ABSTRACT

Increased Computation Using Parallel FPGA Architectures

Stephen L. Dark, M.S.E.C.E.

Mentor: Russell W. Duren, Ph.D.

Two ways to improve algorithm performance in hardware are increasing the speed of each operation, or performing multiple operations simultaneously. However, the percent speed-up for the latter depends upon not only system constraints but also design decisions. When using multiple FPGAs as the implementation target, creating an optimal configuration requires the designer to be aware of many potential issues. A neural network inversion case study is presented in order to give future FPGA algorithm designers insight into the possible problems arising from parallel FPGA implementations. Initial work is performed implementing a large Neural Network and finding its inversion via Particle Swarm Optimization on a single FPGA. This algorithm is later broken up and performed in parallel with multiple FPGAs using several strategies on various hardware and software architectures. At the end, a discussion of the potential issues that arose during these implementations is presented along with some generalized guidelines.

Increased Computation Using Parallel FPGA Architectures

by

Stephen L. Dark, B.S.E.C.E.

A Thesis

Approved by the Department of Electrical and Computer Engineering

_____

Kwang Y. Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Master of Science in Electrical and Computer Engineering

Approved by the Thesis Committee

_____

Russell W. Duren, Ph.D., Chairperson

_____

Michael W. Thompson, Ph.D.

_____

Gregory J. Hamerly, Ph.D.

Accepted by the Graduate School
December 2010

_____

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ASIC – Application Specific Integrated Circuit

BSB – Base System Builder

CF – Compact Flash

CORDIC – COordinate Rotation DIgital Computer

COTS – Commerical off the Shelf

EDK – Embedded Development Kit

FPGA – Field Programmable Gate Array

GUI – Graphical User Interface

IC – Integrated Circuit

DMA – Direct Memory Access

IC – Integrated Circuit

IP – Intellectual Property

LabVIEW – Laboratory Virtual Instrument Electronic Workbench

LUT – Look-up Table

NN – Neural Network

OPB – On-Chip Peripheral Bus

P2P – Peer to Peer

PCB – Printed Circuit Board

PCI – Peripheral Component Interconnect

PCIe – Peripheral Component Interconnect Express

PLB – Processor Local Bus

PPC405 – Power PC 405

PSO – Particle Swarm Optimization

PXIe – PCI eXtensions for Instrumentation Express

RISC – Reduced Instruction Set Computer

SCTL – Single Cycle Timed Loop

SERDES – Serializer/ Deserializer

SIR – Signal to Interference Ratio

SoC – System on a Chip

UART – Universal Asynchronous Receiver Transmitter

UI – User Interface

VHDL – VHSIC Hardware Description Language

VHSIC – Very High Speed Integrated Circuit

VI – Virtual Instrument

XUP – Xilinx University Program

XUP-V2P – Xilinx University Program – Virtex-II Pro

ACKNOWLEDGMENTS

I would like to thank Dr. Duren for providing an interesting and challenging thesis topic. Thanks also to my thesis committee for their continued patience while I worked remotely. Finally, thanks to my parents and fiancée Chelsea, who were always supporting me with their encouragement and love.

DEDICATION

To my mom who was always my biggest fan

CHAPTER ONE

Introduction

Creating very large serial and parallel hardware systems is something that has become well understood in the last few decades especially with the advent of large FPGAs. Larger and larger amounts of customizable hardware logic have enabled hardware engineers to solve an ever increasing array of problems that could not have been previously solved. Engineers now have the ability to customize their designs into entirely serial architectures, entirely parallel architectures, and architectures anywhere in between depending on design needs and constraints. However, there are many times when system complexity is so large that even the fastest and largest FPGAs on the market cannot be leveraged by themselves to complete the task. This is not a challenge that will be resolved in the future with ever increasing amounts of faster logic; this is an ever evolving challenge that will always grow to include tomorrow's unsolvable needs.

More importantly, the silicon IC industry may not be able to sustain its current problem solving progress with current manufacturing limitations on the horizon. Decreasing the computation time of various algorithms has become expected since the invention of the transistor. According to Moore's Law, the number of transistors on a chip will double about every two years. However, the time is nearing that this self-fulfilling prophecy will end, at least temporarily. As of the writing of this thesis, state-of-the-art silicon devices are fabricated using 28nm feature sizes, whereas the size of a single atom is about 1nm. Moore's Law shows no signs of stopping until we reach this

atomic barrier. However, once this barrier arrives, we will be forced to find other methods of increasing computational performance until the next new technology has matured to restart Moore's Law and resurrect the trend. Multiple parallel FPGAs tightly coupled with multi-core microprocessors offer one potential solution.

Modern desktop microprocessors have already made strides towards using parallel architectures as a method of increasing computational performance for general purpose computing. FPGAs offer an additional method for increasing the computational performance of systems by creating custom high speed hardware logic tailored to each design problem. There is a new trend developing that shows the merging of FPGA technology with microprocessors. All of the major FPGA companies have current product offerings that support hard IP (fixed Intellectual Property within silicon) for multipliers, RAM, DRAM, PCIe, microprocessors, etc. In addition, a few embedded microprocessor companies have started developing products that feature a large microprocessor with programmable logic.

As hardware computation moves into the future, hardware computational architectures will become increasingly more complicated and more parallel. In the areas of cutting-edge research and military applications, computational architectures will not have a choice but move to frameworks that include multiple systems. Regardless of the problem, parallel architectures are here to stay, and it will be left to the system designers to create the most optimal system possible. With the state of currently emerging hybrid technology creating ever more complex systems, one of the largest problems facing system designers will be how to effectively and efficiently program these complicated hardware architectures. Questions naturally arise about how best to segment algorithms

between multiple processors and multiple FPGAs, what is the best system topology for a given design problem, what are the best ways to move vast amounts of data for processing, and what are the tradeoffs for cost and time. It is the goal of this thesis to begin to answer some of these questions by using different parallel hardware platforms to implement a large semi-parallel algorithm, real time neural network inversion.

Structure Overview

In order to answer some of the proposed questions, this thesis presents its findings as a result of developing a large neural network inversion algorithm on an FPGA(s) using two different hardware architectures and with two drastically different development environments. In addition, each of these systems implemented the algorithm using different configurations of the hardware with different segmentation strategies of the original algorithm in order to speed up its computation.

Chapter two presents the neural network inversion problem that will be solved throughout the rest of this text using different methods. Following chapter two, chapter three gives a detailed account of the different hardware architectures and their respective capabilities. For each of the hardware architectures presented in chapter three, there is a corresponding different development environment that is described in chapter four. The conclusion of chapter four presents a comparison of the two development environments and their respective advantages and disadvantages.

Beginning in chapter five, a discussion of the different possible algorithm segmentation strategies is presented. The remainder of chapter five describes the mapping of these segmentation strategies to different hardware configurations and the high level system details of each system configuration while chapter six gives a low-level

detailed explanation of the different hardware implementations. Finally, all of the different hardware architectures, hardware configurations, and segmentation strategies join together in chapter seven in order to present what worked best both through analysis and through collected experimental data with final conclusions given in chapter eight.

CHAPTER TWO

The Algorithm: Neural Network Inversion

This chapter presents the acoustic Neural Network (NN) developed in [1] and its inversion via the Particle Swarm Optimization (PSO) algorithm used in [2] that will be implemented using FPGA hardware in later chapters. Neural network inversion makes an appropriate case study because it is neither completely serial nor completely parallel thus it is neither obtuse nor trivial. In addition, it is very computationally intensive and repetitive making it a prime candidate to implement on FPGAs. Using this algorithm as a case study, potential issues that arose from moving this algorithm to a parallel FPGA hardware implementation were exposed. Through this exposure, guidelines for parallel FPGA architectures were developed.

Problem Statement

In Naval sonar systems, it is often desired to have optimized visibility of a particular underwater region. However, in many cases underwater regions of interest are the same regions that are least visible due to either environmental conditions or the sonar system parameters. Shown in Figure 1, is a sonar probe that is capable of imaging its nearby region or “Surveillance Area”. Within this surveillance area, there is a particular target area of interest that needs high sonar visibility. The goal is to find an optimized set of sonar system parameters that produces the highest visibility of the desired underwater region in real-time.

The task of finding an optimized set of input sonar system parameters is accomplished in two parts. First, a feed forward neural network is used to model the Signal to Interference ratio (SIR) of the underwater surveillance area [1]. Second, the

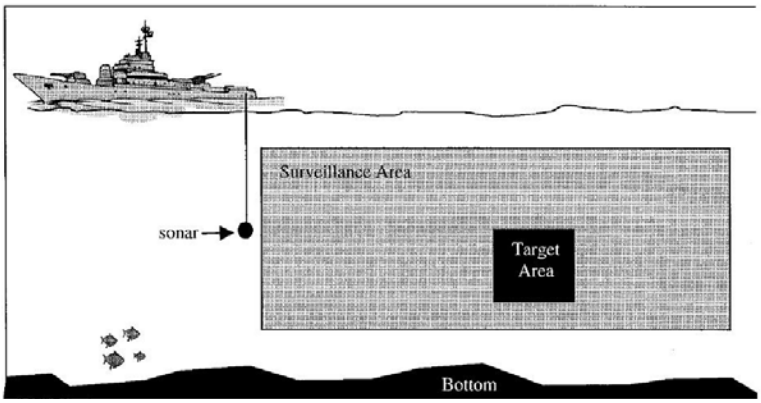


Figure 1: Problem Objective [3]

Particle Swarm Optimization (PSO) search algorithm is used to effectively invert the feed forward neural network, by iterating on the neural network to find the set of inputs that best optimizes the target area [2].

The Acoustic Neural Network

Traditional computer models of the ocean's water profile are computationally intensive. However, previous work has been accomplished training a feed forward neural

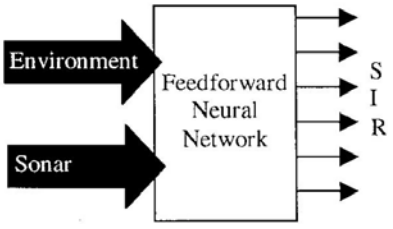


Figure 2: Neural Network I/O [3]

network to perform the same task with much less computational time and intensity shown in Figure 2 [1]. The structure of the proposed neural network consisted of an input layer with twenty-seven input nodes, three hidden layers using, 40 nodes, 50 nodes, and 70 nodes respectively, and an output layer with 1,200 output nodes illustrated in Figure 3. The twenty-seven inputs represent the sonar system parameters and environmental parameters. The sonar system parameters are controllable variables such as center

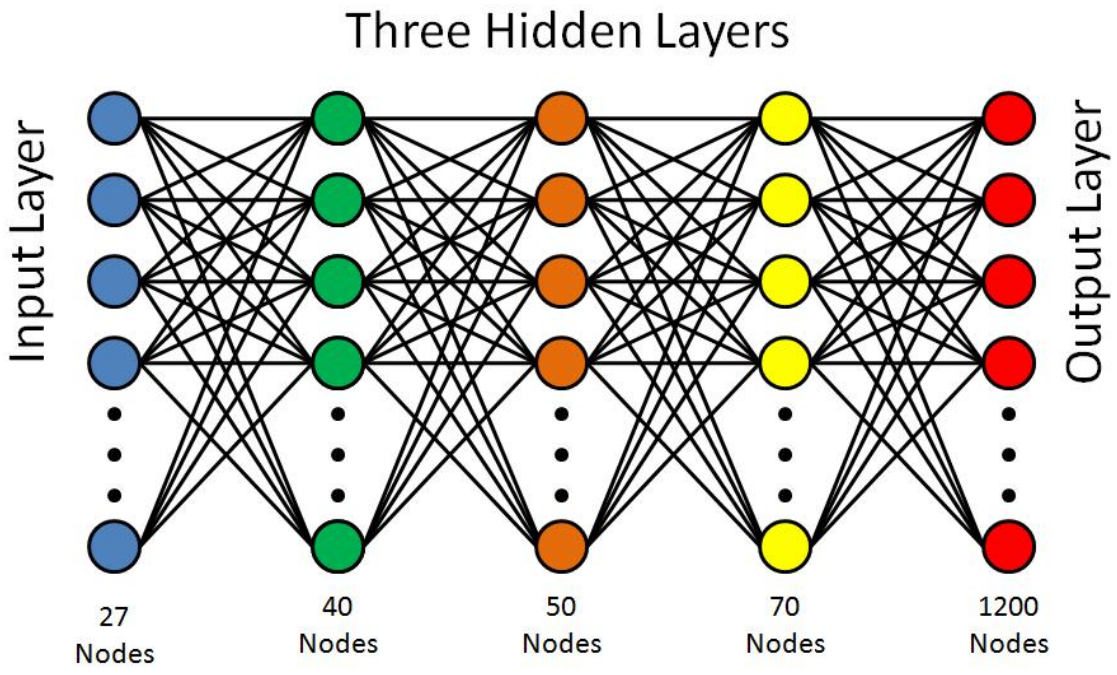


Figure 3: Neural Network Internal Structure

frequency, bandwidth, and vertical steering angle while the environmental parameters are uncontrollable variables such as wind speed, salinity, bottom type, and sound speed. The three hidden layers are design parameters of the neural network, and the 1,200 outputs of the neural network modeled the ocean water profile's Signal to Interference Ratio in dB at points on an 80 by 15 grid. Thus, given the twenty seven sonar and environmental

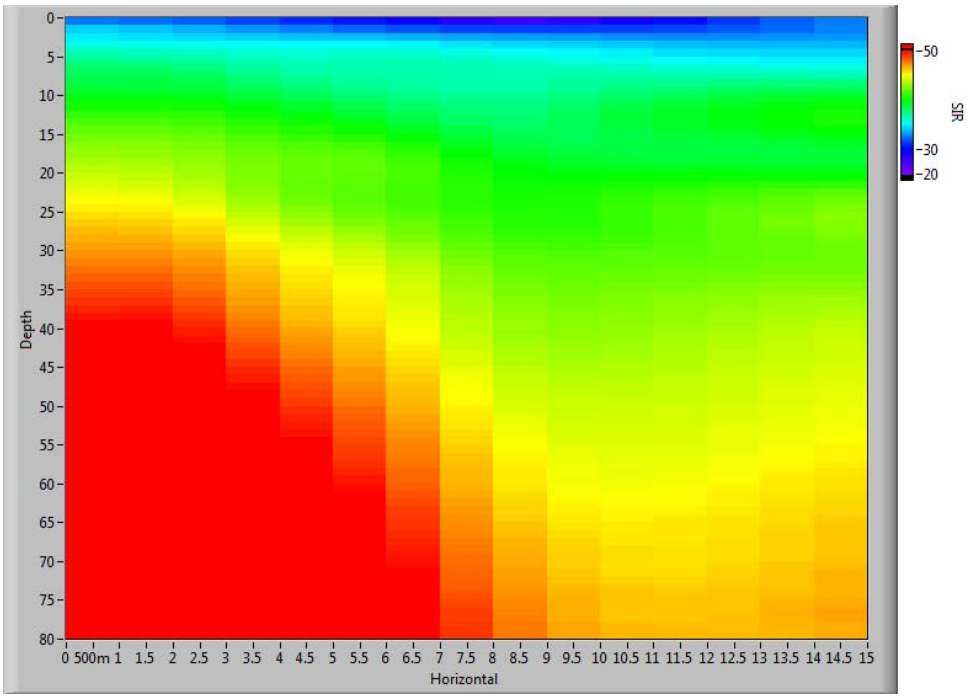


Figure 4: Underwater SIR Profile

parameters as inputs, an 80 by 15 pixel image is obtained of the surveillance area of Figure 1 as demonstrated in Figure 4.

Particle Swarm Optimization

Finding optimized or near optimal input parameters for the "Target Area" of Figure 1 implies finding an inverse of the neural network. The search algorithm used for this case study was Particle Swarm Optimization (PSO). While PSO is not guaranteed to find the optimal solution, it has reliably produced sufficient solutions for the neural network inverse.

In PSO, multiple particles iteratively search throughout the input space to find the best location by either maximizing or minimizing a prescribed fitness function. At algorithm startup, each particle starts out at a randomized location. Then at each

position, each particle calculates its value according to the prescribed fitness function. After every particle has found its value at its current position, the local best value (each

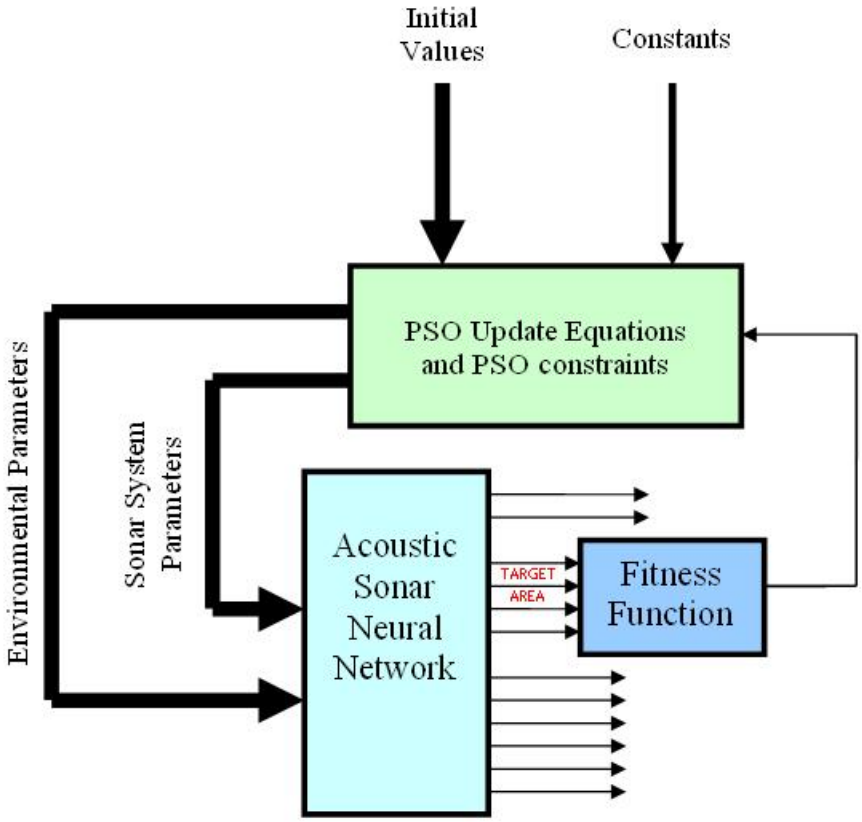


Figure 5: PSO Block Diagram

particle's best value across all iterations) and global best value (the best value of all of the particles across all iterations) are updated by maximizing or minimizing its fitness value.

Finally, each particle updates its position for the next iteration according the equations (2.1) and (2.2) where X[n] is the particle's current position, X[n+1] is the particle's next

$$X[n+1] = X[n] + V[n] \tag{2.1}$$

$$V[n+1] = V[n] + C_L * (X_L - X[n]) + C_G * (X_G - X[n]) \tag{2.2}$$

position, V[n] is the particle's current velocity and V[n+1] is the particle's next velocity. X_L is the local best position, the best location each particle has obtained in its entire iterative history, and X_G is the global best position, the best location of all particles over the iterative history of every particle. Lastly, the constants C_L and C_G, the best local position weight and best global position weight respectively, are used to weight the importance of the local and global best positions and are design parameters of the PSO algorithm. This calculation procedure is depicted as a block diagram in Figure 5.

In the instance of the inverse neural network case study, it is desirable to maximize the Signal to Interference Ratio (SIR) of the target area. Therefore, the prescribed fitness function used for this problem was the L1 norm taken over the target area given by equation (2.3). In addition to the fitness function and update equations,

$$Fitness(\overrightarrow{SIR}) = \left\| \overrightarrow{SIR_{\substack{TARGET \\ AREA}}} \right\| = \sum_{N=\substack{TARGET \\ AREA}} SIR(N) \tag{2.3}$$

there are constraints on the particle swarm algorithm. This includes maximum and minimum values for the particle's position so that particles cannot travel outside of the search space. In addition, there are maximum values on the velocity of the particles in order to keep particles from jumping over and bypassing optimum solutions caused by steep peaks in the search space. In the case that specific inputs are fixed or clamped to certain values (this is the case with the environmental input parameters of the neural network), setting the maximum velocity to zero in the appropriate dimension will cause the particles to not update this dimension.

Hardware Adaptation

Adaptation of the neural network inversion problem to hardware suffers from several possible errors that create differences between the software and hardware implementations. Converting from the conventional floating-point arithmetic to fixed-point arithmetic is one source of possible error. The precision of the fixed point representation suffers from truncation and rounding effects as compared to its floating-point counterpart. Because of the large number of logic resources needed to implement floating-point calculations on FPGAs, floating-point implementations are never used and will most likely not be used for the foreseeable future [4]. However, simulations in [5] verify only a marginal decrease in numerical precision as a result of using a fixed-point representation. In addition, it has been shown that 16-bits of information have sufficient precision for all hardware calculations [5].

Another source of error occurs when implementing the non-linear neural network squashing function. Because of the lack of non-linear mathematical functions in hardware, another low error method for calculating the squashing function must be found. It has been found that a lookup-table produces the best accuracy and lowest latency [5]. While a lookup-table consumes far more memory than the other implementations, there is enough memory remaining to accomplish the task.

The last consideration when making the hardware adaptation is how to create a stochastic parameter used in the particle swarm optimization algorithm. It is known that adding a small amount of random behavior to the update equations increases the quality of the best global location. However, after many simulations in [6], it was found that adding a random function generator to the calculations showed only a marginal increase

in finding a better optimized value. As a result, adding a random number generation function was not included in the hardware implementation.

CHAPTER THREE

Hardware Architecture

The implementation of the inverse neural network algorithm was performed on two different hardware architectures. The first architecture used the Xilinx University Program (XUP) Virtex-II Pro (XUP-V2P) development system as a stand-alone system. This is a low cost system that features many different peripherals that connect directly to the Xilinx Virtex-II FPGA including three high speed duplex serial links for inter-board communication. The second system used the PXIe platform and was composed of National Instruments' hardware. This is a modular platform that interfaces to a general purpose computer through the PXIe bus. At the heart of this system is the Flex-RIO PXIe-7965 that contains a Virtex-5 FPGA that connects to the PXIe bus.

XUP Virtex-II Pro Development System

The Xilinx University Program (XUP) Virtex-II Pro (XUP-V2P) Development System is a hardware platform consisting of the Virtex-II Pro FPGA and many peripheral components. These peripheral components as seen in Figure 6 include a serial RS-232 DCE connection, Compact Flash drive, Serial ATA ports, DDR SDRAM interface slot, 10/100 Ethernet port, and various push buttons, switches, and LEDs. Since this hardware architecture does not connect to a traditional general purpose computer, the entire user interface (UI) has to be built using the peripherals on the XUP-V2P. While this is a burden to the system developer, this architecture allows a fully custom UI while also keeping down the total system cost. The high performance of the Virtex-II Pro FPGA

coupled with the flexibility and low cost of the XUP-V2P system make it a suitable platform for testing and optimizing parallel hardware configurations.

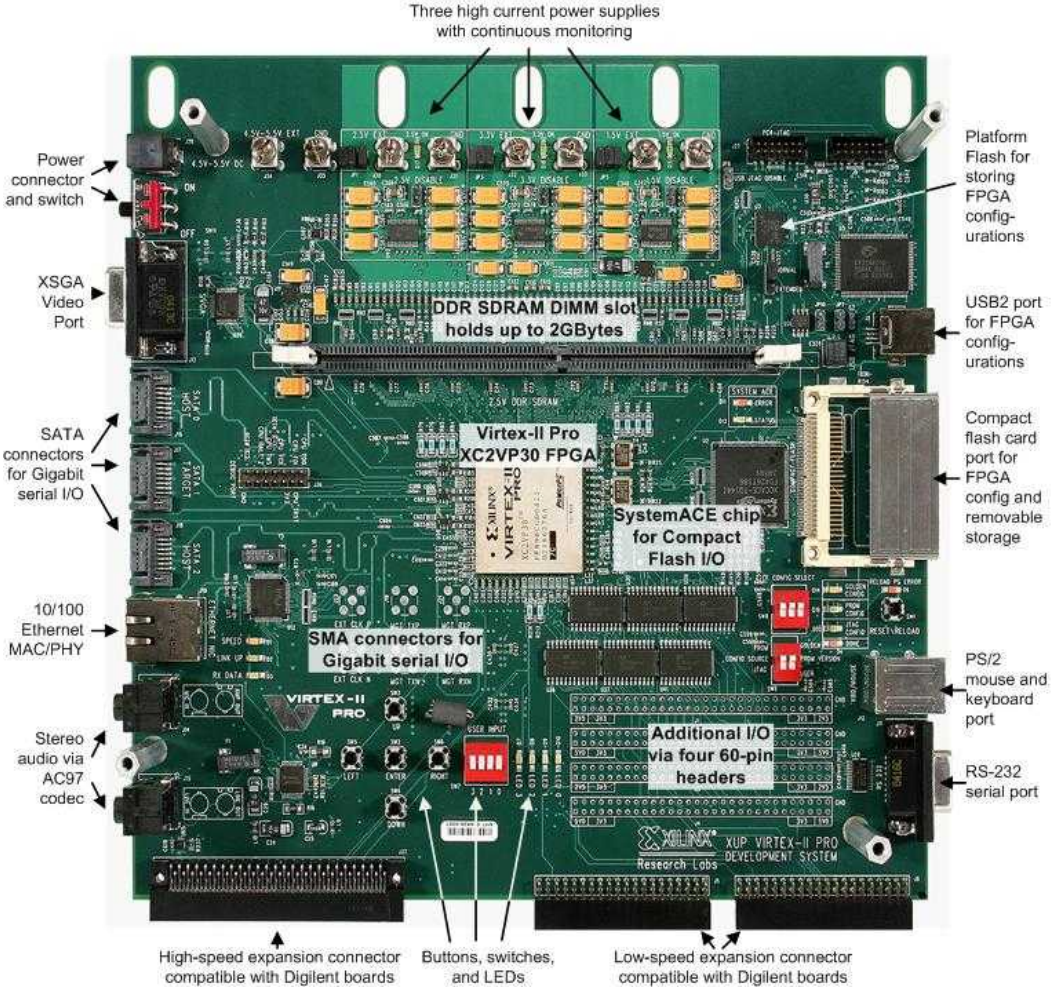


Figure 6: XUP-V2P Development System [7]

Serial Port

The serial port on the XUP-V2P is the primary UI to the system. The XUP-V2P system includes a DB-9 serial connector that connects to a host PC's serial COM port via a straight-through nine pin cable. This enables the end user to interact through the PC using standard Hyper Terminal connection software. On the XUP-V2P PCB, the DB-9 first interfaces through a MAX3388E to condition the voltage levels; the conditioned

signals connect directly to the Virtex-II Pro FPGA where the Universal Asynchronous Receive Transmit (UART) hardware IP core is instantiated as the standard in/out of the PowerPC 405. This allows for easy I/O between the user and standalone executables on the PowerPC 405 when using C commands such as printf. Using this I/O interface, a menu system was implemented to create the UI.

Compact Flash Drive

File I/O is accomplished with the Compact Flash (CF) drive. ASCII based files can be read and written to and from any CompactFlash type-II card or IBM Micro-Drive. This allows for a very convenient means of non-volatile file storage. In addition, the CF drive is used as a means of configuring the Virtex-II Pro fabric and initializing both on (block RAM) and off chip (DRAM) memory. This is accomplished with use of the chipset's System Advanced Configuration Environment (System ACE). Most importantly, the CF drive is used to store the final neural network inputs that result in the optimized target area in order to visualize them on a PC.

Serial ATA

The various parallel XUP-V2P configurations make use of the Serial Advanced Technology Attachment (SATA) electrical characteristics for inter-board high speed full duplex communications. The XUP-V2P system boasts two host SATA connectors and one target SATA connector. This allows for direct communication between the host SATA connector of one XUP-V2P system and the target SATA connector of another. In effect, one XUP-V2P with its three connectors can potentially connect to three other systems directly, or a ring structure can be formed connecting almost an unlimited

number of XUP-V2P systems. The SATA connector connects to the Virtex-II Pro's specialized Rocket IO Multi-Gigabit Transceivers (MGTs) optimized specifically for high speed serial transmissions. Inside of the FPGA fabric, the Aurora protocol IP, developed by Xilinx and discussed in chapter five provides a high speed and low latency interface between the logic used to implement the neural network inversion algorithm inside the Virtex-II Pro fabric and the FPGA's Rocket IO MGTs.

DDR SDRAM

While the XUP-V2P system can support up to 2GB of DDR SDRAM, capacities of only 256 MB are implemented throughout all the XUP-V2P hardware configurations. The DDR SDRAM is connected directly to the Virtex-II Pro where a DDR SDRAM controller is instantiated in the FPGA fabric. This memory is used both as stack/heap storage and program storage.

Xilinx Virtex-II Pro

The Xilinx Virtex-II Pro FPGA comes in many different varieties. The XC2VP30-7 FF896 C is the specific model that is included with the XUP Virtex-II Pro Development System. This model contains two PowerPC 405s, 136 18Kb Block RAMs, 136 18X18 bit multipliers, eight Rocket IO Transceivers, 27,392 four input Look Up Tables (LUTs) and 27,392 flip flops.

PowerPC 405

The PowerPC 405 embedded processor is a RISC CPU with a five stage pipeline (fetch, decode, execute, write-back, and load write-back) with many functional units seen in Figure 7. The PowerPC 405 includes thirty-two 32-bit general purpose registers,

big/little endian support, separate non-blocking instruction and data cache units, an interrupt pin that connects to an external interrupt controller, and various debug support including both internal and external. Most importantly, the PowerPC 405 has a Processor

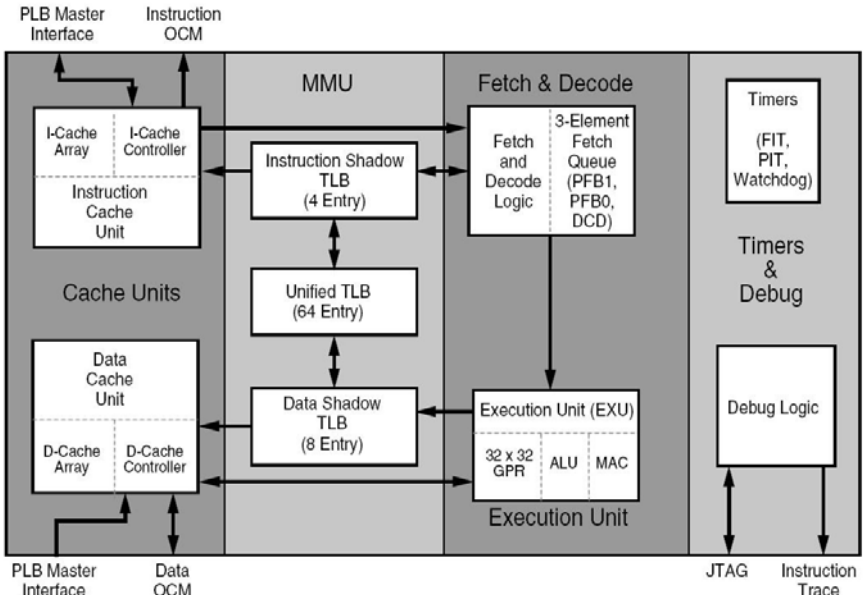


Figure 7: Power PC 405 Processor [8]

Local Bus (PLB) master interface. This interface bus is part of the IBM CoreConnect Bus Architecture that also includes the On-Chip Peripheral Bus (OPB) and the Device Control Register (DCR) bus. The CoreConnect Bus Architecture provides a standard interface for the connection of the many peripherals instantiated in the Virtex-II Pro FPGA fabric to the Power PC 405.

Block RAM

Block RAMs on the Virtex-II Pro can be configured for many purposes. When using the PowerPC 405, different Block RAMs can be configured to contain instruction memory, instruction cache, and data cache. Aside from the PowerPC 405, Block RAMs can be configured as general purpose RAM, or in the case of Neural Networks, Block

RAMs serve the dual purpose of becoming ROMs. This allows for on the chip storage and therefore quick access time of weight information needed in the neural network calculation.

Multipliers

The 18X18 bit multiplies take as inputs two 18-bit values and outputs a 36-bit product. This is a synchronous operation that completes in one clock period. Accessing the multipliers requires instantiating the built in Xilinx primitives. Simply multiplying two values in the HDL does not make use of these one clock operations, but instead synthesize to something much slower using slice logic.

Rocket IO MGTs

The eight Rocket IO MGTs allow for high speed serial communications. Of the eight available, only three can be used with the XUP-V2P system out of the box. This is because only three of the MGTs are connected to the SATA connectors on the XUP-V2P PCB. Each Rocket IO MGT is capable of operating between 622 Mb/s and 3.125 Gb/s. In addition, each MGT self-contains its own clock generator, clock recovery circuitry, 8B/10B encoder/decoder, and 3.125 Gb/s serializer/deserializer.

PXIe Hardware Architecture

The National Instruments PXIe hardware architecture was the second hardware architecure used for implementing the neural network inversion algorithm. This architecture is a modular architecture that allows many different types of modules to be connected in a seamless fashion via the PXIe bus and software running on a PC. Three different modules are used to create a suitable hardware architecture for performing the

neural network inversion computation. These three modules are the PXIe-7965 Virtex-5 processing card, the PXIe-1075 chassis, and the PXIe-8133 controller. The modular architecture of this system and seamless integration with a general purpose computer creates many possibile configurations, making it a suitable platform for parallel computation implementations.

PXIe-7965R FlexRIO

The PXIe-7965R FlexRIO pictured in Figure 8 contains a single Virtex-5 SX95T FPGA that connects to the PXIe bus via a PCIe physical interface ASIC. This connection supports both memory mapped registers and DMA. When using DMA to transfer data,

Figure 8: PXIe-7965R [9]

throughput rates up to 800MB/s are achievable. In addition, when multiple PXIe-7965R's are present within a single system, the PXIe-7965R's can transfer data directly between each other using DMA without the need to use the host computer's memory

controller. This feature is referred to as Peer-to-Peer streaming (P2P) and is used to perform the neural network inversion algorithm in parallel with multiple PXIe-7965Rs. In addition, the PXIe-7965R also contains 512MB of on-board DDR-SDRAM and pins out 132 nets to the front panel of the device.

PXIe-1082 Chassis

The PXIe-1082 is an eight slot chassis used to interconnect the various modules used in the PXIe hardware architecture. Slot one of the chassis supports either an embedded PC controller or a PXIe-PCIe8371 that connects to an external general purpose PC with PCIe x4 capabilities. All slots other than slot one can be used for peripheral

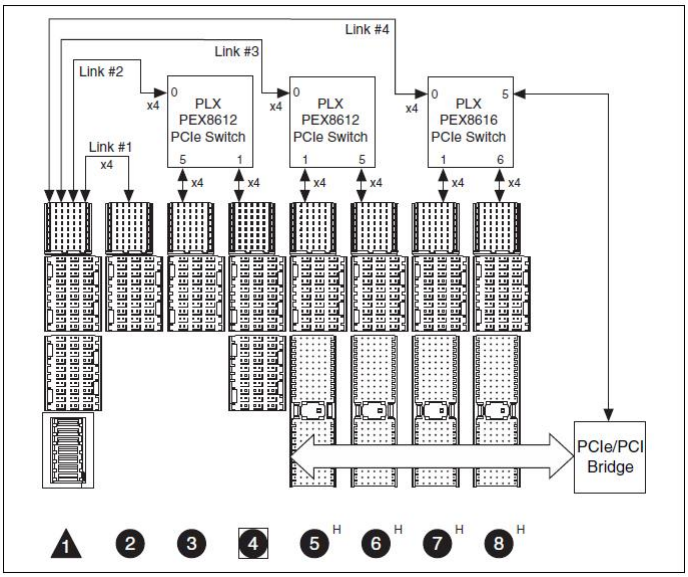


Figure 9: PXIe 1082 Backplane [10]

modules. For the inverse neural network implementation, the PXIe-8133 controller was used in slot one as the general purpose PC embedded controller, and the PXIe-7965Rs were placed in slots five and six. The PXIe-7965Rs were placed in these slots in order to keep the modules interfacing through the same switch as seen in Figure 9. This

minimizes the bus latency since it removes the need for the communication to a switch on the PC.

PXIe-8133 Controller

The PXIe-8133 controller shown in Figure 10 is an embedded industrial PC used to configure the hardware and perform the needed UI functionality. This is a fully functioning PC running Windows XP Professional with keyboard, video, and mouse user interfacing capabilities that is also capable of interfacing to the PXIe bus. Also installed

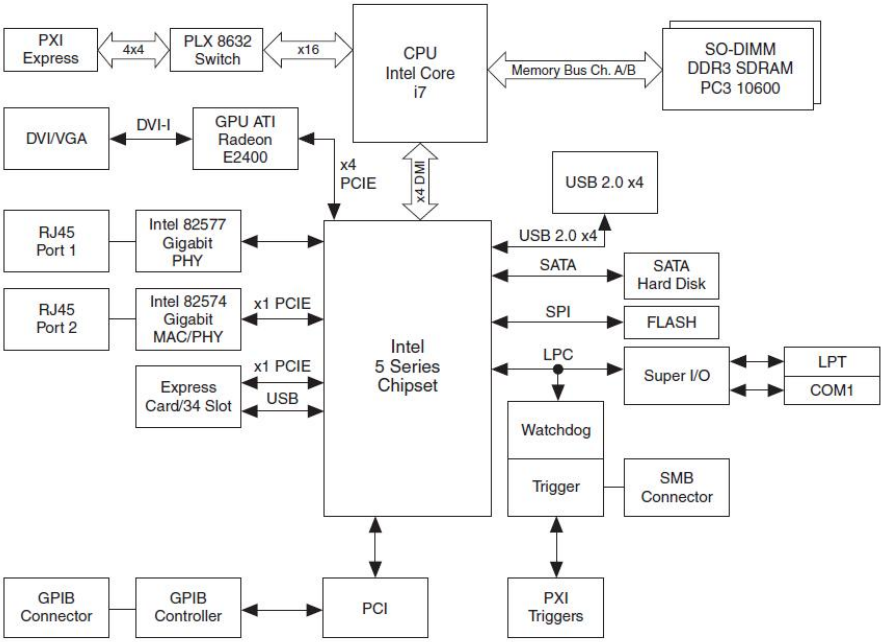


Figure 10: PXIe-8133 Block Diagram [11]

on this system is LabVIEW 2010 including the FPGA module, and NI-RIO 3.5.1. By utilizing commercial off the shelf (COTS) software, a rich user interface is easily achievable very quickly. In addition, the hardware bus interfaces are abstracted away from the developer allowing the developer more time to focus on the algorithm implementation.

CHAPTER FOUR

Development Environments

Two different development environments were used to implement the neural network inversion algorithm on the two different hardware platforms discussed in chapter three. The XUP-V2P hardware architecture used the Xilinx Embedded Development Kit (EDK) as the development environment in order to implement both the system level hardware configuration and the software development, while the PXIe architecture made use of LabVIEW for the software development and LabVIEW FPGA for developing the FPGA algorithm. This chapter gives an overview of both of the development environments and ends with a comparison of the two different environments.

EDK Development Environment

Xilinx EDK is used to quickly build complete hardware and software systems by creating an environment that leverages and interconnects various pieces of Xilinx IP. As a result, with the addition of the PowerPC on the Virtex-II Pro FPGA, EDK allows for the development of entire hardware and software System on a Chip (SoC) embedded designs. When starting with EDK, the EDK project can be created from scratch where the user can build a system using the available EDK IP. However, the recommended starting path makes use of the Base System Builder (BSB) to create the initial EDK project.

The BSB is an EDK wizard that steps through the many hardware configuration options once given information about the hardware target as shown in Figures 11 thru 13. In the case of the XUP-V2P architecture, a board definition file was included with the hardware that allowed for an initial system to be built using the BSB. Once the hardware

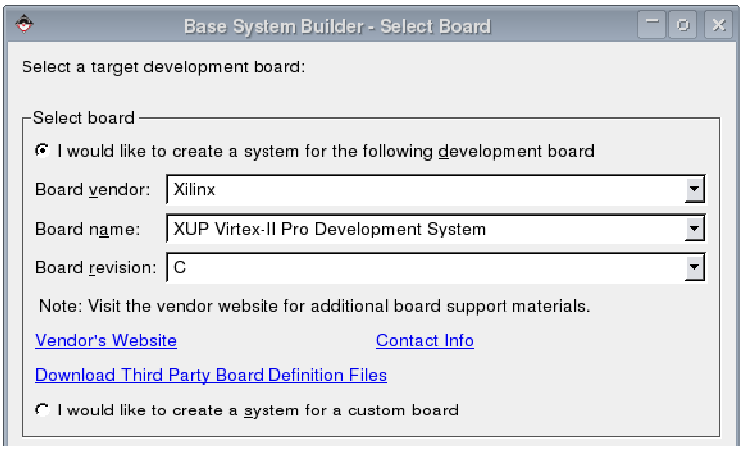


Figure 11: Base System Builder Board Specification Dialog

target was specified as shown in Figure 11, the BSB steps the user through the configuration pages. The first series of configuration pages ask the user to select which peripherals to instantiate in hardware and the configuration for each respective peripheral. Figure 12 shows the inclusion of the "OPB UARTLITE" Xilinx IP and the configuration of the UART.

After the peripheral IP configuration pages, the BSB sets up the IP peripheral that will be used as the "STDIN" and "STDOUT" from the EDK C programming environment as shown in Figure 13. Setting the UART as the "STDIN" and "STDOUT" device makes it possible to create a C program that executes on the PowerPC that delivers a text based console to the user via the Hyper Terminal window on a connected PC. Finally, the BSB configures the address ranges that map each of the peripheral's

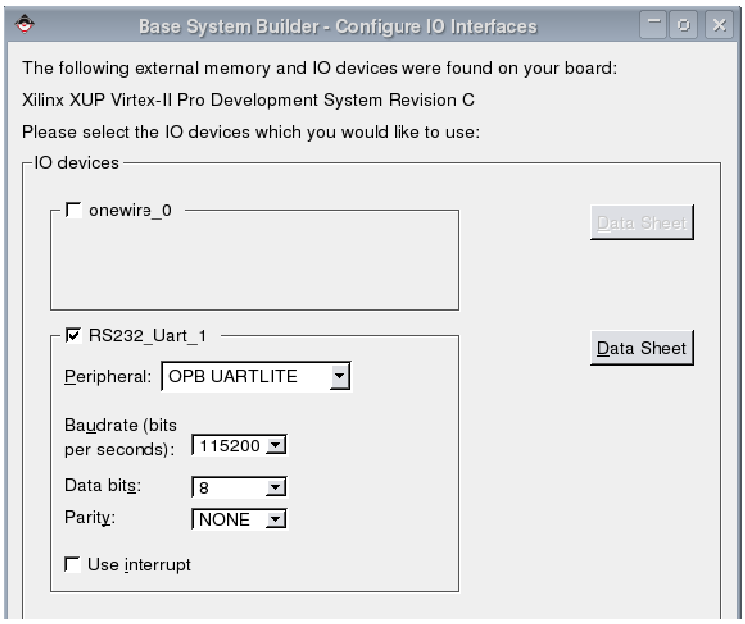


Figure 12: BSB Peripheral Configuration

registers and address ranges to a unique location within the PowerPC's addressable memory. Given in Figure 14 is an example of an EDK project created with the BSB using the XUP-V2P board definition file. As seen in Figure 14, the EDK GUI provides a graphical "System Assembly View" to show how the various pieces of peripheral IP are

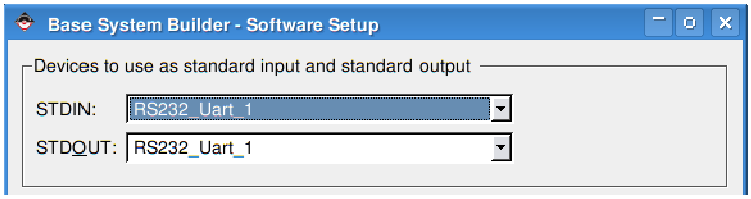


Figure 13: BSB StdIn and StdOut

connected to the PowerPC using the Processor Local Bus (PLB) and On-Chip Peripheral Bus (OPB) buses. While the system in Figure 14 was built using the BSB, it is still possible to add additional IP to the hardware system design using the IP Catalog shown in Figure 15. This library contains many different components and peripherals such as

the IBM PowerPC 405, DDR memory controllers, Ethernet controllers, UART's, DCM's and Reset Logic. In addition, the XUP-V2P hardware included addition IP for integrating the various custom XUP-V2P board peripherals to the FPGA. By simply dragging the various pieces of IP from the IP Catalog to the System Assembly View, the peripherals are automatically connected to either the OBP or PLB internal FPGA buses.

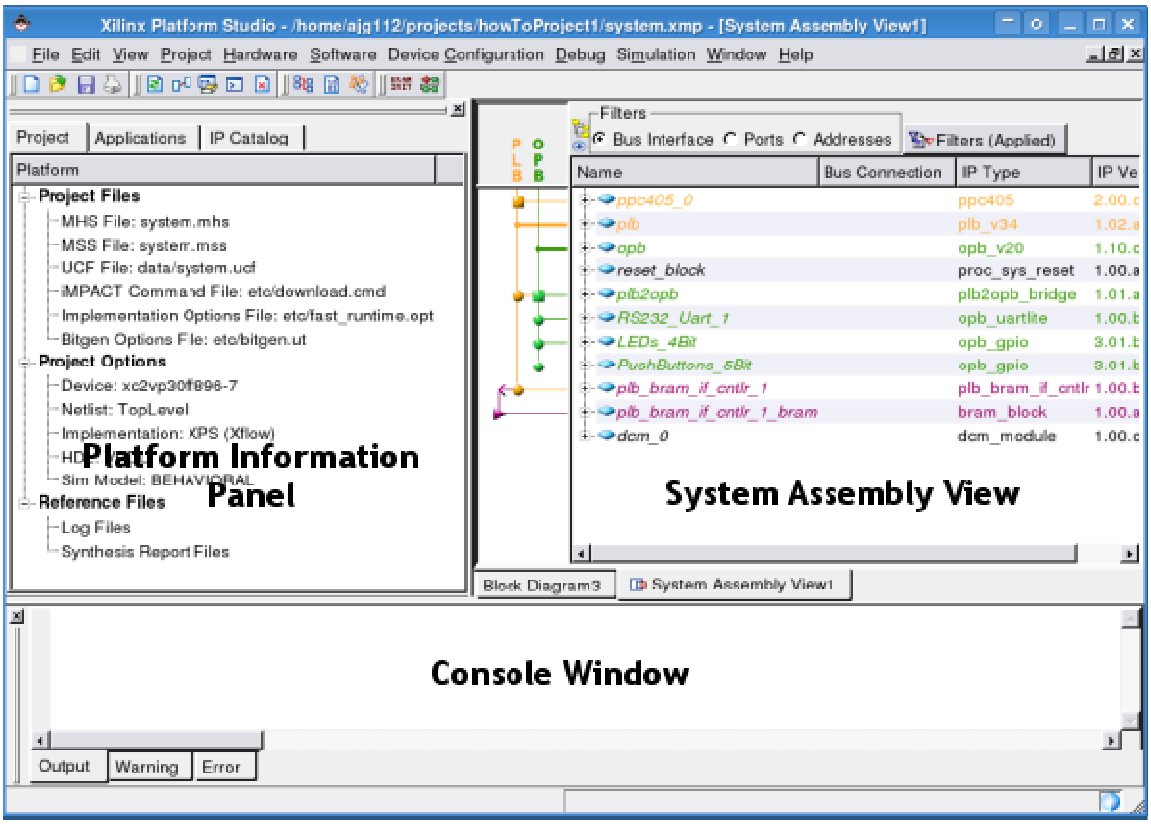


Figure 14: Starting EDK Project GUI

Included with each piece of IP in the "IP Catalog" are associated drives that are accessed using a C API. The application panel shown in Figure 16 is the location within the EDK GUI where C programs are written to interface to the added hardware peripheral in the EDK project. By using the necessary C header files, the calling C program is given access to the peripheral drivers using function calls. The C program developed for the

XUP-V2P system is contained in appendix E, and a screen shot of the menu driven UI via a Hyper Terminal window is given in Figure 26.

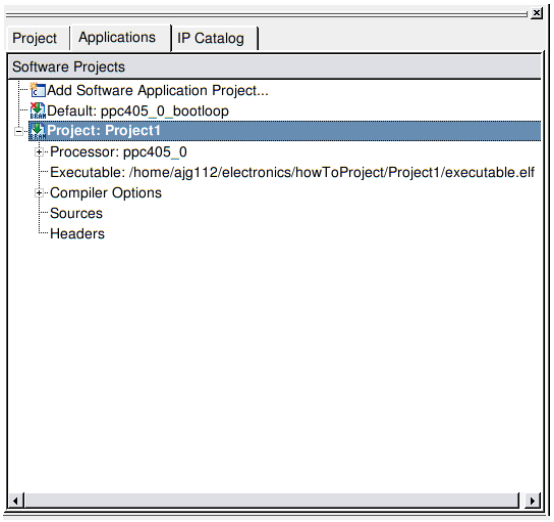


Figure 15: EDK IP Catalog

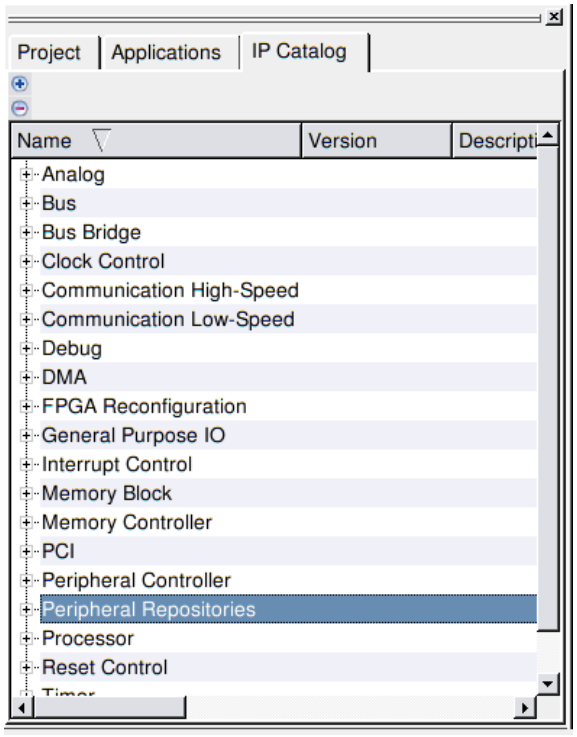


Figure 16: Software Application Panel

Whether the system is created using the BSB or by manually adding IP to the system, the Xilinx IP offerings enable a relatively quick starting point for interfacing the hardware peripherals to the PowerPC. This allows the majority of the development time to be spent on developing the needed custom logic and the PowerPC C program. Using the "Create and Import Peripheral Wizard" shown in Figure 17, custom logic can be

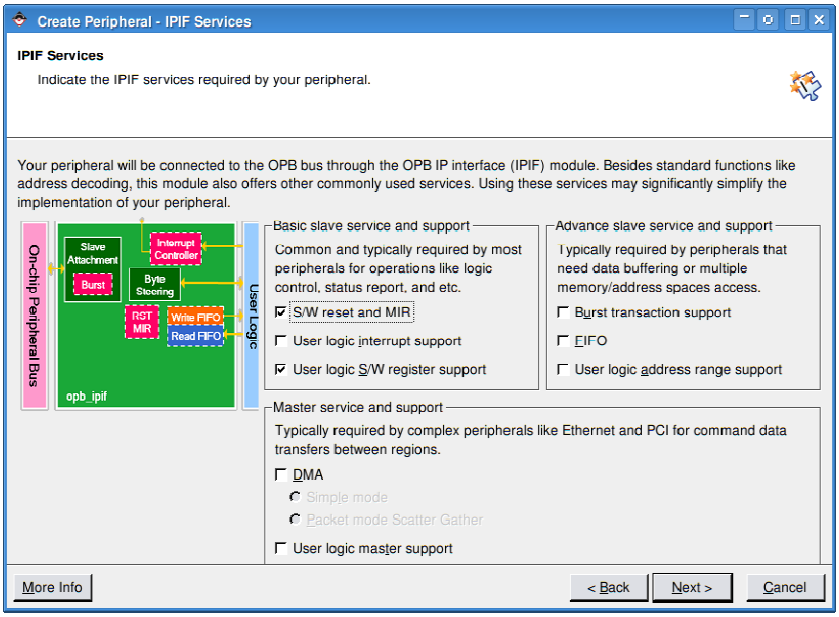


Figure 17: Custom Peripheral Wizard

imported into the EDK development environment with relative ease. By specifying the needed OPB or PLB feature sets, the wizard creates custom wrapper logic in the form of VHDL files that are used to interface custom logic with either the OBP or PLB bus infrastructure. The wizard also creates a template VHDL file that implements the requested registers and address ranges. In the case of the neural network inversion algorithm, the VHDL hardware implementation was wrapped in logic created by the "Create and Import Peripheral Wizard," where the template registers and address ranges

were memory mapped to the algorithm's hardware implementation. The details of the algorithm implementation are discussed in chapter six.

LabVIEW FPGA Development Environment

The LabVIEW FPGA development environment creates a higher level graphical software abstraction to FPGA hardware programming. In addition, this environment creates an easy-to-use interface between LabVIEW code executing on the PC and hardware programmed using LabVIEW FPGA. The environment starts with the creation

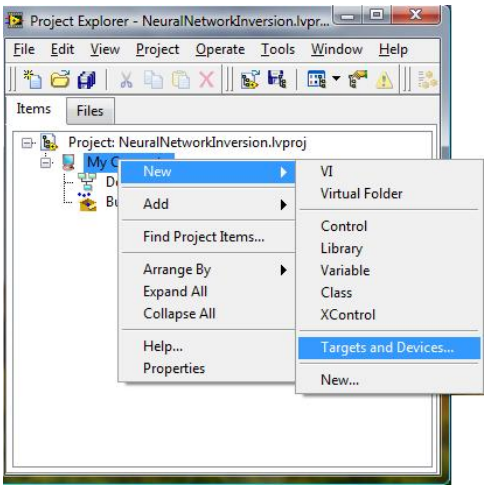


Figure 18: Create LabVIEW FPGA Hardware Target

of a LabVIEW project and defining a hardware target as shown in Figure 18. In Figure 19, a LabVIEW project is shown with two execution targets. The first is the "My Computer" target and contains all of the VIs that execute on the PC or embedded controller. The second target is the "FPGA Target (PXIe-7965R)" target, and contains the LabVIEW FPGA and VHDL code capable of executing on the hardware target.

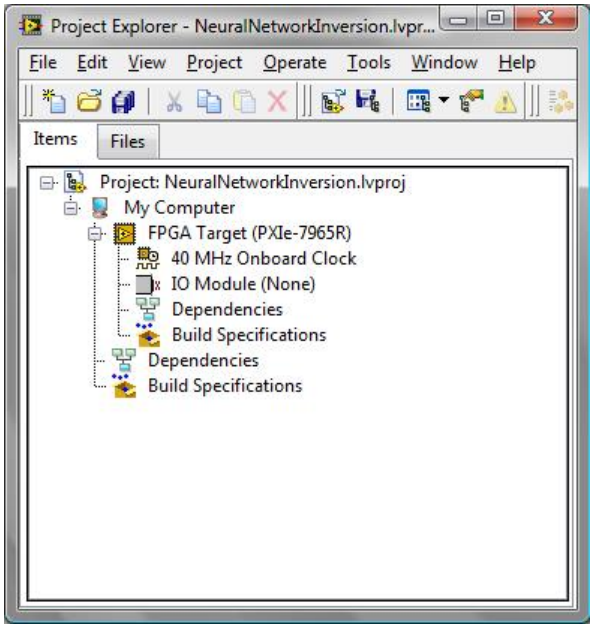


Figure 19: LabVIEW FPGA Project View

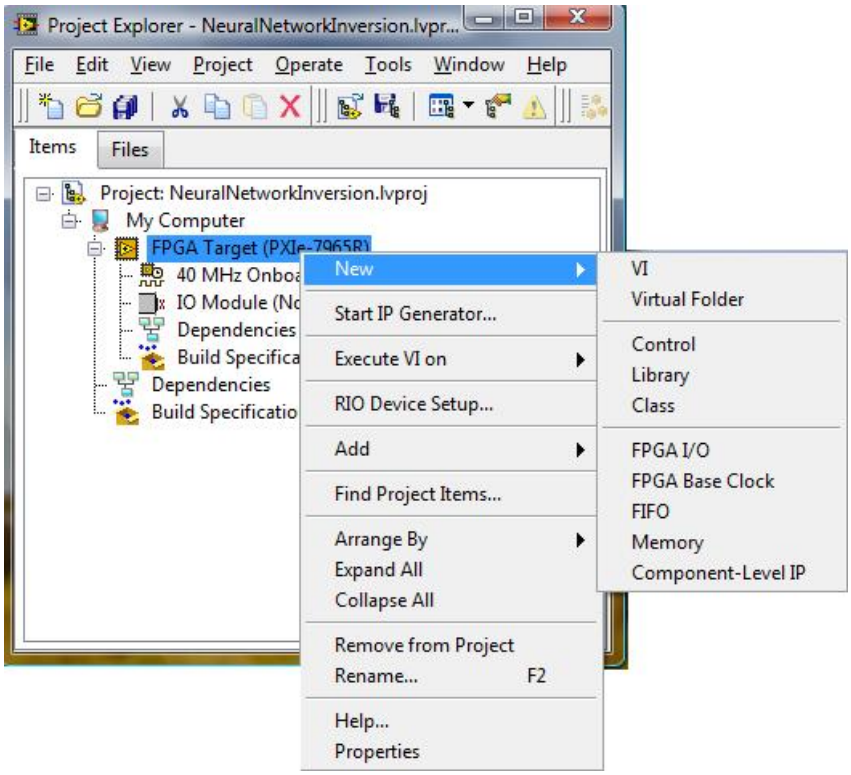


Figure 20: Hardware Target Capabilities

Once the project and hardware target have been created, different functionalities can be added to the hardware by right clicking on the hardware target. This right click menu is given in Figure 20 and includes VI, FPGA I/O, FPGA Base Clock, FIFO, and Memory. VI is the name of the file type used in LabVIEW that contains all of the graphical code. By selecting VI, a new VI is added to the LabVIEW project under the hardware target. In addition to adding programming files, various FPGA I/O and clocks can be added under the hardware target. The types of FPGA I/O and clocks available vary greatly with the specific capabilities of the hardware target. When using the PXIe-7965R hardware target to implement the neural network inversion algorithm, external FPGA I/O was not needed, and only the 40MHz clock was required in order to derive a 120MHz clock used to clock the algorithm's implementation. The FIFO option allows for the creation of many different FIFO implementations including Block RAM, LUTs, and flip-flops. In addition, selecting FIFO allows for the creation of DMA FIFOs that link to and from the host software (software running under "My Computer").

Given in Figure 21 is a simple VI that illustrates some of the many LabVIEW FPGA blocks and constructs. The outer most construct is the Single Cycle Timed Loop

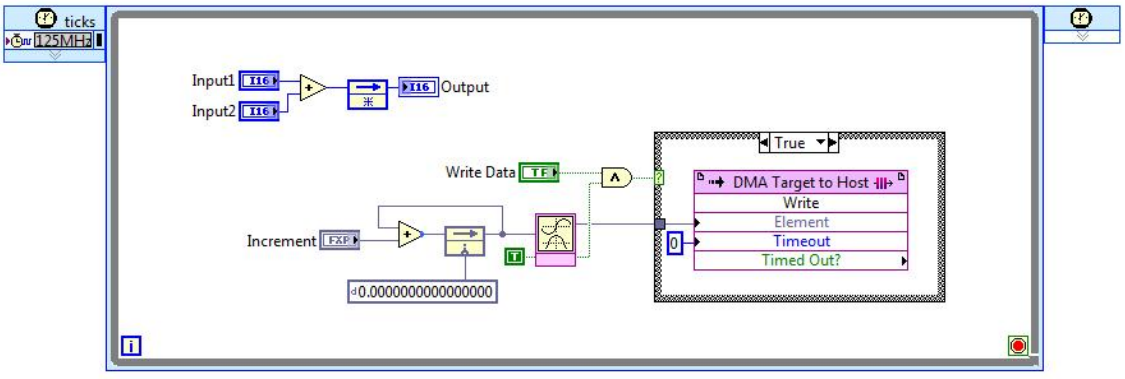


Figure 21: Example LabVIEW FPGA VI

(SCTL) and is used to indicate the execution rate of its interior. In the case of Figure 21, the SCTL is configured to run at 125MHz. As a result, all of the code in the loop will execute every 8 ns. At the top left of Figure 21 are two controls “Input1” and “Input2” that are summed together, delayed (flip-flop), and then outputted using the “Output” indicator. The controls and indicators act as registers that are accessed from the VI running on the PC in software as seen in Figure 22. At the bottom right of Figure 21, code was created to implement an accumulator that was used as an input to the CORDIC

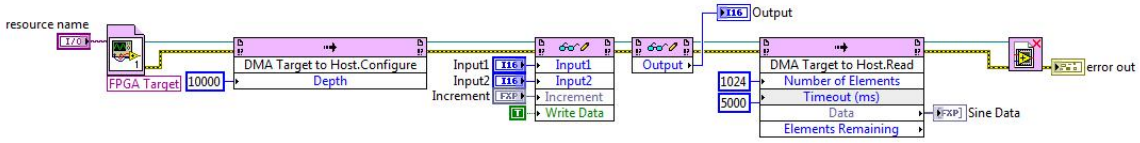


Figure 22: Simple LabVIEW Host Software Example

based sine calculation. When the sine calculation has valid data and the “Write Enable” control is asserted, the data is written to the DMA FIFO. This data is transferred using DMA to the host VI using the PXIe bus and appears in the host software VI using the method node shown in Figure 22.

The last major attribute of the LabVIEW FPGA development environment is the LabVIEW palette. The LabVIEW FPGA palette contains many pieces of IP that ease the development of algorithm implementations. Shown in Figure 23 is part of the LabVIEW FPGA palette. In addition to the IP shown in Figure 23, there also exists a comparison palette, array palette, and boolean palette useful for implementing many types of algorithms. Finally, besides the LabVIEW IP offerings on the palette, LabVIEW FPGA can also import VHDL code. Using the IP Integration Node, VHDL code can be wrapped into the LabVIEW development environment with ease.

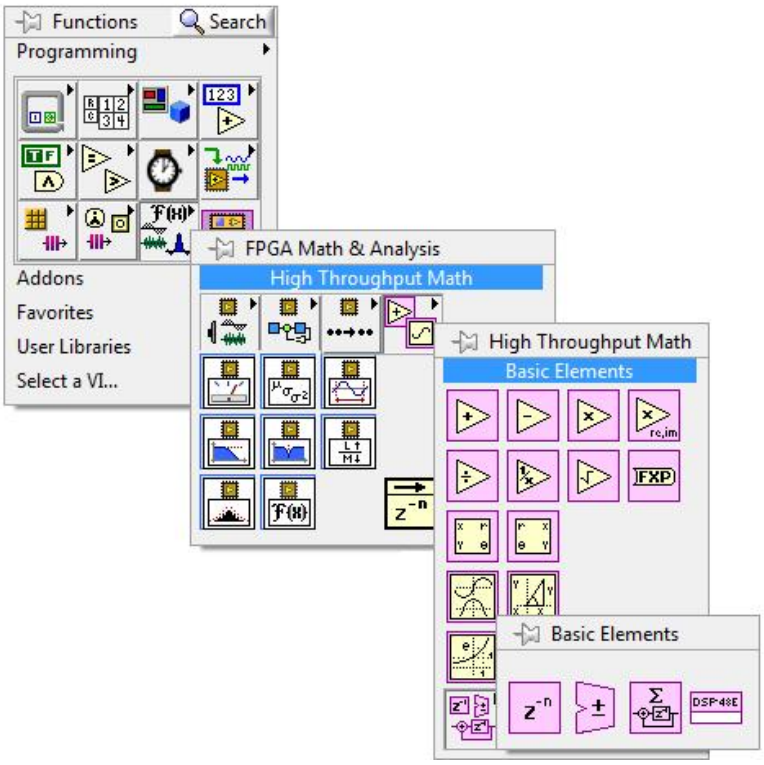


Figure 23: LabVIEW FPGA Palette

Development Environment Comparison

Each of the development environments had its advantages and disadvantages. The first major difference between the two development environments was the language used to implement the algorithm. In the case of EDK, the algorithm was implemented using VHDL, a textual hardware description languages shown in Figure 24 while the LabVIEW FPGA environment was programmed using LabVIEW, a graphical data flow language shown in Figure 25. While the programming language choice is typically a matter of preference and prior familiarity, there are still distinct advantages and disadvantages.

An advantage of the LabVIEW FPGA environment is that the higher level graphical constructs allow for a much faster development time. In the case of the neural

```
162                 Input68          : in  STD_LOGIC_VECTOR(15 downto 0);
163                 Input69          : in  STD_LOGIC_VECTOR(15 downto 0);
164                 Input70          : in  STD_LOGIC_VECTOR(15 downto 0));
165     end component;
166
167 begin
168
169     RegALogic:process(clk,Reset)
170     begin
171         if(Reset = '1') then
172             for count in 0 to 70 loop
173                 RegA(count) <= (others => '0');
174             end loop;
175         elsif(rising_edge(clk)) then
176             if(OutputWrEn_i = '0') then
177                 RegA(conv_integer(RegisterSelect)) <= SquashOut;
178             end if;
179         end if;
180     end process;
181
182
183     RegBlogic: process(clk,Reset)
184     begin
185         if(Reset = '1') then
186             for count in 0 to 70 loop
187                 RegB(count) <= (others => '0');
188             end loop;
189         elsif(rising_edge(clk)) then
190             if(NodeNumber = x"000" or NodeNumber = x"028" or NodeNumber = x"05a" or NodeNumber = x"0a0") then
191
192                 if(NodeNumber = x"000") then
193                 RegB(0) <= Input00;
194                 RegB(1) <= Input01;
195                 RegB(2) <= Input02;
196                 RegB(3) <= Input03;
197                 RegB(4) <= Input04;
```

Figure 24: Example VHDL from EDK

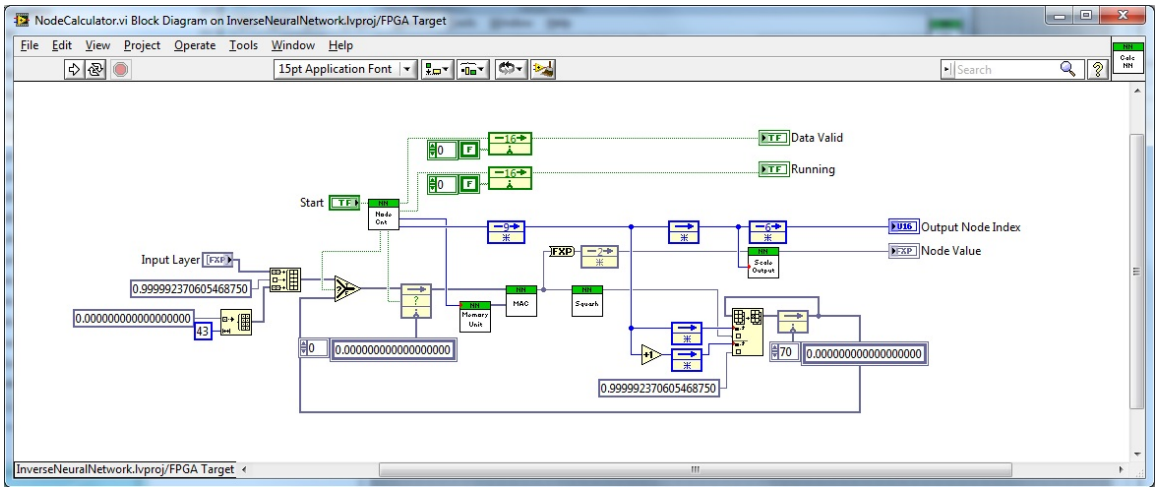


Figure 25: Example Code from LabVIEW FPGA

network inversion algorithm implementation, the development time needed in EDK environment using VHDL was over four times greater than the development time needed

in LabVIEW FPGA. However, the advantage created by these same higher level constructs is also a disadvantage when implementing highly optimized logic operations. Due to the higher level abstraction of LabVIEW FPGA, it is difficult to create optimized LabVIEW FPGA code that is as optimized in both clock speed and/or size as a heavily optimized VHDL implementation. Also as a result of the higher level of abstraction, performing bit level operations while possible, is both time consuming and creates difficult to understand LabVIEW code. Finally, LabVIEW FPGA does not currently support generics unlike VHDL. Correctly written VHDL code using generics creates generalized code that can be reused in many situations by modifying the generics. When using LabVIEW FPGA for example, if the bit width of a signal needs to be modified, the signal's bit width must in many cases be modified in many locations throughout the code.

Aside from the high level abstraction, another reason for the very fast development time in LabVIEW FPGA was due to the integration of the PC. In EDK, all of the available IP was used to create a system similar in functionality to that of a PC.

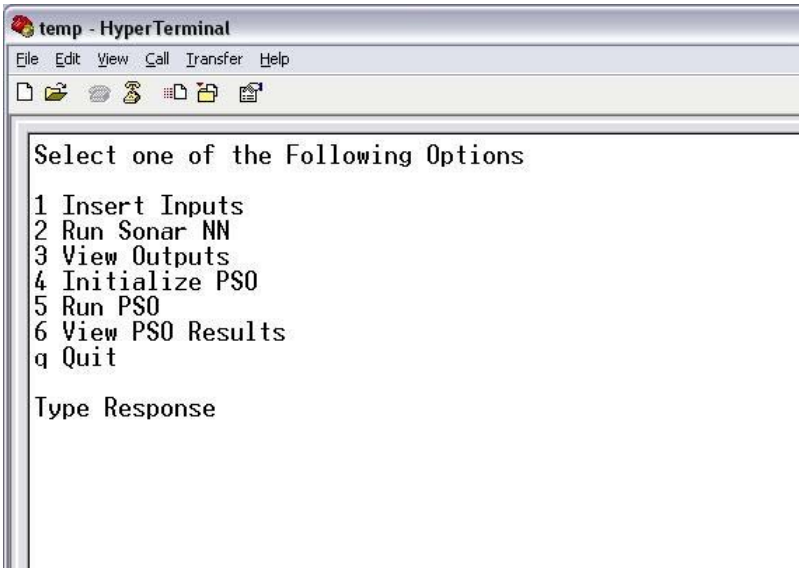


Figure 26: EDK User Interface

While the BSB wizard in EDK eased the creation of the system, creating the EDK hardware and software still took a large effort to get everything working properly. Even with everything working correctly, the end result was a user interface that was much less sophisticated than a typical user interface on modern PCs. The user interface of the neural network inversion implementation using EDK can be seen in Figure 26. In order to graph the data, the output of the algorithm was stored on the CF memory card in order to

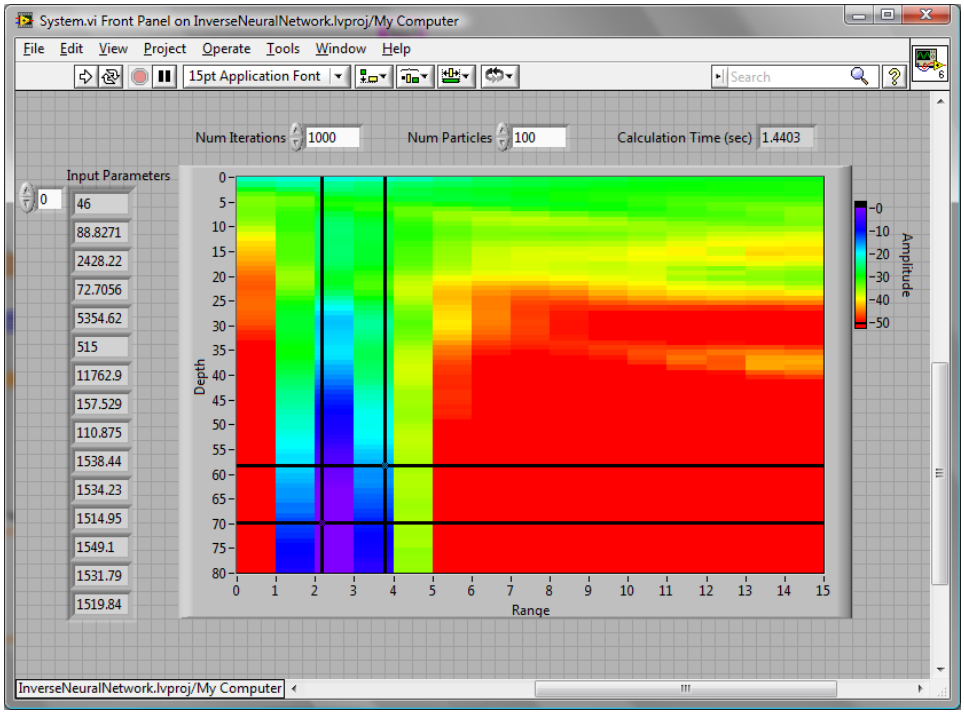


Figure 27: LabVIEW User Interface

transfer it to a PC. While using Ethernet to transfer the data was a possibility using the EDK development environment, it would have required considerable effort to write a secondary PC program to graphically display the data.

Using the LabVIEW development environment, this entire process was unnecessary due to the existence of an actual PC in the system. By using the existing

software infrastructure and running LabVIEW on the PC, a much more polished user interface was created as shown in Figure 27. While a much better user interface was possible using the EDK platform, it would have taken considerable time and effort to develop the keyboard, mouse, and video interfaces in addition to programming a custom graphics engine using C on the PowerPC. Even if this massive effort had been given to this on-chip user interface, it would still not have been as capable as the PC user interface using LabVIEW.

Another advantage of the LabVIEW development environment was the availability of IP and the integration of external IP. As was seen in Figure 23, LabVIEW FPGA has full palettes of IP that range from simple numerical operations, to complex CORDIC IP used to calculate trigonometric functions, to FFTs and windowing functions. In addition, external IP can easily be incorporated into the LabVIEW environment using either the IP Integration Node, or the Component Level IP method. However in EDK, while IP such as the IP generated by the Xilinx Core Generator can be imported directly into the VHDL contained within the custom logic, the EDK environment does not offer any IP that is useful for developing algorithms; all EDK IP offerings are meant to build hardware/software systems by attaching the IP to the OPB and PLB bus infrastructure.

An advantage of the EDK system was its simulation capabilities. Since the EDK system is made of interconnected VHDL peripherals, simulation is the same as a typical VHDL design. For the neural network inversion implementation, ModelSim was used as the VHDL simulator. Simulation was straightforward using VHDL testbenches to test various components at different levels in the design hierarchy. While the simulation of

the custom logic IP was not difficult, a full simulation of the entire system including compiled C code on the PowerPC was still quite difficult.

In contrast, LabVIEW does not offer a complete simulation environment. There are three possible options for hardware simulation, but no one option works best in all situations. One method of simulation in LabVIEW FPGA is “Emulation Mode”. Using this method, it is possible to debug most designs, but this simulation is slow and it is difficult to debug sub components (subVIs). The second type of simulation requires moving the design under test to the “My Computer” target and replacing the SCTL with a for loop. However, some FPGA IP cannot run in this mode and it is still difficult to see multiple signals within the design on a single graph. Instead, multiple graphs, each graphing one signal were created. While it is possible to construct a single graph showing all of the probed signals simultaneously aligned in time, it is a task left to the developer to build the necessary graph programmatically using LabVIEW. The final type of simulation makes use of ModelSim. This simulation method allows for full simulation of the hardware design. However, in order to use this simulation method, a testbench must be created that translates the graphical code that runs on the PC in software into VHDL, a time consuming process. In addition, since the wires on the graphical diagram cannot be named, the developer must infer the computer generated signal name based on the information in the context help.

The final advantage for the EDK environment is that it can be used on any Xilinx FPGA independent of the final PCB while the LabVIEW FPGA environment can only be used to target specific hardware products produced by National Instruments. If the end goal is to prototype a system or create a production system using the appropriate National

Instruments hardware, then this is neither an advantage nor disadvantage. However, if the goal is to create a custom hardware solution using Xilinx FPGAs, then it is not possible to use LabVIEW FPGA to target the custom board.

Comparison Summary

In general, the LabVIEW FPGA environment is well suited for creating systems very quickly. It also successfully makes register reads and writes along with DMA data transfers trivial thereby creating a seamless interface between the FPGA and software implementations. In addition, it does not require a developer to be a digital design expert in order to properly program an algorithm on an FPGA, and it leverages the existing PC hardware and software drivers. While graphical FPGA programming may not be as optimal as VHDL, VHDL is easily imported into this environment if there is a critical need for hardware optimized code. However, it can be difficult to simulate designs in some situations with LabVIEW FPGA, and it cannot be used at all when trying to target a custom PCB.

In contrast, the EDK system is easier to simulate, and it is capable of targeting any Xilinx FPGA. Highly optimized and efficient VHDL code is also achievable, and the environment works well for embedded designs. However, the EDK system does not leverage the PC hardware, and while it attempts to duplicate some of the PC's capabilities, it severely falls short when creating an interactive graphical user interface. Finally, the EDK environment is much more difficult to use and the development time is much longer.

CHAPTER FIVE

Hardware Configuration

The previous two chapters presented the hardware and development environments used for the implementation of the neural network inversion algorithm. This chapter presents the system level perspective for the algorithm's implementation in hardware. First, a discussion is given on the data dependencies present within the algorithm and the potential algorithm segments that would most benefit from a parallel hardware implementation. Finally, three different parallelized approaches are described for the PSO algorithm with their respective hardware configurations.

Neural Network Inversion Data Dependencies

As with almost any algorithm, there are segments, a series of sequential computations, that can benefit from parallelization, and there are segments that either cannot be parallelized or show no positive benefit from being parallelized. The first step towards speeding up any algorithm with a parallel implementation is identifying the segments of the algorithm that can be parallelized. Presented in Figure 28 is a serialized execution sequence of the neural network inversion algorithm described in chapter two. The red lines in the figure indicate where data dependencies exist. For example, at the lowest level of the algorithm, a sequence of multiply and accumulation operations followed by a squashing function result in the calculation of a single node of the neural network. Before the squashing function can execute, it needs a valid input, the result of all of the multiply and accumulation operations. As seen in the Figure 28, many

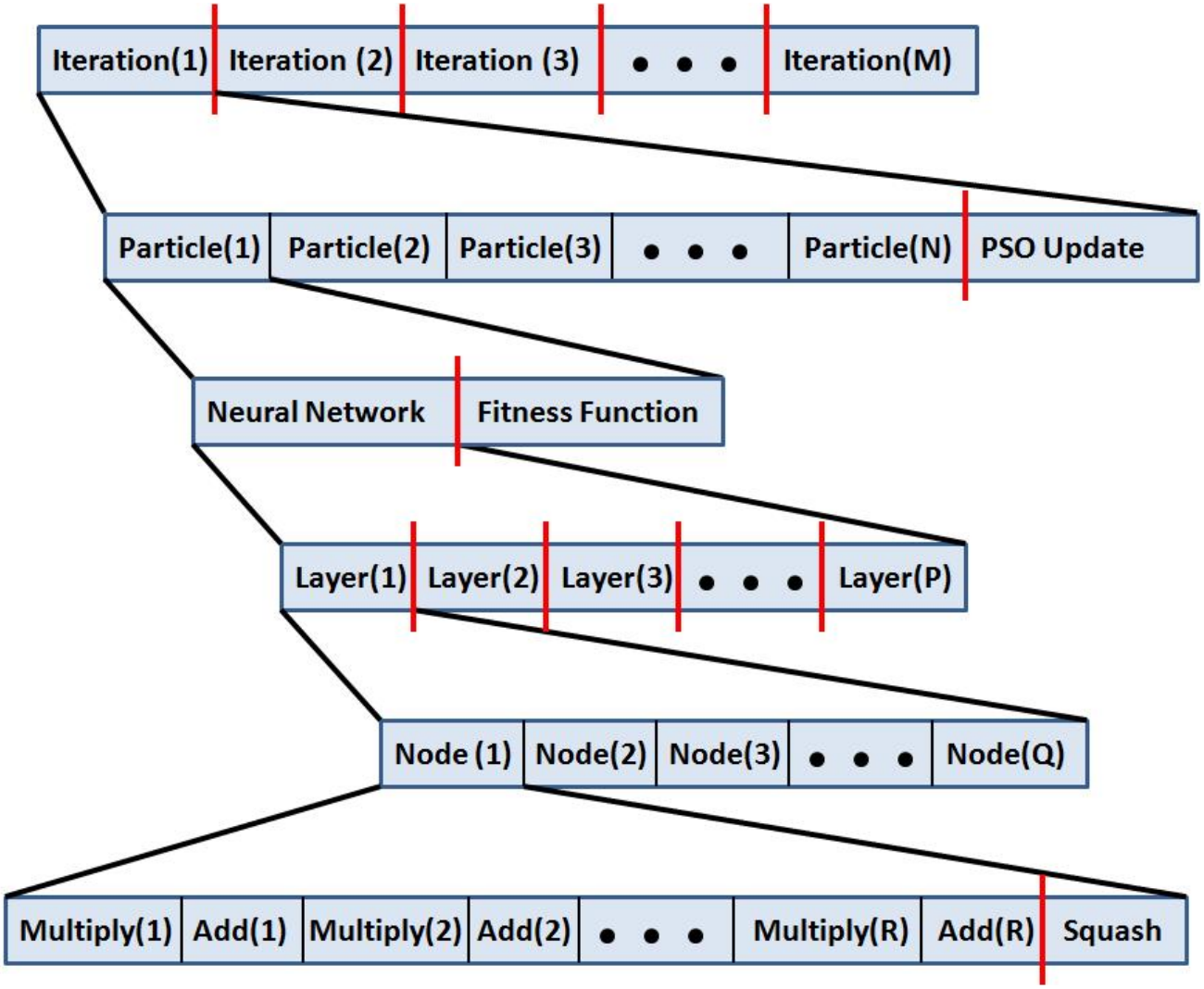


Figure 28: Algorithm Dependencies

other data dependencies exist within the algorithm that keep it from being completely implemented in parallel.

Three major segments within the algorithm were identified as having significant speed-up potential via a parallel implementation. Starting at the lowest level, the first possibility is to parallelize the calculation of a single node of the neural network. Within each node, a long series of multiply and accumulate operations are required before the squashing function can execute. Since the FPGA has many multipliers and can implement a pipelined adder tree, this entire operation can be implemented in parallel in a pipelined fashion with the squashing function performed in the final stage. The next possibility requires speeding up the time required to calculate a single layer of the neural network. Because all of the nodes of the neural network within each layer do not have any data dependencies, the nodes are independent of each other once they are given their inputs, and are thus capable of executing in parallel. The last major possibility is to execute multiple particles of the PSO algorithm in parallel. Again, this is an instance where each particle does not share any dependencies with the other particles until the PSO iteration changes. As a result, each particle can execute in parallel without any changes to the algorithm. These three different algorithm segmentation strategies are presented in the column headings of Table 1.

Hardware Limitations

In order to get the maximum speed-up of the algorithm, ideally it is desirable to perform all things in parallel that are capable of a parallel implementation. However, there are many hardware limitations that restrict the ideal from occurring. FPGAs have a limited number of multipliers, memory, and other logic needed to perform the

calculation. In addition, connecting multiple FPGAs together introduces undesirable communication delays that are discussed in chapter seven. However, even a combination of multiple FPGAs assuming instantaneous FPGA-to-FPGA communication does not allow for the possibility of placing everything in parallel. For example, in the worst case it takes 70 multiples to calculate each node in the last layer of the neural network, there are 1,200 nodes in the final layer of the neural network, and there are 100 particles. Therefore it would require over eight million multipliers in parallel to perform the worse case operation, many orders of magnitude greater than any modern FPGA contains. As a result, none of the possibilities listed above are implemented simultaneously to the fullest theoretical extent.

Table 1: Hardware Implementations

Hardware Configuration	Parallel Node Calculation	Parallel Layer Calculation	Parallel Particle Calculation
XUP-V2P Single Board	X		
XUP-V2P Two Board	X	X	
XUP-V2P Three Board	X	X	
PXIe Hardware Single PXIe-7965R	X		
PXIe Hardware Dual PXIe-7965R	X		X

Three different variations of the parallel implementations were performed on two different hardware architectures. The first variation implements the entire neural network inversion algorithm by calculating each node in parallel. This is accomplished by placing

all of the multipliers in parallel and replacing the accumulation operations with a pipelined adder tree. For this variation, both hardware architectures were used for this implementation in order to compare their respective capabilities. The second variation implements placing each layer of the neural network in parallel in addition to calculating each node in parallel. This variation was performed using only the XUP-V2P hardware platform using the two and three board configurations. A multi-board solution was needed because the Virtex-II Pro FPGA only contains 136 multiplies and 70 multiplies are needed in the worst case node calculation.

The final variation implemented calculated each of the particles in parallel in addition to parallelizing each node. This implementation was only performed on the multi-board PXIe hardware architecture. Because the Virtex-5 FPGA on the PXIe-7965R contained 640 multipliers, multiple particles could have been implemented in parallel on a single PXIe-7965R. However, in order to make a valid comparison of the hardware architectures, the Virtex-5 was resource handicapped according the Virtex-II resource count. Presented in Table 1 is a summary of the various algorithm segmentation approaches and the hardware used for each approach.

System Level Configuration

As summarized in Table 1, there are three different algorithm segmentation methods where different hardware architectures were used for the implementation. This section describes the system level configuration for each of the hardware implementations. The XUP-V2P architecture was used to implement three different configurations. The first is the basic hardware configuration that utilizes a single XUP-V2P development board to parallelize the node calculation of the neural network. The

next two XUP-V2P configurations make use of the high speed SATA connections to connect multiple XUP-V2Ps in both two board and three board configurations. These multi-board XUP-V2P configurations are used to implement the algorithm parallelizing both the node calculation and layer calculation. Figure 29 presents the single board configuration and Figure 30 presents the multi-board configuration.

The PXIe platform is used to implement two configurations. The first utilizes a single PXIe-7965R with a chassis and controller in order to implement the neural network inversion algorithm with the node calculation performed in parallel, the same segmentation method used for the single board XUP-V2P implementation. The second PXIe configuration adds another PXIe-7965R to the single board solution and utilizes Peer-to-Peer (P2P) streaming to directly connect the two PXIe-7965Rs. This configuration implements the algorithm with the node and particle portions of the calculation performed in parallel.

Single Board XUP-V2P Configuration

The single board design, shown in Figure 29, consisted of a single XUP-V2P development board where the entire neural network inversion algorithm was implemented entirely inside the VirtexII-Pro FPGA. Aside from the custom logic used for the neural network inversion implementation (discussed in detail in chapter six), this version also makes use of a UART IP core for user communication, a Compact Flash drive for storing results and boot loading the FPGA configuration file, and the DDR memory used for program storage. Like all of the XUP-V2P designs presented, the single board design utilized Xilinx EDK to create the entire hardware and software system in order to utilize many of the already created generic peripherals.

At the heart of the single board system was the PowerPC microprocessor which connected to all other peripheral using the Processor Local Bus (PLB) and On-Chip Peripheral Bus (OPB). While all of the EDK peripherals contained within the IP library

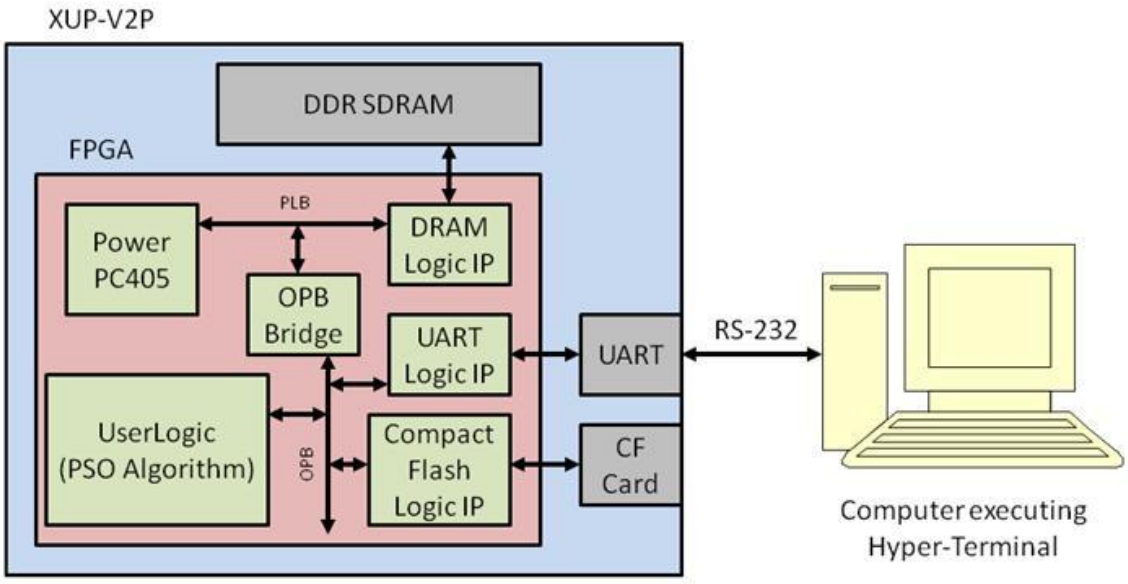


Figure 29: Single Board XUP-V2P Configuration

were accompanied by an included software driver to access the hardware peripheral, the neural network inversion custom logic did not require a driver; a typical driver was unnecessary because all of the control functionality was built into the hardware. With the use of basic pointers in the C programming language, access to the custom logic memory was straight forward.

From the PowerPC's perspective, the neural network inversion peripheral contained segments of software accessible memory. These memory segments were memory mapped to specific locations of the PowerPC's addressable memory determined by configuration within EDK. This allows software running on the PowerPC to transfer data between various peripherals such as the UART and Compact Flash. A C program

was written to use the UART as standard in/out, and the Compact Flash drive as non-volatile file storage. This program allowed users to interact with the neural network inversion peripheral via the UART and Compact Flash. Using the UART, the single board design presented a menu based user interface (UI) that was delivered to the user through an external PC running a Hyper Terminal window as pictured in Figure 29. The XUP-V2P UI is shown in Figure 26 in chapter four. As shown, this interface allows users to do any needed requirement of the system such as reading and writing files to the provided flash and starting the neural network inversion program. Finally, the program utilized a EDK IP hardware based timer counter in order to measure system performance.

This single board configuration was highly successful despite being the simplest of the XUP-V2P configurations and not utilizing multi-board parallelism for increased computational power. Using only a 100MHz system clock, this configuration was able to perform the entire neural network inversion (using 100 agents for 1000 iterations) in 1.4 seconds. This time is comparable to the work in [4], where the SRC-6e computer (dual processor and dual Virtex-II FPGA) was used to calculate the algorithm in 1.8 seconds. For comparison, this was accomplished running on a Pentium 4 in just under two minutes [4]. Finally, this single board XUP-V2P configuration served as a base comparison point in order to evaluate the performance of the multi-board systems.

Multi-board XUP-V2P Configuration

The multi-board configurations were very similar to the single board configuration except that they connected through the Xilinx Aurora IP core in streaming mode using SATA cables to another XUP-V2P system. Because the XUP-V2P systems only contain two host ports and a single target port, a maximum of three point to point

connections can be made. While custom boards could yield more connected systems, these configurations demonstrate the speed-up resulting from this basic configuration. Figure 30 shows the most complicated SATA configuration using the three board design with a UART connection to a PC.

The first SATA system considered is the two board configuration. This is the same system as presented in Figure 30 but with only two XUP-V2P development boards. The two board system was created in order to first demonstrate that the Aurora communication channel was viable and to collect data on throughput and latency. In addition, this system allowed for experimental data to be collected and compared to the expected results in chapter seven.

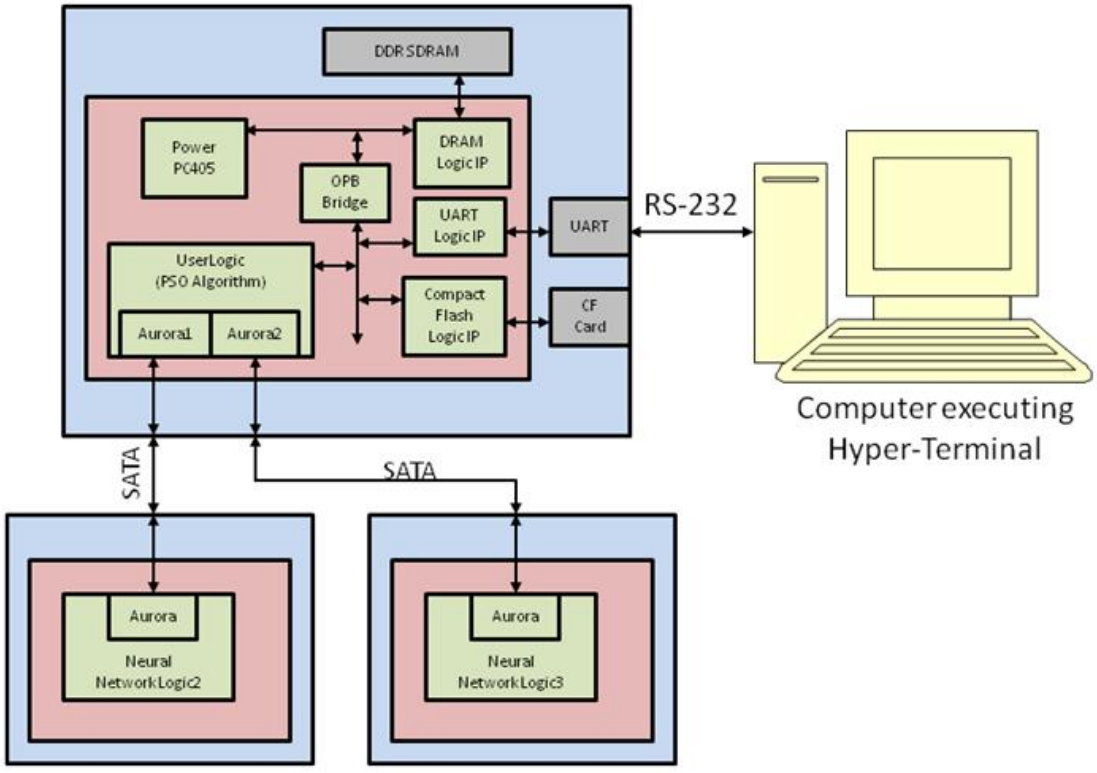


Figure 30: Three Board XUP-V2P Configuration

In this two board configuration, the development board connected to the PC via the UART was designated the master system while the other board was defined as the slave. The master system was created in the same fashion as the single board system using Xilinx EDK. The principle difference is that this multi-board design contains the Aurora core in order to enable multi-board communication. It is important to note that the Aurora core was embedded within the inverse neural network hardware and not connected to the PLB or OPB internal buses. This allows the hardware on the master to connect directly to the hardware contained on the other slave development board and thus eliminates the inefficiencies of moving the data through the processor. However, this also removes much of the flexibility that may be needed in other systems. As a result of this decision, values were hard-coded into the hardware that determined if the system contained one, two, or three developments boards in the complete system.

The slave development boards were developed outside of the EDK environment of the master XUP-V2P system using Xilinx ISE. Since only one board needs to connect to a PC as the UI, only the master needed to use the on-chip PowerPC405. This meant that every slave system only needed instantiated hardware capable of calculating the specific neutral network nodes required by the slave. While EDK could be used for this hardware, Xilinx ISE was more than sufficient.

The multi-board master/slave SATA configurations effectively utilize the slave board as a coprocessor to the master. Part of the task of calculating each layer of the feed forward neural network was given to the slave coprocessor. Because the previous layer of the neural network must be completely calculated before the next can begin, there is a high level of integration between the slave and the master. Each slave must first receive

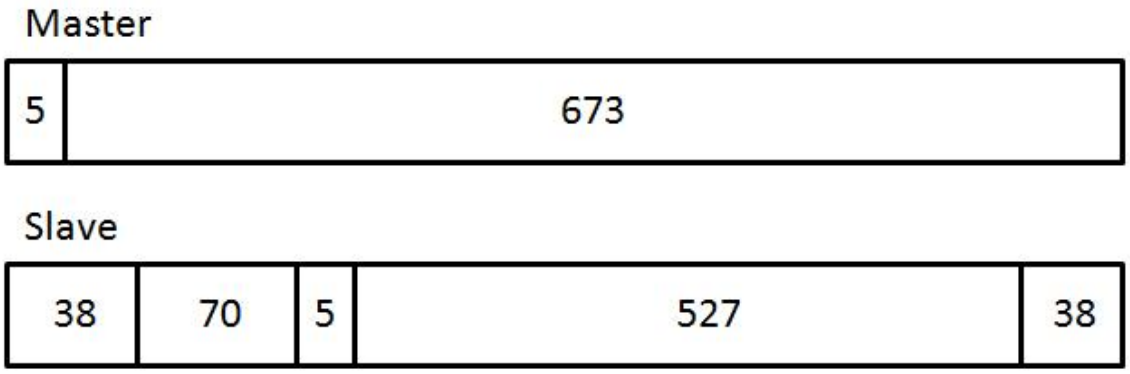


Figure 31: Two Board XUP-V2P Timing Diagram

the complete previous neural network layer, process its specific portion, and then send back the results to the master. For example, in calculating the final neural network layer of 1,200 nodes in the two board configuration, the optimal speed-up occurs when the master calculates output nodes one thru 673 and the slave is responsible for calculating nodes 674 thru 1,200. This imbalance of nodes was due to the communication latency of 38 clock cycles in each direction and the transfer time of 70 clock cycles needed to move the 70 input values from the previous layer. Shown in Figure 31 is the timing diagram needed to achieve the optimal computation time by making the computation time equal for both boards where the five cycle delay represents the pipelining of the algorithm.

Once the SATA channels proved to be successful using two development boards, this method of connection was extended to three XUP-V2P systems. This configuration is presented in Figure 30. If more point-to-point nodes were critically needed, a custom PCB that utilizes all six of the RocketIO transceivers of the Virtex-II Pro could be designed. In addition, newer FPGAs could also be utilized.

Single Board PXIe Configuration

The single board PXIe configuration uses a single PXIe-7965R with a chassis and controller shown in Figure 32. This configuration implements the entire neural network inversion algorithm on the PXIe-7965R's Virtex-5 FPGA with the node calculation

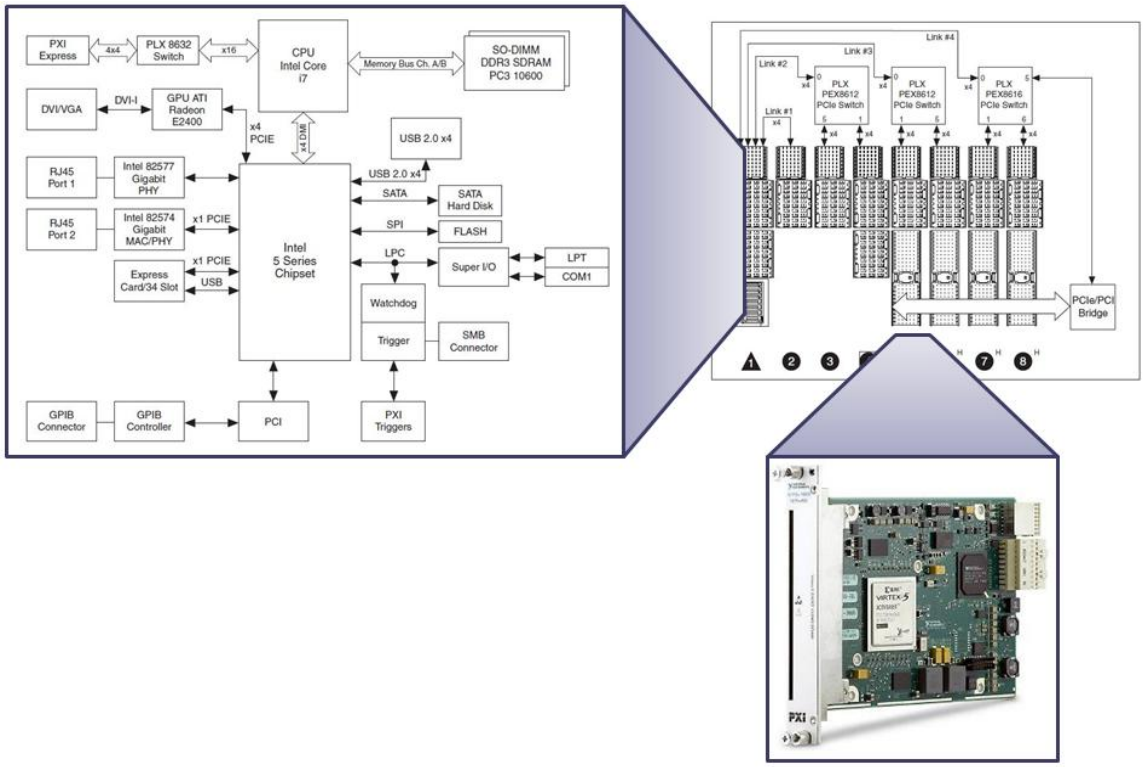


Figure 32: Single Board PXIe Configuration

performed in parallel. While the XUP-V2P architecture used EDK to connect peripherals in order to construct a UI, the PXIe architecture tightly integrates the PC with the FPGA using the standardized PXIe bus. This allows the PXIe architecture to leverage the hardware peripherals of the PC in addition to the well established device drivers for each of the peripherals.

The result is a much more attractive UI with a full interactive graphical display with mouse and keyboard capabilities shown in Figure 27. Instead of the text base user

interface of the XUP-V2P architecture, this interface allows the user to move the cursors (the black lines on the graph in Figure 27) to indicate the target region with great ease. A negative consequence of this approach is the latency and throughput restrictions of the PXIe bus. The PXIe bus had measured latencies of 3.2 microseconds and a data throughput of 800MB/s under optimal configurations. However, since the final result is only returned once at the very end of the computation, the throughput and latency of the PXIe bus was a negligible increase to the total computation time. The logic and PowerPC of the XUP-V2P configurations were located on chip and therefore much faster. In addition, dedicated communication links are used that give very deterministic throughputs with lower latencies. However, if a graphical representation of the data was needed on a PC, the time to move the CF memory card to the PC and graph the results would be orders of magnitude worse than any PXIe latencies.

Multi-Board PXIe Configuration

The multi-board PXIe configuration pictured in Figure 33 adds an additional PXIe-7965R FPGA processing card to the single-board PXIe configuration. This configuration was used to implement the neural network inversion algorithm by calculating both the nodes and particles in parallel. While the XUP-V2P system used Aurora via SATA links to transfer data between FPGAs, the PXIe architecture used Peer-to-Peer (P2P) streaming for data transfer, a FIFO based protocol connected using PCIe data links as shown in Figure 34. Using P2P streaming, each of the PXIe-7965R's are able to communicate with each other using DMA over the PXIe backplane without involving the PC controller. P2P streaming was already integrated into this platform and is accessed through LabVIEW FPGA P2P DMA FIFOs, creating a very easy, effective,

and quick to implement data transfer link. Figure 35 depicts the easy LabVIEW FPGA API primitives used to transfer data using P2P.

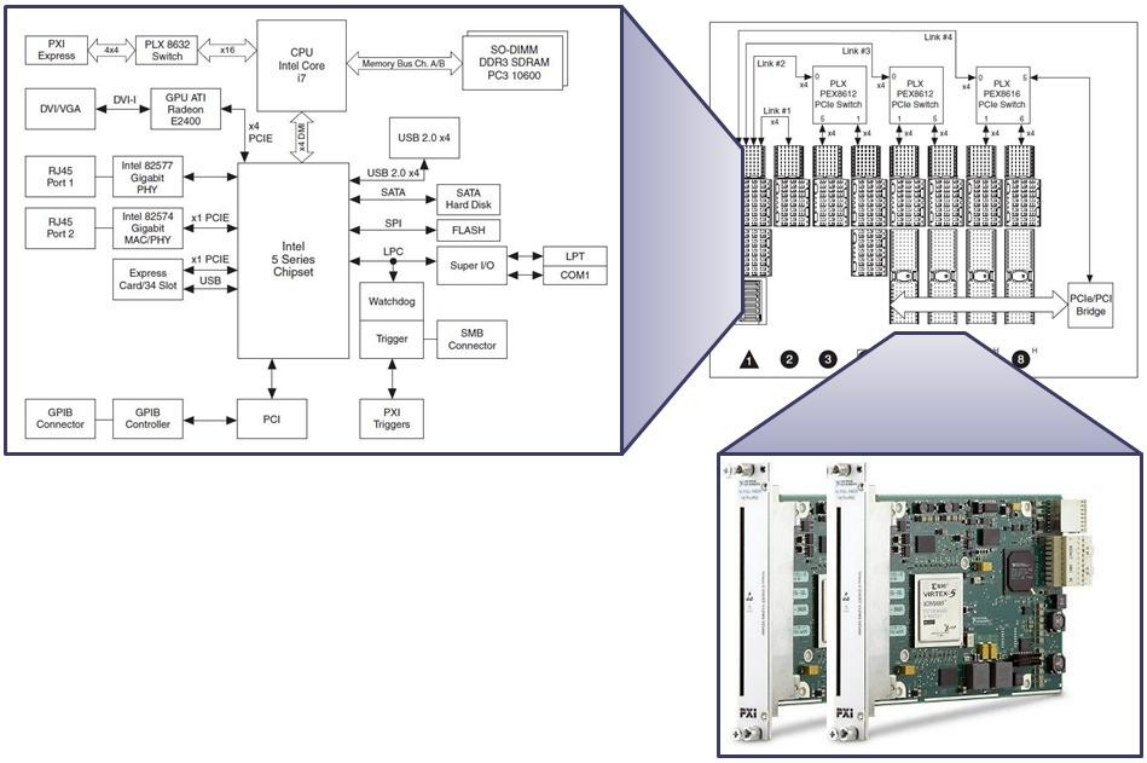


Figure 33: Multi-Board PXIe Configuration

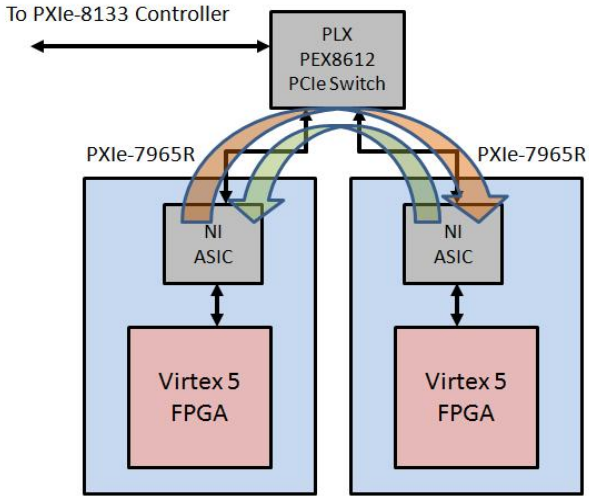


Figure 34: P2P Streaming Block Diagram

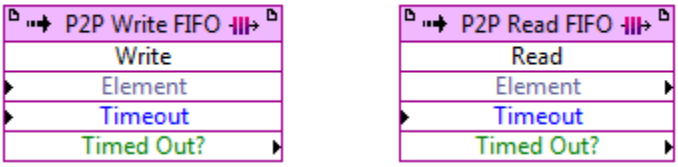


Figure 35: P2P User Interface

CHAPTER SIX

FPGA Hardware Design

As described in chapter five, there are two different hardware architectures and a total of five configuration used for the neural network inversion implementation. Despite different algorithm segmentation strategies used for the different implementations, both hardware designs are very similar. The principle differences between these implementations are the development environments used to create each design and the two different communication interfaces. This chapter describes the various low level hardware implementations using a top down approach for both the XUP-V2P hardware implementation created using EDK, and the PXIe architecture developed using LabVIEW FPGA.

XUP-V2P Hardware Implementation

The first discussed implementation is the XUP-V2P implementation created using the EDK development environment. This system consists of a memory component ("User Logic" in Figure 36), the neural network (NN) components, and the Particle Swarm Optimization (PSO) components. The memory component was responsible for interfacing the NN and PSO components to the OPB bus by creating the registers and the address ranges. In addition, in the multi-board XUP-V2P implementations, the Xilinx Aurora core is used for inter-board communications. The multi-board XUP-V2P high level block diagram can be seen in Figure 36 and the VHDL implementation code is located in appendices A and C.

Memory and Wrapper Logic

In order to take advantage of the EDK design development environment, wrapper logic was applied to the custom NN and PSO logic to interface this custom logic to the rest of the EDK system. Templates for this logic were created by the EDK peripheral creation wizard as presented in chapter four, and these templates were created with a custom number of software accessible registers and memory ranges.

The wrapper of the NN and PSO logic included eight software accessible registers and eight software accessible memory ranges given by Table 2 and Table 3. Once mapped to a unique location in the PowerPC's addressable memory, these registers and address ranges allowed for the direct control of the neural network inversion algorithm

Table 2: Register List

Memory Location	Name	Function
Base Address + 0x00	Start Register	Starts the PSO algorithm
Base Address + 0x04	Done Register	Returns a one when the PSO algorithm has completed
Base Address + 0x08	Iterations	Provides to the PSO algorithm the number of iterations to perform
Base Address + 0x0C	Particles	Provides to the PSO algorithm the number of particles in the search
Base Address + 0x10	Global Constant	Constant C_G from equation (2.2)
Base Address + 0x14	Local Constant	Constant C_L from equation (2.2)
Base Address + 0x18	Current Fitness	Returns the current fitness function evaluation as calculated by equation (2.3)
Base Address + 0x1C	Global Fitness	Returns the best value of the fitness function across all iterations

Table 3: Address Range Definitions

Memory Range	Name	Function
AR0	NN Inputs	Inputs to the Neural Network
AR1	NN Outputs	Outputs of the Neural Network
AR2	Fitness Bitmap	Bitmap used to specify the "Target Area" of Figure 1
AR3	PSO Global Outputs	Returns the 27 inputs values found by the PSO algorithm
AR4	Initial Particle Locations	Specifies the starting position of the particles
AR5	Initial Particle Velocities	Specifies the starting velocity of the particles
AR6	Maximum and Minimum Positions	Specifies the maximum and minimum values of the PSO input space
AR7	Max Velocity	Specifies the maximum velocity of the PSO equation (2.2)

via software memory accesses running on the PowerPC. The memory was arranged such to maximize utilization of the allocated space and the transfer speed. The OPB protocol operates with 32-bit data operations. However, most of the data in the NN and PSO algorithms is 16-bit. In many cases, this resulted in the concatenation of the data where two 16-bit vales were concatenated together to form a single 32-bit data value. This allowed for higher transfer speeds to memory where two values could be sent at once. In addition, memory is much more compact without wasting the other 16-bits.

The registers controlled starting the PSO algorithm, indicating when the algorithm had completed in addition to manipulating other PSO design parameters including the number of particles and iterations for the search. The memory ranges included values

that gave initial placement of the particles' position and velocity, maximum and minimum values for the position, and maximum velocity used by the update equations. The memory ranges also returned the output global best position at the conclusion of the PSO search. Figure 36 shows the connection of the register and address range memory with the various hardware components.

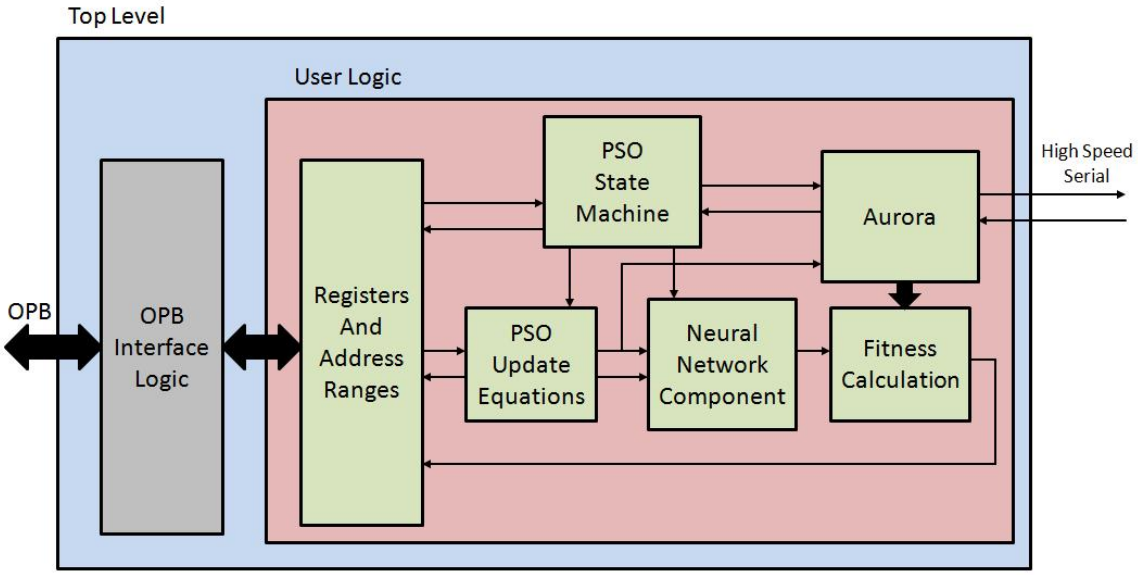


Figure 36: High Level Hardware Block Diagram

Particle Swarm Optimization Hardware Design

The particle swarm optimization (PSO) unit consists of four principle units, the PSO controller, the PSO Update Calculator, the Fitness Calculator, and Neural Network hardware unit pictured in Figure 36. The PSO unit adds a layer of data path and control to the neural network hardware component described in the next section. As shown, the data path consists of the PSO update equations, neural network hardware, fitness value calculator, and memory connected in a circular fashion, where the PSO state machine

controls the indexing of memory and activation of the neural network and PSO calculator at the correct instances in time.

The PSO Update Equations component calculates the update equations found in equations (2.1) and (2.2) and then stores them into memory. This is accomplished using a four stage pipeline shown in Figure 37. Since all of the nodes in a particular layer must be finished before the equations can update, any calculation time of the update equations

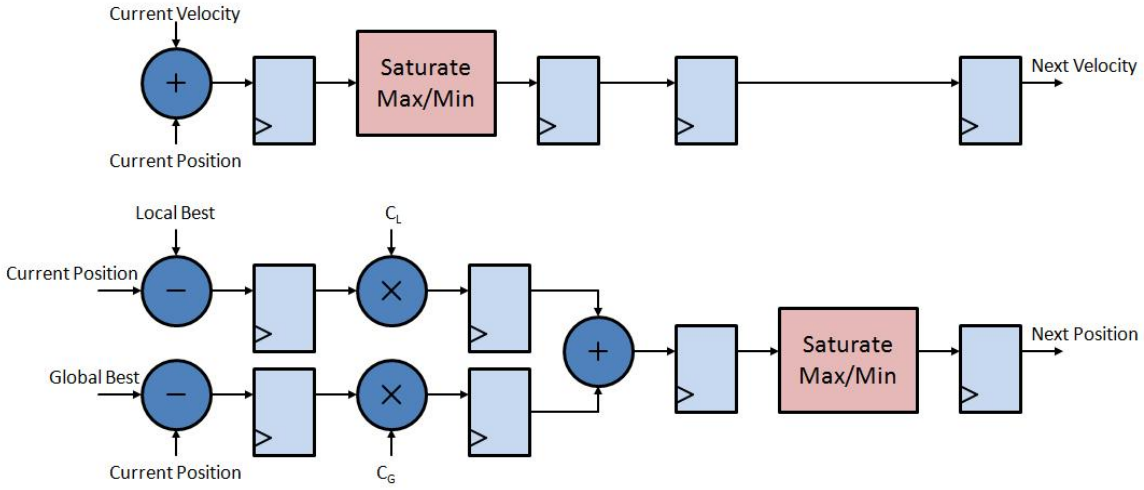


Figure 37: PSO Update Equation Pipeline

are added to the neural network latency. However, the update equations can be performed in parallel and as a result two nodes are updated in parallel. The PSO update calculator is initiated by asserting the start signal. Once started, this block indexes the needed particle and node from memory and calculates the update equations. Once valid outputs appear at the exit of the pipeline, a write enable signal is asserted to memory while an output index is used insert the updated equations into memory.

The fitness calculator sums the absolute value of only those nodes in the target area given by equation (2.3). The fitness calculation is determined by the contents of a

bitmap contained in address range two. In this address range, there is a single bit used to represent each output node. If the corresponding value in the bitmap memory contains a '1', then the corresponding output node of the neural network is considered part of the target area. However, if the corresponding value in the bitmap memory contains a '0', then the corresponding output node of the neural network is not considered part of the target area. The fitness value is calculated as the neural network outputs the output nodes. As a result, when the neural network is finished, the fitness value is known. This value is stored in memory and is used by the PSO update calculator in calculating the update position and velocity.

The PSO state machine coordinates the activation of the neural network, the PSO update calculator, updating the local and global best value and position, and loading neural network input memory with the appropriate particle data. Operation commences when the PSO start signal is asserted by software on the PowerPC. First, the PSO controller loads the input memory of the neural network with the appropriate particle position. Next the neural network is initiated, runs, and terminates with a done signal. Afterwards, the local and global values and positions are updated from the fitness value. Lastly, the PSO update calculator is initiated and ran until it is done by providing the done signal. This process is repeated for each particle and iteration of the particle swarm optimization algorithm.

Neural Network Hardware Design

The Neural Network unit is composed of multiple subunits. These smaller units are the node counter, node calculator, squashing function, and various registers that are depicted in Figure 38. Because of the limited number of FPGA multipliers on the Virtex-

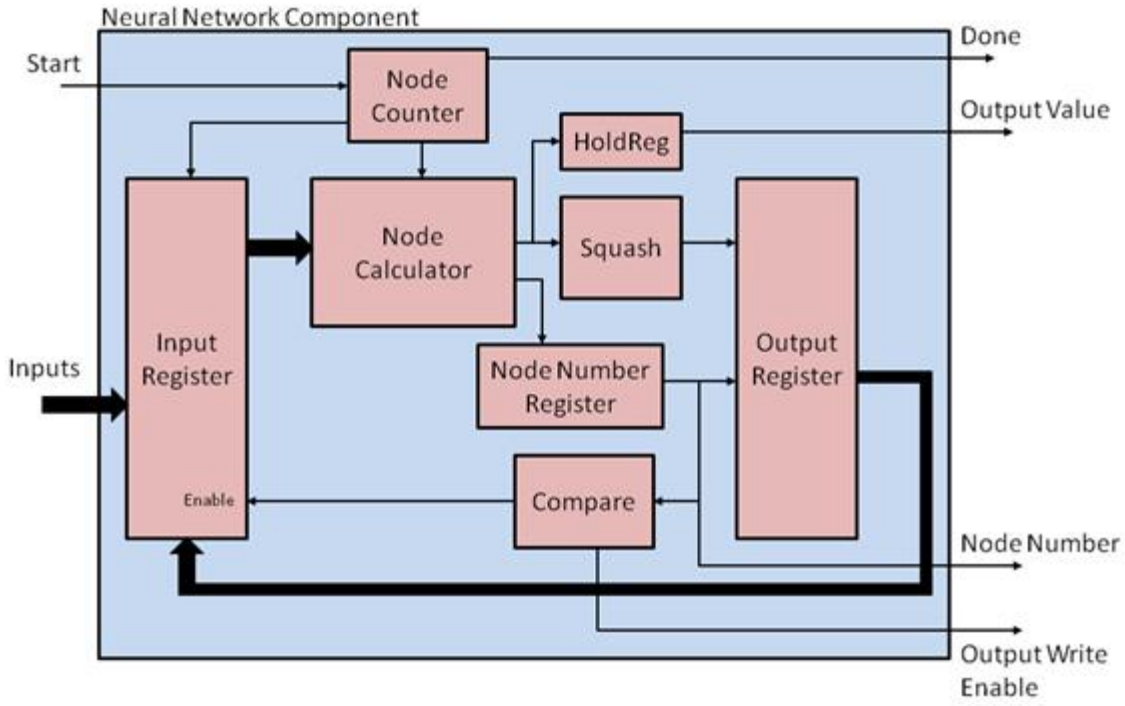


Figure 38: Neural Network Component

II Pro in the single XUP-V2P configuration, multiple nodes cannot be calculated simultaneously in parallel. Instead, the calculation of each node is pipelined such that after five clock cycles of latency, a node of the neural network is calculated every clock cycle. However during the network layer transitions, the pipeline will need to finish execution of the previous layer before calculation may begin on the next due to pipeline latency.

The node counter is the Moore state machine based controller of the neural network. Given the signal to start, the node counter increments from zero to 1,359 once per clock while pausing for five clocks at 40, 90, and 160. This pausing behavior is due to the hidden layers and a calculation latency of five clocks for each layer. Finally, the node counter is also responsible for generating the done control signal when the neural network calculation has completed.

The node calculator takes as inputs the node count and the sixteen bit fixed-point values of the previous layer. In the case of the input layer, the twenty seven inputs are the inputs to the node calculator. Embedded within the node calculator unit are located the fixed-point weights of the neural network. These weights are arranged such that the node count can index the needed weights in parallel. Using the dedicated 71 18X18 multipliers of the Virtex-II Pro, these weights are sign extended to 18-bits and multiplied by the 18-bit signed extended inputs. Finally, all of the multiplier products are summed using a pipelined adder tree and truncated into a 32-bit value. On the output of the node calculator, the node number is outputted along with the corresponding 32-bit summed value.

The squashing function takes as an input the 32-bit summed fixed point value of the node calculator output. This function squashes values according to the equation given in equation (6.1) and graphed in Figure 39. In a previous implementation by [6], the squashing function was implemented using a Taylor series approximation. However, for

$$Squash(x) = \frac{1}{1 + e^{-x}} \tag{6.1}$$

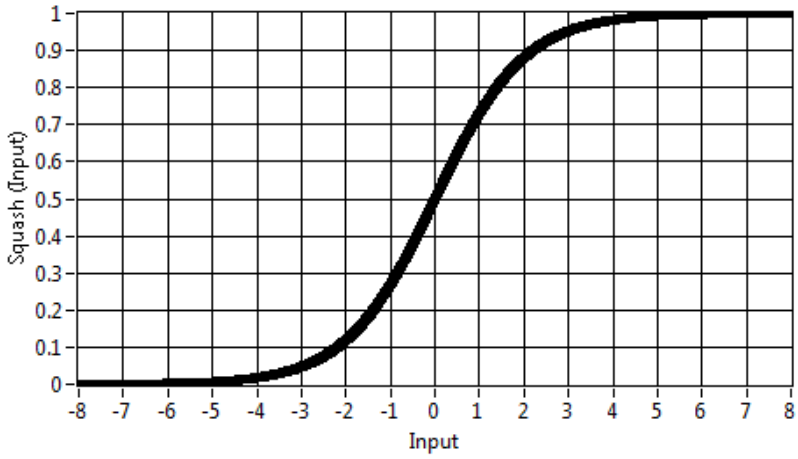


Figure 39: Squashing Function Graph

this implementation, this function was implemented in hardware using a look-up table and two comparisons in order to reduce the pipeline latency. Any input value less than negative eight are given an output value of zero, and any input value greater than eight is given an output value of one. Input values between negative and positive eight are given an output corresponding to the 4KB, 4,096 entry lookup table. While saturating the values outside the negative eight to eight interval and using discrete lookup values has the potential to produce numerical errors, the actual squashing function has no significant errors [6]. The squashed output value of the squashing unit is the final value of the node. Because the squashing adds an addition step to the pipeline, an external register named "Node Register" is used to sync the node value with the appropriate node number.

Last of all, there are two sets of registers, the input registers and the output registers. The output registers are responsible for holding the outputs of the current layer (the layer under calculation). The output registers store the value of the squashed output value (neural network node value), and is indexed by a register index number. The register index number is derived from the node number by subtracting zero, 40, 90, or 160 depending upon if the current layer is the first, second, or third hidden layer or the final output layer. This enables hardware reuse of the registers. The input registers are responsible for providing the previous layer nodes as inputs to the node calculator. The values of the input registers remain static except when the layer of the neural network changes. Upon the event of a layer transition, the contents of output registers are transferred to the input registers.

Aurora

Aurora is a high speed point to point communication protocol created by Xilinx that in the case of the XUP-V2P development system uses SATA electrical characteristics. This protocol is provided using the Xilinx Core Generator and has many customizable features. For this hardware implementation, a full duplex, two byte streaming method was used. Using a 100 MHz reference clock, this configuration is capable of sending 16-bits of data per clock (100 MHz) which has a 2.0 Gb/s data rate when the overhead of clocking compensation and 8B/10B encoding is included. On each end of the streaming protocol, state machines were created in order to control the content sent and received. The data has a 38 clock latency between the send on one side and the receive on the other. This data latency was one of the principle limiting factors when increasing the number of linked XUP-V2Ps as discussed in chapter seven. In addition, the number of SATA connections on the XUP board was limited as discussed in chapter three.

External Peripheral Logic Design

External Peripheral logic was created when using the Aurora protocol to expand the hardware design across multiple XUP-V2P development systems. In this case, slave logic was created that worked in conjunction with the main master EDK peripheral logic to place in parallel the calculation of the neural network. This peripheral was designed to take part of the neural network calculation load off of the main EDK peripheral. Figure 40 illustrates the slave architecture.

During execution of the forward Neural Network, the EDK master peripheral sends the previous layer information to the slave via Aurora. Once the entire layer has

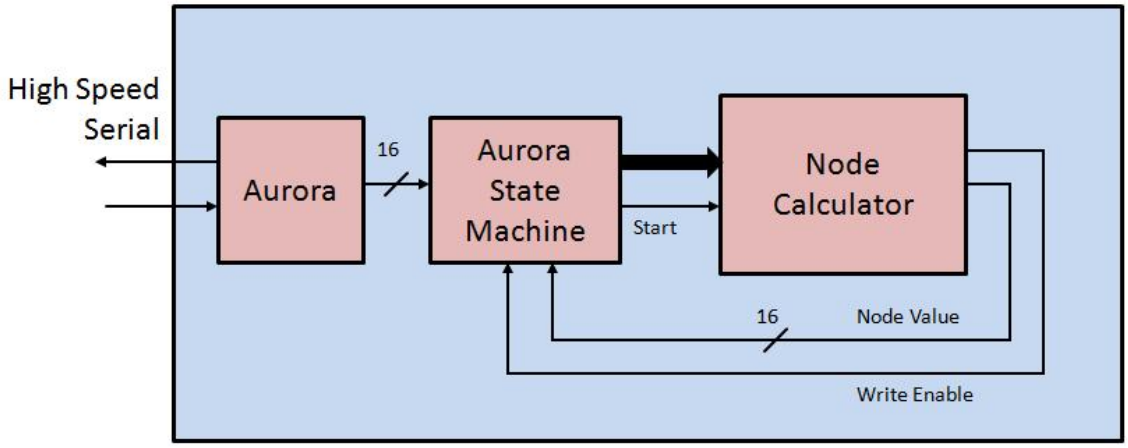


Figure 40: External Peripheral Logic Block Diagram

been sent, the slave peripheral starts calculating a selection of the next Neural Network layer. Finally, the slave peripheral sends its portion of the next layer back to the EDK master peripheral after it has finished its calculations.

The total size of the selection sent to the external peripheral logic varies on the number of slave systems available due to pipeline and communication latencies. In each case, the selection size was chosen to optimize final calculation time. Because each layer inside the neural network must be completely calculated before calculation can begin on the next, the process above has to happen for each layer of the neural network. As a result, communication latencies occur for both the receiving of the previous layer data from the master and the transmission of the new layer to the master for each layer of the neural network. This creates a diminishing percent speed up of the final system for each additional slave processing unit added to the system. See chapter seven for a full analysis of this network topology and the effects of latency on the total algorithm speed-up.

PXIe Implementation

The PXIe architecture implementation using the PXIe-7965R is very similar to the XUP-V2P implementation in structure. Like the XUP-V2P implementation, the PXIe architecture has NN and PSO components. However, the PXIe architecture is written using LabVIEW FPGA and does not need wrapper memory logic in order to interface to the software environment. Instead, all host software to FPGA hardware communication occur using controls and indicators on the FPGA that appear as registers in the host software VI. All of the LabVIEW FPGA code used in the implementation of the neural network inversion algorithm is given in appendices F and G. In addition, the communication links are established using Peer-to-Peer (P2P) streaming over the PXIe bus rather than using Aurora. Finally, the dual PXIe-7965R PXIe implementation used the parallel particle segmentation scheme while the multi-board XUP-V2P architectures made use of the parallel layer implementation.

Particle Swarm Optimization Hardware Design

Given in Figure 41 is the high level block diagram for the PXIe implementation. The algorithm starts when software asserts the “Start” signal to the “PSO State Machine”. Once started, the “PSO State Machine” indexes the “PSO Memory” into the update equations. Once the equations have been updated, the updated values are stored back into memory and are also driven as inputs to the neural network calculation. Then the neural network calculation begins computing once the “PSO State Machine” has asserted the neural network start signal. As the neural network finishes its calculation, the output layer values are passed into the “Fitness Calculation” VI one output node at a time. As was the case in the XUP-V2P implementation, a bit-map is used to indicate to the fitness

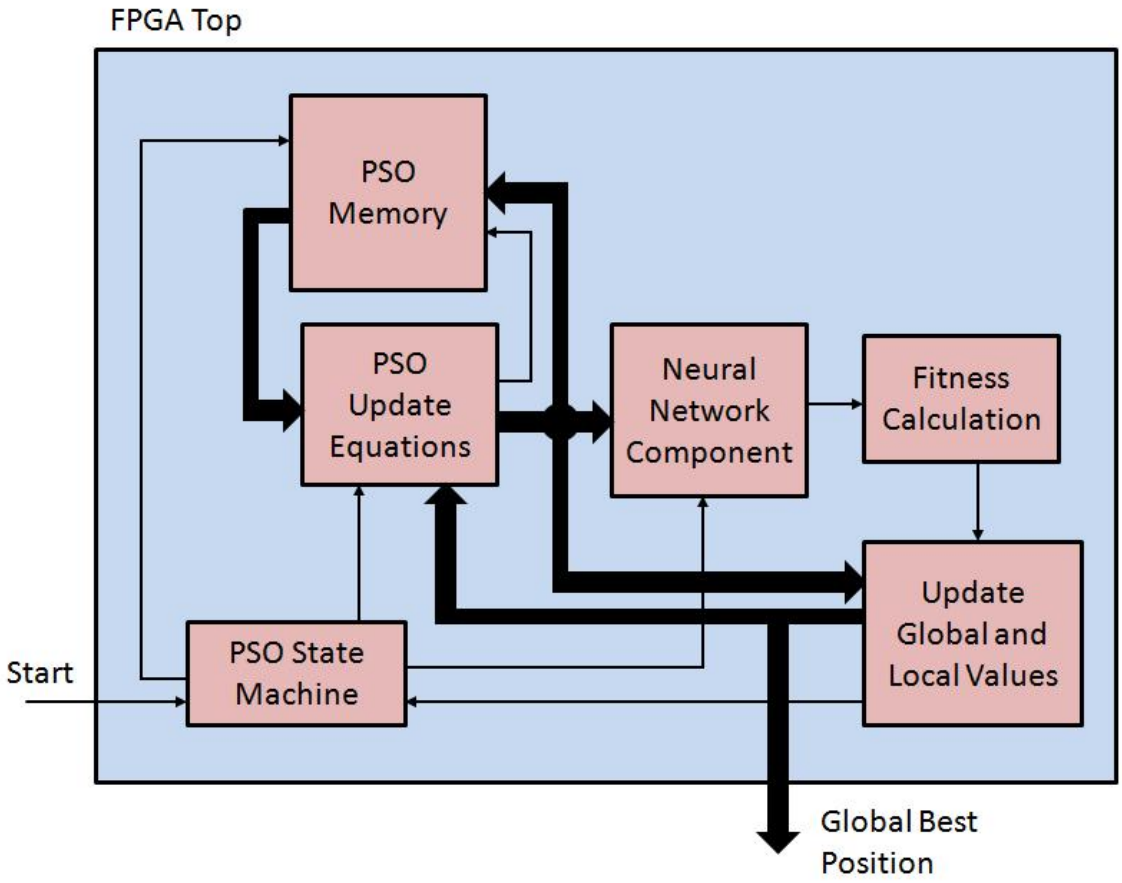


Figure 41: PXIe High Level Block Diagram

calculation when an output node is counted as part of the "Target Area". The value of a one indicates that the output node is to be counted, and the value of zero indicates that the output node is not to be counted. Once the fitness function has been evaluated, the fitness value is passed into the "Update Global and Local Values" component. This component is responsible for maintaining the best local and global fitness values in addition to the locations of each global and local best position. When a new fitness value that is superior to all previous fitness values is found, the "Update Global and Local Best" VI stores the fitness value for later comparisons. In addition, it also updates the best position by

storing the current neural network position. Finally, this process is repeated for each particle and iteration in the PSO algorithm.

Neural Network Hardware Design

The neural network design for the PXIe architecture is nearly identical to the XUP-V2P implementation. Shown in Figure 42 is the LabVIEW FPGA implementation for the neural network component in Figure 41. The neural network calculation begins with the assertion of the "Start" signal shown at the top left of Figure 42. This signal

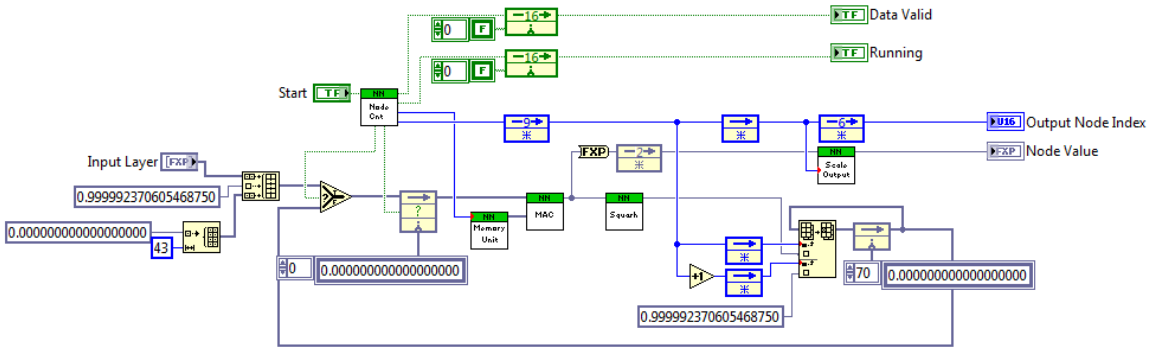


Figure 42: PXIe Neural Network Calculation

initiates the neural network state machine that is used to index the neural network weight memory and to also control the input and output register's indexes. Once initiated, the state machine latches the input layer nodes given as inputs to neural network VI. Then the memory weights are indexed, the input layer is multiplied by the neural network weights, the products are summed and squashed, and the result is stored into the output registers. Once all of the first hidden layer nodes are computed, the input registers are updated to the value of the output registers. Then the process repeats to calculate the second and third hidden layers. On the last iteration (output node iteration), the output nodes are calculated and then scaled before being presented as the output to the VI. As

with the XUP-V2P implementation, the pipeline of the calculation must be flushed before the next layer calculation may begin. As a result, at each layer transition, the state machine pauses in order to flush the pipeline. In the case of the PXIe implementation, the pipeline is ten clock cycles long.

Peer-to-Peer (P2P) Communication

Peer-to-Peer (P2P) is a DMA link between FPGAs using the PXIe bus. Since the PXIe architecture used the parallel particle algorithm segmentation, much less information needed to be passed between the FPGAs. In the case of the XUP-V2P implementation, the complete previous layer was communicated to the secondary FPGA where part of the output layer was calculated and then returned to the master FPGA. This process was repeated for each layer of the neural network calculation and involved many data transfers. However, the PXIe architecture only needs to transmit its global fitness value and global position to the second FPGA only before the beginning of a new iteration. This was because each FPGA implements the entire neural network inversion algorithm and therefore does not need to transfer the layer information. In addition, the local best fitness value and position do not needed to be transmitted because each particle is local to its own FPGA. As a negative consequence of the particle segmentation strategy, extra hardware resources were needed in order to perform the PSO capabilities on both FPGAs whereas the XUP-V2P implementation using the parallel layer segmentation approach did not need to implement the PSO on the slave FPGA. Finally, this parallel segmentation approach has the added benefit that there is only one FPGA file/image because all FPGAs in the system are identical while the XUP-V2P system required a different FPGA for each board in the system.

Given in Figure 43 is the block diagram for the hardware used to control the P2P communication. This implementation is found internal to the "Update Global and Local Best" block from Figure 41. Once the fitness value has been evaluated in Figure 41, the

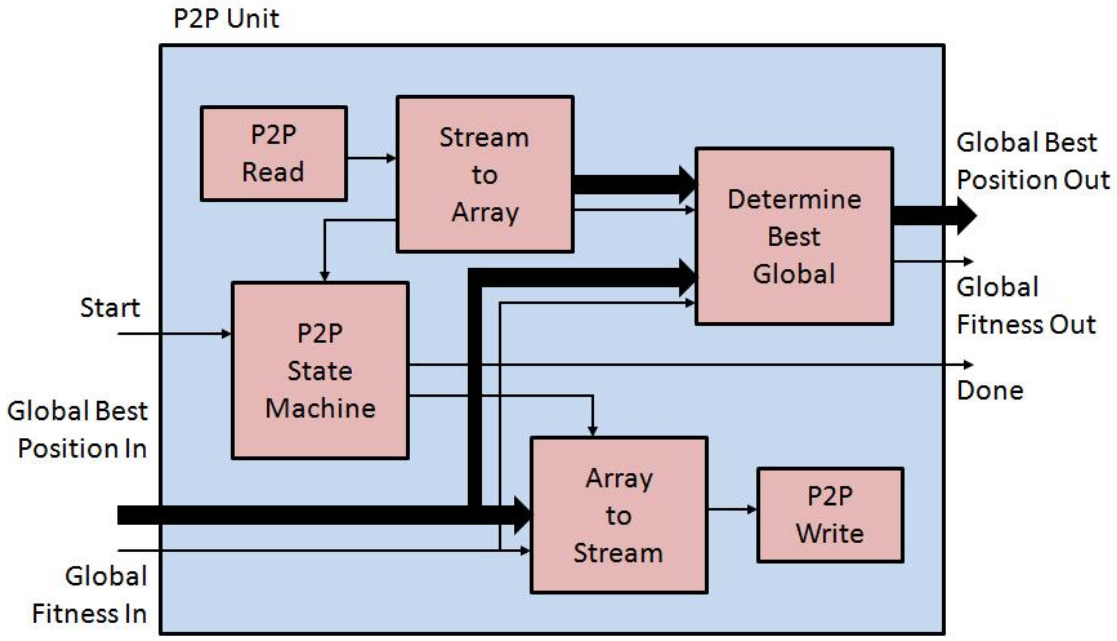


Figure 43: P2P Block Diagram

"Update Global and Local Best" block is used to update the best positions. Upon the execution of the last particle on the FPGA in any iteration, the "Update Global and Local Best" inputs the global best fitness and its associated global best position into the "P2P Unit" and asserts the "Start" signal.

Once start has been asserted, the "P2P State Machine" starts the transfer of global best fitness and global best position to the second FPGA. Before the values can be transmitted to the second FPGA using the P2P FIFO, the array value representing the 27 dimensional value of the input position must be serialized into a stream of 27 values. After all of information has been transmitted to the second FPGA, the "P2P State Machine" waits until all of the needed data has been received from the second FPGA.

Finally, after all of the data has been transmitted and received, a calculation is made on the data to determine if the global fitness value and position should be updated with the information from the second FPGA. If the second FPGA has the better value and position, the global best is updated on the current FPGA and the done signal is asserted. Otherwise, the “P2P Unit” outputs its current global best and asserts done.

CHAPTER SEVEN

Design Considerations

Each of the hardware configurations has its advantages and disadvantages. Below is a discussion of many advantages and disadvantages of the various hardware configurations. Tables summarizing the results are presented at the end.

Latency vs. Pipelining

Pipelining digital logic inside of hardware implementations is very common, and it is the typical solution for either meeting the desired clock rate needs or achieving the maximum clock rate possible. When a particular clock rate must be achieved, calculation speed up is optimal with the fewest pipeline stages while maintaining timing closure. However, in the case of maximizing the computational speed-up, there are classes of algorithms that are impacted negatively due to pipelining. While increasing the total clock speed via pipelining may always seem like a positive outcome, increased pipelining yields increased latency, an undesired new serial component of the design. The question then is to find the class of algorithms that are positively impacted and to find the optimal number of pipelining stages for a given algorithm.

Speed-up is defined as the ratio of the previous total calculation time to the present total calculation time. In equation (7.1), T_{old} represents the previous clock period, and T_{new} represents the present clock period; P represents the total number of pipeline stages added in order to change the clock period from T_{old} to T_{new}, and L is the total execution length without consecutive data dependencies. The speed-up needs to be

greater than one for pipelining to increase the computation speed. If the clock period increase is expressed as clock speed-up (N) given in equation (7.2), equation (7.1) can be rewritten in the form of equation (7.3). It then follows in equation (7.4) that the total number of pipeline stages must be less than the product of the execution length (L) and the clock speed-up (N) minus one for pipelining to increase the computation speed.

$$SpeedUp = \frac{Total\ Execution\ Time_{old}}{Total\ Execution\ Time_{new}} = \frac{L * T_{old}}{L * T_{new} + P * T_{new}} > 1 \qquad (7.1)$$

$$N = \frac{T_{old}}{T_{new}} \qquad (7.2)$$

$$SpeedUp = \frac{L * N}{L + P} > 1 \qquad (7.3)$$

$$P < L * (N - 1) \qquad (7.4)$$

Assuming that the pipeline stages are placed in optimal locations such that all timing paths between pipeline stages are equal, pipelining always increases N until $T_{new}=T_{min}$ where T_{min} is the minimum clock period of the hardware target. Once this critical limit is reached, there is no benefit to adding additional pipeline stages. However, running with the highest possible clock frequency is not always the clock rate for maximum algorithm speed-up. Because of the various electrical propagation times of FPGAs, the number of pipeline stages needed (P) will increase at a faster rate than the clock speed-up (N).

In Figure 44, there are four look-up-tables (LUTs) between two flip-flops to create the period constraint for T_{old}. T_{co} represents the clock-to-out time of the flip-flop, T_{route} is the routing propogation delay, T_{prop} is the propagation time through each the LUTs, and T_{setup} is the setup time needed before the clock of the destination flip-flop in

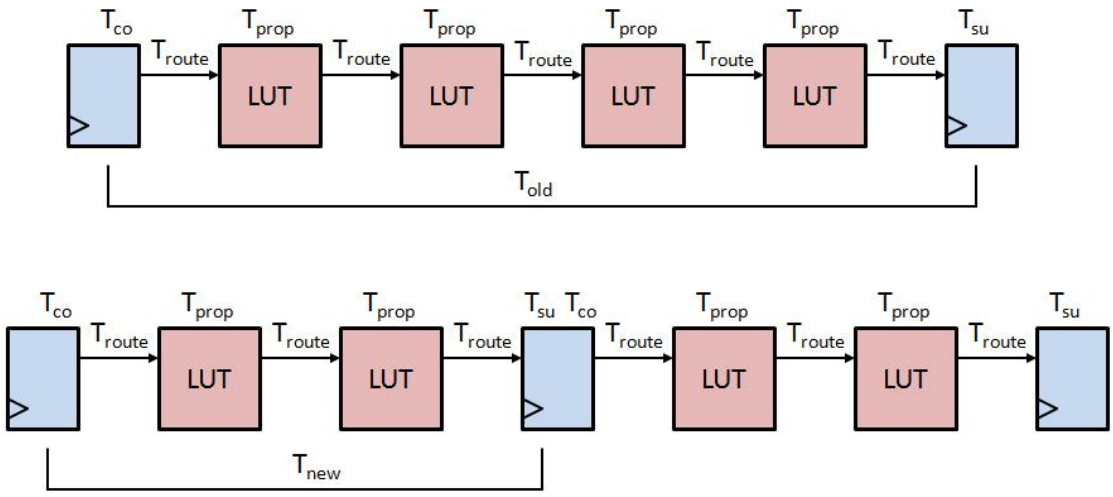


Figure 44: Pipeline Example

order to guarantee timing closure. If a flip-flop is placed after the first two LUTs, then T_{new} is the new flip-flop to flip-flop propagation time assuming the propagation time between the first two flip-flops is equal to the second two flip-flops. As a result of this new flip-flop, many of the propagation times are split up, but additional T_{co}, T_{setup}, and T_{route} times are added to the system. Equations (7.5) and (7.6) give the calculation for the old and new clock periods where #LUTs is the number of LUTs needed, or levels of logic needed, in order to compute the algorithm without pipelining. Equations (7.7) and (7.8) group the propagation times into those that are divided or split up by implementing pipelining, and those that are added as a result of pipelining.

Substituting equations (7.5) thru (7.8) into equation (7.2) yields equation (7.9) in order to relate the clock speed-up (N) as a function of the number of pipeline stages (P). By substituting equation (7.9) into equation (7.3), setting the first derivative equal to zero, and then solving for P reveals equation (7.10). This equation gives the optimal number of pipeline stages as a function of the number of look-up-tables (LUTs) needed for the algorithm, the various propagation times of the FPGA, and the length of the data

dependency free execution length (L). It is required that the number of added pipeline stages must be less than or equal to the number of LUTs required in the critical timing path for the algorithm minus one since there is a minimum of one LUT between each flip-flop. As a result, equation (7.11) indicates when the optimal number of pipeline stages is greater than the number of LUTs minus one indicating that increasing the number of pipeline stages to the point where there is only one LUT between flip-flops is the most optimal pipelined configuration for maximum algorithm speed-up.

$$T_{old} = T_{co} + T_{route} + \#LUTs * \left[T_{prop} + T_{route}\right] + T_{setup} \tag{7.5}$$

$$T_{new} = T_{co} + T_{route} + \frac{\#LUTs}{P+1} * \left[T_{prop} + T_{route}\right] + T_{setup} \tag{7.6}$$

$$T_{add} = T_{co} + T_{route} + T_{setup} \tag{7.7}$$

$$T_{split} = T_{prop} + T_{route} \tag{7.8}$$

$$N = \frac{T_{old}}{T_{new}} = \frac{\#LUTs * T_{split} + T_{add}}{\frac{\#LUTs * T_{split}}{P+1} + T_{add}} \tag{7.9}$$

$$P = \sqrt{\frac{\#LUTs * T_{split} * (L-1)}{T_{add}}} - 1 \tag{7.10}$$

$$L > \frac{\#LUTs * T_{add}}{T_{split}} + 1 \tag{7.11}$$

Note that while these equations show the number of pipeline stages needed in order to maximize the algorithm speed-up, they are only valid with unlimited FPGA resources where the routing times between LUTs and flip-flops are assumed equal. These equations are therefore only valid under ideal circumstances. Typically, routing delays increase with FPGA fullness and has a distribution of values. These equations are meant

to give a formalized justification for pipelining when L becomes moderate in size. In the case where L very is small, a microprocessor based implementation would be preferred in most cases. In addition, these equations don't take into account the effort level of the designer and the resource utilization of the FPGA which are dimensions that typically need to be optimized. However, these equations do show that in general for most problems, pipelining the design to its fullest extent gives the greatest computation speed-up.

In the case of the neural network implementations, pipelining was used in the calculation of each node. Because each layer of the neural network must be completed in order for the next layer's computation to begin, the data dependency free execution length (L) is the number of nodes for each of the layers. Of the 40, 50, 70, and 1,200 nodes at each layer, the first hidden layer with 40 nodes represents the smallest therefore worse case execution length. The logic for computing a single node requires one level of logic for multiplication, at most $\log_4(70)$ addition stages (the Virtex-II Pro contains four input LUTs), a final stage to implement a Block-RAM based look-up-table for the squashing function and an addition pipeline for the double buffer. Since L equals 40, T_{add} and T_{split} are approximately equal, and the maximum number of pipeline stages is no greater than seven, the most optimal amount pipelining occurs when there is a single level of logic between all flip-flops.

Communication Latency and Throughput

Once an FPGA design has expanded beyond the capabilities of one FPGA, it becomes necessary to increase the number of FPGAs to perform the calculation. Assuming that the split calculations running on each FPGA are not autonomous,

communication between the FPGAs is required. The addition of communication links always requires some time to travel between FPGAs and will therefore always add a serial component to an algorithm's calculation. Unlike pipeline latency which is typically small, the time to communicate information from one FPGA to another through communication links can be very large. This section will describe some of the various methods that exist to communicate between FPGAs, and the section on system topology will analyze the speed up from various possible topological configurations.

The first method is a simple direct parallel method. For this method, the FPGAs are connected via an N-bit and M-bit parallel bus interface shown in Figure 45. This method is quick, easy, and very effective. It has very low latency since it does not require communication overhead, and the total data throughput is governed by the bus bit

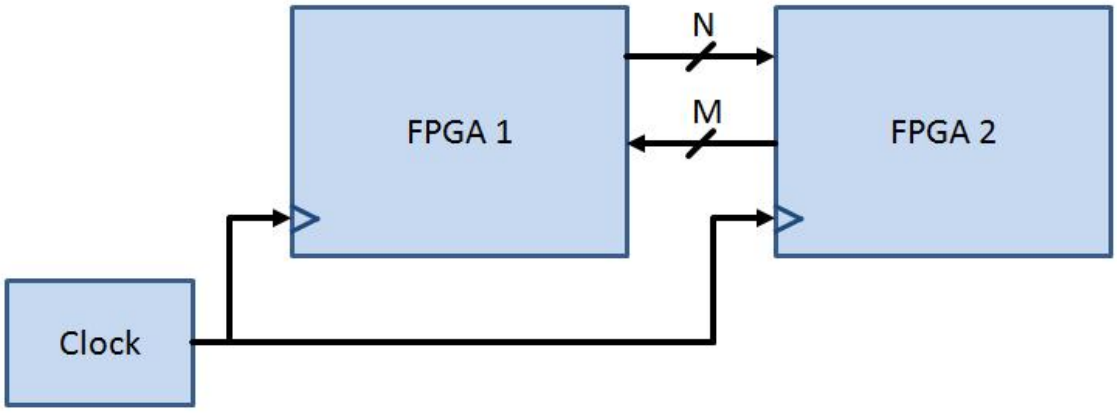


Figure 45: Parallel Direct Connection

width and the FPGA clock rate. However, this method requires many more FPGA pins than other methods, and more importantly, the connected FPGAs must be synchronous to one another. As a result, the number of total connections is greatly hindered, and the clocking of the boards becomes more difficult. This architecture is also very static

because it is not easy to grow or shrink the total number of FPGAs in the architecture. While this platform may be desirable for a single customized solution assuming that it is not limited by the number of inter-connects, it is not well suited for a generalized computing platform.

The second method uses serial communication links instead of the parallel bus previously described as shown in Figure 46. This was the method used to implement the multi-board XUP-V2P solutions using Aurora. While this method is more difficult to implement, it allows for different clocks (not frequency locked but approximately the

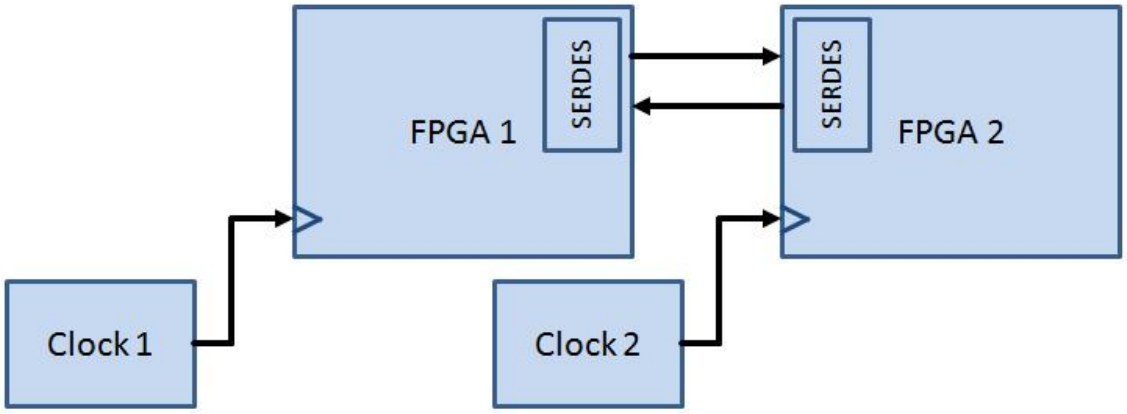


Figure 46: High Speed Serial Direct Connection

same frequency within a specified tolerance) for each FPGA. This is not an advantage if all of the FPGAs are contained within a single PCB; however, when each FPGA is contained within its own PCB, as was the case with the XUP-V2P development boards, this greatly reduces the need for creating a common clocking architecture in order to make the boards synchronous. In addition, there are many fewer electrical pin connections. Even though there are still a limited number of high speed serial links per FPGA, the maximum data transfer using all of the serial links is approximately equal,

depending on the exact FPGA, to the maximum data transfer using all of the available user I/O pins. This is an advantage even if all of the FPGAs are contained within a single PCB because it greatly eases trace routing congestion of the PCB. In the case of multiple PCBs, this greatly reduces the total number of electrical connections. Lastly, by using the serial links of the FPGA for intra-FPGA communication, more of the normal user I/O pins are available for connecting to external peripherals.

All of these advantages come at the cost of increased communication latency and added resource cost to the FPGA. In the case of Aurora, 38 clock cyclers of latency are seen in addition to the total data transfer time. These 38 clock cycles waste valuable computation clock used to compute the algorithm. In addition to latency, valuable FPGA resource are used to perform the data conversions from parallel to serial on the transmit side and serial to parallel on the receive side in addition to clock recovery and error detection/correction. Fortunately, many of the needed logic resources are hard IP within the FPGA and don't waste the configurable resources. But not all of the logic is contained within the hard IP; therefore, there are still some wasted reconfigurable FPGA fabric resources.

The final method uses a common bus to communicate between FPGAs as shown in Figure 47. This was the method used to implement the multi-board PXIe configurations. Of all of the communication types, the bus type is the most flexible where a variable number of processing cards can easily connect to the common bus. In addition, since there is only one connection, the total pin count is kept to a minimum. However, this method has the highest communication latency and the highest resource usage due to the implementation of the bus protocol's specification. The measured

values for the PXIe dual PXIe-7965R configuration were 3.2 µs, and Figure 48 provides additional bus throughput and latency metrics for a wide variety of bus standards. In addition, the data rate, or throughput is variable depending on the level of bus traffic at

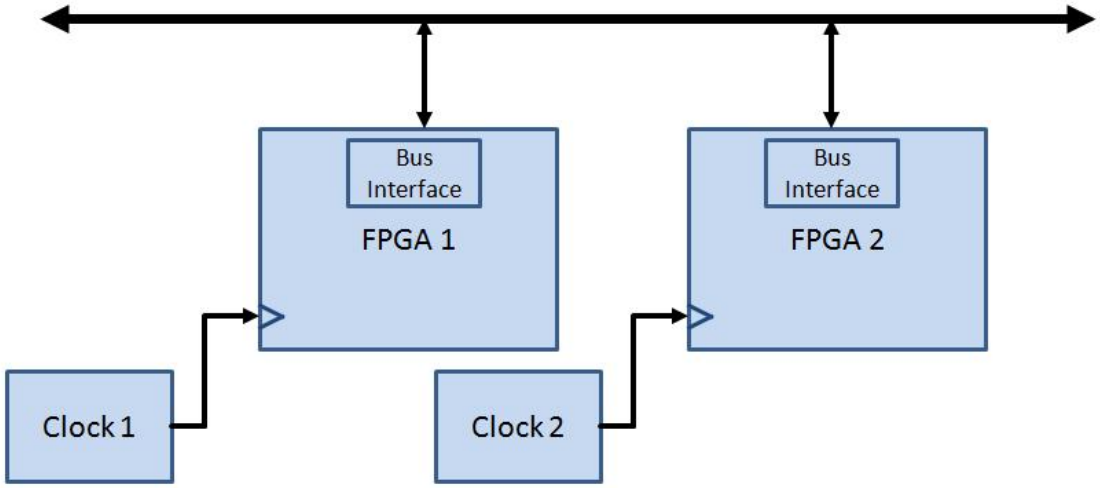


Figure 47: Bus Connection

any given time. While controlling the bus traffic in the system can reduce or completely remove this variability, this decreases some of the gained flexibility by using this connection type. Table 4 presents a summary of the FPGA connection types.

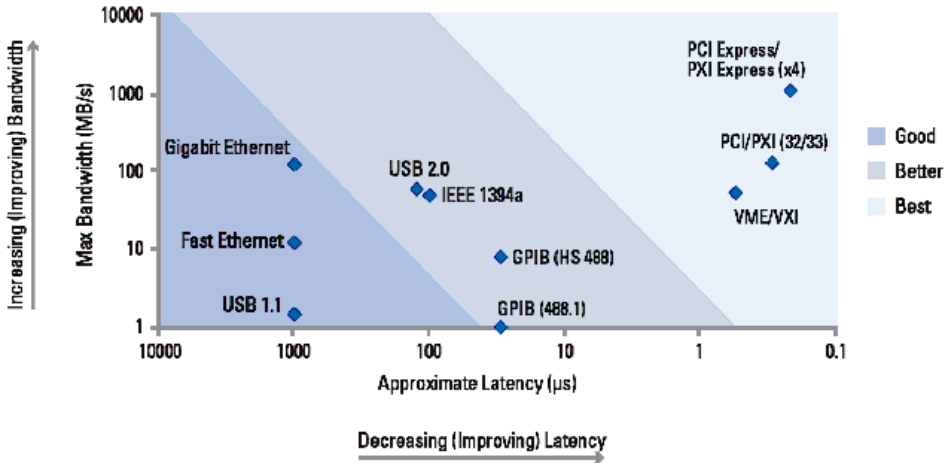


Figure 48: Bus Throughput and Latency

Table 4: Connection Type Comparison

Connection Type	Latency	Data Rate	FPGA Resources	Clocking	#Pins	Flexibility
Parallel	Low	Fixed	Low	Frequency Locked	High	Low
Serial	Medium	Fixed	Medium	Same Frequencies	Medium	Medium
Bus	High	Variable	High	Bus Provided	Low	High

If the only criteria for the best system is maximizing speed-up, then the system with the lowest latency and highest data throughput is desired. It then follows from Table 4 that the best connection type is the parallel connection type. However, the parallel connection type suffers from needing many FPGA pins for communication, and as a result fewer systems can be connected together. The next best option for maximizing speed-up is the serial connection type. While the latency is not as low as the parallel connection type, it is still much lower than the bus connection type. The latency of the Aurora connection was 38 clock cycles which was 0.38 microseconds using a 100MHz clock. In contrast, the PXIe latencies were 3.2 microseconds while most other bus connection types given in Figure 48 are much longer. However, FPGAs still only have a limited number of high speed serial inputs/outputs. Finally, the bus connection type is used as the last option, but as the number of FPGAs in the system becomes large, the latency and data throughput for each FPGA becomes worse due to communication congestion. Ultimately, if the number FPGAs in the system must be large to accommodate the needed algorithm, in general the best solution would result from a combination of all of the connection types. In a complex problem, the needs of each

piece of data in the system are typically not the same. Some data may need lower latency (parallel connection type) and other data might need the largest data throughput (multiple serial connection types) while still other pieces of data may need the flexibility of communicating to all of the FPGAs without strict latency and throughput requirements (bus connection type).

System Topology

Assuming that the hardware system does not use a bus connection type, there are many possible system topologies. These system topologies can be broken into two basic types, a star structure and a ring structure. All other topologies are variations on these two basic types. Underlying the speed-up analysis for both topologies is the goal of speeding up the calculation of an algorithm segment that exists between data dependencies. It is assumed that between these data dependencies, calculations can be performed in parallel and thus capable of speed-up using multiple FPGAs. As a result, this analysis focuses on the best method to move data to various FPGAs within a network of FPGAs for parallel processing, not how best to pipeline an FPGA network.

The first system topology is the star connection method. This method connects multiple FPGAs together with a node at the center branching out to multiple nodes, each directly connected to the center node. An example of this configuration can be seen in Figure 49. Given in equation (7.12), the relationship is given between the calculation time of the center node and a satellite node (a node connected to the center node) in order to achieve the optimal calculation time where T_{Center} is the calculation time of the center node, T_{Comm} is the one way communication time between the nodes, and T_{FPGA2} is the calculation time of a satellite node. Assuming that the cumulative calculation time

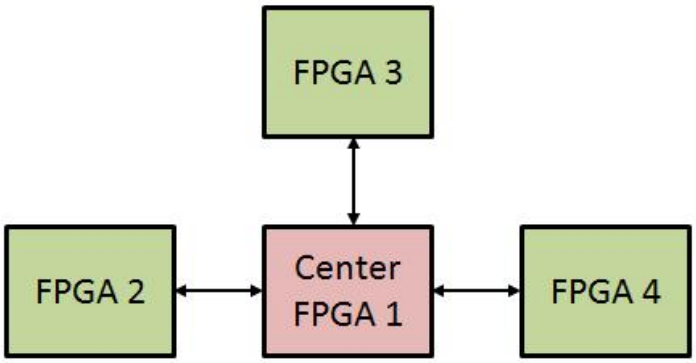


Figure 49: Star Topology

remains equal when an algorithm is split up among multiple nodes, equation (7.13) can be asserted where $T_{Algorithm}$ is the total calculation time of the algorithm before being split up and performed in parallel on multiple FPGAs. Finally, equation (7.14) asserts that each satellite node has the same calculation time as every other satellite node. Using these three equations, the total algorithm speed-up is given by equation (7.16).

$$T_{Center} = 2 * T_{comm} + T_{FPGA2} \tag{7.12}$$

$$T_{Center} + T_{FPGA2} + T_{FPGA3} + \cdots + T_{FPGA\,N} = T_{Algorithm} \tag{7.13}$$

$$T_{FPGA2} = T_{FPGA3} = \cdots = T_{FPGA\,N} \tag{7.14}$$

$$T_{Center} = \frac{T_{Algorithm}}{N} + 2 * T_{Comm} - 2 * \frac{T_{Comm}}{N} \tag{7.15}$$

$$SpeedUp = \frac{T_{Algorithm}}{T_{Center}} = \frac{T_{Algorithm}}{\frac{T_{Algorithm}}{N} + 2 * [T_{Comm} - \frac{T_{Comm}}{N}]} \tag{7.16}$$

This equation is always increasing with the number of nodes in the system, where the speed-up reaches its maximum speed-up with an infinite number of nodes given by equation (7.17). In addition, since speed-up only occurs when equation (7.16) is greater

than one, equation (7.18) gives the requirement in order to see a positive impact on total algorithm speed-up.

$$\lim_{N\to\infty} SpeedUp(N) = \frac{T_{Algorithm}}{2 * T_{Comm}} \tag{7.17}$$

$$T_{Algorithm} > 2 * T_{Comm} \tag{7.18}$$

The ring structure given in Figure 50 distributes segments of the entire calculation around the ring to other nodes where each node performs part of the calculation in parallel. Equation (7.19) gives the relationship between the starting node and any other node in the ring assuming an equal amount of data is sent between each node in the

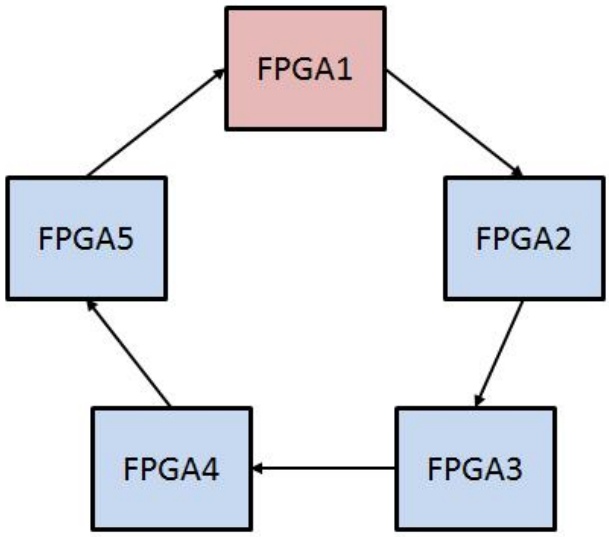


Figure 50: Ring Topology

network, where N is the number of nodes in the network for optimal execution time. As was the case for equation (7.13), equation (7.20) makes the assumption that the cumulative calculation time remains equal when an algorithm is split up among multiple nodes. Finally, equation (7.21) states that all nodes that are not the starting node of the ring network have the same calculation time. Combining these three equations, the

$$T_{FPGA1} = N * T_{comm} + T_{FPGA2} \tag{7.19}$$

$$T_{FPGA1} + T_{FPGA2} + T_{FPGA3} + \cdots + T_{FPGA\,N} = T_{Algorithm} \tag{7.20}$$

$$T_{FPGA2} = T_{FPGA3} = \cdots = T_{FPGA\,N} \tag{7.21}$$

$$T_{FPGA1} = \frac{T_{Algorithm}}{N} + (N-1) * T_{Comm} \tag{7.22}$$

$$SpeedUp = \frac{T_{Algorithm}}{T_{Board1}} = \frac{T_{Algorithm}}{\frac{T_{Algorithm}}{N} + (N-1) * T_{Comm}} \tag{7.23}$$

calculation time of the starting node is given in (7.22) with the total algorithm speed-up given in equation (7.23).

Using a ring structure is only beneficial when the speed-up ratio is greater than one. Therefore, equation (7.24) states the condition necessary in order to have positive speedup. By taking the first derivative of equation (7.23) and equating to zero, it was found that maximum speed-up results when the total number of nodes is given by equation (7.25) where (7.26) gives the maximum speed-up. If the communication time between nodes is expressed by (7.27), equations (7.25) and (7.26) reduce to equations (7.28) and (7.29) respectively.

$$1 < N < \frac{T_{Algorithm}}{T_{Comm}} \tag{7.24}$$

$$N = \frac{\sqrt{T_{Comm} * T_{Algorithm}}}{T_{Comm}} \tag{7.25}$$

$$MaxSpeedUp = \frac{T_{Algorithm} * \sqrt{2 * T_{Algorithm}}}{T_{Comm} * (2 * T_{Algorithm} - \sqrt{T_{Comm} * T_{Algorithm}})} \tag{7.26}$$

$$T_{Comm} = K * T_{Algorithm} \tag{7.27}$$

$$N = \frac{\sqrt{K}}{K} \qquad for\ 0 \leq K \leq 1 \tag{7.28}$$

$$MaxSpeedUp = \frac{\sqrt{K}}{2 * K - K\sqrt{K}} \tag{7.29}$$

In the case of the XUP-V2P implementation, the communication latencies would create critical problems for ring type communication structures. Because each previous layer is needed for calculation of the next layer, the previous layer would need to be distributed to all locations within the ring for parallel computation. In addition, because each layer of the neural network is a different size requiring different amounts of hardware resources, pipelining the design using a ring structure results in a sub-optimal utilization of resources.

Comparing the star topology against the ring topology, the star topology always produces a higher speed-up assuming each topology makes use of the same number FPGAs and has the same communication latency. However, the star topology requires too many electrical connections if the need arises for a massive number of interconnected FPGAs. In this instance, the star topology given in the example above can be extended to connect further satellite nodes to the existing satellite nodes. The speed-up improvements of this configuration are less than with directly connecting to the center node due to increased communication latency. However, these communication latencies would still be less than a ring structure given a large number of FPGAs. The primary reason for using a ring structure is to reduce the total number of connections. While it may seem that a ring structure is more flexible and therefore more general purpose, this is not the case when the assumption cannot be made that the total original algorithm

execution time is constant across the space of all possible algorithms. As a result, the configuration of the ring structure must vary with the algorithm length in order to maximize speed-up as shown by equation (7.25) thus negating any reason to use a ring topology.

Algorithm Segmentation

A problem must first be capable of parallel computation if it is to take advantage of parallel hardware. Many problems have some aspects which could benefit from calculations in parallel. However, many problems must first be restructured in order to take advantage of parallel computation. The neural network inversion problem was chosen because it was inherently parallel with multiple ways to exploit the parallelism. In addition, it is easy to see what parts of the problem are parallel and which parts have serial dependencies. This allows for evaluation of the hardware to be made from the neural network inversion problem.

As seen in previous sections, there are multiple issues that must be considered when implementing an algorithm parallel across multiple FPGAs. In most instances, there are tradeoffs to be made. However, properly segmenting an algorithm according to hardware limitations is one of the most beneficial ways to increase total algorithm speed-up. From the previous section, even when the most optimal topology is chosen for a particular algorithm, the issue of communication latency is still a limiting factor. If however, the algorithm can be segmented in order to reduce the total number of needed communication transfers, then this limiting factor can be reduced. The same is true for pipelining; if an algorithm can be segmented such that data dependencies between the

input data and data in the pipeline rarely exist, then the total speed-up due to pipelining increases.

For the neural network case study, the parallel layer and parallel particle segmentation schemes were implemented using multiple FPGAs. The first focused on speeding up the calculation of each layer within the neural network by sending the calculation for different nodes to different FPGAs. However, the second approach segmented multiple particles of the PSO algorithm. Even though the first implementation had the most optimal topology configuration, and the lower FPGA-to-FPGA communication latency using the high speed serial SATA connections, the PXIe bus architecture proved to be faster simply because of a superior method of segmenting the neural network inversion algorithm.

The parallel particle segmentation scheme was found to be superior to the other segmentation schemes because it reduced the amount of data that needed to be transferred, and because it also required less communication. Since this segmentation approach required less communication, the communication latency hardware limitation had less significance on the total computation speed. Therefore, had the XUP-V2P architecture used the parallel particle segmentation approach, it would have yielded the best computation time only by a slight margin.

Hardware Resource Utilization

Resource utilization of the FPGA is important because this directly influences the cost, size, and power of the overall system. In the case of FPGAs where there are multiple hardware resources, it becomes important to balance the resources. Once any one resource has been completely utilized, it becomes necessary to either restructure the

hardware computation or start using FPGAs in parallel. When implementing the parallel particle segmentation approach, more hardware resources were required on the second FPGA than were needed by the parallel layer segmentation approach. This was due to the parallel particle segmentation requiring the entire neural network inversion algorithm rather than just a subset of the neural network calculation that was needed in the parallel layer approach. However, because the parallel particle approach was more resource balanced than the parallel layer segmentation, it makes better use of resources even though the slave FPGA in the parallel layer segmentation used fewer resources.

Another resource tradeoff results from pipelining. The section on pipelining showed that maximum speed-up generally occurs when the clock speed is maximized. However, this comes at the increase cost of FPGA flip-flops. Finding the optimal point between resource utilization of flip-flops, and the needed to reach a faster clock speed is

Table 5: Resource Utilization Results

Implementation	Slices	Multiplies	18Kb BRAM
XUP-V2P Single	79%	55%	94%
XUP-V2P Double Master	71%	55%	72%
XUP-V2P Double Slave	17%	52%	54%
PXIe-7965R Single	48%	12%	34%
PXIe-7965R Double Master	58%	12%	35%
PXIe-7965R Double Slave	58%	12%	35%

very difficult to find and is typically found using an iterative process. Given in Table 5 are the resource utilizations in terms of percentage used for most of the implementations of the neural network inversion algorithm. The three board XUP-V2P implementation is not given because it matches the resource utilization of the two board implementation. Recall that the XUP-V2P implementations use a Virtex-II Pro VP30 FPGA while the PXIe-7965 uses a Virtex-5 SX95T FPGA.

Design Effort and Financial Cost

Large scale high volume commercial application typically need very low total system cost. As a result, a high emphasis is placed on optimization and reducing the needed hardware resources. At the other end are one-off single system prototypes where the engineering design time is the major contributor to the total cost, and thus less emphasis is placed on optimization. In the middle, a balance must be reached between the cost of each system and the cost of the design time.

The XUP-V2P solution while lower in cost than the National Instruments PXIe architecture took much design effort to accomplish the hardware implementation of the neural network inversion algorithm. One reason for this increased effort was due to the development of system pieces other than those that were not the hardware algorithm. This included connecting and configuring all of the various peripherals included with the XUP-V2P development system. It took about a month just to get the system working with a basic textual user interface, and two months to actually implement the algorithm in VHDL. In contrast, it only took five minutes to get the system working, and a few hours to perfect the interactive GUI using the PXIe architecture with LabVIEW FPGA. This

resulted in a much high percentage of the time allocated to developing the actual hardware algorithm.

The National Instruments solution using the PXIe hardware architecture provided the best platform for quickly developing a working system. The time needed to implement this system was only two weeks. Because the system and all of its peripherals required very little or no setup, nearly all of the development time was spent on implementing the neural network inversion algorithm. In addition, using LabVIEW FPGA in lieu of VHDL required only a quarter of the development time. While the LabVIEW FPGA implementation was slightly less optimal in resource utilization than using VHDL, the time savings more than compensated for resource utilization.

Table 6: Hardware Cost

Hardware	Retail Cost	Academic Cost	High Level Capabilities
XUP-V2P	$1,599.00	$399.00	Virtex-II Pro VP30, 2 PPC 405 Processors, 136 Multiplies
PXIe-7965R	$8,499.00	$7,649.10	Virtex-5 SX95T FPGA, 640 Multiplies
PXIe-1082	$3,299.00	2,569.10	8-Slot PXIe chassis
PXIe-8133	$5,649.00	5,084.10	1.73 GHz quad-core Intel Core i7-820 processor, 2GB DRAM

The financial cost difference of the two systems was significantly different, but the hardware capabilities are also very different. Presented in Table 6 are pieces of hardware used with their respective retail cost, academic cost, and high level capabilities. Shown in Table 7 is a summary of the total system costs and design time for each of the

implementations. In addition, since two very different FPGAs are used, the cost is also presented in terms of cost per multiply.

Table 7: Total System Cost

System	Design Time	Retail Cost	Academic Cost	Cost Per Multiply
XUP-V2P Single	3 months	$1,599.00	$399.00	$11.76
XUP-V2P Double	4 months	$3,198.00	$798.00	$11.76
XUP-V2P Triple	4.25 months	$4,797.00	$1,197.00	$11.76
PXIe System Single	2 weeks	$17,447.00	$15,702.30	$27.26
PXIe System Double	2.5 weeks	$25,946.00	$23,351.40	$20.27

Performance

The performance of the hardware implementations is presented in Table 8. All of the implementations calculate 100 particles for 1,000 iterations. The time is calculated from the when the PSO algorithm is initialized until the last particle finishes its last iteration. In addition, several other configurations have been calculated with their results given.

The calculated time for the "XUP-V2P Single" implementation is found by adding together all of the calculated nodes (40+50+70+1200) for a total of 1360 clock cycles. In addition, there is a five stage pipeline in the node calculation that is counted four times, once for each calculated layer, and a four stage pipeline for the update equations. Finally, the update equation takes ceiling(27/2) clock cycles to calculate

because two dimensions of each particle are calculated simultaneously. Since the clock rate is 100MHz, this results in a calculated time of 1.40 seconds when this process is repeated for 100 particles and 1,000 iterations.

Table 8: Computation Time Results

System	Algorithm Segmentation	System Topology	Calculated Time (sec)	Measured Time (sec)
XUP-V2P Single	Parallel Node	None	1.40	1.42
XUP-V2P Double	Parallel Node Parallel Layer	Star	0.85	0.88
XUP-V2P Triple	Parallel Node Parallel Layer	Star	0.68	0.72
PXIe System Single	Parallel Node	None	1.20	1.21
PXIe System Double	Parallel Node Parallel Particle	Bus	0.60	0.61
XUP-V2P Double	Parallel Node Parallel Particle	Star	0.70	X
XUP-V2P Triple	Parallel Node Parallel Particle	Star	0.48	X
XUP-V2P Triple	Parallel Node Parallel Layer	Ring	0.87	X
PXIe System Double Extreme	Parallel Node Parallel Particle	Bus	0.23	X

In the case of the "XUP-V2P Double" configuration in Table 8, the exact same analysis is used except that the 1,200 outputs nodes have been implemented in parallel across two FPGAs using the star topology. Since two way communication delay in this instance is 2*38+27 clock cycles. Using equation (7.15) the 1,200 clock cycles now becomes 652 clock cycles. The hidden layers of the neural network are not performed in

parallel because they do not satisfy equation (7.18). As a result, the final calculation for the "XUP-V2P Double" implementation is (40+50+70+652) + 5*4 + ceil(27/2)+4 clock cycles resulting in a total time of 0.85 seconds for 100 particles and 1,000 iterations.

The "PXIe System Single" from Table 8 was faster than the "XUP-V2P Single" because of the PXIe system used a 120MHz clock rather than a 100MHz clock. The PXIe single system utilized more pipelining in order to achieve a high clock rate. Had the "PXIe System Single" used a 100MHz clock, it would have had a total computation time of 1.45 seconds, 30 milliseconds slower than the XUP-V2P system due to the extra pipelining. When comparing the two board solutions, the PXIe system was faster because it used a superior algorithm segmentation scheme that minimized FPGA-to-FPGA communication and because it had a high clock rate. If the PXIe system had used a 100MHz clock, then the total computation time would have been 0.73 seconds, 20% faster than the 'XUP-V2P Double" system.

Finally, the PXIe architecture's Virtex-5 FPGA was never fully utilized. There were adequate FPGA resources remaining in the Virtex-5 FPGA to instantiate a duplicate neural network calculation VI. In addition, since the Virtex-5 FPGA has a higher maximum clock speed than the Virtex-II Pro, the algorithm could have achieved a clock rate of at least 160MHz if not faster with minimal extra pipelining. As a result, it is known with high confidence that a total computation time of 0.23 seconds is possible using the "PXIe System Double" hardware.

CHAPTER EIGHT

Conclusions

The implementation of the neural network inversion algorithm was implemented on two different hardware architectures using two very different development environments with three different algorithm segmentation methods. Each of these architectures had several advantages and disadvantages. The XUP-V2P implementations had dedicated low latency FPGA-to-FPGA communication links while also maintaining a very low total system cost as its key advantages. However, this platform required four times the development effort in order to create a working system. In addition, its UI interface was a primitive textual menu based system.

The PXIe system using LabVIEW FPGA development environment was a much more capable system that was also more expandable than the XUP-V2P system. In addition, it offered a rich graphical UI, took a quarter of the development effort when compared to the XUP-V2P, and was more suited to general purpose computing. However, the PXIe system costs significantly more than the XUP-V2P system, even when compared on a dollar per multiplier bases.

In addition to the hardware implementations, an analysis of pipelining, system topology, communication latency and throughput, and algorithm segmentation was given. In general, it is almost always better to pipeline a design to the fullest extent possible. While there are a few situations that are not suited to heavy pipelining, these situations are most likely not well suited for an FPGA implementation. In the analysis of system

topology, it was found that a ring structure is generally never the best topology when there are repetitive data dependencies within an algorithm. If the best performance is needed and the total number of connections needed is obtainable, then a star topology gives best implementation.

In the chapter seven section on communication latency and throughput, it was found that the parallel connection type would give the highest speed-up due to its low latency and high data throughput. However, this connection type was not scalable to a large number of FPGAs. Therefore, the best connection type depended greatly on the number of interconnects needed in a network of FPGAs. In order to maximize speedup in general, if there are a small number of FPGA interconnects, then the parallel connection method is the most optimal; but if the number of FPGAs in system is very large, then assuming that different pieces of data have different optimization requirements, then the optimal system is a hybrid of the parallel, serial, and bus connection types. However, if the goal of the system is to be a generalized computing platform, and if some throughput and latency are sacrificed, a high-speed bus connection is in general the better implementation.

Finally, properly segmenting an algorithm according to hardware limitations is, in many cases, the most effective performance enhancement. In the case of the hardware architectures presented, the FPGA-to-FPGA communication time was the major hardware limitation. In order to show the effects of different algorithm segmentations, a different algorithm segmentation method was used for each of the multi FPGA hardware architectures. The XUP-V2P architecture implemented the parallel layer method while the PXIe system implemented the parallel particle method. It was found that while the

XUP-V2P system had the lowest latency communication link, the PXIe architecture obtained the superior speed-up. In general, the parallel particle segmentation strategy was the best algorithm segmentation method because it decreased the frequency that data was communicated, thus minimizing the negative effects of the hardware communication. Had the XUP-V2P system used the same segmentation strategy, a comparable multisystem speedup would have been observed. Finally, the fastest implementation was able to achieve a computation time of 0.61 seconds using the dual FPGA PXIe hardware platform, while the fastest previous implementation was 1.8 seconds [6]. In addition, it was determined that a computation time of 0.23 seconds was achievable with some minor modifications to the PXIe hardware implementation.

APPENDICES

APPENDIX A

Single Board XUP-V2P VHDL Implementation

PSO_NN_Calc1.vhd

```
------------------------------------------------------------------------------
-- PSO_NN_Calc1.vhd - entity/architecture pair
------------------------------------------------------------------------------
-- IMPORTANT:
-- DO NOT MODIFY THIS FILE EXCEPT IN THE DESIGNATED SECTIONS.
--
-- SEARCH FOR --USER TO DETERMINE WHERE CHANGES ARE ALLOWED.
--
-- TYPICALLY, THE ONLY ACCEPTABLE CHANGES INVOLVE ADDING NEW
-- PORTS AND GENERICS THAT GET PASSED THROUGH TO THE
INSTANTIATION
-- OF THE USER_LOGIC ENTITY.
------------------------------------------------------------------------------
--
--
**************************************************************************
***
-- ** Copyright (c) 1995-2006 Xilinx, Inc.  All rights reserved.           **
-- **                                                                      **
-- ** Xilinx, Inc.                                                         **
-- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"
**
-- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING
PROGRAMS AND       **
-- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,
**
-- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS
FEATURE,       **
-- ** APPLICATION OR STANDARD, XILINX IS MAKING NO
REPRESENTATION          **
-- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF
INFRINGEMENT,     **
-- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY
REQUIRE      **
-- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY
**
```

```
-- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
**
-- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY
WARRANTIES OR        **
-- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM
CLAIMS OF       **
-- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS       **
-- ** FOR A PARTICULAR PURPOSE.                                    **
-- **                                                            **
--
**********************************************************************
***
--
-------------------------------------------------------------------------------
-- Filename:          PSO_NN_Calc1.vhd
-- Version:           1.00.a
-- Description:       Top level design, instantiates IPIF and user logic.
-- Date:              Fri Aug 10 14:04:13 2007 (by Create and Import Peripheral Wizard)
-- VHDL Standard:     VHDL'93
-------------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                  "*_n"
--   clock signals:                       "clk", "clk_div#", "clk_#x"
--   reset signals:                       "rst", "rst_n"
--   generics:                            "C_*"
--   user defined types:                  "*_TYPE"
--   state machine next state:            "*_ns"
--   state machine current state:         "*_cs"
--   combinatorial signals:               "*_com"
--   pipelined or register delay signals:   "*_d#"
--   counter signals:                     "*cnt*"
--   clock enable signals:                "*_ce"
--   internal version of output port:     "*_i"
--   device pins:                         "*_pin"
--   ports:                               "- Names begin with Uppercase"
--   processes:                           "*_PROCESS"
--   component instantiations:            "<ENTITY_>I_<#|FUNC>"
-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v2_00_a;
```

```
use proc_common_v2_00_a.proc_common_pkg.all;
use proc_common_v2_00_a.ipif_pkg.all;
library opb_ipif_v3_01_c;
use opb_ipif_v3_01_c.all;

library PSO_NN_Calc1_v2_00_a;
use PSO_NN_Calc1_v2_00_a.all;

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_BASEADDR              -- User logic base address
--   C_HIGHADDR              -- User logic high address
--   C_OPB_AWIDTH            -- OPB address bus width
--   C_OPB_DWIDTH            -- OPB data bus width
--   C_USER_ID_CODE          -- User ID to place in MIR/Reset register
--   C_FAMILY                -- Target FPGA architecture
--   C_AR0_BASEADDR          -- User address range 0 base address
--   C_AR0_HIGHADDR          -- User address range 0 high address
--   C_AR1_BASEADDR          -- User address range 1 base address
--   C_AR1_HIGHADDR          -- User address range 1 high address
--   C_AR2_BASEADDR          -- User address range 2 base address
--   C_AR2_HIGHADDR          -- User address range 2 high address
--   C_AR3_BASEADDR          -- User address range 3 base address
--   C_AR3_HIGHADDR          -- User address range 3 high address
--   C_AR4_BASEADDR          -- User address range 4 base address
--   C_AR4_HIGHADDR          -- User address range 4 high address
--   C_AR5_BASEADDR          -- User address range 5 base address
--   C_AR5_HIGHADDR          -- User address range 5 high address
--   C_AR6_BASEADDR          -- User address range 6 base address
--   C_AR6_HIGHADDR          -- User address range 6 high address
--   C_AR7_BASEADDR          -- User address range 7 base address
--   C_AR7_HIGHADDR          -- User address range 7 high address
--
-- Definition of Ports:
--   OPB_Clk                 -- OPB Clock
--   OPB_Rst                 -- OPB Reset
--   Sl_DBus                 -- Slave data bus
--   Sl_errAck               -- Slave error acknowledge
--   Sl_retry                -- Slave retry
--   Sl_toutSup              -- Slave timeout suppress
--   Sl_xferAck              -- Slave transfer acknowledge
--   OPB_ABus                -- OPB address bus
--   OPB_BE                  -- OPB byte enable
--   OPB_DBus                -- OPB data bus
```

```
--   OPB_RNW                 -- OPB read/not write
--   OPB_select              -- OPB select
--   OPB_seqAddr             -- OPB sequential address
--   IP2INTC_Irpt            -- Interrupt output to processor
------------------------------------------------------------------------------

entity PSO_NN_Calc1 is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ----------------------
    -- Bus protocol parameters, do not add to or delete
    C_BASEADDR                : std_logic_vector     := X"00000000";
    C_HIGHADDR                : std_logic_vector     := X"0000FFFF";
    C_OPB_AWIDTH              : integer              := 32;
    C_OPB_DWIDTH              : integer              := 32;
    C_USER_ID_CODE            : integer              := 3;
    C_FAMILY                  : string               := "virtex2p";
    C_AR0_BASEADDR            : std_logic_vector     := X"10000000";
    C_AR0_HIGHADDR            : std_logic_vector     := X"10FFFFFF";
    C_AR1_BASEADDR            : std_logic_vector     := X"11000000";
    C_AR1_HIGHADDR            : std_logic_vector     := X"11FFFFFF";
    C_AR2_BASEADDR            : std_logic_vector     := X"12000000";
    C_AR2_HIGHADDR            : std_logic_vector     := X"12FFFFFF";
    C_AR3_BASEADDR            : std_logic_vector     := X"13000000";
    C_AR3_HIGHADDR            : std_logic_vector     := X"13FFFFFF";
    C_AR4_BASEADDR            : std_logic_vector     := X"14000000";
    C_AR4_HIGHADDR            : std_logic_vector     := X"14FFFFFF";
    C_AR5_BASEADDR            : std_logic_vector     := X"15000000";
    C_AR5_HIGHADDR            : std_logic_vector     := X"15FFFFFF";
    C_AR6_BASEADDR            : std_logic_vector     := X"16000000";
    C_AR6_HIGHADDR            : std_logic_vector     := X"16FFFFFF";
    C_AR7_BASEADDR            : std_logic_vector     := X"17000000";
    C_AR7_HIGHADDR            : std_logic_vector     := X"17FFFFFF"
    -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    PSO_Running               : out std_logic;
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ----------------------
```

```
    -- Bus protocol ports, do not add to or delete
    OPB_Clk                     : in  std_logic;
    OPB_Rst                     : in  std_logic;
    Sl_DBus                     : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sl_errAck                   : out std_logic;
    Sl_retry                    : out std_logic;
    Sl_toutSup                  : out std_logic;
    Sl_xferAck                  : out std_logic;
    OPB_ABus                    : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE                      : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_DBus                    : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW                     : in  std_logic;
    OPB_select                  : in  std_logic;
    OPB_seqAddr                 : in  std_logic;
    IP2INTC_Irpt                : out std_logic
    -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );

  attribute SIGIS : string;
  attribute SIGIS of OPB_Clk       : signal is "Clk";
  attribute SIGIS of OPB_Rst       : signal is "Rst";
  attribute SIGIS of IP2INTC_Irpt  : signal is "INTR_LEVEL_HIGH";

end entity PSO_NN_Calc1;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of PSO_NN_Calc1 is

  ---------------------------------------------
  -- Constant: array of address range identifiers
  ---------------------------------------------
  constant ARD_ID_ARRAY               : INTEGER_ARRAY_TYPE   :=
    (
      0  => USER_00,                  -- user logic S/W register address space
      1  => IPIF_RST,                 -- include IPIF S/W Reset/MIR service
      2  => IPIF_INTR,                -- include IPIF Interrupt service
      3  => USER_01,                  -- user address range 0 address space
      4  => USER_02,                  -- user address range 1 address space
      5  => USER_03,                  -- user address range 2 address space
      6  => USER_04,                  -- user address range 3 address space
      7  => USER_05,                  -- user address range 4 address space
      8  => USER_06,                  -- user address range 5 address space
      9  => USER_07,                  -- user address range 6 address space
```

```
    10 => USER_08                    -- user address range 7 address space
   );

  ------------------------------------------
  -- Constant: array of address pairs for each address range
  ------------------------------------------
  constant ZERO_ADDR_PAD              : std_logic_vector(0 to 64-C_OPB_AWIDTH-
1) := (others => '0');

  constant USER_BASEADDR              : std_logic_vector     := C_BASEADDR or
X"00000000";
  constant USER_HIGHADDR              : std_logic_vector     := C_BASEADDR or
X"000000FF";

  constant RST_BASEADDR               : std_logic_vector     := C_BASEADDR or
X"00000100";
  constant RST_HIGHADDR               : std_logic_vector     := C_BASEADDR or
X"000001FF";

  constant INTR_BASEADDR              : std_logic_vector     := C_BASEADDR or
X"00000200";
  constant INTR_HIGHADDR              : std_logic_vector     := C_BASEADDR or
X"000002FF";

  constant ARD_ADDR_RANGE_ARRAY       : SLV64_ARRAY_TYPE     :=
    (
      ZERO_ADDR_PAD & USER_BASEADDR,          -- user logic base address
      ZERO_ADDR_PAD & USER_HIGHADDR,          -- user logic high address
      ZERO_ADDR_PAD & RST_BASEADDR,           -- MIR/Reset register base
address
      ZERO_ADDR_PAD & RST_HIGHADDR,           -- MIR/Reset register high
address
      ZERO_ADDR_PAD & INTR_BASEADDR,          -- interrupt register base
address
      ZERO_ADDR_PAD & INTR_HIGHADDR,          -- interrupt register high
address
      ZERO_ADDR_PAD & C_AR0_BASEADDR,         -- user address range 0 base
address
      ZERO_ADDR_PAD & C_AR0_HIGHADDR,         -- user address range 0 high
address
      ZERO_ADDR_PAD & C_AR1_BASEADDR,         -- user address range 1 base
address
      ZERO_ADDR_PAD & C_AR1_HIGHADDR,         -- user address range 1 high
address
      ZERO_ADDR_PAD & C_AR2_BASEADDR,         -- user address range 2 base
address
```

```
    ZERO_ADDR_PAD & C_AR2_HIGHADDR,          -- user address range 2 high
address
    ZERO_ADDR_PAD & C_AR3_BASEADDR,          -- user address range 3 base
address
    ZERO_ADDR_PAD & C_AR3_HIGHADDR,          -- user address range 3 high
address
    ZERO_ADDR_PAD & C_AR4_BASEADDR,          -- user address range 4 base
address
    ZERO_ADDR_PAD & C_AR4_HIGHADDR,          -- user address range 4 high
address
    ZERO_ADDR_PAD & C_AR5_BASEADDR,          -- user address range 5 base
address
    ZERO_ADDR_PAD & C_AR5_HIGHADDR,          -- user address range 5 high
address
    ZERO_ADDR_PAD & C_AR6_BASEADDR,          -- user address range 6 base
address
    ZERO_ADDR_PAD & C_AR6_HIGHADDR,          -- user address range 6 high
address
    ZERO_ADDR_PAD & C_AR7_BASEADDR,          -- user address range 7 base
address
    ZERO_ADDR_PAD & C_AR7_HIGHADDR           -- user address range 7 high
address
   );

 ----------------------------------------
 -- Constant: array of data widths for each target address range
 ----------------------------------------
 constant USER_DWIDTH                : integer           := 32;

 constant USER_AR0_DWIDTH            : integer           := 32;

 constant USER_AR1_DWIDTH            : integer           := 32;

 constant USER_AR2_DWIDTH            : integer           := 32;

 constant USER_AR3_DWIDTH            : integer           := 32;

 constant USER_AR4_DWIDTH            : integer           := 32;

 constant USER_AR5_DWIDTH            : integer           := 32;

 constant USER_AR6_DWIDTH            : integer           := 32;

 constant USER_AR7_DWIDTH            : integer           := 32;

 constant USER_MAX_AR_DWIDTH         : integer           := 32;
```

```
 constant ARD_DWIDTH_ARRAY              : INTEGER_ARRAY_TYPE   :=
  (
    0  => USER_DWIDTH,                  -- user logic data width
    1  => C_OPB_DWIDTH,                  -- MIR/Reset register data width
    2  => C_OPB_DWIDTH,                  -- interrupt register data width
    3  => USER_AR0_DWIDTH,                -- user address range 0 data width
    4  => USER_AR1_DWIDTH,                -- user address range 1 data width
    5  => USER_AR2_DWIDTH,                -- user address range 2 data width
    6  => USER_AR3_DWIDTH,                -- user address range 3 data width
    7  => USER_AR4_DWIDTH,                -- user address range 4 data width
    8  => USER_AR5_DWIDTH,                -- user address range 5 data width
    9  => USER_AR6_DWIDTH,                -- user address range 6 data width
    10 => USER_AR7_DWIDTH                  -- user address range 7 data width
  );

 ------------------------------------------
 -- Constant: array of desired number of chip enables for each address range
 ------------------------------------------
 constant USER_NUM_CE                : integer           := 8;

 constant USER_NUM_ADDR_RNG             : integer           := 8;

 constant ARD_NUM_CE_ARRAY              : INTEGER_ARRAY_TYPE   :=
  (
    0  => pad_power2(USER_NUM_CE),         -- user logic number of CEs
    1  => 1,                            -- MIR/Reset register - 1 CE
    2  => 16,                            -- interrupt register - 16 CEs
    3  => 1,                            -- user address range 0 - 1 CE
    4  => 1,                            -- user address range 1 - 1 CE
    5  => 1,                            -- user address range 2 - 1 CE
    6  => 1,                            -- user address range 3 - 1 CE
    7  => 1,                            -- user address range 4 - 1 CE
    8  => 1,                            -- user address range 5 - 1 CE
    9  => 1,                            -- user address range 6 - 1 CE
    10 => 1                              -- user address range 7 - 1 CE
  );

 ------------------------------------------
 -- Constant: array of unique properties for each address range
 ------------------------------------------
 constant USER_INCLUDE_DEV_ISC          : boolean           := false;

 constant USER_INCLUDE_DEV_PENCODER     : boolean           := false;

 constant ARD_DEPENDENT_PROPS_ARRAY     :
DEPENDENT_PROPS_ARRAY_TYPE :=
```

```
   (
    0  => (others => 0),                -- user logic slave space dependent properties (none
defined)
    1  => (others => 0),                -- IPIF reset/mir dependent properties (none defined)
    2  => (                             -- IPIF interrupt dependent properties
     EXCLUDE_DEV_ISC      => 1-boolean'pos(USER_INCLUDE_DEV_ISC),
     INCLUDE_DEV_PENCODER =>
boolean'pos(USER_INCLUDE_DEV_PENCODER),
     others => 0),
    3  => (others => 0),                -- user address range 0
    4  => (others => 0),                -- user address range 1
    5  => (others => 0),                -- user address range 2
    6  => (others => 0),                -- user address range 3
    7  => (others => 0),                -- user address range 4
    8  => (others => 0),                -- user address range 5
    9  => (others => 0),                -- user address range 6
    10 => (others => 0)                  -- user address range 7
  );

 ------------------------------------------
 -- Constant: pipeline mode
 -- 1 = include OPB-In pipeline registers
 -- 2 = include IP pipeline registers
 -- 3 = include OPB-In and IP pipeline registers
 -- 4 = include OPB-Out pipeline registers
 -- 5 = include OPB-In and OPB-Out pipeline registers
 -- 6 = include IP and OPB-Out pipeline registers
 -- 7 = include OPB-In, IP, and OPB-Out pipeline registers
 -- Note:
 -- only mode 4, 5, 7 are supported for this release
 ------------------------------------------
 constant PIPELINE_MODEL               : integer            := 5;

 ------------------------------------------
 -- Constant: user core ID code
 ------------------------------------------
 constant DEV_BLK_ID                   : integer            := C_USER_ID_CODE;

 ------------------------------------------
 -- Constant: enable MIR/Reset register
 ------------------------------------------
 constant DEV_MIR_ENABLE               : integer            := 1;

 ------------------------------------------
 -- Constant: array of IP interrupt mode
 -- 1 = Active-high interrupt condition
```

```
 -- 2 = Active-low interrupt condition
 -- 3 = Active-high pulse interrupt event
 -- 4 = Active-low pulse interrupt event
 -- 5 = Positive-edge interrupt event
 -- 6 = Negative-edge interrupt event
 ------------------------------------------
 constant USER_IP_INTR_NUM              : integer            := 1;

 constant IP_INTR_MODE_ARRAY            : INTEGER_ARRAY_TYPE   :=
  (
    0  => 1
  );

 ------------------------------------------
 -- Constant: enable device burst
 ------------------------------------------
 constant DEV_BURST_ENABLE              : integer            := 0;

 ------------------------------------------
 -- Constant: include address counter for burst transfers
 ------------------------------------------
 constant INCLUDE_ADDR_CNTR             : integer            := 0;

 ------------------------------------------
 -- Constant: include write buffer that decouples OPB and IPIC write transactions
 ------------------------------------------
 constant INCLUDE_WR_BUF                : integer            := 0;

 ------------------------------------------
 -- Constant: index for CS/CE
 ------------------------------------------
  constant USER00_CS_INDEX              : integer            :=
get_id_index(ARD_ID_ARRAY, USER_00);

  constant USER00_CE_INDEX              : integer            :=
calc_start_ce_index(ARD_NUM_CE_ARRAY, USER00_CS_INDEX);

  constant USER01_CS_INDEX              : integer            :=
get_id_index(ARD_ID_ARRAY, USER_01);

  constant USER01_CE_INDEX              : integer            :=
calc_start_ce_index(ARD_NUM_CE_ARRAY, USER01_CS_INDEX);

  constant USER02_CS_INDEX              : integer            :=
get_id_index(ARD_ID_ARRAY, USER_02);
```

```
  constant USER02_CE_INDEX        : integer          :=
calc_start_ce_index(ARD_NUM_CE_ARRAY, USER02_CS_INDEX);

  constant USER03_CS_INDEX        : integer          :=
get_id_index(ARD_ID_ARRAY, USER_03);

  constant USER03_CE_INDEX        : integer          :=
calc_start_ce_index(ARD_NUM_CE_ARRAY, USER03_CS_INDEX);

  constant USER04_CS_INDEX        : integer          :=
get_id_index(ARD_ID_ARRAY, USER_04);

  constant USER04_CE_INDEX        : integer          :=
calc_start_ce_index(ARD_NUM_CE_ARRAY, USER04_CS_INDEX);

  constant USER05_CS_INDEX        : integer          :=
get_id_index(ARD_ID_ARRAY, USER_05);

  constant USER05_CE_INDEX        : integer          :=
calc_start_ce_index(ARD_NUM_CE_ARRAY, USER05_CS_INDEX);

  constant USER06_CS_INDEX        : integer          :=
get_id_index(ARD_ID_ARRAY, USER_06);

  constant USER06_CE_INDEX        : integer          :=
calc_start_ce_index(ARD_NUM_CE_ARRAY, USER06_CS_INDEX);

  constant USER07_CS_INDEX        : integer          :=
get_id_index(ARD_ID_ARRAY, USER_07);

  constant USER07_CE_INDEX        : integer          :=
calc_start_ce_index(ARD_NUM_CE_ARRAY, USER07_CS_INDEX);

  constant USER08_CS_INDEX        : integer          :=
get_id_index(ARD_ID_ARRAY, USER_08);

  constant USER08_CE_INDEX        : integer          :=
calc_start_ce_index(ARD_NUM_CE_ARRAY, USER08_CS_INDEX);

  ------------------------------------------
  -- IP Interconnect (IPIC) signal declarations -- do not delete
  -- prefix 'i' stands for IPIF while prefix 'u' stands for user logic
  -- typically user logic will be hooked up to IPIF directly via i<sig>
  -- unless signal slicing and muxing are needed via u<sig>
  ------------------------------------------
```

```
  signal iBus2IP_RdCE                : std_logic_vector(0 to
calc_num_ce(ARD_NUM_CE_ARRAY)-1);
  signal iBus2IP_WrCE                : std_logic_vector(0 to
calc_num_ce(ARD_NUM_CE_ARRAY)-1);
  signal iBus2IP_Data                : std_logic_vector(0 to C_OPB_DWIDTH-1);
  signal iBus2IP_Addr                : std_logic_vector(0 to C_OPB_AWIDTH-1);
  signal iBus2IP_BE                  : std_logic_vector(0 to C_OPB_DWIDTH/8-1);
  signal iBus2IP_RNW                 : std_logic;
  signal iIP2Bus_Data                : std_logic_vector(0 to C_OPB_DWIDTH-1)   :=
(others => '0');
  signal iIP2Bus_Ack                 : std_logic   := '0';
  signal iIP2Bus_Error               : std_logic   := '0';
  signal iIP2Bus_Retry               : std_logic   := '0';
  signal iIP2Bus_ToutSup             : std_logic   := '0';
  signal ZERO_IP2Bus_PostedWrInh     : std_logic_vector(0 to
ARD_ID_ARRAY'length-1)   := (others => '0'); -- work around for XST not taking
(others => '0') in port mapping
  signal ZERO_IP2RFIFO_Data          : std_logic_vector(0 to
ARD_DWIDTH_ARRAY(get_id_index_iboe(ARD_ID_ARRAY,
IPIF_RDFIFO_DATA))-1)   := (others => '0'); -- work around for XST not taking (others
=> '0') in port mapping
  signal ZERO_WFIFO2IP_Data          : std_logic_vector(0 to
ARD_DWIDTH_ARRAY(get_id_index_iboe(ARD_ID_ARRAY,
IPIF_WRFIFO_DATA))-1)   := (others => '0'); -- work around for XST not taking (others
=> '0') in port mapping
  signal iIP2Bus_IntrEvent           : std_logic_vector(0 to
IP_INTR_MODE_ARRAY'length-1)   := (others => '0');
  signal iBus2IP_Clk                 : std_logic;
  signal iBus2IP_Reset               : std_logic;
  signal iBus2IP_CS                  : std_logic_vector(0 to ARD_ID_ARRAY'length-1);
  signal uBus2IP_Data                : std_logic_vector(0 to USER_DWIDTH-1);
  signal uBus2IP_BE                  : std_logic_vector(0 to USER_DWIDTH/8-1);
  signal uBus2IP_RdCE                : std_logic_vector(0 to USER_NUM_CE-1);
  signal uBus2IP_WrCE                : std_logic_vector(0 to USER_NUM_CE-1);
  signal uIP2Bus_Data                : std_logic_vector(0 to USER_DWIDTH-1);
  signal uBus2IP_ArData              : std_logic_vector(0 to USER_MAX_AR_DWIDTH-
1);
  signal uBus2IP_ArBE                : std_logic_vector(0 to
USER_MAX_AR_DWIDTH/8-1);
  signal uBus2IP_ArCS                : std_logic_vector(0 to USER_NUM_ADDR_RNG-
1);
  signal uIP2Bus_ArData              : std_logic_vector(0 to USER_MAX_AR_DWIDTH-
1);

begin
```

```
  ---------------------------------------
  -- instantiate the OPB IPIF
  ---------------------------------------
  OPB_IPIF_I : entity opb_ipif_v3_01_c.opb_ipif
    generic map
    (
      C_ARD_ID_ARRAY              => ARD_ID_ARRAY,
      C_ARD_ADDR_RANGE_ARRAY      => ARD_ADDR_RANGE_ARRAY,
      C_ARD_DWIDTH_ARRAY          => ARD_DWIDTH_ARRAY,
      C_ARD_NUM_CE_ARRAY          => ARD_NUM_CE_ARRAY,
      C_ARD_DEPENDENT_PROPS_ARRAY   =>
ARD_DEPENDENT_PROPS_ARRAY,
      C_PIPELINE_MODEL            => PIPELINE_MODEL,
      C_DEV_BLK_ID                => DEV_BLK_ID,
      C_DEV_MIR_ENABLE            => DEV_MIR_ENABLE,
      C_OPB_AWIDTH                => C_OPB_AWIDTH,
      C_OPB_DWIDTH                => C_OPB_DWIDTH,
      C_FAMILY                    => C_FAMILY,
      C_IP_INTR_MODE_ARRAY        => IP_INTR_MODE_ARRAY,
      C_DEV_BURST_ENABLE          => DEV_BURST_ENABLE,
      C_INCLUDE_ADDR_CNTR         => INCLUDE_ADDR_CNTR,
      C_INCLUDE_WR_BUF            => INCLUDE_WR_BUF
    )
    port map
    (
      OPB_select                  => OPB_select,
      OPB_DBus                    => OPB_DBus,
      OPB_ABus                    => OPB_ABus,
      OPB_BE                      => OPB_BE,
      OPB_RNW                     => OPB_RNW,
      OPB_seqAddr                 => OPB_seqAddr,
      Sln_DBus                    => Sl_DBus,
      Sln_xferAck                 => Sl_xferAck,
      Sln_errAck                  => Sl_errAck,
      Sln_retry                   => Sl_retry,
      Sln_toutSup                 => Sl_toutSup,
      Bus2IP_CS                   => iBus2IP_CS,
      Bus2IP_CE                   => open,
      Bus2IP_RdCE                 => iBus2IP_RdCE,
      Bus2IP_WrCE                 => iBus2IP_WrCE,
      Bus2IP_Data                 => iBus2IP_Data,
      Bus2IP_Addr                 => iBus2IP_Addr,
      Bus2IP_AddrValid            => open,
      Bus2IP_BE                   => iBus2IP_BE,
      Bus2IP_RNW                  => iBus2IP_RNW,
      Bus2IP_Burst                => open,
```

```
    IP2Bus_Data                   => iIP2Bus_Data,
    IP2Bus_Ack                    => iIP2Bus_Ack,
    IP2Bus_AddrAck                => '0',
    IP2Bus_Error                  => iIP2Bus_Error,
    IP2Bus_Retry                  => iIP2Bus_Retry,
    IP2Bus_ToutSup                => iIP2Bus_ToutSup,
    IP2Bus_PostedWrInh            => ZERO_IP2Bus_PostedWrInh,
    IP2RFIFO_Data                 => ZERO_IP2RFIFO_Data,
    IP2RFIFO_WrMark               => '0',
    IP2RFIFO_WrRelease            => '0',
    IP2RFIFO_WrReq                => '0',
    IP2RFIFO_WrRestore            => '0',
    RFIFO2IP_AlmostFull           => open,
    RFIFO2IP_Full                 => open,
    RFIFO2IP_Vacancy              => open,
    RFIFO2IP_WrAck                => open,
    IP2WFIFO_RdMark               => '0',
    IP2WFIFO_RdRelease            => '0',
    IP2WFIFO_RdReq                => '0',
    IP2WFIFO_RdRestore            => '0',
    WFIFO2IP_AlmostEmpty          => open,
    WFIFO2IP_Data                 => ZERO_WFIFO2IP_Data,
    WFIFO2IP_Empty                => open,
    WFIFO2IP_Occupancy            => open,
    WFIFO2IP_RdAck                => open,
    IP2Bus_IntrEvent              => iIP2Bus_IntrEvent,
    IP2INTC_Irpt                  => IP2INTC_Irpt,
    Freeze                        => '0',
    Bus2IP_Freeze                 => open,
    OPB_Clk                       => OPB_Clk,
    Bus2IP_Clk                    => iBus2IP_Clk,
    IP2Bus_Clk                    => '0',
    Reset                         => OPB_Rst,
    Bus2IP_Reset                  => iBus2IP_Reset
  );

------------------------------------------
-- instantiate the User Logic
------------------------------------------
USER_LOGIC_I : entity PSO_NN_Calc1_v2_00_a.user_logic
  generic map
  (
    -- MAP USER GENERICS BELOW THIS LINE ---------------
    --USER generics mapped here
    -- MAP USER GENERICS ABOVE THIS LINE ---------------
```

```
    C_AWIDTH                => C_OPB_AWIDTH,
    C_DWIDTH                => USER_DWIDTH,
    C_MAX_AR_DWIDTH             => USER_MAX_AR_DWIDTH,
    C_NUM_ADDR_RNG              => USER_NUM_ADDR_RNG,
    C_NUM_CE                => USER_NUM_CE,
    C_IP_INTR_NUM               => USER_IP_INTR_NUM
  )
  port map
  (
    -- MAP USER PORTS BELOW THIS LINE ------------------
    PSO_Running                 => PSO_Running,
    -- MAP USER PORTS ABOVE THIS LINE ------------------

    Bus2IP_Clk              => iBus2IP_Clk,
    Bus2IP_Reset            => iBus2IP_Reset,
    IP2Bus_IntrEvent            => iIP2Bus_IntrEvent,
    Bus2IP_Addr             => iBus2IP_Addr,
    Bus2IP_Data             => uBus2IP_Data,
    Bus2IP_BE               => uBus2IP_BE,
    Bus2IP_RNW                  => iBus2IP_RNW,
    Bus2IP_RdCE             => uBus2IP_RdCE,
    Bus2IP_WrCE             => uBus2IP_WrCE,
    IP2Bus_Data             => uIP2Bus_Data,
    IP2Bus_Ack              => iIP2Bus_Ack,
    IP2Bus_Retry            => iIP2Bus_Retry,
    IP2Bus_Error            => iIP2Bus_Error,
    IP2Bus_ToutSup              => iIP2Bus_ToutSup,
    Bus2IP_ArData           => uBus2IP_ArData,
    Bus2IP_ArBE             => uBus2IP_ArBE,
    Bus2IP_ArCS             => uBus2IP_ArCS,
    IP2Bus_ArData           => uIP2Bus_ArData
  );

  ----------------------------------------
  -- hooking up signal slicing
  ----------------------------------------
  uBus2IP_BE <= iBus2IP_BE(0 to USER_DWIDTH/8-1);
  uBus2IP_ArBE <= iBus2IP_BE(0 to USER_MAX_AR_DWIDTH/8-1);
  uBus2IP_Data <= iBus2IP_Data(0 to USER_DWIDTH-1);
  uBus2IP_ArData <= iBus2IP_Data(0 to USER_MAX_AR_DWIDTH-1);
  uBus2IP_ArCS <= iBus2IP_CS(USER01_CS_INDEX to
USER01_CS_INDEX+USER_NUM_ADDR_RNG-1);
  uBus2IP_RdCE <= iBus2IP_RdCE(USER00_CE_INDEX to
USER00_CE_INDEX+USER_NUM_CE-1);
  uBus2IP_WrCE <= iBus2IP_WrCE(USER00_CE_INDEX to
USER00_CE_INDEX+USER_NUM_CE-1);
```

```
  ---------------------------------------
  -- implement data mux
  ---------------------------------------
  DATA_MUX_PROC : process( iBus2IP_CS(USER00_CS_INDEX), uIP2Bus_Data,
uIP2Bus_ArData ) is
  begin

    iIP2Bus_Data <= (others => '0');
    if ( iBus2IP_CS(USER00_CS_INDEX) = '1' ) then
      iIP2Bus_Data(0 to USER_DWIDTH-1) <= uIP2Bus_Data;
    else
      iIP2Bus_Data(0 to USER_MAX_AR_DWIDTH-1) <= uIP2Bus_ArData;
    end if;

  end process DATA_MUX_PROC;

end IMP;
```

user_logic.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_AWIDTH                -- User logic address bus width
--   C_DWIDTH                -- User logic data bus width
--   C_MAX_AR_DWIDTH            -- User logic max data bus width of address ranges
--   C_NUM_ADDR_RNG             -- User logic number of address ranges to be decoded
--   C_NUM_CE                -- User logic chip enable bus width
--   C_IP_INTR_NUM              -- User logic number of interrupt event
--
-- Definition of Ports:
--   Bus2IP_Clk              -- Bus to IP clock
--   Bus2IP_Reset            -- Bus to IP reset
--   IP2Bus_IntrEvent           -- IP to Bus interrupt event
--   Bus2IP_Addr             -- Bus to IP address bus
--   Bus2IP_Data             -- Bus to IP data bus for user logic
--   Bus2IP_BE               -- Bus to IP byte enables for user logic
--   Bus2IP_RNW                 -- Bus to IP read/not write
```

```
--   Bus2IP_RdCE              -- Bus to IP read chip enable for user logic
--   Bus2IP_WrCE              -- Bus to IP write chip enable for user logic
--   IP2Bus_Data              -- IP to Bus data bus for user logic
--   IP2Bus_Ack               -- IP to Bus acknowledgement
--   IP2Bus_Retry             -- IP to Bus retry response
--   IP2Bus_Error             -- IP to Bus error response
--   IP2Bus_ToutSup           -- IP to Bus timeout suppress
--   Bus2IP_ArData            -- Bus to IP data bus for address ranges
--   Bus2IP_ArBE              -- Bus to IP byte enables for address ranges
--   Bus2IP_ArCS              -- Bus to IP chip select for address ranges
--   IP2Bus_ArData            -- IP to Bus data bus for address ranges
------------------------------------------------------------------------------

entity user_logic is
  generic
  ( C_AWIDTH                 : integer           := 32;
    C_DWIDTH                 : integer           := 32;
    C_MAX_AR_DWIDTH          : integer           := 32;
    C_NUM_ADDR_RNG           : integer           := 8;
    C_NUM_CE                 : integer           := 8;
    C_IP_INTR_NUM            : integer           := 1);
  port
  ( Bus2IP_Clk               : in  std_logic;
    Bus2IP_Reset             : in  std_logic;
    IP2Bus_IntrEvent         : out std_logic_vector(0 to C_IP_INTR_NUM-1);
    Bus2IP_Addr              : in  std_logic_vector(0 to C_AWIDTH-1);
    Bus2IP_Data              : in  std_logic_vector(0 to C_DWIDTH-1);
    Bus2IP_BE                : in  std_logic_vector(0 to C_DWIDTH/8-1);
    Bus2IP_RNW               : in  std_logic;
    Bus2IP_RdCE              : in  std_logic_vector(0 to C_NUM_CE-1);
    Bus2IP_WrCE              : in  std_logic_vector(0 to C_NUM_CE-1);
    IP2Bus_Data              : out std_logic_vector(0 to C_DWIDTH-1);
    IP2Bus_Ack               : out std_logic;
    IP2Bus_Retry             : out std_logic;
    IP2Bus_Error             : out std_logic;
    IP2Bus_ToutSup           : out std_logic;
    Bus2IP_ArData            : in  std_logic_vector(0 to C_MAX_AR_DWIDTH-1);
    Bus2IP_ArBE              : in  std_logic_vector(0 to C_MAX_AR_DWIDTH/8-1);
    Bus2IP_ArCS              : in  std_logic_vector(0 to C_NUM_ADDR_RNG-1);
    IP2Bus_ArData            : out std_logic_vector(0 to C_MAX_AR_DWIDTH-1);
          PSO_Running              : out std_logic);
end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------
```

```
architecture IMP of user_logic is

        component Sonar_NN is
                Port(   clk         : in  STD_LOGIC;
                        Start       : in  STD_LOGIC;
                        Reset       : in  STD_LOGIC;
                        Done                    : out STD_LOGIC;
                        OutputWrEn              : out STD_LOGIC;
                        OutNodeNum              : out
STD_LOGIC_VECTOR(11 downto 0);
                        OutputNode    : out STD_LOGIC_VECTOR(15
downto 0);
                        Input00       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input01       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input02       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input03       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input04       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input05       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input06       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input07       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input08       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input09       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input10       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input11       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input12       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input13       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input14       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input15       : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input16       : in  STD_LOGIC_VECTOR(15
downto 0);
```

```
                                    Input17      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input18      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input19      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input20      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input21      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input22      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input23      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input24      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input25      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input26      : in  STD_LOGIC_VECTOR(15
downto 0));
        end component;


        component PSO_Update is
    Port(clk                        : in  std_logic;
        Start                               : in  std_logic;
        Reset                               : in  std_logic;
                        Done                                : out std_logic;
                        Update_Wr_En                        : out std_logic;
                        input_index                 : out std_logic_vector(3 downto 0);
                        output_index                : out std_logic_vector(3 downto 0);
                        constant_global     : in  std_logic_vector(15 downto 0);
                        constant_local      : in  std_logic_vector(15 downto 0);
                        current_position1   : in  std_logic_vector(15 downto 0);
                        current_position2   : in  std_logic_vector(15 downto 0);
                        current_velocity1   : in  std_logic_vector(15 downto 0);
                        current_velocity2   : in  std_logic_vector(15 downto 0);
                        local_best1                 : in  std_logic_vector(15 downto 0);
                        local_best2                 : in  std_logic_vector(15 downto 0);
                        global_best1                : in  std_logic_vector(15 downto 0);
                        global_best2                : in  std_logic_vector(15 downto 0);
                        max_velocity1               : in  std_logic_vector(15 downto 0);
                        max_velocity2               : in  std_logic_vector(15 downto 0);
                        max_position1               : in  std_logic_vector(15 downto 0);
                        max_position2               : in  std_logic_vector(15 downto 0);
                        min_position1               : in  std_logic_vector(15 downto 0);
```

```
                        min_position2               : in  std_logic_vector(15 downto 0);
                        next_position1              : out std_logic_vector(15 downto 0);
                        next_position2              : out std_logic_vector(15 downto 0);
                        next_velocity1              : out std_logic_vector(15 downto 0);
                        next_velocity2              : out std_logic_vector(15 downto 0));

    end component;

---------------------------------------
-- Signals for user logic slave model s/w accessible register example
---------------------------------------
signal status_reg0              : std_logic_vector(0 to C_DWIDTH-1);
signal status_reg1              : std_logic_vector(0 to C_DWIDTH-1);
signal num_iterations            : std_logic_vector(0 to C_DWIDTH-1);
signal num_agents               : std_logic_vector(0 to C_DWIDTH-1);
signal const_CG                 : std_logic_vector(0 to C_DWIDTH-1);
signal const_CL                 : std_logic_vector(0 to C_DWIDTH-1);
signal slv_reg_write_select         : std_logic_vector(0 to 7);
signal slv_reg_read_select          : std_logic_vector(0 to 7);
signal slv_ip2bus_data           : std_logic_vector(0 to C_DWIDTH-1);
signal slv_read_ack             : std_logic;
signal slv_write_ack             : std_logic;
---------------------------------------
-- Signals for user logic interrupt example
---------------------------------------
signal interrupt                : std_logic_vector(0 to C_IP_INTR_NUM-1);
---------------------------------------
-- RAM and register declarations
---------------------------------------
type REG_TYPE1 is array (0 to 31) of std_logic_vector(15 downto 0);
type REG_TYPE2 is array (0 to 63) of std_logic_vector(31 downto 0);
type RAM_TYPE1 is array (0 to 1023) of std_logic_vector(15 downto 0);
type RAM_TYPE2 is array (0 to 2047) of std_logic_vector(31 downto 0);
type RAM_TYPE3 is array (0 to 31) of std_logic_vector(31 downto 0);
type RAM_TYPE4 is array (0 to 15) of std_logic_vector(31 downto 0);
type RAM_TYPE5 is array (0 to 127) of std_logic_vector(31 downto 0);
signal Input_Reg            : REG_TYPE1;
signal Output_RAM1                  : RAM_TYPE1;
signal Output_RAM2                  : RAM_TYPE1;
signal Bitmap_Reg           : REG_TYPE2;
signal PSO_Global_Best      : REG_TYPE1;
signal Agent_Positions      : RAM_TYPE2;
signal Agent_Velocities     : RAM_TYPE2;
signal MaxMin_Position      : RAM_TYPE3;
signal Max_Velocities               : RAM_TYPE4;
signal PSO_Local_Best               : RAM_TYPE2;
```

```
  signal Global_Fitness                 : std_logic_vector(31 downto 0);
  signal Local_Fitness          : RAM_TYPE5;
  signal Current_Fitness        : std_logic_vector(31 downto 0);
  ----------------------------------------
  -- Signals for user logic address range example
  ----------------------------------------
  type DATA_OUT_TYPE is array (0 to C_NUM_ADDR_RNG-1) of std_logic_vector(0
to C_MAX_AR_DWIDTH-1);
  signal ar_data_out              : DATA_OUT_TYPE;
  signal ar_address               : std_logic_vector(0 to 10);
  signal ar_select                : std_logic_vector(0 to 7);
  signal ar_read_enable             : std_logic;
  signal ar_read_ack_dly1           : std_logic;
  signal ar_read_ack              : std_logic;
  signal ar_write_ack             : std_logic;
  signal InputWrEn                : std_logic;
  signal OutRAM1_En                 : std_logic;
  signal OutRAM2_En                 : std_logic;
  signal BitmapWrEn               : std_logic;
  signal GlobalRegWrEn                                          : std_logic;
  signal PSO_GBest_RAM_En           : std_logic;
  signal Agent_Pos_RAM_En           : std_logic;
  signal Agent_Vel_RAM_En           : std_logic;
  signal MaxMin_Pos_RAM_En          : std_logic;
  signal Max_Vel_RAM_En           : std_logic;
  signal Agent_Position_Data_In                 : std_logic_vector(31 downto 0);
  signal Agent_Position_WrAddr_In       : integer;
  signal Agent_Position_RdAddr_In       : integer;
  signal Agent_Velocity_Data_In                 : std_logic_vector(31 downto 0);
  signal Agent_Velocity_WrAddr_In       : integer;
  signal Agent_Velocity_RdAddr_In       : integer;
  signal Local_Addr_In                                          : integer;
  signal RAM1_Data_Out                                          : std_logic_vector(15
downto 0);
  signal RAM2_Data_Out                                          : std_logic_vector(15
downto 0);

  ----------------------------------------
  -- Signal for PSO logic
  ----------------------------------------
  signal NN_Start                                                   : std_logic;
  signal PSO_Start                                              : std_logic;
  signal Update_Start                                       : std_logic;
  signal Reset_i                                                    : std_logic;
  signal NN_Done                                                    : std_logic;
  signal PSO_Done                                                   : std_logic;
```

```
  signal Update_Done                                    : std_logic;
  signal Update_Wr_En                                   : std_logic;
  signal update_cnt_in              : std_logic_vector(3 downto 0);
  signal update_cnt_out             : std_logic_vector(3 downto 0);
  signal current_position_out1      : std_logic_vector(31 downto 0);
  signal current_position_out2      : std_logic_vector(31 downto 0);
  signal current_velocity_out1      : std_logic_vector(31 downto 0);
  signal current_velocity_out2      : std_logic_vector(31 downto 0);
  signal local_best_out             : std_logic_vector(31 downto 0);
  signal global_best_out            : std_logic_vector(31 downto 0);
  signal max_velocity_out           : std_logic_vector(31 downto 0);
  signal maxmin_position_out1       : std_logic_vector(31 downto 0);
  signal maxmin_position_out2       : std_logic_vector(31 downto 0);
  signal next_position_in           : std_logic_vector(31 downto 0);
  signal next_velocity_in           : std_logic_vector(31 downto 0);

  ------------------------------------------
  -- State Machine Signals
  ------------------------------------------
  type state_type is
(initial,load_inputs,start_calc_nn,calc_nn,update_best,update_global_local,update_only_l
ocal,update_agent,update_count,finished1,finished2);
  signal PSO_state                                      : state_type;
  signal PSO_next_state                                 : state_type;
  signal load_cnt                                       :
std_logic_vector(3 downto 0);
  signal load_cnt_dly                                   : std_logic_vector(3 downto
0);
  signal agent_cnt                                      : std_logic_vector(7
downto 0);
  signal iteration_cnt                                  : std_logic_vector(31
downto 0);
  signal LoadInputs                                     : std_logic;
  signal WriteGlobalBest                                : std_logic;
  signal WriteLocalBest                                 : std_logic;
  signal PSO_Running_i                                  : std_logic;
  signal SetAgentCnt0                                   : std_logic;
  signal SetIterationCnt0                               : std_logic;
  signal SetLoadCnt0                                    : std_logic;
  signal IncrAgentCnt                                   : std_logic;
  signal IncrLoadCnt                                    : std_logic;
  signal PSO_Start_NN                                   : std_logic;
  signal Reset_Current_Fitness                          : std_logic;
  signal Update_Rd_En               : std_logic;

  signal OutputWrEn_i                                   : std_logic;
```

```
 signal OutNodeNum_i                                    : std_logic_vector(11
downto 0);
 signal OutputNode_i                                : std_logic_vector(15
downto 0);



begin

        PSO_Running <= not PSO_Running_i;
        NN_Start <= PSO_Start_NN  when PSO_Running_i = '1' else status_reg0(31);
        PSO_Start <= status_reg0(30);
        Reset_i <= status_reg0(29) or Bus2IP_Reset;
  IP2Bus_IntrEvent(0) <= '0';



  slv_reg_write_select <= Bus2IP_WrCE(0 to 7);
  slv_reg_read_select  <= Bus2IP_RdCE(0 to 7);
  slv_write_ack        <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or
Bus2IP_WrCE(3) or Bus2IP_WrCE(4) or Bus2IP_WrCE(5) or Bus2IP_WrCE(6) or
Bus2IP_WrCE(7);
  slv_read_ack         <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2) or
Bus2IP_RdCE(3) or Bus2IP_RdCE(4) or Bus2IP_RdCE(5) or Bus2IP_RdCE(6) or
Bus2IP_RdCE(7);


  -- implement slave model register(s)
  SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
  begin
    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Reset = '1' then
        status_reg0 <= (others => '0');
        status_reg1 <= (others => '0');
                                  num_iterations <= x"0000000a";
        num_agents <= x"0000000a";
                    const_CG <= (others => '0');
        const_CL <= (others => '0');
      else
        case slv_reg_write_select is
          when "10000000" =>
            status_reg0 <= Bus2IP_Data;
          when "01000000" =>
            status_reg1 <= Bus2IP_Data;
                                      when "00100000" =>
```

```
            num_iterations <= Bus2IP_Data;
          when "00010000" =>
            num_agents <= Bus2IP_Data;
                        when "00001000" =>
            const_CG <= Bus2IP_Data;
          when "00000100" =>
            const_CL <= Bus2IP_Data;
          when others => null;
        end case;
      end if;
    end if;
  end process SLAVE_REG_WRITE_PROC;


  SLAVE_REG_READ_PROC : process( slv_reg_read_select, status_reg0, status_reg1,
num_iterations, num_agents, const_CL, const_CG, PSO_Done, NN_Done) is
  begin
    case slv_reg_read_select is
      when "10000000" => slv_ip2bus_data <= status_reg0;
      when "01000000" => slv_ip2bus_data <= status_reg1(0 to 29) & PSO_Done &
NN_Done;
                          when "00100000" => slv_ip2bus_data <= num_iterations;
      when "00010000" => slv_ip2bus_data <= num_agents;
      when "00001000" => slv_ip2bus_data <= const_CG;
      when "00000100" => slv_ip2bus_data <= const_CL;
      when "00000010" => slv_ip2bus_data <= Current_Fitness;
      when "00000001" => slv_ip2bus_data <= Global_Fitness;
      when others => slv_ip2bus_data <= (others => '0');
    end case;
  end process SLAVE_REG_READ_PROC;


  --generates the read acknowledge 1 clock after read enable is presented
  BRAM_RD_ACK_PROC : process( Bus2IP_Clk ) is
  begin
    if (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
      if(Bus2IP_Reset = '1') then
        ar_read_ack_dly1 <= '0';
      else
        ar_read_ack_dly1 <= ar_read_enable;
      end if;
    end if;
  end process BRAM_RD_ACK_PROC;


        ar_select      <= Bus2IP_ArCS;
```

```
  ar_read_enable <= ( Bus2IP_ArCS(0) or Bus2IP_ArCS(1) or Bus2IP_ArCS(2) or
Bus2IP_ArCS(3) or Bus2IP_ArCS(4) or Bus2IP_ArCS(5) or Bus2IP_ArCS(6) or
Bus2IP_ArCS(7) ) and Bus2IP_RNW;
  ar_read_ack    <= ar_read_ack_dly1;
  ar_write_ack   <= ( Bus2IP_ArCS(0) or Bus2IP_ArCS(1) or Bus2IP_ArCS(2) or
Bus2IP_ArCS(3) or Bus2IP_ArCS(4) or Bus2IP_ArCS(5) or Bus2IP_ArCS(6) or
Bus2IP_ArCS(7) ) and not(Bus2IP_RNW);
  ar_address     <= Bus2IP_Addr(C_AWIDTH-13 to C_AWIDTH-3);


        InputWrEn <= not(Bus2IP_RNW) and Bus2IP_ArCS(0);
        INPUTREG:process(Bus2IP_Clk,Reset_i)
        begin
                if(Reset_i = '1') then
                        for index in 0 to 31 loop
                                Input_Reg(index) <= (others => '0');
                        end loop;
                elsif(rising_edge(Bus2IP_Clk)) then
                        if(LoadInputs = '1') then
                                Input_Reg(0+2*conv_integer(load_cnt_dly))
<=current_position_out2(31 downto 16);
                                Input_Reg(1+2*conv_integer(load_cnt_dly))
<=current_position_out2(15 downto 0);
                        elsif(InputWrEn  = '1') then
                                Input_Reg(0+2*conv_integer(ar_address(7 to 10))) <=
Bus2IP_ArData(0 to 15);
                                Input_Reg(1+2*conv_integer(ar_address(7 to 10))) <=
Bus2IP_ArData(16 to 31);
                        end if;
                end if;
        end process;


        OutRAM1_En <=OutputWrEn_i and not(OutNodeNum_i(0));
        OutRAM2_En <=OutputWrEn_i and (OutNodeNum_i(0));
        OUTPUTRAM1:process(Bus2IP_Clk)
        begin
                if(rising_edge(Bus2IP_Clk)) then
                        if(OutRAM1_En='1') then
                                Output_RAM1(conv_integer(OutNodeNum_i(10 downto
1))) <= OutputNode_i;
                        end if;
                        RAM1_Data_Out <=
Output_RAM1(CONV_INTEGER(ar_address(1 to 10)));
                end if;
        end process;
```

```
	OUTPUTRAM2:process(Bus2IP_Clk)
	begin
		if(rising_edge(Bus2IP_Clk)) then
			if(OutRAM2_En='1') then
				Output_RAM2(conv_integer(OutNodeNum_i(10 downto
1))) <= OutputNode_i;
			end if;
			RAM2_Data_Out <=
Output_RAM2(CONV_INTEGER(ar_address(1 to 10)));
		end if;
	end process;


	BitmapWrEn <= not(Bus2IP_RNW) and Bus2IP_ArCS(2);
	BITMAPREG:process(Bus2IP_Clk,Bus2IP_Reset)
	begin
		if(Bus2IP_Reset = '1') then
			for index in 0 to 63 loop
				Bitmap_Reg(index) <= (others => '0');
			end loop;
		elsif(rising_edge(Bus2IP_Clk)) then
			if(BitmapWrEn  = '1') then
			  Bitmap_Reg(conv_integer(ar_address(5 to 10)))  <=
Bus2IP_ArData;
			end if;
		end if;
	end process;

	GlobalRegWrEn <= not(Bus2IP_RNW) and Bus2IP_ArCS(3);
	PSO_GLOBAL_REG:process(Bus2IP_Clk,Reset_i)
	begin
		if(Reset_i = '1') then
			for index in 0 to 31 loop
				PSO_Global_Best(index) <= (others => '0');
			end loop;
		elsif(rising_edge(Bus2IP_Clk)) then

			if(GlobalRegWrEn = '1') then
				PSO_Global_Best(0+2*conv_integer(ar_address(7 to 10)))
<= Bus2IP_ArData(0 to 15);
				PSO_Global_Best(1+2*conv_integer(ar_address(7 to 10)))
<= Bus2IP_ArData(16 to 31);
			elsif(WriteGlobalBest = '1') then
				for index in 0 to 31 loop
					PSO_Global_Best(index) <= Input_Reg(index);
				end loop;
```

```
                        end if;

--                      if(WriteGlobalBest  = '1') then
--                              for index in 0 to 31 loop
--                                      PSO_Global_Best(index) <= Input_Reg(index);
--                              end loop;
--                      end if;
                        global_best_out(31 downto 16) <=
PSO_Global_Best(0+2*conv_integer(update_cnt_in));
                        global_best_out(15 downto 0)  <=
PSO_Global_Best(1+2*conv_integer(update_cnt_in));
                end if;
        end process;




        Local_Addr_In <= 16*conv_integer(agent_cnt)+conv_integer(load_cnt) when
WriteLocalBest = '1' else 16*conv_integer(agent_cnt)+conv_integer(update_cnt_in);
        PSO_LOCAL_RAM:process(Bus2IP_Clk,Reset_i)
        begin
                if(rising_edge(Bus2IP_Clk)) then
                        if(WriteLocalBest  = '1') then
                                PSO_Local_Best(Local_Addr_In) <=
Input_Reg(2*conv_integer(load_cnt)) & Input_Reg(2*conv_integer(load_cnt)+1);
                        end if;
                local_best_out <= PSO_Local_Best(Local_Addr_In);
                end if;
        end process;


        Agent_Pos_RAM_En <= Update_Wr_En or (not(Bus2IP_RNW) and
Bus2IP_ArCS(4));
        Agent_Vel_RAM_En <= Update_Wr_En or (not(Bus2IP_RNW) and
Bus2IP_ArCS(5));
        MaxMin_Pos_RAM_En <= not(Bus2IP_RNW) and Bus2IP_ArCS(6);
        Max_Vel_RAM_En <= not(Bus2IP_RNW) and Bus2IP_ArCS(7);
        Agent_Position_Data_In <= next_position_in when Update_Wr_En = '1' else
Bus2IP_ArData;
        Agent_Velocity_Data_In <= next_velocity_in when Update_Wr_En = '1' else
Bus2IP_ArData;
        Agent_Position_WrAddr_In <=
16*conv_integer(agent_cnt)+conv_integer(update_cnt_out) when (Update_Wr_En = '1')
else conv_integer(ar_address);
```

```
	Agent_Velocity_WrAddr_In <=
16*conv_integer(agent_cnt)+conv_integer(update_cnt_out) when (Update_Wr_En = '1')
else conv_integer(ar_address);
	Agent_Position_RdAddr_In <=
16*conv_integer(agent_cnt)+conv_integer(update_cnt_in) when (Update_Rd_En = '1')
else 16*conv_integer(agent_cnt)+conv_integer(load_cnt) when LoadInputs = '1' else
conv_integer(ar_address);
	Agent_Velocity_RdAddr_In <=
16*conv_integer(agent_cnt)+conv_integer(update_cnt_in) when (Update_Rd_En = '1')
else conv_integer(ar_address);

  RAMS:process(Bus2IP_Clk)
	begin
		if(rising_edge(Bus2IP_Clk)) then

			if(Agent_Pos_RAM_En = '1') then
				Agent_Positions(Agent_Position_WrAddr_In) <=
Agent_Position_Data_In;
			end if;

			if(Agent_Vel_RAM_En = '1') then
				Agent_Velocities(Agent_Velocity_WrAddr_In) <=
Agent_Velocity_Data_In;
			end if;

			if(MaxMin_Pos_RAM_En = '1') then
				MaxMin_Position(conv_integer(ar_address(6 to 10))) <=
Bus2IP_ArData;
			end if;

			if(Max_Vel_RAM_En = '1') then
				Max_Velocities(conv_integer(ar_address(7 to 10))) <=
Bus2IP_ArData;
			end if;

			current_position_out1 <=
Agent_Positions(Agent_Position_WrAddr_In);
			current_position_out2 <=
Agent_Positions(Agent_Position_RdAddr_In);

			current_velocity_out1 <=
Agent_Velocities(Agent_Velocity_WrAddr_In);
			current_velocity_out2 <=
Agent_Velocities(Agent_Velocity_RdAddr_In);
```

```
                        ar_data_out(6) <= MaxMin_Position(conv_integer(ar_address(6 to
10)));
                        maxmin_position_out1 <=
MaxMin_Position(2*conv_integer(update_cnt_in));
                        maxmin_position_out2 <=
MaxMin_Position(1+2*conv_integer(update_cnt_in));

                        ar_data_out(7) <= Max_Velocities(conv_integer(ar_address(7 to
10)));
                        max_velocity_out <=
Max_Velocities(conv_integer(update_cnt_in));

                end if;
        end process;


        ar_data_out(0) <= Input_reg(0+2*CONV_INTEGER(ar_address(7 to 10))) &

        Input_reg(1+2*CONV_INTEGER(ar_address(7 to 10)));
        ar_data_out(1) <= RAM1_Data_Out & RAM2_Data_Out;
        ar_data_out(2) <= Bitmap_reg(CONV_INTEGER(ar_address(5 to 10)));
        ar_data_out(3) <= PSO_Global_Best(0+2*CONV_INTEGER(ar_address(7 to
10))) &

        PSO_Global_Best(1+2*CONV_INTEGER(ar_address(7 to 10)));
        ar_data_out(4) <= current_position_out2;
        ar_data_out(5) <= current_velocity_out2;


 IP2BUS_ARDATA_PROC : process( ar_data_out, ar_select ) is
 begin
  case ar_select is
    when "10000000" => IP2Bus_ArData <= ar_data_out(0);
    when "01000000" => IP2Bus_ArData <= ar_data_out(1);
    when "00100000" => IP2Bus_ArData <= ar_data_out(2);
    when "00010000" => IP2Bus_ArData <= ar_data_out(3);
    when "00001000" => IP2Bus_ArData <= ar_data_out(4);
    when "00000100" => IP2Bus_ArData <= ar_data_out(5);
    when "00000010" => IP2Bus_ArData <= ar_data_out(6);
    when "00000001" => IP2Bus_ArData <= ar_data_out(7);
    when others => IP2Bus_ArData <= (others => '0');
  end case;
 end process IP2BUS_ARDATA_PROC;


        StateMemory: process(Bus2IP_Clk,Reset_i)
```

```
	begin
		if(Reset_i='1') then
			PSO_state <= initial;
			agent_cnt <= (others => '0');
			iteration_cnt <= (others => '0');
			load_cnt <= (others => '0');
		elsif(rising_edge(Bus2IP_Clk)) then
			PSO_state <= PSO_next_state;

			if(SetAgentCnt0 = '1') then
				agent_cnt <= (others => '0');
			elsif(IncrAgentCnt = '1' and agent_cnt < num_agents-1) then
				agent_cnt <= agent_cnt + 1;
			elsif(IncrAgentCnt = '1' and agent_cnt = num_agents-1) then
				agent_cnt <= (others => '0');
			end if;

			if(SetIterationCnt0 = '1') then
				iteration_cnt <= (others => '0');
			elsif(IncrAgentCnt = '1' and agent_cnt = num_agents-1) then
				iteration_cnt <= iteration_cnt + 1;
			end if;

			if(SetLoadCnt0 = '1') then
				load_cnt <= (others => '0');
			elsif(IncrLoadCnt = '1') then
				load_cnt <= load_cnt + 1;
			end if;

			load_cnt_dly <= load_cnt;

		end if;
	end process;

	NextStateLogic:
process(PSO_state,PSO_Start,NN_Done,Update_Done,agent_cnt,load_cnt)
	begin
		case PSO_state is

			when initial =>
			  if(PSO_Start = '1') then
				PSO_next_state <= load_inputs;
			  else PSO_next_state <= initial;
			  end if;
			when load_inputs =>
			  if(load_cnt >= x"e") then
```

```
                    PSO_next_state <= start_calc_nn;
                else PSO_next_state <= load_inputs;
                end if;
            when start_calc_nn =>
                PSO_next_state <= calc_nn;
            when calc_nn =>
                if(NN_Done = '1') then
                    PSO_next_state <= update_best;
                else PSO_next_state <= calc_nn;
                end if;
            when update_best =>
              if(Current_Fitness < Global_Fitness) then
                    PSO_next_state <= update_global_local;
                elsif(Current_Fitness <
Local_Fitness(conv_integer(agent_cnt)) or iteration_cnt = x"00000000") then
                        PSO_next_state <= update_only_local;
                else PSO_next_state <= update_agent;
                end if;
            when update_global_local =>
              if(load_cnt >= x"d") then
                    PSO_next_state <= update_agent;
                else PSO_next_state <= update_global_local;
                end if;
            when update_only_local =>
              if(load_cnt >= x"d") then
                    PSO_next_state <= update_agent;
                else PSO_next_state <= update_only_local;
                end if;
            when update_agent =>
              if(Update_Done = '1') then
                    PSO_next_state <= update_count;
                else PSO_next_state <= update_agent;
                end if;
            when update_count =>
              if(agent_cnt >= num_agents-1 and iteration_cnt >=
num_iterations-1 and PSO_Start = '0') then
                    PSO_next_state <= finished1;
              elsif(agent_cnt >= num_agents-1 and iteration_cnt >=
num_iterations-1 and PSO_Start = '1') then
                    PSO_next_state <= finished2;
                else PSO_next_state <= load_inputs;
                end if;
            when finished1 =>
              if(PSO_Start = '1') then
                    PSO_next_state <= load_inputs;
                else PSO_next_state <= finished1;
```

```
                    end if;
                when finished2 =>
                  if(PSO_Start = '0') then
                        PSO_next_state <= finished1;
                    else PSO_next_state <= finished2;
                    end if;

        end case;
end process;

OutputLogic: process(PSO_state)
begin

        PSO_Done <= '0';
        PSO_Running_i <= '1';
        SetAgentCnt0 <= '0';
        SetIterationCnt0 <= '0';
        SetLoadCnt0 <= '0';
        IncrAgentCnt <= '0';
        IncrLoadCnt <= '0';
        Update_Rd_En <= '0';
        LoadInputs <= '0';
        WriteGlobalBest <= '0';
        WriteLocalBest <= '0';
        PSO_Start_NN <= '0';
        Update_Start <= '0';
        Reset_Current_Fitness <= '0';

        if(PSO_state = initial) then
                PSO_Running_i <= '0';
                SetAgentCnt0 <= '1';
                SetIterationCnt0 <= '1';
                SetLoadCnt0 <= '1';
        end if;

        if(PSO_state = load_inputs) then
                LoadInputs <= '1';
                IncrLoadCnt <= '1';
        end if;

        if(PSO_state = start_calc_nn) then
                PSO_Start_NN <= '1';
                Reset_Current_Fitness <= '1';
        end if;

        if(PSO_state = calc_nn) then
```

```
                PSO_Start_NN <= '1';
        end if;

        if(PSO_state = update_best) then
                SetLoadCnt0 <= '1';
        end if;

        if(PSO_state = update_global_local) then
                IncrLoadCnt <= '1';
                WriteGlobalBest <= '1';
                WriteLocalBest <= '1';
        end if;

        if(PSO_state = update_only_local) then
                IncrLoadCnt <= '1';
                WriteLocalBest <= '1';
        end if;

        if(PSO_state = update_agent) then
                        Update_Start <= '1';
                        Update_Rd_En <= '1';
        end if;

        if(PSO_state = update_count) then
                SetLoadCnt0 <= '1';
                IncrAgentCnt <= '1';
        end if;

        if(PSO_state = finished1) then
                PSO_Done <= '1';
                PSO_Running_i <= '0';
                SetAgentCnt0 <= '1';
                SetIterationCnt0 <= '1';
                SetLoadCnt0 <= '1';
        end if;

        if(PSO_state = finished2) then
                PSO_Done <= '1';
                PSO_Running_i <= '0';
                SetAgentCnt0 <= '1';
                SetIterationCnt0 <= '1';
                SetLoadCnt0 <= '1';
        end if;

end process;
```

```
	FITNESS:process(Bus2IP_Clk,Reset_i)
	begin
		if(Reset_i = '1') then
			Current_Fitness <= (others => '0');
			Global_Fitness <= x"ffffffff";
		elsif(rising_edge(Bus2IP_Clk)) then
			if(Reset_Current_Fitness = '1') then
				Current_Fitness <= (others => '0');
			elsif(Bitmap_reg(conv_integer(OutNodeNum_i(10 downto
5)))(conv_integer(31-OutNodeNum_i(4 downto 0))) = '1') then
				if(OutputWrEn_i = '1' and OutputNode_i(15) = '1') then
					Current_Fitness <= Current_Fitness - (x"ffff" &
OutputNode_i);
				elsif(OutputWrEn_i = '1' and OutputNode_i(15) = '0') then
					Current_Fitness <= Current_Fitness + (x"0000" &
OutputNode_i);
				end if;
			end if;

			if(slv_reg_write_select =  "00000001") then
				Global_Fitness <= Bus2IP_Data;
			elsif(WriteGlobalBest = '1') then
				Global_Fitness <= Current_Fitness;
			end if;

			if(WriteLocalBest = '1') then
				Local_Fitness(conv_integer(agent_cnt)) <=
Current_Fitness;
			end if;

		end if;
	end process;


  ---------------------------------------
  -- Code to drive IP to Bus signals
  ---------------------------------------
  IP2Bus_Data	<= slv_ip2bus_data;
  IP2Bus_Ack	<= slv_write_ack or slv_read_ack or ar_write_ack or ar_read_ack;
  IP2Bus_Error	<= '0';
  IP2Bus_Retry	<= '0';
  IP2Bus_ToutSup	<= '0';

  ------------------------------------------------------------
	-- Component Port Maps
	------------------------------------------------------------
```

```
U1 : Sonar_NN
  port map ( clk => Bus2IP_Clk,
             Start => NN_Start,
             Reset => Reset_i,
                                  Done => NN_Done,
                             OutputWrEn => OutputWrEn_i,
                                  OutNodeNum => OutNodeNum_i,
             OutputNode => OutputNode_i,
             Input00 => Input_reg(0),
             Input01 => Input_reg(1),
             Input02 => Input_reg(2),
             Input03 => Input_reg(3),
             Input04 => Input_reg(4),
             Input05 => Input_reg(5),
             Input06 => Input_reg(6),
             Input07 => Input_reg(7),
             Input08 => Input_reg(8),
             Input09 => Input_reg(9),
             Input10 => Input_reg(10),
             Input11 => Input_reg(11),
             Input12 => Input_reg(12),
             Input13 => Input_reg(13),
             Input14 => Input_reg(14),
             Input15 => Input_reg(15),
             Input16 => Input_reg(16),
             Input17 => Input_reg(17),
             Input18 => Input_reg(18),
             Input19 => Input_reg(19),
             Input20 => Input_reg(20),
             Input21 => Input_reg(21),
             Input22 => Input_reg(22),
             Input23 => Input_reg(23),
             Input24 => Input_reg(24),
             Input25 => Input_reg(25),
             Input26 => Input_reg(26));


       U2 : PSO_Update
  port map ( clk => Bus2IP_Clk,
                                    Start => Update_Start,
                                    Reset => Reset_i,
                                    Done => Update_Done,
                                    Update_Wr_En => Update_Wr_En,
                                    input_index => update_cnt_in,
                                    output_index => update_cnt_out,
                                    constant_global => const_CG(16 to 31),
```

```
                                    constant_local => const_CL(16 to 31),
                                    current_position1 => current_position_out2(31
downto 16),
                                    current_position2 => current_position_out2(15
downto 0),
                                    current_velocity1 => current_velocity_out2(31
downto 16),
                                    current_velocity2 => current_velocity_out2(15
downto 0),
                                    local_best1 => local_best_out(31 downto 16),
                                    local_best2 => local_best_out(15 downto 0),
                                    global_best1 => global_best_out(31 downto 16),
                                    global_best2 => global_best_out(15 downto 0),
                                    max_velocity1 => max_velocity_out(31 downto
16),
                                    max_velocity2 => max_velocity_out(15 downto
0),
                                    max_position1 => maxmin_position_out1(31
downto 16),
                                    max_position2 => maxmin_position_out2(31
downto 16),
                                    min_position1 => maxmin_position_out1(15
downto 0),
                                    min_position2 => maxmin_position_out2(15
downto 0),
                                    next_position1 => next_position_in(31 downto
16),
                                    next_position2 => next_position_in(15 downto 0),
                                    next_velocity1 => next_velocity_in(31 downto
16),
                                    next_velocity2 => next_velocity_in(15 downto
0));


end IMP;
```

PSO_Update.vhd

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

library UNISIM;
use UNISIM.VComponents.all;
```

```
entity PSO_Update is
   Port ( clk                          : in  std_logic;
          Start                        : in  std_logic;
          Reset                        : in  std_logic;
                      Done                          : out std_logic;
                      Update_Wr_En                  : out std_logic;
                      input_index                   : out std_logic_vector(3 downto 0);
                      output_index                  : out std_logic_vector(3 downto 0);
                      constant_global      : in  std_logic_vector(15 downto 0);
                      constant_local       : in  std_logic_vector(15 downto 0);
                      current_position1    : in  std_logic_vector(15 downto 0);
                      current_position2    : in  std_logic_vector(15 downto 0);
                      current_velocity1    : in  std_logic_vector(15 downto 0);
                      current_velocity2    : in  std_logic_vector(15 downto 0);
                      local_best1                   : in  std_logic_vector(15 downto 0);
                      local_best2                   : in  std_logic_vector(15 downto 0);
                      global_best1                  : in  std_logic_vector(15 downto 0);
                      global_best2                  : in  std_logic_vector(15 downto 0);
                      max_velocity1                 : in  std_logic_vector(15 downto 0);
                      max_velocity2                 : in  std_logic_vector(15 downto 0);
                      max_position1                 : in  std_logic_vector(15 downto 0);
                      max_position2                 : in  std_logic_vector(15 downto 0);
                      min_position1                 : in  std_logic_vector(15 downto 0);
                      min_position2                 : in  std_logic_vector(15 downto 0);
                      next_position1                : out std_logic_vector(15 downto 0);
                      next_position2                : out std_logic_vector(15 downto 0);
                      next_velocity1                : out std_logic_vector(15 downto 0);
                      next_velocity2                : out std_logic_vector(15 downto 0));

end PSO_Update;

architecture Behavioral of PSO_Update is

        signal next_position1_i                       : std_logic_vector(15 downto
0);
        signal next_position2_i                       : std_logic_vector(15 downto
0);
        signal next_velocity1_i                       : std_logic_vector(15 downto
0);
        signal next_velocity2_i                       : std_logic_vector(15 downto
0);
        signal diff_local1                            : std_logic_vector(15 downto
0);
        signal diff_local2                            : std_logic_vector(15 downto
0);
```

```
        signal diff_global1                         : std_logic_vector(15 downto
0);
        signal diff_global2                         : std_logic_vector(15 downto
0);
        signal diff_local1_sign_ext         : std_logic_vector(17 downto 0);
        signal diff_local2_sign_ext         : std_logic_vector(17 downto 0);
        signal diff_global1_sign_ext        : std_logic_vector(17 downto 0);
        signal diff_global2_sign_ext        : std_logic_vector(17 downto 0);
        signal constant_local_sign_ext      : std_logic_vector(17 downto 0);
        signal constant_global_sign_ext     : std_logic_vector(17 downto 0);
        signal local_term1                          : std_logic_vector(35 downto
0);
        signal local_term2                          : std_logic_vector(35 downto
0);
        signal global_term1                         : std_logic_vector(35 downto
0);
        signal global_term2                         : std_logic_vector(35 downto
0);
        signal counter                                      : std_logic_vector(3
downto 0);
        signal counter_reg1                         : std_logic_vector(3 downto
0);
        signal counter_reg2                         : std_logic_vector(3 downto
0);
        signal counter_reg3                         : std_logic_vector(3 downto
0);
        signal counter_reg4                         : std_logic_vector(3 downto
0);
        signal counter_reg5                         : std_logic_vector(3 downto
0);
        signal Start_Reg                                    : std_logic;

        signal next_position1_reg1          : std_logic_vector(15 downto 0);
        signal next_position1_reg3          : std_logic_vector(15 downto 0);
        signal next_position1_reg2          : std_logic_vector(15 downto 0);
        signal next_position2_reg1          : std_logic_vector(15 downto 0);
        signal next_position2_reg3          : std_logic_vector(15 downto 0);
        signal next_position2_reg2          : std_logic_vector(15 downto 0);

        signal max_position1_reg1       : std_logic_vector(15 downto 0);
        signal max_position2_reg1       : std_logic_vector(15 downto 0);
        signal min_position1_reg1       : std_logic_vector(15 downto 0);
        signal min_position2_reg1       : std_logic_vector(15 downto 0);

        signal max_velocity1_reg3           : std_logic_vector(15 downto 0);
        signal max_velocity1_reg2           : std_logic_vector(15 downto 0);
```

```
        signal max_velocity1_reg1                   : std_logic_vector(15 downto 0);

        signal max_velocity2_reg3                   : std_logic_vector(15 downto 0);
        signal max_velocity2_reg2                   : std_logic_vector(15 downto 0);
        signal max_velocity2_reg1                   : std_logic_vector(15 downto 0);


begin

        input_index <= counter;
        output_index <= counter_reg5;
        Done <= '1' when (counter = "1110" and counter_reg5 = "1110") else '0';
        Update_Wr_En <= '1' when (counter_reg4 > "0000" and counter_reg5 < "1110")
else '0';

  CNTR:process(clk,Reset)
  begin
    if(Reset = '1') then
            counter <= (others => '0');
                        counter_reg1 <= (others => '0');
                        counter_reg2 <= (others => '0');
                        counter_reg3 <= (others => '0');
                        counter_reg4 <= (others => '0');
                        counter_reg5 <= (others => '0');
                        Start_reg <= '0';
          elsif(rising_edge(clk)) then
                        Start_Reg <= Start;
                counter_reg1 <= counter;
                counter_reg2 <= counter_reg1;
                        counter_reg3 <= counter_reg2;
                        counter_reg4 <= counter_reg3;
                        counter_reg5 <= counter_reg4;
                if(Start = '0' and counter_reg4 = "1110") then
                          counter <= "0000";
                        elsif((Start = '1' and Start_reg = '0') or (counter > "0000" and
counter < "1110")) then
              counter <= counter + 1;
                  end if;
          end if;
  end process;


        DELAY_REGS:process(clk)
        begin
                if(rising_edge(clk)) then
```

```
                        next_position1   <= next_position1_reg2;
--                      next_position1_reg3 <= next_position1_reg2;
                        next_position1_reg2 <= next_position1_reg1;

                        next_position2   <= next_position2_reg2;
--                      next_position2_reg3 <= next_position2_reg2;
                        next_position2_reg2 <= next_position2_reg1;

                        max_position1_reg1 <= max_position1;
                        max_position2_reg1 <= max_position2;
                        min_position1_reg1 <= min_position1;
                        min_position2_reg1 <= min_position2;

                        max_velocity1_reg3 <= max_velocity1;
                        max_velocity1_reg2 <= max_velocity1_reg3;
                        max_velocity1_reg1 <= max_velocity1_reg2;

                        max_velocity2_reg3 <= max_velocity2;
                        max_velocity2_reg2 <= max_velocity2_reg3;
                        max_velocity2_reg1 <= max_velocity2_reg2;

                end if;
        end process;


        UPDATE_EQS:process(clk)
  begin
    if(rising_edge(clk)) then

                        --Calculate the next position
                        next_position1_i <= current_position1 + current_velocity1;
                        next_position2_i <= current_position2 + current_velocity2;

                        --check for max and min position1
                        if(next_position1_i > max_position1_reg1) then
                                next_position1_reg1 <= max_position1_reg1;
                        elsif(next_position1_i < min_position1_reg1) then
                          next_position1_reg1 <= min_position1_reg1;
                        else next_position1_reg1 <= next_position1_i;
                        end if;

                        --check for max and min position2
                        if(next_position2_i > max_position2_reg1) then
                                next_position2_reg1 <= max_position2_reg1;
                        elsif(next_position2_i < min_position2_reg1) then
                          next_position2_reg1 <= min_position2_reg1;
```

```
                        else next_position2_reg1 <= next_position2_i;
                        end if;

                        --Calculate the local term difference
                        diff_local1 <= local_best1 - current_position1;
                        diff_local2 <= local_best2 - current_position2;

                        --Calculate the global term difference
                        diff_global1 <= global_best1 - current_position1;
                        diff_global2 <= global_best2 - current_position2;

                        --add the two terms together
                        next_velocity1_i <= local_term1(30 downto 15) +
global_term1(30 downto 15);
                        next_velocity2_i <= local_term2(30 downto 15) +
global_term2(30 downto 15);

                        --check for max velocity1
                        if(next_velocity1_i(15) = '0' and next_velocity1_i >
max_velocity1_reg1 and max_velocity1_reg1(15) = '0') then
                                next_velocity1 <= max_velocity1_reg1;
                        elsif(next_velocity1_i(15) = '1' and (0-next_velocity1_i) >
max_velocity1_reg1 and max_velocity1_reg1(15) = '0') then
                          next_velocity1 <= 0-max_velocity1_reg1;
                        elsif(next_velocity1_i(15) = '0' and next_velocity1_i > (0-
max_velocity1_reg1) and max_velocity1_reg1(15) = '1') then
                                next_velocity1 <= 0-max_velocity1_reg1;
                        elsif(next_velocity1_i(15) = '1' and (0-next_velocity1_i) > (0-
max_velocity1_reg1) and max_velocity1_reg1(15) = '1') then
                          next_velocity1 <= max_velocity1_reg1;
                        else next_velocity1 <= next_velocity1_i;
                        end if;

                        --check for max velocity2
                        if(next_velocity2_i(15) = '0' and next_velocity2_i >
max_velocity2_reg1 and max_velocity2_reg1(15) = '0') then
                                next_velocity2 <= max_velocity2_reg1;
                        elsif(next_velocity2_i(15) = '1' and (0-next_velocity2_i) >
max_velocity2_reg1 and max_velocity2_reg1(15) = '0') then
                          next_velocity2 <= 0-max_velocity2_reg1;
                        elsif(next_velocity2_i(15) = '0' and next_velocity2_i > (0-
max_velocity2_reg1) and max_velocity2_reg1(15) = '1') then
                                next_velocity2 <= 0-max_velocity2_reg1;
                        elsif(next_velocity2_i(15) = '1' and (0-next_velocity2_i) > (0-
max_velocity2_reg1) and max_velocity2_reg1(15) = '1') then
                          next_velocity2 <= max_velocity2_reg1;
```

```
                    else next_velocity2 <= next_velocity2_i;
                    end if;

                end if;
        end process;

        --sign extent the difference
        diff_local1_sign_ext <= diff_local1(15) & diff_local1(15) & diff_local1;
        diff_local2_sign_ext <= diff_local2(15) & diff_local2(15) & diff_local2;
        diff_global1_sign_ext <= diff_global1(15) & diff_global1(15) & diff_global1;
        diff_global2_sign_ext <= diff_global2(15) & diff_global2(15) & diff_global2;

        --sign extent the constants
        constant_local_sign_ext <= constant_local(15) & constant_local(15) &
constant_local;
        constant_global_sign_ext <= constant_global(15) & constant_global(15) &
constant_global;



  MULT_LOCAL1: MULT18X18S
  port map (
    P => local_term1,
    A => diff_local1_sign_ext,
    B => constant_local_sign_ext,
    C=> clk, CE=>'1', R=>'0');


  MULT_LOCAL2: MULT18X18S
  port map (
    P => local_term2,
    A => diff_local2_sign_ext,
    B => constant_local_sign_ext,
    C=> clk, CE=>'1', R=>'0');


  MULT_GLOBAL1: MULT18X18S
  port map (
    P => global_term1,
    A => diff_global1_sign_ext,
    B => constant_global_sign_ext,
    C=> clk, CE=>'1', R=>'0');

  MULT_GLOBAL2: MULT18X18S
  port map (
    P => global_term2,
```

```
    A => diff_global2_sign_ext,
    B => constant_global_sign_ext,
    C=> clk, CE=>'1', R=>'0');
end Behavioral;
```

Sonar_NN.vhd

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

--all control signals are active high
--clk -> clock
--Start -> Once the Inputs are all set, Start begins the computation
--Reset -> Asynchronous Reset, resets all registers to 0 and sets the NN into a waiting to
start state
--Done -> Specifies when all of the NN Outputs have been calculated
--Input?? -> Inputs to the NN
--OutputWrEN -> Enable specifing when OutputNode has valid data. Also an Enable for
Output RAM
--OutNodeNum -> Node Number of the Output Nodes (0-1199). Used as an address for
Output RAM
--OutputNode -> Output value of the Node specified by OutNodeNum

entity Sonar_NN is
  Port(    clk          : in  STD_LOGIC;
           Start        : in  STD_LOGIC;
           Reset        : in  STD_LOGIC;
                          Ready                    : out STD_LOGIC;
                          Done                     : out STD_LOGIC;
                          OutputWrEn          : out STD_LOGIC;
                          OutNodeNum          : out STD_LOGIC_VECTOR(11
downto 0);
           OutputNode     : out STD_LOGIC_VECTOR(15 downto 0);
           Input00        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input01        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input02        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input03        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input04        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input05        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input06        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input07        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input08        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input09        : in  STD_LOGIC_VECTOR(15 downto 0);
```

```
        Input10      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input11      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input12      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input13      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input14      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input15      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input16      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input17      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input18      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input19      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input20      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input21      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input22      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input23      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input24      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input25      : in  STD_LOGIC_VECTOR(15 downto 0);
        Input26      : in  STD_LOGIC_VECTOR(15 downto 0));
end Sonar_NN;

architecture Behavioral of Sonar_NN is


signal OutputWrEn_i                 : STD_LOGIC;
signal RegisterSelect       : STD_LOGIC_VECTOR(11 downto 0);
signal SquashOut           : STD_LOGIC_VECTOR(7 downto 0);
signal NodeCalcOut          : STD_LOGIC_VECTOR(31 downto 0);
signal NodeNumberOut                        : STD_LOGIC_VECTOR(11 downto 0);
signal NodeNumber                           : STD_LOGIC_VECTOR(11 downto 0);
signal NodeNumber_i       : STD_LOGIC_VECTOR(11 downto 0);
signal HoldReg                                      : STD_LOGIC_VECTOR(15
downto 0);

type type1 is array (0 to 70)   of std_logic_vector (7 downto 0);
type type2 is array (0 to 70)   of std_logic_vector (15 downto 0);
signal RegA             : type1;
signal RegB             : type2;


        component NodeCounter is
                Port ( clk                  : in  STD_LOGIC;
         Start                          : in  STD_LOGIC;
         Reset                          : in  STD_LOGIC;
                                Done                                    : out STD_LOGIC;
                            NodeNumber        : out  STD_LOGIC_VECTOR(11 downto
0));
        end component;
```

```
component Squash
  Port ( Input_Value      : in  STD_LOGIC_VECTOR (31 downto 0);
         Squashed_Value   : out STD_LOGIC_VECTOR (7 downto 0);
         clk              : in  STD_LOGIC);
end component;


component NodeCalc is
  Port(  clk              : in  STD_LOGIC;
                                Reset                          : in  STD_LOGIC;
         NodeNumberIn     : in  STD_LOGIC_VECTOR(11 downto 0);
         NodeNumberOut    : out STD_LOGIC_VECTOR(11 downto 0);
         Output           : out STD_LOGIC_VECTOR(31 downto 0);
         Input00          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input01          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input02          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input03          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input04          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input05          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input06          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input07          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input08          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input09          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input10          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input11          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input12          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input13          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input14          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input15          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input16          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input17          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input18          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input19          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input20          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input21          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input22          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input23          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input24          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input25          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input26          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input27          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input28          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input29          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input30          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input31          : in  STD_LOGIC_VECTOR(15 downto 0);
         Input32          : in  STD_LOGIC_VECTOR(15 downto 0);
```

```
        Input33         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input34         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input35         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input36         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input37         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input38         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input39         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input40         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input41         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input42         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input43         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input44         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input45         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input46         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input47         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input48         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input49         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input50         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input51         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input52         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input53         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input54         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input55         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input56         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input57         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input58         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input59         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input60         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input61         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input62         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input63         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input64         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input65         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input66         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input67         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input68         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input69         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input70         : in  STD_LOGIC_VECTOR(15 downto 0));
  end component;

begin

        RegALogic:process(clk,Reset)
        begin
                if(Reset = '1') then
                        for count in 0 to 70 loop
```

```
                        RegA(count) <= (others => '0');
                    end loop;
                elsif(rising_edge(clk)) then
                    if(OutputWrEn_i = '0') then
                        RegA(conv_integer(RegisterSelect)) <= SquashOut;
                    end if;
                end if;
        end process;


        RegBlogic: process(clk,Reset)
        begin
                if(Reset = '1') then
                    for count in 0 to 70 loop
                        RegB(count) <= (others => '0');
                    end loop;
                elsif(rising_edge(clk)) then
                    if(NodeNumber = x"000" or NodeNumber = x"028" or
NodeNumber = x"05a" or NodeNumber = x"0a0") then

                        if(NodeNumber = x"000") then
          RegB(0) <= Input00;
          RegB(1) <= Input01;
          RegB(2) <= Input02;
          RegB(3) <= Input03;
          RegB(4) <= Input04;
          RegB(5) <= Input05;
          RegB(6) <= Input06;
          RegB(7) <= Input07;
          RegB(8) <= Input08;
          RegB(9) <= Input09;
          RegB(10) <= Input10;
          RegB(11) <= Input11;
          RegB(12) <= Input12;
          RegB(13) <= Input13;
          RegB(14) <= Input14;
          RegB(15) <= Input15;
          RegB(16) <= Input16;
          RegB(17) <= Input17;
          RegB(18) <= Input18;
          RegB(19) <= Input19;
          RegB(20) <= Input20;
          RegB(21) <= Input21;
          RegB(22) <= Input22;
          RegB(23) <= Input23;
          RegB(24) <= Input24;
```

```
            RegB(25) <= Input25;
            RegB(26) <= Input26;
                                    RegB(27) <= "0000010000000000";
                                else
                                    for count in 0 to 27 loop
                                        RegB(count) <= "00000000" &
RegA(count);
                                    end loop;
                                end if;

                                for count in 28 to 39 loop
                                    RegB(count) <= "00000000" & RegA(count);
                                end loop;

                                if(NodeNumber = x"028") then
                                    RegB(40) <= "0000000100000000";
                                else    RegB(40) <= "00000000" & RegA(40);
                                end if;

                                for count in 41 to 49 loop
                                    RegB(count) <= "00000000" & RegA(count);
                                end loop;

                                if(NodeNumber = x"05a") then
                                    RegB(50) <= "0000000100000000";
                                else    RegB(50) <= "00000000" & RegA(50);
                                end if;

                                for count in 51 to 69 loop
                                    RegB(count) <= "00000000" & RegA(count);
                                end loop;

                                if(NodeNumber  = x"0a0") then
                                    RegB(70) <= "0000000100000000";
                                else    RegB(70) <= "00000000" & RegA(70);
                                end if;

                        end if;
                end if;
        end process;


        OutHoldReg:process(clk,Reset)
        begin
                if(Reset = '1') then
                        HoldReg <= (others => '0');
```

```
            elsif(rising_edge(clk)) then
                HoldReg <= NodeCalcOut(24 downto 9);
            end if;
        end process;


        NodeReg:process(clk,Reset)
        begin
            if(Reset = '1') then
                NodeNumberOut <= (others => '0');
            elsif(rising_edge(clk)) then
                NodeNumberOut <= NodeNumber_i;
            end if;
        end process;


        --Output Logic
        OutputWrEn_i <= '1' when NodeNumberOut >= x"0a0" else '0';
        OutputWrEn <= OutputWrEn_i;
        OutputNode <= HoldReg;
        OutNodeNum <= RegisterSelect;


        RegisterSelect <= NodeNumberOut when (NodeNumberOut < x"028") else
                                                NodeNumberOut - 40 when
(NodeNumberOut >= x"028" and NodeNumberOut < x"05a") else
                                                NodeNumberOut - 90 when
(NodeNumberOut >= x"05a" and NodeNumberOut < x"0a0") else
                                                NodeNumberOut - 160 when
(NodeNumberOut >= x"0a0");



  U1 : NodeCounter
    port map ( clk => clk,
            Start => Start,
            Reset => Reset,
                                        Done => Done,
                                NodeNumber => NodeNumber);

  U2 : Squash
    port map ( Input_Value => NodeCalcOut,
            Squashed_Value => SquashOut,
            clk => clk);

  U3 : NodeCalc
```

```
port map ( clk => clk,
                                        Reset => Reset,
        NodeNumberIn => NodeNumber,
        NodeNumberOut => NodeNumber_i,
        Output => NodeCalcOut,
        Input00 => RegB(0),
        Input01 => RegB(1),
        Input02 => RegB(2),
        Input03 => RegB(3),
        Input04 => RegB(4),
        Input05 => RegB(5),
        Input06 => RegB(6),
        Input07 => RegB(7),
        Input08 => RegB(8),
        Input09 => RegB(9),
        Input10 => RegB(10),
        Input11 => RegB(11),
        Input12 => RegB(12),
        Input13 => RegB(13),
        Input14 => RegB(14),
        Input15 => RegB(15),
        Input16 => RegB(16),
        Input17 => RegB(17),
        Input18 => RegB(18),
        Input19 => RegB(19),
        Input20 => RegB(20),
        Input21 => RegB(21),
        Input22 => RegB(22),
        Input23 => RegB(23),
        Input24 => RegB(24),
        Input25 => RegB(25),
        Input26 => RegB(26),
        Input27 => RegB(27),
        Input28 => RegB(28),
        Input29 => RegB(29),
        Input30 => RegB(30),
        Input31 => RegB(31),
        Input32 => RegB(32),
        Input33 => RegB(33),
        Input34 => RegB(34),
        Input35 => RegB(35),
        Input36 => RegB(36),
        Input37 => RegB(37),
        Input38 => RegB(38),
        Input39 => RegB(39),
        Input40 => RegB(40),
```

```
            Input41 => RegB(41),
            Input42 => RegB(42),
            Input43 => RegB(43),
            Input44 => RegB(44),
            Input45 => RegB(45),
            Input46 => RegB(46),
            Input47 => RegB(47),
            Input48 => RegB(48),
            Input49 => RegB(49),
            Input50 => RegB(50),
            Input51 => RegB(51),
            Input52 => RegB(52),
            Input53 => RegB(53),
            Input54 => RegB(54),
            Input55 => RegB(55),
            Input56 => RegB(56),
            Input57 => RegB(57),
            Input58 => RegB(58),
            Input59 => RegB(59),
            Input60 => RegB(60),
            Input61 => RegB(61),
            Input62 => RegB(62),
            Input63 => RegB(63),
            Input64 => RegB(64),
            Input65 => RegB(65),
            Input66 => RegB(66),
            Input67 => RegB(67),
            Input68 => RegB(68),
            Input69 => RegB(69),
            Input70 => RegB(70));
end Behavioral;
```

NodeCalc.vhd

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


entity NodeCalc is
  Port(    clk          : in  STD_LOGIC;
                                Reset                                : in  STD_LOGIC;
        NodeNumberIn   : in  STD_LOGIC_VECTOR(11 downto 0);
        NodeNumberOut  : out STD_LOGIC_VECTOR(11 downto 0);
```

```
Output      : out STD_LOGIC_VECTOR(31 downto 0);
Input00     : in  STD_LOGIC_VECTOR(15 downto 0);
Input01     : in  STD_LOGIC_VECTOR(15 downto 0);
Input02     : in  STD_LOGIC_VECTOR(15 downto 0);
Input03     : in  STD_LOGIC_VECTOR(15 downto 0);
Input04     : in  STD_LOGIC_VECTOR(15 downto 0);
Input05     : in  STD_LOGIC_VECTOR(15 downto 0);
Input06     : in  STD_LOGIC_VECTOR(15 downto 0);
Input07     : in  STD_LOGIC_VECTOR(15 downto 0);
Input08     : in  STD_LOGIC_VECTOR(15 downto 0);
Input09     : in  STD_LOGIC_VECTOR(15 downto 0);
Input10     : in  STD_LOGIC_VECTOR(15 downto 0);
Input11     : in  STD_LOGIC_VECTOR(15 downto 0);
Input12     : in  STD_LOGIC_VECTOR(15 downto 0);
Input13     : in  STD_LOGIC_VECTOR(15 downto 0);
Input14     : in  STD_LOGIC_VECTOR(15 downto 0);
Input15     : in  STD_LOGIC_VECTOR(15 downto 0);
Input16     : in  STD_LOGIC_VECTOR(15 downto 0);
Input17     : in  STD_LOGIC_VECTOR(15 downto 0);
Input18     : in  STD_LOGIC_VECTOR(15 downto 0);
Input19     : in  STD_LOGIC_VECTOR(15 downto 0);
Input20     : in  STD_LOGIC_VECTOR(15 downto 0);
Input21     : in  STD_LOGIC_VECTOR(15 downto 0);
Input22     : in  STD_LOGIC_VECTOR(15 downto 0);
Input23     : in  STD_LOGIC_VECTOR(15 downto 0);
Input24     : in  STD_LOGIC_VECTOR(15 downto 0);
Input25     : in  STD_LOGIC_VECTOR(15 downto 0);
Input26     : in  STD_LOGIC_VECTOR(15 downto 0);
Input27     : in  STD_LOGIC_VECTOR(15 downto 0);
Input28     : in  STD_LOGIC_VECTOR(15 downto 0);
Input29     : in  STD_LOGIC_VECTOR(15 downto 0);
Input30     : in  STD_LOGIC_VECTOR(15 downto 0);
Input31     : in  STD_LOGIC_VECTOR(15 downto 0);
Input32     : in  STD_LOGIC_VECTOR(15 downto 0);
Input33     : in  STD_LOGIC_VECTOR(15 downto 0);
Input34     : in  STD_LOGIC_VECTOR(15 downto 0);
Input35     : in  STD_LOGIC_VECTOR(15 downto 0);
Input36     : in  STD_LOGIC_VECTOR(15 downto 0);
Input37     : in  STD_LOGIC_VECTOR(15 downto 0);
Input38     : in  STD_LOGIC_VECTOR(15 downto 0);
Input39     : in  STD_LOGIC_VECTOR(15 downto 0);
Input40     : in  STD_LOGIC_VECTOR(15 downto 0);
Input41     : in  STD_LOGIC_VECTOR(15 downto 0);
Input42     : in  STD_LOGIC_VECTOR(15 downto 0);
Input43     : in  STD_LOGIC_VECTOR(15 downto 0);
Input44     : in  STD_LOGIC_VECTOR(15 downto 0);
```

```
        Input45     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input46     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input47     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input48     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input49     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input50     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input51     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input52     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input53     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input54     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input55     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input56     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input57     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input58     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input59     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input60     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input61     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input62     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input63     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input64     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input65     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input66     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input67     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input68     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input69     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input70     : in  STD_LOGIC_VECTOR(15 downto 0));
end NodeCalc;

architecture Behavioral of NodeCalc is

signal NodeNumber1_i : STD_LOGIC_VECTOR(11 downto 0);
signal NodeNumber2_i : STD_LOGIC_VECTOR(11 downto 0);
signal NodeNumber3_i : STD_LOGIC_VECTOR(11 downto 0);
signal NodeNumber4_i : STD_LOGIC_VECTOR(11 downto 0);

signal TotalSum_i     : STD_LOGIC_VECTOR(35 downto 0);

signal  Sum0_3_i      : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum4_7_i      : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum8_11_i     : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum12_15_i    : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum16_19_i    : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum20_23_i    : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum24_27_i    : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum28_31_i    : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum32_35_i    : STD_LOGIC_VECTOR(35 downto 0);
```

```
signal  Sum36_39_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum40_43_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum44_47_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum48_51_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum52_55_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum56_59_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum60_63_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum64_67_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum68_70_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum0_15_i    : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum16_31_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum32_47_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum48_63_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum64_70_i   : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum0_31_i    : STD_LOGIC_VECTOR(35 downto 0);
signal  Sum32_70_i   : STD_LOGIC_VECTOR(35 downto 0);



signal Weight00     : STD_LOGIC_VECTOR(15 downto 0);
signal Weight01     : STD_LOGIC_VECTOR(15 downto 0);
signal Weight02     : STD_LOGIC_VECTOR(15 downto 0);
signal Weight03     : STD_LOGIC_VECTOR(15 downto 0);
signal Weight04     : STD_LOGIC_VECTOR(15 downto 0);

!!!CODE REMOVED!!!

signal Weight68     : STD_LOGIC_VECTOR(15 downto 0);
signal Weight69     : STD_LOGIC_VECTOR(15 downto 0);
signal Weight70     : STD_LOGIC_VECTOR(15 downto 0);

signal MultOut00    : STD_LOGIC_VECTOR(35 downto 0);
signal MultOut01    : STD_LOGIC_VECTOR(35 downto 0);
signal MultOut02    : STD_LOGIC_VECTOR(35 downto 0);
signal MultOut03    : STD_LOGIC_VECTOR(35 downto 0);
signal MultOut04    : STD_LOGIC_VECTOR(35 downto 0);

!!!CODE REMOVED!!!

signal MultOut69    : STD_LOGIC_VECTOR(35 downto 0);
signal MultOut70    : STD_LOGIC_VECTOR(35 downto 0);

  component Weigths
    Port(   clk          : in  STD_LOGIC;
          enable         : in  STD_LOGIC;
                                    Reset         : in  STD_LOGIC;
```

```
                                        NodeNumberIn   : in  STD_LOGIC_VECTOR (11
downto 0);
                                        NodeNumberOut  : out  STD_LOGIC_VECTOR
(11 downto 0);
            Weight00      : out  STD_LOGIC_VECTOR (15 downto 0);
            Weight01      : out  STD_LOGIC_VECTOR (15 downto 0);
            Weight02      : out  STD_LOGIC_VECTOR (15 downto 0);
            Weight03      : out  STD_LOGIC_VECTOR (15 downto 0);

            Weight68      : out  STD_LOGIC_VECTOR (15 downto 0);
            Weight69      : out  STD_LOGIC_VECTOR (15 downto 0);
            Weight70      : out  STD_LOGIC_VECTOR (15 downto 0));
    end component;

    component Multiply
      Port(    clk          : in  STD_LOGIC;
            Weight00      : in  STD_LOGIC_VECTOR(15 downto 0);
            Weight01      : in  STD_LOGIC_VECTOR(15 downto 0);
            Weight02      : in  STD_LOGIC_VECTOR(15 downto 0);
            Weight03      : in  STD_LOGIC_VECTOR(15 downto 0);
            Weight04      : in  STD_LOGIC_VECTOR(15 downto 0);

            Weight66      : in  STD_LOGIC_VECTOR(15 downto 0);
            Weight67      : in  STD_LOGIC_VECTOR(15 downto 0);
            Weight68      : in  STD_LOGIC_VECTOR(15 downto 0);
            Weight69      : in  STD_LOGIC_VECTOR(15 downto 0);
            Weight70      : in  STD_LOGIC_VECTOR(15 downto 0);
            PrevNode00    : in  STD_LOGIC_VECTOR(15 downto 0);
            PrevNode01    : in  STD_LOGIC_VECTOR(15 downto 0);
            PrevNode02    : in  STD_LOGIC_VECTOR(15 downto 0);
            PrevNode03    : in  STD_LOGIC_VECTOR(15 downto 0);
            PrevNode04    : in  STD_LOGIC_VECTOR(15 downto 0);

            PrevNode69    : in  STD_LOGIC_VECTOR(15 downto 0);
            PrevNode70    : in  STD_LOGIC_VECTOR(15 downto 0);
            Output00      : out STD_LOGIC_VECTOR(35 downto 0);
            Output01      : out STD_LOGIC_VECTOR(35 downto 0);
            Output02      : out STD_LOGIC_VECTOR(35 downto 0);
            Output03      : out STD_LOGIC_VECTOR(35 downto 0);
            Output04      : out STD_LOGIC_VECTOR(35 downto 0);
            Output05      : out STD_LOGIC_VECTOR(35 downto 0);
            Output06      : out STD_LOGIC_VECTOR(35 downto 0);
            Output07      : out STD_LOGIC_VECTOR(35 downto 0);

            Output68      : out STD_LOGIC_VECTOR(35 downto 0);
            Output69      : out STD_LOGIC_VECTOR(35 downto 0);
```

```
            Output70       : out STD_LOGIC_VECTOR(35 downto 0));
  end component;

begin

  process (clk,Reset)
  begin
    if(Reset = '1') then
      NodeNumberOut <= (others => '0');
                        NodeNumber2_i <= (others => '0');
                        NodeNumber3_i <= (others => '0');
                        NodeNumber4_i <= (others => '0');
    elsif(clk'event and clk = '1') then
                        NodeNumber2_i <= NodeNumber1_i;
                        NodeNumber3_i <= NodeNumber2_i;
                        NodeNumber4_i <= NodeNumber3_i;
      NodeNumberOut <= NodeNumber4_i;
    end if;
  end process;

  Output <= TotalSum_i(31 downto 0);
        TotalSum_i <= Sum0_31_i + Sum32_70_i;

        process(clk,Reset)
        begin
                if(Reset = '1') then
                        Sum0_3_i   <= (others => '0');
                        Sum4_7_i   <= (others => '0');
                        Sum8_11_i  <= (others => '0');
                        Sum12_15_i <= (others => '0');
                        Sum16_19_i <= (others => '0');
                        Sum20_23_i <= (others => '0');
                        Sum24_27_i <= (others => '0');
                        Sum28_31_i <= (others => '0');
                        Sum32_35_i <= (others => '0');
                        Sum36_39_i <= (others => '0');
                        Sum40_43_i <= (others => '0');
                        Sum44_47_i <= (others => '0');
                        Sum48_51_i <= (others => '0');
                        Sum52_55_i <= (others => '0');
                        Sum56_59_i <= (others => '0');
                        Sum60_63_i <= (others => '0');
                        Sum64_67_i <= (others => '0');
                        Sum68_70_i <= (others => '0');
                        Sum0_15_i  <= (others => '0');
                        Sum16_31_i <= (others => '0');
```

```
                Sum32_47_i <= (others => '0');
                Sum48_63_i <= (others => '0');
                Sum64_70_i <= (others => '0');
                Sum0_31_i  <= (others => '0');
                Sum32_70_i <= (others => '0');
            elsif(rising_edge(clk)) then

                Sum0_3_i   <= MultOut00 + MultOut01 + MultOut02 +
MultOut03;
                Sum4_7_i   <= MultOut04 + MultOut05 + MultOut06 +
MultOut07;
                Sum8_11_i  <= MultOut08 + MultOut09 + MultOut10 +
MultOut11;
                Sum12_15_i <= MultOut12 + MultOut13 + MultOut14 +
MultOut15;
                Sum16_19_i <= MultOut16 + MultOut17 + MultOut18 +
MultOut19;
                Sum20_23_i <= MultOut20 + MultOut21 + MultOut22 +
MultOut23;
                Sum24_27_i <= MultOut24 + MultOut25 + MultOut26 +
MultOut27;
                Sum28_31_i <= MultOut28 + MultOut29 + MultOut30 +
MultOut31;
                Sum32_35_i <= MultOut32 + MultOut33 + MultOut34 +
MultOut35;
                Sum36_39_i <= MultOut36 + MultOut37 + MultOut38 +
MultOut39;
                Sum40_43_i <= MultOut40 + MultOut41 + MultOut42 +
MultOut43;
                Sum44_47_i <= MultOut44 + MultOut45 + MultOut46 +
MultOut47;
                Sum48_51_i <= MultOut48 + MultOut49 + MultOut50 +
MultOut51;
                Sum52_55_i <= MultOut52 + MultOut53 + MultOut54 +
MultOut55;
                Sum56_59_i <= MultOut56 + MultOut57 + MultOut58 +
MultOut59;
                Sum60_63_i <= MultOut60 + MultOut61 + MultOut62 +
MultOut63;
                Sum64_67_i <= MultOut64 + MultOut65 + MultOut66 +
MultOut67;
                Sum68_70_i <= MultOut68 + MultOut69 + MultOut70;

                Sum0_15_i  <= Sum0_3_i   + Sum4_7_i   + Sum8_11_i  +
Sum12_15_i;
```

```
                    Sum16_31_i <= Sum16_19_i + Sum20_23_i + Sum24_27_i +
Sum28_31_i;
                    Sum32_47_i <= Sum32_35_i + Sum36_39_i + Sum40_43_i +
Sum44_47_i;
                    Sum48_63_i <= Sum48_51_i + Sum52_55_i + Sum56_59_i +
Sum60_63_i;
                    Sum64_70_i <= Sum64_67_i + Sum68_70_i;

                    Sum0_31_i  <= Sum0_15_i + Sum16_31_i;
                    Sum32_70_i <= Sum32_47_i + Sum48_63_i + Sum64_70_i;

                end if;
        end process;



  U1 : Weigths
    port map ( clk => clk,
               enable => '1',
                                    Reset => Reset,
                                    NodeNumberIn => NodeNumberIn,
                                    NodeNumberOut => NodeNumber1_i,
               Weight00 => Weight00,
               Weight01 => Weight01,
               Weight02 => Weight02,
               Weight03 => Weight03,
               Weight04 => Weight04,

                                !!!CODE REMOVED!!!

               Weight68 => Weight68,
               Weight69 => Weight69,
               Weight70 => Weight70);

  U2 : Multiply
    port map ( clk => clk,
               Weight00 => Weight00,
               Weight01 => Weight01,
               Weight02 => Weight02,
               Weight03 => Weight03,
               Weight04 => Weight04,

                                !!!CODE REMOVED!!!

               Weight66 => Weight66,
               Weight67 => Weight67,
```

```
            Weight68 => Weight68,
            Weight69 => Weight69,
            Weight70 => Weight70,
            PrevNode00 => Input00,
            PrevNode01 => Input01,
            PrevNode02 => Input02,
            PrevNode03 => Input03,
            PrevNode04 => Input04,

                                !!!CODE REMOVED!!!

            PrevNode66 => Input66,
            PrevNode67 => Input67,
            PrevNode68 => Input68,
            PrevNode69 => Input69,
            PrevNode70 => Input70,
            Output00 => MultOut00,
            Output01 => MultOut01,
            Output02 => MultOut02,

                                !!!CODE REMOVED!!!

            Output66 => MultOut66,
            Output67 => MultOut67,
            Output68 => MultOut68,
            Output69 => MultOut69,
            Output70 => MultOut70);

end Behavioral;
```

NodeCounter.vhd

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


entity NodeCounter is
   Port ( clk                  : in  STD_LOGIC;
          Start                : in  STD_LOGIC;
          Reset                : in  STD_LOGIC;
                  Ready        : out STD_LOGIC;
                  Done                 : out STD_LOGIC;
```

```
                        NodeNumber          : out  STD_LOGIC_VECTOR(11 downto
0));
end NodeCounter;

architecture Behavioral of NodeCounter is


type state_type is (inital,count,wait1,wait2,finished,wait3);
signal state              : state_type;
signal next_state : state_type;
signal IncrNodeCount : STD_LOGIC;
signal IncrWaitCount : STD_LOGIC;
signal SetNodeCount0 : STD_LOGIC;
signal SetWaitCount0 : STD_LOGIC;
signal NodeCount  : STD_LOGIC_VECTOR(11 downto 0);
signal WaitCount  : STD_LOGIC_VECTOR(3 downto 0);
constant Latency : integer := 7;

begin

NodeNumber <= NodeCount;

  --ready tells when the first output is available
        --done tells when all of the outputs are available
  Ready_Logic: process(clk,Reset)
        begin
                if(Reset='1') then
                        Ready <= '0';
                elsif(rising_edge(clk)) then
                  if(NodeCount > x"0a7") then
                    Ready <= '1';
                        else Ready <= '0';
                        end if;
                end if;
        end process;


        NodeCounter: process(clk,Reset)
        begin
                if(Reset='1') then
                        NodeCount <= (others => '0');
                elsif(rising_edge(clk)) then
                        if(IncrNodeCount = '1') then
                                NodeCount <= NodeCount + 1;
                        elsif(SetNodeCount0 = '1') then
                                NodeCount <= (others => '0');
```

```
                end if;
        end if;
end process;

WaitCounter: process(clk,Reset)
begin
        if(Reset='1') then
                WaitCount <= (others => '0');
        elsif(rising_edge(clk)) then
                if(IncrWaitCount = '1') then
                        WaitCount <= WaitCount + 1;
                elsif(SetWaitCount0 = '1') then
                        WaitCount <= (others => '0');
                end if;
        end if;
end process;

StateMemory: process(clk,Reset)
begin
        if(Reset='1') then
                state <= inital;
        elsif(rising_edge(clk)) then
                state <= next_state;
        end if;
end process;

NextStateLogic: process(state,Start,NodeCount,WaitCount)
begin
        case state is

                when inital =>
                  if(Start = '1') then
                        next_state <= count;
                  else next_state <= inital;
                  end if;

                when count =>
                        --in decimal  if(NodeCount = 38 or NodeCount = 88 or
NodeCount = 158) then
                        --  40-2=38             40+50-2=88             40+50+70-
2=158
                  if(NodeCount = x"026" or NodeCount = x"058" or NodeCount =
x"09e") then
                        next_state <= wait1;
                  elsif(NodeCount = x"54e") then
                        next_state <= wait2;
```

```
                    else next_state <= count;
                    end if;

                when wait1 =>
                  if(conv_integer(WaitCount) = Latency-2) then
                        next_state <= count;
                    else next_state <= wait1;
                    end if;

                when wait2 =>
                  if(conv_integer(WaitCount) = Latency-1) then
                        next_state <= finished;
                    else next_state <= wait2;
                    end if;

                when finished =>
                  if(Start = '1') then
                        next_state <= finished;
                    else next_state <= wait3;
                    end if;

                when wait3 =>
                  if(Start = '1') then
                        next_state <= count;
                    else next_state <= wait3;
                    end if;

        end case;
end process;

OutputLogic: process(state,Start,NodeCount,WaitCount)
begin

        Done <= '0';
        IncrNodeCount <= '0';
        SetNodeCount0 <= '0';
        IncrWaitCount <= '0';
        SetWaitCount0 <= '0';

        if(state = inital) then
                SetNodeCount0 <= '1';
        end if;

        if(state = count) then
                SetWaitCount0 <= '1';
                IncrNodeCount <= '1';
```

```
                end if;

                if(state = wait1) then
                        IncrWaitCount <= '1';
                end if;

                if(state = wait2) then
                        IncrWaitCount <= '1';
                        SetNodeCount0 <= '1';
                end if;

                if(state = finished) then
                        Done <= '1';
                end if;

                if(state = wait3) then
                        SetNodeCount0 <= '1';
                  Done <= '1';
          end if;

        end process;

end Behavioral;
```

Multiply.vhd

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

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

entity Multiply is
  Port(    clk          : in  STD_LOGIC;
        Weight00      : in  STD_LOGIC_VECTOR(15 downto 0);
        Weight01      : in  STD_LOGIC_VECTOR(15 downto 0);
        Weight02      : in  STD_LOGIC_VECTOR(15 downto 0);
        Weight03      : in  STD_LOGIC_VECTOR(15 downto 0);

                        !!!CODE REMOVED!!!
```

```
        Weight66      : in  STD_LOGIC_VECTOR(15 downto 0);
        Weight67      : in  STD_LOGIC_VECTOR(15 downto 0);
        Weight68      : in  STD_LOGIC_VECTOR(15 downto 0);
        Weight69      : in  STD_LOGIC_VECTOR(15 downto 0);
        Weight70      : in  STD_LOGIC_VECTOR(15 downto 0);
        PrevNode00    : in  STD_LOGIC_VECTOR(15 downto 0);
        PrevNode01    : in  STD_LOGIC_VECTOR(15 downto 0);
        PrevNode02    : in  STD_LOGIC_VECTOR(15 downto 0);
        PrevNode03    : in  STD_LOGIC_VECTOR(15 downto 0);
        PrevNode04    : in  STD_LOGIC_VECTOR(15 downto 0);

                        !!!CODE REMOVED!!!

        PrevNode67    : in  STD_LOGIC_VECTOR(15 downto 0);
        PrevNode68    : in  STD_LOGIC_VECTOR(15 downto 0);
        PrevNode69    : in  STD_LOGIC_VECTOR(15 downto 0);
        PrevNode70    : in  STD_LOGIC_VECTOR(15 downto 0);
        Output00      : out STD_LOGIC_VECTOR(35 downto 0);
        Output01      : out STD_LOGIC_VECTOR(35 downto 0);
        Output02      : out STD_LOGIC_VECTOR(35 downto 0);
        Output03      : out STD_LOGIC_VECTOR(35 downto 0);
        Output04      : out STD_LOGIC_VECTOR(35 downto 0);

                        !!!CODE REMOVED!!!

        Output68      : out STD_LOGIC_VECTOR(35 downto 0);
        Output69      : out STD_LOGIC_VECTOR(35 downto 0);
        Output70      : out STD_LOGIC_VECTOR(35 downto 0));
end Multiply;

architecture Behavioral of Multiply is

signal WeightExtend00  : STD_LOGIC_VECTOR(17 downto 0);
signal WeightExtend01  : STD_LOGIC_VECTOR(17 downto 0);
signal WeightExtend02  : STD_LOGIC_VECTOR(17 downto 0);
signal WeightExtend03  : STD_LOGIC_VECTOR(17 downto 0);

!!!CODE REMOVED!!!

signal WeightExtend67  : STD_LOGIC_VECTOR(17 downto 0);
signal WeightExtend68  : STD_LOGIC_VECTOR(17 downto 0);
signal WeightExtend69  : STD_LOGIC_VECTOR(17 downto 0);
signal WeightExtend70  : STD_LOGIC_VECTOR(17 downto 0);
signal NodeExtend00    : STD_LOGIC_VECTOR(17 downto 0);
signal NodeExtend01    : STD_LOGIC_VECTOR(17 downto 0);
signal NodeExtend02    : STD_LOGIC_VECTOR(17 downto 0);
```

```
signal NodeExtend03    : STD_LOGIC_VECTOR(17 downto 0);
signal NodeExtend04    : STD_LOGIC_VECTOR(17 downto 0);

!!!CODE REMOVED!!!

signal NodeExtend68    : STD_LOGIC_VECTOR(17 downto 0);
signal NodeExtend69    : STD_LOGIC_VECTOR(17 downto 0);
signal NodeExtend70    : STD_LOGIC_VECTOR(17 downto 0);

begin

WeightExtend00 <= Weight00(15) & Weight00(15) & Weight00;
WeightExtend01 <= Weight01(15) & Weight01(15) & Weight01;
WeightExtend02 <= Weight02(15) & Weight02(15) & Weight02;
WeightExtend03 <= Weight03(15) & Weight03(15) & Weight03;

!!!CODE REMOVED!!!

WeightExtend68 <= Weight68(15) & Weight68(15) & Weight68;
WeightExtend69 <= Weight69(15) & Weight69(15) & Weight69;
WeightExtend70 <= Weight70(15) & Weight70(15) & Weight70;
NodeExtend00   <= PrevNode00(15) & PrevNode00(15) & PrevNode00;
NodeExtend01   <= PrevNode01(15) & PrevNode01(15) & PrevNode01;
NodeExtend02   <= PrevNode02(15) & PrevNode02(15) & PrevNode02;
NodeExtend03   <= PrevNode03(15) & PrevNode03(15) & PrevNode03;
NodeExtend04   <= PrevNode04(15) & PrevNode04(15) & PrevNode04;

!!!CODE REMOVED!!!

NodeExtend68   <= PrevNode68(15) & PrevNode68(15) & PrevNode68;
NodeExtend69   <= PrevNode69(15) & PrevNode69(15) & PrevNode69;
NodeExtend70   <= PrevNode70(15) & PrevNode70(15) & PrevNode70;
  Multiply00: MULT18X18S
  port map (
    P => Output00,
    A => WeightExtend00,
    B => NodeExtend00,
    C=>clk, CE=>'1', R=>'0');

  Multiply01: MULT18X18S
  port map (
    P => Output01,
    A => WeightExtend01,
    B => NodeExtend01,
    C=>clk, CE=>'1', R=>'0');
```

```
  Multiply02: MULT18X18S
  port map (
    P => Output02,
    A => WeightExtend02,
    B => NodeExtend02,
    C=>clk, CE=>'1', R=>'0');

!!!CODE REMOVED!!!

  Multiply69: MULT18X18S
  port map (
    P => Output69,
    A => WeightExtend69,
    B => NodeExtend69,
    C=>clk, CE=>'1', R=>'0');

  Multiply70: MULT18X18S
  port map (
    P => Output70,
    A => WeightExtend70,
    B => NodeExtend70,
    C=>clk, CE=>'1', R=>'0');


end Behavioral;
```

Squash.vhd

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


entity Squash is
   Port ( Input_Value          : in  STD_LOGIC_VECTOR (31 downto 0);
          Squashed_Value       : out  STD_LOGIC_VECTOR (7 downto 0);
          clk                          : in  STD_LOGIC);
end Squash;

architecture syn of Squash is

type LUT is array (0 to 4095) of std_logic_vector (7 downto 0);
signal Sigmoid : LUT;
signal Reg_Value : std_logic_vector(11 downto 0);  -- Latched Read Address
```

```
signal OutofRange : std_logic;
signal Sign : std_logic;

begin
  process (clk)
    begin
      if (clk'event and clk = '1') then
                                  Reg_Value <= Input_Value(19 downto 8);
                                  OutofRange <= (Input_Value(31) and Input_Value(30) and
Input_Value(29) and Input_Value(28) and
                                                                  Input_Value(27) and
Input_Value(26) and Input_Value(25) and Input_Value(24) and
                                                                  Input_Value(23) and
Input_Value(22) and Input_Value(21) and Input_Value(20) and Input_Value(19))
                                                                      xor
                                                                 (Input_Value(31) or
Input_Value(30) or  Input_Value(29) or  Input_Value(28) or
                                                                  Input_Value(27) or
Input_Value(26) or  Input_Value(25) or  Input_Value(24) or
                                                                  Input_Value(23) or
Input_Value(22) or  Input_Value(21) or  Input_Value(20) or  Input_Value(19));
                                  Sign <= Input_Value(31);
      end if;
  end process;

        Squashed_Value <= Sigmoid(conv_integer(Reg_Value)) when OutofRange = '0'
else
                        x"ff" when Sign = '0' else x"00";

        --Define LUT constants for Sigmoid
        --Function: 1/(1+exp(-Input))
                Sigmoid(0) <= x"80";
                Sigmoid(1) <= x"80";
                Sigmoid(2) <= x"81";
                Sigmoid(3) <= x"81";
                Sigmoid(4) <= x"81";
                Sigmoid(5) <= x"81";
                Sigmoid(6) <= x"81";
                Sigmoid(7) <= x"82";
                Sigmoid(8) <= x"82";
                Sigmoid(9) <= x"82";

                !!!CODE REMOVED!!!

                Sigmoid(4092) <= x"7f";
                Sigmoid(4093) <= x"7f";
```

```
                Sigmoid(4094) <= x"80";
                Sigmoid(4095) <= x"80";

end syn;
```

Weights.vhd

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

entity Weigths is
  Port(  clk          : in  STD_LOGIC;
         enable       : in  STD_LOGIC;
         Reset        : in  STD_LOGIC;
         NodeNumberIn   : in  STD_LOGIC_VECTOR (11 downto 0);
         NodeNumberOut  : out STD_LOGIC_VECTOR (11 downto 0);
         Weight00     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight01     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight02     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight03     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight04     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight05     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight06     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight07     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight08     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight09     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight10     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight11     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight12     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight13     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight14     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight15     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight16     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight17     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight18     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight19     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight20     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight21     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight22     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight23     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight24     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight25     : out STD_LOGIC_VECTOR (15 downto 0);
         Weight26     : out STD_LOGIC_VECTOR (15 downto 0);
```

```
        Weight27    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight28    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight29    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight30    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight31    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight32    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight33    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight34    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight35    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight36    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight37    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight38    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight39    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight40    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight41    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight42    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight43    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight44    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight45    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight46    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight47    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight48    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight49    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight50    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight51    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight52    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight53    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight54    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight55    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight56    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight57    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight58    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight59    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight60    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight61    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight62    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight63    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight64    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight65    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight66    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight67    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight68    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight69    : out STD_LOGIC_VECTOR (15 downto 0);
        Weight70    : out STD_LOGIC_VECTOR (15 downto 0));
end Weigths;
```

```
architecture syn of Weigths is

  signal NodeNumberReg : STD_LOGIC_VECTOR (11 downto 0);
  signal WeightReg00a  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg00b  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg01a  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg01b  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg02a  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg02b  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg03a  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg03b  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg04a  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg04b  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg05a  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg05b  : STD_LOGIC_VECTOR (15 downto 0);

  !!!CODE REMOVED!!!

  signal WeightReg67a  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg67b  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg68a  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg68b  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg69a  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg69b  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg70a  : STD_LOGIC_VECTOR (15 downto 0);
  signal WeightReg70b  : STD_LOGIC_VECTOR (15 downto 0);
  type ROM_TYPE is array (0 to 1023) of std_logic_vector (15 downto 0);
  constant WeightROM_00  : ROM_TYPE :=(x"fae3",x"0084",x"009b",x"011f",
      x"0164",x"0256",x"ff58",x"004d",x"037c",x"003f",x"ffe0",x"0054",
      x"0399",x"0493",x"020e",x"074a",x"ff39",x"fdd7",x"001a",x"009e",
      x"febd",x"0157",x"00ea",x"ffdf",x"000d",x"0088",x"ff94",x"0048",
      x"fe92",x"ffb2",x"ff26",x"001d",x"02be",x"fe52",x"fdca",x"03c6",
      x"0022",x"00dd",x"fcf4",x"fc25",x"0121",x"fe81",x"fff1",x"ffa5",
      x"ffd9",x"fed3",x"0034",x"002d",x"fe8f",x"022b",x"0011",x"00e8",
      x"ff09",x"000b",x"fe7f",x"ff3e",x"ff92",x"fe95",x"feba",x"00bb",
      x"0022",x"fe99",x"ffb2",x"fe7b",x"fe48",x"ffc6",x"0051",x"fe0d",
      x"fe69",x"0012",x"fd97",x"fe99",x"fecf",x"ff29",x"0075",x"fdb8",
      x"feb2",x"fe41",x"0001",x"ffd6",x"fea1",x"ff90",x"fe8f",x"ff7c",
      x"fd8d",x"002a",x"ff12",x"ff3f",x"015a",x"fff8",x"00c8",x"ff6c",
      x"ff9c",x"ff68",x"ff52",x"ffbf",x"ff2b",x"ff66",x"ff61",x"ff4f",
      x"0135",x"ff54",x"ff9a",x"fe9f",x"feb0",x"ff6f",x"007e",x"ffe8",
      x"01dc",x"ff7b",x"ff3f",x"fe01",x"fe23",x"ff86",x"0191",x"ff5e",
      x"ff71",x"ff51",x"ff8f",x"feeb",x"ffa0",x"0002",x"fe7a",x"0120",
      x"ff7e",x"ff68",x"fe99",x"ff7b",x"fded",x"ff68",x"ff37",x"ff71",
      x"01cc",x"ff1c",x"ff4b",x"ffac",x"ffa2",x"0269",x"fe7f",x"0073",
      x"fec4",x"fe68",x"ff4c",x"ff9c",x"ffab",x"0197",x"ff6a",x"0056",
```

x"ff65",x"0126",x"ff6a",x"0226",x"ff7c",x"ff46",x"0132",x"fe49",
x"ff70",x"ff5b",x"fff4",x"0052",x"ff4a",x"ffc0",x"0029",x"0098",
x"0095",x"005f",x"00a0",x"0167",x"0285",x"03b2",x"0478",x"049c",
x"0411",x"030d",x"01e5",x"00bc",x"ffb1",x"ff1a",x"ff00",x"ff37",
x"ff6b",x"ff7d",x"ffce",x"ffe3",x"ffdf",x"ffb1",x"ff73",x"ff41",
x"feed",x"fe88",x"fe03",x"fd7e",x"fcec",x"fc7d",x"fbdb",x"fb3c",
x"fad9",x"fa46",x"f9af",x"f8fd",x"f82c",x"f76e",x"f6da",x"f63e",
x"f584",x"f486",x"f3d6",x"f305",x"f24a",x"f164",x"f079",x"ef4c",
x"eeb7",x"ee48",x"ee13",x"ed03",x"eb2c",x"eb11",x"ea2b",x"e957",
x"e798",x"e485",x"e54a",x"e3ee",x"e28b",x"e188",x"e041",x"df43",
x"de37",x"dd00",x"db20",x"d982",x"d7e4",x"d6ed",x"d564",x"d459",
x"d3a8",x"d311",x"d2b1",x"d131",x"ff7d",x"ffdd",x"0042",x"0109",
x"01aa",x"0184",x"0124",x"013d",x"0216",x"0399",x"0504",x"05b3",
x"0568",x"041a",x"0265",x"00cf",x"ffb3",x"ff2b",x"ff35",x"ff8d",
x"0004",x"0063",x"00b4",x"00e7",x"0106",x"00f0",x"00bd",x"00be",
x"00c8",x"00b0",x"0059",x"ffee",x"ff84",x"ff1d",x"feb2",x"fe16",
x"fd9d",x"fd1b",x"fca2",x"fc09",x"fb85",x"fb23",x"fa92",x"f9f4",
x"f963",x"f8de",x"f855",x"f79b",x"f6cb",x"f620",x"f58d",x"f4f3",
x"f474",x"f403",x"f3ba",x"f326",x"f233",x"f19e",x"f0fe",x"f002",
x"eefc",x"ee13",x"ed17",x"ec1f",x"eb4c",x"eaa7",x"e98e",x"e8b8",
x"e743",x"e655",x"e551",x"e3b4",x"e25f",x"e13b",x"e06f",x"dfe3",
x"df17",x"de04",x"de06",x"dc48",x"ffa5",x"ffb5",x"fff8",x"00c8",
x"024a",x"036e",x"034a",x"0288",x"025e",x"033d",x"04cd",x"0610",
x"0641",x"052d",x"035a",x"0164",x"ffe1",x"ff2c",x"ff22",x"ff7f",
x"0008",x"007a",x"00d9",x"0126",x"013d",x"0162",x"0187",x"0199",
x"0194",x"018b",x"0142",x"00f6",x"00ad",x"0067",x"fff1",x"ff67",
x"fef6",x"fe7e",x"fde7",x"fd42",x"fce9",x"fca2",x"fc25",x"fbc3",
x"fb83",x"fb25",x"fac1",x"fa6c",x"fa06",x"f982",x"f8f3",x"f8a6",
x"f874",x"f7dd",x"f74e",x"f6f8",x"f67d",x"f61b",x"f5c0",x"f514",
x"f410",x"f356",x"f2f9",x"f283",x"f1d4",x"f137",x"f09d",x"efe2",
x"ef03",x"ee10",x"ecc8",x"eb8f",x"eaf3",x"ead5",x"ea53",x"e9f9",
x"e9a7",x"e8f0",x"e74f",x"e6ea",x"ffb7",x"ff42",x"ff18",x"ff27",
x"000f",x"01e5",x"0328",x"032a",x"02af",x"0286",x"0311",x"03aa",
x"03b4",x"02e0",x"0160",x"ffc8",x"fe7e",x"fdb0",x"fd51",x"fd6c",
x"fdd1",x"fe2d",x"fe7a",x"feba",x"fec9",x"febd",x"fef9",x"ff51",
x"ff56",x"ff08",x"fecd",x"fe96",x"fe3e",x"fde6",x"fd7a",x"fcf6",
x"fc5b",x"fbba",x"fb1b",x"fa8b",x"fa31",x"f9d6",x"f96f",x"f91a",
x"f8b2",x"f863",x"f831",x"f7ef",x"f793",x"f6fb",x"f67d",x"f61e",
x"f608",x"f5bf",x"f526",x"f48c",x"f40a",x"f3c6",x"f332",x"f2bf",
x"f234",x"f1b0",x"f13f",x"f0b3",x"f03d",x"efb6",x"ef1c",x"ee75",
x"edd0",x"ed19",x"ec5a",x"ebb4",x"eb16",x"eadd",x"eae5",x"ea31",
x"e987",x"e8ca",x"e7dc",x"e75f",x"ff8b",x"ff6e",x"ff46",x"ff11",
x"febf",x"ffb9",x"019d",x"0293",x"0279",x"01be",x"00fa",x"00b1",
x"004b",x"ffb5",x"ff0c",x"fe5b",x"fdc9",x"fd72",x"fd42",x"fd14",
x"fd2f",x"fd6e",x"fda4",x"fdf1",x"fe2a",x"fe5a",x"fe63",x"fe86",
x"feb6",x"fea5",x"fe8d",x"fe79",x"fe2c",x"fda1",x"fd21",x"fc94",

x"fc31",x"fbd5",x"fb73",x"faf0",x"faa0",x"fa7b",x"fa38",x"f9ec",
x"f99e",x"f955",x"f929",x"f8c8",x"f882",x"f83e",x"f7c4",x"f749",
x"f710",x"f6d9",x"f674",x"f628",x"f59a",x"f547",x"f4c3",x"f42f",
x"f3b5",x"f374",x"f2eb",x"f242",x"f1b5",x"f164",x"f110",x"f0c6",
x"f026",x"ef6a",x"eeec",x"ee7b",x"ee06",x"ed99",x"ed15",x"ec4c",
x"ebc5",x"eb2c",x"eadb",x"ea4b",x"ff08",x"ff8c",x"ff8d",x"ff8e",
x"ff40",x"ff58",x"0080",x"01d8",x"01fb",x"00d6",x"fef8",x"fd82",
x"fc99",x"fc30",x"fc3b",x"fc82",x"fce3",x"fd21",x"fd55",x"fd4e",
x"fd52",x"fd67",x"fd62",x"fd85",x"fddd",x"fe15",x"fe41",x"fe7c",
x"febb",x"fee9",x"ff0d",x"ff1b",x"fef8",x"fec3",x"fe5f",x"fdfb",
x"fd8a",x"fd2f",x"fd1e",x"fced",x"fcb8",x"fca9",x"fcb3",x"fc7f",
x"fc38",x"fc1b",x"fbe8",x"fb9b",x"fb98",x"fbb0",x"fb7f",x"fad8",
x"fa69",x"fa50",x"fa4e",x"fa2a",x"f9b5",x"f94d",x"f8e0",x"f8d2",
x"f8a9",x"f86b",x"f830",x"f80a",x"f78a",x"f72b",x"f6c5",x"f64c",
x"f5dd",x"f553",x"f549",x"f534",x"f4b7",x"f446",x"f3ed",x"f312",
x"f246",x"f1ee",x"f20e",x"f173",x"ff18",x"ff91",x"ffc2",x"001f",
x"0064",x"0023",x"004f",x"00f9",x"010a",x"ffc4",x"fe03",x"fc12",
x"faa6",x"fa29",x"fa46",x"fb00",x"fbc0",x"fca3",x"fd6c",x"fdd0",
x"fdc0",x"fd7b",x"fdd4",x"fdf7",x"fe8a",x"fef9",x"ff33",x"ff48",
x"ff67",x"ff8e",x"ff99",x"ffa4",x"ffad",x"ffbe",x"ff71",x"ff08",
x"fe8e",x"fe3f",x"fe3a",x"fe76",x"fe9f",x"febb",x"fedd",x"fef2",
x"ff09",x"ff00",x"fed8",x"feec",x"ff27",x"ff3e",x"ff1e",x"fedd",
x"fea2",x"fe73",x"fe52",x"fe51",x"fe4a",x"fddd",x"fd74",x"fdaf",
x"fd86",x"fd8a",x"fd66",x"fcf1",x"fc98",x"fc78",x"fc2b",x"fbdb",
x"fbe4",x"fbb5",x"fb8b",x"fb6a",x"fb28",x"faac",x"fa8c",x"fa03",
x"f931",x"f8df",x"f8dd",x"f8ac",x"ff9f",x"ff97",x"ffc4",x"0056",
x"00e7",x"00e9",x"0058",x"0081",x"0048",x"ff35",x"fd39",x"fbdb",
x"fa54",x"f97e",x"f984",x"fa0f",x"fb74",x"fcbf",x"fdef",x"fe7e",
x"fe73",x"fea3",x"fefe",x"ff58",x"ffc4",x"0016",x"0043",x"0017",
x"ffe6",x"ffdc",x"ffee",x"ffe4",x"fffc",x"0043",x"0029",x"ffc1",
x"ff6e",x"ff29",x"ff3f",x"ffa9",x"fffd",x"0027",x"0043",x"005a",
x"007d",x"0073",x"007c",x"00d4",x"00e5",x"0112",x"012c",x"00df",
x"00f6",x"0116",x"00f4",x"00fa",x"0104",x"00c5",x"0078",x"0094",
x"00bc",x"00b9",x"00d5",x"00ae",x"006d",x"0063",x"0045",x"003a",
x"007f",x"008e",x"004d",x"003c",x"0031",x"ff9b",x"ffa0",x"ff73",
x"ff0e",x"ff0d",x"ff39",x"fef7",x"ffea",x"ff4c",x"ff5d",x"ffde",
x"00d8",x"0190",x"0139",x"006a",x"ff87",x"fe9b",x"fd87",x"fc24",
x"fa97",x"f98f",x"f8e7",x"f9c2",x"faee",x"fc38",x"fda9",x"fe82",
x"fec6",x"ff17",x"ff62",x"ff92",x"0072",x"00da",x"0125",x"00d9",
x"06dd",x"001e",x"ffd8",x"ffb5",x"ffdb",x"0038",x"0039",x"ffe4",
x"ff59",x"ffc5",x"ffd1",x"0000",x"0072",x"003e",x"00d7",x"0114",
x"0137",x"0111",x"00fe",x"014a",x"0185",x"01a5",x"01bf",x"01a7",
x"01b9",x"01bf",x"01b2",x"0182",x"0141",x"014c",x"0107",x"0119",
x"0193",x"0181",x"0188",x"0162",x"0138",x"0145",x"0134",x"0133",
x"0197",x"01bf",x"018b",x"0181",x"01db",x"016c",x"0167",x"019e",
x"0185",x"0161",x"0184",x"0151",x"ffc1",x"ff26",x"fee4",x"ff78",

```
    x"004e",x"014a",x"0178",x"00b0",x"ff83",x"fe6a",x"fda3",x"fcd9",
    x"fbcd",x"fac9",x"fa12",x"fa0a",x"faba",x"fba4",x"fcf2",x"fe1d",
    x"feb2",x"feff",x"ff88",x"0056",x"00dd",x"0141",x"012f",x"00cb",
    x"0060",x"fff0",x"ff5b",x"ff01",x"ff10",x"feee",x"ff3a",x"ff7e",
    x"ff8c",x"ffd6",x"fff8",x"0048",x"00a4",x"010a",x"013e",x"019c",
    x"01ae",x"0190",x"018b",x"015f",x"014e",x"016d",x"01b7",x"01cb",
    x"01cb",x"01b2",x"01b1",x"018c",x"0148",x"0175",x"018e",x"018f",
    x"01e2",x"0219",x"022e",x"0229",x"01f0",x"01e3",x"0240",x"0269",
    x"020f",x"021a",x"01f4",x"01c7",x"01d8",x"01b0",x"01e0",x"0218",
    x"024f",x"0231",x"0217",x"01de",x"0007",x"fefe",x"feeb",x"ff27",
    x"ffca",x"00b2",x"0147",x"00e2",x"ffaa",x"fe6f",x"fda8",x"fd67",
    x"fcfe",x"fbcc",x"fa80",x"f9d4",x"fa37",x"fb28",x"fc31",x"fd4a",
    x"fe26",x"fead",x"ff54",x"0033",x"00ea",x"0181",x"0163",x"00fa",
    x"003f",x"ff9b",x"fec0",x"fe37",x"fdec",x"fde6",x"fe10",x"feae",
    x"fef2",x"ff82",x"ffdc",x"002d",x"00a1",x"0114",x"013d",x"0161",
    x"0132",x"012c",x"012d",x"0113",x"00f7",x"00ef",x"011b",x"0151",
    x"0135",x"00e8",x"00d2",x"00ae",x"00d6",x"00fa",x"00a3",x"00bc",
    x"00d9",x"0119",x"0146",x"0178");
constant WeightROM_01  : ROM_TYPE :=(x"f801",x"002b",x"010b",x"009e",
    x"00ca",x"ffd3",x"ff80",x"005c",x"03d3",x"009e",x"ff8f",x"fec0",
    x"0123",x"02f5",x"ff82",x"0725",x"fefd",x"0080",x"ffc7",x"0030",
    x"fd54",x"019e",x"05e8",x"ff5f",x"ffdb",x"fe8e",x"fdfe",x"0039",

                !!!CODE REMOVED!!!

    x"00e1",x"ffef",x"ff2d",x"fe81",x"fe12",x"fd3b",x"fd16",x"fd9a",
    x"fd0a",x"fc9c",x"fc9b",x"fd09",x"fd42",x"fc74",x"fc05",x"fbd1",
    x"fb77",x"fa94",x"f99f",x"f947",x"f917",x"f872",x"f76b",x"f84d",
    x"f951",x"f9f8",x"f9b4",x"f802",x"f7ad",x"f771",x"f4b1",x"f534",
    x"f5ad",x"f578",x"f54a",x"f4b1");
constant WeightROM_02  : ROM_TYPE :=(x"0008",x"0003",x"0006",x"0002",
    x"0000",x"0000",x"0000",x"0001",x"fffb",x"ffff",x"0001",x"ffed",
    x"0006",x"fff8",x"fffd",x"fffe",x"0003",x"fffe",x"0000",x"ffff",
    x"fffc",x"0000",x"0006",x"0002",x"fff6",x"fffd",x"0003",x"0000",
    x"0001",x"fffc",x"fffa",x"0006",x"fffe",x"0000",x"ffec",x"ffff",

                !!!CODE REMOVED!!!

    x"0167",x"01aa",x"0166",x"0103",x"0068",x"ffc0",x"ff2f",x"ffa6",
    x"fe34",x"fdf5",x"fe75",x"ffd1",x"0086",x"00cb",x"ff8a",x"ff66",
    x"ff2a",x"feb8",x"ff67",x"fffe",x"0068",x"00d4",x"00bf",x"00e1",
    x"00d7",x"0136",x"01a1",x"0196",x"0156",x"00ae",x"0028",x"ffdf",
    x"ff7d",x"ff0d",x"fc8f",x"fe83",x"ff2e",x"ffc8",x"00aa",x"00e8",
    x"00a3",x"00ef",x"00bf",x"0114");
```

```
        !!!CODE REMOVED!!!

  constant WeightROM_105  : ROM_TYPE := (x"fa97",x"0062",x"fb36",x"006a",
      x"fbc8",x"00bb",x"fbbb",x"00eb",x"fb3b",x"00da",x"fad2",x"00d2",
      x"fa97",x"00c2",x"fb15",x"00ce",x"fbfc",x"0124",x"fc3c",x"0163",
      x"fcdc",x"0184",x"fd42",x"0164",x"fd6c",x"016a",x"fe82",x"0141",
      x"fe88",x"015d",x"fe05",x"016e",x"00b5",x"0002",x"003e",x"ffda",

              !!!CODE REMOVED!!!

      x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",
      x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",
      x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",
      x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",
      x"0000",x"0000",x"0000",x"0000");
  constant WeightROM_106  : ROM_TYPE := (x"de84",x"0000",x"de8c",x"0000",
      x"de2c",x"0000",x"ddce",x"0000",x"dd94",x"0000",x"dd92",x"0000",
      x"ddf0",x"0000",x"ddd3",x"0000",x"dcc8",x"0000",x"dc78",x"0000",
      x"dc69",x"0000",x"dbd3",x"0000",x"db49",x"0000",x"dac6",x"0000",
      x"daaa",x"0000",x"dad9",x"0000",x"f30a",x"0000",x"f249",x"0000",
      x"f10b",x"0000",x"f0c6",x"0000",x"efa9",x"0000",x"eed7",x"0000",

              !!!CODE REMOVED!!!

      x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",
      x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",
      x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",
      x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",
      x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",x"0000",
      x"0000",x"0000",x"0000",x"0000");

begin

  NodeNumberOut <= NodeNumberReg;

  process (clk,Reset)
  begin
    if(Reset = '1') then
      NodeNumberReg <= (others => '0');
    elsif(clk'event and clk = '1') then
      NodeNumberReg <= NodeNumberIn;
    end if;
  end process;

  process (clk)
  begin
```

```
    if (clk'event and clk = '1') then
      if(enable = '1') then
        WeightReg00a <= WeightROM_00(conv_integer(NodeNumberIn(9 downto 0)));
        WeightReg01a <= WeightROM_01(conv_integer(NodeNumberIn(9 downto 0)));
        WeightReg02a <= WeightROM_02(conv_integer(NodeNumberIn(9 downto 0)));
        WeightReg03a <= WeightROM_03(conv_integer(NodeNumberIn(9 downto 0)));
        WeightReg04a <= WeightROM_04(conv_integer(NodeNumberIn(9 downto 0)));
        WeightReg05a <= WeightROM_05(conv_integer(NodeNumberIn(9 downto 0)));
        WeightReg06a <= WeightROM_06(conv_integer(NodeNumberIn(9 downto 0)));

                        !!!CODE REMOVED!!!

        WeightReg64b <= WeightROM_103(2*conv_integer(NodeNumberIn(8 downto
0)));
        WeightReg65b <= WeightROM_103(1+2*conv_integer(NodeNumberIn(8
downto 0)));
        WeightReg66b <= WeightROM_104(2*conv_integer(NodeNumberIn(8 downto
0)));
        WeightReg67b <= WeightROM_104(1+2*conv_integer(NodeNumberIn(8
downto 0)));
        WeightReg68b <= WeightROM_105(2*conv_integer(NodeNumberIn(8 downto
0)));
        WeightReg69b <= WeightROM_105(1+2*conv_integer(NodeNumberIn(8
downto 0)));
        WeightReg70b <= WeightROM_106(2*conv_integer(NodeNumberIn(8 downto
0)));
      end if;
    end if;
  end process;

  Weight00 <= WeightReg00a when NodeNumberReg(10) = '0' else WeightReg00b;
  Weight01 <= WeightReg01a when NodeNumberReg(10) = '0' else WeightReg01b;
  Weight02 <= WeightReg02a when NodeNumberReg(10) = '0' else WeightReg02b;
  Weight03 <= WeightReg03a when NodeNumberReg(10) = '0' else WeightReg03b;
  Weight04 <= WeightReg04a when NodeNumberReg(10) = '0' else WeightReg04b;

  !!!CODE REMOVED!!!

  Weight67 <= WeightReg67a when NodeNumberReg(10) = '0' else WeightReg67b;
  Weight68 <= WeightReg68a when NodeNumberReg(10) = '0' else WeightReg68b;
  Weight69 <= WeightReg69a when NodeNumberReg(10) = '0' else WeightReg69b;
  Weight70 <= WeightReg70a when NodeNumberReg(10) = '0' else WeightReg70b;

end syn;
```

APPENDIX B

Single Board XUP-V2P EDK Configuration Files

system.mhs

```
##############################################################################
# Created by Base System Builder Wizard for Xilinx EDK 8.1.02 Build EDK_I.20.4
# Thu Aug 09 11:05:37 2007
# Target Board:  Xilinx XUP Virtex-II Pro Development System Rev C
# Family:        virtex2p
# Device:        xc2vp30
# Package:       ff896
# Speed Grade:          -7
# Processor: PPC 405
# Processor clock frequency: 100.000000 MHz
# Bus clock frequency: 100.000000 MHz
# Debug interface: FPGA JTAG
# On Chip Memory :  16 KB
# Total Off Chip Memory : 256 MB
# - DDR_SDRAM_32Mx64 Single Rank = 256 MB
#
##############################################################################


 PARAMETER VERSION = 2.1.0


 PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX, DIR = I
 PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX, DIR = O
 PORT                fpga_0_SysACE_CompactFlash_SysACE_CLK_pin                =
fpga_0_SysACE_CompactFlash_SysACE_CLK, DIR = I
 PORT                fpga_0_SysACE_CompactFlash_SysACE_MPA_pin                =
fpga_0_SysACE_CompactFlash_SysACE_MPA, DIR = O, VEC = [6:0]
 PORT                fpga_0_SysACE_CompactFlash_SysACE_MPD_pin                =
fpga_0_SysACE_CompactFlash_SysACE_MPD, DIR = IO, VEC = [15:0]
 PORT                fpga_0_SysACE_CompactFlash_SysACE_CEN_pin                =
fpga_0_SysACE_CompactFlash_SysACE_CEN, DIR = O
 PORT                fpga_0_SysACE_CompactFlash_SysACE_OEN_pin                =
fpga_0_SysACE_CompactFlash_SysACE_OEN, DIR = O
 PORT                fpga_0_SysACE_CompactFlash_SysACE_WEN_pin                =
fpga_0_SysACE_CompactFlash_SysACE_WEN, DIR = O
```

PORT fpga_0_SysACE_CompactFlash_SysACE_MPIRQ_pin = fpga_0_SysACE_CompactFlash_SysACE_MPIRQ, DIR = I
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clk_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clk, DIR = O, VEC = [0:2]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clkn_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clkn, DIR = O, VEC = [0:2]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Addr_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Addr, DIR = O, VEC = [0:12]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_BankAddr_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_BankAddr, DIR = O, VEC = [0:1]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CASn_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CASn, DIR = O
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_RASn_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_RASn, DIR = O
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_WEn_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_WEn, DIR = O
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DM_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DM, DIR = O, VEC = [0:7]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQS_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQS, DIR = IO, VEC = [0:7]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQ_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQ, DIR = IO, VEC = [0:63]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CKE_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CKE, DIR = O
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CSn_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CSn, DIR = O
PORT fpga_0_net_gnd_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_1_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_2_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_3_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_4_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_5_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_6_pin = net_gnd, DIR = O
PORT fpga_0_DDR_CLK_FB = ddr_feedback_s, DIR = I, SIGIS = DCMCLK
PORT fpga_0_DDR_CLK_FB_OUT = ddr_clk_feedback_out_s, DIR = O
PORT sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = DCMCLK
PORT sys_rst_pin = sys_rst_s, DIR = I
PORT LED0 = PSO_NN_Calc1_0_PSO_Running, DIR = O

```
BEGIN ppc405
 PARAMETER INSTANCE = ppc405_0
 PARAMETER HW_VER = 2.00.c
 BUS_INTERFACE JTAGPPC = jtagppc_0_0
 BUS_INTERFACE IPLB = plb
 BUS_INTERFACE DPLB = plb
 PORT PLBCLK = sys_clk_s
 PORT C405RSTCHIPRESETREQ = C405RSTCHIPRESETREQ
 PORT C405RSTCORERESETREQ = C405RSTCORERESETREQ
 PORT C405RSTSYSRESETREQ = C405RSTSYSRESETREQ
 PORT RSTC405RESETCHIP = RSTC405RESETCHIP
 PORT RSTC405RESETCORE = RSTC405RESETCORE
 PORT RSTC405RESETSYS = RSTC405RESETSYS
 PORT EICC405EXTINPUTIRQ = EICC405EXTINPUTIRQ
 PORT CPMC405CLOCK = sys_clk_s
END

BEGIN ppc405
 PARAMETER INSTANCE = ppc405_1
 PARAMETER HW_VER = 2.00.c
 BUS_INTERFACE JTAGPPC = jtagppc_0_1
END

BEGIN jtagppc_cntlr
 PARAMETER INSTANCE = jtagppc_0
 PARAMETER HW_VER = 2.00.a
 BUS_INTERFACE JTAGPPC0 = jtagppc_0_0
 BUS_INTERFACE JTAGPPC1 = jtagppc_0_1
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = reset_block
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT Ext_Reset_In = sys_rst_s
 PORT Slowest_sync_clk = sys_clk_s
 PORT Chip_Reset_Req = C405RSTCHIPRESETREQ
 PORT Core_Reset_Req = C405RSTCORERESETREQ
 PORT System_Reset_Req = C405RSTSYSRESETREQ
 PORT Rstc405resetchip = RSTC405RESETCHIP
 PORT Rstc405resetcore = RSTC405RESETCORE
 PORT Rstc405resetsys = RSTC405RESETSYS
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT Dcm_locked = dcm_1_lock
END
```

```
BEGIN plb_v34
 PARAMETER INSTANCE = plb
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT SYS_Rst = sys_bus_reset
 PORT PLB_Clk = sys_clk_s
END

BEGIN opb_v20
 PARAMETER INSTANCE = opb
 PARAMETER HW_VER = 1.10.c
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT SYS_Rst = sys_bus_reset
 PORT OPB_Clk = sys_clk_s
END

BEGIN plb2opb_bridge
 PARAMETER INSTANCE = plb2opb
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_NUM_ADDR_RNG = 1
 PARAMETER C_RNG0_BASEADDR = 0x40000000
 PARAMETER C_RNG0_HIGHADDR = 0x7fffffff
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE MOPB = opb
 PORT PLB_Clk = sys_clk_s
 PORT OPB_Clk = sys_clk_s
END

BEGIN PSO_NN_Calc1
 PARAMETER INSTANCE = PSO_NN_Calc1_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x50000000
 PARAMETER C_HIGHADDR = 0x5000ffff
 PARAMETER C_AR0_BASEADDR = 0x51000000
 PARAMETER C_AR0_HIGHADDR = 0x5100ffff
 PARAMETER C_AR1_BASEADDR = 0x52000000
 PARAMETER C_AR1_HIGHADDR = 0x5200ffff
 PARAMETER C_AR2_BASEADDR = 0x53000000
 PARAMETER C_AR2_HIGHADDR = 0x5300ffff
 PARAMETER C_AR3_BASEADDR = 0x54000000
 PARAMETER C_AR3_HIGHADDR = 0x5400ffff
 PARAMETER C_AR4_BASEADDR = 0x55000000
 PARAMETER C_AR4_HIGHADDR = 0x5500ffff
 PARAMETER C_AR5_BASEADDR = 0x56000000
```

```
 PARAMETER C_AR5_HIGHADDR = 0x5600ffff
 PARAMETER C_AR6_BASEADDR = 0x57000000
 PARAMETER C_AR6_HIGHADDR = 0x5700ffff
 PARAMETER C_AR7_BASEADDR = 0x58000000
 PARAMETER C_AR7_HIGHADDR = 0x5800ffff
 BUS_INTERFACE SOPB = opb
 PORT PSO_Running = PSO_NN_Calc1_0_PSO_Running
END

BEGIN opb_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_ODD_PARITY = 0
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_CLK_FREQ = 100000000
 PARAMETER C_BASEADDR = 0x40600000
 PARAMETER C_HIGHADDR = 0x4060ffff
 BUS_INTERFACE SOPB = opb
 PORT OPB_Clk = sys_clk_s
 PORT Interrupt = RS232_Uart_1_Interrupt
 PORT RX = fpga_0_RS232_Uart_1_RX
 PORT TX = fpga_0_RS232_Uart_1_TX
END

BEGIN opb_sysace
 PARAMETER INSTANCE = SysACE_CompactFlash
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_MEM_WIDTH = 16
 PARAMETER C_BASEADDR = 0x41800000
 PARAMETER C_HIGHADDR = 0x4180ffff
 BUS_INTERFACE SOPB = opb
 PORT OPB_Clk = sys_clk_s
 PORT SysACE_CLK = fpga_0_SysACE_CompactFlash_SysACE_CLK
 PORT SysACE_MPA = fpga_0_SysACE_CompactFlash_SysACE_MPA
 PORT SysACE_MPD = fpga_0_SysACE_CompactFlash_SysACE_MPD
 PORT SysACE_CEN = fpga_0_SysACE_CompactFlash_SysACE_CEN
 PORT SysACE_OEN = fpga_0_SysACE_CompactFlash_SysACE_OEN
 PORT SysACE_WEN = fpga_0_SysACE_CompactFlash_SysACE_WEN
 PORT SysACE_MPIRQ = fpga_0_SysACE_CompactFlash_SysACE_MPIRQ
END
BEGIN plb_ddr
 PARAMETER INSTANCE = DDR_256MB_32MX64_rank1_row13_col10_cl2_5
 PARAMETER HW_VER = 1.11.a
 PARAMETER C_PLB_CLK_PERIOD_PS = 10000
```

```
 PARAMETER C_NUM_BANKS_MEM = 1
 PARAMETER C_NUM_CLK_PAIRS = 4
 PARAMETER C_INCLUDE_BURST_CACHELN_SUPPORT = 1
 PARAMETER C_REG_DIMM = 0
 PARAMETER C_DDR_TMRD = 20000
 PARAMETER C_DDR_TWR = 20000
 PARAMETER C_DDR_TRAS = 60000
 PARAMETER C_DDR_TRC = 90000
 PARAMETER C_DDR_TRFC = 100000
 PARAMETER C_DDR_TRCD = 30000
 PARAMETER C_DDR_TRRD = 20000
 PARAMETER C_DDR_TRP = 30000
 PARAMETER C_DDR_TREFC = 70300000
 PARAMETER C_DDR_AWIDTH = 13
 PARAMETER C_DDR_COL_AWIDTH = 10
 PARAMETER C_DDR_BANK_AWIDTH = 2
 PARAMETER C_DDR_DWIDTH = 64
 PARAMETER C_MEM0_BASEADDR = 0x00000000
 PARAMETER C_MEM0_HIGHADDR = 0x0fffffff
 BUS_INTERFACE SPLB = plb
 PORT PLB_Clk = sys_clk_s
 PORT                                DDR_Addr                                =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Addr
 PORT                              DDR_BankAddr                              =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_BankAddr
 PORT                               DDR_CASn                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CASn
 PORT                                DDR_CKE                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CKE
 PORT                                DDR_CSn                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CSn
 PORT                               DDR_RASn                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_RASn
 PORT                                DDR_WEn                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_WEn
 PORT                                DDR_DM                                  =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DM
 PORT                                DDR_DQS                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQS
 PORT                                DDR_DQ                                  =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQ
 PORT                                DDR_Clk                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clk                     &
ddr_clk_feedback_out_s
 PORT                                DDR_Clkn                                =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clkn & 0b0
```

```
 PORT Clk90_in = clk_90_s
 PORT Clk90_in_n = clk_90_n_s
 PORT PLB_Clk_n = sys_clk_n_s
 PORT DDR_Clk90_in = ddr_clk_90_s
 PORT DDR_Clk90_in_n = ddr_clk_90_n_s
END

BEGIN plb_bram_if_cntlr
 PARAMETER INSTANCE = plb_bram_if_cntlr_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER c_plb_clk_period_ps = 10000
 PARAMETER c_baseaddr = 0xffffc000
 PARAMETER c_highaddr = 0xffffffff
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE PORTA = plb_bram_if_cntlr_1_port
 PORT PLB_Clk = sys_clk_s
END

BEGIN bram_block
 PARAMETER INSTANCE = plb_bram_if_cntlr_1_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = plb_bram_if_cntlr_1_port
END

BEGIN opb_intc
 PARAMETER INSTANCE = opb_intc_0
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_BASEADDR = 0x41200000
 PARAMETER C_HIGHADDR = 0x4120ffff
 BUS_INTERFACE SOPB = opb
 PORT Irq = EICC405EXTINPUTIRQ
 PORT Intr = RS232_Uart_1_Interrupt
END

BEGIN util_vector_logic
 PARAMETER INSTANCE = sysclk_inv
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_SIZE = 1
 PARAMETER C_OPERATION = not
 PORT Op1 = sys_clk_s
 PORT Res = sys_clk_n_s
END

BEGIN util_vector_logic
 PARAMETER INSTANCE = clk90_inv
 PARAMETER HW_VER = 1.00.a
```

```
 PARAMETER C_SIZE = 1
 PARAMETER C_OPERATION = not
 PORT Op1 = clk_90_s
 PORT Res = clk_90_n_s
END

BEGIN util_vector_logic
 PARAMETER INSTANCE = ddr_clk90_inv
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_SIZE = 1
 PARAMETER C_OPERATION = not
 PORT Op1 = ddr_clk_90_s
 PORT Res = ddr_clk_90_n_s
END

BEGIN dcm_module
 PARAMETER INSTANCE = dcm_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_CLK0_BUF = TRUE
 PARAMETER C_CLK90_BUF = TRUE
 PARAMETER C_CLKIN_PERIOD = 10.000000
 PARAMETER C_CLK_FEEDBACK = 1X
 PARAMETER C_DLL_FREQUENCY_MODE = LOW
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT CLKIN = dcm_clk_s
 PORT CLK0 = sys_clk_s
 PORT CLK90 = clk_90_s
 PORT CLKFB = sys_clk_s
 PORT RST = net_gnd
 PORT LOCKED = dcm_0_lock
END

BEGIN dcm_module
 PARAMETER INSTANCE = dcm_1
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_CLK0_BUF = TRUE
 PARAMETER C_CLK90_BUF = TRUE
 PARAMETER C_CLKIN_PERIOD = 10.000000
 PARAMETER C_CLK_FEEDBACK = 1X
 PARAMETER C_DLL_FREQUENCY_MODE = LOW
 PARAMETER C_PHASE_SHIFT = 60
 PARAMETER C_CLKOUT_PHASE_SHIFT = FIXED
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT CLKIN = ddr_feedback_s
 PORT CLK90 = ddr_clk_90_s
 PORT CLK0 = dcm_1_FB
```

```
 PORT CLKFB = dcm_1_FB
 PORT RST = dcm_0_lock
 PORT LOCKED = dcm_1_lock
END

BEGIN opb_timer
 PARAMETER INSTANCE = opb_timer_0
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x60000000
 PARAMETER C_HIGHADDR = 0x6000ffff
 BUS_INTERFACE SOPB = opb
END
```

system.mss

```
 PARAMETER VERSION = 2.2.0


BEGIN OS
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_0
 PARAMETER STDIN = RS232_Uart_1
 PARAMETER STDOUT = RS232_Uart_1
END

BEGIN OS
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_1
END


BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu_ppc405
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc405_0
 PARAMETER COMPILER = powerpc-eabi-gcc
 PARAMETER ARCHIVER = powerpc-eabi-ar
 PARAMETER CORE_CLOCK_FREQ_HZ = 100000000
END

BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu_ppc405
```

```
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc405_1
 PARAMETER COMPILER = powerpc-eabi-gcc
 PARAMETER ARCHIVER = powerpc-eabi-ar
END


BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = jtagppc_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = plbarb
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = plb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = opbarb
 PARAMETER DRIVER_VER = 1.02.a
 PARAMETER HW_INSTANCE = opb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = plb2opb
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = plb2opb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = RS232_Uart_1
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = sysace
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = SysACE_CompactFlash
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = ddr
 PARAMETER DRIVER_VER = 1.00.b
```

```
 PARAMETER HW_INSTANCE = DDR_256MB_32MX64_rank1_row13_col10_cl2_5
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = plb_bram_if_cntlr_1
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = intc
 PARAMETER DRIVER_VER = 1.00.c
 PARAMETER HW_INSTANCE = opb_intc_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = PSO_NN_Calc1
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = PSO_NN_Calc1_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = tmrctr
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER HW_INSTANCE = opb_timer_0
END


BEGIN LIBRARY
 PARAMETER LIBRARY_NAME = xilfatfs
 PARAMETER LIBRARY_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_0
 PARAMETER CONFIG_BUFCACHE_SIZE = 20480
 PARAMETER CONFIG_DIR_SUPPORT = true
 PARAMETER CONFIG_WRITE = true
END
```

APPENDIX C

Three Board XUP-V2P Design VHDL Files

user_logic.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library UNISIM;
use UNISIM.all;

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_AWIDTH                -- User logic address bus width
--   C_DWIDTH                -- User logic data bus width
--   C_MAX_AR_DWIDTH         -- User logic max data bus width of address ranges
--   C_NUM_ADDR_RNG          -- User logic number of address ranges to be decoded
--   C_NUM_CE                -- User logic chip enable bus width
--   C_IP_INTR_NUM           -- User logic number of interrupt event
--
-- Definition of Ports:
--   Bus2IP_Clk              -- Bus to IP clock
--   Bus2IP_Reset            -- Bus to IP reset
--   IP2Bus_IntrEvent        -- IP to Bus interrupt event
--   Bus2IP_Addr             -- Bus to IP address bus
--   Bus2IP_Data             -- Bus to IP data bus for user logic
--   Bus2IP_BE               -- Bus to IP byte enables for user logic
--   Bus2IP_RNW              -- Bus to IP read/not write
--   Bus2IP_RdCE             -- Bus to IP read chip enable for user logic
--   Bus2IP_WrCE             -- Bus to IP write chip enable for user logic
--   IP2Bus_Data             -- IP to Bus data bus for user logic
--   IP2Bus_Ack              -- IP to Bus acknowledgement
--   IP2Bus_Retry            -- IP to Bus retry response
--   IP2Bus_Error            -- IP to Bus error response
--   IP2Bus_ToutSup          -- IP to Bus timeout suppress
--   Bus2IP_ArData           -- Bus to IP data bus for address ranges
--   Bus2IP_ArBE             -- Bus to IP byte enables for address ranges
```

```
--   Bus2IP_ArCS              -- Bus to IP chip select for address ranges
--   IP2Bus_ArData            -- IP to Bus data bus for address ranges
------------------------------------------------------------------------------

entity user_logic is
  generic
  ( C_AWIDTH                  : integer           := 32;
    C_DWIDTH                  : integer           := 32;
    C_MAX_AR_DWIDTH           : integer           := 32;
    C_NUM_ADDR_RNG            : integer           := 8;
    C_NUM_CE                  : integer           := 8;
    C_IP_INTR_NUM             : integer           := 1);
  port
  ( Bus2IP_Clk                : in  std_logic;
    Bus2IP_Reset              : in  std_logic;
    IP2Bus_IntrEvent          : out std_logic_vector(0 to C_IP_INTR_NUM-1);
    Bus2IP_Addr               : in  std_logic_vector(0 to C_AWIDTH-1);
    Bus2IP_Data               : in  std_logic_vector(0 to C_DWIDTH-1);
    Bus2IP_BE                 : in  std_logic_vector(0 to C_DWIDTH/8-1);
    Bus2IP_RNW                : in  std_logic;
    Bus2IP_RdCE               : in  std_logic_vector(0 to C_NUM_CE-1);
    Bus2IP_WrCE               : in  std_logic_vector(0 to C_NUM_CE-1);
    IP2Bus_Data               : out std_logic_vector(0 to C_DWIDTH-1);
    IP2Bus_Ack                : out std_logic;
    IP2Bus_Retry              : out std_logic;
    IP2Bus_Error              : out std_logic;
    IP2Bus_ToutSup            : out std_logic;
    Bus2IP_ArData             : in  std_logic_vector(0 to C_MAX_AR_DWIDTH-1);
    Bus2IP_ArBE               : in  std_logic_vector(0 to C_MAX_AR_DWIDTH/8-1);
    Bus2IP_ArCS               : in  std_logic_vector(0 to C_NUM_ADDR_RNG-1);
    IP2Bus_ArData             : out std_logic_vector(0 to C_MAX_AR_DWIDTH-1);
          TXP1                                                          : out
std_logic;
          TXN1                                                          : out
std_logic;
          RXP1                                                          : in
std_logic;
          RXN1                                                          : in
std_logic;
          TXP2                                                          : out
std_logic;
          TXN2                                                          : out
std_logic;
          RXP2                                                          : in
std_logic;
```

```
        RXN2                                                    : in
std_logic;
   NN_Running_n                                         : out std_logic;
   PSO_Running_n                                        : out std_logic;
   Aurora_Up1_n                                         : out std_logic;
   Aurora_Up2_n                                         : out std_logic);
end entity user_logic;

------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------

architecture IMP of user_logic is

        component Sonar_NN1 is
                Port(   clk         : in  STD_LOGIC;
                        Start       : in  STD_LOGIC;
                        Reset       : in  STD_LOGIC;
                        Done                    : out STD_LOGIC;
                        OutputWrEn          : out STD_LOGIC;
                        OutNodeNum          : out
STD_LOGIC_VECTOR(11 downto 0);
                        OutputNode    : out STD_LOGIC_VECTOR(15
downto 0);
                        Input00     : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input01     : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input02     : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input03     : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input04     : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input05     : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input06     : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input07     : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input08     : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input09     : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input10     : in  STD_LOGIC_VECTOR(15
downto 0);
```

```
                                    Input11      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input12      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input13      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input14      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input15      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input16      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input17      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input18      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input19      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input20      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input21      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input22      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input23      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input24      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input25      : in  STD_LOGIC_VECTOR(15
downto 0);
                                    Input26      : in  STD_LOGIC_VECTOR(15
downto 0));
        end component;


        component PSO_Update is
    Port(clk                        : in  std_logic;
        Start                               : in  std_logic;
        Reset                               : in  std_logic;
                    Done                                : out std_logic;
                    Update_Wr_En                        : out std_logic;
                    input_index                 : out std_logic_vector(3 downto 0);
                    output_index                : out std_logic_vector(3 downto 0);
                    constant_global     : in  std_logic_vector(15 downto 0);
                    constant_local      : in  std_logic_vector(15 downto 0);
                    current_position1   : in  std_logic_vector(15 downto 0);
```

```
                current_position2     : in  std_logic_vector(15 downto 0);
                current_velocity1     : in  std_logic_vector(15 downto 0);
                current_velocity2     : in  std_logic_vector(15 downto 0);
                local_best1                 : in  std_logic_vector(15 downto 0);
                local_best2                 : in  std_logic_vector(15 downto 0);
                global_best1                : in  std_logic_vector(15 downto 0);
                global_best2                : in  std_logic_vector(15 downto 0);
                max_velocity1               : in  std_logic_vector(15 downto 0);
                max_velocity2               : in  std_logic_vector(15 downto 0);
                max_position1               : in  std_logic_vector(15 downto 0);
                max_position2               : in  std_logic_vector(15 downto 0);
                min_position1               : in  std_logic_vector(15 downto 0);
                min_position2               : in  std_logic_vector(15 downto 0);
                next_position1              : out std_logic_vector(15 downto 0);
                next_position2              : out std_logic_vector(15 downto 0);
                next_velocity1              : out std_logic_vector(15 downto 0);
                next_velocity2              : out std_logic_vector(15 downto 0));

    end component;


    component aurora_connect16 is
generic( EXTEND_WATCHDOGS   : boolean := FALSE);
port (
                -- TX Stream Interface
    TX_D            : in  std_logic_vector(0 to 15);
    TX_SRC_RDY_N    : in  std_logic;
    TX_DST_RDY_N    : out std_logic;
                -- RX Stream Interface
    RX_D            : out std_logic_vector(0 to 15);
    RX_SRC_RDY_N    : out std_logic;
                -- Clock Correction Interface
    DO_CC           : in  std_logic;
    WARN_CC         : in  std_logic;
                -- MGT Serial I/O
    RXP             : in std_logic;
    RXN             : in std_logic;
    TXP             : out std_logic;
    TXN             : out std_logic;
                -- MGT Reference Clock Interface
    TOP_REF_CLK     : in std_logic;
                -- Error Detection Interface
    HARD_ERROR      : out std_logic;
    SOFT_ERROR      : out std_logic;
                -- Status
    CHANNEL_UP      : out std_logic;
```

```
        LANE_UP            : out std_logic;
                        -- System Interface
        DCM_NOT_LOCKED     : in std_logic;
        USER_CLK           : in std_logic;
        RESET              : in std_logic;
        POWER_DOWN         : in std_logic;
        LOOPBACK           : in std_logic_vector(1 downto 0));
  end component;


      component STANDARD_CC_MODULE
    port (   WARN_CC       : out std_logic;
         DO_CC         : out std_logic;
         DCM_NOT_LOCKED : in std_logic;
         USER_CLK      : in std_logic;
         CHANNEL_UP    : in std_logic);
  end component;


  component BUFG
     port (
          O : out std_ulogic;
          I : in  std_ulogic);
       end component;

  ----------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ----------------------------------------
  signal status_reg0               : std_logic_vector(0 to C_DWIDTH-1);
  signal status_reg1               : std_logic_vector(0 to C_DWIDTH-1);
  signal num_iterations             : std_logic_vector(0 to C_DWIDTH-1);
  signal num_agents                : std_logic_vector(0 to C_DWIDTH-1);
  signal const_CG                  : std_logic_vector(0 to C_DWIDTH-1);
  signal const_CL                  : std_logic_vector(0 to C_DWIDTH-1);
  signal slv_reg_write_select          : std_logic_vector(0 to 7);
  signal slv_reg_read_select           : std_logic_vector(0 to 7);
  signal slv_ip2bus_data             : std_logic_vector(0 to C_DWIDTH-1);
  signal slv_read_ack                : std_logic;
  signal slv_write_ack               : std_logic;
  signal board1_done                                                  : std_logic;
  signal board2_done                                                  : std_logic;
  signal board3_done                                                  : std_logic;
  signal board2_ready                                          : std_logic;
  signal board3_ready                                          : std_logic;
  signal OutNodeNum2_i                                                : std_logic_vector(11
downto 0);
```

```
  signal OutNodeNum3_i                                    : std_logic_vector(11
downto 0);

  ---------------------------------------
  -- Signals for user logic interrupt example
  ---------------------------------------
  signal interrupt                : std_logic_vector(0 to C_IP_INTR_NUM-1);

  ---------------------------------------
  -- RAM and register declarations
  ---------------------------------------
  type REG_TYPE1 is array (0 to 31) of std_logic_vector(15 downto 0);
  type REG_TYPE2 is array (0 to 37) of std_logic_vector(31 downto 0);
  type RAM_TYPE1 is array (0 to 255) of std_logic_vector(15 downto 0);
  type RAM_TYPE2 is array (0 to 2047) of std_logic_vector(31 downto 0);
  type RAM_TYPE3 is array (0 to 15) of std_logic_vector(15 downto 0);
  type RAM_TYPE4 is array (0 to 15) of std_logic_vector(31 downto 0);
  type RAM_TYPE5 is array (0 to 127) of std_logic_vector(31 downto 0);
  signal Input_Reg              : REG_TYPE1;
  signal Output_RAM1                  : RAM_TYPE1;
  signal Output_RAM2                  : RAM_TYPE1;
  signal Output_RAM3                  : RAM_TYPE1;
  signal Output_RAM4                  : RAM_TYPE1;
  signal Output_RAM5                  : RAM_TYPE1;
  signal Output_RAM6                  : RAM_TYPE1;
  signal Bitmap_Reg             : REG_TYPE2;
  signal PSO_Global_Best      : REG_TYPE1;
  signal Agent_Positions        : RAM_TYPE2;
  signal Agent_Velocities       : RAM_TYPE2;
  signal Max_Position1     : RAM_TYPE3;
  signal Min_Position1     : RAM_TYPE3;
  signal Max_Position2     : RAM_TYPE3;
  signal Min_Position2     : RAM_TYPE3;
  signal Max_Velocities               : RAM_TYPE4;
  signal PSO_Local_Best               : RAM_TYPE2;
  signal Global_Fitness               : std_logic_vector(31 downto 0);
  signal Local_Fitness          : RAM_TYPE5;
  signal Current_Fitness        : std_logic_vector(31 downto 0);
  signal Current_Fitness_Board1       : std_logic_vector(31 downto 0);
  signal Current_Fitness_Board2       : std_logic_vector(31 downto 0);
  signal Current_Fitness_Board3       : std_logic_vector(31 downto 0);

  ---------------------------------------
  -- Signals for user logic address range example
  ---------------------------------------
```

```
  type DATA_OUT_TYPE is array (0 to C_NUM_ADDR_RNG-1) of std_logic_vector(0
to C_MAX_AR_DWIDTH-1);
  signal ar_data_out              : DATA_OUT_TYPE;
  signal ar_address               : std_logic_vector(0 to 10);
  signal ar_address1              : std_logic_vector(0 to 9);
  signal ar_address2              : std_logic_vector(0 to 9);
  signal ar_address3              : std_logic_vector(0 to 9);
  signal ar_select                : std_logic_vector(0 to 7);
  signal ar_read_enable           : std_logic;
  signal ar_read_ack_dly1         : std_logic;
  signal ar_read_ack              : std_logic;
  signal ar_write_ack             : std_logic;
  signal InputWrEn                : std_logic;
  signal OutRAM1_En               : std_logic;
  signal OutRAM2_En               : std_logic;
  signal OutRAM3_En               : std_logic;
  signal OutRAM4_En               : std_logic;
  signal OutRAM5_En               : std_logic;
  signal OutRAM6_En               : std_logic;
  signal BitmapWrEn               : std_logic;
  signal PSO_GBest_RAM_En         : std_logic;
  signal Agent_Pos_RAM_En         : std_logic;
  signal Agent_Vel_RAM_En         : std_logic;
  signal MaxMin_Pos_RAM_En1       : std_logic;
  signal MaxMin_Pos_RAM_En2       : std_logic;
  signal Max_Vel_RAM_En           : std_logic;
  signal Agent_Position_Data_In                 : std_logic_vector(31 downto 0);
  signal Agent_Position_WrAddr_In       : integer;
  signal Agent_Position_RdAddr_In       : integer;
  signal Agent_Velocity_Data_In                 : std_logic_vector(31 downto 0);
  signal Agent_Velocity_WrAddr_In       : integer;
  signal Agent_Velocity_RdAddr_In       : integer;
  signal Local_Addr_In                                  : integer;
  signal Local_Addr_Out                                 : integer;
  signal RAM1_Data_Out                                  : std_logic_vector(15
downto 0);
  signal RAM2_Data_Out                                  : std_logic_vector(15
downto 0);
  signal RAM3_Data_Out                                  : std_logic_vector(15
downto 0);
  signal RAM4_Data_Out                                  : std_logic_vector(15
downto 0);
  signal RAM5_Data_Out                                  : std_logic_vector(15
downto 0);
  signal RAM6_Data_Out                                  : std_logic_vector(15
downto 0);
```

```
  ----------------------------------------
  -- Signal for PSO logic
  ----------------------------------------
  signal NN_Start                                                    : std_logic;
  signal PSO_Start                                                   : std_logic;
  signal Update_Start                                                : std_logic;
  signal Reset_i                                                     : std_logic;
  signal NN_Done                                                     : std_logic;
  signal PSO_Done                                                    : std_logic;
  signal Update_Done                                                 : std_logic;
  signal Update_Wr_En                                                : std_logic;
  signal update_cnt_in              : std_logic_vector(3 downto 0);
  signal update_cnt_out             : std_logic_vector(3 downto 0);
  signal current_position_out1      : std_logic_vector(31 downto 0);
  signal current_position_out2      : std_logic_vector(31 downto 0);
  signal current_velocity_out1      : std_logic_vector(31 downto 0);
  signal current_velocity_out2      : std_logic_vector(31 downto 0);
  signal local_best_out             : std_logic_vector(31 downto 0);
  signal global_best_out            : std_logic_vector(31 downto 0);
  signal max_velocity_out           : std_logic_vector(31 downto 0);
  signal max_position_out1_1        : std_logic_vector(15 downto 0);
  signal min_position_out1_1        : std_logic_vector(15 downto 0);
  signal max_position_out2_1        : std_logic_vector(15 downto 0);
  signal min_position_out2_1        : std_logic_vector(15 downto 0);
  signal max_position_out1_2        : std_logic_vector(15 downto 0);
  signal min_position_out1_2        : std_logic_vector(15 downto 0);
  signal max_position_out2_2        : std_logic_vector(15 downto 0);
  signal min_position_out2_2        : std_logic_vector(15 downto 0);

  signal next_position_in           : std_logic_vector(31 downto 0);
  signal next_velocity_in           : std_logic_vector(31 downto 0);

  ----------------------------------------
  -- PSO State Machine Signals
  ----------------------------------------
  type state_type_pso is
(initial,load_inputs,start_calc_nn,calc_nn,update_best,update_global_local,update_only_l
ocal,update_agent,update_count,finished1,finished2);
  signal PSO_state                                                   : state_type_pso;
  signal PSO_next_state                                              : state_type_pso;
  signal load_cnt                                                    :
std_logic_vector(3 downto 0);
  signal load_cnt_dly                                                : std_logic_vector(3 downto
0);
  signal agent_cnt                                                   : std_logic_vector(7
downto 0);
```

```
  signal iteration_cnt                                    : std_logic_vector(31
downto 0);
  signal LoadInputs                                             : std_logic;
  signal WriteGlobalBest                                  : std_logic;
  signal WriteLocalBest                                    : std_logic;
  signal PSO_Running_i                                          : std_logic;
  signal SetAgentCnt0                                     : std_logic;
  signal SetIterationCnt0                                 : std_logic;
  signal SetLoadCnt0                                       : std_logic;
  signal IncrAgentCnt                                     : std_logic;
  signal IncrLoadCnt                                            : std_logic;
  signal PSO_Start_NN                                           : std_logic;
  signal Reset_Current_Fitness                      : std_logic;
  signal Update_Rd_En               : std_logic;
  signal OutputWrEn_i                                           : std_logic;
  signal OutNodeNum_i                                           : std_logic_vector(11
downto 0);
  signal OutputNode_i                                     : std_logic_vector(15
downto 0);

  ----------------------------------------
  -- Aurora Signals Board 2
  ----------------------------------------
  signal Aurora_Clk                                             : std_logic;
  signal hard_error_i2              : std_logic;
  signal soft_error_i2              : std_logic;
  signal channel_up_i2                : std_logic;
  signal lane_up_i2                 : std_logic;
  signal warn_cc_i2                                             : std_logic;
  signal do_cc_i2                                                     : std_logic;
  signal tx_d2                                                        :
std_logic_vector(0 to 15);
  signal rx_d2                                                        :
std_logic_vector(0 to 15);
  signal tx_src_rdy_n2                                    : std_logic;
  signal tx_dst_rdy_n2                                    : std_logic;
  signal rx_src_rdy_n2                                    : std_logic;

  ----------------------------------------
  -- Aurora Signals Board 3
  ----------------------------------------
  signal hard_error_i3              : std_logic;
  signal soft_error_i3              : std_logic;
  signal channel_up_i3                : std_logic;
  signal lane_up_i3                 : std_logic;
  signal warn_cc_i3                                             : std_logic;
```

```
  signal do_cc_i3                                                    : std_logic;
  signal tx_d3                                                       :
std_logic_vector(0 to 15);
  signal rx_d3                                                       :
std_logic_vector(0 to 15);
  signal tx_src_rdy_n3                                  : std_logic;
  signal tx_dst_rdy_n3                                  : std_logic;
  signal rx_src_rdy_n3                                  : std_logic;

  ----------------------------------------
  -- TX State Machine Signals
  ----------------------------------------
  type state_type_tx is (initial,wait1,pre_xfer,xfer,start1,start2,running,wait2);
  signal tx_state2 : state_type_tx;
  signal tx_next_state2 : state_type_tx;
  signal tx_count2 : std_logic_vector(4 downto 0);
  signal rx_count2 : std_logic_vector(11 downto 0);
  signal tx_state3 : state_type_tx;
  signal tx_next_state3 : state_type_tx;
  signal tx_count3 : std_logic_vector(4 downto 0);
  signal rx_count3 : std_logic_vector(11 downto 0);


begin

        PSO_Running_n <= not PSO_Running_i;
        NN_Start <= PSO_Start_NN  when PSO_Running_i = '1' else status_reg0(31);
        PSO_Start <= status_reg0(30);
        Reset_i <= status_reg0(29) or Bus2IP_Reset;
        NN_Done <= board1_done and board2_done and board3_done;
   IP2Bus_IntrEvent(0) <= '0';
        Aurora_Up1_n <= not ((not hard_error_i2) and (not soft_error_i2) and lane_up_i2
and channel_up_i2);
        Aurora_Up2_n <= not ((not hard_error_i3) and (not soft_error_i3) and lane_up_i3
and channel_up_i3);
        NN_Running_n <= '1' when (tx_state2 = initial or tx_state2 = wait1 or tx_state2 =
wait2 or tx_state3 = initial or tx_state3 = wait1 or tx_state3 = wait2) else '0';


   slv_reg_write_select <= Bus2IP_WrCE(0 to 7);
   slv_reg_read_select  <= Bus2IP_RdCE(0 to 7);
   slv_write_ack        <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or
Bus2IP_WrCE(3) or Bus2IP_WrCE(4) or Bus2IP_WrCE(5) or Bus2IP_WrCE(6) or
Bus2IP_WrCE(7);
```

```
  slv_read_ack        <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2) or
Bus2IP_RdCE(3) or Bus2IP_RdCE(4) or Bus2IP_RdCE(5) or Bus2IP_RdCE(6) or
Bus2IP_RdCE(7);


  -- implement slave model register(s)
  SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
  begin
    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Reset = '1' then
        status_reg0 <= (others => '0');
        status_reg1 <= (others => '0');
                                  num_iterations <= x"0000000a";
        num_agents <= x"0000000a";
                    const_CG <= (others => '0');
        const_CL <= (others => '0');
      else
        case slv_reg_write_select is
          when "10000000" =>
            status_reg0 <= Bus2IP_Data;
          when "01000000" =>
            status_reg1 <= Bus2IP_Data;
                                          when "00100000" =>
            num_iterations <= Bus2IP_Data;
          when "00010000" =>
            num_agents <= Bus2IP_Data;
                        when "00001000" =>
            const_CG <= Bus2IP_Data;
          when "00000100" =>
            const_CL <= Bus2IP_Data;
          when others => null;
        end case;
      end if;
    end if;
  end process SLAVE_REG_WRITE_PROC;


  SLAVE_REG_READ_PROC : process( slv_reg_read_select, status_reg0, status_reg1,
num_iterations, num_agents, const_CL, const_CG, PSO_Done, NN_Done) is
  begin
    case slv_reg_read_select is
      when "10000000" => slv_ip2bus_data <= status_reg0;
      when "01000000" => slv_ip2bus_data <= status_reg1(0 to 29) & PSO_Done &
NN_Done;
                          when "00100000" => slv_ip2bus_data <= num_iterations;
      when "00010000" => slv_ip2bus_data <= num_agents;
```

```
      when "00001000" => slv_ip2bus_data <= const_CG;
      when "00000100" => slv_ip2bus_data <= const_CL;
      when "00000010" => slv_ip2bus_data <= Current_Fitness;
      when "00000001" => slv_ip2bus_data <= Global_Fitness;
      when others => slv_ip2bus_data <= (others => '0');
    end case;
  end process SLAVE_REG_READ_PROC;


  --generates the read acknowledge 1 clock after read enable is presented
  BRAM_RD_ACK_PROC : process( Bus2IP_Clk ) is
  begin
    if (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
      if(Bus2IP_Reset = '1') then
        ar_read_ack_dly1 <= '0';
      else
        ar_read_ack_dly1 <= ar_read_enable;
      end if;
    end if;
  end process BRAM_RD_ACK_PROC;


        ar_select      <= Bus2IP_ArCS;
  ar_read_enable <= ( Bus2IP_ArCS(0) or Bus2IP_ArCS(1) or Bus2IP_ArCS(2) or
Bus2IP_ArCS(3) or Bus2IP_ArCS(4) or Bus2IP_ArCS(5) or Bus2IP_ArCS(6) or
Bus2IP_ArCS(7) ) and Bus2IP_RNW;
  ar_read_ack    <= ar_read_ack_dly1;
  ar_write_ack   <= ( Bus2IP_ArCS(0) or Bus2IP_ArCS(1) or Bus2IP_ArCS(2) or
Bus2IP_ArCS(3) or Bus2IP_ArCS(4) or Bus2IP_ArCS(5) or Bus2IP_ArCS(6) or
Bus2IP_ArCS(7) ) and not(Bus2IP_RNW);
  ar_address     <= Bus2IP_Addr(C_AWIDTH-13 to C_AWIDTH-3);
  ar_address1    <= Bus2IP_Addr(C_AWIDTH-12 to C_AWIDTH-3);
        --472/2=236
  ar_address2    <= Bus2IP_Addr(C_AWIDTH-12 to C_AWIDTH-3) - 236 when
Bus2IP_Addr(C_AWIDTH-12 to C_AWIDTH-3) >= x"0ec" else
Bus2IP_Addr(C_AWIDTH-12 to C_AWIDTH-3);
  --(472+364)/2=418
        ar_address3    <= Bus2IP_Addr(C_AWIDTH-12 to C_AWIDTH-3) - 418 when
Bus2IP_Addr(C_AWIDTH-12 to C_AWIDTH-3) >= x"1a2" else
Bus2IP_Addr(C_AWIDTH-12 to C_AWIDTH-3);


        InputWrEn <= not(Bus2IP_RNW) and Bus2IP_ArCS(0);
        INPUTREG:process(Bus2IP_Clk,Reset_i)
        begin
                if(Reset_i = '1') then
```

```
                        for index in 0 to 31 loop
                                Input_Reg(index) <= (others => '0');
                        end loop;
                elsif(rising_edge(Bus2IP_Clk)) then
                        if(LoadInputs = '1') then
                                Input_Reg(0+2*conv_integer(load_cnt_dly))
<=current_position_out2(31 downto 16);
                                Input_Reg(1+2*conv_integer(load_cnt_dly))
<=current_position_out2(15 downto 0);
                        elsif(InputWrEn  = '1') then
                                Input_Reg(0+2*conv_integer(ar_address(7 to 10))) <=
Bus2IP_ArData(0 to 15);
                                Input_Reg(1+2*conv_integer(ar_address(7 to 10))) <=
Bus2IP_ArData(16 to 31);
                        end if;
                end if;
        end process;


  --Since OPB data width is 32bits, I have made 2 16bit width distributed RAMS.
  --Other approaches resulted in registers, not RAMS, and caused the
  --tools to use up about 70% of the remaining slices.
        OutRAM1_En <=OutputWrEn_i and not(OutNodeNum_i(0));
        OutRAM2_En <=OutputWrEn_i and (OutNodeNum_i(0));
        OutRAM3_En <=(not rx_src_rdy_n2) and not(rx_count2(0));
        OutRAM4_En <=(not rx_src_rdy_n2) and (rx_count2(0));
        OutRAM5_En <=(not rx_src_rdy_n3) and not(rx_count3(0));
        OutRAM6_En <=(not rx_src_rdy_n3) and (rx_count3(0));

        OUTPUTRAM1:process(Bus2IP_Clk)
        begin
                if(Bus2IP_Clk'event and Bus2IP_Clk = '1') then
                        if(OutRAM1_En='1') then
                                Output_RAM1(conv_integer(OutNodeNum_i(8 downto
1))) <= OutputNode_i;
                        end if;
                        RAM1_Data_Out <=
Output_RAM1(CONV_INTEGER(ar_address1(2 to 9)));
                end if;
        end process;
        OUTPUTRAM2:process(Bus2IP_Clk)
        begin
                if(Bus2IP_Clk'event and Bus2IP_Clk = '1') then
                        if(OutRAM2_En='1') then
                                Output_RAM2(conv_integer(OutNodeNum_i(8 downto
1))) <= OutputNode_i;
```

```
                        end if;
                        RAM2_Data_Out <=
Output_RAM2(CONV_INTEGER(ar_address1(2 to 9)));
                end if;
        end process;
        OUTPUTRAM3:process(Bus2IP_Clk)
        begin
                if(Bus2IP_Clk'event and Bus2IP_Clk = '1') then
                        if(OutRAM3_En='1') then
                                Output_RAM3(conv_integer(rx_count2(8 downto 1))) <=
rx_d2;
                        end if;
                        RAM3_Data_Out <=
Output_RAM3(CONV_INTEGER(ar_address2(2 to 9)));
                end if;
        end process;
        OUTPUTRAM4:process(Bus2IP_Clk)
        begin
                if(Bus2IP_Clk'event and Bus2IP_Clk = '1') then
                        if(OutRAM4_En='1') then
                                Output_RAM4(conv_integer(rx_count2(8 downto 1))) <=
rx_d2;
                        end if;
                        RAM4_Data_Out <=
Output_RAM4(CONV_INTEGER(ar_address2(2 to 9)));
                end if;
        end process;
        OUTPUTRAM5:process(Bus2IP_Clk)
        begin
                if(Bus2IP_Clk'event and Bus2IP_Clk = '1') then
                        if(OutRAM5_En='1') then
                                Output_RAM5(conv_integer(rx_count3(8 downto 1))) <=
rx_d3;
                        end if;
                        RAM5_Data_Out <=
Output_RAM5(CONV_INTEGER(ar_address3(2 to 9)));
                end if;
        end process;
        OUTPUTRAM6:process(Bus2IP_Clk)
        begin
                if(Bus2IP_Clk'event and Bus2IP_Clk = '1') then
                        if(OutRAM6_En='1') then
                                Output_RAM6(conv_integer(rx_count3(8 downto 1))) <=
rx_d3;
                        end if;
```

```
                        RAM6_Data_Out <=
Output_RAM6(CONV_INTEGER(ar_address3(2 to 9)));
                end if;
        end process;


        BitmapWrEn <= not(Bus2IP_RNW) and Bus2IP_ArCS(2);
        BITMAPREG:process(Bus2IP_Clk,Bus2IP_Reset)
        begin
                if(Bus2IP_Reset = '1') then
                        for index in 0 to 37 loop
                                Bitmap_Reg(index) <= (others => '0');
                        end loop;
                elsif(rising_edge(Bus2IP_Clk)) then
                        if(BitmapWrEn  = '1') then
                          Bitmap_Reg(conv_integer(ar_address(5 to 10)))  <=
Bus2IP_ArData;
                        end if;
                end if;
        end process;


        PSO_GLOBAL_REG:process(Bus2IP_Clk,Reset_i)
        begin
                if(Reset_i = '1') then
                        for index in 0 to 31 loop
                                PSO_Global_Best(index) <= (others => '0');
                        end loop;
                elsif(rising_edge(Bus2IP_Clk)) then
                        if(WriteGlobalBest  = '1') then
                                for index in 0 to 31 loop
                                        PSO_Global_Best(index) <= Input_Reg(index);
                                end loop;
                        end if;
                        global_best_out(31 downto 16) <=
PSO_Global_Best(0+2*conv_integer(update_cnt_in));
                        global_best_out(15 downto 0)  <=
PSO_Global_Best(1+2*conv_integer(update_cnt_in));
                end if;
        end process;

        Local_Addr_In <= 16*conv_integer(agent_cnt)+conv_integer(load_cnt) when
WriteLocalBest = '1' else 16*conv_integer(agent_cnt)+conv_integer(update_cnt_in);
        PSO_LOCAL_RAM:process(Bus2IP_Clk,Reset_i)
        begin
                if(rising_edge(Bus2IP_Clk)) then
```

```
                        if(WriteLocalBest  = '1') then
                                PSO_Local_Best(Local_Addr_In) <=
Input_Reg(2*conv_integer(load_cnt)) & Input_Reg(2*conv_integer(load_cnt)+1);
                        end if;
                local_best_out <= PSO_Local_Best(Local_Addr_In);
                end if;
        end process;


        Agent_Pos_RAM_En <= Update_Wr_En or (not(Bus2IP_RNW) and
Bus2IP_ArCS(4));
        Agent_Vel_RAM_En <= Update_Wr_En or (not(Bus2IP_RNW) and
Bus2IP_ArCS(5));
        MaxMin_Pos_RAM_En1 <= not(Bus2IP_RNW) and Bus2IP_ArCS(6) and
not(ar_address(10));
        MaxMin_Pos_RAM_En2 <= not(Bus2IP_RNW) and Bus2IP_ArCS(6) and
ar_address(10);
        Max_Vel_RAM_En <= not(Bus2IP_RNW) and Bus2IP_ArCS(7);
        Agent_Position_Data_In <= next_position_in when Update_Wr_En = '1' else
Bus2IP_ArData;
        Agent_Velocity_Data_In <= next_velocity_in when Update_Wr_En = '1' else
Bus2IP_ArData;
        Agent_Position_WrAddr_In <=
16*conv_integer(agent_cnt)+conv_integer(update_cnt_out) when (Update_Wr_En = '1')
else conv_integer(ar_address);
        Agent_Velocity_WrAddr_In <=
16*conv_integer(agent_cnt)+conv_integer(update_cnt_out) when (Update_Wr_En = '1')
else conv_integer(ar_address);
        Agent_Position_RdAddr_In <=
16*conv_integer(agent_cnt)+conv_integer(update_cnt_in) when (Update_Rd_En = '1')
else 16*conv_integer(agent_cnt)+conv_integer(load_cnt) when LoadInputs = '1' else
conv_integer(ar_address);
        Agent_Velocity_RdAddr_In <=
16*conv_integer(agent_cnt)+conv_integer(update_cnt_in) when (Update_Rd_En = '1')
else conv_integer(ar_address);

  RAMS:process(Bus2IP_Clk)
        begin
                if(rising_edge(Bus2IP_Clk)) then

                        if(Agent_Pos_RAM_En = '1') then
                                Agent_Positions(Agent_Position_WrAddr_In) <=
Agent_Position_Data_In;
                        end if;

                        if(Agent_Vel_RAM_En = '1') then
```

```
                        Agent_Velocities(Agent_Velocity_WrAddr_In) <=
Agent_Velocity_Data_In;
                end if;

                if(MaxMin_Pos_RAM_En1 = '1') then
                        Max_Position1(conv_integer(ar_address(6 to 9))) <=
Bus2IP_ArData(0 to 15);
                        Min_Position1(conv_integer(ar_address(6 to 9))) <=
Bus2IP_ArData(16 to 31);
                end if;

                if(MaxMin_Pos_RAM_En2 = '1') then
                        Max_Position2(conv_integer(ar_address(6 to 9))) <=
Bus2IP_ArData(0 to 15);
                        Min_Position2(conv_integer(ar_address(6 to 9))) <=
Bus2IP_ArData(16 to 31);
                end if;

                if(Max_Vel_RAM_En = '1') then
                        Max_Velocities(conv_integer(ar_address(7 to 10))) <=
Bus2IP_ArData;
                end if;

                current_position_out1 <=
Agent_Positions(Agent_Position_WrAddr_In);
                current_position_out2 <=
Agent_Positions(Agent_Position_RdAddr_In);

                current_velocity_out1 <=
Agent_Velocities(Agent_Velocity_WrAddr_In);
                current_velocity_out2 <=
Agent_Velocities(Agent_Velocity_RdAddr_In);

                max_position_out1_1 <=
Max_Position1(conv_integer(update_cnt_in));
                min_position_out1_1 <=
Min_Position1(conv_integer(update_cnt_in));
                max_position_out2_1 <=
Max_Position2(conv_integer(update_cnt_in));
                min_position_out2_1 <=
Min_Position2(conv_integer(update_cnt_in));
                max_position_out1_2 <=
Max_Position1(conv_integer(ar_address(6 to 9)));
                min_position_out1_2 <=
Min_Position1(conv_integer(ar_address(6 to 9)));
```

```
                        max_position_out2_2 <=
Max_Position2(conv_integer(ar_address(6 to 9)));
                        min_position_out2_2 <=
Min_Position2(conv_integer(ar_address(6 to 9)));

                        ar_data_out(7) <= Max_Velocities(conv_integer(ar_address(7 to
10)));
                        max_velocity_out <=
Max_Velocities(conv_integer(update_cnt_in));

                end if;
        end process;


        ar_data_out(0) <= Input_reg(0+2*CONV_INTEGER(ar_address(7 to 10))) &

        Input_reg(1+2*CONV_INTEGER(ar_address(7 to 10)));
        ar_data_out(1) <= RAM1_Data_Out & RAM2_Data_Out when ar_address1 <
x"0ec" else
                RAM3_Data_Out & RAM4_Data_Out when ar_address1 < x"1a2"
else
                                                RAM5_Data_Out &
RAM6_Data_Out;
        ar_data_out(2) <= Bitmap_reg(CONV_INTEGER(ar_address(5 to 10)));
        ar_data_out(3) <= PSO_Global_Best(0+2*CONV_INTEGER(ar_address(7 to
10))) &

        PSO_Global_Best(1+2*CONV_INTEGER(ar_address(7 to 10)));
        ar_data_out(4) <= current_position_out2;
        ar_data_out(5) <= current_velocity_out2;
        ar_data_out(6) <= max_position_out1_2 & min_position_out1_2 when
ar_address(10) = '0' else max_position_out2_2 & min_position_out2_2;


 IP2BUS_ARDATA_PROC : process( ar_data_out, ar_select ) is
 begin
  case ar_select is
    when "10000000" => IP2Bus_ArData <= ar_data_out(0);
    when "01000000" => IP2Bus_ArData <= ar_data_out(1);
    when "00100000" => IP2Bus_ArData <= ar_data_out(2);
    when "00010000" => IP2Bus_ArData <= ar_data_out(3);
    when "00001000" => IP2Bus_ArData <= ar_data_out(4);
    when "00000100" => IP2Bus_ArData <= ar_data_out(5);
    when "00000010" => IP2Bus_ArData <= ar_data_out(6);
    when "00000001" => IP2Bus_ArData <= ar_data_out(7);
    when others => IP2Bus_ArData <= (others => '0');
```

```
  end case;
end process IP2BUS_ARDATA_PROC;


        PSO_StateMemory: process(Bus2IP_Clk,Reset_i)
        begin
                if(Reset_i='1') then
                        PSO_state <= initial;
                        agent_cnt <= (others => '0');
                        iteration_cnt <= (others => '0');
                        load_cnt <= (others => '0');
                elsif(rising_edge(Bus2IP_Clk)) then
                        PSO_state <= PSO_next_state;

                        if(SetAgentCnt0 = '1') then
                                agent_cnt <= (others => '0');
                        elsif(IncrAgentCnt = '1' and agent_cnt < num_agents-1) then
                                agent_cnt <= agent_cnt + 1;
                        elsif(IncrAgentCnt = '1' and agent_cnt = num_agents-1) then
                                agent_cnt <= (others => '0');
                        end if;

                        if(SetIterationCnt0 = '1') then
                                iteration_cnt <= (others => '0');
                        elsif(IncrAgentCnt = '1' and agent_cnt = num_agents-1) then
                                iteration_cnt <= iteration_cnt + 1;
                        end if;

                        if(SetLoadCnt0 = '1') then
                                load_cnt <= (others => '0');
                        elsif(IncrLoadCnt = '1') then
                                load_cnt <= load_cnt + 1;
                        end if;

                        load_cnt_dly <= load_cnt;

                end if;
        end process;

        PSO_NextStateLogic:
process(PSO_state,PSO_Start,NN_Done,Update_Done,agent_cnt,load_cnt)
        begin
                case PSO_state is

                        when initial =>
                          if(PSO_Start = '1') then
```

```
                    PSO_next_state <= load_inputs;
                  else PSO_next_state <= initial;
                  end if;
              when load_inputs =>
                if(load_cnt >= x"e") then
                    PSO_next_state <= start_calc_nn;
                  else PSO_next_state <= load_inputs;
                  end if;
              when start_calc_nn =>
                    PSO_next_state <= calc_nn;
              when calc_nn =>
                    if(NN_Done = '1') then
                      PSO_next_state <= update_best;
                  else PSO_next_state <= calc_nn;
                  end if;
              when update_best =>
                if(Current_Fitness < Global_Fitness) then
                    PSO_next_state <= update_global_local;
                  elsif(Current_Fitness <
Local_Fitness(conv_integer(agent_cnt)) or iteration_cnt = x"00000000") then
                        PSO_next_state <= update_only_local;
                  else PSO_next_state <= update_agent;
                  end if;
              when update_global_local =>
                if(load_cnt >= x"d") then
                    PSO_next_state <= update_agent;
                  else PSO_next_state <= update_global_local;
                  end if;
              when update_only_local =>
                if(load_cnt >= x"d") then
                    PSO_next_state <= update_agent;
                  else PSO_next_state <= update_only_local;
                  end if;
              when update_agent =>
                if(Update_Done = '1') then
                    PSO_next_state <= update_count;
                  else PSO_next_state <= update_agent;
                  end if;
              when update_count =>
                if(agent_cnt >= num_agents-1 and iteration_cnt >=
num_iterations-1 and PSO_Start = '0') then
                    PSO_next_state <= finished1;
                elsif(agent_cnt >= num_agents-1 and iteration_cnt >=
num_iterations-1 and PSO_Start = '1') then
                    PSO_next_state <= finished2;
                  else PSO_next_state <= load_inputs;
```

```
                    end if;
                when finished1 =>
                  if(PSO_Start = '1') then
                        PSO_next_state <= load_inputs;
                    else PSO_next_state <= finished1;
                    end if;
                when finished2 =>
                  if(PSO_Start = '0') then
                        PSO_next_state <= finished1;
                    else PSO_next_state <= finished2;
                    end if;

        end case;
end process;

PSO_OutputLogic: process(PSO_state)
begin

        PSO_Done <= '0';
        PSO_Running_i <= '1';
        SetAgentCnt0 <= '0';
        SetIterationCnt0 <= '0';
        SetLoadCnt0 <= '0';
        IncrAgentCnt <= '0';
        IncrLoadCnt <= '0';
        Update_Rd_En <= '0';
        LoadInputs <= '0';
        WriteGlobalBest <= '0';
        WriteLocalBest <= '0';
        PSO_Start_NN <= '0';
        Update_Start <= '0';
        Reset_Current_Fitness <= '0';

        if(PSO_state = initial) then
                PSO_Running_i <= '0';
                SetAgentCnt0 <= '1';
                SetIterationCnt0 <= '1';
                SetLoadCnt0 <= '1';
        end if;

        if(PSO_state = load_inputs) then
                LoadInputs <= '1';
                IncrLoadCnt <= '1';
        end if;

        if(PSO_state = start_calc_nn) then
```

```
        PSO_Start_NN <= '1';
        Reset_Current_Fitness <= '1';
end if;

if(PSO_state = calc_nn) then
        PSO_Start_NN <= '1';
end if;

if(PSO_state = update_best) then
        SetLoadCnt0 <= '1';
end if;

if(PSO_state = update_global_local) then
        IncrLoadCnt <= '1';
        WriteGlobalBest <= '1';
        WriteLocalBest <= '1';
end if;

if(PSO_state = update_only_local) then
        IncrLoadCnt <= '1';
        WriteLocalBest <= '1';
end if;

if(PSO_state = update_agent) then
                Update_Start <= '1';
                Update_Rd_En <= '1';
end if;

if(PSO_state = update_count) then
        SetLoadCnt0 <= '1';
        IncrAgentCnt <= '1';
end if;

if(PSO_state = finished1) then
        PSO_Done <= '1';
        PSO_Running_i <= '0';
        SetAgentCnt0 <= '1';
        SetIterationCnt0 <= '1';
        SetLoadCnt0 <= '1';
end if;

if(PSO_state = finished2) then
        PSO_Done <= '1';
        PSO_Running_i <= '0';
        SetAgentCnt0 <= '1';
        SetIterationCnt0 <= '1';
```

```
                SetLoadCnt0 <= '1';
            end if;

        end process;

        OutNodeNum2_i <= rx_count2 + 472;
        OutNodeNum3_i <= rx_count3 + 836;
        FITNESS:process(Bus2IP_Clk,Reset_i)
        begin
            if(Reset_i = '1') then
                Current_Fitness <= (others => '0');
                Global_Fitness <= x"ffffffff";
            elsif(rising_edge(Bus2IP_Clk)) then

                if(Reset_Current_Fitness = '1') then
                    Current_Fitness <= (others => '0');
                else Current_Fitness <= Current_Fitness_Board1 +
Current_Fitness_Board2 + Current_Fitness_Board3;
                end if;

                if(Reset_Current_Fitness = '1') then
                    Current_Fitness_Board1 <= (others => '0');

                elsif(Bitmap_reg(conv_integer(OutNodeNum_i(10 downto
5)))(conv_integer(31-OutNodeNum_i(4 downto 0))) = '1') then
                    if(OutputWrEn_i = '1' and OutputNode_i(15) = '1') then
                        Current_Fitness_Board1 <=
Current_Fitness_Board1 - (x"ffff" & OutputNode_i);
                    elsif(OutputWrEn_i = '1' and OutputNode_i(15) = '0') then
                        Current_Fitness_Board1 <=
Current_Fitness_Board1 + (x"0000" & OutputNode_i);
                    end if;
                end if;

                if(Reset_Current_Fitness = '1') then
                    Current_Fitness_Board2 <= (others => '0');

                elsif(Bitmap_reg(conv_integer(OutNodeNum2_i(10 downto
5)))(conv_integer(31-OutNodeNum2_i(4 downto 0))) = '1') then
                    if((not rx_src_rdy_n2) = '1' and rx_d2(0) = '1') then
                        Current_Fitness_Board2 <=
Current_Fitness_Board2 - (x"ffff" & rx_d2);
                    elsif((not rx_src_rdy_n2) = '1' and rx_d2(0) = '0') then
                        Current_Fitness_Board2 <=
Current_Fitness_Board2 + (x"0000" & rx_d2);
                    end if;
```

```
                        end if;

                        if(Reset_Current_Fitness = '1') then
                                Current_Fitness_Board3 <= (others => '0');

                        elsif(Bitmap_reg(conv_integer(OutNodeNum3_i(10 downto
5)))(conv_integer(31-OutNodeNum3_i(4 downto 0))) = '1') then
                                if((not rx_src_rdy_n3) = '1' and rx_d3(0) = '1') then
                                        Current_Fitness_Board3 <=
Current_Fitness_Board3 - (x"ffff" & rx_d3);
                                elsif((not rx_src_rdy_n3) = '1' and rx_d3(0) = '0') then
                                        Current_Fitness_Board3 <=
Current_Fitness_Board3 + (x"0000" & rx_d3);
                                end if;
                        end if;

                        if(WriteGlobalBest = '1') then
                                Global_Fitness <= Current_Fitness;
                        end if;

                        if(WriteLocalBest = '1') then
                                Local_Fitness(conv_integer(agent_cnt)) <=
Current_Fitness;
                        end if;

                end if;
        end process;


        TX_MEM2:process(Bus2IP_Clk)
        begin
                if(Reset_i = '1') then
                        tx_state2 <= initial;
                        tx_count2 <= "00000";
                elsif(rising_edge(Bus2IP_Clk)) then
                        tx_state2 <= tx_next_state2;
                        if(tx_state2 = pre_xfer) then
                          tx_count2 <= "00000";
                        end if;
                        if((not tx_dst_rdy_n2)='1' and tx_state2 = xfer) then
                          tx_count2 <= tx_count2 + 1;
                        end if;
                end if;
        end process;

        TX_MEM3:process(Bus2IP_Clk)
```

```
        begin
                if(Reset_i = '1') then
                        tx_state3 <= initial;
                        tx_count3 <= "00000";
                elsif(rising_edge(Bus2IP_Clk)) then
                        tx_state3 <= tx_next_state3;
                        if(tx_state3 = pre_xfer) then
                          tx_count3 <= "00000";
                        end if;
                        if((not tx_dst_rdy_n3)='1' and tx_state3 = xfer) then
                          tx_count3 <= tx_count3 + 1;
                        end if;
                end if;
        end process;


  TX_NEXT_STATE_LOGIC2:
process(tx_state2,NN_Start,Reset_i,NN_Done,tx_dst_rdy_n2,tx_count2)
        begin
                case tx_state2 is

                        when initial =>
                          if(NN_Start = '1') then
                                        if((not tx_dst_rdy_n2)='1') then
                                                tx_next_state2 <= pre_xfer;
                                        else
                                          tx_next_state2 <= wait1;
                                        end if;
                                else tx_next_state2 <= initial;
                                end if;

                        when wait1 =>
                          if((not tx_dst_rdy_n2)='1') then
                                  tx_next_state2 <= pre_xfer;
                                else tx_next_state2 <= wait1;
                                end if;

                        when pre_xfer =>
                          if((not tx_dst_rdy_n2)='1') then
                                  tx_next_state2 <= xfer;
                                else
                                        tx_next_state2 <= pre_xfer;
                                end if;

                        when xfer =>
                          if(tx_count2 = x"1a") then
```

```
                        tx_next_state2 <= start1;
                    else
                        tx_next_state2 <= xfer;
                    end if;

                when start1 =>
                    if((not tx_dst_rdy_n2)='1') then
                        tx_next_state2 <= start2;
                    else tx_next_state2 <= start1;
                    end if;

                when start2 =>
                    if((not tx_dst_rdy_n2)='1') then
                        tx_next_state2 <= running;
                    else tx_next_state2 <= start2;
                    end if;

                when running =>
                    if(NN_Done = '1') then
                        if(NN_Start = '0') then
                            tx_next_state2 <= initial;
                        else tx_next_state2 <= wait2;
                        end if;
                    else tx_next_state2 <= running;
                    end if;

                when wait2 =>
                    if(NN_Start = '0') then
                        tx_next_state2 <= initial;
                    else tx_next_state2 <= wait2;
                    end if;

            end case;
        end process;


    TX_NEXT_STATE_LOGIC3:
process(tx_state3,NN_Start,Reset_i,NN_Done,tx_dst_rdy_n3,tx_count3)
        begin
            case tx_state3 is

                when initial =>
                    if(NN_Start = '1') then
                        if((not tx_dst_rdy_n3)='1') then
                            tx_next_state3 <= pre_xfer;
                        else
```

```
                    tx_next_state3 <= wait1;
                end if;
        else tx_next_state3 <= initial;
        end if;

when wait1 =>
  if((not tx_dst_rdy_n3)='1') then
          tx_next_state3 <= pre_xfer;
        else tx_next_state3 <= wait1;
        end if;

when pre_xfer =>
  if((not tx_dst_rdy_n3)='1') then
          tx_next_state3 <= xfer;
        else
                tx_next_state3 <= pre_xfer;
        end if;

when xfer =>
  if(tx_count3 = x"1a") then
          tx_next_state3 <= start1;
        else
          tx_next_state3 <= xfer;
        end if;

when start1 =>
  if((not tx_dst_rdy_n3)='1') then
          tx_next_state3 <= start2;
        else tx_next_state3 <= start1;
        end if;

when start2 =>
  if((not tx_dst_rdy_n3)='1') then
          tx_next_state3 <= running;
        else tx_next_state3 <= start2;
        end if;

when running =>
  if(NN_Done = '1') then
                if(NN_Start = '0') then
                        tx_next_state3 <= initial;
                else tx_next_state3 <= wait2;
                end if;
        else tx_next_state3 <= running;
        end if;
```

```
                when wait2 =>
                  if(NN_Start = '0') then
                          tx_next_state3 <= initial;
                        else tx_next_state3 <= wait2;
                        end if;

        end case;
end process;


TX_OUTPUT_LOGIC2: process(tx_count2,tx_state2)
begin

  tx_d2 <= (others => '0');
        tx_src_rdy_n2 <= '0';

        if(tx_state2 = initial) then
                tx_src_rdy_n2 <= '1';
        end if;
        if(tx_state2 = wait1) then
                tx_src_rdy_n2 <= '1';
        end if;
        if(tx_state2 = pre_xfer) then
                tx_d2 <= x"4000";
        end if;
        if(tx_state2 = xfer) then
    tx_d2 <= Input_reg(conv_integer(tx_count2));
        end if;
        if(tx_state2 = start1) then
                tx_d2 <= x"8001";
        end if;
        if(tx_state2 = start2) then
                tx_d2 <= x"8000";
        end if;
        if(tx_state2 = running) then
                tx_src_rdy_n2 <= '1';
        end if;
        if(tx_state2 = wait2) then
                tx_src_rdy_n2 <= '1';
        end if;
end process;


TX_OUTPUT_LOGIC3: process(tx_count3,tx_state3)
begin
```

```
  tx_d3 <= (others => '0');
        tx_src_rdy_n3 <= '0';

        if(tx_state3 = initial) then
                tx_src_rdy_n3 <= '1';
        end if;
        if(tx_state3 = wait1) then
                tx_src_rdy_n3 <= '1';
        end if;
        if(tx_state3 = pre_xfer) then
                tx_d3 <= x"4000";
        end if;
        if(tx_state3 = xfer) then
    tx_d3 <= Input_reg(conv_integer(tx_count3));
        end if;
        if(tx_state3 = start1) then
                tx_d3 <= x"8001";
        end if;
        if(tx_state3 = start2) then
                tx_d3 <= x"8000";
        end if;
        if(tx_state3 = running) then
                tx_src_rdy_n3 <= '1';
        end if;
        if(tx_state3 = wait2) then
                tx_src_rdy_n3 <= '1';
        end if;
end process;


RX_COUNT_LOGIC2:process(Bus2IP_Clk)
begin
        if(Reset_i = '1') then
                rx_count2 <= x"000";
        elsif(rising_edge(Bus2IP_Clk)) then
                if((not rx_src_rdy_n2)='1') then
                  rx_count2 <= rx_count2 + 1;
                elsif(tx_state2 = start1) then
                  rx_count2 <= x"000";
                end if;
        end if;
end process;


RX_COUNT_LOGIC3:process(Bus2IP_Clk)
begin
```

```
        if(Reset_i = '1') then
                rx_count3 <= x"000";
        elsif(rising_edge(Bus2IP_Clk)) then
                if((not rx_src_rdy_n3)='1') then
                  rx_count3 <= rx_count3 + 1;
                elsif(tx_state3 = start1) then
                  rx_count3 <= x"000";
                end if;
        end if;
end process;


BOARD2_DONE_READY_LOGIC:process(Bus2IP_Clk)
begin
        if(Reset_i = '1') then
                board2_done <= '0';
                board2_ready <= '0';
        elsif(rising_edge(Bus2IP_Clk)) then
          if(rx_count2 > x"000") then
                  board2_ready <= '1';
                else board2_ready <= '0';
                end if;
                --364-1=363=0x16b
                if(rx_count2 >= x"16b") then
                  board2_done <= '1';
                else board2_done <= '0';
                end if;
        end if;
end process;

BOARD3_DONE_READY_LOGIC:process(Bus2IP_Clk)
begin
        if(Reset_i = '1') then
                board3_done <= '0';
                board3_ready <= '0';
        elsif(rising_edge(Bus2IP_Clk)) then
          if(rx_count3 > x"000") then
                  board3_ready <= '1';
                else board3_ready <= '0';
                end if;
                if(rx_count3 >= x"16b") then
                  board3_done <= '1';
                else board3_done <= '0';
                end if;
        end if;
end process;
```

```
------------------------------------------
-- Code to drive IP to Bus signals
------------------------------------------
IP2Bus_Data       <= slv_ip2bus_data;
IP2Bus_Ack        <= slv_write_ack or slv_read_ack or ar_write_ack or ar_read_ack;
IP2Bus_Error      <= '0';
IP2Bus_Retry      <= '0';
IP2Bus_ToutSup    <= '0';

------------------------------------------------------------
        -- Component Port Maps
        ------------------------------------------------------------
U1 : Sonar_NN1
  port map ( clk => Bus2IP_Clk,
             Start => NN_Start,
             Reset => Reset_i,
                                  Done => board1_done,
                             OutputWrEn => OutputWrEn_i,
                                  OutNodeNum => OutNodeNum_i,
             OutputNode => OutputNode_i,
             Input00 => Input_reg(0),
             Input01 => Input_reg(1),
             Input02 => Input_reg(2),
             Input03 => Input_reg(3),
             Input04 => Input_reg(4),
             Input05 => Input_reg(5),
             Input06 => Input_reg(6),
             Input07 => Input_reg(7),
             Input08 => Input_reg(8),
             Input09 => Input_reg(9),
             Input10 => Input_reg(10),
             Input11 => Input_reg(11),
             Input12 => Input_reg(12),
             Input13 => Input_reg(13),
             Input14 => Input_reg(14),
             Input15 => Input_reg(15),
             Input16 => Input_reg(16),
             Input17 => Input_reg(17),
             Input18 => Input_reg(18),
             Input19 => Input_reg(19),
             Input20 => Input_reg(20),
             Input21 => Input_reg(21),
             Input22 => Input_reg(22),
             Input23 => Input_reg(23),
             Input24 => Input_reg(24),
             Input25 => Input_reg(25),
```

```
        Input26 => Input_reg(26));


    U2 : PSO_Update
  port map ( clk => Bus2IP_Clk,
                                Start => Update_Start,
                                Reset => Reset_i,
                                Done => Update_Done,
                                Update_Wr_En => Update_Wr_En,
                                input_index => update_cnt_in,
                                output_index => update_cnt_out,
                                constant_global => const_CG(16 to 31),
                                constant_local => const_CL(16 to 31),
                                current_position1 => current_position_out2(31
downto 16),
                                current_position2 => current_position_out2(15
downto 0),
                                current_velocity1 => current_velocity_out2(31
downto 16),
                                current_velocity2 => current_velocity_out2(15
downto 0),
                                local_best1 => local_best_out(31 downto 16),
                                local_best2 => local_best_out(15 downto 0),
                                global_best1 => global_best_out(31 downto 16),
                                global_best2 => global_best_out(15 downto 0),
                                max_velocity1 => max_velocity_out(31 downto
16),
                                max_velocity2 => max_velocity_out(15 downto
0),
--                              max_position1 => maxmin_position_out1(31
downto 16),
--                              max_position2 => maxmin_position_out2(31
downto 16),
--                              min_position1 => maxmin_position_out1(15
downto 0),
--                              min_position2 => maxmin_position_out2(15
downto 0),
                                max_position1 => max_position_out1_1,
                                max_position2 => max_position_out2_1,
                                min_position1 => min_position_out1_1,
                                min_position2 => min_position_out2_1,
                                next_position1 => next_position_in(31 downto
16),
                                next_position2 => next_position_in(15 downto 0),
                                next_velocity1 => next_velocity_in(31 downto
16),
```

```
                                        next_velocity2 => next_velocity_in(15 downto
0));


  U3 : aurora_connect16
       generic map (
    EXTEND_WATCHDOGS       => FALSE)
       port map(  TX_D => tx_d2,
         TX_SRC_RDY_N => tx_src_rdy_n2,
         TX_DST_RDY_N => tx_dst_rdy_n2,
         RX_D => rx_d2,
         RX_SRC_RDY_N => rx_src_rdy_n2,
         DO_CC => do_cc_i2,
                                        WARN_CC => warn_cc_i2,
         RXP => RXP1,
         RXN => RXN1,
         TXP => TXP1,
         TXN => TXN1,
         TOP_REF_CLK => Aurora_Clk,
         HARD_ERROR => hard_error_i2,
         SOFT_ERROR => soft_error_i2,
         CHANNEL_UP => channel_up_i2,
         LANE_UP => lane_up_i2,
         DCM_NOT_LOCKED => '0',
         USER_CLK => Bus2IP_Clk,
         RESET => Bus2IP_Reset,
         POWER_DOWN => '0',
         LOOPBACK => "00");

       U4 : STANDARD_CC_MODULE
         port map(  WARN_CC => warn_cc_i2,
         DO_CC => do_cc_i2,
         DCM_NOT_LOCKED => '0',
         USER_CLK => Bus2IP_Clk,
         CHANNEL_UP => channel_up_i2);

  U5 : aurora_connect16
       generic map (
    EXTEND_WATCHDOGS       => FALSE)
       port map(  TX_D => tx_d3,
         TX_SRC_RDY_N => tx_src_rdy_n3,
         TX_DST_RDY_N => tx_dst_rdy_n3,
         RX_D => rx_d3,
         RX_SRC_RDY_N => rx_src_rdy_n3,
         DO_CC => do_cc_i3,
                                        WARN_CC => warn_cc_i3,
```

```
            RXP => RXP2,
            RXN => RXN2,
            TXP => TXP2,
            TXN => TXN2,
            TOP_REF_CLK => Aurora_Clk,
            HARD_ERROR => hard_error_i3,
            SOFT_ERROR => soft_error_i3,
            CHANNEL_UP => channel_up_i3,
            LANE_UP => lane_up_i3,
            DCM_NOT_LOCKED => '0',
            USER_CLK => Bus2IP_Clk,
            RESET => Bus2IP_Reset,
            POWER_DOWN => '0',
            LOOPBACK => "00");

        U6 : STANDARD_CC_MODULE
          port map(  WARN_CC => warn_cc_i3,
            DO_CC => do_cc_i3,
            DCM_NOT_LOCKED => '0',
            USER_CLK => Bus2IP_Clk,
            CHANNEL_UP => channel_up_i3);


  U7 : BUFG
      port map (  I => Bus2IP_Clk,
                  O => Aurora_Clk);



end IMP;
```

Board_2_Logic.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use WORK.AURORA.all;

library UNISIM;
use UNISIM.all;


entity Board_2_Logic is
  port
```

```
 ( Clk                                                    : in  std_logic;
        Reset_n                                                  : in
std_logic;
   HARD_ERROR                                                 : out std_logic;
   SOFT_ERROR                                                 : out std_logic;
   LANE_UP                                                    : out std_logic;
   CHANNEL_UP                                                 : out std_logic;
   TXP                                                        : out std_logic;
        TXN                                                      : out
std_logic;
        RXP                                                      : in
std_logic;
        RXN                                                      : in
std_logic);
end Board_2_Logic;

architecture Behavioral of Board_2_Logic is

        component Sonar_NN2 is
                Port(   clk         : in  STD_LOGIC;
                        Start       : in  STD_LOGIC;
                        Reset       : in  STD_LOGIC;
                        Ready                : out STD_LOGIC;
                        Done                 : out STD_LOGIC;
                        OutputWrEn           : out STD_LOGIC;
                        OutNodeNum           : out
STD_LOGIC_VECTOR(11 downto 0);
                        OutputNode   : out STD_LOGIC_VECTOR(15
downto 0);
                        Input00      : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input01      : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input02      : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input03      : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input04      : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input05      : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input06      : in  STD_LOGIC_VECTOR(15
downto 0);
                        Input07      : in  STD_LOGIC_VECTOR(15
downto 0);
```

```
                                        Input08      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input09      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input10      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input11      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input12      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input13      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input14      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input15      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input16      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input17      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input18      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input19      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input20      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input21      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input22      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input23      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input24      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input25      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input26      : in  STD_LOGIC_VECTOR(15
downto 0));
        end component;


        component aurora_connect16 is
    generic( EXTEND_WATCHDOGS   : boolean := FALSE);
    port (
                        -- TX Stream Interface
         TX_D               : in  std_logic_vector(0 to 15);
```

```
        TX_SRC_RDY_N       : in  std_logic;
        TX_DST_RDY_N       : out std_logic;
                    -- RX Stream Interface
        RX_D               : out std_logic_vector(0 to 15);
        RX_SRC_RDY_N       : out std_logic;
                    -- Clock Correction Interface
        DO_CC              : in  std_logic;
        WARN_CC            : in  std_logic;
                    -- MGT Serial I/O
        RXP                : in std_logic;
        RXN                : in std_logic;
        TXP                : out std_logic;
        TXN                : out std_logic;
                    -- MGT Reference Clock Interface
        TOP_REF_CLK     : in std_logic;
                    -- Error Detection Interface
        HARD_ERROR         : out std_logic;
        SOFT_ERROR         : out std_logic;
                    -- Status
        CHANNEL_UP         : out std_logic;
        LANE_UP            : out std_logic;
                    -- System Interface
        DCM_NOT_LOCKED     : in std_logic;
        USER_CLK           : in std_logic;
        RESET              : in std_logic;
        POWER_DOWN         : in std_logic;
        LOOPBACK           : in std_logic_vector(1 downto 0));
end component;


       component STANDARD_CC_MODULE
  port (
   -- Clock Compensation Control Interface
         WARN_CC       : out std_logic;
         DO_CC         : out std_logic;
   -- System Interface
         DCM_NOT_LOCKED : in std_logic;
         USER_CLK       : in std_logic;
         CHANNEL_UP     : in std_logic);
end component;


component BUFG
    port (
          O : out std_ulogic;
          I : in  std_ulogic);
```

```
        end component;


  ------------------------------------
  -- Aurora Signals
  ------------------------------------
  signal hard_error_i              : std_logic;
  signal soft_error_i              : std_logic;
  signal channel_up_i               : std_logic;
  signal lane_up_i                 : std_logic;
  signal warn_cc_i                                          : std_logic;
  signal do_cc_i                                                  : std_logic;
  signal tx_d                                                     :
std_logic_vector(0 to 15);
  signal rx_d                                                     :
std_logic_vector(0 to 15);
  signal tx_src_rdy_n                                  : std_logic;
  signal tx_dst_rdy_n                                  : std_logic;
  signal rx_src_rdy_n                                  : std_logic;
  ------------------------------------
  -- RAM and register declarations
  ------------------------------------
  type REG_TYPE is array (0 to 31) of std_logic_vector(15 downto 0);
  type RAM_TYPE is array (0 to 511) of std_logic_vector(15 downto 0);
  signal Input_reg    : REG_TYPE;
  signal Output_RAM   : RAM_TYPE;
  signal status_reg0 : std_logic_vector(15 downto 0);
  signal status_reg1 : std_logic_vector(15 downto 0);
  ------------------------------------
  -- Other Signals
  ------------------------------------
  signal user_clk : std_logic;
  signal Reset : std_logic;
  signal OutputWrEn_i : std_logic;
  signal OutNodeNum_i : std_logic_vector(11 downto 0);
  signal OutputNode_i : std_logic_vector(15 downto 0);
  signal Reset_i : std_logic;
  signal Done_i : std_logic;
  signal Start_i : std_logic;
  signal Ready_i : std_logic;
  ------------------------------------
  -- State Machine Signals
  ------------------------------------
  type state_type1 is (status,node_data);
  type state_type2 is (initial,xfer,done);
  signal rx_state                  : state_type1;
```

```
  signal tx_state    : state_type2;
  signal rx_next_state : state_type1;
  signal tx_next_state : state_type2;
  signal rx_count : std_logic_vector(4 downto 0);
  signal tx_count : std_logic_vector(11 downto 0);
  ----------------------------------------

begin

        Start_i <= status_reg0(0);
        Reset <= not Reset_n;
   Reset_i <= Reset or status_reg0(1);

        HARD_ERROR <= hard_error_i;
        SOFT_ERROR <= soft_error_i;
        LANE_UP <= not lane_up_i;
        CHANNEL_UP <= not channel_up_i;


        STATUS_REG : process(user_clk,Reset_i)
        begin
                if(Reset_i = '1') then
                        status_reg0 <= (others => '0');
                        status_reg1 <= (others => '0');
                elsif(rising_edge(user_clk)) then
                        status_reg1 <= x"000" & "000" & Done_i;
                        if((not rx_src_rdy_n)='1' and rx_state = status) then
                          status_reg0 <= rx_d;
                  end if;
                end if;
        end process;


        INPUT_REG_LOGIC : process(user_clk,Reset_i)
        begin
                if(Reset_i = '1') then
                        for index in 0 to 31 loop
                                Input_reg(index) <= (others => '0');
                        end loop;
                elsif(rising_edge(user_clk)) then
                        if((not rx_src_rdy_n)='1' and rx_state = node_data) then
                                Input_reg(CONV_INTEGER(rx_count)) <= rx_d;
                        end if;
                end if;
        end process;
```

```
	OUTPUTRAM:process(user_clk)
	begin
		if(user_clk'event and user_clk = '1') then
			if(OutputWrEn_i='1') then
				Output_RAM(conv_integer(OutNodeNum_i(8 downto 0)))
<= OutputNode_i;
			end if;
			tx_d <= Output_RAM(CONV_INTEGER(tx_count(8 downto 0)));
		end if;
	end process;


	RX_STATE_MEM:process(user_clk,Reset_i)
	begin
		if(Reset_i = '1') then
			rx_state <= status;
			rx_count <= "00000";
		elsif(rising_edge(user_clk)) then
			rx_state <= rx_next_state;

			--rx_count logic
			if((not rx_src_rdy_n)='1' and rx_state = node_data) then
			  rx_count <= rx_count + 1;
			elsif(rx_state = status) then
			  rx_count <= "00000";
			end if;

		end if;
	end process;


	RX_NEXT_STATE_LOGIC: process(rx_state,rx_src_rdy_n,rx_count)
	begin
		case rx_state is

			when status =>
			  if((not rx_src_rdy_n)='1') then
				if(rx_d(0 to 1) = "10") then
					rx_next_state <= status;
				elsif(rx_d(0 to 1) = "01") then
				  rx_next_state <= node_data;
				end if;
			  else rx_next_state <= status;
			  end if;

			when node_data =>
```

```
                    if(rx_count = x"1a") then
                            rx_next_state <= status;
                        else rx_next_state <= node_data;
                        end if;

        end case;
end process;


TX_STATE_MEM:process(user_clk,Reset_i)
begin
        if(Reset_i = '1') then
                tx_state <= initial;
                tx_count <= x"000";
        elsif(rising_edge(user_clk)) then
                tx_state <= tx_next_state;

                --tx_count logic
                if(tx_state = xfer or tx_next_state = xfer) then
                        if((not tx_dst_rdy_n)='1') then
                     tx_count <= tx_count + 1;
                        end if;
                else tx_count <= x"000";
                end if;

        end if;
end process;


TX_NEXT_STATE_LOGIC:process(tx_state,Start_i,Ready_i,tx_count)
begin
        case tx_state is

                when initial =>
                  if(Ready_i = '1') then
                            tx_next_state <= xfer;
                        else tx_next_state <= initial;
                  end if;

                when xfer =>
                  if(tx_count = x"16c") then
                            tx_next_state <= done;
                        else tx_next_state <= xfer;
                  end if;

                when done =>
```

```
                    if(Start_i = '1') then
                            tx_next_state <= initial;
                          else tx_next_state <= done;
                    end if;

            end case;
        end process;


        TX_OUTPUT_LOGIC:process(tx_state)
        begin
          --tx_d is in the RAM Logic;
            if(tx_state = xfer) then
                    tx_src_rdy_n <= '0';
            else tx_src_rdy_n <= '1';
            end if;
        end process;



U1 : Sonar_NN2
  port map ( clk => user_clk,
            Start => Start_i,
            Reset => Reset_i,
                                        Ready => Ready_i,
                                Done => Done_i,
                        OutputWrEn => OutputWrEn_i,
                                OutNodeNum => OutNodeNum_i,
            OutputNode => OutputNode_i,
            Input00 => Input_reg(0),
            Input01 => Input_reg(1),
            Input02 => Input_reg(2),
            Input03 => Input_reg(3),
            Input04 => Input_reg(4),
            Input05 => Input_reg(5),
            Input06 => Input_reg(6),
            Input07 => Input_reg(7),
            Input08 => Input_reg(8),
            Input09 => Input_reg(9),
            Input10 => Input_reg(10),
            Input11 => Input_reg(11),
            Input12 => Input_reg(12),
            Input13 => Input_reg(13),
            Input14 => Input_reg(14),
            Input15 => Input_reg(15),
            Input16 => Input_reg(16),
```

```
            Input17 => Input_reg(17),
            Input18 => Input_reg(18),
            Input19 => Input_reg(19),
            Input20 => Input_reg(20),
            Input21 => Input_reg(21),
            Input22 => Input_reg(22),
            Input23 => Input_reg(23),
            Input24 => Input_reg(24),
            Input25 => Input_reg(25),
            Input26 => Input_reg(26));


U2 : aurora_connect16
        generic map (
    EXTEND_WATCHDOGS       => FALSE)
          port map(  TX_D => tx_d,
            TX_SRC_RDY_N => tx_src_rdy_n,
            TX_DST_RDY_N => tx_dst_rdy_n,
            RX_D => rx_d,
            RX_SRC_RDY_N => rx_src_rdy_n,
            DO_CC => do_cc_i,
                                        WARN_CC => warn_cc_i,
            RXP => RXP,
            RXN => RXN,
            TXP => TXP,
            TXN => TXN,
            TOP_REF_CLK => user_clk,
            HARD_ERROR => hard_error_i,
            SOFT_ERROR => soft_error_i,
            CHANNEL_UP => channel_up_i,
            LANE_UP => lane_up_i,
            DCM_NOT_LOCKED => '0',
            USER_CLK => user_clk,
            RESET => Reset,
            POWER_DOWN => '0',
            LOOPBACK => "00");

         U3 : STANDARD_CC_MODULE
           port map(  WARN_CC => warn_cc_i,
            DO_CC => do_cc_i,
            DCM_NOT_LOCKED => '0',
            USER_CLK => user_clk,
            CHANNEL_UP => channel_up_i);


U4 : BUFG
```

```
    port map(  I => Clk,
               O => user_clk);


end Behavioral;
```

Board_3_Logic.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use WORK.AURORA.all;

library UNISIM;
use UNISIM.all;


entity Board_3_Logic is
  port
  ( Clk                                                    : in  std_logic;
          Reset_n                                                  : in
std_logic;
    HARD_ERROR                                                 : out std_logic;
    SOFT_ERROR                                                 : out std_logic;
    LANE_UP                                                    : out std_logic;
    CHANNEL_UP                                                 : out std_logic;
    TXP                                                        : out std_logic;
          TXN                                                      : out
std_logic;
          RXP                                                      : in
std_logic;
          RXN                                                      : in
std_logic);
end Board_3_Logic;

architecture Behavioral of Board_3_Logic is

          component Sonar_NN3 is
                    Port(    clk          : in  STD_LOGIC;
                             Start        : in  STD_LOGIC;
                             Reset        : in  STD_LOGIC;
                             Ready                    : out STD_LOGIC;
                             Done                     : out STD_LOGIC;
                             OutputWrEn           : out STD_LOGIC;
```

```
                                OutNodeNum          : out
STD_LOGIC_VECTOR(11 downto 0);
                                OutputNode    : out STD_LOGIC_VECTOR(15
downto 0);
                                Input00       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input01       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input02       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input03       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input04       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input05       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input06       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input07       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input08       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input09       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input10       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input11       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input12       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input13       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input14       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input15       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input16       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input17       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input18       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input19       : in  STD_LOGIC_VECTOR(15
downto 0);
                                Input20       : in  STD_LOGIC_VECTOR(15
downto 0);
```

```
                                        Input21      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input22      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input23      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input24      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input25      : in  STD_LOGIC_VECTOR(15
downto 0);
                                        Input26      : in  STD_LOGIC_VECTOR(15
downto 0));
        end component;


        component aurora_connect16 is
    generic( EXTEND_WATCHDOGS   : boolean := FALSE);
    port (
                    -- TX Stream Interface
        TX_D            : in  std_logic_vector(0 to 15);
        TX_SRC_RDY_N      : in  std_logic;
        TX_DST_RDY_N      : out std_logic;
                    -- RX Stream Interface
        RX_D            : out std_logic_vector(0 to 15);
        RX_SRC_RDY_N      : out std_logic;
                    -- Clock Correction Interface
        DO_CC           : in  std_logic;
        WARN_CC           : in  std_logic;
                    -- MGT Serial I/O
        RXP             : in std_logic;
        RXN             : in std_logic;
        TXP             : out std_logic;
        TXN             : out std_logic;
                    -- MGT Reference Clock Interface
        TOP_REF_CLK     : in std_logic;
                    -- Error Detection Interface
        HARD_ERROR        : out std_logic;
        SOFT_ERROR        : out std_logic;
                    -- Status
        CHANNEL_UP        : out std_logic;
        LANE_UP           : out std_logic;
                    -- System Interface
        DCM_NOT_LOCKED    : in std_logic;
        USER_CLK          : in std_logic;
        RESET           : in std_logic;
        POWER_DOWN        : in std_logic;
```

```
        LOOPBACK          : in std_logic_vector(1 downto 0));
  end component;


      component STANDARD_CC_MODULE
    port (
     -- Clock Compensation Control Interface
        WARN_CC       : out std_logic;
        DO_CC         : out std_logic;
     -- System Interface
        DCM_NOT_LOCKED : in std_logic;
        USER_CLK      : in std_logic;
        CHANNEL_UP    : in std_logic);
  end component;


  component BUFG
      port (
          O : out std_ulogic;
          I : in  std_ulogic);
        end component;


  ------------------------------------------
  -- Aurora Signals
  ------------------------------------------
  signal hard_error_i               : std_logic;
  signal soft_error_i               : std_logic;
  signal channel_up_i                : std_logic;
  signal lane_up_i                  : std_logic;
  signal warn_cc_i                                          : std_logic;
  signal do_cc_i                                                  : std_logic;
  signal tx_d                                                     :
std_logic_vector(0 to 15);
  signal rx_d                                                     :
std_logic_vector(0 to 15);
  signal tx_src_rdy_n                                  : std_logic;
  signal tx_dst_rdy_n                                  : std_logic;
  signal rx_src_rdy_n                                  : std_logic;
  ------------------------------------------
  -- RAM and register declarations
  ------------------------------------------
  type REG_TYPE is array (0 to 31) of std_logic_vector(15 downto 0);
  type RAM_TYPE is array (0 to 511) of std_logic_vector(15 downto 0);
  signal Input_reg    : REG_TYPE;
  signal Output_RAM   : RAM_TYPE;
```

```
  signal status_reg0 : std_logic_vector(15 downto 0);
  signal status_reg1 : std_logic_vector(15 downto 0);
  ----------------------------------------
  -- Other Signals
  ----------------------------------------
  signal user_clk : std_logic;
  signal Reset : std_logic;
  signal OutputWrEn_i : std_logic;
  signal OutNodeNum_i : std_logic_vector(11 downto 0);
  signal OutputNode_i : std_logic_vector(15 downto 0);
  signal Reset_i : std_logic;
  signal Done_i : std_logic;
  signal Start_i : std_logic;
  signal Ready_i : std_logic;
  ----------------------------------------
  -- State Machine Signals
  ----------------------------------------
  type state_type1 is (status,node_data);
  type state_type2 is (initial,xfer,done);
  signal rx_state                    : state_type1;
  signal tx_state    : state_type2;
  signal rx_next_state : state_type1;
  signal tx_next_state : state_type2;
  signal rx_count : std_logic_vector(4 downto 0);
  signal tx_count : std_logic_vector(11 downto 0);
  ----------------------------------------

begin

        Start_i <= status_reg0(0);
        Reset <= not Reset_n;
   Reset_i <= Reset or status_reg0(1);

        HARD_ERROR <= hard_error_i;
        SOFT_ERROR <= soft_error_i;
        LANE_UP <= not lane_up_i;
        CHANNEL_UP <= not channel_up_i;


        STATUS_REG : process(user_clk,Reset_i)
        begin
                if(Reset_i = '1') then
                        status_reg0 <= (others => '0');
                        status_reg1 <= (others => '0');
                elsif(rising_edge(user_clk)) then
                        status_reg1 <= x"000" & "000" & Done_i;
```

```
                    if((not rx_src_rdy_n)='1' and rx_state = status) then
                      status_reg0 <= rx_d;
                  end if;
                end if;
        end process;


        INPUT_REG_LOGIC : process(user_clk,Reset_i)
        begin
                if(Reset_i = '1') then
                        for index in 0 to 31 loop
                                Input_reg(index) <= (others => '0');
                        end loop;
                elsif(rising_edge(user_clk)) then
                        if((not rx_src_rdy_n)='1' and rx_state = node_data) then
                                Input_reg(CONV_INTEGER(rx_count)) <= rx_d;
                        end if;
                end if;
        end process;


        OUTPUTRAM:process(user_clk)
        begin
                if(user_clk'event and user_clk = '1') then
                        if(OutputWrEn_i='1') then
                                Output_RAM(conv_integer(OutNodeNum_i(8 downto 0)))
<= OutputNode_i;
                        end if;
                        tx_d <= Output_RAM(CONV_INTEGER(tx_count(8 downto 0)));
                end if;
        end process;


        RX_STATE_MEM:process(user_clk,Reset_i)
        begin
                if(Reset_i = '1') then
                        rx_state <= status;
                        rx_count <= "00000";
                elsif(rising_edge(user_clk)) then
                        rx_state <= rx_next_state;

                        --rx_count logic
                        if((not rx_src_rdy_n)='1' and rx_state = node_data) then
                          rx_count <= rx_count + 1;
                        elsif(rx_state = status) then
                          rx_count <= "00000";
```

```
                end if;

        end if;
end process;


RX_NEXT_STATE_LOGIC: process(rx_state,rx_src_rdy_n,rx_count)
begin
        case rx_state is

                when status =>
                  if((not rx_src_rdy_n)='1') then
                          if(rx_d(0 to 1) = "10") then
                                  rx_next_state <= status;
                              elsif(rx_d(0 to 1) = "01") then
                                rx_next_state <= node_data;
                              end if;
                        else rx_next_state <= status;
                        end if;

                when node_data =>
                  if(rx_count = x"1a") then
                          rx_next_state <= status;
                        else rx_next_state <= node_data;
                        end if;

        end case;
end process;


TX_STATE_MEM:process(user_clk,Reset_i)
begin
        if(Reset_i = '1') then
                tx_state <= initial;
                tx_count <= x"000";
        elsif(rising_edge(user_clk)) then
                tx_state <= tx_next_state;

                --tx_count logic
                if(tx_state = xfer or tx_next_state = xfer) then
                        if((not tx_dst_rdy_n)='1') then
                     tx_count <= tx_count + 1;
                        end if;
                else tx_count <= x"000";
                end if;
```

```
        end if;
    end process;


    TX_NEXT_STATE_LOGIC:process(tx_state,Start_i,Ready_i,tx_count)
    begin
        case tx_state is

            when initial =>
              if(Ready_i = '1') then
                    tx_next_state <= xfer;
                else tx_next_state <= initial;
              end if;

            when xfer =>
              if(tx_count = x"16c") then
                    tx_next_state <= done;
                else tx_next_state <= xfer;
              end if;

            when done =>
              if(Start_i = '1') then
                    tx_next_state <= initial;
                else tx_next_state <= done;
              end if;

        end case;
    end process;


    TX_OUTPUT_LOGIC:process(tx_state)
    begin
      --tx_d is in the RAM Logic;
        if(tx_state = xfer) then
            tx_src_rdy_n <= '0';
        else tx_src_rdy_n <= '1';
        end if;
    end process;



U1 : Sonar_NN3
  port map ( clk => user_clk,
        Start => Start_i,
        Reset => Reset_i,
                                Ready => Ready_i,
```

```
                                Done => Done_i,
                              OutputWrEn => OutputWrEn_i,
                                OutNodeNum => OutNodeNum_i,
            OutputNode => OutputNode_i,
            Input00 => Input_reg(0),
            Input01 => Input_reg(1),
            Input02 => Input_reg(2),
            Input03 => Input_reg(3),
            Input04 => Input_reg(4),
            Input05 => Input_reg(5),
            Input06 => Input_reg(6),
            Input07 => Input_reg(7),
            Input08 => Input_reg(8),
            Input09 => Input_reg(9),
            Input10 => Input_reg(10),
            Input11 => Input_reg(11),
            Input12 => Input_reg(12),
            Input13 => Input_reg(13),
            Input14 => Input_reg(14),
            Input15 => Input_reg(15),
            Input16 => Input_reg(16),
            Input17 => Input_reg(17),
            Input18 => Input_reg(18),
            Input19 => Input_reg(19),
            Input20 => Input_reg(20),
            Input21 => Input_reg(21),
            Input22 => Input_reg(22),
            Input23 => Input_reg(23),
            Input24 => Input_reg(24),
            Input25 => Input_reg(25),
            Input26 => Input_reg(26));


U2 : aurora_connect16
       generic map (
   EXTEND_WATCHDOGS       => FALSE)
         port map(  TX_D => tx_d,
           TX_SRC_RDY_N => tx_src_rdy_n,
           TX_DST_RDY_N => tx_dst_rdy_n,
           RX_D => rx_d,
           RX_SRC_RDY_N => rx_src_rdy_n,
           DO_CC => do_cc_i,
                                      WARN_CC => warn_cc_i,
           RXP => RXP,
           RXN => RXN,
           TXP => TXP,
```

```
            TXN => TXN,
            TOP_REF_CLK => user_clk,
            HARD_ERROR => hard_error_i,
            SOFT_ERROR => soft_error_i,
            CHANNEL_UP => channel_up_i,
            LANE_UP => lane_up_i,
            DCM_NOT_LOCKED => '0',
            USER_CLK => user_clk,
            RESET => Reset,
            POWER_DOWN => '0',
            LOOPBACK => "00");

        U3 : STANDARD_CC_MODULE
          port map(  WARN_CC => warn_cc_i,
            DO_CC => do_cc_i,
            DCM_NOT_LOCKED => '0',
            USER_CLK => user_clk,
            CHANNEL_UP => channel_up_i);


  U4 : BUFG
    port map(  I => Clk,
            O => user_clk);


end Behavioral;
```

Sonar_NN_2.vhd

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

--all control signals are active high
--clk -> clock
--Start -> Once the Inputs are all set, Start begins the computation
--Reset -> Asynchronous Reset, resets all registers to 0 and sets the NN into a waiting to
start state
--Done -> Specifies when all of the NN Outputs have been calculated
--Input?? -> Inputs to the NN
--OutputWrEN -> Enable specifing when OutputNode has valid data. Also an Enable for
Output RAM
--OutNodeNum -> Node Number of the Output Nodes (0-1199). Used as an address for
Output RAM
```

```
--OutputNode -> Output value of the Node specified by OutNodeNum

entity Sonar_NN2 is
  Port(    clk          : in  STD_LOGIC;
           Start        : in  STD_LOGIC;
           Reset        : in  STD_LOGIC;
                                  Ready                          : out STD_LOGIC;
                                  Done                           : out STD_LOGIC;
                                  OutputWrEn             : out STD_LOGIC;
                                  OutNodeNum             : out STD_LOGIC_VECTOR(11
downto 0);
           OutputNode     : out STD_LOGIC_VECTOR(15 downto 0);
           Input00        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input01        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input02        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input03        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input04        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input05        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input06        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input07        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input08        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input09        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input10        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input11        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input12        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input13        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input14        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input15        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input16        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input17        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input18        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input19        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input20        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input21        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input22        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input23        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input24        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input25        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input26        : in  STD_LOGIC_VECTOR(15 downto 0));
end Sonar_NN2;

architecture Behavioral of Sonar_NN2 is


signal OutputWrEn_i                    : STD_LOGIC;
signal RegisterSelect       : STD_LOGIC_VECTOR(11 downto 0);
```

```
signal SquashOut          : STD_LOGIC_VECTOR(7 downto 0);
signal NodeCalcOut        : STD_LOGIC_VECTOR(31 downto 0);
signal NodeNumberOut                    : STD_LOGIC_VECTOR(11 downto 0);
signal NodeNumber                       : STD_LOGIC_VECTOR(11 downto 0);
signal NodeNumber_i       : STD_LOGIC_VECTOR(11 downto 0);
signal HoldReg                                  : STD_LOGIC_VECTOR(15
downto 0);

type type1 is array (0 to 70)   of std_logic_vector (7 downto 0);
type type2 is array (0 to 70)   of std_logic_vector (15 downto 0);
signal RegA             : type1;
signal RegB             : type2;


        component NodeCounter2 is
                Port ( clk                  : in  STD_LOGIC;
         Start                          : in  STD_LOGIC;
         Reset                          : in  STD_LOGIC;
                                Ready                   : out STD_LOGIC;
                                Done                            : out STD_LOGIC;
                           NodeNumber       : out STD_LOGIC_VECTOR(11 downto
0));
        end component;


  component Squash
    Port ( Input_Value      : in  STD_LOGIC_VECTOR (31 downto 0);
         Squashed_Value   : out STD_LOGIC_VECTOR (7 downto 0);
         clk            : in  STD_LOGIC);
  end component;


  component NodeCalc2 is
    Port(  clk          : in  STD_LOGIC;
                                Reset                   : in  STD_LOGIC;
         NodeNumberIn     : in  STD_LOGIC_VECTOR(11 downto 0);
         NodeNumberOut    : out STD_LOGIC_VECTOR(11 downto 0);
         Output         : out STD_LOGIC_VECTOR(31 downto 0);
         Input00        : in  STD_LOGIC_VECTOR(15 downto 0);
         Input01        : in  STD_LOGIC_VECTOR(15 downto 0);
         Input02        : in  STD_LOGIC_VECTOR(15 downto 0);
         Input03        : in  STD_LOGIC_VECTOR(15 downto 0);
         Input04        : in  STD_LOGIC_VECTOR(15 downto 0);
         Input05        : in  STD_LOGIC_VECTOR(15 downto 0);
         Input06        : in  STD_LOGIC_VECTOR(15 downto 0);
         Input07        : in  STD_LOGIC_VECTOR(15 downto 0);
```

```
Input08      : in  STD_LOGIC_VECTOR(15 downto 0);
Input09      : in  STD_LOGIC_VECTOR(15 downto 0);
Input10      : in  STD_LOGIC_VECTOR(15 downto 0);
Input11      : in  STD_LOGIC_VECTOR(15 downto 0);
Input12      : in  STD_LOGIC_VECTOR(15 downto 0);
Input13      : in  STD_LOGIC_VECTOR(15 downto 0);
Input14      : in  STD_LOGIC_VECTOR(15 downto 0);
Input15      : in  STD_LOGIC_VECTOR(15 downto 0);
Input16      : in  STD_LOGIC_VECTOR(15 downto 0);
Input17      : in  STD_LOGIC_VECTOR(15 downto 0);
Input18      : in  STD_LOGIC_VECTOR(15 downto 0);
Input19      : in  STD_LOGIC_VECTOR(15 downto 0);
Input20      : in  STD_LOGIC_VECTOR(15 downto 0);
Input21      : in  STD_LOGIC_VECTOR(15 downto 0);
Input22      : in  STD_LOGIC_VECTOR(15 downto 0);
Input23      : in  STD_LOGIC_VECTOR(15 downto 0);
Input24      : in  STD_LOGIC_VECTOR(15 downto 0);
Input25      : in  STD_LOGIC_VECTOR(15 downto 0);
Input26      : in  STD_LOGIC_VECTOR(15 downto 0);
Input27      : in  STD_LOGIC_VECTOR(15 downto 0);
Input28      : in  STD_LOGIC_VECTOR(15 downto 0);
Input29      : in  STD_LOGIC_VECTOR(15 downto 0);
Input30      : in  STD_LOGIC_VECTOR(15 downto 0);
Input31      : in  STD_LOGIC_VECTOR(15 downto 0);
Input32      : in  STD_LOGIC_VECTOR(15 downto 0);
Input33      : in  STD_LOGIC_VECTOR(15 downto 0);
Input34      : in  STD_LOGIC_VECTOR(15 downto 0);
Input35      : in  STD_LOGIC_VECTOR(15 downto 0);
Input36      : in  STD_LOGIC_VECTOR(15 downto 0);
Input37      : in  STD_LOGIC_VECTOR(15 downto 0);
Input38      : in  STD_LOGIC_VECTOR(15 downto 0);
Input39      : in  STD_LOGIC_VECTOR(15 downto 0);
Input40      : in  STD_LOGIC_VECTOR(15 downto 0);
Input41      : in  STD_LOGIC_VECTOR(15 downto 0);
Input42      : in  STD_LOGIC_VECTOR(15 downto 0);
Input43      : in  STD_LOGIC_VECTOR(15 downto 0);
Input44      : in  STD_LOGIC_VECTOR(15 downto 0);
Input45      : in  STD_LOGIC_VECTOR(15 downto 0);
Input46      : in  STD_LOGIC_VECTOR(15 downto 0);
Input47      : in  STD_LOGIC_VECTOR(15 downto 0);
Input48      : in  STD_LOGIC_VECTOR(15 downto 0);
Input49      : in  STD_LOGIC_VECTOR(15 downto 0);
Input50      : in  STD_LOGIC_VECTOR(15 downto 0);
Input51      : in  STD_LOGIC_VECTOR(15 downto 0);
Input52      : in  STD_LOGIC_VECTOR(15 downto 0);
Input53      : in  STD_LOGIC_VECTOR(15 downto 0);
```

```
        Input54         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input55         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input56         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input57         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input58         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input59         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input60         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input61         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input62         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input63         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input64         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input65         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input66         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input67         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input68         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input69         : in  STD_LOGIC_VECTOR(15 downto 0);
        Input70         : in  STD_LOGIC_VECTOR(15 downto 0));
    end component;

begin

        RegALogic:process(clk,Reset)
        begin
                if(Reset = '1') then
                        for count in 0 to 70 loop
                                RegA(count) <= (others => '0');
                        end loop;
                elsif(rising_edge(clk)) then
                        if(OutputWrEn_i = '0') then
                                RegA(conv_integer(RegisterSelect)) <= SquashOut;
                        end if;
                end if;
        end process;


        RegBlogic: process(clk,Reset)
        begin
                if(Reset = '1') then
                        for count in 0 to 70 loop
                                RegB(count) <= (others => '0');
                        end loop;
                elsif(rising_edge(clk)) then
                        if(NodeNumber = x"000" or NodeNumber = x"028" or
NodeNumber = x"05a" or NodeNumber = x"0a0") then

                                if(NodeNumber = x"000") then
```

```
RegB(0) <= Input00;
RegB(1) <= Input01;
RegB(2) <= Input02;
RegB(3) <= Input03;
RegB(4) <= Input04;
RegB(5) <= Input05;
RegB(6) <= Input06;
RegB(7) <= Input07;
RegB(8) <= Input08;
RegB(9) <= Input09;
RegB(10) <= Input10;
RegB(11) <= Input11;
RegB(12) <= Input12;
RegB(13) <= Input13;
RegB(14) <= Input14;
RegB(15) <= Input15;
RegB(16) <= Input16;
RegB(17) <= Input17;
RegB(18) <= Input18;
RegB(19) <= Input19;
RegB(20) <= Input20;
RegB(21) <= Input21;
RegB(22) <= Input22;
RegB(23) <= Input23;
RegB(24) <= Input24;
RegB(25) <= Input25;
RegB(26) <= Input26;
                RegB(27) <= "0000010000000000";
        else
                for count in 0 to 27 loop
                        RegB(count) <= "00000000" &
RegA(count);
                end loop;
        end if;

        for count in 28 to 39 loop
                RegB(count) <= "00000000" & RegA(count);
        end loop;

        if(NodeNumber = x"028") then
                RegB(40) <= "0000000100000000";
        else    RegB(40) <= "00000000" & RegA(40);
        end if;

        for count in 41 to 49 loop
                RegB(count) <= "00000000" & RegA(count);
```

```
                        end loop;

                        if(NodeNumber = x"05a") then
                                RegB(50) <= "0000000100000000";
                        else    RegB(50) <= "00000000" & RegA(50);
                        end if;

                        for count in 51 to 69 loop
                                RegB(count) <= "00000000" & RegA(count);
                        end loop;

                        if(NodeNumber  = x"0a0") then
                                RegB(70) <= "0000000100000000";
                        else    RegB(70) <= "00000000" & RegA(70);
                        end if;

                end if;
        end if;
end process;


OutHoldReg:process(clk,Reset)
begin
        if(Reset = '1') then
                HoldReg <= (others => '0');
        elsif(rising_edge(clk)) then
                HoldReg <= NodeCalcOut(24 downto 9);
        end if;
end process;


NodeReg:process(clk,Reset)
begin
        if(Reset = '1') then
                NodeNumberOut <= (others => '0');
        elsif(rising_edge(clk)) then
                NodeNumberOut <= NodeNumber_i;
        end if;
end process;


--Output Logic
OutputWrEn_i <= '1' when NodeNumberOut >= x"0a0" else '0';
OutputWrEn <= OutputWrEn_i;
OutputNode <= HoldReg;
OutNodeNum <= RegisterSelect;
```

```
        RegisterSelect <= NodeNumberOut when (NodeNumberOut < x"028") else
                                                    NodeNumberOut - 40 when
(NodeNumberOut >= x"028" and NodeNumberOut < x"05a") else
                                                    NodeNumberOut - 90 when
(NodeNumberOut >= x"05a" and NodeNumberOut < x"0a0") else
                                                    NodeNumberOut - 160 when
(NodeNumberOut >= x"0a0");



  U1 : NodeCounter2
    port map ( clk => clk,
               Start => Start,
               Reset => Reset,
                                        Ready => Ready,
                                        Done => Done,
                                NodeNumber => NodeNumber);

  U2 : Squash
    port map ( Input_Value => NodeCalcOut,
               Squashed_Value => SquashOut,
               clk => clk);

  U3 : NodeCalc2
    port map ( clk => clk,
                                        Reset => Reset,
               NodeNumberIn => NodeNumber,
               NodeNumberOut => NodeNumber_i,
               Output => NodeCalcOut,
               Input00 => RegB(0),
               Input01 => RegB(1),
               Input02 => RegB(2),
               Input03 => RegB(3),
               Input04 => RegB(4),
               Input05 => RegB(5),
               Input06 => RegB(6),
               Input07 => RegB(7),
               Input08 => RegB(8),
               Input09 => RegB(9),
               Input10 => RegB(10),
               Input11 => RegB(11),
               Input12 => RegB(12),
               Input13 => RegB(13),
               Input14 => RegB(14),
               Input15 => RegB(15),
               Input16 => RegB(16),
```

```
Input17 => RegB(17),
Input18 => RegB(18),
Input19 => RegB(19),
Input20 => RegB(20),
Input21 => RegB(21),
Input22 => RegB(22),
Input23 => RegB(23),
Input24 => RegB(24),
Input25 => RegB(25),
Input26 => RegB(26),
Input27 => RegB(27),
Input28 => RegB(28),
Input29 => RegB(29),
Input30 => RegB(30),
Input31 => RegB(31),
Input32 => RegB(32),
Input33 => RegB(33),
Input34 => RegB(34),
Input35 => RegB(35),
Input36 => RegB(36),
Input37 => RegB(37),
Input38 => RegB(38),
Input39 => RegB(39),
Input40 => RegB(40),
Input41 => RegB(41),
Input42 => RegB(42),
Input43 => RegB(43),
Input44 => RegB(44),
Input45 => RegB(45),
Input46 => RegB(46),
Input47 => RegB(47),
Input48 => RegB(48),
Input49 => RegB(49),
Input50 => RegB(50),
Input51 => RegB(51),
Input52 => RegB(52),
Input53 => RegB(53),
Input54 => RegB(54),
Input55 => RegB(55),
Input56 => RegB(56),
Input57 => RegB(57),
Input58 => RegB(58),
Input59 => RegB(59),
Input60 => RegB(60),
Input61 => RegB(61),
Input62 => RegB(62),
```

```
            Input63 => RegB(63),
            Input64 => RegB(64),
            Input65 => RegB(65),
            Input66 => RegB(66),
            Input67 => RegB(67),
            Input68 => RegB(68),
            Input69 => RegB(69),
            Input70 => RegB(70));
end Behavioral;
```

Sonar_NN_3.vhd

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

--all control signals are active high
--clk -> clock
--Start -> Once the Inputs are all set, Start begins the computation
--Reset -> Asynchronous Reset, resets all registers to 0 and sets the NN into a waiting to
start state
--Done -> Specifies when all of the NN Outputs have been calculated
--Input?? -> Inputs to the NN
--OutputWrEN -> Enable specifing when OutputNode has valid data. Also an Enable for
Output RAM
--OutNodeNum -> Node Number of the Output Nodes (0-1199). Used as an address for
Output RAM
--OutputNode -> Output value of the Node specified by OutNodeNum

entity Sonar_NN3 is
  Port(    clk          : in  STD_LOGIC;
           Start        : in  STD_LOGIC;
           Reset        : in  STD_LOGIC;
                          Ready                    : out STD_LOGIC;
                          Done                     : out STD_LOGIC;
                          OutputWrEn          : out STD_LOGIC;
                          OutNodeNum          : out STD_LOGIC_VECTOR(11
downto 0);
           OutputNode     : out STD_LOGIC_VECTOR(15 downto 0);
           Input00        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input01        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input02        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input03        : in  STD_LOGIC_VECTOR(15 downto 0);
           Input04        : in  STD_LOGIC_VECTOR(15 downto 0);
```

```
        Input05     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input06     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input07     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input08     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input09     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input10     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input11     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input12     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input13     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input14     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input15     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input16     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input17     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input18     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input19     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input20     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input21     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input22     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input23     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input24     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input25     : in  STD_LOGIC_VECTOR(15 downto 0);
        Input26     : in  STD_LOGIC_VECTOR(15 downto 0));
end Sonar_NN3;

architecture Behavioral of Sonar_NN3 is


signal OutputWrEn_i                 : STD_LOGIC;
signal RegisterSelect       : STD_LOGIC_VECTOR(11 downto 0);
signal SquashOut            : STD_LOGIC_VECTOR(7 downto 0);
signal NodeCalcOut          : STD_LOGIC_VECTOR(31 downto 0);
signal NodeNumberOut                    : STD_LOGIC_VECTOR(11 downto 0);
signal NodeNumber                       : STD_LOGIC_VECTOR(11 downto 0);
signal NodeNumber_i         : STD_LOGIC_VECTOR(11 downto 0);
signal HoldReg                                  : STD_LOGIC_VECTOR(15
downto 0);

type type1 is array (0 to 70)   of std_logic_vector (7 downto 0);
type type2 is array (0 to 70)   of std_logic_vector (15 downto 0);
signal RegA             : type1;
signal RegB             : type2;


        component NodeCounter3 is
                Port ( clk                  : in  STD_LOGIC;
         Start                          : in  STD_LOGIC;
```

```
        Reset                   : in  STD_LOGIC;
                                Ready                   : out STD_LOGIC;
                                Done                    : out STD_LOGIC;
                            NodeNumber      : out STD_LOGIC_VECTOR(11 downto
0));
        end component;


  component Squash
    Port ( Input_Value      : in  STD_LOGIC_VECTOR (31 downto 0);
           Squashed_Value   : out STD_LOGIC_VECTOR (7 downto 0);
           clk              : in  STD_LOGIC);
  end component;


  component NodeCalc3 is
    Port(  clk              : in  STD_LOGIC;
                                Reset                   : in  STD_LOGIC;
           NodeNumberIn     : in  STD_LOGIC_VECTOR(11 downto 0);
           NodeNumberOut    : out STD_LOGIC_VECTOR(11 downto 0);
           Output           : out STD_LOGIC_VECTOR(31 downto 0);
           Input00          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input01          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input02          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input03          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input04          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input05          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input06          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input07          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input08          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input09          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input10          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input11          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input12          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input13          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input14          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input15          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input16          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input17          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input18          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input19          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input20          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input21          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input22          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input23          : in  STD_LOGIC_VECTOR(15 downto 0);
           Input24          : in  STD_LOGIC_VECTOR(15 downto 0);
```

```
Input25      : in  STD_LOGIC_VECTOR(15 downto 0);
Input26      : in  STD_LOGIC_VECTOR(15 downto 0);
Input27      : in  STD_LOGIC_VECTOR(15 downto 0);
Input28      : in  STD_LOGIC_VECTOR(15 downto 0);
Input29      : in  STD_LOGIC_VECTOR(15 downto 0);
Input30      : in  STD_LOGIC_VECTOR(15 downto 0);
Input31      : in  STD_LOGIC_VECTOR(15 downto 0);
Input32      : in  STD_LOGIC_VECTOR(15 downto 0);
Input33      : in  STD_LOGIC_VECTOR(15 downto 0);
Input34      : in  STD_LOGIC_VECTOR(15 downto 0);
Input35      : in  STD_LOGIC_VECTOR(15 downto 0);
Input36      : in  STD_LOGIC_VECTOR(15 downto 0);
Input37      : in  STD_LOGIC_VECTOR(15 downto 0);
Input38      : in  STD_LOGIC_VECTOR(15 downto 0);
Input39      : in  STD_LOGIC_VECTOR(15 downto 0);
Input40      : in  STD_LOGIC_VECTOR(15 downto 0);
Input41      : in  STD_LOGIC_VECTOR(15 downto 0);
Input42      : in  STD_LOGIC_VECTOR(15 downto 0);
Input43      : in  STD_LOGIC_VECTOR(15 downto 0);
Input44      : in  STD_LOGIC_VECTOR(15 downto 0);
Input45      : in  STD_LOGIC_VECTOR(15 downto 0);
Input46      : in  STD_LOGIC_VECTOR(15 downto 0);
Input47      : in  STD_LOGIC_VECTOR(15 downto 0);
Input48      : in  STD_LOGIC_VECTOR(15 downto 0);
Input49      : in  STD_LOGIC_VECTOR(15 downto 0);
Input50      : in  STD_LOGIC_VECTOR(15 downto 0);
Input51      : in  STD_LOGIC_VECTOR(15 downto 0);
Input52      : in  STD_LOGIC_VECTOR(15 downto 0);
Input53      : in  STD_LOGIC_VECTOR(15 downto 0);
Input54      : in  STD_LOGIC_VECTOR(15 downto 0);
Input55      : in  STD_LOGIC_VECTOR(15 downto 0);
Input56      : in  STD_LOGIC_VECTOR(15 downto 0);
Input57      : in  STD_LOGIC_VECTOR(15 downto 0);
Input58      : in  STD_LOGIC_VECTOR(15 downto 0);
Input59      : in  STD_LOGIC_VECTOR(15 downto 0);
Input60      : in  STD_LOGIC_VECTOR(15 downto 0);
Input61      : in  STD_LOGIC_VECTOR(15 downto 0);
Input62      : in  STD_LOGIC_VECTOR(15 downto 0);
Input63      : in  STD_LOGIC_VECTOR(15 downto 0);
Input64      : in  STD_LOGIC_VECTOR(15 downto 0);
Input65      : in  STD_LOGIC_VECTOR(15 downto 0);
Input66      : in  STD_LOGIC_VECTOR(15 downto 0);
Input67      : in  STD_LOGIC_VECTOR(15 downto 0);
Input68      : in  STD_LOGIC_VECTOR(15 downto 0);
Input69      : in  STD_LOGIC_VECTOR(15 downto 0);
Input70      : in  STD_LOGIC_VECTOR(15 downto 0));
```

```
  end component;

begin

        RegALogic:process(clk,Reset)
        begin
                if(Reset = '1') then
                        for count in 0 to 70 loop
                                RegA(count) <= (others => '0');
                        end loop;
                elsif(rising_edge(clk)) then
                        if(OutputWrEn_i = '0') then
                                RegA(conv_integer(RegisterSelect)) <= SquashOut;
                        end if;
                end if;
        end process;


        RegBlogic: process(clk,Reset)
        begin
                if(Reset = '1') then
                        for count in 0 to 70 loop
                                RegB(count) <= (others => '0');
                        end loop;
                elsif(rising_edge(clk)) then
                        if(NodeNumber = x"000" or NodeNumber = x"028" or
NodeNumber = x"05a" or NodeNumber = x"0a0") then

                                if(NodeNumber = x"000") then
          RegB(0) <= Input00;
          RegB(1) <= Input01;
          RegB(2) <= Input02;
          RegB(3) <= Input03;
          RegB(4) <= Input04;
          RegB(5) <= Input05;
          RegB(6) <= Input06;
          RegB(7) <= Input07;
          RegB(8) <= Input08;
          RegB(9) <= Input09;
          RegB(10) <= Input10;
          RegB(11) <= Input11;
          RegB(12) <= Input12;
          RegB(13) <= Input13;
          RegB(14) <= Input14;
          RegB(15) <= Input15;
          RegB(16) <= Input16;
```

```
            RegB(17) <= Input17;
            RegB(18) <= Input18;
            RegB(19) <= Input19;
            RegB(20) <= Input20;
            RegB(21) <= Input21;
            RegB(22) <= Input22;
            RegB(23) <= Input23;
            RegB(24) <= Input24;
            RegB(25) <= Input25;
            RegB(26) <= Input26;
                                    RegB(27) <= "0000010000000000";
                                else
                                    for count in 0 to 27 loop
                                        RegB(count) <= "00000000" &
RegA(count);
                                    end loop;
                                end if;

                                for count in 28 to 39 loop
                                    RegB(count) <= "00000000" & RegA(count);
                                end loop;

                                if(NodeNumber = x"028") then
                                    RegB(40) <= "0000000100000000";
                                else    RegB(40) <= "00000000" & RegA(40);
                                end if;

                                for count in 41 to 49 loop
                                    RegB(count) <= "00000000" & RegA(count);
                                end loop;

                                if(NodeNumber = x"05a") then
                                    RegB(50) <= "0000000100000000";
                                else    RegB(50) <= "00000000" & RegA(50);
                                end if;

                                for count in 51 to 69 loop
                                    RegB(count) <= "00000000" & RegA(count);
                                end loop;

                                if(NodeNumber  = x"0a0") then
                                    RegB(70) <= "0000000100000000";
                                else    RegB(70) <= "00000000" & RegA(70);
                                end if;

                        end if;
```

```
		end if;
	end process;


	OutHoldReg:process(clk,Reset)
	begin
		if(Reset = '1') then
			HoldReg <= (others => '0');
		elsif(rising_edge(clk)) then
			HoldReg <= NodeCalcOut(24 downto 9);
		end if;
	end process;


	NodeReg:process(clk,Reset)
	begin
		if(Reset = '1') then
			NodeNumberOut <= (others => '0');
		elsif(rising_edge(clk)) then
			NodeNumberOut <= NodeNumber_i;
		end if;
	end process;


	--Output Logic
	OutputWrEn_i <= '1' when NodeNumberOut >= x"0a0" else '0';
	OutputWrEn <= OutputWrEn_i;
	OutputNode <= HoldReg;
	OutNodeNum <= RegisterSelect;


	RegisterSelect <= NodeNumberOut when (NodeNumberOut < x"028") else
                                                NodeNumberOut - 40 when
(NodeNumberOut >= x"028" and NodeNumberOut < x"05a") else
                                                NodeNumberOut - 90 when
(NodeNumberOut >= x"05a" and NodeNumberOut < x"0a0") else
                                                NodeNumberOut - 160 when
(NodeNumberOut >= x"0a0");



  U1 : NodeCounter3
    port map ( clk => clk,
          Start => Start,
          Reset => Reset,
                                    Ready => Ready,
```

```
                                        Done => Done,
                                    NodeNumber => NodeNumber);

U2 : Squash
  port map ( Input_Value => NodeCalcOut,
             Squashed_Value => SquashOut,
             clk => clk);

U3 : NodeCalc3
  port map ( clk => clk,
                                        Reset => Reset,
             NodeNumberIn => NodeNumber,
             NodeNumberOut => NodeNumber_i,
             Output => NodeCalcOut,
             Input00 => RegB(0),
             Input01 => RegB(1),
             Input02 => RegB(2),
             Input03 => RegB(3),
             Input04 => RegB(4),
             Input05 => RegB(5),
             Input06 => RegB(6),
             Input07 => RegB(7),
             Input08 => RegB(8),
             Input09 => RegB(9),
             Input10 => RegB(10),
             Input11 => RegB(11),
             Input12 => RegB(12),
             Input13 => RegB(13),
             Input14 => RegB(14),
             Input15 => RegB(15),
             Input16 => RegB(16),
             Input17 => RegB(17),
             Input18 => RegB(18),
             Input19 => RegB(19),
             Input20 => RegB(20),
             Input21 => RegB(21),
             Input22 => RegB(22),
             Input23 => RegB(23),
             Input24 => RegB(24),
             Input25 => RegB(25),
             Input26 => RegB(26),
             Input27 => RegB(27),
             Input28 => RegB(28),
             Input29 => RegB(29),
             Input30 => RegB(30),
             Input31 => RegB(31),
```

```
            Input32 => RegB(32),
            Input33 => RegB(33),
            Input34 => RegB(34),
            Input35 => RegB(35),
            Input36 => RegB(36),
            Input37 => RegB(37),
            Input38 => RegB(38),
            Input39 => RegB(39),
            Input40 => RegB(40),
            Input41 => RegB(41),
            Input42 => RegB(42),
            Input43 => RegB(43),
            Input44 => RegB(44),
            Input45 => RegB(45),
            Input46 => RegB(46),
            Input47 => RegB(47),
            Input48 => RegB(48),
            Input49 => RegB(49),
            Input50 => RegB(50),
            Input51 => RegB(51),
            Input52 => RegB(52),
            Input53 => RegB(53),
            Input54 => RegB(54),
            Input55 => RegB(55),
            Input56 => RegB(56),
            Input57 => RegB(57),
            Input58 => RegB(58),
            Input59 => RegB(59),
            Input60 => RegB(60),
            Input61 => RegB(61),
            Input62 => RegB(62),
            Input63 => RegB(63),
            Input64 => RegB(64),
            Input65 => RegB(65),
            Input66 => RegB(66),
            Input67 => RegB(67),
            Input68 => RegB(68),
            Input69 => RegB(69),
            Input70 => RegB(70));
end Behavioral;
```

NodeCounter2.vhd

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

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity NodeCounter2 is
   Port ( clk                    : in  STD_LOGIC;
          Start                  : in  STD_LOGIC;
          Reset                  : in  STD_LOGIC;
                      Ready      : out STD_LOGIC;
                      Done                  : out STD_LOGIC;
                      NodeNumber      : out  STD_LOGIC_VECTOR(11 downto
0));
end NodeCounter2;

architecture Behavioral of NodeCounter2 is


type state_type is (initial,count,wait1,wait2,finished,wait3);
signal state            : state_type;
signal next_state : state_type;
signal IncrNodeCount : STD_LOGIC;
signal IncrWaitCount : STD_LOGIC;
signal SetNodeCount0 : STD_LOGIC;
signal SetWaitCount0 : STD_LOGIC;
signal NodeCount  : STD_LOGIC_VECTOR(11 downto 0);
signal WaitCount  : STD_LOGIC_VECTOR(3 downto 0);
constant Latency : integer := 7;

begin

NodeNumber <= NodeCount;

  --Tells when the first output is available
        --Allows the driver to start data transfer with the available data
        --done tells when the process is finished
  Ready_Logic: process(clk,Reset)
        begin
              if(Reset='1') then
                    Ready <= '0';
              elsif(rising_edge(clk)) then
                if(NodeCount > x"0a7") then
                  Ready <= '1';
                    else Ready <= '0';
                    end if;
              end if;
        end process;
```

```
NodeCounter: process(clk,Reset)
begin
	if(Reset='1') then
		NodeCount <= (others => '0');
	elsif(rising_edge(clk)) then
		if(IncrNodeCount = '1') then
			NodeCount <= NodeCount + 1;
		elsif(SetNodeCount0 = '1') then
			NodeCount <= (others => '0');
		end if;
	end if;
end process;

WaitCounter: process(clk,Reset)
begin
	if(Reset='1') then
		WaitCount <= (others => '0');
	elsif(rising_edge(clk)) then
		if(IncrWaitCount = '1') then
			WaitCount <= WaitCount + 1;
		elsif(SetWaitCount0 = '1') then
			WaitCount <= (others => '0');
		end if;
	end if;
end process;

StateMemory: process(clk,Reset)
begin
	if(Reset='1') then
		state <= initial;
	elsif(rising_edge(clk)) then
		state <= next_state;
	end if;
end process;

NextStateLogic: process(state,Start,NodeCount,WaitCount)
begin
	case state is

		when initial =>
		  if(Start = '1') then
			next_state <= count;
		else next_state <= initial;
		end if;

		when count =>
```

```
                        --in decimal  if(NodeCount = 38 or NodeCount = 88 or
NodeCount = 158) then
                        --  40-2=38          40+50-2=88          40+50+70-
2=158
                    if(NodeCount = x"026" or NodeCount = x"058" or NodeCount =
x"09e") then
                        next_state <= wait1;
                    --40+50+70+364-2=522
                    --elseif(NodeCount = 522) then
                    elsif(NodeCount = x"20a") then
                        next_state <= wait2;
                    else next_state <= count;
                    end if;

                when wait1 =>
                  if(conv_integer(WaitCount) = Latency-2) then
                        next_state <= count;
                    else next_state <= wait1;
                    end if;

                when wait2 =>
                  if(conv_integer(WaitCount) = Latency-1) then
                        next_state <= finished;
                    else next_state <= wait2;
                    end if;

                when finished =>
                  if(Start = '1') then
                        next_state <= finished;
                    else next_state <= wait3;
                    end if;

                when wait3 =>
                  if(Start = '1') then
                        next_state <= count;
                    else next_state <= wait3;
                    end if;

        end case;
    end process;

    OutputLogic: process(state,Start,NodeCount,WaitCount)
    begin

        Done <= '0';
        IncrNodeCount <= '0';
```

```
                SetNodeCount0 <= '0';
                IncrWaitCount <= '0';
                SetWaitCount0 <= '0';

                if(state = initial) then
                        SetNodeCount0 <= '1';
                end if;

                if(state = count) then
                        SetWaitCount0 <= '1';
                        IncrNodeCount <= '1';
                end if;

                if(state = wait1) then
                        IncrWaitCount <= '1';
                end if;

                if(state = wait2) then
                        IncrWaitCount <= '1';
                        SetNodeCount0 <= '1';
                end if;

                if(state = finished) then
                        Done <= '1';
                end if;

                if(state = wait3) then
                        SetNodeCount0 <= '1';
                        Done <= '1';
                end if;

        end process;

end Behavioral;
```

NodeCounter3.vhd

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


entity NodeCounter3 is
   Port ( clk                   : in  STD_LOGIC;
```

```
        Start                           : in  STD_LOGIC;
        Reset                           : in  STD_LOGIC;
                        Ready           : out STD_LOGIC;
                        Done                    : out STD_LOGIC;
                        NodeNumber      : out  STD_LOGIC_VECTOR(11 downto
0));
end NodeCounter3;

architecture Behavioral of NodeCounter3 is


type state_type is (initial,count,wait1,wait2,finished,wait3);
signal state            : state_type;
signal next_state : state_type;
signal IncrNodeCount : STD_LOGIC;
signal IncrWaitCount : STD_LOGIC;
signal SetNodeCount0 : STD_LOGIC;
signal SetWaitCount0 : STD_LOGIC;
signal NodeCount  : STD_LOGIC_VECTOR(11 downto 0);
signal WaitCount  : STD_LOGIC_VECTOR(3 downto 0);
constant Latency : integer := 7;

begin

NodeNumber <= NodeCount;

  --Tells when the first output is available
        --Allows the driver to start data transfer with the available data
        --done tells when the process is finished
  Ready_Logic: process(clk,Reset)
        begin
                if(Reset='1') then
                        Ready <= '0';
                elsif(rising_edge(clk)) then
                  if(NodeCount > x"0a7") then
                    Ready <= '1';
                        else Ready <= '0';
                        end if;
                end if;
        end process;

        NodeCounter: process(clk,Reset)
        begin
                if(Reset='1') then
                        NodeCount <= (others => '0');
                elsif(rising_edge(clk)) then
```

```
                    if(IncrNodeCount = '1') then
                        NodeCount <= NodeCount + 1;
                    elsif(SetNodeCount0 = '1') then
                        NodeCount <= (others => '0');
                    end if;
                end if;
        end process;

        WaitCounter: process(clk,Reset)
        begin
                if(Reset='1') then
                        WaitCount <= (others => '0');
                elsif(rising_edge(clk)) then
                        if(IncrWaitCount = '1') then
                                WaitCount <= WaitCount + 1;
                        elsif(SetWaitCount0 = '1') then
                                WaitCount <= (others => '0');
                        end if;
                end if;
        end process;

        StateMemory: process(clk,Reset)
        begin
                if(Reset='1') then
                        state <= initial;
                elsif(rising_edge(clk)) then
                        state <= next_state;
                end if;
        end process;

        NextStateLogic: process(state,Start,NodeCount,WaitCount)
        begin
                case state is

                        when initial =>
                          if(Start = '1') then
                                next_state <= count;
                            else next_state <= initial;
                            end if;

                        when count =>
                                --in decimal  if(NodeCount = 38 or NodeCount = 88 or
NodeCount = 158) then
                                --  40-2=38             40+50-2=88             40+50+70-
2=158
```

```
                        if(NodeCount = x"026" or NodeCount = x"058" or NodeCount =
x"09e") then
                                next_state <= wait1;
                            --40+50+70+364-2=522
                            --elseif(NodeCount = 522) then
                            elsif(NodeCount = x"20a") then
                                    next_state <= wait2;
                            else next_state <= count;
                            end if;

                        when wait1 =>
                          if(conv_integer(WaitCount) = Latency-2) then
                                next_state <= count;
                            else next_state <= wait1;
                            end if;

                        when wait2 =>
                          if(conv_integer(WaitCount) = Latency-1) then
                                next_state <= finished;
                            else next_state <= wait2;
                            end if;

                        when finished =>
                          if(Start = '1') then
                                next_state <= finished;
                            else next_state <= wait3;
                            end if;

                        when wait3 =>
                          if(Start = '1') then
                                next_state <= count;
                            else next_state <= wait3;
                            end if;

                end case;
        end process;

        OutputLogic: process(state,Start,NodeCount,WaitCount)
        begin

                Done <= '0';
                IncrNodeCount <= '0';
                SetNodeCount0 <= '0';
                IncrWaitCount <= '0';
                SetWaitCount0 <= '0';
```

```
                if(state = initial) then
                        SetNodeCount0 <= '1';
                end if;

                if(state = count) then
                        SetWaitCount0 <= '1';
                        IncrNodeCount <= '1';
                end if;

                if(state = wait1) then
                        IncrWaitCount <= '1';
                end if;

                if(state = wait2) then
                        IncrWaitCount <= '1';
                        SetNodeCount0 <= '1';
                end if;

                if(state = finished) then
                        Done <= '1';
                end if;

                if(state = wait3) then
                        SetNodeCount0 <= '1';
                        Done <= '1';
                end if;

        end process;

end Behavioral;
```

APPENDIX D

Three Board XUP-V2P SATA Design EDK Configuration Files

system.mhs

```
##############################################################################
# Created by Base System Builder Wizard for Xilinx EDK 8.1.02 Build EDK_I.20.4
# Thu Aug 09 11:05:37 2007
# Target Board:  Xilinx XUP Virtex-II Pro Development System Rev C
# Family:        virtex2p
# Device:        xc2vp30
# Package:       ff896
# Speed Grade:          -7
# Processor: PPC 405
# Processor clock frequency: 100.000000 MHz
# Bus clock frequency: 100.000000 MHz
# Debug interface: FPGA JTAG
# On Chip Memory :  16 KB
# Total Off Chip Memory : 256 MB
# - DDR_SDRAM_32Mx64 Single Rank = 256 MB
#
##############################################################################


 PARAMETER VERSION = 2.1.0


 PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX, DIR = I
 PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX, DIR = O
 PORT                fpga_0_SysACE_CompactFlash_SysACE_CLK_pin                =
fpga_0_SysACE_CompactFlash_SysACE_CLK, DIR = I
 PORT                fpga_0_SysACE_CompactFlash_SysACE_MPA_pin                =
fpga_0_SysACE_CompactFlash_SysACE_MPA, DIR = O, VEC = [6:0]
 PORT                fpga_0_SysACE_CompactFlash_SysACE_MPD_pin                =
fpga_0_SysACE_CompactFlash_SysACE_MPD, DIR = IO, VEC = [15:0]
 PORT                fpga_0_SysACE_CompactFlash_SysACE_CEN_pin                =
fpga_0_SysACE_CompactFlash_SysACE_CEN, DIR = O
 PORT                fpga_0_SysACE_CompactFlash_SysACE_OEN_pin                =
fpga_0_SysACE_CompactFlash_SysACE_OEN, DIR = O
 PORT                fpga_0_SysACE_CompactFlash_SysACE_WEN_pin                =
fpga_0_SysACE_CompactFlash_SysACE_WEN, DIR = O
```

PORT fpga_0_SysACE_CompactFlash_SysACE_MPIRQ_pin = fpga_0_SysACE_CompactFlash_SysACE_MPIRQ, DIR = I
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clk_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clk, DIR = O, VEC = [0:2]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clkn_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clkn, DIR = O, VEC = [0:2]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Addr_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Addr, DIR = O, VEC = [0:12]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_BankAddr_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_BankAddr, DIR = O, VEC = [0:1]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CASn_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CASn, DIR = O
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_RASn_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_RASn, DIR = O
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_WEn_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_WEn, DIR = O
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DM_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DM, DIR = O, VEC = [0:7]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQS_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQS, DIR = IO, VEC = [0:7]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQ_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQ, DIR = IO, VEC = [0:63]
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CKE_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CKE, DIR = O
PORT fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CSn_pin = fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CSn, DIR = O
PORT fpga_0_net_gnd_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_1_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_2_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_3_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_4_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_5_pin = net_gnd, DIR = O
PORT fpga_0_net_gnd_6_pin = net_gnd, DIR = O
PORT fpga_0_DDR_CLK_FB = ddr_feedback_s, DIR = I, SIGIS = DCMCLK
PORT fpga_0_DDR_CLK_FB_OUT = ddr_clk_feedback_out_s, DIR = O
PORT sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = DCMCLK
PORT sys_rst_pin = sys_rst_s, DIR = I
PORT LED0 = PSO_NN_Calc3_0_PSO_Running_n, DIR = O

```
 PORT LED1 = PSO_NN_Calc3_0_NN_Running_n, DIR = O
 PORT LED2 = PSO_NN_Calc3_0_Aurora_Up1_n, DIR = O
 PORT LED3 = PSO_NN_Calc3_0_Aurora_Up2_n, DIR = O
 PORT MGT_TXP1 = PSO_NN_Calc3_0_TXP1, DIR = O
 PORT MGT_TXN1 = PSO_NN_Calc3_0_TXN1, DIR = O
 PORT MGT_RXP1 = PSO_NN_Calc3_0_RXP1, DIR = I
 PORT MGT_RXN1 = PSO_NN_Calc3_0_RXN1, DIR = I
 PORT MGT_TXP2 = PSO_NN_Calc3_0_TXP2, DIR = O
 PORT MGT_TXN2 = PSO_NN_Calc3_0_TXN2, DIR = O
 PORT MGT_RXP2 = PSO_NN_Calc3_0_RXP2, DIR = I
 PORT MGT_RXN2 = PSO_NN_Calc3_0_RXN2, DIR = I


BEGIN ppc405
 PARAMETER INSTANCE = ppc405_0
 PARAMETER HW_VER = 2.00.c
 BUS_INTERFACE JTAGPPC = jtagppc_0_0
 BUS_INTERFACE IPLB = plb
 BUS_INTERFACE DPLB = plb
 PORT PLBCLK = sys_clk_s
 PORT C405RSTCHIPRESETREQ = C405RSTCHIPRESETREQ
 PORT C405RSTCORERESETREQ = C405RSTCORERESETREQ
 PORT C405RSTSYSRESETREQ = C405RSTSYSRESETREQ
 PORT RSTC405RESETCHIP = RSTC405RESETCHIP
 PORT RSTC405RESETCORE = RSTC405RESETCORE
 PORT RSTC405RESETSYS = RSTC405RESETSYS
 PORT EICC405EXTINPUTIRQ = EICC405EXTINPUTIRQ
 PORT CPMC405CLOCK = sys_clk_s
END

BEGIN ppc405
 PARAMETER INSTANCE = ppc405_1
 PARAMETER HW_VER = 2.00.c
 BUS_INTERFACE JTAGPPC = jtagppc_0_1
END

BEGIN jtagppc_cntlr
 PARAMETER INSTANCE = jtagppc_0
 PARAMETER HW_VER = 2.00.a
 BUS_INTERFACE JTAGPPC0 = jtagppc_0_0
 BUS_INTERFACE JTAGPPC1 = jtagppc_0_1
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = reset_block
 PARAMETER HW_VER = 1.00.a
```

```
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT Ext_Reset_In = sys_rst_s
 PORT Slowest_sync_clk = sys_clk_s
 PORT Chip_Reset_Req = C405RSTCHIPRESETREQ
 PORT Core_Reset_Req = C405RSTCORERESETREQ
 PORT System_Reset_Req = C405RSTSYSRESETREQ
 PORT Rstc405resetchip = RSTC405RESETCHIP
 PORT Rstc405resetcore = RSTC405RESETCORE
 PORT Rstc405resetsys = RSTC405RESETSYS
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT Dcm_locked = dcm_1_lock
END

BEGIN plb_v34
 PARAMETER INSTANCE = plb
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT SYS_Rst = sys_bus_reset
 PORT PLB_Clk = sys_clk_s
END

BEGIN opb_v20
 PARAMETER INSTANCE = opb
 PARAMETER HW_VER = 1.10.c
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT SYS_Rst = sys_bus_reset
 PORT OPB_Clk = sys_clk_s
END

BEGIN plb2opb_bridge
 PARAMETER INSTANCE = plb2opb
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_NUM_ADDR_RNG = 1
 PARAMETER C_RNG0_BASEADDR = 0x40000000
 PARAMETER C_RNG0_HIGHADDR = 0x7fffffff
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE MOPB = opb
 PORT PLB_Clk = sys_clk_s
 PORT OPB_Clk = sys_clk_s
END

BEGIN PSO_NN_Calc3
 PARAMETER INSTANCE = PSO_NN_Calc3_0
 PARAMETER HW_VER = 1.00.a
```

```
 PARAMETER C_BASEADDR = 0x50000000
 PARAMETER C_HIGHADDR = 0x5000ffff
 PARAMETER C_AR0_BASEADDR = 0x51000000
 PARAMETER C_AR0_HIGHADDR = 0x5100ffff
 PARAMETER C_AR1_BASEADDR = 0x52000000
 PARAMETER C_AR1_HIGHADDR = 0x5200ffff
 PARAMETER C_AR2_BASEADDR = 0x53000000
 PARAMETER C_AR2_HIGHADDR = 0x5300ffff
 PARAMETER C_AR3_BASEADDR = 0x54000000
 PARAMETER C_AR3_HIGHADDR = 0x5400ffff
 PARAMETER C_AR4_BASEADDR = 0x55000000
 PARAMETER C_AR4_HIGHADDR = 0x5500ffff
 PARAMETER C_AR5_BASEADDR = 0x56000000
 PARAMETER C_AR5_HIGHADDR = 0x5600ffff
 PARAMETER C_AR6_BASEADDR = 0x57000000
 PARAMETER C_AR6_HIGHADDR = 0x5700ffff
 PARAMETER C_AR7_BASEADDR = 0x58000000
 PARAMETER C_AR7_HIGHADDR = 0x5800ffff
 BUS_INTERFACE SOPB = opb
 PORT PSO_Running_n = PSO_NN_Calc3_0_PSO_Running_n
 PORT NN_Running_n = PSO_NN_Calc3_0_NN_Running_n
 PORT Aurora_Up1_n = PSO_NN_Calc3_0_Aurora_Up1_n
 PORT Aurora_Up2_n = PSO_NN_Calc3_0_Aurora_Up2_n
 PORT RXN1 = PSO_NN_Calc3_0_RXN1
 PORT RXP1 = PSO_NN_Calc3_0_RXP1
 PORT TXN1 = PSO_NN_Calc3_0_TXN1
 PORT TXP1 = PSO_NN_Calc3_0_TXP1
 PORT RXN2 = PSO_NN_Calc3_0_RXN2
 PORT RXP2 = PSO_NN_Calc3_0_RXP2
 PORT TXN2 = PSO_NN_Calc3_0_TXN2
 PORT TXP2 = PSO_NN_Calc3_0_TXP2
END

BEGIN opb_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_ODD_PARITY = 0
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_CLK_FREQ = 100000000
 PARAMETER C_BASEADDR = 0x40600000
 PARAMETER C_HIGHADDR = 0x4060ffff
 BUS_INTERFACE SOPB = opb
 PORT OPB_Clk = sys_clk_s
 PORT Interrupt = RS232_Uart_1_Interrupt
```

```
 PORT RX = fpga_0_RS232_Uart_1_RX
 PORT TX = fpga_0_RS232_Uart_1_TX
END

BEGIN opb_sysace
 PARAMETER INSTANCE = SysACE_CompactFlash
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_MEM_WIDTH = 16
 PARAMETER C_BASEADDR = 0x41800000
 PARAMETER C_HIGHADDR = 0x4180ffff
 BUS_INTERFACE SOPB = opb
 PORT OPB_Clk = sys_clk_s
 PORT SysACE_CLK = fpga_0_SysACE_CompactFlash_SysACE_CLK
 PORT SysACE_MPA = fpga_0_SysACE_CompactFlash_SysACE_MPA
 PORT SysACE_MPD = fpga_0_SysACE_CompactFlash_SysACE_MPD
 PORT SysACE_CEN = fpga_0_SysACE_CompactFlash_SysACE_CEN
 PORT SysACE_OEN = fpga_0_SysACE_CompactFlash_SysACE_OEN
 PORT SysACE_WEN = fpga_0_SysACE_CompactFlash_SysACE_WEN
 PORT SysACE_MPIRQ = fpga_0_SysACE_CompactFlash_SysACE_MPIRQ
END

BEGIN plb_ddr
 PARAMETER INSTANCE = DDR_256MB_32MX64_rank1_row13_col10_cl2_5
 PARAMETER HW_VER = 1.11.a
 PARAMETER C_PLB_CLK_PERIOD_PS = 10000
 PARAMETER C_NUM_BANKS_MEM = 1
 PARAMETER C_NUM_CLK_PAIRS = 4
 PARAMETER C_INCLUDE_BURST_CACHELN_SUPPORT = 1
 PARAMETER C_REG_DIMM = 0
 PARAMETER C_DDR_TMRD = 20000
 PARAMETER C_DDR_TWR = 20000
 PARAMETER C_DDR_TRAS = 60000
 PARAMETER C_DDR_TRC = 90000
 PARAMETER C_DDR_TRFC = 100000
 PARAMETER C_DDR_TRCD = 30000
 PARAMETER C_DDR_TRRD = 20000
 PARAMETER C_DDR_TRP = 30000
 PARAMETER C_DDR_TREFC = 70300000
 PARAMETER C_DDR_AWIDTH = 13
 PARAMETER C_DDR_COL_AWIDTH = 10
 PARAMETER C_DDR_BANK_AWIDTH = 2
 PARAMETER C_DDR_DWIDTH = 64
 PARAMETER C_MEM0_BASEADDR = 0x00000000
 PARAMETER C_MEM0_HIGHADDR = 0x0fffffff
 BUS_INTERFACE SPLB = plb
 PORT PLB_Clk = sys_clk_s
```

```
 PORT                                    DDR_Addr                                =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Addr
 PORT                                    DDR_BankAddr                            =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_BankAddr
 PORT                                    DDR_CASn                                =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CASn
 PORT                                    DDR_CKE                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CKE
 PORT                                    DDR_CSn                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_CSn
 PORT                                    DDR_RASn                                =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_RASn
 PORT                                    DDR_WEn                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_WEn
 PORT                                    DDR_DM                                  =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DM
 PORT                                    DDR_DQS                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQS
 PORT                                    DDR_DQ                                  =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_DQ
 PORT                                    DDR_Clk                                 =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clk                 &
ddr_clk_feedback_out_s
 PORT                                    DDR_Clkn                                =
fpga_0_DDR_256MB_32MX64_rank1_row13_col10_cl2_5_DDR_Clkn & 0b0
 PORT Clk90_in = clk_90_s
 PORT Clk90_in_n = clk_90_n_s
 PORT PLB_Clk_n = sys_clk_n_s
 PORT DDR_Clk90_in = ddr_clk_90_s
 PORT DDR_Clk90_in_n = ddr_clk_90_n_s
END

BEGIN plb_bram_if_cntlr
 PARAMETER INSTANCE = plb_bram_if_cntlr_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER c_plb_clk_period_ps = 10000
 PARAMETER c_baseaddr = 0xffffc000
 PARAMETER c_highaddr = 0xffffffff
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE PORTA = plb_bram_if_cntlr_1_port
 PORT PLB_Clk = sys_clk_s
END

BEGIN bram_block
 PARAMETER INSTANCE = plb_bram_if_cntlr_1_bram
 PARAMETER HW_VER = 1.00.a
```

```
 BUS_INTERFACE PORTA = plb_bram_if_cntlr_1_port
END

BEGIN opb_intc
 PARAMETER INSTANCE = opb_intc_0
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_BASEADDR = 0x41200000
 PARAMETER C_HIGHADDR = 0x4120ffff
 BUS_INTERFACE SOPB = opb
 PORT Irq = EICC405EXTINPUTIRQ
 PORT Intr = RS232_Uart_1_Interrupt
END

BEGIN util_vector_logic
 PARAMETER INSTANCE = sysclk_inv
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_SIZE = 1
 PARAMETER C_OPERATION = not
 PORT Op1 = sys_clk_s
 PORT Res = sys_clk_n_s
END

BEGIN util_vector_logic
 PARAMETER INSTANCE = clk90_inv
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_SIZE = 1
 PARAMETER C_OPERATION = not
 PORT Op1 = clk_90_s
 PORT Res = clk_90_n_s
END

BEGIN util_vector_logic
 PARAMETER INSTANCE = ddr_clk90_inv
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_SIZE = 1
 PARAMETER C_OPERATION = not
 PORT Op1 = ddr_clk_90_s
 PORT Res = ddr_clk_90_n_s
END

BEGIN dcm_module
 PARAMETER INSTANCE = dcm_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_CLK0_BUF = TRUE
 PARAMETER C_CLK90_BUF = TRUE
 PARAMETER C_CLKIN_PERIOD = 10.000000
```

```
 PARAMETER C_CLK_FEEDBACK = 1X
 PARAMETER C_DLL_FREQUENCY_MODE = LOW
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT CLKIN = dcm_clk_s
 PORT CLK0 = sys_clk_s
 PORT CLK90 = clk_90_s
 PORT CLKFB = sys_clk_s
 PORT RST = net_gnd
 PORT LOCKED = dcm_0_lock
END

BEGIN dcm_module
 PARAMETER INSTANCE = dcm_1
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_CLK0_BUF = TRUE
 PARAMETER C_CLK90_BUF = TRUE
 PARAMETER C_CLKIN_PERIOD = 10.000000
 PARAMETER C_CLK_FEEDBACK = 1X
 PARAMETER C_DLL_FREQUENCY_MODE = LOW
 PARAMETER C_PHASE_SHIFT = 60
 PARAMETER C_CLKOUT_PHASE_SHIFT = FIXED
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT CLKIN = ddr_feedback_s
 PORT CLK90 = ddr_clk_90_s
 PORT CLK0 = dcm_1_FB
 PORT CLKFB = dcm_1_FB
 PORT RST = dcm_0_lock
 PORT LOCKED = dcm_1_lock
END

BEGIN opb_timer
 PARAMETER INSTANCE = opb_timer_0
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x60000000
 PARAMETER C_HIGHADDR = 0x6000ffff
 BUS_INTERFACE SOPB = opb
END
```

system.mss

```
 PARAMETER VERSION = 2.2.0


BEGIN OS
 PARAMETER OS_NAME = standalone
```

```
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_0
 PARAMETER STDIN = RS232_Uart_1
 PARAMETER STDOUT = RS232_Uart_1
END

BEGIN OS
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_1
END


BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu_ppc405
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc405_0
 PARAMETER COMPILER = powerpc-eabi-gcc
 PARAMETER ARCHIVER = powerpc-eabi-ar
 PARAMETER CORE_CLOCK_FREQ_HZ = 100000000
END

BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu_ppc405
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc405_1
 PARAMETER COMPILER = powerpc-eabi-gcc
 PARAMETER ARCHIVER = powerpc-eabi-ar
END


BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = jtagppc_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = plbarb
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = plb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = opbarb
 PARAMETER DRIVER_VER = 1.02.a
```

```
 PARAMETER HW_INSTANCE = opb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = plb2opb
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = plb2opb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = RS232_Uart_1
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = sysace
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = SysACE_CompactFlash
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = ddr
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER HW_INSTANCE = DDR_256MB_32MX64_rank1_row13_col10_cl2_5
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = plb_bram_if_cntlr_1
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = intc
 PARAMETER DRIVER_VER = 1.00.c
 PARAMETER HW_INSTANCE = opb_intc_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = PSO_NN_Calc3
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = PSO_NN_Calc3_0
END

BEGIN DRIVER
```

```
 PARAMETER DRIVER_NAME = tmrctr
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER HW_INSTANCE = opb_timer_0
END


BEGIN LIBRARY
 PARAMETER LIBRARY_NAME = xilfatfs
 PARAMETER LIBRARY_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_0
 PARAMETER CONFIG_BUFCACHE_SIZE = 20480
 PARAMETER CONFIG_DIR_SUPPORT = true
 PARAMETER CONFIG_WRITE = true
END
```

APPENDIX E

XUP-V2P EDK User Interface Code

main.c

```
#include "xparameters.h"
#include <sysace_stdio.h>
#include <xuartlite_l.h>
#include "uart.h"
#include "stdio.h"
#include "math.h"


//====================================================

void Insert_Inputs_Menu(void);
void Run_NN(void);
void View_Outputs_Menu(void);
void get27inputs(short int *data,char *filename);
void WriteOutputs(int *databaseaddr,char *filename);
void SetBitmap(void);
void Initialize_PSO(unsigned int num_agents,unsigned int num_iterations);
void Run_PSO();
void View_PSO();

int main (void) {

unsigned char menu_input;


        uartInit(STDOUT_BASEADDRESS);
        clearScreen();

        do{
                clearScreen();
                SetBitmap();
                printf("Select one of the Following Options\n\n\r");
                printf("1 Insert Inputs\n\r");
                printf("2 Run Sonar NN\n\r");
                printf("3 View Outputs\n\r");
                printf("4 Initialize PSO\n\r");
```

```
		printf("5 Run PSO\n\r");
		printf("6 View PSO Results\n\r");
		printf("q Quit\n\n\r");
		printf("Type Response ");
		do{
			menu_input = uartRead();
		}while(menu_input != '1' && menu_input!= '2' && menu_input!= '3' &&
menu_input!= '4' && menu_input!= '5' && menu_input!= '6' && menu_input !='q');

		switch (menu_input) {
			case '1' :
				Insert_Inputs_Menu();
				break;
			case '2' :
				Run_NN();
				break;
			case '3' :
				View_Outputs_Menu();
				break;
			case '4' :
				Initialize_PSO(100,1000);
				SetBitmap();
				break;
			case '5' :
				Run_PSO();
				break;
			case '6' :
				View_PSO();
				break;
		}
	}while(menu_input != 'q');

  printf("Goodbye");
  return 0;
}

void Insert_Inputs_Menu(void)
{
	unsigned char NodeNum;
	char input_file[] = "a:\\inputs1.txt";
	unsigned short int inputs[32]={0};
	int count;
	unsigned int temp;
	unsigned int* Pointer;
	unsigned char Decision = '0';
```

```
	unsigned int* Control_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_BASEADDR;
	unsigned int* Input_Pointer = (unsigned int
*)XPAR_PSO_NN_CALC2_0_AR0_BASEADDR;
	unsigned int* Output_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_AR1_BASEADDR;
	unsigned int* PSO_Result_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_AR3_BASEADDR;



	do{
		clearScreen();
		SetBitmap();
		print("Make a Selection from the Menu\n\n\r");
		print("1 Use Default Values\n\r");
		print("2 Set Values from File\n\r");
		print("3 Set Values from PSO Result\n\r");
		print("q Return to Main Menu\n\n\r");

		do{
			Decision = uartRead();
		}while(Decision!='1' && Decision!='2' && Decision!='3'&&
Decision!='q');

		switch (Decision)
		{
			case '1' :
				clearScreen();
				Pointer = Input_Pointer;
				*Pointer = 0x00ea00a8;
				Pointer++;
				*Pointer = 0x20aa005c;
				Pointer++;
				*Pointer = 0xc9500234;
				Pointer++;
				*Pointer = 0x4f36021a;
				Pointer++;
				*Pointer = 0x01d20bff;
				Pointer++;
				*Pointer = 0x0bff0bfe;
				Pointer++;
				*Pointer = 0x0bff0c02;
				Pointer++;
				*Pointer = 0x0c020c00;
				Pointer++;
```

```
                *Pointer = 0x0bfe0bf8;
                Pointer++;
                *Pointer = 0x0bf50bed;
                Pointer++;
                *Pointer = 0x0beb0be4;
                Pointer++;
                *Pointer = 0x0bd80bd1;
                Pointer++;
                *Pointer = 0x0bc50bc1;
                Pointer++;
                *Pointer = 0x0bbe0000;
                xil_printf("Finished Inserting Default Values\n\r");
                xil_printf("Press any key to continue\n\r");
                uartRead();
                break;

        case '2' :
                clearScreen();
                xil_printf("Press 'c' to change the filename or any other key
to proceed\n\r");
                do{
                  xil_printf("Current filename is %s\r",input_file);
                        Decision = uartRead();
                        if(Decision == 'c' && input_file[9] != '9')
                          input_file[9]++;
                        else if(input_file[9] == '9')
                          input_file[9] = '0';
                }while(Decision=='c');
                Pointer = Input_Pointer;
                xil_printf("Getting Inputs from Compact Flash with file
name %s\n\r",input_file);
                get27inputs(inputs,input_file);
                for(count=0;count<14;count++){
                        temp = inputs[2*count];
                        temp <<= 16;
                        *Pointer = temp | inputs[2*count+1];
                        putnum(*Pointer);
                        xil_printf("\n\r");
                        Pointer++;
                }
                xil_printf("Press any key to continue\n\r");
                uartRead();
                break;

        case '3' :
                clearScreen();
```

```
                                for(count=0;count<14;count++)
                                        *(Input_Pointer + count) =
*(PSO_Result_Pointer+count);
                                xil_printf("Finished Inserting PSO Result Values\n\r");
                                xil_printf("Press any key to continue\n\r");
                                uartRead();
                                break;
                        }

        }while(Decision!= 'q');

}

void Run_NN(void)
{
        unsigned int* Control_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_BASEADDR;
        unsigned int* CntTmr_Pointer = (unsigned int *)
XPAR_OPB_TIMER_0_BASEADDR;
        unsigned int elapsed_clks;
        float elapsed_time;

        clearScreen();
        printf("Running Neural Network\n\r");
        *(CntTmr_Pointer+1)=0;//Reset Value for timer
        *(CntTmr_Pointer)=0x00000020;//Reset
        *(CntTmr_Pointer)=0x00000080;//Start Counter
        *Control_Pointer = 0x01;//Start NN
        *Control_Pointer = 0x00;//NN is still running until finished but start signal is set
low

        while(*(Control_Pointer+1) != 1 && *(Control_Pointer+1) != 3)
        {
                //loop until NN is finished
        }
        *(CntTmr_Pointer)=0x00000000;//Stop Counter
        elapsed_clks = *(CntTmr_Pointer+2);
        elapsed_time = (float)elapsed_clks;
        elapsed_time /= 100;
        printf("Neural Network Finished\n\n\r");
        printf("Number of clocks for the calculation is %d\n\r",elapsed_clks);
        printf("With a Clock Frequency of 100MHz, the total elapsed time for
the\n\rcalculation is %f us\n\r",elapsed_time);
        printf("Press any key to continue\n\r");
        uartRead();
}
```

```
void View_Outputs_Menu(void)
{
	unsigned int* Control_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_BASEADDR;
	unsigned int* Input_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_AR0_BASEADDR;
	unsigned int* Output_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_AR1_BASEADDR;

	int NodeNum = 0;
	unsigned int raw_value;
	unsigned int* NodeNum_Location = Output_Pointer;
	unsigned char Decision = '0';
	char output_file[] = "a:\\NN_Outs1.txt";

	int Value1,Value2,Value3;
	float OutValue1,OutValue2;
	int count = 0;


	do{
		clearScreen();
		printf("Select the Desired Output Node to View\n\n\r");
		printf("a Go Back 100 Nodes\n\r");
		printf("s Go Back 10 Node\n\r");
		printf("d Move forward 10 Node\n\r");
		printf("f Move forward 100 Nodes\n\r");
		printf("w Write Outputs to Compact Flash\n\r");
		printf("q Return to Main Menu\n\n\r");

		printf(" Node   Fixed Point   Actual\n");
		printf("Number     Value      Value\n");

		for(count = 0;count<5;count++)
		{
			raw_value = * NodeNum_Location;

			Value1 = raw_value & 0xffff0000;
			Value1 >>= 16;
			Value1 &= 0x0000ffff;
			printf("  %d        %X",NodeNum*2,Value1);
			Value1 <<= 16; // sign extend
			Value1 >>= 20;
			printf("      %d\n",Value1);

			Value2 = raw_value & 0x0000ffff;
```

```
                printf("  %d        %X",NodeNum*2+1,Value2);
                Value2 <<= 16; //sign extend
                Value2 >>= 20;
                printf("      %d\n",Value2);

                NodeNum_Location += 1;
                NodeNum += 1;
        }

        do{
                Decision = uartRead();
        }while(Decision!='a' && Decision!='s' && Decision!='d' &&
Decision!='f' && Decision!='w' && Decision!='q');

        switch (Decision)
        {
                case 'a' :
                        NodeNum -= 55;
                        NodeNum_Location -= 55;
                        break;
                case 's' :
                        NodeNum -= 10;
                        NodeNum_Location -= 10;
                        break;
                case 'd' :
                        NodeNum += 0;
                        NodeNum_Location += 0;
                        break;
                case 'f' :
                        NodeNum += 45;
                        NodeNum_Location += 45;
                        break;
                case 'w' :
                        clearScreen();
                        xil_printf("Press 'c' to change the filename or any other key
to proceed\n\r");
                        do{
                          xil_printf("Current filename is %s\r",output_file);
                                Decision = uartRead();
                                if(Decision == 'c' && output_file[10] != '9')
                                  output_file[10]++;
                                else if(output_file[10] == '9')
                                  output_file[10] = '0';
                        }while(Decision=='c');
                        xil_printf("Writting Outputs to Compact Flash with file
name %s\n\r",output_file);
```

```
                                WriteOutputs(Output_Pointer,output_file);
                                NodeNum -= 5;
                                NodeNum_Location -= 5;
                                xil_printf("Press any key to continue\n\r");
                                uartRead();
                                break;
                        }
        }while(Decision!= 'q');
}


void View_PSO()
{
        unsigned int* Global_Result_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_AR3_BASEADDR;

        int NodeNum = 0;
        unsigned int raw_value, raw_value2;
        unsigned int* NodeNum_Location = Global_Result_Pointer;
        unsigned int* NodeNum_Location2 = Global_Result_Pointer+7;
        unsigned char Decision = '0';

        int Value1,Value2,Value3,Value4;
        float OutValue1,OutValue2,OutValue3,OutValue4;
        int count = 0;

        clearScreen();
        printf(" Node   Fixed Point   Actual      Node   Fixed Point   Actual\n\r");
        printf("Number     Value      Value       Number     Value      Value\n\r");

        for(count = 0;count<7;count++)
        {
                raw_value = *NodeNum_Location;
                raw_value2 = *NodeNum_Location2;

                Value1 = raw_value & 0xffff0000;
                Value1 >>= 16;
                Value1 &= 0x0000ffff;
                printf("  %d         %X",NodeNum*2,Value1);
                Value1 <<= 16; // sign extend
                Value1 >>= 17;
                printf("      %d",Value1);

                Value3 = raw_value2 & 0xffff0000;
                Value3 >>= 16;
                Value3 &= 0x0000ffff;
```

```
                printf("       %d        %X",NodeNum*2+14,Value3);
                Value3 <<= 16; // sign extend
                Value3 >>= 17;
                printf("     %d\n",Value3);

                Value2 = raw_value & 0x0000ffff;
                printf("  %d        %X",NodeNum*2+1,Value2);
                Value2 <<= 16; //sign extend
                Value2 >>= 17;
                printf("     %d",Value2);

                Value4 = raw_value2 & 0x0000ffff;
                printf("       %d        %X",NodeNum*2+15,Value4);
                Value4 <<= 16; //sign extend
                Value4 >>= 17;
                printf("     %d\n",Value4);

                NodeNum_Location += 1;
                NodeNum_Location2 += 1;
                NodeNum += 1;
        }
        uartRead();
}


void Run_PSO()
{
        unsigned int* Control_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_BASEADDR;
        unsigned int* CntTmr_Pointer = (unsigned int *)
XPAR_OPB_TIMER_0_BASEADDR;
        unsigned int elapsed_clks;
        float elapsed_time;
        int fitness_value;

        clearScreen();
        print("Running Particle Swarm Optimization\n\r");
        printf("Number of Agents = %d\n\r",*(Control_Pointer+2));
        printf("Number of Iterations = %d\n\n\r",*(Control_Pointer+3));
        *(CntTmr_Pointer+1)=0;//Reset Value
        *(CntTmr_Pointer)=0x00000020;//Reset
        *(CntTmr_Pointer)=0x00000080;//Start Counter
        *Control_Pointer = 0x02;//Start
        *Control_Pointer = 0x00;//NN is still running until finished but start signal is set
low
```

```
	while(*(Control_Pointer+1) != 3)
	{
		//loop until PSO is finished
	}

	*(CntTmr_Pointer)=0x00000000;//Stop Counter
	elapsed_clks = *(CntTmr_Pointer+2);
	elapsed_time = (float)elapsed_clks;
	elapsed_time /= 100000000;
	fitness_value = *(Control_Pointer+7);
	printf("Fitness Number = %d\n\n\r",fitness_value);
	printf("Number of Clocks for Calculation = %d\n\r",elapsed_clks);
	printf("With a Clock Frequency of 100MHz, the total elapsed time for
the\n\rcalculation is %f seconds\n\r",elapsed_time);
	printf("Press any key to continue\n\r");
	uartRead();
}


void Initialize_PSO(unsigned int num_agents,unsigned int num_iterations)
{
	unsigned int* Control_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_BASEADDR;
	unsigned int* Agent_Position_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_AR4_BASEADDR;
	unsigned int* Agent_Velocity_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_AR5_BASEADDR;
	unsigned int* MaxMinPosition_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_AR6_BASEADDR;
	unsigned int* MaxVelocity_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_AR7_BASEADDR;
	unsigned int* Pointer;
	char positionMax[] = "a:\\XmaxFP.txt";
	char positionMin[] = "a:\\XminFP.txt";
	char velocityMax[] = "a:\\VmaxFP.txt";
	unsigned short int xmax[32]={0};
	unsigned short int xmin[32]={0};
	unsigned short int vmax[32]={0};
	unsigned int value1, value2;
	unsigned int count,count2;
	float randval1,randval2;

	clearScreen();
	print("Starting PSO Initialization\n\r");

	//Reset
```

```
	*(Control_Pointer)=4;
	*(Control_Pointer)=0;

	get27inputs(xmax,positionMax);
	get27inputs(xmin,positionMin);
	get27inputs(vmax,velocityMax);

	//Initialize the agents position
	Pointer = Agent_Position_Pointer;
	for(count=0;count<num_agents;count++)
		for(count2=0;count2<16;count2++){
			randval1=(float)rand()/((float)0x7fffffff);
			randval2=(float)rand()/((float)0x7fffffff);
			value1=xmin[2*count2]+(int)(randval1*(xmax[2*count2]-
xmin[2*count2]));
			value2=xmin[2*count2+1]+(int)(randval2*(xmax[2*count2+1]-
xmin[2*count2+1]));
			value1 <<= 16;
			value2 &= 0x0000ffff;
			*Pointer = value1 | value2;
			Pointer++;
			//agents[count*3*16+count2] = value1 | value2;//Initialize Current
Position
			//agents[count*3*16+count2+32] = value1 | value2;//Initialize Best
Local Position
		}

	//Initialize the agents velocity
	Pointer = Agent_Velocity_Pointer;
	for(count=0;count<num_agents;count++)
		for(count2=0;count2<16;count2++){
			randval1=(float)rand()/((float)0x7fffffff);
			randval2=(float)rand()/((float)0x7fffffff);
			value1= -vmax[2*count2]+(int)(randval1*2*vmax[2*count2]);
			value2= -
vmax[2*count2+1]+(int)(randval2*2*vmax[2*count2+1]);
			value1 <<= 16;
			value2 &= 0x0000ffff;
			*Pointer = value1 | value2;
			Pointer++;
			//agents[count*3*16+count2+16] = value1 | value2;//Initialize
Current Velocity
		}


	//Set the number of agents and iterations
```

```
	*(Control_Pointer+2) = num_iterations;
	*(Control_Pointer+3) = num_agents;

	//Set PSO Constants
	//Use last 16bits S.XXX XXXX XXXX XXXX
//	*(Control_Pointer+4) = 0x00006000;//CGlobal=75%
//	*(Control_Pointer+5) = 0x00002000;//CLocal=25%
//	*(Control_Pointer+4) = 0x00004000;//CGlobal=50%
//	*(Control_Pointer+5) = 0x00004000;//CLocal=50%
	*(Control_Pointer+4) = 0x00002000;//CGlobal=25%
	*(Control_Pointer+5) = 0x00006000;//CLocal=75%

	//Set the max and min positions in the PSO
	Pointer = MaxMinPosition_Pointer;
	for(count=0;count<27;count++){
		value1 = xmax[count];
		value1 <<= 16;
		value2 = xmin[count];
		value2 &= 0x0000ffff;
		*Pointer = value1 | value2;
		Pointer++;
	}

	//Set the max velocity in the PSO
	Pointer = MaxVelocity_Pointer;
	for(count=0;count<14;count++){
		value1 = vmax[2*count];
		value1 <<= 16;
		value2 = vmax[2*count+1];
		value2 &= 0x0000ffff;
		*Pointer = value1 | value2;
		Pointer++;
	}

	print("PSO Initialization Finished\n\r");
	print("Press any key to continue\n\r");
	uartRead();
}

void SetBitmap()
{
	unsigned int* Bitmap_Pointer = (unsigned int *)
XPAR_PSO_NN_CALC2_0_AR2_BASEADDR;


	*(Bitmap_Pointer)   = 0x00000000;
```

```
	//optimizes columns 6, 7, and 8
	*(Bitmap_Pointer+12) = 0x0000ffff;
	*(Bitmap_Pointer+13) = 0xffffffff;
	*(Bitmap_Pointer+14) = 0xffffffff;
	*(Bitmap_Pointer+15) = 0xffffffff;
	*(Bitmap_Pointer+16) = 0xffffffff;
	*(Bitmap_Pointer+17) = 0xffffffff;
	*(Bitmap_Pointer+18) = 0xffffffff;
	*(Bitmap_Pointer+19) = 0xffffffff;

	*(Bitmap_Pointer+37)= 0x00000000;
}


void get27inputs(short int *data,char *filename)
{
	int NUM_INPUTS = 27;
	unsigned char databuffer[170];
	SYSACE_FILE *infile;
	unsigned short int datatemp;
	int count1, count2, numread;

	infile = sysace_fopen(filename, "r");

	if(infile != 0){
		//xil_printf("Reading file : %s\n\r", filename);

		numread = sysace_fread(databuffer, 1, 6*NUM_INPUTS, infile);
		if(numread == 6*NUM_INPUTS){
			for(count1=0;count1<NUM_INPUTS;count1++){
				datatemp=0;
				for(count2=0;count2<4;count2++){
					if(databuffer[6*count1+count2] >= '0' &&
databuffer[6*count1+count2] <= '9')
						databuffer[6*count1+count2] -= 48;
					else if(databuffer[6*count1+count2] >= 'a' &&
databuffer[6*count1+count2] <= 'f')
						databuffer[6*count1+count2] -= 87;
					else databuffer[6*count1+count2] = 0;

					datatemp <<= 4;
					datatemp |= (short int)databuffer[6*count1+count2];
				}
				data[count1]=datatemp;
			}
		}
```

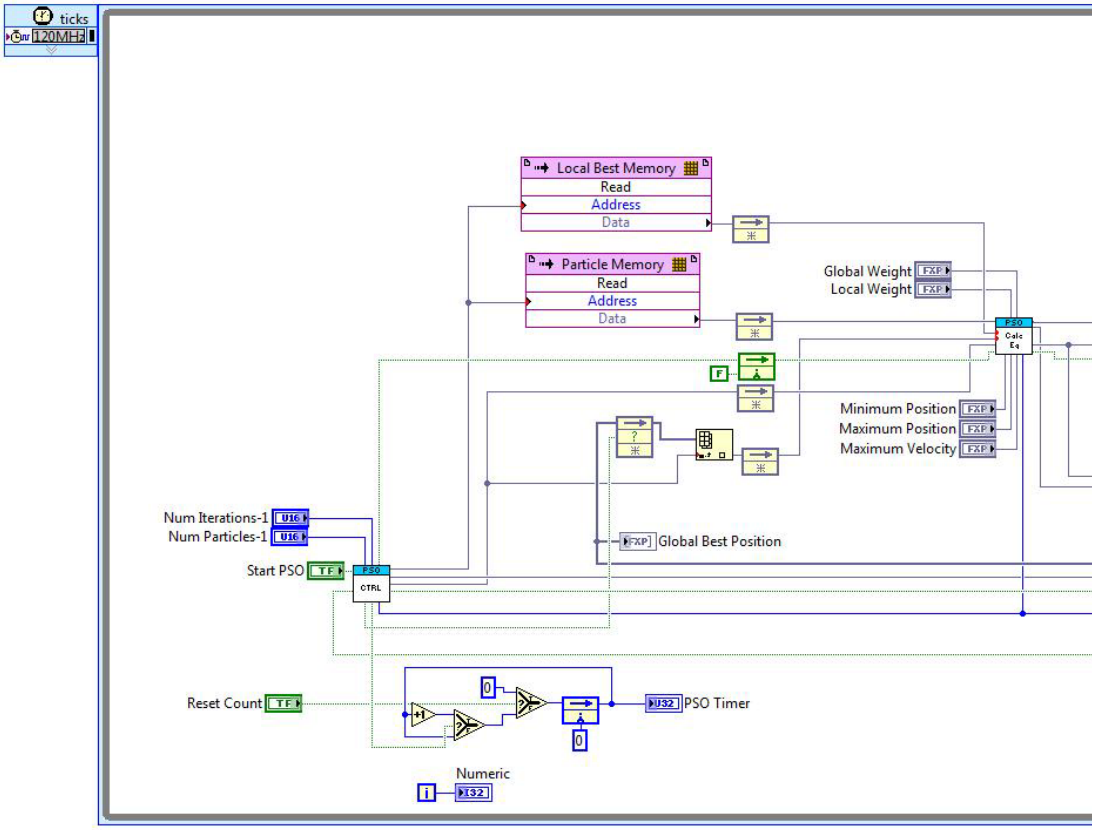
```
                else xil_printf("There are not a correct number of entries in the file
%s",filename);
        }
        else
                xil_printf("Error Openning file : %s\n\r", filename);
        sysace_fclose(infile);

}

void WriteOutputs(int *databaseaddr,char *filename)
{
        int NUM_OUTPUTS = 1200;
        unsigned char databuffer[10];
        SYSACE_FILE *outfile;
        int count1, count2, success;
        unsigned int* NodeNum_Location = databaseaddr;
        unsigned int value;
        unsigned int temp;
        unsigned int andvalue;

        outfile = sysace_fopen(filename, "w");

        if(outfile != 0){
                xil_printf("Writting file : %s\n\r", filename);

                for(count1=0;count1<600;count1++){

                        value = *(NodeNum_Location+count1);

                        temp = value >> 28;
                        temp &= 0x0000000f;
                        databuffer[0] = (char) temp;

                        temp = value >> 24;
                        temp &= 0x0000000f;
                        databuffer[1] = (char) temp;

                        temp = value >> 20;
                        temp &= 0x0000000f;
                        databuffer[2] = (char) temp;

                        temp = value >> 16;
                        temp &= 0x0000000f;
                        databuffer[3] = (char) temp;

                        databuffer[4] = '\n';
```

```
                        temp = value >> 12;
                        temp &= 0x0000000f;
                        databuffer[5] = (char) temp;

                        temp = value >> 8;
                        temp &= 0x0000000f;
                        databuffer[6] = (char) temp;

                        temp = value >> 4;
                        temp &= 0x0000000f;
                        databuffer[7] = (char) temp;

                        temp = value >> 0;
                        temp &= 0x0000000f;
                        databuffer[8] = (char) temp;

                        databuffer[9] = '\n';

                        for(count2=0;count2<4;count2++){
                                if(databuffer[count2] >= 0x00 && databuffer[count2] <=
0x09)
                                        databuffer[count2] += 48;
                                else if(databuffer[count2] >= 0x0a && databuffer[count2]
<= 0x0f)
                                        databuffer[count2] += 87;
                        }
                        for(count2=5;count2<9;count2++){
                                if(databuffer[count2] >= 0x00 && databuffer[count2] <=
0x09)
                                        databuffer[count2] += 48;
                                else if(databuffer[count2] >= 0x0a && databuffer[count2]
<= 0x0f)
                                        databuffer[count2] += 87;
                        }
                        success = sysace_fwrite(databuffer, 1, 10, outfile);
                }
                xil_printf("Finished Writting\n\r");
        }
        else
                xil_printf("Error Openning file : %s\n\r", filename);
        sysace_fclose(outfile);     }
```

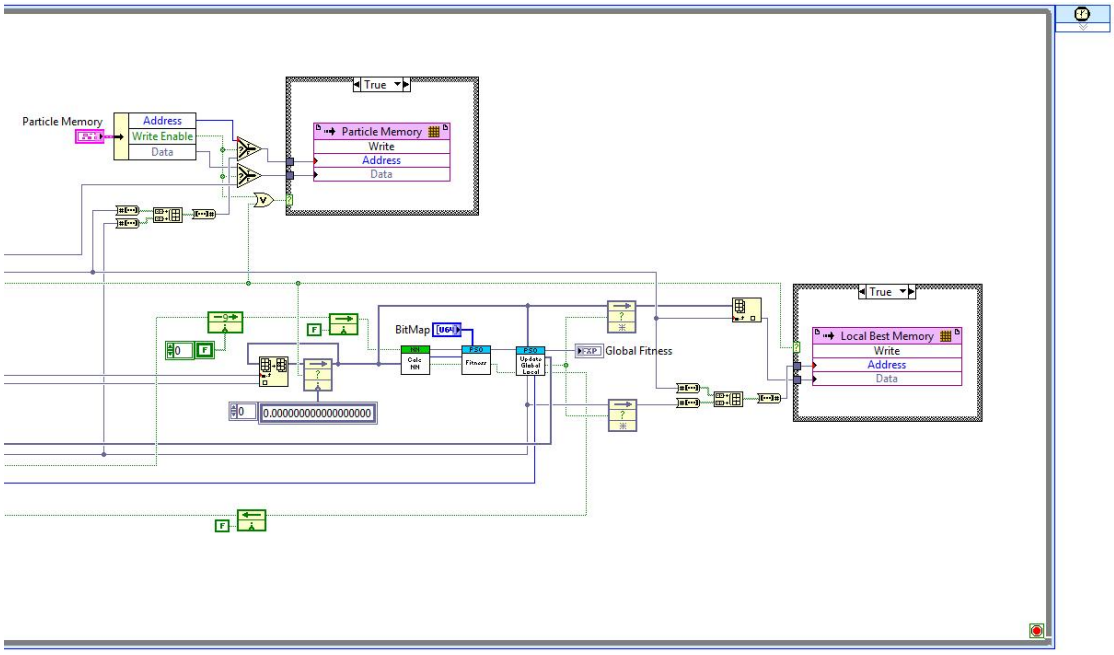
APPENDIX F

Single Board PXIe LabVIEW FPGA Code

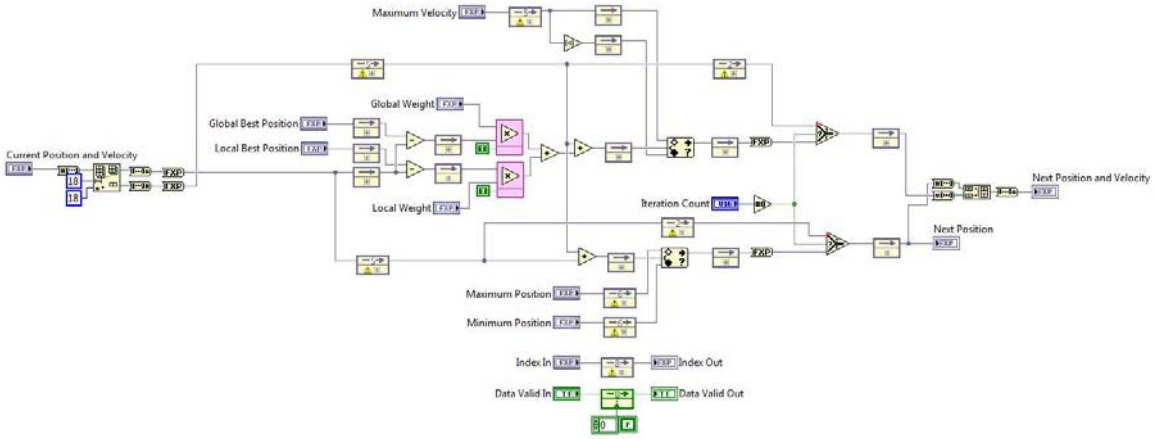
FPGA Top.vi



F.1: FPGA Top (Left)

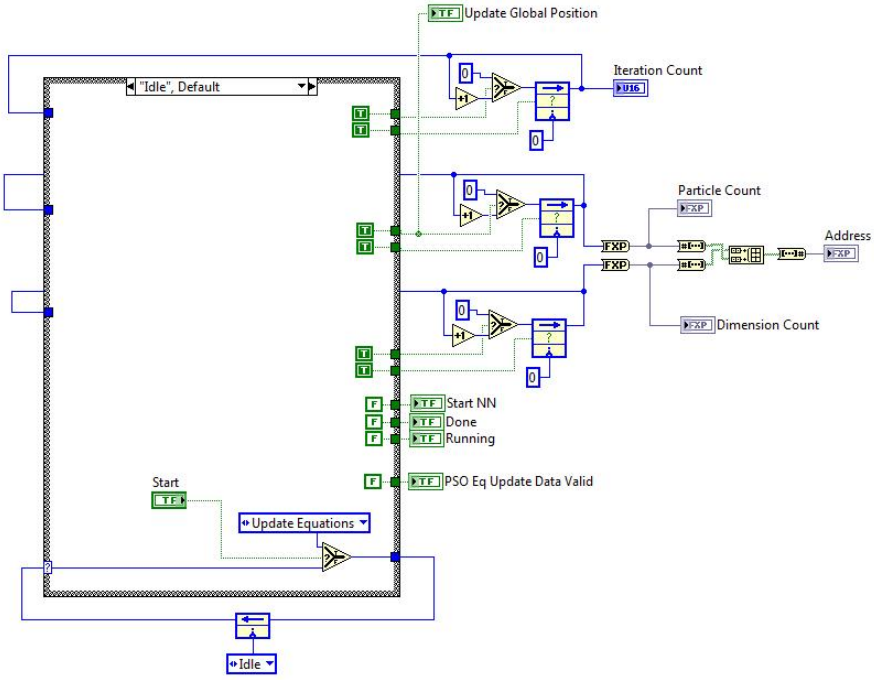


F.2: FPGA Top (Right)

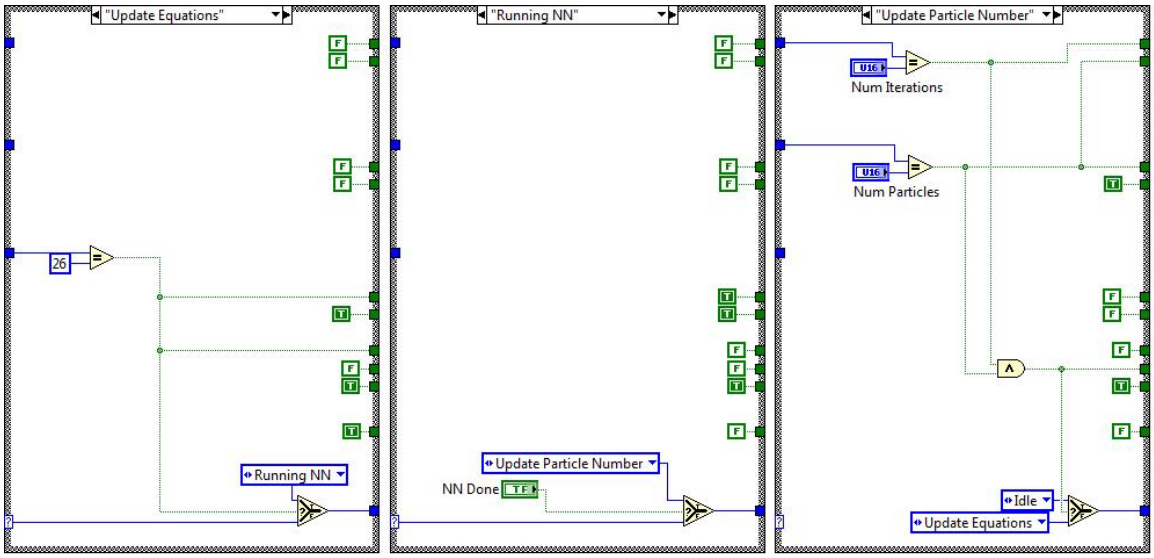


F.3: Update Equations

PSO Controller.vi

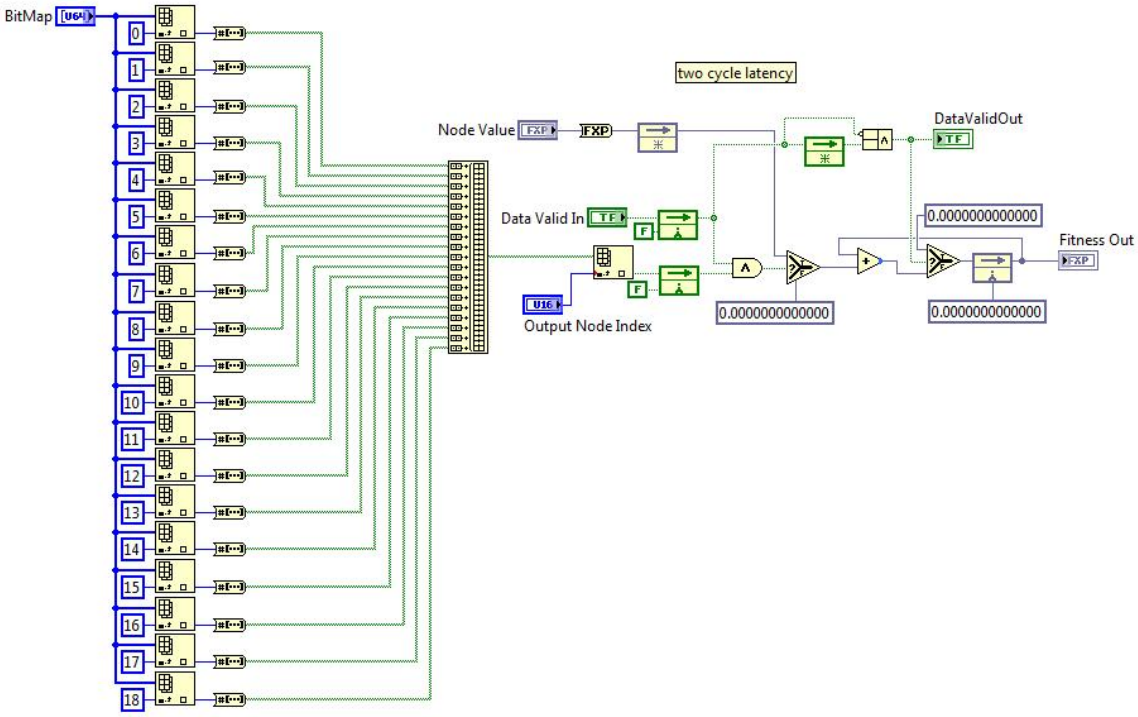


F.4: PSO Controller – Idle



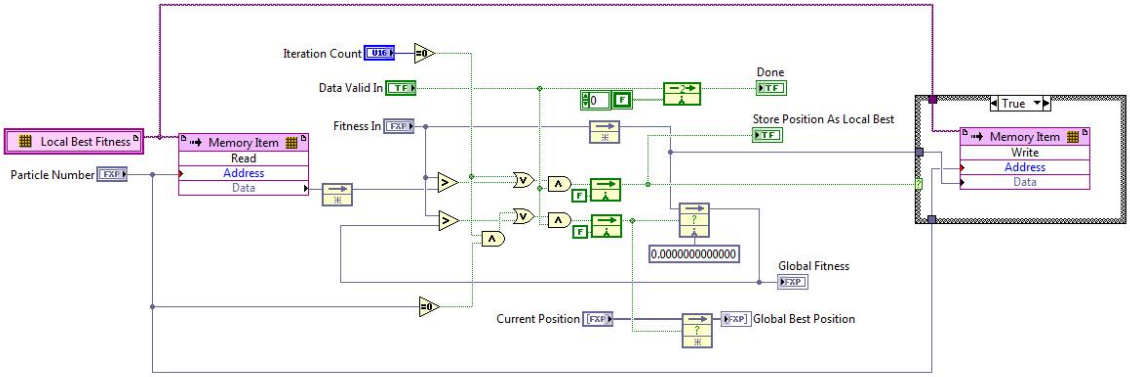
F.5: PSO Controller Cases

FitnessCalculator.vi



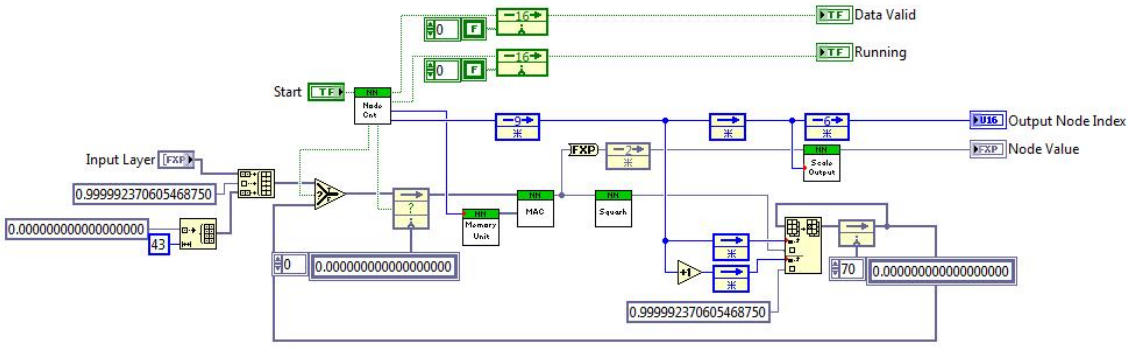
F.6: Fitness Calculator

UpdateGlobalLocal.vi



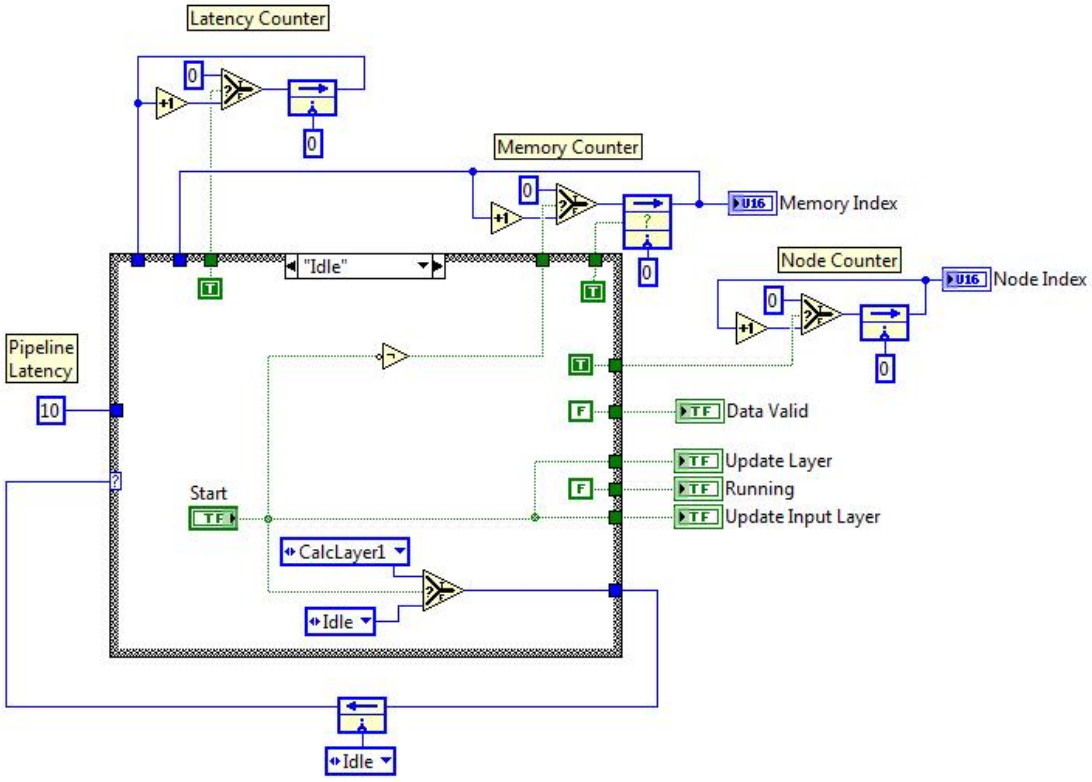
F.7: Update Global Local

NodeCalculator.vi

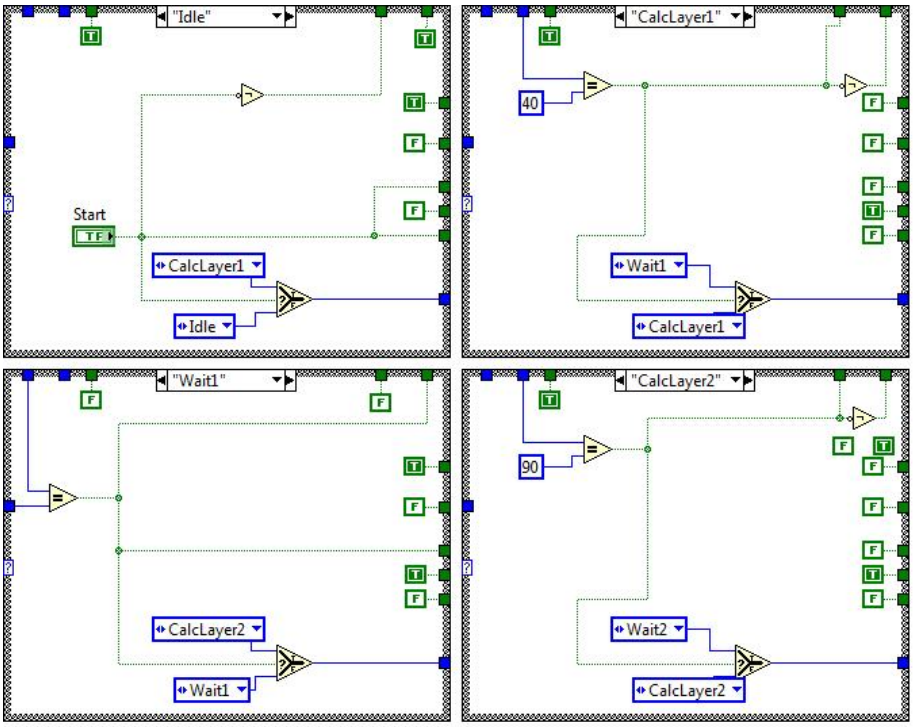


F.8: Node Calculator

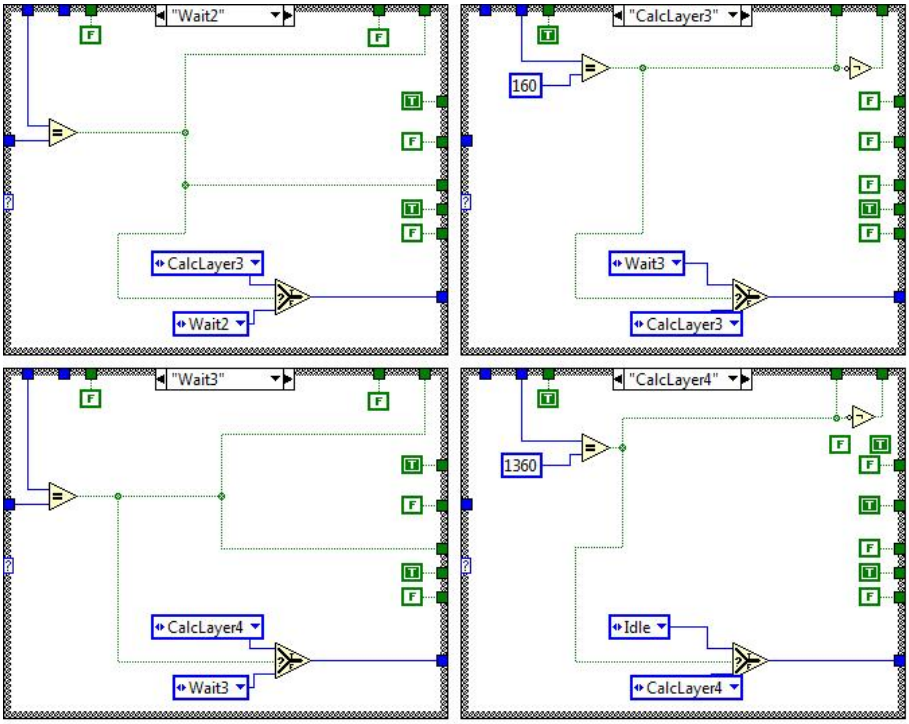
NodeCounter.vi



F.9: Node Counter

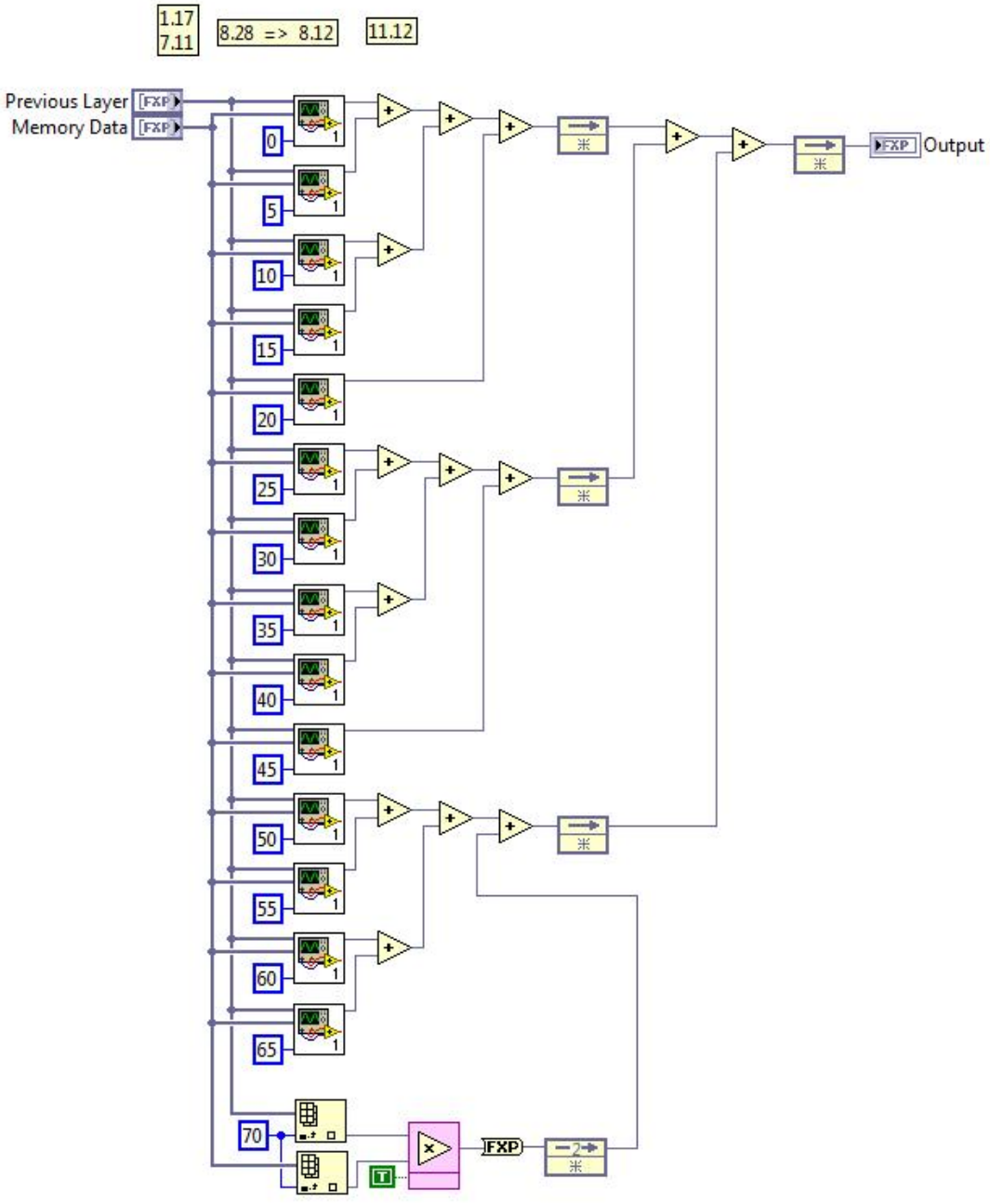


F.10: Node Counter Cases 1



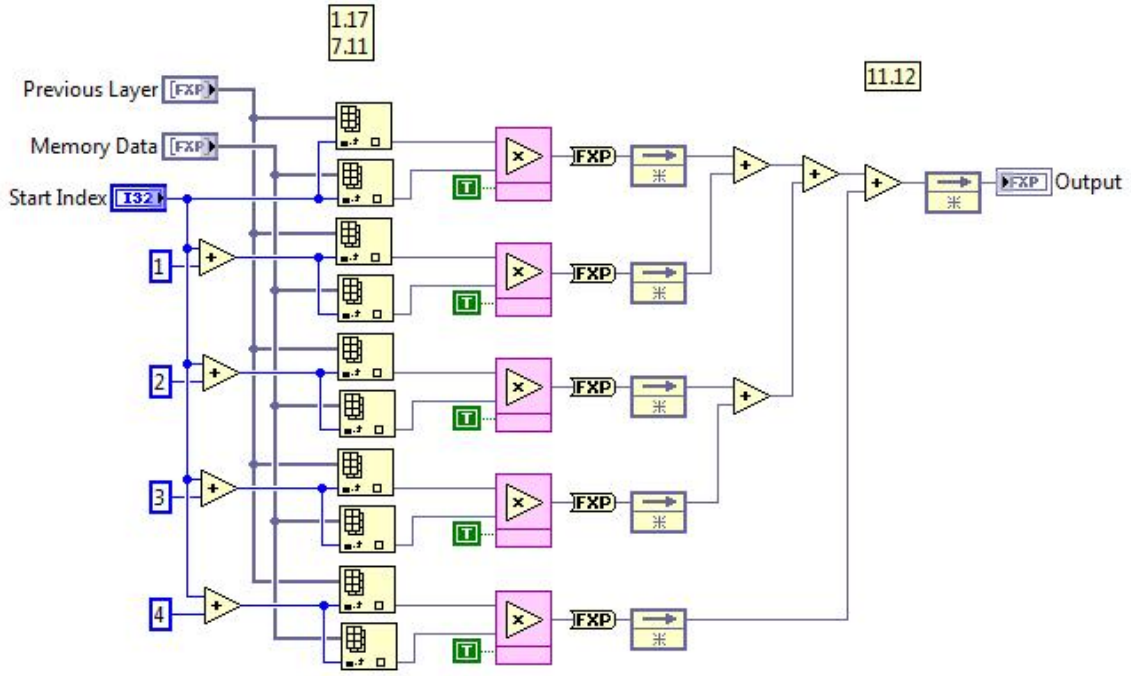
F.11: Node Counter Cases 2

MultiplyAdd.vi



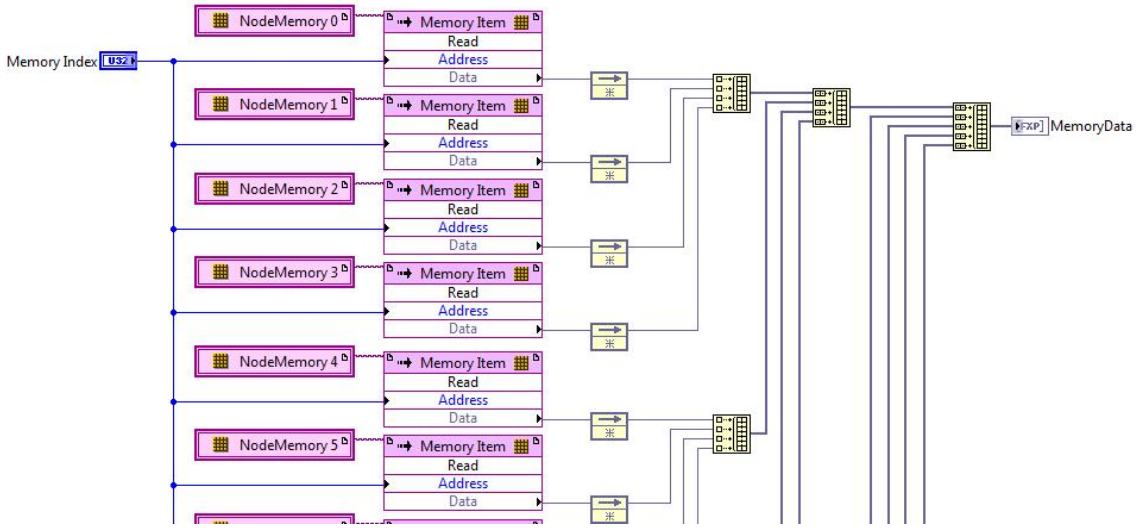
F.12: Multiply Add

MultiplyAdd5.vi



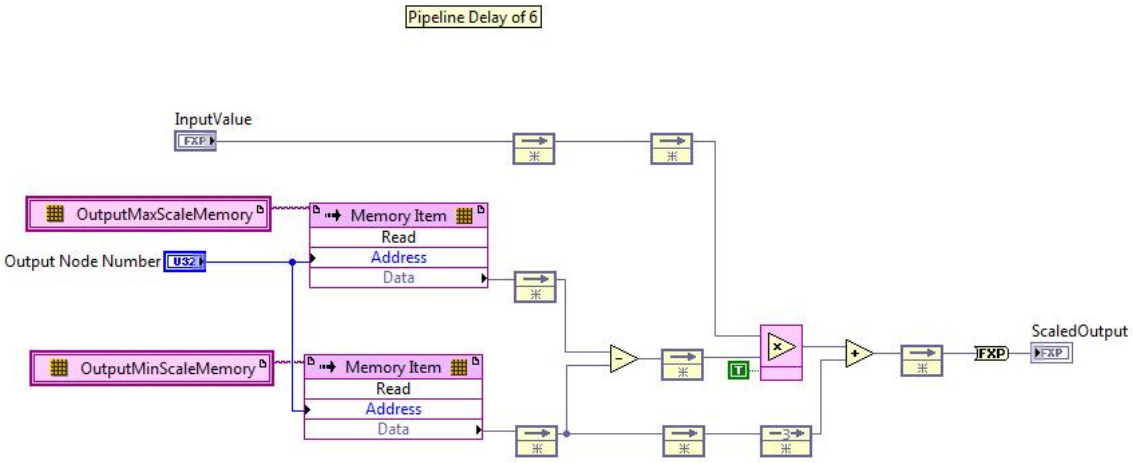
F.13: Multiply Add Five Input

MemoryUnit.vi



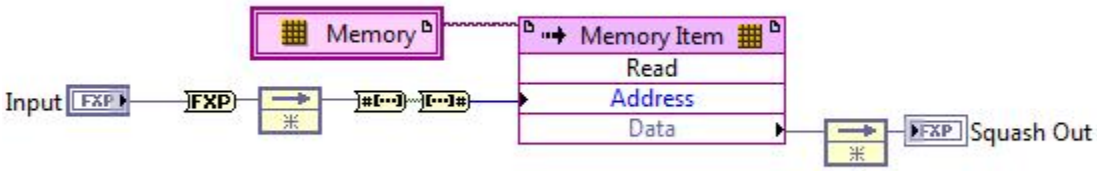
F.14: Memory Unit

ScaleOutput.vi



F.15: Scale Output

Squash.vi

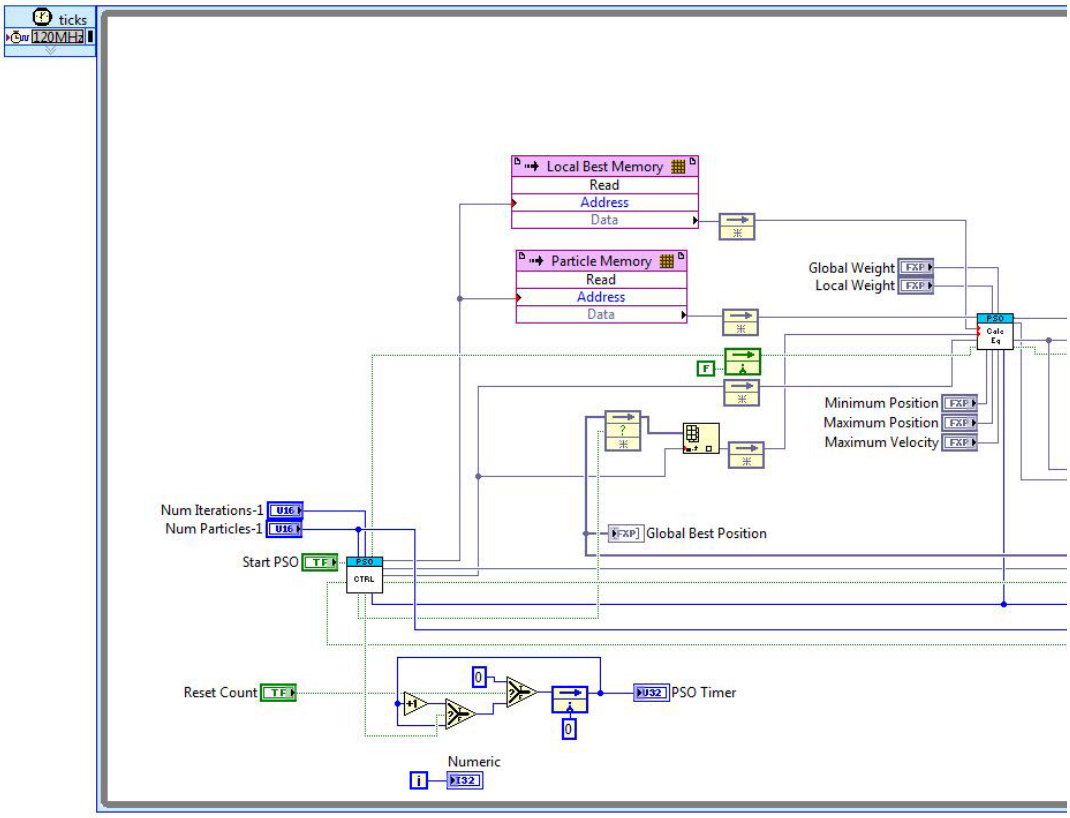


F.16: Squash

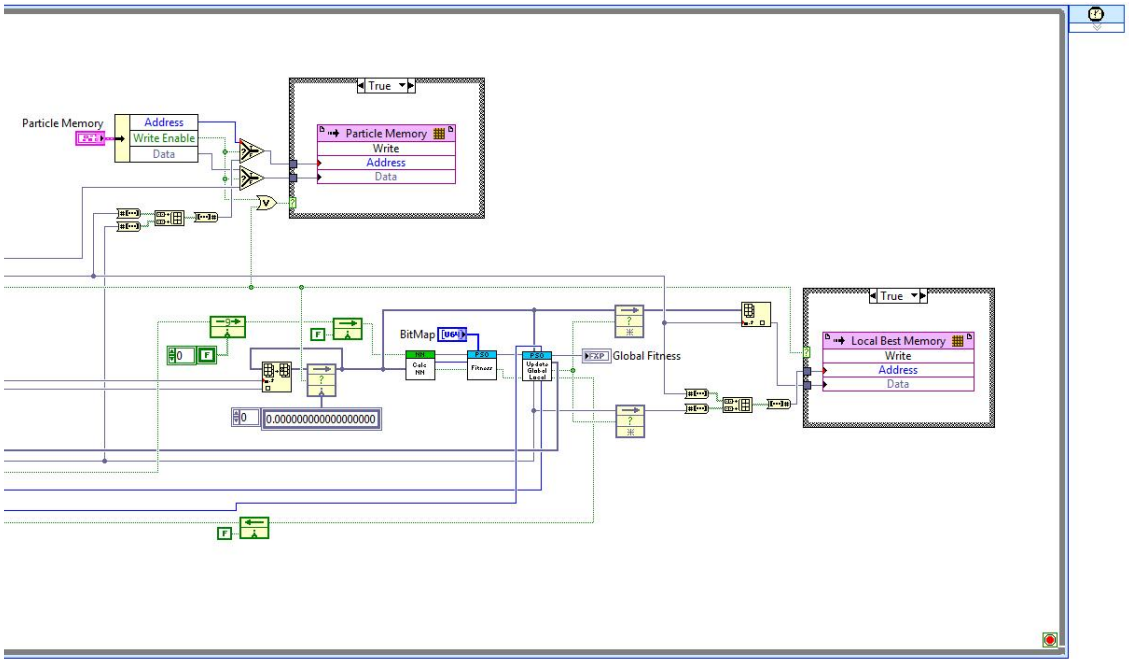
APPENDIX G

Multi Board PXIe LabVIEW FPGA Code

FPGA Top.vi

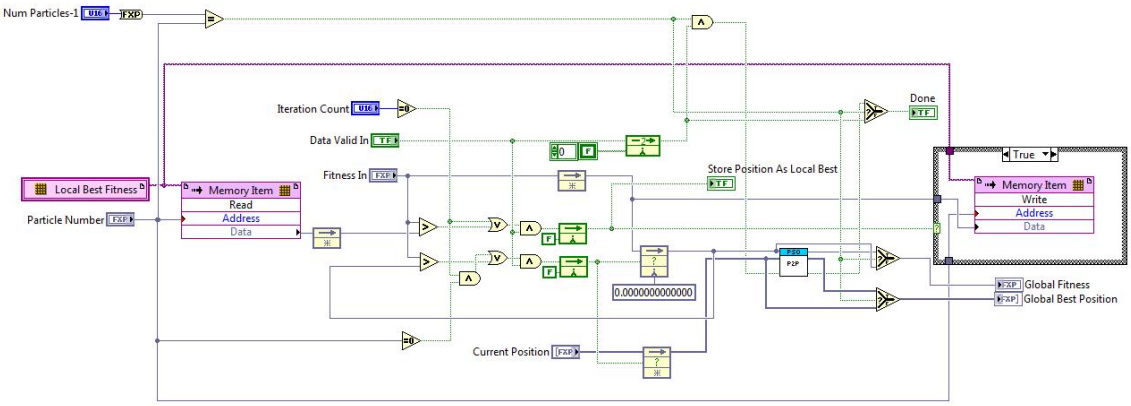


G.1: FPGA Top (Left) with P2P



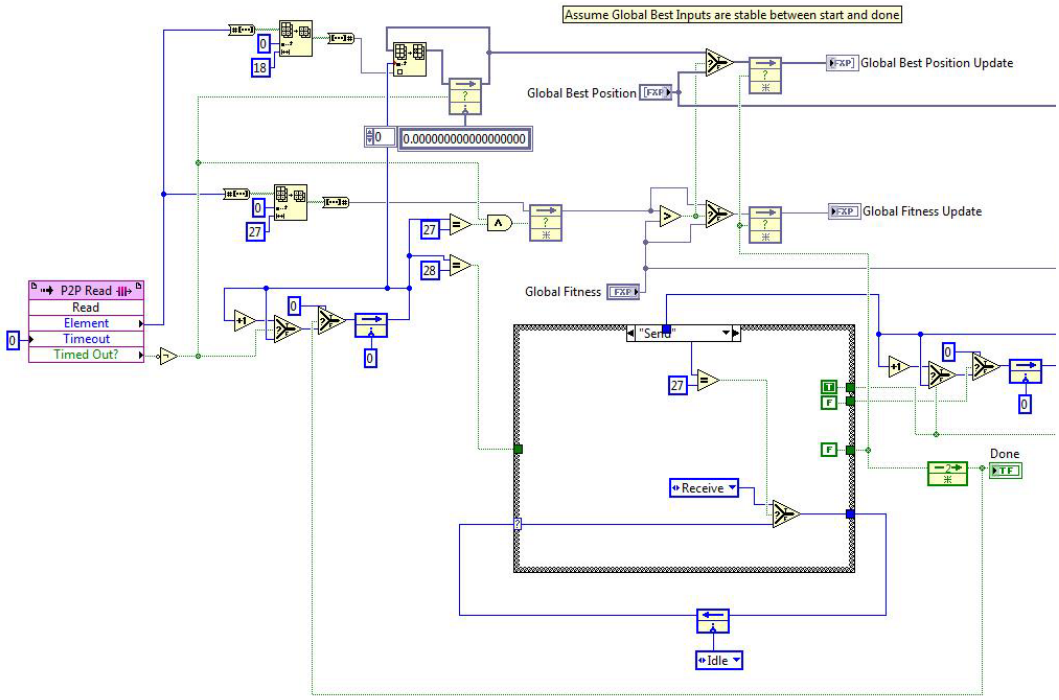
G.2: FPGA Top (Right) with P2P

UpdateGlobalLocal.vi

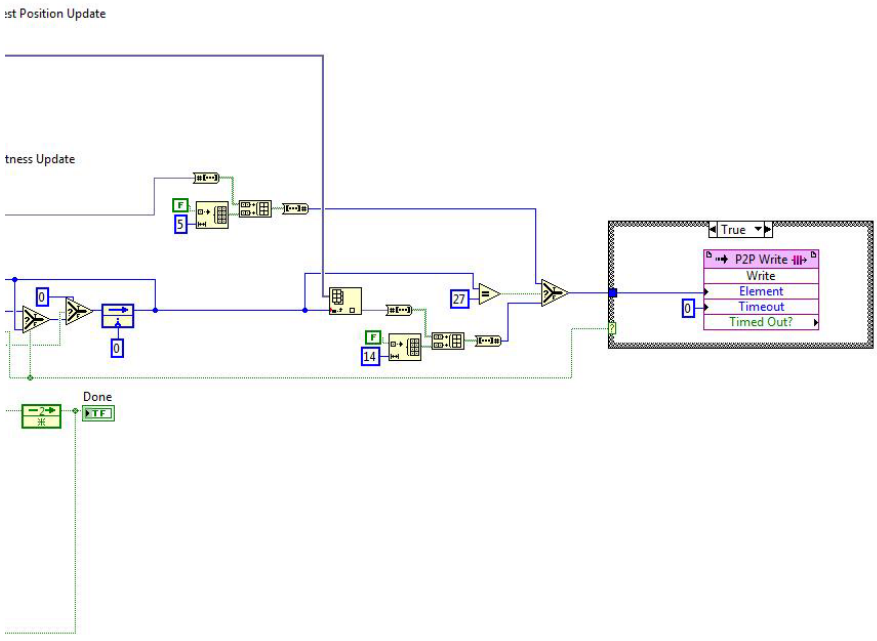


G.3: Update Global Local with P2P

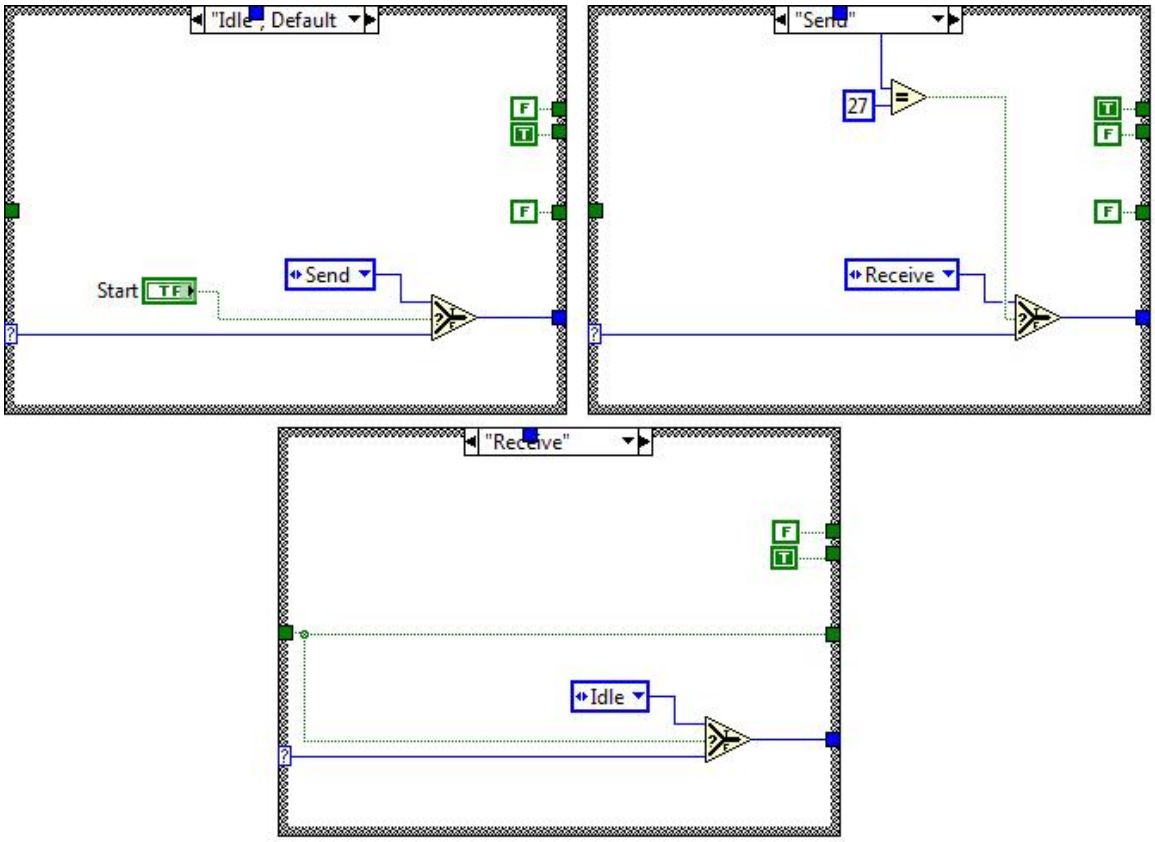
P2P Unit.vi



G.4: P2P Unit (Left)



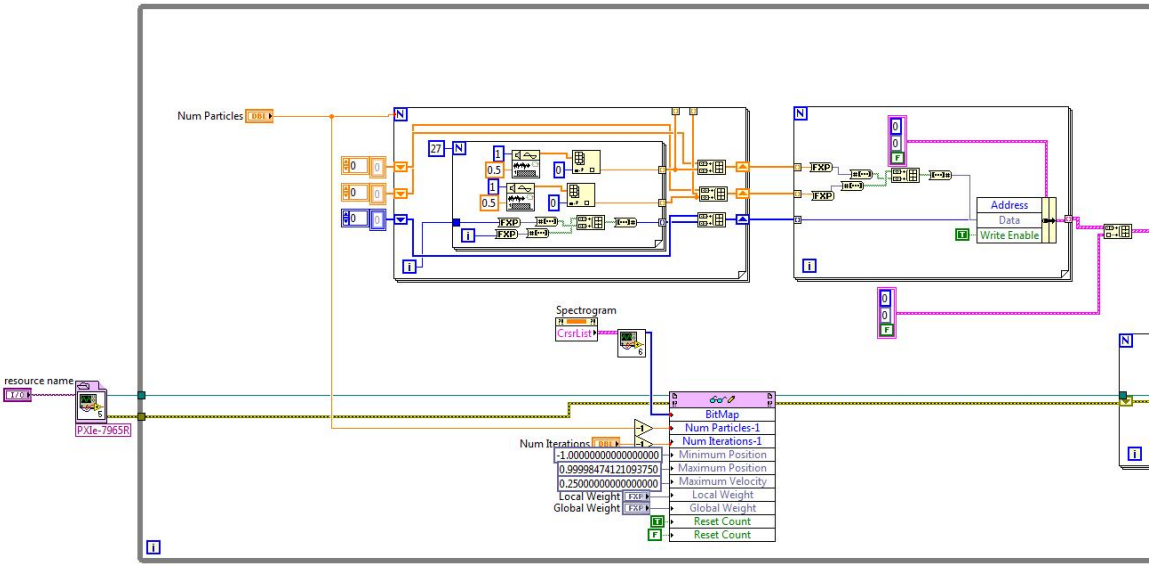
G.5: P2P Unit (Right)



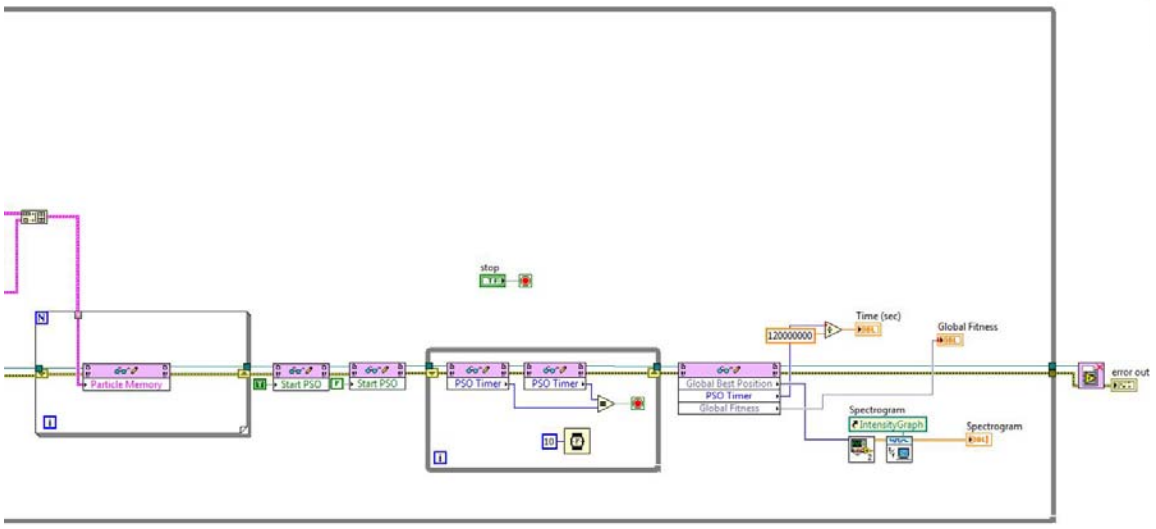
G.6: P2P Unit Cases

APPENDIX H

Host PXIe LabVIEW Code



H.1: Host Application (Left)



H.2: Host Application (Right)

BIBLIOGRAPHY

[1] B. B. Thompson, R. J. Marks, M. A. El-Sharkawi, W. J. Fox, and R. T. Miyamoto, "Inversion of neural network underwater acoustic model for estimation of bottom parameters using modified particle swarm optimizers," in *Proc. Int. Joint Conf. Neural Netw.*, 2003, pp. 1301–1306 [Online]. Available: http://marksmannet.com/RobertMarks/REPRINTS/2003-07_InversionOfNeuralNetworkUnderwater.pdf

[2] W. L. J. Fox, R. J. Marks, M. U. Hazen, C. J. Eggen, and M. A. El-Sharkawi, Environmentally adaptive sonar control in a tactical setting," *Impact Environmentally Variability Acoustic Predictions on Sonar Performance* pp. 595–602, 2002 [Online]. Available: http://marksmannet.com/RobertMarks/REPRINTS/2002_EnvironmentallyAdaptiveSonar.pdf

[3] C. A. Jensen, R. D. Reed, R. J. Marks, M. A. El-Sharkawi, J. Jung, R. T. Miyamoto, G. M. Anderson, and C. J. Eggen, "Inversion of feedforward neural networks: Algorithms and applications," *Proc. IEEE*, vol. 87, no. 9, pp. 1536–1549, Sep. 1999 [Online]. Available: http://marksmannet.com/RobertMarks/REPRINTS/1999- 09_InversionOfFeedforward.pdf

[4] J. Zhu and P. Sutton, "FPGA implementation of neural networks – A survey of a decade of progress," in *Proc. 13th Int. Conf. Field-Programmable Logic Appl.*, 2003, pp. 1062–1066

[5] R. W. Duren, R. J. Marks II, P. D. Reynolds, and M. L. Trumbo, "Real-time inversion on the SRC-6e reconfigurable computer," *IEEE Transactions on Neural Networks*, vol. 18, no. 3, pp. 889-901, May 2007 [Online] Available: http://marksmannet.com/RobertMarks/REPRINTS/2007_RealTimeNeuralNetworkInversion.pdf

[6] P. D. Reynolds, R. W. Duren, M. L. Trumbo, and R. J. Marks II, "FPGA implementation of particle swarm optimization for inversion of large neural networks," in *Proc. 2005 IEEE Swarm Intell. Symp.*, Pasadena, CA, Jun. 8–10, 2005, pp. 389–392 [Online]. Available: http://www.baylor.edu/content/services/document.php?id=15541

[7] Xilinx, Inc., "Xilinx University Program Virtex-II Pro Development System: Hardware Reference Manual,"San Jose, CA, 2009 [Online]. Available: http://www.xilinx.com/univ/XUPV2P/Documentation/ug069.pdf

[8] Xilinx, Inc., “Virtex-II Pro and Virtex-II Pro X Platform FPGAs : Complete Data Sheet,” San Jose, CA, 2007 [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf

[9] National Instruments, Inc., “NI FlexRIO FPGA Modules” Austin, TX, 2010 [Online]. Available: http://www.ni.com/pdf/products/us/cat_flexriofpga.pdf

[10] Xilinx, Inc., “Virtex-5 User Guide” San Jose, CA,2010 [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug190.pdf

[11] National Instruments, Inc., “NI PXIe-1082 User Manual” Austin, TX, 2010 [Online]. Available: http://www.ni.com/pdf/manuals/372752b.pdf

[12] National Instruments, Inc., “NI PXIe-8133 User Manual” Austin, TX, 2010 [Online]. Available: http://www.ni.com/pdf/manuals/372870b.pdf