12-2015

Implementation of a Neuromorphic Development Platform with DANNA

Jason Yen-Shen Chan
University of Tennessee - Knoxville, jchan5@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Jason Yen-Shen Chan entitled "Implementation of a Neuromorphic Development Platform with DANNA." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Mark Dean, Major Professor

We have read this thesis and recommend its acceptance:

John D. Birdwell, Garrett Rose

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Implementation of a Neuromorphic Development Platform with DANNA

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Jason Yen-Shen Chan

December 2015

Abstract

Neuromorphic computing is the use of artificial neural networks to address complex and/or non-traditional computational problems. The specialized computing field has been growing in interest during the past few years. Specialized hardware that function as neural networks can be utilized to solve a broad set of problems less suited for traditional computing architectures, such as pattern classification, anomaly detection, and adaptive controls. However, these hardware platforms have neural network structures that are static, being limited to only perform a specific application, and cannot be used for other tasks. In this paper, the feasibility of a development platform utilizing a dynamic artificial neural network for researchers is discussed.

Table of Contents

List of Figures

Chapter 1

Introduction

The field of neuromorphic computing has grown in prevalence over the years. To date, many research groups have tried to implement artificial neural networks using both hardware and software combined. There are a handful of hardware platforms hosting artificial neural networks well documented by research groups. However, these platforms hold static networks, being configured in a specific manner, making them usable for only specific applications.

Neuroscience-Inspired Dynamic Architecture (NIDA) was presented as an alternate software approach to artificial neural networks. With a focus on making simple networks dynamic and adaptive, they allow for large network structures with comparable capabilities against other approaches. The architecture utilizes simple neurons and synapses to create large array structures suited to perform tasks and solve problems suited for neuromorphic computing systems.

A hardware platform that utilizes the NIDA computing architecture was developed. Dynamically Adaptive Neural Network Arrays (DANNA) utilizes the same architecture and principles as NIDA networks. DANNAs are 2-D network structures better suited for implementation on a hardware platform. DANNA elements, which can be programmed as a neuron or a synapse, are tiled together to form an array structure, and they can be programmed to accomplish the same tasks as NIDA.

Previous work has been done with DANNA networks showcasing their functionality on a Field-Programmable Gate Array (FPGA) [21].

In an effort to apply DANNA to other applications, a development kit (DDK) was proposed. Comprised of a single board computer, a communications interface, and an FPGA module, it can contain a programmable DANNA array DANNA as well as the necessary hardware to operate and monitor the DANNA networks. This standalone system contains all of the necessary hardware needed to operate DANNA networks, so other researchers may utilize it for their own applications.

The DANNA implementation used for the kit improves upon the problems presented in the original implementation while retaining the same architecture to create neural networks. It now features an increased number of connections per element as well as a new monitoring capability for elements and a new communications interface. Resolving issues found in the original design, large array structures have been implemented in the DANNA kit. Testing the new design confirms its behavior and led to other enhancements. While the kit has been confirmed to have worked, more work can be performed on its components allowing for larger array structures to be built, interfaced, and monitored.

The following sections of this report detail the work done with DANNA works that culminate in the creation of the DANNA development kit.

Chapter 2

Related Work

Neuroscience-inspired computation is a rapidly growing field in the realm of computing. With technology advancing, more and more research has been conducted in the field with the goal of mimicking a human brain to perform advanced computations unsuited for traditional computing. However, creating an artificial neural network (ANNs) for computation proves to be a challenge. They tend to be constructed with dynamic element with predefined operators, such as unit delay elements, and are trained with machine learning (ML) methods whereas biological neural networks are inherently parallel and function as distributed systems that incorporate state information as memory and dynamics for behavior based on prior stimuli [20].

Currently, there have been many implementations of ANNs by many groups in both software, through modeling and simulation, and hardware, through analog and digital constructs [20]. Each group has their own unique method of approaching the problem posed by neuromorphic computing, with some groups utilizing existing architecture and others developing unique architectures to solve the same problems. Some notable hardware examples include Neurogrid by Stanford University [17], SpiNNaker by the University of Manchester [24], the TrueNorth Computer by IBM [6], and BrainScaleS by the Human Brain Project [25].

Neurogrid, by Stanford University, is a neuromorphic multi-chip system for simulating large-scale neural models in real-time [17]. It is capable of modeling complex neuron structures and synapse connections on each of its custom chips, referred to as Neurocores. Using a combination of both analog and digital circuits, Neurogrid succeeds in implementing common elements found in neural structures allowing for versatility. Specialized software works with the Neurogrid platform to map neural networks onto the hardware through the use of a USB controller, and a CPLD acts as an interface between the controller and the Neurocores. The current Neurogrid hardware platform contains 16 Neurocores, each capable of simulating a 256×256 array structure, allowing for very complex networks all on a 6.5×7.5 in^2 board. An image of the platform can be shown below in figure 2.1.

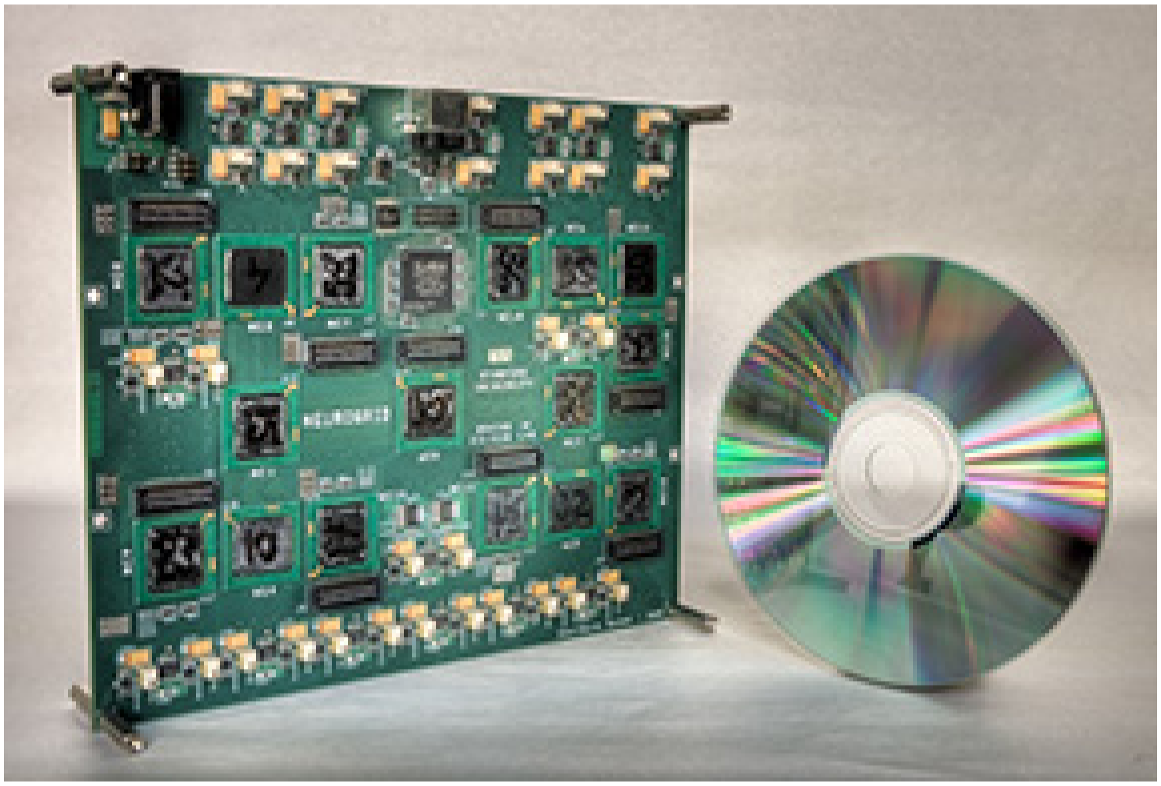


Figure 2.1: Neurogrid Board [18]

SpiNNaker is a biologically inspired, massively parallel computing engine designed to model and simulate spiking neural networks [24]. It is designed as an array of nodes, with each node containing a unique chip multiprocessor (CMP) die and an SDRAM die, and each CMP holds 18 ARM968 processors, each capable of simulating 1000 spiking neurons. The processors contain separate instruction and data memory as well as both off-chip and on-chip RAM all interconnected through a network which also communicates with an on-chip router which sends packets off-chip to other processors. Processor communication is based off of a neurobiological-inspired multicast infrastructure, which uses a packet-switched network to emulate the high connectivity found in biological systems with a router that repeats packets to several destinations. A die plot of the SpiNNaker CMP is shown below in figure 2.2. The fourth generation of the SpiNNaker system contains 864 processors in 48 chips all on one board which have been shown to simulate large networks containing a million neurons and a billion synapses, and it is hoped that multiple boards can be interconnected together to form larger arrays.

TrueNorth is a brain-inspired neurosynaptic processor with its architecture inspired by the human brain and the cortical column, a cluster of densely interconnected neurons which form the canonical unit of the brain [16]. It is composed of 4096 neurosynaptic cores tiled in a 2-D array structure, which contain an aggregate of 1 million neurons and 256 million synapses. Each core is composed of tightly coupled neurons for computation, synapses for memory, and axons and dendrites for communication. The processor is driven by a parallel, event-driven architecture which implement spiking neurons by programming neurons and the connectivity between them. The TrueNorth processors are designed in such a way that can be tiled into 2-D array structures, and this allows for large networks of interconnected neurons built from the neurosynaptic cores.

The Human Brain Project by the European Commission conducted several projects in an effort to develop its own neuromorphic computing platform. One project, BrainScaleS, attempts to develop a large-scale artificial neural system

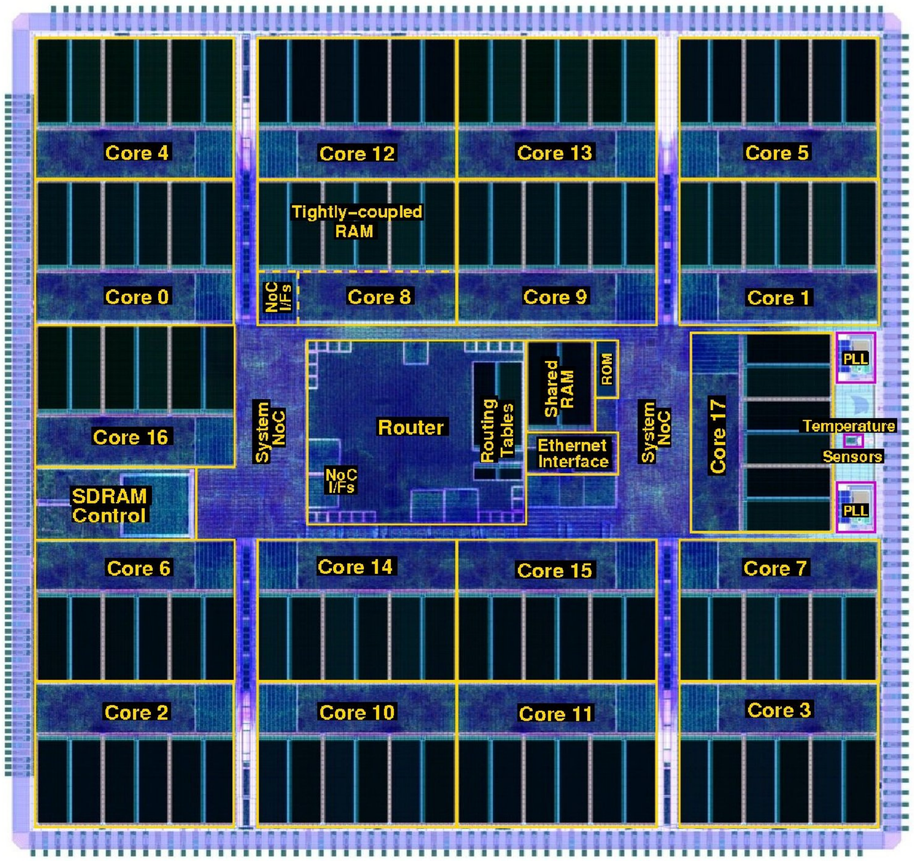


Figure 2.2: SpiNNaker CMP Die [9]

on a physical platform that implements many systems found in contemporary computational neuroscience [25]. The platform is comprised of specialized wafer-integrated VLSI chips, all interconnected together with each one containing 4×10^7 analog synapses and up to 180,000 neurons. External configuration and monitoring of the system is carried through a dedicated communications network through the use of application specific integrated circuits (ASICs) and FPGAs [30]. Testing shows that the system is capable of accelerated emulation of spiking neural networks.

In spite of these technological achievements in the realm of neuromorphic computing, there are some downsides to the hardware. Most hardware implementations are static structures, by which their neural networks have their components fixed at specific locations in hardware [20]. This fixed placement of neural components limits the network to work for only specific applications. Also, the hardware lacks the ability flexibly adapt to changing conditions and inputs. Furthermore, many hardware implementations are limited by the use of components found in von Neumann based systems such as RAM, buses, processors, and sequential instruction processing. This is most evident in the SpiNNaker project through it use of ARM cores [24]. All of the drawbacks of current platforms have inspired development of a dynamic, adaptable neuromorphic platform with an approach of creating complex neural networks at the expense of less complex neural elements. The result is a simpler platform capable of providing similar functionality when compared to other complex hardware platforms.

Neuroscience-Inspired Dynamic Architecture (NIDA), is a biologically-inspired architecture used to develop neuromorphic networks [28]. NIDA networks are unique in that information is distributed throughout the network in neurons and synapses via charge and delay, respectively, and that the network functions in a discrete event framework represented by spiking behavior.

NIDA networks are comprised of neurons and synapses [28]. Neurons are located at fixed points in 3-D space, and each one acts as an accumulator with a associated threshold value, while synapses are defined as paths between neurons passing information (represented as charge), which is delayed because of a synapse's propagation velocity. Synapses have associated weight values that may be inhibitory or excitatory. Delay throughout the network is defined by a synapse's length, which corresponds to the distance between the two neurons that the synapse connects. Depending on the synapse's delay, both long and short-term information may be stored in the network.

Neurons accumulate charge and fire, at which point they enter a refractory period in which they may still accumulate charge but cannot fire. Synapses delay charge for a certain amount of time before firing to model delay between neurons. The state of the network, which includes the charges accumulated by neurons and events traveling along synapses, evolves in an event-driven simulation. An example of a neuron's behavior is shown in figure 2.3. The far-left image shows charge traveling along a synapse to a neuron. The middle-left image shows the charge reaching the neuron.

Once received, the neuron adds the receiving charge to its stored, accumulated charge. The accumulated charge exceeds the neuron's threshold, so the neuron fires. The accumulated charge travels along the synapses connected to the neuron as shown in the far-right image.

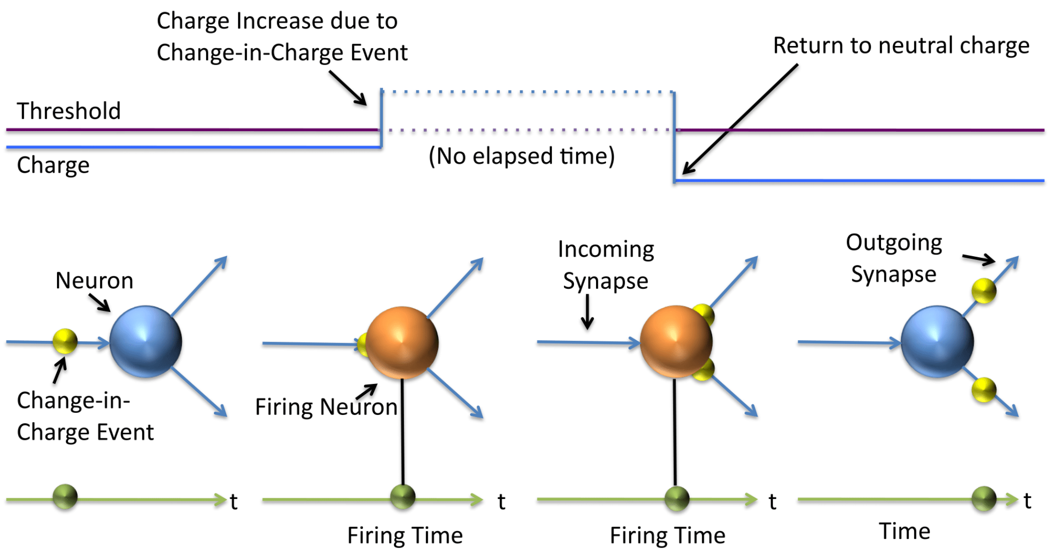


Figure 2.3: Element Operation [26]

NIDA networks are unique in that they use evolutionary optimization tools to adjust the network based on changing conditions [28]. It trains over the network structure and parameters, including neuron thresholds and synapse weights, as well as the dynamics found within the network. Starting with an initial group of networks created by user information, they experience mutation and crossover producing new networks that are optimized to perform a specific task. Using this method results in networks optimized for various applications such as supervised tasks and reinforcement learning.

The work done with NIDA shows its potential as a new computing architecture for neuromorphic computing. Simulations have shown its application in tasks such as anomaly detection and controls. NIDA is primarily a software simulation-based system, and the work done has potential applications in hardware. Implementing the NIDA architecture in hardware would allow for more usage in applications than with just a software platform.

Chapter 3

DANNA

A DANNA is a system architecture designed to support implementation of artificial neural networks emulated in hardware. As stated previously, Schuman et al. introduced a new type of computing architecture, the Neuroscience-Inspired Dynamic Architecture (NIDA). DANNAs can be thought of as a 2-D hardware implementation of NIDA. Their uniqueness lies in the fact that their physical layout is not exclusive to one specific application. Being dynamically programmable, the arrays can be reconfigured to form a wide variety of network structures limited only by size of the DANNA array.

With the use of evolutionary optimization tools to train element connections and interconnections, DANNA arrays can be reprogrammed to adapt to changing conditions allowing for high adaptability and flexibility when compared to other hardware models [20]. DANNA arrays also carry the advantage in that their implementation is highly simplistic. By relying only on select capabilities of neurons and synapses rather than all characteristics of a biological network, element complexity is reduced while still retaining the ability to perform various tasks through complex array structures.

3.1 Overview

A DANNA array utilizes basic neural components integrated into a generic structure referred to as an "element". Elements are arranged in a 2-dimensional structure using a nearest neighbor connection matrix. When an element fires, it sends its weight to all connected elements. Supporting structures that are included within the DANNA array are: a pseudo-random number generator (to support randomization of input sampling for each input) and a clocking module. All elements receive signals from these modules. A finite-state machine module (FSM) supports communications between the DANNA and a host computer.

The DANNA array structure operates through commands sent by a host processor. A communications interface receives command data sent from the host and stores it in a First-In-First-Out (FIFO), the command FIFO. An FSM for the FIFO reads from the FIFO and concatenates the data into commands. These commands go to a programming FSM, which interpret the commands and act upon the DANNA array. In return, the DANNA array returns status packets, which the FIFO FSM reads and breaks up into data for a second FIFO, the response FIFO. The communications interface reads from the FIFO and returns the data back to the host computer. A block diagram showing the DANNA array implementation and associated modules can be shown below in figure 3.1.

3.2 DANNA Element

DANNA arrays are comprised of basic re-programmable elements that can act as a neuron or a synapse depending on the network. The representation of the neuron is highly simplified, but the trade off allows for complex networks rather than complex neurons [27]. Each neuron in the network has a programmable threshold and a refractory period. Neurons connect each other through directed synapses, with each neuron containing a set of synapses going to other neurons as well as a set

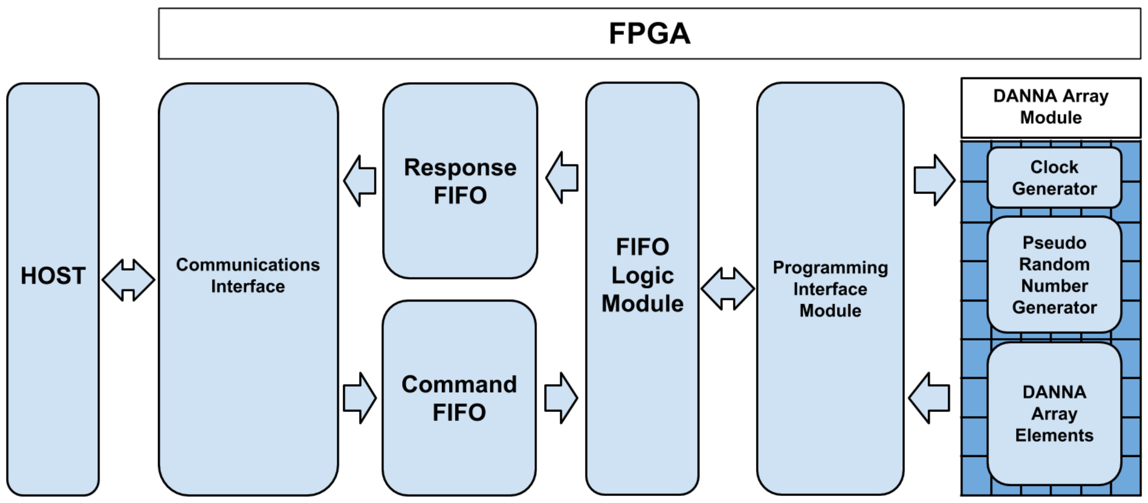


Figure 3.1: Block Diagram of DANNA Implementation

of synapses coming from other neurons. Neurons operate by accumulating charge from its synapses until its threshold is reached. After it has been reached, the neuron will fire its output, and its charge will reset to a bias level. After firing, it enters a refractory period for one network cycle during which it can accumulate charge, but it cannot fire.

A synapse can be defined by the neurons it links together, as each synapse links one neuron to another [27]. Each one contains a distance between the two neurons and a weight value representing the strength of the connection, with the distance representing how many network cycles it takes for charge to travel across the synapse. The primary operation of a synapse in a DANNA work is to adapt and transmit a weighted firing signal based on the firing rate of its input neuron, the firing conditions of the corresponding output neuron, and its programmable distance, utilizing a ”distance” FIFO to simulate a synapse transmitting a set of fire events based on its length. A synapse can have one of its I/O ports enabled as an input and one I/O port enabled as an output. When it receives a fire event from its input neuron, it stores it in its distance FIFO which shifts out stored events; when it reaches the output, the current weight stored in the synapse is transmitted as a firing event at its associated output port. When a synapse fires and its transmitted charge causes the neuron at it output port to fire, then the synapse experiences long-term potentiation (LTP) and increases its weight by a fixed value; on the other hand, if the neuron at its output port fires but not as a result of the synapse's charge, then it experiences long-term depression (LTD) and decreases its weight by a set value. Once a synapse experiences LTP or LTD, it goes into a refractory period during which it cannot adjust its weight until the moment has passed.

A DANNA element utilizes processes alongside functional components to support behavior as a neuron and a synapse. During each network cycle, an element samples each input port, acquires the incoming fire condition, loading it into the synapse FIFO as necessary, and checks the fire condition of the element assigned to the output port. The initial input is randomized through the use of a pseudo-random number generator global to all elements, after which it checks the remaining inputs in a sequential order. Next, the element will accumulate the acquired input weight with the charge state and compare the accumulated result with its programmed threshold if the element is set as a neuron. As a synapse, the accumulator will hold the LTP/LTD weight, and it will increase or decrease its weight depending on the fire condition of the element sampled at its output port. Finally, the element as a neuron will fire its threshold at its output and resets its accumulator to its bias value if the charge exceeds its threshold while it will fire its output if a fire event is present at the synapse FIFO output if the element is a synapse.

A DANNA element contains all functions required to operate as a synapse or a neuron. Physically, it features programmable registers to hold specific values: a Threshold/Weight Register which holds the neuron threshold or a synapse weight, a Synapse Distance Register to hold the distance value of the synapse, an Input Enable Register to specify the active inputs from which to sample, an Output Select Register which specifies the connected element to monitor for LTP/LTD if the element is a synapse, an Output Register to hold an element's weight if a synapse or a threshold if a neuron, and an LTP/LTD refractory period register to hold the length of the refractory period of a synapse. Other important features of a DANNA element include ports that communicate fire events from neurons and weights from synapses, multiplexers and latches to latch the selected input fire and weight, an accumulator consisting of an adder, comparator, and a latch to hold and calculate a neuron's charge or a synapse's weight as well as comparing the calculated charge to the neuron's threshold value, the Synapse Distance FIFO previously mentioned to store firing events for synapses and maintain delays between the events, and a counter to implement the LTP/LTD synapse refractory period [20].

The elements in a DANNA array are self-contained, but there are some array features that correspond to all elements. For instance, when programming an element, an address signal is sent to all elements in the array. Each element has a corresponding address value, and if the signal sent to the DANNA element matches the value, it uses the corresponding data to program its internal registers. Another global function is a global input select signal telling elements what input port to sample from. All elements sample inputs in the same order ensuring equal operation among all elements. The DANNA array also contains other essential components necessary for operation, such as a clocking module, a pseudo-random number generator, a programming interface module, and a FIFO module.

3.3 Clocking Module

The clocking module contains a clock generator that creates global array clocks that are necessary for the array's operation. There are four primary clock signals: the Global Net Clock (GNC), the Acquire Fire Clock (AFC), the Accumulator Enable Clock (AEC), and the Accumulator Clock (AC). The GNC controls the overall operation of the array, with one network cycle being defined in terms of one GNC cycle. The AFC samples fires from all inputs of an element, and it operates at a faster frequency than the GNC so as to sample all inputs of an element. The AC and AEC both work together to operate the accumulator of the element, with the AC running at a faster frequency than the AFC and the AEC running at the same clock speed as the AFC but with a 90 degree phase shift. These clock signals go to all elements and modules of the DANNA array.

3.4 Psuedo-Random Number Generator

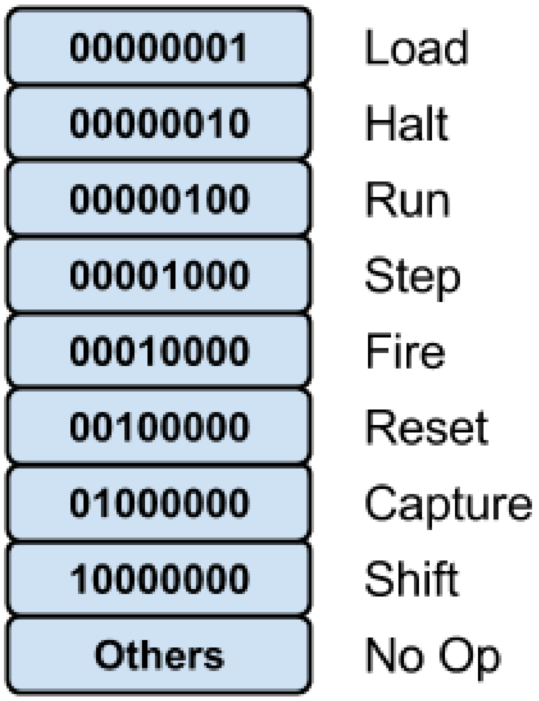
The pseudo-random number generator is a module that selects an input to be sampled during each AFC cycle. It sends a global signal to all elements, indicating which input port to sample, starting with a pseudo-random input. It iterates from the initial input until all inputs have been sampled. It uses a 63-bit linear feedback shift register to generate a seed value that is used to determine the starting input in a pseudo-random fashion allowing for all inputs to be treated equally so no one input gets priority over others.

3.5 Programming Interface Module

The programming interface module helps configure and operate the DANNA. It parses commands received from the host computer and acts according to the operation specified by the command. It also receives data from the DANNA array and returns status packets back to the host computer during specific events. Commands sent to the programming interface allow it to perform actions on the DANNA such as running the DANNA, halting the DANNA operation, programming a DANNA element, run the DANNA for a specified number of network cycles, reset a DANNA configuration, send external fires to a DANNA array, and monitor DANNA elements. This is all done through a unique programming structure specified for the DANNA.

3.6 DANNA Programming Structure

DANNA commands are identified by their operation code, a unique one-byte value corresponding to a specific action to be performed. The list of current operation codes are shown in 3.2. The codes are programmed in a one-hot encoding scheme for easy interpretation. Operations supported by the programming structure include loading/programming an element, halting the DANNA array, running the DANNA array, "stepping" the array for a specified number of network cycles, sending fires to external input elements, resetting the array, capture values of an element, and shifting captured values to the programming interface. Other unspecified codes are considered "no-ops" where no action is taken during a network cycle.



00000001	Load
00000010	Halt
00000100	Run
00001000	Step
00010000	Fire
00100000	Reset
01000000	Capture
10000000	Shift
Others	No Op

Figure 3.2: DANNA Commands Operation Code

The bytes that follow the operation code are payload data whose usage depends on the particular command. The full command layout can be shown in 3.3. This structure is designed in such a way that all commands are the same length with the difference being how the bytes are used. With certain commands such as the reset and halt command, the following bytes after the operation code are not used whereas with the load command, the following bytes are used to program an element.

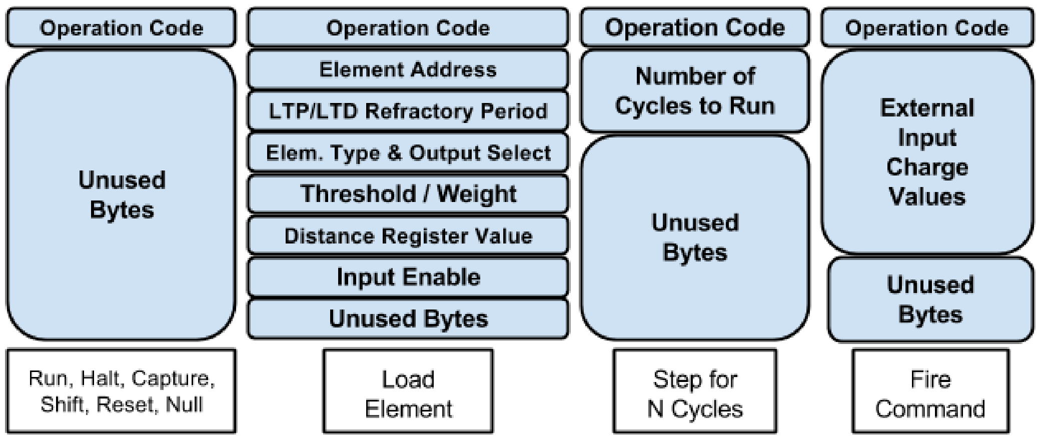


Figure 3.3: DANNA Commands and their Structure

The DANNA array returns status packets during response conditions. These conditions include when the array is sending an external fire off of the array to the host computer, when the array has been halted through the use of a halt or step command, or when the array receives a shift command to shift data of the elements off of the array. The packets are of a fixed length, and is structured to indicate various information of the DANNA array. Figure 3.4 shows the structure of a status packet. The first eight bytes indicate the time-stamp of the response in little-endian order, and the following 32 bytes are weights from output elements when fired. The shift output contains data captured in shift registers when a shift command is received. The status flags take up a byte, with one bit indicating the presence of shift output data and another bit indicating whether the response is a result of a halt or step command. The last two bytes are a configuration ID used to identify the DANNA programming implementation.

Commands and status packets are sent to the programming interface through the use of FIFOs making it independent of the method used to communicate with the DANNA array. The FIFO logic is the communications module that reads and writes data to/from the FIFOs. It is through these FIFOs that the DANNA array

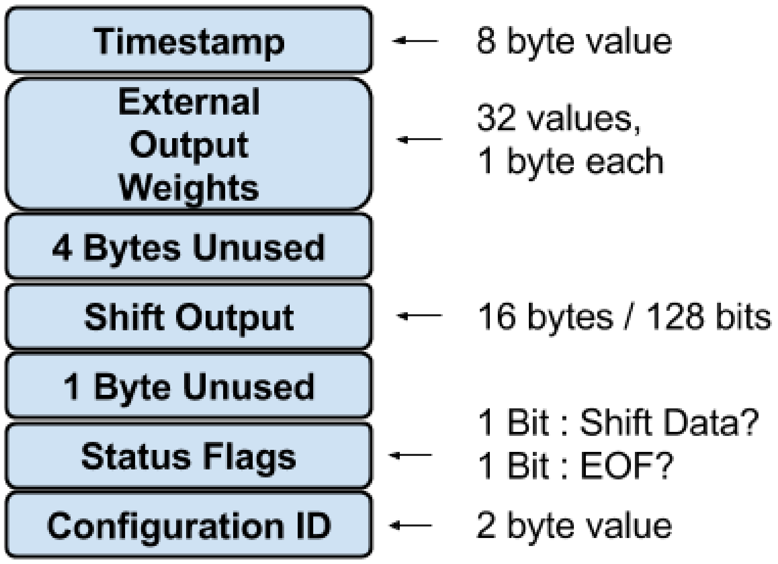


Figure 3.4: DANNA Status Packet

communicates with the outside world. The external host processor sends commands to the DANNA array which are stored in the command FIFO. The FIFO logic module reads these commands and sends the commands to the programming interface. When the programming interface has data to send off of the DANNA array, it sends the data to the FIFO logic which stores it in the response FIFO to be read as shown previously in Figure 3.1.

3.7 Initial Implementation

The initial DANNA implementation showcased its functionality on an FPGA, with some modules being configured to support the implementation at the time. Elements were connected to eight other elements, designated as the eight closest neighbor elements. Clocking of the DANNA array was done in consideration to the eight connections of each element. During each network cycle, an element had to sample all of its eight inputs and accumulate any charge received. Thus, the GNC was 1 MHz, the AFC was 8 MHz, the AC was 16 MHz, and the AEC was 8 MHz with a

90 degree phase shift from the AFC. Likewise, the pseudo-random number generator was configured to only support the eight connections of an element. The DANNA array was subjected to limited monitoring, with status packets being sent to the host computer when an edge element fires or when the array was halted due to a halt or step command. Communication between the host computer and the DANNA array was performed through PCI Express. This was accomplished through the use of Xillybus, an FPGA IP core allowing for direct memory access over PCI Express between a computer and an FPGA [15]. The core was implemented on the FPGA, and it communicated with the command and response FIFOs to send commands and receive packets to/from the DANNA array. All the host computer needed to do was to invoke the Xillybus drivers in their application to send and receive data.

The programming structure of the DANNA array was structured around the current build of the DANNA array at the time. The array implementation only supported sixteen external inputs and outputs, and the commands and status packets reflected that [21]. Commands were limited to 20 bytes, and status packets were fixed at 24 bytes. Capture and shift commands were not part of the programming interface. The only information contained in a status packet were the weights of an edge element if it fired, and the time-stamp at which the fire occurred. Aside from that, the status packets do not tell much about the array itself. Individual elements could not be monitored, so the status of elements at specific cycles could not be tracked.

While it was shown to be successful, there were challenges with the design. For instance, with the eight element connection scheme, there were issues when designing networks, as elements could potentially block other elements from communicating with one another. An element had to be configured to allow communication between elements if they were not next to each other. Another issue involved the pseudo-random number generator. The linear-feedback shift register used for the pseudo-random number generator was constructed with 16-bit shift register elements concatenated to form one large shift register. This design was simplistic and functional, but it could not be reset. When it tried to reset, the shift registers would not revert back to the initial seed value. As a result, the DANNA array could not start back at the same random sequence during subsequent runs.

There were also issues facing the PCI Express communication. The Xillybus FPGA core was very complex, and implementing it in the FPGA design caused some timing issues. The core was specifically designed for PCI Express communication, so the DANNA FPGAs needed to have PCI Express support, and the host computer had to be able to support PCI Express as well. The communications method only utilized a fraction of the throughput available with PCI Express. Also, a consequence of utilizing PCI Express with Xillybus forced the host computer to be reset during each run. Every time the FPGA was reprogrammed with a new DANNA array, the host computer had to be rebooted without killing power to the FPGA so that the original design would not be lost.

Chapter 4

Enhanced DANNA

The challenges with the previous implementation of the DANNA array showcased the need for improvement in its design. Initially, changes were made to the element design to enhance its capabilities. This led to adjusting the supporting modules as well. Altogether, these changes removed the drawbacks associated with the previous design allowing for a more flexible implementation of DANNA.

4.1 Element Improvements

In the new implementation, the element connectivity was enhanced. Elements now connect to sixteen elements instead of eight which eliminates blocking and improves the fan-out/fan-in of each element. An element not only connects to its eight nearest neighbor elements, but it also connects to the following eight elements in the same direction. An example of the new connection can be shown in 4.1. The center element in red connects to sixteen other elements indicated by the arrows coming out of the element.

Increasing the number of connections per element required changes to some internal element signals and registers which carried over to other modules. For instance, the input select register which controls which inputs are enabled for an element had to be increased from eight to sixteen bits. Similarly, the output

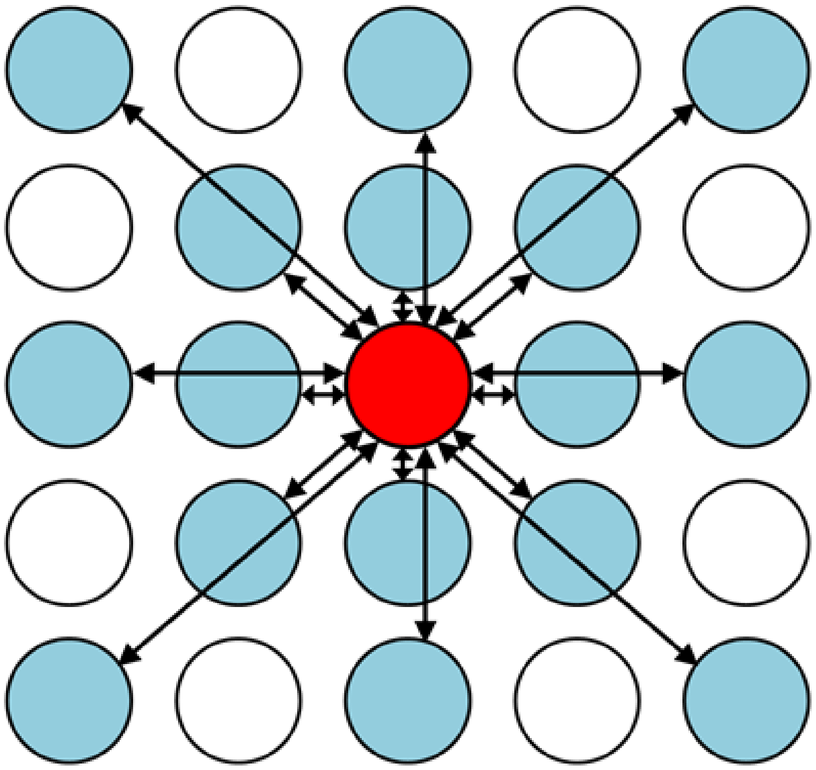


Figure 4.1: Element Connection

enable register which specified which element to monitor for a synapse was increased from three to four bits. The internal multiplexers were also expanded in size to accommodate the new connections.

Elements can now be monitored in the new DANNA implementation through the use of a JTAG-like structure built using shift registers within the element. Through the use of capture and shift commands, elements can send out information about themselves at specific clock cycles. The capture command tells the shift registers to capture values of an element into the shift register, while the shift command shifts data from one element to another, e.g. in a JTAG-like fashion. The shifted data gets put into a DANNA status packet which is sent to the host.

4.1.1 Element Monitoring

Monitoring of the array is an important feature that allows insight to the status of the array elements at a given time. It is implemented in the DANNA through the use of shift registers that capture and shift values off of the array. A diagram of the procedure is shown below in figure 4.2.

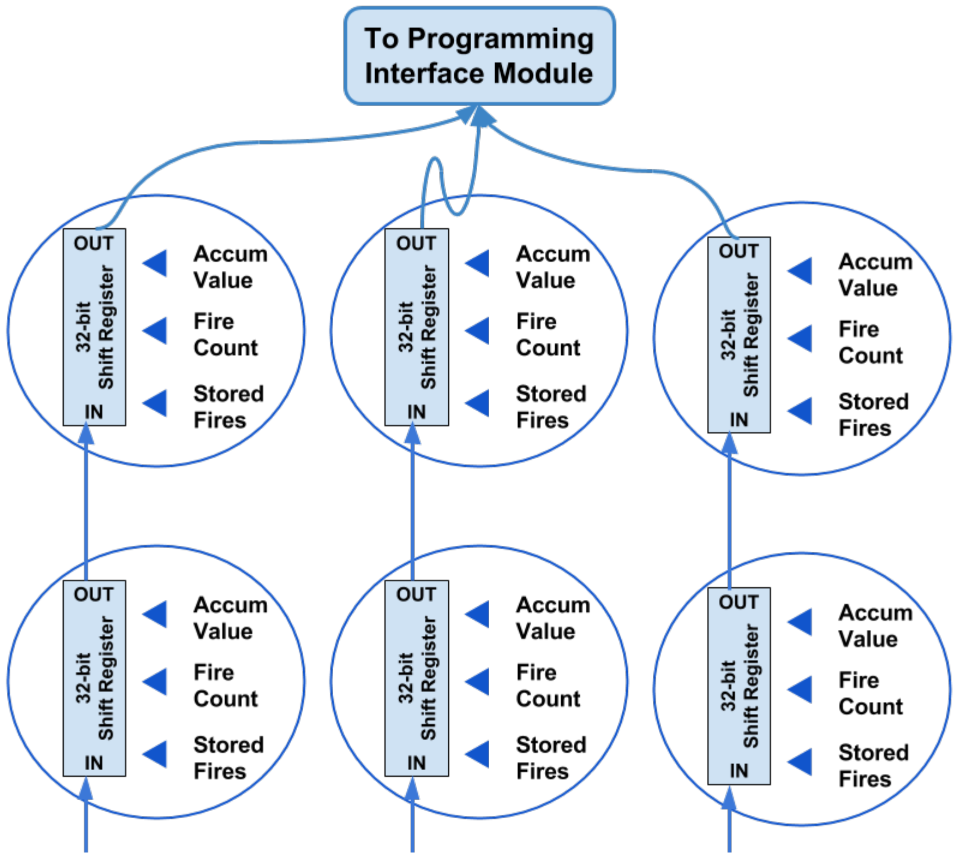


Figure 4.2: Element Monitoring Diagram

In the software, it utilize two commands, capture and shift, to consolidate and retrieve the data. The capture command loads three important pieces of data into each element's shift register: an element's current accumulator value, the number of times an element has fired since the last capture command, and the the number of fires stored in its synapse distance register. The accumulator value informs of the accumulated charge since its last fire if the element is a neuron, or it represents the current synapse weight including the effects of long term potentiation and depression. Once the values have been loaded into the shift register, the shift command shifts out the concatenated values one bit per column per command to the next element above in each column of the array. At the top of each column, the bits are shifted into the programming interface which sends the bits of data to the interface board via a status pscket. Capturing and shifting data are global, so when the commands are sent, all elements capture and shift their data even if they are not programmed.

While the monitoring implementation is a useful feature to get the sense of the array at a given time, there are some limitations to it. One noticeable drawback is the amount of time it takes to get the data of every element. Capturing data and shifting data off-chip takes one GNC cycle. Given an 80×80 array, it can take approximately 2500 GNC cycles (5 ms) to shift all of the data off-chip. In that time frame, the array can continue to experience important events which will be missed while shifting data. This is mitigated by the adjustment that capturing and shifting of data can be accomplished when the array is halted. Also, there is a limitation to the number of columns that can be supported by the programming interface with the current limit set at 120 columns.

4.2 DANNA Module Improvements

The clocking module was adjusted to allow sampling of all sixteen connections of an element during a whole network cycle. This required changing the ratios of the clock signals in reference to the global network cycle. In the new implementation, the Global Network Clock now operates at 0.5 MHz with the other clock signals remaining the same.

The pseudo-random number generator was also adjusted to support the increased amount of connections. It now outputs a 4-bit signal to account for each connection, so during each network cycle, an element samples all sixteen of its inputs. Also, the implementation for the linear-feedback shift register was redone so that the random number sequence may be properly reset. This was accomplished by using a flip-flop implementation of the shift registers interlinked together so that they may be reset to its original seed value when the DANNA array is reset.

The DANNA programming structure also faced some modification to accommodate the changes. The number of inputs supported by the DANNA array was increased to 32 inputs and 32 outputs. While the general structure of the DANNA commands stayed the same, their length was increased to 36 bytes. The format of status packets remained the same as shown in 3.4, but their length was increased to 64 bytes to accommodate the added output edge elements as well as the element monitoring functionality.

The various improvements to DANNA eliminated the problems from the previous implementation, allowing for an enhanced version suitable for the DANNA development kit.

Chapter 5

DANNA Development Kit

The DANNA Development Kit is a hardware kit that allows researchers to take advantage of DANNA networks through a hardware platform. It utilizes the new implementation of DANNA to make its usage more portable and user friendly while eliminating the drawbacks of the previous implementation. The kit is self contained and requires no outside drivers or software. An image of the development kit assembled can be shown in figure 5.1.

It is comprised of an host computer module shown as the green board, an FPGA module shown as the red board, and a communications module shown as the blue board to interface between the two. The host computer sends commands to the DANNA and receives status packets. The FPGA module hosts the DANNA implementation, and the communications module allow for data transfer between the two boards. The host computer and the communications module communicate through a Universal Serial Bus (USB) while the communications module and the FPGA module communicate via an FPGA Mezzanine Card (FMC) interconnection. A JTAG port on the FPGA module communicates with a workstation to program the DANNA array implementation.

A block diagram of the entire system is shown in 5.2, with the perspective being from the point of view of the host computer showing the connections between

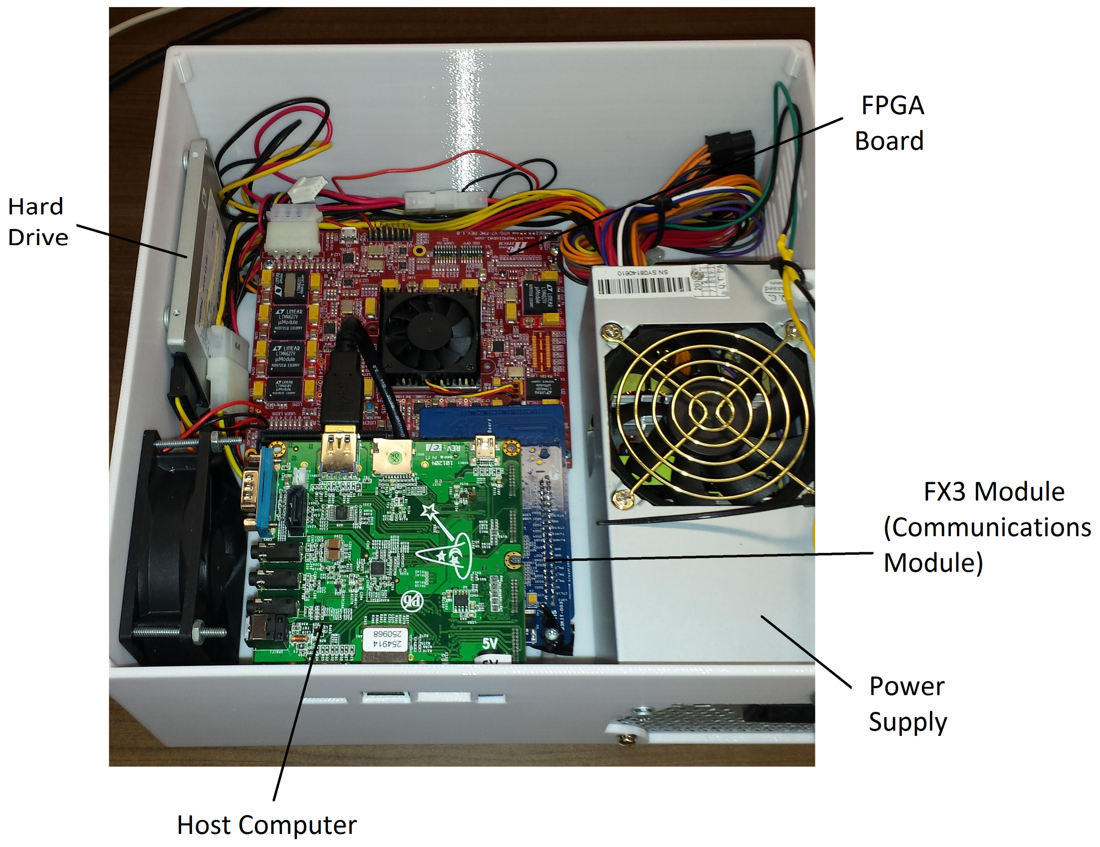


Figure 5.1: DANNA Development Kit

the modules. The host computer sends DANNA commands through USB to the communications module. The communications module buffers the commands received and indicates to the FPGA module that there is data to be read. An FSM module on the FPGA reads the available data and stores it in the command FIFO to be read by the DANNA array. Similarly, when a DANNA status packet is to be sent out, the FIFO logic module stores it in the response FIFO, which is read by the FSM and written to the communications module so that the host computer can read the data via USB.

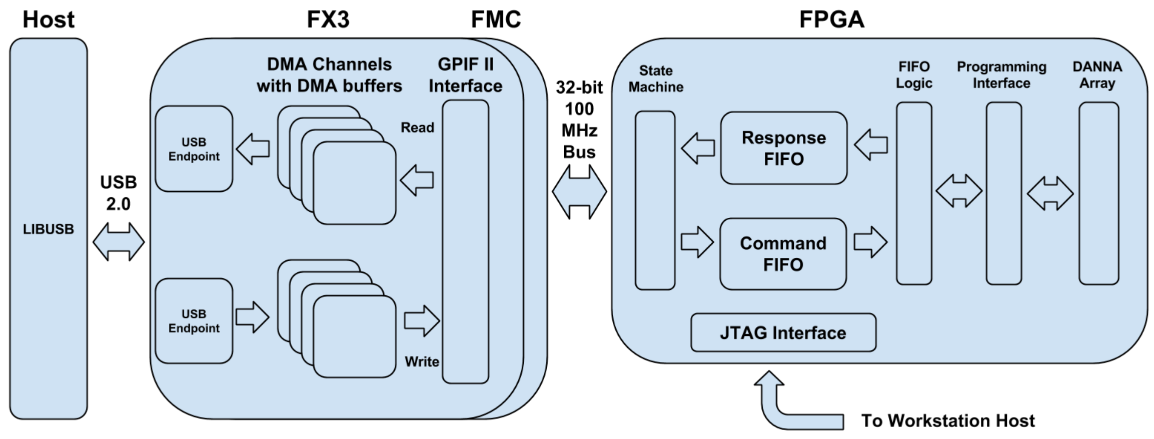


Figure 5.2: Block Diagram of Kit

5.1 Host Computer

The host computer board is a ARM-based single-board computer that configures the DANNA, sends external fires to the DANNA array, and receives status packets back from the DANNA array. It also runs the optimization engine necessary for evolutionary optimization of the DANNA array. To serve its purpose, several single board computers were considered for use in the kit, with criteria being given to available memory, processing power, the operating system that can be run on the board, its physical size, and cost. Some examples of computers considered for the kit included the UDOO Quad, the Wandboard Quad, the CuBox-i4, and the ODROID-XU3. However, it was decided that the Wandboard Quad Single Board Computer would be used for the kit due to its higher memory capacity of 2 GB, its processing power with its Freescale i.MX6 Quad-Core ARM-based processor, and its relatively lower price than its competitors [12]. The Wandboard Quad is shown in figure 5.3, and it is configured to run various operating systems, including Linux. The development kit utilizes Ubuntu 14.04 as the operating system.

Figure 5.3: Wandboard Single Board Computer [12]

Other features of the Wandboard include a SATA port for access to a hard drive, a USB OTG Host port that allows the Wandboard to act as a USB host, and an HDMI port for a display to be attached. There is also an Ethernet port for Internet access as well as to allow users to remotely log into the system from a separate workstation without the need for a display or a keyboard.

5.2 Communications Module

The communications module interfaces between the DANNA array and the host computer allowing them to send and receive data packets from one another. For the kit, it is meant to replace Xillybus removing the dependency on PCI Express from the previous DANNA implementation. Several ideas were considered as to how the interface should be, including utilizing a small FPGA to act as the interface. However, with cost and size as a factor, it was decided that the Cypress EZ-USB

FX3 SuperSpeed Explorer Kit be used to act as the interface. Figure 5.4 shows an image of the FX3 Explorer Kit.

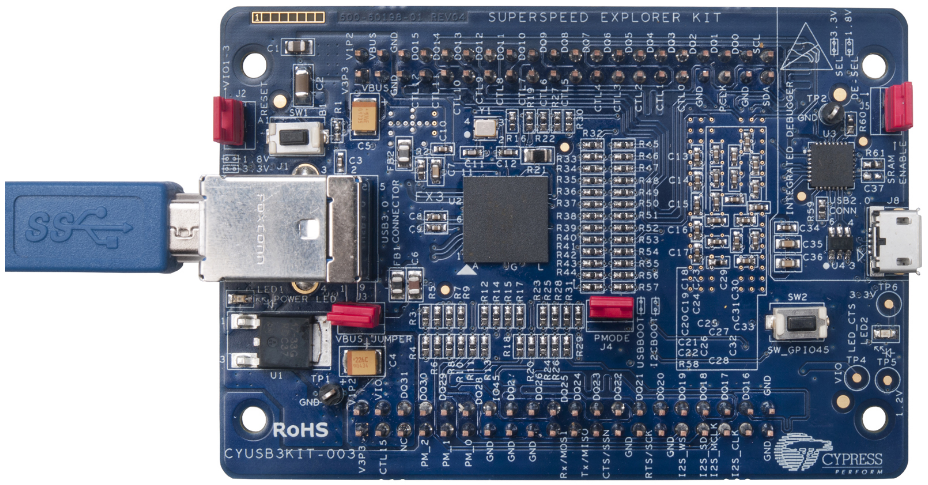


Figure 5.4: Cypress EZ-USB FX3 Super Speed Explorer Kit [3]

The Cypress FX3 SuperSpeed Explorer Kit is a development kit that utilizes the Cypress EZ-USB FX3 USB 3.0 peripheral controller allowing for USB 3.0 device functionality to any external processor [3]. The FX3 controller contains an ARM9 processor for high performance as well as a fully configurable, General Programmable Interface (GPIF$^{\text{TM}}$II) along with other basic communication peripherals such as I2C, I2S, and UART implemented through GPIO pins. Through use of DMA channels and the GPIF interface, the FX3 can provide communication between a USB host and an external processor, which, for the kit, is an FPGA.

5.3 FPGA Module

The FPGA module holds the DANNA array including all components necessary to run it as well as the state machine necessary for USB communication. The FPGA was required to contain enough logic cells to support them. Acceptable FPGAs that met the requirements include the Xilinx Virtex-7 XC7V485T, the XC7V690T, and the

XC7V2000T FPGAs, with them being used in testing the previous implementation of DANNA [20]. Seeing their results with DANNA previously prompted their use in the development kit.

For the development kit, the module hosting the FPGA would be constrained by the physical dimensions of the box. It had to be small enough to fit in the case while still being able to fully support a DANNA module without any loss of functionality. FPGA boards used in the previous DANNA implementation included the Xilinx Virtex-7 FPGA VC707 Evaluation Kit and the Xilinx Virtex-7 FPGA VC709 Connectivity Kit. While they were suitable for prototyping with the PCI Express interface, they were considered too large physically for use in the kit.

With physical size being a constraint, other FPGA modules were considered for the kit. Some research was done in finding an FPGA board that could support the Virtex-7 FPGA as well as be small enough to fit in the kit. It was decided that the Virtex-7 FPGA-FMC Module by Hitech Global will be used as the FPGA board for the kit. As shown in figure 5.5, it contains a Xilinx Virtex-7 FPGA in a small form factor of $5.5in. \times 4.5in.$ [5]. Other peripherals include three FMC connectors, a DDR3 memory SODIMM socket capable of supporting up to 8 GB, two Samtec QSE and QTE connectors with six GTX/GTH Serial Tranceivers, a flash memory module for configuration and storage, a USB port for UART communication, and an I2C bus switch.

5.4 Software

Moving from PCI Express to USB required a different method of communication for software to interact with the DANNA array. The previous implementation took the array configuration file, wrote it to a specified file, and the Xillybus drivers were left to handle the communication with the DANNA array. However, for Xillybus to work, the host machine must be rebooted every time the FPGA is reprogrammed with a

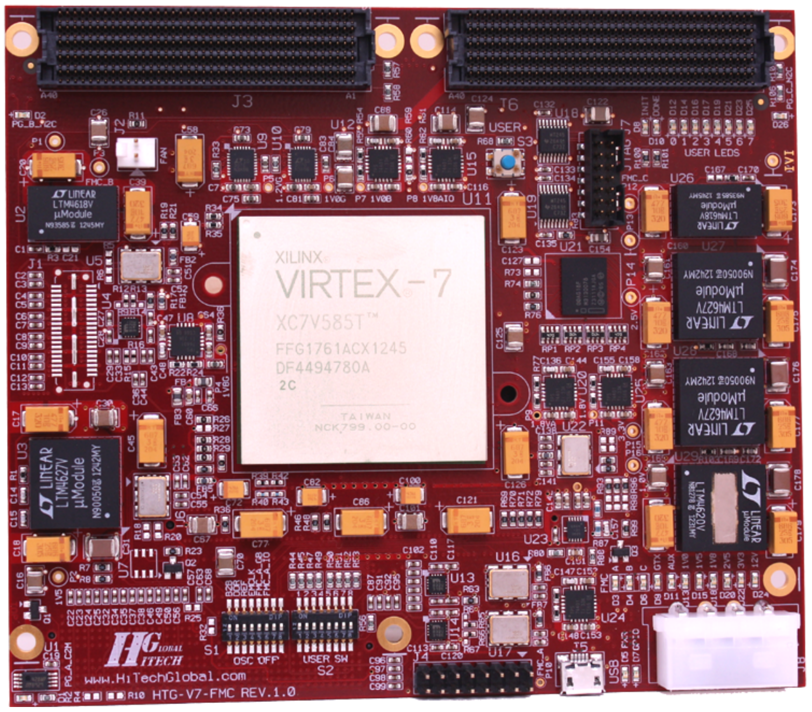


Figure 5.5: Hitech Global Virtex-7 FPGA FMC Module [5]

new configuration. USB communication eliminates this issue by eliminating the need to reboot the host computer and the FPGA module every time.

Once the communication method was decided upon, the software needed to be changed to allow for USB communication. There are many ways of implementing USB communication with a computer. Developing a custom USB driver was considered, but due to the limited knowledge of designing a kernel driver, research was done on finding a pre-existing driver. Microsoft provides a generic USB driver, WinUSB (Winusb.sys), for USB devices that consists of a kernel-mode driver and a user-mode dynamic link library that allows management of USB devices with user-mode software [1]. However, the driver and required components are limited to Windows operating systems. The software for USB communication had to be Linux compatible. After some research, it was decided that the software would use LIBUSB to perform USB operations.

LIBUSB is a C based library that enables generic access to USB devices [7]. It is platform independent, allowing it to be used in many different operating systems including Linux, Windows, and OS X, and it supports all versions of the USB protocol up to the latest SuperSpeed protocol. The API allows the user to interact with USB devices through function calls allowing for user applications to send and receive data from USB devices. LIBUSB was integrated into the DANNA software code on the host platform.

Chapter 6

Build and Initial Testing

The creation of the development kit came about the need to allow other researchers to easily use DANNA in various applications. This required changes to the physical implementation of DANNA to make it more portable as described in chapters 4 and 5. This section highlights the physical changes and testing procedures made to support the DANNA implementation in the kit.

6.1 Physical Layout

The DANNA development kit was envisioned to be contained in a portable package that is easy to move around and set up. With this in mind, the main components of the kit were selected for their capabilities as well as their physical size. The Wandboard computer connects to the Cypress FX3 Dev Kit through the use of a USB cable, while the FX3 was attached to the FPGA module through the FMC connector on the module. This was accomplished through the use of the Cypress FMC Interconnect Board for the FX3. As shown in figure 6.1, the board attaches to the headers on the bottom of the FX3 kit, and it can be attached to any FMC receptacle allowing the selected FPGA-FMC boards to have USB functionality.

Aside from the three main components, the DANNA development kit also contains a solid state drive to store array configurations as needed when using DANNA for

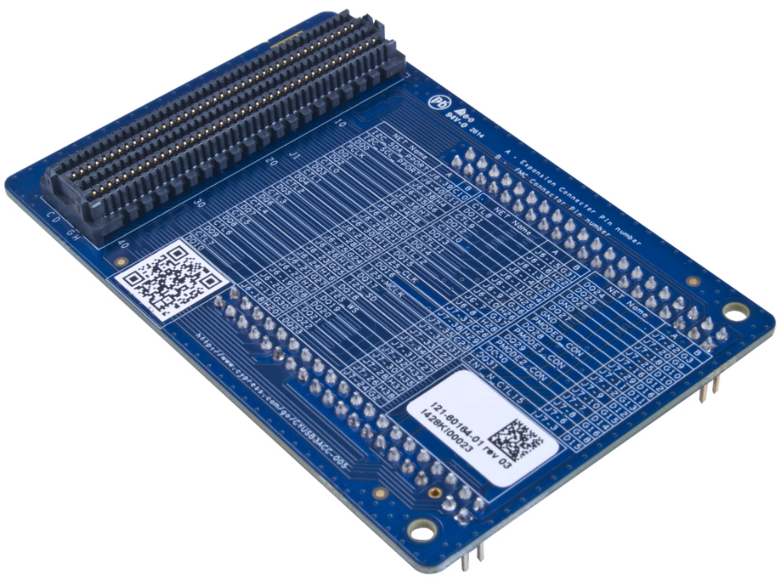


Figure 6.1: Cypress FMC Interconnect Board for the FX3 Dev Kit [4]

applications. The Wandboard itself is placed in a location allowing for easy access to its Ethernet port, HDMI port, and USB OTG port for connecting peripherals as needed. The entire kit is powered by an mini-ATX power supply allowing for full functionality of all modules, and all components are contained in a custom 3-D printed case.

6.2 Communication Interface

From the previous implementation of DANNA, it was clear that the communication between the host computer and the DANNA array needed to be changed. It was impractical for researchers to require a workstation capable of supporting PCI Express in order to run DANNA. Also, the usage of PCI Express underutilized its potential throughput due to the amount of data being transferred. Therefore, a replacement for PCI Express was sought for the DANNA Development Kit.

Initially, the idea for the communications replacement was to develop a custom protocol through the use of another FPGA. The FPGA would be separate from the

DANNA array, and the protocol would communicate with the FIFOs on the DANNA array. While this method provides a lot of flexibility, the complexity of designing a reliable interface would have been time consuming. Also, interfacing with the FPGA hosting the DANNA array would be complex due the lack of I/O pins on the board.

With time and complexity of the interface being an issue, some research was done on existing protocols to see whether they could be incorporated into the DANNA development kit. The RS-232 (UART) method was a popular method which can be easily incorporated onto an FPGA. However, the protocol was very slow, and would not provide the necessary bandwidth needed for the DANNA array. Another method that was considered was Gigabit Ethernet, and Xilinx provides IP cores to integrate it into an FPGA design. However, the unfamiliarity with the protocol as well as the hardware required made it more of a challenge to incorporate.

The USB protocol provided more than enough bandwidth to satisfy the needs of the DANNA array, but implementing it on the FPGA was a challenge. Some companies have developed their own proprietary software incorporating USB with their FPGAs, such as Opal Kelly and their FrontPanel SDK [2]. There is an open-source project known as FPGALink which seeks to integrate USB functionality between an FPGA and a PC [23]. Unfortunately, there were difficulties with the hardware and software requirements that made it unsuitable for the project, such as the PCs requiring specific software drivers and the FPGAs needing specific USB controllers such as the Cypress FX2LP USB controller. However, in an attempt to find a way to integrate the USB controller, the Cypress FX3 USB controller was discovered to be a viable substitute for the outdated FX2LP controller without the need for specialized software.

The Cypress FX3 USB controller can be configured to function as a Slave FIFO interface to allow any external master device to have USB functionality [29]. Because of its simplicity and ease of use, the FX3 Slave FIFO interface seemed like a natural fit to send and receive data from the DANNA array. It operates by creating Direct Memory Access (DMA) channels with a programmable number of DMA buffers to hold data. Through socket interfaces, the DMA channels can send and receive data coming from a host computer or the external processor to which it is attached to. A block diagram of this interface is shown in figure 6.2 showing the bi-directional data bus, an address bus, DMA flags, operation signals, and a clock signal. Flags A and B indicate the availability of a DMA buffer that can be written to and flags C and D represent a DMA buffer that can be read from. The signals SLWR and PKTEND control write operations while SLRD and SLOE control read operations.

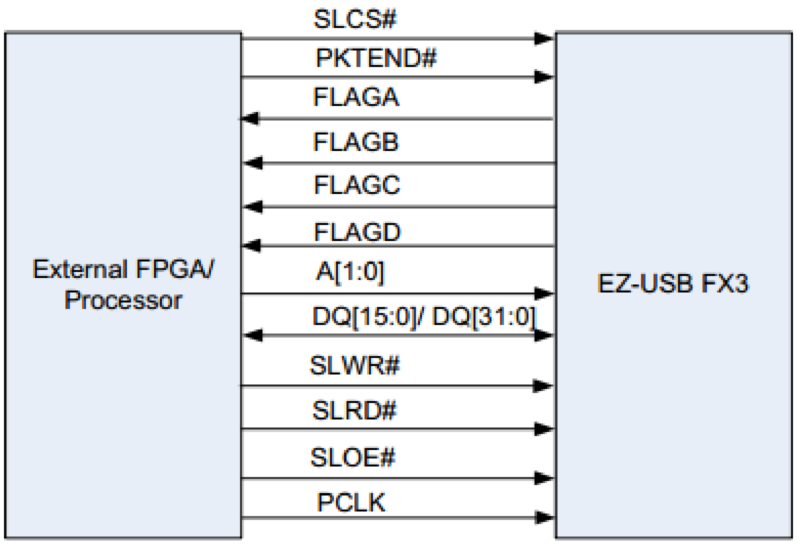


Figure 6.2: Slave FIFO Interface Diagram

The interface utilizes DMA channels: one for receiving data from the FPGA to be read by the host computer (the P2U channel), and one to send data from the host computer to the FPGA (the U2P channel) [29]. The DMA channels interact between the USB host and the GPIF interface through the use of sockets to perform data transfers to and from the FPGA. Each DMA channel can have a number of buffers of a specified size to hold data. DMA channels also utilize sockets to act as producers or consumers of data through which end systems can interact with. The U2P channel uses a USB socket as a producer, which can be accessed by the host computer through a USB endpoint, and it uses a GPIF socket as a consumer allowing an external processor to read data. Similarly, the P2U channel uses a GPIF socket as a producer to receive data from the FPGA, and it uses a USB socket as a consumer for the host to read data through another endpoint. The GPIF interface interacts with the sockets, and DMA flags going to the FPGA indicate the availability of a buffer that can be read or written to by the FPGA. DMA flags are unique to each GPIF socket, and they must be monitored by the FPGA.

On the FPGA, a state machine monitors the DMA flags and acts accordingly depending on whether the flags represent a DMA channel to be read from or a DMA channel that can be written to. This state machine, as shown in figure 6.3, is independent of the DANNA array and can be adjusted to fit the needs of the FX3 firmware. The state machine as well as the GPIF interface operates at 100 MHz. The FPGA acts as the master of the GPIF interface, and the state machine sends signals to the GPIF interface to drive a particular data transfer as well as an address signal to specify which socket to interact with. The GPIF interface interprets the signals and performs a read or write operation accordingly.

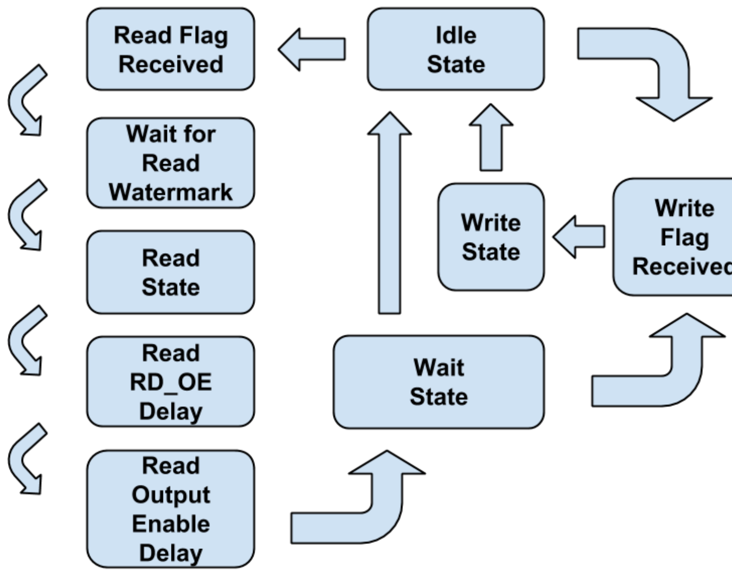


Figure 6.3: Slave FIFO FSM Diagram

The state machine operates with the FX3 at 100 MHz while the DANNA array reads a command from the command FIFO every GNC cycle. The FSM is designed such that reading data from the FX3 has more priority than writing data to the FX3. When there is a DMA buffer available to be read, the FSM goes through the read states on the left. The FSM goes through the states on the right when there is data to be written to the FX3.

6.3 Initial Design, Implementation, and Testing

The Slave FIFO project by Cypress was used as a reference design for the FPGA I/O implementation. The original project was developed for a Spartan-6 FPGA so some components were modified for use with the Virtex-7 FPGA. The clock generator used in the original project was replaced with an clocking wizard IP block that utilized the clock sources available on the FPGA module to create the necessary clock signals. Likewise, their FIFO module was replaced with two FIFO Generator IP blocks, so that the state machine on the FPGA can write data to one FIFO while reading data from a separate FIFO. Also, the locations for the FMC pin placements were adjusted to account for their different locations on the FPGA.

The changes were made incrementally with verification of each change being done through testing. The project was tested through the use of an application provided by Cypress as part of its FX3 Development Kit for Windows. It interacts with the Cypress FX3 SuperSpeed Explorer Kit by loading firmware onto the kit and performing data transfers to/from the device. Testing was performed by sending data out and reading the same data back in a loop to verify the accuracy of the design. Along with using the utility, LIBUSB code was written to send out data and read data back in. This was to confirm that the design still functioned correctly on a Linux platform.

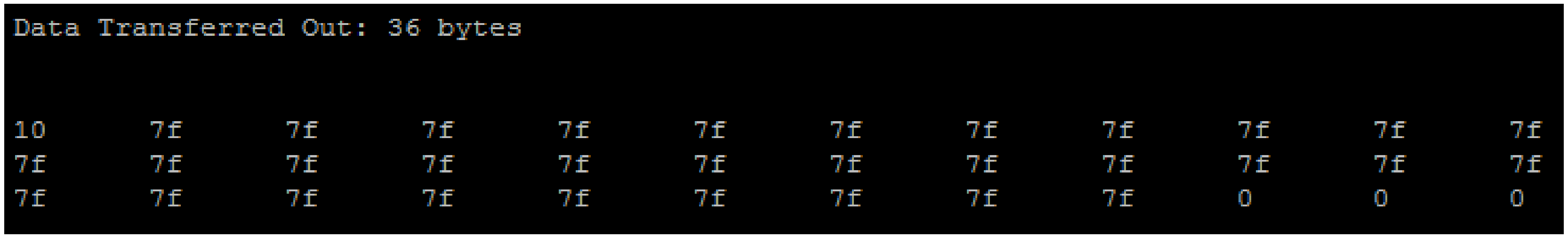
Once the design was verified to have worked, it was integrated with a DANNA array. Working with a small array for testing, the host computer sent out commands and read status packets through LIBUSB to verify that the DANNA array still functioned with the new communications interface. An example of a transfer can be shown below, with a fire command being sent to the DANNA array in figure 6.4 and the FPGA signals of the state machine shown in 6.5. The returning status packet signals is shown in figure 6.6 with the data read back shown in 6.7.

The images show a fire command being sent to the array. Figure 6.4 shows the command sent from the Wandboard with each individual byte of the command shown in hexadecimal. The first byte identifies the operation code of the command, with the fire operation represented as 16. The following thirty-two bytes shows the weight value being sent to the designated input elements. Here, all thirty-two inputs receive a weight of 127. The last three bytes are unused in the command. Probes on the FPGA show the data being read into the command FIFO by the FSM in figure 6.5 with the red line indicating the start of the command. This confirms that the command data was sent to the FPGA correctly.

Once the command was received, the programming interface module sends the weights to the input elements. They receive it, and operate according to their configuration. The DANNA array was programmed to produce a fire response when a fire command was received, so the output elements fire, causing the programming interface module to send a status packet back. The first eight bytes show the time-stamp indicating when the edge elements fired while the following thirty-two bytes show the weight of the indicated output element. No status flags are set, so the status packet does not contain any shift data nor is the packet the result of a halt or a step command.

The FIFO logic module receives the status packet and writes it to the response FIFO. The FSM reads from the FIFO and writes the data to a buffer in the FX3. Figure 6.6 shows the FSM writing the status packet to the FX3 with the data bytes shown in hexadecimal. The last four bytes of the packet are indicated by the yellow marker with the data shown in the second column on the left.

Figure 6.7 shows the bytes of the received packet by the Wandboard in hexadecimal. The time-stamp indicates that the status packet occurred at cycle 323,602,420 from when the DANNA array first started running. The external output weights indicate that the third output element fired with an weight of 129 and the fifth output element fired with a weight of 127. The configuration ID can be observed as the last two bytes, and the lack of any status flags at byte 62 indicate that this is a response from the fire command. The data of the fire command and the response packet can be verified by comparing the probed signals with the data sent out and received indicating that the USB communication interface is functional.



```
Data Transferred Out: 36 bytes


10      7f      7f      7f      7f      7f      7f      7f      7f      7f      7f      7f
7f      7f      7f      7f      7f      7f      7f      7f      7f      7f      7f      7f
7f      7f      7f      7f      7f      7f      7f      7f      7f      0       0       0
```

Figure 6.4: Fire Command - Sending Command to DANNA

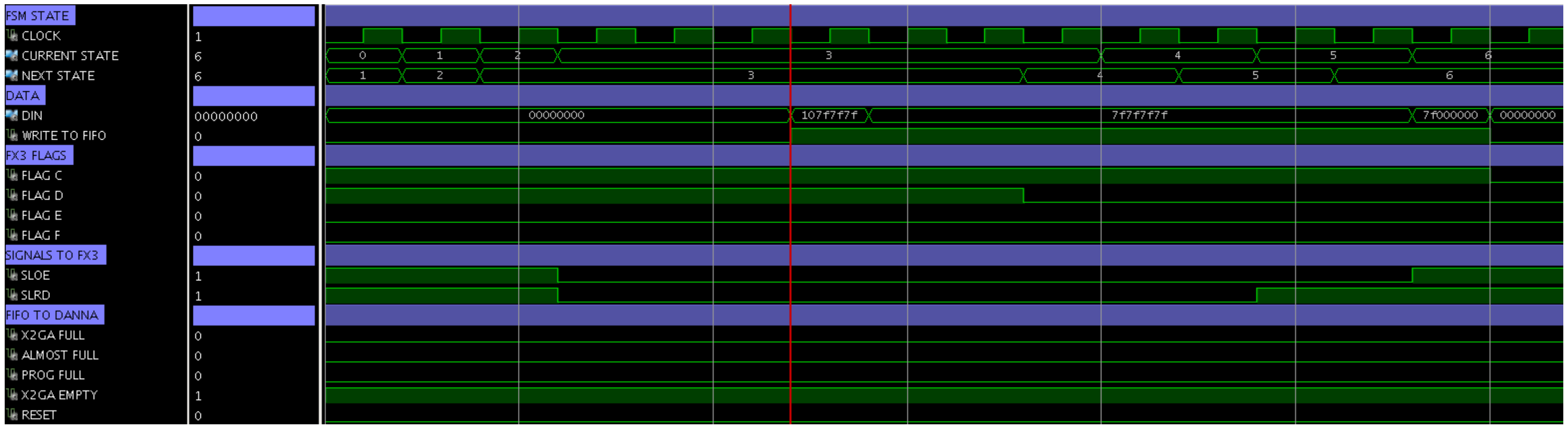


Figure 6.5: Fire Command - Finite State Machine Signals

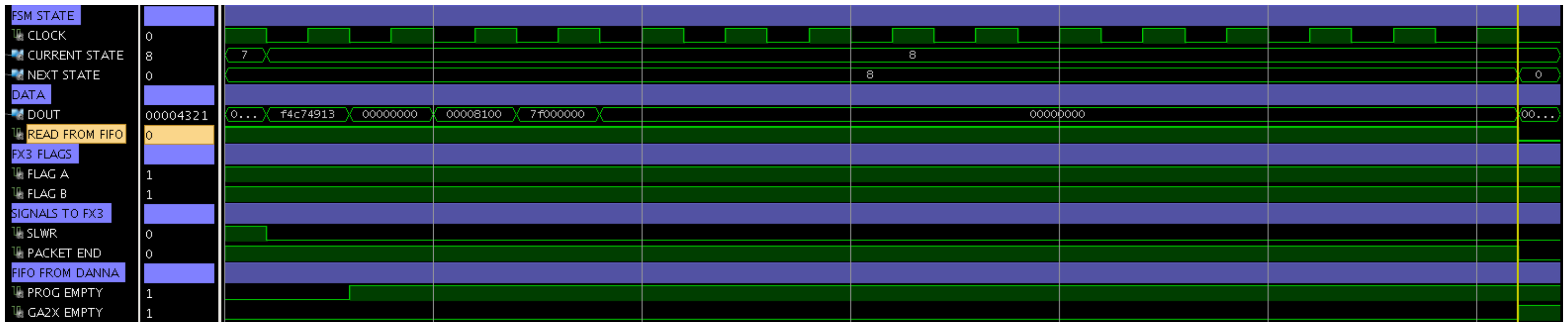
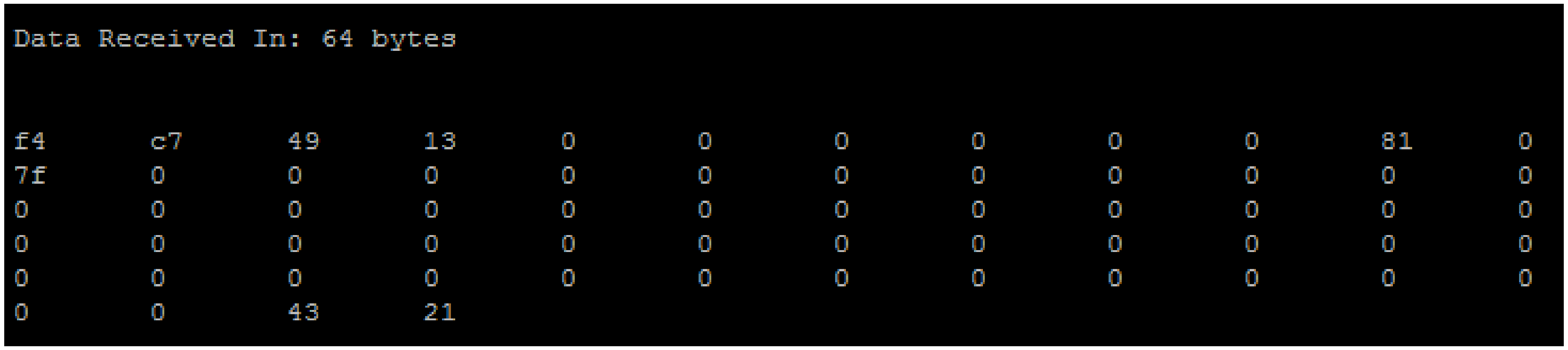


Figure 6.6: Status Packet - Receiving Packet from DANNA



```
Data Received In: 64 bytes


f4      c7      49      13      0       0       0       0       0       0       81      0
7f      0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       43      21
```

Figure 6.7: Status Packet - Data Received

During testing, the firmware on the FX3 as well as the state machine on the FPGA underwent some changes to make it more suitable for the DANNA application. The initial firmware consisted of two DMA channels previously mentioned with one channel handling transfers from the USB host to the FPGA and another handling transfers from the FPGA to the host. Each channel had two DMA buffers, and the buffer size was dependent on the USB speed, 512 bytes at USB 2.0. The DMA channels were configured such that whenever a transfer is initiated along a channel, the operation must be performed through a callback function initiated by the FX3 to commit the data to the DMA buffers. Also, the USB endpoints at the USB host were configured as bulk endpoints, which only allowed for bulk transfers to occur.

Changes to the Slave FIFO firmware were made to increase its potential usage with the DANNA array. DMA channels were reconfigured to automatically drive data during transfers without any need for a callback function. By removing the CPU intervention found with manual channels, data transfers can occur much faster than with callbacks. The buffer sizes were fixed at 512 bytes to accommodate the USB 2.0 standard supported by the Wandboard. Changes to the GPIF interface included changing the endian-ness of data in transfers to ensure that the order of data sent to the DANNA array is correct.

While testing the new design, it was found that there was notable latency when reading data from the FX3 which appears to be caused by LIBUSB and the underlying USB protocol. Testing was performed by sending one command to the DANNA array at a time. The DMA flags took around fifty to sixty microseconds to update the availability of a new buffer to be read from. Whenever the FSM read from a new buffer, the command FIFO would have already been empty for several microseconds, and with the array operating at a period of two microseconds, this caused a delay that propagated throughout the entire DANNA array. This realization led to the fact that the data transfers needed to contain enough data such that when the next transfer comes around, the command FIFO would maintain cycle accurate sequencing with the DANNA's operation.

By sending out commands to the DANNA, results show that transferring one command at a time was not enough to fill the command FIFO and maintain cycle accurate command sequencing. Several tests were ran by sending different size amounts of DANNA commands to the FX3 to determine the recommended amount of data per transfer. It was determined that the data transfers need to get ahead on the command FIFO on the FPGA, and the best way to do that is to transfer enough data to fill the FIFO to ensure that hides the USB transfer overhead between DMA buffer operations. Also, to ensure that no data is lost when filling up the FIFOs, the state machine is configured such that it only reads a DMA buffer when there is enough room in the FIFO to read a complete DMA buffer. This done through programmable FIFO signals indicating how full the command FIFO is.

Changes were also made to the FX3 firmware in order to have it transfer data faster. The initial setup of the firmware had the U2P DMA channel going from the host computer to the FPGA contain only one producer socket and one consumer socket. Having only one consumer socket contributed to the large amount of latency when switching between DMA buffers. By reconfiguring the DMA channel to have two consumer sockets, the channel can transfer data faster by having the sockets work together to switch between DMA buffers rather than have one socket perform all of the switching. The FX3 firmware had to be changed in order to support this new configuration by specifying a new GPIF socket to be used and reconfiguring the U2P DMA channel to support the additional socket. A second set of DMA flags had to be created for the new socket, and the GPIF interface was reconfigured to reflect the new socket. A block diagram of the new structure can be shown in figure 6.8. The DMA channel coming from the USB host now has two sockets to transfer data through, with socket 3 being represented with flags C and D while socket 1 using flags E and F.

The addition set of DMA flags caused changes to the state machine on the FPGA. Now, during a read operation, the FSM tracks two sets of flags and ensures that it reads from the right DMA buffer by having it alternate reading between sockets.

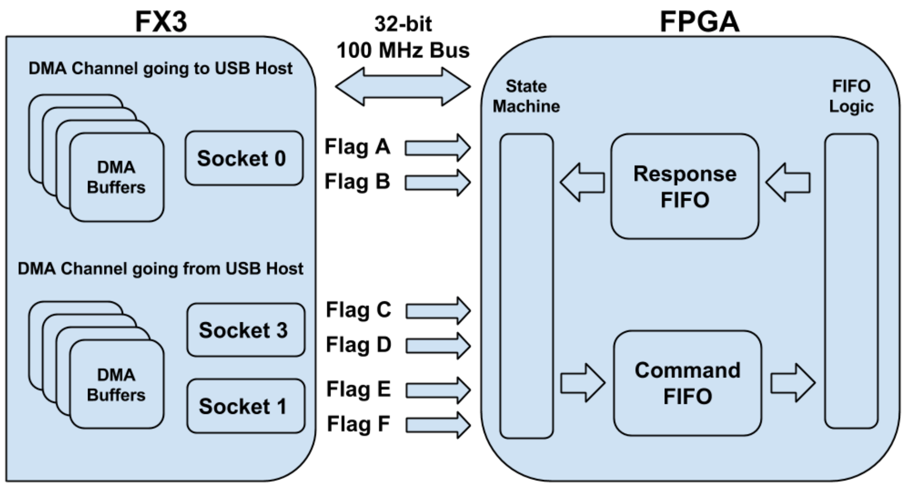


Figure 6.8: FX3-to-FPGA Socket Switching Implementation

The DMA channel is configured to transfer data in the socket order specified during channel configuration, and the state machine is configured such that after it reads from one socket, it is set to read data from the next anticipated socket to ensure that data will not be read out of order. If no data is sent after a long period of time, the FSM will revert back to a neutral state, where it can read from any socket of the DMA channel so that data will not be read out of order during a new DANNA run.

Testing with the additional socket shows significant improvement during data transfers. Results show that it takes around ten to twelve microsecond between reading from DMA sockets during the first few buffers. As the command FIFO on the FPGA gets full, the state machine stops reading data until there is enough room to read an entire DMA buffer. By waiting until there is enough room, the process slows down to a point where after immediately reading from one socket, another buffer is available to be read from the next socket. In this fashion, data can be read by the FPGA quickly showing very little delay between buffers.

6.4 FIFO Reset Implementation

Previously, when working with the PCI Express implementation, the FIFOs on the FPGA could not be reset. Because of the usage of Xillybus, this was not a issue at the time. However, through testing with the USB implementation, there were some cases where stale data remaining in the response FIFO causes response data to be misaligned. In an effort to eliminate the possibility of stale data, the FIFOs on the FPGAs had to be reset between DANNA runs.

The Cypress FX3 SuperSpeed Explorer Kit uses GPIO pins to implement its various communication peripherals. In implementing USB communication with the kit, it was found that the firmware could be used to drive specified pins going to the FPGA which can trigger it to perform specific actions. Therefore, by using a GPIO pin, the firmware could be triggered to send a signal to the FPGA to trigger the command and response FIFOs to be reset. The kit contained plenty of pins specified for interfaces such as I2C and I2S. Since they were not being currently used, some of pins were re-purposed for the kit.

Changes were made to the FX3 firmware to allow free use of the spare pins. During initialization, an unused GPIO pin intended for use by the I2S peripheral was overwritten to allow the firmware to use it. Overwriting a GPIO pin holds the risk of damaging its functionality for the intended peripheral. However, considering how the DANNA array was not using I2S, the risk was determined acceptable at the time. A function in the firmware that handled USB control transfers was expanded to handle USB vendor requests allowing for interaction with the GPIO pin.

Whenever the host application performs a USB control transfer indicating a vendor request with a specific request value, the firmware would interpret the control transfer and drive the specified GPIO pin for two microseconds. During this time, the FPGA would treat the signal as a reset signal and reset both of the FIFOs on the FPGA. This signal was also used to reset the Slave FIFO FSM on the FPGA to a default state to ensure that it was configured correctly during a new run.

Chapter 7

Challenges

Throughout the work done on the DANNA development kit, some issues were faced during creation of the DANNA arrays. The Virtex-7 FPGAs contained different amount of resources, with the 485T containing 75,900 logic slices, the 690T containing 108,300 slices, and the 2000T containing 305,400 slices [33]. The amount of resources on each FPGA limited the possible DANNA array size. With the previous implementation of DANNA, the 485T could support a maximum of around 256 elements, while the 690T and the 2000T could support 2,500 elements and 6,400 elements, respectively [20]. Using this resource as a guideline, it was decided that the HTG-V7 FMC-FPGA module would host a XC7V690T FPGA. As the DANNA implementation changed, the amount of utilized resources also changed to reflect it.

Some issues were faced during the array creation. These issues presented themselves throughout the design process causing some problems with the DANNA array implementation. The software used to build the arrays is the Vivado 2014.2 Design Suite by Xilinx. There was very little control over how the software routes signals together, so timing issues could propagate throughout the entire array. While the timing issues were not as prevalent with the previous implementation of DANNA, they were still significant enough to cause problems throughout the whole design. For instance, when a fire response was expected at a specific cycle, it was found to occur one or two network cycles later than anticipated. To overcome these issues, processes that utilize the same sensitivity list were combined, and large signal vectors were split up into smaller signal vectors between modules and recombined back to its original length. The synthesis settings were also adjusted such that the DANNA array hierarchy was fully flattened in an effort to improve timing. Implementing these changes allowed for a more robust implementation of DANNA.

Another prevalent issue is with the Vivado Design Suite itself. During one point in the development timeline of the kit, the software would experience a segmentation fault and fail to synthesize the DANNA design with no warning as to what could have caused the issue. This seems to occur with large arrays greater than 2,200 elements. Some work was done in an effort to determine what caused the software to experience the fault, but results have been largely indeterminate. However, through tinkering, some larger arrays were able to be synthesized and implemented.

7.1 690T FPGA

The new DANNA implementation consumed more resources than before, limiting the amount elements that can be supported by the development kit. During development, results showed that the maximum number of elements that can be supported by the 690T FPGA for the current implementation is 2,209 elements, or an array size of 47×47. The final implementation routing done by the Vivado software can be shown in figure 7.1.

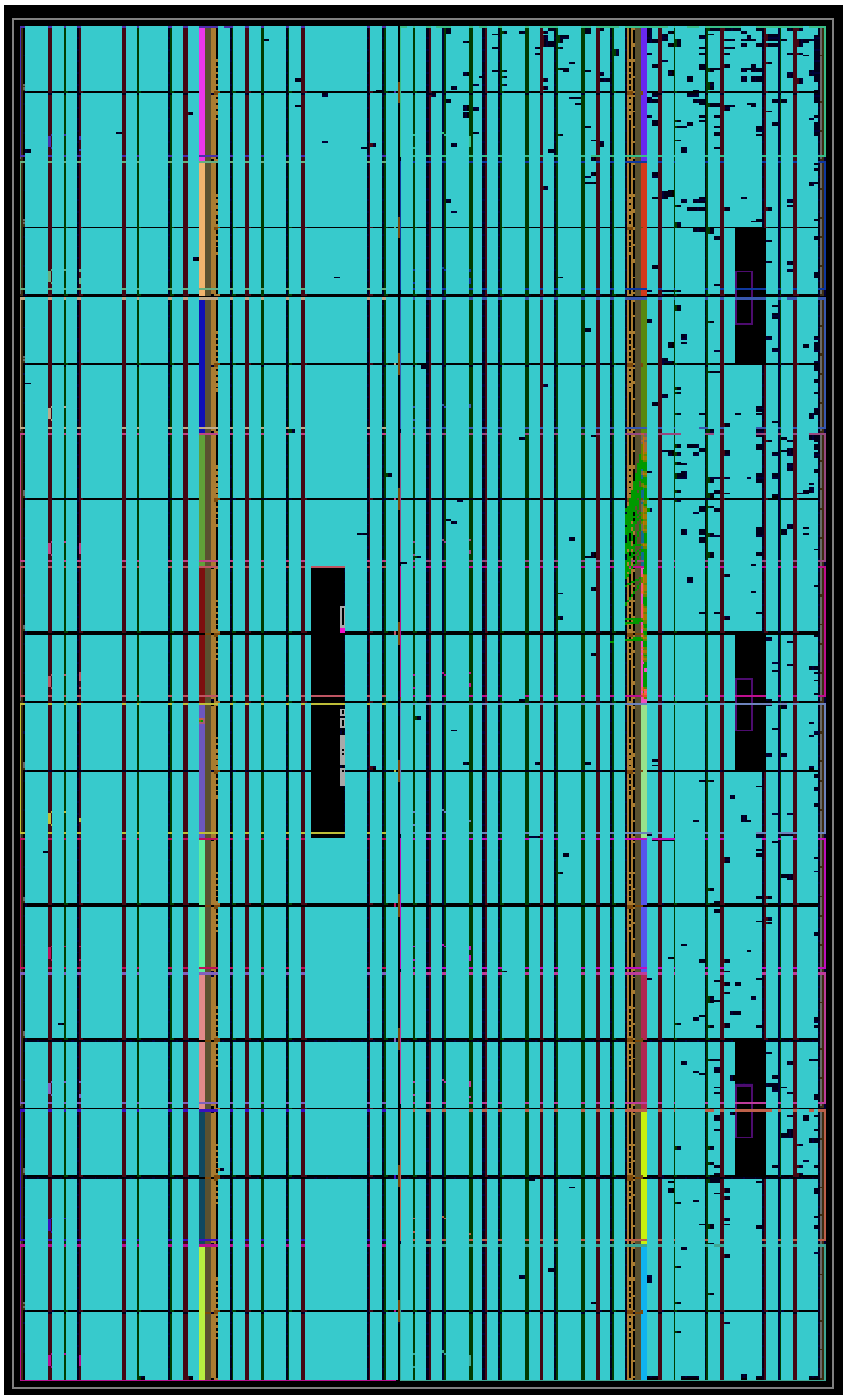


Figure 7.1: 47×47 DANNA Array Implementation

The resulting implementation utilized a majority of the space for the targeted DANNA design. According to the post-implementation reports, the design utilized 79% of the available look-up tables, 31% of the available flip-flops, and 44% of the available buffers on the FPGA. The estimated timing summary reports that the worst-case setup time is 1.052 ns, while the worst-case hold time is -3.087 ns. Despite the slack times reported by Vivado, they did not result in faulty or intermittent operation during testing.

7.2 2000T FPGA

To expand upon the capabilities of the DANNA array, a larger FPGA was used for prototyping in order to achieve a larger array. The prototyping board used is the HTG-700 Xilinx Virtex-7 V2000T PCI Express Development Board shown below in Figure 7.2. It contains three FMC connectors, a DDR3 SODIMM module supporting up to 8 GB, four SMA ports to allow for external clocks, a USB-UART bridge controller, and PCI Express Gen2/Gen3 edge connector [14]. The board hosts a Xilinx Virtex-7 XC7V2000T FPGA, being approximately three times larger than the 690T FPGA [13]. This FPGA module was utilized with the DANNA array implementation, and it also contains FMC ports which allows the new DANNA implementation to be used.

The structure of the 2000T is vastly different than the 690T, with the 2000T containing four super-logic regions whereas the 690T only contained one [32]. This caused timing issues throughout the DANNA array. Because of the 2000T's structure, a clocking tree was spread out throughout the array to ensure that each DANNA element received clocks with delays that tracked signals across the array. This involved partitioning the array into row-like structures, where each row structure contains clock buffers for each of the four DANNA array clocks as well as a pseudo-random number generator synchronized with the linear-feedback shift register. This unique

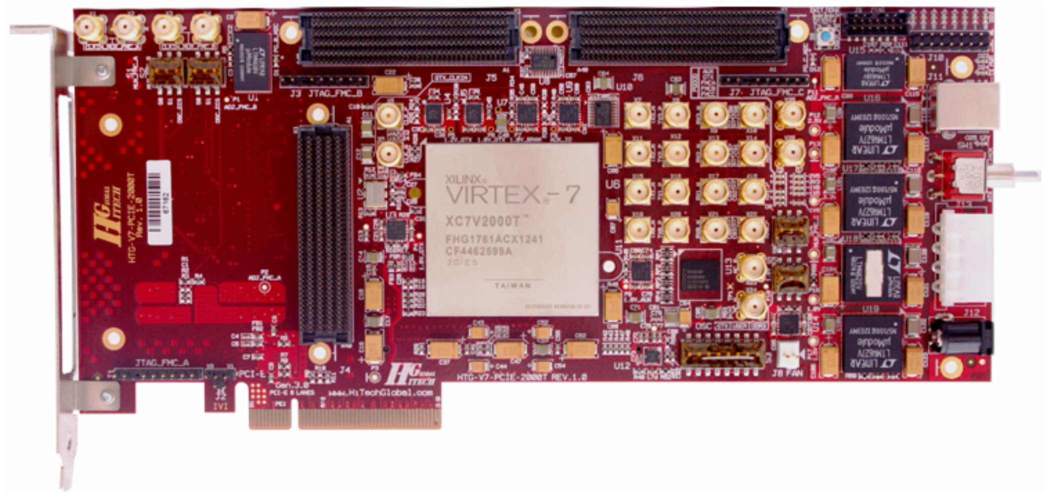


Figure 7.2: Hitech Global Virtex-7 V2000T PCI Express Development Board [14]

structure allows for distribution of clocks and the random number generator so that one structure does not drive the signals for each element.

Despite the setbacks presented by the software and the FPGA structure, a DANNA array was able to be implemented on the 2000T. Issues with the segmentation fault caused progress to stagnate slightly, but recent attempts have yielded a successful array implementation of larger arrays over 2,500 elements. However, it took several hours of run time for the arrays to synthesize, with a 50×50 array taking nine hours and a 70×70 array up to seventy-two hours. Despite the unusual development time, the success of the implementation showcases the potential of the 2000T. With the new DANNA implementation, the 2000T can support up to 5,625 elements in a 75×75 array, and the routing can be shown below in figure 7.3.

The implemented 75×75 array utilized 72% of the available look-up tables, 28% of the available flip-flops, and 20% of the available buffers on the FPGA. The total on-chip power utilization was approximated as 2.522 watts.Timing issues were more prevalent with this FPGA and design, with the worst-case setup time being -0.951 ns and the worst-case hold time being -4.5 ns. Despite the estimated timing issues reported by Vivado, testing indicates proper operation of the arrays.

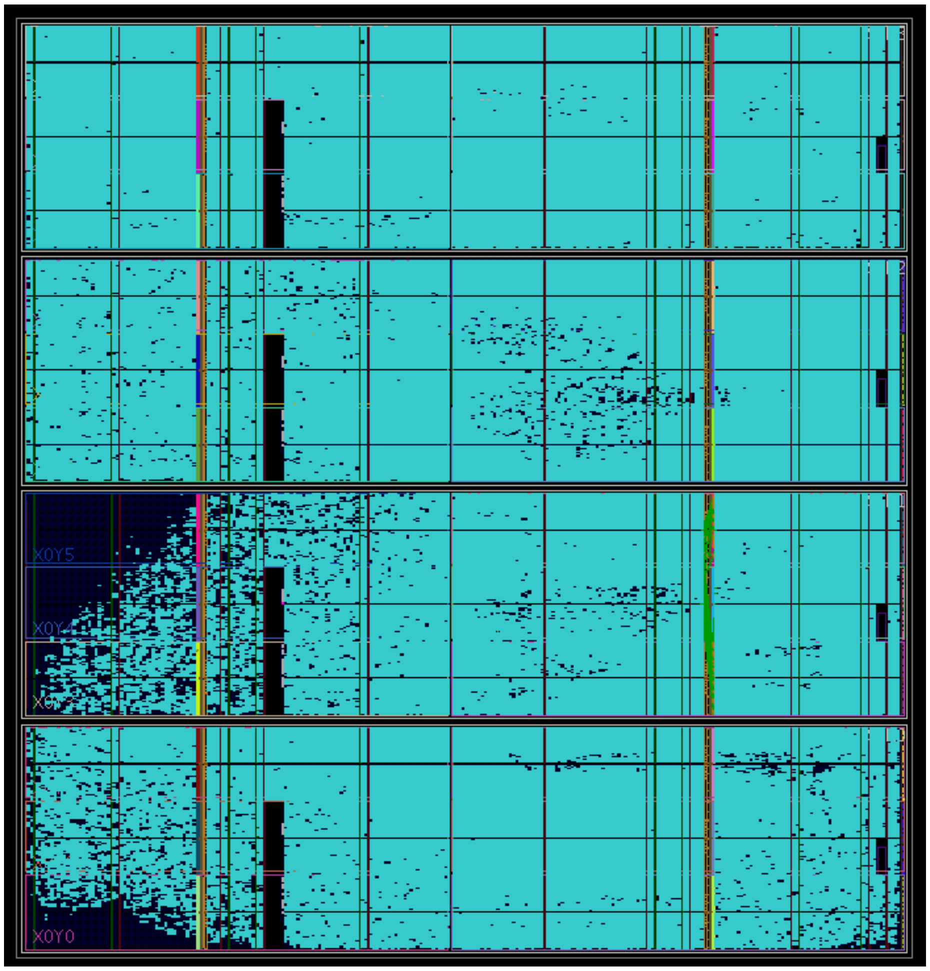


Figure 7.3: 75 × 75, FPGA Routing

7.3 Comparison with VLSI Design

A related project was conducted in parallel to the construction of the DANNA development kit. Its goal was to create a VLSI implementation of the current DANNA implementation utilizing existing CMOS technology in an effort to improve the architecture's density, speed, and power consumption [19]. Using ASIC tools, an implementation of a 75 × 75 DANNA array hosting 5,625 elements was achieved, with the resulting routing placement shown in figure 7.4. The routing placement of the same-size array highlight the differences between the VLSI implementation and the FPGA implementation. The routing done by the Vivado software for the FPGA is constrained by the physical placements of the FPGA's various logic blocks and cells. With the VLSI routing, the constraints are much more flexible, allowing for a more robust routing placement.

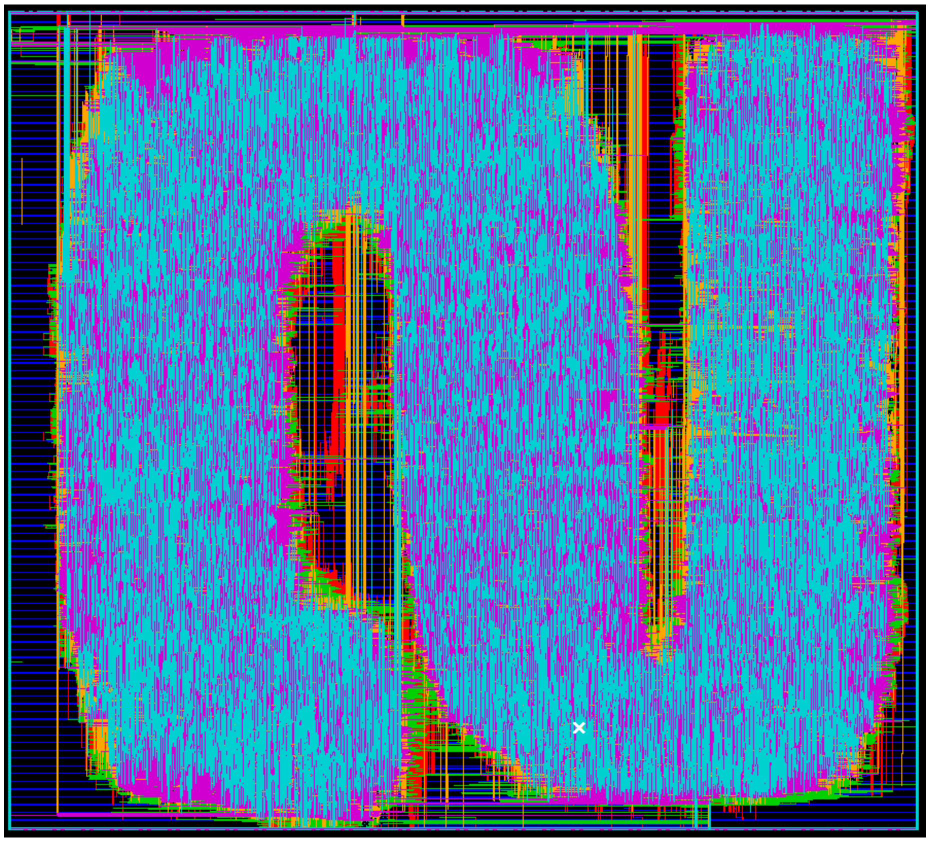


Figure 7.4: 75 × 75, VLSI Routing [19]

Chapter 8

Future Improvements

Major improvements have been made to the DANNA implementation for the development kit. However, more work can be done upon it to enhance its capabilities even further. Improvements can be made to the communications interface, the DANNA element, and the DANNA array itself.

8.1 Communication Improvement

One major improvement to the communications interface would be to improve the latency between the FX3 and the FPGA. Despite the viability shown with the USB implementation, there is still a visible latency associated with the transfers. Using a host computer with USB 3.0 SuperSpeed capability will help improve the latency when compared to USB 2.0. USB 3.0 Revision 1 allows for a maximum data rate of 10 Gigabits-per-second (Gbps) which allows for a faster rate of transfer by approximately 20 times than USB 2.0 at its max data rate of 480 Megabits-per-second (Mbps) [31]. The Cypress EZ-USB FX3 USB 3.0 peripheral controller can be easily modified to work with the enhanced protocol with little modification to the finite-state-machine on the FPGA.

Switching over to USB 3.0 would require changing the host computer board to one that can support USB 3.0. However, there are very few single board computers that currently support the protocol. With that being said, one board is being taken into consideration to replace the Wandboard. The ODROID-XU4 is a small single board computer that is capable of acting as a USB 3.0 host. It hosts a Samsung Exynos5422 Cortex-A15 and Cortex-A7 Octa core quad-core CPUs along with 2 GB of DDR3 RAM, Gigabit Ethernet, and is capable of running Ubuntu 15.04, making it more than a suitable replacement for the Wandboard Quad [8]. An image of the board is shown below in figure 8.1.

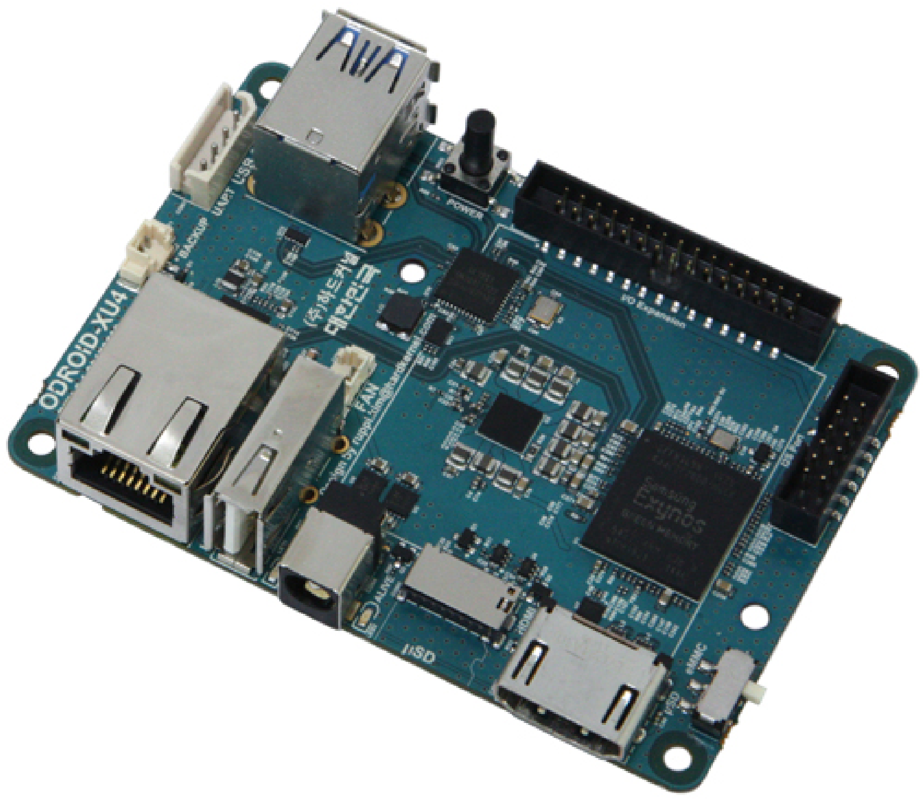


Figure 8.1: ODROID-XU4 Single Board Computer [8]

8.2 DANNA Element Improvement

An improvement to the element itself would be to expand its functionality while retaining its simplistic programming model. Other hardware platforms are known to model many complex structures of neural systems. DANNA has been shown to accomplish large, complex array structures with simple element structures, but in order to have it behave more like a biological system, increasing the element functionality is a necessity.

Expanding the versatility of the DANNA element would increase its usefulness in modeling more complex systems. The challenge lies in expanding its configurations without interfering with functionality and increasing its resources.

There are plans into configuring the DANNA element to function as a central pattern generator. Central pattern generators are neuronal circuits that produce rhythmic motor patterns without inputs that carry timing information [22]. These patterns have been shown to be attributed to behavior such as walking, breathing, and flying [22]. By incorporating such an important structure in the DANNA, the implementation would be better suited to model involuntary actions found in neural systems. Currently, there are plans to incorporate the function with little change to the element, but more work is needed before it can be confirmed as viable.

8.3 External Input Implementation

Currently, elements of the DANNA array only interact through commands. It is not set up to receive inputs directly from the outside world. Allowing it to do so would expand the versatility of the DANNA array and allow it to stand on par with other hardware platforms that implement sensors such as True North.

Incorporating the DANNA array to receive external inputs directly is relatively straightforward. The DANNA array can receive the data directly from external inputs by adjusting the implementation code to do so. The external inputs can be easily incorporated through the use of the second FMC connector on the HTG-V7 FPGA board currently in use. The amount of pins on the FMC connector would allow up to 32 inputs to be attached to the FPGA, with each external input requiring the use of nine signals, eight for weight and one to indicate a fire. This can be accomplished through the use of a breakout board for the FMC connector.

8.4 FPGA Improvements

While improvements can be clearly made with the communications interface, more work can be done on the FPGA module itself to expand the capabilities of the DANNA array. The HTG-V7 FMC-FPGA module supports Virtex-7 FPGAs up to the XC7V2000T FPGA. By replacing the 690T FPGA with a 2000T FPGA, the development kit can make use of larger arrays and expand its capabilities. While some success has been shown with the 2000T FPGA already on hand, more work can be done on it in order to improves its timing. Understanding more about how the SLR regions are utilized in the FPGA may hold the key to improve timing throughout the DANNA array.

Another approach to this is to use separate FPGA modules to construct arrays on different FPGAs and interlink them together to form one large array. A single FPGA module has already been shown to hold a large DANNA array. If multiple FPGA modules could be leveraged, they can possibly be interlinked together to form one large array. The Hitech Global Virtex 7 Quad V2000T Emulation / ASIC Prototyping Board shown in figure 8.2 contains four Virtex-7 XC7V2000T FPGAs, a Kintex-7 FPGA, eight FMC connectors, and four DDR3 SODIMM modules which fits the requirements for the proposed array design [10]. Its unique layout allows for a wide variety of Hitech Global FPGA boards to be interconnected allowing for greater expansion of FPGAs.

Xilinx has produced a new series of FPGAs utilizing a new design architecture that supersedes the Virtex-7 FPGAs. Known as the UltraScale architecture, FPGAs utilizing that architecture use both monolithic and stacked silicon interconnect technology to allow for high capacity, bandwidth, and performance to meet a vast spectrum of system requirements [34]. The Virtex Ultrascale FPGAs contain a large

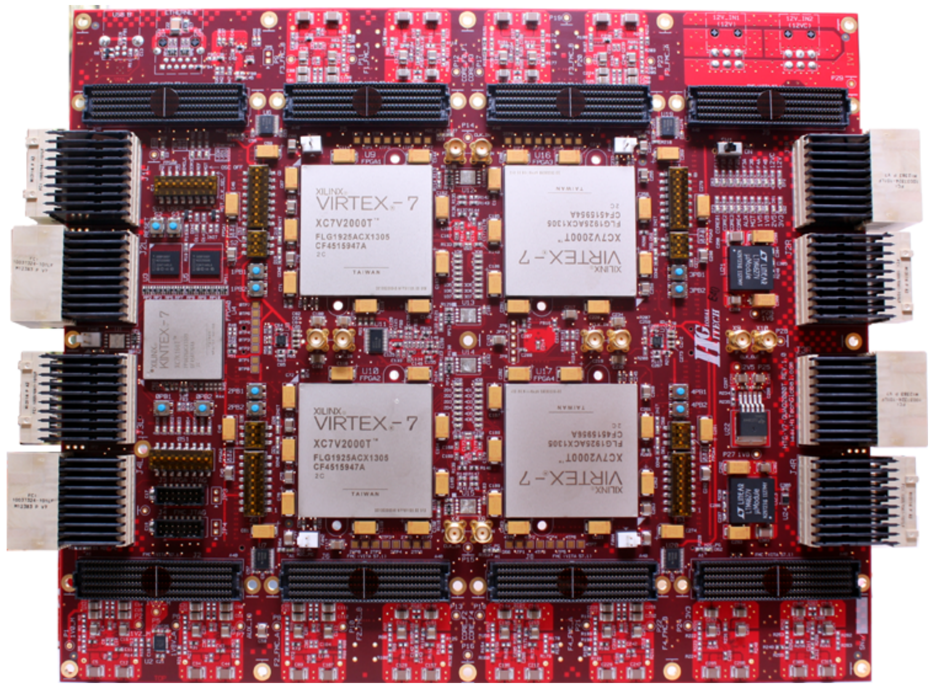


Figure 8.2: Hitech Global Virtex 7 Quad V2000T Emulation / ASIC Prototyping Board[10]

number of logic cells, flip-flops, and look-up tables, even more so than the 690T FPGA. Utilizing this new FPGA could also allow for larger array sizes. Hitech Global provides several FPGA boards that utilize the Ultrascale FPGAs. Its Virtex UltraScale 900 Gig Optical Networking Platform hosts a Virtex Ultrascale FPGA on a small $9.25in. \times 8.1in.$ board which makes it an ideal candidate for use in the DANNA development kit [11].

8.5 File Transfer Improvements

Throughout the testing of the communications interface, some issues were found regarding the size of files that could be sent out to the FX3. Files as large as 90 KB were able to be sent to the FX3 without error. However, issues occured when sending files as large as 3 MB. For some reason, during the transfer, the DMA flags all

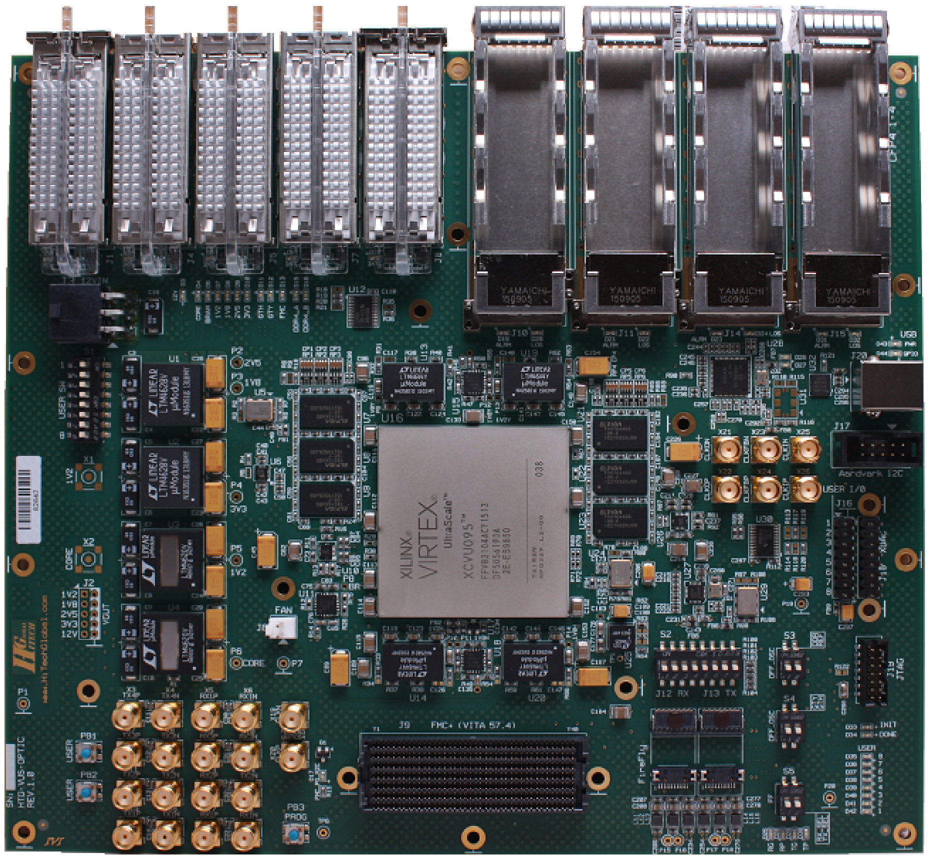


Figure 8.3: Hitech Global Virtex UltraScale 900 Gig Optical Networking Platform [11]

go high, putting the state machine in a position where it no longer reads any useful data. This error puts a limit on how large a transfer could be when performing USB transfers.

Chapter 9

Conclusion

DANNAs have been shown to function as a dynamic, neuromorphic platform capable of adapting their behavior to changing inputs and conditions. However, the previous implementation contained issues that made it difficult for other researchers to use. Creation of a DANNA development kit and the new DANNA implementation subverts the problems allowing other researchers to take advantage of neural network arrays in a portable package. While great steps have been made to make DANNA more accessible, additional work to the kit as well as the DANNA implementation itself will improve usability and capability of these neural networks.

Bibliography

[1] WinUSB (Winusb.sys). 32

[2] FrontPanel - Opal Kelly, 2013. 36

[3] Cypress EZ-USB®FX3$^{\text{TM}}$SuperSpeed Explorer Kit. Technical report, 2015. vi, 30

[4] CYUSB3ACC-005 FMC Interconnect Board for the EZ-USB®FX3$^{\text{TM}}$SuperSpeed Explorer Kit, 2015. vi, 35

[5] Hitech Global Virtex-7 FPGA FMC Module, 2015. vi, 31, 32

[6] IBM Research: Brain-inspired Chip, 2015. 3

[7] LIBUSB.info, 2015. 33

[8] ODROID-XU4, 2015. vii, 58

[9] SpiNNaker, 2015. vi, 6

[10] Virtex 7 Quad V2000T Emulation / ASIC Prototyping Board, 2015. vii, 60, 61

[11] Virtex UltraScale? 900 Gig Optical Networking Platform, 2015. vii, 61, 62

[12] Wandboard Computer, 2015. vi, 28, 29

[13] Xilinx 7 Series FPGAs Overview, 2015. 53

[14] Xilinx Virtex$^{\text{TM}}$-7 V2000T PCI Express Development Board, 2015. vii, 53, 54

[15] Xillybus, 2015. 19

[16] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, Gi-Joon Nam, B. Taba, M. Beakes, B. Brezzo, J.B. Kuang, R. Manohar, W.P. Risk, B. Jackson, and D.S. Modha. TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 34(10):1537–1557, Oct 2015. 5

[17] B.V. Benjamin, Peiran Gao, E. McQuinn, S. Choudhary, A.R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J.V. Arthur, P.A. Merolla, and K. Boahen. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proceedings of the IEEE*, 102(5):699–716, May 2014. 3, 4

[18] B.V. Benjamin, Peiran Gao, E. McQuinn, S. Choudhary, A.R. Chandrasekaran, J.M. Bussat, R. Alvarez-Icaza, J.V. Arthur, P.A Merolla, and K. Boahen. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proceedings of the IEEE*, 102(5):699–716, May 2014. vi, 4

[19] Christopher Daffron. DANNA A Neuromorphic Computing VLSI Chip. Master's thesis, University of Tennessee, 2015. vii, 55, 56

[20] Mark E. Dean, Catherine D. Schuman, and J. Douglas Birdwell. Dynamic Adaptive Neural Network Array. pages 129–141, 2014. 3, 6, 10, 14, 31, 50

[21] ECE 402 Senior Design Group. Neuromorphic Hardware using a FPGA, May 2014. 2, 19

[22] Eve Marder and Dirk Bucher. Central pattern generators and the control of rhythmic movements. *Current Biology*, 11(23):R986 – R996, 2001. 59

[23] Chris McClelland. FPGALink, 2013. 36

[24] E. Painkras, L.A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D.R. Lester, A.D. Brown, and S.B. Furber. SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation. *Solid-State Circuits, IEEE Journal of*, 48(8):1943–1953, Aug 2013. 3, 5, 7

[25] J. Schemmel, D. Bru?derle, A. Gru?bl, M. Hock, K. Meier, and S. Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 1947–1950, May 2010. 3, 6

[26] Catherine D. Schuman. *Neuroscience-Inspired Dynamic Architectures Dissertation Defense*. PhD thesis, University of Tennessee, November 2014. vi, 8

[27] Catherine D. Schuman and J. Douglas Birdwell. Dynamic Artificial Neural Networks with Affective Systems. *PLoS ONE*, 8(11):e80455, 11 2013. 11, 12

[28] C.D. Schuman, J.D. Birdwell, and M. Dean. Neuroscience-inspired inspired dynamic architectures. In *Biomedical Science and Engineering Center Conference (BSEC), 2014 Annual Oak Ridge National Laboratory*, pages 1–4, May 2014. 7, 8

[29] Cypress Semiconductor. AN65974 - Designing with the EZ-USB®FX3™Slave FIFO Interface. Technical report, 2015. 36, 37

[30] V. Thanasoulis, J. Partzsch, B. Vogginger, C. Mayr, and R. Schuffny. Long-term pulse stimulation and recording in an accelerated neuromorphic system. In *Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on*, pages 590–592, Dec 2012. 6

[31] USB.org. *Universal Serial Bus 3.1 Specification*, rev. 1 edition, July 2013. 57

[32] Xilinx. Large FPGA Methodology Guide. Technical report, 2012. 53

[33] Xilinx. 7 Series FPGAs Overview. Technical report, 2015. 50

[34] Xilinx. UltraScale Architecture and Product Overview. Technical report, 2015. 60

Vita

Jason Chan is from Knoxville, TN. He graduated from Farragut High School in 2010, and he continued his studies at the University of Tennessee in Knoxville, pursuing a Bachelor of Science degree in Electrical Engineering from the Department of Electrical Engineering and Computer Science in the College of Engineering. He graduated with this degree in May of 2014. In the same month, he began pursuing a Master of Science degree in Electrical Engineering at the University of Tennessee. He graduated with that degree in December of 2015.