

An 8 Bit by 8 Bit Booth Multiplier

Rick Fenster - 22597217

November 30, 2015

Contents

1	Introduction	6
2	Project Requirements	7
3	Design Overview	7
3.1	Description of the Device	7
3.2	Design Methodology	8
3.3	Functional Description of Non-Pipelined Variant	8
3.4	Functional Description of Pipelined Variant	9
4	Design of Fundamental Gates	10
4.1	NOT Gate	10
4.2	AND2 Gate	10
4.3	NAND2 Gate	11
4.4	NAND3 Gate	11
4.5	OR2 Gate	12
4.6	XOR2 Gate	12
5	Design of Basic Functional Blocks	13
5.1	The Half-Adder	13
5.2	The Full Adder	14
5.3	D Flip Flop	15
5.4	Two to One Multiplexer	18
6	Design of Higher Level Functional Blocks	19
6.1	9 Bit Incrementer	19
6.2	Two's Complementer	20
6.3	9 Bit Barrel Shifter	22
6.4	Ripple Carry Adder Blocks	24
7	Building the Adders	25
7.1	Introduction to the Carry Select Adder	25
7.2	Carry Select Adder Blocks	26
7.3	Carry Select Adders	27
8	Booth Units and the Booth Adder	29
8.1	Booth Decoder	29
8.2	Booth Units	31
8.3	The Booth Adder	31
9	Non-Pipelined Implementation	32
9.1	Introduction	32
9.2	The Operand Register	33
9.3	The Booth8 Multiplier	36

10 Pipelined Implementation	38
10.1 Introduction	38
10.2 Pipeline Stage 0	38
10.3 Pipeline Stage 1	39
10.4 Top Level Implementation	41
11 Synthesis and Analysis of the Multiplier	43
11.1 Introduction	43
11.2 Results for Non-Pipelined Version	43
11.3 Results for the Pipelined Version	43
11.4 Alternate Method for Pipelined Version	44
11.4.1 Pipeline Stage 0	44
11.4.2 Pipeline Stage 1	44
11.5 Conclusions and Improvements	45
Appendices	45
A VHDL Source Code	45
A.1 Logic Gates	45
A.1.1 AND2	45
A.1.2 NAND2	45
A.1.3 NAND3	46
A.1.4 NOT	46
A.1.5 OR2	47
A.1.6 XOR2	47
A.2 Small Functional Blocks	48
A.2.1 D Flip Flop with Active Low Reset and Positive Edge Clock	48
A.2.2 Full Adder	49
A.2.3 Half Adder	50
A.2.4 2 To 1 Multiplexer	51
A.3 Adders	52
A.3.1 2 Bit Carry Select Block	52
A.3.2 4 Bit Carry Select Block	53
A.3.3 10 Bit Carry Select Adder	54
A.3.4 12 Bit Carry Select Adder	55
A.3.5 14 Bit Carry Select Adder	56
A.3.6 9 Bit Incrementer	58
A.3.7 4 Bit Ripple Carry Adder Block	59
A.4 Higher Level Functional Blocks	60
A.4.1 Booth Adder, Device Under Test	60
A.4.2 9 Bit Barrel Shifter	61
A.4.3 Booth Adder	62
A.4.4 Booth Decoder	64
A.4.5 Booth Unit 0 (16 Bit)	65
A.4.6 Booth Unit 1 (14 Bit)	66
A.4.7 Booth Unit 2 (12 Bit)	67
A.4.8 Booth Unit 3 (10 Bit)	68

A.4.9	9 Bit Two's Complementer	69
A.5	Registers	70
A.5.1	8 Bit Operand Register	70
A.5.2	10 Bit Operand Register	71
A.5.3	14 Bit Operand Register	72
A.5.4	16 Bit Operand Register	73
A.6	Non-Pipelined Implementation	74
A.7	Pipelined Implementation	76
A.7.1	Top-Level Implementation	76
A.7.2	Pipeline Stage 0	78
A.7.3	Pipeline Stage 1	80
B	Test Scripts	82
B.1	Logic Gates	82
B.1.1	AND2	82
B.1.2	NAND2	82
B.1.3	NAND3	83
B.1.4	NOT	83
B.1.5	OR2	84
B.1.6	XOR2	84
B.2	Small Functional Blocks	85
B.2.1	D Flip Flop with Active Low Reset and Positive Edge Clock	85
B.2.2	Full Adder	85
B.2.3	Half Adder	86
B.2.4	2 To 1 Multiplexer	87
B.3	Adders	87
B.3.1	2 Bit Carry Select Block	87
B.3.2	4 Bit Carry Select Block	88
B.3.3	10 Bit Carry Select Adder	89
B.3.4	12 Bit Carry Select Adder	89
B.3.5	14 Bit Carry Select Adder	90
B.3.6	9 Bit Incrementer	91
B.3.7	4 Bit Ripple Carry Adder Block	91
B.4	Higher Level Functional Blocks	92
B.4.1	Booth Adder, Device Under Test	92
B.4.2	9 Bit Barrel Shifter	93
B.4.3	Booth Adder	94
B.4.4	Booth Decoder	95
B.4.5	Booth Unit 0 (16 Bit)	95
B.4.6	Booth Unit 1 (14 Bit)	97
B.4.7	Booth Unit 2 (12 Bit)	98
B.4.8	Booth Unit 3 (10 Bit)	99
B.4.9	9 Bit Two's Complementer	100
B.5	Registers	101
B.5.1	8 Bit Operand Register	101
B.6	Non-Pipelined Implementation	102

B.7	Pipelined Implementation	104
B.7.1	Booth8 Pipeline Stage 0	104
B.7.2	Booth8 Pipeline Stage 1	105
B.7.3	Booth8 Pipeline Top Level Implementation	106
C	Synthesis and Timing Reports	108
C.1	Non-Pipelined Implementation	108
C.1.1	PrecisionRTL Area Report	108
C.1.2	Xilinx ISE Timing Report	110
C.2	Pipelined Implementation	116
C.2.1	PrecisionRTL Area Report	116
C.2.2	Xilinx ISE Timing Report	118
C.3	Compartmentalized Pipelined Implementation	123
C.3.1	PrecisionRTL Area Report for Stage 0 Combinational Logic	123
C.3.2	Xilinx ISE Timing Report for Stage 0 Combinational Logic	127
C.3.3	PrecisionRTL Area Report for Stage 0 Registers	135
C.3.4	Xilinx ISE Timing Report for Stage 0 Registers	140
C.3.5	PrecisionRTL Area Report for Stage 1 Combinational Logic	144
C.3.6	Xilinx ISE Timing Report for Stage 1 Combinational Logic	147
C.3.7	PrecisionRTL Area Report for Stage 1 Registers	154
C.3.8	Xilinx ISE Timing Report for Stage 1 Registers	156

1 Introduction

In any aspect of computing, the speed of the arithmetic unit is of great concern. Because of this, the implementation of the ALU must be decided carefully. The goal of this project is to build a fast 8 bit by 8 bit multiplier with an output of 16 bits, *focused on speed*. By focusing on speed, the delay time is intended to be reduced, while the area and power consumption of the device are expected to be focused less on.

By specifications provided for the project, the multiplier must accept 8 bit signed inputs and output a 16 bit resultant. Because of this, the theoretical smallest and largest input values are 127 and -128. With this in mind, it can be safely assumed that the largest output value of the multiplier is 16384 or **0b0100000000000000**. Additionally, the smallest possible resultant is -16256 or **0b1100000010000000**.

The architecture chosen for this multiplier is a **radix-2 Booth multiplier**. A Booth multiplier achieves a reasonable compromise on speed and size because it does not need additional supporting logic for counters such as those needed in serial or serial/parallel multipliers. Instead, the multiplication is performed by generating partial products from decoding chunks of the multiplier and then manipulating the multiplicand by negating, shifting, zeroing or applying any combination to it. In the case of an 8 bit by 8 bit radix-2 Booth multiplier, there will be four partial products generated and then added together to obtain a final result. That being said, the Booth multiplier requires sign extensions to be functional which adds overhead for addition. The values and their corresponding operations can be seen in Table 1.

Table 1: Booth Radix-2 Operation Codes

<i>Input</i>	<i>Operation</i>
000	0
001	1
010	1
011	2
100	-2
101	-1
110	-1
111	-0

The multiplicand is taken and a zero is added on the right. Going from right to left, the three values are taken and decoded to determine what operation is performed on the partial product. The values taken overlap on the right-most side. This means that the most significant bit of the first multiplicand operator is the least significant bit of the second operator.

As an example, let us assume the multiplicand is **0b10110000** and the multiplier is **0b01110110**. This implies

$$0b10110000 \times 0b01110110 \tag{1}$$

A zero is appended to the right side to obtain **011101100**. The modified multiplicand is decoded as shown in Table 1.

Starting with the first partial product, the partial product is computed then sign extended to 16 bits. The next partial product is added but offset by two bits then sign extended. This is continued

until there are no remaining partial products The solution is shown in Figure 1.

Figure 1: Sample Booth Multiplier Decoding

<i>Partial Product</i>	<i>Input</i>	<i>Result</i>
0	100	Multiplicand * -2
1	011	Multiplicand * 2
2	110	Multiplicand * -1
3	011	Multiplicand * 2

Figure 2: Sample Booth Multiplication Example

0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	Partial Product 0
1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	Partial Product 1
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	Partial Product 2
1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	Partial Product 3
1	1	0	1	1	0	1	1	0	0	1	0	0	0	0	0	Sum

When converted into decimal and treated as a signed number, the result obtained is -9440, which is correct because the operation performed was:

$$-80 \times 118 = -9440 \quad (2)$$

2 Project Requirements

There are distinct requirements for the project. They are as follows:

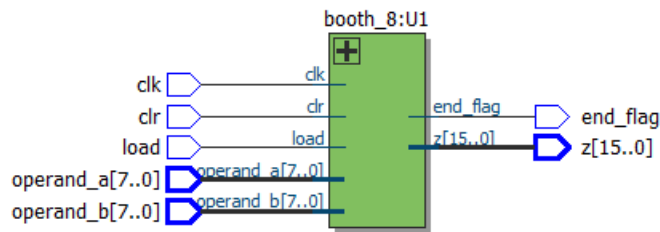
- The goal of the project is to design an 8 bit by 8 bit multiplier
- The project must be written in structural VHDL
- The operands A and B must be written into registers on the negative edge of the LOAD flag
- An END signal must be asserted when the multiplication is complete
- The result must be available on the Z port
- When the CLEAR signal is asserted, all registers will be reset.

3 Design Overview

3.1 Description of the Device

This section details the multiplier and its inputs and outputs. The multiplier has two 8 bit inputs to load both the multiplicand and multiplier. The LOAD signal is a negative edge triggered signal, while the clock is a positive edge triggered signal. CLR is asserted high to clear all registers.

Figure 3: Booth8 Logic Block



3.2 Design Methodology

Since the design requirements constitute a structural implementation, each unit must be carefully designed without the use of sequential VHDL features such as processes. Because of this, the Booth8 multiplier is designed from a *bottom-up* method. Each individual unit and gate is purposefully built and then verified. Once the units have been verified, they are used in other larger modules which are exposed to same process of verification. Additionally, the top level module being the multiplier will be subjected to multiplying its theoretical smallest and largest values along with zeros to ensure proper functionality. It should also be said that the design is focused to allow for code reuse with smaller functional blocks are re-used to build larger units.

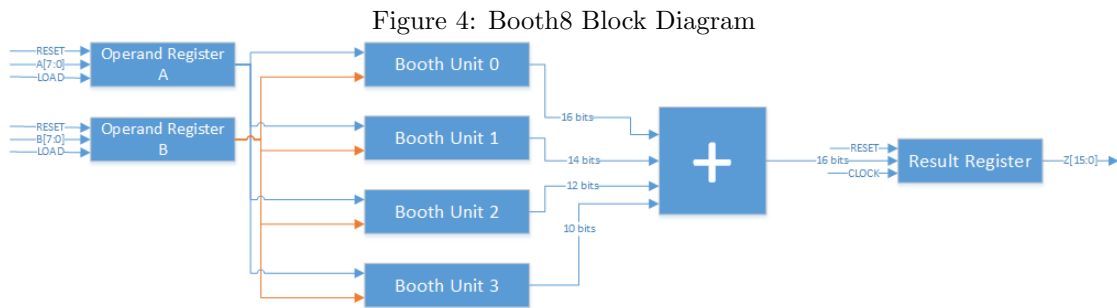
The Booth multiplier has some substantial speed benefits as described previously due to the ability to substantially reduce the number of partial products needed. That being said, sign extending comes at a cost. Sign extensions are performed with additional fan-out, which ultimately delays performance and increases overall power consumption.

Also, while there are less additions needed, the additions are still large. Because of this, simple ripple carry adders will not suffice because of their inherent delay. Since speed is of utmost concern in this implementation, carry select adders are used. The use of these adders are purposefully selected so that redundant additions are not performed such as adding zeros where partial products are offset. This will be discussed in greater detail later on in the report.

Each module is subjected to verification by a simulated test-bench provided by a .do file that is executed in Mentor Graphics ModelSim. Appendix B has the listings of every .do file used when designing the multiplier.

3.3 Functional Description of Non-Pipelined Variant

The non-pipelined version of the Booth8 multiplier loads the operands A and B into registers when LOAD transitions from logic high to logic low. The output of these registers are then fed to sub-circuits referred to as *Booth Units* that perform the necessary operation decoded from the multiplier. Once retrieved, the outputs of the Booth Units are fed into a series of carry select adders to produce the result. The obtained result is then passed onto a resultant register. The output of the resultant is fed to the Z port. A block diagram of the layout is shown in Figure 4.

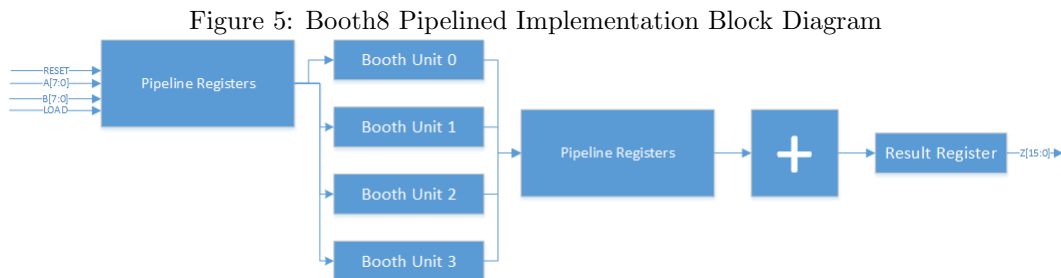


3.4 Functional Description of Pipelined Variant

The pipelined variant operates in a similar manner as the non-pipelined variant. The difference lies in the implementation of additional registers which serve as pipeline stages. The implementation involves two pipeline stages:

1. Partial products generated by Booth Units and data valid flag stored in registers
2. Partial products added, producing the final result and stored into a register, along with the data valid flag

A simplified block diagram can be seen in Figure 5



4 Design of Fundamental Gates

In this section, the design of each basic functional gate is demonstrated, detailed and verified.

4.1 NOT Gate

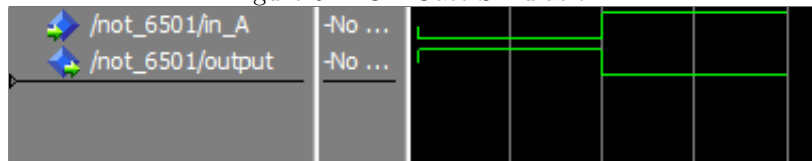
The first component designed is also the simplest, being an inverter. An inverter simply takes its logic input and inverts it. The functionality can be described by the truth table shown in Table 2.

Table 2: NOT Truth Table

<i>Input</i>	<i>Output</i>
0	1
1	0

The simulation has rendered the following, yet appropriate results shown in Figure 6.

Figure 6: NOT Gate Simulation



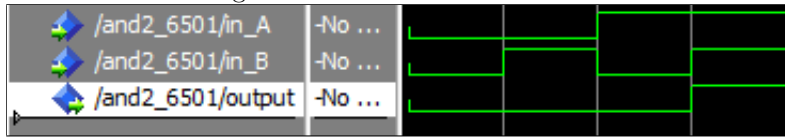
4.2 AND2 Gate

The AND2 gate implements a two input AND gate. Functionally, an AND gate's output should be zero unless all inputs present are equal to a logic 1. The functionality is shown in Table 3 and the simulation results can be seen in Figure 7.

Table 3: AND2 Truth Table

<i>A</i>	<i>B</i>	<i>Output</i>
0	0	0
0	1	0
1	0	0
1	1	1

Figure 7: AND2 Gate Simulation



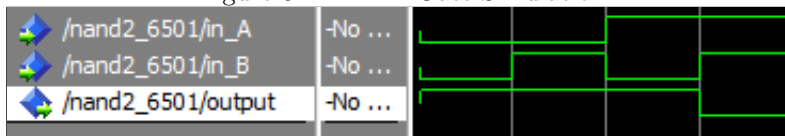
4.3 NAND2 Gate

A NAND gate is described as a logic gate which outputs a logic 1 under all conditions *except when all inputs are logic 1*. It can also be thought as the inverted output of an AND gate. With this in mind, the truth table is presented in Table 4. The simulation results can be seen in Figure 8.

Table 4: NAND2 Truth Table

A	B	$Output$
0	0	1
0	1	1
1	0	1
1	1	0

Figure 8: NAND2 Gate Simulation



4.4 NAND3 Gate

The NAND3 gate has a similar behaviour as the NAND2 gate except the logic of outputting a logic 0 when all inputs are logic 1 extends to three inputs. Table 5 contains the truth table. Simulation results are seen in Figure 9.

Figure 9: NAND3 Gate Simulation



Table 5: NAND3 Truth Table

<i>A</i>	<i>B</i>	<i>C</i>	<i>Output</i>
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

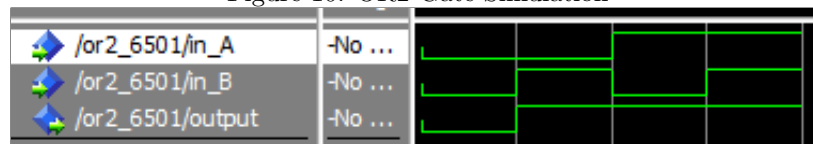
4.5 OR2 Gate

The next logic gate is the OR2 gate. The OR2 gate is a two input OR gate. This implies that the output of the gate will be a logic 1 as long as at least input is a logic 1 as well. The functionality can be described in Table 6. Figure 10 demonstrates the simulation of the OR2 Gate.

Table 6: OR2 Truth Table

<i>A</i>	<i>B</i>	<i>Output</i>
0	0	0
0	1	1
1	0	1
1	1	1

Figure 10: OR2 Gate Simulation



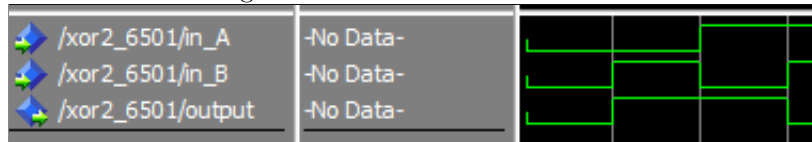
4.6 XOR2 Gate

The last gate built for this project is an XOR2 Gate. An XOR gate outputs a logic 1 when the number of logic 1 inputs are odd. If the number of logic high inputs is 0, it is treated as an even value. The truth table is shown in Table 7 and simulation results are shown in Figure 11.

Table 7: XOR2 Truth Table

<i>A</i>	<i>B</i>	<i>Output</i>
0	0	0
0	1	1
1	0	1
1	1	0

Figure 11: XOR2 Gate Simulation



5 Design of Basic Functional Blocks

With the basic gates described in Chapter 4, functional blocks that can perform basic yet useful functions can be derived. These functional blocks are designed in a structural manner, implying that the basic gates are instanced and linked together. These new functional blocks are tested in a similar manner as the gates by automated scripts that test the device. The source code and test scripts for these are available in Appendices A and B respectively.

The basic functional blocks designed provide desired functionalities such as storing bits and adding bits together. With various instances and implementations of these devices, bigger functional blocks that are essential to the design of the Booth8 multiplier can be made.

5.1 The Half-Adder

The first basic functional block is the half adder. A half adder is a simple adder that adds two inputs together and provides two single bit outputs. The first output is the carry out, while the second is the sum. To get the numerical value of a half adder's output, the carry out is treated as the most significant bit, while the sum is considered to be the least significant bit. Since the half adder has only two inputs available, it is used for only certain functional blocks such as incrementers. The half adder's behaviour is defined in Table 8.

Table 8: Half Adder Truth Table

<i>A</i>	<i>B</i>	<i>Carry Out</i>	<i>Sum</i>
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Through means of analysis, it can be determined that the governing equations for the half adder

are:

$$Carry_{out} = A \cdot B \quad (3)$$

and

$$Sum = A + B \quad (4)$$

When synthesized, the circuit shown in Figure 12 is produced. Simulating the half adder yields desired results as shown in Figure 13.

Figure 12: Half Adder Circuit

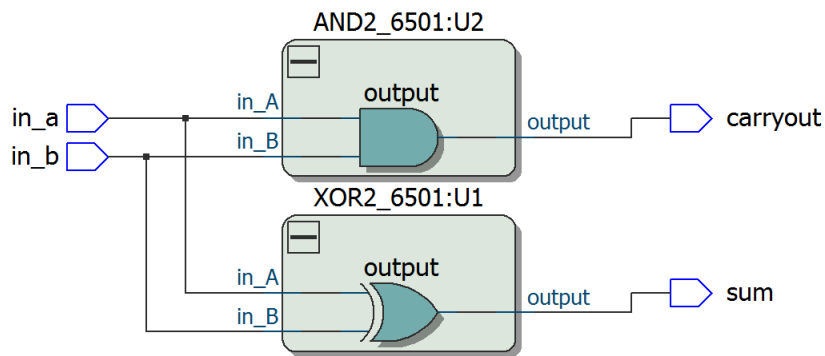
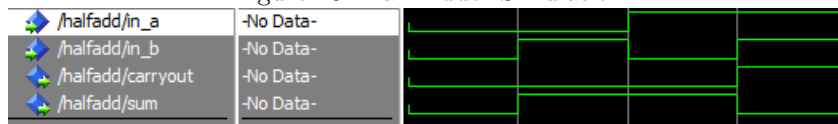


Figure 13: Half Adder Simulation



5.2 The Full Adder

The Full Adder performs the addition of three bits rather than the two bits of the half adder. Because of this characteristic, the Full Adder can be used for a wider variety of applications than a half adder but at a cost. Additional gates are required and a larger delay is present. With that being said, the Full Adder is capable of reaching an output of 3 in decimal versus the largest output of 2 for the Half Adder. The truth table for a Full Adder is shown in Table 9.

The equations for the Full Adder are defined as follows:

$$Sum = A \oplus B \oplus Carry_{in} \quad (5)$$

and

$$Carry_{out} = (A \cdot B) + (Carry_{in} \cdot (A \oplus B)) \quad (6)$$

The circuit for the Full Adder is shown in Figure 14. When simulated, the Full Adder produces the desired results as shown in Figure 15.

Table 9: Full Adder Truth Table

<i>A</i>	<i>B</i>	<i>Carry</i>	<i>Carry Out</i>	<i>Sum</i>
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figure 14: Full Adder Circuit

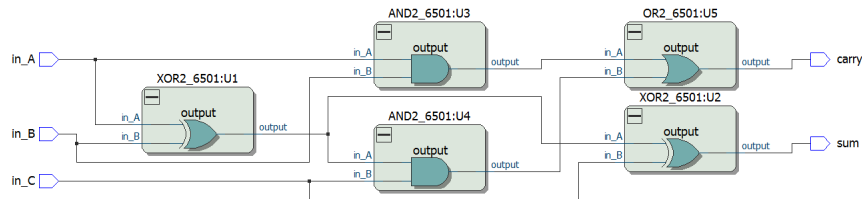
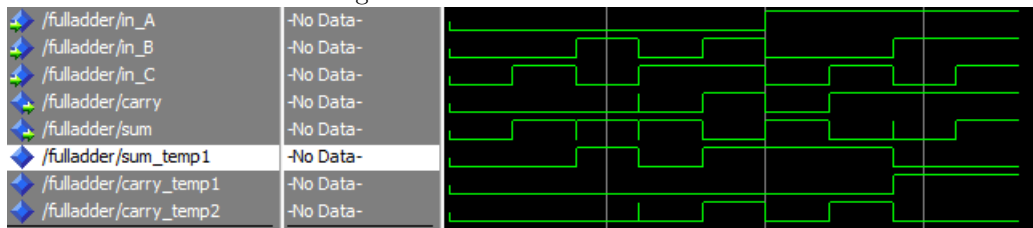


Figure 15: Full Adder Simulation



5.3 D Flip Flop

The next basic functional block that is needed for the multiplier is the D Flip Flop. A D Flip Flop is a functional block that stores one bit and can form the basis of a class of circuits known as *sequential circuits* that require a clock and memory. Registers of size *n* can be made from generating *n* number of D Flip Flops to store each bit in the nibble, word or block of data. The control signals of the register such as loads, output enables, clocks are resets are all tied to the same input so that each D Flip Flop can reset, load or update in tandem.

The requirements of the multiplier require that any register derived from these flip flops must have a reset, so an *active-low* reset is implemented. Registers derived from this D Flip Flop have certain signals inverted to comply with the project specifications.

This implementation of the D Flip Flop has a *positive edge* triggered clock and an *active low* reset. The behaviour of the D Flip Flop can be described in Table 10. Simulations yield the results shown in Figure 16.

Table 10: D Flip Flop Truth Table

<i>Reset</i>	<i>Data In</i>	<i>Clock</i>	Q_{next}	$\overline{Q_{next}}$
0	X	X	0	1
1	0	↑	0	1
1	1	↑	1	0

Figure 16: D Flip Flop Simulation

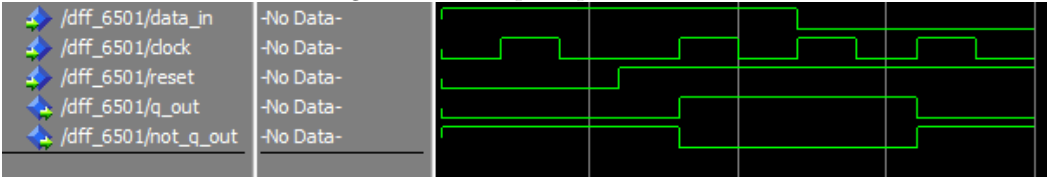
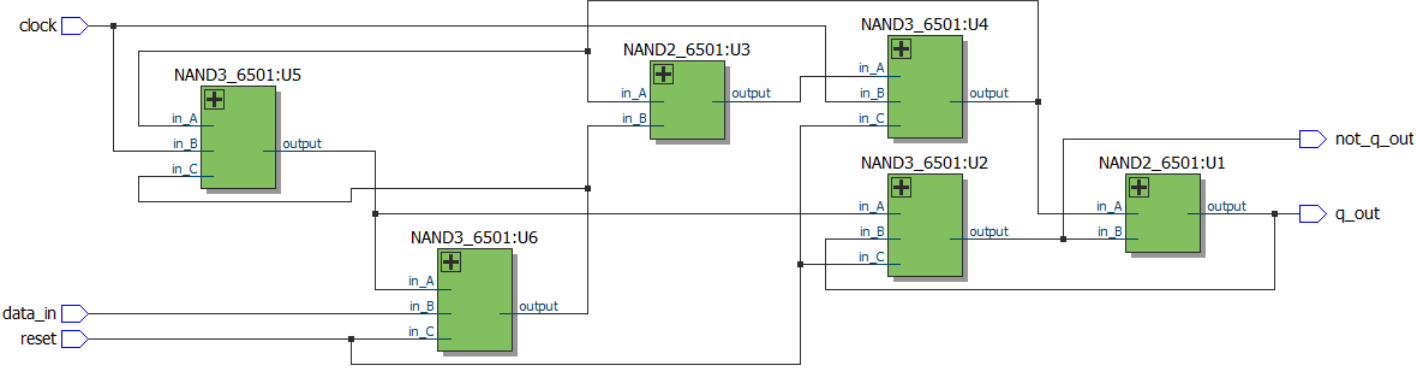


Figure 17: D Flip Flop Circuit



5.4 Two to One Multiplexer

A two to one multiplexer is a switch that has a select switch to control the input. The truth table is shown in Table 11.

Table 11: 2 To 1 Multiplexer

A	B	Select	Output
0	X	0	0
1	X	0	1
X	0	1	0
X	1	1	1

The equation for the multiplexer is

$$Y = (\overline{Sel} \cdot A) + (Sel \cdot B) \quad (7)$$

Figure 18: Two To One Multiplexer Circuit

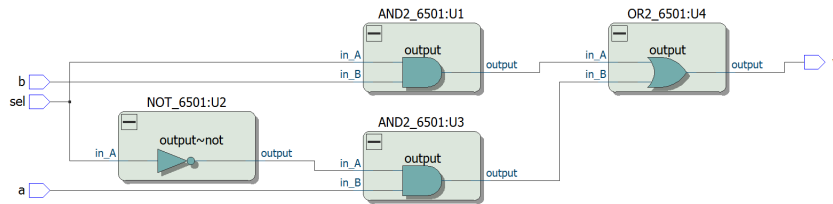
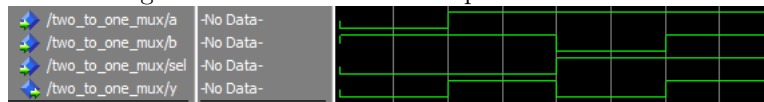


Figure 19: Two To One Multiplexer Simulation



6 Design of Higher Level Functional Blocks

In this section, the design of more complex functional blocks is discussed. Devices such as incrementers, registers and twos complementers are described. These functional blocks are built from the functional blocks described previously along with the logic gates also designed.

6.1 9 Bit Incrementer

The 9 Bit Incrementer is a functional block that increments the input by one. In order to optimize for area and power, only half adders are used. The design is a chain of half adders with the carry in of the first half adder set to a logic 1. The carry out of each half adder is fed to the next half adder along with the current bit. The synthesized layout is shown in Figure 20. This functional block is used to perform the operation of producing the two's complement of an input. Testing on the module was performed in ModelSim. Results can be seen in Figure 21.

Figure 20: 9 Bit Incrementer Circuit

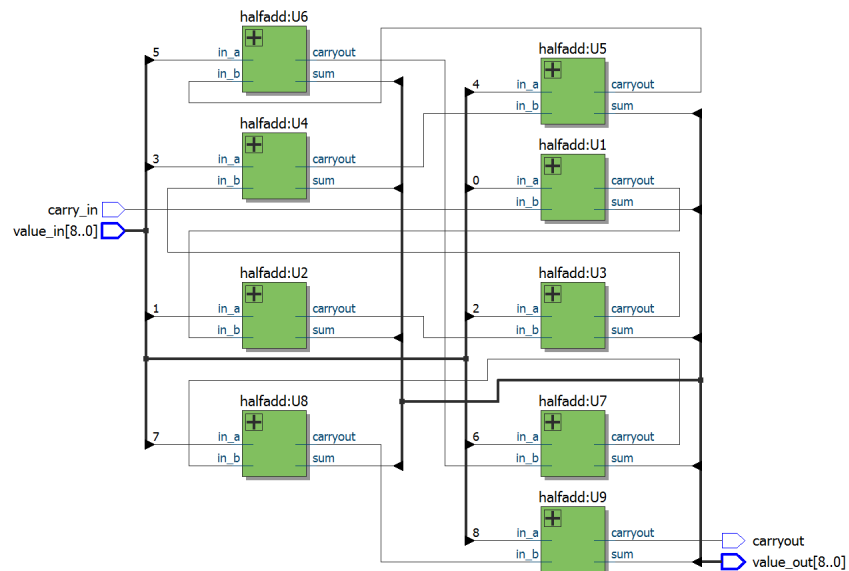


Figure 21: 9 Bit Incrementer Simulation

/increment_9bit/value_in	-No ...	110001111	101111110	111111111	000000000
/increment_9bit/carry_in	-No ...				
/increment_9bit/value_out	-No ...	110010000	101111110	000000000	000000001
/increment_9bit/carryout	-No ...				

6.2 Two's Complementer

The purpose of this circuit is to compute the two's complement of the input. The two's complement is defined as a signed negative number that is produced by complementing a positive number then adding one. This can be expressed in equation 8.

$$-Value_{in} = \overline{Value_{in}} + 1 \quad (8)$$

The range of two's complement numbers is from -2^{n-1} to $2^{n-1}-1$. Table 12 shows the values available for 3 bit strings. Given that the Booth multiplier takes 8 bit inputs, the smallest value accepted is -128 and the largest value accepted is 127. This functional block has an enable that is fed to an XOR gate for each bit of the input. It takes an 8 bit input and sign extends the value by one bit. This is done to ensure that the sign is still retained for later stages. The result is fed to parallel XOR gates which perform the complementing if enabled. The output of each XOR gate is fed into the 9 bit incrementer along with the enable signal as the initial carry in. The layout is shown in Figure 22. A simulation involving complementing three numbers is shown in Figure 23.

Table 12: 3 Bit Two's Complement Values

Binary	Decimal
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Figure 22: Two's Complementer Circuit

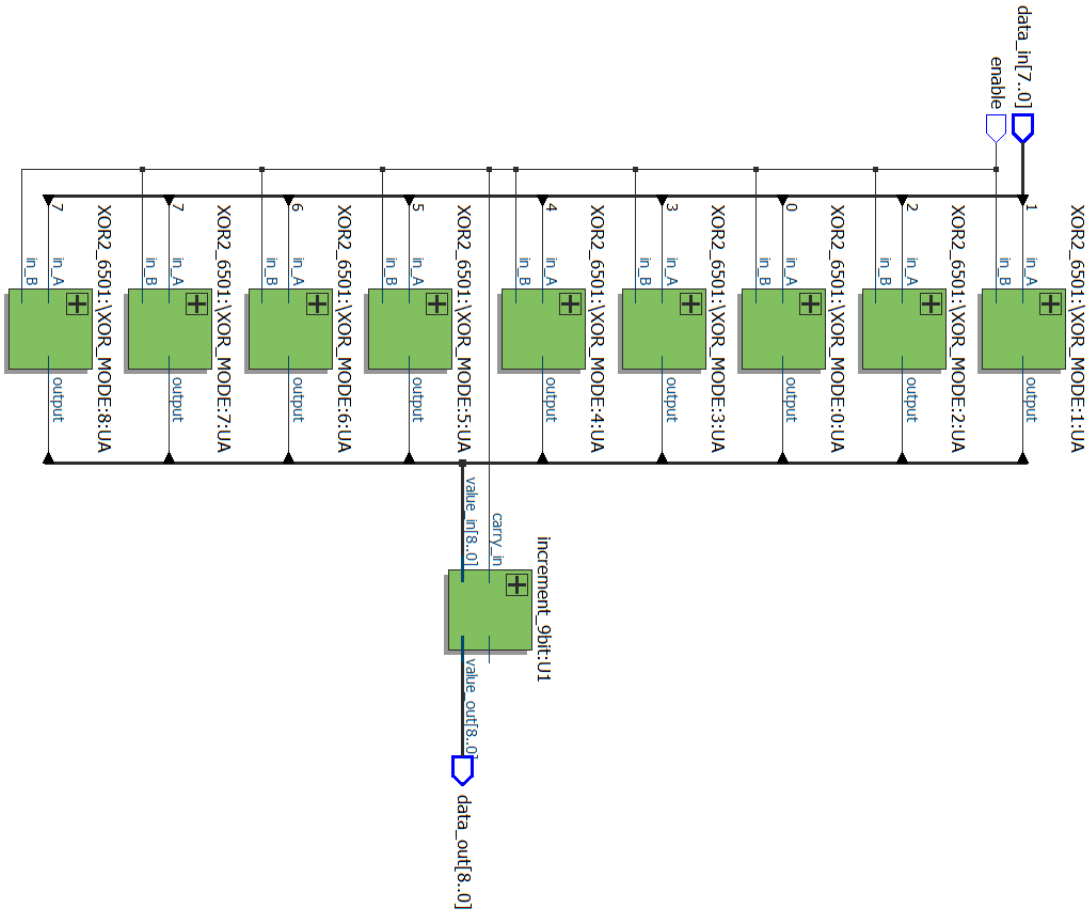
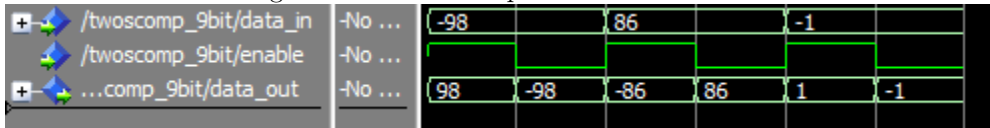


Figure 23: Two's Complementer Simulation



6.3 9 Bit Barrel Shifter

A barrel shifter is a combinational circuit that is able to perform different shifting operations depending on the values of the control signals. Barrel shifters tend to be derived from cascading multiplexers. In the case of the Booth multiplier, a 9 bit barrel shifter is needed to easily perform the multiplication operations needed. To multiply by two, a single shift to the left is needed. Table 13 describes the functionality of the barrel shifter.

Table 13: 9 Bit Barrel Shifter Functionality

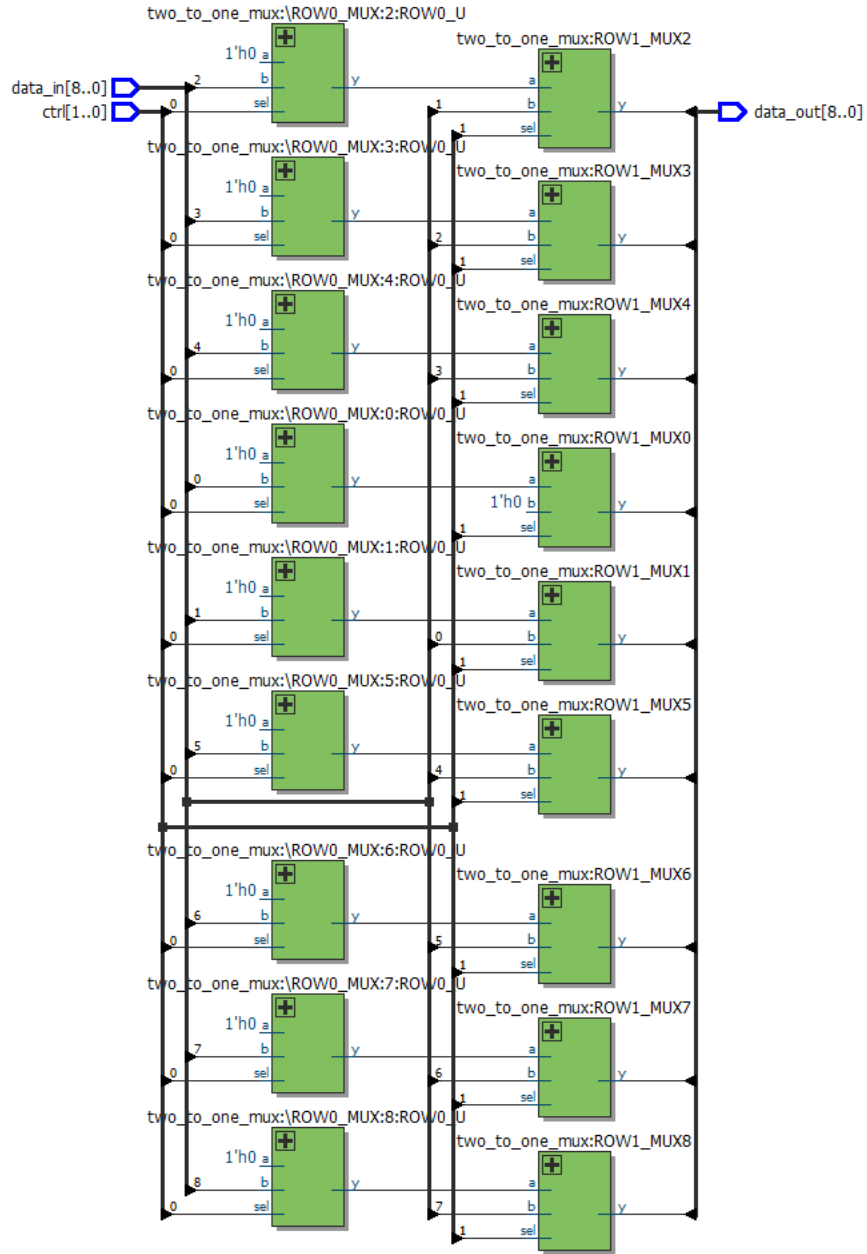
S_1	S_0	Operation
0	0	Input $\times 0$
0	1	Input $\times 1$
1	X	Input $\times 2$

Figure 24: 9 Bit Barrel Shifter Simulation

/barrel_shift_9bit/data_in	-No ...	453	218	1	
/barrel_shift_9bit/ctrl	-No ...	0	1	2	3
/barrel_shift_9bit/data_out	-No ...	0	453	394	0
			218	436	0
				1	2

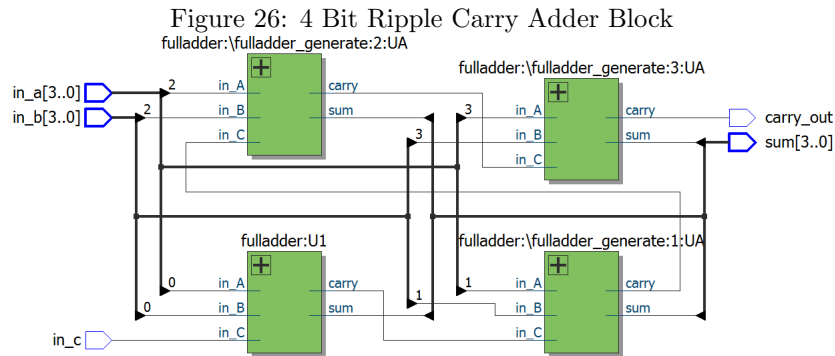
The circuit is shown in Figure 25. A simulation was performed to validate the circuit. Please note that for ease of reading, the Unsigned ModelSim radix is used. The results are shown in Figure 24.

Figure 25: 9 Bit Barrel Shifter



6.4 Ripple Carry Adder Blocks

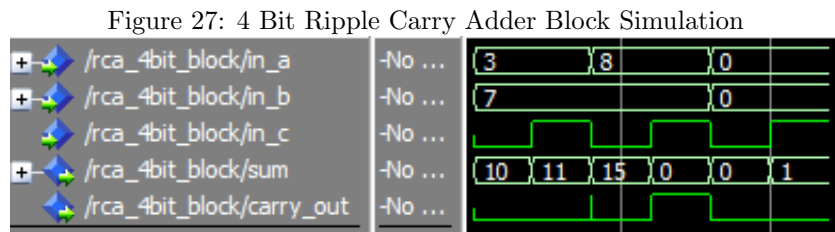
In order to build larger carry select adders, small ripple carry adder blocks are needed. For this multiplier, a 4 bit version is built. With that said, an n-bit Ripple Carry Adder is n full adders tied together with the goal of adding two n bit numbers together. The carry out of the first stage is connected to the carry in of the next. This is repeated n number of times to achieve the addition of two numbers. The four bit Ripple Carry Adder block is shown in Figure 26.



The simulation runs through three cases:

- A general purpose test addition
- Overflow test
- Zeros test

These results can be seen in Figure 27.



7 Building the Adders

7.1 Introduction to the Carry Select Adder

In a multiplier, the use of adders is critical. For this multiplier, the main metric of concern is speed which dictates the selection of adders. For the sake of code reuse and ease of implementation, carry select adders are used.

A *carry select adder* (CSA) is a two number adder that is split into multiple stages. Aside from the first stage, each stage operates has a pair of Ripple Carry Adder (RCA) blocks performing the addition of a chunk of the two operands. One RCA in the pair has a carry in of 0, while the other has a carry in of 1. A set of multiplexers will select the correct sum and carry out from the two RCA blocks depending on the carry out of the previous stage.

Ideally, the CSA is divided into stages that each process an equal number of bits. By doing so, the delay can be decreased substantially. As an example, let us assume an 8 bit Carry Select Adder with 4 bit RCA stages. Since the addition occurs in parallel, we can then claim that the simplified delay of the CSA is the following:

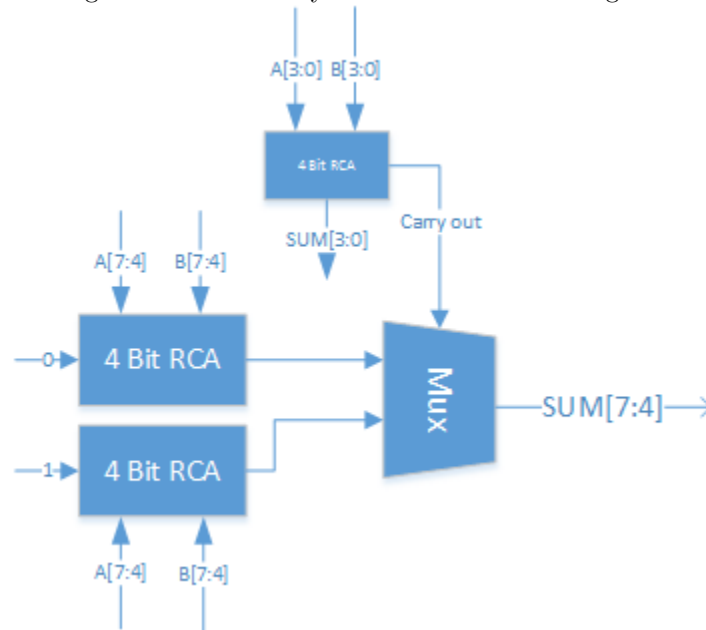
$$\tau_{csa} = \tau_{mux} + 4 \times \tau_{fa} \quad (9)$$

If we were to use an 8 bit Ripple Carry Adder, the delay would be

$$\tau_{rca} = 8 \times \tau_{fa} \quad (10)$$

It can be assumed that the delay of a multiplexer is less than a full adder. As shown, the delay savings are substantial but do come at a cost. The power consumption and area required to implement a CSA is rather demanding. A block diagram of the 8 bit CSA is shown in Figure 28.

Figure 28: 8 Bit Carry Select Adder Block Diagram



7.2 Carry Select Adder Blocks

In this multiplier, two Carry Select Adder blocks are used: a 2 bit and a 4 bit block. As a general rule, it is frowned upon to produce RCAs of greater lengths than 4 bits due to propagation delay. For the sake of brevity, only the 4 bit block will be discussed. The source code and testing script for the 2 bit variant is available in the Appendices. An n bit CSA block is a combinational circuit composed two n bit RCAs and n+1 multiplexers. The implementation of a 4 bit CSA Block is shown in Figure 29 and the simulation in Figure 30.

Figure 29: 4 Bit CSA Block Circuit

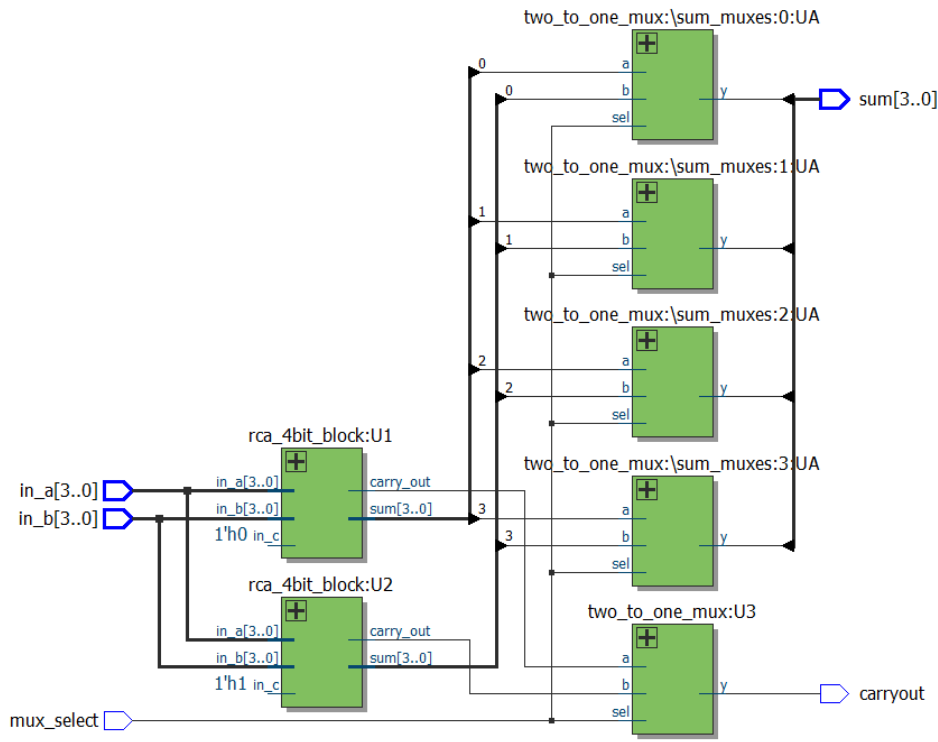


Figure 30: 4 Bit CSA Block Simulation

/csa_4bit_block/in_a	-No ...	7		8		0	
/csa_4bit_block/in_b	-No ...	7				0	
...4bit_block/mux_select	-No ...						
/csa_4bit_block/sum	-No ...	14	15	15	0	0	1
/csa_4bit_block/carryout	-No ...						

7.3 Carry Select Adders

The Booth multiplier uses three Carry Select Adders with differing sizes. The CSAs come in 10 bit, 12 bit and 14 bit variants. These adders are purpose-built so that they perform the exact amount of additions needed to compute the partial products for the multiplication operation.

The 10 bit version is built with two 4 bit CSA blocks and one 2 bit CSA. Since the 2 bit CSA block takes less time to complete than the 4 bit one, the only delay to consider is the additional multiplexer. The simplified delay can be expressed as:

$$\tau_{csa} = 2 \times \tau_{mux} + 4 \times \tau_{fa} \quad (11)$$

The 12 bit version is built from three 4 bit CSA, which allows for a simplified delay of

$$\tau_{csa} = 3 \times \tau_{mux} + 4 \times \tau_{fa} \quad (12)$$

And lastly, the 14 bit version is built from three four bit CSA block and one 2 bit CSA block. The simplified delay equation for this version is:

$$\tau_{csa} = 4 \times \tau_{mux} + 4 \times \tau_{fa} \quad (13)$$

For simulation purposes, the 10 bit CSA is shown in Figure 32 and a simulation is shown in Figure 31.

Figure 31: 10 Bit CSA Simulation





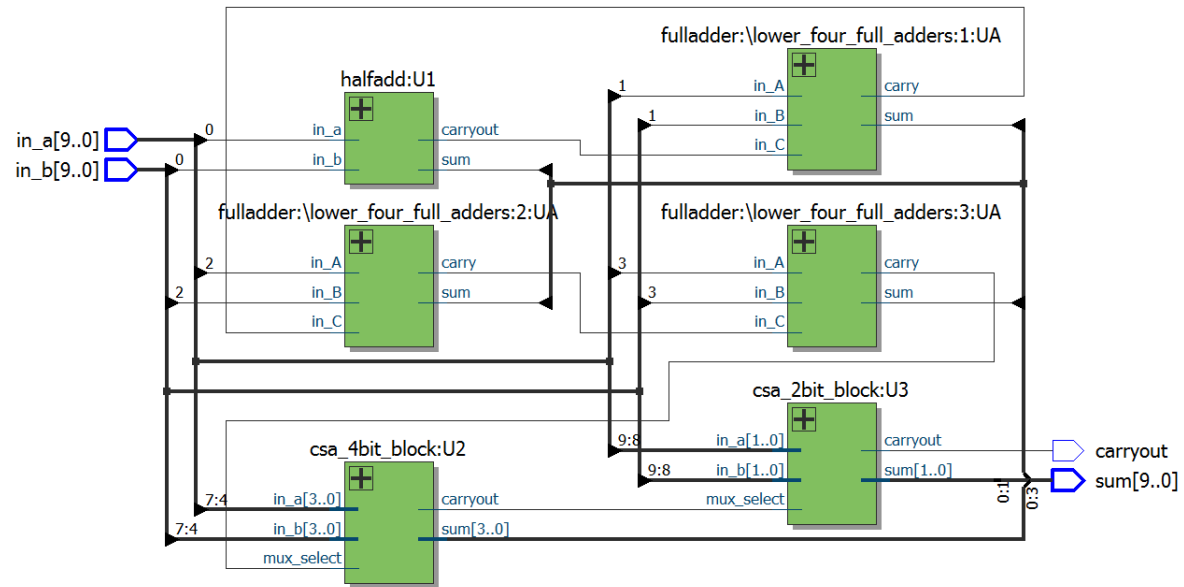
 /csa_10bit/in_a	-No ...	879	476	512	1012
 /csa_10bit/in_b	-No ...	46	198	511	484
 /csa_10bit/carryout	-No ...				
 /csa_10bit/sum	-No ...	925	674	1023	472

Figure 32: 10 Bit CSA



8 Booth Units and the Booth Adder

This section details the implementation of logical blocks that perform the operations described in Table 1 to generate partial products, along with a purpose-built adder to add the four partial products.

8.1 Booth Decoder

The functional unit called the Booth Decoder simply provides appropriate control signals based on the chunk of the multiplier passed to it. Its outputs are fed to the control input of the barrel shifter. The truth table of the Booth Decoder is the following:

Table 14: Booth Decoder Truth Table

D_2	D_1	D_0	$CTRL_1$	$CTRL_0$	<i>Operation</i>
0	0	0	0	0	Input \times 0
0	0	1	0	1	Input \times 1
0	1	0	0	1	Input \times 1
0	1	1	1	0	Input \times 2
1	0	0	1	0	Input \times -2
1	0	1	0	1	Input \times -1
1	1	0	0	1	Input \times -1
1	1	1	0	0	Input \times 0

The Boolean equations for this functional block are:

$$CTRL_1 = (D_2 \cdot \overline{D_1} \cdot \overline{D_0}) + (\overline{D_2} \cdot D_1 \cdot D_0) \quad (14)$$

and

$$CTRL_0 = \overline{D_1} \cdot D_0 + D_1 \cdot \overline{D_0} \quad (15)$$

The circuit implemented is shown in Figure 34. When simulated and tested, the results are shown in Figure 33.

Figure 33: Booth Decoder Simulation

/booth_decoder/data_in	-No Data-	000	001	010	011	100	101	110	111
/booth_decoder/ctrl	-No Data-	00	01		10	10	01		00

Figure 34: Booth Decoder Circuit

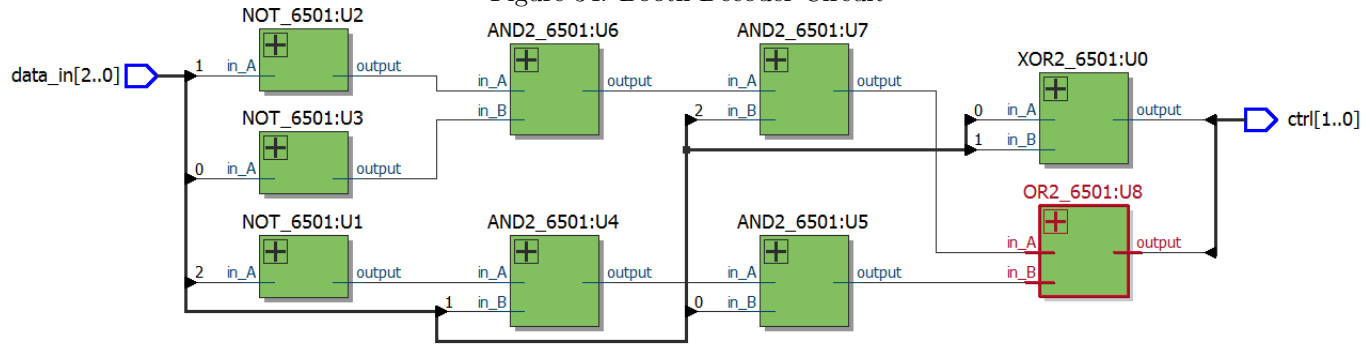
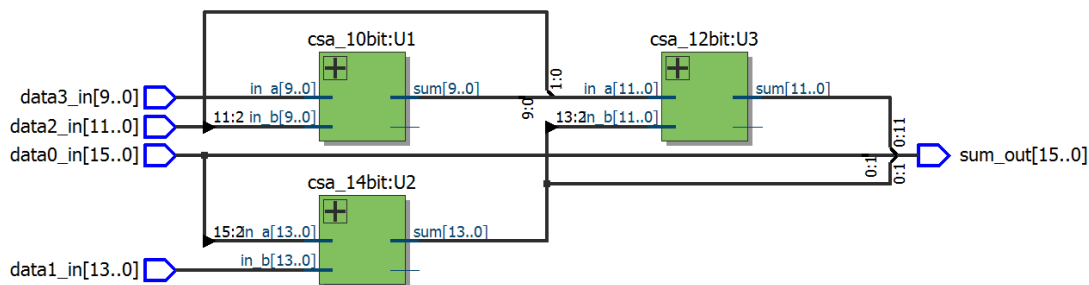


Figure 37: Booth Adder Circuit



To aid in testing, a sample testbench is made. This testbench includes the Booth Units as well and fundamentally is the combinational logic needed for the multiplier. This module is known as *Booth Adder, Device Under Test* or *ba-dut*. A set of multiplication operations are performed and the results are checked to see if the device operates correctly. A wide variety of operations are performed to test the carry of adders, check the behaviour when an operand is zero and lastly, general random multiplication. The script used to drive the testbench is available in the Appendix. The list of multiplications performed are:

- $127 \times 0 = 0$
- $127 \times 127 = 16129$
- $-128 \times 127 = -16256$
- $-56 \times -91 = 5096$
- $-1 \times -1 = 1$
- $-2 \times -1 = 2$
- $-24 \times -12 = 288$
- $107 \times -96 = 10272$
- $-3 \times 127 = -381$

The schematic is shown in Figure 38 and the simulation results are found in Figure 39.

9 Non-Pipelined Implementation

9.1 Introduction

In this section, the non-pipelined implementation of the multiplier is detailed. As detailed in the first chapter, the Booth multiplier functions with a *clock* and an active low *LOAD* signal to load the operands into registers. The *CLEAR* signal is active high and clears the registers.

In regards to outputs, the result is placed on the *Z Port*. Additionally, an *END* flag is present. The *END* flag is used to denote if the result is valid. It is asserted when a multiplication has finished on the positive edge of the clock.

Figure 38: Booth Adder, Device Under Test Circuit

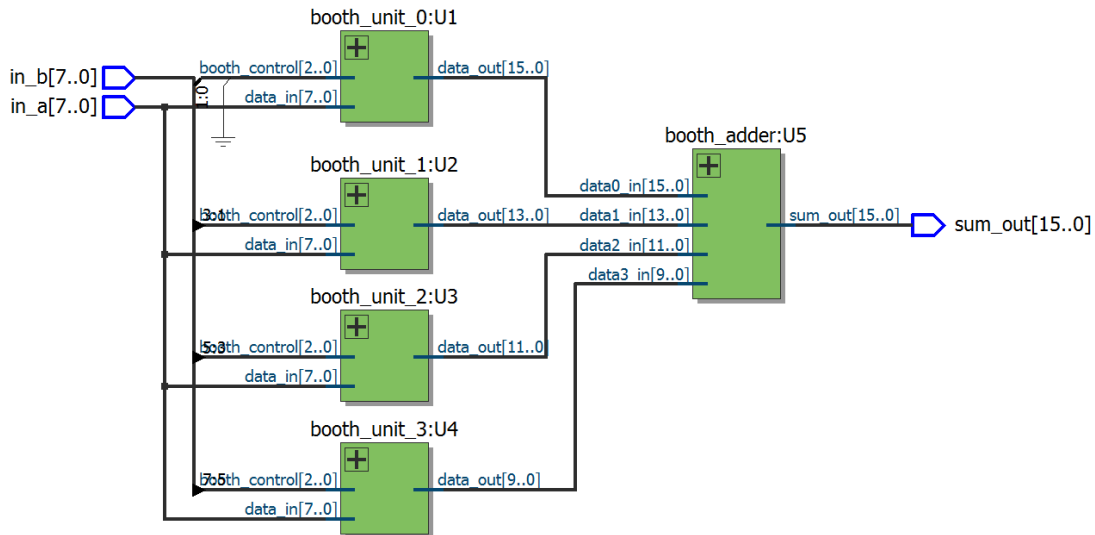


Figure 39: Booth Adder, Device Under Test Simulation

/ba_dut/in_a	-No ...	127		-128		-56	-1	-2	-24	107	-3
/ba_dut/in_b	-No ...	0	127		-1	-91	-1		-12	96	127
/ba_dut/sum_out	-No ...	0	16129	-16256	128	5096	1	2	288	10272	-381

9.2 The Operand Register

Before the multiplier can be built, the operand register unit must be detailed first. It is composed of the D Flip Flops detailed in Section 5.3. The operand register has a capacity of 1 byte or 8 bits. There are also two inverters present within the register. The first register is to invert the clock so that the register may function on the negative edge of the *LOAD* signal. The other inverter is present for the asynchronous reset. The project requirements dictate that the asynchronous reset must be active high. By using the inverter, this requirement is met.

In the simulation, three tests are performed:

- A simple data load of 0b10101010 or 170 in unsigned decimal with no reset
- Asynchronous reset
- Clocking test to see if the register will update without the clock

The results of these tests are shown in Figure 40. The implementation is shown in Figure 41.

Figure 40: 8 Bit Operand Register Test

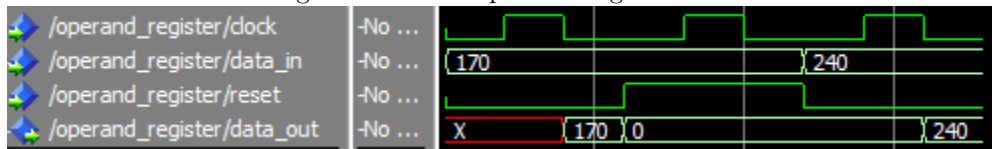
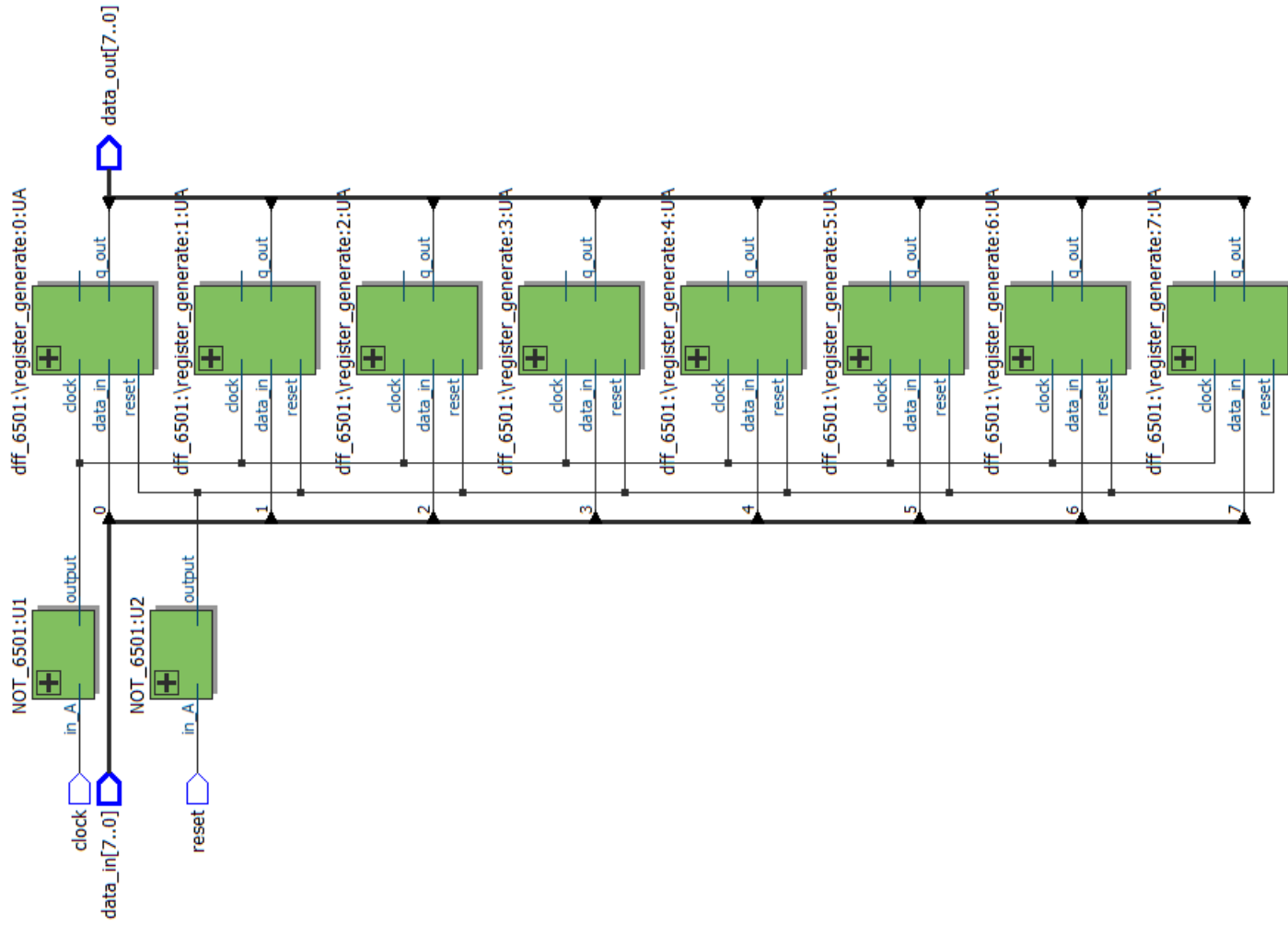


Figure 41: 8 Bit Operand Register Circuit

35



9.3 The Booth8 Multiplier

At this point, the multiplier is finally implemented. An inversion of the *LOAD* signal is fed through a D Flip Flop to dictate if the result produced is valid. If *LOAD* is high, it is likely that the operands and the result are likely going to change very soon. Additionally, the D Flip Flop is used so that if the reset is active, any device that reads the output from the multiplier will know that the result obtained is not valid due to resetting. Two inverters are present to invert the load and clear signals rather than have the two inverters implemented on each register. A testing script was produced to validate the circuit. It performs the following tests:

- Asynchronous Reset
- $127 \times 127 = 16129$
- Asynchronous Reset
- $-128 \times -128 = 16384$
- $-65 \times 83 = 16129$
- $107 \times 96 = 10272$
- $-3 \times 127 = -381$
- Valid Data/END Flag

The results are shown in Figure 42 and the implementation in Figure 43.

Figure 42: Booth8 Multiplier Test

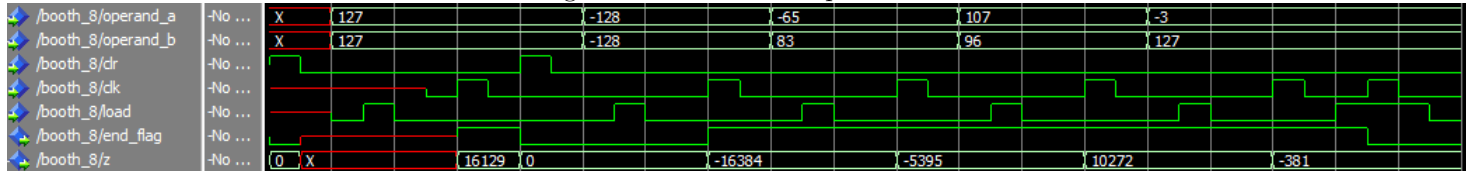
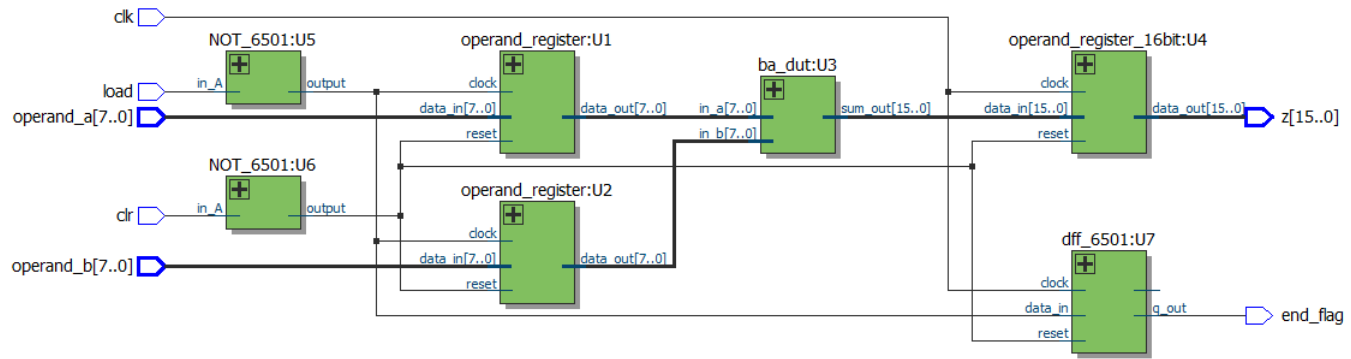


Figure 43: Booth8 Multiplier Implementation



10 Pipelined Implementation

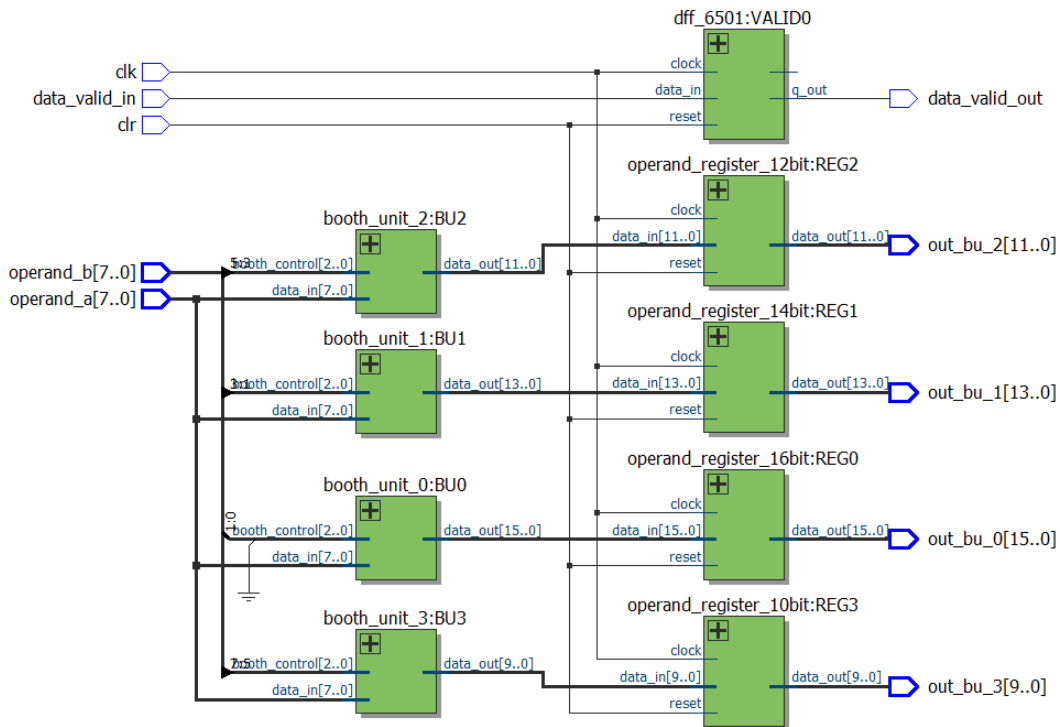
10.1 Introduction

This section details the pipelined implementation of the Booth8 multiplier. Given the implementation, there are only two pipeline stages. The operation of the pipelined variant requires two clock cycles to yield a result. There are two operand registers to store the multiplicand and multiplier. These are controlled by the negative edge of the *LOAD* signal. When the *clr* signal is asserted, all registers are cleared including the valid data registers. This is done so that the *END* flag is set low and it can be seen that the results on the Z port are not valid.

10.2 Pipeline Stage 0

The first pipeline stage contains the Booth Units and the necessary registers to contain the results of the Booth Units and the data valid flag. These registers are controlled by the clock, despite the fact that the *LOAD* flag controls the change of operators. It takes the output of the two operand registers and feeds them into the pipeline stage, along with the data valid signal to ensure that the data going through is valid. It outputs the results of Booth Units 0 to 3 and passes the data valid flag along the pipeline. The implementation is shown in Figure 44.

Figure 44: Booth8 Pipeline Stage 0 Circuit

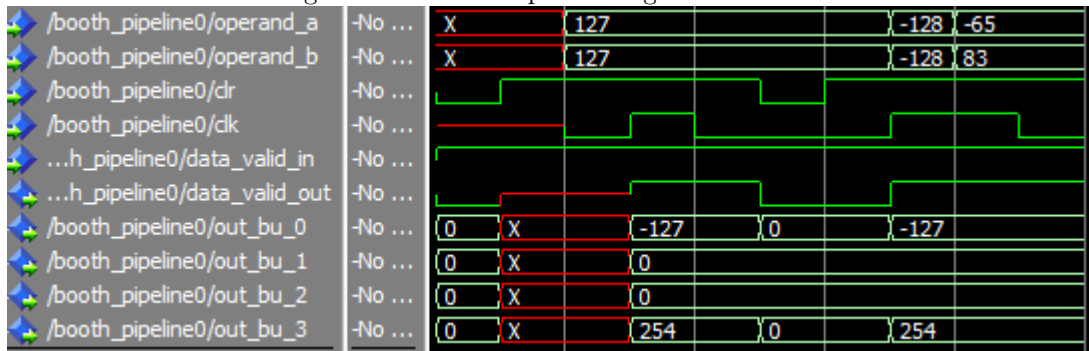


When testing, the following test was performed:

- Initial Reset
- Load operands 0b11111111, 0b11111111
- Do asynchronous clear
- Load operands 0b10111111, 0b01010011

The simulation provided the desired results as seen in Figure 45.

Figure 45: Booth8 Pipeline Stage 0 Simulation



10.3 Pipeline Stage 1

This is the final stage of the pipeline. It contains the Booth Adder and two registers. The stage simply takes the results of the Booth Units and adds them as demonstrated previously. One register is present to store the final result, while the other is to pass on the data valid flag to the end of the pipeline. The implementation is shown in Figure 47. To validate the module, the following tests were performed:

- Initial Reset
- Load partial products of $127 \times 0 = 0$
- Load partial products of $127 \times 127 = 16129$ then perform an asynchronous reset and try to rewrite the value
- Load partial products of $-128 \times -128 = 16384$
- $-65 \times 83 = -5395$

The results of the simulation are shown in Figure 46 and the implementation in Figure 47.

Figure 46: Booth8 Pipeline Stage 1 Simulation

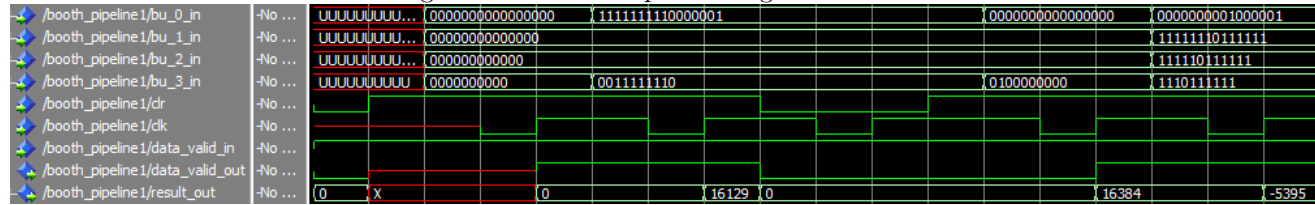
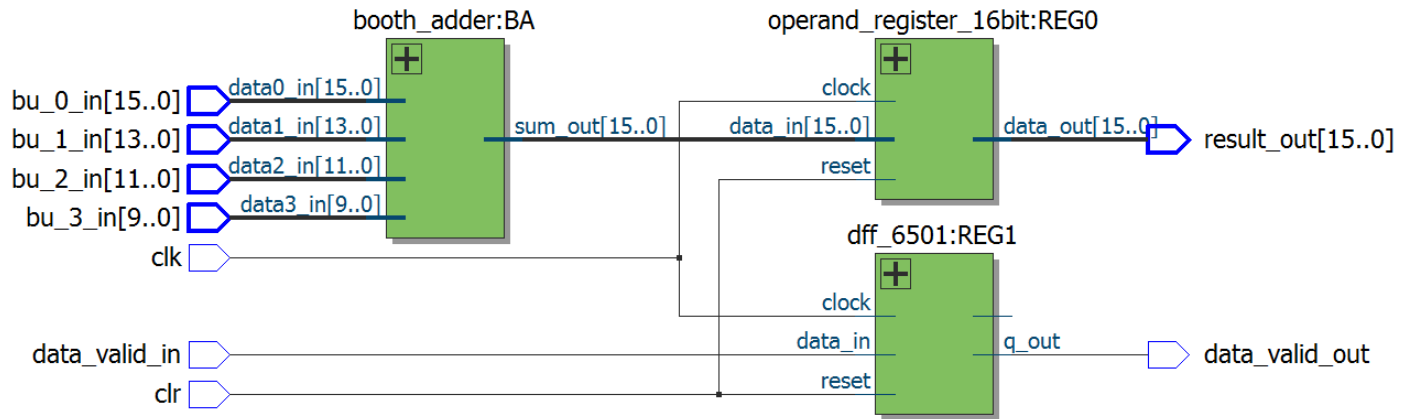


Figure 47: Booth8 Pipeline Stage 1 Circuit



10.4 Top Level Implementation

This section details the complete implementation of the pipelined multiplier. Within the module, the two pipeline stages are instanced along with two inverters for control signals and the two operand registers. A similar testing scheme used to validate the non-pipelined version is used to verify this module. The tests have been modified to account for pipeline flushing and propagation. Simulation results are shown in Figure 48 and the implementation in Figure 49. The testing scheme is as follows:

- Initial Reset
- $127 \times 127 = 16129$
- Pipeline Flush
- $-128 \times -128 = 16384$
- $-65 \times 83 = -5395$
- Push remaining items through pipeline and verify valid flag

$$\frac{1}{\tau_{delay}} = Frequency \quad (16)$$

11 Synthesis and Analysis of the Multiplier

11.1 Introduction

In this section, the Booth8 multiplier is synthesized for implementation on an field-programmable grid array (FPGA). By synthesizing, the source code in VHDL is mapped to physical constructs on the FPGA. For this project, the FPGA used is a Xilinx Virtex-II Pro.

A simplified FPGA has numerous "slices", which are discrete units that contain a look-up table (LUT) and some number of flip flops. A look-up table has n inputs and returns one output. When synthesizing HDL code, the synthesizer will use these look-up tables to implement combinational logic and possibly infer the flip flops in the LUT. Once laid out, the synthesis tool will wire and link the slices to produce the circuits described in the HDL. A timing analysis can also be performed to determine if a circuit can operate at a given speed. The features of FPGA design toolkits is beyond the scope of this report, for more information please refer to the ISE Manual.

The tools used for this section are Mentor Graphics PrecisionRTL to generate and synthesize the circuits. PrecisionRTL also details the area usage and can perform timing analysis if needed. When performing synthesis, PrecisionRTL will produce a netlist called an .EDIF file that will dictate to the programming file generator (in this case, ISE) how to lay out and wire the FPGA internally.

When determining the maximum frequency, the largest delay is taken and its reciprocal is the maximum frequency. This can be computed by the following equation:

11.2 Results for Non-Pipelined Version

When synthesizing the non-pipelined version, the total number of slices used was 163 or 1.19% of all available logic slices. The full report from PrecisionRTL regarding the area and resource used is shown below in Appendix C.

The timing analysis was performed with ISE. There were no timing constraints used so the default settings were simply applied. These results are just for discussion purposes and are not to serve as any basis for comparison. The longest delay found was between *load* and z_{14} at 17.905 ns. By using the equation presented above, it can be seen that the maximum frequency is **55.85 MHz**.

11.3 Results for the Pipelined Version

When synthesizing the pipelined version, the total number of slices used was found to be 269 or 1.96% of all available slices. Similarly to the non-pipelined version, the full report from PrecisionRTL is available in Appendix C.

Similarly to the non-pipelined version, the timing analysis was performed with ISE. No timing constraints were used and the frequency of the FPGA was set to a maximum of 100 MHz as recommended. The longest delay was found to be 20.551 ns. The frequency was found to be **48.66 MHz**.

The results obtained from synthesis are not entirely acceptable but do make sense. Regarding area, the resource usage reported by PrecisionRTL does make sense. A pipelined variant should

consume more resources due to the additional registers and control logic. An increase in resource usage by 0.77% is acceptable if there is a return on throughput.

The problematic results are the ones obtained by the timing analysis. Despite no user determined timing constraints, the timing results are not logical. By pipelining, the throughput goes up because at each cycle, there is less signal propagation. In the case of the multiplier, the pipeline technique serves to reduce the amount of propagation through combinational circuitry by splitting the path into stages such as computing the partial products then feeding the obtained partial products into the adder module during the next clock cycle. According to the results obtained, the non-pipelined version is approximately 14.7% faster which goes against the technique just mentioned. Fortunately, there is a logical explanation for this.

Traditionally, systems designed in VHDL are not designed in a structural manner but rather by behavioural descriptions. As mentioned in the project requirements, the multiplier is built in a structural manner which renders it susceptible to peculiar issues in analysis, implementation and possibly simulation as well. Regarding timing issues, the synthesizer is unable to infer flip flops when a design is structural. When executing the analysis, the tool simply sees extended combinational logic with combinational loops for modules that are in fact flip flops. Additionally, the synthesis tool simply will display a warning for combinational loops because there is a possibility of unknown or indeterminate states. Since these modules are seen as combinational units, they are considered additional paths for delay. This becomes especially apparent with the increased delay for the pipelined version of the multiplier given the additional registers used. Fortunately, there is a way to overcome this issue.

11.4 Alternate Method for Pipelined Version

The reasoning behind pipelining is to divide the logic into stages that are separated by registers. By doing so, the delay of the overall circuit is divided into discrete chunks. The clock frequency can then be derived to account for the slowest stage. A stage is defined as the path from one pipeline register to the next. In the case of this Booth multiplier, there are two stages. The first stage is from the operand registers, leading into the Booth units to produce the partial products. The second stage is from the pipeline registers to the Booth Adder, leading into the result register.

To test the pipeline stage delays, the registers and combinational logic of each are separated into chunks and evaluated separately. An evaluation consists of being synthesized through PrecisionRTL and analyzed for timing in ISE. The highest delay time produced by ISE is taken as the delay used for the chunk. The pipeline stage delay can then be computed as the following:

$$\tau_{stage} = \tau_{CL} + \tau_{Registers} \quad (17)$$

11.4.1 Pipeline Stage 0

For Pipeline Stage 0, the largest delay for the combinational logic was found to be 8.244 ns. For the registers, the largest delay found was 10.782 ns. The result total delay was found to be 19.026 ns. With this value, the frequency found was **52.56 MHz**. The timing results can be found in Appendix C

11.4.2 Pipeline Stage 1

The combinational logic for the second pipeline stage had a maximum delay of 11.129 ns. The registers had a maximum delay of 7.074 ns. The total delay through the stage was found to be

18.133 ns. The frequency found is **54.94 MHz**.

11.5 Conclusions and Improvements

The maximum operating frequency for the pipelined version was found to be 52.56 MHz by using alternate method. This is slower than the non-pipelined variant but is understandable and a better result than the initial 48.66 MHz. The multiplier does not pipeline well due to the fact that it has large blocks of combinational logic and numerous registers to fill at each stage. With that said, the pipelined version has a maximum frequency within 10% of the non-pipelined version. The multiplier would require a redesign to pipeline better and because of this, should be used in a non-pipelined version.

Appendices

A VHDL Source Code

In this section, the VHDL source code for the 8 bit by 8 bit Booth Multiplier is available. This chapter is split into subsections to better organize the source code listings.

A.1 Logic Gates

A.1.1 AND2

```
-- AND2 Gate
-- Purpose: To implement a two input AND Gate

-- Library declarations
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity AND2_6501 is
    port
    (
        in_A, in_B : in std_logic;
        output : out std_logic
    );
end AND2_6501;

-- Begin architecture
architecture AND2_6501_IMP of AND2_6501 is
begin
    -- Do AND function
    output <= in_A and in_B;
end AND2_6501_IMP;
```

A.1.2 NAND2

```
-- NAND Gate
-- Written by Richard Fenster
```

```

-- Library declarations
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity NAND2_6501 is
    port
    (
        in_A, in_B : in std_logic;
        output : out std_logic
    );
end NAND2_6501;

-- Begin architecture
architecture NAND2_6501_IMP of NAND2_6501 is
begin
    -- Do NAND function
    output <= in_A nand in_B;
end NAND2_6501_IMP;

```

A.1.3 NAND3

```

-- NAND3 Gate
-- Written by Richard Fenster

-- Library declarations
library ieee;
use ieee.std_logic_1164.all;

-- Begin Entity
entity NAND3_6501 is
    port
    (
        in_A, in_B, in_C : in std_logic;
        output : out std_logic
    );
end NAND3_6501;
-- Begin arch

architecture NAND3_6501_ARCH of NAND3_6501 is
begin
    -- Do NOT (A AND B AND C) which is equivalent to A NAND B NAND C
    output <= NOT (in_A and (in_B and in_C));
end NAND3_6501_ARCH;

```

A.1.4 NOT

```

-- NOT Gate
-- Written by Richard Fenster

-- Library declarations
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity NOT_6501 is

```

```

        port
        (
            in_A : in std_logic;
            output : out std_logic
        );
end NOT_6501;

-- Begin architecture
architecture NOT_6501_IMP of NOT_6501 is
begin
    -- Do inversion
    output <= NOT(in_A);
end NOT_6501_IMP;

```

A.1.5 OR2

```

-- OR2 Gate
-- Purpose: To implement a two input OR Gate

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity OR2_6501 is
    port
    (
        in_A, in_B : in std_logic;
        output : out std_logic
    );
end OR2_6501;

-- Begin architecture
architecture OR2_6501_IMP of OR2_6501 is
begin
    -- Do A or B
    output <= in_A or in_B;
end OR2_6501_IMP;

```

A.1.6 XOR2

```

-- XOR2 Gate
-- Purpose: To implement a two input XOR gate

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity XOR2_6501 is
    port
    (
        in_A, in_B : in std_logic;
        output : out std_logic
    );
end XOR2_6501;

```

```

-- Begin architecture
architecture XOR2_6501_IMP of XOR2_6501 is
begin
    -- Do A xor B
    output <= in_A xor in_B;
end XOR2_6501_IMP;

```

A.2 Small Functional Blocks

A.2.1 D Flip Flop with Active Low Reset and Positive Edge Clock

```

-- Positive Edge Triggered Flip Flop
-- Purpose: Store a bit, positive edge triggered and active low reset

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity dff_6501 is
    port
    (
        data_in : in std_logic;
        clock : in std_logic;
        reset : in std_logic;
        not_q_out : out std_logic;
        q_out : out std_logic
    );
end dff_6501;

-- Begin architecture
architecture rtl of dff_6501 is

    -- Define components
    component NAND2_6501
        port
        (
            in_A, in_B : in std_logic;
            output : out std_logic
        );
    end component;

    component NAND3_6501
        port
        (
            in_A, in_B, in_C : in std_logic;
            output : out std_logic
        );
    end component;

    -- Temporary signals
    signal s, r, not_q, q, t1, t2 : std_logic;
begin
    -- Build NANDs with feedback
    U1: NAND2_6501 port map (s, not_q, q);
    U2: NAND3_6501 port map (r, q, reset, not_q);
    U3: NAND2_6501 port map (s, t2, t1);
    U4: NAND3_6501 port map (t1, clock, reset, s);
    U5: NAND3_6501 port map (s, clock, t2, r);

```



```

    U6: NAND3_6501 port map (r, data_in, reset, t2);
    -- Cast outputs
    q_out <= q;
    not_q_out <= not_q;

end rtl;

```

A.2.2 Full Adder

```

-- Full Adder for COEN6501
-- Purpose: Does addition of 3 bits

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Entity definition
entity fulladder is
    port
    (
        in_A, in_B, in_C : in std_logic;
        carry, sum : out std_logic
    );
end fulladder;

-- Architecture definition
architecture fulladder_arch of fulladder is
    -- Define signals
    signal sum_temp1, carry_temp1, carry_temp2, carry_temp3 : std_logic;
    -- Declaration of XOR2 gate
    component XOR2_6501
        port
        (
            in_A, in_B : in std_logic;
            output : out std_logic
        );
    end component;
    -- Declaration of OR2 gate
    component OR2_6501
        port
        (
            in_A, in_B : in std_logic;
            output : out std_logic
        );
    end component;
    -- Declaration of AND2 gate
    component AND2_6501
        port
        (
            in_A, in_B : in std_logic;
            output : out std_logic
        );
    end component;
begin
    -- Create XOR gates for sum
    U1: XOR2_6501 port map(in_A, in_B, sum_temp1);
    U2: XOR2_6501 port map(sum_temp1, in_C, sum);

```

```

        -- Create AND Gates for carry generation
        U3: AND2_6501 port map (in_A, in_B, carry_temp1);
        U4: AND2_6501 port map (sum_temp1, in_C, carry_temp2);
        -- Create OR Gate for carry out generation
        U5: OR2_6501 port map (carry_temp1, carry_temp2, carry);
end fulladder_arch;

```

A.2.3 Half Adder

```

-- Half Adder for COEN6501
-- Purpose: Does addition of 2 bits

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin Entity
entity halfadd is
    port
    (
        in_a : in std_logic;
        in_b : in std_logic;
        carryout : out std_logic;
        sum : out std_logic
    );
end halfadd;

-- Begin architecture
architecture halfadd_arch of halfadd is

    -- Define components
    component XOR2_6501
        port
        (
            in_a : in std_logic;
            in_b : in std_logic;
            output : out std_logic
        );
    end component;

    component AND2_6501
        port
        (
            in_a : in std_logic;
            in_b : in std_logic;
            output : out std_logic
        );
    end component;
begin
    -- Implement circuit
    U1: XOR2_6501 port map (in_a, in_b, sum);
    U2: AND2_6501 port map (in_a, in_b, carryout);
end halfadd_arch;

```

A.2.4 2 To 1 Multiplexer

```
-- 2 to 1 Mux for COEN6501
-- Purpose: Switches between two inputs, depending on state of control signal

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity two_to_one_mux is
    port
    (
        a, b, sel : in std_logic;
        y : out std_logic
    );
end two_to_one_mux;

-- Begin architecture
architecture rtl of two_to_one_mux is

    -- Define components
    component AND2_6501
        port
        (
            in_A, in_B : in std_logic;
            output : out std_logic
        );
    end component;
    component NOT_6501
        port
        (
            in_A : in std_logic;
            output : out std_logic
        );
    end component;
    component OR2_6501
        port
        (
            in_A, in_B : in std_logic;
            output : out std_logic
        );
    end component;
    -- define linking signals
    signal temp0, temp1, temp2 : std_logic;
begin
    -- Build circuit
    U1: AND2_6501 port map(sel, b, temp0);
    U2: NOT_6501 port map (sel, temp1);
    U3: AND2_6501 port map (temp1,a, temp2);
    U4: OR2_6501 port map (temp0, temp2, y);
end rtl;
```

A.3 Adders

A.3.1 2 Bit Carry Select Block

```
-- Two Bit Carry Select Block
-- Purpose: 2 Bit block of adders for carry select adders

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity csa_2bit_block is
    port
    (
        in_a : in std_logic_vector (1 downto 0);
        in_b : in std_logic_vector (1 downto 0);
        mux_select : in std_logic;
        sum : out std_logic_vector (1 downto 0);
        carryout : out std_logic
    );
end csa_2bit_block;

-- Begin architecture
architecture rtl of csa_2bit_block is

    -- Define components
    component fulladder
        port
        (
            in_A, in_B, in_C : in std_logic;
            carry, sum : out std_logic
        );
    end component;

    component two_to_one_mux
        port
        (
            a, b, sel : in std_logic;
            y : out std_logic
        );
    end component;

    -- Define intermediate signals
    signal temp_sum0 : std_logic_vector (1 downto 0);
    signal temp_sum1 : std_logic_vector (1 downto 0);
    signal temp_carry0a, temp_carry0b : std_logic;
    signal temp_carry1a, temp_carry1b : std_logic;

begin
    -- Generate RCA for carry equal to 0
    U1A: fulladder port map (in_a(0), in_b(0), '0', temp_carry0a, temp_sum0(0));
    U1B: fulladder port map (in_a(1), in_b(1), temp_carry0a, temp_carry0b, temp_sum0(1));
    -- Generate RCA for carry equal to 1
    U2A: fulladder port map (in_a(0), in_b(0), '1', temp_carry1a, temp_sum1(0));
    U2B: fulladder port map (in_a(1), in_b(1), temp_carry1a, temp_carry1b, temp_sum1(1));
    -- Generate Multiplexers for sum
    sum_muxes: for i in 1 downto 0 generate
```

```

        UA: two_to_one_mux port map (temp_sum0(i), temp_sum1(i), mux_select, sum(i));
    end generate;

    -- Generate multiplexer for carry
    U3: two_to_one_mux port map (temp_carry0b, temp_carry1b, mux_select, carryout);
end rtl;

```

A.3.2 4 Bit Carry Select Block

```

-- Four Bit Carry Select Block
-- Purpose: 4 Bit block of adders for carry select adders
-- Written on November 10, 2015

```

```

library ieee;
use ieee.std_logic_1164.all;

```

```

entity csa_4bit_block is

```

```

    port
    (
        in_a : in std_logic_vector (3 downto 0);
        in_b : in std_logic_vector (3 downto 0);
        mux_select : in std_logic;
        sum : out std_logic_vector (3 downto 0);
        carryout : out std_logic
    );

```

```

end csa_4bit_block;

```

```

architecture rtl of csa_4bit_block is

```

```

    component rca_4bit_block
        port
        (
            in_a : in std_logic_vector (3 downto 0);
            in_b : in std_logic_vector (3 downto 0);
            in_c : in std_logic;
            sum : out std_logic_vector (3 downto 0);
            carry_out : out std_logic
        );

```

```

end component;

```

```

    component two_to_one_mux

```

```

        port
        (
            a, b, sel : in std_logic;
            y : out std_logic
        );

```

```

end component;

```

```

    signal temp_sum0 : std_logic_vector (3 downto 0);
    signal temp_sum1 : std_logic_vector (3 downto 0);
    signal temp_carry0 : std_logic;
    signal temp_carry1 : std_logic;

```

```

begin

```

```

    -- Generate RCA for carry equal to 0
    U1: rca_4bit_block port map (in_a, in_b, '0', temp_sum0, temp_carry0);
    -- Generate RCA for carry equal to 1
    U2: rca_4bit_block port map (in_a, in_b, '1', temp_sum1, temp_carry1);

```

```

-- Generate Multiplexers for sum
sum_muxes: for i in 3 downto 0 generate
    UA: two_to_one_mux port map (temp_sum0(i), temp_sum1(i), mux_select, sum(i));
end generate;

-- Generate multiplexer for carry
U3: two_to_one_mux port map (temp_carry0, temp_carry1, mux_select, carryout);
end rtl;

```

A.3.3 10 Bit Carry Select Adder

```

-- 10 Bit Carry Select Adder
-- Purpose: Adds two 10 bit numbers by Carry Select method (2 blocks of 4 bits, 1 block of 2 bits)

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity csa_10bit is
    port
    (
        in_a : in std_logic_vector (9 downto 0);
        in_b : in std_logic_vector (9 downto 0);
        sum : out std_logic_vector (9 downto 0);
        carryout : out std_logic
    );
end csa_10bit;

-- Begin architecture
architecture rtl of csa_10bit is

    -- Define components
    component csa_4bit_block
        port
        (
            in_a : in std_logic_vector (3 downto 0);
            in_b : in std_logic_vector (3 downto 0);
            mux_select : in std_logic;
            sum : out std_logic_vector (3 downto 0);
            carryout : out std_logic
        );
    end component;

    component csa_2bit_block
        port
        (
            in_a : in std_logic_vector (1 downto 0);
            in_b : in std_logic_vector (1 downto 0);
            mux_select : in std_logic;
            sum : out std_logic_vector (1 downto 0);
            carryout : out std_logic
        );
    end component;

    component halfadd
        port
        (

```

```

        in_a : in std_logic;
        in_b : in std_logic;
        carryout : out std_logic;
        sum : out std_logic
    );
end component;

component fulladder
port
(
    in_A, in_B, in_C : in std_logic;
    carry, sum : out std_logic
);
end component;

-- Define signals
signal temp_carry : std_logic_vector (3 downto 0);
signal carryout_temp : std_logic;
begin
    -- Instance half adder for sum 0
    U1: halfadd port map (in_a(0), in_b(0), temp_carry(0), sum(0));
    -- Generate full adders for remaining first three bits
    lower_four_full_adders: for i in 1 to 3 generate
        UA: fulladder port map (in_a(i), in_b(i), temp_carry(i-1), temp_carry(i),
→ sum(i));
    end generate;
    -- Instance 4 bit carry select adder block
    U2: csa_4bit_block port map (in_a (7 downto 4), in_b (7 downto 4), temp_carry(3),
→ sum(7 downto 4), carryout_temp);
    U3: csa_2bit_block port map (in_a (9 downto 8), in_b (9 downto 8), carryout_temp,
→ sum(9 downto 8), carryout);
end rtl;

```

A.3.4 12 Bit Carry Select Adder

```

-- 12 Bit Carry Select Adder
-- Purpose: Adds two 12 bit numbers by Carry Select method (3 blocks of 4 bits)

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity csa_12bit is
    port
    (
        in_a : in std_logic_vector (11 downto 0);
        in_b : in std_logic_vector (11 downto 0);
        sum : out std_logic_vector (11 downto 0);
        carryout : out std_logic
    );
end csa_12bit;

-- Begin architecture
architecture rtl of csa_12bit is

    component csa_4bit_block

```

```

    port
    (
        in_a : in std_logic_vector (3 downto 0);
        in_b : in std_logic_vector (3 downto 0);
        mux_select : in std_logic;
        sum : out std_logic_vector (3 downto 0);
        carryout : out std_logic
    );
end component;

component halfadd
port
(
    in_a : in std_logic;
    in_b : in std_logic;
    carryout : out std_logic;
    sum : out std_logic
);
end component;

component fulladder
port
(
    in_A, in_B, in_C : in std_logic;
    carry, sum : out std_logic
);
end component;

-- Define signals
signal temp_carry : std_logic_vector (3 downto 0);
signal temp_muxselect : std_logic;
begin
    -- Instance half adder for sum 0
    U1: halfadd port map (in_a(0), in_b(0), temp_carry(0), sum(0));
    -- Generate full adders for remaining first three bits
    lower_three_full_adders: for i in 1 to 3 generate
        UA: fulladder port map (in_a(i), in_b(i), temp_carry(i-1), temp_carry(i),
→ sum(i));
    end generate;
    -- Instance 4 bit carry select adder blocks
    U2: csa_4bit_block port map (in_a (7 downto 4), in_b (7 downto 4), temp_carry(3),
→ sum(7 downto 4), temp_muxselect);
    U3: csa_4bit_block port map (in_a (11 downto 8), in_b (11 downto 8), temp_muxselect,
→ sum(11 downto 8), carryout);
end rtl;

```

A.3.5 14 Bit Carry Select Adder

```

-- 14 Bit Carry Select Adder
-- Purpose: Adds two 14 bit numbers by Carry Select method (3 blocks of 4 bits, 1 block of 2)

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity csa_14bit is

```



```

port
(
    in_a : in std_logic_vector (13 downto 0);
    in_b : in std_logic_vector (13 downto 0);
    sum : out std_logic_vector (13 downto 0);
    carryout : out std_logic
);
end csa_14bit;

-- Begin architecture
architecture rtl of csa_14bit is

    -- Define components
    component csa_4bit_block
    port
    (
        in_a : in std_logic_vector (3 downto 0);
        in_b : in std_logic_vector (3 downto 0);
        mux_select : in std_logic;
        sum : out std_logic_vector (3 downto 0);
        carryout : out std_logic
    );
    end component;

    component csa_2bit_block
    port
    (
        in_a : in std_logic_vector (1 downto 0);
        in_b : in std_logic_vector (1 downto 0);
        mux_select : in std_logic;
        sum : out std_logic_vector (1 downto 0);
        carryout : out std_logic
    );
    end component;

    component halfadd
    port
    (
        in_a : in std_logic;
        in_b : in std_logic;
        carryout : out std_logic;
        sum : out std_logic
    );
    end component;

    component fulladder
    port
    (
        in_A, in_B, in_C : in std_logic;
        carry, sum : out std_logic
    );
    end component;

    -- Define signals
    signal temp_carry : std_logic_vector (4 downto 0);
    signal temp_muxselect : std_logic_vector (1 downto 0);
    begin
        -- Instance half adder for sum 0

```

```

    U1: halfadd port map (in_a(0), in_b(0), temp_carry(0), sum(0));
    -- Generate full adders for remaining first three bits
    lower_three_full_adders: for i in 1 to 3 generate
        UA: fulladder port map (in_a(i), in_b(i), temp_carry(i-1), temp_carry(i),
→ sum(i));
        end generate;
        -- Instance 4 bit carry select adder block
    U2: csa_4bit_block port map (in_a (7 downto 4), in_b (7 downto 4), temp_carry(3),
→ sum(7 downto 4), temp_muxselect(0));
        -- Instance 4 bit carry select adder block
    U3: csa_4bit_block port map (in_a (11 downto 8), in_b (11 downto 8),
→ temp_muxselect(0), sum(11 downto 8), temp_muxselect(1));
        -- Instance 2 bit carry select adder block
    U4: csa_2bit_block port map (in_a (13 downto 12), in_b (13 downto 12),
→ temp_muxselect(1), sum(13 downto 12), carryout);

end rtl;

```

A.3.6 9 Bit Incrementer

```

--          9 Bit Incrementer for Two's Complementer
--          Purpose: To add 9 bits, in use for 2's complementer

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity increment_9bit is
    port
    (
        value_in : in std_logic_vector (8 downto 0);
        carry_in  : in std_logic;
        value_out  : out std_logic_vector (8 downto 0);
        carryout  : out std_logic
    );
end increment_9bit;

-- Begin architecture
architecture rtl of increment_9bit is
    -- Define components
    component halfadd
        port
        (
            in_a : in std_logic;
            in_b : in std_logic;
            sum  : out std_logic;
            carryout : out std_logic
        );
    end component;
    -- Carry propagate signal
    signal temp_carry : std_logic_vector (8 downto 0);

begin
    -- Define halfadder chain
    U1: halfadd port map (value_in(0), carry_in, value_out(0), temp_carry(0));
    U2: halfadd port map (value_in(1), temp_carry(0), value_out(1), temp_carry(1));
    U3: halfadd port map (value_in(2), temp_carry(1), value_out(2), temp_carry(2));

```

```

        U4: halfadd port map (value_in(3), temp_carry(2), value_out(3), temp_carry(3));
        U5: halfadd port map (value_in(4), temp_carry(3), value_out(4), temp_carry(4));
        U6: halfadd port map (value_in(5), temp_carry(4), value_out(5), temp_carry(5));
        U7: halfadd port map (value_in(6), temp_carry(5), value_out(6), temp_carry(6));
        U8: halfadd port map (value_in(7), temp_carry(6), value_out(7), temp_carry(7));
        U9: halfadd port map (value_in(8), temp_carry(7), value_out(8), carryout);
end rtl;

```

A.3.7 4 Bit Ripple Carry Adder Block

```

-- Four Bit Ripple Carry Adder
-- Purpose: To add 4 bits. Used as blocks for carry select adders

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity rca_4bit_block is
    port
    (
        in_a : in std_logic_vector (3 downto 0);
        in_b : in std_logic_vector (3 downto 0);
        in_c : in std_logic;
        sum : out std_logic_vector (3 downto 0);
        carry_out : out std_logic
    );
end rca_4bit_block;

-- Begin architecture
architecture rtl of rca_4bit_block is
    -- Define components
    component fulladder
        port
        (
            in_a, in_b, in_c : in std_logic;
            carry, sum : out std_logic
        );
    end component;

    -- Define carry vectory
    signal temp_carry : std_logic_vector (3 downto 0);
begin
    U1: fulladder port map (in_a(0), in_b(0), in_c, temp_carry(0), sum(0));
    -- Generate full adder chain
    fulladder_generate: for i in 1 to 3 generate
        UA: fulladder port map (in_a(i), in_b(i), temp_carry(i-1), temp_carry(i),
        ↪ sum(i));
    end generate;
    -- Cast end of vector as actual carry out
    carry_out <= temp_carry(3);
end rtl;

```

A.4 Higher Level Functional Blocks

A.4.1 Booth Adder, Device Under Test

```
-- Booth Adder, Device Under Test
-- Purpose: Circuit simplification by instancing Logic Block with Booth Units and Adders

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity ba_dut is
  port
  (
    in_a : in std_logic_vector (7 downto 0);
    in_b : in std_logic_vector (7 downto 0);
    sum_out : out std_logic_vector (15 downto 0)
  );
end ba_dut;

-- Begin architecture
architecture rtl of ba_dut is

  -- Define components
  component booth_unit_0
    port
    (
      data_in : in std_logic_vector (7 downto 0);
      booth_control : in std_logic_vector (2 downto 0);
      data_out : out std_logic_vector (15 downto 0)
    );
  end component;

  component booth_unit_1
    port
    (
      data_in : in std_logic_vector (7 downto 0);
      booth_control : in std_logic_vector (2 downto 0);
      data_out : out std_logic_vector (13 downto 0)
    );
  end component;

  component booth_unit_2 is
    port
    (
      data_in : in std_logic_vector (7 downto 0);
      booth_control : in std_logic_vector (2 downto 0);
      data_out : out std_logic_vector (11 downto 0)
    );
  end component;

  component booth_unit_3
    port
    (
      data_in : in std_logic_vector (7 downto 0);
      booth_control : in std_logic_vector (2 downto 0);
      data_out : out std_logic_vector (9 downto 0)
    );
  end component;
end architecture;
```

```

    );
end component;

component booth_adder
  port
  (
    data0_in : in std_logic_vector (15 downto 0);
    data1_in : in std_logic_vector (13 downto 0);
    data2_in : in std_logic_vector (11 downto 0);
    data3_in : in std_logic_vector (9 downto 0);
    sum_out : out std_logic_vector (15 downto 0)
  );
end component;

-- Define result vectors and control signals
signal booth_control0, booth_control1, booth_control2, booth_control3 : std_logic_vector (2
→  downto 0);
signal booth_result0 : std_logic_vector (15 downto 0);
signal booth_result1 : std_logic_vector (13 downto 0);
signal booth_result2 : std_logic_vector (11 downto 0);
signal booth_result3 : std_logic_vector (9 downto 0);

begin
  -- Create Booth signals
  booth_control0 (2 downto 1) <= in_b (1 downto 0);
  booth_control0 (0) <= '0';

  booth_control1 <= in_b (3 downto 1);
  booth_control2 <= in_b (5 downto 3);
  booth_control3 <= in_b (7 downto 5);
  -- Instance Booth Units
  U1: booth_unit_0 port map (in_a, booth_control0, booth_result0);
  U2: booth_unit_1 port map (in_a, booth_control1, booth_result1);
  U3: booth_unit_2 port map (in_a, booth_control2, booth_result2);
  U4: booth_unit_3 port map (in_a, booth_control3, booth_result3);
  -- Instance Booth Adder
  U5: booth_adder port map (booth_result0, booth_result1, booth_result2, booth_result3,
→  sum_out);
end rtl;

```

A.4.2 9 Bit Barrel Shifter

```

--      9 Bit Barrel Shifter
--      Purpose: To carry out shifts
--              00 = Zero out contents, 01 = No shift
--              10 = Shift left

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity barrel_shift_9bit is
  port
  (
    data_in : in std_logic_vector (8 downto 0);
    ctrl : in std_logic_vector (1 downto 0);

```

```

        data_out : out std_logic_vector (8 downto 0)
    );
end barrel_shift_9bit;

-- Begin architecture
architecture rtl of barrel_shift_9bit is

    -- Define components
    component two_to_one_mux
        port
        (
            a, b, sel : in std_logic;
            y : out std_logic
        );
    end component;

    -- Define temporary signal vectory for passing to second MUX row
    signal temp : std_logic_vector (8 downto 0);

begin

    -- First row: If ctrl = 00, then zero
    --                               If ctrl = 01, then do nothing (x1)
    ROW0_MUX: for i in 0 to 8 generate
        ROW0_U: two_to_one_mux port map ('0', data_in(i), ctrl(0), temp(i));
    end generate;

    -- Second row: If ctrl = 1X, then shift right by two (x2)
    ROW1_MUX0: two_to_one_mux port map (temp(0), '0', ctrl(1), data_out(0));
    ROW1_MUX1: two_to_one_mux port map (temp(1), data_in(0), ctrl(1), data_out(1));
    ROW1_MUX2: two_to_one_mux port map (temp(2), data_in(1), ctrl(1), data_out(2));
    ROW1_MUX3: two_to_one_mux port map (temp(3), data_in(2), ctrl(1), data_out(3));
    ROW1_MUX4: two_to_one_mux port map (temp(4), data_in(3), ctrl(1), data_out(4));
    ROW1_MUX5: two_to_one_mux port map (temp(5), data_in(4), ctrl(1), data_out(5));
    ROW1_MUX6: two_to_one_mux port map (temp(6), data_in(5), ctrl(1), data_out(6));
    ROW1_MUX7: two_to_one_mux port map (temp(7), data_in(6), ctrl(1), data_out(7));
    ROW1_MUX8: two_to_one_mux port map (temp(8), data_in(7), ctrl(1), data_out(8));

end rtl;

```

A.4.3 Booth Adder

```

-- Booth Adder
-- Purpose: To add Booth Partial Products

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity booth_adder is
    port
    (
        data0_in : in std_logic_vector (15 downto 0);
        data1_in : in std_logic_vector (13 downto 0);
        data2_in : in std_logic_vector (11 downto 0);
        data3_in : in std_logic_vector (9 downto 0);
        sum_out  : out std_logic_vector (15 downto 0)
    );
end booth_adder;

```

```

-- Begin architecture
architecture rtl of booth_adder is

    -- Define components
    component csa_10bit
        port
        (
            in_a : in std_logic_vector (9 downto 0);
            in_b : in std_logic_vector (9 downto 0);
            sum : out std_logic_vector (9 downto 0);
            carryout : out std_logic
        );
    end component;

    component csa_12bit
        port
        (
            in_a : in std_logic_vector (11 downto 0);
            in_b : in std_logic_vector (11 downto 0);
            sum : out std_logic_vector (11 downto 0);
            carryout : out std_logic
        );
    end component;

    component csa_14bit
        port
        (
            in_a : in std_logic_vector (13 downto 0);
            in_b : in std_logic_vector (13 downto 0);
            sum : out std_logic_vector (13 downto 0);
            carryout : out std_logic
        );
    end component;
    -- Define result vectors from carry select adders and temporary signals
    signal csa_14_sum_out : std_logic_vector (13 downto 0);
    signal csa_12_sum_out : std_logic_vector (11 downto 0);
    signal csa_10_sum_out : std_logic_vector (9 downto 0);
    signal temp_12_bit : std_logic_vector (11 downto 0);
    signal temp_14_bit : std_logic_vector (13 downto 0);

begin
    -- Assign first two sums out
    sum_out (1 downto 0) <= data0_in (1 downto 0);
    sum_out (3 downto 2) <= temp_14_bit (1 downto 0);
    -- Get resultant B
    temp_12_bit(1 downto 0) <= data2_in (1 downto 0);
    U1: csa_10bit port map (data3_in, data2_in(11 downto 2), temp_12_bit(11 downto 2));

    -- Get resultant A
    U2: csa_14bit port map (data0_in(15 downto 2), data1_in, temp_14_bit);

    -- Do final sum
    U3: csa_12bit port map (temp_12_bit, temp_14_bit (13 downto 2), sum_out(15 downto
→ 4));

end rtl;

```

A.4.4 Booth Decoder

```
-- Booth Signal Decoder
-- Purpose: To generate appropriate Booth control signals

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity booth_decoder is
  port
  (
    data_in : in std_logic_vector (2 downto 0);
    ctrl : out std_logic_vector (1 downto 0)
  );
end booth_decoder;

-- Begin architecture
architecture rtl of booth_decoder is

  -- Define components
  component XOR2_6501
    port
    (
      in_A, in_B : in std_logic;
      output : out std_logic
    );
  end component;
  component OR2_6501
    port
    (
      in_A, in_B : in std_logic;
      output : out std_logic
    );
  end component;
  component AND2_6501
    port
    (
      in_A, in_B : in std_logic;
      output : out std_logic
    );
  end component;
  component NOT_6501
    port
    (
      in_A : in std_logic;
      output : out std_logic
    );
  end component;

  -- Define linking signals
  signal temp0, NOTB, NOTC, NOTA, temp1, temp2, temp3, temp4 : std_logic;

begin
  -- Instance components
  U0: XOR2_6501 port map (data_in(0), data_in(1), ctrl(0));
  U1: NOT_6501 port map (data_in(2), NOTA);
```



```

        U2: NOT_6501 port map (data_in(1), NOTB);
        U3: NOT_6501 port map (data_in(0), NOTC);
        U4: AND2_6501 port map (NOTA, data_in(1), temp0);
        U5: AND2_6501 port map (temp0, data_in(0), temp1);
        U6: AND2_6501 port map (NOTB, NOTC, temp2);
        U7: AND2_6501 port map (temp2, data_in(2), temp3);
        U8: OR2_6501 port map (temp3, temp1, ctrl(1));
end rtl;

```

A.4.5 Booth Unit 0 (16 Bit)

```

-- Booth Unit 0
-- Purpose: To generate the first Booth partial product

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity booth_unit_0 is
    port
    (
        data_in : in std_logic_vector (7 downto 0);
        booth_control : in std_logic_vector (2 downto 0);
        data_out : out std_logic_vector (15 downto 0)
    );
end booth_unit_0;

-- Begin architecture
architecture rtl of booth_unit_0 is

    -- Define components
    component twoscomp_9bit
        port
        (
            data_in : in std_logic_vector (7 downto 0);
            enable : in std_logic;
            data_out : out std_logic_vector(8 downto 0)
        );
    end component;

    component barrel_shift_9bit
        port
        (
            data_in : in std_logic_vector (8 downto 0);
            ctrl : in std_logic_vector (1 downto 0);
            data_out : out std_logic_vector (8 downto 0)
        );
    end component;

    component booth_decoder
        port
        (
            data_in : in std_logic_vector (2 downto 0);
            ctrl : out std_logic_vector (1 downto 0)
        );
    end component;

```

```

-- Define control signal for Barrel shifter
signal barrel_shift_ctrl : std_logic_vector (1 downto 0);
-- Signal for two's complement result
signal twoscomp_out : std_logic_vector (8 downto 0);
-- Sign extend bit for 9 bit result
signal sign_extend : std_logic;
-- Barrel shifter result
signal barrel_shift_out : std_logic_vector (8 downto 0);
begin
    U1: booth_decoder port map (booth_control, barrel_shift_ctrl);
    U2: twoscomp_9bit port map (data_in, booth_control(2), twoscomp_out);
    U3: barrel_shift_9bit port map (twoscomp_out, barrel_shift_ctrl, barrel_shift_out);
    -- Assign the rest of the signals
    sign_extend <= barrel_shift_out(8);
    data_out (8 downto 0) <= barrel_shift_out;
    data_out (15 downto 9) <= (sign_extend, sign_extend, sign_extend, sign_extend,
→ sign_extend, sign_extend, sign_extend);
end rtl;

```

A.4.6 Booth Unit 1 (14 Bit)

```

-- Booth Unit 1
-- Purpose: To generate the second Booth partial product

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity booth_unit_1 is
    port
    (
        data_in : in std_logic_vector (7 downto 0);
        booth_control : in std_logic_vector (2 downto 0);
        data_out : out std_logic_vector (13 downto 0)
    );
end booth_unit_1;

-- Begin architecture
architecture rtl of booth_unit_1 is

    -- Define components
    component twoscomp_9bit
        port
        (
            data_in : in std_logic_vector (7 downto 0);
            enable : in std_logic;
            data_out : out std_logic_vector(8 downto 0)
        );
    end component;

    component barrel_shift_9bit
        port
        (
            data_in : in std_logic_vector (8 downto 0);
            ctrl : in std_logic_vector (1 downto 0);
            data_out : out std_logic_vector (8 downto 0)
        );
    end component;

```

```

    );
end component;

component booth_decoder
port
(
    data_in : in std_logic_vector (2 downto 0);
    ctrl : out std_logic_vector (1 downto 0)
);
end component;

-- Define control signal for Barrel shifter
signal barrel_shift_ctrl : std_logic_vector (1 downto 0);
-- Signal for two's complement result
signal twoscomp_out : std_logic_vector (8 downto 0);
-- Sign extend bit for 9 bit result
signal sign_extend : std_logic;
-- Barrel shifter result
signal barrel_shift_out : std_logic_vector (8 downto 0);
begin
    U1: booth_decoder port map (booth_control, barrel_shift_ctrl);
    U2: twoscomp_9bit port map (data_in, booth_control(2), twoscomp_out);
    U3: barrel_shift_9bit port map (twoscomp_out, barrel_shift_ctrl, barrel_shift_out);
    -- Assign the rest of the signals
    sign_extend <= barrel_shift_out(8);
    data_out (8 downto 0) <= barrel_shift_out;
    data_out (13 downto 9) <= (sign_extend, sign_extend, sign_extend, sign_extend,
↪ sign_extend);
end rtl;

```

A.4.7 Booth Unit 2 (12 Bit)

```

-- Booth Unit 2
-- Purpose: To generate the third Booth partial product

-- Define Libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity booth_unit_2 is
    port
    (
        data_in : in std_logic_vector (7 downto 0);
        booth_control : in std_logic_vector (2 downto 0);
        data_out : out std_logic_vector (11 downto 0)
    );
end booth_unit_2;

-- Begin architecture
architecture rtl of booth_unit_2 is

    -- Define components
    component twoscomp_9bit
        port
        (
            data_in : in std_logic_vector (7 downto 0);

```

```

        enable : in std_logic;
        data_out : out std_logic_vector(8 downto 0)
    );
end component;

component barrel_shift_9bit
port
(
    data_in : in std_logic_vector (8 downto 0);
    ctrl : in std_logic_vector (1 downto 0);
    data_out : out std_logic_vector (8 downto 0)
);
end component;

component booth_decoder
port
(
    data_in : in std_logic_vector (2 downto 0);
    ctrl : out std_logic_vector (1 downto 0)
);
end component;

-- Define control signal for Barrel shifter
signal barrel_shift_ctrl : std_logic_vector (1 downto 0);
-- Signal for two's complement result
signal twoscomp_out : std_logic_vector (8 downto 0);
-- Sign extend bit for 9 bit result
signal sign_extend : std_logic;
-- Barrel shifter result
signal barrel_shift_out : std_logic_vector (8 downto 0);
begin
    U1: booth_decoder port map (booth_control, barrel_shift_ctrl);
    U2: twoscomp_9bit port map (data_in, booth_control(2), twoscomp_out);
    U3: barrel_shift_9bit port map (twoscomp_out, barrel_shift_ctrl, barrel_shift_out);
    -- Assign the rest of the signals
    sign_extend <= barrel_shift_out(8);
    data_out (8 downto 0) <= barrel_shift_out;
    data_out (11 downto 9) <= (sign_extend, sign_extend, sign_extend);
end rtl;

```

A.4.8 Booth Unit 3 (10 Bit)

```

-- Booth Unit 3
-- Purpose: To generate the fourth Booth partial product

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity booth_unit_3 is
port
(
    data_in : in std_logic_vector (7 downto 0);
    booth_control : in std_logic_vector (2 downto 0);
    data_out : out std_logic_vector (9 downto 0)
);

```

```

end booth_unit_3;

-- Begin architecture
architecture rtl of booth_unit_3 is

    component twoscomp_9bit
        port
        (
            data_in : in std_logic_vector (7 downto 0);
            enable : in std_logic;
            data_out : out std_logic_vector(8 downto 0)
        );
    end component;

    component barrel_shift_9bit
        port
        (
            data_in : in std_logic_vector (8 downto 0);
            ctrl : in std_logic_vector (1 downto 0);
            data_out : out std_logic_vector (8 downto 0)
        );
    end component;

    component booth_decoder
        port
        (
            data_in : in std_logic_vector (2 downto 0);
            ctrl : out std_logic_vector (1 downto 0)
        );
    end component;

    -- Define control signal for Barrel shifter
    signal barrel_shift_ctrl : std_logic_vector (1 downto 0);
    -- Signal for two's complement result
    signal twoscomp_out : std_logic_vector (8 downto 0);
    -- Barrel shifter result
    signal barrel_shift_out : std_logic_vector (8 downto 0);
begin
    U1: booth_decoder port map (booth_control, barrel_shift_ctrl);
    U2: twoscomp_9bit port map (data_in, booth_control(2), twoscomp_out);
    U3: barrel_shift_9bit port map (twoscomp_out, barrel_shift_ctrl, barrel_shift_out);
    data_out (8 downto 0) <= barrel_shift_out;
    data_out (9) <= barrel_shift_out(8);
end rtl;

```

A.4.9 9 Bit Two's Complementer

```

--          9 Bit Two's Complementer
--          Purpose: To generate 2's complement

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity twoscomp_9bit is
    port

```

```

    (
        data_in : in std_logic_vector (7 downto 0);
        enable : in std_logic;
        data_out : out std_logic_vector(8 downto 0)
    );
end twoscomp_9bit;

-- Begin architecture
architecture twoscomp9bit_arch of twoscomp_9bit is

    -- Define components
    component increment_9bit
        port
        (
            value_in : in std_logic_vector (8 downto 0);
            carry_in : in std_logic;
            value_out : out std_logic_vector (8 downto 0);
            carryout : out std_logic
        );
    end component;

    component XOR2_6501
        port
        (
            in_a : in std_logic;
            in_b : in std_logic;
            output : out std_logic
        );
    end component;

    -- Store input as temporary vector
    signal temp_value : std_logic_vector (8 downto 0);
    -- Resultant of complementing if enabled
    signal xor_out : std_logic_vector (8 downto 0);
    -- Temp carryout
    signal temp_carryout : std_logic;
begin
    -- Sign extend
    temp_value(7 downto 0) <= data_in (7 downto 0);
    temp_value(8) <= data_in(7);

    -- Generate XOR gates
    XOR_MODE: for i in 0 to 8 generate
        UA: XOR2_6501 port map (temp_value(i), enable, xor_out(i));
    end generate XOR_MODE;
    U1: increment_9bit port map (xor_out, enable, data_out);
end twoscomp9bit_arch;

```

A.5 Registers

A.5.1 8 Bit Operand Register

```

--      8 Bit Operand Register
--      Purpose: To store 8 bit multiplication operand

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

```

```

-- Begin entity
entity operand_register is
  port
  (
    data_in : in std_logic_vector(7 downto 0);
    clock, reset : in std_logic;
    data_out : out std_logic_vector(7 downto 0)
  );
end operand_register;
-- Architecture definition of operand register
architecture operand_register_arch of operand_register is
  -- Declared signals
  signal clock_invert, reset_invert : std_logic;
  signal not_q : std_logic_vector (7 downto 0);
  -- Component declaration for D Flip Flop
  component DFF_6501
  port
  (
    data_in : in std_logic;
    clock : in std_logic;
    reset : in std_logic;
    not_q_out : out std_logic;
    q_out : out std_logic
  );
  end component;

  begin
  -- Structural definition
  -- Generate registers
  register_generate: for i in 0 to 7 generate
    UA: DFF_6501 port map (data_in(i), clock, reset, not_q(i), data_out(i));
  end generate;
end operand_register_arch;

```

A.5.2 10 Bit Operand Register

```

--      10 Bit Operand Register
--      Purpose: To store 10 bit multiplication partial product

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity operand_register_10bit is
  port
  (
    data_in : in std_logic_vector(9 downto 0);
    clock, reset : in std_logic;
    data_out : out std_logic_vector(9 downto 0)
  );
end operand_register_10bit;
-- Architecture definition of operand register
architecture operand_register_arch of operand_register_10bit is
  -- Declared signals
  signal not_q : std_logic_vector (9 downto 0);

```

```

-- Component declaration for D Flip Flop
component DFF_6501
  port
  (
    data_in : in std_logic;
    clock   : in std_logic;
    reset   : in std_logic;
    not_q_out : out std_logic;
    q_out   : out std_logic
  );
end component;
-- Declaration of NOT Gate
component NOT_6501
  port
  (
    in_A : in std_logic;
    output : out std_logic
  );
end component;
begin
-- Structural definition
-- Generate registers
register_generate: for i in 0 to 9 generate
  UA: DFF_6501 port map (data_in(i), clock, reset, not_q(i), data_out(i));
end generate;
end operand_register_arch;

```

A.5.3 14 Bit Operand Register

```

--      14 Bit Operand Register
--      Purpose: To store 14 bit multiplication operand

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity operand_register_14bit is
  port
  (
    data_in : in std_logic_vector(13 downto 0);
    clock, reset : in std_logic;
    data_out : out std_logic_vector(13 downto 0)
  );
end operand_register_14bit;
-- Architecture definition of operand register
architecture operand_register_arch of operand_register_14bit is
  -- Declared signals
  signal clock_invert, reset_invert : std_logic;
  signal not_q : std_logic_vector (13 downto 0);
  -- Component declaration for D Flip Flop
  component DFF_6501
    port
    (
      data_in : in std_logic;
      clock   : in std_logic;
      reset   : in std_logic;
      not_q_out : out std_logic;
    );
  end component;

```



```

        q_out : out std_logic
    );
end component;
-- Declaration of NOT Gate
component NOT_6501
    port
    (
        in_A : in std_logic;
        output : out std_logic
    );
end component;
begin
-- Structural definition
-- Generate registers
    register_generate: for i in 0 to 13 generate
        UA: DFF_6501 port map (data_in(i), clock, reset, not_q(i), data_out(i));
    end generate;
end operand_register_arch;

```

A.5.4 16 Bit Operand Register

```

--      16 Bit Operand Register
--      Purpose: To store 16 bit multiplication partial product

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity operand_register_16bit is
    port
    (
        data_in : in std_logic_vector(15 downto 0);
        clock, reset : in std_logic;
        data_out : out std_logic_vector(15 downto 0)
    );
end operand_register_16bit;
-- Architecture definition of operand register
architecture operand_register_arch of operand_register_16bit is
-- Declared signals
    signal clock_invert, reset_invert : std_logic;
    signal not_q : std_logic_vector (15 downto 0);
-- Component declaration for D Flip Flop
    component DFF_6501
        port
        (
            data_in : in std_logic;
            clock : in std_logic;
            reset : in std_logic;
            not_q_out : out std_logic;
            q_out : out std_logic
        );
    end component;
-- Declaration of NOT Gate
    component NOT_6501
        port
        (
            in_A : in std_logic;

```

```

        output : out std_logic
    );
end component;
begin
    -- Structural definition
    -- Generate registers
    register_generate: for i in 0 to 15 generate
        UA: DFF_6501 port map (data_in(i), clock, reset, not_q(i), data_out(i));
    end generate;
end operand_register_arch;

```

A.6 Non-Pipelined Implementation

```

-- 8 bit Booth Multiplier
-- Purpose: To multiply 8 bit signed numbers

-- Define libraries
library ieee;
use ieee.std_logic_1164.all;
-- Begin entity
entity booth_8 is
    port
    (
        operand_a : in std_logic_vector (7 downto 0);
        operand_b : in std_logic_vector (7 downto 0);
        clr : in std_logic;
        clk : in std_logic;
        load: in std_logic;
        z : out std_logic_vector (15 downto 0);
        end_flag : out std_logic
    );
end booth_8;
-- Begin architecture
architecture rtl of booth_8 is
    -- Component declarations
    component operand_register
        port
        (
            data_in : in std_logic_vector(7 downto 0);
            clock, reset : in std_logic;
            data_out : out std_logic_vector(7 downto 0)
        );
    end component;

    component ba_dut
        port
        (
            in_a : in std_logic_vector (7 downto 0);
            in_b : in std_logic_vector (7 downto 0);
            sum_out : out std_logic_vector (15 downto 0)
        );
    end component;
    component AND2_6501
        port
        (
            in_A, in_B : in std_logic;
            output : out std_logic

```

```

    );
end component;

component NOT_6501
port
(
    in_A : in std_logic;
    output : out std_logic
);
end component;

component dff_6501
port
(
    data_in : in std_logic;
    clock : in std_logic;
    reset : in std_logic;
    not_q_out : out std_logic;
    q_out : out std_logic
);
end component;

component operand_register_16bit
port
(
    data_in : in std_logic_vector(15 downto 0);
    clock, reset : in std_logic;
    data_out : out std_logic_vector(15 downto 0)
);
end component;
-- Signal declarations
signal load_ctrl : std_logic;
signal booth_operand_a : std_logic_vector (7 downto 0);
signal booth_operand_b : std_logic_vector (7 downto 0);
signal booth_result : std_logic_vector (15 downto 0);
signal not_load: std_logic;
signal not_clr : std_logic;
signal result_reg_out : std_logic_vector (15 downto 0);
signal valid_data, not_valid_data : std_logic;
begin

    -- Create operand registers
    U1: operand_register port map (operand_a, not_load, not_clr, booth_operand_a);
    U2: operand_register port map (operand_b, not_load, not_clr, booth_operand_b);

    -- Create Booth Behemoth
    U3: ba_dut port map (booth_operand_a, booth_operand_b, booth_result);

    -- Create result register
    U4: operand_register_16bit port map (booth_result, clk, not_clr, result_reg_out);

    -- Cast register out to Z port
    z <= result_reg_out;

    -- Create data valid flag generator
    -- Negate load
    U5: NOT_6501 port map (load, not_load);
    -- Negate reset

```

```

        U6: NOT_6501 port map (clr, not_clr);
        -- Generate DFF to store
        U7: dff_6501 port map (not_load, clk, not_clr, not_valid_data, valid_data);
        -- Cast to END flag
        end_flag <= valid_data;
end rtl;

```

A.7 Pipelined Implementation

A.7.1 Top-Level Implementation

```

-- 8 bit Booth Multiplier
-- Purpose: To multiply 8 bit signed numbers

-- Library declarations
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity booth_8_pipelined is
    port
    (
        operand_a : in std_logic_vector (7 downto 0);
        operand_b : in std_logic_vector (7 downto 0);
        clr : in std_logic;
        clk : in std_logic;
        load: in std_logic;
        z : out std_logic_vector (15 downto 0);
        end_flag : out std_logic
    );
end booth_8_pipelined;

-- Begin architecture
architecture rtl of booth_8_pipelined is
    -- Declaration of Operand Register
    component operand_register
        port
        (
            data_in : in std_logic_vector(7 downto 0);
            clock, reset : in std_logic;
            data_out : out std_logic_vector(7 downto 0)
        );
    end component;

    -- Declaration of Pipeline Stage 0
    component booth_pipeline0
        port
        (
            operand_a : in std_logic_vector (7 downto 0);
            operand_b : in std_logic_vector (7 downto 0);
            clk : in std_logic;
            clr : in std_logic;
            data_valid_in : in std_logic;
            out_bu_0 : out std_logic_vector (15 downto 0);
            out_bu_1 : out std_logic_vector (13 downto 0);
            out_bu_2 : out std_logic_vector (11 downto 0);
            out_bu_3 : out std_logic_vector (9 downto 0);
            data_valid_out : out std_logic
        );
    end component;

```

```

    );
end component;

-- Declaration of Pipeline Stage 1
component booth_pipeline1
  port
  (
    bu_0_in : in std_logic_vector (15 downto 0);
    bu_1_in : in std_logic_vector (13 downto 0);
    bu_2_in : in std_logic_vector (11 downto 0);
    bu_3_in : in std_logic_vector (9 downto 0);
    data_valid_in : in std_logic;
    clk : in std_logic;
    clr : in std_logic;
    result_out : out std_logic_vector (15 downto 0);
    data_valid_out : out std_logic
  );
end component;

component NOT_6501
  port
  (
    in_A : in std_logic;
    output : out std_logic
  );
end component;

-- Valid data signals for pipeline
signal valid_data : std_logic_vector (1 downto 0);
-- Inverted signals for control signals
signal not_clr, not_load : std_logic;
-- Operand vectors
signal mult_operand_a, mult_operand_b : std_logic_vector (7 downto 0);
-- Pipeline 0 output vectors
signal p0_out0 : std_logic_vector (15 downto 0);
signal p0_out1 : std_logic_vector (13 downto 0);
signal p0_out2 : std_logic_vector (11 downto 0);
signal p0_out3 : std_logic_vector (9 downto 0);
-- Pipeline 1 output vector
signal p1_result : std_logic_vector (15 downto 0);

begin
  -- Create control signal inverters
  NOT0: NOT_6501 port map (load, not_load);
  NOT1: NOT_6501 port map (clr, not_clr);

  -- Create operand registers
  OP_A: operand_register port map(operand_a, not_load, not_clr, mult_operand_a);
  OP_B: operand_register port map (operand_b, not_load, not_clr, mult_operand_b);

  -- Pipeline Stage 0
  PIPE0: booth_pipeline0 port map (mult_operand_a, mult_operand_b, clk, not_clr,
→ not_load, p0_out0, p0_out1, p0_out2, p0_out3, valid_data(0));

  -- Pipeline Stage 1
  PIPE1: booth_pipeline1 port map (p0_out0, p0_out1, p0_out2, p0_out3, valid_data(0),
→ clk, not_clr, p1_result, valid_data(1));
  -- Set valid flag

```

```

        end_flag <= valid_data(1);
        -- Set output
        z <= p1_result;
end rtl;

```

A.7.2 Pipeline Stage 0

```

-- Booth_8 Pipeline Stage 0
-- Purpose: To store Booth Unit results

-- Library declaration
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity booth_pipeline0 is
    port
    (
        operand_a : in std_logic_vector (7 downto 0);
        operand_b : in std_logic_vector (7 downto 0);
        clk : in std_logic;
        clr : in std_logic;
        data_valid_in : in std_logic;
        out_bu_0 : out std_logic_vector (15 downto 0);
        out_bu_1 : out std_logic_vector (13 downto 0);
        out_bu_2 : out std_logic_vector (11 downto 0);
        out_bu_3 : out std_logic_vector (9 downto 0);
        data_valid_out : out std_logic
    );
end booth_pipeline0;

-- Begin architecture
architecture rtl of booth_pipeline0 is

    -- Component definitions
    component DFF_6501
        port
        (
            data_in : in std_logic;
            clock : in std_logic;
            reset : in std_logic;
            not_q_out : out std_logic;
            q_out : out std_logic
        );
    end component;

    component operand_register_16bit
        port
        (
            data_in : in std_logic_vector(15 downto 0);
            clock, reset : in std_logic;
            data_out : out std_logic_vector(15 downto 0)
        );
    end component;

    component operand_register_14bit
        port
        (

```

```

        data_in : in std_logic_vector(13 downto 0);
        clock, reset : in std_logic;
        data_out : out std_logic_vector(13 downto 0)
    );
end component;

component operand_register_12bit
port
(
    data_in : in std_logic_vector(11 downto 0);
    clock, reset : in std_logic;
    data_out : out std_logic_vector(11 downto 0)
);
end component;

component operand_register_10bit
port
(
    data_in : in std_logic_vector(9 downto 0);
    clock, reset : in std_logic;
    data_out : out std_logic_vector(9 downto 0)
);
end component;

component booth_unit_0
port
(
    data_in : in std_logic_vector (7 downto 0);
    booth_control : in std_logic_vector (2 downto 0);
    data_out : out std_logic_vector (15 downto 0)
);
end component;

component booth_unit_1
port
(
    data_in : in std_logic_vector (7 downto 0);
    booth_control : in std_logic_vector (2 downto 0);
    data_out : out std_logic_vector (13 downto 0)
);
end component;

component booth_unit_2
port
(
    data_in : in std_logic_vector (7 downto 0);
    booth_control : in std_logic_vector (2 downto 0);
    data_out : out std_logic_vector (11 downto 0)
);
end component;

component booth_unit_3
port
(
    data_in : in std_logic_vector (7 downto 0);
    booth_control : in std_logic_vector (2 downto 0);
    data_out : out std_logic_vector (9 downto 0)
);

```

```

end component;

-- Signal assignments
-- Booth Control Vector
signal booth_control_vector : std_logic_vector (8 downto 0);
-- Booth Unit output vectors
signal bu_0_val : std_logic_vector (15 downto 0);
signal bu_1_val : std_logic_vector (13 downto 0);
signal bu_2_val : std_logic_vector (11 downto 0);
signal bu_3_val : std_logic_vector (9 downto 0);
-- Data valid not signal
signal not_data_valid_out : std_logic;
begin
    -- Cast Booth Control Vector
    booth_control_vector (8 downto 1) <= operand_b;
    booth_control_vector(0) <= '0';

    -- Generate Booth Unit 0 and its register
    BU0: booth_unit_0 port map (operand_a, booth_control_vector(2 downto 0), bu_0_val);
    REG0: operand_register_16bit port map (bu_0_val, clk, clr, out_bu_0);
    -- Generate Booth Unit 1 and its register
    BU1: booth_unit_1 port map (operand_a, booth_control_vector(4 downto 2), bu_1_val);
    REG1: operand_register_14bit port map (bu_1_val, clk, clr, out_bu_1);
    -- Generate Booth Unit 2 and its register
    BU2: booth_unit_2 port map (operand_a, booth_control_vector(6 downto 4), bu_2_val);
    REG2: operand_register_12bit port map (bu_2_val, clk, clr, out_bu_2);
    -- Generate Booth Unit 3 and its register
    BU3: booth_unit_3 port map (operand_a, booth_control_vector(8 downto 6), bu_3_val);
    REG3: operand_register_10bit port map (bu_3_val, clk, clr, out_bu_3);
    -- Generate valid data flag dff
    VALID0: DFF_6501 port map (data_valid_in, clk, clr, not_data_valid_out,
    => data_valid_out);
end rtl;

```

A.7.3 Pipeline Stage 1

```

-- Booth_8 Pipeline Stage 1
-- Purpose: To store Booth Adder results

-- Library declaration
library ieee;
use ieee.std_logic_1164.all;

-- Begin entity
entity booth_pipeline1 is
    port
    (
        bu_0_in : in std_logic_vector (15 downto 0);
        bu_1_in : in std_logic_vector (13 downto 0);
        bu_2_in : in std_logic_vector (11 downto 0);
        bu_3_in : in std_logic_vector (9 downto 0);
        data_valid_in : in std_logic;
        clk : in std_logic;
        clr : in std_logic;
        result_out : out std_logic_vector (15 downto 0);
        data_valid_out : out std_logic
    );

```



```

end booth_pipeline1;

-- Begin architecture
architecture rtl of booth_pipeline1 is
  -- Instance Booth Adder
  component booth_adder
    port
    (
      data0_in : in std_logic_vector (15 downto 0);
      data1_in : in std_logic_vector (13 downto 0);
      data2_in : in std_logic_vector (11 downto 0);
      data3_in : in std_logic_vector (9 downto 0);
      sum_out  : out std_logic_vector (15 downto 0)
    );
  end component;
  -- Instance result register
  component operand_register_16bit
    port
    (
      data_in : in std_logic_vector(15 downto 0);
      clock, reset : in std_logic;
      data_out : out std_logic_vector(15 downto 0)
    );
  end component;
  -- Instance DFF
  component dff_6501
    port
    (
      data_in : in std_logic;
      clock : in std_logic;
      reset : in std_logic;
      not_q_out : out std_logic;
      q_out : out std_logic
    );
  end component;
  -- Signal for negated data valid out
  signal not_data_valid_out : std_logic;
  -- Vector for result of add
  signal booth_adder_result : std_logic_vector (15 downto 0);
begin
  -- Create Booth Adder
  BA: booth_adder port map (bu_0_in, bu_1_in, bu_2_in, bu_3_in, booth_adder_result);
  -- Create result register
  REG0: operand_register_16bit port map (booth_adder_result, clk, clr, result_out);
  -- Create register for valid data
  REG1: dff_6501 port map (data_valid_in, clk, clr, not_data_valid_out,
    => data_valid_out);
end rtl;

```

B Test Scripts

In this section, the test scripts are presented. These are .do files which are executed in Mentor Graphics' ModelSim. By using these scripts, all the simulations can be simply automated by executing one instruction in the command-line interface rather than manually forcing every value and execution.

B.1 Logic Gates

B.1.1 AND2

```
# Compile and load module
vcom -reportprogress 300 -work work
↳ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/gates/AND2_6501.vhd
vsim work.and2_6501
# Add all signals to wave
add wave sim:/and2_6501/*

# Combinational Test
force in_A 0
force in_B 0
run 1
force in_A 0
force in_B 1
run 1
force in_A 1
force in_B 0
run 1
force in_A 1
force in_B 1
run 1
```

B.1.2 NAND2

```
# Compile and load module
vcom -reportprogress 300 -work work
↳ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/gates/NAND2_6501.vhd
vsim work.nand2_6501

# Add signals to wave
add wave sim:/nand2_6501/*

# Combinational test
force in_A 0
force in_B 0
run 2
force in_A 0
force in_B 1
run 2
force in_A 1
force in_B 0
run 2
force in_A 1
force in_B 1
run 2
```

B.1.3 NAND3

```
# Compile and load module
vcom -reportprogress 300 -work work
↪ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/gates/NAND3_6501.vhd
vsim work.nand3_6501

# Add signals to wave
add wave sim:/nand3_6501/*

# Combinational test
force in_A 0
force in_B 0
force in_C 0
run 2
force in_A 0
force in_B 0
force in_C 1
run 2
force in_A 0
force in_B 1
force in_C 0
run 2
force in_A 0
force in_B 0
force in_C 1
run 2
force in_A 0
force in_B 1
force in_C 1
run 2
force in_A 1
force in_B 0
force in_C 0
run 2
force in_A 1
force in_B 0
force in_C 1
run 2
force in_A 1
force in_B 1
force in_C 0
run 2
force in_A 1
force in_B 1
force in_C 1
run 2
```

B.1.4 NOT

```
# Compile and load module
vcom -reportprogress 300 -work work
↪ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/gates/NOT_6501.vhd
vsim work.not_6501

# Add signals to wave
add wave sim:/not_6501/*
```

```
# Combinational test
force in_A 0
run 2
force in_A 1
run 2
```

B.1.5 OR2

```
# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/gates/OR2_6501.vhd
vsim work.or2_6501
```

```
# Add signals to wave
add wave sim:/or2_6501/*
```

```
# Combinational test
force in_A 0
force in_B 0
run 2
force in_A 0
force in_B 1
run 2
force in_A 1
force in_B 0
run 2
force in_A 1
force in_B 1
run 2
```

B.1.6 XOR2

```
# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/pipeline/XOR2_6501.vhd
vsim work.xor2_6501
```

```
# Add to wave
add wave sim:/xor2_6501/*
```

```
# Combinational test
force in_A 0
force in_B 0
run 2
force in_A 0
force in_B 1
run 2
force in_A 1
force in_B 0
run 2
force in_A 1
force in_B 1
run 2
```

B.2 Small Functional Blocks

B.2.1 D Flip Flop with Active Low Reset and Positive Edge Clock

```
# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/small_blocks/dff_6501.vhd
vsim work.dff_6501

# Add signals to wave
add wave sim:/dff_6501/*

# Test reset
force data_in 1
force reset 0
force clock 0
run 2
force clock 1
run 2
force clock 0
run 2

# Test set
force reset 1
run 2
force clock 1
run 2
force clock 0
run 2

# Test reset (not-async)
force data_in 0
force clock 1
run 2
force clock 0
run 2
force clock 1
run 2
force clock 0
run 2
```

B.2.2 Full Adder

```
# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/small_blocks/fulladder.vhd
vsim work.fulladder

# Add signals to wave
add wave sim:/fulladder/*

# Combinational test
force in_A 0
force in_B 0
force in_C 0
run 2
force in_A 0
force in_B 0
```

```

force in_C 1
run 2
force in_A 0
force in_B 1
force in_C 0
run 2
force in_A 0
force in_B 0
force in_C 1
run 2
force in_A 0
force in_B 1
force in_C 1
run 2
force in_A 1
force in_B 0
force in_C 0
run 2
force in_A 1
force in_B 0
force in_C 1
run 2
force in_A 1
force in_B 1
force in_C 0
run 2
force in_A 1
force in_B 1
force in_C 1
run 2

```

B.2.3 Half Adder

```

# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/small_blocks/halfadd.vhd
vsim work.halfadd

# Add signals to wave
add wave sim:/halfadd/*

# Combinational test
force in_A 0
force in_B 0
run 2
force in_A 0
force in_B 1
run 2
force in_A 1
force in_B 0
run 2
force in_A 1
force in_B 1
run 2

```

B.2.4 2 To 1 Multiplexer

```
# Compile and load module
vcom -reportprogress 300 -work work
↪ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/small_blocks/two_to_one_mux.vhd
vsim work.two_to_one_mux

# Add signals to wave
add wave sim:/two_to_one_mux/a
add wave sim:/two_to_one_mux/b
add wave sim:/two_to_one_mux/sel
add wave sim:/two_to_one_mux/y

# Test cases for a
force a 0
force b 1
force sel 0
run 2
force a 1
force b 1
force sel 0
run 2

# Test cases for b
force a 1
force b 0
force sel 1
run 2
force a 1
force b 1
force sel 1
run 2
```

B.3 Adders

B.3.1 2 Bit Carry Select Block

```
# Compile and load module
vcom -reportprogress 300 -work work
↪ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/adders/csa_2bit_block.vhd
vsim work.csa_2bit_block

# Add signals to wave
add wave sim:/csa_2bit_block/in_a
add wave sim:/csa_2bit_block/in_b
add wave sim:/csa_2bit_block/mux_select
add wave sim:/csa_2bit_block/sum
add wave sim:/csa_2bit_block/carryout

# General purpose test
force in_a 01
force in_b 01
force mux_select 0
run 2

force in_a 01
force in_b 01
force mux_select 1
```

```

run 2

# Overflow test
force in_a 10
force in_b 01
force mux_select 0
run 2

force in_a 10
force in_b 01
force mux_select 1
run 2

# Zero test
force in_a 00
force in_b 00
force mux_select 0
run 2

force in_a 00
force in_b 00
force mux_select 1
run 2

```

B.3.2 4 Bit Carry Select Block

```

# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/adders/csa_4bit_block.vhd
vsim work.csa_4bit_block

# Add signals to wave
add wave sim:/csa_4bit_block/in_a
add wave sim:/csa_4bit_block/in_b
add wave sim:/csa_4bit_block/mux_select
add wave sim:/csa_4bit_block/sum
add wave sim:/csa_4bit_block/carryout

# General purpose test
force in_a 0111
force in_b 0111
force mux_select 0
run 2

force in_a 0111
force in_b 0111
force mux_select 1
run 2

# Overflow test
force in_a 1000
force in_b 0111
force mux_select 0
run 2

force in_a 1000
force in_b 0111
force mux_select 1

```



```

run 2

# Zero test
force in_a 0000
force in_b 0000
force mux_select 0
run 2

force in_a 0000
force in_b 0000
force mux_select 1
run 2

```

B.3.3 10 Bit Carry Select Adder

```

# Compile and load module
vcom -reportprogress 300 -work work
↪ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/adders/csa_10bit.vhd
vsim work.csa_10bit

# Add signals to wave
add wave sim:/csa_10bit/in_a
add wave sim:/csa_10bit/in_b
add wave sim:/csa_10bit/carryout
add wave sim:/csa_10bit/sum

# General purpose test
force in_a 1101101111
force in_b 0000101110

run 2

force in_a 0111011100
force in_b 0011000110
run 2

# Overflow test
force in_a 1000000000
force in_b 0111111111
run 2

force in_a 1111110100
force in_b 0111100100
run 2

# Zero test
force in_a 0000000000
force in_b 0000000000
force mux_select 0
run 2

```

B.3.4 12 Bit Carry Select Adder

```

# Compile and load module
vcom -reportprogress 300 -work work
↪ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/adders/csa_12bit.vhd

```

```

vsim work.csa_12bit

# Add signals to wave
add wave sim:/csa_12bit/in_a
add wave sim:/csa_12bit/in_b
add wave sim:/csa_12bit/carryout
add wave sim:/csa_12bit/sum

# General purpose test
force in_a 110110111111
force in_b 000010111011

run 2

force in_a 011101110000
force in_b 001100011000
run 2

# Overflow test
force in_a 100000000000
force in_b 011111111111
run 2

force in_a 111111010000
force in_b 011110010000
run 2

# Zero test
force in_a 000000000000
force in_b 000000000000
force mux_select 0
run 2

```

B.3.5 14 Bit Carry Select Adder

```

# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/adders/csa_14bit.vhd
vsim work.csa_14bit

# Add signals to wave
add wave sim:/csa_14bit/in_a
add wave sim:/csa_14bit/in_b
add wave sim:/csa_14bit/carryout
add wave sim:/csa_14bit/sum

# General purpose test
force in_a 11011011111111
force in_b 00001011101111

run 2

force in_a 01110111000000
force in_b 00110001100000
run 2

```

```

# Overflow test
force in_a 10000000000000
force in_b 011111111111111
run 2

force in_a 11111101000000
force in_b 01111001000000
run 2

# Zero test
force in_a 0000000000000000
force in_b 0000000000000000
force mux_select 0
run 2

```

B.3.6 9 Bit Incrementer

```

# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/adders/increment_9bit.vhd
vsim work.increment_9bit

# Add signals to wave
add wave sim:/increment_9bit/value_in -radix unsigned
add wave sim:/increment_9bit/carry_in
add wave sim:/increment_9bit/value_out -radix unsigned
add wave sim:/increment_9bit/carryout

# Test values
force value_in 110001111
force carry_in 1
run 2
force value_in 101111110
force carry_in 0
run 2
force value_in 111111111
force carry_in 1
run 2
force value_in 000000000
force carry_in 1
run 2

```

B.3.7 4 Bit Ripple Carry Adder Block

```

# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/adders/rca_4bit_block.vhd
vsim work.rca_4bit_block

# Add signals to wave
add wave sim:/rca_4bit_block/in_a
add wave sim:/rca_4bit_block/in_b
add wave sim:/rca_4bit_block/in_c
add wave sim:/rca_4bit_block/sum
add wave sim:/rca_4bit_block/carry_out

# General purpose test

```

```
force in_a 0011
force in_b 0111
force in_c 0
run 2
```

```
force in_a 0011
force in_b 0111
force in_c 1
run 2
```

```
# Overflow test
force in_a 1000
force in_b 0111
force in_c 0
run 2
```

```
force in_a 1000
force in_b 0111
force in_c 1
run 2
```

```
# Zero test
force in_a 0000
force in_b 0000
force in_c 0
run 2
```

```
force in_a 0000
force in_b 0000
force in_c 1
run 2
```

B.4 Higher Level Functional Blocks

B.4.1 Booth Adder, Device Under Test

```
# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/func_blocks/ba_dut.vhd
vsim work.ba_dut
```

```
# Add signals to wave
add wave sim:/ba_dut/in_a
add wave sim:/ba_dut/in_b
add wave sim:/ba_dut/sum_out
```

```
# Test 0: 127 x 0 = 0
force in_a 01111111
force in_b 00000000
run 1
```

```
# Test 1: 127 x 127 = 16129
force in_a 01111111
force in_b 01111111
run 1
```

```
# Test 2: -128 x 127 = -16256
force in_a 10000000
```

```

force in_b 01111111
run 1

# Test 3: -128 x -1 = 128
force in_b 11111111
run 1

# Test 4: -56 x -91 = 5096
force in_a 11001000
force in_b 10100101
run 1

# Test 5: -1 x -1 = 1
force in_a 11111111
force in_b 11111111
run 1

# Test 6: -2 x -1 = 2
force in_a 11111110
force in_b 11111111
run 1

# Test 7: -24 x -12 = 288
force in_a 11101000
force in_b 11110100
run 1

# Test 8: 107 x -96 = 10272
force in_a 01101011
force in_b 01100000
run 1

# Test 9: -3 x 127 = -381
force in_a 11111101
force in_b 01111111
run 1

```

B.4.2 9 Bit Barrel Shifter

```

# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/func_blocks/barrel_shift_9bit.vhd
vsim work.barrel_shift_9bit

# Add signals to wave
add wave sim:/barrel_shift_9bit/data_in
add wave sim:/barrel_shift_9bit/ctrl
add wave sim:/barrel_shift_9bit/data_out

# Do 0'ing out
force data_in 111000101
force ctrl 00
run 1

# Perform no shift
force data_in 111000101
force ctrl 01
run 1

# Perform double

```

```

force data_in 111000101
force ctrl 10
run 1
# Perform double again
force data_in 111000101
force ctrl 11
run 1
# Do 0'ing out
force data_in 011011010
force ctrl 00
run 1
# Perform no shift
force data_in 011011010
force ctrl 01
run 1
# Perform double
force data_in 011011010
force ctrl 10
run 1
# Perform double again
force data_in 011011010
force ctrl 11
run 1
# Do 0'ing out
force data_in 000000001
force ctrl 00
run 1
# Perform no shift
force data_in 000000001
force ctrl 01
run 1
# Perform double
force data_in 000000001
force ctrl 10
run 1
# Perform double again
force data_in 000000001
force ctrl 11
run 1

```

B.4.3 Booth Adder

```

# Compile and load module
vcom -reportprogress 300 -work work
↪ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/func_blocks/booth_adder.vhd
vsim work.booth_adder

# Add signals to wave
add wave sim:/booth_adder/data0_in
add wave sim:/booth_adder/data1_in
add wave sim:/booth_adder/data2_in
add wave sim:/booth_adder/data3_in
add wave sim:/booth_adder/sum_out

# Test 0: 127 x 0 = 0
force data0_in 0000000000000000
force data1_in 0000000000000000

```

```

force data2_in 000000000000
force data3_in 0000000000
run 1

# Test 1: 127 x 127 = 16129
force data0_in 1111111110000001
force data1_in 0000000000000000
force data2_in 0000000000000000
force data3_in 00111111110
run 1

# Test 2: -128 x -128 = -16384
force data0_in 0000000000000000
force data1_in 0000000000000000
force data2_in 0000000000000000
force data3_in 11000000000
run 1

# Test 3: -65 x 83 = 5229
force data0_in 0000000001000001
force data1_in 1111110111111111
force data2_in 1111101111111111
force data3_in 1110111111111111
run 1

```

B.4.4 Booth Decoder

```

# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/func_blocks/booth_decoder.vhd
vsim work.booth_decoder
# Add signals to wave
add wave sim:/booth_decoder/data_in
add wave sim:/booth_decoder/ctrl

# Combinational Logic Test
force data_in 000
run 1
force data_in 001
run 1
force data_in 010
run 1
force data_in 011
run 1
force data_in 100
run 1
force data_in 101
run 1
force data_in 110
run 1
force data_in 111
run 1

```

B.4.5 Booth Unit 0 (16 Bit)

```

# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/func_blocks/booth_unit_0.vhd

```

```

vsim work.booth_unit_0
# Add signals to wave
add wave sim:/booth_unit_0/data_in
add wave sim:/booth_unit_0/booth_control
add wave sim:/booth_unit_0/data_out

# Zeros Test
force data_in 00000000
force booth_control 000
run 2
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
run 2
force booth_control 110
run 2
force booth_control 111
run 2

# Negative Overflow Test
force data_in 11111111
force booth_control 000
run 2
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
run 2
force booth_control 110
run 2
force booth_control 111
run 2

# Positive Overflow Test
force data_in 01111111
force booth_control 000
run 2
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
run 2

```



```
force booth_control 110
run 2
force booth_control 111
run 2
```

B.4.6 Booth Unit 1 (14 Bit)

```
# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/func_blocks/booth_unit_1.vhd
vsim work.booth_unit_1
# Add signals to wave
add wave sim:/booth_unit_1/data_in
add wave sim:/booth_unit_1/booth_control
add wave sim:/booth_unit_1/data_out

# Zeros Test
force data_in 00000000
force booth_control 000
run 2
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
run 2
force booth_control 110
run 2
force booth_control 111
run 2

# Negative Overflow Test
force data_in 11111111
force booth_control 000
run 2
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
run 2
force booth_control 110
run 2
force booth_control 111
run 2

# Positive Overflow Test
force data_in 01111111
force booth_control 000
run 2
```

```
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
run 2
force booth_control 110
run 2
force booth_control 111
run 2
```

B.4.7 Booth Unit 2 (12 Bit)

```
# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/func_blocks/booth_unit_2.vhd
vsim work.booth_unit_2
# Add signals to wave
add wave sim:/booth_unit_2/data_in
add wave sim:/booth_unit_2/booth_control
add wave sim:/booth_unit_2/data_out

# Zeros Test
force data_in 00000000
force booth_control 000
run 2
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
run 2
force booth_control 110
run 2
force booth_control 111
run 2

# Negative Overflow Test
force data_in 11111111
force booth_control 000
run 2
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
```

```

run 2
force booth_control 110
run 2
force booth_control 111
run 2

# Positive Overflow Test
force data_in 01111111
force booth_control 000
run 2
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
run 2
force booth_control 110
run 2
force booth_control 111
run 2

```

B.4.8 Booth Unit 3 (10 Bit)

```

# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/func_blocks/booth_unit_3.vhd
vsim work.booth_unit_3

# Add signals to wave
add wave sim:/booth_unit_3/data_in
add wave sim:/booth_unit_3/booth_control
add wave sim:/booth_unit_3/data_out

# Zeros Test
force data_in 00000000
force booth_control 000
run 2
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
run 2
force booth_control 110
run 2
force booth_control 111
run 2

# Negative Overflow Test
force data_in 11111111

```

```

force booth_control 000
run 2
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
run 2
force booth_control 110
run 2
force booth_control 111
run 2

# Positive Overflow Test
force data_in 01111111
force booth_control 000
run 2
force booth_control 001
run 2
force booth_control 010
run 2
force booth_control 011
run 2
force booth_control 100
run 2
force booth_control 101
run 2
force booth_control 110
run 2
force booth_control 111
run 2

```

B.4.9 9 Bit Two's Complementer

```

# Compile and load module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/func_blocks/twoscomp_9bit.vhd
vsim work.twoscomp_9bit

# Add signals to wave
add wave sim:/twoscomp_9bit/data_in
add wave sim:/twoscomp_9bit/enable
add wave sim:/twoscomp_9bit/data_out

# Test case 0: Complement operand
force data_in 10011110
force enable 1
run 2

force enable 0
run 2

# Test case 1: Complement operand
force data_in 01010110

```

```

force enable 1
run 2

force enable 0
run 2

# Test case 2: Complement operand
force data_in 11111111
force enable 1
run 2

force enable 0
run 2

```

B.5 Registers

B.5.1 8 Bit Operand Register

```

# Compile and load the module
vcom -reportprogress 300 -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/registers/operand_register.vhd
vsim work.operand_register

# Add signals to wave
add wave sim:/operand_register/clock
add wave sim:/operand_register/data_in
add wave sim:/operand_register/reset
add wave sim:/operand_register/data_out

# Load Register Test: Reset not active, simple data load
force clock 0
force data_in 10101010
force reset 0
run 2
force clock 1
run 2
force clock 0
run 2

# Asynchronous Reset Test
force reset 1
run 2
force clock 1
run 2
force clock 0
run 2

# Clock indifference test
force reset 0
force data_in 11110000
run 2

# Now just load value
force clock 1
run 2
force clock 0
run 2

```

B.6 Non-Pipelined Implementation

```
# Compile and load module
vcom -work work
↪ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/impl/non-pipelined/booth_8.vhd
vsim work.booth_8

# Add signals to wave
add wave sim:/booth_8/operand_a
add wave sim:/booth_8/operand_b
add wave sim:/booth_8/clr
add wave sim:/booth_8/clock
add wave sim:/booth_8/load
add wave sim:/booth_8/end_flag
add wave sim:/booth_8/z

# Do initial reset
force clr 1
run 1
force clr 0
run 1

# Test 1: 127 x 127 = 16129
force operand_a 01111111
force operand_b 01111111
force load 0
run 1
force load 1
run 1
force load 0
run 1
force clk 0
run 1
force clk 1
run 1
force clk 0
run 1

# Asynchronous Clear
force clr 1
run 1
force clr 0
run 1

# Test 2: -128 x -128 = -16384
force operand_a 10000000
force operand_b 10000000
force load 0
run 1
force load 1
run 1
force load 0
run 1
force clk 0
run 1
force clk 1
run 1
force clk 0
```

```

run 1

# Test 3: -65 x 83 = 5229
force operand_a 10111111
force operand_b 01010011
force load 0
run 1
force load 1
run 1
force load 0
run 1
force clk 0
run 1
force clk 1
run 1
force clk 0
run 1

# Test 4: 107 x -96 = 10272
force operand_a 01101011
force operand_b 01100000
force load 0
run 1
force load 1
run 1
force load 0
run 1
force clk 0
run 1
force clk 1
run 1
force clk 0
run 1

# Test 5: -3 x 127 = -381
force operand_a 11111101
force operand_b 01111111
force load 0
run 1
force load 1
run 1
force load 0
run 1
force clk 0
run 1
force clk 1
run 1
force clk 0
run 1

# Test 6: END Flag
force load 1
run 1
force clk 1
run 1
force clk 0
run 1

```

```
force load 0
run 1
```

B.7 Pipelined Implementation

B.7.1 Booth8 Pipeline Stage 0

```
# Compile and load module
vcom -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/impl/pipeline/booth_pipeline0.vhd
vsim work.booth_pipeline0

# Add signals to wave
add wave sim:/booth_pipeline0/operand_a
add wave sim:/booth_pipeline0/operand_b
add wave sim:/booth_pipeline0/clr
add wave sim:/booth_pipeline0/clk
add wave sim:/booth_pipeline0/data_valid_in
add wave sim:/booth_pipeline0/data_valid_out
add wave sim:/booth_pipeline0/out_bu_0
add wave sim:/booth_pipeline0/out_bu_1
add wave sim:/booth_pipeline0/out_bu_2
add wave sim:/booth_pipeline0/out_bu_3

# Do initial reset
force data_valid_in 1
force clr 0
run 1
force clr 1
run 1

# Test 1
force operand_a 01111111
force operand_b 01111111
force clk 0
run 1
force clk 1
run 1
force clk 0
run 1

# Test 2 Asynchronous Clear
force clr 0
run 1
force clr 1
run 1
force operand_a 10000000
force operand_b 10000000
force clk 1
run 1
force clk 0

# Test 3: Load after Asynchronous Clear
force operand_a 10111111
force operand_b 01010011
force clk 1
run 1
```



```
force clk 0
run 1
```

B.7.2 Booth8 Pipeline Stage 1

```
# Compile and load module
vcom -work work
↳ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/impl/pipeline/booth_pipeline1.vhd
vsim work.booth_pipeline1
```

```
# Add signals to wave
add wave sim:/booth_pipeline1/bu_0_in
add wave sim:/booth_pipeline1/bu_1_in
add wave sim:/booth_pipeline1/bu_2_in
add wave sim:/booth_pipeline1/bu_3_in
add wave sim:/booth_pipeline1/clr
add wave sim:/booth_pipeline1/clk
add wave sim:/booth_pipeline1/data_valid_in
add wave sim:/booth_pipeline1/data_valid_out
add wave sim:/booth_pipeline1/result_out
```

```
# Do initial reset
force data_valid_in 1
force clr 0
run 1
force clr 1
run 1
```

```
# Test 0: 127 x 0 = 0
force bu_0_in 0000000000000000
force bu_1_in 0000000000000000
force bu_2_in 0000000000000000
force bu_3_in 0000000000
run 1
force clk 0
run 1
force clk 1
run 1
```

```
# Test 1: 127 x 127 = 16129 with asynchronous reset
force bu_0_in 1111111110000001
force bu_1_in 0000000000000000
force bu_2_in 0000000000000000
force bu_3_in 0011111110
run 1
force clk 0
run 1
force clk 1
run 1
force clr 0
run 1
force clk 0
run 1
force clk 1
run 1
force clr 1
run 1
```

```

# Test 2: -128 x -128 = 16384
force bu_0_in 0000000000000000
force bu_1_in 0000000000000000
force bu_2_in 0000000000000000
force bu_3_in 010000000000
run 1
force clk 0
run 1
force clk 1
run 1

```

```

# Test 3: -65 x 83 = -5395
force bu_0_in 0000000001000001
force bu_1_in 1111111011111111
force bu_2_in 1111101111111111
force bu_3_in 1110111111111111
run 1
force clk 0
run 1
force clk 1
run 1

```

B.7.3 Booth8 Pipeline Top Level Implementation

```

# Compile and load module
vcom -work work
→ E:/UserData/Dropbox/Concordia/Graduate/COEN6501/Project/src/impl/pipeline/booth_8_pipelined.vhd
vsim work.booth_8_pipelined

# Add signals to wave
# Add divider
add wave -noupdate -divider -height 32 Inputs
add wave sim:/booth_8_pipelined/operand_a
add wave sim:/booth_8_pipelined/operand_b
add wave sim:/booth_8_pipelined/clr
add wave sim:/booth_8_pipelined/clk
add wave sim:/booth_8_pipelined/load
# Add divider
add wave -noupdate -divider -height 32 Pipeline
add wave sim:/booth_8_pipelined/mult_operand_a
add wave sim:/booth_8_pipelined/mult_operand_b
add wave sim:/booth_8_pipelined/p0_out0
add wave sim:/booth_8_pipelined/p0_out1
add wave sim:/booth_8_pipelined/p0_out2
add wave sim:/booth_8_pipelined/p0_out3
add wave sim:/booth_8_pipelined/p1_result
add wave sim:/booth_8_pipelined/valid_data
# Add divider
add wave -noupdate -divider -height 32 Outputs
add wave sim:/booth_8_pipelined/end_flag
add wave sim:/booth_8_pipelined/z

# Do initial reset
force clr 1
run 1
force clr 0
run 1

```

```
# Test 1: 127 x 127 = 16129
```

```
force operand_a 01111111
```

```
force operand_b 01111111
```

```
force load 0
```

```
run 1
```

```
force load 1
```

```
run 1
```

```
force load 0
```

```
run 1
```

```
force clk 0
```

```
run 1
```

```
force clk 1
```

```
run 1
```

```
force clk 0
```

```
run 1
```

```
# Asynchronous Clear
```

```
force clr 1
```

```
run 1
```

```
force clr 0
```

```
run 1
```

```
# Test 2: -128 x -128 = 16384
```

```
force operand_a 10000000
```

```
force operand_b 10000000
```

```
force load 0
```

```
run 1
```

```
force load 1
```

```
run 1
```

```
force load 0
```

```
run 1
```

```
force clk 0
```

```
run 1
```

```
force clk 1
```

```
run 1
```

```
force clk 0
```

```
run 1
```

```
# Test 3: -65 x 83 = -5395
```

```
force operand_a 10111111
```

```
force operand_b 01010011
```

```
force load 0
```

```
run 1
```

```
force load 1
```

```
run 1
```

```
force load 0
```

```
run 1
```

```
force clk 0
```

```
run 1
```

```
force clk 1
```

```
run 1
```

```
force clk 0
```

```
run 1
```

```
# Pass through the pipe
```

```
force clk 1
```

```

run 1
force clk 0
run 1
force clk 1
run 1
force clk 0
run 1
force clk 1
run 1
force clk 0
run 1

```

C Synthesis and Timing Reports

C.1 Non-Pipelined Implementation

C.1.1 PrecisionRTL Area Report

```

*****
Device Utilization for 2VP30ff896
*****

```

Resource	Used	Avail	Utilization
IOs	36	556	6.47%
Global Buffers	0	16	0.00%
LUTs	326	27392	1.19%
CLB Slices	163	13696	1.19%
Dffs or Latches	0	29060	0.00%
Block RAMs	0	136	0.00%
Block Multipliers	0	136	0.00%
Block Multiplier Dffs	0	4896	0.00%
GT_CUSTOM	0	8	0.00%

```

*****

```

```

Library: work    Cell: booth_8    View: rtl

```

```

*****

```

Cell	Library	References	Total Area
IBUF	xcv2p	19 x	
LUT4	xcv2p	3 x	1 3 LUTs
OBUF	xcv2p	17 x	
booth_unit_0	work	1 x	26 gates 25 LUTs
booth_unit_1	work	1 x	28 gates 28 LUTs
booth_unit_2	work	1 x	28 gates 28 LUTs
booth_unit_3	work	1 x	28 gates 28 LUTs
csa_10bit	work	1 x	25 gates 25 LUTs
csa_12bit	work	1 x	35 gates

```

csa_14bit          work      1 x    35    35 LUTs
                  work      1 x    27    27 gates
                  work      1 x    24    24 LUTs
operand_register   work      1 x    33    33 gates
                  work      1 x    33    33 LUTs
operand_register_16bit work    1 x    65    65 gates
                  work      1 x    65    65 LUTs
operand_register_unfolded0 work    1 x    32    32 gates
                  work      1 x    32    32 LUTs

```

```

Number of ports :      36
Number of nets :     164
Number of instances :   49
Number of references to this view : 0

```

```

Total accumulated area :
Number of LUTs :      326
Number of gates :     332
Number of accumulated instances : 362

```

```

*****
IO Register Mapping Report
*****
Design: work.booth_8.rtl

```

Port	Direction	INFF	OUTFF	TRIFF
operand_a(7)	Input			
operand_a(6)	Input			
operand_a(5)	Input			
operand_a(4)	Input			
operand_a(3)	Input			
operand_a(2)	Input			
operand_a(1)	Input			
operand_a(0)	Input			
operand_b(7)	Input			
operand_b(6)	Input			
operand_b(5)	Input			
operand_b(4)	Input			
operand_b(3)	Input			
operand_b(2)	Input			
operand_b(1)	Input			

operand_b(0)	Input				
clr	Input				
clk	Input				
load	Input				
z(15)	Output				
z(14)	Output				
z(13)	Output				
z(12)	Output				
z(11)	Output				
z(10)	Output				
z(9)	Output				
z(8)	Output				
z(7)	Output				
z(6)	Output				
z(5)	Output				
z(4)	Output				
z(3)	Output				
z(2)	Output				
z(1)	Output				
z(0)	Output				
end_flag	Output				

Total registers mapped: 0

C.1.2 Xilinx ISE Timing Report

Release 10.1 Trace (lin64)

Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.

```
/nfs/sw_cmc/linux-64/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce -ise
/nfs/home/r/r_fenste/COEN6501/synth_test/np/ise/booth_8/booth_8.ise -intstyle
ise -e 3 -s 7 -xml booth_8 booth_8.ncd -o booth_8.twr booth_8.pcf -ucf
booth_8.ucf
```

Design file: booth_8.ncd

Physical constraint file: booth_8.pcf

Device,package,speed: xc2vp30,ff896,-7 (PRODUCTION 1.94 2008-01-09)
Report level: error report

Environment Variable	Effect
NONE	No environment variables were set

INFO:Timing:2698 - No timing constraints found, doing default enumeration.
INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths option. All paths that are not constrained will be reported in the unconstrained paths section(s) of the report.
INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on a 50 Ohm transmission line loading model. For the details of this model, and for more information on accounting for different loading conditions, please see the device datasheet.

Data Sheet report:

All values displayed in nanoseconds (ns)

Pad to Pad

Source Pad	Destination Pad	Delay
clk	end_flag	6.861
clk	z(0)	5.300
clk	z(1)	4.920
clk	z(2)	5.951
clk	z(3)	6.310
clk	z(4)	7.122
clk	z(5)	6.817
clk	z(6)	7.016
clk	z(7)	7.485
clk	z(8)	8.949
clk	z(9)	7.963
clk	z(10)	7.750
clk	z(11)	7.163
clk	z(12)	7.260
clk	z(13)	7.647
clk	z(14)	8.079
clk	z(15)	6.786
clr	end_flag	6.710
clr	z(0)	8.007
clr	z(1)	8.925
clr	z(2)	10.712
clr	z(3)	11.585
clr	z(4)	12.889
clr	z(5)	13.375
clr	z(6)	13.689
clr	z(7)	14.438
clr	z(8)	15.712
clr	z(9)	15.189
clr	z(10)	16.774
clr	z(11)	16.736
clr	z(12)	17.509

clr	z(13)		17.660
clr	z(14)		17.742
clr	z(15)		16.857
load	end_flag		6.408
load	z(0)		7.846
load	z(1)		8.801
load	z(2)		10.538
load	z(3)		11.748
load	z(4)		13.052
load	z(5)		13.538
load	z(6)		13.852
load	z(7)		14.601
load	z(8)		15.875
load	z(9)		15.352
load	z(10)		16.937
load	z(11)		16.899
load	z(12)		17.672
load	z(13)		17.823
load	z(14)		17.905
load	z(15)		17.020
operand_a(0)	z(0)		6.915
operand_a(0)	z(1)		7.381
operand_a(0)	z(2)		8.296
operand_a(0)	z(3)		9.424
operand_a(0)	z(4)		10.728
operand_a(0)	z(5)		11.214
operand_a(0)	z(6)		11.528
operand_a(0)	z(7)		12.277
operand_a(0)	z(8)		13.551
operand_a(0)	z(9)		13.028
operand_a(0)	z(10)		14.613
operand_a(0)	z(11)		14.575
operand_a(0)	z(12)		15.348
operand_a(0)	z(13)		15.499
operand_a(0)	z(14)		15.581
operand_a(0)	z(15)		14.696
operand_a(1)	z(1)		7.758
operand_a(1)	z(2)		9.545
operand_a(1)	z(3)		9.846
operand_a(1)	z(4)		11.182
operand_a(1)	z(5)		11.668
operand_a(1)	z(6)		11.982
operand_a(1)	z(7)		12.778
operand_a(1)	z(8)		14.052
operand_a(1)	z(9)		13.529
operand_a(1)	z(10)		15.458
operand_a(1)	z(11)		15.420
operand_a(1)	z(12)		16.193
operand_a(1)	z(13)		16.344
operand_a(1)	z(14)		16.426
operand_a(1)	z(15)		15.541
operand_a(2)	z(2)		8.043
operand_a(2)	z(3)		8.541
operand_a(2)	z(4)		9.852
operand_a(2)	z(5)		10.338
operand_a(2)	z(6)		10.652
operand_a(2)	z(7)		11.401
operand_a(2)	z(8)		12.675

operand_a(2)	z(9)		12.152
operand_a(2)	z(10)		13.737
operand_a(2)	z(11)		13.699
operand_a(2)	z(12)		14.472
operand_a(2)	z(13)		14.623
operand_a(2)	z(14)		14.705
operand_a(2)	z(15)		13.820
operand_a(3)	z(3)		9.695
operand_a(3)	z(4)		11.006
operand_a(3)	z(5)		11.492
operand_a(3)	z(6)		11.806
operand_a(3)	z(7)		12.555
operand_a(3)	z(8)		13.829
operand_a(3)	z(9)		13.306
operand_a(3)	z(10)		14.891
operand_a(3)	z(11)		14.853
operand_a(3)	z(12)		15.626
operand_a(3)	z(13)		15.777
operand_a(3)	z(14)		15.859
operand_a(3)	z(15)		14.974
operand_a(4)	z(4)		8.743
operand_a(4)	z(5)		9.379
operand_a(4)	z(6)		9.681
operand_a(4)	z(7)		10.597
operand_a(4)	z(8)		11.871
operand_a(4)	z(9)		11.384
operand_a(4)	z(10)		13.208
operand_a(4)	z(11)		13.170
operand_a(4)	z(12)		13.943
operand_a(4)	z(13)		14.094
operand_a(4)	z(14)		14.176
operand_a(4)	z(15)		13.291
operand_a(5)	z(5)		9.761
operand_a(5)	z(6)		10.063
operand_a(5)	z(7)		11.193
operand_a(5)	z(8)		12.467
operand_a(5)	z(9)		11.944
operand_a(5)	z(10)		13.718
operand_a(5)	z(11)		13.680
operand_a(5)	z(12)		14.453
operand_a(5)	z(13)		14.604
operand_a(5)	z(14)		14.686
operand_a(5)	z(15)		13.801
operand_a(6)	z(6)		10.003
operand_a(6)	z(7)		11.461
operand_a(6)	z(8)		12.735
operand_a(6)	z(9)		12.212
operand_a(6)	z(10)		14.141
operand_a(6)	z(11)		14.103
operand_a(6)	z(12)		14.876
operand_a(6)	z(13)		15.027
operand_a(6)	z(14)		15.109
operand_a(6)	z(15)		14.224
operand_a(7)	z(7)		10.896
operand_a(7)	z(8)		12.170
operand_a(7)	z(9)		11.647
operand_a(7)	z(10)		13.209
operand_a(7)	z(11)		13.171

operand_a(7)	z(12)		13.944
operand_a(7)	z(13)		14.095
operand_a(7)	z(14)		14.177
operand_a(7)	z(15)		13.292
operand_b(0)	z(0)		6.126
operand_b(0)	z(1)		6.869
operand_b(0)	z(2)		8.302
operand_b(0)	z(3)		8.660
operand_b(0)	z(4)		9.971
operand_b(0)	z(5)		10.457
operand_b(0)	z(6)		10.771
operand_b(0)	z(7)		11.520
operand_b(0)	z(8)		12.794
operand_b(0)	z(9)		12.271
operand_b(0)	z(10)		13.856
operand_b(0)	z(11)		13.818
operand_b(0)	z(12)		14.591
operand_b(0)	z(13)		14.742
operand_b(0)	z(14)		14.824
operand_b(0)	z(15)		13.939
operand_b(1)	z(1)		7.288
operand_b(1)	z(2)		8.638
operand_b(1)	z(3)		10.235
operand_b(1)	z(4)		11.539
operand_b(1)	z(5)		12.025
operand_b(1)	z(6)		12.339
operand_b(1)	z(7)		13.088
operand_b(1)	z(8)		14.362
operand_b(1)	z(9)		13.839
operand_b(1)	z(10)		15.424
operand_b(1)	z(11)		15.386
operand_b(1)	z(12)		16.159
operand_b(1)	z(13)		16.310
operand_b(1)	z(14)		16.392
operand_b(1)	z(15)		15.507
operand_b(2)	z(2)		8.458
operand_b(2)	z(3)		9.944
operand_b(2)	z(4)		11.248
operand_b(2)	z(5)		11.734
operand_b(2)	z(6)		12.048
operand_b(2)	z(7)		12.797
operand_b(2)	z(8)		14.071
operand_b(2)	z(9)		13.548
operand_b(2)	z(10)		15.133
operand_b(2)	z(11)		15.095
operand_b(2)	z(12)		15.868
operand_b(2)	z(13)		16.019
operand_b(2)	z(14)		16.101
operand_b(2)	z(15)		15.216
operand_b(3)	z(3)		9.741
operand_b(3)	z(4)		11.045
operand_b(3)	z(5)		11.531
operand_b(3)	z(6)		11.845
operand_b(3)	z(7)		12.594
operand_b(3)	z(8)		13.868
operand_b(3)	z(9)		13.345
operand_b(3)	z(10)		14.930
operand_b(3)	z(11)		14.892

operand_b(3)	z(12)		15.665
operand_b(3)	z(13)		15.816
operand_b(3)	z(14)		15.898
operand_b(3)	z(15)		15.013
operand_b(4)	z(4)		7.396
operand_b(4)	z(5)		9.426
operand_b(4)	z(6)		10.069
operand_b(4)	z(7)		10.671
operand_b(4)	z(8)		11.945
operand_b(4)	z(9)		11.422
operand_b(4)	z(10)		11.471
operand_b(4)	z(11)		11.720
operand_b(4)	z(12)		12.988
operand_b(4)	z(13)		13.139
operand_b(4)	z(14)		13.221
operand_b(4)	z(15)		12.336
operand_b(5)	z(5)		8.242
operand_b(5)	z(6)		8.885
operand_b(5)	z(7)		10.072
operand_b(5)	z(8)		11.346
operand_b(5)	z(9)		11.576
operand_b(5)	z(10)		11.927
operand_b(5)	z(11)		12.176
operand_b(5)	z(12)		13.444
operand_b(5)	z(13)		13.595
operand_b(5)	z(14)		13.720
operand_b(5)	z(15)		13.202
operand_b(6)	z(6)		7.878
operand_b(6)	z(7)		9.715
operand_b(6)	z(8)		10.989
operand_b(6)	z(9)		10.466
operand_b(6)	z(10)		10.742
operand_b(6)	z(11)		10.991
operand_b(6)	z(12)		12.259
operand_b(6)	z(13)		12.410
operand_b(6)	z(14)		12.845
operand_b(6)	z(15)		12.327
operand_b(7)	z(7)		9.798
operand_b(7)	z(8)		11.072
operand_b(7)	z(9)		10.549
operand_b(7)	z(10)		10.591
operand_b(7)	z(11)		10.840
operand_b(7)	z(12)		12.108
operand_b(7)	z(13)		12.259
operand_b(7)	z(14)		12.837
operand_b(7)	z(15)		12.319
-----+-----+			

Analysis completed Sun Nov 22 15:44:05 2015

Trace Settings:

Trace Settings

Peak Memory Usage: 300 MB

C.2 Pipelined Implementation

C.2.1 PrecisionRTL Area Report

 Device Utilization for 2VP30ff896

Resource	Used	Avail	Utilization
I/Os	36	556	6.47%
Global Buffers	0	16	0.00%
LUTs	538	27392	1.96%
CLB Slices	269	13696	1.96%
Dffs or Latches	0	29060	0.00%
Block RAMs	0	136	0.00%
Block Multipliers	0	136	0.00%
Block Multiplier Dffs	0	4896	0.00%
GT_CUSTOM	0	8	0.00%

Library: work Cell: booth_8_pipelined View: rtl

Cell	Library	References	Total Area
IBUF	xcv2p	19 x	
LUT4	xcv2p	6 x 1	6 LUTs
OBUF	xcv2p	17 x	
booth_unit_0	work	1 x 26	26 gates
		25	25 LUTs
booth_unit_1	work	1 x 28	28 gates
		28	28 LUTs
booth_unit_2	work	1 x 28	28 gates
		28	28 LUTs
booth_unit_3	work	1 x 28	28 gates
		28	28 LUTs
csa_10bit	work	1 x 25	25 gates
		25	25 LUTs
csa_12bit	work	1 x 35	35 gates
		35	35 LUTs
csa_14bit	work	1 x 42	42 gates
		41	41 LUTs
operand_register	work	1 x 33	33 gates
		33	33 LUTs
operand_register_10bit	work	1 x 39	39 gates
		39	39 LUTs
operand_register_12bit	work	1 x 46	46 gates
		45	45 LUTs
operand_register_14bit	work	1 x 52	52 gates
		51	51 LUTs
operand_register_16bit	work	1 x 61	61 gates
		60	60 LUTs
operand_register_16bit_unfolded0	work	1 x 62	62 gates
		62	62 LUTs
operand_register_unfolded0	work	1 x 32	32 gates

```

Number of ports :          36
Number of nets :          221
Number of instances :      56
Number of references to this view : 0

```

```

Total accumulated area :
Number of LUTs :          538
Number of gates :         545
Number of accumulated instances : 574

```

```

*****
IO Register Mapping Report

```

```

*****

```

```

Design: work.booth_8_pipelined.rtl

```

Port	Direction	INFF	OUTFF	TRIFF
operand_a(7)	Input			
operand_a(6)	Input			
operand_a(5)	Input			
operand_a(4)	Input			
operand_a(3)	Input			
operand_a(2)	Input			
operand_a(1)	Input			
operand_a(0)	Input			
operand_b(7)	Input			
operand_b(6)	Input			
operand_b(5)	Input			
operand_b(4)	Input			
operand_b(3)	Input			
operand_b(2)	Input			
operand_b(1)	Input			
operand_b(0)	Input			
clr	Input			
clk	Input			
load	Input			

z(15)	Output				
z(14)	Output				
z(13)	Output				
z(12)	Output				
z(11)	Output				
z(10)	Output				
z(9)	Output				
z(8)	Output				
z(7)	Output				
z(6)	Output				
z(5)	Output				
z(4)	Output				
z(3)	Output				
z(2)	Output				
z(1)	Output				
z(0)	Output				
end_flag	Output				

Total registers mapped: 0

C.2.2 Xilinx ISE Timing Report

```

-----
Release 10.1 Trace (lin64)
Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.

/nfs/sw_cmc/linux-64/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce -ise
/nfs/home/r/r_fenste/COEN6501/synth_test/pipelined/ise/booth_8_pipelined/booth_8_pipelined.ise
-intstyle ise -e 3 -s 7 -xml booth_8_pipelined booth_8_pipelined.ncd -o
booth_8_pipelined.twr booth_8_pipelined.pcf -ucf booth_8_pipelined.ucf

Design file:          booth_8_pipelined.ncd
Physical constraint file: booth_8_pipelined.pcf
Device,package,speed: xc2vp30,ff896,-7 (PRODUCTION 1.94 2008-01-09)
Report level:        error report

Environment Variable  Effect
-----
NONE                 No environment variables were set
-----

```

INFO:Timing:2698 - No timing constraints found, doing default enumeration.
 INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths option. All paths that are not constrained will be reported in the unconstrained paths section(s) of the report.
 INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on a 50 Ohm transmission line loading model. For the details of this model, and for more information on accounting for different loading conditions, please see the device datasheet.

Data Sheet report:

 All values displayed in nanoseconds (ns)

Pad to Pad

Source Pad	Destination Pad	Delay
clk	end_flag	9.479
clk	z(0)	8.288
clk	z(1)	9.700
clk	z(2)	9.909
clk	z(3)	10.074
clk	z(4)	10.807
clk	z(5)	11.868
clk	z(6)	12.688
clk	z(7)	13.920
clk	z(8)	13.935
clk	z(9)	14.771
clk	z(10)	14.532
clk	z(11)	15.436
clk	z(12)	16.159
clk	z(13)	16.631
clk	z(14)	16.547
clk	z(15)	16.653
clr	end_flag	8.800
clr	z(0)	11.602
clr	z(1)	12.721
clr	z(2)	12.790
clr	z(3)	13.588
clr	z(4)	14.474
clr	z(5)	15.535
clr	z(6)	16.355
clr	z(7)	17.587
clr	z(8)	17.602
clr	z(9)	18.438
clr	z(10)	18.199
clr	z(11)	19.334
clr	z(12)	20.057
clr	z(13)	20.529
clr	z(14)	20.445
clr	z(15)	20.551
load	end_flag	7.564
load	z(0)	10.051
load	z(1)	11.170
load	z(2)	11.336
load	z(3)	11.111

load	z(4)		11.997
load	z(5)		13.082
load	z(6)		13.879
load	z(7)		15.110
load	z(8)		15.125
load	z(9)		15.961
load	z(10)		15.838
load	z(11)		17.212
load	z(12)		17.935
load	z(13)		18.407
load	z(14)		18.323
load	z(15)		18.429
operand_a(0)	z(0)		9.648
operand_a(0)	z(1)		10.767
operand_a(0)	z(2)		10.295
operand_a(0)	z(3)		10.186
operand_a(0)	z(4)		11.072
operand_a(0)	z(5)		12.133
operand_a(0)	z(6)		12.953
operand_a(0)	z(7)		14.185
operand_a(0)	z(8)		14.200
operand_a(0)	z(9)		15.036
operand_a(0)	z(10)		14.859
operand_a(0)	z(11)		16.233
operand_a(0)	z(12)		16.956
operand_a(0)	z(13)		17.428
operand_a(0)	z(14)		17.344
operand_a(0)	z(15)		17.450
operand_a(1)	z(1)		10.268
operand_a(1)	z(2)		9.866
operand_a(1)	z(3)		10.137
operand_a(1)	z(4)		10.995
operand_a(1)	z(5)		12.056
operand_a(1)	z(6)		12.876
operand_a(1)	z(7)		14.108
operand_a(1)	z(8)		14.123
operand_a(1)	z(9)		14.959
operand_a(1)	z(10)		14.830
operand_a(1)	z(11)		16.204
operand_a(1)	z(12)		16.927
operand_a(1)	z(13)		17.399
operand_a(1)	z(14)		17.315
operand_a(1)	z(15)		17.421
operand_a(2)	z(2)		9.160
operand_a(2)	z(3)		9.674
operand_a(2)	z(4)		10.384
operand_a(2)	z(5)		11.635
operand_a(2)	z(6)		12.432
operand_a(2)	z(7)		13.614
operand_a(2)	z(8)		13.629
operand_a(2)	z(9)		14.465
operand_a(2)	z(10)		14.391
operand_a(2)	z(11)		15.765
operand_a(2)	z(12)		16.488
operand_a(2)	z(13)		16.960
operand_a(2)	z(14)		16.876
operand_a(2)	z(15)		16.982
operand_a(3)	z(3)		9.411

operand_a(3)	z(4)		10.223
operand_a(3)	z(5)		11.745
operand_a(3)	z(6)		12.542
operand_a(3)	z(7)		13.724
operand_a(3)	z(8)		13.739
operand_a(3)	z(9)		14.575
operand_a(3)	z(10)		14.501
operand_a(3)	z(11)		15.875
operand_a(3)	z(12)		16.598
operand_a(3)	z(13)		17.070
operand_a(3)	z(14)		16.986
operand_a(3)	z(15)		17.092
operand_a(4)	z(4)		10.041
operand_a(4)	z(5)		11.296
operand_a(4)	z(6)		12.093
operand_a(4)	z(7)		13.275
operand_a(4)	z(8)		13.290
operand_a(4)	z(9)		14.525
operand_a(4)	z(10)		14.763
operand_a(4)	z(11)		16.137
operand_a(4)	z(12)		16.860
operand_a(4)	z(13)		17.332
operand_a(4)	z(14)		17.248
operand_a(4)	z(15)		17.354
operand_a(5)	z(5)		10.340
operand_a(5)	z(6)		11.137
operand_a(5)	z(7)		12.657
operand_a(5)	z(8)		12.672
operand_a(5)	z(9)		13.620
operand_a(5)	z(10)		13.936
operand_a(5)	z(11)		15.310
operand_a(5)	z(12)		16.033
operand_a(5)	z(13)		16.505
operand_a(5)	z(14)		16.421
operand_a(5)	z(15)		16.527
operand_a(6)	z(6)		10.301
operand_a(6)	z(7)		11.935
operand_a(6)	z(8)		11.987
operand_a(6)	z(9)		13.350
operand_a(6)	z(10)		13.588
operand_a(6)	z(11)		14.962
operand_a(6)	z(12)		15.685
operand_a(6)	z(13)		16.157
operand_a(6)	z(14)		16.073
operand_a(6)	z(15)		16.179
operand_a(7)	z(7)		11.724
operand_a(7)	z(8)		11.739
operand_a(7)	z(9)		13.061
operand_a(7)	z(10)		13.299
operand_a(7)	z(11)		14.673
operand_a(7)	z(12)		15.396
operand_a(7)	z(13)		15.868
operand_a(7)	z(14)		15.784
operand_a(7)	z(15)		15.890
operand_b(0)	z(0)		8.712
operand_b(0)	z(1)		9.660
operand_b(0)	z(2)		9.702
operand_b(0)	z(3)		9.973

operand_b(0)	z(4)		10.706
operand_b(0)	z(5)		11.767
operand_b(0)	z(6)		12.587
operand_b(0)	z(7)		13.819
operand_b(0)	z(8)		13.834
operand_b(0)	z(9)		14.670
operand_b(0)	z(10)		14.431
operand_b(0)	z(11)		15.335
operand_b(0)	z(12)		16.058
operand_b(0)	z(13)		16.530
operand_b(0)	z(14)		16.446
operand_b(0)	z(15)		16.552
operand_b(1)	z(1)		9.968
operand_b(1)	z(2)		10.091
operand_b(1)	z(3)		10.343
operand_b(1)	z(4)		11.207
operand_b(1)	z(5)		12.268
operand_b(1)	z(6)		13.088
operand_b(1)	z(7)		14.320
operand_b(1)	z(8)		14.335
operand_b(1)	z(9)		15.171
operand_b(1)	z(10)		14.932
operand_b(1)	z(11)		15.836
operand_b(1)	z(12)		16.559
operand_b(1)	z(13)		17.031
operand_b(1)	z(14)		16.947
operand_b(1)	z(15)		17.053
operand_b(2)	z(2)		10.474
operand_b(2)	z(3)		10.098
operand_b(2)	z(4)		10.984
operand_b(2)	z(5)		12.045
operand_b(2)	z(6)		12.865
operand_b(2)	z(7)		14.097
operand_b(2)	z(8)		14.112
operand_b(2)	z(9)		14.948
operand_b(2)	z(10)		14.709
operand_b(2)	z(11)		15.812
operand_b(2)	z(12)		16.535
operand_b(2)	z(13)		17.007
operand_b(2)	z(14)		16.923
operand_b(2)	z(15)		17.029
operand_b(3)	z(3)		10.888
operand_b(3)	z(4)		11.774
operand_b(3)	z(5)		12.835
operand_b(3)	z(6)		13.655
operand_b(3)	z(7)		14.887
operand_b(3)	z(8)		14.902
operand_b(3)	z(9)		15.738
operand_b(3)	z(10)		15.499
operand_b(3)	z(11)		16.403
operand_b(3)	z(12)		17.126
operand_b(3)	z(13)		17.598
operand_b(3)	z(14)		17.514
operand_b(3)	z(15)		17.620
operand_b(4)	z(4)		9.758
operand_b(4)	z(5)		10.500
operand_b(4)	z(6)		11.487
operand_b(4)	z(7)		12.764

operand_b(4)	z(8)		12.779
operand_b(4)	z(9)		13.615
operand_b(4)	z(10)		13.468
operand_b(4)	z(11)		14.287
operand_b(4)	z(12)		15.227
operand_b(4)	z(13)		15.699
operand_b(4)	z(14)		15.615
operand_b(4)	z(15)		15.658
operand_b(5)	z(5)		10.004
operand_b(5)	z(6)		10.991
operand_b(5)	z(7)		13.354
operand_b(5)	z(8)		13.369
operand_b(5)	z(9)		14.205
operand_b(5)	z(10)		14.107
operand_b(5)	z(11)		14.699
operand_b(5)	z(12)		15.639
operand_b(5)	z(13)		16.111
operand_b(5)	z(14)		16.027
operand_b(5)	z(15)		16.302
operand_b(6)	z(6)		9.995
operand_b(6)	z(7)		11.930
operand_b(6)	z(8)		11.945
operand_b(6)	z(9)		12.781
operand_b(6)	z(10)		12.959
operand_b(6)	z(11)		13.778
operand_b(6)	z(12)		14.718
operand_b(6)	z(13)		15.190
operand_b(6)	z(14)		15.106
operand_b(6)	z(15)		14.953
operand_b(7)	z(7)		12.229
operand_b(7)	z(8)		12.244
operand_b(7)	z(9)		13.147
operand_b(7)	z(10)		13.258
operand_b(7)	z(11)		14.077
operand_b(7)	z(12)		15.017
operand_b(7)	z(13)		15.489
operand_b(7)	z(14)		15.405
operand_b(7)	z(15)		15.458

-----+-----+-----+
Analysis completed Sun Nov 22 16:03:50 2015

Trace Settings:

Trace Settings

Peak Memory Usage: 304 MB

C.3 Compartmentalized Pipelined Implementation

C.3.1 PrecisionRTL Area Report for Stage 0 Combinational Logic

```
*****
Device Utilization for 2VP30ff896
*****
Resource                Used      Avail      Utilization
```

```

-----
IOs                72      556     12.95%
Global Buffers    0        16       0.00%
LUTs              110     27392    0.40%
CLB Slices        55     13696    0.40%
Dffs or Latches   0      29060    0.00%
Block RAMs        0        136      0.00%
Block Multipliers 0        136      0.00%
Block Multiplier Dffs 0     4896     0.00%
GT_CUSTOM         0         8       0.00%
-----

```

Library: work Cell: booth_pipeline0 View: rtl

Cell	Library	References	Total Area
GND	xcv2p	1 x	
IBUF	xcv2p	16 x	
OBUF	xcv2p	52 x	
OBUFT	xcv2p	1 x	
VCC	xcv2p	1 x	
booth_unit_0	work	1 x	26 gates 26 LUTs
booth_unit_1	work	1 x	28 gates 28 LUTs
booth_unit_2	work	1 x	28 gates 28 LUTs
booth_unit_3	work	1 x	28 gates 28 LUTs

Number of ports : 72
Number of nets : 123
Number of instances : 75
Number of references to this view : 0

Total accumulated area :
Number of LUTs : 110
Number of gates : 110
Number of accumulated instances : 181

IO Register Mapping Report

Design: work.booth_pipeline0.rtl

Port	Direction	INFF	OUTFF	TRIFF
operand_a(7)	Input			
operand_a(6)	Input			
operand_a(5)	Input			

operand_a(4)	Input			
operand_a(3)	Input			
operand_a(2)	Input			
operand_a(1)	Input			
operand_a(0)	Input			
operand_b(7)	Input			
operand_b(6)	Input			
operand_b(5)	Input			
operand_b(4)	Input			
operand_b(3)	Input			
operand_b(2)	Input			
operand_b(1)	Input			
operand_b(0)	Input			
clk	Input			
clr	Input			
data_valid_in	Input			
out_bu_0(15)	Output			
out_bu_0(14)	Output			
out_bu_0(13)	Output			
out_bu_0(12)	Output			
out_bu_0(11)	Output			
out_bu_0(10)	Output			
out_bu_0(9)	Output			
out_bu_0(8)	Output			
out_bu_0(7)	Output			
out_bu_0(6)	Output			
out_bu_0(5)	Output			
out_bu_0(4)	Output			
out_bu_0(3)	Output			

out_bu_0(2)	Output			
out_bu_0(1)	Output			
out_bu_0(0)	Output			
out_bu_1(13)	Output			
out_bu_1(12)	Output			
out_bu_1(11)	Output			
out_bu_1(10)	Output			
out_bu_1(9)	Output			
out_bu_1(8)	Output			
out_bu_1(7)	Output			
out_bu_1(6)	Output			
out_bu_1(5)	Output			
out_bu_1(4)	Output			
out_bu_1(3)	Output			
out_bu_1(2)	Output			
out_bu_1(1)	Output			
out_bu_1(0)	Output			
out_bu_2(11)	Output			
out_bu_2(10)	Output			
out_bu_2(9)	Output			
out_bu_2(8)	Output			
out_bu_2(7)	Output			
out_bu_2(6)	Output			
out_bu_2(5)	Output			
out_bu_2(4)	Output			
out_bu_2(3)	Output			
out_bu_2(2)	Output			
out_bu_2(1)	Output			
out_bu_2(0)	Output			

out_bu_3(9)	Output				
out_bu_3(8)	Output				
out_bu_3(7)	Output				
out_bu_3(6)	Output				
out_bu_3(5)	Output				
out_bu_3(4)	Output				
out_bu_3(3)	Output				
out_bu_3(2)	Output				
out_bu_3(1)	Output				
out_bu_3(0)	Output				
data_valid_out	Output				

Total registers mapped: 0

C.3.2 Xilinx ISE Timing Report for Stage 0 Combinational Logic

Release 10.1 Trace (lin64)

Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.

```
/nfs/sw_cmc/linux-64/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce -ise
/nfs/home/r/r_fenste/COEN6501/synth_test2/stage0_combinationallogic/ise/stage0_comb/stage0_comb.ise
-intstyle ise -e 3 -s 7 -xml booth_pipeline0 booth_pipeline0.ncd -o
booth_pipeline0.twr booth_pipeline0.pcf -ucf booth_pipeline0.ucf
```

```
Design file: booth_pipeline0.ncd
Physical constraint file: booth_pipeline0.pcf
Device,package,speed: xc2vp30,ff896,-7 (PRODUCTION 1.94 2008-01-09)
Report level: error report
```

```
Environment Variable Effect
-----
NONE No environment variables were set
```

```
INFO:Timing:2698 - No timing constraints found, doing default enumeration.
INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths
option. All paths that are not constrained will be reported in the
unconstrained paths section(s) of the report.
INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on
a 50 Ohm transmission line loading model. For the details of this model,
and for more information on accounting for different loading conditions,
please see the device datasheet.
```

Data Sheet report:

All values displayed in nanoseconds (ns)

Pad to Pad

Source Pad	Destination Pad	Delay
operand_a(0)	out_bu_0(0)	4.412
operand_a(0)	out_bu_0(1)	5.334
operand_a(0)	out_bu_0(2)	4.709
operand_a(0)	out_bu_0(3)	5.850
operand_a(0)	out_bu_0(4)	6.463
operand_a(0)	out_bu_0(5)	7.314
operand_a(0)	out_bu_0(6)	7.034
operand_a(0)	out_bu_0(7)	6.715
operand_a(0)	out_bu_0(8)	7.123
operand_a(0)	out_bu_0(9)	7.556
operand_a(0)	out_bu_0(10)	7.063
operand_a(0)	out_bu_0(11)	7.697
operand_a(0)	out_bu_0(12)	7.057
operand_a(0)	out_bu_0(13)	7.053
operand_a(0)	out_bu_0(14)	7.038
operand_a(0)	out_bu_0(15)	7.262
operand_a(0)	out_bu_1(0)	5.390
operand_a(0)	out_bu_1(1)	5.201
operand_a(0)	out_bu_1(2)	5.181
operand_a(0)	out_bu_1(3)	5.687
operand_a(0)	out_bu_1(4)	6.485
operand_a(0)	out_bu_1(5)	6.348
operand_a(0)	out_bu_1(6)	5.996
operand_a(0)	out_bu_1(7)	6.127
operand_a(0)	out_bu_1(8)	6.392
operand_a(0)	out_bu_1(9)	6.252
operand_a(0)	out_bu_1(10)	6.479
operand_a(0)	out_bu_1(11)	7.110
operand_a(0)	out_bu_1(12)	7.245
operand_a(0)	out_bu_1(13)	6.851
operand_a(0)	out_bu_2(0)	4.897
operand_a(0)	out_bu_2(1)	5.023
operand_a(0)	out_bu_2(2)	6.286
operand_a(0)	out_bu_2(3)	5.266
operand_a(0)	out_bu_2(4)	6.701
operand_a(0)	out_bu_2(5)	6.120
operand_a(0)	out_bu_2(6)	6.504
operand_a(0)	out_bu_2(7)	5.982
operand_a(0)	out_bu_2(8)	7.209
operand_a(0)	out_bu_2(9)	6.726
operand_a(0)	out_bu_2(10)	6.973
operand_a(0)	out_bu_2(11)	7.042
operand_a(0)	out_bu_3(0)	5.265
operand_a(0)	out_bu_3(1)	5.762
operand_a(0)	out_bu_3(2)	5.262
operand_a(0)	out_bu_3(3)	5.396
operand_a(0)	out_bu_3(4)	6.137
operand_a(0)	out_bu_3(5)	6.600
operand_a(0)	out_bu_3(6)	6.170
operand_a(0)	out_bu_3(7)	5.913
operand_a(0)	out_bu_3(8)	6.362

operand_a(0)	out_bu_3(9)		6.495
operand_a(1)	out_bu_0(1)		5.296
operand_a(1)	out_bu_0(2)		5.086
operand_a(1)	out_bu_0(3)		6.303
operand_a(1)	out_bu_0(4)		6.692
operand_a(1)	out_bu_0(5)		7.543
operand_a(1)	out_bu_0(6)		7.263
operand_a(1)	out_bu_0(7)		6.944
operand_a(1)	out_bu_0(8)		7.352
operand_a(1)	out_bu_0(9)		7.785
operand_a(1)	out_bu_0(10)		7.292
operand_a(1)	out_bu_0(11)		7.926
operand_a(1)	out_bu_0(12)		7.286
operand_a(1)	out_bu_0(13)		7.282
operand_a(1)	out_bu_0(14)		7.267
operand_a(1)	out_bu_0(15)		7.491
operand_a(1)	out_bu_1(1)		5.346
operand_a(1)	out_bu_1(2)		5.510
operand_a(1)	out_bu_1(3)		6.321
operand_a(1)	out_bu_1(4)		6.681
operand_a(1)	out_bu_1(5)		6.544
operand_a(1)	out_bu_1(6)		6.192
operand_a(1)	out_bu_1(7)		6.323
operand_a(1)	out_bu_1(8)		6.588
operand_a(1)	out_bu_1(9)		6.448
operand_a(1)	out_bu_1(10)		6.675
operand_a(1)	out_bu_1(11)		7.306
operand_a(1)	out_bu_1(12)		7.441
operand_a(1)	out_bu_1(13)		7.047
operand_a(1)	out_bu_2(1)		5.378
operand_a(1)	out_bu_2(2)		6.472
operand_a(1)	out_bu_2(3)		5.812
operand_a(1)	out_bu_2(4)		7.106
operand_a(1)	out_bu_2(5)		6.525
operand_a(1)	out_bu_2(6)		6.909
operand_a(1)	out_bu_2(7)		6.387
operand_a(1)	out_bu_2(8)		7.614
operand_a(1)	out_bu_2(9)		7.131
operand_a(1)	out_bu_2(10)		7.378
operand_a(1)	out_bu_2(11)		7.447
operand_a(1)	out_bu_3(1)		6.064
operand_a(1)	out_bu_3(2)		5.399
operand_a(1)	out_bu_3(3)		5.820
operand_a(1)	out_bu_3(4)		6.538
operand_a(1)	out_bu_3(5)		7.001
operand_a(1)	out_bu_3(6)		6.571
operand_a(1)	out_bu_3(7)		6.314
operand_a(1)	out_bu_3(8)		6.763
operand_a(1)	out_bu_3(9)		6.896
operand_a(2)	out_bu_0(2)		5.502
operand_a(2)	out_bu_0(3)		6.042
operand_a(2)	out_bu_0(4)		5.670
operand_a(2)	out_bu_0(5)		6.356
operand_a(2)	out_bu_0(6)		6.269
operand_a(2)	out_bu_0(7)		6.597
operand_a(2)	out_bu_0(8)		7.127
operand_a(2)	out_bu_0(9)		7.560
operand_a(2)	out_bu_0(10)		7.067

operand_a(2)	out_bu_0(11)		7.701
operand_a(2)	out_bu_0(12)		7.061
operand_a(2)	out_bu_0(13)		7.057
operand_a(2)	out_bu_0(14)		7.042
operand_a(2)	out_bu_0(15)		7.266
operand_a(2)	out_bu_1(2)		5.639
operand_a(2)	out_bu_1(3)		6.205
operand_a(2)	out_bu_1(4)		6.308
operand_a(2)	out_bu_1(5)		6.219
operand_a(2)	out_bu_1(6)		6.051
operand_a(2)	out_bu_1(7)		5.983
operand_a(2)	out_bu_1(8)		6.438
operand_a(2)	out_bu_1(9)		6.298
operand_a(2)	out_bu_1(10)		6.525
operand_a(2)	out_bu_1(11)		7.156
operand_a(2)	out_bu_1(12)		7.291
operand_a(2)	out_bu_1(13)		6.897
operand_a(2)	out_bu_2(2)		6.078
operand_a(2)	out_bu_2(3)		5.645
operand_a(2)	out_bu_2(4)		6.021
operand_a(2)	out_bu_2(5)		5.440
operand_a(2)	out_bu_2(6)		5.689
operand_a(2)	out_bu_2(7)		5.167
operand_a(2)	out_bu_2(8)		6.296
operand_a(2)	out_bu_2(9)		5.813
operand_a(2)	out_bu_2(10)		6.060
operand_a(2)	out_bu_2(11)		6.129
operand_a(2)	out_bu_3(2)		4.762
operand_a(2)	out_bu_3(3)		5.394
operand_a(2)	out_bu_3(4)		5.189
operand_a(2)	out_bu_3(5)		5.533
operand_a(2)	out_bu_3(6)		5.134
operand_a(2)	out_bu_3(7)		5.685
operand_a(2)	out_bu_3(8)		5.845
operand_a(2)	out_bu_3(9)		5.978
operand_a(3)	out_bu_0(3)		6.134
operand_a(3)	out_bu_0(4)		6.053
operand_a(3)	out_bu_0(5)		6.739
operand_a(3)	out_bu_0(6)		6.652
operand_a(3)	out_bu_0(7)		6.980
operand_a(3)	out_bu_0(8)		7.510
operand_a(3)	out_bu_0(9)		7.943
operand_a(3)	out_bu_0(10)		7.450
operand_a(3)	out_bu_0(11)		8.084
operand_a(3)	out_bu_0(12)		7.444
operand_a(3)	out_bu_0(13)		7.440
operand_a(3)	out_bu_0(14)		7.425
operand_a(3)	out_bu_0(15)		7.649
operand_a(3)	out_bu_1(3)		6.434
operand_a(3)	out_bu_1(4)		6.684
operand_a(3)	out_bu_1(5)		6.595
operand_a(3)	out_bu_1(6)		6.427
operand_a(3)	out_bu_1(7)		6.359
operand_a(3)	out_bu_1(8)		6.814
operand_a(3)	out_bu_1(9)		6.674
operand_a(3)	out_bu_1(10)		6.901
operand_a(3)	out_bu_1(11)		7.532
operand_a(3)	out_bu_1(12)		7.667

operand_a(3)	out_bu_1(13)		7.273
operand_a(3)	out_bu_2(3)		5.560
operand_a(3)	out_bu_2(4)		6.092
operand_a(3)	out_bu_2(5)		5.511
operand_a(3)	out_bu_2(6)		5.760
operand_a(3)	out_bu_2(7)		5.238
operand_a(3)	out_bu_2(8)		6.367
operand_a(3)	out_bu_2(9)		5.884
operand_a(3)	out_bu_2(10)		6.131
operand_a(3)	out_bu_2(11)		6.200
operand_a(3)	out_bu_3(3)		5.265
operand_a(3)	out_bu_3(4)		5.262
operand_a(3)	out_bu_3(5)		5.611
operand_a(3)	out_bu_3(6)		5.212
operand_a(3)	out_bu_3(7)		5.763
operand_a(3)	out_bu_3(8)		5.923
operand_a(3)	out_bu_3(9)		6.056
operand_a(4)	out_bu_0(4)		4.307
operand_a(4)	out_bu_0(5)		5.476
operand_a(4)	out_bu_0(6)		5.233
operand_a(4)	out_bu_0(7)		6.030
operand_a(4)	out_bu_0(8)		6.637
operand_a(4)	out_bu_0(9)		7.070
operand_a(4)	out_bu_0(10)		6.577
operand_a(4)	out_bu_0(11)		7.211
operand_a(4)	out_bu_0(12)		6.571
operand_a(4)	out_bu_0(13)		6.567
operand_a(4)	out_bu_0(14)		6.552
operand_a(4)	out_bu_0(15)		6.776
operand_a(4)	out_bu_1(4)		5.032
operand_a(4)	out_bu_1(5)		5.401
operand_a(4)	out_bu_1(6)		5.410
operand_a(4)	out_bu_1(7)		5.478
operand_a(4)	out_bu_1(8)		5.910
operand_a(4)	out_bu_1(9)		5.770
operand_a(4)	out_bu_1(10)		5.997
operand_a(4)	out_bu_1(11)		6.628
operand_a(4)	out_bu_1(12)		6.763
operand_a(4)	out_bu_1(13)		6.369
operand_a(4)	out_bu_2(4)		5.577
operand_a(4)	out_bu_2(5)		5.210
operand_a(4)	out_bu_2(6)		5.978
operand_a(4)	out_bu_2(7)		5.456
operand_a(4)	out_bu_2(8)		6.557
operand_a(4)	out_bu_2(9)		6.074
operand_a(4)	out_bu_2(10)		6.321
operand_a(4)	out_bu_2(11)		6.390
operand_a(4)	out_bu_3(4)		4.827
operand_a(4)	out_bu_3(5)		5.745
operand_a(4)	out_bu_3(6)		5.315
operand_a(4)	out_bu_3(7)		4.986
operand_a(4)	out_bu_3(8)		5.310
operand_a(4)	out_bu_3(9)		5.443
operand_a(5)	out_bu_0(5)		5.885
operand_a(5)	out_bu_0(6)		5.281
operand_a(5)	out_bu_0(7)		5.743
operand_a(5)	out_bu_0(8)		6.350
operand_a(5)	out_bu_0(9)		6.783

operand_a(5)	out_bu_0(10)		6.290
operand_a(5)	out_bu_0(11)		6.924
operand_a(5)	out_bu_0(12)		6.284
operand_a(5)	out_bu_0(13)		6.280
operand_a(5)	out_bu_0(14)		6.265
operand_a(5)	out_bu_0(15)		6.489
operand_a(5)	out_bu_1(5)		5.713
operand_a(5)	out_bu_1(6)		5.317
operand_a(5)	out_bu_1(7)		5.090
operand_a(5)	out_bu_1(8)		5.522
operand_a(5)	out_bu_1(9)		5.382
operand_a(5)	out_bu_1(10)		5.609
operand_a(5)	out_bu_1(11)		6.240
operand_a(5)	out_bu_1(12)		6.375
operand_a(5)	out_bu_1(13)		5.981
operand_a(5)	out_bu_2(5)		5.496
operand_a(5)	out_bu_2(6)		5.824
operand_a(5)	out_bu_2(7)		5.302
operand_a(5)	out_bu_2(8)		5.929
operand_a(5)	out_bu_2(9)		5.446
operand_a(5)	out_bu_2(10)		5.693
operand_a(5)	out_bu_2(11)		5.762
operand_a(5)	out_bu_3(5)		5.641
operand_a(5)	out_bu_3(6)		5.211
operand_a(5)	out_bu_3(7)		5.056
operand_a(5)	out_bu_3(8)		5.509
operand_a(5)	out_bu_3(9)		5.642
operand_a(6)	out_bu_0(6)		4.732
operand_a(6)	out_bu_0(7)		5.691
operand_a(6)	out_bu_0(8)		6.298
operand_a(6)	out_bu_0(9)		6.731
operand_a(6)	out_bu_0(10)		6.238
operand_a(6)	out_bu_0(11)		6.872
operand_a(6)	out_bu_0(12)		6.232
operand_a(6)	out_bu_0(13)		6.228
operand_a(6)	out_bu_0(14)		6.213
operand_a(6)	out_bu_0(15)		6.437
operand_a(6)	out_bu_1(6)		5.345
operand_a(6)	out_bu_1(7)		5.149
operand_a(6)	out_bu_1(8)		5.581
operand_a(6)	out_bu_1(9)		5.441
operand_a(6)	out_bu_1(10)		5.668
operand_a(6)	out_bu_1(11)		6.299
operand_a(6)	out_bu_1(12)		6.434
operand_a(6)	out_bu_1(13)		6.040
operand_a(6)	out_bu_2(6)		5.967
operand_a(6)	out_bu_2(7)		5.445
operand_a(6)	out_bu_2(8)		6.614
operand_a(6)	out_bu_2(9)		6.131
operand_a(6)	out_bu_2(10)		6.378
operand_a(6)	out_bu_2(11)		6.447
operand_a(6)	out_bu_3(6)		5.415
operand_a(6)	out_bu_3(7)		5.269
operand_a(6)	out_bu_3(8)		5.332
operand_a(6)	out_bu_3(9)		5.465
operand_a(7)	out_bu_0(7)		5.761
operand_a(7)	out_bu_0(8)		5.284
operand_a(7)	out_bu_0(9)		5.717

operand_a(7)	out_bu_0(10)		5.224
operand_a(7)	out_bu_0(11)		5.858
operand_a(7)	out_bu_0(12)		5.218
operand_a(7)	out_bu_0(13)		5.214
operand_a(7)	out_bu_0(14)		5.199
operand_a(7)	out_bu_0(15)		5.423
operand_a(7)	out_bu_1(7)		5.565
operand_a(7)	out_bu_1(8)		5.524
operand_a(7)	out_bu_1(9)		5.384
operand_a(7)	out_bu_1(10)		5.611
operand_a(7)	out_bu_1(11)		6.242
operand_a(7)	out_bu_1(12)		6.377
operand_a(7)	out_bu_1(13)		5.983
operand_a(7)	out_bu_2(7)		5.210
operand_a(7)	out_bu_2(8)		5.996
operand_a(7)	out_bu_2(9)		5.513
operand_a(7)	out_bu_2(10)		5.760
operand_a(7)	out_bu_2(11)		5.829
operand_a(7)	out_bu_3(7)		5.166
operand_a(7)	out_bu_3(8)		5.315
operand_a(7)	out_bu_3(9)		5.448
operand_b(0)	out_bu_0(0)		4.401
operand_b(0)	out_bu_0(1)		4.447
operand_b(0)	out_bu_0(2)		4.127
operand_b(0)	out_bu_0(3)		3.991
operand_b(0)	out_bu_0(4)		4.462
operand_b(0)	out_bu_0(5)		5.242
operand_b(0)	out_bu_0(6)		4.603
operand_b(0)	out_bu_0(7)		4.744
operand_b(0)	out_bu_0(8)		5.672
operand_b(0)	out_bu_0(9)		6.105
operand_b(0)	out_bu_0(10)		5.612
operand_b(0)	out_bu_0(11)		6.246
operand_b(0)	out_bu_0(12)		5.606
operand_b(0)	out_bu_0(13)		5.602
operand_b(0)	out_bu_0(14)		5.587
operand_b(0)	out_bu_0(15)		5.811
operand_b(1)	out_bu_0(1)		5.349
operand_b(1)	out_bu_0(2)		5.329
operand_b(1)	out_bu_0(3)		6.315
operand_b(1)	out_bu_0(4)		6.740
operand_b(1)	out_bu_0(5)		7.591
operand_b(1)	out_bu_0(6)		7.311
operand_b(1)	out_bu_0(7)		7.140
operand_b(1)	out_bu_0(8)		7.670
operand_b(1)	out_bu_0(9)		8.103
operand_b(1)	out_bu_0(10)		7.610
operand_b(1)	out_bu_0(11)		8.244
operand_b(1)	out_bu_0(12)		7.604
operand_b(1)	out_bu_0(13)		7.600
operand_b(1)	out_bu_0(14)		7.585
operand_b(1)	out_bu_0(15)		7.809
operand_b(1)	out_bu_1(0)		5.420
operand_b(1)	out_bu_1(1)		6.243
operand_b(1)	out_bu_1(2)		6.443
operand_b(1)	out_bu_1(3)		6.177
operand_b(1)	out_bu_1(4)		6.246
operand_b(1)	out_bu_1(5)		6.123

operand_b(1)	out_bu_1(6)		6.016
operand_b(1)	out_bu_1(7)		5.526
operand_b(1)	out_bu_1(8)		6.470
operand_b(1)	out_bu_1(9)		6.330
operand_b(1)	out_bu_1(10)		6.557
operand_b(1)	out_bu_1(11)		7.188
operand_b(1)	out_bu_1(12)		7.323
operand_b(1)	out_bu_1(13)		6.929
operand_b(2)	out_bu_1(0)		4.749
operand_b(2)	out_bu_1(1)		5.448
operand_b(2)	out_bu_1(2)		5.648
operand_b(2)	out_bu_1(3)		5.382
operand_b(2)	out_bu_1(4)		5.451
operand_b(2)	out_bu_1(5)		5.328
operand_b(2)	out_bu_1(6)		5.352
operand_b(2)	out_bu_1(7)		4.862
operand_b(2)	out_bu_1(8)		5.675
operand_b(2)	out_bu_1(9)		5.535
operand_b(2)	out_bu_1(10)		5.762
operand_b(2)	out_bu_1(11)		6.393
operand_b(2)	out_bu_1(12)		6.528
operand_b(2)	out_bu_1(13)		6.134
operand_b(3)	out_bu_1(1)		6.130
operand_b(3)	out_bu_1(2)		6.319
operand_b(3)	out_bu_1(3)		6.445
operand_b(3)	out_bu_1(4)		6.528
operand_b(3)	out_bu_1(5)		6.483
operand_b(3)	out_bu_1(6)		6.087
operand_b(3)	out_bu_1(7)		6.585
operand_b(3)	out_bu_1(8)		6.544
operand_b(3)	out_bu_1(9)		6.404
operand_b(3)	out_bu_1(10)		6.631
operand_b(3)	out_bu_1(11)		7.262
operand_b(3)	out_bu_1(12)		7.397
operand_b(3)	out_bu_1(13)		7.003
operand_b(3)	out_bu_2(0)		4.629
operand_b(3)	out_bu_2(1)		4.952
operand_b(3)	out_bu_2(2)		6.147
operand_b(3)	out_bu_2(3)		5.145
operand_b(3)	out_bu_2(4)		5.358
operand_b(3)	out_bu_2(5)		4.801
operand_b(3)	out_bu_2(6)		4.741
operand_b(3)	out_bu_2(7)		4.839
operand_b(3)	out_bu_2(8)		5.954
operand_b(3)	out_bu_2(9)		5.471
operand_b(3)	out_bu_2(10)		5.718
operand_b(3)	out_bu_2(11)		5.787
operand_b(4)	out_bu_2(0)		4.190
operand_b(4)	out_bu_2(1)		4.809
operand_b(4)	out_bu_2(2)		6.004
operand_b(4)	out_bu_2(3)		5.002
operand_b(4)	out_bu_2(4)		5.215
operand_b(4)	out_bu_2(5)		4.663
operand_b(4)	out_bu_2(6)		4.598
operand_b(4)	out_bu_2(7)		4.701
operand_b(4)	out_bu_2(8)		5.816
operand_b(4)	out_bu_2(9)		5.333
operand_b(4)	out_bu_2(10)		5.580

operand_b(4)	out_bu_2(11)		5.649
operand_b(5)	out_bu_2(1)		5.647
operand_b(5)	out_bu_2(2)		6.531
operand_b(5)	out_bu_2(3)		5.830
operand_b(5)	out_bu_2(4)		6.183
operand_b(5)	out_bu_2(5)		5.725
operand_b(5)	out_bu_2(6)		6.122
operand_b(5)	out_bu_2(7)		5.625
operand_b(5)	out_bu_2(8)		6.740
operand_b(5)	out_bu_2(9)		6.257
operand_b(5)	out_bu_2(10)		6.504
operand_b(5)	out_bu_2(11)		6.573
operand_b(5)	out_bu_3(0)		4.345
operand_b(5)	out_bu_3(1)		5.707
operand_b(5)	out_bu_3(2)		5.606
operand_b(5)	out_bu_3(3)		5.900
operand_b(5)	out_bu_3(4)		6.029
operand_b(5)	out_bu_3(5)		5.280
operand_b(5)	out_bu_3(6)		5.522
operand_b(5)	out_bu_3(7)		5.376
operand_b(5)	out_bu_3(8)		5.885
operand_b(5)	out_bu_3(9)		6.018
operand_b(6)	out_bu_3(0)		4.072
operand_b(6)	out_bu_3(1)		5.438
operand_b(6)	out_bu_3(2)		5.337
operand_b(6)	out_bu_3(3)		5.631
operand_b(6)	out_bu_3(4)		5.760
operand_b(6)	out_bu_3(5)		5.011
operand_b(6)	out_bu_3(6)		5.253
operand_b(6)	out_bu_3(7)		5.107
operand_b(6)	out_bu_3(8)		5.616
operand_b(6)	out_bu_3(9)		5.749
operand_b(7)	out_bu_3(1)		5.686
operand_b(7)	out_bu_3(2)		5.674
operand_b(7)	out_bu_3(3)		5.968
operand_b(7)	out_bu_3(4)		5.788
operand_b(7)	out_bu_3(5)		6.251
operand_b(7)	out_bu_3(6)		5.821
operand_b(7)	out_bu_3(7)		5.695
operand_b(7)	out_bu_3(8)		6.023
operand_b(7)	out_bu_3(9)		6.156
-----+-----+-----+			

Analysis completed Mon Nov 30 17:59:20 2015

Trace Settings:

Trace Settings

Peak Memory Usage: 298 MB

C.3.3 PrecisionRTL Area Report for Stage 0 Registers

Device Utilization for 2VP30ff896

Resource	Used	Avail	Utilization
I/Os	108	556	19.42%
Global Buffers	0	16	0.00%
LUTs	208	27392	0.76%
CLB Slices	104	13696	0.76%
Dffs or Latches	0	29060	0.00%
Block RAMs	0	136	0.00%
Block Multipliers	0	136	0.00%
Block Multiplier Dffs	0	4896	0.00%
GT_CUSTOM	0	8	0.00%

Library: work Cell: booth_pipeline0 View: rtl

Cell	Library	References	Total Area
GND	xcv2p	1 x	
IBUF	xcv2p	54 x	
OBUF	xcv2p	52 x	
OBUFT	xcv2p	1 x	
VCC	xcv2p	1 x	
operand_register_10bit	work	1 x	40 gates 40 LUTs
operand_register_12bit	work	1 x	48 gates 48 LUTs
operand_register_14bit	work	1 x	56 gates 56 LUTs
operand_register_16bit	work	1 x	65 gates 64 LUTs

Number of ports : 108
Number of nets : 215
Number of instances : 113
Number of references to this view : 0

Total accumulated area :
Number of LUTs : 208
Number of gates : 209
Number of accumulated instances : 317

IO Register Mapping Report

Design: work.booth_pipeline0.rtl

Port	Direction	INFF	OUTFF	TRIFF
in_bu_0(15)	Input			
in_bu_0(14)	Input			

out_bu_2(7)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_2(6)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_2(5)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_2(4)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_2(3)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_2(2)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_2(1)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_2(0)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_3(9)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_3(8)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_3(7)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_3(6)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_3(5)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_3(4)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_3(3)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_3(2)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_3(1)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
out_bu_3(0)	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
data_valid_out	Output				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+

Total registers mapped: 0

C.3.4 Xilinx ISE Timing Report for Stage 0 Registers

```
-----
Release 10.1 Trace (lin64)
Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.

/nfs/sw_cmc/linux-64/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce -ise
/nfs/home/r/r_fenste/COEN6501/synth_test2/stage0_register/ise/stage0_reg/stage0_reg.ise
-intstyle ise -e 3 -s 7 -xml booth_pipeline0 booth_pipeline0.ncd -o
booth_pipeline0.twr booth_pipeline0.pcf -ucf booth_pipeline0.ucf

Design file: booth_pipeline0.ncd
Physical constraint file: booth_pipeline0.pcf
Device,package,speed: xc2vp30,ff896,-7 (PRODUCTION 1.94 2008-01-09)
Report level: error report

Environment Variable Effect
-----
```

NONE

No environment variables were set

INFO:Timing:2698 - No timing constraints found, doing default enumeration.
INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths option. All paths that are not constrained will be reported in the unconstrained paths section(s) of the report.
INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on a 50 Ohm transmission line loading model. For the details of this model, and for more information on accounting for different loading conditions, please see the device datasheet.

Data Sheet report:

All values displayed in nanoseconds (ns)

Pad to Pad

Source Pad	Destination Pad	Delay
clk	out_bu_0(0)	6.069
clk	out_bu_0(1)	7.394
clk	out_bu_0(2)	7.014
clk	out_bu_0(3)	7.144
clk	out_bu_0(4)	7.507
clk	out_bu_0(5)	7.400
clk	out_bu_0(6)	7.854
clk	out_bu_0(7)	6.596
clk	out_bu_0(8)	4.866
clk	out_bu_0(9)	7.330
clk	out_bu_0(10)	5.429
clk	out_bu_0(11)	4.807
clk	out_bu_0(12)	6.137
clk	out_bu_0(13)	6.854
clk	out_bu_0(14)	6.484
clk	out_bu_0(15)	6.218
clk	out_bu_1(0)	7.621
clk	out_bu_1(1)	8.704
clk	out_bu_1(2)	8.596
clk	out_bu_1(3)	7.006
clk	out_bu_1(4)	6.420
clk	out_bu_1(5)	5.726
clk	out_bu_1(6)	6.696
clk	out_bu_1(7)	7.495
clk	out_bu_1(8)	4.886
clk	out_bu_1(9)	7.529
clk	out_bu_1(10)	6.393
clk	out_bu_1(11)	5.955
clk	out_bu_1(12)	7.471
clk	out_bu_1(13)	6.877
clk	out_bu_2(0)	5.922
clk	out_bu_2(1)	6.883
clk	out_bu_2(2)	6.340
clk	out_bu_2(3)	6.742
clk	out_bu_2(4)	6.561
clk	out_bu_2(5)	7.902

clk	out_bu_2(6)		6.599
clk	out_bu_2(7)		6.661
clk	out_bu_2(8)		7.787
clk	out_bu_2(9)		7.115
clk	out_bu_2(10)		7.035
clk	out_bu_2(11)		7.343
clk	out_bu_3(0)		7.122
clk	out_bu_3(1)		6.758
clk	out_bu_3(2)		7.019
clk	out_bu_3(3)		6.711
clk	out_bu_3(4)		7.226
clk	out_bu_3(5)		7.479
clk	out_bu_3(6)		7.816
clk	out_bu_3(7)		7.064
clk	out_bu_3(8)		6.591
clk	out_bu_3(9)		5.817
clr	out_bu_0(0)		6.415
clr	out_bu_0(1)		7.609
clr	out_bu_0(2)		8.280
clr	out_bu_0(3)		7.389
clr	out_bu_0(4)		7.273
clr	out_bu_0(5)		9.202
clr	out_bu_0(6)		9.585
clr	out_bu_0(7)		6.711
clr	out_bu_0(8)		6.080
clr	out_bu_0(9)		7.576
clr	out_bu_0(10)		6.060
clr	out_bu_0(11)		5.685
clr	out_bu_0(12)		7.114
clr	out_bu_0(13)		7.501
clr	out_bu_0(14)		7.415
clr	out_bu_0(15)		7.011
clr	out_bu_1(0)		8.502
clr	out_bu_1(1)		9.455
clr	out_bu_1(2)		10.782
clr	out_bu_1(3)		7.239
clr	out_bu_1(4)		6.906
clr	out_bu_1(5)		5.713
clr	out_bu_1(6)		7.328
clr	out_bu_1(7)		7.796
clr	out_bu_1(8)		4.938
clr	out_bu_1(9)		8.874
clr	out_bu_1(10)		7.271
clr	out_bu_1(11)		7.142
clr	out_bu_1(12)		8.860
clr	out_bu_1(13)		7.523
clr	out_bu_2(0)		6.197
clr	out_bu_2(1)		7.649
clr	out_bu_2(2)		6.702
clr	out_bu_2(3)		6.521
clr	out_bu_2(4)		7.066
clr	out_bu_2(5)		8.140
clr	out_bu_2(6)		7.575
clr	out_bu_2(7)		7.176
clr	out_bu_2(8)		8.079
clr	out_bu_2(9)		7.437
clr	out_bu_2(10)		7.561
clr	out_bu_2(11)		9.357

clr	out_bu_3(0)		6.770
clr	out_bu_3(1)		7.670
clr	out_bu_3(2)		8.102
clr	out_bu_3(3)		7.634
clr	out_bu_3(4)		8.077
clr	out_bu_3(5)		7.773
clr	out_bu_3(6)		9.603
clr	out_bu_3(7)		7.632
clr	out_bu_3(8)		7.577
clr	out_bu_3(9)		6.617
in_bu_0(0)	out_bu_0(0)		5.389
in_bu_0(1)	out_bu_0(1)		6.651
in_bu_0(2)	out_bu_0(2)		6.089
in_bu_0(3)	out_bu_0(3)		5.334
in_bu_0(4)	out_bu_0(4)		5.680
in_bu_0(5)	out_bu_0(5)		6.040
in_bu_0(6)	out_bu_0(6)		6.951
in_bu_0(7)	out_bu_0(7)		6.535
in_bu_0(8)	out_bu_0(8)		5.729
in_bu_0(9)	out_bu_0(9)		5.636
in_bu_0(10)	out_bu_0(10)		5.544
in_bu_0(11)	out_bu_0(11)		5.101
in_bu_0(12)	out_bu_0(12)		6.113
in_bu_0(13)	out_bu_0(13)		5.989
in_bu_0(14)	out_bu_0(14)		5.747
in_bu_0(15)	out_bu_0(15)		6.055
in_bu_1(0)	out_bu_1(0)		5.782
in_bu_1(1)	out_bu_1(1)		7.348
in_bu_1(2)	out_bu_1(2)		7.341
in_bu_1(3)	out_bu_1(3)		6.392
in_bu_1(4)	out_bu_1(4)		6.435
in_bu_1(5)	out_bu_1(5)		5.360
in_bu_1(6)	out_bu_1(6)		5.779
in_bu_1(7)	out_bu_1(7)		6.427
in_bu_1(8)	out_bu_1(8)		5.168
in_bu_1(9)	out_bu_1(9)		5.302
in_bu_1(10)	out_bu_1(10)		5.449
in_bu_1(11)	out_bu_1(11)		5.815
in_bu_1(12)	out_bu_1(12)		6.293
in_bu_1(13)	out_bu_1(13)		5.480
in_bu_2(0)	out_bu_2(0)		5.864
in_bu_2(1)	out_bu_2(1)		6.197
in_bu_2(2)	out_bu_2(2)		5.546
in_bu_2(3)	out_bu_2(3)		5.127
in_bu_2(4)	out_bu_2(4)		5.768
in_bu_2(5)	out_bu_2(5)		6.238
in_bu_2(6)	out_bu_2(6)		5.651
in_bu_2(7)	out_bu_2(7)		4.969
in_bu_2(8)	out_bu_2(8)		6.669
in_bu_2(9)	out_bu_2(9)		5.960
in_bu_2(10)	out_bu_2(10)		6.573
in_bu_2(11)	out_bu_2(11)		5.861
in_bu_3(0)	out_bu_3(0)		5.328
in_bu_3(1)	out_bu_3(1)		5.893
in_bu_3(2)	out_bu_3(2)		5.911
in_bu_3(3)	out_bu_3(3)		5.685
in_bu_3(4)	out_bu_3(4)		6.248
in_bu_3(5)	out_bu_3(5)		5.861

```

in_bu_3(6)   |out_bu_3(6)   |   6.067|
in_bu_3(7)   |out_bu_3(7)   |   5.017|
in_bu_3(8)   |out_bu_3(8)   |   4.882|
in_bu_3(9)   |out_bu_3(9)   |   5.936|
-----+-----+-----+

```

Analysis completed Mon Nov 30 18:08:07 2015

Trace Settings:

Trace Settings

Peak Memory Usage: 299 MB

C.3.5 PrecisionRTL Area Report for Stage 1 Combinational Logic

Device Utilization for 2VP30ff896

```

*****
Resource                Used    Avail    Utilization
-----
IOs                      72      556     12.95%
Global Buffers           0        16       0.00%
LUTs                     102    27392     0.37%
CLB Slices                51    13696     0.37%
Dffs or Latches          0    29060     0.00%
Block RAMs                0       136     0.00%
Block Multipliers         0       136     0.00%
Block Multiplier Dffs    0     4896     0.00%
GT_CUSTOM                 0         8     0.00%
-----

```

Library: work Cell: booth_pipeline1_comb View: rtl

Cell	Library	References	Total Area
GND	xcv2p	1 x	
IBUF	xcv2p	52 x	
OBUF	xcv2p	16 x	
OBUFT	xcv2p	1 x	
VCC	xcv2p	1 x	
csa_10bit	work	1 x	25 gates 25 LUTs
csa_12bit	work	1 x	35 gates 35 LUTs
csa_14bit	work	1 x	42 gates 42 LUTs

```

Number of ports :           72
Number of nets :           163
Number of instances :       74
Number of references to this view : 0

```



```

Total accumulated area :
  Number of LUTs :          102
  Number of gates :         102
  Number of accumulated instances : 173

```

```

*****
IO Register Mapping Report
*****

```

```

Design: work.booth_pipeline1_comb.rtl

```

Port	Direction	INFF	OUTFF	TRIFF
bu_0_in(15)	Input			
bu_0_in(14)	Input			
bu_0_in(13)	Input			
bu_0_in(12)	Input			
bu_0_in(11)	Input			
bu_0_in(10)	Input			
bu_0_in(9)	Input			
bu_0_in(8)	Input			
bu_0_in(7)	Input			
bu_0_in(6)	Input			
bu_0_in(5)	Input			
bu_0_in(4)	Input			
bu_0_in(3)	Input			
bu_0_in(2)	Input			
bu_0_in(1)	Input			
bu_0_in(0)	Input			
bu_1_in(13)	Input			
bu_1_in(12)	Input			
bu_1_in(11)	Input			
bu_1_in(10)	Input			
bu_1_in(9)	Input			
bu_1_in(8)	Input			

bu_1_in(7)	Input			
bu_1_in(6)	Input			
bu_1_in(5)	Input			
bu_1_in(4)	Input			
bu_1_in(3)	Input			
bu_1_in(2)	Input			
bu_1_in(1)	Input			
bu_1_in(0)	Input			
bu_2_in(11)	Input			
bu_2_in(10)	Input			
bu_2_in(9)	Input			
bu_2_in(8)	Input			
bu_2_in(7)	Input			
bu_2_in(6)	Input			
bu_2_in(5)	Input			
bu_2_in(4)	Input			
bu_2_in(3)	Input			
bu_2_in(2)	Input			
bu_2_in(1)	Input			
bu_2_in(0)	Input			
bu_3_in(9)	Input			
bu_3_in(8)	Input			
bu_3_in(7)	Input			
bu_3_in(6)	Input			
bu_3_in(5)	Input			
bu_3_in(4)	Input			
bu_3_in(3)	Input			
bu_3_in(2)	Input			
bu_3_in(1)	Input			

bu_3_in(0)	Input			
data_valid_in	Input			
clk	Input			
clr	Input			
result_out(15)	Output			
result_out(14)	Output			
result_out(13)	Output			
result_out(12)	Output			
result_out(11)	Output			
result_out(10)	Output			
result_out(9)	Output			
result_out(8)	Output			
result_out(7)	Output			
result_out(6)	Output			
result_out(5)	Output			
result_out(4)	Output			
result_out(3)	Output			
result_out(2)	Output			
result_out(1)	Output			
result_out(0)	Output			
data_valid_out	Output			

Total registers mapped: 0

C.3.6 Xilinx ISE Timing Report for Stage 1 Combinational Logic

Release 10.1 Trace (lin64)

Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.

```
/nfs/sw_cmc/linux-64/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce -ise
/nfs/home/r/r_fenste/COEN6501/synth_test2/stage1_comb/ise/stage1_comb/stage1_comb.ise
-intstyle ise -e 3 -s 7 -xml booth_pipeline1_comb booth_pipeline1_comb.ncd -o
booth_pipeline1_comb.twr booth_pipeline1_comb.pcf -ucf
/nfs/home/r/r_fenste/COEN6501/synth_test2/stage1_comb/stage1_comb_impl_1/booth_pipeline1_comb.ucf
```

Design file: booth_pipeline1_comb.ncd

Physical constraint file: booth_pipeline1_comb.pcf
 Device,package,speed: xc2vp30,ff896,-7 (PRODUCTION 1.94 2008-01-09)
 Report level: error report

Environment Variable	Effect
NONE	No environment variables were set

INFO:Timing:2698 - No timing constraints found, doing default enumeration.
 INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths option. All paths that are not constrained will be reported in the unconstrained paths section(s) of the report.
 INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on a 50 Ohm transmission line loading model. For the details of this model, and for more information on accounting for different loading conditions, please see the device datasheet.

Data Sheet report:

All values displayed in nanoseconds (ns)

Pad to Pad

Source Pad	Destination Pad	Delay
bu_0_in(0)	result_out(0)	3.155
bu_0_in(1)	result_out(1)	3.339
bu_0_in(2)	result_out(2)	4.124
bu_0_in(2)	result_out(3)	4.357
bu_0_in(2)	result_out(4)	6.506
bu_0_in(2)	result_out(5)	6.333
bu_0_in(2)	result_out(6)	7.693
bu_0_in(2)	result_out(7)	8.716
bu_0_in(2)	result_out(8)	9.261
bu_0_in(2)	result_out(9)	10.513
bu_0_in(2)	result_out(10)	10.298
bu_0_in(2)	result_out(11)	10.781
bu_0_in(2)	result_out(12)	10.854
bu_0_in(2)	result_out(13)	11.129
bu_0_in(2)	result_out(14)	10.477
bu_0_in(2)	result_out(15)	10.881
bu_0_in(3)	result_out(3)	4.469
bu_0_in(3)	result_out(4)	6.329
bu_0_in(3)	result_out(5)	6.156
bu_0_in(3)	result_out(6)	7.516
bu_0_in(3)	result_out(7)	8.539
bu_0_in(3)	result_out(8)	9.084
bu_0_in(3)	result_out(9)	10.336
bu_0_in(3)	result_out(10)	10.121
bu_0_in(3)	result_out(11)	10.604
bu_0_in(3)	result_out(12)	10.677
bu_0_in(3)	result_out(13)	10.952
bu_0_in(3)	result_out(14)	10.300
bu_0_in(3)	result_out(15)	10.704
bu_0_in(4)	result_out(4)	5.146

bu_0_in(4)	result_out(5)		5.195
bu_0_in(4)	result_out(6)		6.540
bu_0_in(4)	result_out(7)		7.361
bu_0_in(4)	result_out(8)		7.906
bu_0_in(4)	result_out(9)		9.158
bu_0_in(4)	result_out(10)		8.943
bu_0_in(4)	result_out(11)		9.426
bu_0_in(4)	result_out(12)		9.499
bu_0_in(4)	result_out(13)		9.774
bu_0_in(4)	result_out(14)		9.122
bu_0_in(4)	result_out(15)		9.526
bu_0_in(5)	result_out(5)		5.059
bu_0_in(5)	result_out(6)		6.252
bu_0_in(5)	result_out(7)		7.275
bu_0_in(5)	result_out(8)		7.820
bu_0_in(5)	result_out(9)		9.072
bu_0_in(5)	result_out(10)		8.857
bu_0_in(5)	result_out(11)		9.340
bu_0_in(5)	result_out(12)		9.413
bu_0_in(5)	result_out(13)		9.688
bu_0_in(5)	result_out(14)		9.036
bu_0_in(5)	result_out(15)		9.440
bu_0_in(6)	result_out(6)		5.772
bu_0_in(6)	result_out(7)		5.716
bu_0_in(6)	result_out(8)		6.085
bu_0_in(6)	result_out(9)		7.337
bu_0_in(6)	result_out(10)		7.770
bu_0_in(6)	result_out(11)		8.458
bu_0_in(6)	result_out(12)		9.310
bu_0_in(6)	result_out(13)		9.585
bu_0_in(6)	result_out(14)		9.134
bu_0_in(6)	result_out(15)		9.538
bu_0_in(7)	result_out(7)		5.084
bu_0_in(7)	result_out(8)		5.629
bu_0_in(7)	result_out(9)		6.881
bu_0_in(7)	result_out(10)		7.248
bu_0_in(7)	result_out(11)		7.936
bu_0_in(7)	result_out(12)		8.788
bu_0_in(7)	result_out(13)		9.063
bu_0_in(7)	result_out(14)		8.612
bu_0_in(7)	result_out(15)		9.016
bu_0_in(8)	result_out(8)		5.710
bu_0_in(8)	result_out(9)		6.403
bu_0_in(8)	result_out(10)		7.347
bu_0_in(8)	result_out(11)		8.035
bu_0_in(8)	result_out(12)		8.887
bu_0_in(8)	result_out(13)		9.162
bu_0_in(8)	result_out(14)		8.711
bu_0_in(8)	result_out(15)		9.115
bu_0_in(9)	result_out(9)		5.764
bu_0_in(9)	result_out(10)		6.373
bu_0_in(9)	result_out(11)		7.061
bu_0_in(9)	result_out(12)		7.913
bu_0_in(9)	result_out(13)		8.188
bu_0_in(9)	result_out(14)		7.737
bu_0_in(9)	result_out(15)		8.141
bu_0_in(10)	result_out(10)		5.767
bu_0_in(10)	result_out(11)		6.643

bu_0_in(10)	result_out(12)	7.391
bu_0_in(10)	result_out(13)	7.666
bu_0_in(10)	result_out(14)	8.666
bu_0_in(10)	result_out(15)	8.348
bu_0_in(11)	result_out(11)	6.088
bu_0_in(11)	result_out(12)	6.655
bu_0_in(11)	result_out(13)	6.930
bu_0_in(11)	result_out(14)	7.808
bu_0_in(11)	result_out(15)	7.502
bu_0_in(12)	result_out(12)	5.929
bu_0_in(12)	result_out(13)	6.585
bu_0_in(12)	result_out(14)	7.620
bu_0_in(12)	result_out(15)	7.407
bu_0_in(13)	result_out(13)	7.134
bu_0_in(13)	result_out(14)	8.412
bu_0_in(13)	result_out(15)	8.027
bu_0_in(14)	result_out(14)	6.444
bu_0_in(14)	result_out(15)	6.059
bu_0_in(15)	result_out(15)	5.698
bu_1_in(0)	result_out(2)	3.761
bu_1_in(0)	result_out(3)	3.726
bu_1_in(0)	result_out(4)	5.840
bu_1_in(0)	result_out(5)	5.667
bu_1_in(0)	result_out(6)	7.027
bu_1_in(0)	result_out(7)	8.050
bu_1_in(0)	result_out(8)	8.595
bu_1_in(0)	result_out(9)	9.847
bu_1_in(0)	result_out(10)	9.632
bu_1_in(0)	result_out(11)	10.115
bu_1_in(0)	result_out(12)	10.188
bu_1_in(0)	result_out(13)	10.463
bu_1_in(0)	result_out(14)	9.811
bu_1_in(0)	result_out(15)	10.215
bu_1_in(1)	result_out(3)	3.758
bu_1_in(1)	result_out(4)	5.824
bu_1_in(1)	result_out(5)	5.651
bu_1_in(1)	result_out(6)	7.011
bu_1_in(1)	result_out(7)	8.034
bu_1_in(1)	result_out(8)	8.579
bu_1_in(1)	result_out(9)	9.831
bu_1_in(1)	result_out(10)	9.616
bu_1_in(1)	result_out(11)	10.099
bu_1_in(1)	result_out(12)	10.172
bu_1_in(1)	result_out(13)	10.447
bu_1_in(1)	result_out(14)	9.795
bu_1_in(1)	result_out(15)	10.199
bu_1_in(2)	result_out(4)	5.031
bu_1_in(2)	result_out(5)	4.829
bu_1_in(2)	result_out(6)	5.999
bu_1_in(2)	result_out(7)	7.022
bu_1_in(2)	result_out(8)	7.567
bu_1_in(2)	result_out(9)	8.819
bu_1_in(2)	result_out(10)	8.604
bu_1_in(2)	result_out(11)	9.087
bu_1_in(2)	result_out(12)	9.160
bu_1_in(2)	result_out(13)	9.435
bu_1_in(2)	result_out(14)	8.783
bu_1_in(2)	result_out(15)	9.187

bu_1_in(3)	result_out(5)		4.673
bu_1_in(3)	result_out(6)		5.866
bu_1_in(3)	result_out(7)		6.889
bu_1_in(3)	result_out(8)		7.434
bu_1_in(3)	result_out(9)		8.686
bu_1_in(3)	result_out(10)		8.471
bu_1_in(3)	result_out(11)		8.954
bu_1_in(3)	result_out(12)		9.027
bu_1_in(3)	result_out(13)		9.302
bu_1_in(3)	result_out(14)		8.650
bu_1_in(3)	result_out(15)		9.054
bu_1_in(4)	result_out(6)		5.294
bu_1_in(4)	result_out(7)		5.835
bu_1_in(4)	result_out(8)		6.380
bu_1_in(4)	result_out(9)		7.632
bu_1_in(4)	result_out(10)		7.694
bu_1_in(4)	result_out(11)		8.382
bu_1_in(4)	result_out(12)		9.234
bu_1_in(4)	result_out(13)		9.509
bu_1_in(4)	result_out(14)		9.058
bu_1_in(4)	result_out(15)		9.462
bu_1_in(5)	result_out(7)		5.456
bu_1_in(5)	result_out(8)		6.001
bu_1_in(5)	result_out(9)		7.253
bu_1_in(5)	result_out(10)		7.746
bu_1_in(5)	result_out(11)		8.434
bu_1_in(5)	result_out(12)		9.286
bu_1_in(5)	result_out(13)		9.561
bu_1_in(5)	result_out(14)		9.110
bu_1_in(5)	result_out(15)		9.514
bu_1_in(6)	result_out(8)		5.361
bu_1_in(6)	result_out(9)		6.213
bu_1_in(6)	result_out(10)		7.396
bu_1_in(6)	result_out(11)		8.084
bu_1_in(6)	result_out(12)		8.936
bu_1_in(6)	result_out(13)		9.211
bu_1_in(6)	result_out(14)		8.760
bu_1_in(6)	result_out(15)		9.164
bu_1_in(7)	result_out(9)		5.895
bu_1_in(7)	result_out(10)		6.504
bu_1_in(7)	result_out(11)		7.192
bu_1_in(7)	result_out(12)		8.044
bu_1_in(7)	result_out(13)		8.319
bu_1_in(7)	result_out(14)		7.868
bu_1_in(7)	result_out(15)		8.272
bu_1_in(8)	result_out(10)		5.568
bu_1_in(8)	result_out(11)		6.876
bu_1_in(8)	result_out(12)		7.443
bu_1_in(8)	result_out(13)		7.718
bu_1_in(8)	result_out(14)		8.286
bu_1_in(8)	result_out(15)		7.980
bu_1_in(9)	result_out(11)		6.716
bu_1_in(9)	result_out(12)		7.283
bu_1_in(9)	result_out(13)		7.558
bu_1_in(9)	result_out(14)		8.152
bu_1_in(9)	result_out(15)		7.846
bu_1_in(10)	result_out(12)		5.663
bu_1_in(10)	result_out(13)		6.667

bu_1_in(10)	result_out(14)	7.505
bu_1_in(10)	result_out(15)	7.489
bu_1_in(11)	result_out(13)	7.410
bu_1_in(11)	result_out(14)	8.688
bu_1_in(11)	result_out(15)	8.303
bu_1_in(12)	result_out(14)	6.801
bu_1_in(12)	result_out(15)	6.416
bu_1_in(13)	result_out(15)	6.295
bu_2_in(0)	result_out(4)	4.481
bu_2_in(0)	result_out(5)	3.782
bu_2_in(0)	result_out(6)	4.967
bu_2_in(0)	result_out(7)	4.848
bu_2_in(0)	result_out(8)	5.370
bu_2_in(0)	result_out(9)	6.622
bu_2_in(0)	result_out(10)	6.407
bu_2_in(0)	result_out(11)	6.890
bu_2_in(0)	result_out(12)	6.826
bu_2_in(0)	result_out(13)	7.101
bu_2_in(0)	result_out(14)	6.308
bu_2_in(0)	result_out(15)	6.578
bu_2_in(1)	result_out(5)	3.711
bu_2_in(1)	result_out(6)	5.074
bu_2_in(1)	result_out(7)	4.955
bu_2_in(1)	result_out(8)	5.477
bu_2_in(1)	result_out(9)	6.729
bu_2_in(1)	result_out(10)	6.514
bu_2_in(1)	result_out(11)	6.997
bu_2_in(1)	result_out(12)	6.933
bu_2_in(1)	result_out(13)	7.208
bu_2_in(1)	result_out(14)	6.415
bu_2_in(1)	result_out(15)	6.685
bu_2_in(2)	result_out(6)	3.925
bu_2_in(2)	result_out(7)	5.084
bu_2_in(2)	result_out(8)	5.629
bu_2_in(2)	result_out(9)	6.881
bu_2_in(2)	result_out(10)	7.216
bu_2_in(2)	result_out(11)	7.573
bu_2_in(2)	result_out(12)	8.140
bu_2_in(2)	result_out(13)	8.415
bu_2_in(2)	result_out(14)	7.622
bu_2_in(2)	result_out(15)	8.512
bu_2_in(3)	result_out(7)	5.020
bu_2_in(3)	result_out(8)	5.565
bu_2_in(3)	result_out(9)	6.817
bu_2_in(3)	result_out(10)	7.250
bu_2_in(3)	result_out(11)	7.607
bu_2_in(3)	result_out(12)	8.174
bu_2_in(3)	result_out(13)	8.449
bu_2_in(3)	result_out(14)	7.656
bu_2_in(3)	result_out(15)	8.546
bu_2_in(4)	result_out(8)	4.791
bu_2_in(4)	result_out(9)	6.273
bu_2_in(4)	result_out(10)	7.301
bu_2_in(4)	result_out(11)	7.501
bu_2_in(4)	result_out(12)	8.012
bu_2_in(4)	result_out(13)	8.287
bu_2_in(4)	result_out(14)	7.520
bu_2_in(4)	result_out(15)	8.597

bu_2_in(5)	result_out(9)		6.028
bu_2_in(5)	result_out(10)		6.889
bu_2_in(5)	result_out(11)		7.089
bu_2_in(5)	result_out(12)		7.764
bu_2_in(5)	result_out(13)		8.039
bu_2_in(5)	result_out(14)		7.246
bu_2_in(5)	result_out(15)		8.185
bu_2_in(6)	result_out(10)		6.149
bu_2_in(6)	result_out(11)		6.740
bu_2_in(6)	result_out(12)		7.307
bu_2_in(6)	result_out(13)		7.582
bu_2_in(6)	result_out(14)		7.328
bu_2_in(6)	result_out(15)		8.405
bu_2_in(7)	result_out(11)		6.831
bu_2_in(7)	result_out(12)		7.398
bu_2_in(7)	result_out(13)		7.673
bu_2_in(7)	result_out(14)		6.892
bu_2_in(7)	result_out(15)		7.969
bu_2_in(8)	result_out(12)		6.204
bu_2_in(8)	result_out(13)		6.434
bu_2_in(8)	result_out(14)		6.711
bu_2_in(8)	result_out(15)		7.788
bu_2_in(9)	result_out(13)		5.900
bu_2_in(9)	result_out(14)		6.900
bu_2_in(9)	result_out(15)		7.977
bu_2_in(10)	result_out(14)		5.798
bu_2_in(10)	result_out(15)		6.456
bu_2_in(11)	result_out(15)		6.851
bu_3_in(0)	result_out(6)		4.590
bu_3_in(0)	result_out(7)		5.099
bu_3_in(0)	result_out(8)		5.644
bu_3_in(0)	result_out(9)		6.896
bu_3_in(0)	result_out(10)		7.220
bu_3_in(0)	result_out(11)		7.577
bu_3_in(0)	result_out(12)		8.144
bu_3_in(0)	result_out(13)		8.419
bu_3_in(0)	result_out(14)		7.626
bu_3_in(0)	result_out(15)		8.516
bu_3_in(1)	result_out(7)		5.224
bu_3_in(1)	result_out(8)		5.769
bu_3_in(1)	result_out(9)		7.021
bu_3_in(1)	result_out(10)		7.334
bu_3_in(1)	result_out(11)		7.691
bu_3_in(1)	result_out(12)		8.258
bu_3_in(1)	result_out(13)		8.533
bu_3_in(1)	result_out(14)		7.740
bu_3_in(1)	result_out(15)		8.630
bu_3_in(2)	result_out(8)		4.551
bu_3_in(2)	result_out(9)		5.746
bu_3_in(2)	result_out(10)		6.584
bu_3_in(2)	result_out(11)		7.119
bu_3_in(2)	result_out(12)		7.686
bu_3_in(2)	result_out(13)		7.961
bu_3_in(2)	result_out(14)		7.168
bu_3_in(2)	result_out(15)		7.880
bu_3_in(3)	result_out(9)		6.982
bu_3_in(3)	result_out(10)		7.843
bu_3_in(3)	result_out(11)		8.043

```

bu_3_in(3) |result_out(12) | 8.718|
bu_3_in(3) |result_out(13) | 8.993|
bu_3_in(3) |result_out(14) | 8.200|
bu_3_in(3) |result_out(15) | 9.139|
bu_3_in(4) |result_out(10) | 6.544|
bu_3_in(4) |result_out(11) | 7.307|
bu_3_in(4) |result_out(12) | 7.874|
bu_3_in(4) |result_out(13) | 8.149|
bu_3_in(4) |result_out(14) | 7.466|
bu_3_in(4) |result_out(15) | 8.543|
bu_3_in(5) |result_out(11) | 7.154|
bu_3_in(5) |result_out(12) | 7.721|
bu_3_in(5) |result_out(13) | 7.996|
bu_3_in(5) |result_out(14) | 7.203|
bu_3_in(5) |result_out(15) | 8.229|
bu_3_in(6) |result_out(12) | 6.574|
bu_3_in(6) |result_out(13) | 7.117|
bu_3_in(6) |result_out(14) | 7.394|
bu_3_in(6) |result_out(15) | 8.471|
bu_3_in(7) |result_out(13) | 5.700|
bu_3_in(7) |result_out(14) | 6.545|
bu_3_in(7) |result_out(15) | 7.622|
bu_3_in(8) |result_out(14) | 6.907|
bu_3_in(8) |result_out(15) | 6.607|
bu_3_in(9) |result_out(15) | 6.415|
-----+-----+-----+

```

Analysis completed Mon Nov 30 18:16:46 2015

Trace Settings:

Trace Settings

Peak Memory Usage: 297 MB

C.3.7 PrecisionRTL Area Report for Stage 1 Registers

Device Utilization for 2VP20ff896

```

*****
Resource                Used    Avail    Utilization
-----
IOs                      36      556      6.47%
Global Buffers          0         16      0.00%
LUTs                     68    18560     0.37%
CLB Slices               34     9280     0.37%
Ddfs or Latches         0    20228     0.00%
Block RAMs              0         88      0.00%
Block Multipliers       0         88      0.00%
Block Multiplier Ddfs   0     3168     0.00%
GT_CUSTOM               0         8       0.00%
-----

```

Library: work Cell: booth_pipeline1_reg View: rtl

Cell	Library	References	Total Area
IBUF	xcv2p	19 x	
LUT3	xcv2p	1 x 1	1 LUTs
LUT4	xcv2p	3 x 1	3 LUTs
OBUF	xcv2p	17 x	
operand_register_16bit	work	1 x 65	65 gates
		64	64 LUTs

Number of ports : 36
Number of nets : 75
Number of instances : 41
Number of references to this view : 0

Total accumulated area :
Number of LUTs : 68
Number of gates : 69
Number of accumulated instances : 104

IO Register Mapping Report

Design: work.booth_pipeline1_reg.rtl

Port	Direction	INFF	OUTFF	TRIFF
data_valid_in	Input			
clk	Input			
clr	Input			
result_in(15)	Input			
result_in(14)	Input			
result_in(13)	Input			
result_in(12)	Input			
result_in(11)	Input			
result_in(10)	Input			
result_in(9)	Input			
result_in(8)	Input			
result_in(7)	Input			
result_in(6)	Input			
result_in(5)	Input			

result_in(4)	Input			
result_in(3)	Input			
result_in(2)	Input			
result_in(1)	Input			
result_in(0)	Input			
result_out(15)	Output			
result_out(14)	Output			
result_out(13)	Output			
result_out(12)	Output			
result_out(11)	Output			
result_out(10)	Output			
result_out(9)	Output			
result_out(8)	Output			
result_out(7)	Output			
result_out(6)	Output			
result_out(5)	Output			
result_out(4)	Output			
result_out(3)	Output			
result_out(2)	Output			
result_out(1)	Output			
result_out(0)	Output			
data_valid_out	Output			

Total registers mapped: 0

C.3.8 Xilinx ISE Timing Report for Stage 1 Registers

Release 10.1 Trace (lin64)

Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.

```
/nfs/sw_cmc/linux-64/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce -ise
/nfs/home/r/r_fenste/COEN6501/synth_test2/stage1_reg/ise/stage1_reg/stage1_reg.ise
-intstyle ise -e 3 -s 7 -xml booth_pipeline1_reg booth_pipeline1_reg.ncd -o
booth_pipeline1_reg.twr booth_pipeline1_reg.pcf -ucf booth_pipeline1_reg.ucf
```

Design file: booth_pipeline1_reg.ncd
 Physical constraint file: booth_pipeline1_reg.pcf
 Device,package,speed: xc2vp20,ff896,-7 (PRODUCTION 1.94 2008-01-09)
 Report level: error report

Environment Variable	Effect
NONE	No environment variables were set

INFO:Timing:2698 - No timing constraints found, doing default enumeration.
 INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths option. All paths that are not constrained will be reported in the unconstrained paths section(s) of the report.
 INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on a 50 Ohm transmission line loading model. For the details of this model, and for more information on accounting for different loading conditions, please see the device datasheet.

Data Sheet report:

All values displayed in nanoseconds (ns)

Pad to Pad

Source Pad	Destination Pad	Delay
clk	data_valid_out	5.925
clk	result_out(0)	5.491
clk	result_out(1)	7.073
clk	result_out(2)	5.882
clk	result_out(3)	5.826
clk	result_out(4)	5.869
clk	result_out(5)	6.408
clk	result_out(6)	4.892
clk	result_out(7)	5.056
clk	result_out(8)	4.903
clk	result_out(9)	4.951
clk	result_out(10)	6.156
clk	result_out(11)	5.328
clk	result_out(12)	6.154
clk	result_out(13)	5.396
clk	result_out(14)	6.324
clk	result_out(15)	6.989
clr	data_valid_out	6.159
clr	result_out(0)	6.493
clr	result_out(1)	6.975
clr	result_out(2)	6.649
clr	result_out(3)	6.069
clr	result_out(4)	6.274
clr	result_out(5)	6.207
clr	result_out(6)	5.501
clr	result_out(7)	5.113
clr	result_out(8)	5.287
clr	result_out(9)	5.463
clr	result_out(10)	6.587

clr	result_out(11)	5.917
clr	result_out(12)	6.085
clr	result_out(13)	5.985
clr	result_out(14)	7.074
clr	result_out(15)	6.949
data_valid_in	data_valid_out	4.972
result_in(0)	result_out(0)	5.497
result_in(1)	result_out(1)	6.323
result_in(2)	result_out(2)	5.931
result_in(3)	result_out(3)	5.446
result_in(4)	result_out(4)	5.635
result_in(5)	result_out(5)	5.745
result_in(6)	result_out(6)	5.059
result_in(7)	result_out(7)	5.020
result_in(8)	result_out(8)	5.281
result_in(9)	result_out(9)	4.962
result_in(10)	result_out(10)	5.578
result_in(11)	result_out(11)	5.055
result_in(12)	result_out(12)	5.003
result_in(13)	result_out(13)	5.092
result_in(14)	result_out(14)	5.874
result_in(15)	result_out(15)	5.825

-----+-----+-----+

Analysis completed Mon Nov 30 18:23:49 2015

Trace Settings:

Trace Settings

Peak Memory Usage: 271 MB