COEN 6501 Project Fall 2021 Specification

Jun Huang, Dawei Zuo, Yuelin Yao, and 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 Conetnts

Introduction **6**

Heading II . . . 6

Heading III . . . 6

Project Requirement **7**

Heading II . . . 7

Heading III . . . 7

Overview of the Design **8**

Heading II . . . 8

Heading III . . . 8

Multiplication in Three Operands **9**

Radix-4 Booth Algorithm Logic in Details . . . 9

Example in Signed Number Multiplication . . . 9

Example in Unsigned Number Multiplication . . . 10

8-bit Radix-4 Booth Multiplier Circuit Implementation . . . 11

Overall Circuit Design and RTL Description . . . 11

Blocks Design . . . 11

8-bit Triple Operands Multiplier Circuit Implementation . . . 15

16-bit Triple Operands Multiplier Circuit Implementation . . . 17

Carry Select Adder **18**

Heading II . . . 18

Heading III . . . 18

Overflow Handling **19**

Heading II . . . 19

Heading III . . . 19

Non-pipelining Implementation **20**

Heading II . . . 20

Heading III . . . 20

Pipelining Implementation **21**

Heading II 21

Heading III 21

Synthesis and Analysis of the Arithmetic Circuit **22**

Heading II 22

Heading III 22

References **23**

List of Figures

1 The Decoded Code Blocks of the Signed Multiplier 10
2 Process of the Algorithm for Signed Number 10
3 The Decoded Code Blocks of the Unsigned Multiplier 11
4 Process of the Slgorithm for Unisgned Number 11
5 Synthesized RTL Diagram of 8-bit Radix-4 Booth Multiplier 12
6 Synthesized RTL Diagram of Complement Generator 12
7 Simulation Wave Diagram of Complement Generator 13
8 Synthesized RTL Diagram of Booth Stage 0 Block 13
9 Simulation Wave Diagram of Booth Stage 0 Block 14
10 Synthesized RTL Diagram of Booth Stage 1 Block 14
11 Simulation Wave Diagram of Booth Stage 1 Block 15
12 Synthesized RTL Diagram of Triple 8-bit Operands Multiplier Circuit 16
13 Simulation Wave Diagram of Triple 8-bit Operands Multiplier Circuit 16
14 Synthesized RTL Diagram of Triple 16-bit Operands Multiplier Circuit 17
15 Simulation Wave Diagram of Triple 16-bit Operands Multiplier Circuit 17
16 Synthesized RTL Diagram of 16-bit Radix-4 Booth Multiplier 24

List of Tables

1 Code Table of Radix-4 Booth Algorithm 9

Introduction

Cervantes Saavedra, Raffel, and Wilson (1999) En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor.

Heading II

Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lantejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda.

Heading III

El resto della concluían sayo de velarte, calzas de velludo para las fiestas, con sus pantuflos de lo mesmo, y los días de entresemana se honraba con su vellorí de lo más fino.

Heading IV. Tenía en su casa una ama que pasaba de los cuarenta, y una sobrina que no llegaba a los veinte, y un mozo de campo y plaza, que así ensillaba el rocín como tomaba la podadera. Frisaba la edad de nuestro hidalgo con los cincuenta años; era de complexión recia, seco de carnes, enjuto de rostro, gran madrugador y amigo de la caza.

Heading V. Quieren decir que tenía el sobrenombre de Quijada, o Quesada, que en esto hay alguna diferencia en los autores que deste caso escriben; aunque por conjeturas verosímiles se deja entender que se llamaba Quijana.

Project Requirement

Heading II

Heading III

Heading IV.

Heading V.

Overview of the Design

Heading II

Heading III

Heading IV.

Heading V.

Multiplication in Three Operands

The very first component of the ALU should be the circuit that calculating the result of $A^2 * B$. Hence the multiplication of two 8-bit operands circuit 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 design for the sighed number. In this case, the implemented algorithm requires 2/n + 1 partial product for 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 1 is shown. Then all partial products can be derived from the code blocks and the "Code Table" which is presented in Table 1.

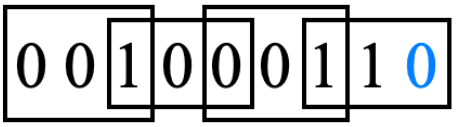
Table 1

Code Table of Radix-4 Booth Algorithm

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

Figure 1

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 with shifting two bits more than the previous product. Then perform the addition. The process of the algorithm is described in Figure 2.

Figure 2

Process of the Algorithm for Signed Number

110	1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0	partial product 1
001	0 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	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 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.

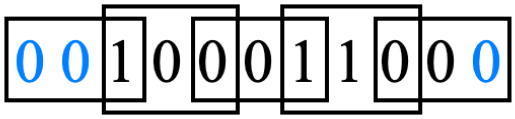
After the performance, the result *0b0000101111000010* is 3010 in decimal which is the product of 86 and 35. As can be observed at the table, subtraction can be done by adding the negation of the number which is represented by 2's complement.

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 into 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 4 and Figure 4 are shown.

Figure 3

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 4

Process of the Slgorithm for Unisgned Number

000	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 1 1 0 1 0 1 1 0 0	partial product 2
001	0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0	partial product 3
100	1 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 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 previous discussion, the booth multiplier component should contain the following blocks: 1. A 9-bit complementor for the negation of the multiplicand; 2. Five booth stage units for 5 partial products; 3. Four 16-bit adders for sum up the partial products. Figure 5 presents the synthesized RTL diagram of the multiplier circuit.

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

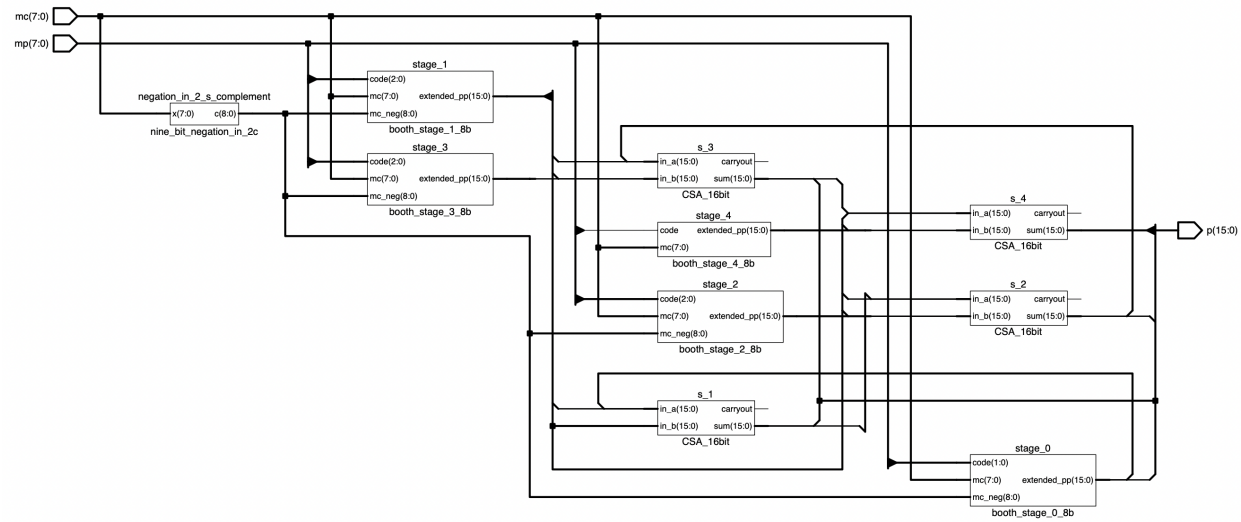
Blocks Design

This section will discuss the design of the complementor 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 6. The logic of the generator that uses a 2-to-1 mux is straightforward as Expression (1)

Figure 5

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

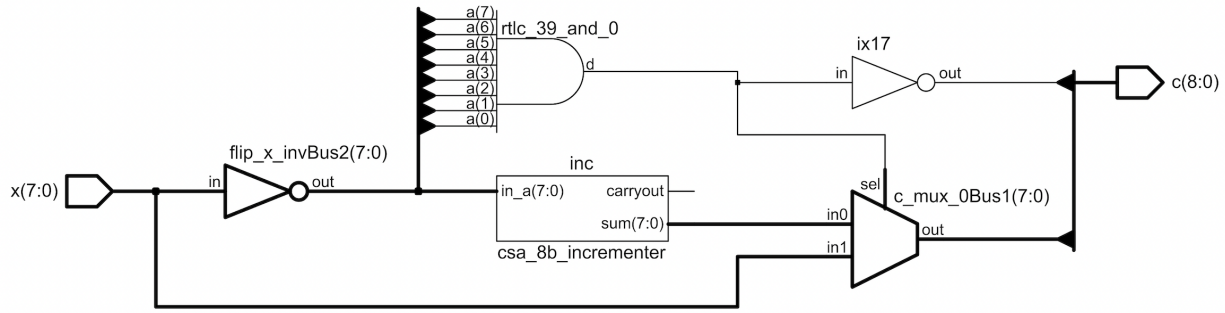


described. The simulation result is shown in Figure 7.

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

Figure 6

Synthesized RTL Diagram of Complement Generator



Booth Stage 0. Figure 8 and Expression (2) 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 right most 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 9.

Figure 7

Simulation Wave Diagram of Complement Generator

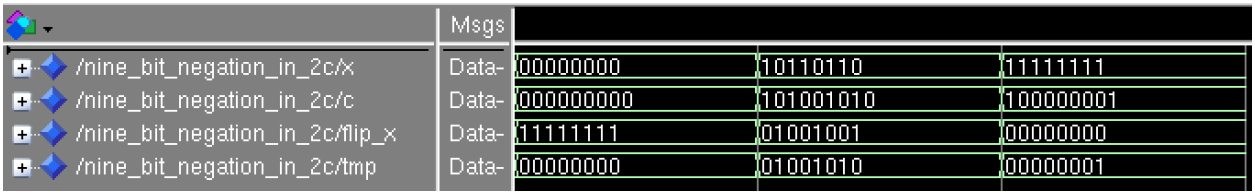
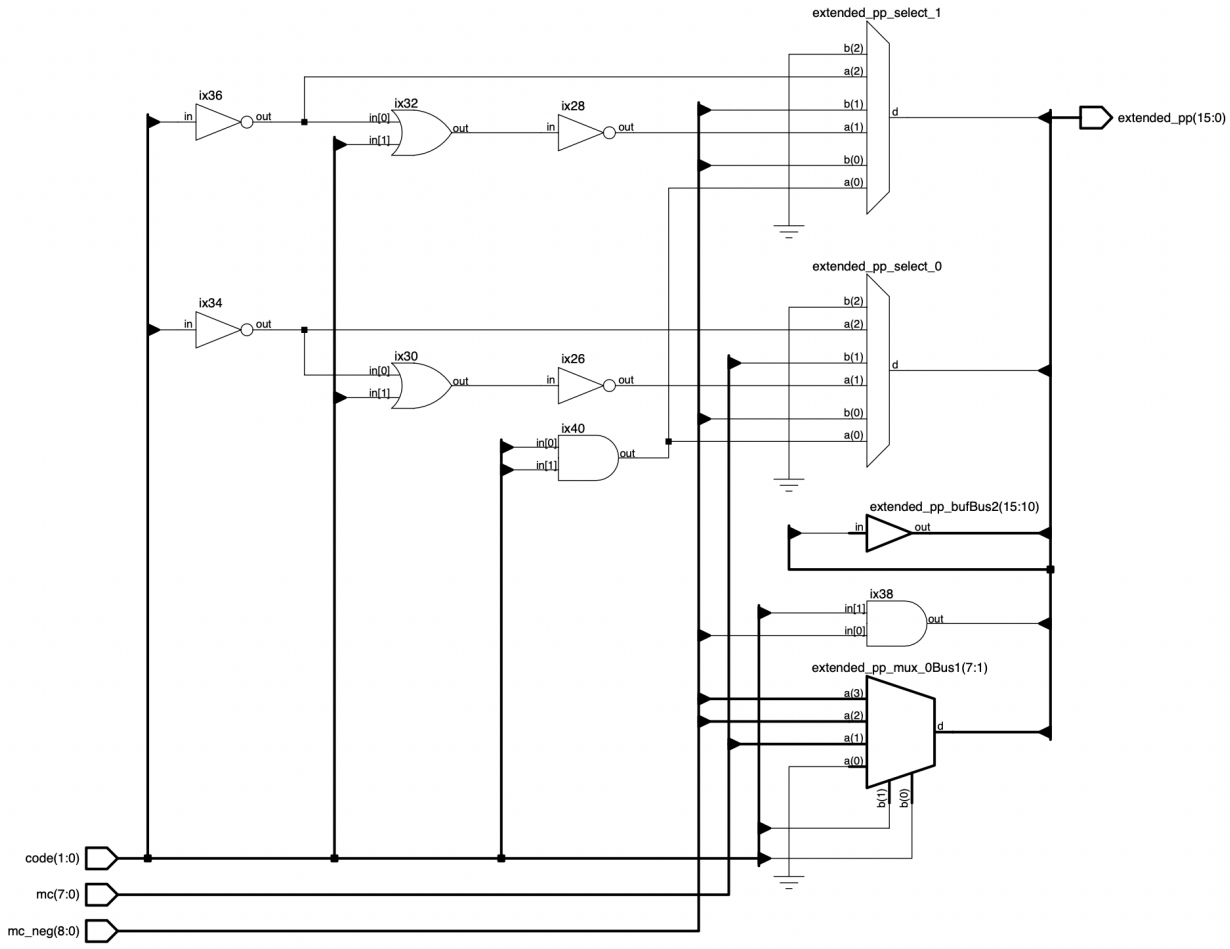


Figure 8

Synthesized RTL Diagram of Booth Stage 0 Block



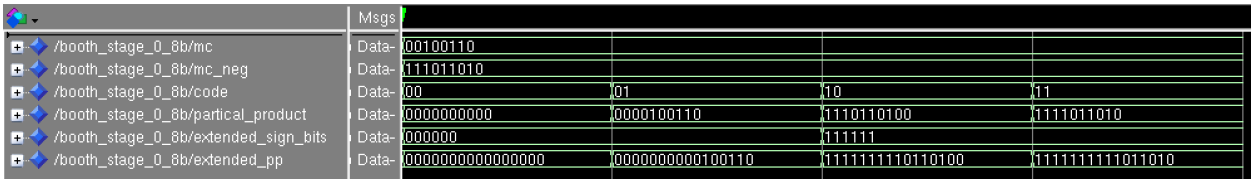
$$partial_product = \begin{cases} 0000000000, & \text{if } code = 00 \\ concat(00, mc), & \text{if } code = 01 \\ concat(mc_neg, 0), & \text{if } code = 10 \\ concat(mc_neg(8), mc_neg), & \text{else } code = others \end{cases} \tag{2}$$

$$extended_sign_bits = (others => partical_product(9))$$

$$extended_pp = concat(extended_sign_bits, partial_product)$$

Figure 9

Simulation Wave Diagram of Booth Stage 0 Block

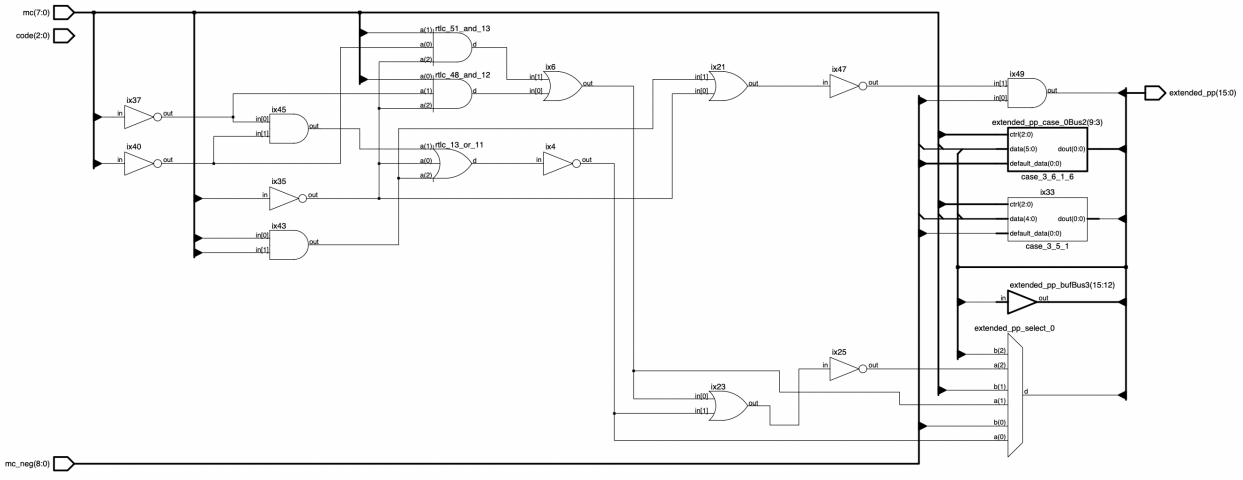


Note. Four code values were provided to simulate the *extended_pp*.

Booth Stage 1. Figure 10 and Expression (3) 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 11.

Figure 10

Synthesized RTL Diagram of Booth Stage 1 Block



$$partial_product = \begin{cases} 0000000000, & \text{if } code = 000|111 \\ concat(00, mc), & \text{if } code = 001|010 \\ concat(0, mc, 0), & \text{if } code = 011 \\ concat(mc_neg, 0), & \text{if } code = 100 \\ concat(mc_neg(8), mc_neg), & \text{else } code = others \end{cases} \tag{3}$$

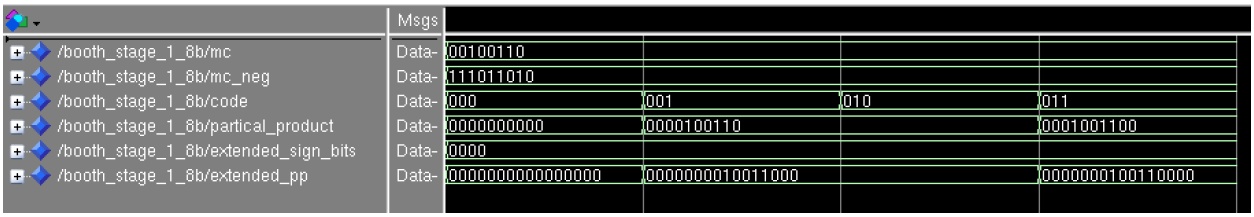
$$extended_sign_bits = (others => partical_product(9))$$

$$extended_pp = concat(extended_sign_bits, partial_product, 00)$$

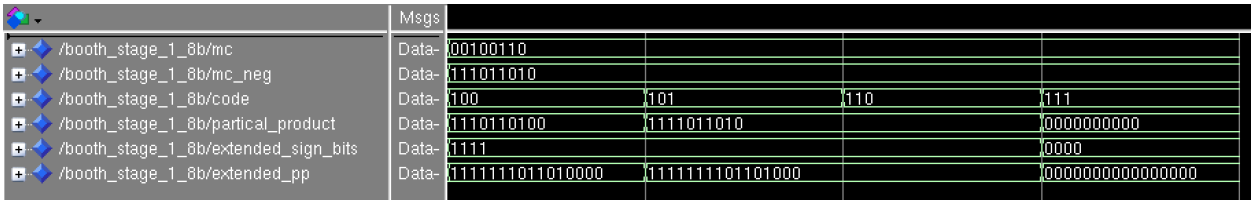
Figure 11

Simulation Wave Diagram of Booth Stage 1 Block

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



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



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 booth stage 4 block, it only takes the MSB from the operand multiplier. Expression (4) shows its logic.

$$extended_pp = \begin{cases} 0000000000000000, & \text{if } code = 0 \\ concat(mc, 00000000), & \text{else } code = others \end{cases} \tag{4}$$

8-bit Triple Operands Multiplier Circuit Implementation

Once the 16-bit product of A^2 which 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 a 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 number together will be the result of $A^2 * B$.

Figure 12 presents the RTL description of the circuit and Figure 13 presents the simulation result. Expression (5) shows the arithmetic process of the $A^2 * B$.

$$\begin{aligned} aa &= a * a \\ laa_b &= aa[7,0] * b \\ haa_b &= aa[15,8] * b \\ aab_m &= concat(haa_b, 00000000) \\ aab_l &= concat(00000000, laa_b) \\ p &= aab_m + aab_l \end{aligned} \tag{5}$$

Figure 12

Synthesized RTL Diagram of Triple 8-bit Operands Multiplier Circuit

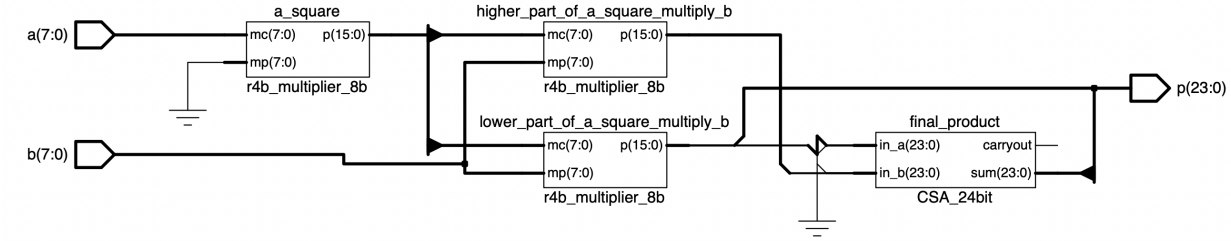
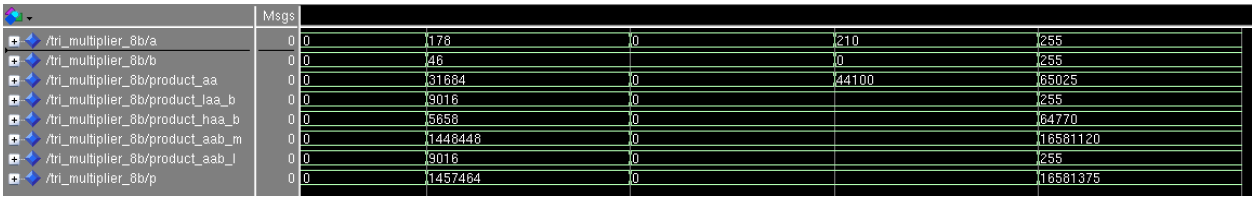


Figure 13

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. 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 bulid every block from scratch. **This is one of the drawbacks of the booth algorithm: lack of expandability.**

Figure 14 presents the RTL description of the triple 16-bit operands multiplier circuit and Figure 15 presents the simulation result. Figure 16 presents the RTL description of the 16-bit booth multiplier circuit.

Figure 14

Synthesized RTL Diagram of Triple 16-bit Operands Multiplier Circuit

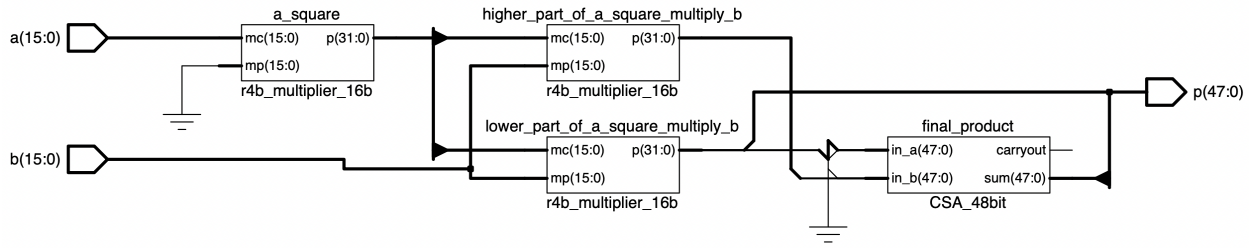
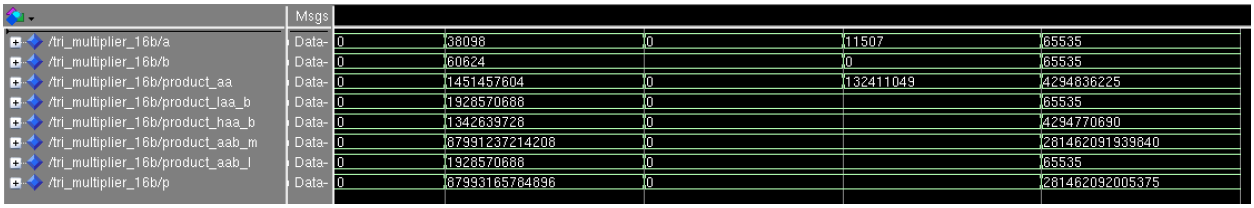


Figure 15

Simulation Wave Diagram of Triple 16-bit Operands Multiplier Circuit



Carry Select Adder

Heading II

Heading III

Heading IV.

Heading V.

Overflow Handling

Heading II

Heading III

Heading IV.

Heading V.

Non-pipelining Implementation

Heading II

Heading III

Heading IV.

Heading V.

Pipelining Implementation

Heading II

Heading III

Heading IV.

Heading V.

Synthesis and Analysis of the Arithmetic Circuit

Heading II

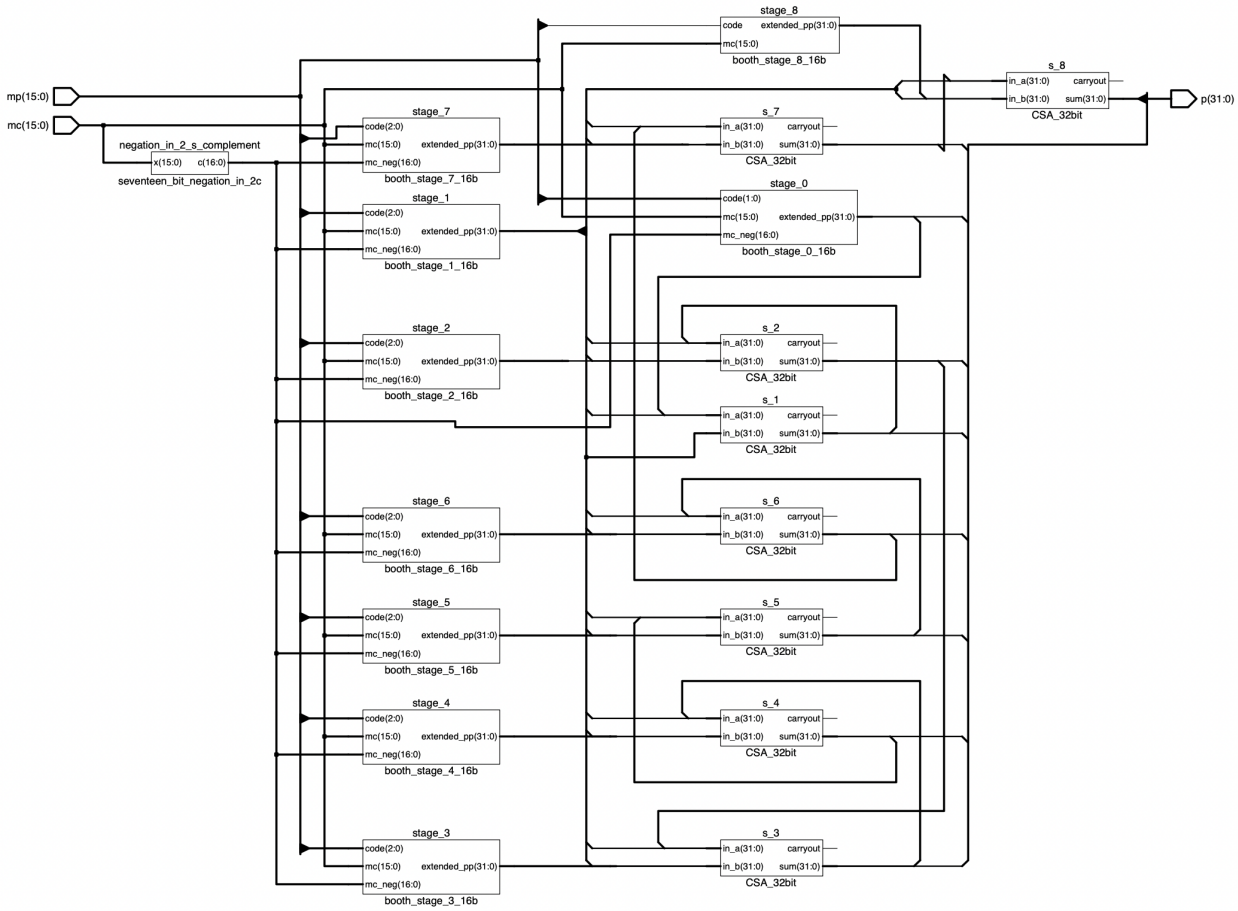
Heading III

Heading IV.

Heading V.

References

Cervantes Saavedra, M. d., Raffel, B., & Wilson, D. d. A. (1999). *Don Quijote: a new translation, backgrounds and contexts, criticism* (1st ed ed.). New York: W.W. Norton.

Figure 16

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