

COEN 6501 Project Fall 2021 Report

Jun Huang, Dawei Zuo, Yuelin Yao, and Xuesi Feng

Department of Electrical and Computer Engineering, Concordia University

COEN 6501: Digital Design and Synthesis

Dr. Marwan Ammar

December 6th, 2021

Table of Contents

Introduction	7
Project Requirement	8
Carry Select Adder	9
Full Adder	9
Two to one multiplexer	10
4-bit Ripple Carry Adder	11
4-bit Carry Select Adder	12
16-bit Carry Select Adder	14
24-bit Carry Select Adder	15
24-bit CSA Incrementor	16
Multiplication in Three Operands	17
Radix-4 Booth Algorithm Logic in Details	17
Example in Signed Number Multiplication	17
Example in Unsigned Number Multiplication	18
8-bit Radix-4 Booth Multiplier Circuit Implementation	19
Overall Circuit Design and RTL Description	19
Blocks Design	19
8-bit Triple Operands Multiplier Circuit Implementation	23
16-bit Triple Operands Multiplier Circuit Implementation	25
Overflow Handling	26
Method 1: Negation Output	26
Overflow Detection	26
Handler Implementation	26
Method 2: Display Z Separately with Two Clock Cycles	26
End Flag Generator	28
Non-pipelined Implementation	29
Operating Circuit	29

Output Process	30
Pipelined Implementation	31
8-bit Operands Implementation with Negation Overflow Handler	31
8-bit Operands Implementation with Output Separation Overflow Handler	31
16-bit Operands Implementation and Simulation	34
Synthesis and Analysis of the Arithmetic Circuit	35
Introduction	35
Timing Information	35
Radix-4 Booth Multiplier	35
CSA	35
Implementation Circuit	35
Area and Power Results for the Non-Pipelined Version	36
Implementation with Negation Output	36
Area and Power Results for the Pipelined Version	36
Implementation with Negation Output	36
Implementation with Separation Output	37
Appendices	38
Area Report for Non-Pipelined Version with Negation Output	38
Area Report for Pipelined Version with Negation Output	40
Area Report for Pipelined Version with Separation Output	42
Timing Analysis Report for 8-bit Radix-4 Booth Multiplier Component	44
Timing Analysis Report for 16-bit CSA Component	47
Timing Analysis Report for 24-bit CSA Component	52
Timing Analysis Report for Non-pipelined Implementation with Negation Output	61
Timing Analysis Report for Pipelined Implementation with Negation Output	62
Timing Analysis Report for Pipelined Implementation with Separation Output	64

List of Figures

1	The RTL Diagram of the Required ALU	8
2	Synthesized RTL Diagram of Full Adder Block	10
3	Simulation Wave Diagram of Full Adder Block	10
4	Synthesized RTL Diagram of 2-to-1 Multiplexer Block	11
5	Simulation Wave Diagram of 2-to-1 Multiplexer Block	11
6	Synthesized RTL Diagram of 4-bit Ripple Carry Adder Block	12
7	Simulation Wave Diagram of 4-bit Ripple Carry Adder Block	12
8	Synthesized RTL Diagram of 4-bit Carry Select Adder Block	13
9	Simulation Wave Diagram of 4-bit Carry Select Adder Block	13
10	Synthesized RTL Diagram of 16-bit Carry Select Adder Block	14
11	Simulation Wave Diagram of 16-bit Carry Select Adder Block	14
12	Synthesized RTL Diagram of 24-bit Carry Select Adder Block	15
13	Simulation Wave Diagram of 24-bit Carry Select Adder Block	15
14	Synthesized RTL Diagram of 24-bit Incrementor Block	16
15	Simulation Wave Diagram of 24-bit Incrementor Block	16
16	The Decoded Code Blocks of the Signed Multiplier	18
17	Process of the Algorithm for Signed Number	18
18	The Decoded Code Blocks of the Unsigned Multiplier	19
19	Process of the Slgorithm for Unisgned Number	19
20	Synthesized RTL Diagram of 8-bit Radix-4 Booth Multiplier	20
21	Synthesized RTL Diagram of Complement Generator	20
22	Simulation Wave Diagram of Complement Generator	20
23	Synthesized RTL Diagram of Booth Stage 0 Block	21
24	Simulation Wave Diagram of Booth Stage 0 Block	22
25	Synthesized RTL Diagram of Booth Stage 1 Block	22
26	Simulation Wave Diagram of Booth Stage 1 Block	23
27	Synthesized RTL Diagram of Triple 8-bit Operands Multiplier Circuit	24
28	Simulation Wave Diagram of Triple 8-bit Operands Multiplier Circuit	24
29	Synthesized RTL Diagram of Triple 16-bit Operands Multiplier Circuit	25
30	Simulation Wave Diagram of Triple 16-bit Operands Multiplier Circuit	25
31	Synthesized RTL Diagram of the Zero Detector	26

32	Simulation Wave Diagram of the Zero Detector	26
33	Synthesized RTL Diagram of the Overflow Handler of Method 1	27
34	Simulation Wave Diagram of the Overflow Handler of Method 1	27
35	Synthesized RTL Diagram of the Overflow Handler of Method 2	27
36	Simulation Wave Diagram of the Overflow Handler of Method 2	27
37	ASMD Chart of the FSM of Different End Flag Generator	28
38	Synthesized RTL Diagram of Non-pipelined Operating Circuit	29
39	Simulation Wave Diagram of Non-pipelined Operating Circuit	29
40	Synthesized RTL Diagram of Non-pipelined Circuit	30
41	Simulation Wave Diagram of Non-pipelined Circuit	30
42	Simulation Wave Diagram of the 8-bit Pipelined ALU Circuit with Negation Output	32
43	Synthesized RTL Diagram of the 8-bit Operands Pipelined ALU Circuit and the Arithmetic Operation Block	32
44	Simulation Wave Diagram of the 8-bit Pipelined ALU Circuit with Separation Output	33
45	Synthesized Diagram of the 8-bit Pipelined ALU Circuit with Separation Output	33
46	Simulation Wave Diagram of the 16-bit Pipelined ALU Circuit with Negation Output	34
47	Synthesized RTL Diagram of 16-bit Radix-4 Booth Multiplier	67

List of Tables

1	Components of the ALU Circuit	7
2	Full Adder Truth Table	9
3	2-to-1 Multiplexer Truth Table	11
4	Code Table of Radix-4 Booth Algorithm	17
5	A Visual Representation Table of the Pipelined Processe with Negation Overflow Handler	31
6	A Visual Representation Table of the Pipelined Processing with Output Separation Overflow Handler .	33
7	Power Information for Non-pipelined Version	36
8	Power Information for Pipelined Version with Negation Output	36
9	Power Information for Pipelined Version with Separation Output	37

Introduction

This is the project report of the course **COEN 6501: Digital Design and Synthesis**. The project requires an implementation of an ALU circuit design that perform the equation of $Z = \frac{1}{4}[A^2 * B] + 1$. More details of the requirements will be discussed in the next section. Breaking down the project requirements, the team should implement the following component designs to complete the ALU:

Table 1

Components of the ALU Circuit

Functional Components	Other Components
n-bit adder design	2-bit right shifter
n-bit multiplier with 2 and 3 operands	n-bit incrementor
overflow handler	negative edge registers
<i>END_FLAG</i> generator	

As per adder design, the team chose the “**Carry Select Adder**” implementation. This adder design is suitable for bit width extension.

As per multiplier design, the team chose the “**Radix-4 Booth Algorithm**” implementation which is an elegant design for reducing both area occupation and delay.

As per the overflow handler, the team provides two solutions: 1. giving a negation which is impossible for the demanded equation to indicate the overflow; 2. dividing the full result into the higher part and the lower part and outputting them to the *Z* port alternately within two clock cycles.

As per the *END_FLAG* generator, the team chose to implement a FSM for indicating the valid *END_FLAG* signal. As per the register, falling edge clock event sensitive register will be used in the project, the registers for input *A* and *B* should use *LOAD* signal as the clock event which means those two registers are asynchronous, the rest of the registers should use *Clock* signal as the clock event.

After completing all component designs, the ALU will be implemented in both pipelined and non-pipelined designs.

The project is designed, implemented, simulated, and synthesized with the environments of:

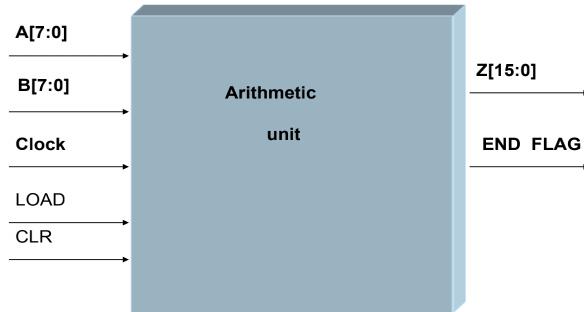
- **ModelSim SE-64 6.6g May 23 2012 Linux 3.10.0-1160.45.1.el7.x86_64**
- **Precision RTL Synthesis 64-bit 2016.1.0.15 (Production Release)**
- **XILINX ISE and Project Navigator 10.1 (lin64)**
- **GHDL 2.0.0-dev (1.0.0.r849.gc56db233)**

The synthesized report is based on the “2VP20ff896” device with default timing constraints provided by *Precision*.

Project Requirement

The project should implement ALU circuit described in Figure 1

Figure 1
The RTL Diagram of the Required ALU



The project contains the following requirements for signals:

- The operands A and B are latched into register RA and RB when $LOAD$ signal transit from high to low.
- The unit outputs the results in 16-bit register RZ output port.
- Each calculation starts with a $LOAD$ signal and ends with an END_FLAG signal.
- The $CLEAR$ signal will clear all registers to '0'.
- The unit performs the arithmetic operation until END_FLAG becomes high.
- The 16-bit product shall be loaded into the 16-bit Z port.
- The design shall be structural.

The project contains the following extra features:

- **(Accomplished)** Expansion of the method for 16 bits operand.
- **(Accomplished)** Pipelining of the design.
- Multiply Accumulate for additional operands.

Carry Select Adder

During the implementation of a multiplier, adders are needed. Different adder choices can have different effects on the delay and area. The outputs of ripple carry adder rely on the output carry of lower levels, so the RCA has an extremely long output chain and path. A carry select adder has a pair of Ripple Carry Adder performing the addition of a chunk of the two operands and a multiplexer to select the correct sum and carry out from the two RCAs. Compared to RCA, the carry select adder is a more efficient parallel adder.

In carry select adder, both sum and carry outputs are calculated for two alternatives: the input carry C_{in} '0' and '1'. Once the input carry is loaded, the correct calculation is chosen by a multiplexer to produce the desired output. Instead of waiting for the carry to calculate the sum, the sum will be correctly output as soon as the input carry delivered. The time used to compute the sum is then reduced that results in a good improvement in speed. Also, it can be formed into higher bit adders by cascading. So that extending the algorithm will be easier with the usage of CSAs.

The following will introduce the structures of the desired carry select adders used in this project.

Full Adder

The full adder is the fundamental digital component of various arithmetic logic unit, the circuit, which is composed of **XOR** gate, **AND** gate and **OR** gate, adds three inputs and produces two outputs. The full adder differs from the half adder in that the full adder has an input carry C_{in} , so that it can handle the carry in from the lower bit and output its carry out.

Multiple one-bit full adders can be cascaded to obtain a multi-bit full adder, which is used as a method to design subsequent circuits. The full adder is more widely used due to the feature that it can perform the addition of three bits. But at the same time, it requires additional gates. As a result, its delay increases. The truth table for the full adder is shown in Table 2.

Table 2
Full Adder Truth Table

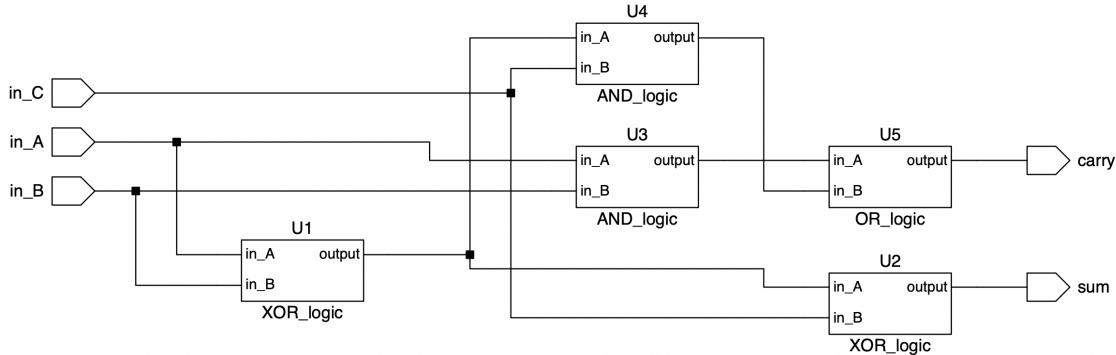
A	B	Carry _{in}	Carry _{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The equation for the full adder is defined at Expression (1).

$$\begin{aligned} \text{Sum} &= A \oplus B \oplus C_{in} \\ C_{out} &= (A \bullet B) + (C_{in} \bullet (A \oplus B)) \end{aligned} \quad (1)$$

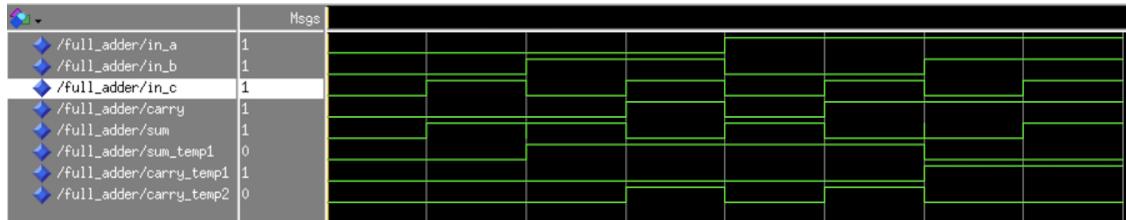
The circuit for the full adder is shown in Figure 2.

Figure 2
Synthesized RTL Diagram of Full Adder Block



The simulation results for the full adder are shown in Figure 3.

Figure 3
Simulation Wave Diagram of Full Adder Block



Two to one multiplexer

The 2-to-1 multiplexer is a selector that has a switch to control the input, the circuit which consists of **AND** gate, **OR** gate and **NOT** gate. For a 2-to-1 multiplexer, the inputs are A and B , Sel is the select signal and Z is the output. Depending on the select signal, the output is connected to either of the inputs. If $Sel = 0$, then the output will be switched to input a , whereas if $Sel = 1$, then the output will be switched to input b . The truth table is shown in Table 3.

The equation for the multiplexer is defined at Expression (2)

$$Y = (\overline{Sel} \bullet A) + (Sel \bullet B) \quad (2)$$

Table 3
2-to-1 Multiplexer Truth Table

A	B	Select	Output
0	-	0	0
1	-	0	1
-	0	1	0
-	1	1	1

The circuit for the multiplexer is shown in Figure 4.

The simulation results for the multiplexer are shown in Figure 5.

Figure 4
Synthesized RTL Diagram of 2-to-1 Multiplexer Block

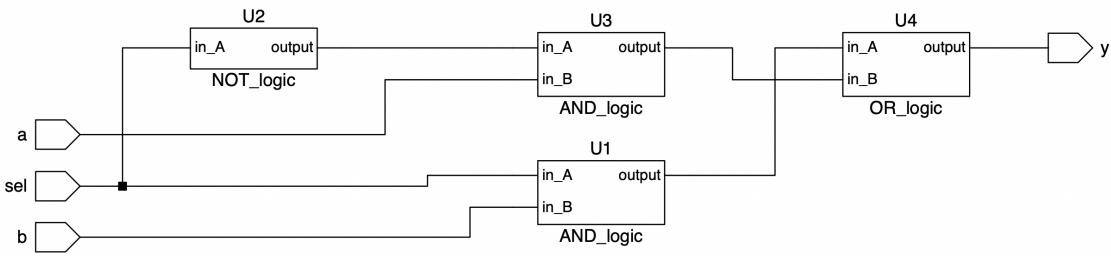
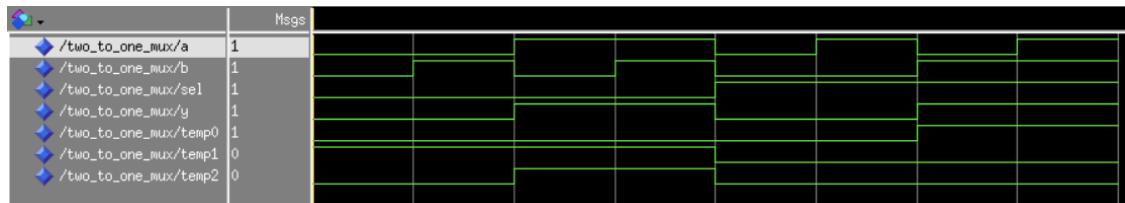


Figure 5
Simulation Wave Diagram of 2-to-1 Multiplexer Block

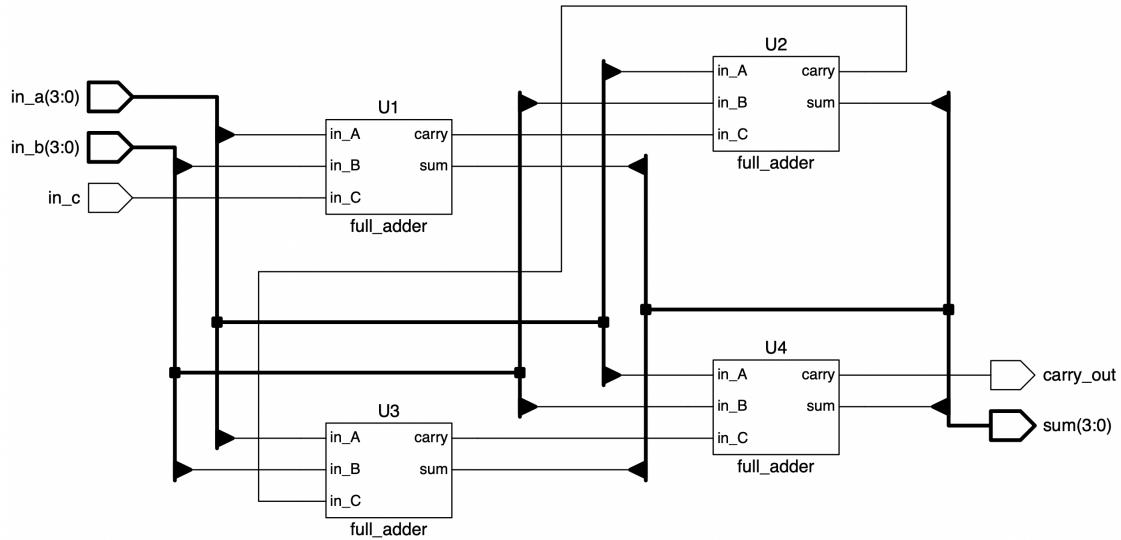


4-bit Ripple Carry Adder

In order to build n-bit carry select adders which are suitable for this project, a 4-bit ripple carry adder block is the basic component. A ripple carry adder is cascaded in parallel by multiple full adder circuits, in which the carry out of each full adder is the carry in of the succeeding next most significant full adder. Four full adders are tied together to build the 4-bit ripple carry adder block for this project. Where A and B are 4-bit inputs, sum is the addition output of A and B , $Carry_{out}$ is the output carry which depends on C_{in} , C_1 , C_2 , C_3 . The circuit for the 4-bit ripple carry adder block is shown in Figure 6.

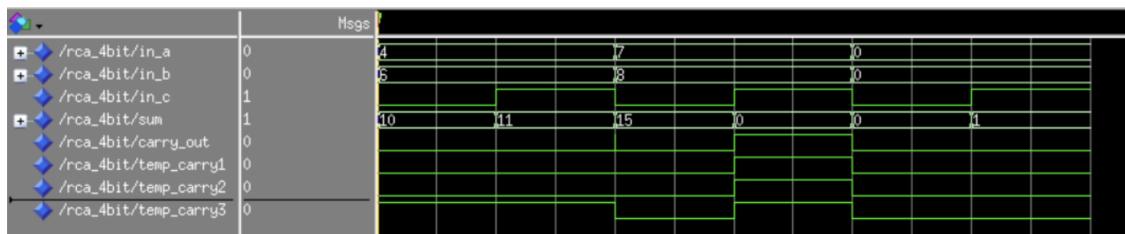
The selected simulation cases are shown below and the simulation results for the 4-bit ripple carry adder are shown in Figure 7.

Figure 6
Synthesized RTL Diagram of 4-bit Ripple Carry Adder Block



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

Figure 7
Simulation Wave Diagram of 4-bit Ripple Carry Adder Block



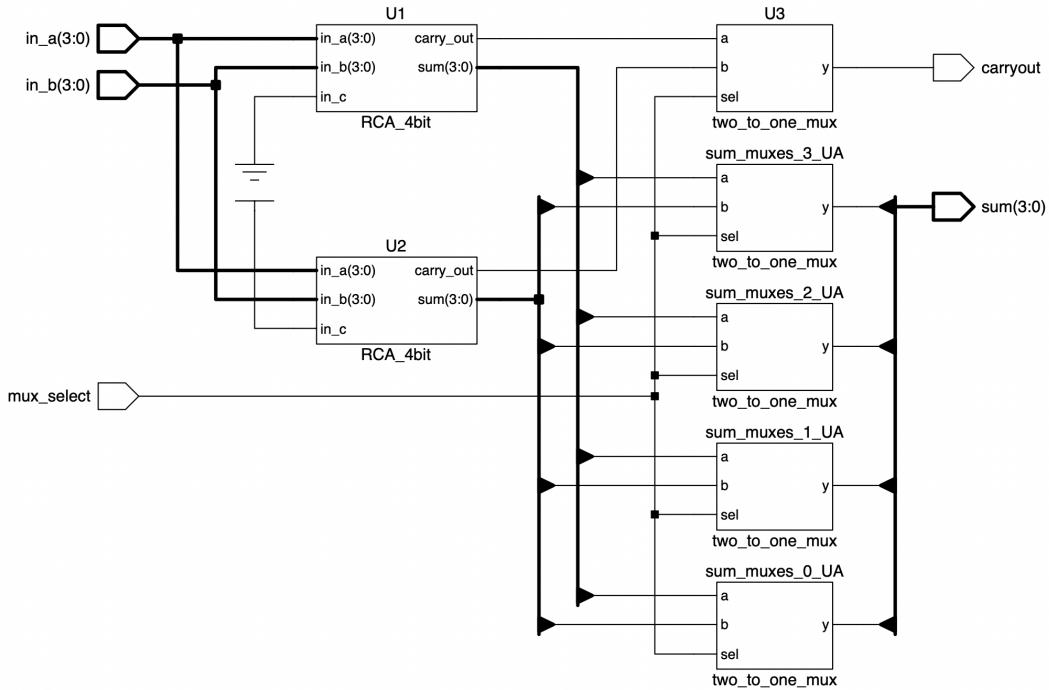
4-bit Carry Select Adder

A basic building block size of carry select adder is four. A 4-bit carry select adder consists of two parallel 4-bit ripple carry adders and 2-to-1 multiplexers to perform the calculation twice. One of the 4-bit RCA block assumes that the input carry is 0 (RCA_0), the other assumes that the input carry is 1 (RCA_1). After the two results are calculated, the correct sum, as well as the correct carry out, is then selected with the multiplexer once the correct carry in is known. The delay equation for the 4-bit carry select adder is defined below:

$$T_{CSA} = T_{mux} + 4 \times T_{full_adder} \quad (3)$$

The circuit for the 4-bit carry select adder is shown in Figure 8.

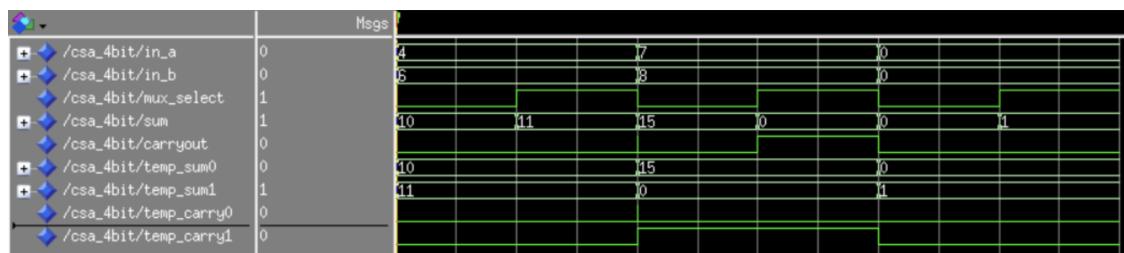
Figure 8
Synthesized RTL Diagram of 4-bit Carry Select Adder Block



If the input carry C_{in} from the lower level is 0, the output carry of the RCA_0 is selected as the output carry of this 4-bit CSA block. If the input carry C_{in} from the lower level is 1, the output carry of the RCA_1 is selected as the output carry. At the same time C_{in} is used as the selection signal of 2-to-1 multiplexer to control whether the output of S3 to S0 comes from the RCA_0 or the RCA_1.

The simulation results for the 4-bit carry select adder are shown in Figure 9.

Figure 9
Simulation Wave Diagram of 4-bit Carry Select Adder Block



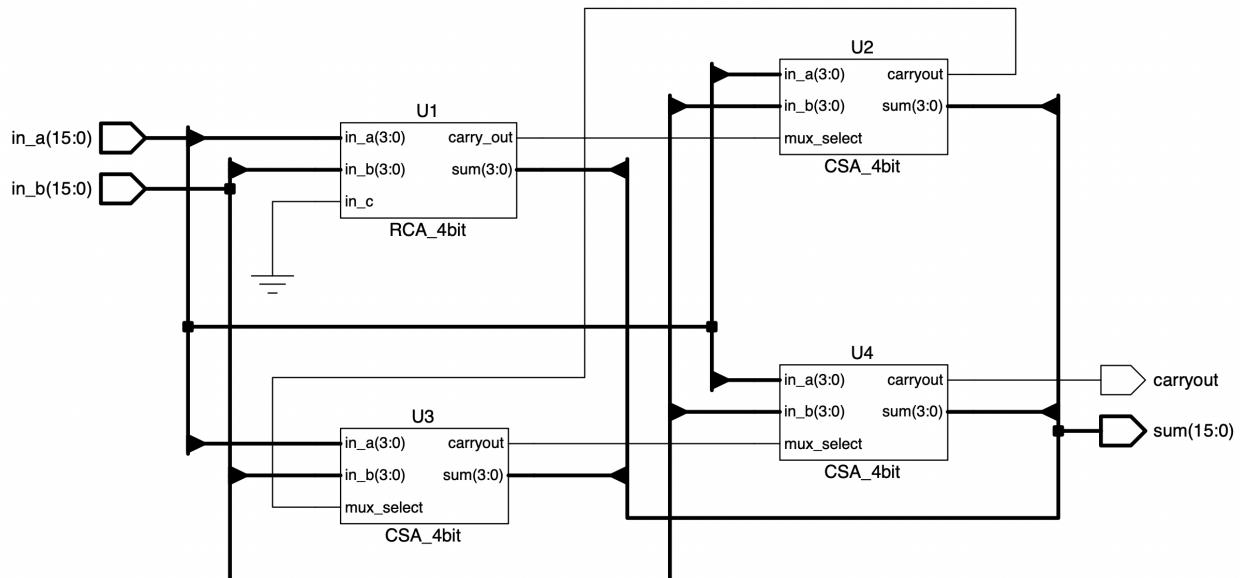
16-bit Carry Select Adder

In this project, 16-bit carry select adder is used. A 16-bit carry select adder can be created using three 4-bit CSA blocks and one 4-bit RCA blocks. The first block is a 4-bit RCA, the inputs are two binary numbers from multiplier, so that there is no input carry and the C_{in} can be set to 0. Then the delay of this adder will be the delay of the four full adders, plus the delay of the three MUXs. The delay equation for the 16-bit carry select adder is defined below:

$$T_{CSA} = 3 \times T_{mux} + 4 \times T_{full_adder} \quad (4)$$

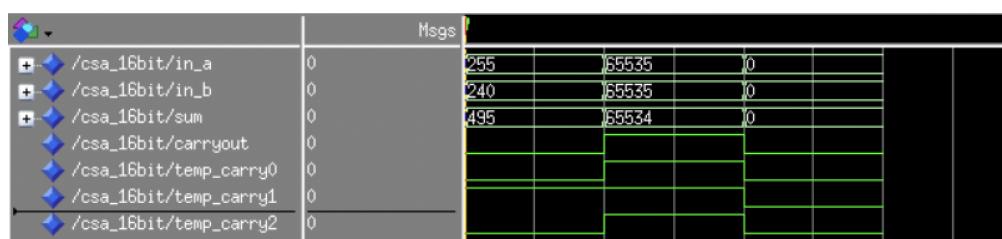
The circuit for the 16-bit carry select adder is shown in Figure 10.

Figure 10
Synthesized RTL Diagram of 16-bit Carry Select Adder Block



The simulation results for the 16-bit carry select adder are shown in Figure 11.

Figure 11
Simulation Wave Diagram of 16-bit Carry Select Adder Block



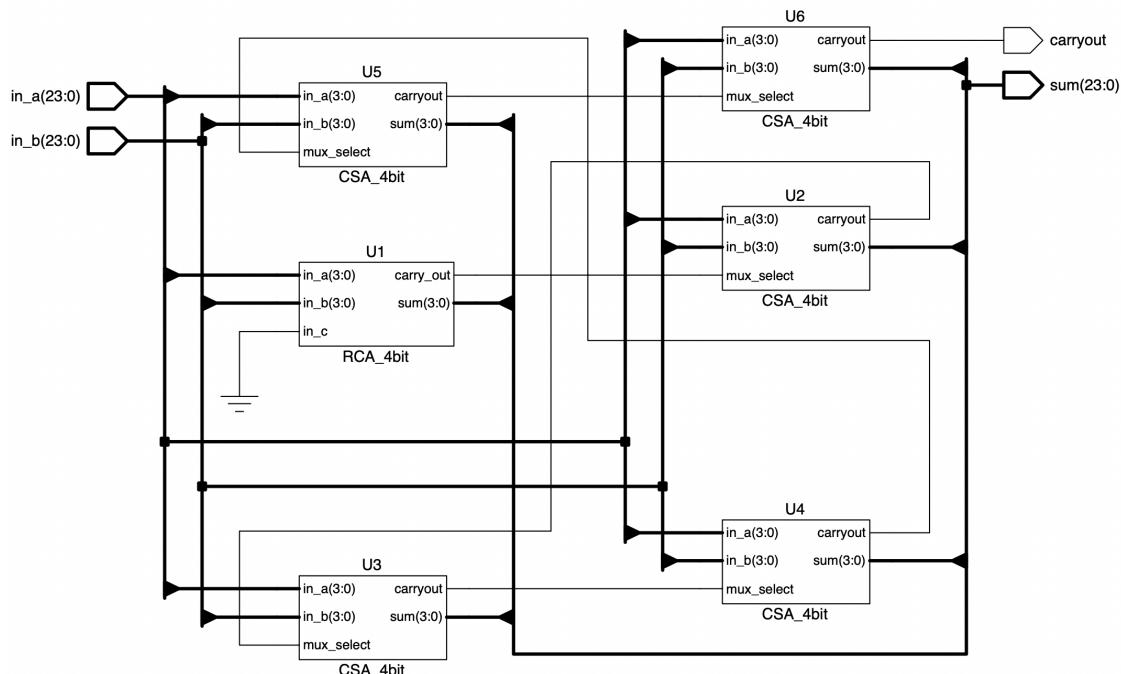
24-bit Carry Select Adder

Same principle as a 16-bit CSA, a 24-bit CSA is built up with five 4-bit CSA blocks and one 4-bit RCA block. Also, the first block is 4-bit RCA, the inputs are two binary numbers from multiplier, so that there is no input carry and the C_{in} can be set to 0. The delay equation for the 16-bit carry select adder is defined below:

$$T_{CSA} = 5 \times T_{mux} + 4 \times T_{full_adder} \quad (5)$$

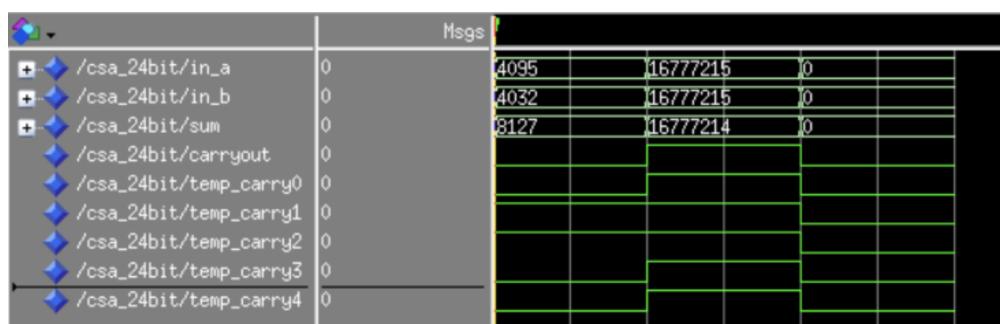
The circuit for the 24-bit carry select adder is shown in Figure 10.

Figure 12
Synthesized RTL Diagram of 24-bit Carry Select Adder Block



The simulation results for the 24-bit carry select adder are shown in Figure 13.

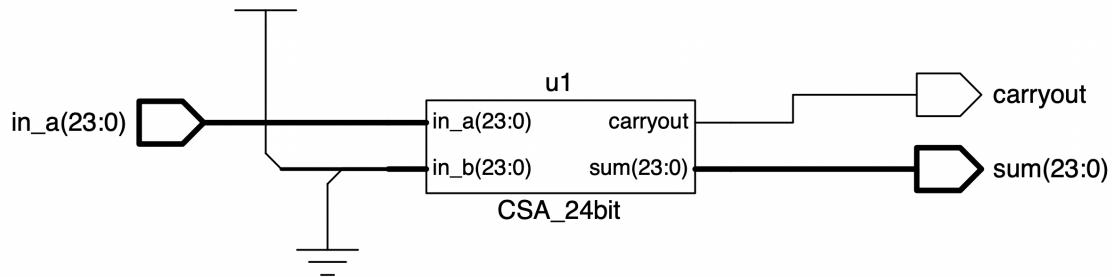
Figure 13
Simulation Wave Diagram of 24-bit Carry Select Adder Block



24-bit CSA Incrementor

The incrementor used in this project were designed on the basis of the CSA. The principle is to define the input B of the CSA as a constant with the lowest bit of 1 and the rest of 0. Due to different bit requirements, a N-bit incrementor can be implemented by N-bit CSA. A 24-bit incrementor is used in this project. The circuit for the 24-bit incrementor is shown in Figure 14.

Figure 14
Synthesized RTL Diagram of 24-bit Incrementor Block



The simulation results for the 24-bit incrementor are shown in Figure 15.

Figure 15
Simulation Wave Diagram of 24-bit Incrementor Block



Multiplication in Three Operands

The very first component of the ALU should be the circuit that calculates the result of $A^2 * B$. Hence the multiplication of two 8-bit operands circuits should be designed first, then the circuit for multiplying the product of the square A with the B should be designed later.

In general, the multiplication of two 8 bits operands in the shift-and-add algorithm or the radix-2 booth algorithm requires 8 steps of calculating eight partial products and then adding them together. By using the modified booth algorithm which is also known as radix-4 booth algorithm, the number of the partial products can be reduced to $2/n$ where n is the bit length of the operand.

Not only because it's faster, but also because it saves more area compared with the previous two algorithms. Hence the project chooses to implement the radix-4 booth algorithm as the multiplication component of the ALU.

There is an extra partial product the circuit should consider since the booth algorithm is designed for the signed number. In this case, the implemented algorithm requires $2/n + 1$ partial product for the unsigned number to reach the final answer. This will be discussed in the following sections.

Radix-4 Booth Algorithm Logic in Details

Booth algorithm calculates the partial product by examining the “Code Table” on the second operand, the multiplier. The table requires certain blocks of bits from the right side to the left side of the multiplier, and for each block, the table provides the partial product respectively. Radix-2 requires 2 bits while radix-4 requires 3 bits. This is how the radix-4 algorithm reduces the partial products to a half. In this manner, the overlaps will occur in the partial products, hence the algorithm will do subtraction according to the “Code Table”.

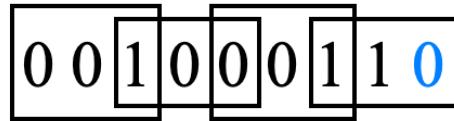
Example in Signed Number Multiplication

When the algorithm is used for calculating two 8-bit signed number $0b01010110$ which is 86 in decimal, and $0b00100011$ which is 35 in decimal, the multiplier will be decoded as Figure 16 is shown. Then all partial products can be derived from the code blocks and the “Code Table” which is presented in Table 4.

Table 4
Code Table of Radix-4 Booth Algorithm

Code Blocks	Partial Product
000/111	0
001/010	$1 * \text{multiplicand}$
011	$2 * \text{multiplicand}$
100	$-2 * \text{multiplicand}$
101/110	$-1 * \text{multiplicand}$

Figure 16
The Decoded Code Blocks of the Signed Multiplier



Note. The blue bit is an extra bit which is added on the right of the LSB of the multiplier for completing the first code block.

The algorithm takes four blocks of code from the right to the left and retrieves the corresponding product by shifting two bits more than the previous product. Then perform the addition. The process of the algorithm is described in Figure 17.

Figure 17
Process of the Algorithm for Signed Number

110	1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0	partial product 1
001	0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0	partial product 2
100	1 1 1 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0	partial product 3
001	0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0	partial product 4
	0 0 0 0 1 0 1 1 1 1 0 0 0 0 0 1 0	

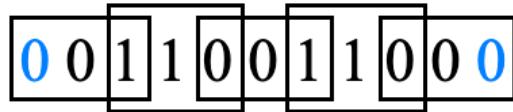
Note. Bits in green color represent the product is extended to 16 bits. Bits in red color represent the shifted 2 bits leftward.

After the performance, the result $0b0000101111000010$ is 3010 in decimal which is the product of 86 and 35. As can be observed at the table, subtraction can be done by adding the negation of the number which is represented by 2's complement.

Example in Unsigned Number Multiplication

Applying the algorithm to unsigned operands is a little bit different. Because the MSB of the operand is treated as a valid number rather than the sign, the operands should extend to 9-bit by adding a “0” to the MSB. Hence an extra partial product will be added to the product. For instance, multiplicand $0b010010101$ which is 149 in decimal and multiplier $0b011001100$ which is 204 in decimal can be operated in the process as Figure 18 and Figure 19 are shown.

Figure 18
The Decoded Code Blocks of the Unsigned Multiplier



Note. Two “0” are added to the MSB to complete the last code block. The second zero represent the sign bit of the operand which in this case will always be positive number.

Figure 19
Process of the Slgorithm for Unisgned Number

000	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	partial product 1
110	1 1 1 1 1 1	1 0 1 1 0 1 0 1 1 0 1 1 0 0 0	partial product 2
001	0 0 0 0 1 0 0	1 0 1 0 1 0 1 0 1 0 1 0 0 0 0	partial product 3
110	1 1 0 1 1 0 1	0 1 1 0 1 0 1 1 0 0 0 0 0 0	partial product 4
001	1 0 0 1 0 1 0	1 0 0 1 0 1 0 1 0 0 0 0 0 0	partial product 5
		0 1 1 1 0 1 1 0 1 0 1 1 1 1 0 0	

Note. Bits in green color represent the product is extended to 16 bits. Bits in red color represent the shifted 2 bits leftward.

8-bit Radix-4 Booth Multiplier Circuit Implementation

Overall Circuit Design and RTL Description

As the previous discussion, the booth multiplier component should contain the following blocks: 1. A 9-bit complement generator for the negation of the multiplicand; 2. Five booth stage units for 5 partial products; 3. Four 16-bit adders to sum up the partial products. Figure 20 presents the synthesized RTL diagram of the multiplier circuit.

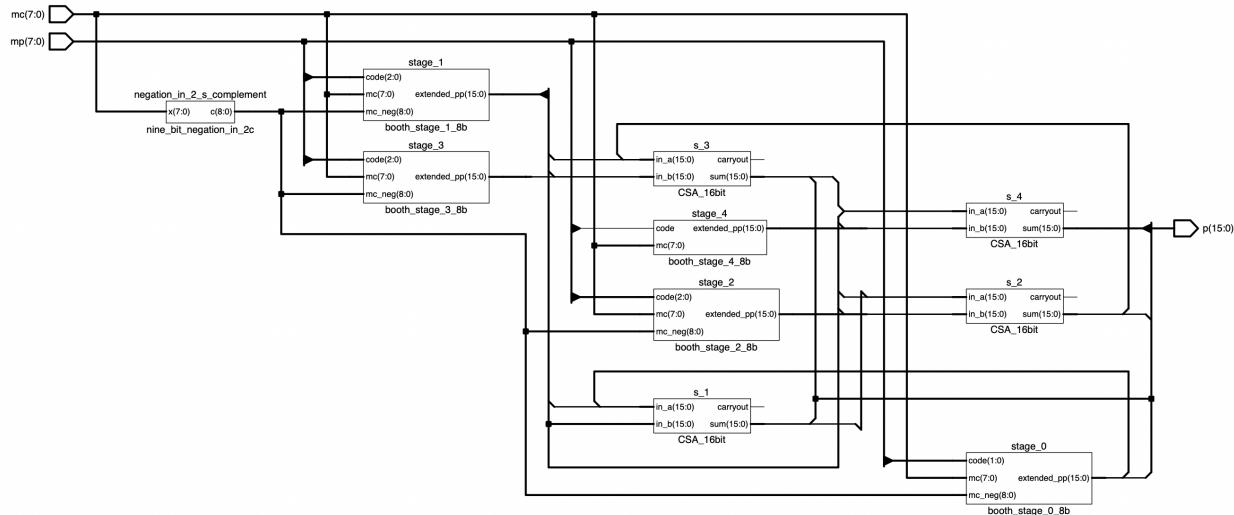
The circuit will first calculate a 9-bit negation of operand multiplicand in 2’s complement. Then pass through all five booth stage blocks to get five 16-bit partial products. And finally sums those partial products up.

Blocks Design

This section will discuss the design of the complement generator and five of the booth stages, the 16-bit CSA will be discussed in Section “Carry Select Adder”.

The 9-bit Complement Generator for the Negation of the Multiplier. Since the circuit uses negation addition to represent the subtraction, a complement generator should be introduced. The RTL diagram of the generator is shown in Figure 21. The logic of the generator that uses a 2-to-1 mux is straightforward as Expression (6) described. The simulation result is shown in Figure 22.

Figure 20
Synthesized RTL Diagram of 8-bit Radix-4 Booth Multiplier



$$c = \begin{cases} concat(0, x), & \text{if } x = 00000000 \\ concat(1, (not\ x)) + 1, & \text{otherwise} \end{cases} \quad (6)$$

Figure 21
Synthesized RTL Diagram of Complement Generator

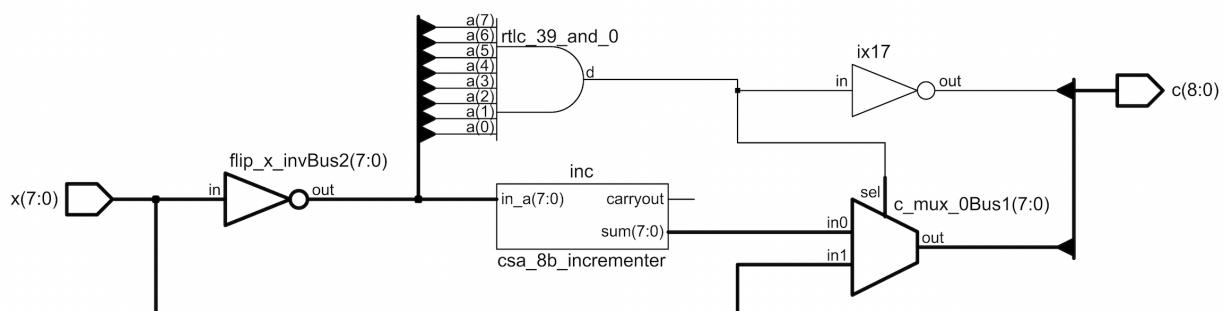
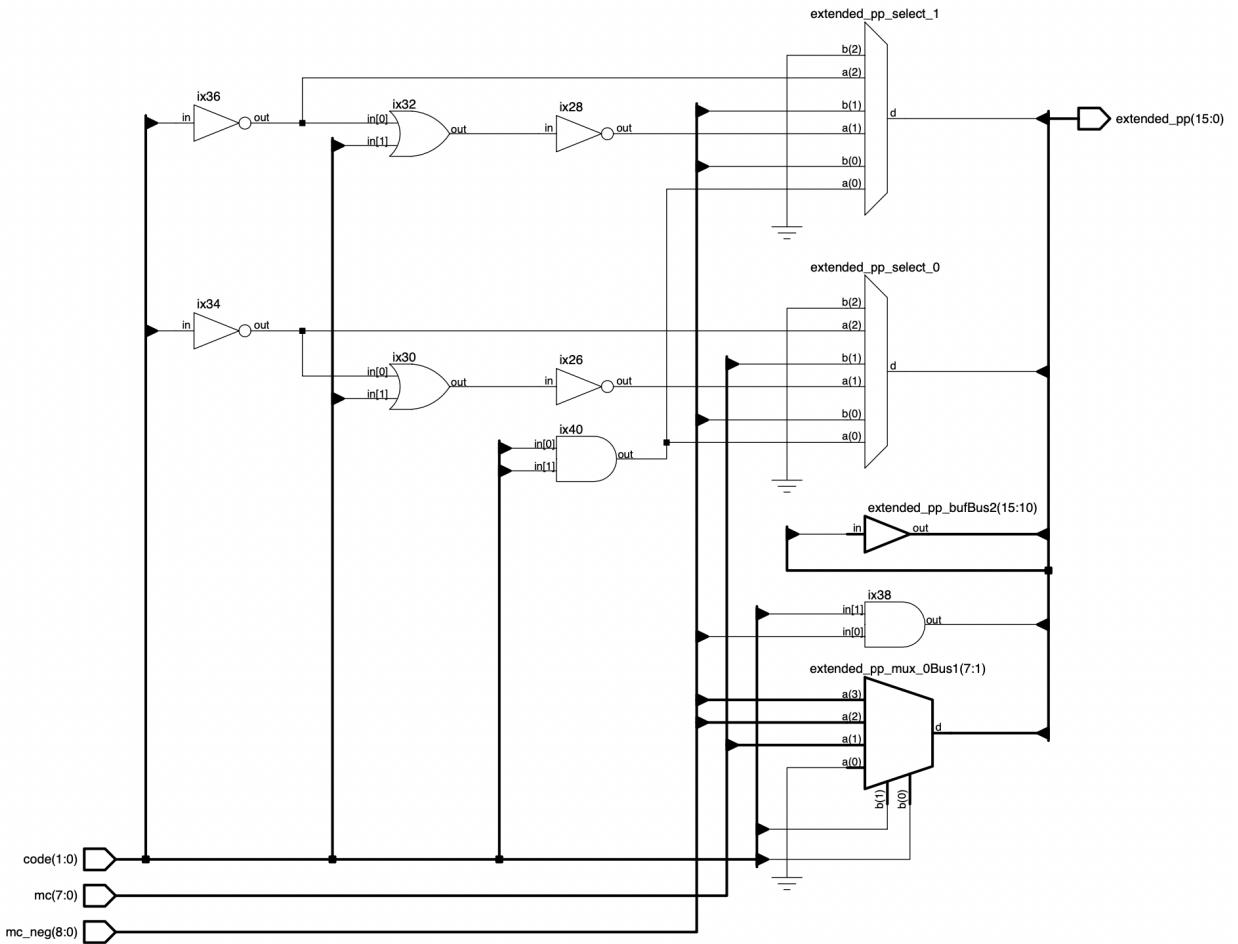


Figure 22
Simulation Wave Diagram of Complement Generator

	Msgs		
+ /nine_bit_negation_in_2c/x	Data-	00000000	10110110
+ /nine_bit_negation_in_2c/c	Data-	000000000	101001010
+ /nine_bit_negation_in_2c/flip_x	Data-	11111111	01001001
+ /nine_bit_negation_in_2c/tmp	Data-	00000000	01001010
			11111111
			100000001
			000000000
			000000001

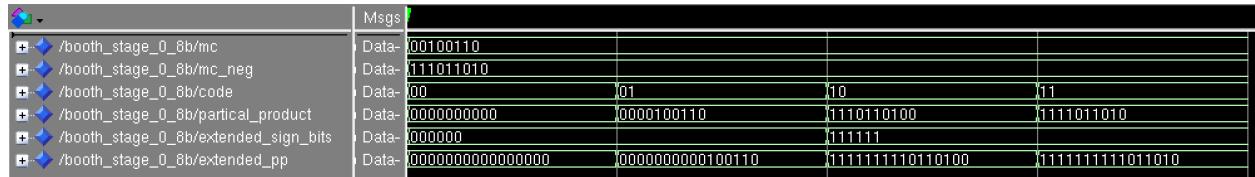
Booth Stage 0. Figure 23 and Expression (7) present the hardware implementation and the logic expression of the block. As the figure suggested, a booth stage block takes the 8-bit operand multiplicand and its 9-bit negation and two of the rightmost bits of the operand multiplier as input, and composes the 16-bit extended partial product as output by a 4-to-1 mux. The width of the *extended_sign_bits* will be 6. The simulation result is shown in Figure 24.

Figure 23
Synthesized RTL Diagram of Booth Stage 0 Block



$$\begin{aligned}
 \text{partial_product} = & \begin{cases} 0000000000, & \text{if } \text{code} = 00 \\ \text{concat}(00, \text{mc}), & \text{if } \text{code} = 01 \\ \text{concat}(\text{mc_neg}, 0), & \text{if } \text{code} = 10 \\ \text{concat}(\text{mc_neg}(8), \text{mc_neg}), & \text{else } \text{code} = \text{others} \end{cases} \\
 \text{extended_sign_bits} &= (\text{others} \Rightarrow \text{partial_product}(9)) \\
 \text{extended_pp} &= \text{concat}(\text{extended_sign_bits}, \text{partial_product})
 \end{aligned} \tag{7}$$

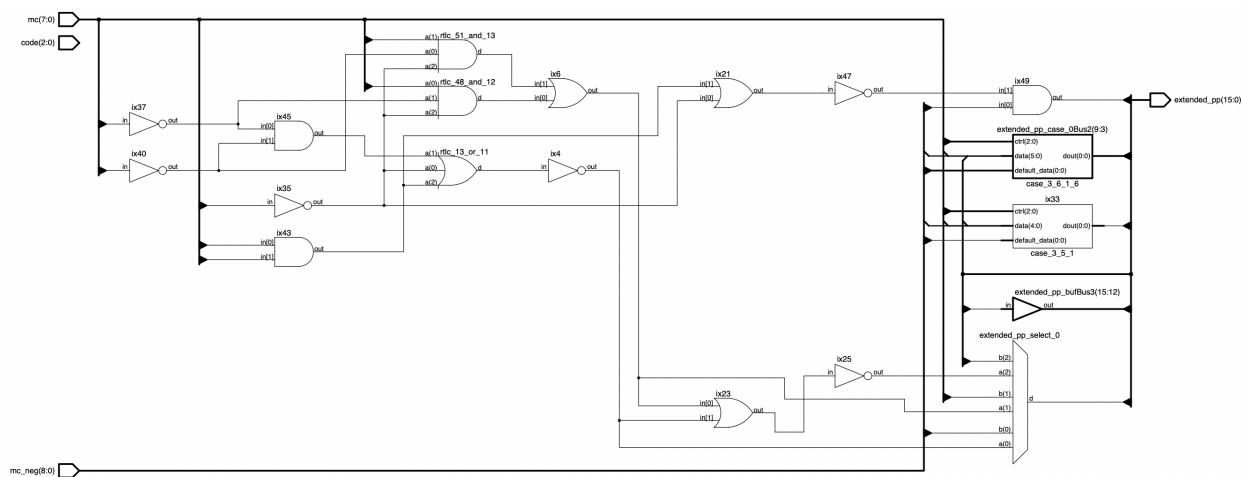
Figure 24
Simulation Wave Diagram of Booth Stage 0 Block



Note. Four code values were provided to simulate the *extended_pp*.

Booth Stage 1. Figure 25 and Expression (8) present the hardware implementation and the logic expression of the block. Different from booth stage 0 block, this stage takes 3 bits from the operand multiplier. With considering the right shift during the algorithm, the width of the *extended_sign_bits* will be 4, and “00” will be added to the end. The simulation result is shown in Figure 26.

Figure 25
Synthesized RTL Diagram of Booth Stage 1 Block



$$\begin{aligned}
partial_product = & \begin{cases} 0000000000, & \text{if } code = 000|111 \\ concat(00, mc), & \text{if } code = 001|010 \\ concat(0, mc, 0), & \text{if } code = 011 \\ concat(mc_neg, 0), & \text{if } code = 100 \\ concat(mc_neg(8), mc_neg), & \text{else } code = others \end{cases} \\
extended_sign_bits = & (others \Rightarrow partial_product(9)) \\
extended_pp = & concat(extended_sign_bits, partial_product, 00)
\end{aligned} \tag{8}$$

Figure 26
Simulation Wave Diagram of Booth Stage 1 Block

(a) With Code Values: 000/001/010/011

	Msgs			
+ /booth_stage_1_8b/mc	Data- 00100110			
+ /booth_stage_1_8b/mc_neg	Data- 111011010			
+ /booth_stage_1_8b/code	Data- 000	001	010	011
+ /booth_stage_1_8b/partial_product	Data- 0000000000	0000100110		0001001100
+ /booth_stage_1_8b/extended_sign_bits	Data- 0000			
+ /booth_stage_1_8b/extended_pp	Data- 0000000000000000	0000000010011000		0000000100110000

(b) With Code Values: 100/101/110/111

	Msgs			
+ /booth_stage_1_8b/mc	Data- 00100110			
+ /booth_stage_1_8b/mc_neg	Data- 111011010			
+ /booth_stage_1_8b/code	Data- 100	101	110	111
+ /booth_stage_1_8b/partial_product	Data- 1110110100	1111011010		00000000000
+ /booth_stage_1_8b/extended_sign_bits	Data- 1111			0000
+ /booth_stage_1_8b/extended_pp	Data- 1111111011010000	1111111101101000		0000000000000000

Note. Eight code values were provided to simulate the *extended_pp*.

Booth Stage 2, 3, and 4. Booth stage 2 and 3 blocks share the same idea of booth stage 1 except they shift more bit to the right. As for the booth stage 4 block, it only takes the MSB from the operand multiplier. Expression (9) shows its logic.

$$extended_pp = \begin{cases} 0000000000000000, & \text{if } code = 0 \\ concat(mc, 00000000), & \text{else } code = others \end{cases} \tag{9}$$

8-bit Triple Operands Multiplier Circuit Implementation

Once the 16-bit product of A^2 which is marked as *product_aa* is calculated, it will then multiply with the 8-bit input *B*. To perform multiplication with a 16-bit operand and an 8-bit operand, the circuit divides the 16-bit operand into two 8-bit operands. This is to reuse the 8-bit multiplier block that is designed before.

The product of multiplying the higher 8-bit of *product_aa* and *B* will be marked as *product_haa_b*, and it will be extended to 24 bits by shifting 8 bits rightwards. The product of multiplying the lower 8-bit of *product_aa* and *B* will be marked as *product_laa_b*, and it will be extended to 24 bits by adding 8 zeros to its left. Then adds those two extended 24-bit numbers together will be the result of $A^2 * B$.

Figure 27 presents the RTL description of the circuit and Figure 28 presents the simulation result.

Expression (10) shows the arithmetic process of the $A^2 * B$.

$$\begin{aligned}
 aa &= a * a \\
 laa_b &= aa[7,0] * b \\
 haa_b &= aa[15,8] * b \\
 aab_m &= concat(haa_b, 00000000) \\
 aab_l &= concat(00000000, laa_b) \\
 p &= aab_m + aab_l
 \end{aligned} \tag{10}$$

Figure 27
Synthesized RTL Diagram of Triple 8-bit Operands Multiplier Circuit

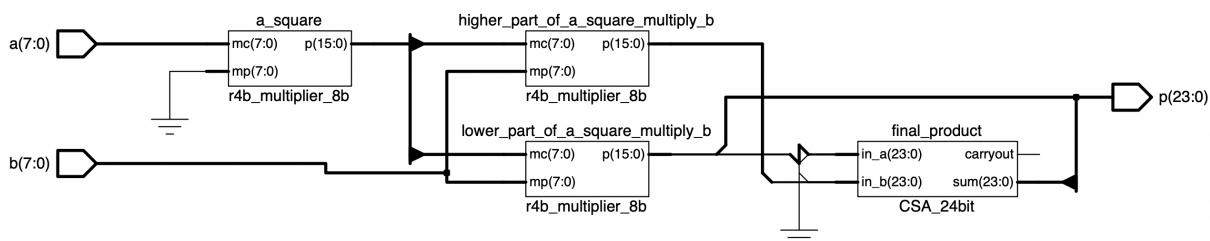
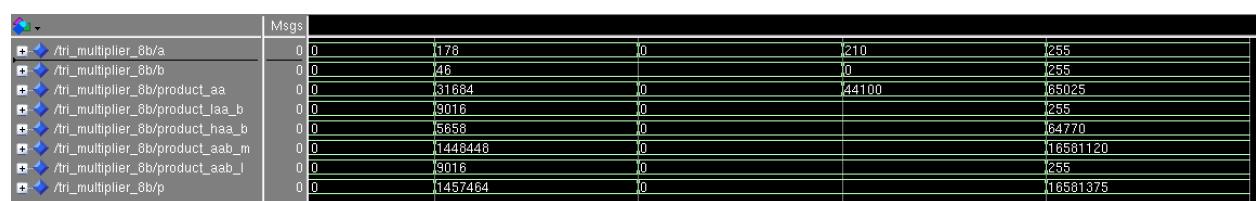


Figure 28
Simulation Wave Diagram of Triple 8-bit Operands Multiplier Circuit



16-bit Triple Operands Multiplier Circuit Implementation

Like the 8-bit version, the 16-bit implementation of the booth multiplier will need 9 blocks of booth stage to calculate 9 partial products and a 17-bit negation complement generator. Because of the characteristic of the algorithm, the 16-bit version can not reuse or extend from the 8-bit version circuit designed before. Hence the circuit will need to build every block from scratch. **This is one of the drawbacks of the booth algorithm: lack of expandability.**

Figure 29 presents the RTL description of the triple 16-bit operands multiplier circuit and Figure 30 presents the simulation result. Figure 47 presents the RTL description of the 16-bit booth multiplier circuit.

Figure 29

Synthesized RTL Diagram of Triple 16-bit Operands Multiplier Circuit

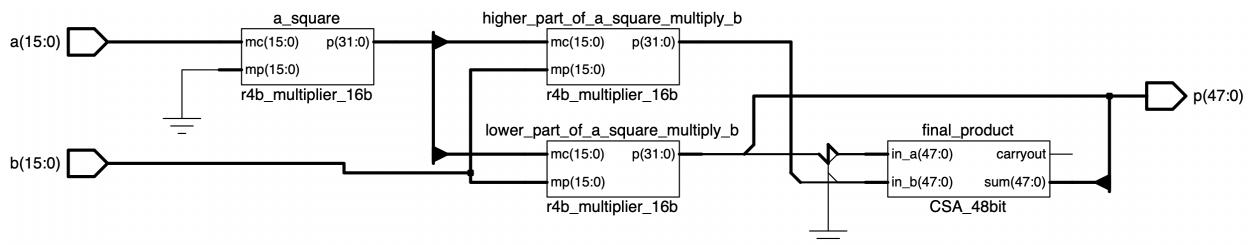


Figure 30

Simulation Wave Diagram of Triple 16-bit Operands Multiplier Circuit

	Msgs
+ ⚡ /tri_multiplier_16b/a	Data- 0 38096 0 11507 65535
+ ⚡ /tri_multiplier_16b/b	Data- 0 60624 0 0 65535
+ ⚡ /tri_multiplier_16b/product_aa	Data- 0 1451457604 0 132411049 4294967295
+ ⚡ /tri_multiplier_16b/product_laa_b	Data- 0 1928570688 0 0 65535
+ ⚡ /tri_multiplier_16b/product_haa_b	Data- 0 1342639728 0 4294770690 0
+ ⚡ /tri_multiplier_16b/product_aab_m	Data- 0 67991237214208 0 281462091939840 0
+ ⚡ /tri_multiplier_16b/product_aab_l	Data- 0 1928570688 0 65535 0
+ ⚡ /tri_multiplier_16b/p	Data- 0 67993165784896 0 281462092005375 0

Overflow Handling

In normal process, output Z is a 24-bit unsigned number from the equation since A and B are 8-bit numbers.

Here are two methods to convert 24-bit unsigned number into 16-bit signed number.

Method 1: Negation Output

Overflow Detection

To detect whether a 24-bit unsigned number is overflow for a 16-bit signed slot, just simply using a 9-bit or gate for the higher 9-bit of the 24-bit unsigned number.

The RTL description is shown in Figure 31 and the simulation result of block is shown in Figure 32.

Figure 31
Synthesized RTL Diagram of the Zero Detector

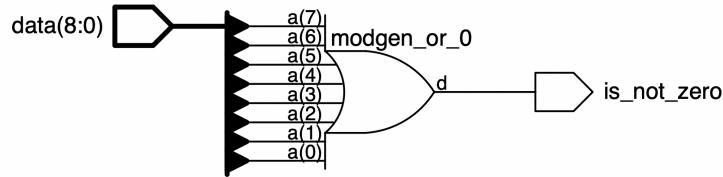
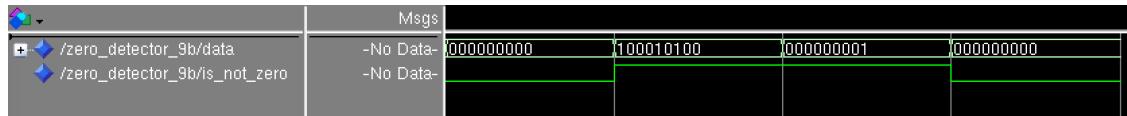


Figure 32
Simulation Wave Diagram of the Zero Detector



Handler Implementation

Once the 24-bit result is detected as an overflowing number, the circuit will just output a negative number to indicate the overflow happens. The RTL description of the implementation is shown in Figure 33 and the simulation result of block is shown in Figure 34.

Method 2: Display Z Separately with Two Clock Cycles

In this method, the circuit will output the 24-bit number into two parts. The higher part of the 24-bit result which is a 9-bit number will be stored in a register with concatenating a “1000000” to its front; The rest of the 15-bit will be stored in another register with concatenating a “0” to its front. Then the circuit will output those two parts to Z alternately by the clock cycles.

Figure 33
Synthesized RTL Diagram of the Overflow Handler of Method 1

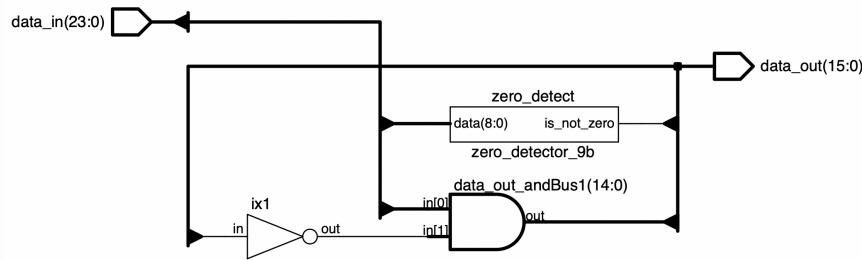
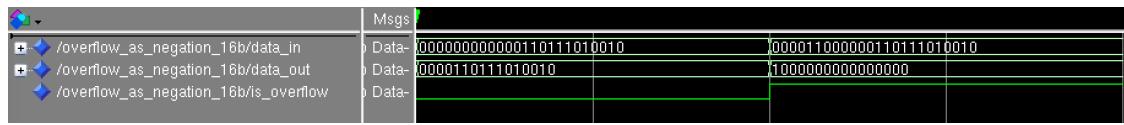


Figure 34
Simulation Wave Diagram of the Overflow Handler of Method 1



The drawback of this method is obvious: **the result will need two clock cycles to be fully obtained**. Hence the input of the ALU circuit should have a clock of the time interval between every set of A and B input to avoid output overlaps. This method acts like a 2 states FSM which RTL description is shown in Figure 35. The simulation result of block is shown in Figure 36.

Figure 35
Synthesized RTL Diagram of the Overflow Handler of Method 2

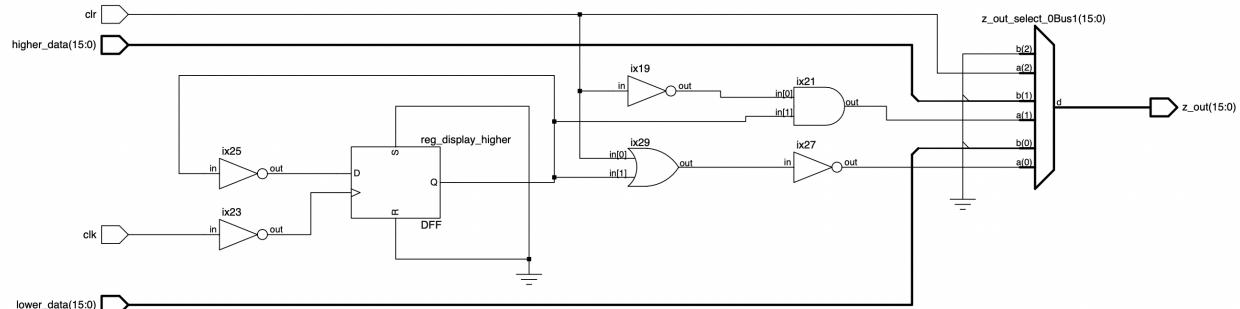
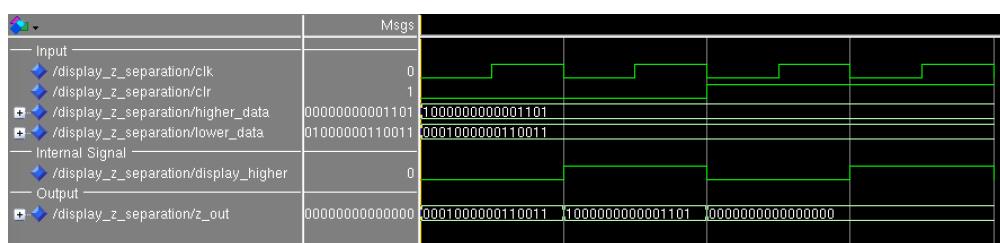


Figure 36
Simulation Wave Diagram of the Overflow Handler of Method 2



End Flag Generator

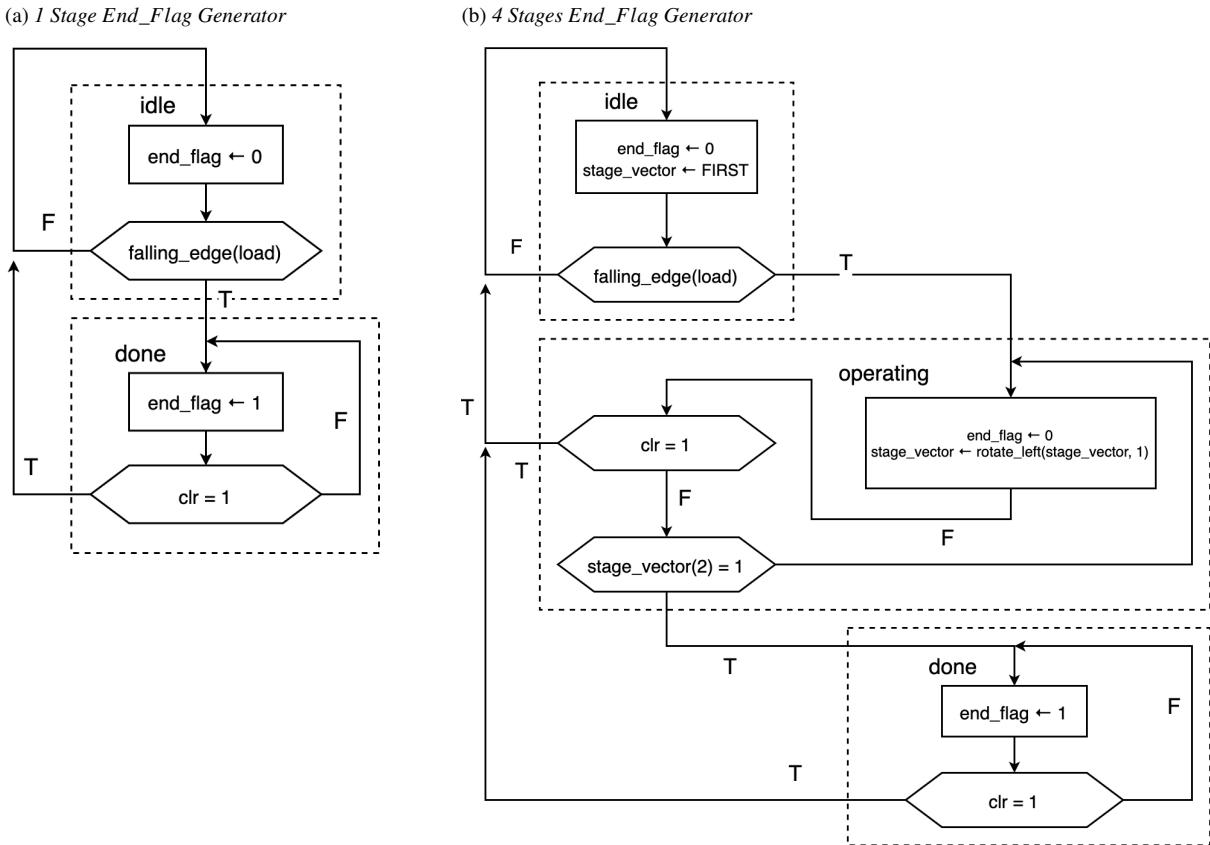
To generate a valid *END_FLAG* signal for the ALU, a Timed Mealy State Machine is introduced into the circuit. The ASMD charts of the 1 stage and 4 stages *END_FLAG* generator are presented in Figure 37.

For pipelined circuit, the *END_FLAG* will result in the output after counting 4 synchronized clock cycles when the *LOAD* signal goes from high to low. The counting of the clock cycle is implemented by rotating a vector signal to avoid introducing an extra addition circuit.

For non-pipelined circuit, the *END_FLAG* will result in the output of a clock later after the *LOAD* signal goes from high to low.

Figure 37

ASMD Chart of the FSM of Different End Flag Generator



Note. In (b), the “stege_vector” is a 3 bit vector signal and the “FIRST” is “001”.

Non-pipelined Implementation

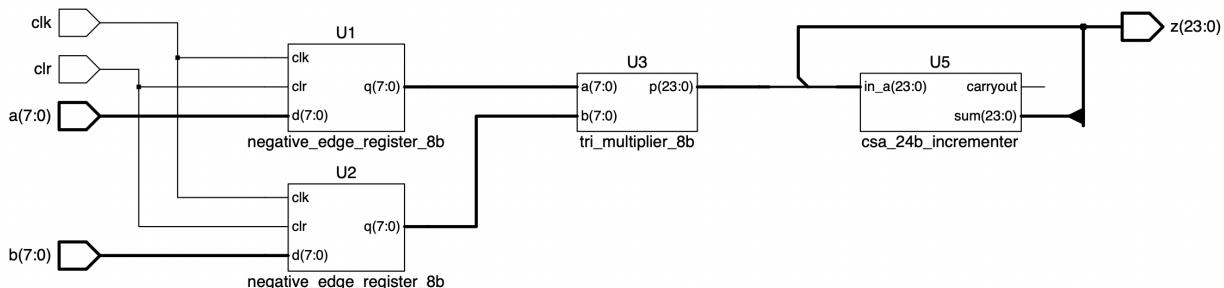
For non-pipelined implementation, the arithmetic logic block functions with a clock and an active low signal *LOAD* to load the operands *A* and *B* into the operand registers. After finishing the arithmetic operations, the result will be stored in the output register with a high *END_FLAG* signal. The *END_FLAG* signal is used to denote whether the output is valid, it is asserted when the calculation has finished. The *CLEAR* signal will clear all the operand and output registers to ‘0’ when it is active high.

Operating Circuit

Once the operands *A* and *B* are loaded into the operand registers. The output of these registers is then fed into the designed multiplier to perform the multiplication calculation required by the arithmetic unit. The output of the multiplication unit is then fed into the shifter unit. After being shifted by four bits to the right, the result is then fed into the 24-bit incrementor. The resulting result then passed into a result register pending final overflow processing. The result output of the overflow process is then fed into the *Z*-port.

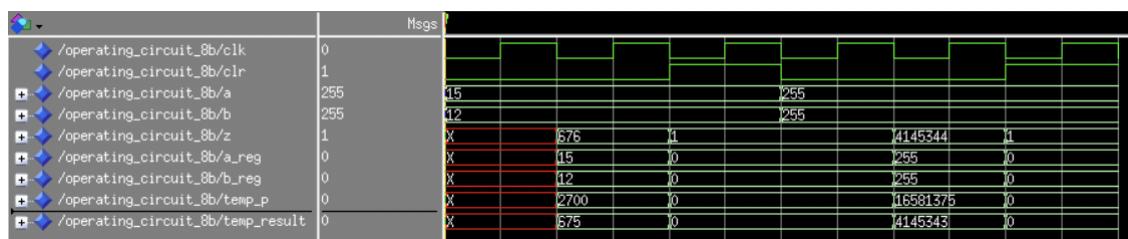
The operating circuit for the non-pipelined implementation is shown in Figure 38.

Figure 38
Synthesized RTL Diagram of Non-pipelined Operating Circuit



The test simulation results for the non-pipelined operating circuit are shown in Figure 39.

Figure 39
Simulation Wave Diagram of Non-pipelined Operating Circuit

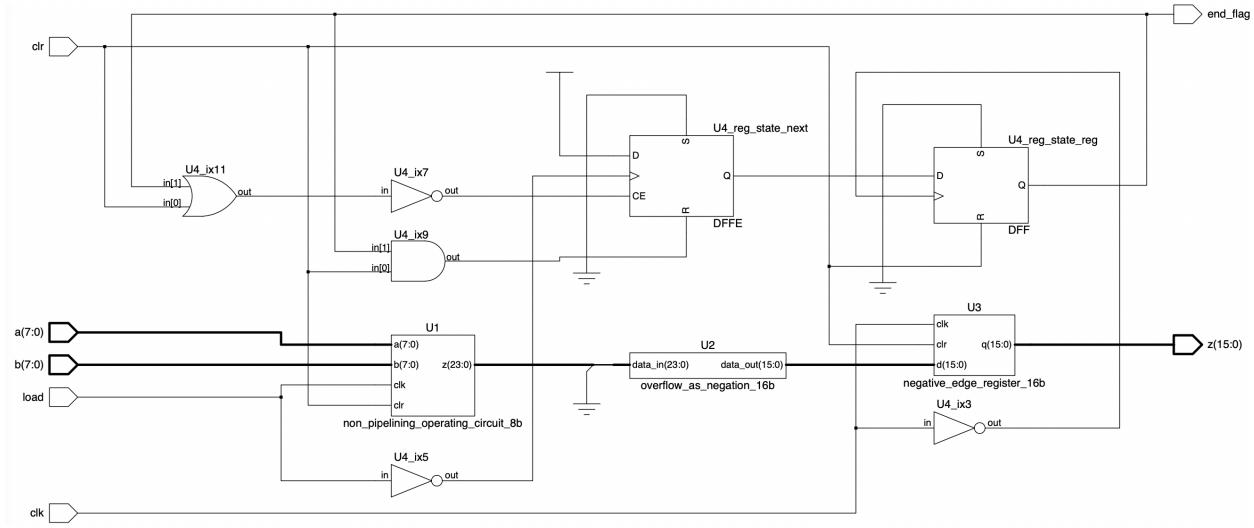


Note. Signal with suffix “_after_reg” means the value of the signal is an output of a register

Output Process

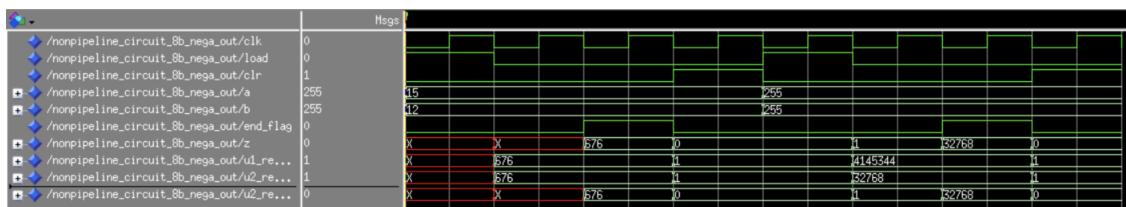
In the non-pipelined implementation, negation output is used to meet the project requirements of 16-bit Z-port outputs, which just represent the result as overflow. The implementation of the whole circuit is shown in Figure 40. The test simulation results for the negation output are shown in Figure 41.

Figure 40
Synthesized RTL Diagram of Non-pipelined Circuit



Note. A 1 stage END_FLAG generator is used.

Figure 41
Simulation Wave Diagram of Non-pipelined Circuit



Note. Signal with suffix “_after_reg” means the value of the signal is an output of a register.

Pipelined Implementation

For pipelined implementation, a set of stage registers is placed after every arithmetic operation block for memorizing the outcome of every stage. The required operation $Z = \frac{1}{4}[A^2 * B] + 1$ can be divided into three stages of calculation with one overflow handling stage. Stage 1 should obtain the product of A and A and B . Then stage 2 should perform two bits right shifting operation on the result of stage 1, this operation represents the $\frac{1}{4}$. After that, stage 3 will add number 1 to the result of stage 2. Finally, stage 4 should handle the bit width of the arithmetic result into the required width.

8-bit Operands Implementation with Negation Overflow Handler

By this design, the circuit will obtain the result in 4 synchronized clock cycles after the falling edge event of the *load* signal. Table 5 shows how the registers in this pipelined circuit will latch the values. Figure 43 presents the RTL description of the circuit and its arithmetic block.

Table 5

A Visual Representation Table of the Pipelined Process with Negation Overflow Handler

clk	A Reg	B Reg	Stage 1 Reg	Satge 2 Reg	Satge 3 Reg	Stage 4 Reg(Z)
0	12	3	-	-	-	-
1	100	100	432	-	-	-
2	-	-	1000000	108	-	-
3	-	-	-	250000	109	-
4	-	-	-	-	250001	109
5	-	-	-	-	-	-32768

Figure 42 shows the simulation wave result of the pipelined ALU circuit. As can be observed with the figure, once the *load* signal goes from 1 to 0, after 4 clock cycles, *z* obtains the result of the input. And the circuit is pipelining the data by the falling edge event of *load* signal and the *clock*. If the *clk* signal was asserted, all registers will be reset and *end_flag* turned into 0.

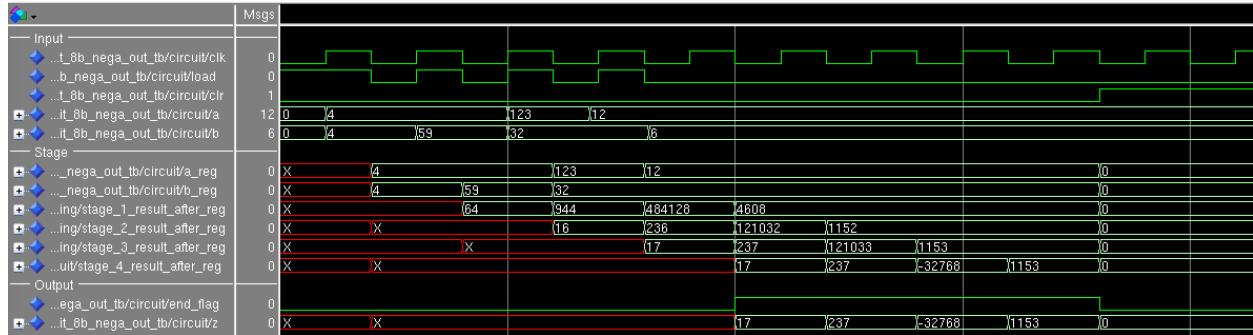
8-bit Operands Implementation with Output Separation Overflow Handler

With separation overflow handling discussed before, the full result can be obtained after three stages of calculation with two overflow handling stages. Before the full data reach the *Z* port, the 24-bit result will be separated into 2 parts: the higher part stored in one register and the lower part stored in another register. The handler takes those two data and transfers them to *Z* port alternately by the clock.

Table 6 presents how the *z* will take over the data and Figure 44 shows the simulation of the design. Figure 45 presents the RTL description of the design.

Figure 42

Simulation Wave Diagram of the 8-bit Pipelined ALU Circuit with Negation Output

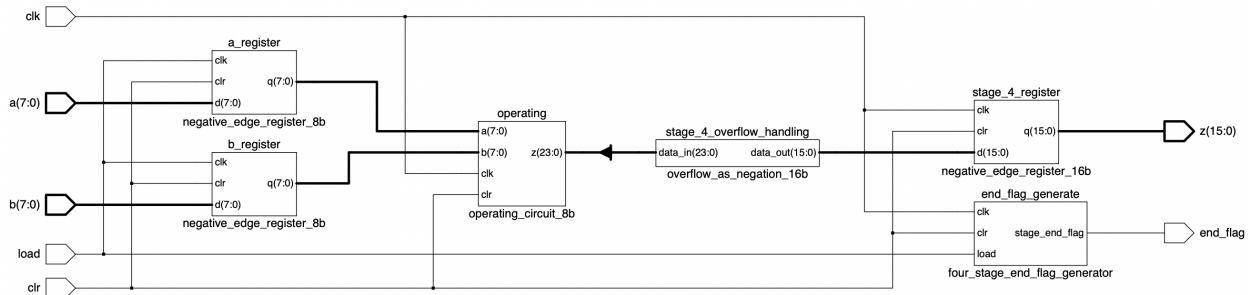


Note. Signal with suffix “_after_reg” means the value of the signal is an output of a register.

Figure 43

Synthesized RTL Diagram of the 8-bit Operands Pipelined ALU Circuit and the Arithmetic Operation Block

(a) Top Design of the Circuit



(b) Arithmetic Operation Block

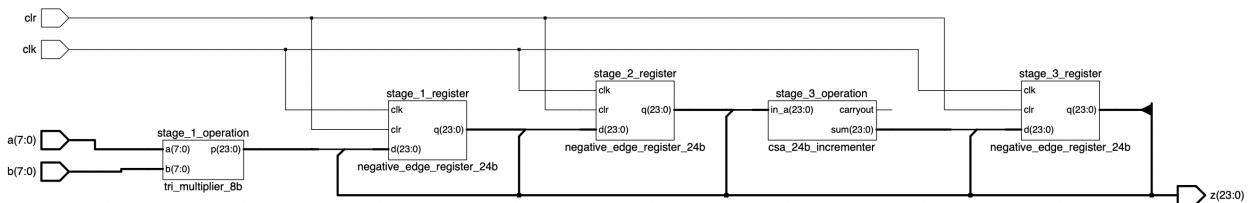


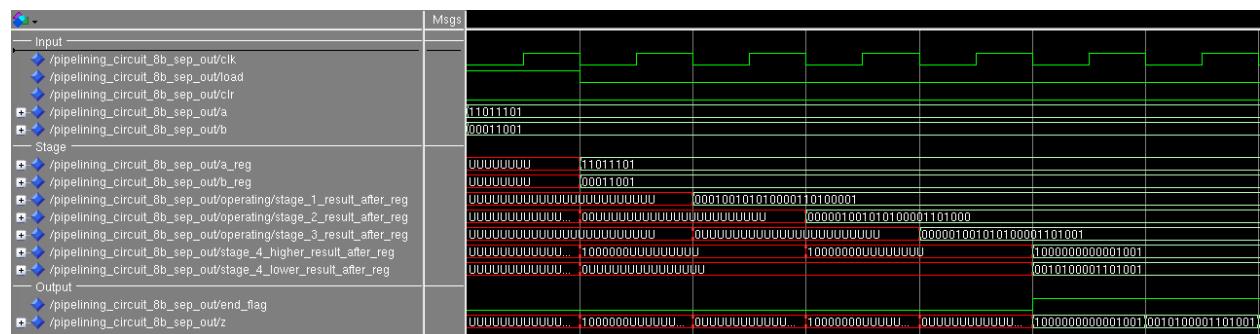
Table 6

A Visual Representation Table of the Pipelined Processing with Output Separation Overflow Handler

clk	Stage 3 Reg	Stage 4 Reg for Higher	Stage 4 Reg for Lower	Z
...				
3	000110110000110100101101	-	-	-
4	-	1000000000110110	0000110100101101	1000000000110110
6	-	1000000000110110	0000110100101101	0000110100101101

Figure 44

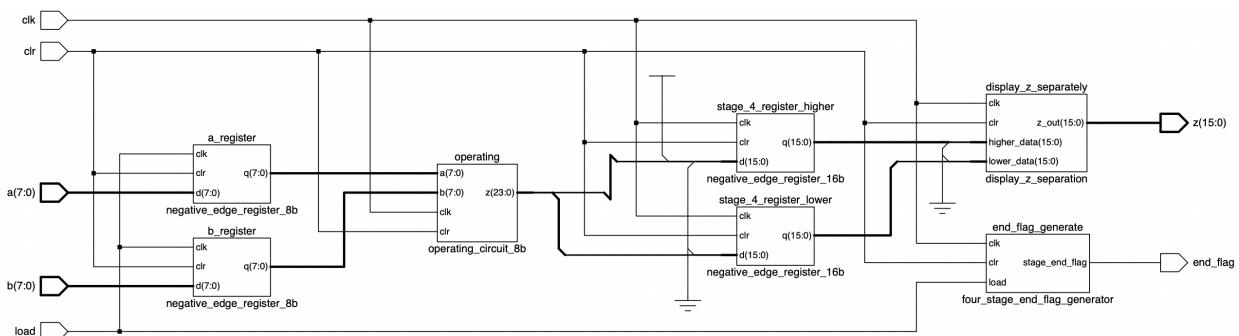
Simulation Wave Diagram of the 8-bit Pipelined ALU Circuit with Separation Output



Note. Signal with suffix “_after_reg” means the value of the signal is an output of a register. The value of Z will retrieve from the previous two registers alternately.

Figure 45

Synthesized Diagram of the 8-bit Pipelined ALU Circuit with Separation Output

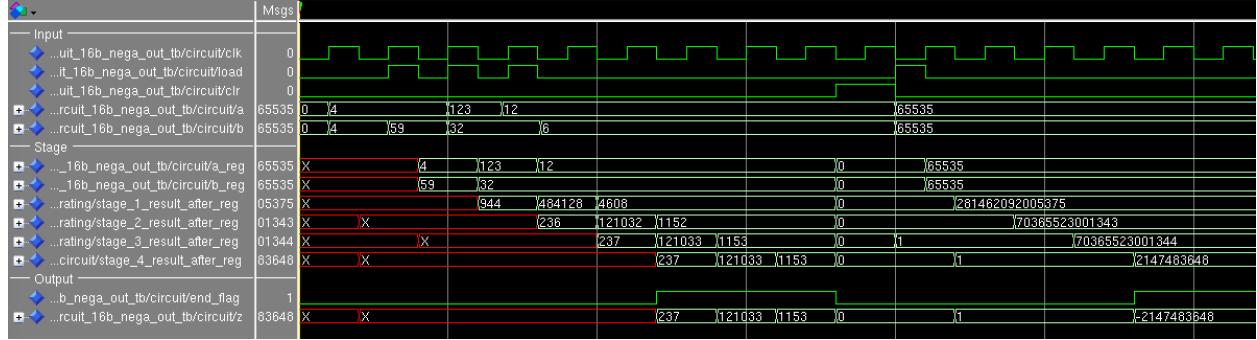


16-bit Operands Implementation and Simulation

By now there will no necessary for discussing the implementation of the extended design since all details were presented in the previous sections. Figure 46 presents the testbench simulation of the bit-width extended circuit.

Figure 46

Simulation Wave Diagram of the 16-bit Pipelined ALU Circuit with Negation Output



Note. Signal with suffix “_after_reg” means the value of the signal is an output of a register.

Synthesis and Analysis of the Arithmetic Circuit

Introduction

The arithmetic circuit is synthesized for implementation on a field-programmable grid array (FPGA). Xilinx Virtex-II Pro is used as the FPGA in this project. Mentor Graphics PrecisionRTL is used to generate and synthesize the circuits, which also details the area usage and can perform timing analysis if needed. Xilinx ISE is used for getting the timing and power information.

For timing information, two main operation modules of the arithmetic circuit are considered. The reciprocal of the largest delay is taken to determine the maximum frequency. The equation between delay and maximum frequency is as follows:

$$\frac{1}{T_{delay}} = Frequency \quad (11)$$

Timing Information

Radix-4 Booth Multiplier

The timing analysis was performed with Xilinx ISE. The default settings were applied since there were no timing constraints used. The longest delay was found between mp(3) and p(14) at 15.793 ns. By using the equation presented above, the maximum frequency is 63.32 MHz. The full timing report from Xilinx ISE is shown in [Appendix Timing Analysis Report for 8-bit Radix-4 Booth Multiplier Component](#).

CSA

16-bit CSA. The timing analysis was performed with Xilinx ISE. The default settings were applied since there were no timing constraints used. The longest delay was found between in_a(0) and carryout at 8.490 ns. By using the equation presented above, the maximum frequency is 117.79 MHz. The full timing report from Xilinx ISE is shown in [Appendix Timing Analysis Report for 16-bit CSA Component](#).

24-bit CSA. The timing analysis was performed with Xilinx ISE. The default settings were applied since there were no timing constraints used. The longest delay was found between in_a(0) and sum(22) at 9.675 ns. By using the equation presented above, the maximum frequency is 103.36 MHz. The full timing report from Xilinx ISE is shown in [Appendix Timing Analysis Report for 24-bit CSA Component](#).

Implementation Circuit

The full timing report from Xilinx ISE for non-pipelined version and pipelined version regarding the timing source values are shown in [Appendix Timing Analysis Report for Non-pipelined Implementation with Negation](#)

Output, Appendix Timing Analysis Report for Pipelined Implementation with Negation Output, and Appendix Timing Analysis Report for Pipelined Implementation with Separation Output.

Area and Power Results for the Non-Pipelined Version

Implementation with Negation Output

Area. When synthesizing the non-pipelined version, the total number of slices used was 377 or 4.06% of all available logic slices. The full report from Mentor Graphics PrecisionRTL regarding the area and resource used is shown in Appendix Area Report for Non-Pipelined Version with Negation Output.

Power. The power information was performed with Xilinx ISE. The results are shown in the Table 7.

Table 7
Power Information for Non-pipelined Version

Source	Voltage(V)	Power(W)
Vccint	1.5	0.06
Vccaux	2.5	0.025
Vcco25	2.5	0.003
Total power: 0.088W		

Area and Power Results for the Pipelined Version

Implementation with Negation Output

Area. When synthesizing the pipelined negation output version, the total number of slices used was 380 or 4.09% of all available logic slices. The full report from Mentor Graphics PrecisionRTL regarding the area and resource used is shown in Appendix Area Report for Pipelined Version with Negation Output.

Power. The power information was performed with Xilinx ISE. The results are shown in the Table 8.

Table 8
Power Information for Pipelined Version with Negation Output

Source	Voltage(V)	Power(W)
Vccint	1.5	0.06
Vccaux	2.5	0.025
Vcco25	2.5	0.003
Total power: 0.088W		

Implementation with Separation Output

When synthesizing the pipelined separation output version, the total number of slices used was 379 or 4.08% of all available logic slices. The full report from Mentor Graphics PrecisionRTL regarding the area and resource used is shown in Appendix Area Report for Pipelined Version with Separation Output.

Power. The power information was performed with Xilinx ISE. The results are shown in the Table 9.

Table 9

Power Information for Pipelined Version with Separation Output

Source	Voltage(V)	Power(W)
Vccint	1.5	0.06
Vccaux	2.5	0.025
Vcco25	2.5	0.003
Total power: 0.088W		

Appendices

Area Report for Non-Pipelined Version with Negation Output

```

// Precision RTL Synthesis 64-bit 2016.1.0.15 (Production Release) Wed Jun 8 09:35:56 PDT 2016
//
// Copyright (c) Mentor Graphics Corporation, 1996–2016, All Rights Reserved.
// Portions copyright 1991–2008 Compuware Corporation
// UNPUBLISHED, LICENSED SOFTWARE.
// CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
// PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS
//
// Running on Linux hu_ju@grace.encs.concordia.ca #1 SMP Tue Oct 12 08:40:46 CDT 2021
// 3.10.0-1160.45.1.el7.x86_64 x86_64
//
// Start time Mon Dec 6 09:38:30 2021
*****Device Utilization for 2VP20ff896*****
*****Resource Used Avail Utilization*****

```

Resource	Used	Avail	Utilization
IOs	36	556	6.47%
Global Buffers	2	16	12.50%
LUTs	754	18560	4.06%
CLB Slices	377	9280	4.06%
Dffs or Latches	34	20228	0.17%
Block RAMs	0	88	0.00%
Block Multipliers	0	88	0.00%
Block Multiplier Dffs	0	3168	0.00%
GT_CUSTOM	0	8	0.00%

```

*****Library: work      Cell: nonpipeline_circuit_8b_nega_out      View: arch*****
*****Cell          Library    References    Total Area*****

```

Cell	Library	References	Total Area
BUFGP	xcv2p	2 x	
CSA_24bit	work	1 x	40 gates
			40 LUTs
CSA_24bit_unfolded0	work	1 x	41 gates
			39 LUTs
FDCE_1	xcv2p	1 x	1 Dffs or Latches
FDC_1	xcv2p	33 x	33 Dffs or Latches
IBUF	xcv2p	17 x	
LUT2	xcv2p	2 x	2 LUTs
LUT3	xcv2p	14 x	14 LUTs
LUT4	xcv2p	5 x	5 LUTs
OBUF	xcv2p	17 x	
VCC	xcv2p	1 x	
r4b_multiplier_8b	work	1 x	7 MUXF5
			207 LUTs
			218 gates
r4b_multiplier_8b_unfolded0	work	1 x	2 MUXF5
			214 LUTs
			229 gates
r4b_multiplier_8b_unfolded1	work	1 x	4 MUXF5
			233 LUTs
			252 gates

```

Number of ports :           36
Number of nets :           236
Number of instances :       97
Number of references to this view :   0

Total accumulated area :
Number of Dffs or Latches : 34
Number of LUTs :            754
Number of MUXF5 :           13
Number of gates :           803
Number of accumulated instances : 840

```

***** IO Register Mapping Report *****					
Design: work.nonpipeline_circuit_8b_nega_out.arch					
Port	Direction	INFF	OUTFF	TRIFF	
clk	Input				
load	Input				
clr	Input				
a(7)	Input				
a(6)	Input				
a(5)	Input				
a(4)	Input				
a(3)	Input				
a(2)	Input				
a(1)	Input				
a(0)	Input				
b(7)	Input				
b(6)	Input				
b(5)	Input				
b(4)	Input				
b(3)	Input				
b(2)	Input				
b(1)	Input				
b(0)	Input				
end_flag	Output				
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 |      |      |
+-----+-----+-----+-----+
Total registers mapped: 0
```

Area Report for Pipelined Version with Negation Output

```
// Precision RTL Synthesis 64-bit 2016.1.0.15 (Production Release) Wed Jun  8 09:35:56 PDT 2016
//
// Copyright (c) Mentor Graphics Corporation, 1996-2016, All Rights Reserved.
// Portions copyright 1991-2008 Compuware Corporation
// UNPUBLISHED, LICENSED SOFTWARE.
// CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
// PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS
//
// Running on Linux hu_ju@grace.encs.concordia.ca #1 SMP Tue Oct 12 08:40:46 CDT 2021
// 3.10.0-1160.45.1.el7.x86_64 x86_64
//
// Start time Mon Dec  6 09:38:30 2021
```

```
*****  
Device Utilization for 2VP20ff896  
*****
```

Resource	Used	Avail	Utilization
IOs	36	556	6.47%
Global Buffers	2	16	12.50%
LUTs	760	18560	4.09%
CLB Slices	380	9280	4.09%
Dffs or Latches	106	20228	0.52%
Block RAMs	0	88	0.00%
Block Multipliers	0	88	0.00%
Block Multiplier Dffs	0	3168	0.00%
GT_CUSTOM	0	8	0.00%

```
*****  
Library: work      Cell: pipelining_circuit_8b_nega_out      View: arch  
*****
```

Cell	Library	References	Total Area
BUFGP	xcv2p	2 x	
FDCE_1	xcv2p	1 x	1 Dffs or Latches
FDCPE_1	xcv2p	1 x	1 Dffs or Latches
FDC_1	xcv2p	34 x	1 34 Dffs or Latches
FDE_1	xcv2p	3 x	3 Dffs or Latches
GND	xcv2p	1 x	
IBUF	xcv2p	17 x	
LUT2	xcv2p	5 x	5 LUTs
LUT3	xcv2p	18 x	18 LUTs
LUT4	xcv2p	4 x	4 LUTs
OBUF	xcv2p	17 x	
VCC	xcv2p	1 x	
operating_circuit_8b	work	1 x	67 Dffs or Latches
			781 gates
			734 LUTs
			13 MUXF5
Number of ports :			36
Number of nets :			146
Number of instances :			105
Number of references to this view :			0
Total accumulated area :			
Number of Dffs or Latches :			106
Number of LUTs :			760
Number of MUXF5 :			13
Number of gates :			811
Number of accumulated instances :			920

```
*****
```

IO Register Mapping Report

Design: work.pipeline_circuit_8b_nega_out.arch

Port	Direction	INFF	OUTFF	TRIFF
clk	Input			
load	Input			
clr	Input			
a(7)	Input			
a(6)	Input			
a(5)	Input			
a(4)	Input			
a(3)	Input			
a(2)	Input			
a(1)	Input			
a(0)	Input			
b(7)	Input			
b(6)	Input			
b(5)	Input			
b(4)	Input			
b(3)	Input			
b(2)	Input			
b(1)	Input			
b(0)	Input			
end_flag	Output			
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			

Total registers mapped: 0

Area Report for Pipelined Version with Separation Output

```
// Precision RTL Synthesis 64-bit 2016.1.0.15 (Production Release) Wed Jun 8 09:35:56 PDT 2016
//
// Copyright (c) Mentor Graphics Corporation, 1996-2016, All Rights Reserved.
// Portions copyright 1991-2008 Compuware Corporation
// UNPUBLISHED, LICENSED SOFTWARE.
// CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
// PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS
//
// Running on Linux hu_ju@poise.encs.concordia.ca #1 SMP Tue Oct 12 08:40:46 CDT 2021
3.10.0-1160.45.1.e17.x86_64 x86_64
//
// Start time Sat Dec 4 21:22:54 2021
```

Device Utilization for 2VP20ff896

Resource	Used	Avail	Utilization
IOs	36	556	6.47%
Global Buffers	2	16	12.50%
LUTs	757	18560	4.08%
CLB Slices	379	9280	4.08%
Dffs or Latches	115	20228	0.57%
Block RAMs	0	88	0.00%
Block Multipliers	0	88	0.00%
Block Multiplier Dffs	0	3168	0.00%
GT_CUSTOM	0	8	0.00%

Library: work Cell: pipelining_circuit_8b_sep_out View: arch

Cell	Library	References	Total Area
BUFGP	xcv2p	2 x	
FDCE_1	xcv2p	1 x	1 Dffs or Latches
FDCPE_1	xcv2p	1 x	1 Dffs or Latches
FDC_1	xcv2p	42 x	42 Dffs or Latches
FDE_1	xcv2p	3 x	3 Dffs or Latches
FD_1	xcv2p	1 x	1 Dffs or Latches
GND	xcv2p	1 x	
IBUF	xcv2p	17 x	
LUT1	xcv2p	1 x	1 LUTs
LUT2	xcv2p	4 x	4 LUTs
LUT3	xcv2p	11 x	11 LUTs
LUT4	xcv2p	9 x	9 LUTs
OBUF	xcv2p	17 x	
VCC	xcv2p	1 x	
operating_circuit_8b	work	1 x	67 Dffs or Latches
		781	781 gates
		734	734 LUTs
		13	13 MUXF5

Number of ports : 36

Number of nets : 153

Number of instances : 112

Number of references to this view : 0

Total accumulated area :

Number of Dffs or Latches : 115

Number of LUTs : 757

Number of MUXF5 : 13

Number of gates : 809

Number of accumulated instances : 927

IO Register Mapping Report

Design: work.pipelining_circuit_8b_sep_out.arch

Port	Direction	INFF	OUTFF	TRIFF
clk	Input			
load	Input			
clr	Input			
a(7)	Input			
a(6)	Input			
a(5)	Input			
a(4)	Input			
a(3)	Input			
a(2)	Input			
a(1)	Input			
a(0)	Input			
b(7)	Input			
b(6)	Input			
b(5)	Input			
b(4)	Input			
b(3)	Input			
b(2)	Input			
b(1)	Input			
b(0)	Input			
end_flag	Output			
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			

Total registers mapped: 0

Timing Analysis Report for 8-bit Radix-4 Booth Multiplier Component

Release 10.1 Trace (lin64)
 Copyright (c) 1995–2008 Xilinx, Inc. All rights reserved.
 /nfs/sw_cmc/x86_64.EL7/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce
 r4b_multiplier_8b_out r4b_multiplier_8b.pcf -v 10 -o r4b_multiplier_8b_out

Design file: r4b_multiplier_8b_out.ncd
 Physical constraint file: r4b_multiplier_8b.pcf
 Device, package, speed: xc2vp20,ff896,-7 (PRODUCTION 1.94 2008-01-09)
 Report level: verbose report, limited to 10 items per constraint

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
mc(0)	lp(0)	4.7751
mc(0)	lp(1)	5.2071
mc(0)	lp(2)	5.7291
mc(0)	lp(3)	6.0661
mc(0)	lp(4)	6.9091
mc(0)	lp(5)	7.9961
mc(0)	lp(6)	8.6151
mc(0)	lp(7)	10.6511
mc(0)	lp(8)	12.2591
mc(0)	lp(9)	12.6671
mc(0)	lp(10)	12.9661
mc(0)	lp(11)	14.1251
mc(0)	lp(12)	14.4961
mc(0)	lp(13)	14.9091
mc(0)	lp(14)	15.3961
mc(0)	lp(15)	15.5761
mc(1)	lp(1)	4.8551
mc(1)	lp(2)	6.2741
mc(1)	lp(3)	6.5221
mc(1)	lp(4)	7.3791
mc(1)	lp(5)	8.4401
mc(1)	lp(6)	8.9251
mc(1)	lp(7)	10.7671
mc(1)	lp(8)	12.3751
mc(1)	lp(9)	12.7831
mc(1)	lp(10)	13.0821
mc(1)	lp(11)	14.2411
mc(1)	lp(12)	14.6121
mc(1)	lp(13)	15.0251
mc(1)	lp(14)	15.5121
mc(1)	lp(15)	15.6921
mc(2)	lp(2)	5.5671
mc(2)	lp(3)	5.8151
mc(2)	lp(4)	6.6721
mc(2)	lp(5)	7.8341
mc(2)	lp(6)	8.4481
mc(2)	lp(7)	10.0971
mc(2)	lp(8)	11.7051

mc(2)	p(9)	12.113
mc(2)	p(10)	12.412
mc(2)	p(11)	13.571
mc(2)	p(12)	13.942
mc(2)	p(13)	14.355
mc(2)	p(14)	14.842
mc(2)	p(15)	15.022
mc(3)	p(3)	6.101
mc(3)	p(4)	6.944
mc(3)	p(5)	8.005
mc(3)	p(6)	8.504
mc(3)	p(7)	10.540
mc(3)	p(8)	12.148
mc(3)	p(9)	12.556
mc(3)	p(10)	12.855
mc(3)	p(11)	14.014
mc(3)	p(12)	14.385
mc(3)	p(13)	14.798
mc(3)	p(14)	15.285
mc(3)	p(15)	15.465
mc(4)	p(4)	6.224
mc(4)	p(5)	7.296
mc(4)	p(6)	7.797
mc(4)	p(7)	9.658
mc(4)	p(8)	11.266
mc(4)	p(9)	11.674
mc(4)	p(10)	11.973
mc(4)	p(11)	13.132
mc(4)	p(12)	13.503
mc(4)	p(13)	13.916
mc(4)	p(14)	14.459
mc(4)	p(15)	14.583
mc(5)	p(5)	6.170
mc(5)	p(6)	8.217
mc(5)	p(7)	10.253
mc(5)	p(8)	11.861
mc(5)	p(9)	12.269
mc(5)	p(10)	12.568
mc(5)	p(11)	13.727
mc(5)	p(12)	14.098
mc(5)	p(13)	14.511
mc(5)	p(14)	14.998
mc(5)	p(15)	15.178
mc(6)	p(6)	7.075
mc(6)	p(7)	9.111
mc(6)	p(8)	10.719
mc(6)	p(9)	11.486
mc(6)	p(10)	11.744
mc(6)	p(11)	12.903
mc(6)	p(12)	13.274
mc(6)	p(13)	13.687
mc(6)	p(14)	14.349
mc(6)	p(15)	14.354
mc(7)	p(7)	8.767
mc(7)	p(8)	10.379
mc(7)	p(9)	11.082
mc(7)	p(10)	11.412
mc(7)	p(11)	12.499
mc(7)	p(12)	12.870
mc(7)	p(13)	13.283
mc(7)	p(14)	14.183
mc(7)	p(15)	14.007
mp(0)	p(0)	4.654
mp(0)	p(1)	4.745
mp(0)	p(2)	5.239
mp(0)	p(3)	5.487
mp(0)	p(4)	6.344
mp(0)	p(5)	7.405
mp(0)	p(6)	7.911
mp(0)	p(7)	9.935
mp(0)	p(8)	11.543
mp(0)	p(9)	11.951
mp(0)	p(10)	12.250
mp(0)	p(11)	13.409

mp(0)	p(12)	13.780
mp(0)	p(13)	14.193
mp(0)	p(14)	14.680
mp(0)	p(15)	14.860
mp(1)	p(1)	4.802
mp(1)	p(2)	5.448
mp(1)	p(3)	6.972
mp(1)	p(4)	7.662
mp(1)	p(5)	8.723
mp(1)	p(6)	9.012
mp(1)	p(7)	10.404
mp(1)	p(8)	12.012
mp(1)	p(9)	12.420
mp(1)	p(10)	12.858
mp(1)	p(11)	13.878
mp(1)	p(12)	14.249
mp(1)	p(13)	14.662
mp(1)	p(14)	15.512
mp(1)	p(15)	15.336
mp(2)	p(2)	3.943
mp(2)	p(3)	6.481
mp(2)	p(4)	7.171
mp(2)	p(5)	8.232
mp(2)	p(6)	8.521
mp(2)	p(7)	9.913
mp(2)	p(8)	11.521
mp(2)	p(9)	11.929
mp(2)	p(10)	12.367
mp(2)	p(11)	13.387
mp(2)	p(12)	13.758
mp(2)	p(13)	14.171
mp(2)	p(14)	15.021
mp(2)	p(15)	14.845
mp(3)	p(3)	7.253
mp(3)	p(4)	7.943
mp(3)	p(5)	9.004
mp(3)	p(6)	9.293
mp(3)	p(7)	10.685
mp(3)	p(8)	12.293
mp(3)	p(9)	12.701
mp(3)	p(10)	13.139
mp(3)	p(11)	14.159
mp(3)	p(12)	14.530
mp(3)	p(13)	14.943
mp(3)	p(14)	15.793
mp(3)	p(15)	15.617
mp(4)	p(4)	5.157
mp(4)	p(5)	6.749
mp(4)	p(6)	7.529
mp(4)	p(7)	8.809
mp(4)	p(8)	10.417
mp(4)	p(9)	10.825
mp(4)	p(10)	11.124
mp(4)	p(11)	12.283
mp(4)	p(12)	12.654
mp(4)	p(13)	13.067
mp(4)	p(14)	13.554
mp(4)	p(15)	13.734
mp(5)	p(5)	7.298
mp(5)	p(6)	8.078
mp(5)	p(7)	9.358
mp(5)	p(8)	10.966
mp(5)	p(9)	11.374
mp(5)	p(10)	11.673
mp(5)	p(11)	12.832
mp(5)	p(12)	13.203
mp(5)	p(13)	13.616
mp(5)	p(14)	14.103
mp(5)	p(15)	14.283
mp(6)	p(6)	4.562
mp(6)	p(7)	6.143
mp(6)	p(8)	7.187
mp(6)	p(9)	7.754
mp(6)	p(10)	7.944

mp(6)	p(11)	8.617
mp(6)	p(12)	8.769
mp(6)	p(13)	9.182
mp(6)	p(14)	9.669
mp(6)	p(15)	9.849
mp(7)	p(7)	6.892
mp(7)	p(8)	7.936
mp(7)	p(9)	8.503
mp(7)	p(10)	8.693
mp(7)	p(11)	9.366
mp(7)	p(12)	9.519
mp(7)	p(13)	9.932
mp(7)	p(14)	10.419
mp(7)	p(15)	10.599

Analysis completed Mon Dec 6 12:11:08 2021

Trace Settings:

Trace Settings

Peak Memory Usage: 274 MB

Timing Analysis Report for 16-bit CSA Component

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

/nfs/sw_cmc/x86_64.EL7/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce
CSA_16bit_out CSA_16bit.pcf -v 10 -o CSA_16bit_out

Design file: CSA_16bit_out.ncd
Physical constraint file: CSA_16bit.pcf
Device, package, speed: xc2vp20,ff896,-7 (PRODUCTION 1.94 2008-01-09)
Report level: verbose report, limited to 10 items per constraint

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
in_a(0)	carryout	8.490
in_a(0)	sum(0)	3.854
in_a(0)	sum(1)	3.708
in_a(0)	sum(2)	4.935
in_a(0)	sum(3)	5.379
in_a(0)	sum(4)	5.854
in_a(0)	sum(5)	6.001
in_a(0)	sum(6)	6.781
in_a(0)	sum(7)	6.356
in_a(0)	sum(8)	6.888
in_a(0)	sum(9)	7.419
in_a(0)	sum(10)	7.268

in_a(0)	sum(11)	7.159
in_a(0)	sum(12)	8.420
in_a(0)	sum(13)	8.297
in_a(0)	sum(14)	8.270
in_a(0)	sum(15)	7.601
in_a(1)	carryout	8.169
in_a(1)	sum(1)	4.025
in_a(1)	sum(2)	4.614
in_a(1)	sum(3)	5.058
in_a(1)	sum(4)	5.533
in_a(1)	sum(5)	5.680
in_a(1)	sum(6)	6.460
in_a(1)	sum(7)	6.035
in_a(1)	sum(8)	6.567
in_a(1)	sum(9)	7.098
in_a(1)	sum(10)	6.947
in_a(1)	sum(11)	6.838
in_a(1)	sum(12)	8.099
in_a(1)	sum(13)	7.976
in_a(1)	sum(14)	7.949
in_a(1)	sum(15)	7.280
in_a(2)	carryout	8.226
in_a(2)	sum(2)	4.675
in_a(2)	sum(3)	5.115
in_a(2)	sum(4)	5.590
in_a(2)	sum(5)	5.737
in_a(2)	sum(6)	6.517
in_a(2)	sum(7)	6.092
in_a(2)	sum(8)	6.624
in_a(2)	sum(9)	7.155
in_a(2)	sum(10)	7.004
in_a(2)	sum(11)	6.895
in_a(2)	sum(12)	8.156
in_a(2)	sum(13)	8.033
in_a(2)	sum(14)	8.006
in_a(2)	sum(15)	7.337
in_a(3)	carryout	7.122
in_a(3)	sum(3)	3.813
in_a(3)	sum(4)	4.493
in_a(3)	sum(5)	4.640
in_a(3)	sum(6)	5.420
in_a(3)	sum(7)	4.995
in_a(3)	sum(8)	5.520
in_a(3)	sum(9)	6.051
in_a(3)	sum(10)	5.900
in_a(3)	sum(11)	5.791
in_a(3)	sum(12)	7.052
in_a(3)	sum(13)	6.929
in_a(3)	sum(14)	6.902
in_a(3)	sum(15)	6.233
in_a(4)	carryout	7.804
in_a(4)	sum(4)	3.842
in_a(4)	sum(5)	3.799
in_a(4)	sum(6)	5.148
in_a(4)	sum(7)	5.437
in_a(4)	sum(8)	6.202
in_a(4)	sum(9)	6.733
in_a(4)	sum(10)	6.582
in_a(4)	sum(11)	6.473
in_a(4)	sum(12)	7.734
in_a(4)	sum(13)	7.611
in_a(4)	sum(14)	7.584
in_a(4)	sum(15)	6.915
in_a(5)	carryout	7.998
in_a(5)	sum(5)	4.471
in_a(5)	sum(6)	5.398
in_a(5)	sum(7)	5.732
in_a(5)	sum(8)	6.396
in_a(5)	sum(9)	6.927
in_a(5)	sum(10)	6.776
in_a(5)	sum(11)	6.667
in_a(5)	sum(12)	7.928
in_a(5)	sum(13)	7.805
in_a(5)	sum(14)	7.778

in_a(5)	sum(15)	7.109
in_a(6)	carryout	7.520
in_a(6)	sum(6)	4.825
in_a(6)	sum(7)	5.167
in_a(6)	sum(8)	5.918
in_a(6)	sum(9)	6.449
in_a(6)	sum(10)	6.298
in_a(6)	sum(11)	6.189
in_a(6)	sum(12)	7.450
in_a(6)	sum(13)	7.327
in_a(6)	sum(14)	7.300
in_a(6)	sum(15)	6.631
in_a(7)	carryout	6.419
in_a(7)	sum(7)	4.570
in_a(7)	sum(8)	4.817
in_a(7)	sum(9)	5.348
in_a(7)	sum(10)	5.197
in_a(7)	sum(11)	5.088
in_a(7)	sum(12)	6.349
in_a(7)	sum(13)	6.226
in_a(7)	sum(14)	6.199
in_a(7)	sum(15)	5.530
in_a(8)	carryout	6.645
in_a(8)	sum(8)	4.062
in_a(8)	sum(9)	4.299
in_a(8)	sum(10)	4.702
in_a(8)	sum(11)	5.078
in_a(8)	sum(12)	6.575
in_a(8)	sum(13)	6.452
in_a(8)	sum(14)	6.425
in_a(8)	sum(15)	5.756
in_a(9)	carryout	6.389
in_a(9)	sum(9)	4.390
in_a(9)	sum(10)	4.349
in_a(9)	sum(11)	4.863
in_a(9)	sum(12)	6.319
in_a(9)	sum(13)	6.196
in_a(9)	sum(14)	6.169
in_a(9)	sum(15)	5.500
in_a(10)	carryout	6.472
in_a(10)	sum(10)	4.255
in_a(10)	sum(11)	4.946
in_a(10)	sum(12)	6.402
in_a(10)	sum(13)	6.279
in_a(10)	sum(14)	6.252
in_a(10)	sum(15)	5.583
in_a(11)	carryout	6.365
in_a(11)	sum(11)	4.478
in_a(11)	sum(12)	6.295
in_a(11)	sum(13)	6.172
in_a(11)	sum(14)	6.145
in_a(11)	sum(15)	5.476
in_a(12)	carryout	5.291
in_a(12)	sum(12)	4.070
in_a(12)	sum(13)	4.103
in_a(12)	sum(14)	4.209
in_a(12)	sum(15)	4.986
in_a(13)	carryout	5.373
in_a(13)	sum(13)	4.528
in_a(13)	sum(14)	4.291
in_a(13)	sum(15)	5.064
in_a(14)	carryout	5.261
in_a(14)	sum(14)	4.132
in_a(14)	sum(15)	4.952
in_a(15)	carryout	4.582
in_a(15)	sum(15)	4.908
in_b(0)	carryout	8.265
in_b(0)	sum(0)	3.889
in_b(0)	sum(1)	3.739
in_b(0)	sum(2)	4.710
in_b(0)	sum(3)	5.154
in_b(0)	sum(4)	5.629
in_b(0)	sum(5)	5.776
in_b(0)	sum(6)	6.556

in_b(0)	sum(7)	6.131
in_b(0)	sum(8)	6.663
in_b(0)	sum(9)	7.194
in_b(0)	sum(10)	7.043
in_b(0)	sum(11)	6.934
in_b(0)	sum(12)	8.195
in_b(0)	sum(13)	8.072
in_b(0)	sum(14)	8.045
in_b(0)	sum(15)	7.376
in_b(1)	carryout	8.074
in_b(1)	sum(1)	3.965
in_b(1)	sum(2)	4.519
in_b(1)	sum(3)	4.963
in_b(1)	sum(4)	5.438
in_b(1)	sum(5)	5.585
in_b(1)	sum(6)	6.365
in_b(1)	sum(7)	5.940
in_b(1)	sum(8)	6.472
in_b(1)	sum(9)	7.003
in_b(1)	sum(10)	6.852
in_b(1)	sum(11)	6.743
in_b(1)	sum(12)	8.004
in_b(1)	sum(13)	7.881
in_b(1)	sum(14)	7.854
in_b(1)	sum(15)	7.185
in_b(2)	carryout	7.760
in_b(2)	sum(2)	4.203
in_b(2)	sum(3)	4.649
in_b(2)	sum(4)	5.124
in_b(2)	sum(5)	5.271
in_b(2)	sum(6)	6.051
in_b(2)	sum(7)	5.626
in_b(2)	sum(8)	6.158
in_b(2)	sum(9)	6.689
in_b(2)	sum(10)	6.538
in_b(2)	sum(11)	6.429
in_b(2)	sum(12)	7.690
in_b(2)	sum(13)	7.567
in_b(2)	sum(14)	7.540
in_b(2)	sum(15)	6.871
in_b(3)	carryout	6.991
in_b(3)	sum(3)	3.884
in_b(3)	sum(4)	4.457
in_b(3)	sum(5)	4.604
in_b(3)	sum(6)	5.384
in_b(3)	sum(7)	4.959
in_b(3)	sum(8)	5.389
in_b(3)	sum(9)	5.920
in_b(3)	sum(10)	5.769
in_b(3)	sum(11)	5.660
in_b(3)	sum(12)	6.921
in_b(3)	sum(13)	6.798
in_b(3)	sum(14)	6.771
in_b(3)	sum(15)	6.102
in_b(4)	carryout	7.435
in_b(4)	sum(4)	3.932
in_b(4)	sum(5)	3.903
in_b(4)	sum(6)	4.779
in_b(4)	sum(7)	5.097
in_b(4)	sum(8)	5.833
in_b(4)	sum(9)	6.364
in_b(4)	sum(10)	6.213
in_b(4)	sum(11)	6.104
in_b(4)	sum(12)	7.365
in_b(4)	sum(13)	7.242
in_b(4)	sum(14)	7.215
in_b(4)	sum(15)	6.546
in_b(5)	carryout	7.490
in_b(5)	sum(5)	4.429
in_b(5)	sum(6)	4.834
in_b(5)	sum(7)	5.123
in_b(5)	sum(8)	5.888
in_b(5)	sum(9)	6.419
in_b(5)	sum(10)	6.268

in_b(5)	sum(11)	6.159
in_b(5)	sum(12)	7.420
in_b(5)	sum(13)	7.297
in_b(5)	sum(14)	7.270
in_b(5)	sum(15)	6.601
in_b(6)	carryout	7.369
in_b(6)	sum(6)	4.602
in_b(6)	sum(7)	4.989
in_b(6)	sum(8)	5.767
in_b(6)	sum(9)	6.298
in_b(6)	sum(10)	6.147
in_b(6)	sum(11)	6.038
in_b(6)	sum(12)	7.299
in_b(6)	sum(13)	7.176
in_b(6)	sum(14)	7.149
in_b(6)	sum(15)	6.480
in_b(7)	carryout	6.498
in_b(7)	sum(7)	4.649
in_b(7)	sum(8)	4.896
in_b(7)	sum(9)	5.427
in_b(7)	sum(10)	5.276
in_b(7)	sum(11)	5.167
in_b(7)	sum(12)	6.428
in_b(7)	sum(13)	6.305
in_b(7)	sum(14)	6.278
in_b(7)	sum(15)	5.609
in_b(8)	carryout	6.260
in_b(8)	sum(8)	3.993
in_b(8)	sum(9)	4.090
in_b(8)	sum(10)	4.317
in_b(8)	sum(11)	4.716
in_b(8)	sum(12)	6.190
in_b(8)	sum(13)	6.067
in_b(8)	sum(14)	6.040
in_b(8)	sum(15)	5.371
in_b(9)	carryout	6.362
in_b(9)	sum(9)	4.323
in_b(9)	sum(10)	4.403
in_b(9)	sum(11)	4.836
in_b(9)	sum(12)	6.292
in_b(9)	sum(13)	6.169
in_b(9)	sum(14)	6.142
in_b(9)	sum(15)	5.473
in_b(10)	carryout	6.298
in_b(10)	sum(10)	4.422
in_b(10)	sum(11)	4.772
in_b(10)	sum(12)	6.228
in_b(10)	sum(13)	6.105
in_b(10)	sum(14)	6.078
in_b(10)	sum(15)	5.409
in_b(11)	carryout	6.611
in_b(11)	sum(11)	4.724
in_b(11)	sum(12)	6.541
in_b(11)	sum(13)	6.418
in_b(11)	sum(14)	6.391
in_b(11)	sum(15)	5.722
in_b(12)	carryout	5.745
in_b(12)	sum(12)	4.256
in_b(12)	sum(13)	4.214
in_b(12)	sum(14)	4.663
in_b(12)	sum(15)	5.436
in_b(13)	carryout	5.472
in_b(13)	sum(13)	4.632
in_b(13)	sum(14)	4.390
in_b(13)	sum(15)	5.163
in_b(14)	carryout	5.121
in_b(14)	sum(14)	4.605
in_b(14)	sum(15)	4.812
in_b(15)	carryout	4.810
in_b(15)	sum(15)	5.131

Analysis completed Mon Dec 6 12:12:40 2021

Trace Settings:

Trace Settings

Peak Memory Usage: 272 MB

Timing Analysis Report for 24-bit CSA Component

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

/nfs/sw_cmc/x86_64.EL7/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce
CSA_24bit_out CSA_24bit.pcf -v 10 -o CSA_24bit_out

Design file: CSA_24bit_out.ncd
Physical constraint file: CSA_24bit.pcf
Device, package, speed: xc2vp20,ff896,-7 (PRODUCTION 1.94 2008-01-09)
Report level: verbose report, limited to 10 items per constraint

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
in_a(0)	lcarryout	8.431
in_a(0)	lsum(0)	4.000
in_a(0)	lsum(1)	4.153
in_a(0)	lsum(2)	4.549
in_a(0)	lsum(3)	5.312
in_a(0)	lsum(4)	5.759
in_a(0)	lsum(5)	5.737
in_a(0)	lsum(6)	5.962
in_a(0)	lsum(7)	5.828
in_a(0)	lsum(8)	6.534
in_a(0)	lsum(9)	6.564
in_a(0)	lsum(10)	6.343
in_a(0)	lsum(11)	7.066
in_a(0)	lsum(12)	7.342
in_a(0)	lsum(13)	7.649
in_a(0)	lsum(14)	7.031
in_a(0)	lsum(15)	7.419
in_a(0)	lsum(16)	8.274
in_a(0)	lsum(17)	9.007
in_a(0)	lsum(18)	8.430
in_a(0)	lsum(19)	8.117
in_a(0)	lsum(20)	8.474
in_a(0)	lsum(21)	9.582
in_a(0)	lsum(22)	9.675
in_a(0)	lsum(23)	9.316
in_a(1)	lcarryout	8.098
in_a(1)	lsum(1)	3.732
in_a(1)	lsum(2)	4.216
in_a(1)	lsum(3)	4.979
in_a(1)	lsum(4)	5.426

in_a(1)	sum(5)	5.404
in_a(1)	sum(6)	5.629
in_a(1)	sum(7)	5.495
in_a(1)	sum(8)	6.201
in_a(1)	sum(9)	6.231
in_a(1)	sum(10)	6.010
in_a(1)	sum(11)	6.733
in_a(1)	sum(12)	7.009
in_a(1)	sum(13)	7.316
in_a(1)	sum(14)	6.698
in_a(1)	sum(15)	7.086
in_a(1)	sum(16)	7.941
in_a(1)	sum(17)	8.674
in_a(1)	sum(18)	8.097
in_a(1)	sum(19)	7.784
in_a(1)	sum(20)	8.141
in_a(1)	sum(21)	9.249
in_a(1)	sum(22)	9.342
in_a(1)	sum(23)	8.983
in_a(2)	carryout	7.334
in_a(2)	sum(2)	3.791
in_a(2)	sum(3)	4.210
in_a(2)	sum(4)	4.999
in_a(2)	sum(5)	4.977
in_a(2)	sum(6)	5.202
in_a(2)	sum(7)	4.731
in_a(2)	sum(8)	5.437
in_a(2)	sum(9)	5.467
in_a(2)	sum(10)	5.246
in_a(2)	sum(11)	5.969
in_a(2)	sum(12)	6.245
in_a(2)	sum(13)	6.552
in_a(2)	sum(14)	5.934
in_a(2)	sum(15)	6.322
in_a(2)	sum(16)	7.177
in_a(2)	sum(17)	7.910
in_a(2)	sum(18)	7.333
in_a(2)	sum(19)	7.020
in_a(2)	sum(20)	7.377
in_a(2)	sum(21)	8.485
in_a(2)	sum(22)	8.578
in_a(2)	sum(23)	8.219
in_a(3)	carryout	7.916
in_a(3)	sum(3)	4.799
in_a(3)	sum(4)	4.730
in_a(3)	sum(5)	4.708
in_a(3)	sum(6)	4.933
in_a(3)	sum(7)	5.313
in_a(3)	sum(8)	6.019
in_a(3)	sum(9)	6.049
in_a(3)	sum(10)	5.828
in_a(3)	sum(11)	6.551
in_a(3)	sum(12)	6.827
in_a(3)	sum(13)	7.134
in_a(3)	sum(14)	6.516
in_a(3)	sum(15)	6.904
in_a(3)	sum(16)	7.759
in_a(3)	sum(17)	8.492
in_a(3)	sum(18)	7.915
in_a(3)	sum(19)	7.602
in_a(3)	sum(20)	7.959
in_a(3)	sum(21)	9.067
in_a(3)	sum(22)	9.160
in_a(3)	sum(23)	8.801
in_a(4)	carryout	8.059
in_a(4)	sum(4)	4.649
in_a(4)	sum(5)	4.631
in_a(4)	sum(6)	4.875
in_a(4)	sum(7)	5.456
in_a(4)	sum(8)	6.162
in_a(4)	sum(9)	6.192
in_a(4)	sum(10)	5.971
in_a(4)	sum(11)	6.694
in_a(4)	sum(12)	6.970

in_a(4)	sum(13)	7.277
in_a(4)	sum(14)	6.659
in_a(4)	sum(15)	7.047
in_a(4)	sum(16)	7.902
in_a(4)	sum(17)	8.635
in_a(4)	sum(18)	8.058
in_a(4)	sum(19)	7.745
in_a(4)	sum(20)	8.102
in_a(4)	sum(21)	9.210
in_a(4)	sum(22)	9.303
in_a(4)	sum(23)	8.944
in_a(5)	carryout	7.908
in_a(5)	sum(5)	4.709
in_a(5)	sum(6)	4.220
in_a(5)	sum(7)	5.305
in_a(5)	sum(8)	6.011
in_a(5)	sum(9)	6.041
in_a(5)	sum(10)	5.820
in_a(5)	sum(11)	6.543
in_a(5)	sum(12)	6.819
in_a(5)	sum(13)	7.126
in_a(5)	sum(14)	6.508
in_a(5)	sum(15)	6.896
in_a(5)	sum(16)	7.751
in_a(5)	sum(17)	8.484
in_a(5)	sum(18)	7.907
in_a(5)	sum(19)	7.594
in_a(5)	sum(20)	7.951
in_a(5)	sum(21)	9.059
in_a(5)	sum(22)	9.152
in_a(5)	sum(23)	8.793
in_a(6)	carryout	7.860
in_a(6)	sum(6)	4.313
in_a(6)	sum(7)	5.257
in_a(6)	sum(8)	5.963
in_a(6)	sum(9)	5.993
in_a(6)	sum(10)	5.772
in_a(6)	sum(11)	6.495
in_a(6)	sum(12)	6.771
in_a(6)	sum(13)	7.078
in_a(6)	sum(14)	6.460
in_a(6)	sum(15)	6.848
in_a(6)	sum(16)	7.703
in_a(6)	sum(17)	8.436
in_a(6)	sum(18)	7.859
in_a(6)	sum(19)	7.546
in_a(6)	sum(20)	7.903
in_a(6)	sum(21)	9.011
in_a(6)	sum(22)	9.104
in_a(6)	sum(23)	8.745
in_a(7)	carryout	6.844
in_a(7)	sum(7)	4.239
in_a(7)	sum(8)	5.153
in_a(7)	sum(9)	4.643
in_a(7)	sum(10)	4.211
in_a(7)	sum(11)	5.479
in_a(7)	sum(12)	5.755
in_a(7)	sum(13)	6.062
in_a(7)	sum(14)	5.444
in_a(7)	sum(15)	5.832
in_a(7)	sum(16)	6.687
in_a(7)	sum(17)	7.420
in_a(7)	sum(18)	6.843
in_a(7)	sum(19)	6.530
in_a(7)	sum(20)	6.887
in_a(7)	sum(21)	7.995
in_a(7)	sum(22)	8.088
in_a(7)	sum(23)	7.729
in_a(8)	carryout	7.678
in_a(8)	sum(8)	5.815
in_a(8)	sum(9)	5.376
in_a(8)	sum(10)	5.208
in_a(8)	sum(11)	6.313
in_a(8)	sum(12)	6.589

in_a(8)	sum(13)	6.896
in_a(8)	sum(14)	6.278
in_a(8)	sum(15)	6.666
in_a(8)	sum(16)	7.521
in_a(8)	sum(17)	8.254
in_a(8)	sum(18)	7.677
in_a(8)	sum(19)	7.364
in_a(8)	sum(20)	7.721
in_a(8)	sum(21)	8.829
in_a(8)	sum(22)	8.922
in_a(8)	sum(23)	8.563
in_a(9)	carryout	7.191
in_a(9)	sum(9)	4.443
in_a(9)	sum(10)	4.721
in_a(9)	sum(11)	5.826
in_a(9)	sum(12)	6.102
in_a(9)	sum(13)	6.409
in_a(9)	sum(14)	5.791
in_a(9)	sum(15)	6.179
in_a(9)	sum(16)	7.034
in_a(9)	sum(17)	7.767
in_a(9)	sum(18)	7.190
in_a(9)	sum(19)	6.877
in_a(9)	sum(20)	7.234
in_a(9)	sum(21)	8.342
in_a(9)	sum(22)	8.435
in_a(9)	sum(23)	8.076
in_a(10)	carryout	7.255
in_a(10)	sum(10)	3.949
in_a(10)	sum(11)	5.890
in_a(10)	sum(12)	6.166
in_a(10)	sum(13)	6.473
in_a(10)	sum(14)	5.855
in_a(10)	sum(15)	6.243
in_a(10)	sum(16)	7.098
in_a(10)	sum(17)	7.831
in_a(10)	sum(18)	7.254
in_a(10)	sum(19)	6.941
in_a(10)	sum(20)	7.298
in_a(10)	sum(21)	8.406
in_a(10)	sum(22)	8.499
in_a(10)	sum(23)	8.140
in_a(11)	carryout	6.146
in_a(11)	sum(11)	4.597
in_a(11)	sum(12)	5.057
in_a(11)	sum(13)	5.364
in_a(11)	sum(14)	4.746
in_a(11)	sum(15)	5.134
in_a(11)	sum(16)	5.989
in_a(11)	sum(17)	6.722
in_a(11)	sum(18)	6.145
in_a(11)	sum(19)	5.832
in_a(11)	sum(20)	6.189
in_a(11)	sum(21)	7.297
in_a(11)	sum(22)	7.390
in_a(11)	sum(23)	7.031
in_a(12)	carryout	6.335
in_a(12)	sum(12)	4.187
in_a(12)	sum(13)	4.327
in_a(12)	sum(14)	4.683
in_a(12)	sum(15)	4.925
in_a(12)	sum(16)	6.178
in_a(12)	sum(17)	6.911
in_a(12)	sum(18)	6.334
in_a(12)	sum(19)	6.021
in_a(12)	sum(20)	6.378
in_a(12)	sum(21)	7.486
in_a(12)	sum(22)	7.579
in_a(12)	sum(23)	7.220
in_a(13)	carryout	6.483
in_a(13)	sum(13)	4.628
in_a(13)	sum(14)	4.832
in_a(13)	sum(15)	5.045
in_a(13)	sum(16)	6.326

in_a(13)	sum(17)	7.059
in_a(13)	sum(18)	6.482
in_a(13)	sum(19)	6.169
in_a(13)	sum(20)	6.526
in_a(13)	sum(21)	7.634
in_a(13)	sum(22)	7.727
in_a(13)	sum(23)	7.368
in_a(14)	carryout	5.886
in_a(14)	sum(14)	4.205
in_a(14)	sum(15)	4.419
in_a(14)	sum(16)	5.729
in_a(14)	sum(17)	6.462
in_a(14)	sum(18)	5.885
in_a(14)	sum(19)	5.572
in_a(14)	sum(20)	5.929
in_a(14)	sum(21)	7.037
in_a(14)	sum(22)	7.130
in_a(14)	sum(23)	6.771
in_a(15)	carryout	7.038
in_a(15)	sum(15)	5.208
in_a(15)	sum(16)	4.912
in_a(15)	sum(17)	5.645
in_a(15)	sum(18)	5.068
in_a(15)	sum(19)	4.755
in_a(15)	sum(20)	7.081
in_a(15)	sum(21)	8.189
in_a(15)	sum(22)	8.282
in_a(15)	sum(23)	7.923
in_a(16)	carryout	5.556
in_a(16)	sum(16)	3.936
in_a(16)	sum(17)	4.664
in_a(16)	sum(18)	4.139
in_a(16)	sum(19)	5.027
in_a(16)	sum(20)	5.599
in_a(16)	sum(21)	6.707
in_a(16)	sum(22)	6.800
in_a(16)	sum(23)	6.441
in_a(17)	carryout	5.858
in_a(17)	sum(17)	5.088
in_a(17)	sum(18)	4.441
in_a(17)	sum(19)	5.329
in_a(17)	sum(20)	5.901
in_a(17)	sum(21)	7.009
in_a(17)	sum(22)	7.102
in_a(17)	sum(23)	6.743
in_a(18)	carryout	5.291
in_a(18)	sum(18)	4.223
in_a(18)	sum(19)	4.762
in_a(18)	sum(20)	5.334
in_a(18)	sum(21)	6.442
in_a(18)	sum(22)	6.535
in_a(18)	sum(23)	6.176
in_a(19)	carryout	4.170
in_a(19)	sum(19)	4.576
in_a(19)	sum(20)	4.281
in_a(19)	sum(21)	5.474
in_a(19)	sum(22)	5.567
in_a(19)	sum(23)	5.208
in_a(20)	carryout	6.298
in_a(20)	sum(20)	5.182
in_a(20)	sum(21)	4.900
in_a(20)	sum(22)	6.041
in_a(20)	sum(23)	6.345
in_a(21)	carryout	5.506
in_a(21)	sum(21)	4.491
in_a(21)	sum(22)	5.170
in_a(21)	sum(23)	5.540
in_a(22)	carryout	4.732
in_a(22)	sum(22)	4.812
in_a(22)	sum(23)	4.909
in_a(23)	carryout	3.837
in_a(23)	sum(23)	5.078
in_b(0)	carryout	8.127
in_b(0)	sum(0)	4.142

in_b(0)	sum(1)	4.033
in_b(0)	sum(2)	4.245
in_b(0)	sum(3)	5.008
in_b(0)	sum(4)	5.455
in_b(0)	sum(5)	5.433
in_b(0)	sum(6)	5.658
in_b(0)	sum(7)	5.524
in_b(0)	sum(8)	6.230
in_b(0)	sum(9)	6.260
in_b(0)	sum(10)	6.039
in_b(0)	sum(11)	6.762
in_b(0)	sum(12)	7.038
in_b(0)	sum(13)	7.345
in_b(0)	sum(14)	6.727
in_b(0)	sum(15)	7.115
in_b(0)	sum(16)	7.970
in_b(0)	sum(17)	8.703
in_b(0)	sum(18)	8.126
in_b(0)	sum(19)	7.813
in_b(0)	sum(20)	8.170
in_b(0)	sum(21)	9.278
in_b(0)	sum(22)	9.371
in_b(0)	sum(23)	9.012
in_b(1)	carryout	7.847
in_b(1)	sum(1)	3.664
in_b(1)	sum(2)	3.965
in_b(1)	sum(3)	4.728
in_b(1)	sum(4)	5.175
in_b(1)	sum(5)	5.153
in_b(1)	sum(6)	5.378
in_b(1)	sum(7)	5.244
in_b(1)	sum(8)	5.950
in_b(1)	sum(9)	5.980
in_b(1)	sum(10)	5.759
in_b(1)	sum(11)	6.482
in_b(1)	sum(12)	6.758
in_b(1)	sum(13)	7.065
in_b(1)	sum(14)	6.447
in_b(1)	sum(15)	6.835
in_b(1)	sum(16)	7.690
in_b(1)	sum(17)	8.423
in_b(1)	sum(18)	7.846
in_b(1)	sum(19)	7.533
in_b(1)	sum(20)	7.890
in_b(1)	sum(21)	8.998
in_b(1)	sum(22)	9.091
in_b(1)	sum(23)	8.732
in_b(2)	carryout	7.246
in_b(2)	sum(2)	3.733
in_b(2)	sum(3)	4.133
in_b(2)	sum(4)	4.936
in_b(2)	sum(5)	4.914
in_b(2)	sum(6)	5.139
in_b(2)	sum(7)	4.643
in_b(2)	sum(8)	5.349
in_b(2)	sum(9)	5.379
in_b(2)	sum(10)	5.158
in_b(2)	sum(11)	5.881
in_b(2)	sum(12)	6.157
in_b(2)	sum(13)	6.464
in_b(2)	sum(14)	5.846
in_b(2)	sum(15)	6.234
in_b(2)	sum(16)	7.089
in_b(2)	sum(17)	7.822
in_b(2)	sum(18)	7.245
in_b(2)	sum(19)	6.932
in_b(2)	sum(20)	7.289
in_b(2)	sum(21)	8.397
in_b(2)	sum(22)	8.490
in_b(2)	sum(23)	8.131
in_b(3)	carryout	7.911
in_b(3)	sum(3)	4.794
in_b(3)	sum(4)	4.645
in_b(3)	sum(5)	4.623

in_b(3)	sum(6)	4.848
in_b(3)	sum(7)	5.308
in_b(3)	sum(8)	6.014
in_b(3)	sum(9)	6.044
in_b(3)	sum(10)	5.823
in_b(3)	sum(11)	6.546
in_b(3)	sum(12)	6.822
in_b(3)	sum(13)	7.129
in_b(3)	sum(14)	6.511
in_b(3)	sum(15)	6.899
in_b(3)	sum(16)	7.754
in_b(3)	sum(17)	8.487
in_b(3)	sum(18)	7.910
in_b(3)	sum(19)	7.597
in_b(3)	sum(20)	7.954
in_b(3)	sum(21)	9.062
in_b(3)	sum(22)	9.155
in_b(3)	sum(23)	8.796
in_b(4)	carryout	7.918
in_b(4)	sum(4)	4.130
in_b(4)	sum(5)	4.101
in_b(4)	sum(6)	4.180
in_b(4)	sum(7)	5.315
in_b(4)	sum(8)	6.021
in_b(4)	sum(9)	6.051
in_b(4)	sum(10)	5.830
in_b(4)	sum(11)	6.553
in_b(4)	sum(12)	6.829
in_b(4)	sum(13)	7.136
in_b(4)	sum(14)	6.518
in_b(4)	sum(15)	6.906
in_b(4)	sum(16)	7.761
in_b(4)	sum(17)	8.494
in_b(4)	sum(18)	7.917
in_b(4)	sum(19)	7.604
in_b(4)	sum(20)	7.961
in_b(4)	sum(21)	9.069
in_b(4)	sum(22)	9.162
in_b(4)	sum(23)	8.803
in_b(5)	carryout	7.994
in_b(5)	sum(5)	4.795
in_b(5)	sum(6)	4.026
in_b(5)	sum(7)	5.391
in_b(5)	sum(8)	6.097
in_b(5)	sum(9)	6.127
in_b(5)	sum(10)	5.906
in_b(5)	sum(11)	6.629
in_b(5)	sum(12)	6.905
in_b(5)	sum(13)	7.212
in_b(5)	sum(14)	6.594
in_b(5)	sum(15)	6.982
in_b(5)	sum(16)	7.837
in_b(5)	sum(17)	8.570
in_b(5)	sum(18)	7.993
in_b(5)	sum(19)	7.680
in_b(5)	sum(20)	8.037
in_b(5)	sum(21)	9.145
in_b(5)	sum(22)	9.238
in_b(5)	sum(23)	8.879
in_b(6)	carryout	8.183
in_b(6)	sum(6)	4.636
in_b(6)	sum(7)	5.580
in_b(6)	sum(8)	6.286
in_b(6)	sum(9)	6.316
in_b(6)	sum(10)	6.095
in_b(6)	sum(11)	6.818
in_b(6)	sum(12)	7.094
in_b(6)	sum(13)	7.401
in_b(6)	sum(14)	6.783
in_b(6)	sum(15)	7.171
in_b(6)	sum(16)	8.026
in_b(6)	sum(17)	8.759
in_b(6)	sum(18)	8.182
in_b(6)	sum(19)	7.869

in_b(6)	sum(20)	8.226
in_b(6)	sum(21)	9.334
in_b(6)	sum(22)	9.427
in_b(6)	sum(23)	9.068
in_b(7)	carryout	6.978
in_b(7)	sum(7)	4.075
in_b(7)	sum(8)	4.777
in_b(7)	sum(9)	5.068
in_b(7)	sum(10)	4.969
in_b(7)	sum(11)	5.613
in_b(7)	sum(12)	5.889
in_b(7)	sum(13)	6.196
in_b(7)	sum(14)	5.578
in_b(7)	sum(15)	5.966
in_b(7)	sum(16)	6.821
in_b(7)	sum(17)	7.554
in_b(7)	sum(18)	6.977
in_b(7)	sum(19)	6.664
in_b(7)	sum(20)	7.021
in_b(7)	sum(21)	8.129
in_b(7)	sum(22)	8.222
in_b(7)	sum(23)	7.863
in_b(8)	carryout	7.478
in_b(8)	sum(8)	5.396
in_b(8)	sum(9)	4.957
in_b(8)	sum(10)	5.008
in_b(8)	sum(11)	6.113
in_b(8)	sum(12)	6.389
in_b(8)	sum(13)	6.696
in_b(8)	sum(14)	6.078
in_b(8)	sum(15)	6.466
in_b(8)	sum(16)	7.321
in_b(8)	sum(17)	8.054
in_b(8)	sum(18)	7.477
in_b(8)	sum(19)	7.164
in_b(8)	sum(20)	7.521
in_b(8)	sum(21)	8.629
in_b(8)	sum(22)	8.722
in_b(8)	sum(23)	8.363
in_b(9)	carryout	7.302
in_b(9)	sum(9)	4.694
in_b(9)	sum(10)	4.832
in_b(9)	sum(11)	5.937
in_b(9)	sum(12)	6.213
in_b(9)	sum(13)	6.520
in_b(9)	sum(14)	5.902
in_b(9)	sum(15)	6.290
in_b(9)	sum(16)	7.145
in_b(9)	sum(17)	7.878
in_b(9)	sum(18)	7.301
in_b(9)	sum(19)	6.988
in_b(9)	sum(20)	7.345
in_b(9)	sum(21)	8.453
in_b(9)	sum(22)	8.546
in_b(9)	sum(23)	8.187
in_b(10)	carryout	7.051
in_b(10)	sum(10)	4.065
in_b(10)	sum(11)	5.686
in_b(10)	sum(12)	5.962
in_b(10)	sum(13)	6.269
in_b(10)	sum(14)	5.651
in_b(10)	sum(15)	6.039
in_b(10)	sum(16)	6.894
in_b(10)	sum(17)	7.627
in_b(10)	sum(18)	7.050
in_b(10)	sum(19)	6.737
in_b(10)	sum(20)	7.094
in_b(10)	sum(21)	8.202
in_b(10)	sum(22)	8.295
in_b(10)	sum(23)	7.936
in_b(11)	carryout	5.568
in_b(11)	sum(11)	4.038
in_b(11)	sum(12)	4.479
in_b(11)	sum(13)	4.786

in_b(11)	sum(14)	4.168
in_b(11)	sum(15)	4.556
in_b(11)	sum(16)	5.411
in_b(11)	sum(17)	6.144
in_b(11)	sum(18)	5.567
in_b(11)	sum(19)	5.254
in_b(11)	sum(20)	5.611
in_b(11)	sum(21)	6.719
in_b(11)	sum(22)	6.812
in_b(11)	sum(23)	6.453
in_b(12)	carryout	6.685
in_b(12)	sum(12)	4.207
in_b(12)	sum(13)	4.721
in_b(12)	sum(14)	5.034
in_b(12)	sum(15)	5.266
in_b(12)	sum(16)	6.528
in_b(12)	sum(17)	7.261
in_b(12)	sum(18)	6.684
in_b(12)	sum(19)	6.371
in_b(12)	sum(20)	6.728
in_b(12)	sum(21)	7.836
in_b(12)	sum(22)	7.929
in_b(12)	sum(23)	7.570
in_b(13)	carryout	6.511
in_b(13)	sum(13)	5.082
in_b(13)	sum(14)	4.859
in_b(13)	sum(15)	5.101
in_b(13)	sum(16)	6.354
in_b(13)	sum(17)	7.087
in_b(13)	sum(18)	6.510
in_b(13)	sum(19)	6.197
in_b(13)	sum(20)	6.554
in_b(13)	sum(21)	7.662
in_b(13)	sum(22)	7.755
in_b(13)	sum(23)	7.396
in_b(14)	carryout	6.082
in_b(14)	sum(14)	4.649
in_b(14)	sum(15)	4.652
in_b(14)	sum(16)	5.925
in_b(14)	sum(17)	6.658
in_b(14)	sum(18)	6.081
in_b(14)	sum(19)	5.768
in_b(14)	sum(20)	6.125
in_b(14)	sum(21)	7.233
in_b(14)	sum(22)	7.326
in_b(14)	sum(23)	6.967
in_b(15)	carryout	6.397
in_b(15)	sum(15)	4.567
in_b(15)	sum(16)	4.438
in_b(15)	sum(17)	5.171
in_b(15)	sum(18)	4.594
in_b(15)	sum(19)	4.281
in_b(15)	sum(20)	6.440
in_b(15)	sum(21)	7.548
in_b(15)	sum(22)	7.641
in_b(15)	sum(23)	7.282
in_b(16)	carryout	6.788
in_b(16)	sum(16)	4.447
in_b(16)	sum(17)	5.186
in_b(16)	sum(18)	5.371
in_b(16)	sum(19)	6.259
in_b(16)	sum(20)	6.831
in_b(16)	sum(21)	7.939
in_b(16)	sum(22)	8.032
in_b(16)	sum(23)	7.673
in_b(17)	carryout	5.548
in_b(17)	sum(17)	4.931
in_b(17)	sum(18)	4.131
in_b(17)	sum(19)	5.019
in_b(17)	sum(20)	5.591
in_b(17)	sum(21)	6.699
in_b(17)	sum(22)	6.792
in_b(17)	sum(23)	6.433
in_b(18)	carryout	5.807

in_b(18)	sum(18)	4.444
in_b(18)	sum(19)	5.278
in_b(18)	sum(20)	5.850
in_b(18)	sum(21)	6.958
in_b(18)	sum(22)	7.051
in_b(18)	sum(23)	6.692
in_b(19)	carryout	4.534
in_b(19)	sum(19)	4.482
in_b(19)	sum(20)	4.405
in_b(19)	sum(21)	5.852
in_b(19)	sum(22)	5.945
in_b(19)	sum(23)	5.586
in_b(20)	carryout	6.296
in_b(20)	sum(20)	5.044
in_b(20)	sum(21)	4.457
in_b(20)	sum(22)	5.955
in_b(20)	sum(23)	6.330
in_b(21)	carryout	5.867
in_b(21)	sum(21)	5.138
in_b(21)	sum(22)	5.629
in_b(21)	sum(23)	5.933
in_b(22)	carryout	4.975
in_b(22)	sum(22)	5.328
in_b(22)	sum(23)	5.425
in_b(23)	carryout	3.720
in_b(23)	sum(23)	4.961

Analysis completed Mon Dec 6 12:08:32 2021

Trace Settings:

Trace Settings

Peak Memory Usage: 273 MB

Timing Analysis Report for Non-pipelined Implementation with Negation Output

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

/nfs/sw_cmc/x86_64.EL7/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce
nonpipeline_circuit_8b_nega_out_out nonpipeline_circuit_8b_nega_out.pcf -v 10
-o nonpipeline_circuit_8b_nega_out_out

Design file: nonpipeline_circuit_8b_nega_out_out.ncd
Physical constraint file: nonpipeline_circuit_8b_nega_out.pcf
Device, package, speed: xc2vp20,ff896,-7 (PRODUCTION 1.94 2008-01-09)
Report level: verbose report, limited to 10 items per constraint

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)

Setup/Hold to clock load

+	+	+	+	+	+
Setup to Hold to			Clock		

Source	clk (edge)	clk (edge)	Internal Clock(s)	Phase	
a(0)	-0.199(F)	0.963(F) load_int		0.000	
a(1)	0.244(F)	0.600(F) load_int		0.000	
a(2)	-0.148(F)	0.923(F) load_int		0.000	
a(3)	0.072(F)	0.744(F) load_int		0.000	
a(4)	-0.231(F)	0.997(F) load_int		0.000	
a(5)	-0.288(F)	1.029(F) load_int		0.000	
a(6)	-0.378(F)	1.108(F) load_int		0.000	
a(7)	-0.075(F)	0.858(F) load_int		0.000	
b(0)	-0.125(F)	0.885(F) load_int		0.000	
b(1)	0.514(F)	0.372(F) load_int		0.000	
b(2)	0.110(F)	0.709(F) load_int		0.000	
b(3)	0.849(F)	0.119(F) load_int		0.000	
b(4)	-0.135(F)	0.905(F) load_int		0.000	
b(5)	-0.296(F)	1.029(F) load_int		0.000	
b(6)	0.420(F)	0.457(F) load_int		0.000	
b(7)	-0.136(F)	0.903(F) load_int		0.000	
clr	2.809(F)	-1.406(F) load_int		0.000	

Clock clk to Pad

Destination	clk (edge)	Internal Clock(s)	Clock		Phase	
end_flag	4.842(F) clk_int		0.000			
z(0)	4.891(F) clk_int		0.000			
z(1)	5.199(F) clk_int		0.000			
z(2)	4.851(F) clk_int		0.000			
z(3)	4.792(F) clk_int		0.000			
z(4)	5.502(F) clk_int		0.000			
z(5)	4.641(F) clk_int		0.000			
z(6)	6.244(F) clk_int		0.000			
z(7)	5.292(F) clk_int		0.000			
z(8)	4.836(F) clk_int		0.000			
z(9)	5.035(F) clk_int		0.000			
z(10)	4.842(F) clk_int		0.000			
z(11)	4.914(F) clk_int		0.000			
z(12)	5.070(F) clk_int		0.000			
z(13)	4.663(F) clk_int		0.000			
z(14)	5.238(F) clk_int		0.000			
z(15)	5.274(F) clk_int		0.000			

Clock to Setup on destination clock clk

Source Clock	Src : Rise	Src : Fall	Src : Rise	Src : Fall	
	Dest : Rise	Dest : Rise	Dest : Fall	Dest : Fall	
load					31.846

Clock to Setup on destination clock load

Source Clock	Src : Rise	Src : Fall	Src : Rise	Src : Fall	
	Dest : Rise	Dest : Rise	Dest : Fall	Dest : Fall	
clk					1.854

Analysis completed Mon Dec 6 09:58:52 2021

Trace Settings:

Trace Settings

Peak Memory Usage: 283 MB

Timing Analysis Report for Pipelined Implementation with Negation Output

```
/nfs/sw_cmc/x86_64.EL7/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce
pipelining_circuit_8b_nega_out_out pipelining_circuit_8b_nega_out.pcf -v 10 -o
pipelining_circuit_8b_nega_out_out
```

Design file: pipelining_circuit_8b_nega_out_out.ncd
 Physical constraint file: pipelining_circuit_8b_nega_out.pcf
 Device, package, speed: xc2vp20,ff896,-7 (PRODUCTION 1.94 2008-01-09)
 Report level: verbose report, limited to 10 items per constraint

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)

Setup/Hold to clock load

Source	Setup to clk (edge)	Hold to clk (edge)	Internal Clock(s)	Clock	Phase
a(0)	-0.344(F)	1.074(F) load_int		0.000	
a(1)	-0.305(F)	1.042(F) load_int		0.000	
a(2)	-0.146(F)	0.923(F) load_int		0.000	
a(3)	-0.334(F)	1.073(F) load_int		0.000	
a(4)	-0.383(F)	1.112(F) load_int		0.000	
a(5)	-0.285(F)	1.026(F) load_int		0.000	
a(6)	-0.349(F)	1.078(F) load_int		0.000	
a(7)	-0.341(F)	1.079(F) load_int		0.000	
b(0)	-0.071(F)	0.841(F) load_int		0.000	
b(1)	-0.297(F)	1.029(F) load_int		0.000	
b(2)	-0.372(F)	1.099(F) load_int		0.000	
b(3)	-0.284(F)	1.020(F) load_int		0.000	
b(4)	-0.395(F)	1.124(F) load_int		0.000	
b(5)	-0.345(F)	1.081(F) load_int		0.000	
b(6)	-0.349(F)	1.072(F) load_int		0.000	
b(7)	-0.269(F)	1.001(F) load_int		0.000	
clr	3.698(F)	-1.760(F) load_int		0.000	

Clock clk to Pad

Destination	clk (edge) to PAD	Internal Clock(s)	Clock	Phase
end_flag	5.473(F) clk_int		0.000	
z(0)	5.654(F) clk_int		0.000	
z(1)	5.613(F) clk_int		0.000	
z(2)	5.788(F) clk_int		0.000	
z(3)	5.461(F) clk_int		0.000	
z(4)	5.487(F) clk_int		0.000	
z(5)	5.551(F) clk_int		0.000	
z(6)	6.076(F) clk_int		0.000	
z(7)	5.764(F) clk_int		0.000	
z(8)	5.613(F) clk_int		0.000	
z(9)	5.722(F) clk_int		0.000	
z(10)	5.633(F) clk_int		0.000	
z(11)	5.485(F) clk_int		0.000	
z(12)	5.587(F) clk_int		0.000	
z(13)	5.650(F) clk_int		0.000	
z(14)	5.599(F) clk_int		0.000	
z(15)	5.627(F) clk_int		0.000	

Clock to Setup on destination clock clk							
Source Clock	Src:Rise	Src:Fall	Dest:Rise	Dest:Fall	Dest:Fall		
clk						2.3021	
load						25.6791	

Clock to Setup on destination clock load							
Source Clock	Src:Rise	Src:Fall	Dest:Rise	Dest:Fall	Dest:Fall		
clk						2.0901	
load							

Analysis completed Mon Dec 6 09:45:17 2021

Trace Settings:

Trace Settings

Peak Memory Usage: 284 MB

Timing Analysis Report for Pipelined Implementation with Separation Output

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

/nfs/sw_cmc/x86_64.EL7/tools/xilinx_10.1/ISE/bin/lin64/unwrapped/trce
pipelining_circuit_8b_sep_out_out pipelining_circuit_8b_sep_out.pcf -v 10 -o
pipelining_circuit_8b_sep_out_out

Design file: pipelining_circuit_8b_sep_out_out.ncd
Physical constraint file: pipelining_circuit_8b_sep_out.pcf
Device, package, speed: xc2vp20,ff896,-7 (PRODUCTION 1.94 2008-01-09)
Report level: verbose report, limited to 10 items per constraint

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)

Setup/Hold to clock load

Source	Setup to	Hold to	Internal Clock(s)	Clock	Phase
	clk (edge)	clk (edge)			
a(0)	-0.343(F)	1.080(F) load_int		0.0001	
a(1)	0.103(F)	0.727(F) load_int		0.0001	
a(2)	-0.102(F)	0.872(F) load_int		0.0001	
a(3)	-0.009(F)	0.796(F) load_int		0.0001	
a(4)	-0.370(F)	1.106(F) load_int		0.0001	
a(5)	-0.332(F)	1.075(F) load_int		0.0001	
a(6)	-0.344(F)	1.080(F) load_int		0.0001	
a(7)	0.132(F)	0.693(F) load_int		0.0001	

b(0)	0.422(F)	0.447(F) load_int	0.000
b(1)	0.493(F)	0.395(F) load_int	0.000
b(2)	0.212(F)	0.631(F) load_int	0.000
b(3)	0.366(F)	0.519(F) load_int	0.000
b(4)	0.624(F)	0.300(F) load_int	0.000
b(5)	0.244(F)	0.611(F) load_int	0.000
b(6)	-0.071(F)	0.858(F) load_int	0.000
b(7)	0.514(F)	0.386(F) load_int	0.000
clr	3.319(F)	-1.655(F) load_int	0.000

Clock clk to Pad

Destination	clk (edge)	Internal Clock(s)	Clock	Phase
end_flag	6.489(F) clk_int		0.000	
z(0)	7.015(F) clk_int		0.000	
z(1)	6.694(F) clk_int		0.000	
z(2)	6.691(F) clk_int		0.000	
z(3)	6.704(F) clk_int		0.000	
z(4)	6.424(F) clk_int		0.000	
z(5)	6.841(F) clk_int		0.000	
z(6)	7.539(F) clk_int		0.000	
z(7)	7.166(F) clk_int		0.000	
z(8)	6.438(F) clk_int		0.000	
z(9)	7.024(F) clk_int		0.000	
z(10)	7.017(F) clk_int		0.000	
z(11)	6.599(F) clk_int		0.000	
z(12)	6.321(F) clk_int		0.000	
z(13)	7.068(F) clk_int		0.000	
z(14)	6.231(F) clk_int		0.000	
z(15)	5.675(F) clk_int		0.000	

Clock to Setup on destination clock clk

Source Clock	Src : Rise	Src : Fall	Src : Rise	Src : Fall	
	Dest : Rise	Dest : Rise	Dest : Fall	Dest : Fall	
clk					2.142
load					27.165

Clock to Setup on destination clock load

Source Clock	Src : Rise	Src : Fall	Src : Rise	Src : Fall	
	Dest : Rise	Dest : Rise	Dest : Fall	Dest : Fall	
clk					1.596

Pad to Pad

Source Pad	Destination Pad	Delay
clr	z(0)	6.115
clr	z(1)	5.384
clr	z(2)	5.798
clr	z(3)	6.013
clr	z(4)	6.151
clr	z(5)	6.563
clr	z(6)	5.968
clr	z(7)	5.854
clr	z(8)	5.022
clr	z(9)	5.889
clr	z(10)	5.727
clr	z(11)	6.330
clr	z(12)	5.039
clr	z(13)	6.379
clr	z(14)	4.655
clr	z(15)	4.769

Analysis completed Mon Dec 6 11:49:22 2021

Trace Settings:

Trace Settings

Peak Memory Usage: 284 MB

Figure 47
Synthesized RTL Diagram of 16-bit Radix-4 Booth Multiplier

