A 64-WAY HYPERCUBE INTERCONNECTED SINGLE INSTRUCTION, MULTIPLE DATA ARCHITECTURE FOR FIELD PROGRAMMABLE GATE ARRAYS

by

Katrina Jean-Marie Werger

BS, Pennsylvania State University, 2002

Submitted to the Graduate Faculty of

the School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2003

UNIVERSITY OF PITTSBURGH

SCHOOL OF ENGINEERING

This thesis was presented

by

Katrina Jean-Marie Werger

It was defended on

November 25, 2003

and approved by

Alex K. Jones, Assistant Professor, Department of Electrical Engineering

Benjamin Levine, Assistant Professor, Department of Electrical Engineering

Marlin H. Mickle, Professor, Department of Electrical Engineering

Thesis Advisor: Raymond R. Hoare, Assistant Professor, Department of Electrical Engineering

ABSTRACT

A 64-WAY HYPERCUBE INTERCONNECTED SINGLE INSTRUCTION, MULTIPLE DATA ARCHITECTURE FOR FIELD PROGRAMMABLE GATE ARRAYS

Katrina Jean-Marie Werger, MS

University of Pittsburgh, 2003

The architecture of modern FPGAs contain over one thousand 512-bit memory banks, over five hundred 4k-bit memory banks, and over one hundred thousand logic elements. This inherent parallelism of an FPGA makes it an ideal platform for a multiprocessor architecture. In addition to embedded memory, hundreds of ASIC multipliers are embedded into modern FPGA architectures. This thesis introduces three Single-Instruction-Multiple-Data architectures comprised of 2, 4, 8, 16, 32, 64 and 88 processing elements. The first architecture uses configurable logic to implement the processing elements while second and third architectures are built around ASIC multipliers and use configurable logic to implement customizable instruction. All of the architectures described in this thesis are controlled by a central instruction stream. The 64 interconnected processor SIMD design operates at 94 MHz, and utilizes 73% of the DSP blocks available in the Altera Stratix EPS80F1508C6 device but only 24% of the look-up table logic. The remaining 76% of the logic cells are available for custom instructions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

1. ALU – Arithmetic Logic Unit
2. ASIC – Application Specific Integrated Circuit
3. DFT – Discrete Fourier Transform
4. DSP – Digital Signal Processing
5. FFT – Fast Fourier Transform
6. FPGA – Field Programmable Gate Array
7. FSM – Finite State Machine
8. IOE – Input / Output Element
9. LAB – Logic Array Blocks
10. LE – Logic Element
11. LUT – Look-up Table
12. MIMD – Multiple Instruction Multiple Data
13. PE – Processing Element
14. RAM – Random Access Memory
15. ROM – Read Only Memory
16. SIMD – Single Instruction Multiple Data
17. VLIW – Very Long Instruction Word

1.0 INTRODUCTION

Parallel processing has remained an active research area for the past two decades which has led to the development and commercialization of numerous parallel architectures. Over this time there have been many advances made and today more logic is available to the designer on a single chip than used to be only possible on a cabinet-level of integration. Currently, Field Programmable Gate Arrays (FGPA) are available that contain hundreds of memories of various sizes ranging from 512 bits to 512k bits and over 100,000 logic elements. This paper presents parallel architectures that utilize the inherent parallel nature of these devices.

1.1 Motivation

Today, the largest FPGAs contain over 100,000 logic elements and over a megabit of memory within a single device. The problem is that the efficient utilization of these resources is complex and requires much design time. Much of this design time is spent in placing the design on the chip to achieve an optimal maximum clock frequency. This thesis presents a general parallel FPGA architecture that can be programmed in software and downloaded to the device, eliminating the need for a parallel architecture to design from scratch for different applications. The design flow for a custom parallel architecture compared to the design flow using the architecture presented in this paper is shown in Figure 1.

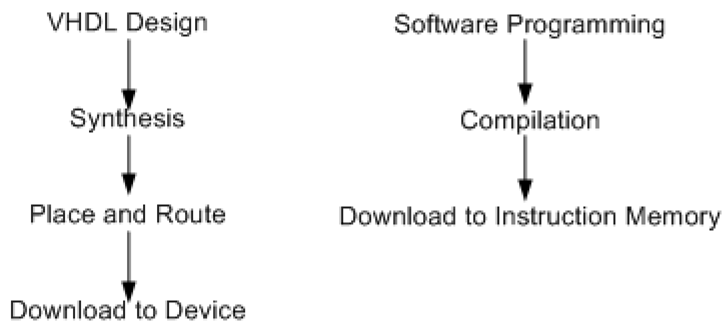

Figure 1: Design Flow for Custom Parallel Architecture and Generic Parallel Architecture

1.2 Classifications of Parallel Architectures

There are numerous ways of organizing the control structure and processing elements of a parallel computer, according to Flynn's classification [1]. The use of a single instruction multiple data (SIMD) in which a single control structure executes a single program that controls a large number of ALUs that each have their own data storage is explored in this thesis. This structure is very efficient from a logic utilization perspective and has come back into popularity with instruction set extensions for the commercially available processors from Intel, Sun, AMD, and Motorola [2].

Other classes of parallel architectures can be utilized within an FPGA architecture as well. These include very log instruction word (VLIW) architectures in which each of the function units is given a separate instruction by a single control unit [8,14]. These architectures are more flexible but require more logic and sophisticated compiler technology.

Multiple instructions, multiple data (MIMD) architectures are also possible within an FPGA and can be found within the Xilinx Vertex II Pro device family with two or even four processors within one device [9]. Synchronization and coordination of these parallel systems on a chip is handled through the FPGA logic. However, the current soft-core processors are much larger than that an ALU and thus, fewer processors can fit within a single device. The 16 bit Altera NIOS processor occupies 2100 logic elements [12], while the 8 bit ALU described in this paper requires only 393 logic elements when implemented in look-up table logic and only 96 logic elements when the ASIC multipliers are utilized.

1.3 Related Work

The ILLIAC IV [21] is a SIMD architecture that contains 64 64-bit processing units that each contain a processing element and a local memory. The control unit for this architecture can access the data in all of the processing elements, while the individual processing elements can only operate on their own memory. The interconnection for this architecture allows each processing element to have a direct link to its four neighbors in a rectangular array structure. Each processing element used 210 printed circuit boards and the local memory required four boards for 2048 64-bit word memory locations. The cost for this architecture was $31 million dollars.

Another SIMD architecture is the IBM GF11 [21]. This architecture consists of 566 32-bit processing elements. The design allowed for a peak performance of 11.4 GFLOPS with each processor being capable of 20 MFLOPS. The architecture was built with 22” x 16” circuit boards that each contained 642 chips. Each of these boards implemented on processing element, therefore the entire system required 566 of these 22” x 16” boards.

The SIMD architecture is useful in areas where the same operation needs to be applied to many different pieces of data. The applications for this type of architecture include image processing where manipulations are executed on each pixel [3,4], execution of genetic algorithms [5], solving scheduling problems [6], and key searches for encryption algorithms [7].

The SIMD architecture in [7] uses an FPGA as the platform for development of a 95 processing element array for an encryption key search engine. Due to the nature of the key search there is no communication between the processing elements. The maximum clock frequency achieved in this design is 68 MHz on a Xilinx Virtex 1000E-6 FPGA. The FPGA architecture presented in this thesis is a hypercube connected array of 64 processing elements that execute a general arithmetic instruction set with the availability to enhance the set with custom instructions.

1.4 Overview of Prototype Architectures

There are three different SIMD parallel architectures presented in this thesis. Each successive architecture introduces a series of improvements on the previous architecture and adds performance to the system.

The first architecture features an array of up to 64 processing elements. This design is implemented on the Stratix FPGA using the look-up table logic. The instruction set for the first prototype consists of unsigned arithmetic and logical operations, but does not support multiplication.

The second implementation maps the processing elements to the DSP blocks of the Stratix FPGA. The use of the DSP blocks for the processing elements allows for over 80% of the device to be available for custom instructions. The use of the DSP blocks also allows for the introduction of multiplication instructions as well as a special $a*b + c*d$ instruction that takes the same number of clock cycles to execute as the rest of the arithmetic instructions.

The final prototype addresses the issue of communication between the processors. The communication scheme implemented is a hypercube design. Additional improvements made to the final design include increased bit width from eight to nine bits and the inclusion of signed arithmetic instructions. There is also a reduction in the number of clock cycles needed to execute certain instructions. This design also allows for increased flexibility for applications by introducing the ability to turn individual processors on and off.

All of the prototype architectures introduced by this thesis are broken up into a single control unit and up to either 64 or 88 individual processing elements. These processing elements are arranged into a SIMD architecture where each processing element operates only on its local memory. Each instruction is fetched and decoded by the control unit which broadcasts the decoded instruction to each Arithmetic Logic Unit (ALU). Every ALU simultaneously executes the same instruction by operating on the data stored in its local memory.

2.0 FPGA PARALLELISM

Due to the parallel nature, high frequency, and high density of currently available FPGAs they make an ideal platform for the implementation of a massively parallel architecture. One chip can hold multiple processors and control units that will work together simultaneously. The FPGA architecture can be broken down into three major parts [12]. The first part of the architecture consists of logic cells, embedded memory blocks, and *DSP blocks*. A *DSP block* consists of embedded ASIC multipliers and an adder or accumulator output stage to increase the performance of common Digital Signal Processing operations. The Input/Output cells make up the second part. The final part of the FPGA device architecture is the internal interconnection.

The logic cells, embedded memory blocks, and the DSP blocks make up the first part of the architecture. The major part of the logic cell, which is also known as the logic element, is a look-up table (LUT). This LUT is used to generate the logic function of the cell. The logic cell also contains multiple mutiplexers and an internal register. The LUTs within the logic cells can be reprogrammed to change the logic function and allow for the FPGA chip to be reused. Each FPGA could contain over 100,000 of these logic cells and many embedded memory blocks, making this device a suitable platform for parallel processing. These logic cells and memories can be used to create a single processor and if enough logic cell and memory combinations are available, many processors can be created on a single device. The DSP blocks can be configured as multipliers, multiply and add blocks, or multiply and accumulate blocks.

2.1 Altera Stratix Architecture

The SIMD architectures presented in this thesis are all targeted to the Stratix FPGA chip provided by Altera [11], but it should be noted that Xilinx also offers embedded multipliers in their Virtex II [10] and Virtex II Pro [9] devices. According to *Stratix Device Handbook, Volume 1* [11], the device is arranged as follows and shown in Figure 2:

> Stratix devices contain a two-dimensional row- and column-based architecture to implement custom logic. A series of column and row interconnects of varying length and speed provides signal interconnects between logic array blocks (LABs), memory block structures, and DSP blocks.
>
> The logic array consists of LABs, with 10 logic elements (LEs) in each LAB. An LE is a small unit of logic providing efficient implementation of user logic functions. LABs are grouped into rows and columns across the device.
>
> M512 RAM blocks are simple dual-port memory blocks with 512 bits plus parity (576) bits. These blocks provide dedicated simple dual-port or single-port memory up to 18-bits wide at up to 318 MHz. M512 blocks are grouped into columns across the device in between certain LABs.
>
> M4K RAM blocks are true dual-port memory blocks with 4K bits plus parity (4,608 bits). These blocks provide dedicated true dual-port, simple dual-port, or single-port memory up to 36-bits wide at up to 291 MHz. These blocks are grouped into columns across the device in between certain LABs.
>
> M-RAM blocks are true dual-port memory blocks with 512K bits plus parity (589,824 bits). These blocks provide dedicated true dual-port, simple dual-port, or single-port memory up to 144-bits wide at up to 269 MHz. Several M-RAM blocks are located individually or in pairs within the device's logic array.
>
> Digital signal processing (DSP) blocks can implement up to either eight full-precision 9 x 9-bit multipliers, four full-precision 18 x 18-bit multipliers, or one full-precision 36 x 36-bit multiplier with add or subtract features. These blocks also contain 18-bit input

shift registers for digital signal processing applications, including FIR and infinite impulse response (IIR) filters. DSP blocks are grouped into two columns in each device.

Each Stratix device I/O pin is fed by an I/O element (IOE) located at the end of LAB rows and columns around the periphery of the device. I/O pin support numerous single-ended and differential I/O standards. Each IOE contain a bidirectional I/O buffer and six registers for registering input, output, and output-enable signals. When used with dedicated clocks, these registers provide exceptional performance and interface support with external memory devices such as DDR SDRAM, FCRAM, ZBT, and QDR SRAM devices.

High-speed serial interface channels support transfers at up to 840 Mbps using LVDS, LVPECL, 3.3-V PCML, or HyperTransport technology I/O standards.

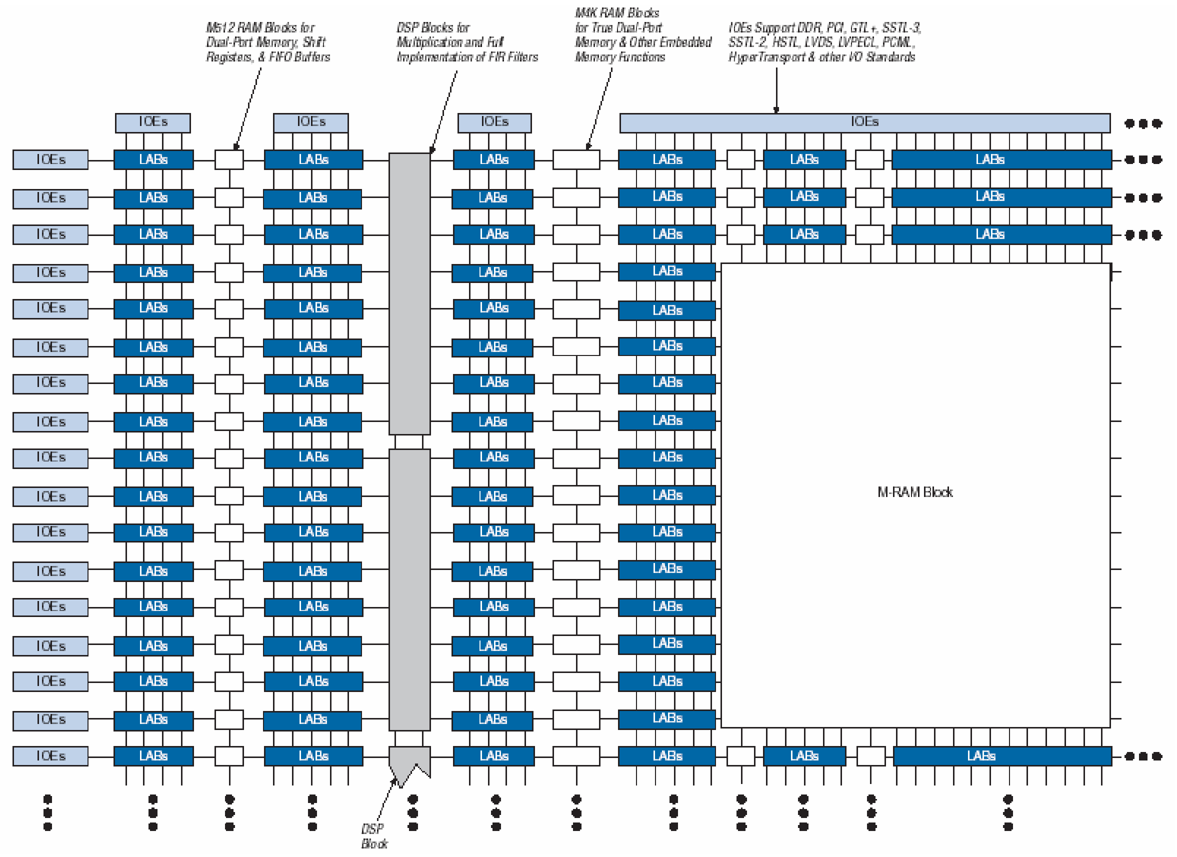
M512 RAM Blocks for Dual-Port Memory, Shift Registers, & FIFO Buffers
DSP Blocks for Multiplication and Full Implementation of FIR Filters
M4K RAM Blocks for True Dual-Port Memory & Other Embedded Memory Functions
IOEs Support DDR, PCI, GTL+, SSTL-3, SSTL-2, HSTL, LVDS, LVPECL, PCML, HyperTransport & other I/O Standards
IOEs
LABs
M-RAM Block
DSP Block

Figure 2: Top Level Architecture of Stratix FPGA Device [11]

2.1.1 Logic Elements

A single logic element, the smallest unit in the Altera Stratix device, is composed of multiple multiplexers, a LUT, control units, and a D-flip flop. A four-input LUT generates any function of four variables. The input multiplexer can select the external input data or the feedback data from the previous register. Based on the configuration of the LAB control signals, a logic element can be operated in two different modes, the normal mode and the dynamic arithmetic mode.

The normal mode can generate any Boolean function of four variables. The dynamic arithmetic mode is designed for implementing adders, accumulators, and other arithmetic functions. Each of the processors in the first prototype needs these two modes because each ALU includes arithmetic functions, ADD and SUB, and basic logic functions, AND, OR, NOT and XOR. In the second and third designs these logic elements will be utilized for the control unit and for custom instructions for the individual PEs. One of the objectives of this architecture is to utilize as few of these resources as possible for the core architecture so that they can be utilized for custom applications and for expansion of the instruction set.

2.1.2 Embedded Memory Elements

The Altera Stratix architecture contains three unique embedded memory blocks: M512, M4k, and M-RAM shown in Figure 2[11]. M512, M4k and M-RAM contain total of 576, 4k, and 576k memory bits, respectively. Depending on the width of the words, each memory block can support varying configurations. For example, M512 provides 512x1, 256x2, 128x4, 64x8 and others configurations. The embedded memory blocks can implement simple dual-port Random

Access Memory (RAM) blocks, true dual-port RAM blocks, First-In First-Out (FIFO) blocks, Read Only Memory (ROM), or shift registers.

The difference between the Simple and True Dual-Port RAM is the number of the input and output data ports. The True Dual-Port RAM has two input data ports and two output data ports. The Simple Dual-Port RAM has two clock domain inputs, a read address input, and a write address input, but has only one input data port and one output data port. In order to achieve high performance, the True Dual-Port RAM blocks, which are M4K memory blocks, are used in the prototype design to store the local data of the PEs. Table 1 shows the quantity of each memory size for five different device sizes.

Table 1: Stratix Device Features [11]

Feature	**EP1S10**	**EP1S30**	**EP1S40**	**EP1S80**	**EP1S120**
LEs	10,570	32,470	41,250	79,040	114,140
M512 RAM Blocks	94	295	384	767	1118
M4k RAM Blocks	60	171	183	364	520
M-RAM Blocks	1	4	4	9	12
DSP Blocks	6	12	14	22	28
Embedded Multipliers	48	96	112	176	224

2.1.3 DSP Blocks

The high level architecture of the Altera Straix *DSP block* is shown in Figure 3. These blocks are optimized for DSP applications and can be used for high speed parallel processing. The functionality of DSP blocks includes multiplication, addition, subtraction, accumulation, and summation. Each DSP block can be configured as up to eight 9x9 bit multipliers, four 18x18 bit multipliers, or one 36x36 bit multiplier. The different Stratix devices contain from 14 to 28 DSP blocks which allows for up to 224 nine-by-nine multipliers on one chip.

The DSP block consists of the multiplier block, the adder/output block, and interconnection and control signals. The multiplier portion of the DSP block is broken down into input registers, the multiplier, and a pipeline register. The multiplier block allows for signed or unsigned operands, but the operands for each multiplier in the DSP block must have the same sign type. The pipeline register is placed at the output of the multiplier to increase the performance of the 36x36 multiplier when used alone or the smaller multipliers when the adder output stages are used.

The adder/output block consists of an adder/subtractor/accumulator block, a summation block, an output interface, and output registers. The block can be used as strictly an output interface for the multiplier. It can also be used as an accumulator to allow for multiply and accumulate functions. Finally, this block can also be configured as a single level adder or a two level adder. This block must be used following a multiplier and can not be used independently.

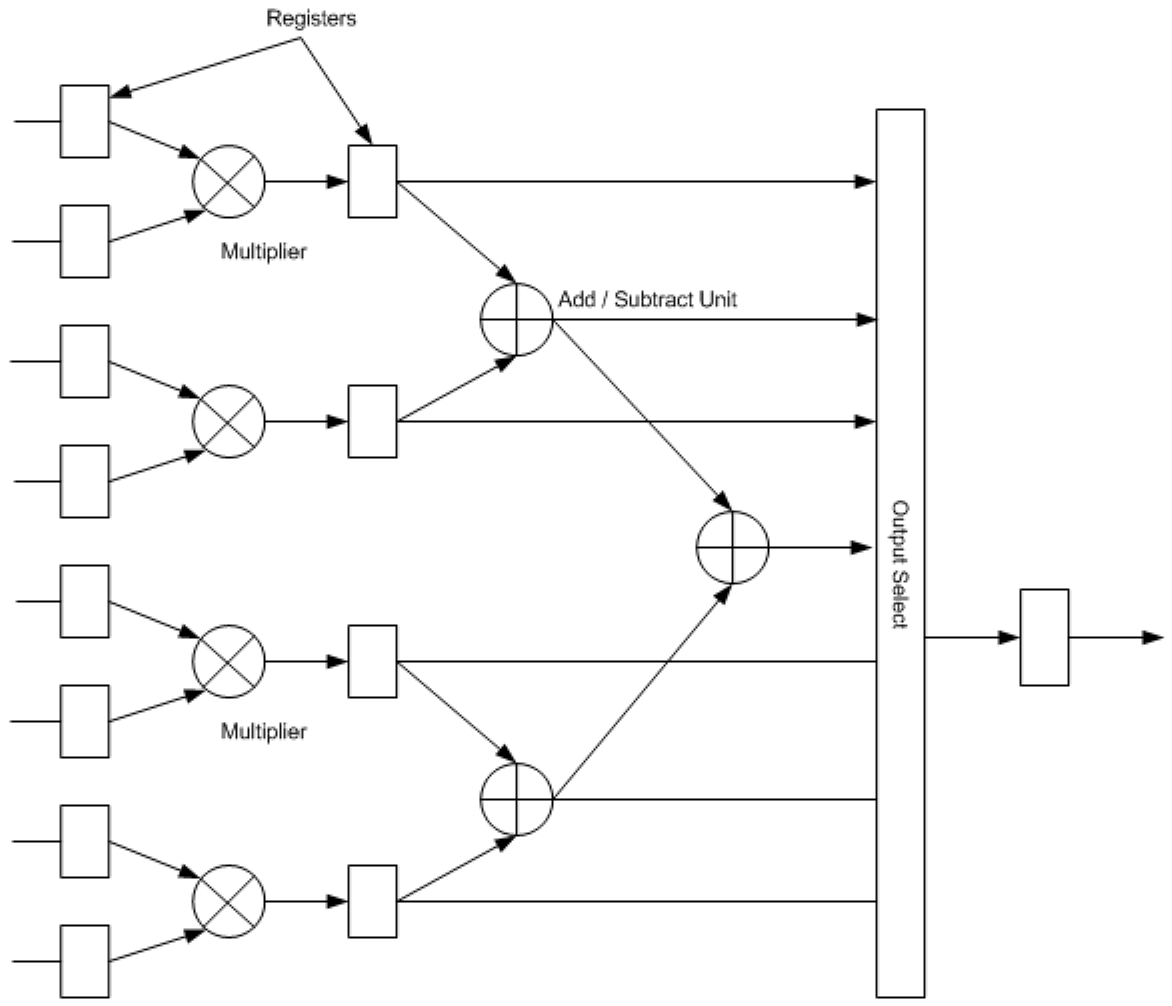

Figure 3: DSP Block Architecture [11]

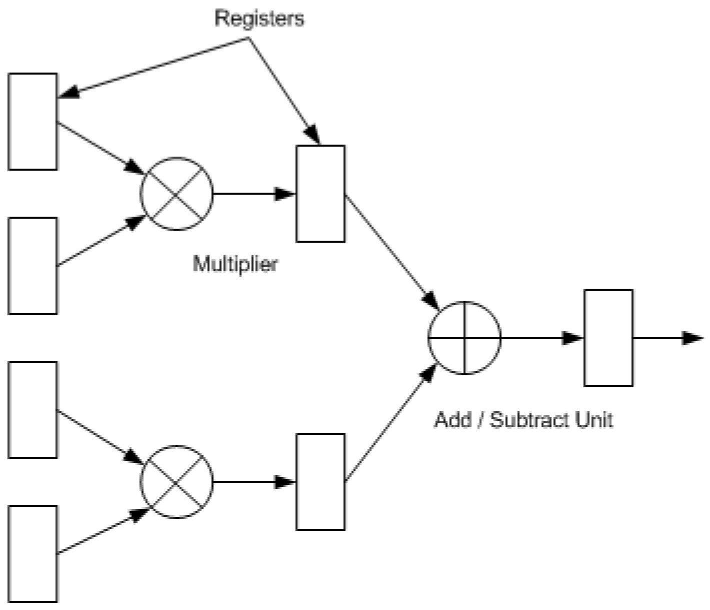

Figure 4: DSP Block - 2 Multipliers and Add Configuration [11]

Figure 4 shows the configuration of the DSP blocks used for two of the prototypes described in this paper. In the first of the designs DSP ALU designs, a DSP block is broken up into four 8x8 multiply and add or subtract blocks. In the second design, the multiply and add or subtract blocks are 9x9 instead of 8x8 to get the most precision available with four units in each block. This allows for 4 ALUs to be placed into each DSP block with the instructions add, sub, multiply, $a*b + c*d$, and $a*b – c*d$. Since four ALUs can be placed in each DSP block, there is the possibility for up to 112 processors in the largest Stratix FPGA [11].

3.0 PROCESSING ELEMENTS

This section describes the architecture of the individual processing elements for each of the designs. A high level diagram of the entire prototype architecture is shown in Figure 5. This figure shows how all the processing elements execute instructions from a single stream.

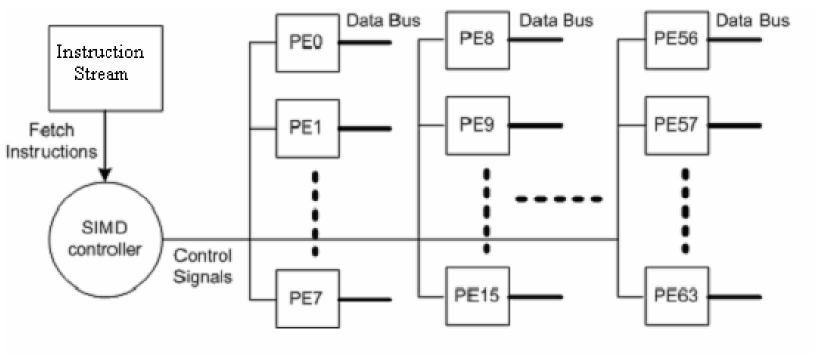

Figure 5 : High Level Architecture

The architectures were designed to take advantage of the inherent parallelism of the Stratix FPGA. An array of 1, 2, 4, 8, 16, 32, 64, or 88 processing elements is connected to a single controller and instruction stream. Each processing element is identical and contains an ALU, local memory, an input data bus to store values to the RAM, and an output data bus to read values from the RAM as shown in Figure 6 . As each instruction comes in, a single controller fetches and decodes the instruction and then each of the processing elements executes the instruction in parallel.

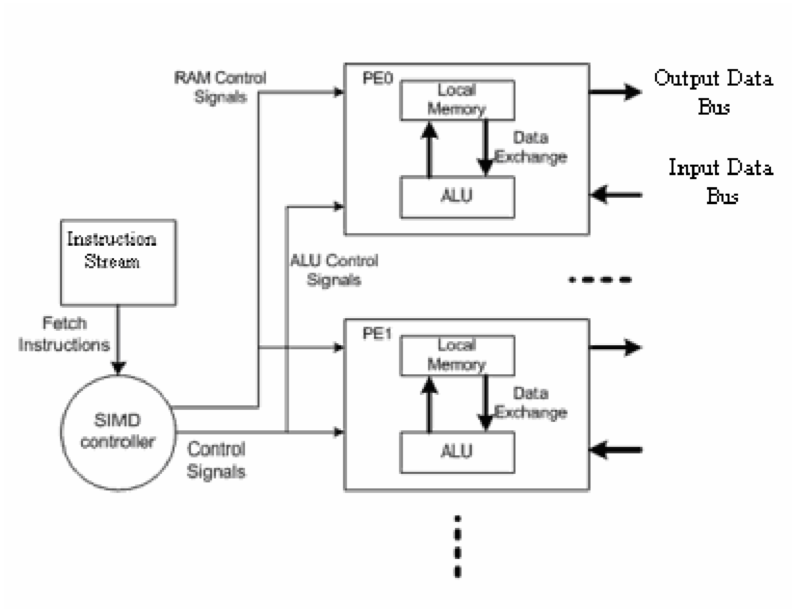

Figure 6 : Processing Element Architecture

The local memory of each of the processing elements is made up of one of the M4k RAM blocks of the Stratix FPGA. This 4k memory allows for 512 8-bit registers for each ALU. The 4k memory was chosen for the register file because it is a fully dual ported memory to allow for reading both register operands simultaneously. As was shown in Table 1, the EP1S120 has 520 M4k memories therefore there it is possible to have up to 520 ALU and register file combinations on this chip if the ALUs are implemented in the look up table logic and each one requires approximately 200 logic elements.

3.1 Logic Element Arithmetic Logic Unit

That data path for the first architecture is shown in Figure 7. The main design focus of this architecture was the creation of a simple ALU with only basic functionality that would be implemented in look-up table logic. The purpose of the limited functionality is to keep the ALU small enough to maximize the parallelism of the FPGA. In this design, the data cannot be shared between ALUs. Adding communication between the processors is an issue that will be dealt in the final evolution of the design.

The ALU of the first architecture executes basic arithmetic and logical operations. The instruction set includes addition, subtraction, logical operations, and shifting operations (logical, arithmetic, and circular). This ALU does not support multiplication or division. This small set of instructions could be useful in the area of image processing for applications such as brightening an image where a constant value is added to every pixel or for taking the complement of an image [18].

For every computational operation there is an instruction that allows the ALU to operate on two registers and store their result in a third register or operate on a register and a constant. There are also memory access instructions that allow the same constant to be written to the same register of each register file, or the input data lines of each ALU can be read to allow a different constant to be written to the same register number of each register file. The final memory access instruction reads a register from every local memory and writes the data to the corresponding output line of the processing element.

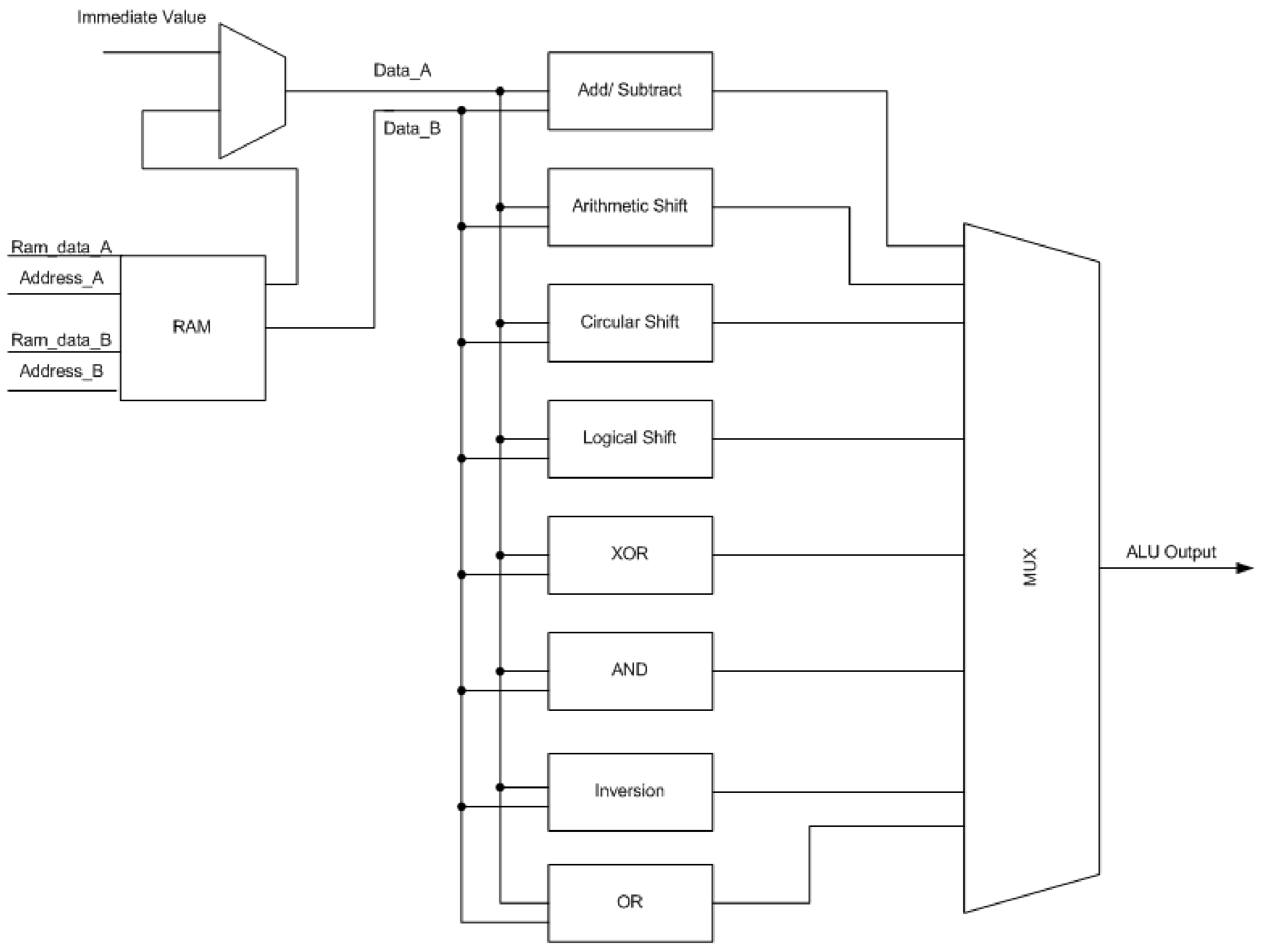

Figure 7: Data path for LUT ALU Implementation

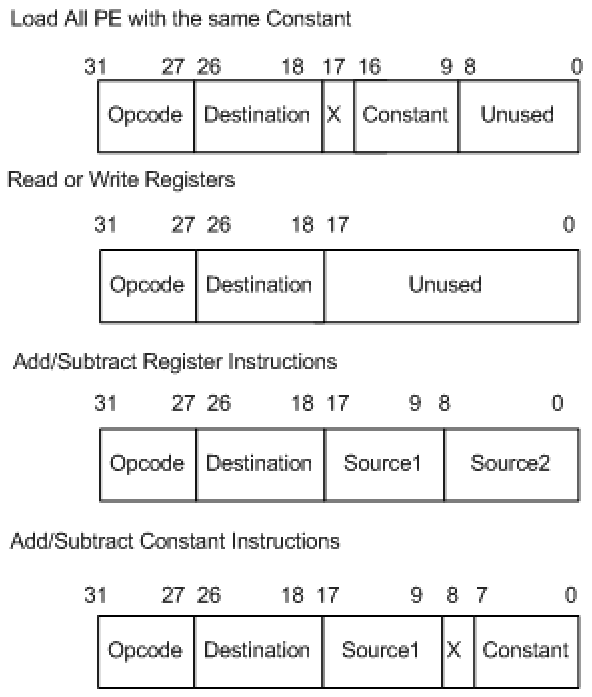

Figure 8: Instruction Format Logic Element ALU

The instructions are a fixed length of 32 bits which includes 5 bits for the operation code, 9 bits each for the 2 source registers, and 9 bits for the destination register as shown in Figure 8. If a constant value is used, the value of the constant takes the place of the second source register in the instruction format. A list of all the supported instructions is shown in Table 2.

Table 2 : ALU Instructions

Instruction	**Definition**
ADD, ADDC, SUB, SUBC	Add/Subtract Register or Constant
INC	Increment a Register
AND, ANDC, OR, ORC, XOR, XORC	AND, OR, XOR Register or Constant
NOT	Logical NOT a Register
SLL, SLLC, SRL, SRLC	Shift Left/Right Logical by a register value or constant
SLA, SLAC, SRA, SRAC	Shift Left/Right Arithmetic by a register of constant
SLC, SLCC, SRC, SRCC	Shift Left/Right Circular by a register or constant
STOREC	Store the same constant to the local memory of every ALU
STORE	Store the different ALU input values to their respective register files
READ	Read the value of a certain register from every ALU

3.2 Initial Arithmetic Logic Unit Using ASIC Multipliers

The second architecture was designed to take advantage of the large number of embedded ASIC multipliers located within the DSP blocks of the modern FPGAs as well as the overall parallelism of the architecture. The use of these DSP blocks allow for multiplication, *a*b+c*d*, and *a*b-c*d* to be added to the instruction set. The creation of the ALU within the DSP blocks allows the architecture to utilize less logic cells. With more logic cells available, custom instructions can also be added to the instruction set.

3.2.1 Latency Hiding of the Instruction Stream

In order to increase the clock speed for this architecture a register tree is added. The addition of this register tree increases the clock speeding without using significantly more chip resources. Starting with groups of two processing elements, a register for control signals and input data is added. An additional register is added to groups of four processing elements. This repeats in a traditional binary tree structure. The register tree was started with groups of two processing elements instead of adding a register at each PE due to the requirement of the DSP blocks that only two add/subtract signals are available for each block.

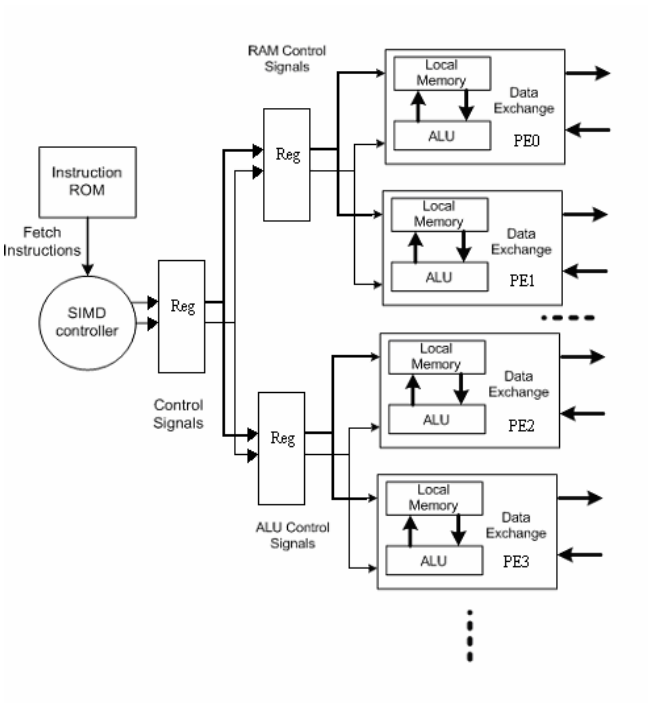

Figure 9: Processing Element Architecture with Register Tree

3.2.2 DSP Block Instructions

The core of this design is to gain as much functionality as possible out of the DSP blocks to take advantage of their increased performance over the logic elements of the chip. The implementation of the arithmetic operation within the DSP blocks leaves the majority of the logic elements available for application specific instructions. The design takes the multiply and add/subtract feature from the DSP block and designs the control signals around it. The use of the DSP block allows for an ALU that consists only of arithmetic functions which include addition, subtraction, multiplication, $a*b + c*d$, and $a*b – c*d$. The use of the DSP blocks does not allow for any logical operations, but the need for those types of instructions is implemented by two custom instruction blocks. The custom instruction blocks can be configured by the user to add two instructions specific to the user's application. These instructions can be configured to implement any combinational function of two eight bit variables and will use the LUT logic of the FPGA. In this design, the data cannot be shared between ALUs which is an issue that will be dealt with in the next version of the design.

The DSP Block ALU was generated using the Altera Megawizard tool for the Mentor Graphics HDL Designer program [19]. The Altera Megawizard tool allows the hardware designer to choose the DSP Block configuration based on the bit widths of the multipliers, how many multipliers in each unit, and if the multiplier is to accept signed, unsigned, or variable inputs. The output block is also configured using this tool and can be configured as an add/subtract unit or as an accumulator. The multiplier block configuration is shown in Figure 10. The use of this tool allows for these blocks to be directly mapped to the DSP Blocks on the Stratix FPGA during place and route.

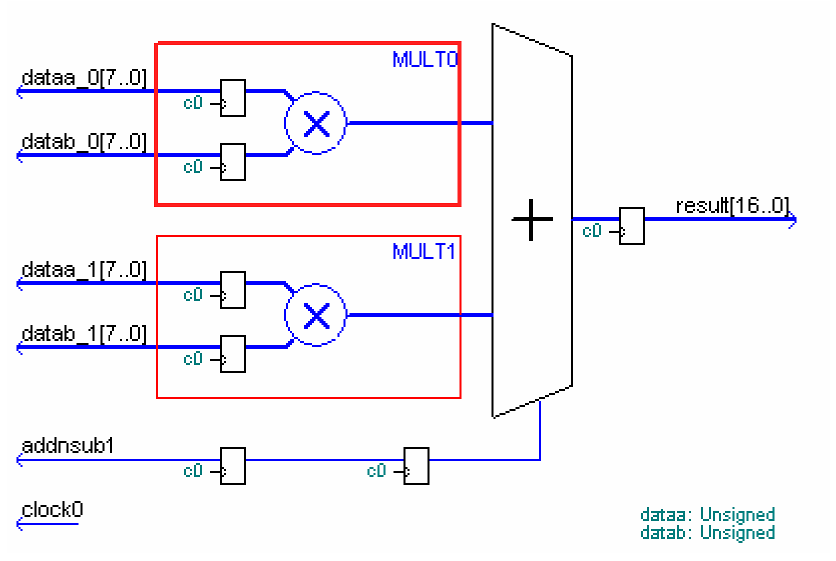

Figure 10: Altera MegaWizard Multiply and Add Block [19]

This design uses two of the multiplier blocks with an add/subtract output unit. The DSP Block design requires the adder output block to be used with multiplier blocks. The adder output stage can not be accessed separately from the multipliers. Therefore all arithmetic instructions execute the *a*b* $\underline{+}$ *c*d* operation with changes made to the inputs to achieve the other instructions. The data path for the DSP unit is show in Figure 11.

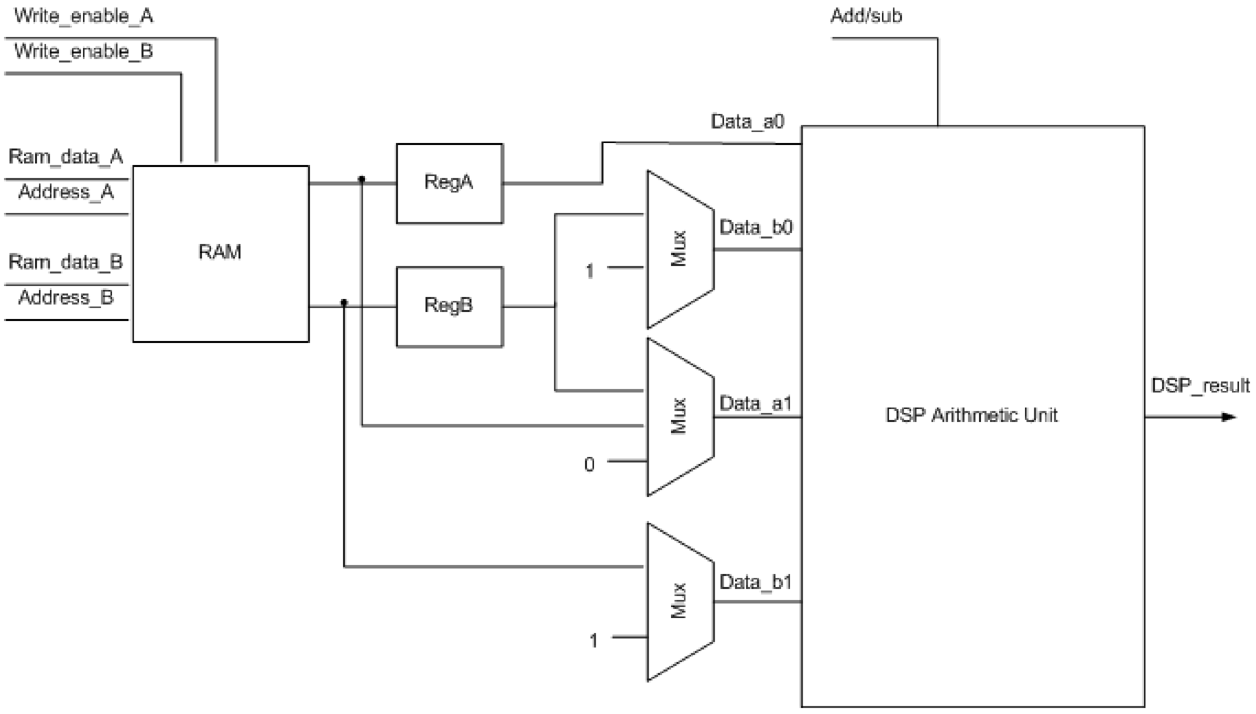

Figure 11: Data path for DSP Block Arithmetic Unit

The DSP block ALU executes basic arithmetic functions. The instruction set includes addition, subtraction, multiplication, *a*b + c*d*, and *a*b – c*d*. For add, subtract, and multiply there is an instruction that allows the ALU to operate on two registers and store the result in a third register. The *a*b + c*d* and *a*b – c*d* operations have instructions that allow four registers to be operated on and the result stored in two consecutive memory locations. The least significant 8 bits of the result are stored in the lower memory location and the most significant 8 bits are stored in the higher memory location (i.e. little endian). Since the multiplication of two 8 bit numbers produces a result of 16 bits, the addition of two of those results could require 17 bits. The use of two consecutive memory locations only allows for the storage of 16 bits, therefore in the case of overflow the output is saturated to all ones. A different scheme for handling overflow is implemented in the next version of the design. There is also a memory load instruction that reads the input data lines of each ALU to allow different values to be written to the same register number of each register file. The memory read instruction reads a register from every local memory and writes the data to the corresponding output line of the processing element. The instructions, as shown in Figure 12, are a fixed length of 49 bits which includes 4 bits for the operation code, 9 bits each for the four source registers, and 9 bits for the destination register.

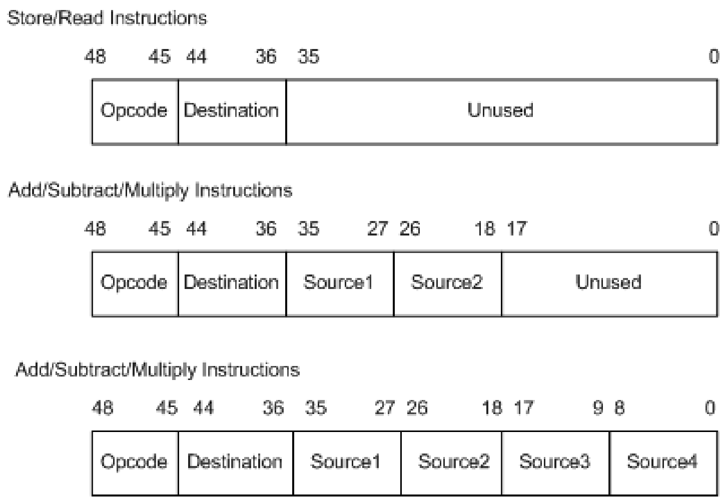

Figure 12: Instruction Format for Architecture 2

As indicated in their marketing name, these DSP blocks are highly optimized for a variety of signal processing applications. The *a*b + c*d* instructions make the DSP block able to calculate 8x8 matrix multiplications using only seven instructions on the 64 processing element prototype. The 64 processing element architecture is a perfect candidate for an 8x8 matrix multiplication where each processor will only have to calculate one cell of the result matrix. The performance of the different architectures on matrix multiplication is discussed in more detail in Section 6.

Table 3 : Architecture 2 ALU Instructions

Instruction	**Definition**
ADD	unsigned addition
SUB	unsigned subtraction
MULT	unsigned multiplication
MULTA	unsigned a*b + c*d
MULTS	unsigned a*b – c*d
STORE	Store the different ALU input values to their respective register files
READ	Read the value of a certain register from every ALU

3.2.3 Custom Instructions

To demonstrate that custom instruction could be added to the PE, two custom instruction blocks available to be user configured. The custom instructions for this prototype are set to logical AND and OR operations, but they could be set to any function of two 8-bit inputs using LUT logic. Since the current configuration using the DSP blocks uses only 17% of the LUT logic with 88 processors running, there is 83% of the chip available to implement more complicated custom instructions, as shown in Figure 13.

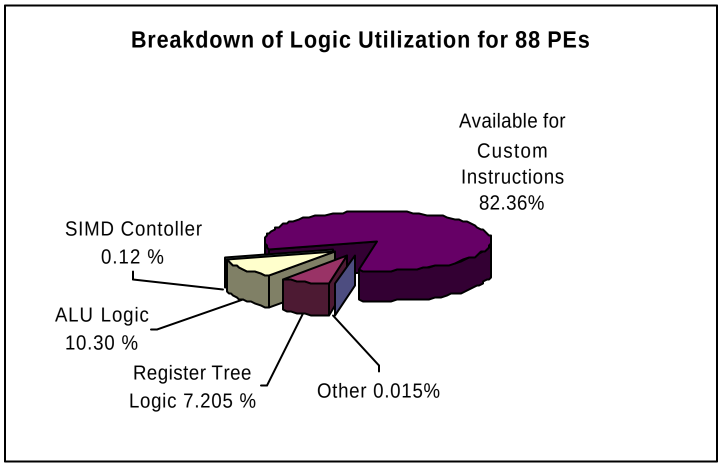

Figure 13: Breakdown of Logic Utilization for 88 PEs

3.3 Second ALU Architecture Using ASIC Mulipliers

The third and final architecture extends the DSP block architecture described above. A communication scheme is implemented in this design that allows every processor to transfer data to every other processor using a maximum of six transfers in the 64 processing element case. The bit width is extended to 9 bits which is the maximum that is supported to fit four processors in each DSP block and to create 512 registers in the 4k-bit memories. This design supports all the instructions implemented in the second architecture as well as the introduction of signed operations. The final change to this design is the addition of a status register for each processing element that can enable or disable the processor. The addition of the status register allows for processors to "sit out" an instruction, therefore increasing the flexibility of the design.

The processing element core is similar to the core of the previous design. A change had to be made to the register tree to allow for signed arithmetic. While each DSP block will accommodate two separate addition/subtraction signals, they will only allow one signal for each operand for the entire block. Due to this limitation, the register tree starts out at every four processors as show in Figure 14.

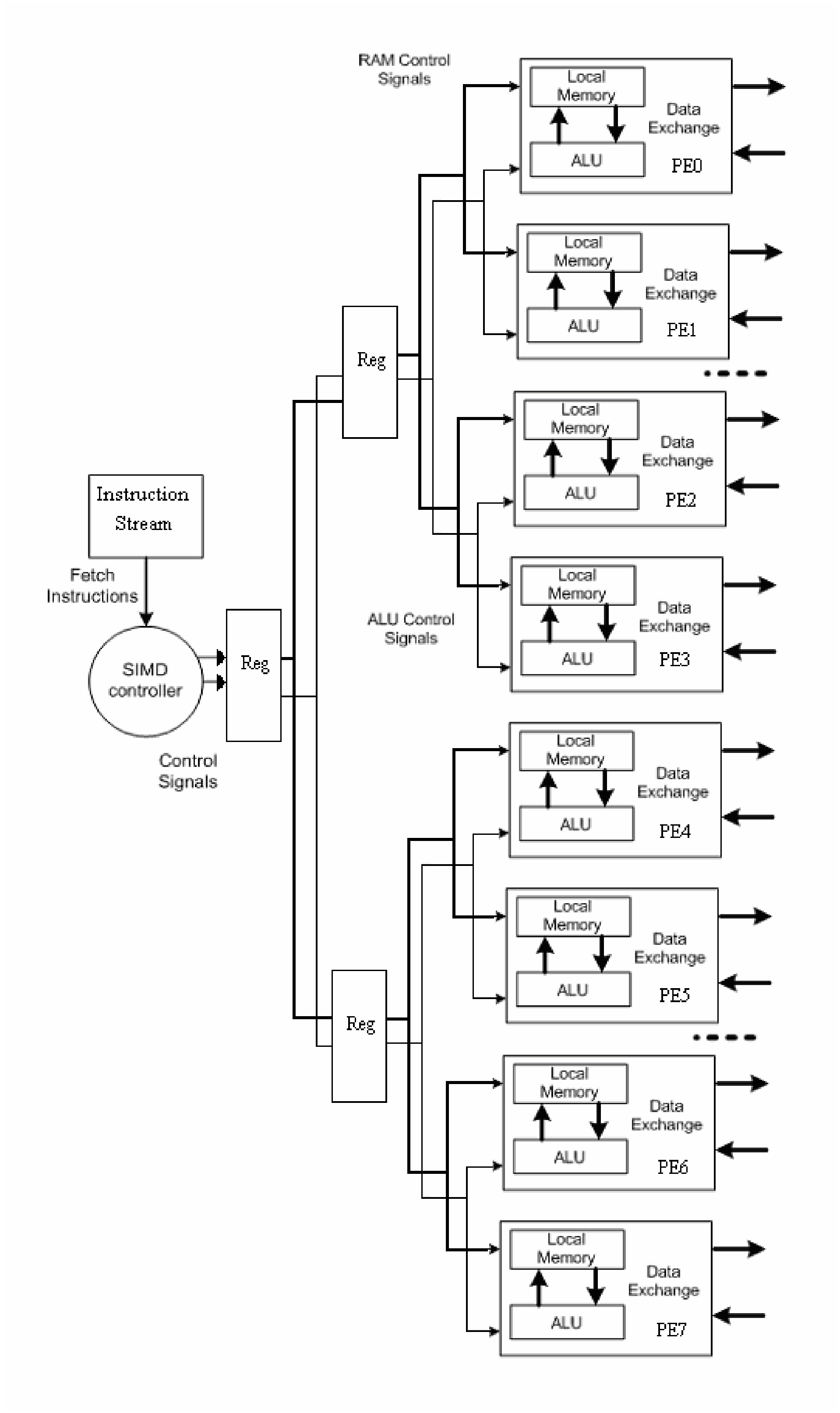

Figure 14: Register Tree Starting at 4 Processing Elements

Another major change is the addition of the status register. The output of this register controls a select line for multiplexers that determine if the write enable for the output registers and ram are grounded or controlled by the instruction stream. All of the registers are initialized to zero, therefore all processors start out enabled. In order to turn a processor off a special instruction is used to write a one to this register.

To reduce the number of clock cycles required for addition, subtraction, and multiplication, the a0 input into the DSP block ALU is taken directly from the output of the RAM instead of from the register. This allows for one clock cycle to be eliminated. The changes made to the data path from the previous architecture are shown in Figure 15.

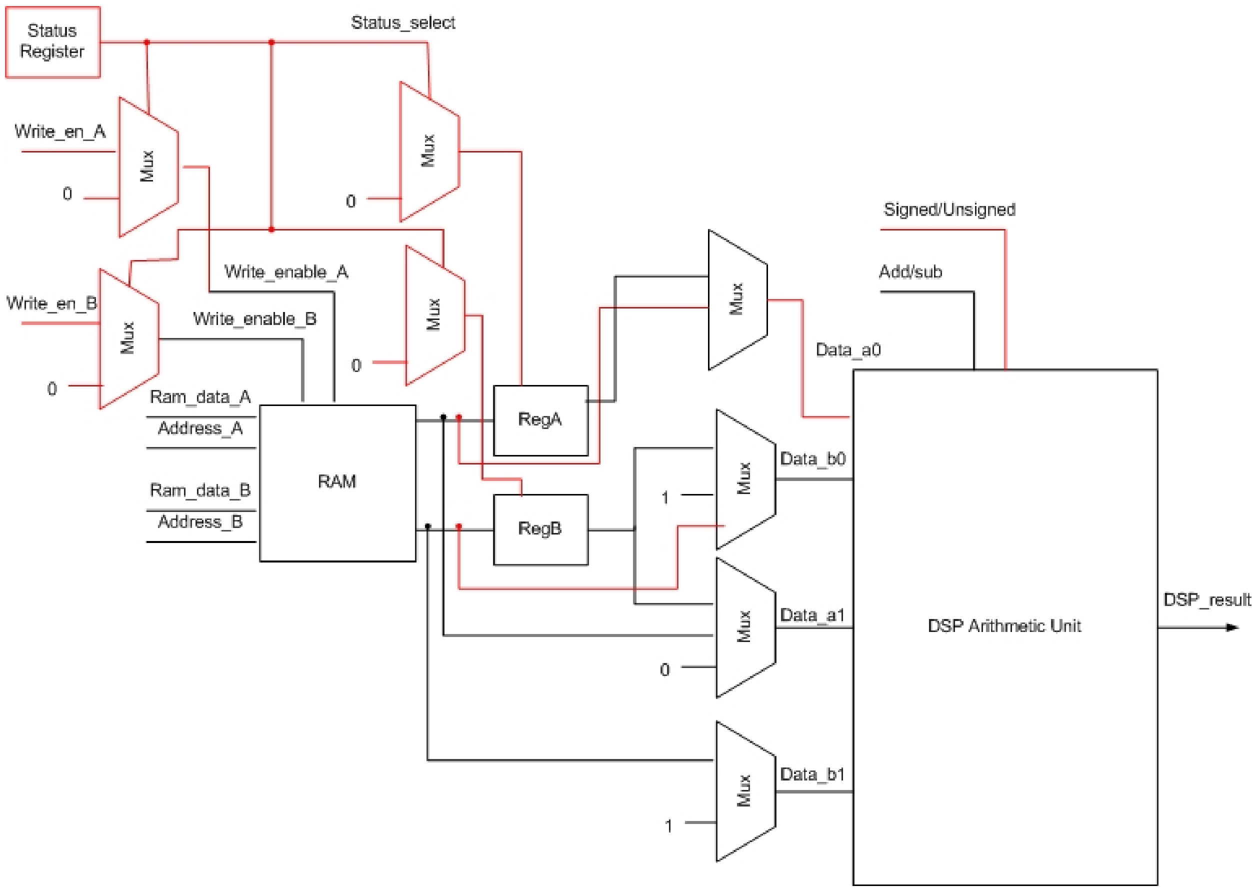

Figure 15: Changes to the DSP Block Datapath

The addition of the signed instructions increases the complexity of the saturation arithmetic. In future designs, the use of saturation arithmetic to handle overflow will be optional and other overflow handling techniques will also be available. Due to its applications in image processing, saturation arithmetic is used in this design. In the case of overflow during an unsigned add instruction the output is set to 511 while during a signed add instruction, the output is saturated at 255. For unsigned subtraction the saturation value is zero and the saturation value for signed subtraction is negative 256. The saturation values are different for the $a*b + c*d$ and $a*b - c*d$ instructions. Overflow that occurs during an unsigned $a*b + c*d$ is saturated to 18 bits of ones. For the unsigned $a*b - c*d$ operation, the overflow value is set to zero. The signed case of these two instructions will not create an overflow case. The multiplication of two 9 bit number consisting of 8 bits of data and a sign bit will create a result no larger than 17 bits (16 bits of data and a sign bit) the addition or subtraction of these two 17 bit numbers will not overflow the 18 bits of storage that are available.

The instruction set for this architecture is an extension of the set implemented in the second design. All the instructions discussed in the previous design are also implemented in this architecture. Those instructions as well as the new instructions take 9-bit operands instead of 8-bit operands. The new instructions include signed arithmetic instructions. There are now signed and unsigned versions of the addition, subtraction, multiplication, $a*b + c*d$, and $a*b - c*d$ instructions. Due to the addition of the status register to enable or disable each of the processing elements, an instruction has been included to change the value of that status register. Instructions are also added to this design to allow for the communication between processors. The communication scheme is discussed in more detail in Section 5: Communication Network.

A list of the instructions available for this design not including the instructions for interprocessor communication is shown in Table 4.

Table 4 : Architecture 3 ALU Instructions

Instruction	**Definition**
ADDU	unsigned addition
SUBU	unsigned subtraction
MULTU	unsigned multiplication
MULTAU	unsigned a*b + c*d
MULTSU	unsigned a*b – c*d
ADD	signed addition
SUB	signed subtraction
MULT	signed multiplication
MULTA	signed a*b + c*d
MULTS	signed a*b – c*d
STORE	Store the different ALU input values to their respective register files
READ	Read the value of a certain register from every ALU

For the one to thirty-two processing element architectures, the instruction format is similar to the format illustrated in Figure 12 with the opcode field increased to five bits to allow for the extension of the instruction set. The signed instructions have the same format as their unsigned counterparts. The format for the additional instruction to write to the status register is shown in Figure 16. When the architecture includes sixty-four processing elements, the instruction format had to be changed to allow for enough bits to write to the status register of every processor. In this case the instruction format consists of five bits of the opcode and 64 to allow a status bit for every processing element. All instructions with the exception of the write to status register instruction do not use the bits 0 to 18. This altered instruction format is shown in Figure 17.

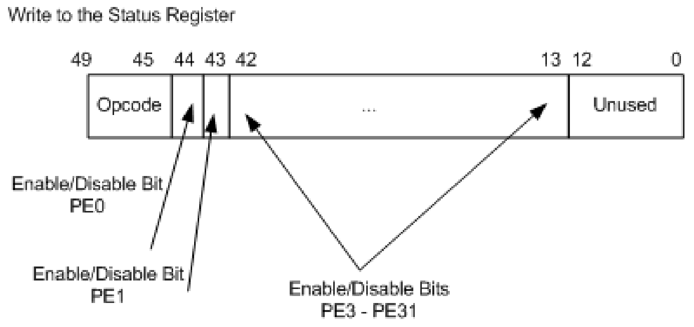

Figure 16: Write Status Register Instruction Format 1 - 32 PE

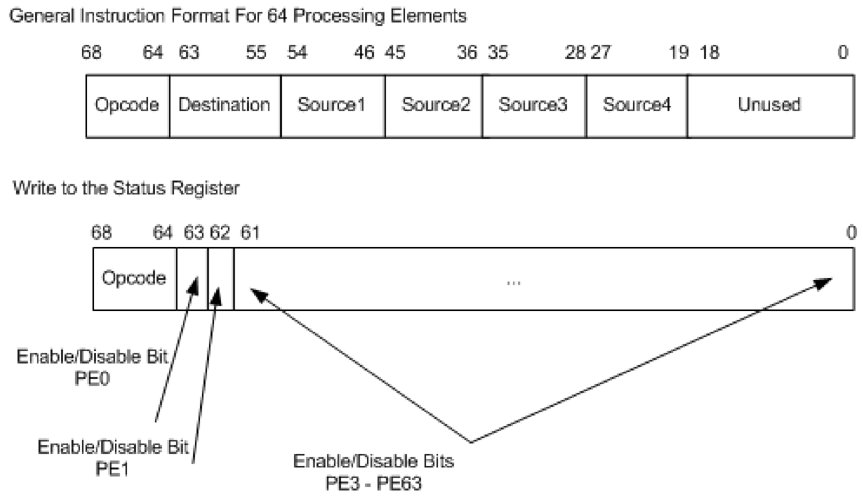

Figure 17: Instruction Format for 64 PE

4.0 CONTROL UNIT

All of the different designs operate using a SIMD architecture which allows for only a single control unit. The control unit for the first design is different from than that of the other two. The first control unit uses a ROM to decode the instructions while the second and third designs to do not use a ROM, but instead use a more complicated finite state machine.

4.1 Control Unit for Logic Element ALU

This control unit is made up of a ROM and a finite state machine (FSM). The ROM is used to decode the control signals for the ALUs. The FSM controls the fetch, decode, execute, and write back cycles of the ALUs. Operations that require the ALU to compute a value take 5 clock cycles each. The clock cycles consist of reading the control signals from the ROM, executing the instruction, storing the output to a temporary register and changing the control signals from read to write, and writing the output back to the RAM, and a wait cycle. The wait cycle is needed because the RAM block writes on the negative clock edge and the controller is rising edge. Allowing an extra clock cycle guarantees that there will not be any data hazards for the next instruction. The memory access instructions require only 3 cycles to execute because they do not require the storage of a value to a temporary register or the changing of control signals from read to write. This control unit utilizes 166 logic cells and can operate at a maximum frequency of 261 MHz.

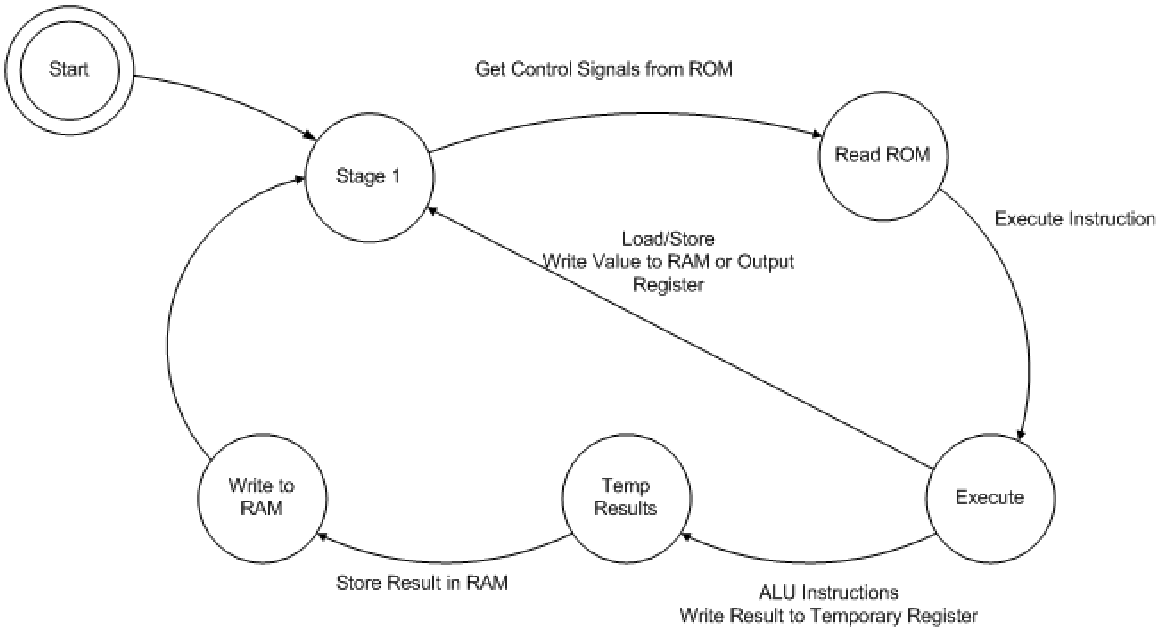

Figure 18: Diagram of Control Unit for LUT ALU

4.2 Control for Initial ASIC Multiplier ALU

This control unit is a finite state machine (FSM) that fetches the instruction, decodes it, and controls the execute cycle and write back cycle of all the ALUS. The FSM produces outputs that are sent to a decoder that determines the values of the individual control signals of the processing elements. Operations that require the use of the DSP block take six clock cycles each. The custom instructions that only use LUT logic take five clock cycles. The load instruction takes three clock cycles. Finally, to read a value from the register file to the output lines take four clock cycles, the output value is multiplexed in order to use less I/O pins. The lower order four bits of the result are available after the third clock cycle and the higher order four bits are available after the fourth clock cycle. The control unit for this design has a maximum frequency of 422 MHz and occupies 104 logic cells.

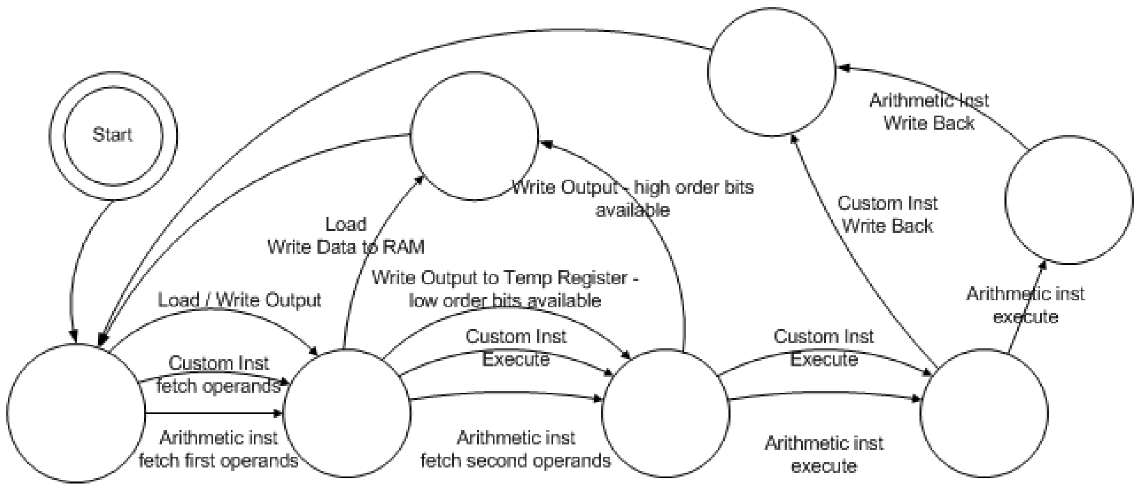

Figure 19: Diagram of Controller for First DSP ALU

4.3 Control for Second ASIC Multpilier ALU

This ALU is implemented with two different controllers. The second controller saves clock cycles for certain instructions, but causes the design to achieve a slower maximum clock frequency. The performance results for both controllers are compared in Section 7. The first control unit for this design is similar to the controller described previously and is shown in Figure 20. The control for the signed operations are the same as the control for the unsigned operation with the exception of setting the sign bit control signal high for these instructions. Due to a change in the data path for this architecture that takes the input to the ALU for addition, subtraction, and multiplication directly from the RAM instead of from the register following the RAM, one clock cycle was removed from the controller for these operations. This controller was also changed to allow for an instruction to enable or disable each individual processing element which takes two clock cycles. Control for the instructions that move data between the processors were also added and take three clock cycles to read the data from all hypercube dimensions.

These instructions will be discussed in more detain in the communication section of the architecture. The maximum frequency of this controller is 341 MHz. It utilizes 238 logic cells.

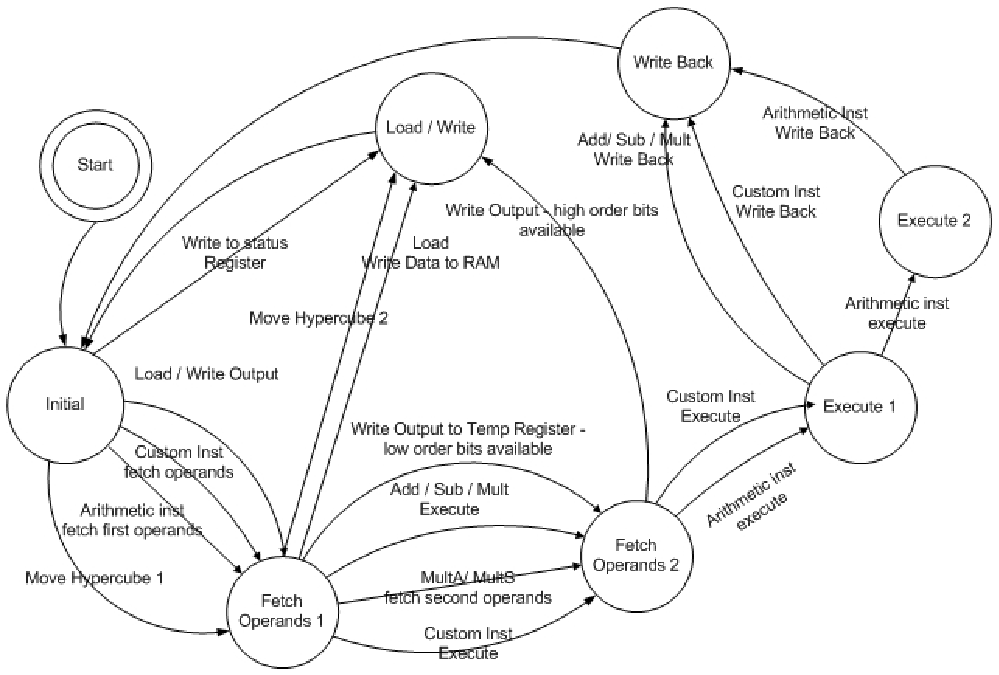

Figure 20: Diagram of First Controller for Second DSP ALU

The second controller for this design reduces the number of clock cycles required to execute custom instructions. With the new controller they now take four clock cycles instead of five. A clock cycle was also able to be removed from each of the move instructions, therefore to read from any dimension takes two clock cycles with this controller instead of three as mentioned previously. The diagram of this controller is shown in Figure 21. This controller requires 247 logic cells and can operate at a maximum frequency of 325 MHz.

Table 5 shows the maximum frequency and logic utilization for each of the control units.

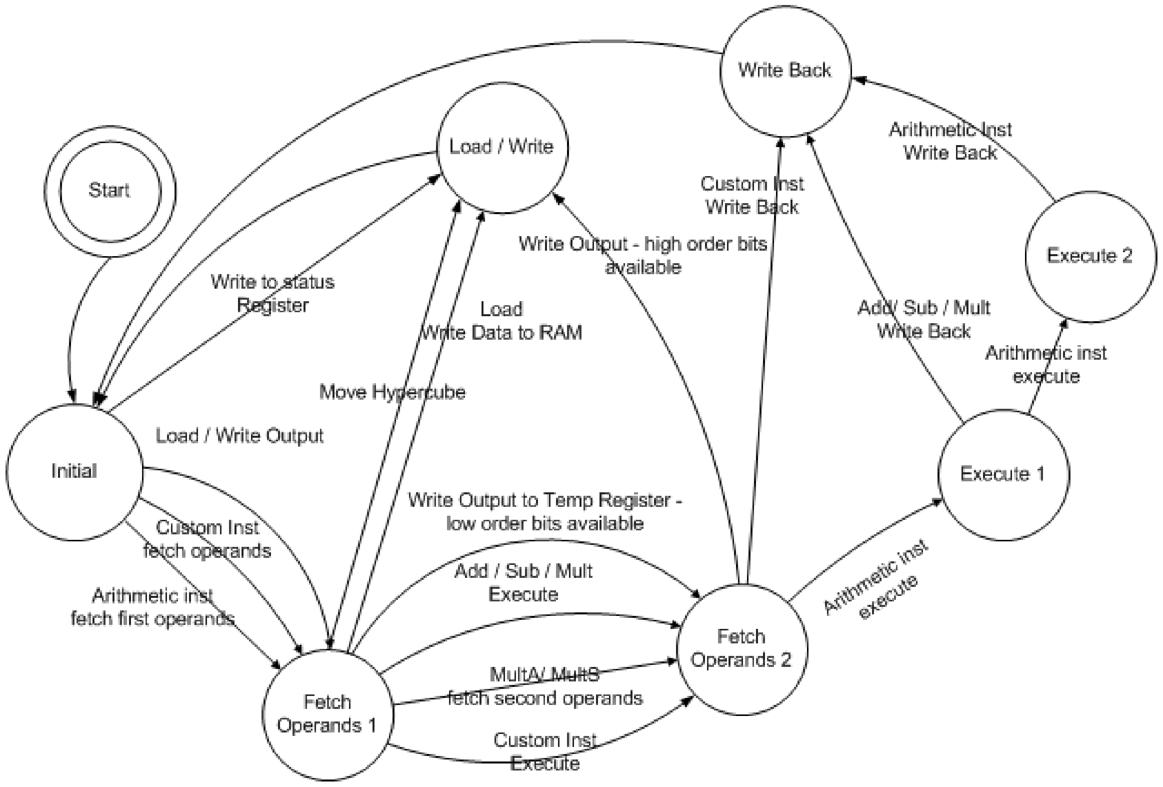

Figure 21: Diagram of Second Controller for Second DSP ALU

Table 5: Controller Maximum Frequency and Logic Cell Utilization

Control Unit	Maximum Frequency (MHz)	Number of Logic Cells
LUT Controller	261	166
Initial DSP Controller	422	104
Second DSP Controller	341	238
Third DSP Controller	325	247

5.0 COMMUNICATION NETWORK

The communication scheme implemented is a hypercube formation. This interconnection allows every processor to transmit data to every other processor in a maximum of six transfers in the 64 processor case. The 64 processor case is considered a six dimensional hypercube where the number of processors is equal to 2^d and *d* is the number of dimensions [17]. To construct the hypercube each processor is given a unique binary number to identify it that has the same number of bits as the dimensions of the hypercube. On each dimension a processor is directly connected to only one other processor. The connections are chosen by connecting the processors that only differ by the bit position corresponding to the dimension of the connection. For example in dimension zero, processing elements zero and one are connected as well as processing elements two and three. Figure 22 shows the connections for hypercube of one to three dimensions. Figure 23 shows the hypercube interconnection for the 64 processor case illustrating all six dimensions. The hypercube layout allows all processors to communicate without using the amount of resources that would be required to have each processor transmit data to all processors directly. Figure 24 illustrates how the six dimensional hypercube organization is mapped to the Stratix FPGA architecture based on the locations of the DSP blocks on the device. The first and second dimensions connect processors that are located within the same DSP block while the rest of the dimensions connect DSP blocks to each other. Since the DSP blocks are organized into columns on opposite sides of the chip, the sixth dimension connects DSP blocks in the different columns.

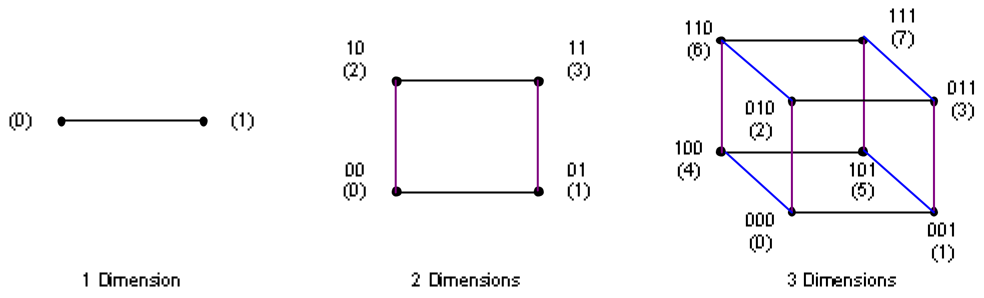

Figure 22: 1 to 3 Dimensional Hypercubes

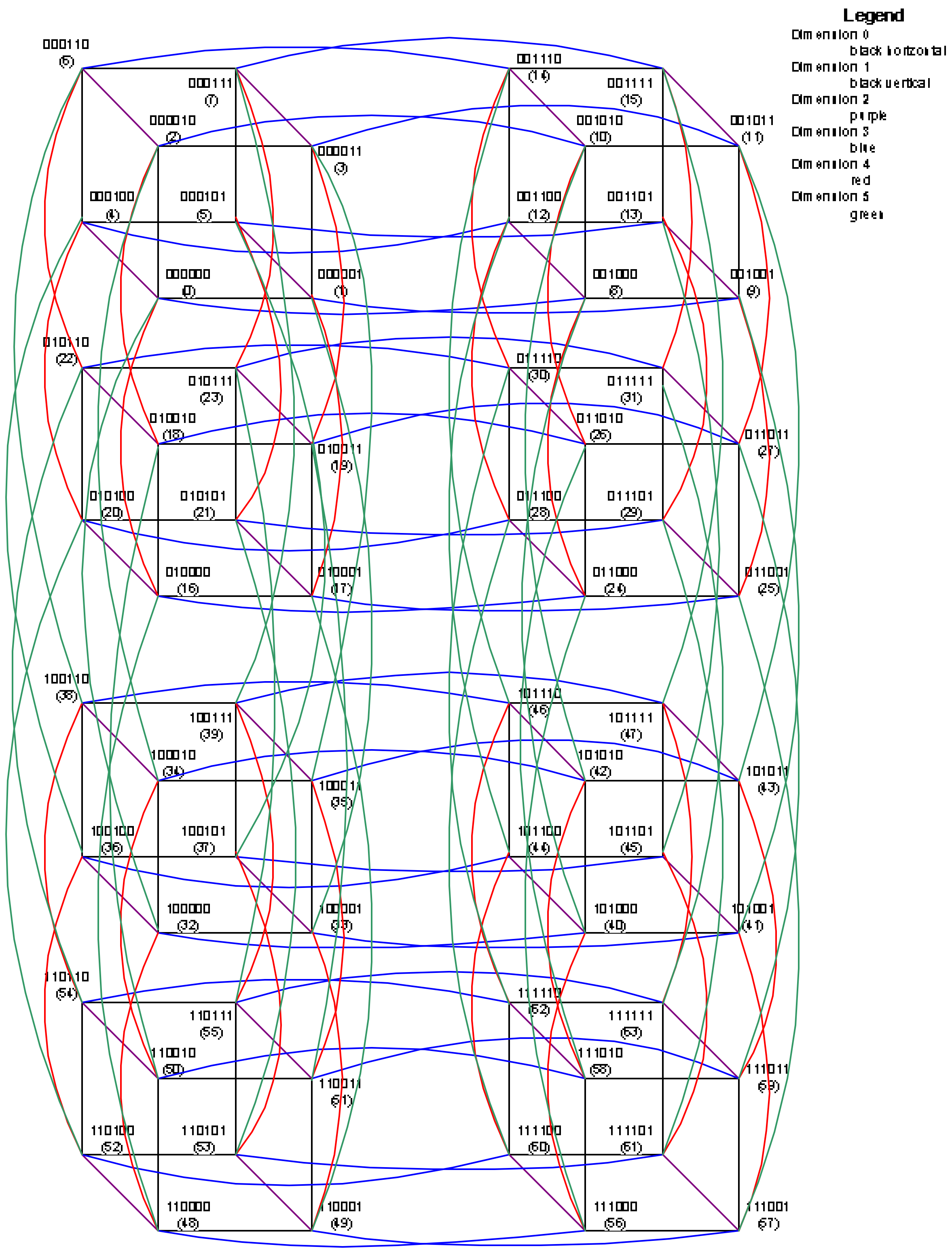

Figure 23: Six Dimensional Hypercube Representation

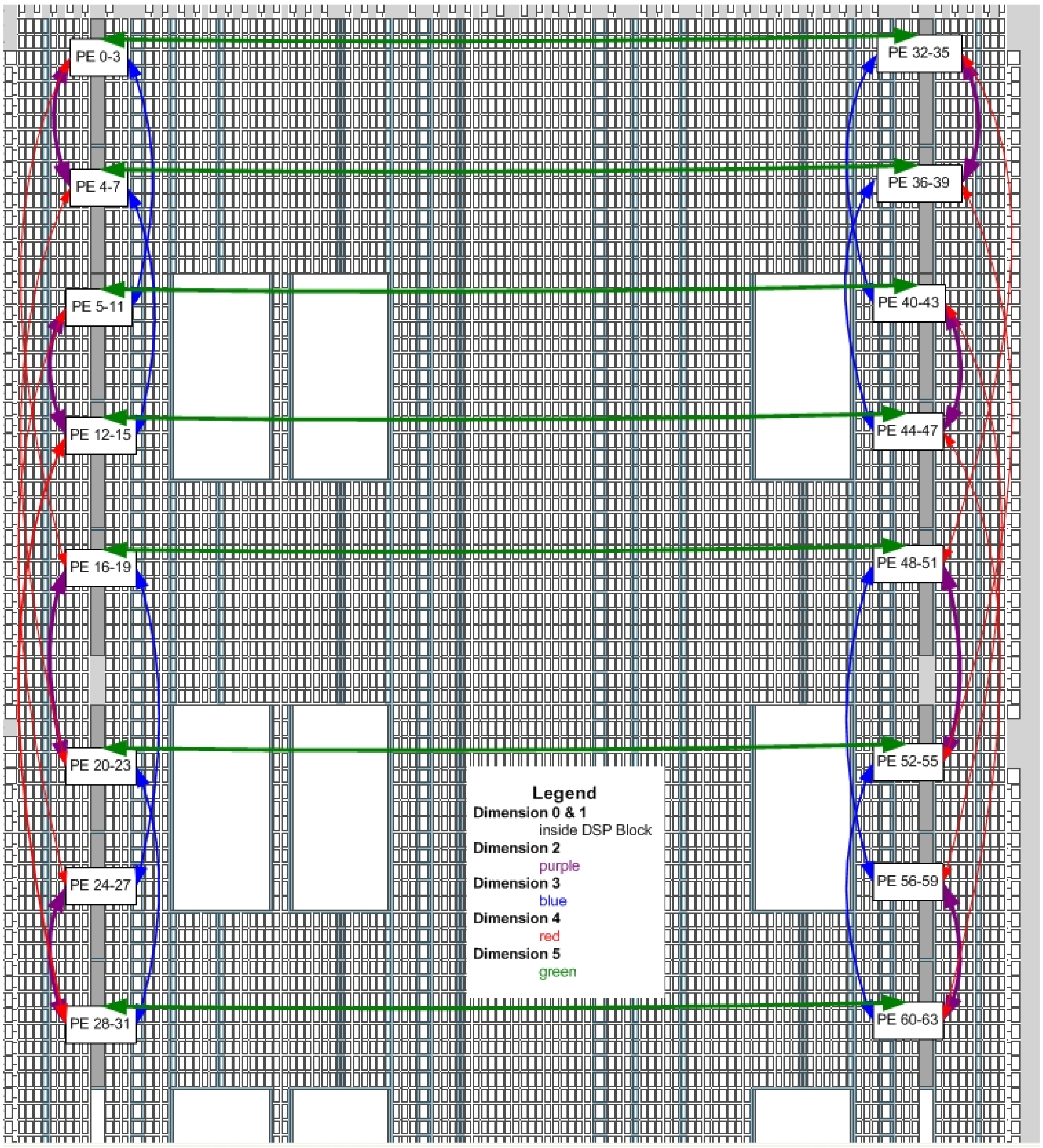

Figure 24: Processor Communication Dimensions in DSP Blocks

Since the register file for the processing elements contains a true dual ported RAM every data transfer instruction moves the contents of two registers. The data transfer process starts with an instruction that writes the data of the two specified registers to the communication lines and takes two clock cycles. The first clock cycle reads the values from the RAM, the second writes that value to a temporary register. The other instructions available for communication are those that choose the dimension of the hypercube to read from. The rest of the instructions listed in Table 6 take three clock cycles to execute. The instruction that writes the data to the temporary register and the instruction that reads the value from the specified dimension were implemented separately to give the user the ability to change the status of the processors in between the write and read operations. Therefore data can be moved from one subset of the processing elements to a different subset of the processing elements. Each of the Move instructions selects the input data from the specified dimension as shown in Figure 25.

Table 6: Communication Instructions

WRITE	Writes the value of a register to the interprocessor communication lines
MOVE0	Reads the communication lines from dimension zero
MOVE1	Reads the communication lines from dimension one *available for 4 or more PE
MOVE2	Reads the communication lines fromdimension two *available for 8 or more PE
MOVE3	Reads the communication lines from dimension three *available for 16 or more PE
MOVE4	Reads the communication lines from dimension four *available for 32 or more PE
MOVE5	Reads the communication lines from dimension five *available for 64 PE

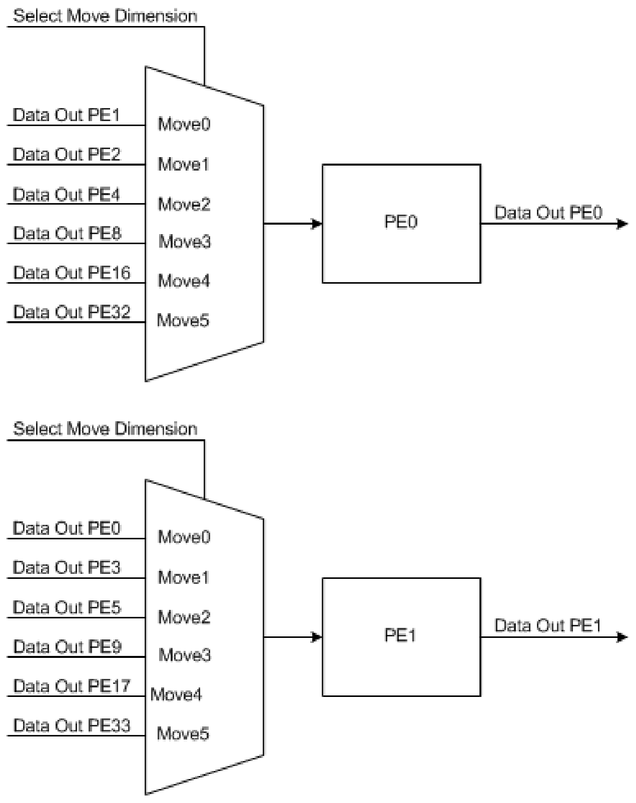

Figure 25: Data Transfer between Processing Elements

6.0 PERFORMANCE RESULTS

In this section the performance and chip utilization results are discussed for all of the architectures. The operation speeds of the prototypes are based on the place and route results from the Quartus II design automation tool and include both logic and routing delays. This section will also discuss the performance of the DSP block designs for applications with a high degree of parallelism such as matrix multiplication and the calculation of the Fast Fourier Transform (FFT).

6.1 LUT Architecture

The operation speeds of the first prototype are shown in Table 7. It shows the logic cell utilization, the memory utilization and the maximum frequency of the prototype. The Altera Stratix device, EP1S40F1020, is targeted for placing 2, 4, 8, 16 and 32 processing elements (PE) and another Stratix device, EP1S80F1508, to implement the 64-PE prototype. The reason the EP1S80 device is chosen to implement 64 processors is because it has more input/output (I/O) pins. Since each processing element has an independent input data path and output data path for its local memory block, the number of the I/O pins on an FPGA device becomes the bottleneck of the performance measurement. This bottleneck is reduced in the next design by multiplexing the output data lines.

6.1.1 Maximum Clock Frequency Analysis

Table 7 : Place and Route Results for Prototypes with 2, 4, 8, 16, 32 and 64 PEs

Altera Stratix EP1S40F1020C5				
	Logic Cells (%)	Memory Bits (%)	Max. frequency	
1 Control Unit			Without Logic Lock	With Logic Lock
2 ALU	1.7 %	0.24 %	99 MHz	99 MHz
4 ALU	2.9 %	0.47 %	95 MHz	96 MHz
8 ALU	5.4 %	0.95 %	97 MHz	96 MHz
16 ALU	10.4 %	1.9 %	95 MHz	98 MHz
32 ALU	20.3 %	3.82 %	90 MHz	95 MHz
Altera Stratix EP1S80F1508C6				
64 ALU	21 %	3.5 %	78 MHz	N/A

The maximum frequency for the prototypes is given with and without using Logic Lock. The frequencies listed without Logic Lock are the default timings produced using the Quartus II tool with the optimization for speed option set, but without any other constraints. The floor plan generated without any constraints for the eight processing element case is shown in Figure 26.

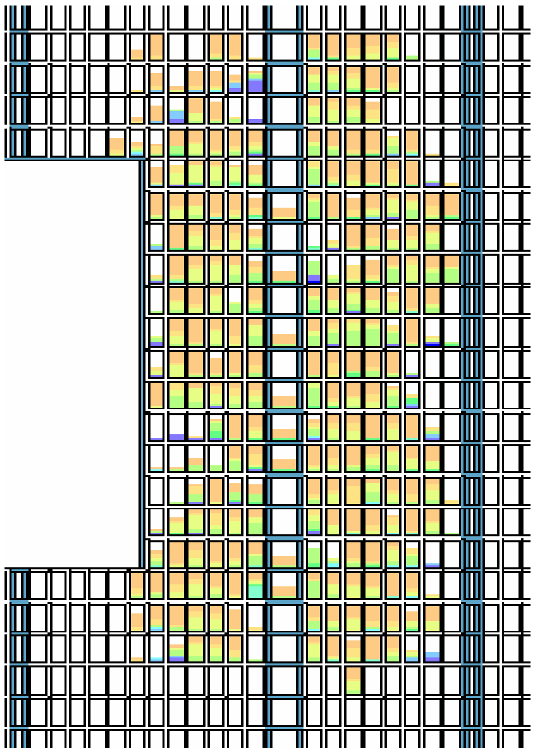
Figure 26: FPGA Floor Plan for 8 PE without Constraining the Logic Placement

According to the place and route result from Quartus II, the critical path is from the ALU to the local RAM from the same PE. The logic lock approach is used to improve the overall design performance [13]. This approach can be utilized to constrain logic structures to a specific region of the floor plan of a device. Since the critical path is from the ALU to the local RAM from the same PE, these two objects can be placed in close logic lock regions. Therefore, this data path is no longer the issue for the critical path. For the multiple-processor design, constraining the ALU to be placed close to its local RAM is important in order to increase the clock speed. Figure 27 illustrates the floor plan for eight processing elements by using the logic lock approach. The middle column in Figure 27 is dedicated for the M4K memory blocks and logic array blocks (LABs) are located beside it.

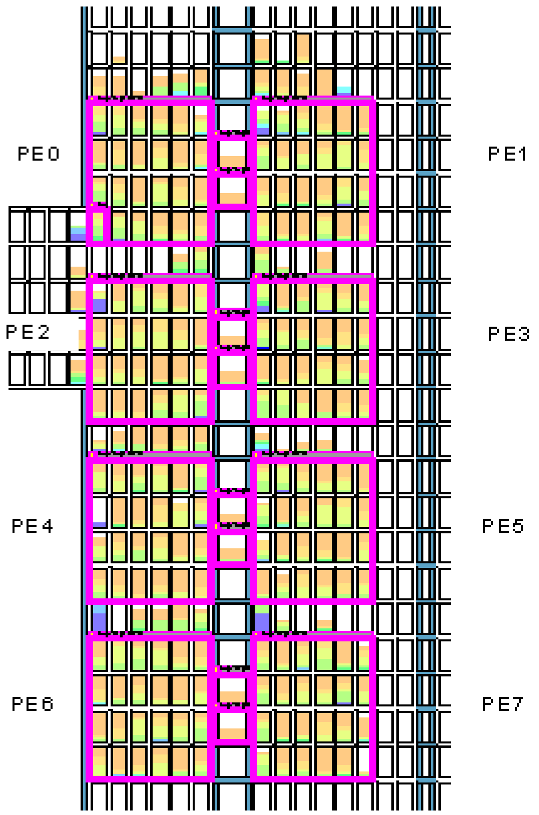

Figure 27: FPGA Floor Plan for 8 PE Constraining the Logic Placement

6.1.2 Chip Utilization

Figure 28 presents the chip utilization of these prototypes. The chip utilization was analyzed for the EP1S80F1508 Stratix device. This FPGA contains 79,040 logic elements, 1211 I/O pins, 364 M4k RAM blocks and 7,427,520 total bits of memory. 64 PE's only used 21% of the chip's total logic elements while using 98% of the chip's I/O pins. Since the bottleneck of this design is I/O support and not logic area, more processors can be implemented in future designs by using parallel to series conversion for the output data buses. The I/O problem can also be eliminated by loading the registers of the processors serially from a main memory and not allowing each processor to communicate off chip.

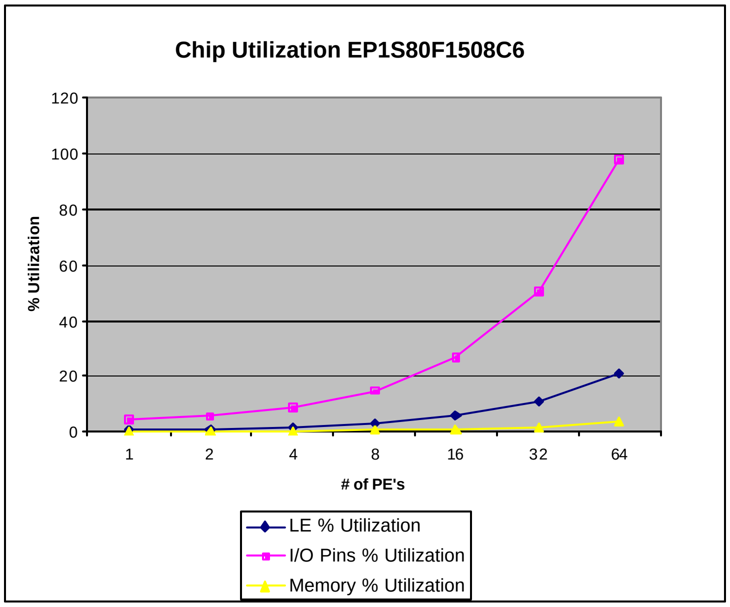

Figure 28: Chip Utilization Results

6.2 DSP Architecture

The target device for placing 2, 4, 8, 16 and 32 PEs is the Altera Stratix EP1S40F1020C5 with the EP1S80F1508 used to implement the 64 and 88 PE prototypes The reason the EP1S80F1508C6 device is chosen to implement 64 or 88 processors is because it has more DSP blocks and more input/output (I/O) pins. The number of DSP blocks on the FPGA device becomes the bottleneck of the design.

6.2.1 Performance Comparison With and Without Register Tree

As mentioned in the architecture section, in order to offset the delay of the control stream, the stream is pipelined by using a register tree. This enables the effect that routing has on the design to be seen since the register tree hides some of this delay. The performance results are compared between the design with the register tree and the design without the register tree to show the tradeoffs between clock speed and resource allocation.

Figure 29 and Figure 30 show that the addition of the register tree increases the clock speed without a significant increase to the number of logic elements used. Table 8 shows the values achieved using Quartus II for place and route of the design with the register tree. The tool options had to be customized as not to do redundant register reduction and compress the register tree into a single register.

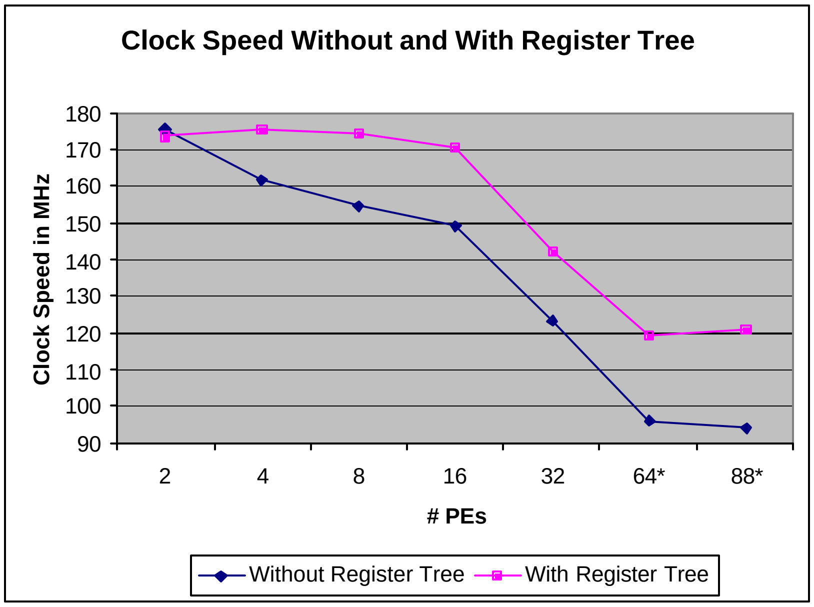

Figure 29: Clock Speed with and without Register Tree

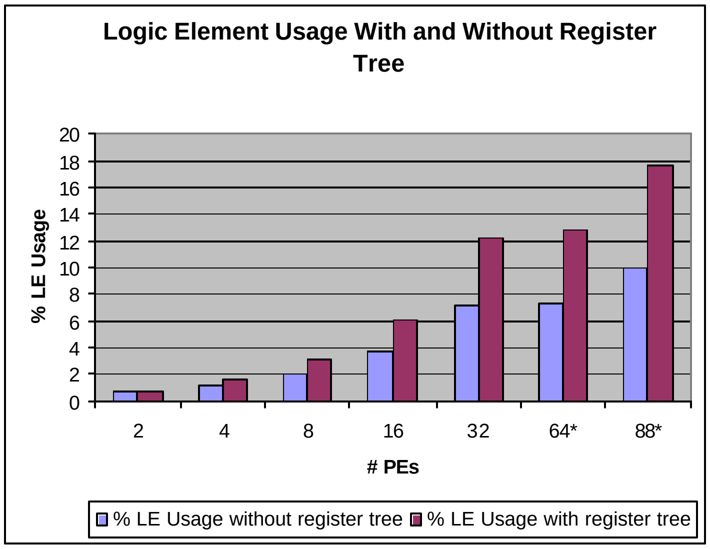

Figure 30: Logic Element Usage with and without Register Tree

Table 8: Post Place-and-Route Resource Utilization for 2, 4, 8, 16, 32, 64, and 88 Processing Elements.

Altera Stratix EP1S40F1020C5				
	Logic Cells (%)	**Memory Bits (%)**	**DSP Blocks (%)**	**Max. frequency**
1 Control Unit				**Default (MHz)**
2 ALU	0.85 %	0.24 %	3.57 %	173
4 ALU	1.67%	0.48 %	7.14 %	175
8 ALU	3.12%	0.96 %	14.29 %	174
16 ALU	6.15 %	1.93 %	28.57 %	170
32 ALU	12.24 %	3.87 %	57.14 %	141
Altera Stratix EP1S80F1508C6				
64 ALU	12.72 %	3.58 %	72.73 %	119
88 ALU	17.64 %	4.92 %	100 %	121

6.2.2 DSP Blocks vs. LUT Implementation

In order to illustrate the performance advantages gained by using the DSP blocks, the same architecture was also implemented by using standard look-up table logic on the same device. This was done by changing the mapping of the Altera MegaWizard ALU block from

DSP block to logic elements. This architecture is not compared to the LUT architecture described in the previous section because that architecture does not implement multiplication and would therefore not be an accurate comparison. The maximum clock speed is compared between the two designs in Figure 31.

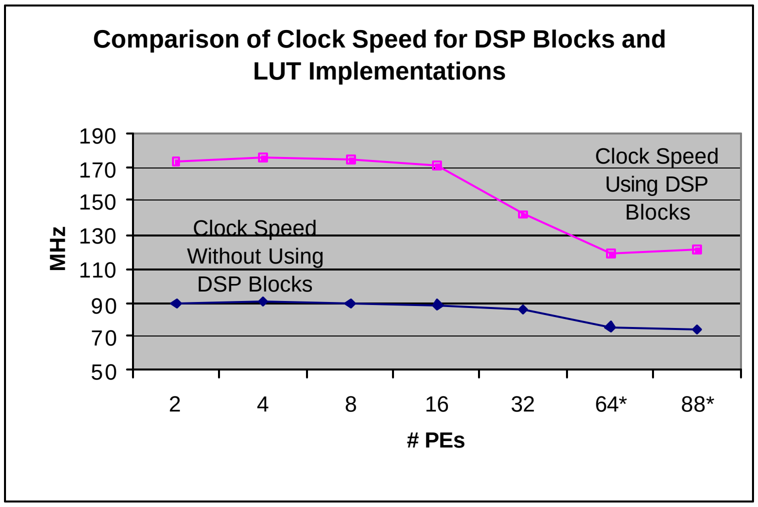

Figure 31: Clock Speed Comparison

After the initial place and route results that gave the maximum clock frequencies above were analyzed for the DSP block implementation it was determined that the critical path was from the ALU to the local memory of the same processing element. In order to achieve a higher maximum clock frequency the local memories were constrained to be adjacent to their respective processing element. The critical path for 64 processing elements before the constraints is shown in Figure 32. Figure 33 shows the constrained placement of the processing elements and their local memories. These place and route constraints increased the maximum clock frequency of the 64 processing element design from 119 MHz to 143 MHz and the of the 88 processing element design from 121 MHz to 128 MHz.

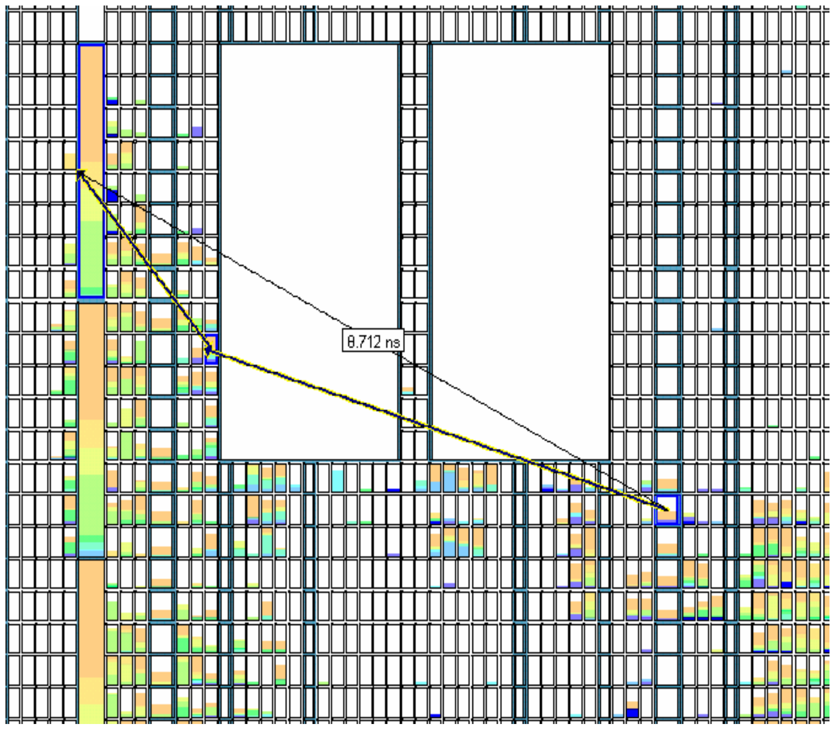

Figure 32: Unconstrained Critcal Path for 64 Processing Elements

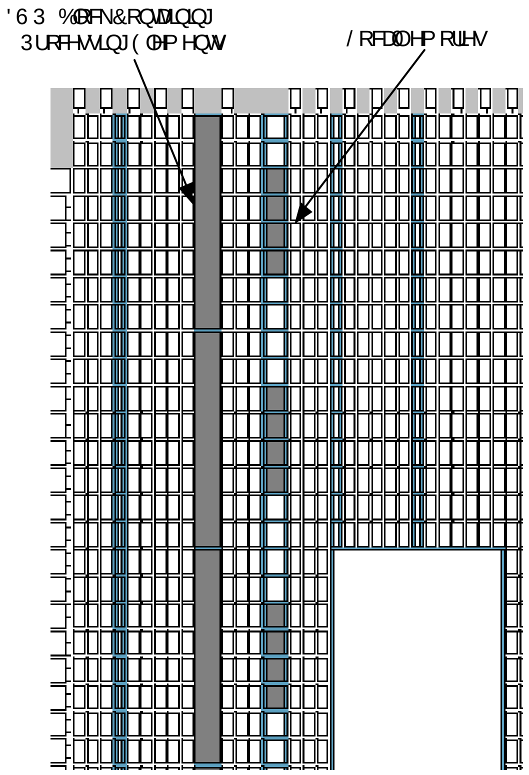

Figure 33: Constraining the Location of Local Memories

The number of logic elements available for custom instructions is compared between the two implementations in Figure 34. It is shown that using the DSP blocks allows for 87% of the chip to be available for custom instructions in the case of 88 processors whereas the LUT implementation uses half of the device area.

Since the 88 processors are implemented on the EP1S80F1508C6, this percentage increase is equivalent to over 26,000 logic elements. Not only are there thousands more logic elements available for user defined instructions, but each processor in the DSP design runs 53 MHz faster. The aggregate speed increase across the SIMD array is 4.6 GHz when using the DSP blocks.

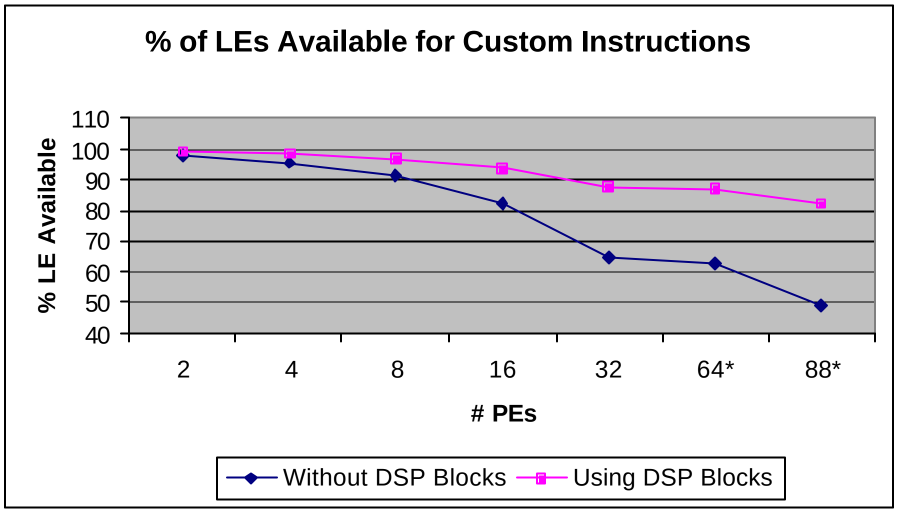

Figure 34: Percentage of Logic Elements Available for Custom Instructions

6.2.3 Matrix Multiplication Example

This prototype is an ideal platform for the calculation of matrix multiplication. The inclusion of an *a*b + c*d* operation in the instruction set allows for matrix multiplication to be completed in few instructions. For an n x n matrix there are n^2 elements that need to be calculated. Each element requires that n multiplies be computed and $n-1$ additions. Using the *a*b+ c*d* instruction, two of the multiplications and one of the additions can be calculated in a single instruction. Letting y equal the number of these instructions required,

$$y = \lfloor n / 2 \rfloor \qquad \text{(Eq 1)}$$

After the use of these instructions, there are still $n \bmod 2$ multiplications left over and $n–1–y$ additions to be performed. Therefore, the total instructions required for the calculation of each element (i_e) in the result matrix is

$$i_e = n - 1 + n \bmod 2 \qquad \text{(Eq 2)}$$

Since there are n^2 elements to be calculated the total instructions (i_t) required for a single processor is

$$i_t = i_e * n^2 \qquad \text{(Eq 3)}$$

Letting p equal the number of processing elements in the architecture being used, the number of instructions can be divided amongst the processors to be completed in parallel up to the total number of elements to be calculated. For example, if there are only 4 elements to be calculated, the use of 8 processors does not increase the performance of using 4 processors because only 4 of the 8 processor array will be used and the rest will remain idle during the calculation. Therefore the total number of instructions for a specific number of processing elements (i_p) is

$$i_p = \left\lfloor (i_t / (\min(p, n^2)) \right\rfloor \qquad \text{(Eq 4)}$$

The number of clock cycles required to execute the matrix multiplication (C) since each instruction takes six clock cycles plus the extra clock cycles required to propagate through the register tree is

$$C = i_p * 6 + \lceil \log_2 p \rceil \qquad \text{(Eq 5)}$$

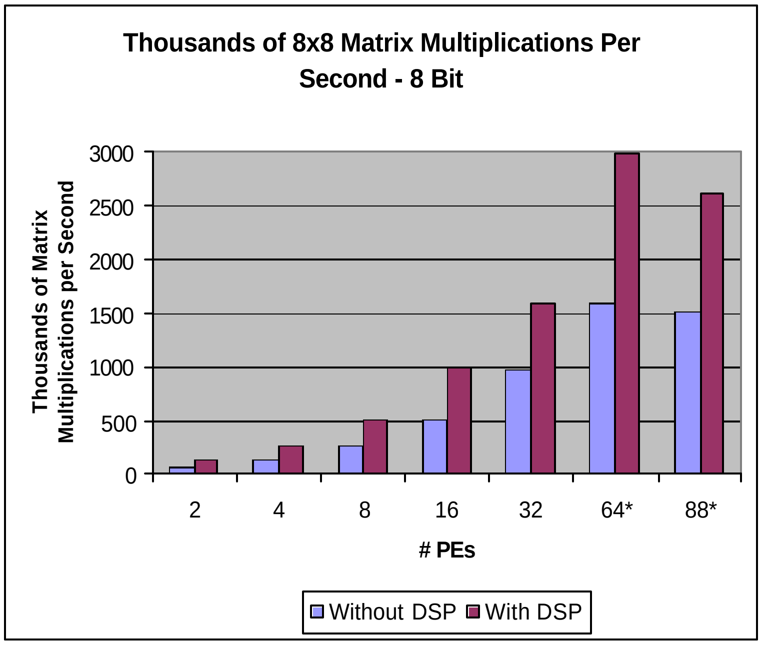

Figure 35: Thousands of 8x8 Matrix Multiplications per Second

Figure 35 shows that using the DSP blocks allows for almost 1.5 million more matrix multiplications per second in the DSP implementation over the LUT design when using 64 processors.

6.2.4 Space Utilization

Figure 36 presents the chip utilization of these prototypes. The chip utilization was analyzed for the EP1S40F1020C5 Stratix device for 2 through 32 processing elements and for the EP1S80F1508C6 for 64 or 88 processing elements. 88 PE's only used 17% of the chip's total logic elements while using 91% of the chip's I/O pins, and 100% of the DSP blocks. Figure 37 shows the post place and route floor plan for the LUT implementation of the ALU. The post place and route floor plan for the DSP Block implementation is shown in Figure 38. Since the bottleneck of this design is DSP block support and not logic area, custom instruction instructions can be implemented with 83% of the logic still available.

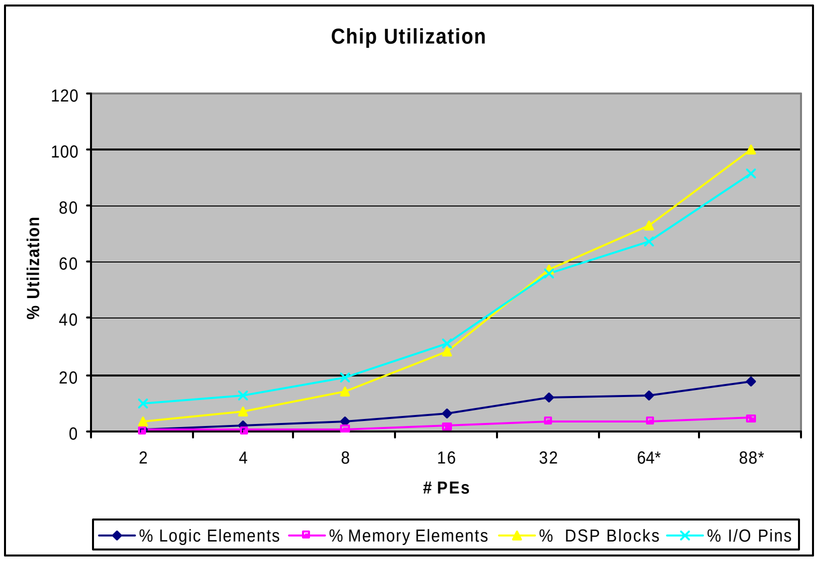

Figure 36: Utilization Results

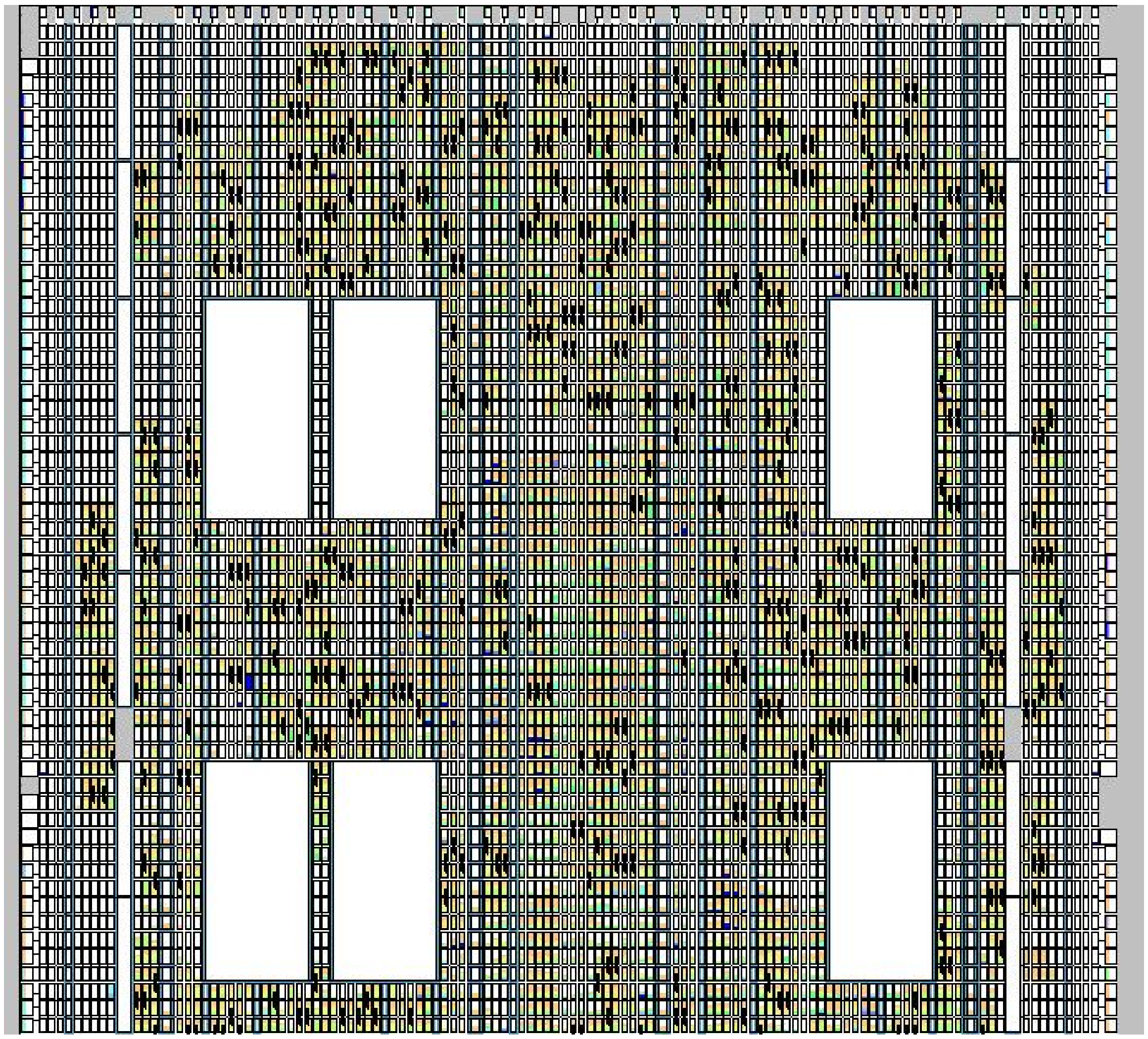
Figure 37: Post Place and Route Floor Plan of 88 PE (LUT)

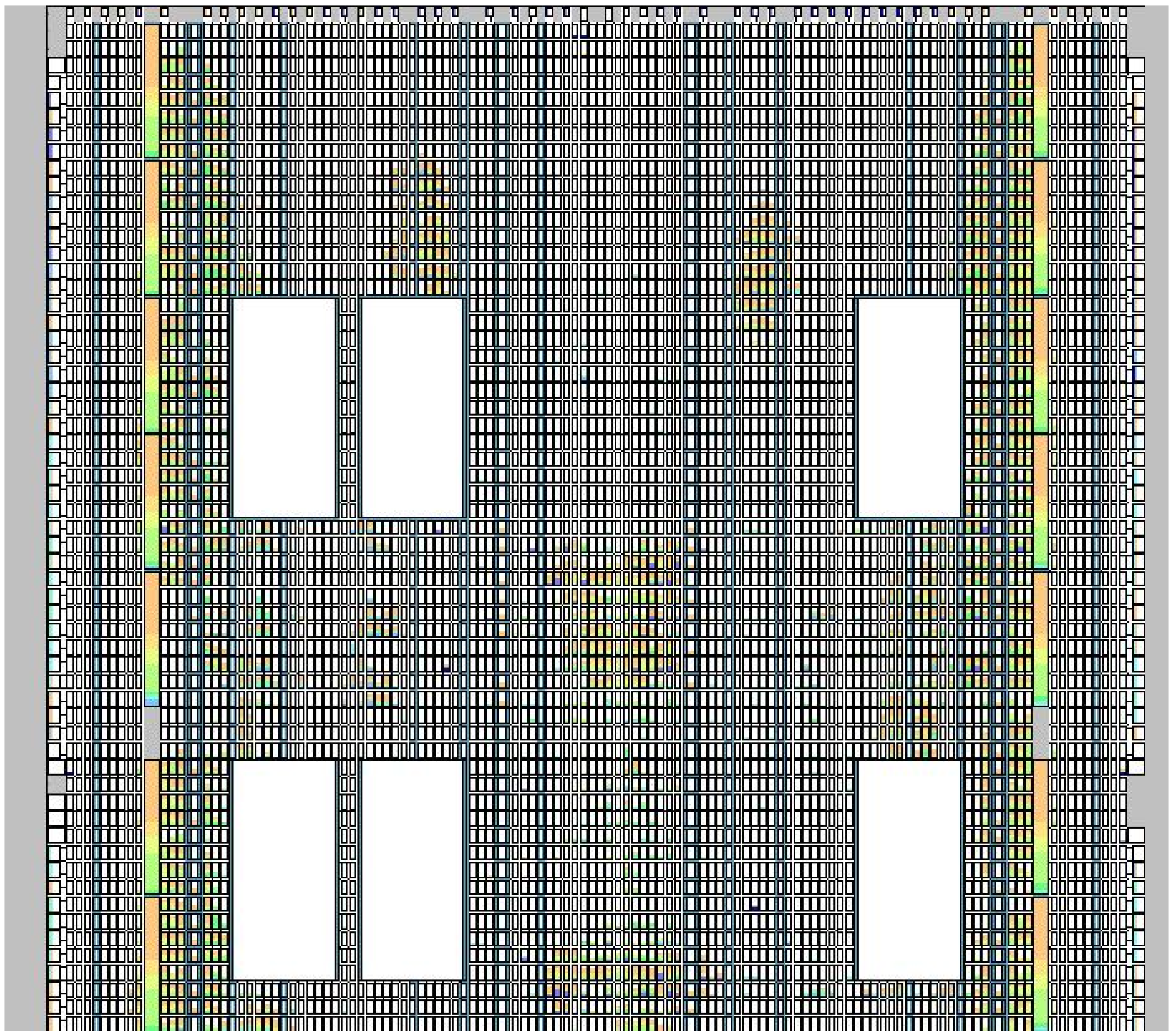

Figure 38: Post Place and Route Floor Plan of 88 PE (DSP)

6.3 DSP Architecture with Communication

The Altera Stratix EP1S40F1020C5 is the target device for placing 2, 4, 8, 16 and 32 PEs and the EP1S80F1508 is used to implement the 64 PE prototype. The EP1S40F1020C5 does not have enough DSP Blocks to allow for the implementation of the 64 processing elements. Since 88 is not a power of two, it can not be mapped to the hypercube communication scheme and therefore is not implemented in this case. The availability of the communication scheme allows for more complex algorithms to be executed on the design such as the FFT.

6.3.1 Maximum Clock Frequency Analysis

Table 9 shows the post place and route resource allocation and maximum frequency results achieved for the architectures with the hypercube communication implementation and the first controller. Table 10 shows the post place and route data for the communication architecture with the second control unit. A graph in Figure 39 of the maximum clock frequency is shown for this design with the second control unit compared to the maximum frequency achieved with the previous DSP architecture. As shown, the addition of the communication network causes the maximum frequency to increase. The values in Table 9 and Table 10 are a baseline for future improvements in the communication of the processing elements. Table 11 shows the breakdown of the delay between logic cells and interconnections. As shown in Figure 40, the interconnect delay accounts for a large majority of the delay, over 80 percent in all cases. One possible solution to reduce this delay is to add an increasing number of registers to each of the hypercube communication dimensions. In this case, each higher dimension will require more clock cycles, but the overall clock speed will be increased.

Table 9: Post Place-and-Route Resource Utilization First Control Unit

Altera Stratix EP1S40F1020C5				
	Logic Cells (%)	**Memory Bits (%)**	**DSP Blocks (%)**	**Max. frequency**
1 Control Unit				**Default (MHz)**
2 ALU	1.57 %	0.27 %	3.57 %	120
4 ALU	2.42 %	0.54 %	7.14 %	136
8 ALU	4.94 %	1.08 %	14.29 %	122
16 ALU	9.72 %	2.17 %	28.57 %	119
32 ALU	19.77 %	4.34 %	57.14 %	113
Altera Stratix EP1S80F1508C6				
64 ALU	24.03 %	4.01 %	72.73 %	97

Table 10: Post Place-and-Route Resource Utilization First Control Unit

Altera Stratix EP1S40F1020C5				
	Logic Cells (%)	**Memory Bits (%)**	**DSP Blocks (%)**	**Max. frequency**
1 Control Unit				**Default (MHz)**
2 ALU	1.60 %	0.27 %	3.57 %	111
4 ALU	2.40 %	0.54 %	7.14 %	134
8 ALU	5.04 %	1.08 %	14.29 %	129
16 ALU	9.71 %	2.17 %	28.57 %	124
32 ALU	19.75 %	4.34 %	57.14 %	109
Altera Stratix EP1S80F1508C6				
64 ALU	24.02 %	4.01 %	72.73 %	94

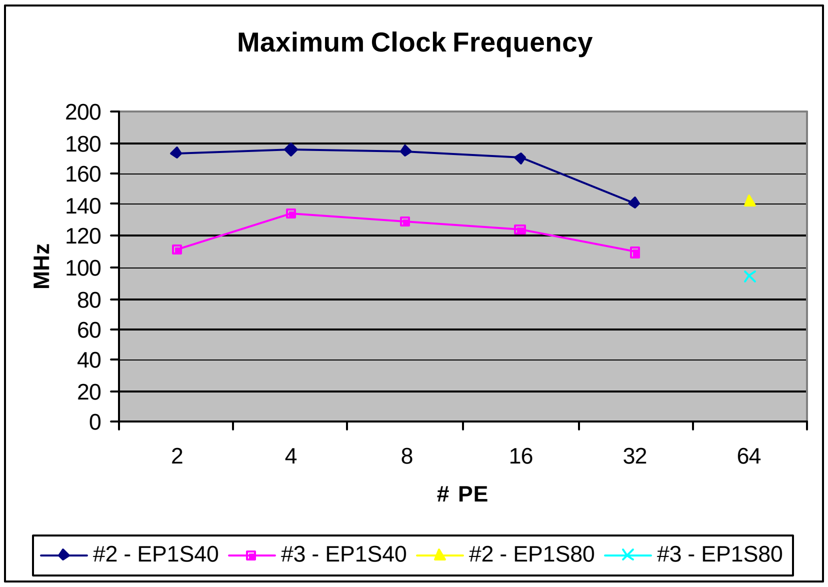

Figure 39: Maximum Frequency Analysis For 1st and 2nd DSP Architectures

Table 11: Total Delay Divided Between Cells and Interconnect 2nd Controller

# PE	Delay(us)	Cell Delay (us)	% Cell	Interconnect (us)	% Interconnect
2	8.717	1.436	16.5	7.281	83.5
4	7.07	1.156	16.4	5.914	83.6
8	7.341	1.33	18.1	6.011	81.7
16	7.753	1.275	16.4	6.478	83.6
32	8.807	0.964	10.9	7.843	89.1
64	9.971	1.662	16.7	8.309	83.3

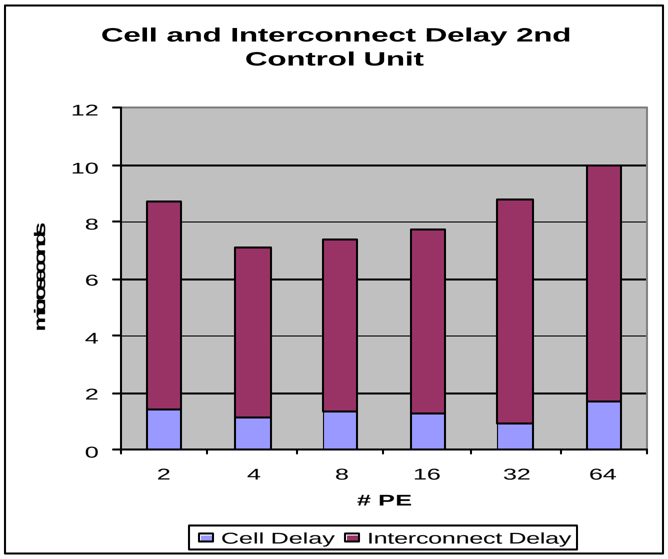

Figure 40: Cell and Interconnect Delay for Second Control Unit Design

6.3.2 Chip Utilization

The chip utilization of these prototypes using the first control unit is shown in Figure 41 and using the second control unit is shown in Figure 42. The chip utilization was analyzed for the EP1S40F1020C5 Stratix device for 2 through 32 processing elements and for the EP1S80F1508C6 for 64 processing elements. The post place and route floor plans for the implementation with the second controller are shown in Appendix A. Despite which controller is used, 64 PE's only used 24% of the chip's total logic elements while using 80% of the chip's I/O pins, and 73% of the DSP blocks. The bottleneck of this design is DSP block and not the logic elements. This design implements 64 processing elements with a fully connected communication pattern and still has 76% of the device available for custom instructions.

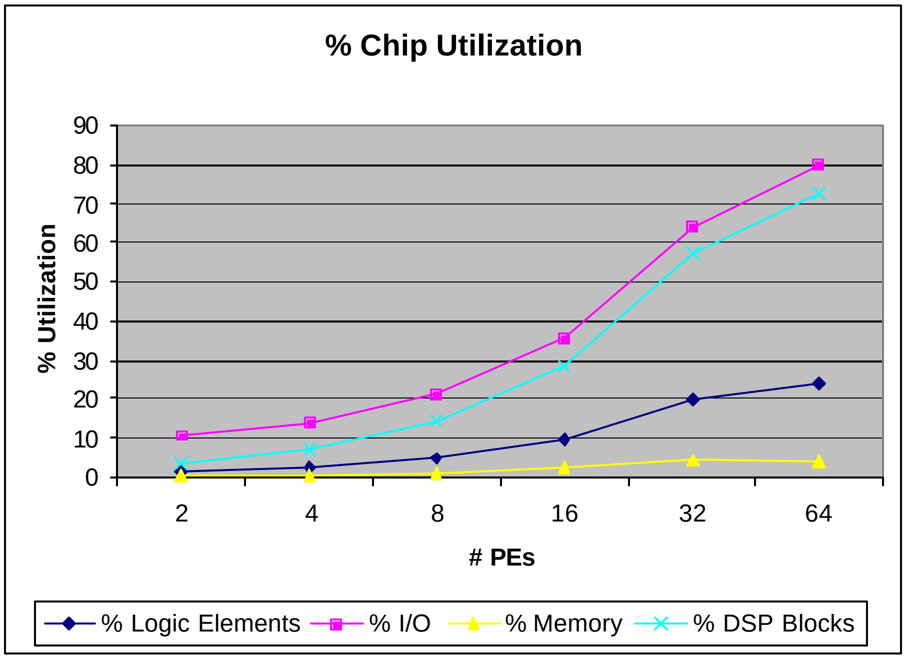

Figure 41: Chip Utilization for First Control Unit

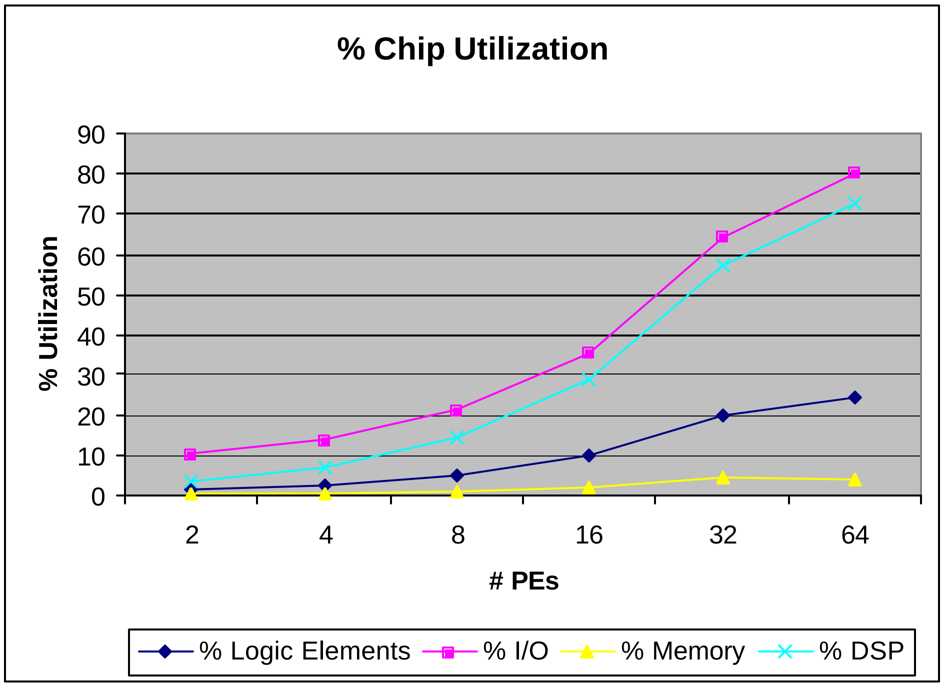

Figure 42: Chip Utilization for Second Control Unit

6.3.3 Application: Fast Fourier Transform

The Fast Fourier Transform (FFT) function can be implemented on the 64 processing element prototype. This algorithm features a high degree of parallelism and the high number of complex multiplications take advantage of the availability of the *a*b+c*d* and *a*b-c*d* instructions. The FFT algorithm is a method to quickly calculate the Discrete Fourier Transform (DFT). The DFT takes and input sequence of x(n) in the time domain and returns a sequence X(k) in the frequency domain. The relationship between X(k) and x(n) is:

$$X(k) = \sum_{j=0}^{N-1} x(n) \cdot e^{\frac{-j2\Pi k}{N}} \qquad \text{(Eq 6) [20]}$$

where N is the number of points in the DFT, the FFT algorithm can be used when N is a power of 2. This algorithm is calculated using the butterfly graph shown in Figure 43, in order to calculate each butterfly graph using the hypercube connected processing elements, the graph is reordered as shown in Figure 44[15].

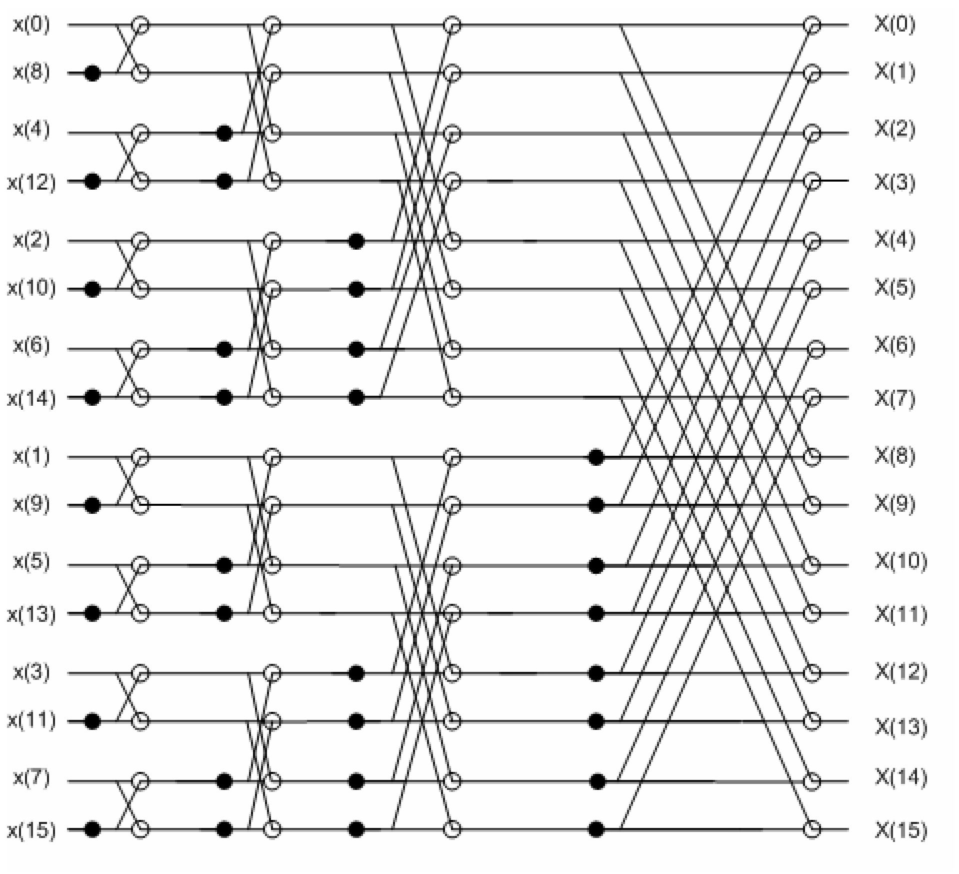

Figure 43: Butterfly Graph for 16 Point FFT [15]

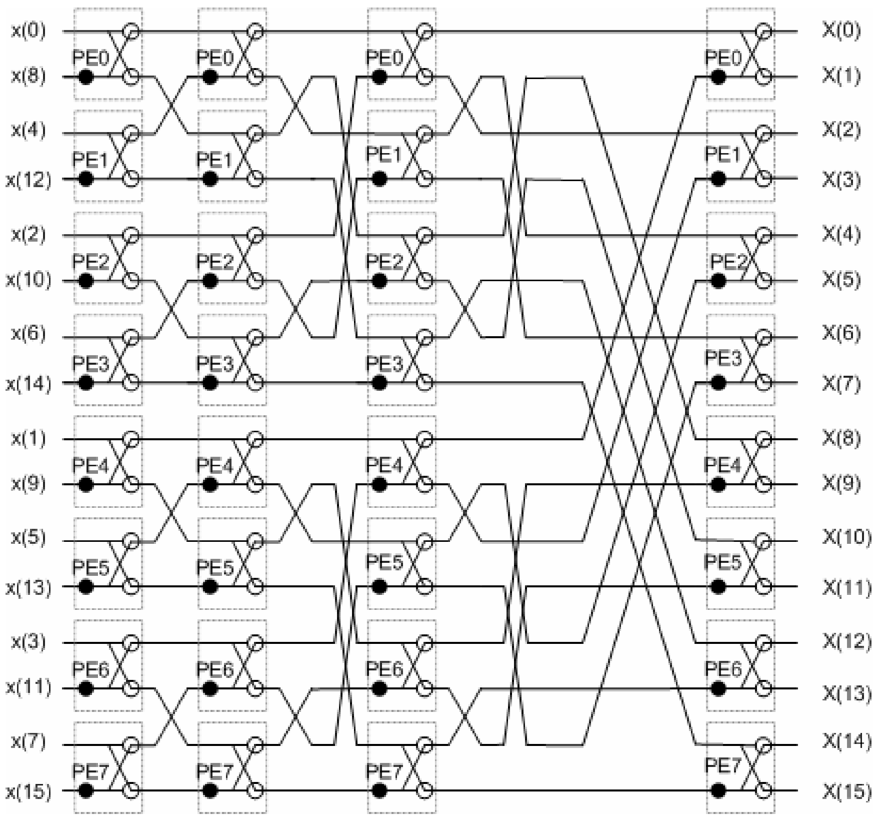

Figure 44: Reordered Butterfly Graph for 16 Point FFT [15]

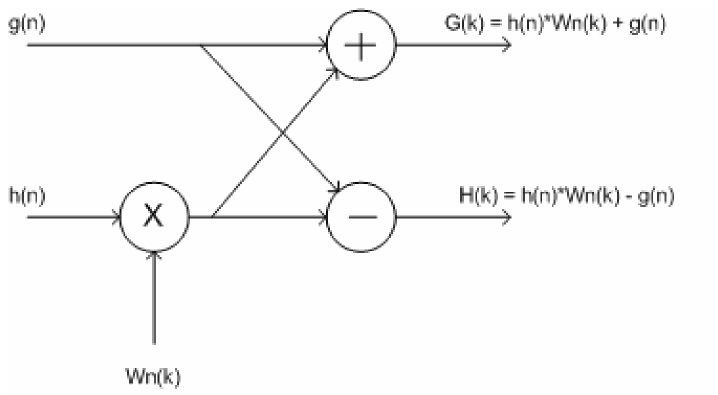

Figure 45: Detailed View of FFT Butterfly Calculation

The prototype will calculate the FFT on signed numbers with a nine bit real part and nine bit imaginary part. The input vector is scaled so that each $-1 < x(n) < 1$. As shown in the detailed butterfly graph in Figure 45. If the input vector is scaled, the values of h(n) and g(n) can increase by a factor of two at most during each butterfly calculation [16]. Due to this, the possibility of overflow can be eliminated by shifting the values at the end of each butterfly calculation to the right one place. The final result will give X(k) scaled by N. In the prototype, one of the custom instructions is used to shift to the right one place. The equations for each of the butterflies are:

$$g(n) = re[g(n)] + j\ img[g(n)] \qquad \text{(Eq 7)}$$

$$h(n) = re[h(n)] + j\ img[h(n)]$$

$$Wn = re[Wn(k)] + j\ img[Wn(k)]$$

Where $Wn(k) = e^{\frac{-j2\Pi k}{N}}$ and is pre-calculated and loaded into the processing elements prior to execution. Table 12 shows the instructions used to calculate the Butterfly graph and number of clock cycles required for each of control units described for the second DSP architecture. Since there are 64 processing elements, 64 butterfly calculations can be executed in parallel resulting in the calculation of a 128 point FFT.

Table 12: Instruction to Execute FFT Butterfly

Butterfly Instructions	**clock cycles control #1**	**clock cycles control #2**
re[h(n)]* re[Wn(k)] - img[h(n)]* img[Wn(k)] => re[A]	6	6
re[h(n)]* img[Wn(k)] + re[Wn(k)]* img[h(n)] => img[A]	6	6
shift re[g(n)] => re[g(n)]	5	4
shift img[g(n)] => img[g(n)]	5	4
shift re[A] => re[A]	5	4
shift img[A] => img[A]	5	4
re[g(n)] + re[A] => re[G(k)]	5	5
img[g(n)]+ img [A] => img[G(k)]	5	5
re[g(n)] - re[A] => re[H(k)]	5	5
img[g(k)] – img [A] => img [H(k)]	5	5
Total Clock Cycles	**52**	**48**
Time per Clock Cycle (µs)	0.0103	0.0106
Calculation Time per Butterfly (µs)	**0.536**	**0.509**

After the calculation of the first butterflies, the data needs to be exchanged across the first hypercube dimension. This transfer is done according to Figure 44. The instructions needed to achieve this transfer are shown in Table 13. After this series of instructions, the data is in the correct registers to repeat the butterfly calculations.

Table 13: Instructions to Move Data in Dimension 0 for FFT

Instructions to Move Data in Dimension 0 for FFT	**clock cycles control #1**	**clock cycles control #2**
Copy re[H(k)]	5	5
Copy img[H(k)]	5	5
Write G(k)	2	2
Disable odd # processing elements (write to status register)	2	2
Move Dimension 0	3	2
Write H(k)	2	2
Enable even PE and Disable odd PE	2	2
Read dimension 0	3	2
Enable all processing elements	2	2
Total Clock Cycles	**26**	**24**
Time per Clock Cycle (μs)	0.0103	0.0106
Calculation Time per Data Transfer (μs)	**0.268**	**0.254**

After the second butterfly is calculated the data needs to be moved over the first dimension as described above. Next, the data is moved over the second dimension using the same procedure with the exception that different processing elements are enabled or disabled during the write to status register instruction. The total number of clock cycles required to execute the 128 point FFT is shown in Table 14. It is shown in this table that though the second control unit operates at a slower maximum frequency, the reduction of clock cycles allows it to calculate the 128 point FFT more quickly than the first controller.

Table 14: Instructions Sets for Calculation of 128 Point FFT

Instruction Sets for Calculation of 128 pt FFT	**clock cycles control #1**	**clock cycles control #2**
Butterfly Instruction Set	52	48
Move Dimension 0	26	24
Butterfly Instruction Set	52	48
Move Dimension 0	26	24
Move Dimension 1	26	24
Butterfly Instruction Set	52	48
Move Dimension 0	26	24
Move Dimension 1	26	24
Move Dimension 2	26	24
Butterfly Instruction Set	52	48
Move Dimension 0	26	24
Move Dimension 1	26	24
Move Dimension 2	26	24
Move Dimension 3	26	24
Butterfly Instruction Set	52	48
Move Dimension 0	26	24
Move Dimension 1	26	24
Move Dimension 2	26	24
Move Dimension 3	26	24
Move Dimension 4	26	24
Butterfly Instruction Set	52	48
Move Dimension 0	26	24
Move Dimension 1	26	24
Move Dimension 2	26	24
Move Dimension 3	26	24
Move Dimension 4	26	24
Move Dimension 5	26	24
Butterfly Instruction Set	52	48
Total Clock Cycles	**910**	**840**
Time per Clock Cycle (μs)	0.0102	0.0106
Calculation Time per 128 pt FFT (μs)	**9.282**	**8.904**

This architecture is able to calculate a 128 point FFT in 8.93 μs. The multiple FPGA architecture described in [15] takes 50 μs to execute the same algorithm. The use of the 64 processing array also introduces increased precision with the use of 9 bit number instead of 8 bit. The 128 point FFT can be done completely in parallel using the DSP block architecture presented in this thesis. An FFT with more than 128 points can be calculated by executing multiple 128 point FFTs, then moving the data accordingly and executing additional butterfly calculations. The time required to compute FFTs of 128, 256, 512, and 1024 points is shown for this architecture and the multiple FPGA architecture in [15].

Table 15: Compute Time Comparison for FFTs of 128, 256, 512, and 1024 points

Number of Points	Compute Time (μs) for 64 interconnected PE	Compute Time (μs) for architecture in [15]
128	9	50
256	21	130
512	44	290
1024	90	640

Many different companies offer commercially available FFT processors. Altera has an FFT MegaCore function [22] that is optimized to use the Stratix DSP blocks. When implemented on the EP1S10F780C5 the 1024 point radix 2 FFT requires 22 μs to execute. The 1024 point FFT can be calculated on a 400 MHz Motorola MPC7400 in 63 μs [23]. In this case, the FFT is fixed-point and makes use of the AltiVec vector technology in the MPC7400. If the 1024 point FFT was calculated on the 400 MHz Motorola MPC7400 using only scalar instructions, it would take 453 μs. Zarlink Semiconductor offers a stand alone FFT processor PDSP16515A that can compute a 1024 point FFT in 87 μs [24].

The 64 processing element hypercube interconnected array performs the FFT only 3 μs slower than the stand alone FFT chip from Zarlink Semiconductor, but 4.2 times slower than the

IP core from Altera for the Stratix FPGA. The FFT MegaCore function is designed to only implement the FFT and does not offer the same flexibility for other instruction sets as the 64 processing element array presented in this paper. The introduction of the a pipelined data path for this architecture would increase the instruction throughput by up to a factor of 5 allowing it to compete with the FFT MegaCore function as well as retain its ability to execute custom instruction sets.

7.0 CONCLUSIONS AND FUTURE DIRECTIONS

Since the architecture of current FPGAs contain more than 100,000 logic elements, over 500 4k-bit memories, and over one hundred embedded multipliers there is the possibility for massive parallelism. The SIMD architecture presented in this paper gives a baseline for the potential of parallel processing in an FPGA. The contributions to the area of parallel processing presented in this paper include:

- The use of an FPGA as a platform for parallel processing.
- The design of a SIMD array of processing elements that exploits the increased performance of embedded ASIC multipliers over the look up table logic. The instruction set for this architecture includes an $a*b + c*d$ instruction that takes one clock cycle to execute. The multiply and add instruction is useful for applications such as matrix multiplication and the multiplication of complex numbers.
- A register tree was added to reduce the affect the delay due to interconnect. The addition of the register tree increased the maximum clock frequency on average by 17 MHz per processing element. The logic available for custom instructions was reduced by 7.6% in the worst case and by only 3% on average.

- The same architecture was implemented using both logic cells and the embedded multiplier blocks. For the 2, 4, 8, 16, 32, 64, 88 processing element architectures, the maximum clock frequency was increased on average by 73 MHz by using the embedded multiplier blocks. The area available for custom instructions was increased by an average of 14.5% of the target device. In the case of 88 processing elements, the clock frequency increase was 52 MHz and the logic elements available were increased by 33%.

- The instruction set for an 8x8 matrix multiplication requires only 7 instructions when using 64 processing elements. Using the first DSP block architecture, 64 processing elements operate at 143 MHz, this allows for over 2,900 Million operations per second to be calculated when computing an 8x8 matrix multiplication.

- A 64 processing element array designed is connected in a hypercube communication scheme that allows each processor to transmit data to every other processing element in a maximum of six transfers.

- For the hypercube connected architecture, two different control unit structures were evaluated. The first control design required an extra clock cycle for custom instructions and an extra clock cycle to transmit data over the hypercube dimensions. This control unit allowed the 64 processing element architecture to operate at a maximum frequency of 97 MHz. The second controller for this design with the elimination of one clock cycle for custom instructions and one clock cycle for each of the move instructions operated at a maximum frequency of 94 MHz. Through the FFT benchmark, it is shown that the second control design executes the instructions more quickly due to the clock cycle reduction. The number of clock cycles saved out weighs the slower frequency for the FFT instruction set.

- The architecture contains the flexibility to augment the instruction set with application specific instructions. The hypercube connected 64 processing element design has 76% of the FPGA logic cells available for the implementation of these custom instructions.

- The hypercube connection allows for an FFT to be calculated on the processing element array. The 64 processing element prototype allows for a 128 point FFT to be computed completely in parallel. FFTs with a higher number of points can be calculated by breaking them up in to groups of 128 points and then rearranging the data to compute the additional butterfly calculations.

A weakness to the current design is that each multiply and add or subtract operation takes six clock cycles, all other arithmetic instructions take five clock cycles, and each custom operation takes four clock cycles to complete. The implementation of a pipeline ALU to improve performance should be addressed in future designs. A pipelined ALU could, however, increase the size of each ALU leaving less room for custom instructions.

The current clock speed achieved could also be increased by hand optimizing the placement of logic elements in Quartus II using their Logic Lock tool [13]. Another way to increase the maximum clock frequency is reduce the interconnect delay. This delay can be reduced by the introduction of an increasing number of registers added to each dimension of the hypercube communication scheme. The addition of these registers will make each higher communication dimension more costly and therefore would require the complier to limit the communication at higher dimensions as much as possible.

The limiting factor for this design is the number of DSP blocks available on the target Stratix FPGA. If the *a*b+c*d* instruction is eliminated, the adder output stage of the DSP block would not be used and twice as many processing elements could be implemented on a single device. Since the adder output stage within the DSP block would not be used, an add/subtract unit would need to be implemented in logic elements. The resource allocation and performance will be analyzed compared to the current design. The removal of the *a*b + c*d* instruction could hurt performance in applications such as the FFT and matrix multiplication. The implementation of the adder in look-up table units will reduce the amount of logic available for custom instructions. Another area to explore for increased parallelism is the use of future FPGA's that have more embedded multipliers.

APPENDIX A

Post Place and Route Floor Plan Views Showing Critical Path

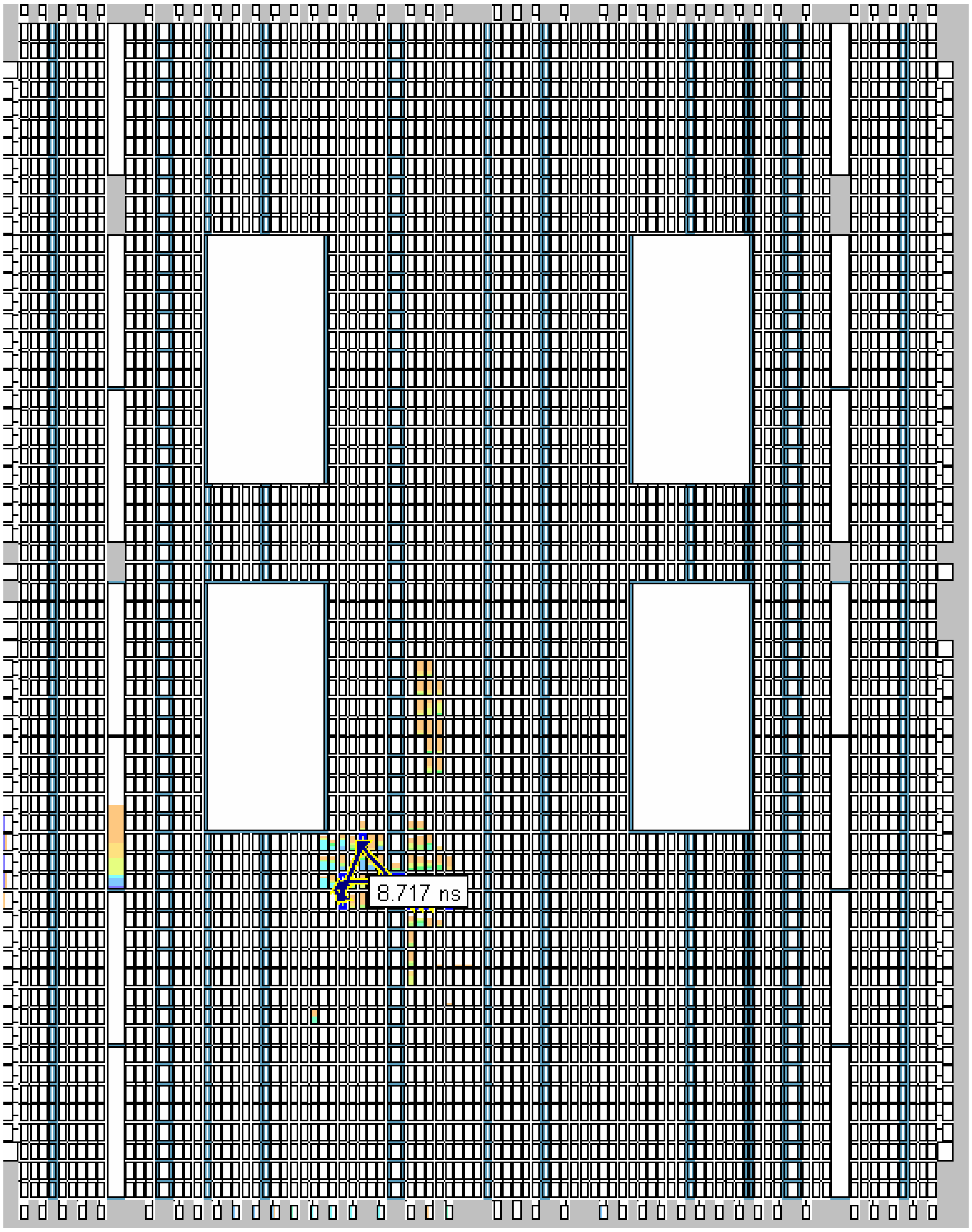

Figure 46: Post Place and Route Floor Plan Views Showing Critical Path for 2 PE Using 3rd Controller

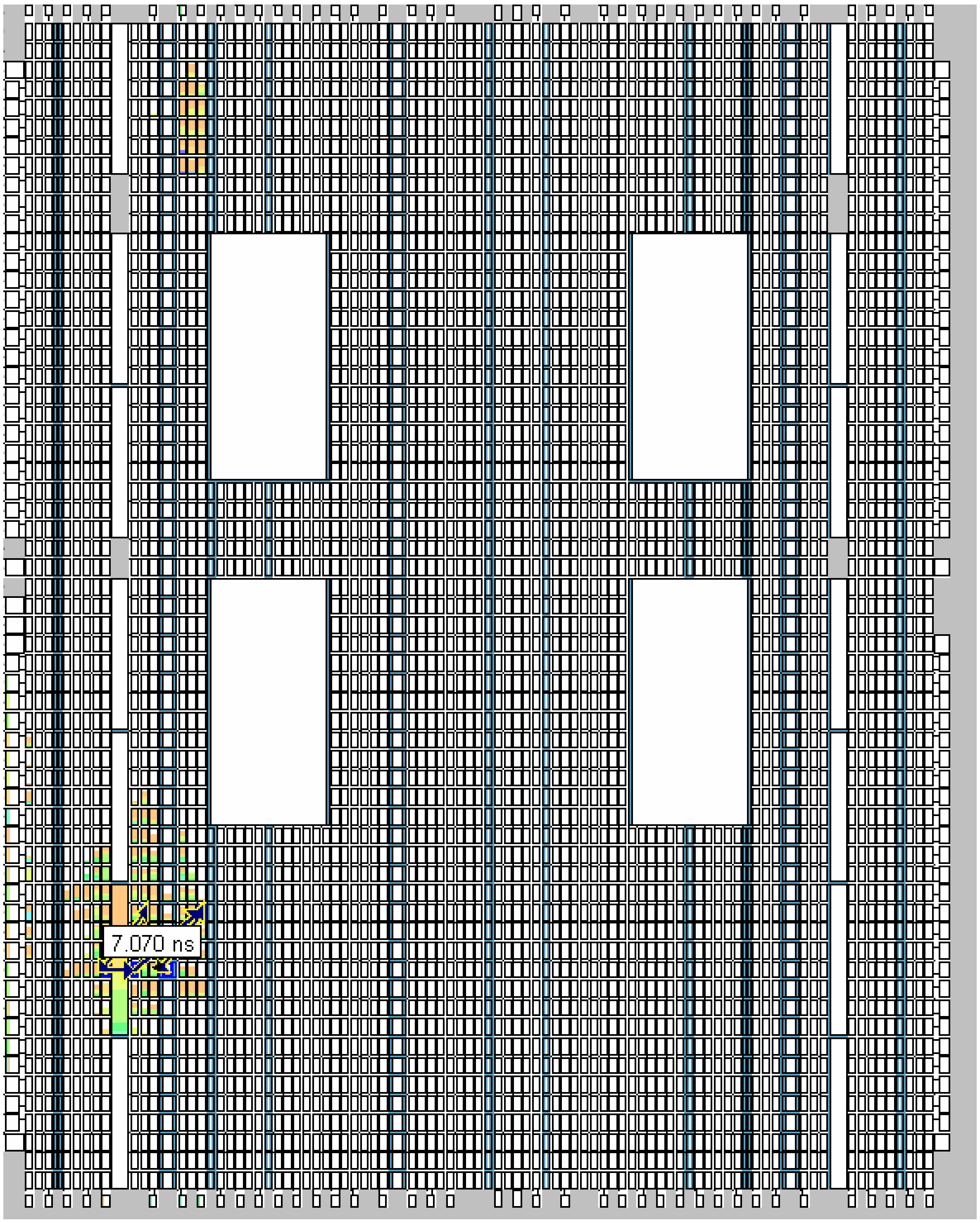

Figure 47: Post Place and Route Floor Plan Views Showing Critical Path for 4 PE Using 3rd Controller

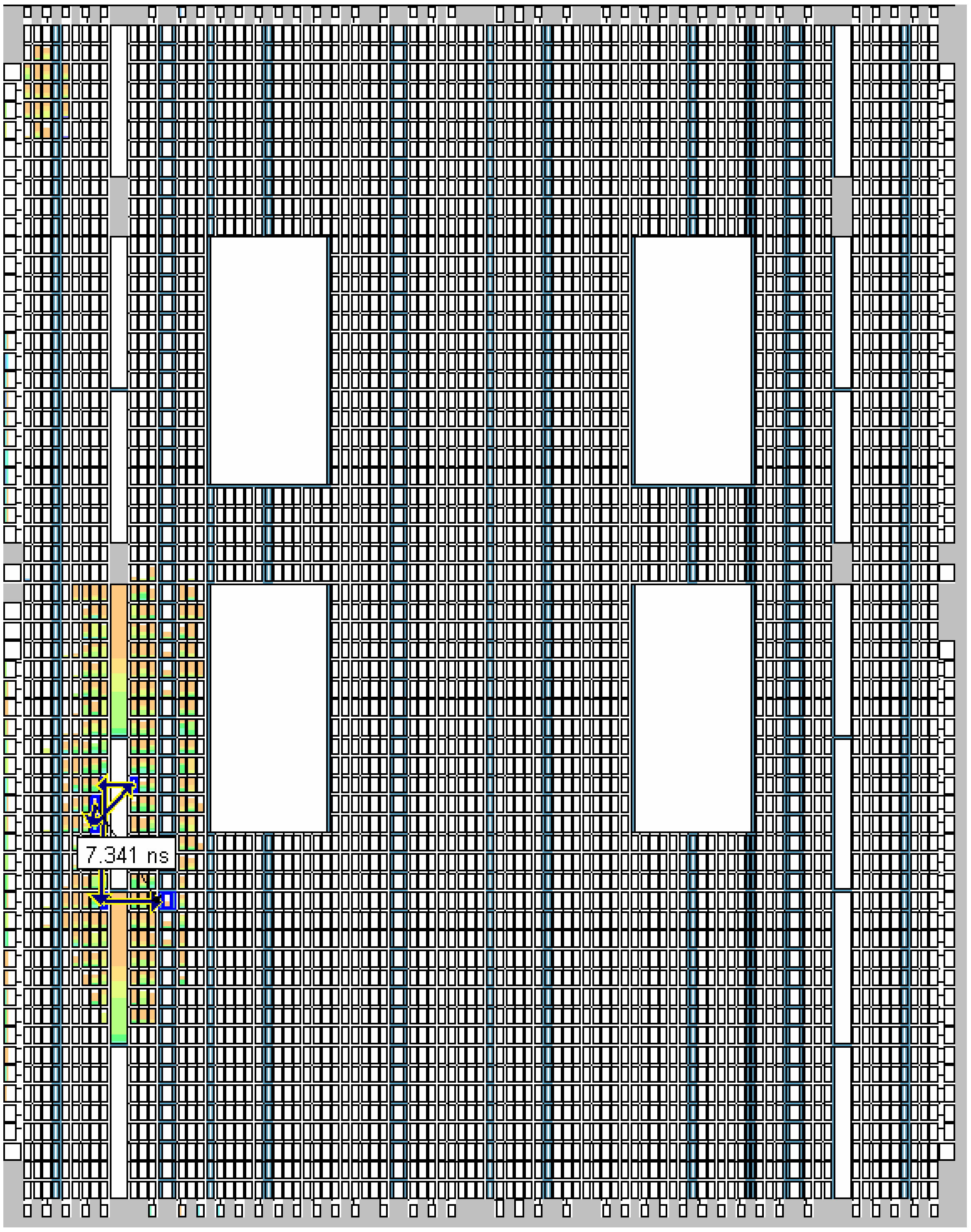

Figure 48: Post Place and Route Floor Plan Views Showing Critical Path for 8 PE Using 3rd Controller

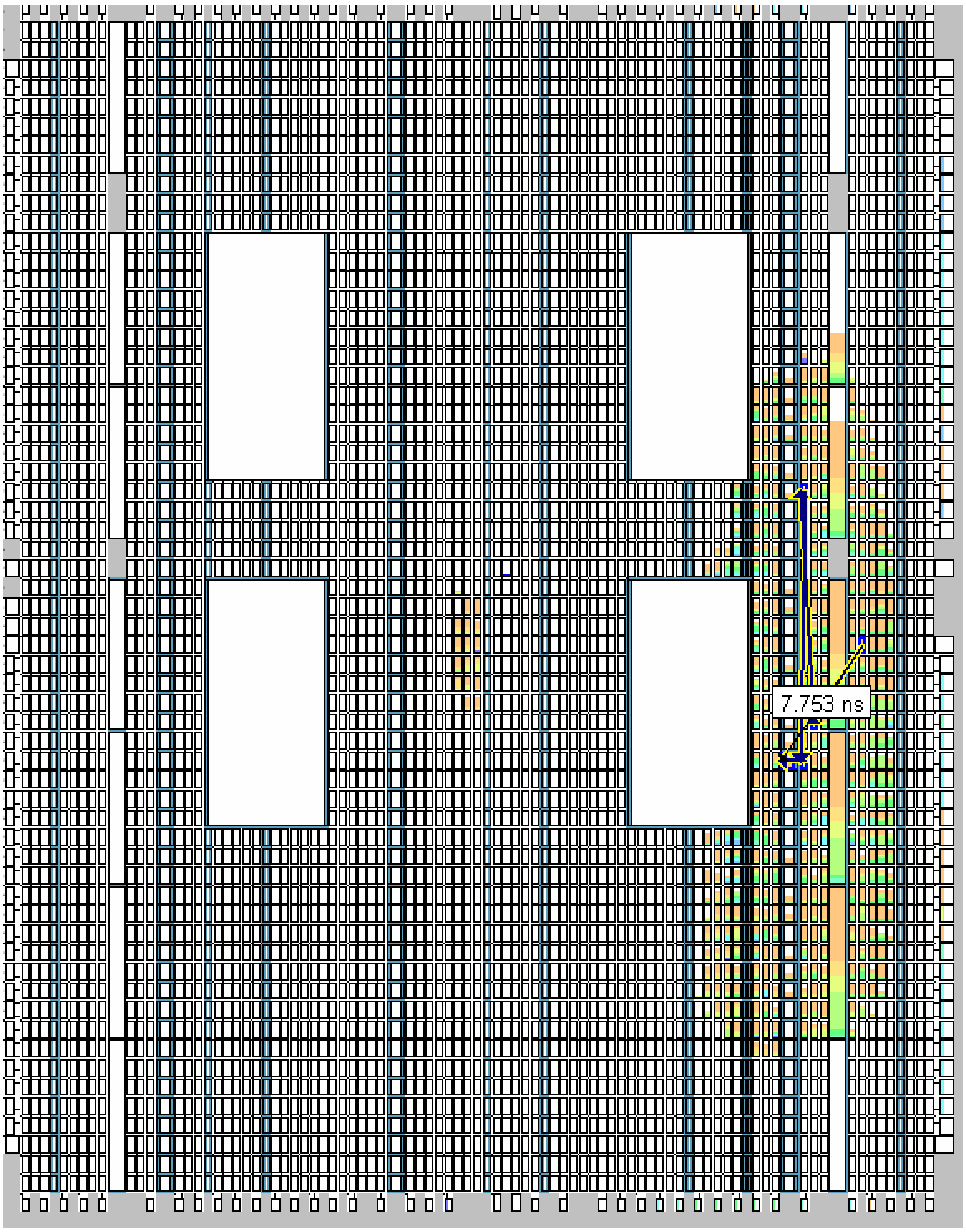

Figure 49: Post Place and Route Floor Plan Views Showing Critical Path for 16 PE Using 3rd Controller

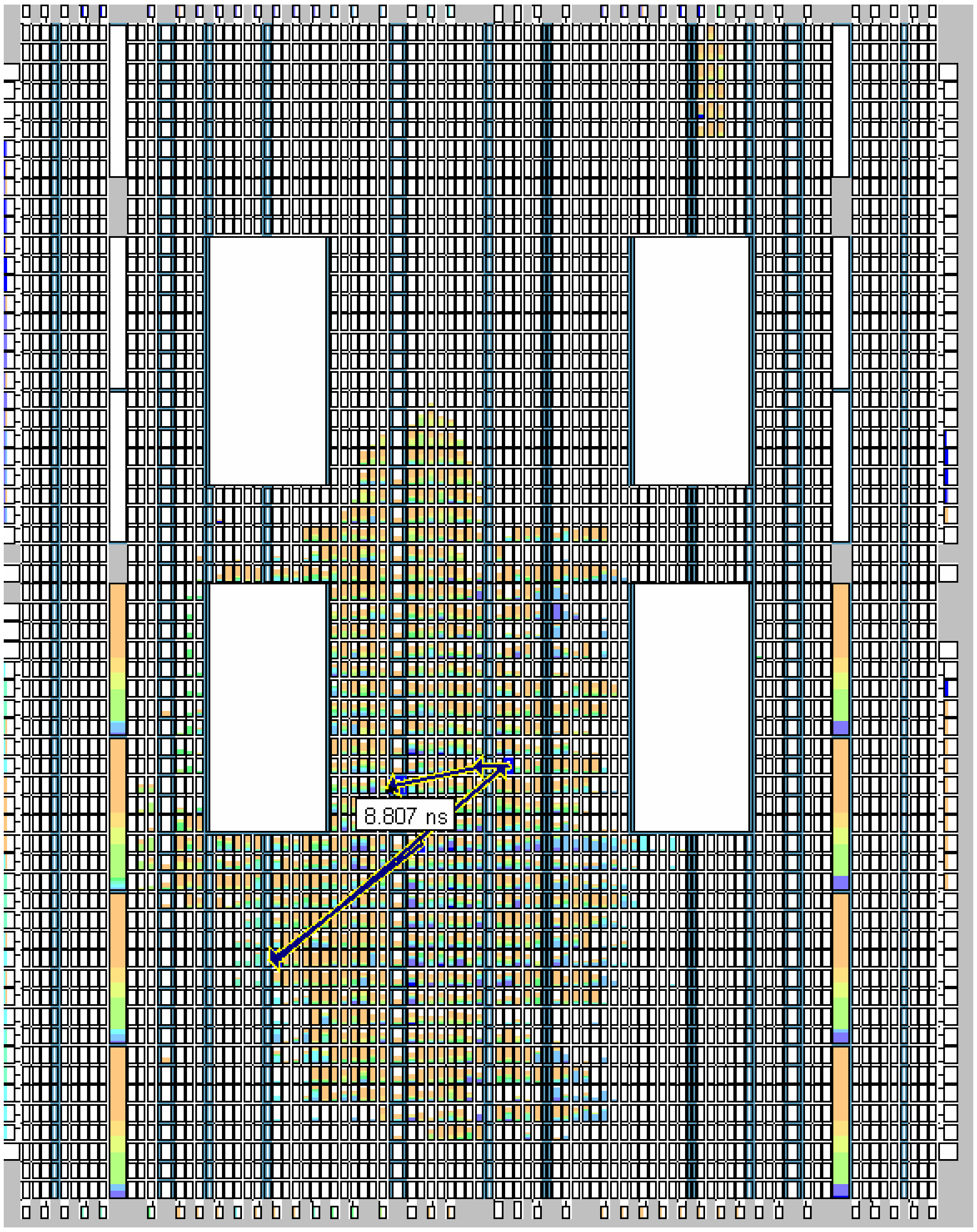

Figure 50: Post Place and Route Floor Plan Views Showing Critical Path for 32 PE Using 3rd Controller

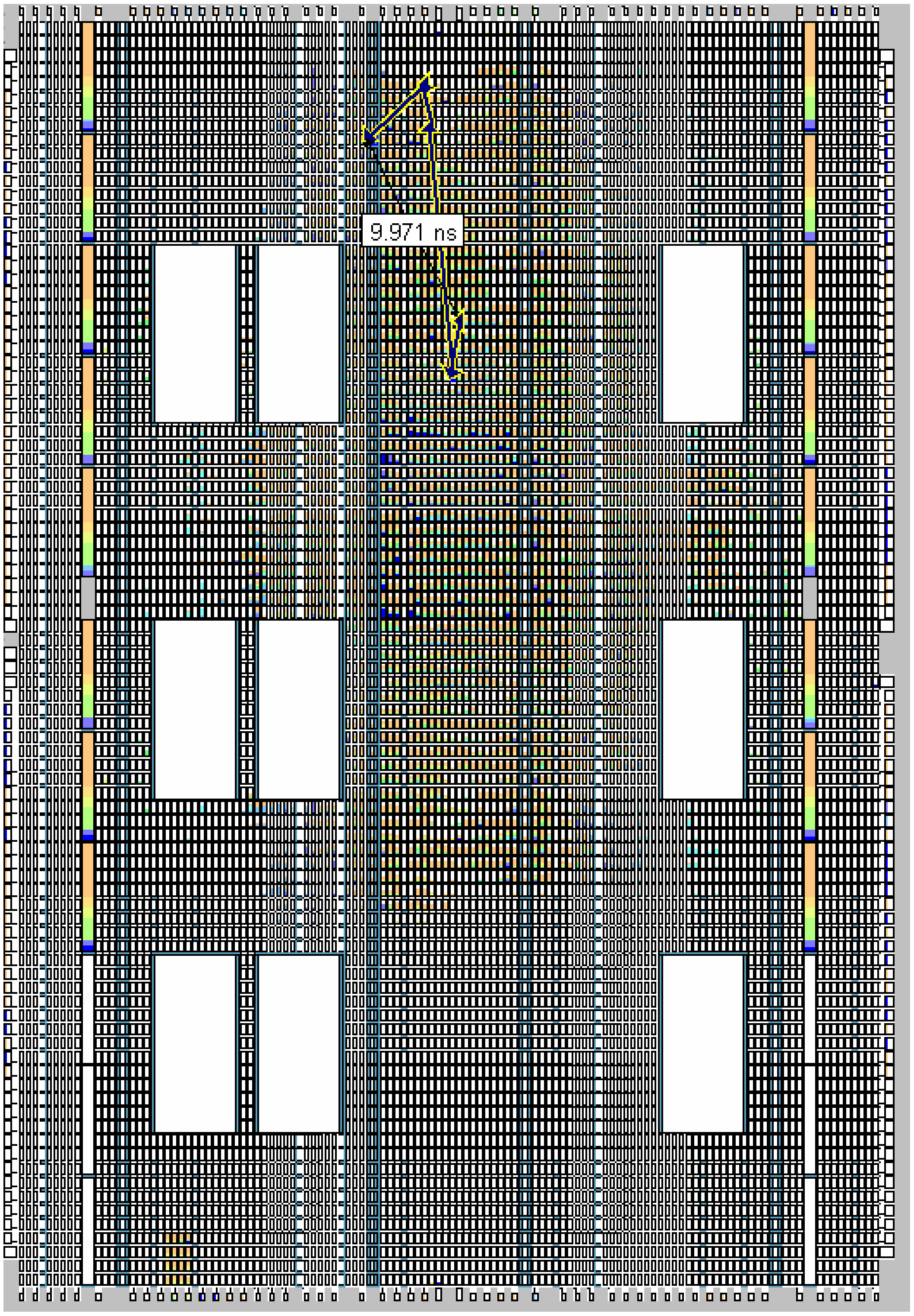

Figure 51: Post Place and Route Floor Plan Views Showing Critical Path for 64 PE Using 3rd Controller

BIBLIOGRAPHY

1. J.L. Hennesy, and D.A. Patterson, *Computer Architecture: A Quanitative Approach*. Morgan-Kaufmann, 1996. Chapter 7, pp. 634-765.

2. D. Talla, L.K. John, V. Lapinskii, and B.L. Evans, "Evaluating Signal Processing and Multimedia Applications on SIMD, VLIW, and Superscalar Architecture," *Proceedings of the 2000 International Conference on Computer Design*, Austin, USA, 2000, pp. 163-172.

3. D. Andrews, C. Kancler, and B. Wealand, "An Embedded Real-Time SIMD Processor Array for Image Processing," *Proceedings of the 4th International Workshop on Parallel and Distributed Real-Time Systems*, Honolulu, USA, 1996, pp. 131-134.

4. M. Sunwoo, S. Ong, B. Ahn, and K.Lee, "Design and Implementation of a Parallel Image Processor Chip for a SIMD Array Processor," *Proceedings of the International Conference on Application Specific Array Processors*, Strasbourg, France, 1995, pp. 66-75

5. T. Inoue, M. Sano, and Y. Takahashi, "Design of a Processing Element of a SIMD Computer for Genetic Algorithms," *High Performance Computing on the Information Superhighway*, Seoul, South Korea, 1997, pp. 688-691.

6. K. Chen, S. Chang, T. Chiueh, P.B. Luh, and X. Zhao, "SIMD Architecture for Job Shop Scheduling Problem Solving," *The 2001 IEEE International Symposium on Circuits and Systems*, Sydney, Australia, May 2001, pp. 530-533.

7. S. Li, G. Cheuk, K. Lee, and P. Leong, "FPGA-based SIMD Processor," *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2003, pp. 267-268.

8. H.F Ugurdag and C.A Papachristou, "A VLIW Architecture Based on Shifting Register Files," *Proceedings of the 26th Annual International Symposium on Microarchitechture*, Austin, USA, 1993, pp. 263-268.

9. Xilinx Virtex II Pro Platform FPGAS: Functional Description. ver 2.9, Oct 2003 http://direct.xilinx.com/bvdocs/publications/ds083-2.pdf

10. Xilinx Virtex II Platform FGPAs: Detailed Description. ver 3.1, Oct 2003 http://direct.xilinx.com/bvdocs/publications/ds031-2.pdf

11. Altera Inc. Stratix Device Family Data Sheet: Stratix Device Handbook. vol. 1, ver 1.2, May 2003 http://www.altera.com/literature/hb/stx/stratix_vol_1.pdf

12. D. Reed and R. Hoare, "An SoC Solution for Massive Parallel Processing," *Proceedings of International Parallel and Distributed Processing Symposium, IPDPS 2002*, Fort Lauderdale, USA, 2002, pp. 15-19.

13. Altera Inc. Using the LogicLock Methodology in the Quartus II Design Software ver 3.3, June 2003 http://www.altera.com/literature/an/an161.pdf

14. S.P. Kim, R.R. Hoare, and H.G. Dietz, "VLIW Across Multiple Superscalar Processors on A Single Chip," *International Conference on Parallel Architectures and Compilation Techniques*, 1997, pp. 166-175.

15. S. Periyacheri, A. Jones, A. Nayak, D. Zaretsky, P. Banerjee, N. Shenoy, and A. Choudhary, "Functions in Reconfigurable Hardware for Matrix and Signal Processing Operations in Matlab," *Proceedings of the IASTED International Conference (Parallel and Distributed Computing Systems*), Cambridge, USA, 1999 pp. 663-669.

16. P. Kabal and B. Sayar, "Performance of Fixed-Point FFT's: Rounding and Scaling Considerations," *IEEE International Conference on Acoustics, Speech, and Signal Processing,* April 1986, pp. 221-224.

17. V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin/Cummings, 1994, Chapter 3, pp. 65-116.

18. R. Hoare, D. Swope, S. Bailey, "A Width Expansion of MMX/SIMD Processing Architecture on an FPGA," *Proceedings of the IASTED International Conference on Parallel and Distributed Computing Systems*, Cambridge, USA, Nov. 2002, pp. 566-571.

19. Altera Inc. Stratix Device Handbook Ch 6: Using the DSP Blocks in Stratix & Stratix GX Device. vol 2, July 2003, http://www.altera.com/literature/hb/stx/ch6_vol_2.pdf

20. S. K. Mitra, Digital Signal Processing: A Computer-Based Approach, McGraw-Hill, 2001, Chapter 8, pp. 515-582.

21. G. S. Almasi and A. Gottlieb, Highly Parallel Computing, Benjamin/Cummings, 1994, Chapter 9, pp. 413-467.

22. Altera Inc. FFT MegaCore Function http://www.altera.com/products/ip/dsp/transforms/m-ham-fft.html.

23. Motorola Application Note, Complex Fixed-Point Fast Fourier Transform Optimization for AltiVec. rev 2.1, June 2003, http://e-www.motorola.com/files/32bit/doc/app_note/AN2114.pdf.

24. Zarlink Semconductor, PDSP16515A Stand Alone FFT Processor Advance Information. Issue 2.0, April 1999, http://assets.zarlink.com/DS/zarlink_PDSP16515A_APR_99.pdf.