Delay-based Pattern Recognition using Field-Programmable Gate Arrays

C. H. ANG

A thesis submitted in fulfilment of the requirements for the degree of

Master of Philosophy

School of Electrical and Information Engineering

THE UNIVERSITY OF SYDNEY

2013

ABSTRACT

This thesis describes the design and implementation of two pattern recognition systems on field-programmable gate arrays (FPGAs) that operate based on ‘time delays’.

The idea was inspired by the concept of spiking neural networks (SNNs) which suggests information processing in biological neural systems is based on precise timing of action potentials or spikes. Both systems developed process patterns in the form of *spatiotemporal spike sequences* – patterns of spikes distributed over a population of neurons (“space”) and time. The pattern processor in both systems is a time-delay network consisting of *programmable delays* and *coincidence detectors*, which respectively perform pattern learning and matching. The network is implemented using an innovative ‘clock-free’ design approach that exploits the architecture and hardware resources of FPGAs.

The first system performs pattern learning and recognition tasks while the second operates as an *auto-associative memory* – a type of memory where stored data is retrieved via partial presentation of the original copy. Both systems demonstrate effective and fast processing of pattern recognition tasks with relatively low hardware cost.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my supervisor, Assoc. Prof. Philip Leong for his guidance, inspiring ideas and advices, precious time in reviewing my work, encouragement and constant support throughout the entire period of my studies.

I would also like to thank my associate supervisor, Prof. André van Schaik, Assoc. Prof. Craig Jin and Dr. Alistair McEwan for their valuable advices, discussions and feedback on my work.

I am grateful to Dr. Richard Davis, Dr. Elaine Ou and Dr. Roberto Cardu for their valuable comments and discussions, as well as for reviewing my manuscripts.

I thank all my colleagues at the University of Sydney for their daily help, sharing of ideas and support.

Finally, I thank my parents, and my wife, for their love, sacrifices and continuous support.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Aims and Motivation

Humans are granted with highly sophisticated ability to learn and distinguish patterns, for example, recognizing a face, reading handwriting and understanding spoken words. Research in *pattern recognition* has been undertaken intensively for the past five decades with a common goal to establish similar capabilities in machines. However, design and implementation of a pattern recognizer that is realistic and efficient on hardware remain a challenging task.

There are varieties of pattern recognition techniques, ranging from simple matching of patterns against stored templates to more complex methods of using artificial neural networks (ANNs) [30]. The work in this thesis was inspired by the idea of *spiking neural networks* (SNNs). Neurons in the nervous system communicate using short electrical pulses called *action potentials*, or *spikes* [42, 44]. SNNs represent the most recent generation of ANN models where precise timing of individual spikes is believed to be playing an important role in the functioning of the brain [1, 42]. Biological studies and the concept of SNNs suggest that information is encoded in the 'time relationship' or 'delays' between spikes [43, 46, 47].

Field-programmable gate arrays (FPGAs), on the other hand, are programmable logic devices that offer a great scale of flexibility in design and implementation of digital circuits due to their reprogrammability. These devices consist of hundreds of thousands of logic elements with massive programmable connections [59, 60]. Readily available and affordable to users, FPGAs are widely popular for hardware implementation and design prototyping [41, 7, 15, 23, 39, 54, 48].

The aim of this thesis is to develop pattern recognition systems that process patterns based on ‘time delays’ on FPGAs. The motivation of this work is two-fold. Firstly, we would like to explore the use of time delays in pattern recognition processing and realize the idea into concrete hardware implementations on FPGAs. Secondly, we hope the hardware models developed and the approach we used may contribute to the discovery of an innovative design methodology for creating fast-processing circuits, for applications including but not limited to pattern recognition. In particular, we aim to develop fast-processing pattern recognition models by exploiting the architecture and hardware resources of FPGAs.

1.2 Thesis Outline

Research in pattern recognition has a long history. Chapter 2 introduces the basics of pattern recognition and provides the background details that led us to the idea of developing our pattern recognition systems. The fundamentals of SNNs and FPGAs are introduced in this chapter. We also gain insight into existing works of other researchers and introduce the idea of our pattern recognition models.

In Chapter 3, we describe the functionality and the design of our pattern recognition model, and demonstrate how such model can be implemented by taking advantage of the FPGA architecture. We explore how information could be encoded into spike patterns through time delays and explain how our model learns and recognizes spike patterns.

Based on the same principle of using time delays, we develop a memory model on an FPGA in Chapter 4. The memory belongs to the family of *auto-associative memories* where data is retrieved through partial presentation of the original copy. The system stores and recalls spike patterns.

In Chapter 5, we present the results for tests and hardware implementation for the two systems developed in Chapter 3 and 4, respectively. The results provide a good demonstration of the viability of implementing a pattern recognizer based on time delays.

In the final chapter, we summarize our work in this thesis and provide possible directions for future work.

CHAPTER 2

BACKGROUND

2.1 Introduction

This chapter provides the background for the work we intend to present in this thesis.

The overview of pattern recognition and its various approaches is first provided in Section 2.2. Section 2.3 introduces the fundamental concept of spiking neural networks (SNNs), which provides the inspiration for our work in developing a pattern recognition system based on time delays. An overview of existing pattern recognition models developed by other researchers is covered in Section 2.4, while Section 2.5 provides an introduction to field-programmable gate arrays (FPGAs) and examples of FPGA-based pattern recognizers. Finally, the last section of this chapter describes the basic idea of a delay-based pattern recognition system we aim to develop.

2.2 Pattern Recognition

In machine learning, *pattern recognition* is the study of how machines can learn to distinguish patterns of interest and make sound decisions about the categories of the patterns [30]. Research in pattern recognition has been around for about 50 years. However, pattern recognition systems with high speed processing power, low cost and small size, and efficient implementation, remain elusive.

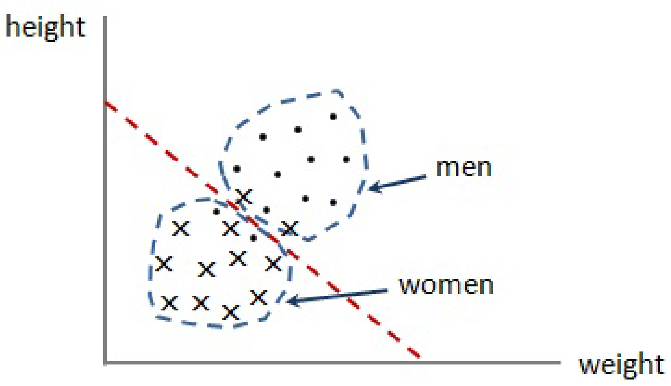


Figure 2.1: Example of a simple statistical classification

There are four best known approaches for pattern recognition: 1) template matching, 2) statistical classification, 3) syntactic or structural matching, and 4) artificial neural networks (ANNs).

Template matching is one of the classical and the simplest approaches to pattern recognition. In this approach, patterns to be recognized are directly matched against stored templates or prototypes. The templates themselves could be learned from available training samples. However, template matching is computationally demanding and may not be the most efficient and effective approach to pattern recognition when dealing with complex or noisy patterns.

The *statistical classification* approach classifies patterns based on the statistical distributions of features. Each pattern is usually represented by a point in a representation space of the features and classes of patterns are represented by regions in that feature space. For example, suppose the average value of height and weight for women is 165 cm and 57 kg, respectively, and for men is 180 cm and 72 kg, respectively; a statistical classifier may estimate the probability distribution of the two features, *i.e.* height and weight, from training samples and establish a decision boundary for classification, as illustrated in Figure 2.1. A person with height 182 cm and weight 75 kg will then be classified as a man in this example.

Syntactic or structural matching adopts a hierarchical approach to pattern recognition. This method decomposes a given complex pattern into sub-patterns which are themselves built from yet simpler sub-patterns. The given complex pattern is then classified based on the interrelationships between the sub-patterns and itself. For

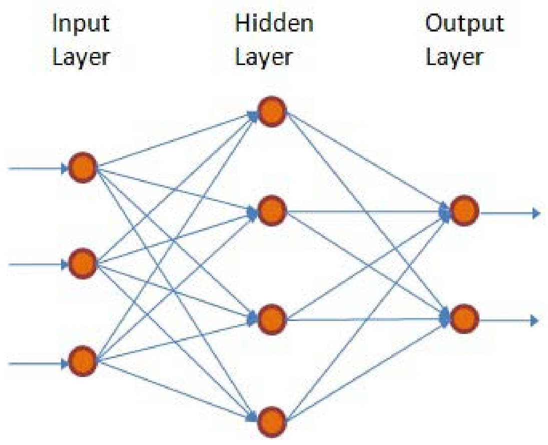


Figure 2.2: A simple feedforward artificial neural network

example, a complex pattern such as an animal may be described in terms of its sub-patterns such as head, limbs, legs, or tail, and be classified into an appropriate category based on relationships between the sub-patterns and its complex pattern.

Artificial neural networks (ANNs) can be applied to a variety of problems including pattern recognition and are inspired by biological neural systems [53, 18]. This approach has attracted significant attention due to its ability to learn complex non-linear input-output relationships. The learning process often involves updating network configurations and connection weights so that a network can efficiently adapt itself to learn patterns and perform classification tasks [32, 52, 17, 56]. Among various types of ANN models, feedforward networks such as multilayer perceptrons, redial basis function networks, Kohonen's self-organizing map and SNNs are commonly used for pattern recognition tasks [2, 13, 65, 31].

2.3 Spiking Neural Networks

Spiking neural networks (SNNs) are the third generation of ANN models that include the factor of time in addition to neuronal and synaptic mechanisms modeled in previous generations of ANNs [42, 61]. These ANN models take into account the precise firing times of neurons, which are believed to be main features in cognitive processing [1, 19].

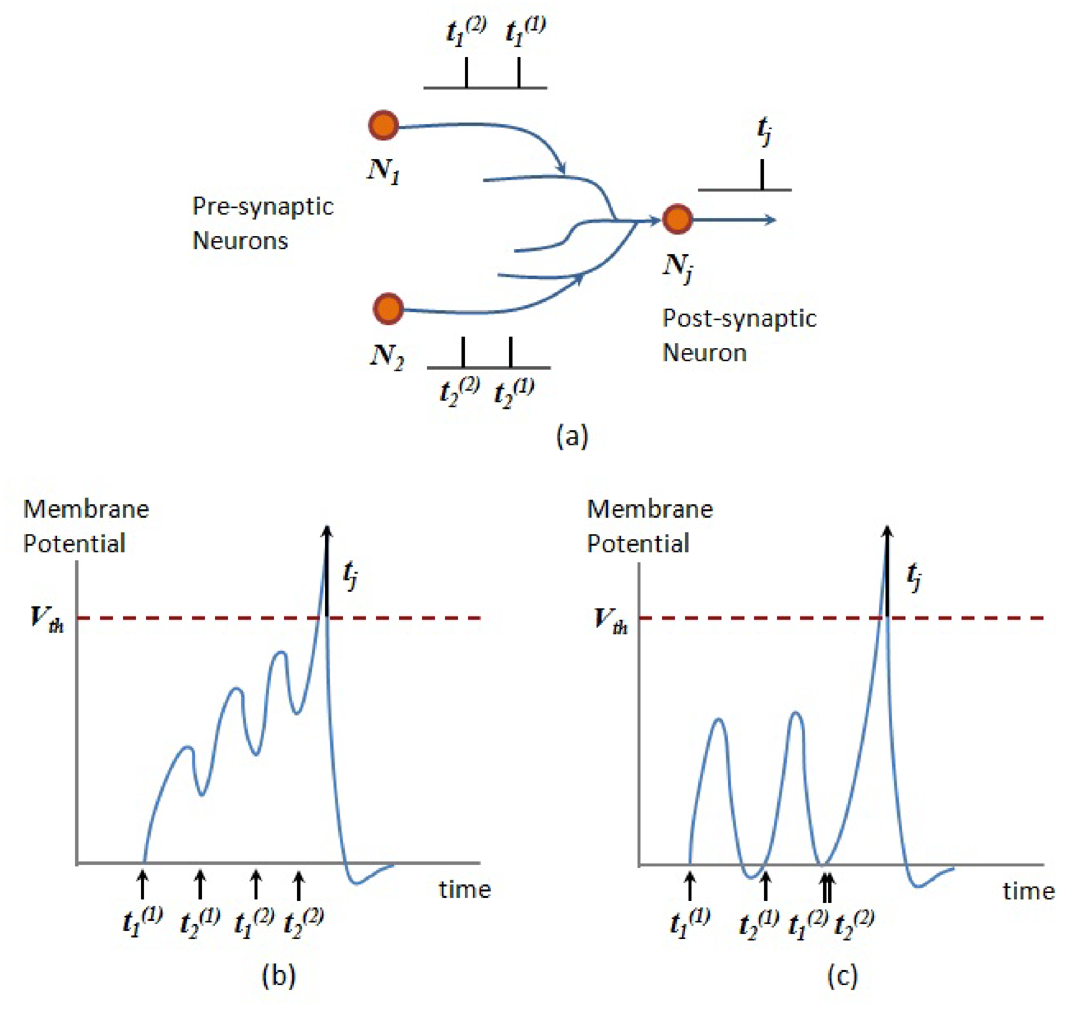


Figure 2.3: (a) A simple spiking neural network; (b) Spike emission of an integrator neuron; (c) Spike emission of a coincidence detector neuron

Neurons in the nervous system process and transmit information using *action potentials*, or *spikes* [42, 44]. Information is believed to be encoded in the time delays between spikes [43, 46, 47, 8]. In general, the biological model of a neuron emits a spike whenever the temporal integration of incoming action potentials generated by its pre-synaptic neurons exceeds a given threshold, V_{th}, as illustrated in Figure 2.3 (a) and (b) [27, 29, 33]. Nevertheless, the specific firing behavior of a neuron may however vary depending on parameters such as the threshold of membrane potential and the inter-spike interval of pre-synaptic neuron emissions. The firing behavior may vary from the role of "integrator", as in Figure 2.3 (b), to the role of "coincidence detector", as in Figure 2.3 (c) [35, 49]. For integrator, most of the input spikes integrated over a period of time contribute to the emission of an output spike; while for coincidence detector, only quasi-synchronously arriving input spikes trigger an output spike emission. Integrator and coincidence detector are two of the best known spiking neuron models suggested in biological research [28, 35].

Biological and theoretical results have shown that SNNs are potentially more powerful than traditional ANNs [43, 45, 63], and are able to perform signal-processing tasks in a robust and energy-efficient manner. Due to these advantages, SNNs have attracted attention in various bio-sensing applications including olfactory sensing [11, 34], auditory systems [25, 36, 37, 71], image processing [12, 40] and pattern recognition [73, 57, 6].

2.4 Existing Works on SNN and Time Delay-based Pattern Recognition

Various SNN and time delay-based pattern recognition systems were studied by researchers. This section provides examples of theoretical and simulation models of such systems. Examples for hardware models in FPGA implementations are given in Section 2.5.3.

Hopfield presented a computational model for pattern recognition based on action potential timing [26]. He suggested that a given pattern in the form of analog variables could be represented by a pattern consisting of action potentials or spikes occurring in a given time relationship, and the recognition computation in this representation could be performed by a network that uses time delays and coincidence detection. The time delays are organized in a way such that the spikes of a pattern, which occur at different times, arrive simultaneously at a coincidence detection neuron, which performs recognition.

An adaptive SNN architecture with an online learning procedure for visual pattern recognition was proposed by Wysoski *et al.* [74]. The architecture comprises four layers of integrate-and-fire neurons. The network learns different views of an object through training samples presented to it online and adaptively changes its structure to respond optimally to different visual patterns. The system performs face recognition by collecting multiple frames of visual data for processing before making a final decision.

Gupta and Long presented a 2-layer SNN model for character recognition [21]. The network consists of integrate-and-fire neurons and uses spike time-dependent plasticity (STDP) for learning, where STDP is a well-known type of learning rule based on the order of pre- and post-synaptic neurons' firing times [9, 58, 62]. Their results showed that 43 out of a set of 48 characters were successfully recognized by the network. Unlike Hopfield's model mentioned earlier, which emphasizes representation of a generic analog input pattern via a set of neurons firing with a time relationship; an input pattern in the models from Gupta and Long, and Wysoski *et al.*, is a set of image pixels each represented by a neuron with constant "ON" and "OFF" states to represent pixel contrast.

Apart from SNN-based pattern recognition models, there are other models that operate based on time delays such as time-delay neural network-based models [22, 38, 70]. However, these models are primarily used to work with continuous data especially in speech processing and time series prediction, and fundamentally different from SNN-based models that process information-encoded temporal spike patterns. Our goal for this thesis is to develop pattern recognition systems with functionality based on the fundamentals of SNNs, *i.e.* similar to how the brain possibly performs pattern recognition tasks through the use of time delays. We aim to develop basic pattern recognition models that work with temporal spike patterns and potentially be useful for development of more complex pattern recognition systems in the future.

2.5 Field-Programmable Gate Arrays

Since their invention in the mid-1980s, *field-programmable gate arrays* (FPGAs) have grown significantly in popularity due to their effective programmability and reconfigurability [53, 10, 5, 16, 20, 24, 51]. These advantages allow different design choices to be evaluated and adopted in a very short time. Unlike custom application-specific integrated circuit (ASIC) implementations, FPGAs are readily available at reasonable cost and allow great reduction in a development cycle.

2.5.1 Architecture and logic resources

The architecture of an FPGA consists of an array of programmable logic blocks with interconnect resources, as well as Input/Output blocks on the border of the chip, as illustrated in Figure 2.4.

The logic blocks constitute the main logic resources for implementing sequential and combinational circuits. The logic resources in each logic block are usually organized as small units of *logic elements* or *cells*. Each logic element often consists of two basic components – an *n*-input look-up table (LUT) and a register. An *n*-input LUT is basically a function generator that can implement any boolean function of *n* variables, while a register is a basic programmable storage element. The terminology used in the organization of logic resources may vary from one FPGA vendor to another. For example, Xilinx refers logic blocks as 'configurable logic blocks' (CLBs) [76], while Altera refers them as 'logic array blocks' (LABs) [3].

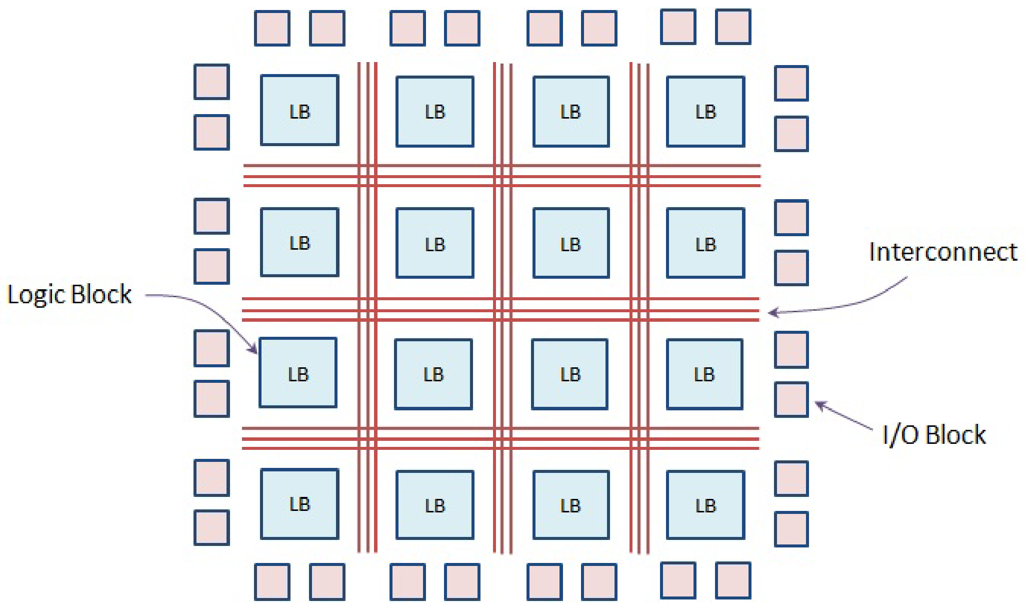


Figure 2.4: Basic FPGA architecture

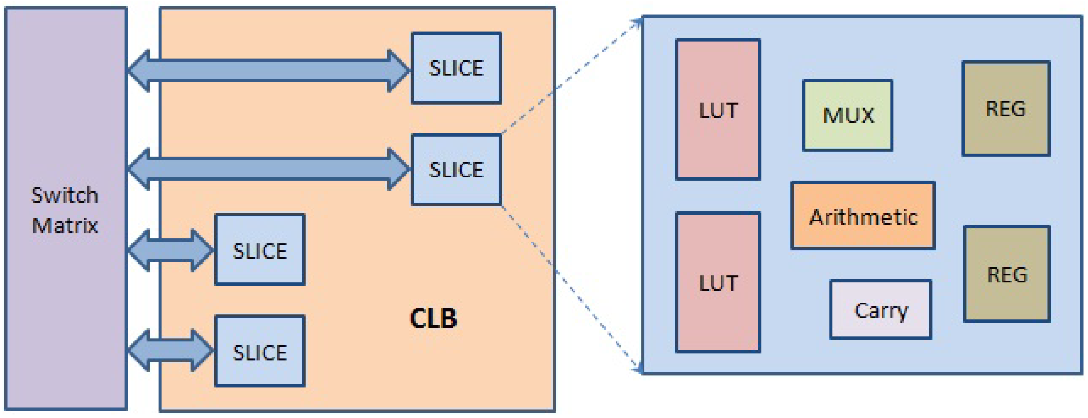


Figure 2.5: Logic resources in a Xilinx Spartan-3E CLB

Figure 2.5 illustrates an example of a CLB in a Xilinx Spartan-3E device. The logic resources are organized as ‘slices’ in each CLB. Each slice contains two logic cells, where each of them comprises a 4-input LUT and a register. There are also additional hardware features in a slice, such as multiplexers (muxes), carry and arithmetic logic for implementing circuits that would otherwise require additional LUTs. Next to every CLB, there is a ‘switch matrix’ that provides programmable connections between slices of the same or different CLB(s) via interconnect.

In addition to the basic logic resources, modern FPGAs nowadays come with embedded higher-level logic functions that are commonly used such as multipliers, memories and processors. Having these commonly-used functions embedded into the chip allows savings in area and better speed performance compared to building them from basic logic resources. The availability of embedded processors such as PowerPC and ARM enables the development of a-system-on-a-reconfigurable-chip. Besides these hard-macro processors, there are also soft processors such as MicroBlaze and Nios II that can be implemented using the FPGA logic resources.

2.5.2 Interconnect resources

Interconnect resources are programmable routing channels between functional entities in an FPGA, such as logic elements, Input/Output blocks and embedded memories [66]. They are usually segmented into different lengths and geometrically optimized for optimum connectivity.

Figure 2.6 illustrates the organization of interconnect resources in a Xilinx Spartan-3E FPGA. There are four types of interconnect segments in the device – single, double, hex (shown respectively in the top, middle and bottom boxes of the figure) and long lines. Single lines route signals to neighboring CLBs horizontally, vertically and diagonally. Double lines route signals to every first and second CLB away horizontally and vertically, in four directions; while hex lines route signals to every third and sixth CLB away, also in four directions. Long lines span across the chip and connect to one out of every six CLBs.

Placement and routing of hardware logic are often optimized by FPGA design tools. The design tools place and route associated logic within a logic block or adjacent logic blocks to optimize for speed performance and area efficiency.

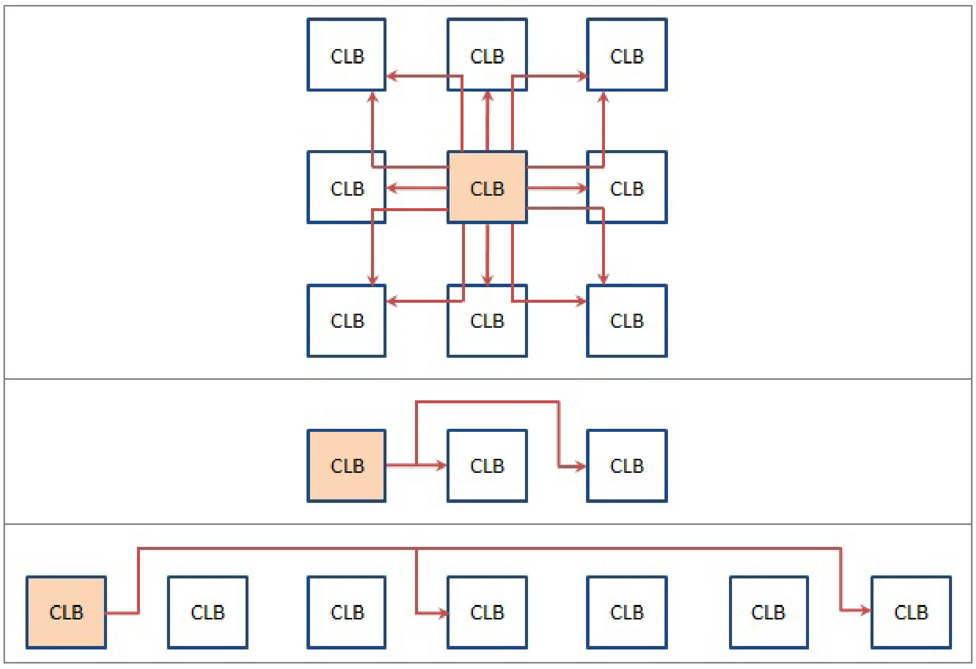


Figure 2.6: Interconnect resources in a Xilinx Spartan-3E device

2.5.3 SNN-based pattern recognizers on FPGAs

Apart from theoretical models, hardware implementations of SNN models were also explored in research [68, 69, 12, 14, 50, 64, 67]. This section provides examples of SNN-based pattern recognizers implemented on FPGAs.

An FPGA implementation of a frequency discriminator using a 3-layer SNN was presented by Upegui *et al*. [67]. Each layer of the network contains 10 neurons. Each neuron was implemented using a finite state machine as control unit, a memory to store parameters, and other sequential logic to perform learning and computations. The hardware model demonstrated positive response to waveforms of different frequencies.

Caron *et al.* presented an FPGA-based SNN for pattern recognition where matching is achieved through synchronization of firing neurons [12]. The SNN uses a bit slice architecture where neurons are organized into slices or columns of 1-bit wide. Each neuron is implemented using a block RAM for synaptic weight storage, along with serial adders and sequential logic to perform computations. The system demonstrated accurate pattern matching results in an image recognition task.

Schrauwen *et al.* proposed a speech recognition system on an FPGA using a liquid state machine (LSM), a recurrent network of spiking neurons where only the output layer is trained [64]. The system uses a serial-processing, serial-arithmetic architecture to achieve savings in hardware resources. The work showed that a relatively small implementation of a real-time speech recognizer compared with existing solutions is viable with a tradeoff in speed.

Another example of FPGA-based pattern recognition system using an SNN, also for speech recognition application, was presented by Cassidy *et al*. [14]. The architecture consists of an array of 32 identical integrate-and-fire neurons, each of them implemented using a 16-bit digital accumulator and memories. The system demonstrated consistent spike responses to respective input speech patterns.

2.6 The Central Idea

Having introduced the background in previous sections, this section describes the basic idea of our work that was fundamentally inspired by the concept of SNNs.

As described in Section 2.4 and 2.5.3, various pattern recognition models based on SNNs were proposed by researchers. These models are either in theoretical or simulation form, or in hardware implemented using conventional design practices and standard digital circuits. In our work, we look into the use of time delays for pattern recognition tasks and explore a novel design approach for implementing a delay-based pattern recognition system on an FPGA. The basis of our pattern recognition model is fundamentally similar to that in Hopfield’s theoretical model described in Section 2.4, which uses a time-delay network and coincidence detection for recognition computation. We further expand from this basis into a more concrete hardware model and realize it on an FPGA with a novel and unique design approach.

FPGAs consist of rich hardware resources which are potentially useful for implementing delays. We develop our pattern recognition circuit by exploiting these massive resources available on FPGAs. Typically, processing speed of a standard digital system designed using conventional synchronous approach is often limited by clock frequencies of the circuit. Power consumption is also relatively high in synchronous circuits when operating at a high clock rate. We aim to build a hardware model for pattern recognition that uses only combinational logic and interconnect resources, with no sequential clocking elements. This unique design approach would allow high-speed processing of pattern recognition tasks with relatively low hardware cost. The approach could possibly improve processing speed of a digital system to a level beyond conventional synchronous circuits could achieve, along with the benefit of power savings. Furthermore, exploitation of interconnect resources for implementing delays would allow significant savings on logic elements and free up those resources for other logic implementations. The approach may also potentially contribute to development of fast-processing circuits beyond FPGA platforms in the future since delays implemented using wires are easier and less expensive to fabricate.

Based on this basic idea and similar approach, we also develop an auto-associative memory in Chapter 4. The memory is fully implemented on an FPGA with pattern learning capability.

As SNNs are also often referred to as pulsed neural networks, the terms “spike” and “pulse” are used interchangeably in this thesis.

CHAPTER 3

DELAY-BASED PATTERN RECOGNITION

3.1 Introduction

The aim of this chapter is to demonstrate the feasibility of FPGA implementation of a delay-based pattern recognition circuit. The circuit is inspired by the concept of spiking neural networks (SNNs) where time delays are believed to be crucial in neurobiological processing [44, 46, 47]. FPGAs consist of massive logic and interconnect resources which are potentially useful for implementing delays. The work in this chapter looks into the use of time delays for pattern recognition and the exploitation of FPGA hardware resources as programmable delays for implementing a delay-based pattern recognition circuit. In particular, the work explores the design of a basic programmable delay line that serves as a fundamental structure for building such pattern recognition circuits. A prototype is developed to demonstrate a complete system with pattern learning capability.

3.1.1 Programmable delays and pattern recognition

Inspired by the concept of SNNs, we propose a basic pattern recognition circuit that uses time delays in the processing of patterns. The patterns are in the form of spike or pulse sequences.

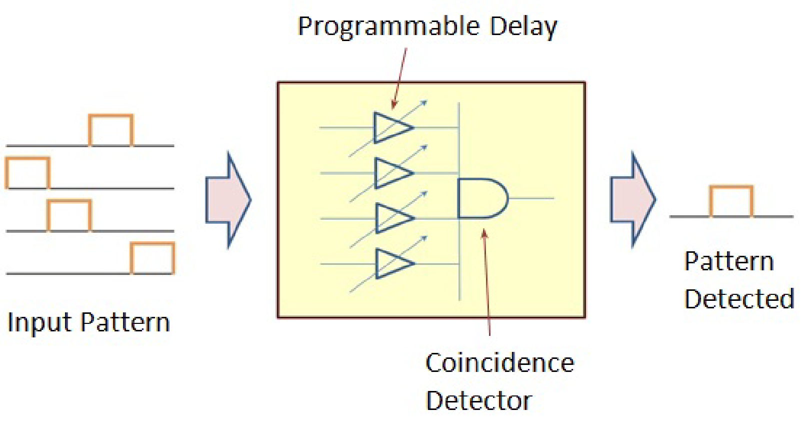


Figure 3.1: Delay-based pattern recognition

The pattern recognition circuit consists of two basic components – *programmable delay lines* and a *coincidence detector*. As illustrated in Figure 3.1, for an input pattern presented to the circuit, each of the spikes is fed through a dedicated programmable delay line that is connected to the coincidence detector. If the programmable delay lines could be trained through delay adaptation such that coincidence of delayed spikes is triggered at the coincidence detector, then one could train the circuit to remember a given pattern and the circuit would detect the trained pattern when the same pattern is presented to it again. The coincidence detector is an AND gate and a detection of coincidence of the delayed spikes results in an assertion of an output spike indicating that the pattern is detected.

3.1.2 Spatiotemporal spike sequence

Neurons in the nervous system communicate using actions potentials, which are, in general, referred to as ‘spikes’, or ‘pulses’. A delay-based pattern recognition circuit described in this chapter processes patterns in the form of *spatiotemporal spike sequences*. A spatiotemporal spike sequence is a pattern of spikes distributed over both a population of neurons (“space”) and time.

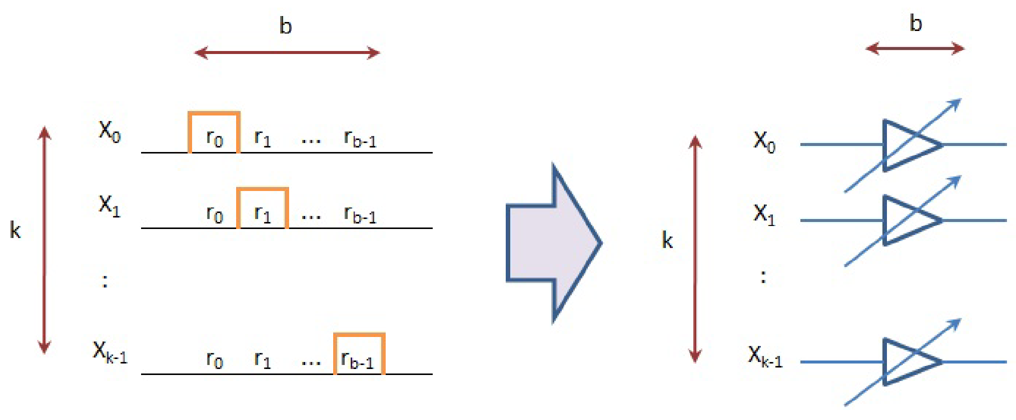


Figure 3.2: Mapping of spatiotemporal spike sequence onto programmable delay lines

We define a spatiotemporal spike sequence to be represented by a vector of k elements $\{x_0, x_1, \ldots, x_{k-1}\}$ with each element having b temporal values $\{r_0, r_1, \ldots, r_{b-1}\}$, as illustrated in Figure 3.2. Each temporal value could represent a piece of data and they all have a pulse width, t_{res}, with a zero time gap between pulses. For example, for a k-element vector with 8 temporal values, each of the x_i elements could represent an integer in {0, 1, … , 7}, while a vector with 26 temporal values could represent a set of characters {A, B, … , Z}. Through this temporal coding scheme, a vector of a data type is translated into a sequence of pulses, or spikes. For a k-element, b-temporal value vector, the total number of possible patterns that could be produced is b^k.

In terms of hardware mapping and representation, each x_i element of a k-element, b-temporal value vector could be represented by a programmable delay line capable of producing b different delays, as illustrated in Figure 3.2.

3.1.3 FPGA interconnect resources

The architecture of an FPGA consists of massive logic and interconnect resources. The design of programmable delay lines described in this chapter effectively utilizes those hardware resources and takes advantage of the interconnect routing architecture for building a delay-based pattern recognition circuit.

Without loss of generality, Xilinx Spartan-3E family FPGAs were used in this work due to their low cost and high performance benefits. Interconnect scheme of these devices is previously described in Section 2.5.2 of Chapter 2.

3.2 Design and Implementation of Programmable Delay Lines

In this section, we describe the design of a basic programmable delay line that serves as a basic structure for constructing longer programmable delay lines in a delay-based pattern recognition circuit. The implementation of the primitives used in a programmable delay line that enable the programmability of the delay line is also described. The last part of this section illustrates how an FPGA-targeted pattern recognition circuit could be constructed by using these basic programmable delay lines.

3.2.1 Basic programmable delay line

The size of programmable delay lines used in a pattern recognition circuit varies depending on the needs and specifications of the patterns to be trained. For patterns with larger pulse width and number of temporal values, *i.e.* larger t_{res} and b, larger amount of delays are required and hence longer programmable delay lines.

We describe the design of a *basic programmable delay line* that serves as a fundamental structure for building longer programmable delay lines for the size of interest. The architecture is based on the physical layout organization of logic and interconnect resources in an FPGA. The architecture effectively utilizes the logic and interconnect resources available to create a programmable delay line. Figure 3.3 illustrates the architecture of the basic programmable delay line. It consists of a chain of LUTs each from adjacent CLBs connecting one after another serially. The basic programmable delay line is extendable to a longer programmable delay line by appending more LUTs to the end of the delay line.

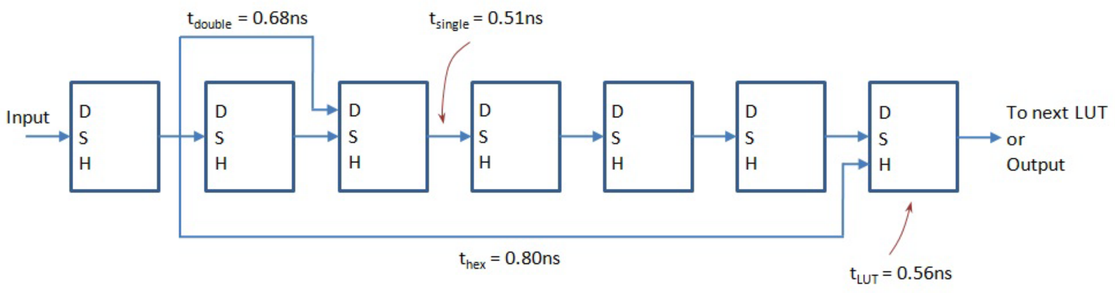


Figure 3.3: Architecture of basic programmable delay line

The LUTs are the basic primitives of the basic programmable delay line. Each of the LUTs functions as a ‘delay switch’ as well as a connection point for interconnect segments. The delay switch is a 3-to-1 multiplexer (mux) providing options to select between a single, a double, or a hex line connection.

The topology of the delay switch connections is organized in the same fashion as the FPGA’s interconnect scheme. Figure 3.3 illustrates the connections between the LUTs using each type of interconnect segment. Note that the output from each LUT provides a single, double, and hex line connection to the respective receiving LUTs (not shown explicitly for every LUT in the figure). For example, the output from the first LUT is routed to the second LUT via a single line; to the third LUT via a double line; and to the seventh LUT via a hex line. Similarly, the output from the second LUT is routed to the third LUT via a single line; to the fourth LUT via a double line; and to the eighth LUT (if exists, when appended for building a longer programmable delay line) via a hex line.

Conforming to the fundamentals of conductivity, the longer an interconnect wire the larger the resistance and capacitance, and hence resulting in a larger delay. Table 3.1 shows a set of characterization results obtained via the vendor’s timing analysis tools. A piece of single line produces an average delay of 0.51 ns, while a double and a hex line produce average delays of 0.68 ns and 0.80 ns, respectively. Each delay switch has an average delay of 0.56 ns. For estimation, we could approximate one LUT-interconnect pair as producing a 1 ns delay.

	Delays (ns)			
	Single line	Double line	Hex line	LUT (delay switch)
	0.52	0.66	0.79	0.58
	0.49	0.68	0.78	0.58
	0.50	0.71	0.80	0.55
	0.51	0.70	0.79	0.56
	0.48	0.70	0.81	0.56
	0.51	0.67	0.81	0.55
	0.53	0.68	0.78	0.58
	0.53	0.67	0.79	0.55
	0.49	0.66	0.81	0.54
	0.50	0.66	0.81	0.55
Average	**0.51**	**0.68**	**0.80**	**0.56**

Table 3.1: Characterized delays of interconnects and LUTs

Delay switch settings for DS0 to DS6	Resulting LUTs and interconnects used	Delays (ns)
S,X,X,X,X,X,H	2 LUTs + 1 hex	2.13
S,X,D,X,D,X,D	4 LUTs + 3 doubles	4.09
S,X,D,X,D,S,S	5 LUTs + 2 doubles + 2 singles	5.22
S,X,D,S,S,S,S	6 LUTs + 1 double + 4 singles	6.18
S,S,S,S,S,S,S	7 LUTs + 6 singles	7.15

Table 3.2: Resulting delays against different delay switch settings

By applying appropriate settings to each of the delay switches, a basic 7-LUT programmable delay line with delay switches DS0 to DS6 from input to output as shown in Figure 3.3 is able to produce a series of variable delays ranging from 2 ns to 7 ns, as shown in Table 3.2. For example, a {DS0,DS1,DS2,DS3,DS4,DS5,DS6} = {S,X,X,X,X,X,H} setting on the delay switches DS0 to DS6 produces a delay of 2.13 ns; a {S,X,D,X,D,X,D} setting gives a 4.09 ns delay; and a {S,S,S,S,S,S,S} setting gives a 7.15 ns delay; where S, D, H, X denote single, double, hex line connection and don't care, respectively. The resulting delays against different delay switch settings were measured and characterized from simulation. The results also agree with estimation based on summation of individual average delays of LUTs and interconnect wires by using the values in Table 3.1.

With the extendable nature of the FPGA fabric, a greater range of delays can be achieved by extending the delay line, creating a longer variable delay line.

3.2.2 Programmable delay switches

In order to allow programmability of the delay line, each of the LUTs on a delay line is implemented as a 16-bit addressable shift register (SRL16), a built-in feature available in Spartan-3E LUTs. This is shown in Figure 3.4. The A[3:0] inputs provide access to any bit in the shift register through the Q output.

The A[2:0] inputs on each of the LUTs are connected to a hex, a double, and a single line connection, respectively. A[3] is unused and connected to ground. To select which of the A[3:0] inputs is output through Q, we determine the truth table values for the output versus the A[3:0] inputs. This is simply Q = A[m], where $m \in \{0, 1, 2, 3\}$, and A[m] is the selected input that we would like to output through Q, as shown in Table 3.3. This logic implementation can also be thought of as a 3-to-1 mux with selection determined via input and output mapping. By shifting in the truth table values for Q output into the shift register, we can reflect which of the A[2:0] inputs (hex, double, or single line connection) is output through Q. More precisely, a LUT will be configured to select single line connection by shifting Q[15:0] = AAAA into the shift register, while Q[15:0] = CCCC and Q[15:0] = F0F0 are used for double and hex line connections, respectively. Hence, a delay line is programmable to generate a desired delay by shifting an appropriate shift register value into each of the LUTs.

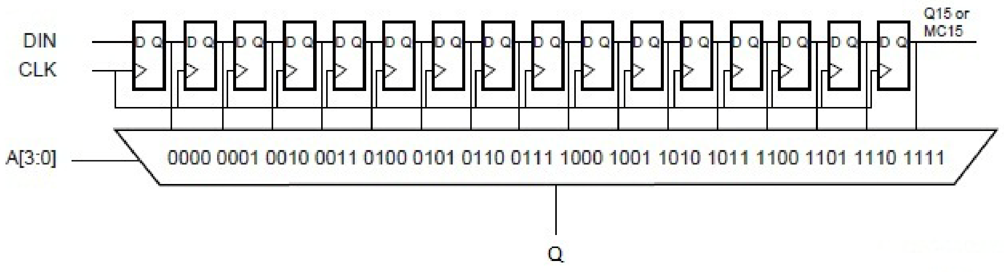


Figure 3.4: Implementing LUT as 16-bit shift register

Ground (unused)	**Hex**	**Double**	**Single**	
A[3]	**A[2]**	**A[1]**	**A[0]**	**Q**
0	0	0	0	Q[0]
0	0	0	1	Q[1]
0	0	1	0	Q[2]
0	0	1	1	Q[3]
0	1	0	0	Q[4]
0	1	0	1	Q[5]
0	1	1	0	Q[6]
0	1	1	1	Q[7]
1	0	0	0	Q[8]
1	0	0	1	Q[9]
1	0	1	0	Q[10]
1	0	1	1	Q[11]
1	1	0	0	Q[12]
1	1	0	1	Q[13]
1	1	1	0	Q[14]
1	1	1	1	Q[15]

	Q[15:0]
Single	AAAA
Double	CCCC
Hex	F0F0

Table 3.3: Shift register values for configuring delay switches to select between single, double, or hex line connections

3.2.3 Pattern recognition array

CLBs in a standard Xilinx FPGA constitute the main logic resources for implementing digital circuits. The CLBs are arranged in a regular array of rows and columns. Each CLB in a Spartan-3E FPGA comprises four slices, two on each left and right side of the CLB. Each slice contains two LUTs. The two slices on the left are called SLICEM and support both logic and shift register functions that are useful for implementing programmable delay switches.

The basic programmable delay line presented serves as a fundamental structure for building programmable delay lines in a delay-based pattern recognition circuit. By taking advantage of the array-based CLB architectural layout organization of logic and interconnect resources in Spartan-3E FPGAs, a pattern recognition circuit consisting of programmable delay lines could be implemented as a block of $N{\times}N$ CLBs array, with two LUTs per CLB in each horizontal and vertical direction, as illustrated in Figure 3.5.

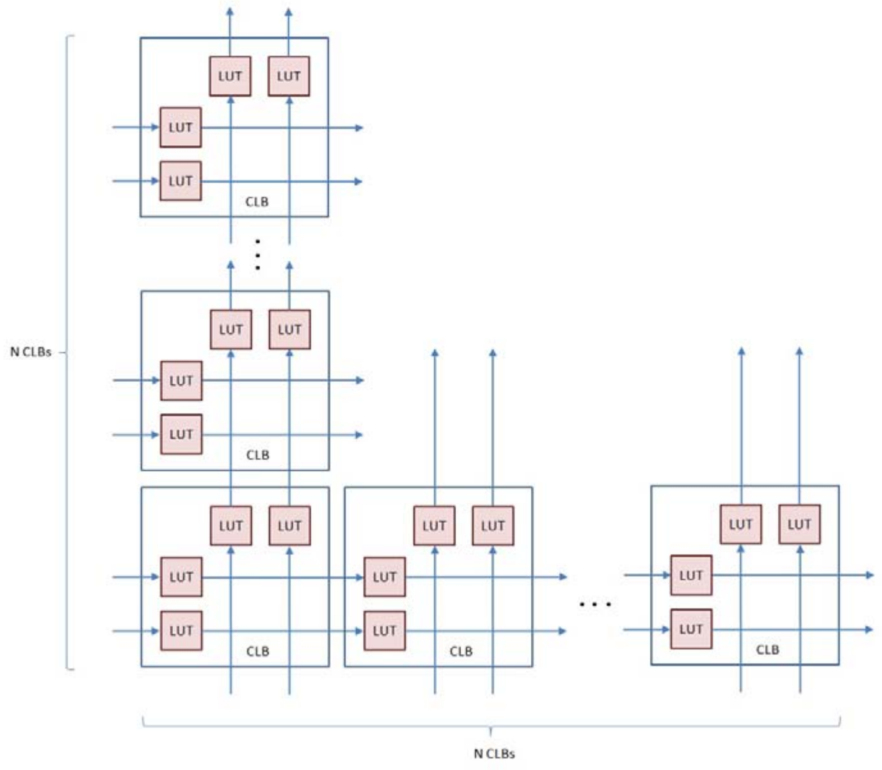


Figure 3.5: $N \times N$ CLBs pattern recognition array

All the LUTs illustrated in the figure are SLICEM LUTs. The total number of LUTs in the array, L, is therefore:

$$L = 2N \times 2N = 4N^2 \tag{3.1}$$

Figure 3.6 illustrates an example of a 8×8 CLBs pattern recognition array that stores and detects patterns of a 4-element vector $\{x_0, x_1, x_2, x_3\}$. The first two elements of the vector x_0 and x_1 are stored via the horizontal programmable delay lines while the last two elements x_2 and x_3 are stored via the vertical programmable delay lines. For example, x_0 is stored via the bottom-most horizontal delay line in each CLB row. The delay lines in each direction are connected one after another from one CLB row or column to the next CLB row or column, for respective vector elements. The connections between the horizontal or vertical delay lines are made via the shortest available routing

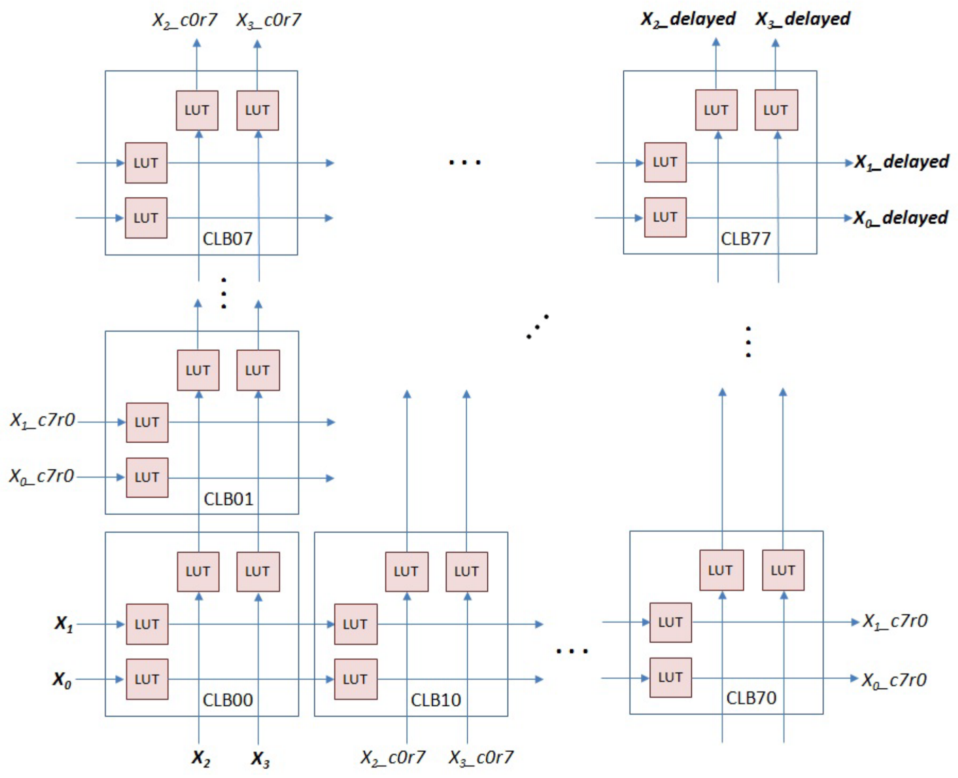


Figure 3.6: 8×8 CLBs pattern recognition array storing a 4-element vector patterns

path determined by the vendor’s FPGA design tools. The direction of a programmable delay line is flexible to be defined differently. For example, the horizontal delay lines may go alternating from left to right and from right to left instead of always from left to right as shown in Figure 3.6. In addition, a programmable delay line may also be formed with a mix of horizontal and vertical delay lines as long as there are unused LUT and interconnect resources available.

The pattern recognition array has the flexibility of storing and detecting different and independent sets of spatiotemporal patterns of *k*-element, *b*-temporal value vectors. Each set of patterns could have its own specifications in terms of number of elements and delay requirements. The specifications of the patterns could also potentially be defined differently and may not necessarily be limited to what is defined in Section 3.1.2. For example, each x_i element of a *k*-element vector could potentially be having different number of temporal values instead of the same.

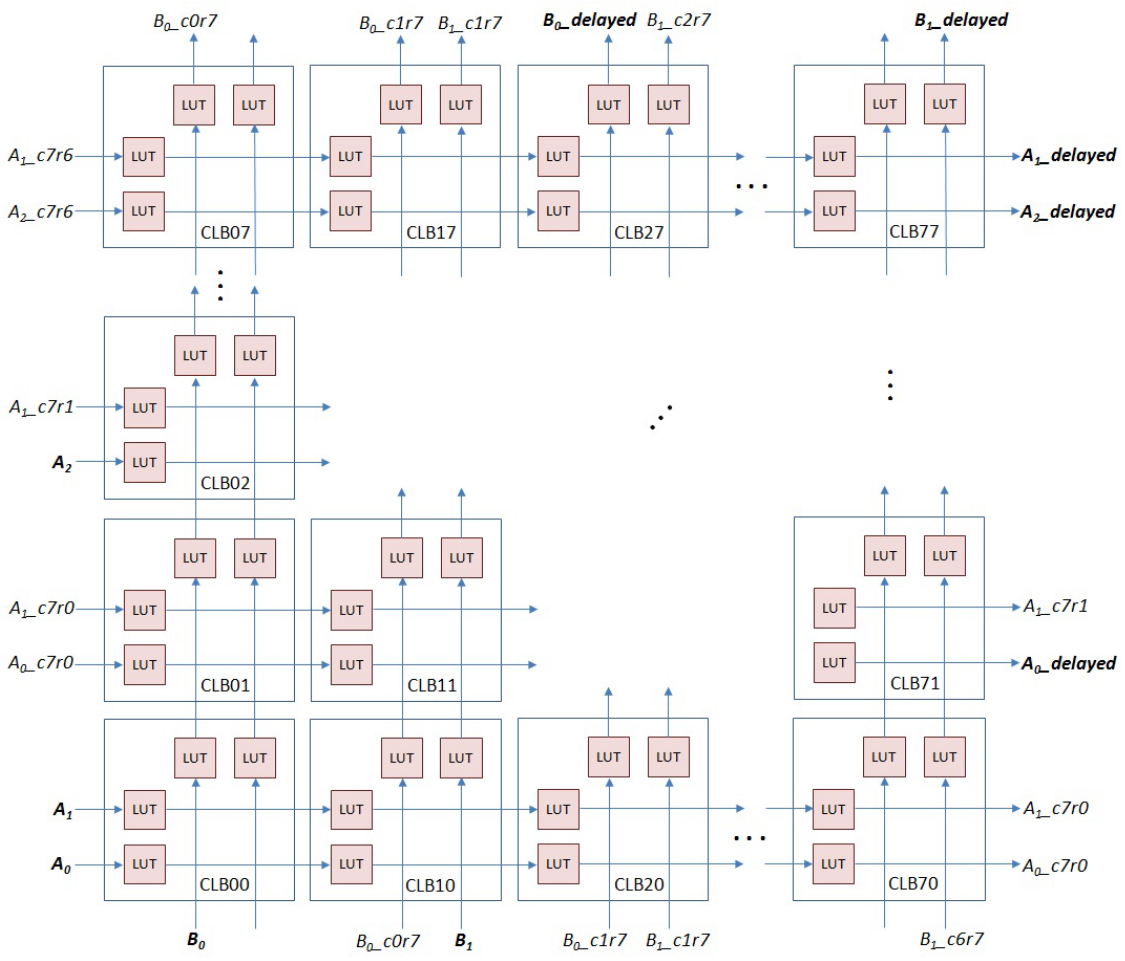


Figure 3.7: A pattern recognition array storing two different and independent sets of patterns, $\{A_0, A_1, A_2\}$ and $\{B_0, B_1\}$

Figure 3.7 illustrates an example of a pattern recognition array that stores two different and independent sets of spatiotemporal patterns – a set of 3-element vector patterns $\{A_0, A_1, A_2\}$ and a set of 2-element vector patterns $\{B_0, B_1\}$. The first set of patterns $\{A_0, A_1, A_2\}$ is stored via the horizontal programmable delay lines while the second set $\{B_0, B_1\}$ is stored via vertical programmable delay lines.

The length of a programmable delay line may vary, depending on the needs of the amount of delay required. For example, the A_0 element in the first set of patterns that requires lesser amount of delay compared to A_1 element may take up 2 rows of the horizontal programmable delay lines, while the latter may take up 8 rows of the programmable delay lines.

The input and output of a programmable delay line could start and end at any LUTs within the array as long as there are sufficient LUT and interconnect resources available to meet the pattern requirements.

3.3 Training

The network can be trained to recognize a given pattern through delay adaptation of the input spikes X_i to a target spike X_{target} using a training algorithm. The pseudo code of the training algorithm is described as below.

Pseudo code of training algorithm

```
1) Initialize all programmable delay lines to the minimum delay
2) For each $X_i$ spike in the pattern,
        delay $X_i$ by incrementing the delay of its programmable delay line until it
        coincides with $X_{target}$ spike
```

Each X_i spike is fed trough a dedicated programmable delay line. In the initialization stage of the training process, each of the programmable delay lines is initialized to the minimum delay. After that, the training algorithm adapts the delay of each of the input spikes X_i such that it coincides with the target spike X_{target} by incrementing the delay of the associated programmable delay line, as illustrated in Figure 3.8.

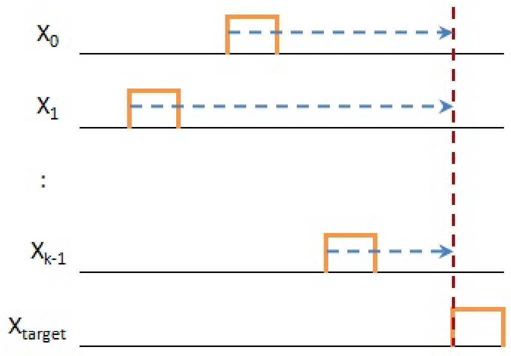


Figure 3.8: Delay adaptation of input spikes to a target spike

The training is deemed to be completed when all of the input spikes X_i are adapted to the target spike X_{target} to cause a coincidence. After the training, when the same pattern is introduced to the circuit, the output of the circuit will be asserted to indicate a detection of the trained pattern.

3.4 Delay-based Pattern Recognition Prototype

To demonstrate the proof-of-concept of the delay-based pattern recognition circuit and the feasibility of its FPGA implementation, a prototype consisting of a 10×10 CLBs pattern recognition array with pattern learning capability is implemented.

Figure 3.9 shows the block diagram of the implemented pattern recognition prototype. The entire circuit was implemented on a Xilinx Spartan-3E XC3S1600E family FPGA that uses 90nm process technology with logic resources density of 33,192 logic cells. The training process is orchestrated by a MicroBlaze soft processor. The shift registers of the LUTs on each programmable delay line are connected one after another as a chain. The configuration values of the delay switches are fed serially into the programmable delay lines from the MicroBlaze processor through a Fast Simplex Link (FSL) interface and a parallel-to-serial converter. A pattern generator is implemented as a state machine that generates spatiotemporal patterns of X_i spikes and a target spike X_{target}. The entire circuit including the training logic utilizes 13.5% (4,494/33,192) of the total logic cells available on the FPGA while the pattern recognition array takes up 1.3% (416/33,192).

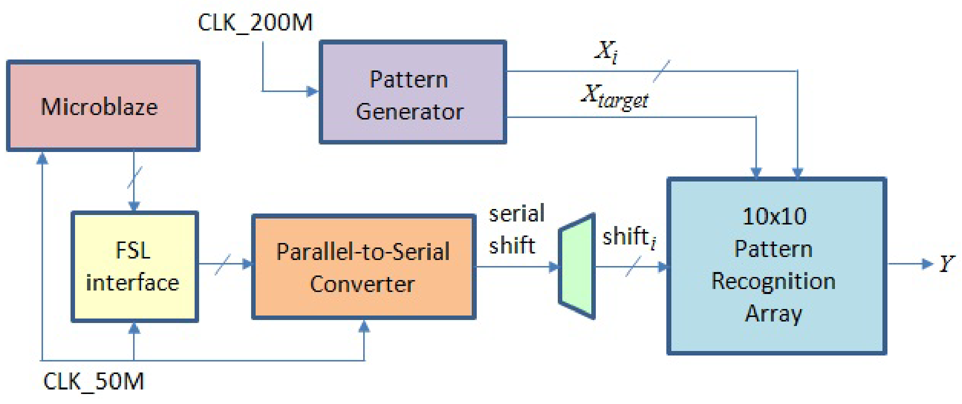


Figure 3.9: Delay-based pattern recognition prototype

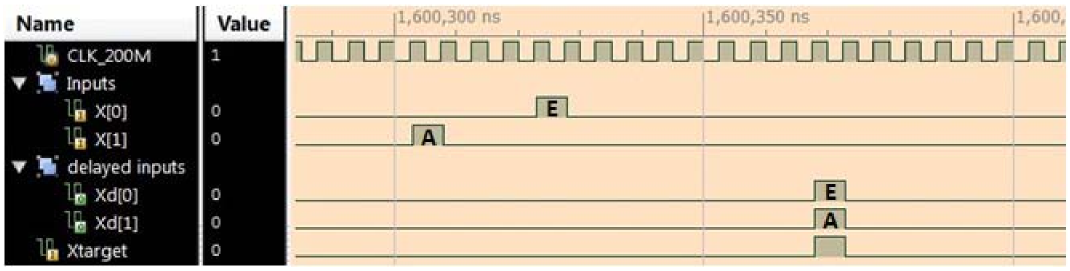


Figure 3.10: Simulation result of a trained pattern {x_0, x_1} = {E, A}

The array is configured to train and detect patterns of a 2-element, 8-temporal value vector {x_0, x_1}, with a pulse width t_{res} of 5 ns. Each of the x_i elements represent a set of 8 characters {A, B, … , H}. x_0 and x_1 are fed in through the horizontal and vertical programmable delay lines respectively. The programmable delay lines in each direction are cascaded one after another to build a long programmable delay line with a sufficient amount of delay to store the pattern.

Figure 3.10 shows the simulation result of a pattern {x_0, x_1} = {E, A} after training. Before training, x_0 and x_1 spikes do not coincide with each other or the target spike X_{target}. After training, x_0 and x_1 spikes are adapted to X_{target} and coincidence occurs. When the same pattern {E, A} is introduced to the circuit again, the output of the AND gate is asserted indicating that the trained pattern is detected.

3.5 Utilization and Capacity

The size of a delay-based pattern recognition circuit may vary depending on the specifications of the patterns to be stored. This section discusses about estimation of hardware resource utilization of programmable delay lines for storing and detecting patterns of a given *k*-element, *b*-temporal value vector. The capacity of a pattern recognition array for storing and detecting different and independent sets of patterns is also discussed in this section.

3.5.1 Hardware utilization of programmable delay lines

As illustrated in Figure 3.2, for a programmable delay line capable of producing *b* temporal values of different delays of equal width t_{res}, the total delay could be produced by the programmable delay line is:

$$D_{line} = b \times t_{res} \tag{3.2}$$

Furthermore, as discussed in Section 3.2.1, based on characterization results of interconnect and LUT delays, we could approximate one LUT-interconnect pair producing a delay of 1 ns. We could therefore estimate the utilization of a programmable delay line, U_{line}, in terms of the number of LUTs, as:

$$U_{line} \approx D_{line} = b \times t_{res} \tag{3.3}$$

Hence, for a given *k*-element, *b*-temporal value vector, the total utilization of programmable delay lines in terms of the number of LUTs in a delay-based pattern recognition circuit, U_{set}, could be estimated as:

$$U_{set} \approx U_{line} \times k \tag{3.4}$$

For example, for a 4-element, 8-temporal value vector, which could produce a total of $8^4 = 4096$ possible spatiotemporal patterns, if a pulse width t_{res} of 3 ns is used, the utilization of the circuit is $U_{set} \approx 96$ LUTs, which is very compact in size.

3.5.2 Capacity of a pattern recognition array

As proposed in Section 3.2.3, a delay-based pattern recognition circuit consisting of programmable delay lines of the presented design could be implemented as a block of $N \times N$ CLBs array. Here, we discuss the potential storage capacity of a pattern recognition array.

We define the capacity, C, as the total number of different and independent sets of spatiotemporal patterns, each represents a k-element and b-temporal value vector, storable by an $N \times N$ CLBs pattern recognition array. Based on calculations in terms of number of LUTs from Equation 3.1 and Equation 3.4, the capacity of a pattern recognition array could be estimated as:

$$C \approx L/U_{set} = 4N^2 / (b \times t_{res} \times k) \tag{3.5}$$

For example, for a 30×30 CLBs array, the capacity for storing different and independent sets of patterns each of a 4-input, 8-temporal value vector is $C \approx 4 \times 30^2 / (8 \times 3 \times 4) = 37$ sets of patterns, where each set is having $8^4 = 4096$ possible spatiotemporal patterns. With the approximation of one LUT-interconnect pair producing a delay of 1 ns, the capacity of a pattern recognition array could be increased in proportion to the number of LUTs in the array. Hence, for modern FPGAs with high hardware resource density, a larger pattern recognition array could be realized for storing a large number of patterns.

CHAPTER 4

A BIOLOGICALLY-INSPIRED AUTO-ASSOCIATIVE MEMORY

4.1 Introduction

A conventional memory, in the context of engineering or computing, stores a data element at a unique address and is capable of retrieving the data element back upon presentation of the complete address. In contrast, an *auto-associative memory* is a type of "content-addressable" memory which does not require an address in order to retrieve data, but instead retrieve a data element in response to a partial presentation of the original copy [72].

In this chapter, the design and implementation of a biologically-inspired auto-associative memory on an FPGA is presented. The design is conceptualized and developed based on a spiking neural network (SNN) model. The architecture effectively utilizes the massive logic and interconnect resources available in an FPGA to model axonal delay elements in biological neural networks. Time delays, rather than binary values are used to represent numeric data; and coincidence is used to perform pattern matching.

Figure 4.1 illustrates the function of the SNN-based auto-associative memory. The system first learns a certain input pattern through a training process. The memory consists of programmable delays and coincidence detectors. The input pattern is stored by adapting the programmable delays connected to the coincidence detectors. After

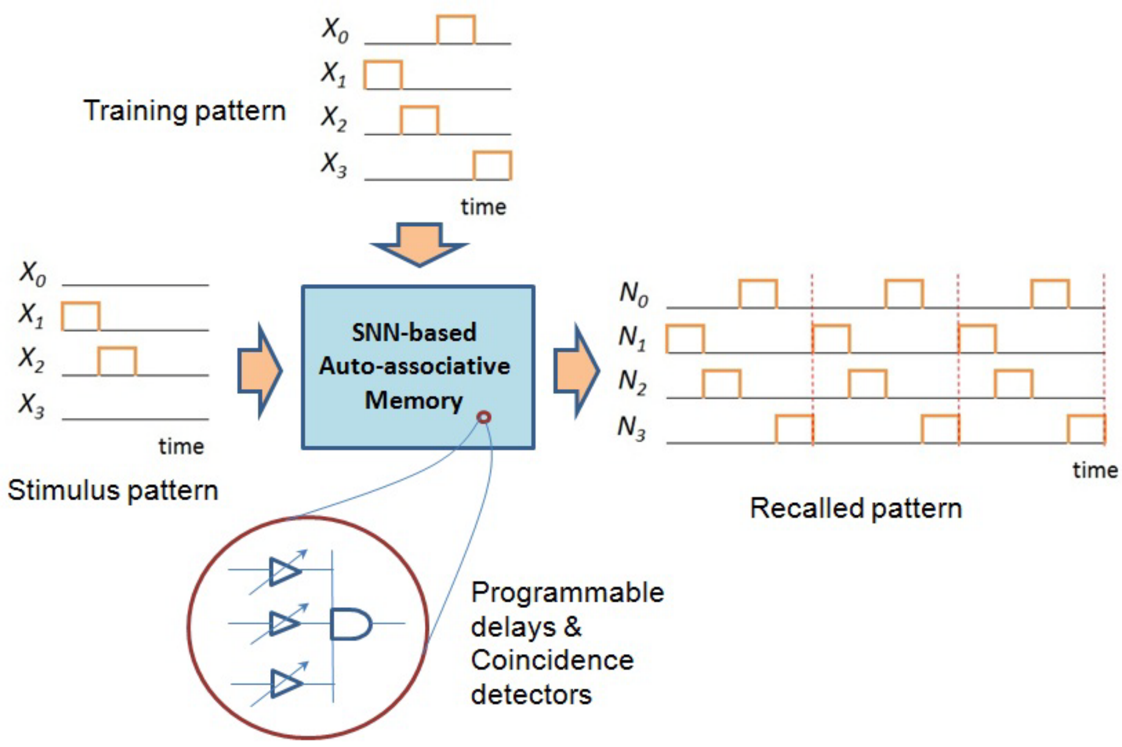


Figure 4.1: SNN-based auto-associative memory

training, when a partial input pattern is presented, the complete version of the training pattern will be retrieved from the memory and a sustained periodic replay of the pattern effected.

Similar to the pattern recognition prototype developed in Chapter 3, the pattern processing unit of the memory is also implemented using a clockless design approach. The system is however implemented on an FPGA from a different vendor to demonstrate that such delay-based pattern recognition models and the clockless design approach are viable across different FPGA platforms and not only limited to specific FPGA devices.

4.2 Spatiotemporal Spike Sequence

Similar to the delay-based pattern recognition system developed in Chapter 3, the auto-associative memory stores and reproduces memories in the form of *spatiotemporal spike sequences*.

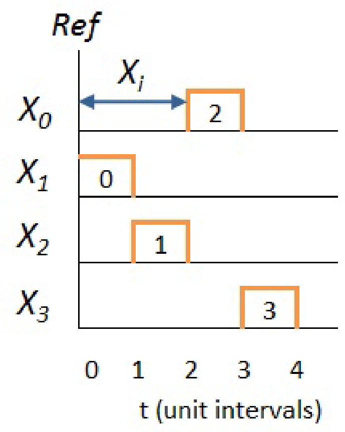


Figure 4.2: A temporal pattern $\{x_0, x_1, x_2, x_3\} = \{2, 0, 1, 3\}$

Here, we define a spatiotemporal spike sequence to be represented by a vector of k elements $\{x_0, x_1, \ldots , x_{k-1}\}$, with each x_i representing the time between a reference signal and the rising edge of a pulse. Through this temporal coding scheme, a vector of real numbers is translated into a sequence of pulses, or spikes, as shown in Figure 4.2.

4.3 Derivation of Model

The SNN-based memory model developed in this chapter was derived from a Java-based spiking memory model described by Wills [72]. Wills' work aims to develop a SNN-based auto-associative memory model capable of storing and retrieving memories in the form of spatiotemporal spike sequences or patterns. The model developed by Wills stores a number of different spatiotemporal spike sequences as an auto-associative memory so that any stored pattern can be recalled by the network when presented with a partial version of that pattern.

In Wills' model, each neuron is endowed with several multi-input coincidence detectors each of which may be used to detect spikes. The input spikes to a neuron's coincidence detector are referred to as the 'context spikes' for the given neuron. There are time-delay connections between the neuron outputs and the inputs to other neurons' coincidence detectors. The recall of a particular spatiotemporal spike sequence is manifested by the recurrent activation of that spatiotemporal spike sequence within the population of neurons. Thus, there is a closed network of neurons that will spike repeatedly if stimulated in the correct temporal sequence. Because each neuron may have multiple coincidence detectors, each neuron can participate in multiple spatiotemporal spike patterns. As well, this auto-associative memory model can store multiple patterns and recall them concurrently.

We explore a SNN model that stores a single simple pattern in comparison to Wills' software-based model which can store and concurrently recall multiple patterns. In our simplified model, each neuron in the network is associated with only one coincidence detector that responds to the context spikes of a particular stored pattern, and in turn contributes to the recurrent activation of that single pattern.

While many of the existing FPGA implementations of SNNs are based on standard digital designs with sequential logic [12, 14, 64, 67], where speed performance is often limited by clock frequencies of such circuits; the SNN developed in this chapter uses only combinational logic and no sequential clocking elements are involved in the feedforward path. Hence, the memory has the potential to process patterns at very high speed and low latency, at a level beyond a conventional synchronous circuit could achieve.

4.4 Architecture

Having introduced the ideas behind our SNN-based auto-associative memory model, this section describes precisely the architecture of the memory.

4.4.1 Programmable delay lines

Information, *i.e.* patterns, are stored via *programmable delay lines* that interconnect the neurons. A *k*-input pattern requires *k* spikes to represent that pattern with each spike corresponding to the output of a neuron. Thus, the SNN model for the *k*-input pattern consists of *k* neurons, each with a multi-input coincidence detector. The input spikes to a neuron's coincidence detector are referred to as the 'context spikes' for the given

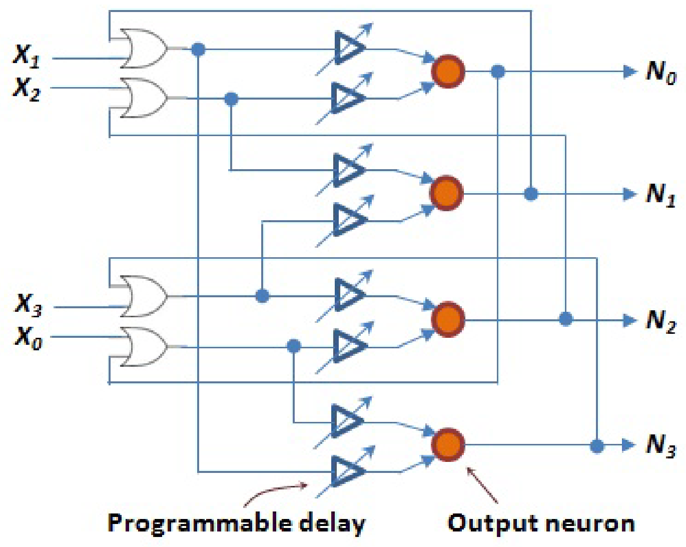


Figure 4.3: Network configuration of a 4-input auto-associative memory with two contexts

neuron. The inputs of the coincidence detector are driven by a subset of the other *k-1* neurons, which are in turn referred to as the ‘context spike neurons’. Each input of the coincidence detector is connected to its context spike neuron through a programmable delay line. If each multi-input coincidence detector receives *c* context spikes, then a total of $k \times c$ programmable delay lines are required to form the network between each output neuron and its associated context spike neurons. Figure 4.3 illustrates a network configuration of a SNN-based auto-associative memory for the case of $k = 4$ and $c = 2$.

Figure 4.4 illustrates the detailed architecture for the programmable delay lines. The inputs of an output neuron N_i are connected to its context spike neurons N_j through programmable delay lines. Using the programmable delay lines, the interconnection of each output neuron with its associated context spike neurons forms a closed feedback network that drives the recurrent activation of the stored spike pattern. In comparison to the delay line architecture developed in Chapter 3, which explores the utilization of inter-logic block routing resources for implementing delays, the design in this chapter focuses on utilizing the interconnect resources within a logic block.

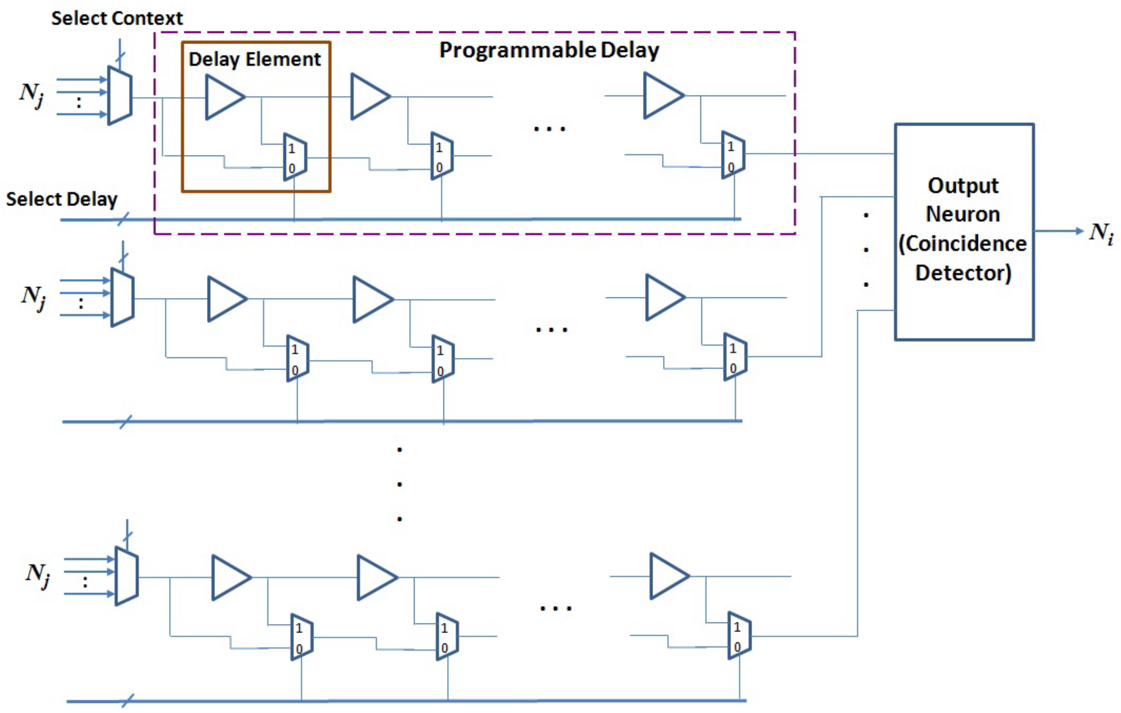


Figure 4.4: Architecture of programmable delay lines

The spikes of an input pattern consist of pulses of equal width t_{spike}. Each programmable delay line consists of a cascade of n delay elements, where n is determined based on the maximum programmable delay, d_{max}, for the delay line. d_{max} is essentially the maximum allowable delay for the context spike to be adapted to the target spike. With similar design goals as in Chapter 3, the delay lines are designed such that they nicely fit the FPGA architecture and allow compact block-structure implementations. Each delay element is well represented by a Logic Array Block (LAB) in an Altera Cyclone II FPGA. The delay of each delay element can be set as long delay d_{long} or short delay d_{short}, through the configuration of the control multiplexer (mux). The long delay is an interconnection of 15 logic elements (LEs) while the short delay is a direct connection of a single wire, as illustrated in Figure 4.5. The total delay of an n-delay element delay line, d_{line}, must satisfy $nd_{short} \leq d_{line} \leq nd_{long}$. The feedback connection from an output neuron to a delay line introduces an additional delay of

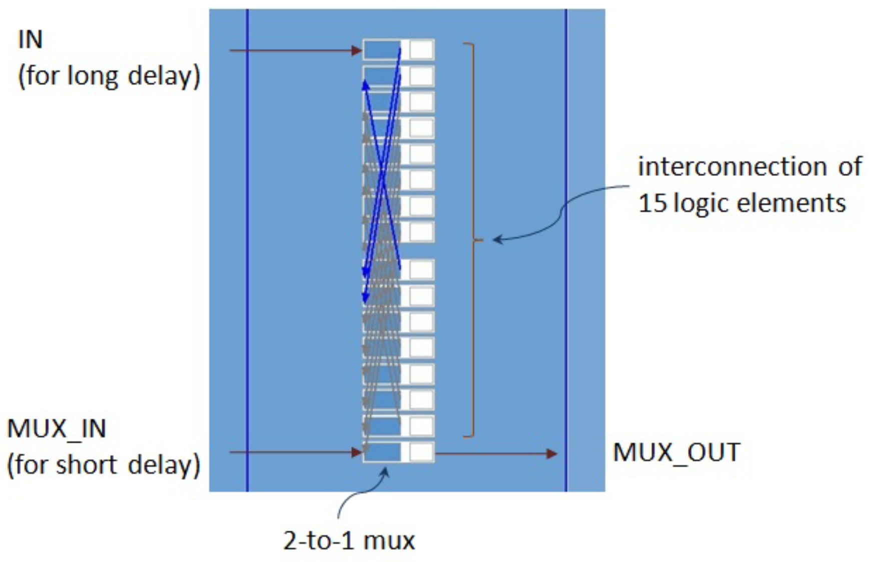


Figure 4.5: Delay element

$d_{feedback}$. The total delay imposed on a spike, d_{total}, is therefore:

$$d_{total} = d_{line} + d_{feedback} \tag{4.1}$$

4.4.2 Coincidence detectors

A *coincidence detector* detects the coincidence of delayed context spikes and is implemented using a logical ‘AND’ gate. Since an AND operation between two coincidence pulses with a small time offset produces a pulse with smaller width, the recurrently recalled pattern will eventually disappear. To overcome this problem, two solutions were implemented. The second solution was an improvement over the first one.

In the first solution, the output of the AND gate is delayed and regenerated through a two-input OR gate, as shown in Figure 4.6. The delay element used here is also implemented using interconnect resources. With a wider pulse at the output of the OR gate, there is also a possibility that all the outputs of the network will eventually settle at the logical high state. Hence, a toggle flip-flop is used to select between the

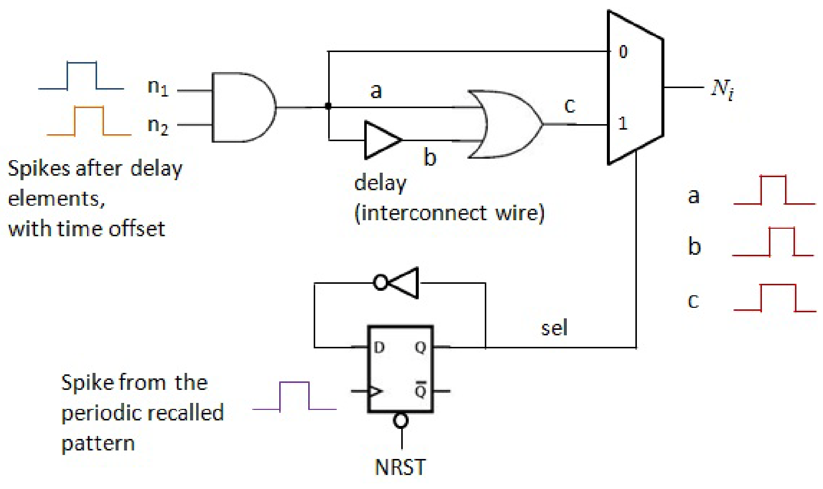

n_1
n_2
Spikes after delay elements, with time offset
a
b
c
delay (interconnect wire)
0
1
N_i
sel
D
Q
$\bar{Q}$
NRST
Spike from the periodic recalled pattern

Figure 4.6: Pulse-muxing coincidence detector

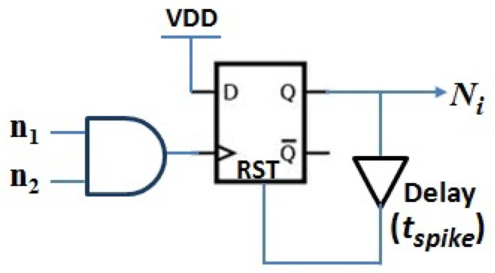


Figure 4.7: Coincidence detector with constant-width pulse generator

smaller and larger pulse over time, maintaining the original pulse width and sustaining the recall of the pattern indefinitely.

The second solution uses less hardware resources and provides much better control of the pulse width. In this solution, the output of the AND gate is connected to a pulse generator circuit that produces pulses of constant width. The output spike from the AND gate triggers the D flip-flop to logical high state which later gets reset back to low state through a fixed-delay feedback that resets the D flip-flop, as shown in Figure 4.7. The delay element used in the feedback loop is also implemented using interconnect resources. With this solution, all the spikes in the pattern are always maintained with a constant pulse width and the recall of the pattern can therefore be sustained indefinitely.

4.5 Training Algorithm

A particular spike pattern is stored in the SNN using a training algorithm consisting of two steps – establishing the correct context neuron interconnection and setting the correct delays for the programmable delay lines, as described in the pseudo code below.

Pseudo code of training algorithm

```
1) For each $N_i$ spike in the pattern,
    • Identify a set of preceding context spikes $N_j$ that triggers $N_i$
    • Make a connection from each context spike neuron $N_j$ to $N_i$ neuron
2) For each context spike $N_j$ of $N_i$ spike,
    • Adapt the delay of $N_j$ spike such that it coincides with $N_i$ spike
```

For each output neuron N_i representing a spike in a pattern, the algorithm identifies a set of preceding context spike neurons N_j that triggers the target output neuron N_i. The architecture developed in this chapter uses simple 4-input patterns (k = 4) for testing to demonstrate the feasibility of the design and the number of preceding context spikes c is set to 2. The training pattern is treated with wrap-around in the time domain, *i.e.* the last spike in the pattern is treated as being the one preceding the first spike. This configuration enables the recurrent recall of the pattern. The recall of the pattern is achieved through presentation of a sub-pattern with contiguous spikes, *i.e.* a subset of spikes with neighboring time relationship in the original pattern. To set the delay from the context spike to the target spike, the algorithm calculates the delay between the context spike and the target spike, $d_{context,target}$, for achieving coincidence; and determines the number of delay elements required, n_{req}, based on a delay function. The delay function, D, in terms of the number of delay elements, n, is initially characterized and obtained from simulation. The delay muxes on the programmable delay lines are then configured appropriately as according to the value of n_{req} calculated.

For example, consider a temporal pattern of $\{x_0, x_1, x_2, x_3\}$ = {2, 0, 1, 3}, each x_i spike in the pattern is represented by the output neuron N_i spike, respectively. Each spike has width t_{spike} and the period of the pattern is 4 unit intervals. For N_0 spike in this pattern, its preceding context spikes are N_2 and N_1 spikes. The delay from N_2 spike to N_0

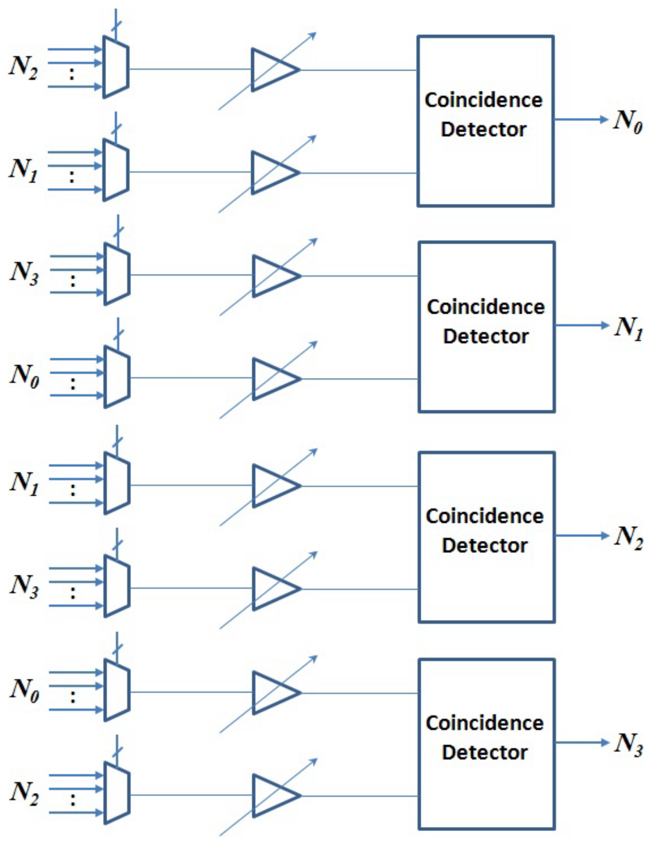


Figure 4.8: Connections between context neurons and the target neurons of an example pattern {2, 0, 1, 3}

spike, $d_{2,0}$; and from N_1 spike to N_0 spike, $d_{1,0}$; is equal to 1 unit interval and 2 unit intervals, respectively. The algorithm makes a connection from each of the preceding context spike neurons N_2 and N_1 to the target neuron N_0 through appropriate configurations of the context muxes as in Figure 4.8. Similarly, for each of N_1, N_2 and N_3, the algorithm identifies the preceding context spikes and makes connections from the preceding context spikes neurons to the target neurons. The connections between the N_0, N_1, N_2 and N_3 target neurons and their respective context spike neurons for the example pattern are shown in Figure 4.8. The recall of the pattern can be achieved via presentation of a contiguous sub-pattern such as $\{x_1, x_2\} = \{0, 1\}$, $\{x_0, x_2\} = \{2, 1\}$, $\{x_0, x_3\} = \{2, 3\}$ or $\{x_1, x_3\} = \{0, 3\}$.

4.6 Implementation

This section describes a complete implementation of the SNN-based auto-associative memory on a single FPGA.

4.6.1 The complete system

Figure 4.9 shows the top-level block diagram of the entire memory system.

A 400MHz PLL-clocked counter and a set of registers are used to capture the timing of the training pattern. A ‘pattern start’ pulse is given to the system to signal the start of the training pattern and activate the counter. The registers store the timing for each spike of the training pattern. The Nios II soft processor runs a C-program that executes the training algorithm, reads the spike timings from the registers, calculates the delays between spikes, and applies appropriate settings on context neurons connections and delay configurations. This base design serves as a representation of the potential for the utilization of the massive logic and interconnect resources available in an FPGA as delay elements for building a fast-processing SNN-based memory architecture. The system is capable of learning and recalling a 4-input pattern with temporal coding $\{x_0, x_1, x_2, x_3\}$ where each x_i represents a real number encoded in the pattern.

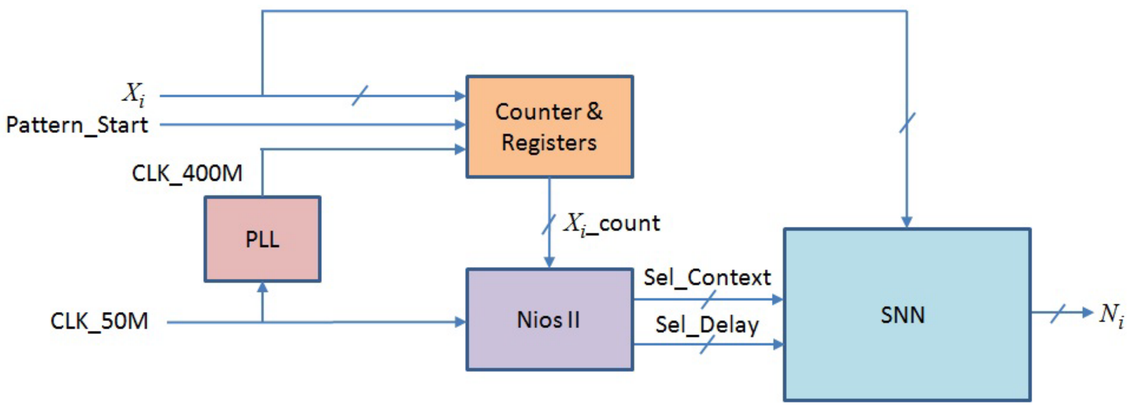


Figure 4.9: Complete system of the SNN-based auto-associative memory

The generation of an input pattern to the auto-associative memory is achieved using a hardware state machine that produces the spatiotemporal spike sequence. It is run on a PLL clock with a frequency of $1/t_{spike}$ for generating the required input pattern.

While the duration of an action potential emitted by a biological neuron is typically 1-2 ms [55], the SNN-based memory developed in the initial stage was configured to work with a pulse width t_{spike} of 60 ns to save on hardware resources and keep the entire network in manageable size since larger pulse widths would require more delay elements. In order to examine the maximum speed performance, the memory was later optimized for processing with patterns of smaller pulse width, which is later discussed in Section 4.6.2.

The delay function that is used to determine the number of delay elements required for achieving coincidence was characterized and obtained from simulation. By plotting the amount of delay, D, against the number of delay elements, n, we obtain the delay function:

$$D = 6n + 29 \tag{4.2}$$

where D is expressed in nanoseconds. The plot of the delay function is shown in Figure 4.10.

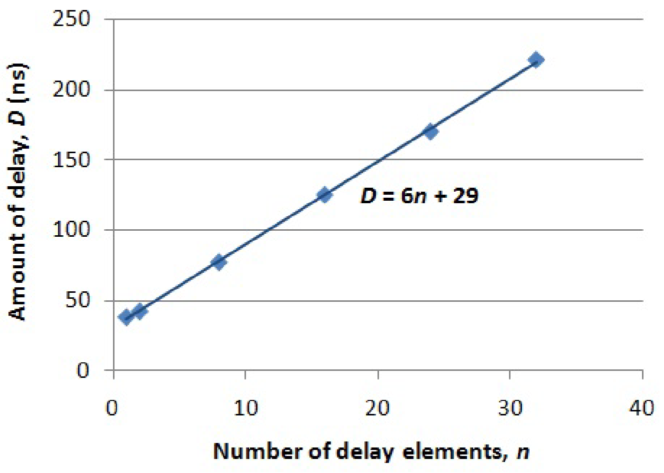


Figure 4.10: Plot of delay function

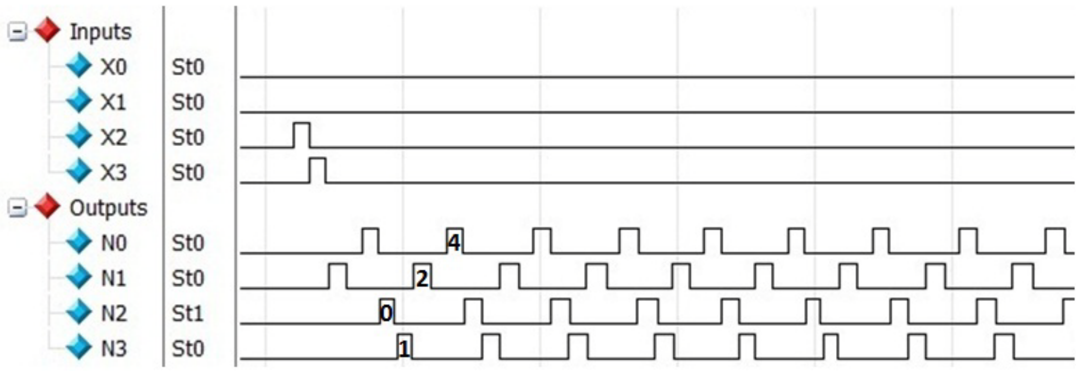


Figure 4.11: Simulation result of pattern $\{x_0, x_1, x_2, x_3\} = \{4, 2, 0, 1\}$

The simulation results found that one LAB delay element provides a delay of 6 ns when enabled, *i.e.* d_{long} = 6 ns. In this work, since we used simple patterns for which the context spike delay was no more than 3 unit intervals (*i.e.* d_{max} = 60 ns × 3 = 180 ns), it was determined that 32 delay elements per delay line was used to provide sufficient context spike delay of up to 180 ns. Equation 4.2 verifies that one delay element does indeed provide a 6 ns delay and also implies that there is an overhead of 29 ns propagation delay across the delay line and the feedback connection.

Figure 4.11 shows a sample simulation result of an input pattern with temporal coding $\{x_0, x_1, x_2, x_3\} = \{4, 2, 0, 1\}$. Each spike of the pattern is of a 60 ns pulse width and the period of the pattern is 5 unit intervals, or 300 ns. By applying the algorithm on the input pattern, the preceding context spikes for each of N_0, N_1, N_2 and N_3 output neurons are N_1 and N_3, N_3 and N_2, N_0 and N_1, and N_2 and N_0, respectively. The respective number of delay elements required to adapt each of the context spikes to the target spike was also determined by applying the delay function in Equation 4.2. For the N_0 spike in this pattern, the context spikes N_1 and N_3 were delayed by 120 ns (15 delay elements) and 180 ns (25 delay elements), respectively, for them to achieve coincidence that results in the triggering of N_0 spike. In the figure, N_1 spike was triggered first in response to the input of its context spikes x_2 and x_3.

Output Neuron	Context Neurons	Delay (unit DEs)
N_0	*N_1*	15
	N_3	25
N_1	*N_3*	5
	N_2	15
N_2	*N_0*	5
	N_1	25
N_3	*N_2*	5
	N_0	15

Table 4.1: Context neurons and delay settings of each output neuron

Table 4.1 shows the settings of the context neurons for this pattern and the respective number of delay elements required to achieve coincidence for each output neuron as a result of the learning process. The recall of the pattern was successfully triggered by stimulating the first two spikes of the pattern, *i.e.* $\{x_2, x_3\} = \{0, 1\}$.

The entire auto-associative memory system was implemented and tested on an Altera Cyclone II EP2C35 family FPGA, which uses 90nm technology and allows resource utilization of up to 33,216 LEs, with 16 LEs per LAB.

Figure 4.12 shows the test results of pattern {4, 2, 0, 1} captured on an oscilloscope. The outputs of N_0, N_1, N_2 and N_3 neurons were shown in the order from top to bottom of the figure respectively. The pattern was successfully recalled and sustains itself indefinitely.

Table 4.2 indicates the hardware resource utilization for the implementation of the auto-associative memory. The entire system utilizes 22% (7,206/33,216) of the total LEs available on the FPGA while the SNN takes up 13% (4,305/33,216). It is interesting to see that the SNN is implemented exactly as expected where the circuit consists of almost entirely combinational logic (4,304/4,305). The LUTs associated with the interconnect wires used to implement the delay elements takes up 95% (4,096/4,305) of the total LEs used in the SNN while logic takes up 5% (209/4,305). The LUTs are used to buffer the interconnect wires and not for logic. For a delay line consisting of 32 LABs in the presented architecture, the resulting total delay is 221 ns. Interconnect contributed 149.5 ns of the total delay and LUTs 71.5 ns. Because interconnect

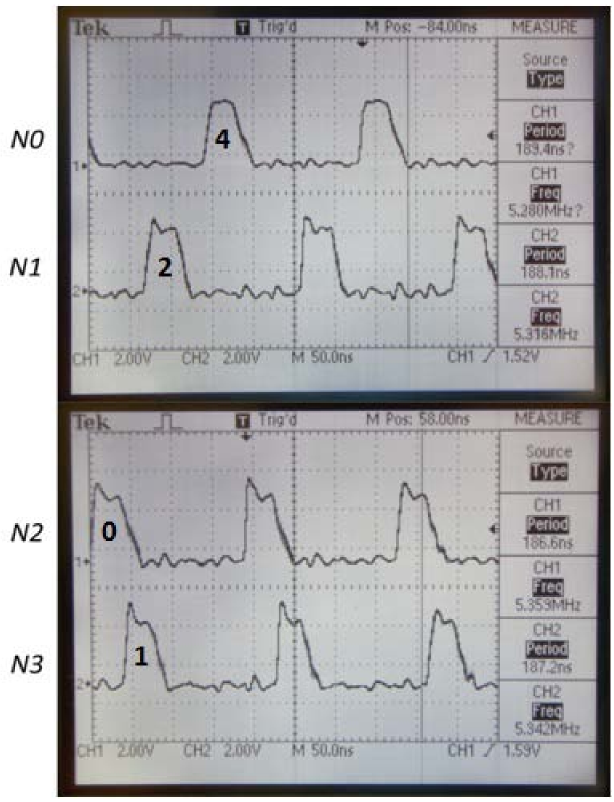


Figure 4.12: The recall of pattern {4, 2, 0, 1} captured on oscilloscope

	Entire System	**SNN**
Total logic elements	7,206 / 33,216 (22%)	4,305 / 33,216 (13%)
Combinational functions	6,957 / 33,216 (21%)	4,304 / 33,216 (13%)
Dedicated logic registers	1,510 / 33,216 (5%)	4 / 33,216 (<1%)
Delay element's LUTs	-	4,096 / 4,305 (95%)
Logic	-	209 / 4,305 (5%)

Table 4.2: Resource utilization on cyclone II FPGA

contributes more to the delay than the LUTs by area, it is more efficient to add more delay as necessary by increasing the amount of routing used in the interconnect.

The figure for SNN utilization in terms of total LEs consumption also agrees with the estimation based on architectural parameters described by Equation 4.3:

$$U_{SNN} \approx [(B_{line} \times L) + B_{cd}] \times E_{LAB} \tag{4.3}$$

Here, B_{line} represents the total number of LABs utilized in a delay line, which in turn is made up of the number of LABs utilized for context selection mux, m_{mux}, and the number of delay elements, n. L is the total number of delay lines in the SNN and can be represented by the term ($k \times c$), where k is the number of inputs to the network and c is the number of context neuron connections, as described in Section 4.4. B_{cd} represents the total number of LABs utilized for coincidence detectors which is essentially the product of the number of LABs utilized for one coincidence detector, m_{cd}, and the number of SNN outputs, k. E_{LAB} is the total number of LEs available per LAB (*i.e.* 16 for Cyclone II FPGAs). Equation 4.3 can therefore be more specifically expressed as:

$$U_{SNN} \approx [(m_{mux} + n) \times (k \times c) + (k \times m_{cd})] \times E_{LAB} \qquad (4.4)$$

m_{mux} is 1, m_{cd} is 2, n is 32 while k and c are 4 and 2, respectively, for the presented SNN.

Given the compact size of the SNN, the capacity of the memory can be expanded by replicating the SNN for multiple pattern storage. The total number of storable patterns, P, on a given FPGA can therefore be estimated by:

$$P \approx E_{FPGA} / U_{SNN} \qquad (4.5)$$

where E_{FPGA} is the total number of LEs available on an FPGA. With the availability of high-density FPGAs such as Stratix IV with 820k LEs, the total number of storable patterns could be up to 200 if the memory is implemented on such a platform.

4.6.2 Optimizing for speed performance and size

To examine the maximum speed performance of the memory, it is interesting to optimize the architecture for operating with even smaller pulse width. The architecture was later optimized to work with t_{spike} as small as 6 ns.

The optimization is achieved by reducing the length of the programmable delay lines since lesser delay elements are required to process patterns with smaller pulse width. The optimized architecture not only could allow a rapid processing of patterns with considerably small pulse widths and inter-spike intervals but also smaller and more compact in terms of size. The delay function for the optimized memory is:

$$D = 6n + 6 \tag{4.6}$$

The delay produced by one LAB delay element when enabled remained at 6 ns while the overhead propagation delay is reduced significantly to 6 ns.

The auto-associative memory with the optimized architecture was tested with the same set of random patterns and correct operation was achieved in all instances both in simulation and in hardware. Figure 4.13 shows the recall of a temporal pattern $\{x_0, x_1, x_2, x_3\} = \{1, 0, 4, 2\}$ captured on an oscilloscope. In the pattern, each spike has a 6 ns pulse width and the period of the pattern is 5 unit intervals, or 30 ns.

The context spikes N_1 and N_2 were delayed by 6 ns ($n_{req} = 0$) and 12 ns ($n_{req} = 1$) respectively to achieve coincidence and cause triggering of the N_0 spike. The recall of the pattern can be successfully triggered from a partial representation of the pattern, *i.e.* $\{x_0, x_1\} = \{1, 0\}$. Note that N_3 spike was triggered first in response to the input of its context spikes x_0 and x_1.

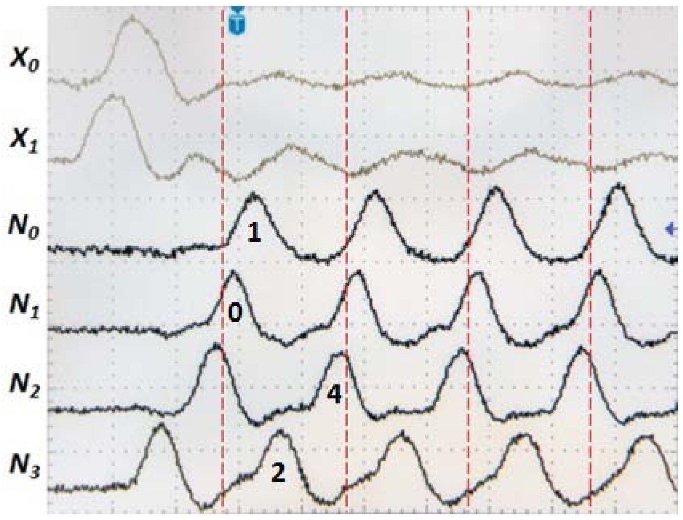


Figure 4.13: Recall of pattern {1, 0, 4, 2} captured on oscilloscope; voltage 5V/div, time 10ns/div

In terms of hardware resource utilization, the entire system utilizes 8% (2,764/33,216) of the total LEs available on the FPGA while the SNN takes up less than 1% (328/33,216). Given the compact size of the optimized architecture, approximately 2500 patterns could be stored in high-density FPGAs such as Stratix IV.

CHAPTER 5

RESULTS

5.1 Introduction

Having described the design and implementation of the delay-based pattern recognition circuit and the auto-associative memory in Chapter 3 and 4, respectively, this chapter presents the results of tests and hardware utilization of the two systems.

5.2 Delay-based Pattern Recognition Circuit

This section elaborates on the delays evaluation of programmable delay lines presented in Chapter 3. It also presents the simulation result of a manually-trained pattern as well as the test results for the pattern recognition prototype described in the same chapter. The hardware utilization and physical layout of the pattern recognition prototype are also presented.

5.2.1 Delays of programmable delay lines

As described in Chapter 3, programmable delay lines in a delay-based pattern recognition circuit play an important role in adjusting the arrival time of spikes of a temporal pattern for triggering a coincidence event. A pattern is trained and recognized through the detection of coincidence of delayed spikes.

Delay switch settings for DS0 to DS6	**Resulting LUTs and interconnects used**	**Delays (ns)**
S,X,X,X,X,X,H	2 LUTs + 1 hex	2.13
S,X,D,X,D,X,D	4 LUTs + 3 doubles	4.09
S,X,D,X,D,S,S	5 LUTs + 2 doubles + 2 singles	5.22
S,S,S,X,D,X,D	5 LUTs + 2 doubles + 2 singles	5.19
S,S,X,D,X,D,S	5 LUTs + 2 doubles + 2 singles	5.21
S,X,D,S,S,S,S	6 LUTs + 1 double + 4 singles	6.18
S,S,S,S,X,D,S	6 LUTs + 1 double + 4 singles	6.17
S,S,S,S,S,X,D	6 LUTs + 1 double + 4 singles	6.18
S,S,S,S,S,S,S	7 LUTs + 6 singles	7.15

Table 5.1: Resulting delays against different combinations of delay switch settings for 7-LUT programmable delay line

The design of the programmable delay line allows many combinations of delay switch settings that produce a series of variable delays. A spike has various signal paths to propagate through a delay line depending on the settings of the delay switches. Each of the various signal paths may have a different number of LUTs and the type of interconnects used along the path, and hence a different resulting delay.

A basic 7-LUT programmable delay line as described in Section 3.2.1 of Chapter 3 was simulated to evaluate the delays produced across different combinations of delay switch settings. Table 5.1 shows the list of all possible combinations of delay switch settings and the simulated delays for these settings. Some of the combinations of the settings resulted in the same number and type of LUTs and interconnects, and hence gave a similar delay. For example, the settings of {S,X,D,X,D,S,S}, {S,S,S,X,D,X,D} and {S,S,X,D,X,D,S} all resulted in the use of 5 LUTs, 2 double line and 2 single line connections, and produced a delay of about 5.2 ns. The signal path resulted from each of these three settings is illustrated in Figure 5.1 (a), (b) and (c), respectively.

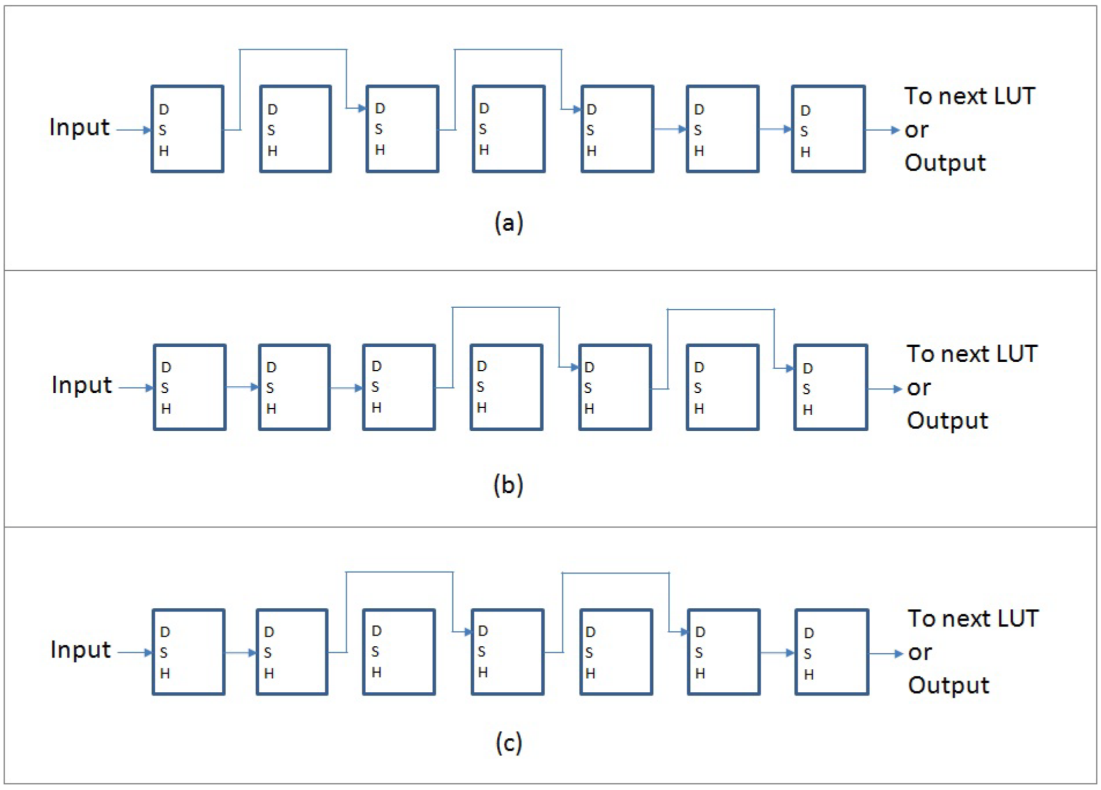


Figure 5.1 (a)-(c): Resulting signal path for delay switch settings of {S,X,D,X,D,S,S}, {S,S,S,X,D,X,D} and {S,S,X,D,X,D,S}, respectively

Longer programmable delay lines with more combinations of hex line connections and signal paths were also evaluated and similar results were obtained. Table 5.2 shows the list of possible combinations of delay switch settings and the simulated delays for a 9-LUT programmable delay line. In this example, both {S,X,X,X,X,X,H,X,D} and {S,X,D,X,X,X,X,X,H} resulted in the use of 3 LUTs, 1 hex line and 1 double line connections, and produced a similar delay of close to 3.1 ns. The resulting signal path for each of these settings is illustrated in Figure 5.2 (a) and (b), respectively.

Delay switch settings for DS0 to DS8	Resulting LUTs and interconnects used	Delays (ns)
S,X,X,X,X,X,H,X,D	3 LUTs + 1 hex + 1 double	3.08
S,X,D,X,X,X,X,X,H	3 LUTs + 1 hex + 1 double	3.05
S,X,X,X,X,X,H,S,S	4 LUTs + 1 hex + 2 singles	3.71
S,S,X,X,X,X,X,H,S	4 LUTs + 1 hex + 2 singles	3.62
S,S,S,X,X,X,X,X,H	4 LUTs + 1 hex + 2 singles	3.64

Table 5.2: Resulting delays against different combinations of delay switch settings with hex line connections for 9-LUT programmable delay line

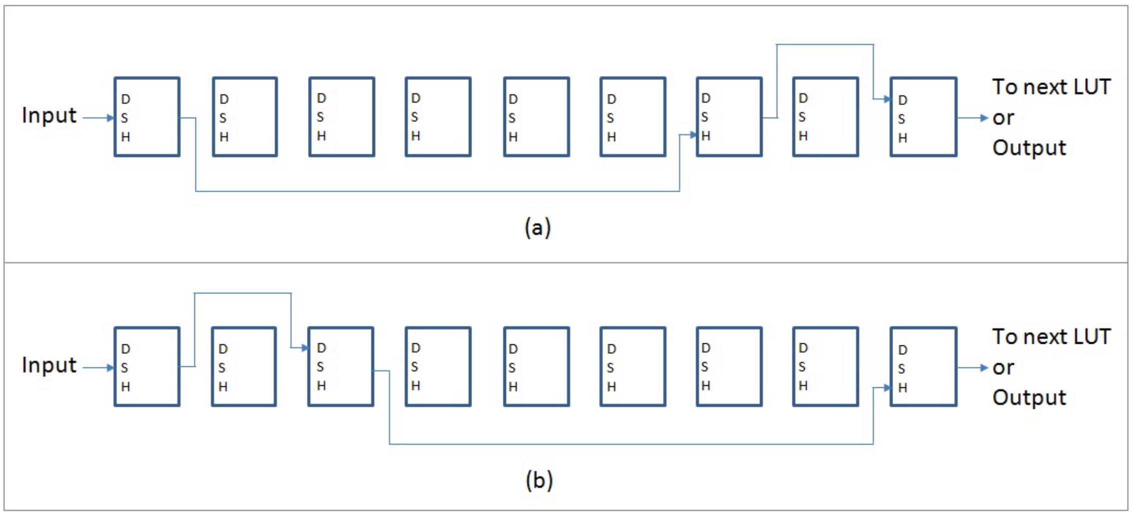


Figure 5.2 (a) and (b): Resulting signal path for delay switch settings of {S,X,X,X,X,X,H,X,D} and {S,X,D,X,X,X,X,X,H}, respectively

5.2.2 Simulation of pattern recognition

A simulation was set up for the testing of pattern recognition of five ASCII characters, each having 26 possible values from ‘A’ to ‘Z’, which in this case is detecting patterns of a 5-element, 26-temporal value vector.

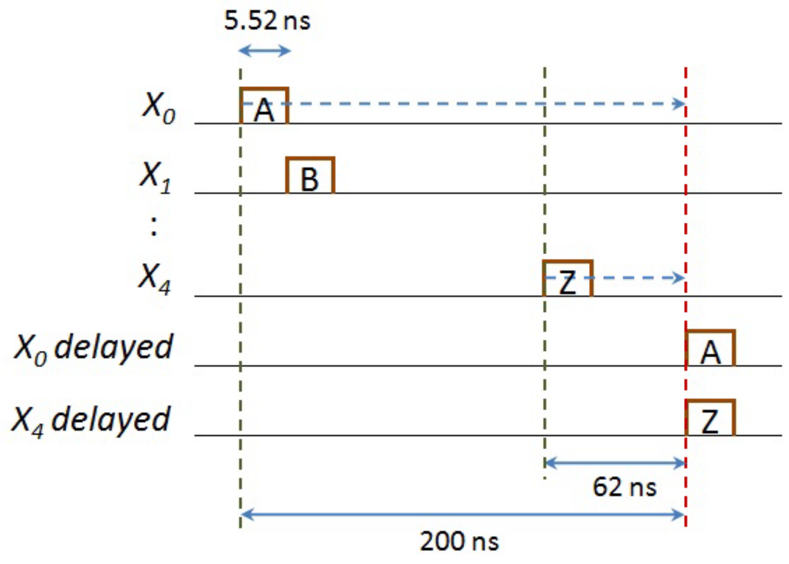


Figure 5.3: Encoding of characters ‘A’ to ‘Z’ into temporal space of 200 ns

Five programmable delay lines were implemented on a Xilinx Spartan-3E FPGA and simulated using the vendor’s FPGA design tools. Each delay line is a cascade of 9 rows of 20-LUT delay lines that gives variable delays ranging from 62 ns to 200 ns. This range of delays is sufficient for the testing of patterns with 26 temporal values each having a 5 ns pulse width. Figure 5.3 illustrates the encoding of characters ‘A’ to ‘Z’ into the temporal space of this delay range. The delays of characters ‘A’ and ‘Z’ respectively represent the maximum and minimum delays needed. Character ‘A’ requires a delay of 200 ns while ‘Z’ needs 62 ns. All the 26 temporal values are fit into the temporal space with pulse width of 5.52 ns.

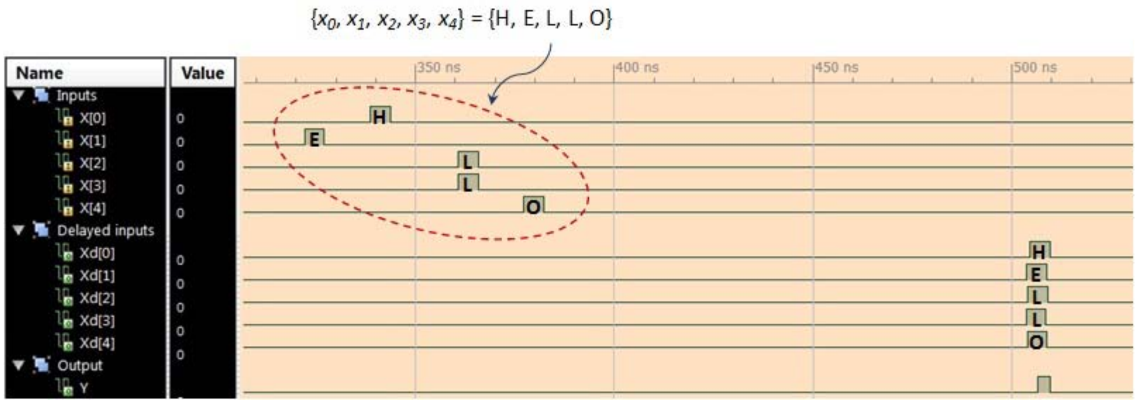


Figure 5.4: Simulation result of the detection of a manually-trained 5-element, 26-temporal value vector pattern

By knowing the value needed for delaying a particular temporal value, it is straight forward to manually train a pattern by configuring the delay switch settings of the delay lines. Figure 5.4 shows the simulation result of the detection of a manually trained pattern. The pattern represents a ‘HELLO’ word, *i.e.* {x_0, x_1, x_2, x_3, x_4} = {H, E, L, L, O}, and all the spikes were manually adapted to trigger coincidence.

5.2.3 Pattern recognition prototype

As presented in Section 3.4 of Chapter 3, a prototype was developed to demonstrate the feasibility of the implementation of a delay-based pattern recognition circuit. The prototype was implemented on a Xilinx Spartan-3E XC3S1600E device available on a Spartan-3E development board. The board comes with a 50MHz oscillator, a 64MByte SDRAM, 4 slide switches, 4 push-button switches, 8 surface-mount LEDs and three 6-pin expansion connectors that are useful for design testing [75]. It also has an on-board USB-based FPGA download interface.

The prototype comprises a pattern recognition array, pattern learning blocks and a pattern generator. The FPGA has a density of 33,192 logic cells. Figure 5.5 shows the physical layout of the prototype on the FPGA. The pattern recognition array consists of

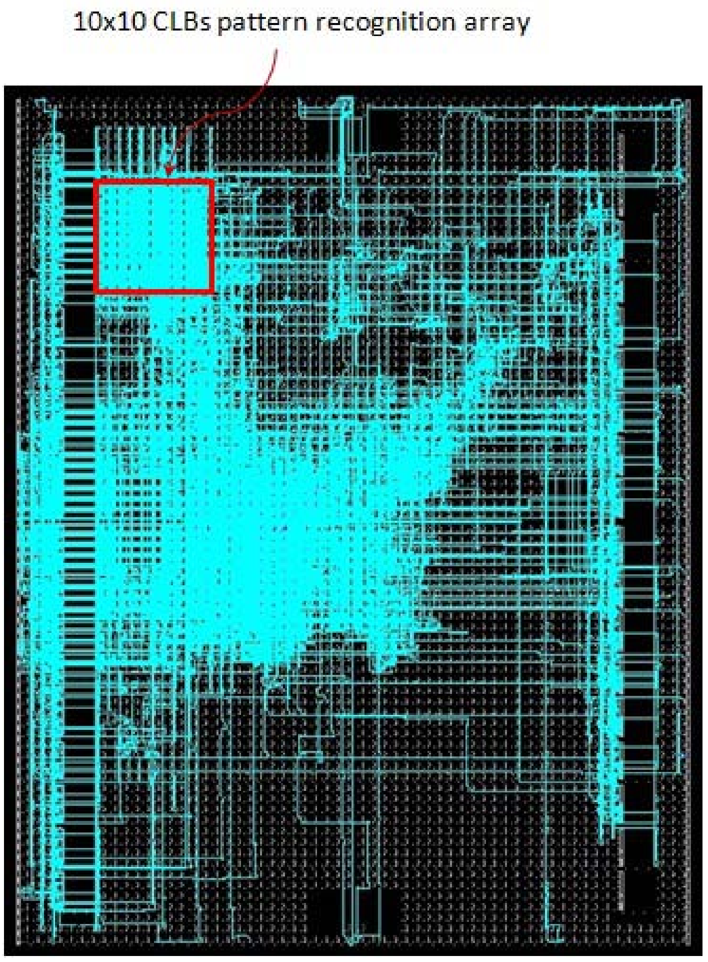


Figure 5.5: Physical layout of pattern recognition prototype on Xilinx Spartan-3E XC3S1600E FPGA

a 10×10 CLBs array of programmable delay lines and a coincidence detector. It utilizes 416/33,192 (1.3%) of the total logic cells available on the FPGA. The entire system including the pattern learning blocks and the pattern generator utilizes 4,494/33,192 (13.5%) of logic cells.

5.2.4 Detection of patterns with pattern recognition prototype

This section shows the results obtained from both simulation and hardware for the tests conducted on the pattern recognition prototype.

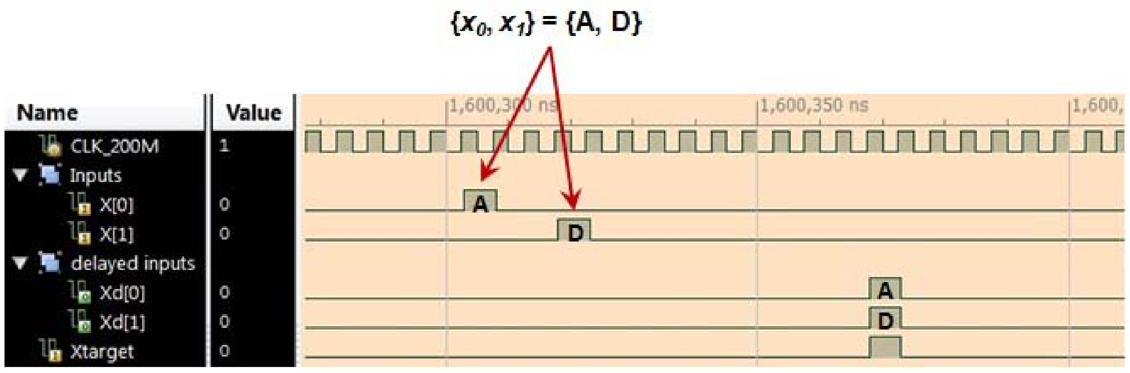


Figure 5.6: Simulation result for the detection of pattern {x_0, x_1} = {A, D} on pattern recognition prototype

The pattern recognition array is configured to train and detect patterns of a 2-element and 8-temporal value vector, {x_0, x_1}. Each x_i element represents a set of 8 characters {A, B, …, H}. x_0 and x_1 are fed in through the horizontal and vertical delay lines, respectively. The delay lines in each horizontal and vertical direction are cascaded to build a long delay line with sufficient delay to store the pattern.

Figure 5.6 shows the simulation result for the detection of a pattern {x_0, x_1} = {A, D} with a pulse width of 5 ns. In the initialization stage of the training process, each delay line was initialized to produce minimum delay. Each input spike was then delayed incrementally until it coincided with X_{target} spike, inserted after the last spike of the training pattern. Upon completion of training, the signals for the delayed version of x_0 and x_1 spikes, *i.e.* xd_0 and xd_1, were adapted to X_{target} and coincidence of the two spikes resulted. After training, when the same pattern {A, D} is presented to the circuit again, the output of the coincidence detector is asserted indicating the detection of the trained pattern.

The prototype was implemented and tested on a Spartan-3E FPGA for the learning and recognition of various 2-element, 8-temporal value vector patterns. Each delay line for the x_i spike was measured to provide a series of variable delays ranging from 30 ns to 71 ns, which was sufficient to encode patterns of an 8-temporal value vector with 5 ns pulse width. All patterns tested were successfully trained and detected by the prototype. The results from the hardware tests are shown in Figure 5.7. The signals were output through the 6-pin expansion connectors.

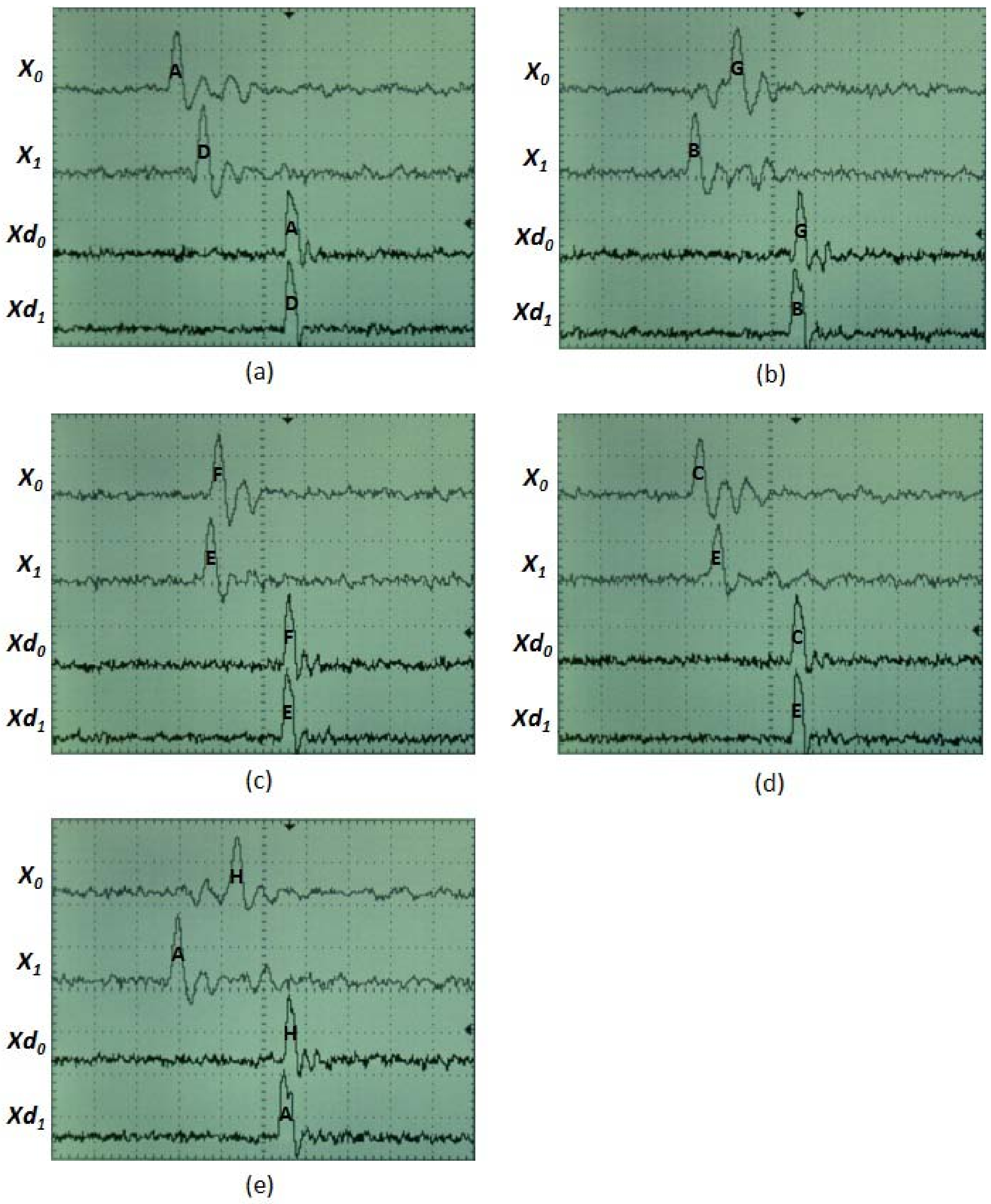


Figure 5.7 (a)-(e): Results of the detection of various 2-element, 8-temporal value vector patterns {A, D}, {G, B}, {F, E}, {C, E} and {H, A}, respectively, on Spartan-3E FPGA; voltage 2V/div, time 25ns/div

5.3 Auto-associative Memory

This section elaborates on the delays simulated on the auto-associative memory presented in Chapter 4, for both pre- and post-optimized versions of the system. The section also describes the pulse-narrowing effect experienced in the recall of patterns and presents results in relation to before and after solutions to the problem are implemented. The results for the hardware tests and resource utilization are also presented in this section.

5.3.1 Delay as a function of delay elements

The 4-input auto-associative memory with two context spike inputs per coincidence detector described in Chapter 4 was initially developed to process patterns with pulse width t_{spike} of 60 ns and context spike delay of no more than 3 unit intervals, *i.e.* 60 ns × 3 = 180 ns.

From simulation, it was measured that one delay element gives a delay of 6 ns when enabled. Each programmable delay line in the auto-associative memory consists of 32 delay elements to give sufficient context spike delay of up to 180 ns. The amount of delay produced by a delay line against the number of delay elements enabled was measured and collected from simulation. Four delay lines were randomly picked for the measurement of delay. The plot of the delay, *D*, against the number of delay elements enabled, *n*, is shown in Figure 5.8. From the plot, the delay is increasing linearly with the number of delay elements enabled, which is technically true since all the delay elements are identical. The plot can be represented by an equation, which is referred to as the *delay function*.

$$D = 6n + 29 \tag{5.1}$$

It was observed that when all the delay elements are disabled, there is an overhead of 29 ns propagation delay across the delay line and the feedback connection.

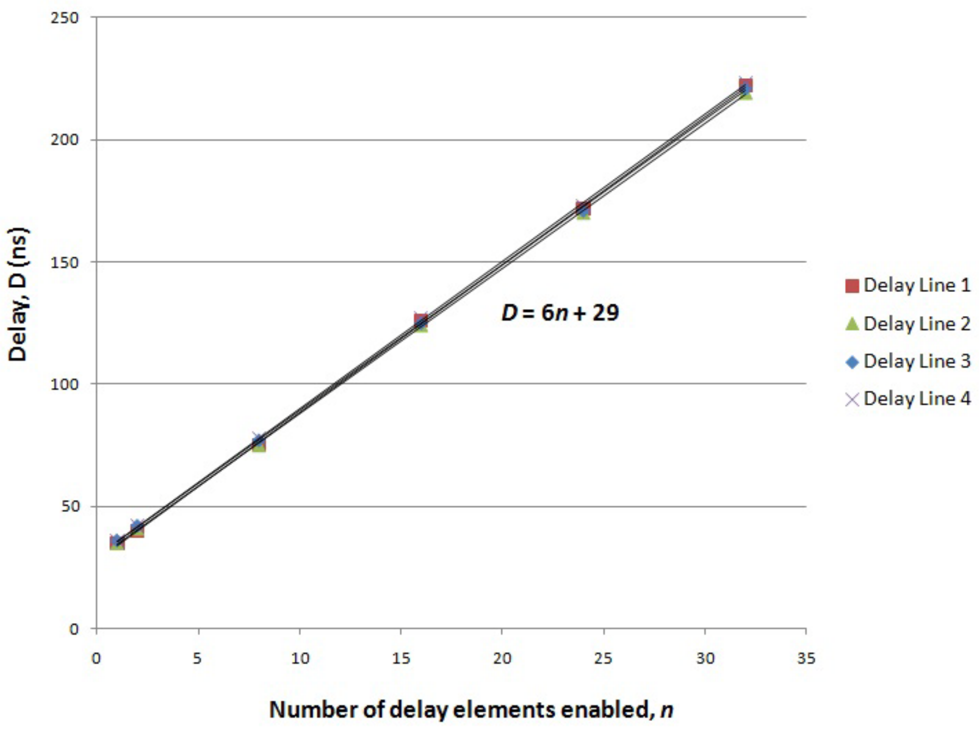


Figure 5.8: Plot of delay for 32-delay-element delay lines

In order to examine the performance of the memory in terms of speed, the design was later optimized for operating with smaller pulse width. The optimization was achieved by reducing the length of the programmable delay lines since lesser delay elements are needed for patterns with smaller pulse width. The design was optimized to process patterns with pulse width as small as the delay of a delay element, *i.e.* 6 ns. Each delay line was shortened from 32 delay elements to just 2 delay elements. Each delay element still provides a 6 ns delay when enabled while the overhead propagation delay was reduced significantly to just 6 ns. The optimized design processes patterns with pulse width of 6 ns and context spike delay of no more than 3 unit intervals. The 6 ns propagation delay is also being utilized as a delay for 1 unit interval. For example, for the case of context spike delay of 1 unit interval, *i.e.* 6 ns, both delay elements on the delay line are disabled to meet the delay requirement. The plot of the delay against the number of delay elements enabled for four randomly selected delay lines is shown in Figure 5.9. The delay function for the optimized memory is expressed in Equation 5.2.

$$D = 6n + 6 \tag{5.2}$$

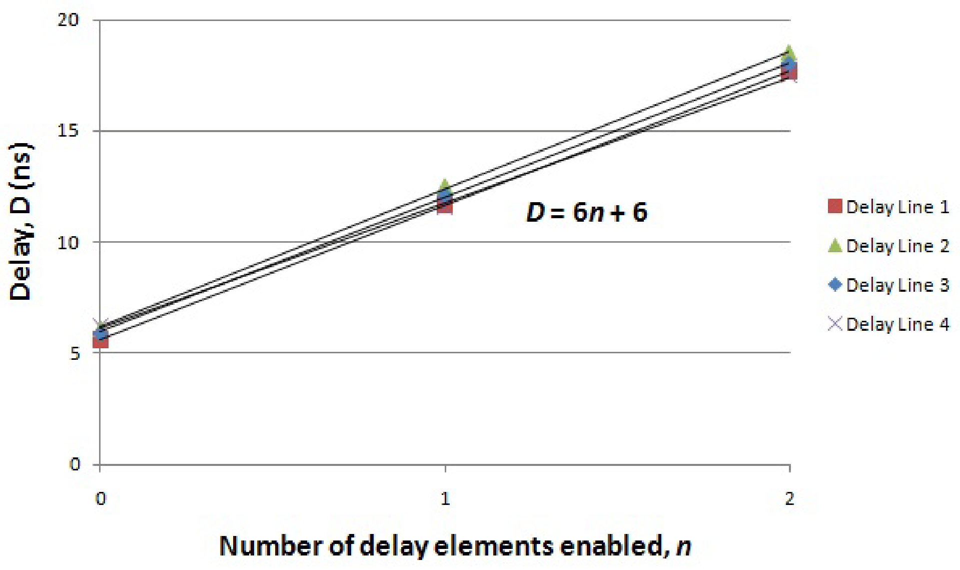


Figure 5.9: Plot of delay for 2-delay-element delay lines

5.3.2 Coincidence pulses and recall of patterns

A coincidence detector in the auto-associative memory is implemented using an ‘AND’ gate. Since two coincidence pulses arrive at a coincidence detector may have a small time difference, an AND operation between the two pulses produces an output pulse with smaller width. Each output pulse from a coincidence detector is fed back to the delay lines and the stored pattern is subsequently recalled for the next cycle. Due to the pulse-narrowing effect, the recurrently recalled pattern will eventually disappear over time, as shown in Figure 5.10.

With the two solutions described in Section 4.4.2 of Chapter 4, *i.e.* pulse-muxing coincidence detector and coincidence detector with constant-width pulse generator, the recall of a pattern can be sustained indefinitely. Figure 5.11 shows the simulation result for the recall of a pattern using the pulse-muxing coincidence detector. The output pulse of a coincidence detector is toggled between a smaller and a larger pulse over time and hence sustaining the recall of a pattern.

The solution with constant-width pulse generator provides a more efficient fix to the pulse-narrowing effect. Patterns recalled with this solution are always maintained with a constant pulse width and sustained indefinitely, as shown in Figure 5.12.

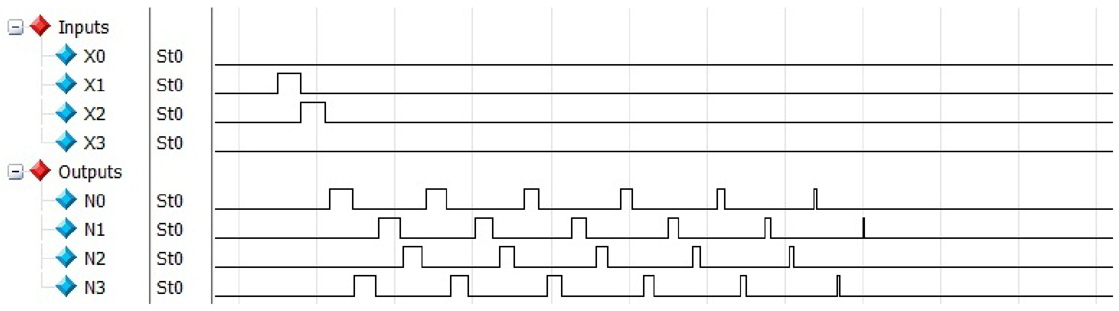


Figure 5.10: Pulse-narrowing effect on a recalled pattern

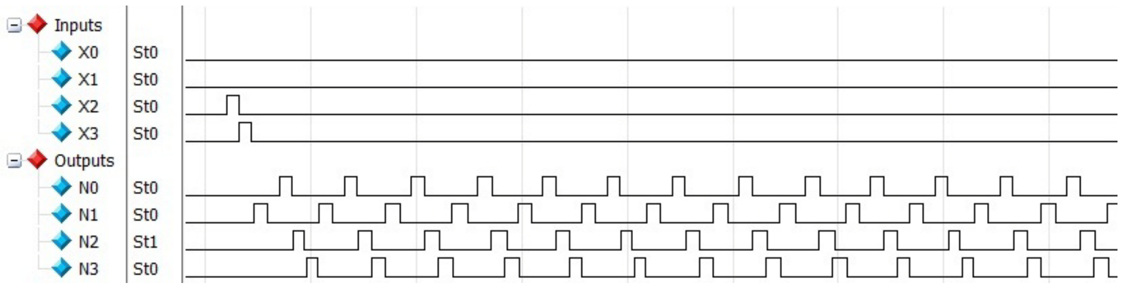


Figure 5.11: Recall of pattern {4, 2, 0, 1} with pulse-muxing coincidence detectors

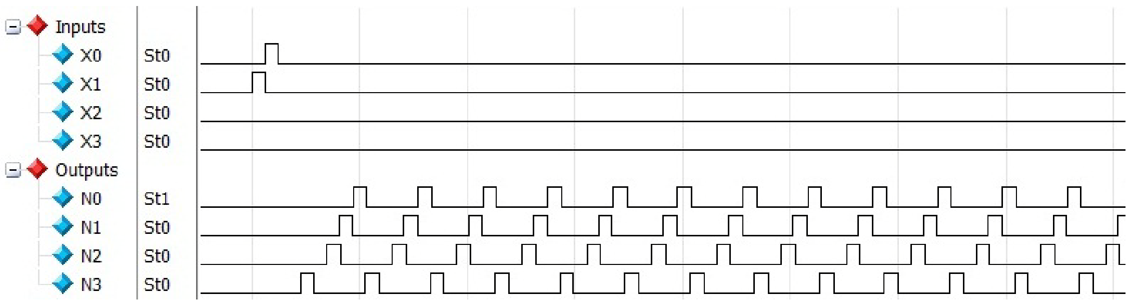


Figure 5.12: Recall of pattern {1, 0, 4, 2,} using coincidence detectors with constant-width pulse generator

5.3.3 Hardware test results

The auto-associative memory developed was simulated and tested on an Altera Cyclone II FPGA with twelve simple 4-input patterns. The settings of the pattern include spikes in random as well as diagonal directions, as illustrated in Figure 5.13. A perfect success rate of learning and recalling the patterns was achieved from the testing. The network successfully learned from the training patterns and recalled them recurrently.

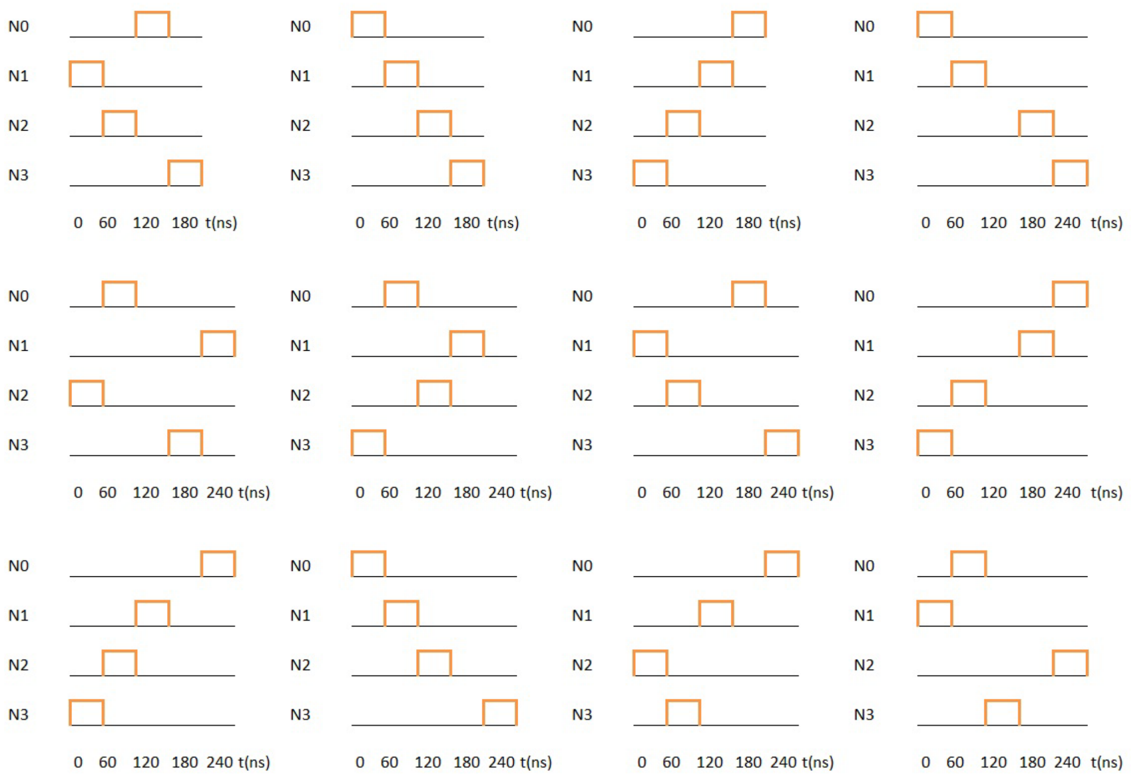


Figure 5.13: Random patterns tested on the auto-associative memory

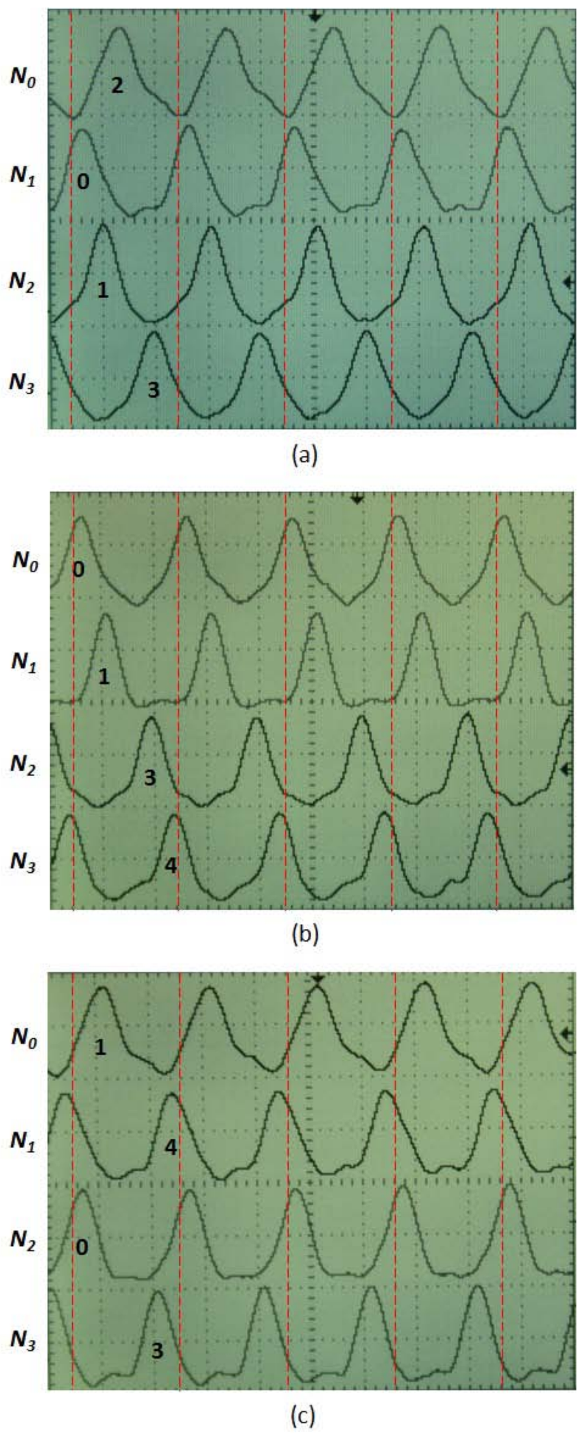


Figure 5.14 (a)-(c): Results of the recall of patterns {2, 0, 1, 3}, {0, 1, 3, 4} and {1, 4, 0, 3}, respectively, on Cyclone II FPGA; voltage 2V/div, time 10ns/div

Pattern	Output Neuron	Context Neurons	Delay (unit DEs)
{2, 0, 1, 3}	N_0	N_2	0
		N_1	1
	N_1	N_3	1
		N_0	2
	N_2	N_1	0
		N_3	2
	N_3	N_0	0
		N_2	1
{0, 1, 3, 4}	N_0	N_3	0
		N_2	1
	N_1	N_0	0
		N_3	1
	N_2	N_1	1
		N_0	2
	N_3	N_2	0
		N_1	2
{1, 4, 0, 3}	N_0	N_2	0
		N_1	1
	N_1	N_3	0
		N_0	2
	N_2	N_1	0
		N_3	1
	N_3	N_0	1
		N_2	2

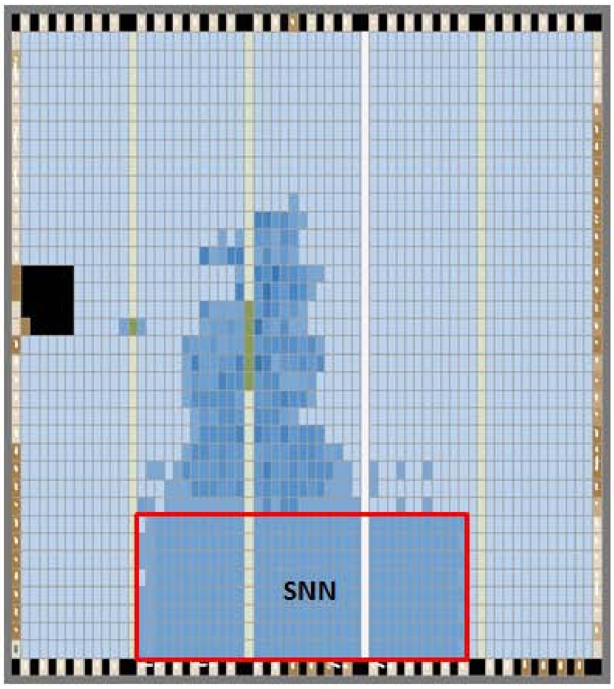
Table 5.3: Context neurons and delay settings of trained patterns

Figure 5.14 (a), (b) and (c) show the hardware test results for the recall of patterns {2, 0, 1, 3}, {0, 1, 3, 4} and {1, 4, 0, 3}, respectively. The pulse width for each spike of the patterns is 6 ns and the period for each pattern is 5 unit intervals, *i.e.* 30 ns. The settings of the context neurons and the respective number of delay elements required resulted from the training process for these patterns are shown in Table 5.3.

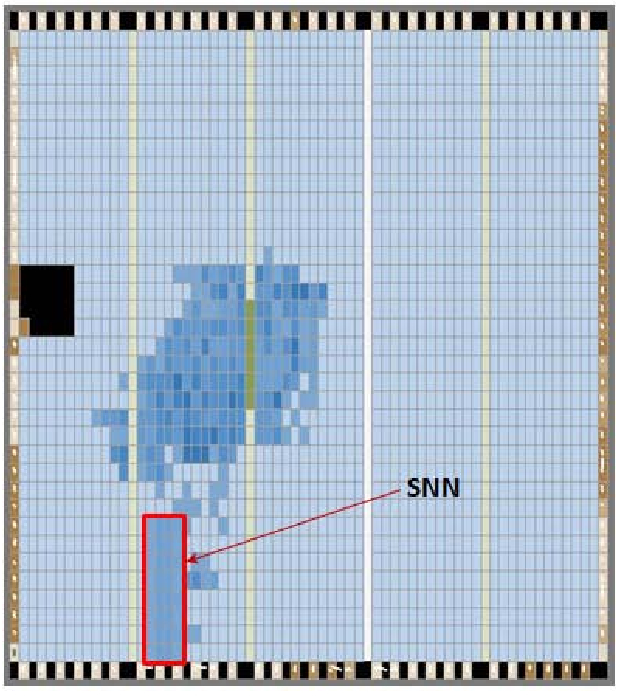
5.3.4 Hardware resource utilization

An Altera DE2 board was used for the testing of the auto-associative memory. The board comes with a Cyclone II EP2C35 family FPGA, which was used for the implementation of the memory. In addition to the FPGA, the board also equipped with other useful hardware features such as a 50MHz oscillator, an 8MByte SDRAM, 4 push-button switches, 18 toggle switches, 27 user LEDs and two 40-pin expansion connectors [4]. The FPGA is configured through an on-board USB download interface.

The Cyclone II device has a total of 33,216 logic elements. Figure 5.15 (a) and (b) show the physical layout of the auto-associative memory for the case of pre and post optimization, respectively. The pre-optimized system processes patterns with pulse width of 60 ns and the SNN consists of 32-delay-element delay lines. The entire system utilizes 22% (7,206/33,216) of the total logic elements available on the FPGA while the SNN takes up 13% (4,305/33,216). The size of the SNN is significantly reduced for the case of the optimized system where the length of the delay lines is reduced to 2 delay elements. The hardware utilization for the entire optimized system is 8% (2,764/33,216) of the total logic elements available while the SNN consumed less than 1% (328/33,216). The optimized system not only improved in size but also processes patterns at higher speed with pulse width of 6 ns.



(a)



(b)

Figure 5.15: Physical layout of auto-associative memory on Cyclone II EP2C35 FPGA with (a) 32-delay-element delay lines SNN; and (b) the optimized SNN with 2-delay-element delay lines

5.4 Summary

The results from the pattern recognition prototype and the auto-associative memory both demonstrate that an implementation of a time-delay pattern recognition circuit by exploiting the logic and interconnect resources in an FPGA is achievable, and the systems developed effectively perform simple learning and recognition tasks.

Table 5.4 compares our implementations with some of the existing pattern recognizers developed by other researchers, described previously in Section 2.5.3 of Chapter 2. Although the scale and functionality are fundamentally different across those systems, the comparison gives a high-level idea of the uniqueness and benefits of our models in terms of design approach and implementation. From the table, it can be seen that all the three existing pattern recognizers are implemented using conventional design approach with standard sequential and combinational logic, and clock; with processing speed in the range of 50-100 MHz. In contrast, the pattern processor (*i.e.* SNN) in our systems is implemented using a 'clock-free' design approach with LUTs and interconnect resources, which enables high speed processing of patterns with pulse width as small as 5 ns. The unique design approach not only gives advantage in processing speed but also allows a relatively compact implementation. In the same table, the Image Recognizer developed by Caron *et al.* is among the fastest with the system clocked at 100 MHz. The system however consumes a relatively large amount of logic resources with a total of ~32,242 logic cells. For systems with slower processing speed such as the Speed Recognizer proposed by Cassidy *et al.*, at a clock rate of 50 MHz, the logic utilization is also close to 20,000 logic cells. In comparison, the pattern processor in our systems that could process patterns with pulse width of 5 ns only utilizes 416 logic cells. The hardware consumption for the programmable delays is relatively low in both of our systems and that gives plenty of room for larger implementations aiming to store more patterns or patterns of larger scale. In addition, the implementation of learning module using soft processor allows flexibility for training algorithm modification with relatively low impact on hardware size.

Pattern Recognition Systems	**Main Functional Blocks**	**Hardware Utilization**	**Processing Speed**
Frequency Discriminator (Upegui *et al.*)	3-layer SNN: 30 neurons Neuron hardware: • control unit – FSM • weights – memory • learning modules	~3000 LCs (on Spartan-2 XC2S200)	Clocked at 54.4 MHz
Image Recognizer (Caron *et al.*)	SNN: 648 neurons Neuron hardware: • weights – BRAMs • synapses – adders • membrane model	~32424 LCs 126×36Kb BRAMs (on Virtex-5 XC5VSX50T)	Clocked at 100 MHz
Speech Recognizer (Cassidy *et al.*)	SNN: 32 neurons Neuron hardware: • 16-bit accumulator • dual-port memory	~19168 LCs 10×18Kb BRAMs (on Spartan-3 XC3S1500)	Clocked at 50 MHz
Delay-based Pattern Recognition Prototype	10×10 CLBs SNN Array • programmable delay lines • coincidence detector Learning module – MicroBlaze processor	SNN: 416 LCs System: 4494 LCs (on Spartan-3E XC3S1600E)	SNN: clock-free Pulse width: 5ns
Auto-associative Memory	1-layer SNN: 4 neurons • programmable delay lines • coincidence detectors Learning module – Nios II processor	SNN: 328 LEs System: 7206 LEs (on Cyclone II EP2C35)	SNN: clock-free Pulse width: 6ns

Table 5.4: Comparison of example pattern recognizers with our systems; FSM – finite state machine; BRAMs – block RAMs; LCs – logic cells; LEs – logic elements

CHAPTER 6

CONCLUSION

6.1 Summary and Key Insights

In this thesis, we have developed two pattern recognition systems that process spatiotemporal spike patterns on field-programmable gate arrays (FPGAs).

Borrowing the idea from spiking neural networks (SNNs), which suggests the brain possibly processes information based on action potential timing; the systems were developed with a similar principle: perform pattern learning and recognition tasks through the use of time delays. Patterns being processed are in the form of spatiotemporal spike sequences, which represent the spiking activity of neurons over time. Information is encoded in spike patterns via time delays. Both systems developed use a time-delay network consisting of programmable delay lines and coincidence detectors to process patterns.

The first system, *i.e.* the Delay-based Pattern Recognition Prototype in Chapter 3, learns spatiotemporal spike patterns and performs recognition of learned patterns. Learning is achieved through adaptation of delay lines to spike timing of patterns while recognition is attained via coincidence detection of delayed spikes. The design of a programmable delay line is explored with the exploitation of FPGA architecture and interconnect scheme. The delay lines are extendable for patterns with larger delay requirements and can be organized into a compact array for better area efficiency. The test results demonstrate that a hardware realization of a time-delay pattern recognizer is viable, and that FPGA logic and interconnect resources are effective for implementing programmable delays. The system developed has a 10×10 CLBs array of delay lines and a coincidence detector, and consumes only 416 logic cells. The pattern recognizer demonstrates processing of spike patterns with 2 elements and 8 temporal values, and 5 ns pulse width.

The second system, *i.e.* the Auto-associative Memory in Chapter 4, performs memory tasks of storing and recalling spatiotemporal spike patterns. The system operates as an auto-associative memory where a complete pattern is retrieved via partial presentation of the original copy. The memory learns a pattern through identification of context spikes and adaptation of delay lines to the timing of context spikes. Memory recall is achieved through coincidence of delayed context spikes. The system uses a closed feedback SNN that drives recurrent recall of a stored pattern. The delay lines are built from small units of delay elements and the delay produced by each delay line exhibits a linear relationship with the number of active delay elements. The SNN utilizes only 328 logic elements for storing a 4-input pattern with 6 ns pulse width. The compact size of the SNN is beneficial for implementation of memory blocks of larger storage capacity through replication of the SNN.

Unlike existing pattern recognition systems which are usually implemented using conventional design practices and standard circuitries; we explored and adopted a ‘clock-free’ design approach in the implementation of our models. The pattern processing unit in our systems uses only combinational logic and interconnect resources. This unique design approach allows fast processing of patterns with pulse width as small as 5 ns. The approach may potentially provide inspiration to the design of ultra-high performance digital processors for applications beyond pattern recognition.

6.2 Future Directions

Having demonstrated the feasibility of realizing a delay-based pattern recognizer, for future work, it would be interesting to explore a ‘user-interactive’ pattern recognizer with external inputs from the environment. Given the fast-processing capability of the models developed in this thesis, it would be beneficial to extend the research into developing a pattern recognizer that could learn and recognize patterns introduced directly from the environment, and perform specific user tasks.

Patterns in the real world are complex. A second possible area of study is to explore a more sophisticated model for complex pattern recognition tasks. A complex pattern such as a speech can be broken down into levels of sub-patterns such as sentences, words and characters. One possible approach is to model a pattern recognizer in a "hierarchical manner" where computation and recognition of higher-level complex patterns could be made via processing of sub-patterns from lower levels.

A third interesting area to look at is to exploit the present architecture for working with temporal patterns of longer duration, with minimum increase in hardware cost. A possible method is to divide a long temporal pattern into shorter "time blocks" for processing. This could possibly be achieved through a more sophisticated software algorithm and some hardware memory resources.

A fourth consideration is to make the present systems to be more automated and easily accustomed to target patterns of different sizes and timing requirements. For example, in the present systems, the length of delay lines for a target pattern is manually determined via simulation. It would be beneficial if the systems could automatically determine the length of delay lines when presented with a target pattern.

Another consideration is to assess the robustness of these delay-based systems against temperature variation. Since delays may vary across different temperatures, patterns trained at one temperature may not be detected at another temperature. Hence, it would be beneficial to incorporate temperature variation factors into pattern learning so that delays trained are tolerable across different temperatures.

BIBLIOGRAPHY

[1] M. Abeles. *Corticonics: Neural Circuits of the Cerebral Cortex*. Cambridge University Press, 1991.

[2] J. N. Allen, H. S. Abdel-Aty-Zohdy and R. L. Ewing. Plasticity recurrent spiking neural networks for olfactory pattern recognition. In *48th Midwest Symposium on Circuits and Systems*, volume 2, pages 1741-1744, 2005.

[3] Altera Corp. *Cyclone II Device Handbook*, volume 1, 2008.

[4] Altera Corp. *DE2 Development and Education Board User Manual*, version 1.4, 2006.

[5] A. Armato, E. Nardini, A. Lanata, G. Valenza, C. Mancuso, E. P. Scilingo and D. De Rossi. An FPGA based arrhythmia recognition system for wearable applications. In *Ninth International Conference on Intelligent Systems Design and Applications 2009*, pages 660-664, 2009.

[6] L. Bako, S. T. Brassai, I. Szkely and M. A. Baczo. Hardware implementation of delay-coded spiking-RBF neural network for unsupervised clustering. In *11th International Conference on Optimization of Electrical and Electronic Equipment, 2008*, pages 51-56, 2008.

[7] S. Bellis, K. M. Razeeb, C. Saha, K. Delaney, C. O'Mathuna, A. Pounds-Cornish, G. de Souza, M. Colley, H. Hagras, G. Clarke, V. Callaghan, C. Argyropoulos, C. Karistianos and G. Nikiforidis. FPGA implementation of spiking neural networks - an initial step towards building tangible collaborative autonomous

agents. *Proceedings of the IEEE International Conference on Field-Programmable Technology 2004*, pages 449-452, 2004.

[8] G. Bi and M. Poo. Distributed synaptic modification in neural networks induced by patterned stimulation. *Nature*, volume 401, pages 792-796, 1999.

[9] G. Bi and M. Poo. Synaptic Modification by correlated activity: Hebb's postulate revisited. *Annual Review of Neuroscience*, volume 24, pages 139-166, 2001.

[10] J. J. Blake, L. P. Maguire, T. M. McGinnity, B. Roche and L. J. McDaid. The implementation of fuzzy systems, neural networks and fuzzy neural networks using FPGAs. *Information Sciences*, volume 112, pages 151-168, 1998.

[11] C. D. Brody and J. J. Hopfield. Simple networks for spike-timing-based computation, with application to olfactory processing. *Neuron*, volume 37, pages 843-852, 2003.

[12] L. C. Caron, F. Mailhot and J. Rouat. FPGA implementation of a spiking neural network for pattern matching. In *IEEE International Symposium on Circuits and Systems 2011*, pages 649-652, 2011.

[13] G. A. Carpenter. Neural network models for pattern recognition and associative memory. *Neural Networks*, volume 2, pages 243-257, 1989.

[14] A. Cassidy, S. Denham, P. Kanold and A. Andreou. FPGA based silicon spiking neural array. In *IEEE Biomedical Circuits and Systems Conference 2007*, pages 75-78, 2007.

[15] K. Cheung, S. R. Schultz and P. H. W. Leong. A parallel spiking neural network simulator. *Proceedings of the IEEE International Conference on Field-Programmable Technology 2009*, pages 247-254, 2009.

[16] J. Cho, S. Mirzaei, J. Oberg and R. Kastner. FPGA-based face detection system using Haar classifiers. *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 103-112, 2009.

[17] R. J. Duro and J. S. Reyes. Discrete-time backpropagation for training synaptic delay-based artificial neural networks. *IEEE Transactions on Neural Networks*, volume 10, pages 779-789, 1999.

[18] C. W. Eurich, K. Pawelzik, U. Ernst, A. Thiel, J. D. Cowan and J. G. Milton. Delay adaptation in the nervous system. *Neurocomputing*, volume 32-33, pages 741-748, 2000.

[19] W. Gerstner. Time structure of the activity in neural network models. *Physical Review E*, volume 51, page 738, 1995.

[20] B. Glackin, L. P. Maguire and T. M. McGinnity. Intrinsic and extrinsic implementation of a bio-inspired hardware system. *Information Sciences*, volume 161, pages 1-19, 2004.

[21] A. Gupta and L. N. Long. Character recognition using spiking neural networks. In *International Joint Conference on Neural Networks 2007*, pages 53-58, 2007.

[22] J. B. Hampshire, II and A. H. Waibel. A novel objective function for improved phoneme recognition using time-delay neural networks. *IEEE Transactions on Neural Networks*, volume 1, pages 216-228, 1990.

[23] E. Heinrich, R. Joost, M. Luder and R. Salomon. Precise indoor localization with low-cost field-programmable gate arrays. In *IEEE Workshop on Merging Fields of Computational Intelligence and Sensor Technology 2011*, pages 23-28, 2011.

[24] E. Heinrich, R. Joost and R. Salomon. A digital implementation of the nucleus laminaris. In *International Joint Conference on Neural Networks 2011*, pages 1461-1465, 2011.

[25] E. Heinrich, R. Joost and R. Salomon. Learning from the barn owl auditory system: A bio-inspired localization hardware architecture. In *IEEE Congress on Evolutionary Computation 2011*, pages 216-221, 2011.

[26] J. J. Hopfield. Pattern recognition computation using action potential timing for stimulus representation. *Nature*, volume 376, pages 33-36, 1995.

[27] E. M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, volume 14, pages 1569-1572, 2003.

[28] E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, volume 15, pages 1063-1070, 2004.

[29] E. M. Izhikevich. Polychronization: Computation with spikes. *Neural Computation*, volume 18, pages 245-282, 2006.

[30] A. K. Jain, R. P. W. Duin and M. Jianchang. Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* volume 22, pages 4-37, 2000.

[31] C. T. Jin, P. L. Rolandi and P. H. W. Leong. Non-volatile programmable pulse computation cell. *Electronics Letters*, volume 35, pages 1413-1414, 1999.

[32] R. Kempter, W. Gerstner and J. L. van Hemmen. Hebbian learning and spiking neurons. *Physical Review E*, volume 59, page 4498, 1999.

[33] W. M. Kistler, W. Gerstner and J. L. van Hemmen. Reduction of the Hodgkin-Huxley equations to a single-variable threshold model. *Neural Computation,* volume 9, pages 1015-1015, 1997.

[34] T. Koickal, A. Hamilton, S. Tan, J. Covington, J. Gardner and T. Pearce. Analog VLSI circuit implementation of an adaptive neuromorphic olfaction chip. *IEEE Transactions on Circuits and Systems I: Regular Papers*, volume 54, pages 60-73, 2007.

[35] P. König, A. K. Engel and W. Singer. Integrator or coincidence detector? The role of the cortical neuron revisited. *Trends in Neurosciences*, volume 19, pages 130-137, 1996.

[36] M. Kugler, V. Benso, S. Kuroyanagi and A. Iwata. A novel approach for hardware based sound classification. In M. Köppen, N. Kasabov and G. Coghill, editors, *Advances in Neuro-Information Processing*, volume 5507, pages 859-866. Springer Berlin / Heidelberg, 2009.

[37] M. Kugler, K. Iwasa, V. Benso, S. Kuroyanagi and A. Iwata. A complete hardware implementation of an integrated sound localization and classification system based on spiking neural networks. In M. Ishikawa, K. Doya, H. Miyamoto and T. Yamakawa, editors, *Neural Information Processing*, volume 4985, pages 577-587. Springer Berlin / Heidelberg, 2008.

[38] K. J. Lang, A. H. Waibel and G. E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, volume 3, pages 23-43, 1990.

[39] M. P. Leong, C. T. Jin and P. H. W. Leong. Parameterized module generator for an FPGA-based electronic cochlea. In *9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines 2001*, pages 21-30, 2001.

[40] T. Lindblad and J. M. Kinser. *Image Processing Using Pulse-Coupled Neural Networks*, Springer-Verlag NewYork Inc., Secaucus, NJ, USA, 2005.

[41] J. Liu and D. Liang. A survey of FPGA-based hardware implementation of ANNs. In *International Conference on Neural Networks and Brain 2005*, pages 915-918, 2005.

[42] W. Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, pages 1659-1671, 1997.

[43] W. Maass. Fast sigmoidal networks via spiking neurons. *Neural Computation*, volume 9, pages 279-304, 1997.

[44] W. Maass. On the relevance of time in neural computation and learning. In M. Li and A. Maruoka, editors, *Algorithmic Learning Theory*, volume 1316, pages 364-384. Springer Berlin / Heidelberg, 1997.

[45] W. Maass, T. Natschläger and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, volume 14, pages 2531-2560, 2002.

[46] W. Maass and M. Schmitt. On the complexity of learning for a spiking neuron (extended abstract). *Proceedings of the Tenth Annual Conference on Computational Learning Theory*, pages 54-61, Nashville, Tennessee, United States, 1997.

[47] W. Maass and M. Schmitt. On the complexity of learning for spiking neurons with temporal coding. *Information and Computation*, volume 153, pages 26-46, 1999.

[48] L. P. Maguire, T. M. McGinnity, B. Glackin, A. Ghani, A. Belatreche and J. Harkin. Challenges for large-scale implementations of spiking neural networks on FPGAs. *Neurocomputing*, volume 71, pages 13-29, 2007.

[49] H. Markram, J. Lubke, M. Frotscher and B. Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, volume 275, pages 213-215, January 10, 1997.

[50] S. Maya, R. Reynoso, C. Torres and M. Arias-Estrada. Compact spiking neural network implementation in FPGA. In R. Hartenstein and H. Grünbacher, editors, *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, volume 1896, pages 270-276. Springer Berlin / Heidelberg, 2000.

[51] R. McCready. Real-time face detection on a configurable hardware system. In R. Hartenstein and H. Grünbacher, editors, *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, volume 1896, pages 157-162. Springer Berlin / Heidelberg, 2000.

[52] D. Meunier and H. Paugam-Moisy. Evolutionary supervision of a dynamical neural network allows learning with on-going weights. *Proceedings of the IEEE International Joint Conference on Neural Networks 2005*, volume 3, pages 1493-1498, 2005.

[53] J. Misra and I. Saha. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, doi:10.1016/j.neucom.2010.03.021, 2010.

[54] V. Nair, P. Laprise and J. J. Clark. An FPGA-based people detection system. *EURASIP Journal on Applied Signal Processing*, volume 2005, pages 1047-1061, 2005.

[55] H. Paugam-Moisy. *Spiking Neuron Networks – A Survey*. IDIAP Research Institute, Martigny, Switzerland, 2006.

[56] H. Paugam-Moisy, R. Martinez and S. Bengio. Delay learning and polychronization for reservoir computing. *Neurocomputing*, volume 71, pages 1143-1158, 2008.

[57] M. Pearson, M. Nibouche, A. G. Pipe, C. Melhuish, I. Gilhespy, B. Mitchison, K. Gurney, T. Prescott and P. Redgrave. A biologically inspired FPGA based implementation of a tactile sensory system for object recognition and texture discrimination. In *International Conference on Field Programmable Logic and Applications 2006*, pages 1-4, 2006.

[58] J. Pfister, T. Toyoizumi, D. Barber and W. Gerstner. Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Computation*, volume 18, pages 1318-1348, 2006.

[59] M. Saldana, L. Shannon and P. Chow. The routability of multiprocessor network topologies in FPGAs. *Proceedings of the 2006 International Workshop on System-level Interconnect Prediction*, pages 49-56, Munich, Germany, 2006.

[60] M. Saldana, L. Shannon, J. S. Yue, S. Bian, J. Craig and P. Chow. Routability of network topologies in FPGAs. *IEEE Transactions on Very Large Scale Integration Systems*, volume 15, pages 948-951, 2007.

[61] M. Schæfer, T. Schoenauer, C. Wolff, G. Hartmann, H. Klar and U. Rückert. Simulation of spiking neural networks – architectures and implementations. *Neurocomputing*, volume 48, pages 647-679, 2002.

[62] J. Schemmel, A. Grubl, K. Meier and E. Mueller. Implementing synaptic plasticity in a VLSI spiking neural network model. In *International Joint Conference on Neural Networks 2006*, pages 1-6, 2006.

[63] M. Schmitt. On computing Boolean functions by a spiking neuron. *Annals of Mathematics and Artificial Intelligence*, volume 24, pages 181-191, 1998.

[64] B. Schrauwen, M. D'Haene, D. Verstraeten and J. V. Campenhout. Compact hardware liquid state machines on FPGA for real-time speech recognition. *Neural Networks*, volume 21, pages 511-523, 2008.

[65] J. Sohn, B. Zhang and B. Kaang. Temporal pattern recognition using a spiking neural network with delays. In *International Joint Conference on Neural Networks 1999*, volume 4, pages 2590-2593, 1999.

[66] A. Tavaragiri, J. Couch and P. Athanas. Exploration of FPGA interconnect for the design of unconventional antennas. *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 219-226, Monterey, CA, USA, 2011.

[67] A. Upegui, C. A. Peña-Reyes and E. Sanchez. An FPGA platform for on-line topology exploration of spiking neural networks. *Microprocessors and Microsystems*, volume 29, pages 211-223, 2005.

[68] A. van Schaik. Building blocks for electronic spiking neural networks. *Neural Networks*, volume 14, pages 617-628, 2001.

[69] K. Van Sickle and H. Abdel-Aty-Zohdy. A reconfigurable spiking neural network digital ASIC simulation and implementation. *Proceedings of the IEEE National Aerospace & Electronics Conference 2009*, pages 275-280, 2009.

[70] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano and K. J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, volume 37, pages 328-339, 1989.

[71] B. Webb and T. Scutt. A simple latency-dependent spiking-neuron model of cricket phonotaxis. *Biological Cybernetics*, volume 82, pages 247-269, 2000.

[72] S. A. Wills. *Computation with Spiking Neurons*. PhD thesis, University of Cambridge, UK, 2004.

[73] S. Wysoski, L. Benuskova and N. Kasabov. On-line learning with structural adaptation in a network of spiking neurons for visual pattern recognition. In S. Kollias, A. Stafylopatis, W. Duch and E. Oja, editors, *Artificial Neural Networks – ICANN 2006*, volume 4131, pages 61-70. Springer Berlin / Heidelberg, 2006.

[74] S. G. Wysoski, L. Benuskova and N. Kasabov. Fast and adaptive network of spiking neurons for multi-view visual pattern recognition. *Neurocomputing*, volume 71, pages 2563-2575, 2008.

[75] Xilinx Inc. *MicroBlaze Development Kit Spartan-3E 1600E Edition User Guide*, version 1.1, 2007.

[76] Xilinx Inc. *Spartan-3 Generation FPGA User Guide*, version 1.7, 2010.

PUBLICATIONS

- C. H. Ang, C. Jin, A. van Schaik and P. H. W. Leong. Spiking neural network-based auto-associative memory using FPGA interconnect delays. *Proceedings of the IEEE International Conference on Field-Programmable Technology 2011*, pages 1-4, 2011.
- C. H. Ang, A. L. McEwan, A. van Schaik, C. Jin and P. H. W. Leong. FPGA implementation of biologically-inspired auto-associative memory. *Electronics Letters*, volume 48, pages 148-149, 2012.