector;
                                        width: in integer;
                                        lowbit: in integer)

return std_logic_vector is
  variable s_p : std_logic_vector(a'length + b'length-1 downto 0);
          begin
            s_p := a*b;
                  return s_p(lowbit+width-1 downto lowbit);
end function;

end selection;

architecture Behavioural of selection is

constant step : std_logic_vector(44 downto 0) :=
"000000000000000"&"00000010100011110101110000101010";

signal count_int : std_logic_vector(44 downto 0) := (others =>'0');
signal count_out : std_logic_vector(44 downto 0);


begin

  process(CLOCK)
            begin
                  if rising_edge(CLOCK) then
                        if                                  count_int                                  =
"001001110001000"&"00000000000000000000000000000000" then
                                          count_int <= (others =>'0');
                                    else
                  count_int                                                                          <=
adder(              count_int,"000000000000001"&"00000000000000000000000000000000",45,45);

                                    end if;
                                    count_out <= mult(count_int,step,45,45);
                                    t_out <= count_out;
                                    pulse_out <= light;
```

```
                                        if count_int <= light then
                                              control_out <= '1' ;
                                        else
                        control_out <= '0' ;
                  end  if;

              end if;



  end process;


end Behavioural;
```

U2:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ga1 is
 port( t  :in std_logic_vector( 44 downto 0);
                                CLOCK :in  std_logic;
                                control :in std_logic;
          pulse :  in std_logic_vector (44 downto 0);
          ga1_out :out  std_logic_vector( 44 downto 0)
);
end ga1;




architecture netlist of ga1 is

component mux1 port  (sel: in std_logic;
                  d0 : in std_logic_vector( 44 downto 0);
                                                      d1 : in std_logic_vector( 44 downto 0);
                                                      d_out    :    out    std_logic_vector(    44
downto 0)
                                                   );
end component;

component ga1off port (t_in : in std_logic_vector (44 downto 0);
                     p_in:  in std_logic_vector (44 downto 0);
                                                            CLOCK :in  std_logic;
                                                            data1_in  :  in  std_logic_vector
(44 downto 0);
                                                            data2_in  :  in  std_logic_vector
(44 downto 0);
```

```
                                                    out1 : out std_logic_vector (44
downto 0);
                                                    address1 : out  integer;
                                                    address2 : out  integer
                                                );

end component;

component memory1 port(
                address1_in : in integer;
                                                    address2_in : in integer;
                                                    data1_out       :   out
std_logic_vector( 44 downto 0);
                                                    data2_out       :   out
std_logic_vector( 44 downto 0)
                                                    );
end component;


component ga1on port (t_in : in std_logic_vector (44 downto 0);
                p_in:  in std_logic_vector (44 downto 0);
                                                    CLOCK :in  std_logic;
                                                    data3_in  :  in  std_logic_vector
(44 downto 0);
                                                    out2 : out std_logic_vector (44
downto 0);
                                                    address3 : out  integer

                                                );

end component;

component memory2 port(
                address3_in : in integer;

                                                    data3_out       :   out
std_logic_vector( 44 downto 0)
                                                    );
end component;

signal s1,s2,s3,s4,s7 : std_logic_vector (44 downto 0);
signal s5,s6,s8 : integer;

begin

  U1 : ga1off port map ( t, pulse, CLOCK,s3, s4, s1, s5, s6);
  U2 : memory1 port map ( s5, s6, s3, s4);
  U3:  ga1on  port map ( t, pulse, CLOCK,s7, s2, s8);
  U4 : memory2 port map ( s8, s7);
  U5:  mux1   port map (control, s1,s2,ga1_out);


end netlist;
```

U3:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ga1on is
port (t_in : in std_logic_vector (44 downto 0);
                  p_in:  in std_logic_vector (44 downto 0);
                                                                  CLOCK :in  std_logic;
                                                                  data3_in  :  in  std_logic_vector
(44 downto 0);

                                                                  out2 : out std_logic_vector (44
downto 0);
                                                                  address3 : out  integer

                                                                );

function adder(a: in std_logic_vector;
          b:in std_logic_vector;
                                    width: in integer;
                                    lowbit: in integer)

return std_logic_vector is
  variable s_p : std_logic_vector(a'length + b'length-1 downto 0);
          begin
            s_p := a+b;
                  return s_p(lowbit+width-1 downto lowbit);
end function;

function sub(a: in std_logic_vector;
          b:in std_logic_vector;
                                    width: in integer;
                                    lowbit: in integer)

return std_logic_vector is
  variable s_p : std_logic_vector(a'length + b'length-1 downto 0);
          begin
            s_p := a-b;
                  return s_p(lowbit+width-1 downto lowbit);
end function;


function mult(a: in std_logic_vector;
          b:in std_logic_vector;
                                    width: in integer;
                                    lowbit: in integer)

return std_logic_vector is
  variable s_p : std_logic_vector(a'length + b'length-1 downto 0);
          begin
            s_p := a*b;
```

```
                return s_p(lowbit+width-1 downto lowbit);
end function;

end ga1on;

architecture Behavioural of ga1on is

constant tau_ChR : std_logic_vector := 
"100000000000000"&"110001001110101001001010100011";
constant QEtrans : std_logic_vector := 
"000000000000000"&"100110011001100110011001100110";
constant F : std_logic_vector := 
"000000000000000"&"001001110101100011100010000110";

signal s1,s2,s3 : std_logic_vector(44 downto 0);

begin
   process(CLOCK,t_in,data3_in)
          begin
                s1<= mult(QEtrans, F, 45,45);
                s2<= mult(t_in, tau_ChR, 45,45);
                address3 <= to_integer(unsigned(s2));

                --s3<= 
adder("000000000000001"&"00000000000000000000000000000",data3_in,45,45);
                 s3<=  "000000000000001"&"00000000000000000000000000000"+data3_in;
                out2<= mult(s1,s3,45,45);
               end process;

end Behavioural;
```

U4:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ode is
generic ( Gd1 : std_logic_vector (44 downto 0) := 
"000000000000101"&"010110011001100110011001100110";
        e_ct: std_logic_vector (44 downto 0) := 
"000000000000101"&"000000101000111101011100001010";
                e_tc: std_logic_vector (44 downto 0) := 
"000000000000101"&"000001010001111010111000010100";
                Gd2 : std_logic_vector (44 downto 0) := 
"000000000000101"&"000001010001111010111000010100";
                Gr_d : std_logic_vector (44 downto 0) := 
"000000000000101"&"000000000010101110101111110110"
                );
```

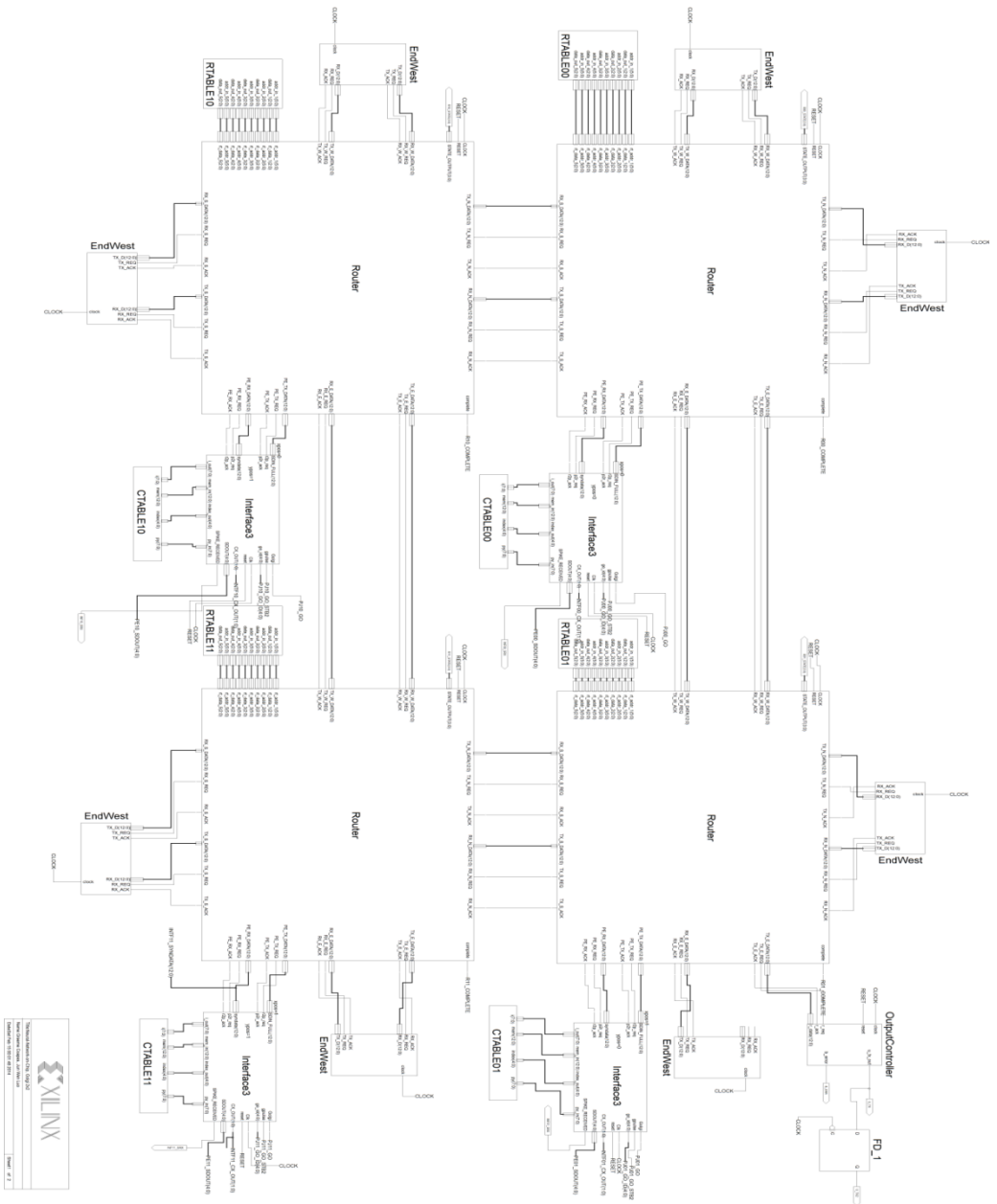
```
port (  ga1_in  :in std_logic_vector( 44 downto 0);
                        CLOCK :in  std_logic;
                        ga2_in : in std_logic_vector( 44 downto 0);
                                o1      :out std_logic_vector (44 downto 0);
                                o2      :out std_logic_vector (44 downto 0);
                                o3     : out std_logic_vector (44 downto 0)
                    );


function adder(a: in std_logic_vector;
            b:in std_logic_vector;
                                        width: in integer;
                                        lowbit: in integer)

return std_logic_vector is
   variable s_p : std_logic_vector(width-1 downto 0);
           begin
             s_p := a+b;
                    --return s_p(lowbit+width-1 downto lowbit);
                    return s_p;
end function;

function sub(a: in std_logic_vector;
            b:in std_logic_vector;
                                        width: in integer;
                                        lowbit: in integer)

return std_logic_vector is
   variable s_p : std_logic_vector(width-1 downto 0);
           begin
             s_p := a-b;
                    --return s_p(lowbit+width-1 downto lowbit);
                    return s_p;
end function;


function mult(a: in std_logic_vector;
            b:in std_logic_vector;
                                        width: in integer;
                                        lowbit: in integer)

return std_logic_vector is
   variable s_p : std_logic_vector(a'length + b'length-1 downto 0);
           begin
             s_p := a*b;
                    return s_p(lowbit+width-1 downto lowbit);
end function;

function delay ( a: in std_logic_vector (44 downto 0);
              clk: in std_logic)
return std_logic_vector is
    variable r : std_logic_vector(44 downto 0);
     begin

                                if(clk = '1')then
                                     r := a;
                                else
              r:= (others =>'0')        ;
          end if;
```

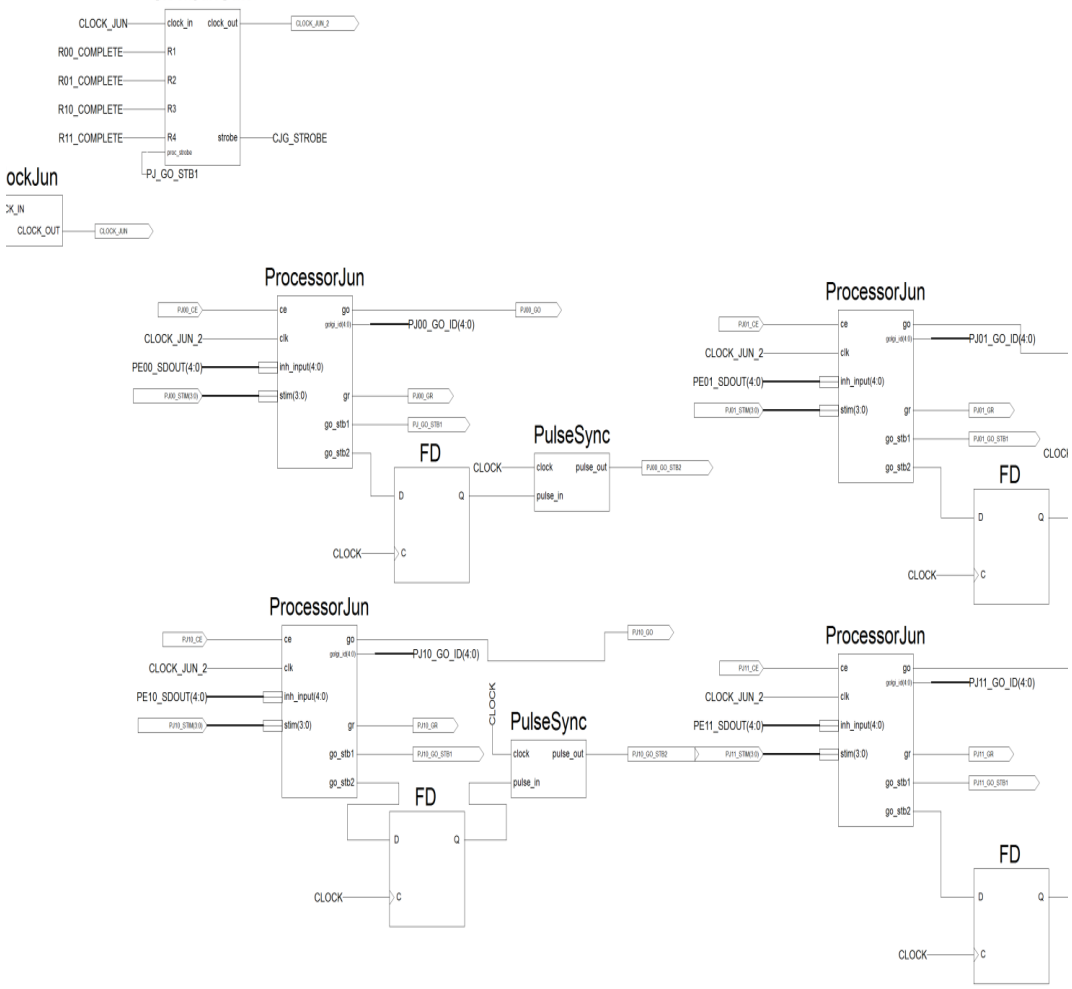
```
      return r;
 end function;



end ode;

architecture Behavioural of ode is

signal s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s12,so1,so2,so3 : std_logic_vector(44 downto 0) :=
(others =>'0');
signal s13,s14,s15,s16,s17,s18,s19,s21 : std_logic_vector(44 downto 0) := (others =>'0');
signal s22,s23,s24,s25,s26,s28 : std_logic_vector(44 downto 0) := (others =>'0');

begin
   process(CLOCK,ga1_in,ga2_in)
            begin
                  if(rising_edge(CLOCK)) then
                    s1 <= Gd1 + e_ct;
                        --s2<= adder(s1,ga1_in,45,45);
                        s2 <= s1 + ga1_in;

        s3<= mult(s2,so1, 45,45);

        s4<= sub( ga1_in, s3,45,45);

        s5<= sub(e_tc, ga1_in, 45,45);
        s6<= mult(s5,so2,45,45);
        s7<= mult(ga1_in,so3,45,45);
        s8<= sub(s6,s7,45,45);
        s9<= sub(s4,s8,45,45);
        s10<= mult(s9,"000000000000101"&"00000010100011110101110000 1010",45,45);
        so1<= adder(s10,s12,45,45);
        s12<= delay(so1,CLOCK);
                        s13<= mult(e_ct,so1,45,45);
                        s14<= Gd2 + e_tc;
                        s15<= mult(so2,s14,45,45);
                        s16<= sub(s13,s15,45,45);
                        s17<= mult(ga2_in,so3,45,45);
                        s18<= adder(s17,s16,45,45);
                        s19<=
mult(s18,"000000000000101"&"00000010100011110101110000 1010",45,45);
                        so2 <= adder(s19,s21,45,45);
                        s21<= delay(so2,CLOCK);
                        s22<= mult(Gd2,so2,45,45);
                        s23<= adder(ga2_in, Gr_d,45,45);
                        s24<= mult(s23, so3, 45,45);
                        s25<= sub(s22, s24,45,45);
                        s26<=
mult(s25,"000000000000101"&"00000010100011110101110000 1010",45,45);
                        so3<= adder(s26,s28,45,45);
                        s28<= delay(so3,CLOCK);
                        o1<=so1;
                        o2<=so2;
                        o3<=so3;
                  end if;

  end process;
```

```
end Behavioural;

----------------------------------------------------------------------------------------------------U5:
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity current_g is

port   (  imax    : in std_logic_vector(44 downto 0);
                    CLOCK :in  std_logic;
                          O1    :  in std_logic_vector (44 downto 0);
                          O2    :  in std_logic_vector (44 downto 0);
                          current :out std_logic_vector (44 downto 0)
                          );

function adder(a: in std_logic_vector;
          b:in std_logic_vector;
                                    width: in integer;
                                    lowbit: in integer)

return std_logic_vector is
  variable s_p : std_logic_vector(width-1 downto 0);
         begin
           s_p := a+b;
                 --return s_p(lowbit+width-1 downto lowbit);
                 return s_p;
end function;

function sub(a: in std_logic_vector;
          b:in std_logic_vector;
                                    width: in integer;
                                    lowbit: in integer)

return std_logic_vector is
  variable s_p : std_logic_vector(width-1 downto 0);
         begin
           s_p := a-b;
                 --return s_p(lowbit+width-1 downto lowbit);
                 return s_p;
end function;


function mult(a: in std_logic_vector;
          b:in std_logic_vector;
                                    width: in integer;
                                    lowbit: in integer)

return std_logic_vector is
  variable s_p : std_logic_vector(a'length + b'length-1 downto 0);
```

```
            begin
              s_p := a*b;
                    return s_p(lowbit+width-1 downto lowbit);
end function;

function delay ( a: in std_logic_vector (44 downto 0);
              clk: in std_logic)
return std_logic_vector is
     variable r : std_logic_vector(44 downto 0);
      begin

                                    if(clk = '1')then
                                        r := a;
                                    else
               r:= (others =>'0')        ;
           end if;

        return r;
 end function;




end current_g;

architecture Behavioural of current_g is

signal s1,s2,s3,s4 : std_logic_vector(44 downto 0);

begin
   process(CLOCK,imax)
              begin
                        s1                                                                                  <=
mult("000000000001010"&"00000000000000000000000000000",imax,45,45);
                                    s2 <= mult(s1,O1,45,45);
                                    s3 <= mult(s1,O2,45,45);
                                    s4                                                                      <=
mult(s3,"000000000000000"&"00010100011110101110000101000 1",45,45);
                                    current<= adder(s4,s2,45,45);
   end process;


end Behavioural;
```

D. Schematic figures of two-by-two frame-based network-on-chip system*

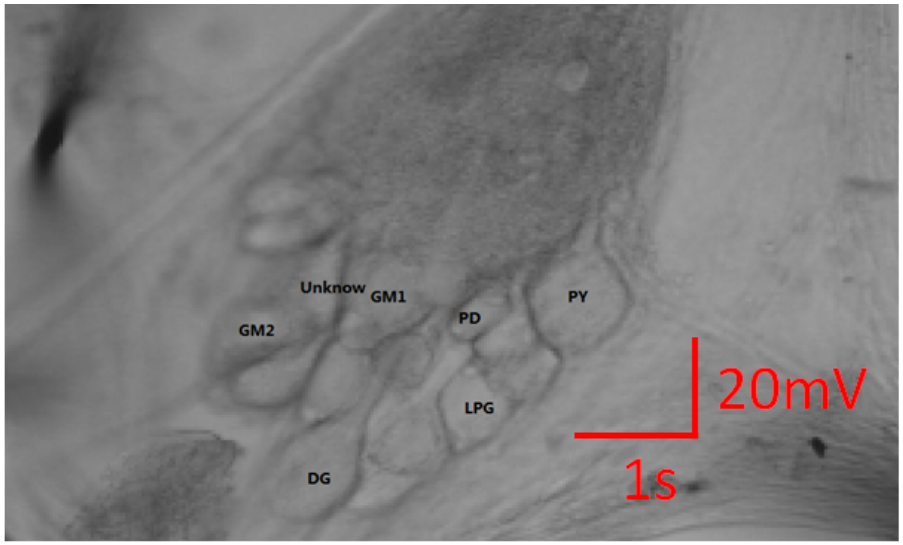


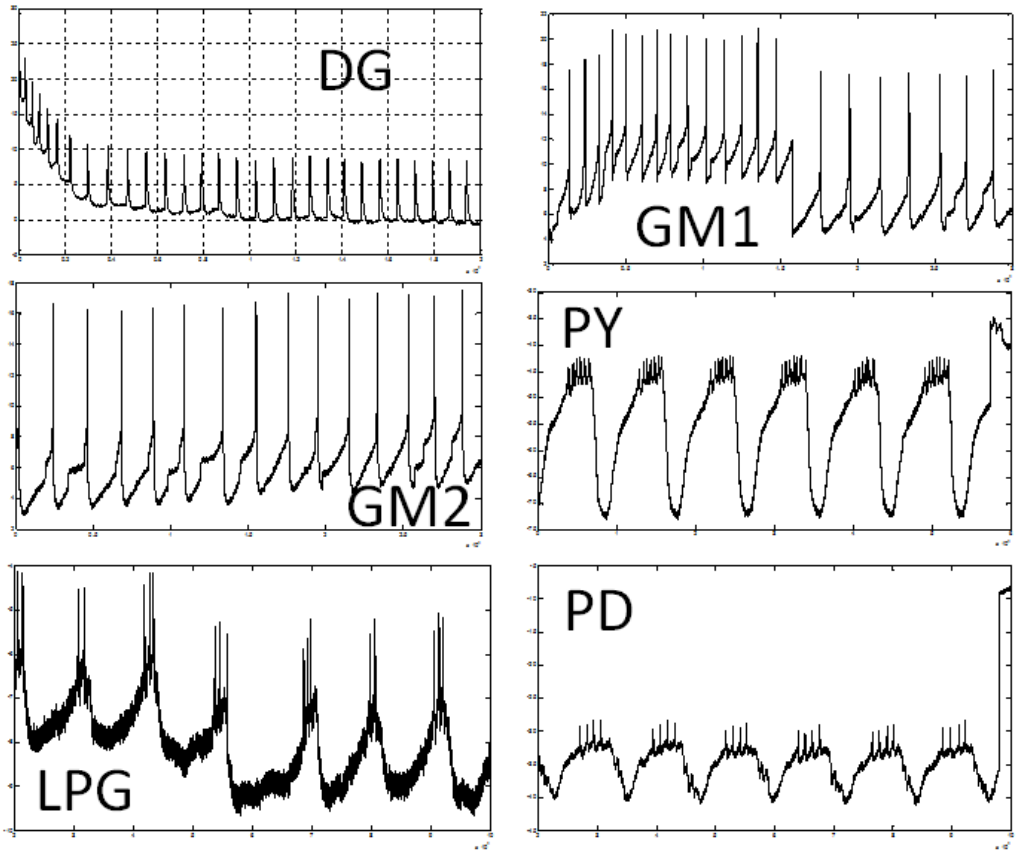

CLOCK_JUN
clock_in
clock_out
CLOCK_JUN_2
R00_COMPLETE
R1
R01_COMPLETE
R2
R10_COMPLETE
R3
R11_COMPLETE
R4
strobe
CJG_STROBE
proc_strobe
PJ_GO_STB1
ockJun
K_IN
CLOCK_OUT
CLOCK_JUN
ProcessorJun
ce
go
PJ00_GO_ID(4:0)
CLOCK_JUN_2
clk
PE00_SDOUT(4:0)
inh_input(4:0)
stim(3:0)
gr
go_stb1
go_stb2
PulseSync
FD
CLOCK
clock
pulse_out
pulse_in
D
Q
C
PJ01_GO_ID(4:0)
PE01_SDOUT(4:0)
PJ10_GO_ID(4:0)
PE10_SDOUT(4:0)
PJ11_GO_ID(4:0)
PE11_SDOUT(4:0)

E. STG mapping results and closed-loop system set-up

E.1 mapping

Since the entire STG network has only approximately 24 neurons, it is feasible to identify each neuron location and character and to recognize the pacemaker neurons. The mapped results are shown below:





E.2 Closed-loop system set-up

The Virtex-4 DSP board was integrated into a standard PC as a digital processor, and the electro-psychological device is an Axoclamp 900A Amplifier. The overall system is shown below.