Floating Point Coprocessor
ECE 337
Final Report
Wednesday 11:30 Lab Section 3

Andy Shi
Matt King
Kyunghoon Jung
Tasha Weidler
TA: Yunus Akhtar

May 6, 2015

1. Executive Summary

In many aspects of computing, performance is defined as the throughput of the system. *How quickly can the system process data?* This is an important question to ask when there is a precedence on the time it takes to reach a solution as well as the validity of the solution. Hardware applications large and small may contain many processing units, usually consisting of a primary central processing unit (CPU) and many coprocessors that augment the functionality of the main processor. Offloading specific tasks to dedicated coprocessors for tasks like I/O integration, arithmetic functions, and graphics processing can significantly increase the throughput of the overall system.

The objective of this design project is to create a floating point arithmetic coprocessor (FPU) capable of addition and subtraction, as well as more advanced functions such as multiplication and calculation of sine and cosine values. The purpose of the FPU in context of a larger system is to handle floating point arithmetic, reserving CPU resources for higher-level program management. The design of the FPU will focused on a specific function, making an ASIC implementation ideal. The remainder of this proposal will detail the technical aspects of an ASIC implementation of the FPU.

2. Design Specifications

2.1 Interfacing Specifications

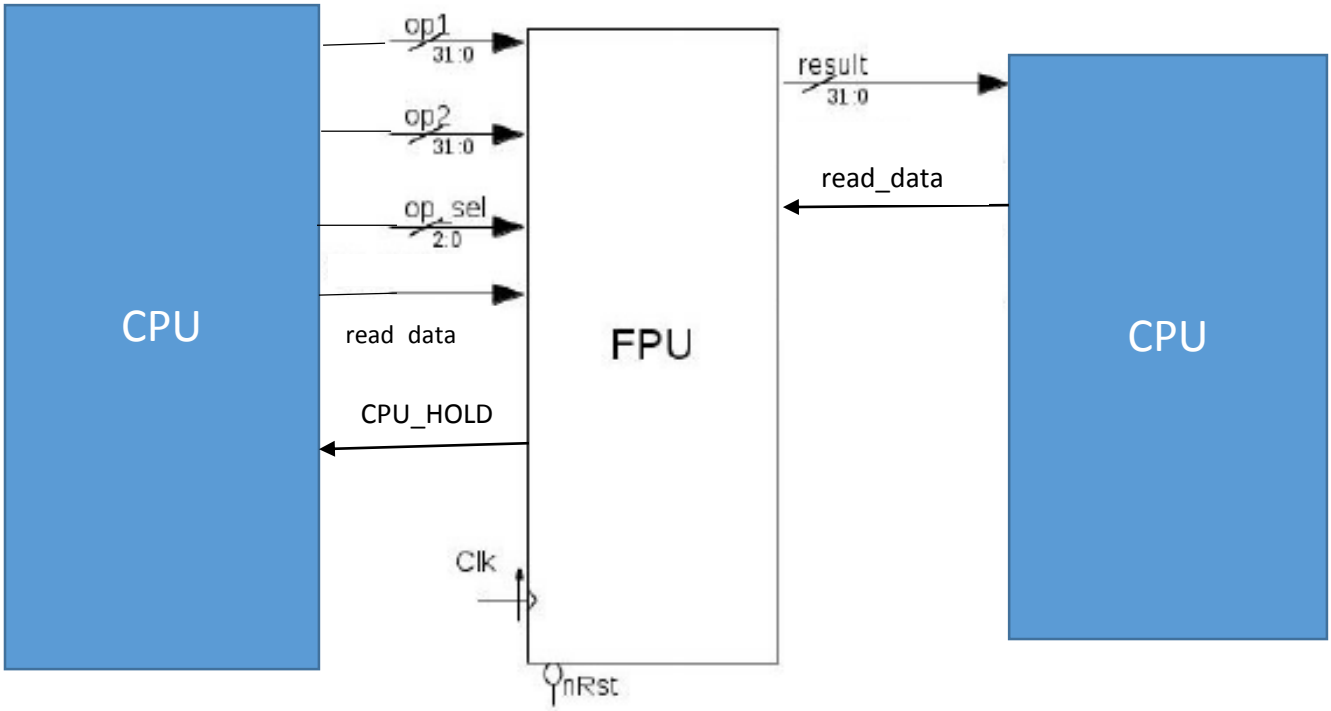


Figure 1: Chip Layout Overview

This design adopts the IEEE-754 Single Precision Floating Point Standard. The way this works, as shown in Figure 2, is that the first bit is the sign bit, it makes the whole variable into a positive or negative value. The next 8 bits, bits two through nine, are the exponent bits which hold the information of the binary power of the variable. Then the last 23 bits are called the mantissa, they are the fractional value of the variable. The mantissa is multiplied by the exponent bits to get the decimal value. Since this standard describes many types of floating point data and operations, some generalizations and assumptions will be made for the simplicity of this design, such as using a single rounding scheme as well as exception handling of NaN and infinity conditions.

The features of this design include performing five different operations on 32-bit floating point operands. These operations are addition, subtraction, multiplication, sine, and cosine.

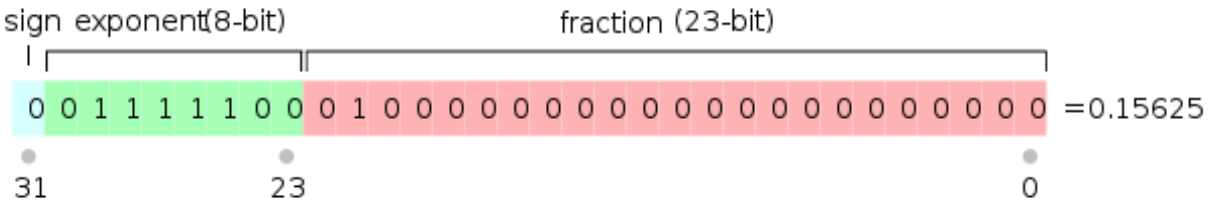


Figure 3: IEEE-754 Single Precision Floating Point Standard Illustration

2.2 Operational Characteristics

The Operation of the design will be achieved by use of an input decoding module to determine the operation requested, and then blocks for individual functions. The FPU will be designed to facilitate operations in parallel. If an addition operation is being performed and the module receives a multiply operation, it will start the multiply operation before the addition operation is completed. This ensures maximum usage of the available resources to increase the overall throughput of the module.

The flow of data through each block will be similar for each operation requested. There will be a sub-block to “pre-process" the data to allow the floating point numbers to be operated on, the operation sub-block itself, and then the “post-process" sub-block to normalize data. The input and output decode blocks will handle sending data to and from each operation block.

Signal Name	Type	Num. Bits	Description
clk	Input	1	250 MHz System Clock
n_rst	Input	1	System wide active low reset
op1	Input	32	Input operand 1. This operand must be a valid number conforming to the IEEE-754 Standard. Input will be in normalized form.
op2	Input	32	Input operand 2. This operand must be a valid number conforming to the IEEE-754 Standard. Input will be in normalized form.
op_sel	Input	3	Operation select. 000: add/sub, 010: mul, 011: sine, 100: cosine.
cpu_hold	Output	1	A signal sent to the CPU when the input is full with 8 operations to be performed.
op_strobe	Input	1	A signal telling the input FIFO buffer when to write data.
cpu_pop	Input	1	A signal telling the output FIFO buffer when to read data.
result	Output	32	Output result. The output will also be a number conforming to the IEEE-754 standard.

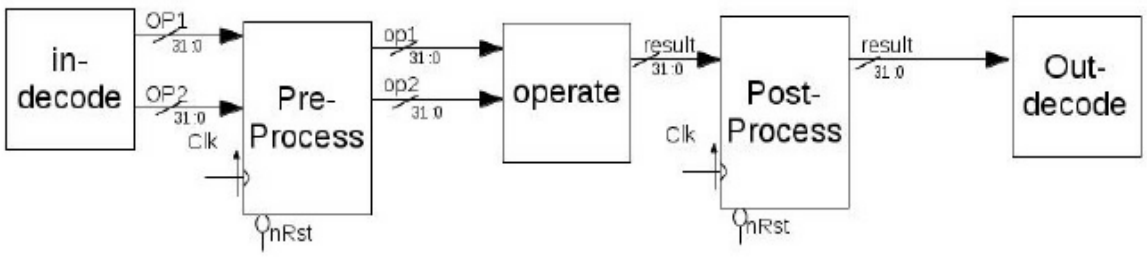


Figure 2: Data Flow through Block

The functionality of the module will be split into several functional blocks. Descriptions and signals associated with each block are described below:

Input Decode

- **Module Name:** indecode
- **File Name:** indecode.sv
- This block takes all the system level inputs and determines which operation is requested and what to do with the data inputs.
- In order to facilitate the parallel operation of the arithmetic operations, the decode block will add the requested opcode and operands to a FIFO buffer. This allows the output block to determine the order in which the operations were requested.
- The arithmetic units will use a shared operand bus, and each operation will have a dedicated start signal to tell that block the values on the bus are for that operation and to begin computing results.
- This block is partly combinational, acting as a multiplexer for the different operations and ties the system level inputs to the specific blocks in the module. This block also contains a FIFO buffer that holds eight instances of opcodes and operands.

Signal Name	Type	Num. Bits	Data Format or Active High/Low	Description
clk	Input	1	Active High	250 MHz System Clock
n_rst	Input	1	Active Low	System wide active low reset
op1	Input	32	Single-Precision Floating Point (32 bit)	Input operand 1. This operand must be a valid number conforming to the IEEE-754 Standard. Input will be in normalized form.
op2	Input	32	Single-Precision Floating Point (32 bit)	Input operand 2. This operand must be a valid number conforming to the IEEE-754 Standard. Input will be in normalized form.

op_sel	Input	3	Unsigned binary number	Operation select. 000: add, 001: mul, 011: sine, 100: cosine.
add_busy	Input	1	Active High	Signal asserted when an add or subtract operation is currently taking place
mul_busy	Input	1	Active High	Signal asserted when a multiply operation is currently taking place
sine_busy	Input	1	Active High	Signal asserted when a sine or cosine operation is currently taking place
out_fifo_hold	Input	1	Active High	Signal from the output decode block that is asserted when the output FIFO buffer is full.
op_strobe	Input	1	Active High	Signal to tell the input and opcode FIFO buffers to read in data.
op1_out	Output	32	Single-Precision Floating Point (32 bit)	Output to the shared operand 1 bus.
op2_out	Output	32	Single-Precision Floating Point (32 bit)	Output to the shared operand 2 bus.
add_start	Output	1	Active High	Signal to tell the add block to begin computation.
mul_start	Output	1	Active High	Signal to tell the multiply block to begin computation.
sine_start	Output	1	Active High	Signal to tell the sine block to begin computation.
opcode_out	Output	3	Unsigned binary number	Output of opcode to the opcode FIFO buffer.
opbuff_strobe	Output	1	Active High	Signal to tell the opcode FIFO buffer to read data
cpu_hold	Output	1	Active High	A signal sent to the CPU when the input is full with 8 operations to be performed.

Add & Subtract

- **Module Name:** addsub
- **File Name:** addsub.sv
- This block takes the two operands as inputs and adds them together. The mode input switches to subtraction by using radix complement.
- The operands will be input in a normalized format, however the block will handle processing the operands such that the exponents will match, allowing the operation to take place. The block will then normalize the result prior to outputting the value.

Signal Name	Type	Num. Bits	Data Format or Active High/Low	Description
clk	Input	1	Active High	250 MHz System Clock
n_rst	Input	1	Active Low	System wide active low reset.
add_start	Input	1	Active High	Signal to tell block to begin computation.
add_serv	Input	1	Active High	When this signal is asserted, that means the operation is complete and the add_busy signal is reset.
op1	Input	32	Single-Precision Floating Point (32 bit)	Input operand 1. This operand must be a valid number conforming to the IEEE-754 Standard.
op2	Input	32	Single-Precision Floating Point (32 bit)	Input operand 2. This operand must be a valid number conforming to the IEEE-754 Standard.
add_busy	Output	1	Active High	Signal asserted when an add or subtract operation is currently taking place
add_result	Output	32	Single-Precision Floating Point (32 bit)	Output result. The output will also be a number conforming to the IEEE-754 Standard.
add_done	Output	1	Active High	Signal is asserted when the add or subtract operation is complete.

Multiply

- **Module Name:** multiply
- **File Name:** multiply.sv
- This block takes the two operands as inputs and multiplies them together.
- The block will handle adjusting the exponents to allow the operation to take place and then normalize the data prior to output.

Signal Name	Type	Num. Bits	Data Format or Active High/Low	Description
clk	Input	1	Active High	250 MHz System Clock
n_rst	Input	1	Active Low	System wide active low reset.
mul_start	Input	1	Active High	Signal to tell block to begin computation.
mul_serv	Input	1	Active High	When this signal is asserted, that means the operation is complete and the mul_busy signal is reset.
op1	Input	32	Single-Precision Floating Point (32 bit)	Input operand 1. This operand must be a valid number conforming to the IEEE-754 Standard.
op2	Input	32	Single-Precision Floating Point (32 bit)	Input operand 2. This operand must be a valid number conforming to the IEEE-754 Standard.
mul_busy	Output	1	Active High	Signal asserted when a multiply operation is currently taking place
mul_result	Output	32	Single-Precision Floating Point (32 bit)	Output result. The output will also be a number conforming to the IEEE-754 Standard.
mul_done	Output	1	Active High	Signal is asserted when the multiply operation is complete.

Sine & Cosine

- **Module Name:** sincos
- **File Name:** sincos.sv
- This block takes one radian input and outputs the sine or cosine of that value.
- This block will use a Taylor series representation of the sine or cosine value. The series is an addition of the input divided by increasing factorials. This block will make use of the inverse of these factorials stored in memory, and the multiply it by the input.
- This block will have dedicated add and multiply components to free the add and multiply blocks to allow parallel operations to take place.

Signal Name	Type	Num. Bits	Data Format or Active High/Low	Description
clk	Input	1	Active High	250 MHz System Clock.
n_rst	Input	1	Active Low	System wide active low reset.
opx	Input	32	Single-Precision Floating Point (32 bit)	Input operand. This operand must be a valid number conforming to the IEEE-754 Standard.
sine_result	Output	32	Single-Precision Floating Point (32 bit)	Sine output result. The output will also be a number conforming to the IEEE-754 Standard.
cosine_result	Output	32	Single-Precision Floating Point (32 bit)	Cosine output result. The output will also be a number conforming to the IEEE-754 Standard.
sine_done	Output	1	Active High	Signal is asserted when the sine/cosine operation is completed.

Opcode FIFO Buffer

- **Module Name:** fifobuff
- **File Name:** fifobuff.sv
- This block is a first in first out buffer responsible for storing the order in which the operations were requested. This allows the design to perform different operations in parallel and output the results in the order in which their requests were made.

Signal Name	Type	Num. Bits	Data Format or Active High/Low	Description
clk	Input	1	Active High	250 MHz System Clock.
n_rst	Input	1	Active Low	System wide active low reset.
read	Input	1	Active High	Signal telling the buffer to read in the opcode.
write	Input	1	Active High	Signal telling the buffer to write data.
opcode_in	Input	3	Unsigned binary number	Opcode data to be read into the buffer.
opcode_out	Output	3	Unsigned binary number	Next opcode to be output.

Output Decode

- **Module Name:** outdecode
- **File Name:** outdecode.sv
- This block handles the system level outputs.
- This block contains a FIFO buffer to determine which result value to output.

Signal Name	Type	Num. Bits	Data Format or Active High/ Low	Description
clk	Input	1	Active High	250 MHz System Clock.
n_rst	Input	1	Active Low	System wide active low reset.
add_result	Input	32	Single-Precision Floating Point (32 bit)	Result value from the add/subtract block.
mul_result	Input	32	Single-Precision Floating Point (32 bit)	Result from the multiply block.
sine_result	Input	32	Single-Precision Floating Point (32 bit)	Result from the sine/cosine block.
add_done	Input	1	Active High	Signal is asserted when the add/subtract operation is completed.
mul_done	Input	1	Active High	Signal is asserted when the multiply operation is completed.
sine_done	Input	1	Active High	Signal is asserted when the sine/cosine operation is completed.
opcode_out	Input	3	Unsigned binary number	Next opcode to be output.
fifo_out	Input	1	Active High	Signal telling the buffer to read in the opcode.
cpu_pop	Input	1	Active High	Signal telling the output FIFO buffer to read data.
op_fifo_pop	Input	1	Active High	Signal telling the output FIFO buffer to write the opcode data.
add_serv	Output	1	Active High	When this signal is asserted, that means the operation is complete and the add_busy signal is reset.
mul_serv	Output	1	Active High	When this signal is asserted, that means the operation is complete and the mul_busy signal is reset.
sine_serv	Output	1	Active High	When this signal is asserted, that means the operation is complete and the sine_busy signal is reset.
out_fifo_hold	Output	1	Active High	Signal to tell the input decode block that the output FIFO buffer is full.
result	Output	32	Single-Precision Floating Point (32 bit)	Result of the operation performed.

2.3 Requirements

The target application for this design is to be interfaced with a primary processor via two 32-bit input buses, and a 2-bit operation select bus. Output will be interfaced to the primary processor via a 32-bit result bus and a single bit overflow error bus. This design calls for a relatively high pin count, and modifications could be made to optimize chip footprint by utilizing a single bi-directional 32-bit bus to handle transmission of operands and results. However, since the scope of this project is focused on specifically the FPU rather than integration, the goal will be to create a simple general solution that could be easily optimized for specific integration. The goal for the timing of the system will be to optimize the design such that a system clock of 250 MHz can be achieved.

It is assumed that all data passed to the FPU will be in a normalized format, and the CPU will never send the FPU invalid data not in this format. When the busy flag is asserted, the CPU will wait for the FPU to free resources before sending additional data.

3. Final Design

3.1 Design Architecture

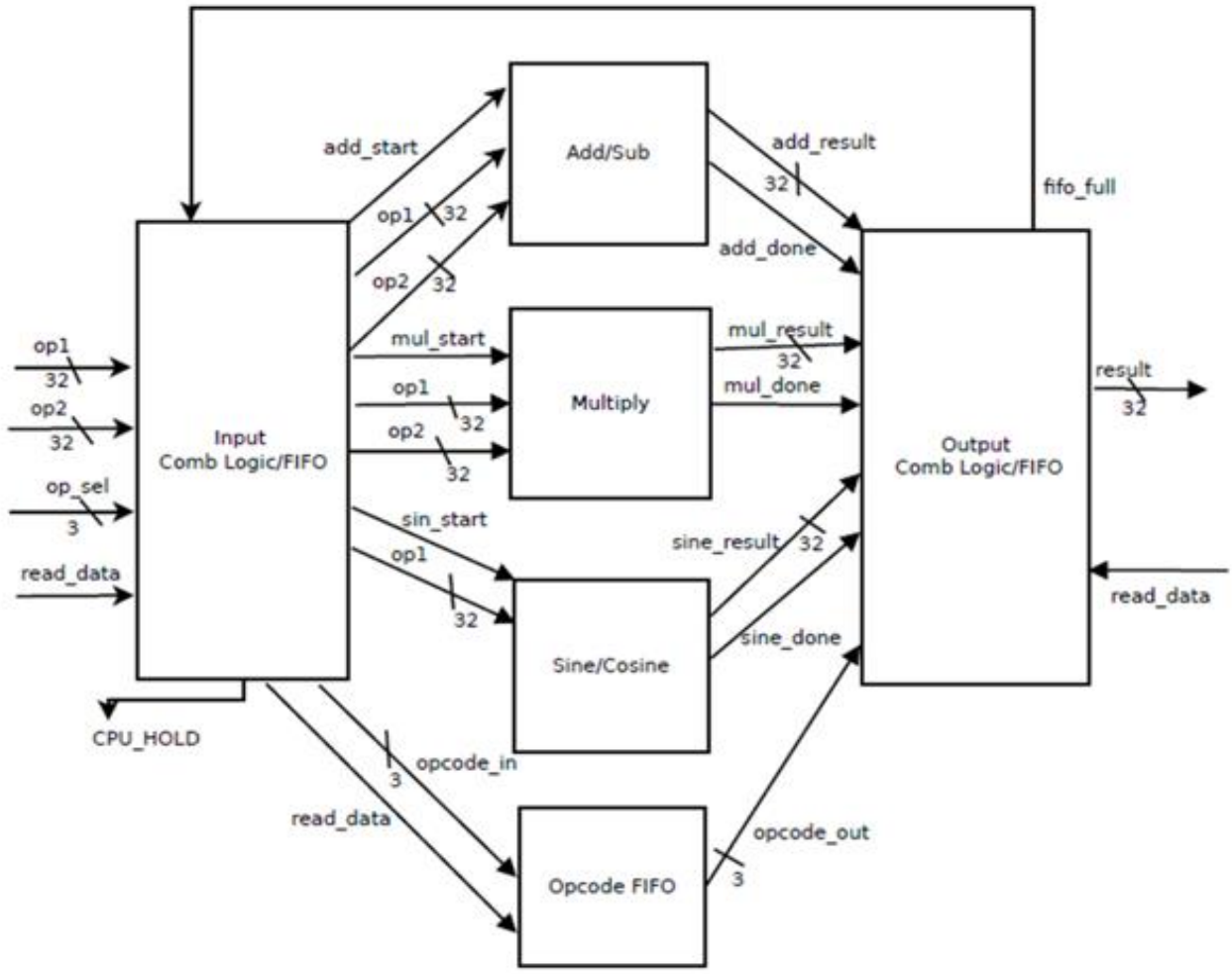
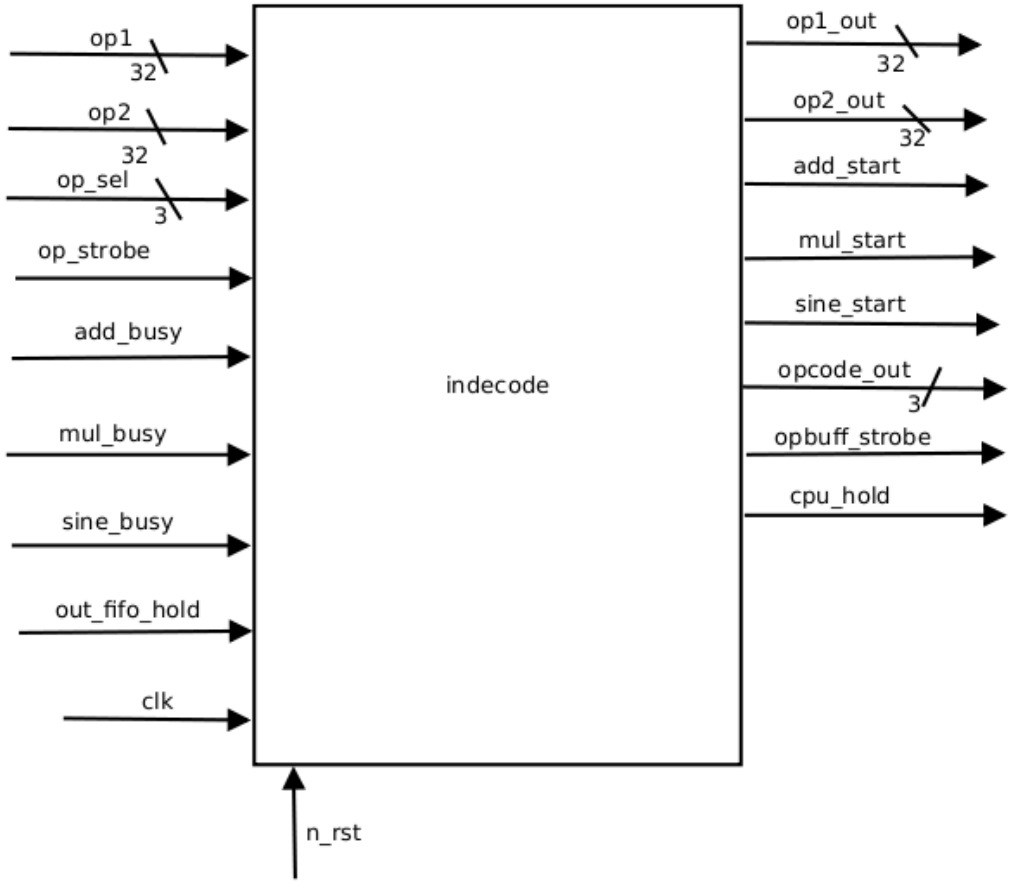
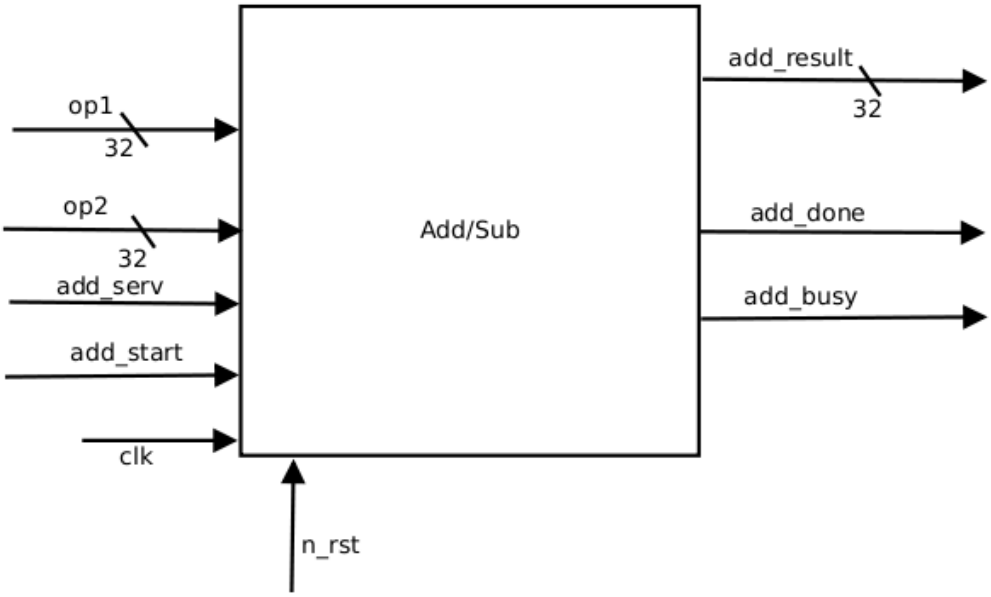


Figure 4: Top-Level Diagram

3.2 Functional Block Diagrams

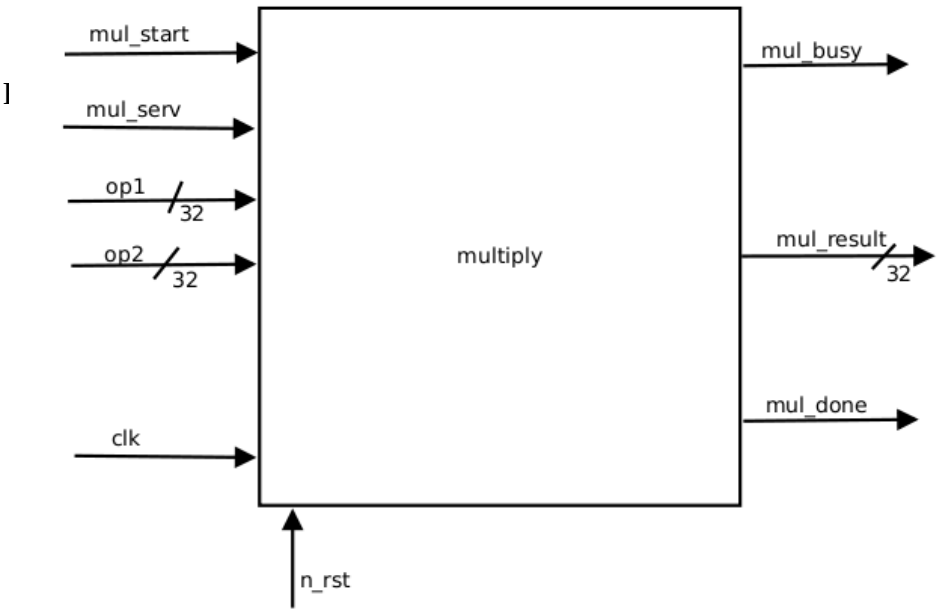


Input Combinational Logic/FIFO: Top-Level block that holds 8 instances of two 32-bit floating point operands and a 3-bit operation select code. When the CPU has sent eight operations to be performed, this block sends a FIFO_FULL signal back to the CPU. Also decides which operation is to be performed next and sends the operands to the correct block. This block also decides which operation blocks are busy and which are ready to perform the next operation, and it sends the operation select codes to the Opcode FIFO block.

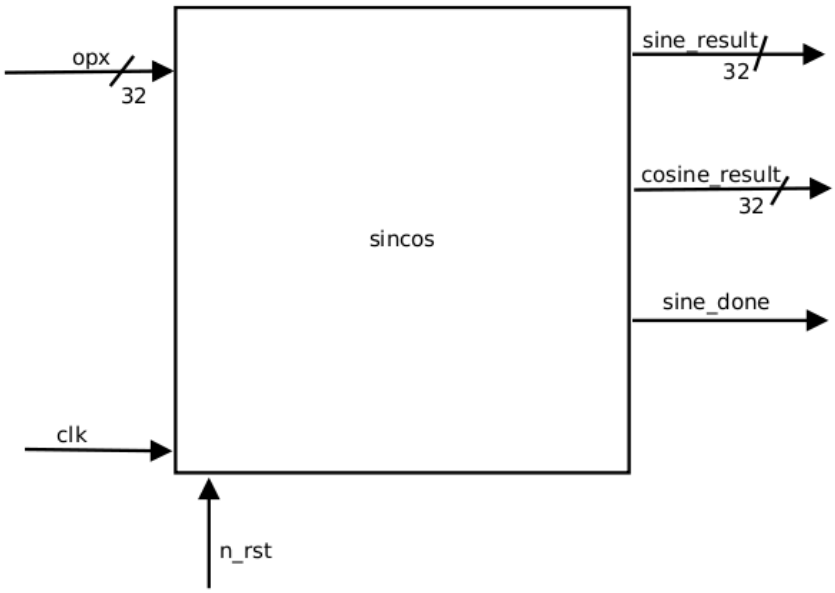


Add/Sub: Top-Level block that takes in two 32-bit floating point operands, performs addition or subtraction, and produces the 32-bit result.

]



Multiply: Top-Level block that takes in two 32-bit floating point operands, multiplies them together, and produces the 32-bit result.



Sine/Cosine: Top-Level block that takes in one 32-bit floating point operand and calculates the sine or cosine using Taylor Series and a duplicate set of the Add/Sub and Multiply blocks.
Opcode FIFO: Top-Level block that takes in 8 instances of operation select codes and decides the order the operations came from the CPU and produces them in that order.

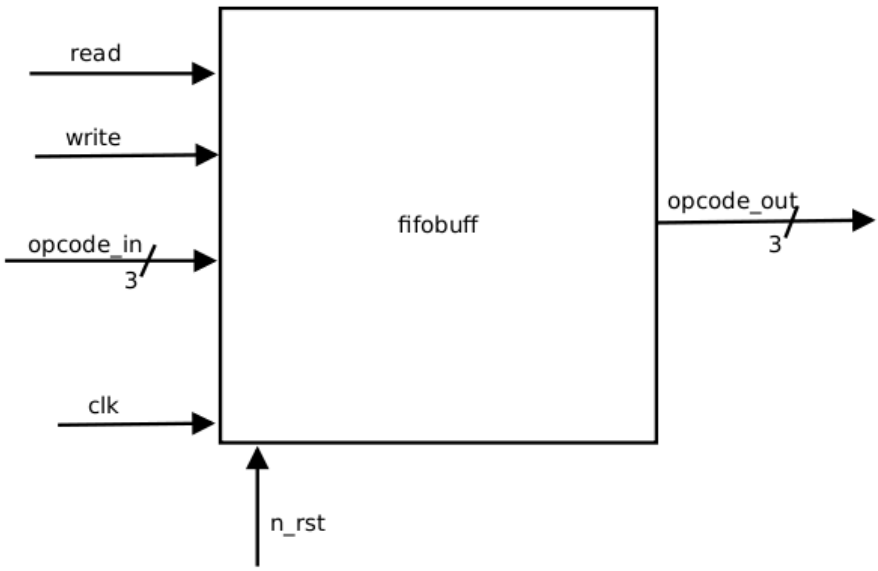


Figure 9: Block Diagram for the Opcode FIFO Buffer Module Block

Opcode FIFO Buffer: Top-Level block that only contains a FIFO buffer. This buffer takes in eight instances of operation select codes and outputs them to the outdecode block in the same order that they were received.

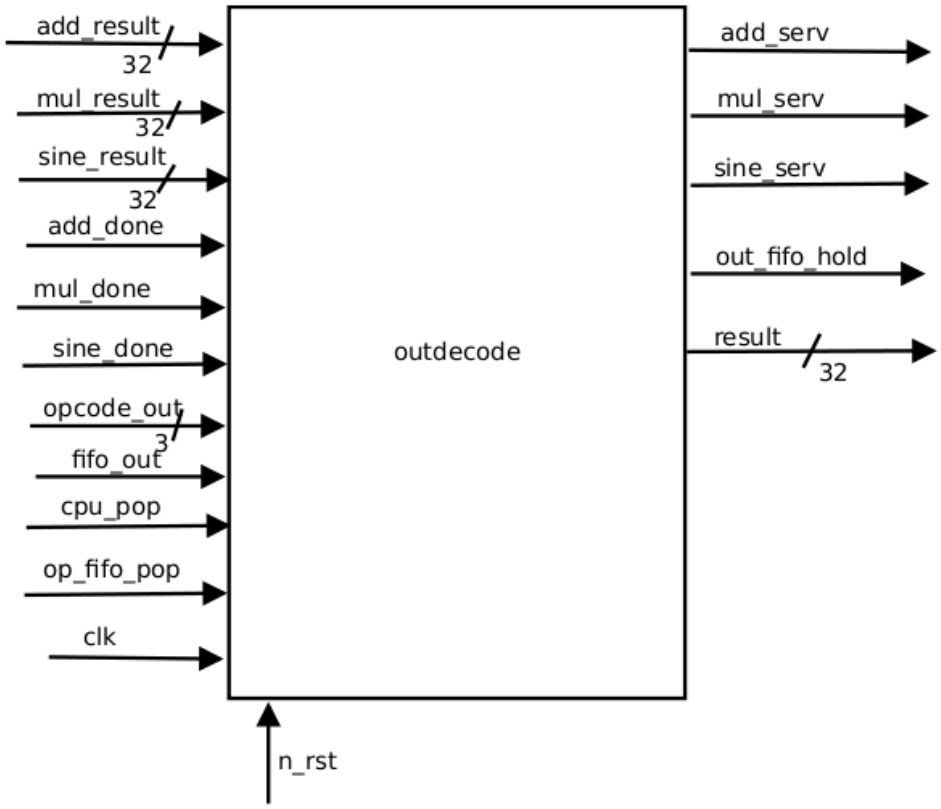


Figure 10: Block Diagram for the Output Decode Module Block

Output Combinational Logic/FIFO: Top-Level block that takes in the operation select codes and the results of the operations and pairs them together and uses a FIFO to take in 8 instances of operation results and produces them to the CPU in the order they were received from the CPU.

3.3 Standards and Protocols

Floating point variable is used everywhere. From CPUs to GPUs, floating point is essential in calculating exponentials, fractions, and especially divisions that involve fractions and decimal points which require certain amount of precision. Therefore, we chose the protocol that is widely used in the world of computer science, which is IEEE 754 Single-Precision floating point standard. By implementing IEEE 754 standard, users can utilize all the applications that uses floating point variables, without it, they'll have to implement this operation based on a software which takes a lot longer in operation time because they have to treat each bit as a separate variable. Floating point is used very widely, thus it is a perfect implementation as a standalone ASIC.

The IEEE 754 Single Precision floating point is a 32 bit binary variable, the most significant bit being the sign bit, controls the sign of the floating point variable. The 2^{nd} to 9^{th} bit of the floating point variable is called the exponent bit, which determines the power the floating point variable is raised to in terms of binary number. The 10^{th} bit to the last bit consists of a fractional value, which is essentially an addition of binary fractions. The analogy is easy when this is compared to a decimal point scientific notation. For example, consider a number 5380; this is written in scientific notation as this: 5.38×10^3. 5.38 represents the fraction bit, represented in decimal value, and the power 3 on 10 represents the exponent value. This analogy works exactly the same in floating point variables.

We implemented 5 different functions in our floating point processor, which are addition, subtraction, multiplication, sine, and cosine. Multiplication was easiest among them to implement. In essence, multiplication is done by adding the exponent bit against each other, and manipulating exponent bit to show the correct result. Subtraction is just flipping the sign of the first bit and using addition, it is just one additional step before addition. Addition, you have to go through the normalization process. If the floating point variables have same exponent bit, then we don't have to worry about normalization. This is not the case for the most time. They often have different exponent values, and for proper operation and precision, we have to normalize the operand that has lesser exponent value of the two and match it with the operand that has the bigger exponent value of the two. This process was tricky because it involved some bit-shifting in the fraction bit.

The toughest standard we followed was the sine and cosine function. We utilized the Taylor Series implementation of sine and cosine and approximated it to the precision of 5 decimal digits. We decided to use the first 5 terms of the Taylor Series for the sine and cosine function.

Overall, the input and output FIFO handles the instructions based on whether the previous calculations are completed, and queues the CPU request of these calculations based on their chronological order.

3.4 Timing Area Budgets

Core Area Calculations				
Name of Block	Category	Gate/FF Count	Area (um^2)	Comments
Input FIFO Registers	Reg. w/ Reset	600	1,440,000	Holds 8 instances of 2 operands and opcode
Input FIFO Logic	Combinational	32	24,000	Decides what values go into the FIFO registers
Input Logic	Combinational	12	9,000	Decides which operation to perform on which values
Add/Sub Register	Reg. w/ Reset	1	2,400	Resets the values in the Add/Sub module
Add/Sub Logic	Combinational	20	15,000	Adds or Subtracts two 32-bit floating point numbers
Multiply Register	Reg. w/ Reset	1	2,400	Resets the values in the Multiply module
Multiply Logic	Combinational	20	15,000	Multiplies two 32-bit floating point numbers
Sine/Cosine Register	Reg. w/ Reset	1	2,400	Clocks the Sine/Cosine Module
Sine/Cosine Logic	Combinational	400	300,000	Finds the sine or cosine of a 32-bit floating point number
Opcode FIFO Registers	Reg. w/ Reset	500	1,200,000	Holds 8 instances of opcodes
Opcode FIFO Logic	Combinational	16	12,000	Decides what values go into the FIFO registers
Output Logic	Combinational	2	1,500	Decides which results to display
Output FIFO Registers	Reg. w/ Reset	600	1,440,000	Holds 8 instances of results
Output FIFO Logic	Combinational	16	12,000	Decides what values go into the FIFO registers
Total Core Area			4,475,700	
Chip Area Calculations (units in um or um^2)				
Number of I/O Pads:			111	
I/O Pad Dimensions:	100	by	150	
I/O Based Padframe Dimension	3,075	by	3,075	
Core Dimensions	2,116	by	2,116	
Core Based Padframe Dimensio	2,566	by	2,566	
Final Padframe Dimensions:	3,075	by	3,075	
Final Chip Area:			9,455,625	

Figure 11: Area Budget Spreadsheet

Combinational Logic	Propagation Delay (ns)	Ending Component	Setup Time or Propagation Delay (ns)	Total Path Delay (ns)	Target Clock Period (ns)
Input Logic(1 clk cycle), Add/Sub Logic(3 clk cycles)	40	Add/Sub Register	0.1	40.2	4
Input Logic(1 clk cycle), Multiply Logic(1 clk cycle)	20	Multiply Register	0.1	20.2	4
Input Logic(1 clk cycle), Sine/Cosine Logic(5 clk cycles)	450	Sine/Cosine Register	0.1	450.2	4
Output Logic(1 clk cycle), Output FIFO Logic(0.5 clk cycles)	1.5	Output FIFO Register	0.1	1.7	4
Output Logic(1 clk cycle), Output FIFO Logic(0.5 clk cycles)	1.5	Output FIFO Register	0.1	1.7	4

Figure 12: Timing Budget Spreadsheet

Timing Report:

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : fp_layout
Version: G-2012.06
Date   : Wed May  6 19:42:27 2015
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: typical   Library: osu05_stdcells
Wire Load Model Mode: top

  Startpoint: LD/ADD_SUB/op2_r_reg[24]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: LD/OUTPUT/fifo_in_reg[22]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Point                                       Incr       Path
  ---------------------------------------------------------------------------
  clock clk (rise edge)                     0.00       0.00
  clock network delay (ideal)               0.00       0.00
  LD/ADD_SUB/op2_r_reg[24]/CLK (DFFSR)                0.00 #     0.00 r
  LD/ADD_SUB/op2_r_reg[24]/Q (DFFSR)                  0.51       0.51 f
  LD/ADD_SUB/U515/Y (BUFX2)                 0.20       0.71 f
  LD/ADD_SUB/sub_177/B[1] (addsub_DW01_sub_32)        0.00       0.71 f
  LD/ADD_SUB/sub_177/U72/Y (INVX2)                    0.09       0.81 r
  LD/ADD_SUB/sub_177/U51/Y (NOR2X1)                   0.14       0.95 f
  LD/ADD_SUB/sub_177/U48/Y (OAI21X1)                  0.20       1.15 r
  LD/ADD_SUB/sub_177/U33/Y (AOI21X1)                  0.13       1.28 f
  LD/ADD_SUB/sub_177/U68/Y (BUFX2)                    0.25       1.53 f
  LD/ADD_SUB/sub_177/U70/Y (XNOR2X1)                  0.23       1.76 r
  LD/ADD_SUB/sub_177/DIFF[4] (addsub_DW01_sub_32)     0.00       1.76 r
  LD/ADD_SUB/srl_178/SH[4] (addsub_DW_rightsh_4)      0.00       1.76 r
  LD/ADD_SUB/srl_178/U221/Y (INVX4)                   0.20       1.96 f
  LD/ADD_SUB/srl_178/U216/Y (INVX8)                   0.21       2.16 r
  LD/ADD_SUB/srl_178/U64/Y (NOR2X1)                   0.25       2.41 f
  LD/ADD_SUB/srl_178/U48/Y (MUX2X1)                   0.15       2.57 r
  LD/ADD_SUB/srl_178/U16/Y (NOR2X1)                   0.15       2.72 f
  LD/ADD_SUB/srl_178/B[10] (addsub_DW_rightsh_4)      0.00       2.72 f
  LD/ADD_SUB/U409/Y (BUFX2)                 0.29       3.01 f
  LD/ADD_SUB/sub_194/A[10] (addsub_DW01_sub_49)       0.00       3.01 f
  LD/ADD_SUB/sub_194/U331/Y (OR2X2)                   0.23       3.24 f
  LD/ADD_SUB/sub_194/U138/Y (NAND2X1)                 0.15       3.39 r
  LD/ADD_SUB/sub_194/U126/Y (NOR2X1)                  0.14       3.53 f
  LD/ADD_SUB/sub_194/U301/Y (NAND3X1)                 0.22       3.75 r
  LD/ADD_SUB/sub_194/U344/Y (OAI21X1)                 0.16       3.90 f
  LD/ADD_SUB/sub_194/U10/Y (AOI21X1)                  0.15       4.06 r
  LD/ADD_SUB/sub_194/U4/Y (OAI21X1)                   0.14       4.19 f
```

```
LD/ADD_SUB/sub_194/U284/Y (XOR2X1)                  0.25       4.45 f
LD/ADD_SUB/sub_194/DIFF[23] (addsub_DW01_sub_49)        0.00       4.45 f
LD/ADD_SUB/U300/Y (BUFX2)                          0.26       4.71 f
LD/ADD_SUB/U628/Y (OAI21X1)                        0.19       4.90 r
LD/ADD_SUB/U229/Y (AND2X2)                         0.29       5.19 r
LD/ADD_SUB/U2191/Y (AOI22X1)                       0.13       5.31 f
LD/ADD_SUB/U2192/Y (OAI21X1)                       0.26       5.57 r
LD/ADD_SUB/U2207/Y (AOI22X1)                       0.13       5.70 f
LD/ADD_SUB/U1007/Y (OAI21X1)                       0.17       5.87 r
LD/ADD_SUB/U2280/Y (INVX2)                         0.12       5.99 f
LD/ADD_SUB/U2286/Y (OAI21X1)                       0.15       6.14 r
LD/ADD_SUB/U372/Y (AND2X1)                         0.14       6.28 r
LD/ADD_SUB/U2421/Y (AOI21X1)                       0.12       6.40 f
LD/ADD_SUB/U2422/Y (OAI21X1)                       0.14       6.54 r
LD/ADD_SUB/U2593/Y (AOI22X1)                       0.13       6.68 f
LD/ADD_SUB/U734/Y (NAND2X1)                        0.13       6.81 r
LD/ADD_SUB/U333/Y (AOI22X1)                        0.13       6.94 f
LD/ADD_SUB/U2594/Y (OAI21X1)                       0.13       7.07 r
LD/ADD_SUB/U1294/Y (AND2X2)                        0.16       7.23 r
LD/ADD_SUB/U2704/Y (AOI21X1)                       0.12       7.35 f
LD/ADD_SUB/U690/Y (NAND2X1)                        0.13       7.48 r
LD/ADD_SUB/U2705/Y (AOI21X1)                       0.11       7.59 f
LD/ADD_SUB/U2706/Y (OAI21X1)                       0.21       7.80 r
LD/ADD_SUB/U1117/Y (AND2X2)                        0.16       7.96 r
LD/ADD_SUB/U491/Y (BUFX2)                          0.18       8.14 r
LD/ADD_SUB/U3038/Y (AOI21X1)                       0.12       8.26 f
LD/ADD_SUB/U3074/Y (OAI21X1)                       0.13       8.40 r
LD/ADD_SUB/U3097/Y (NOR2X1)                        0.17       8.57 f
LD/ADD_SUB/U3098/Y (NAND2X1)                       0.13       8.70 r
LD/ADD_SUB/add_result[22] (addsub)                 0.00       8.70 r
LD/OUTPUT/add_result[22] (outdecode)               0.00       8.70 r
LD/OUTPUT/U770/Y (INVX2)                           0.08       8.78 f
LD/OUTPUT/U775/Y (OAI21X1)                         0.10       8.88 r
LD/OUTPUT/fifo_in_reg[22]/D (DFFSR)                0.00       8.88 r
data arrival time                                             8.88

clock clk (rise edge)                              2.00       2.00
clock network delay (ideal)                        0.00       2.00
LD/OUTPUT/fifo_in_reg[22]/CLK (DFFSR)              0.00       2.00 r
library setup time                                -0.22       1.78
data required time                                            1.78
--------------------------------------------------------------------------------
data required time                                            1.78
data arrival time                                            -8.88
--------------------------------------------------------------------------------
```

4. Verification

Verification Plan Summary

What to Verify	Design Module(s) Involved	Verification Procedure Summary	DSSC(s) Proved	Use in Final Demo	Comments
Correctness of Add operation results	Add/Sub Block	Compare program result to test case result	DSSCs 1 & 5	Yes	Use Add/Sub test bench
Correctness of Subtract operation results	Add/Sub Block	Compare program result to test case result	DSSCs 1 & 5	Yes	Use Add/Sub test bench
Correctness of Multiply operation results	Multiply Block	Compare program result to test case result	DSSCs 1 & 5	Yes	Use Multiply test bench
Correctness of Sine operation results	Sine/Cosine Block	Compare program result to test case result	DSSCs 1 & 5	Yes	Use Sine/Cosine test bench
Correctness of Cosine operation results	Sine/Cosine Block	Compare program result to test case result	DSSCs 1 & 5	Yes	Use Sine/Cosine test bench
Input FIFO operates as expected when full	Input Decode Block	CPU receives a hold signal when full	DSSC 4	Yes	Use test bench for entire program
Output FIFO operates as expected when full	Output Decode Block	CPU receives a hold signal when full	DSSC 4	Yes	Use test bench for entire program
All FIFOs receive and produce data as expected	Input Decode Block, Opcode FIFO Block, Output Decode Block	Check that when Add, Sine, and Multiply operations are received in that order, Add, Sine, and Multiply results are produced in that order	DSSC 3 & 4	Yes	Use test bench for entire program
Check that all operations can be done in 400 clock cycles	Add/Sub Block, Multiply Block, Sine/Cosine Block	Compare number of clock cycles determined by the test bench to 60 clock cycles	DSSC 2	Yes	See the result of the test bench (determine clock cycles used)

Detailed Verification Test Breakouts
Demo Tests

Correctness of Add operation results

- Shown in Demo: Yes
- DSSC(s) Proved: 1 & 5
- Highest Level of Design Module(s) involved:
  - Add/Sub Block
- Test bench Expectations/Requirements:
  - Run test bench that tests the Add/Sub block
- IEEE 754 Single-precision Floating Point variable calculation referenced
- Post processing is needed, normalization of the results
- Main Verification Test Steps:
  1. Run the test bench that feeds opcodes and operands to the program and compares the results to the actual calculated result.

Correctness of Subtract operation results

- Shown in Demo: Yes
- DSSC(s) Proved: 1 & 5
- Highest Level of Design Module(s) involved:
  - Add/Sub Block
- Test bench Expectations/Requirements:
  - Run the test bench that tests the Add/Sub Block
- IEEE 754 Single-precision Floating Point variable calculation referenced
- Post processing is needed, normalization of the results
- Main Verification Test Steps:

  1. Run the test bench that feeds opcodes and operands to the program and compares the results to the actual calculated result.

Correctness of Multiply operation results

- Shown in Demo: Yes
- DSSC(s) Proved: 1 & 5
- Highest Level of Design Module(s) involved:
  - Multiply Block
- Test bench Expectations/Requirements:
  - Run the test bench that tests the Multiply Block
- IEEE 754 Single-precision Floating Point variable calculation referenced
- Post processing is needed, normalization of the results
- Main Verification Test Steps:

  1. Run the test bench that feeds opcodes and operands to the program and compares the results to the actual calculated result.

Correctness of Sine operation results

- Shown in Demo: Yes
- DSSC(s) Proved: 1 & 5
- Highest Level of Design Module(s) involved:
  - Sine/Cosine Block
- Test bench Expectations/Requirements:
  - Run the test bench that tests the Sine/Cosine Block
- IEEE 754 Single-precision Floating Point variable calculation referenced
- Taylor Series referenced from http://mathworld.wolfram.com/TaylorSeries.html
- Post processing is needed, normalization of the results
- Main Verification Test Steps:

  1. Run the test bench that feeds opcodes and an operand to the program and compares the results to the actual calculated result.

Correctness of Cosine operation results

- Shown in Demo: Yes
- DSSC(s) Proved: 1 & 5
- Highest Level of Design Module(s) involved:
  - Sine/Cosine Block
- Test bench Expectations/Requirements:
  - Run the test bench that tests the Sine/Cosine Block
- IEEE 754 Single-precision Floating Point variable calculation referenced
- Taylor Series referenced from http://mathworld.wolfram.com/TaylorSeries.html
- Post processing is needed, normalization of the results
- Main Verification Test Steps:

  1. Run the test bench that feeds opcodes and an operand to the program and compares the results to the actual calculated result.

Input FIFO operates as expected when full

- Shown in Demo: Yes
- DSSC(s) Proved: 4
- Highest Level of Design Module(s) involved:
  - Input Comb Logic/FIFO Block
- Test bench Expectations/Requirements:
  - Run the test bench that tests the whole program
- No external or premade references are needed
- No pre/post processing is needed
- Main Verification Test Steps:
  1. Run the test bench that feeds more than eight instances of opcodes and operands to the program
  2. Check that the CPU receives a triggered FIFO_FULL signal

Output FIFO operates as expected when full

- Shown in Demo: Yes
- DSSC(s) Proved: 4
- Highest Level of Design Module(s) involved:
    - Output Comb Logic/FIFO Block
- Test bench Expectations/Requirements:
    - Run the test bench that test the whole program
- No external or premade references are needed
- No pre/post processing is needed
- Main Verification Test Steps:
    1. Run the test bench that feeds more than eight instances of results to the program
    2. Check that the CPU receives a triggered FIFO_FULL signal

All FIFOs receive and produce data as expected

- Shown in Demo: Yes
- DSSC(s) Proved: 3 & 4
- Highest Level of Design Module(s) involved:
    - Input Comb Logic/FIFO Block
    - Output Comb Logic/FIFO Block
- Test bench Expectations/Requirements:
    - Run the test bench that tests the whole program
- No external or premade references are needed
- No pre/post processing is needed
- Main Verification Test Steps:
    1. Run the test bench that feeds multiple instances of data in a specific order and see if they come out in the same specific order.

Check that all operations can be done in 400 clock cycles (2.5MHz)

- Shown in Demo: Yes
- DSSC(s) Proved: 2
- Highest Level of Design Module(s) involved: Add/Sub block, Multiply block, Sine/Cosine block
- Test bench Expectations/Requirements: Add/Sub block runs in less than 50 clock cycles, Multiply block runs in less than 20 clock cycles, Sine/Cosine uses less than 400 clock cycles.
- No external or premade references are needed
- No pre/post processing is needed
- Main Verification Test Steps:
    1. Prepare a test bench for each blocks
    2. Run the test bench and run the test bench for 400 ns.
    3. See if all the results output in 400 ns. (1ns clock cycle, 1GHz clock)

5. Layout

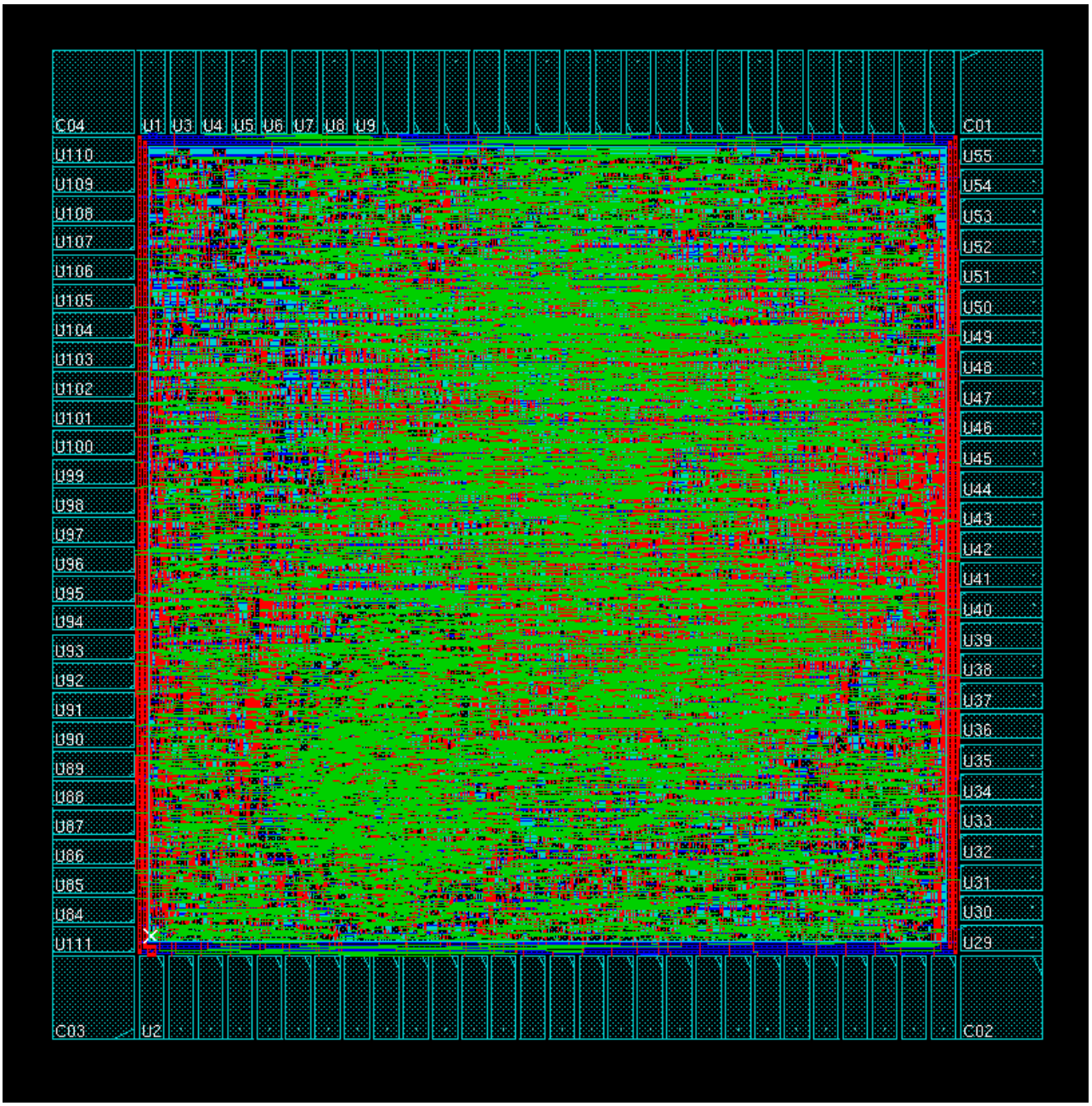


Figure 13: Layout for the Entire FPU

Aspect Ratio: 50:50
Dimensions: 3600x3600 um
Area in mm2 = 4,919,886,000.00000 mm^2
Number of Gates = 518
Number of I/O Pads = 111

6. Results

Fixed Success Criteria

1. Test benches exist for all top level components and the entire design. The test benches for the entire design can be demonstrated or documented to cover all of the functional requirements given in the design specific success criteria. (2 pts)

- Completed

2. Entire design synthesizes completely, without any inferred latches, timing arcs, and, sensitivity list warnings. (4pts)

- Completed

3. Source and mapped version of the complete design behave the same for all test cases. The mapped version simulates without timing errors except at time zero. (2 pts)

- Completed

4. A complete IC layout is produced that passes all geometry and connectivity checks. (2pts)

- Completed

5. The entire design complies with targets for area, pin count, throughput (if applicable), and clock rate. The final targets for these parameters will be determined by course staff based on your design review. Failure to reach any of the targets will result a score of 1 out of 2 provided that you are within 50% on area, 10% on pin count, and 25% on throughput. Doing worse in any category will result in a score of 0. (2 pts)

a. Area: 4mm x 4mm
   - -4,919,886.000000 um^2

b. Pin Count: 164 (161-pin AHB-Lite + power, ground, clock)
   - -104 pins total

c. Clock Period: 100 MHz
   - -250 MHz, 4ns clock

Design Specific Criteria

1. Demonstrate by simulation of Verilog test benches that the complete design is able to perform five different operations, add, subtract, multiply, sine, and cosine. (3 pts)
    - Completed

2. Demonstrate by simulation of Verilog test benches that the complete design is able to perform the multiply, sine, and cosine operations within 60 clock cycles. (1 pt)
    - Completed

3. Demonstrate by simulation of Verilog test benches that the complete design can take in multiple operation requests before producing a result. (1 pt)
    - Completed

4. Demonstrate by simulation of Verilog test benches that the complete design is able to consecutively take in an add or subtract, a multiply, or a sine or cosine opcode, perform each of these operations at the same time, and produce the outputs in the order that the opcodes were received. (2 pts)
    - Completed

5. Demonstrate by simulation of Verilog test benches that the complete design is able to yield the correct results for each operation for add, subtract, multiply, sine, and cosine within 3 decimal places. (1 pt)
    - Completed

Appendix A

Account and directory where all of the files are located: mg71/ECE337/Project/verilog-fp

Top level structural VERILOG code source

- Addition and Subtraction Module: source/ addsub.sv
- Opcode FIFO Buffer Module: source/fifobuff.sv
- Input Decode Module: source/indecode.sv
- Multiplication Module: source/multiple.sv
- Output Decode Module: source/outdecode.sv
- Sine and Cosine Module: source/sincos.sv
- Overall Top-Level File Combining All Modules: source/wrapper.sv

Test Benches

- Addition and Subtraction Module Test Bench: source/tb_addsub.sv
- Opcode FIFO Buffer Module Test Bench: source/tb_fifobuff.sv
- Input Decode Module Test Bench: source/tb_indecode.sv
- Multiplication Module Test Bench: source/tb_multiple.sv
- Output Decode Module Test Bench: source/tb_outdecode.sv
- Sine and Cosine Module Test Bench: source/tb_sincos.sv
- Overall Top-Level Test Bench: source/tb_wrapper.sv

Appendix B

Input Decode Module Results

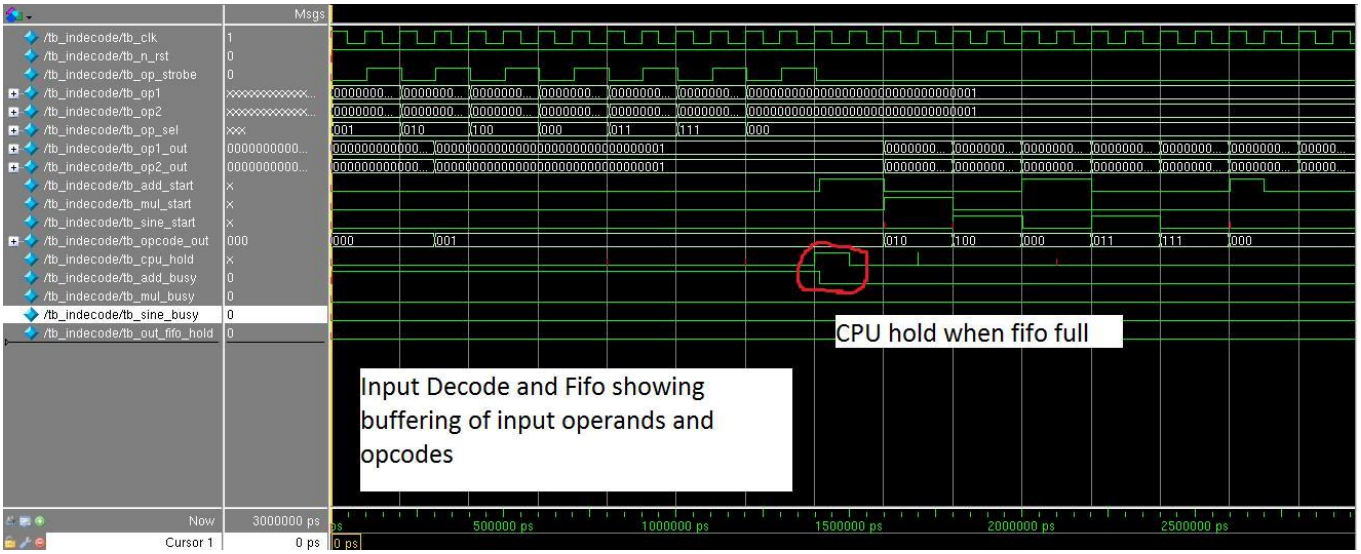


Figure 14: Simulation Waveform for the Input Decode Module

Add/Subtract Module Results

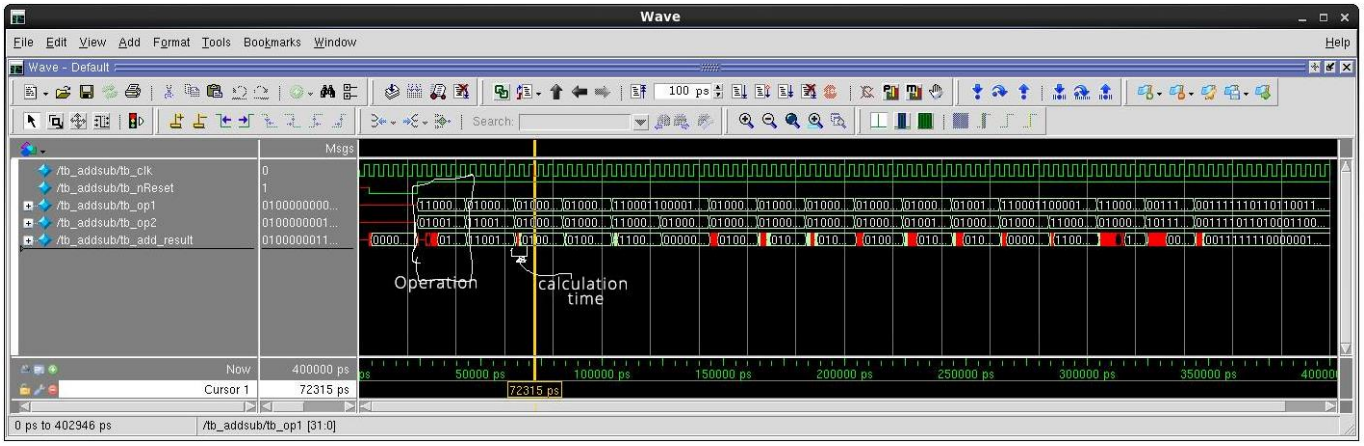


Figure 15: Simulation Waveform for the Add/Sub Block

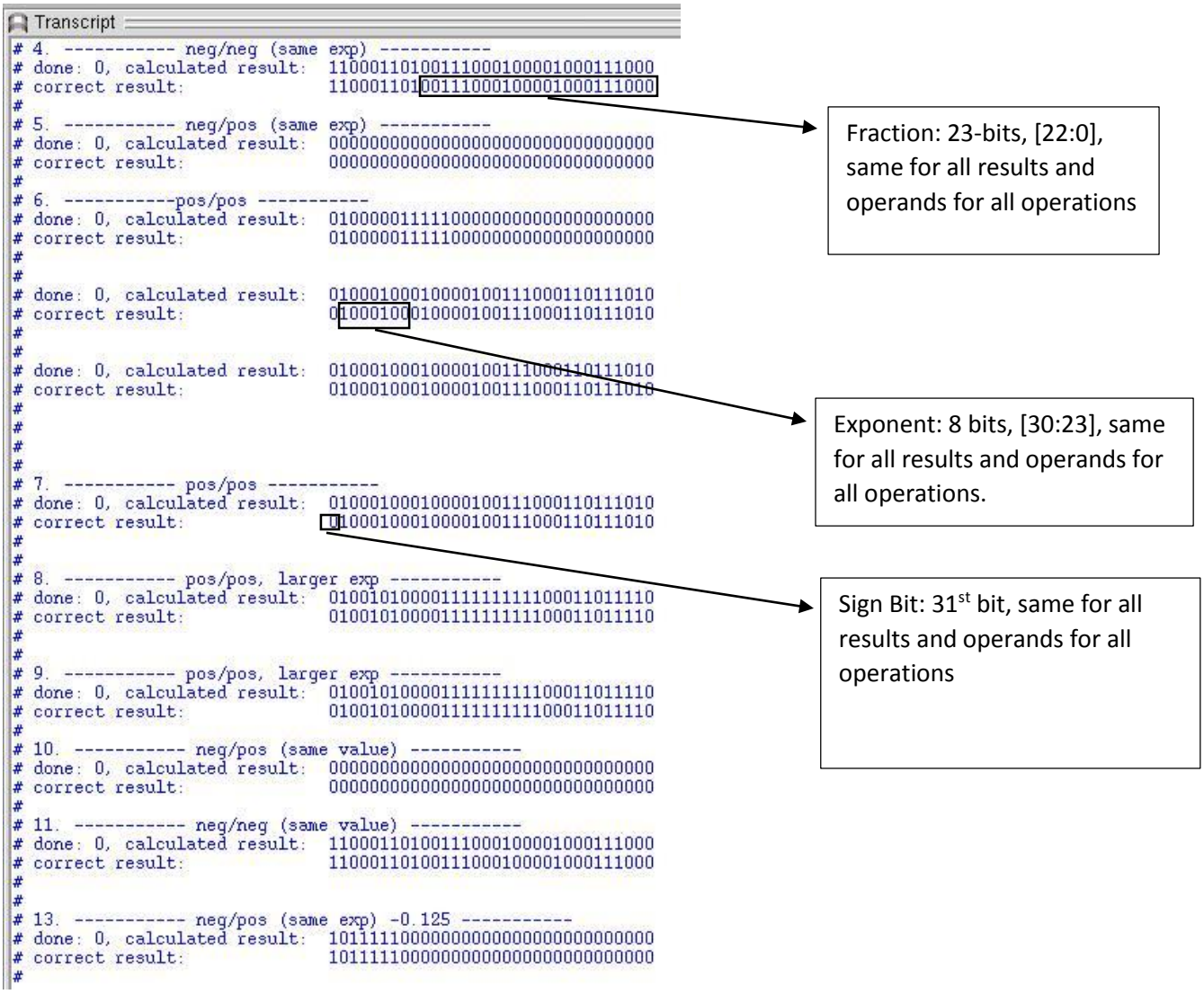


Figure 16: Results for the Test Cases for the Add/Sub Block

Multiply Module Results

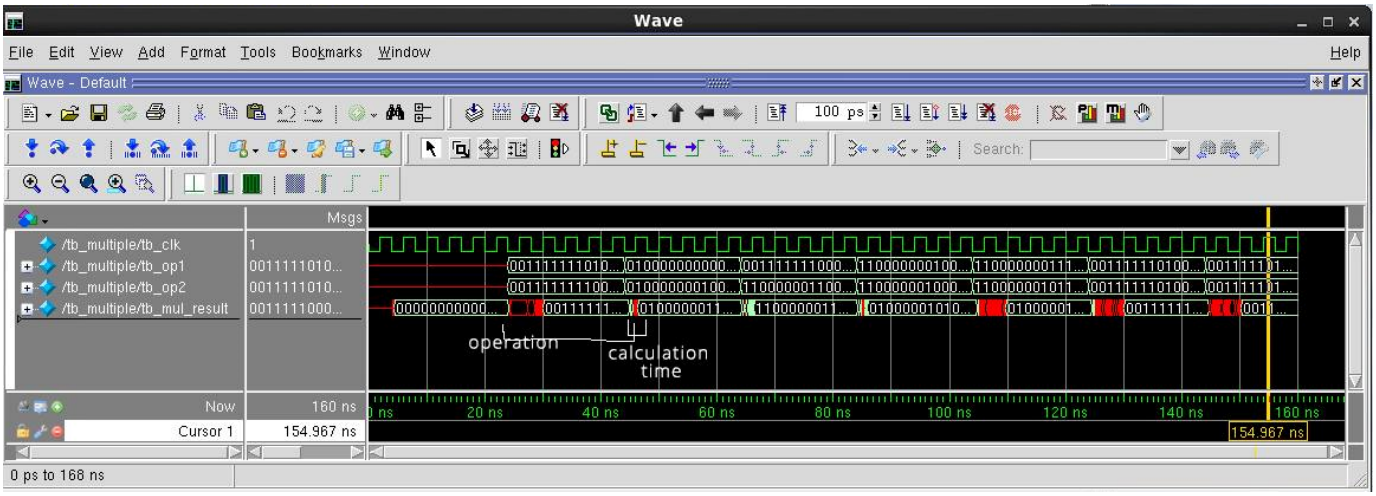


Figure 17: Simulation Waveform for the Multiplication Block

```
# ----------- reset -----------
# correct result:              00000000000000000000000000000000
# done: 0, calculated result:  00000000000000000000000000000000
#
# ----------- pos/pos -----------
# correct result:              00111111111110000000000000000000
# done: 1, calculated result:  00111111111110000000000000000000
#
#
# ----------- pos/pos -----------
# correct result:              01000000110000000000000000000000
# done: 1, calculated result:  01000000110000000000000000000000
#
#
# ----------- pos/neg -----------
# correct result:              11000000110000000000000000000000
# done: 1, calculated result:  11000000110000000000000000000000
#
#
# ----------- neg/neg -----------
# correct result:              01000001010000000000000000000000
# done: 1, calculated result:  01000001010000000000000000000000
#
#
```

Figure 18: Results for the Test Cases for the Multiplication Block

Sine/Cosine Module Results

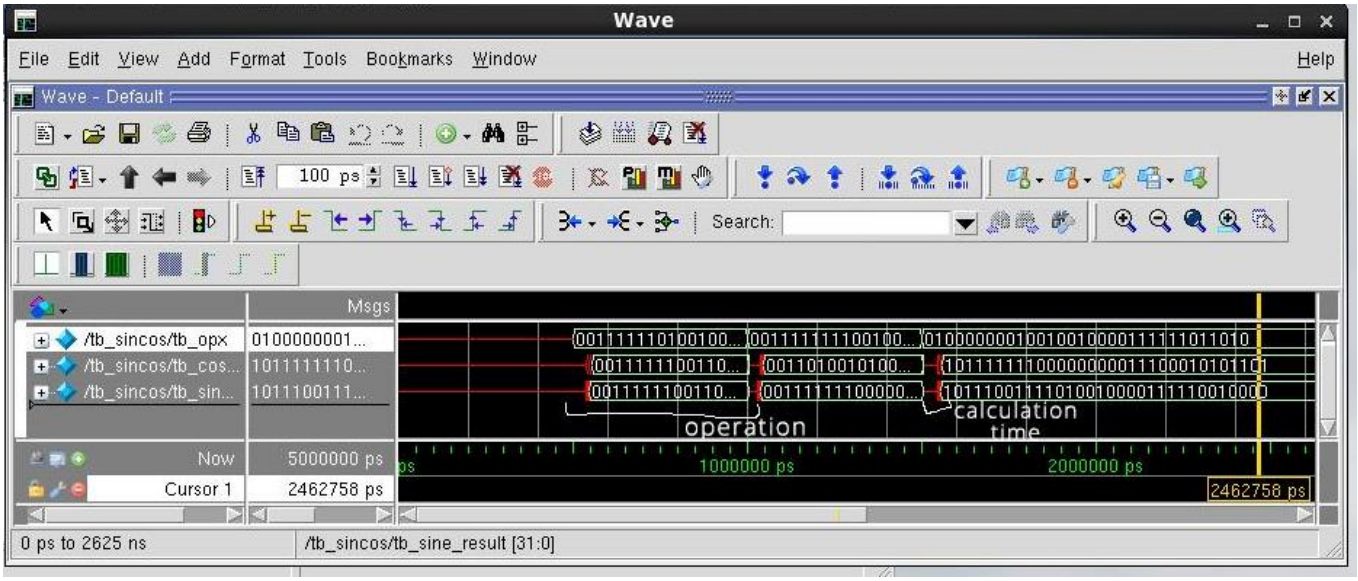


Figure 19: Simulation Waveform for the Sine/Cosine Block

```
#
# ----------- pi/4 -----------
# correct result:              0.707106, 0.707106
# correct result:               00111111100110101000001001110011 0
# done:     sine result:      00111111100110101000001001111000 1
# done:     cosine result:    00111111100110101000001001111001 10
#
# ----------- pi/2 -----------
# correct result:                1.0, 0.0
# correct result:               00111111110000000000000000000000, 0
# done:     sine result:      00111111110000000100101000100010 1
# done:     cosine result:    00110100101000001011101101010101
#
# ----------- pi -----------
# correct result:                0.0, -1.0
# correct result:               0, 10111111110000000000000000000000
# done:     sine result:      10111001111010010000111110010000
# done:     cosine result:    10111111110000000001110001010110 1

VSIM 5>
```

Now: 5 us Delta: 2 | /tb_sincos/tb_sine_result [31:0]

Figure 20: Results for the Test Cases for the Sine/Cosine Block

Overall Results

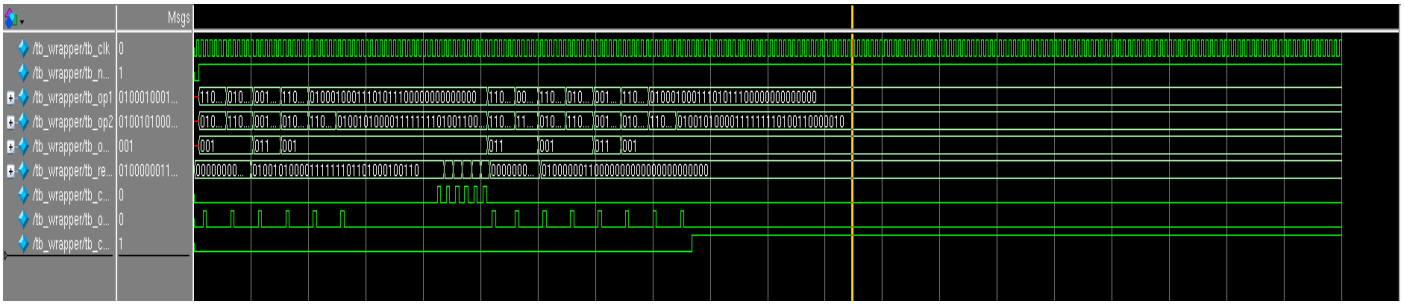


Figure 21: Simulation Waveform for the Overall FPU Minus the Sine/Cosine Block

Overall FPU Results:

Power on Reset, waiting...
Queing Add operation...
Queing Add operation...
Queing Mul operation...
Queing Add operation...
Queing Add operation...
Queing Add operation...
Waiting for operation to complete...
Pulling result...
done: calculated result: 01001010000111111101101000100110
correct result: 01001010000111111101101000100110
values match
Pulling result...
done: calculated result: 11001010000111111101101000100110
correct result: 11001010000111111101101000100110
values match
Pulling result...
done: calculated result: 00111111111100000000000000000000
correct result: 00111111111100000000000000000000
values match
Pulling result...
done: calculated result: 01001010000111111101101000100110
correct result: 01001010000111111101101000100110
values match
Pulling result...
done: calculated result: 11001010000111111101101000100110
correct result: 11001010000111111101101000100110
values match
Pulling result...
done: calculated result: 01001010000111111111100011011110
correct result: 01001010000111111111100011011110
values match
Filling Fifo...

```
# Queing Mul operation...
# Fifo Full: 0
# Queing Mul operation...
# Fifo Full: 0
# Queing Add operation...
# Fifo Full: 0
# Queing Add operation...
# Fifo Full: 0
# Queing Mul operation...
# Fifo Full: 0
# Queing Add operation...
# Fifo Full: 0
# Queing Add operation...
# Fifo Full: 0
# Queing Add operation...
# Fifo Full: 1
```

Coverage Reports:

Coverage Summary by Structure:

Design Scope ◂	Coverage (%) ◂
tb_fifobuff	75.63%
push	100.00%
pull	100.00%
DUT	73.05%

Coverage Summary by Type:

Weighted Average:				75.63%
Coverage Type ◂	**Bins ◂**	**Hits ◂**	**Misses ◂**	**Coverage (%) ◂**
Statement	88	81	7	92.04%
Branch	38	31	7	81.57%
Toggle	152	81	71	53.28%

Figure 22: Coverage Report for the Opcode FIFO Buffer Module Test Bench

Coverage Summary by Structure:

Design Scope ◂	Coverage (%) ◂
tb_indecode	59.40%
push	100.00%
DUT	56.95%

Coverage Summary by Type:

Weighted Average:				59.40%
Coverage Type ◂	**Bins ◂**	**Hits ◂**	**Misses ◂**	**Coverage (%) ◂**
Statement	126	104	22	82.53%
Branch	49	36	13	73.46%
Toggle	2870	637	2233	22.19%

Report generated by ModelSim on Wed 06 May 2015 10:51:29 PM EDT

Figure 23: Coverage Report for the Input Decode Module Test Bench

Coverage Summary by Structure:

Design Scope	Coverage (%)
tb_multiple	62.69%
MULTI	58.96%

Coverage Summary by Type:

Weighted Average:				62.69%
Coverage Type	**Bins**	**Hits**	**Misses**	**Coverage (%)**
Statement	55	53	2	96.36%
Branch	4	3	1	75.00%
Toggle	580	97	483	16.72%

Report generated by ModelSim on Wed 06 May 2015 10:46:21 PM EDT

Figure 24: Coverage Report for the Multiplication Module Test Bench

Coverage Summary by Structure:

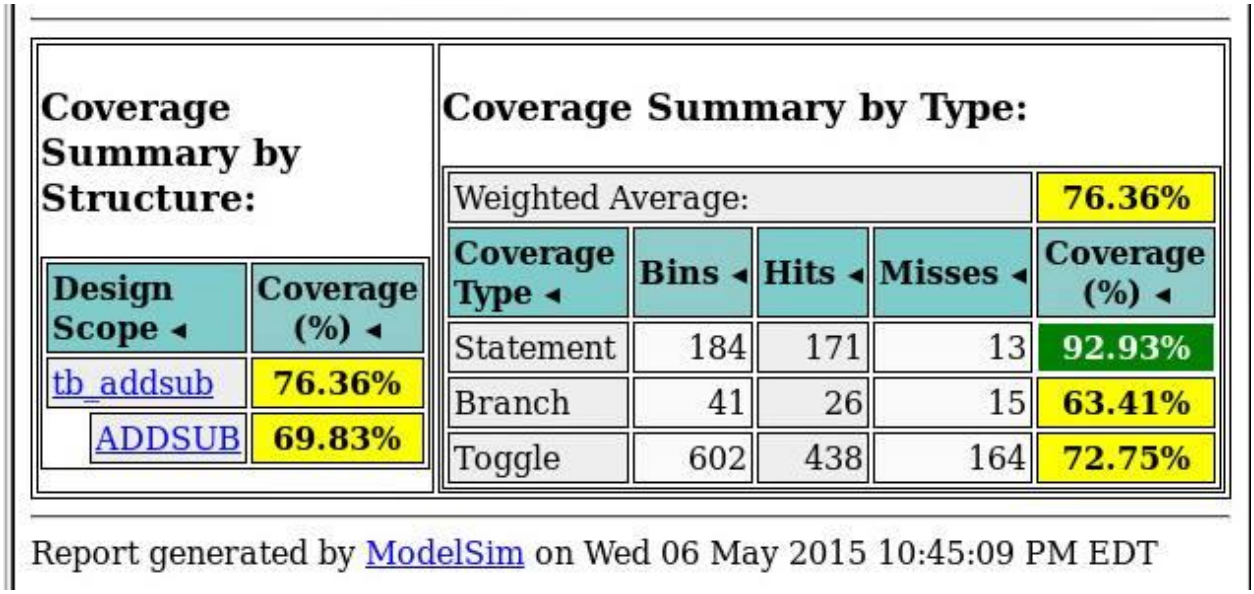
Design Scope	Coverage (%)
tb_sincos	57.39%
SINCOS	55.58%

Coverage Summary by Type:

Weighted Average:				57.39%
Coverage Type	**Bins**	**Hits**	**Misses**	**Coverage (%)**
Statement	173	157	16	90.75%
Branch	43	26	17	60.46%
Toggle	1178	247	931	20.96%

Report generated by ModelSim on Wed 06 May 2015 10:47:59 PM EDT

Figure 25: Coverage Report for the Sine and Cosine Module Test Bench



Coverage Summary by Structure:

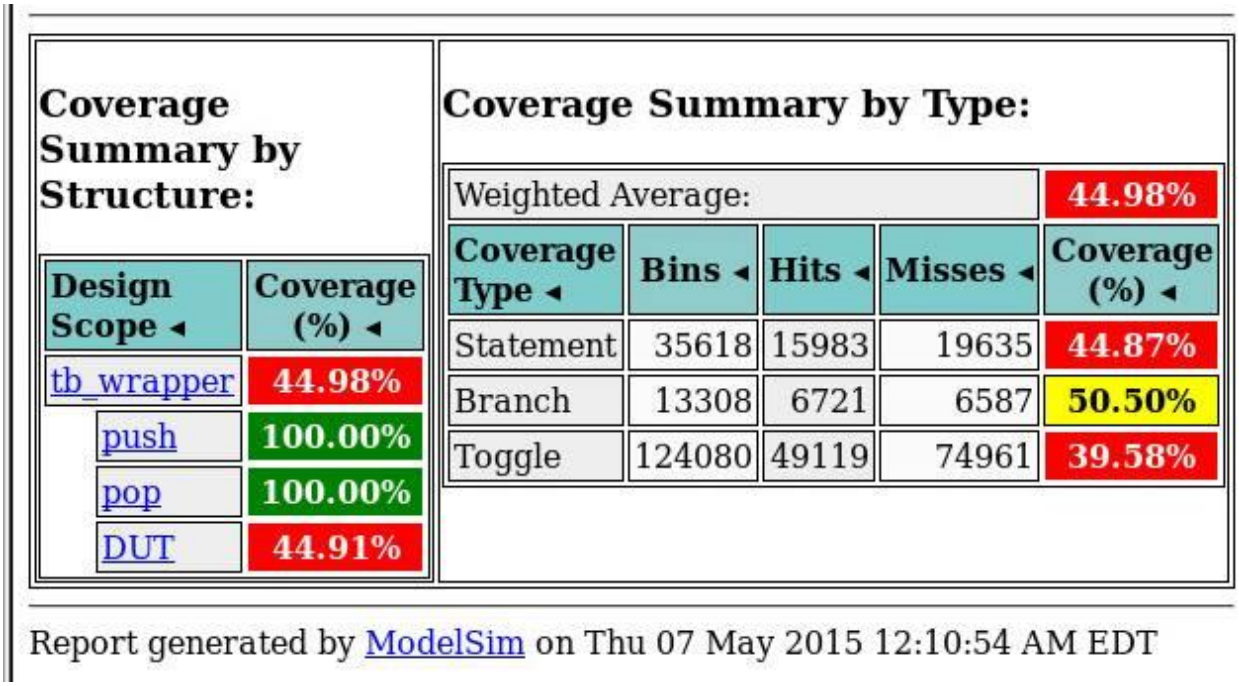
Design Scope	Coverage (%)
tb_addsub	76.36%
ADDSUB	69.83%

Coverage Summary by Type:

Weighted Average:				76.36%
Coverage Type	**Bins**	**Hits**	**Misses**	**Coverage (%)**
Statement	184	171	13	92.93%
Branch	41	26	15	63.41%
Toggle	602	438	164	72.75%

Report generated by ModelSim on Wed 06 May 2015 10:45:09 PM EDT

Figure 26: Coverage Report for the Addition and Subtraction Module Test Bench



Coverage Summary by Structure:

Design Scope	Coverage (%)
tb_wrapper	44.98%
push	100.00%
pop	100.00%
DUT	44.91%

Coverage Summary by Type:

Weighted Average:				44.98%
Coverage Type	**Bins**	**Hits**	**Misses**	**Coverage (%)**
Statement	35618	15983	19635	44.87%
Branch	13308	6721	6587	50.50%
Toggle	124080	49119	74961	39.58%

Report generated by ModelSim on Thu 07 May 2015 12:10:54 AM EDT

Figure 27: Coverage Report for the Overall FPU Test Bench