

CERTIFICATION OF APPROVAL

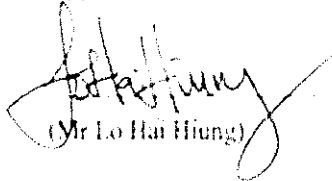
Performance Comparison Of Radix-Based Multiplier Designs

by

Kelly Liew Suet Swee

A project dissertation submitted to the
Electrical and Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL AND ELECTRONICS ENGINEERING)

Approved by,



(Mr. Lo Hai Hiong)

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
January 2012

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



KELLY LIEW SUET SWEE

ABSTRACT

Fast multiplication is used to replace the conventional multiplier to increase the performance and efficiency of the multiplier since multipliers are becoming more important in Digital Signal Processing. Multipliers designed in this project were the Radix-based Multiplier inclusive of Radix-2, Radix-4, Radix-8, Radix-16 and Radix-32 Booth Encoding multipliers. These Radix-based multipliers are able to increase the compression time, contribute to a great savings in silicon area and also the number of stages to be added that is known as speed. The speed and the partial products in these Radix-based multipliers reduced significantly compared to the common addition and shift multiplication.

In this Final Year Project, the Radix-based Booth Encoding multipliers were designed, logic simulation was conducted and logic synthesizer was performed to obtain the area and timing. The relative performance of each multiplier was compared to determine the suitable type of Digital Signal Processing applications in terms of its speed and area performances.

The Project began by defining the problem statement and identifying the objectives and outcomes of the project. Next, the Radix-based multipliers were designed using Verilog Hardware Description Language. It was then logic simulated using Modelsim, simulation software produced by Mentor Graphics to verify the multiplier designs created. Then, the designs were synthesized in Leonardo Spectrum to obtain the performance parameters such as area and timing of the Radix-based multipliers. The synthesis process was done by synthesizing it in TSMC 0.35-microns ASIC standard cell library.

An analysis of the performance obtained were then compared in order to determine which type of Radix-based multipliers give better results in different aspects of performances. The performance of Radix-based designs will then be compared to the five other multiplier designs performances created previously by Chris Lee inclusive of Array, Wallace, Dadda and Reduced-Area multipliers. The Project ended with a conclusion and recommendations.

ACKNOWLEDGEMENT

First of all I would like to thank my UTP Supervisor for my Final Year Project, Mr Lo Hai Hiung for all the guidance and advices throughout the duration of completing my Final Year Project. Without his encouragements and motivations, I would not be able to complete my project on time. His high demands and expectations drove me into producing the outcome and fulfilling the objectives of my project.

Secondly, I would like show my warmest gratitude to Edmond Ang from Emerald System Design located in Penang for setting up a special workplace for me in the company in order for me to do my logic synthesis using Leonardo Spectrum software. Edmond was very helpful in assisting me throughout the duration that I was at Emerald System Design. He never failed to lend me a helping hand whenever I needed guidance.

Besides, I would also like to acknowledge Dr Zuhairi and the Final Year Project coordinators who were very cooperative and patience whenever I go and see them for any advices or problem encountered. Without them, the submission and application process would not have run smoothly.

Here, I would also want to express a special thanks my university, Universiti Teknologi Petronas for introducing Final Year Project I and Final Year Project II for two semesters which had taught me a lot ranging from technical knowledge, soft-skills and also communication skills. With the skills learned in the university, hard work and self-determination, I managed to overcome the challenges and it was truly a bitter sweet memory.

TABLE OF CONTENTS

CERTIFICATION	i
ABSTRACT.....	iii
ACKNOWLEDGEMENT	iv
CHAPTER 1: INTRODUCTION	
1.1 Background of Study	1
1.1.1 Booth Encoding Multiplier	1
1.1.2 Radix-2 Booth Encoding Multiplier	3
1.1.3 Radix-4 Booth Encoding Multiplier	5
1.1.4 Radix-8 Booth Encoding Multiplier	7
1.1.5 Radix-16 Booth Encoding Multiplier	8
1.1.6 Radix-32 Booth Encoding Multiplier	10
1.1.7 Array Multiplier	12
1.1.8 Wallace Multiplier	12
1.1.9 Dadda Multiplier	14
1.1.10 Reduced-Area Multiplier	15
1.2 Problem Statement	16
1.3 Objective	17
1.4 Scope of Study	17
CHAPTER 2: LITERATURE REVIEW	
2.1 Comparison of 32-bit Multipliers for Various Performance Measures	19
2.2 54x54-bit Radix-4 Multiplier Based on Modified Booth Algorithm	20
2.3 A Performance Comparison Study on Multiplier Designs	20
2.4 Binary Multiplication of Radix-32 and Radix-256	21
2.5 A 16x16 Bit Modified Radix-16 Booth Encoded Parallel Multiplier	22
CHAPTER 3: METHODOLOGY	
3.1 Flow Chart	23
3.2 Project Activities	24
3.2.1 Project Definition and Objective Identification	24
3.2.2 Literature Review	24
3.2.3 Further Research and Study	24
3.2.4 Design Entry in Verilog HDL	24
3.2.5 Logic Simulation	25
3.2.6 Logic Synthesis	25
3.2.7 Results Analysis in Timing and Area	25
3.3 Materials and Equipments	25

3.3.1 Hardware	25
3.3.2 Software	26
3.4 Gantt Chart	26
CHAPTER 4: DISCUSSIONS AND RESULTS	
4.1 Results of Area and Delay Comparison Performances among 32-bits Multiplier Designs	27
4.1.1 Results of Area Comparison among 32-bits Multiplier Designs	27
4.1.2 Results of Delay Comparison among 32-bits Multiplier Designs	33
4.2 Discussions	40
CHAPTER 5: CONCLUSION	42
REFERENCES	44
APPENDICES	46

LIST OF FIGURES

Figure 1.1: Example of Booth multiplication

Figure 1.2: Dot-notation representation of Radix-2 Booth Encoding multiplier

Figure 1.3: Dot diagram of Radix-4 Booth Encoding multiplier

Figure 1.4: Blocks of three are divided in the multiplier

Figure 1.5: Example of Radix-4 Booth Encoding multiplication

Figure 1.6: Dot-notation of Radix-8 Booth Encoding multiplier

Figure 1.7: Dot-notation of Radix-16 Booth Encoding multiplier

Figure 1.8: Dot diagram representing Radix-32 Booth Encoding multiplier

Figure 1.9: Dot-notation of 8x8 Wallace multiplier

Figure 1.10: Dot diagram of 8x8 Dadda multiplier

Figure 1.11: Dot diagram of 8x8 Reduced-Area

Figure 1.12: An example of the common multiplication

Figure 3.1: Project flow chart

Figure 4.1: Area changes in Area-Optimized Among Radix-based Booth Encoding Multiplier Designs

Figure 4.2: Area changes in Speed-Optimized Among Radix-based Booth Encoding Multiplier Designs

Figure 4.3: Area changes in Auto-Optimized Among Radix-based Booth Encoding Multiplier Designs

Figure 4.4: Area changes in Area-Optimized Among 32-bits Multiplier Designs

Figure 4.5: Area changes in Speed-Optimized Among 32-bits Multiplier Designs

Figure 4.6: Area changes in Auto-Optimized Among 32-bits Multiplier Designs

Figure 4.7: Delay changes summarized in the Speed-Optimized mode of Radix-based Booth Encoding multipliers

Figure 4.8: Delay changes summarized in the Area-Optimized mode of Radix-based Booth Encoding multipliers

Figure 4.9: Delay changes summarized in the Auto-Optimized mode of Radix-based Booth Encoding multipliers

Figure 4.10: Delay changes summarized in the Speed-Optimized mode of 32-bits multipliers

Figure 4.11: Delay changes summarized in the Area-Optimized mode
of 32-bits multipliers

Figure 4.12: Delay changes summarized in the Auto-Optimized mode
of 32-bits multipliers

LIST OF TABLES

Table 1.1: Booth Algorithm Table

Table 1.2: Combinations of C_k and S_k for Radix-2 Booth Encoding multiplier

Table 1.3: Combinations of C_k and S_k for Radix-4 Booth Encoding multiplier

Table 1.4: Combinations of C_k and S_k for Radix-8 Booth Encoding multiplier

Table 1.5: Combinations of C_k and S_k for Radix-16 Booth Encoding multiplier

Table 1.6: Combinations of C_k and S_k for Radix-32 Booth Encoding multiplier

Table 4.1: Area comparison of Radix-based multipliers on Area-Optimized, Speed-Optimized
and Auto-Optimized

Table 4.2: Area performance of 32-bits multipliers in Area-Optimized, Speed-Optimized
and Auto-Optimized modes

Table 4.3: Delay comparison of Radix-based multipliers in Area-Optimized, Speed-Optimized
and Auto-Optimized

Table 4.4: AD and AD^2 computed for all the Radix-based Booth Encoding
designs in all three modes

Table 4.5: Delay performance of 32-bits multipliers in Area-Optimized, Speed-Optimized
and Auto-Optimized modes

Table 4.6: AD and AD^2 computed for 32-bits multiplier designs

Booth Encoding algorithm is one of the most well-known techniques used to reduce the number of partial products added while multiplying the multiplicand and the multiplier. Figure 1.1 above is an example showing how Booth multiplication is being done. Reduction in the number of partial products relies upon the number of bits encoded. It multiplies two signed binary numbers using two's complement notation. [3] It is done by repeatedly adding one of the two predetermined values given by A and S to a Product given by P and right arithmetic shift is performed on P. As referred to Figure 1.1, A represents the Multiplicand, S is the 2's complement form of the Multiplicand and P represents the Product. Arithmetically shift is done by shifting a single place to the right. The length of A, S and P are equaled to $x+y+1$, where x and y is the number of bits in the multiplicand or the multiplier. The value of A will be filled with the multiplicand at the leftmost bits while the value of S will be filled with the two's complement value of the multiplicand at the leftmost bits. The remaining bits left for A and S will be filled with zeros. [4] For the value P, it will be filled with the value of the multiplier at the rightmost bits and the remaining bits are padded with zeros. The two least significant bits of P will then be determined according to Table 1.1.

Table 1.1: Booth Algorithm Table

Two least significant bits of P	Procedure
00	Do nothing. P is directly used in the next step
01	P + A. Overflow ignored
10	P + S. Overflow ignored
11	Do nothing. P is directly in the next step

The loop according to Table 1.1 is repeated depending on the number of bits in the multiplicand or multiplier. The final result of the Booth Encoding multiplication between the multiplicand and the multiplier is the value of P obtained with the least significant bit of P ignored.

Booth Encoding algorithm here is not directly applied in modern arithmetic circuits but serves as a tool in understanding the Radix-based versions of this encoding discussed later.

1.1.2 Radix-2 Booth Encoding Multiplier

In Radix-2 Booth Encoding multiplier, two 16-bits signed numbers are multiplied and a total of 16 partial products are produced and added to obtain the final product of the multiplication. If we let A and B be two n-bit two's complement binary numbers where A represents the multiplicand and B represents the multiplier. The 16-bits of A are represented as $A_{15} A_{14} A_{13} \dots A_2 A_1 A_0$ and the 16-bits of B is represented as $B_{15} B_{14} B_{13} \dots B_2 B_1 B_0$. Both of the multiplicand and the multiplier are in the form of signed binary numbers in two's complement. [5] So, the general equation for the Product:

$$P = A \times B \dots\dots\dots \text{Equation 1}$$

$$A \times B = A \times \{(S_{n-1} \times 2^{n-1}) + (S_{n-2} \times 2^{n-2}) + (S_{n-3} \times 2^{n-3}) + \dots + (S_2 \times 2^2) + (S_1 \times 2^1) + (S_0 \times 2^0)\} \dots\dots\dots \text{Equation 2}$$

where S_k for $n-1 \leq k \leq 0$, is a value that depends on the value of B and can be determined as will be explained next and n represents the number of bits present in the multiplicand or the multiplier. [6]

Radix-2 Booth Encoding multiplier is determined by a collection of 2-bits and it is represented as C_k where $n-1 \leq k \leq 0$. For C_0 , B is appended by a bit of 0 at the least significant bit and is represented by Z. This results in a 16-bits collection as displayed below.

For $n-1 \leq k \leq 1$:

$$C_k = (B_k B_{k-1}) \dots\dots\dots \text{Equation 3}$$

For $k= 0$:

$$C_0 = (B_k Z) \dots\dots\dots \text{Equation 4}$$

The value of S_k with all the possible combinations values of a pair C_k is presented in Table 1.2 below.

Table 1.2: Combinations of C_k and S_k for Radix-2 Booth Encoding multiplier.

C_k	S_k
00, 11	0
01	+1
10	-1

Then, the Radix-2 booth encoding multiplication is rewritten as follows:

$$A \times B = (A \times P_{n-1}) + (A \times P_{n-2}) + (A \times P_{n-3}) + \dots + (A \times P_2) + (A \times P_1) + (A \times P_0) \dots \dots \dots \text{Equation 5}$$

where $P_{n-1} = S_{n-1} \times 2^{n-1}$, $P_{n-2} = S_{n-2} \times 2^{n-2}$, $P_{n-3} = S_{n-3} \times 2^{n-3}$, ..., $P_2 = S_2 \times 2^2$, $P_1 = S_1 \times 2^1$, $P_0 = S_0 \times 2^0$

The final product of the Radix-2 Booth Encoding multiplication is shown below:

$$A \times B = PP_{n-1} \times 2^{n-1} + PP_{n-2} \times 2^{n-2} + PP_{n-3} \times 2^{n-3} + \dots + PP_2 \times 2^2 + PP_1 \times 2^1 + PP_0 \times 2^0 \dots \dots \dots \text{Equation 6}$$

where $PP_{n-1} = (A \times P_{n-1})$, $PP_{n-2} = (A \times P_{n-2})$, $PP_{n-3} = (A \times P_{n-3})$, ..., $PP_2 = (A \times P_2)$, $PP_1 = (A \times P_1)$, $PP_0 = (A \times P_0)$

where PP_{n-1} , PP_{n-2} , PP_{n-3} , ..., P_2 , P_1 , P_0 are known as the partial products of the multiplication.

The dot-notation that represents the 16x16 bits of Radix-2 Booth Encoding multiplier is shown in Figure 1.2 below.

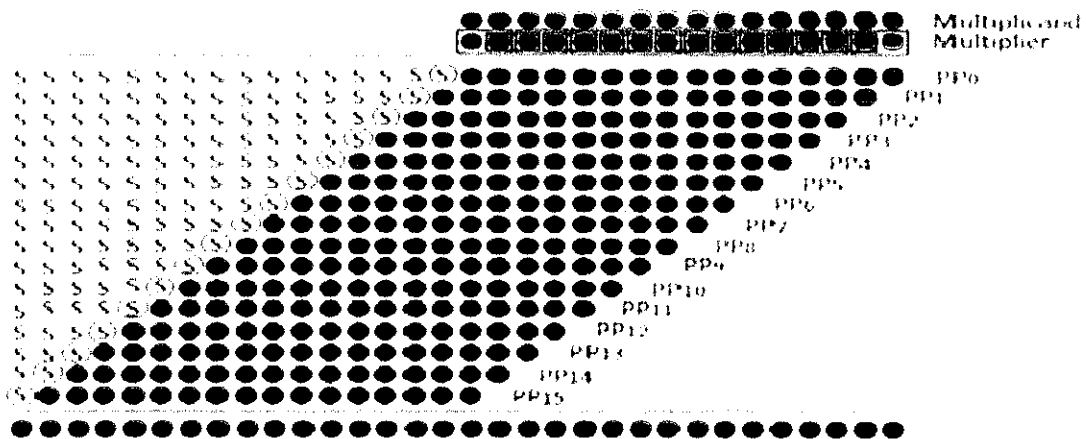


Figure 1.2: Dot notation representation of Radix-2 Booth Encoding multiplier.

1.1.3 Radix-4 Booth Encoding Multiplier

The main cause of delay in the critical path of the multiplication is mainly caused by the addition of the partial products. The delay in the critical path is longer at the normal multiplication is because there are a total of n-partial products need to be computed. [7] However for Radix-4 multiplications, when a 3-bit encoding scheme is used, the number of partial products is reduced by half. It means that if the multiplicand and the multiplier as 16-bits, the partial products produced by Radix-4 Booth Encoding multiplier will be reduced to 8-bits. A collection of 3-bits are taken in one C_k where $N-1 \leq k \leq 0$ and $N=n/2$. At the same time, a factor of 2 can be multiplied to reduce the number of partial products as shown in the equation below.

For $N-1 \leq k \leq 1$:

$$C_k = (B_{2k+1} B_{2k} B_{2k-1}) \dots \dots \dots \text{Equation 7}$$

For $k = 0$ and $Z = 0$:

$$C_k = (B_{2k+1} B_{2k} Z) \dots \dots \dots \text{Equation 8}$$

All the partial products are then multiplied by 2^x and when x is odd, it will be eliminated. So, instead of shifting a partial product by 1-bit before summing the partial products to obtain the final product, we will shift by 2-bits. This means that instead of multiplying it by 2^1 in each partial product, we will multiply it by 2^2 . The value of S_k for all the [8] possible combinations of a pair C_k is changed as shown in Table 1.3.

Table 1.3: Combinations of C_k and S_k for Radix-4 Booth Encoding multiplier.

C_k	S_k
000, 111	0
001, 010	+1
011	+2
100	-2
101, 110	-1

The Radix-4 Booth Encoding general multiplication can be written as stated below:

$$A \times B = (A \times S_{N-1} \times 2^{2*(N-1)}) + (A \times S_{N-2} \times 2^{2*(N-2)}) + \dots + (A \times S_1 \times 2^2) + (A \times S_0 \times 2^0) \dots \dots \dots \text{Equation 9}$$

So, the final product of Radix-4 Booth Encoding multiplier is shown below:

$$A \times B = (PP_{N-1} \times 2^{2*(N-1)}) + (PP_{N-2} \times 2^{2*(N-2)}) + \dots + (PP_1 \times 2^2) + (PP_0 \times 2^0) \dots \dots \dots \text{Equation 10}$$

where $PP_{N-1} = A \times S_{N-1}$, $PP_{N-2} = A \times S_{N-2}$, ..., $PP_1 = A \times S_1$, $PP_0 = A \times S_0$ and PP_{N-1} , PP_{N-2} , ..., PP_1 , PP_0 represents the partial products.

The dot-notation representation of the Radix-4 Booth Encoding multiplier can be seen as shown in Figure 1.3.

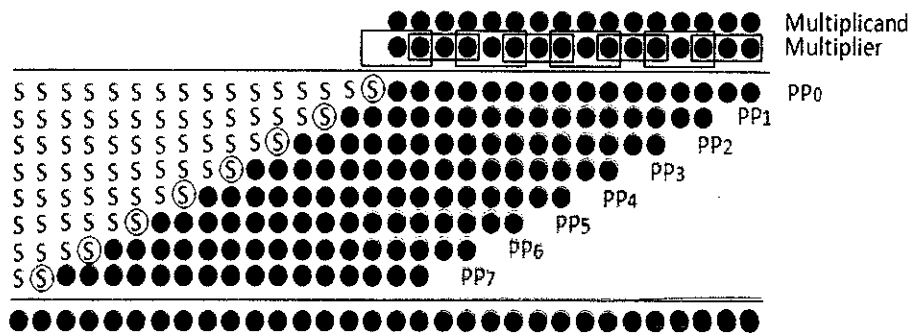


Figure 1.3: Dot diagram of Radix-4 Booth Encoding multiplier.

The grouping blocks of three starts from the least significant bit. [9] [10] Since the rightmost block only uses two bits, a zero can be assumed as the third bit. Overlapping of the blocks is necessary so that whatever happened in the previous block will be known. It is as shown in Figure 1.4 below.

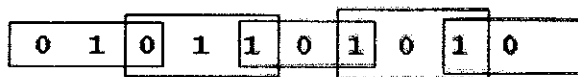


Figure 1.4: Blocks of three are divided in the multiplier.

An example of Radix-4 Booth Encoding multiplication using a multiplicand of +11 multiplied with a multiplier of +19 is explained in Figure 1.5.

Multiplicand	0 0 1 0 1 1
Multiplier	<u>0 1 0 0 1 1</u>
Booth Encoding of multiplier	<u>1 1 -1</u>
-1 x Multiplicand (negative sign extended)	1 1 1 1 1 1 0 1 0 1
1 x Multiplicand	0 0 1 0 1 1
1 x Multiplicand	<u>0 0 1 0 1 1</u>
	0 0 1 1 0 1 0 0 0 1

Figure 1.5: Example of Radix-4 Booth Encoding multiplication

1.1.4 Radix-8 Booth Encoding Multiplier

Radix-8 Booth Encoding multiplier has the same basic concept as Radix-4 Booth Encoding multiplier except that it uses a 4-bit encoding scheme to produce one third the number of partial products. The number of partial products in Radix-8 Booth Encoding multiplier can also be reduced by multiplying a factor of 3 instead of multiplying it with a factor of 2 as discussed in Radix-4 Booth Encoding multiplication. As shown below is a representation of the equation of the collection of 4-bits are taken in one C_k .

where $N-1 \leq k \leq 0$ and $N=n/3$.

For $N-1 \leq k \leq 1$:

$$C_k = (B_{3k+2} B_{3k+1} B_{3k} B_{3k-1}) \dots \text{Equation 11}$$

For $k = 0$ and $Z = 0$:

$$C_k = (B_{3k+2} B_{3k+1} R_{3k} Z) \dots \text{Equation 12}$$

The number of partial products of Radix-8 Booth Encoding multiplier reduces from 16-bits to 5-bits for a 16-bits multiplicand and multiplier value. Instead of shifting a partial product by 2-bits before adding the partial products to get the final product as used in Radix-4 Booth Encoding multiplier, we shift each of the partial products by 3-bits. This means that instead of multiplying it by 2^2 , we multiply it by 2^3 in each partial product generated. All the possible combinations of a pair of C_k and the value of S_k are as follow in Table 1.4.

Table 1.4: Combinations of C_k and S_k for Radix-8 Booth Encoding multiplier.

C_k	S_k
0000, 1111	0
0001, 0010	+1
0011, 0100	+2
0101, 0110	+3
0111	+4
1000	-4
1001, 1010	-3
1011, 1100	-2
1101, 1110	-1

Radix-8 Booth Encoding multiplier is rewritten as:

$$A \times B = (A \times S_{N-1} \times 2^{3*(N-1)}) + (A \times S_{N-2} \times 2^{3*(N-2)}) + \dots + (A \times S_1 \times 2^3) + (A \times S_0 \times 2^0) \dots \dots \dots \text{Equation 13}$$

The final product is given as:

$$A \times B = (PP_{N-1} \times 2^{3*(N-1)}) + (PP_{N-2} \times 2^{3*(N-2)}) + \dots + (PP_1 \times 2^3) + (PP_0 \times 2^0) \dots \dots \dots \text{Equation 14}$$

where $PP_{N-1} = A \times S_{N-1}$, $PP_{N-2} = A \times S_{N-2}$, ..., $PP_1 = A \times S_1$, $PP_0 = A \times S_0$ while PP_{N-1} , PP_{N-2} , ..., PP_1 , PP_0 are the partial products.

Figure 1.6 below represents the dot diagram of the 16x16 bits of Radix-8 Booth Encoding multiplier with 6 partial products generated.

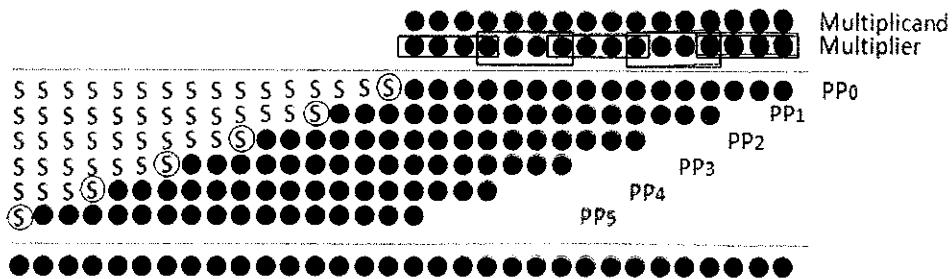


Figure 1.6: Dot-notation of Radix-8 Booth Encoding multiplier.

1.1.5 Radix-16 Booth Encoding Multiplier

The discussion continued on with a higher radix-based multiplication of Radix-16 Booth Encoding multiplication. Radix-16 Booth Encoding multiplier uses a 5-bit encoding

scheme to produce one fourth number of partial products. A factor of 4 is multiplied in the equation to reduce the number of partial products generated by Radix-16 Booth Encoding. Hence, a collection of 5-bits are taken in one C_k where $N-1 \leq k \leq 0$ and $N=n/4$ and further reduces the partial product to one fourth.

For $N-1 \leq k \leq 1$:

$$C_k = (B_{4k+3} B_{4k+2} B_{4k+1} B_{4k} B_{4k-1}) \dots \dots \dots \text{Equation 15}$$

For $k = 0$ and $Z = 0$:

$$C_k = (B_{4k+3} B_{4k+2} B_{4k+1} R_{4k} Z) \dots \dots \dots \text{Equation 16}$$

Each of the partial products will then be shifted by 4-bits in order to obtain the final product of the multiplication. So, 2^4 are multiplied in each of the partial products in each of the partial products. As the higher Radix-based Booth Encoding multiplier reaches, the more complex is the combinations of one pair of C_k and S_k as shown in Table 1.5.

Table 1.5: Combinations of C_k and S_k for Radix-16 Booth Encoding multiplier.

C_k	S_k
0000, 1111	0
0001, 0010	+1
0011, 00100	+2
00101, 00110	+3
00111, 01000	+4
01001, 01010	+5
01011, 01100	+6
01101, 01110	+7
01111	+8
10000	-8
10001, 10010	-7
10011, 10100	-6
10101, 10110	-5
10111, 11000	-4
11001, 11010	-3
11011, 11100	-2
11101, 11110	-1

The multiplication of Radix-16 Booth Encoding multiplier can be written as shown below:

$$A \times B = (A \times S_{N-1} \times 2^{4*(N-1)}) + (A \times S_{N-2} \times 2^{4*(N-2)}) + \dots + (A \times S_1 \times 2^4) + (A \times S_0 \times 2^0) \dots \dots \dots \text{Equation 17}$$

So, the final product is:

$$A \times B = (PP_{N-1} \times 2^{4*(N-1)}) + (PP_{N-2} \times 2^{4*(N-2)}) + \dots + (PP_1 \times 2^4) + (PP_0 \times 2^0) \dots \dots \dots \text{Equation 18}$$

where $PP_{N-1} = A \times S_{N-1}$, $PP_{N-2} = A \times S_{N-2}$, ..., $PP_1 = A \times S_1$, $PP_0 = A \times S_0$ and PP_{N-1} , PP_{N-2} , ..., PP_1 , PP_0 represents the partial products.

The dot-notation of Radix-16 booth encoding multiplier can be best represented as follow in Figure 1.7.

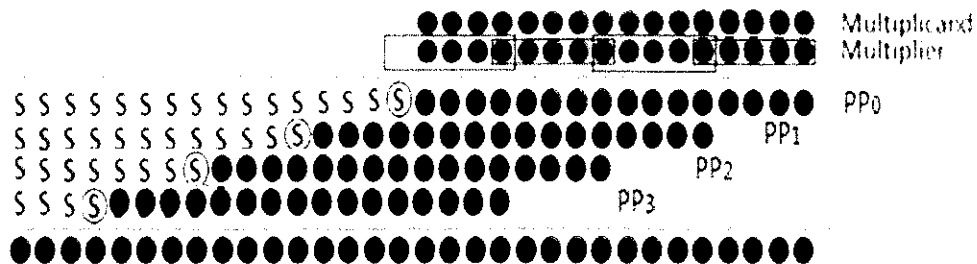


Figure 1.7: Dot diagram of Radix-16 Booth Encoding multiplier

1.1.6 Radix-32 Booth Encoding Multiplier

Radix-32 is further developed where it uses a 6-bit encoding method to reduce the number of partial products to one fifth. In order to reduce the number of partial products of the Radix-based multiplier, a factor of 5 is multiplied in the equation below.

$$A \times B = (A \times S_{N-1} \times 2^{5*(N-1)}) + (A \times S_{N-2} \times 2^{5*(N-2)}) + \dots + (A \times S_1 \times 2^5) + (A \times S_0 \times 2^0) \dots \dots \dots \text{Equation 19}$$

The multiplication of Radix-32 Booth Encoding multiplier is largely determined by a collection of 6-bits as one C_k where $N-1 \leq k \leq 0$ and $N=n/5$ as displayed.

For $N-1 \leq k \leq 1$:

$$C_k = (B_{5k+4} B_{5k+3} B_{5k+2} B_{5k+1} B_{5k} B_{5k-1}) \dots \text{Equation 20}$$

For $k = 0$ and $Z = 0$:

$$C_k = (B_{5k+4} B_{5k+3} B_{5k+2} B_{5k+1} R_{5k} Z) \dots \text{Equation 21}$$

Each of the partial products of the Radix-32 Booth Encoding multiplier is then shifted by 5-bits. This means that 2^5 will be multiplied in each of the partial products computed in the equation. Table 1.6 shows the combinations of one pair of C_k together with the value of S_k .

Table 1.6: Combinations of C_k and S_k for Radix-32 Booth Encoding multiplier.

C_k	S_k
000000, 111111	0
000001, 000010	1 x Multiplicand
000011, 000100	2 x Multiplicand
000101, 000110	3 x Multiplicand
000111, 001000	4 x Multiplicand
001001, 001010	5 x Multiplicand
001011, 001100	6 x Multiplicand
001101, 001110	7 x Multiplicand
001111, 010000	8 x Multiplicand
010001, 010010	9 x Multiplicand
010011, 010100	10 x Multiplicand
010101, 010110	11 x Multiplicand
010111, 011000	12 x Multiplicand
011001, 011010	13 x Multiplicand
011011, 011100	14 x Multiplicand
011101, 011110	15 x Multiplicand
011111	16 x Multiplicand
100000	-16 x Multiplicand
100001, 100010	-15 x Multiplicand
100011, 100100	-14 x Multiplicand
100101, 100110	-13 x Multiplicand
100111, 010111	-12 x Multiplicand
101001, 101010	-11 x Multiplicand
101011, 101100	-10 x Multiplicand
101101, 101110	-9 x Multiplicand
101111, 110000	-8 x Multiplicand
110001, 110010	-7 x Multiplicand
110011, 110100	-6 x Multiplicand
110101, 110110	-5 x Multiplicand
110111, 111000	-4 x Multiplicand
111001, 111010	-3 x Multiplicand
111011, 111100	-2 x Multiplicand
111101, 111110	-1 x Multiplicand

The final product of Radix-32 Booth Encoding multiplication is equivalent to:

$$A \times B = (PP_{N-1} \times 2^{5*(N-1)}) + (PP_{N-2} \times 2^{5*(N-2)}) + \dots + (PP_1 \times 2^5) + (PP_0 \times 2^0) \dots \dots \dots \text{Equation 22}$$

where $PP_{N-1} = A \times S_{N-1}$, $PP_{N-2} = A \times S_{N-2}$, ..., $PP_1 = A \times S_1$, $PP_0 = A \times S_0$ and PP_{N-1} , PP_{N-2} , ..., PP_1 , PP_0 represents the partial products.

Figure 1.8 is a dot diagram representation of a 16x16 bits Radix-32 Booth Encoding multiplier where 3 partial products are produced as shown below. This method reduces the timing of the multiplication but at the same time the area of the multiplier gets a lot more complicated.

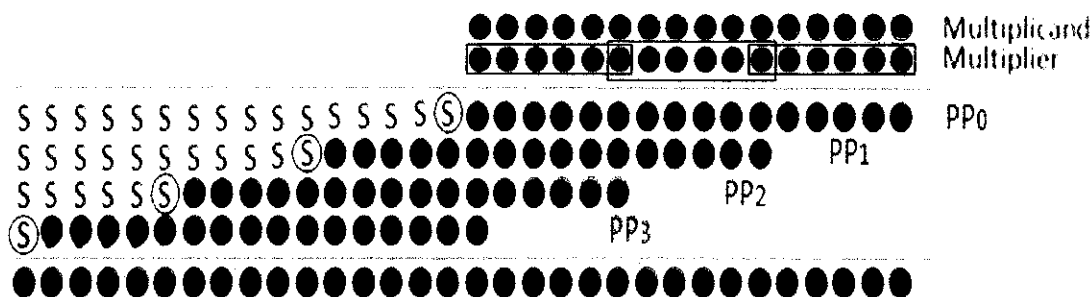


Figure 8: Dot diagram representing Radix-32 Booth Encoding multiplier.

1.1.7 Array Multiplier

The Array multiplier algorithm uses a simple addition and shifting method which largely comprises of AND gates and adders. Each of the partial products present in the array multiplier is generated by multiplying the multiplicand with the multiplier. The partial products computed are shifted according to the bit orders and addition is being done after that. When array multiplication is applied, we need to add as many numbers of partial products as there are present in the multiplier bits. [11]

1.1.8 Wallace Multiplier

The main advantage of implementing Wallace tree is because there is only a reduction stage delay of the order $O(\log n)$ and each layer has $O(1)$ propagation delay. [12] The logarithm of $O(\log n)$ only present in the reduction layers and each layer has $O(1)$

propagation delay. The partial products are made up of $O(1)$ while the final addition is $O(\log n)$. So, the multiplication only consists of $O(\log n)$ and this is not any slower than addition being computed.

The logarithm increase in delay will contribute to major speed gain over array designs which have a linear increase in delay at the higher structural complexity cost. The partial products that are scaled down based on Wallace algorithm that depends on the height of the matrix present in each reduction delay stage in the multiplier. The height of the matrix in the j^{th} reduction stage, w_j is given by the following recursive equations as shown below.

$$w_0 = N$$

$$w_{j+1} = 2 * \lfloor w_j / 3 \rfloor + w_j \bmod 3$$

Figure 1.9 below shows an example of a representation of the 8x8 dot-notation of the Wallace multiplier.

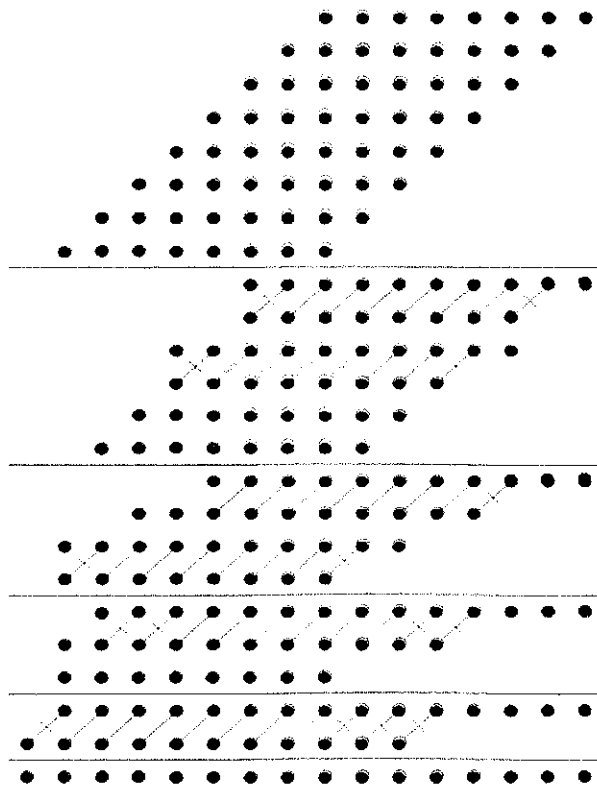


Figure 1.9: Dot-notation of 8x8 Wallace multiplier.

1.1.9 Dadda Multiplier

Dadda multiplier is very similar to Wallace multiplier but it is slightly faster in speed and requires fewer gate logics. [13] However, the part that makes Dadda multiplier different from Wallace multiplier is that Dadda multiplier does not attempt to reduce as many partial products in each layer but instead it performs as few reductions as possible. This makes the Dadda multipliers less expensive in the reduction phase but contain longer numbers in each stage, thus requiring slightly bigger carry-propagate adders.

The formation of partial products for this particular design presented here is the same as the Wallace multiplier design. [14] However, the reduction stage is different where a series of recursive steps were used to determine the heights of each stage and the number of additions needed in order to achieve the heights of each stage.

Here is an explanation of the reduction stage in the Dadda multiplier design. We make $d_1 = 2$ and $d_{j+1} = \lceil 1.5 * d_j \rceil$ where d_j represents the height of the matrix for the j th stage. This is repeated until the biggest j th stage is achieved which includes the original N height matrix to contain at least one column that has more than d_j dots. Starting from the end in the j th stage, (3, 2) and (2, 2) counters are placed in order to achieve a reduced matrix. As a result, the multiplier only has columns with more than d_j dots as it receives carries from (3, 2) and (2, 2) counters that has been reduced. Then, we make $j = j - 1$ and the reduced matrix step as mentioned previously was repeated until a matrix with a height of $N=2$ generated. This happens when $j = 1$.

The example of the dot-notation diagram for the 8x8 bit Dadda multiplier is as shown in Figure 1.10.

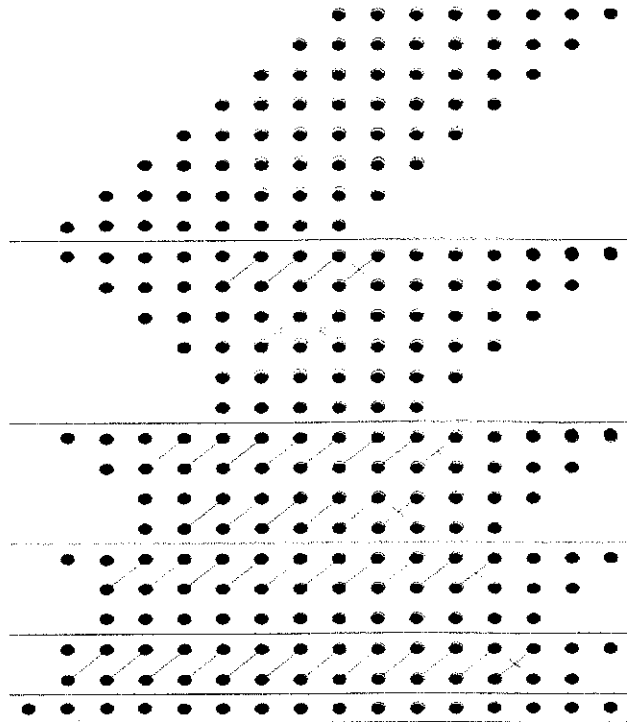


Figure 1.10: Dot diagram of 8x8 Dadda multiplier.

1.1.10 *Reduced-Area Multiplier*

Reduced-Area multiplier discussed here is a refinement of the Wallace and Dadda scheme. The huge difference in the reduction scheme of the Reduced-Area multiplier compared to Wallace and Dadda algorithm includes the maximum number of (3,2) counters used in each of the reduction stage is used as early as possible to minimize the number of bits to be routed in the next stage. [15] On the other hand, the (2,2) counter are carefully placed to reduced the word size of the carry-propagate adder. Figure 1.11 shows the dot diagram representation for an 8x8 bit Reduced-Area multiplier.

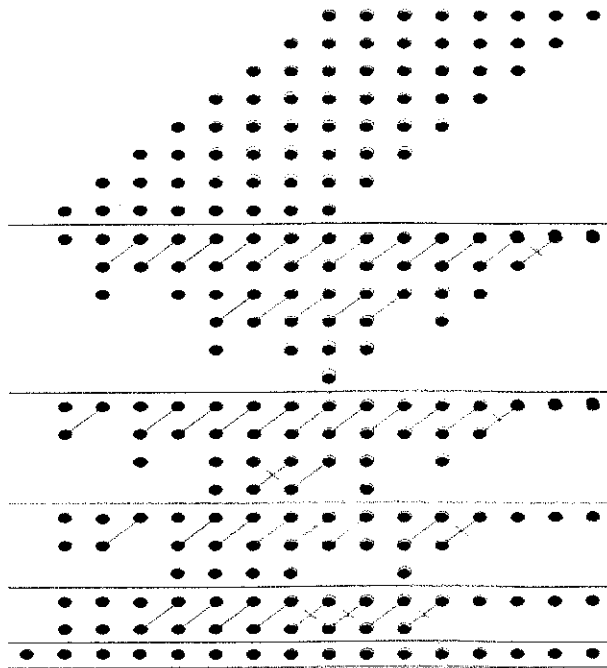


Figure 1.11: Dot diagram of 8x8 Reduced-Area multiplier.

1.2 Problem Statement

Multipliers are the most critical components for the efficient implementation of Digital Signal Processing (DSP) and Cryptographic algorithms. As the non-memory sub-block of the microprocessor with the largest size and delay, multipliers have a big impact on the cycle time of the microprocessor. For this purpose, the use of Booth Encoding multiplier has been a popular method to reduce the number of partial product in a multiplication in order to reduce the delay of the partial products accumulation and to simplify the partition of the multiplier into several shorter stages.

The basic multiplication approach is done by shifting and adding. That can be implemented by shifting the multiplicand the appropriate number of columns and multiply it with the value of the digit in the column of the multiplier at each column in the multiplier to obtain a partial product. The partial products obtained are then added overall to obtain the final result. Figure 1.12 below shows an example of the normal long multiplication is being calculated.

Multiplicand:	0 0 1 0 1 1
Multiplier:	<u>0 1 0 0 1 1</u>
	0 0 1 0 1 1
	0 0 1 0 1 1
	0 0 0 0 0 0
	0 0 0 0 0 0
	<u>0 0 1 0 1 1</u>
	0 0 1 1 0 1 0 0 0 1

Figure 1.12: An example of the common multiplication

The performance given by each of the multipliers designed in this project was synthesized and analyzed in terms of area and timing in order to determine the most suitable type of applications that can be used. Multipliers being compared in this project include Radix-based Booth Encoding multipliers which are Radix-2, Radix-4, Radix-8, Radix-16 and Radix-32, Array multiplier, Wallace multiplier, Dadda multiplier and Reduced-Area multiplier.

1.3 Objective

The objectives of this project include:

1. To compare the performance in term of area and timing synthesized by the multiplier designs.
2. To determine if the speed of the Radix-based multipliers reduced as the Radix-based multipliers goes higher.
3. To determine if the area of the Radix-based multipliers increased as the Radix-based multipliers go higher.

1.4 Scope of Study

The project focuses comparing the performance of Radix-based Booth Encoding multiplier designs created using Verilog HDL. The performances were then compared based on area and speed of each of the designs created. The scope of the project is widely based on Computer Arithmetic where it shows how the algorithm of the multipliers are invented and implemented.

Besides that, Verilog HDL needs to be mastered in order to design the Radix-based Encoding multipliers. The scope of this project also includes the knowledge of logic simulation and logic synthesizing the designs in order to obtain the performance in three modes which are Area-Optimized mode, Speed-Optimized mode and Auto-Optimized mode in Leonardo Spectrum.

CHAPTER 2

2. LITERATURE REVIEW

2.1 Comparison of 32-bit Multipliers for Various Performance Measures [16]

This technical paper written by S.Shah, A. J. Al-Khabb and D. Al-Khabb from Concordia University, Montreal, was about the comparison of five various 32-bits multiplier to obtain different performance measures. The multiplier designs that were being compared are Array multiplier, Modified Booth Radix-4 multiplier, Optimized Wallace Tree multiplier, Combined Modified Booth-Wallace Tree multiplier and Twin Pipe Serial Parallel multiplier.

This technical paper started off with an introduction on the multiplier industry and the advantages of each of the multipliers. It continued on to the architecture of Array multiplier, Modified Booth multiplier, Wallace Tree multiplier, Twin Pipe Serial-Parallel multiplier and Combined Modified Booth-Wallace Tree multiplier. Next, it stated the result of the multipliers and findings after synthesizing each multiplier used in this research by targeting the minx FPGA 4052XL-1HQ240C. The comparison performances of each of the multiplier examined are based on the synthesis reports obtained. The part number and speed grade of the FPGA device with same design constraints used in the synthesis process for each multiplier are the same. The performance findings in this technical paper include area, delay and power consumption. This paper also included the calculation of delay x power, area x power, area x delay and area x delay².

Based on the results obtained, the delay of Wallace Tree multiplier and Combined Booth-Wallace Tree multiplier was almost the same and it had the least value. Both Wallace Tree multiplier and Combined Booth Wallace Tree multiplier were the fastest among five multipliers tested in this research. The delay x power calculation obtained was also the least for Wallace Tree multiplier and Combined Booth Wallace Tree multiplier. So, Wallace Tree multiplier and Combined Booth Wallace Tree multiplier were the best multiplier when the performance was measured by delay and delay x power. Serial Parallel multiplier was the best multiplier choice for the performance based on reduced area, power consumption, area x power and area x delay. If the performance was

determined by area x delay², Modified Booth-Wallace Tree multiplier would be the most suitable choice to use.

2.2 54x54-bit Radix-4 Multiplier Based on Modified Booth Algorithm [17]

This technical paper written by Ki-seon Cho, Jong-on Park, Jin-seok Hong, Goang-seog Choi from Samsung Electronics, Korea, described a low power and high speed multiplier that was suitable for standard cell-based ASIC design. The whole design was done using Verilog HDL and implemented using an available EDA tool chain. This paper started with an introduction on the benefits of fast parallel multiplier and the advantages that Booth multipliers provide in the Digital Signal Processing industry. It continued on describing the architecture of the 54x54 bit multiplier, the circuit design of Booth Encoder based on Modified Booth Algorithm, comparator and conditional sum adder, comparisons of the proposed design methods and conventional design methods used. It finally ended with the discussion of the simulation results obtained.

The simulation results obtained from the research done gave the following propagation time of critical path of 3.25ns at 2.5V on a 0.18um process technology. It is almost 21% faster in speed compared to the conventional multiplier.

2.3 A Performance Comparison Study on Multiplier Designs [18]

This technical paper written by Chris Y.H. Lee, Lo Hai Hiung , Sean W.F. Lee and Nor Hisham Hamid from Universiti Teknologi PETRONAS and Emerald Systems Sdn. Bhd. investigates the relative performance of Array multiplier, Wallace multiplier, Dadda multiplier and Reduced Area multiplier designed in Verilog HDL and are synthesized using TSMC 0.35-micron ASCII Design Kit standard cell library. [14] The data performance was extracted after the designs were synthesized in Leonardo Spectrum for area, speed and auto-optimization modes.

This paper started with an introduction of the usage of multipliers and previous research done on multipliers and their performance results. It continued with the algorithm description of the Array multiplier architecture, Wallace multiplier architecture, Dadda multiplier architecture and Reduced Area multiplier architecture. It proceeds on by

describing with the process flow of the research and followed by a discussion on the results obtained after the synthesis of the multiplier designs. The discussion was divided into three sections which were area comparison, delay comparison and comparison of 32x32 bit tree designs. The paper was ended off with a conclusion of which multiplier was best-suited with what applications.

The results obtained indicate that Wallace multiplier was best selected for high speed applications and independent of area constraints if it was based on the comparison of 32x32 bit design. The Dadda and Reduced Area multiplier were best selected for speed when they were synthesized to minimized area.

2.4 Binary Multiplication of Radix-32 and Radix-256 [19]

This technical paper was prepared by Peter-Michael Seidal, Lee D McFearin and David W Matula from the Department of Computer Science and Engineering, Southern Methodist University in Dallas, Texas. This paper focuses on the generation and reduction of partial products in Radix-32 and Radix-256 multiplier designs. The partial products of Radix-32 and Radix-256 multiplier designs created in this paper depended on the multiples of radices 7 or 11. The proposed designs were then compared to the traditional Booth recoding method.

It started with an introduction of binary multiplication and explanations of the number of partial product produced in Radix-4, Radix-8 and Radix-32 and Radix-256 multiplier designs as proposed in this paper. A further description about the theory and algorithm of binary multiplication, higher radix partial product generation, secondary radix operand conversion and the multiplier designs of Radix-32 and Radix-256 architecture.

The delays of the proposed designs were compared with the partial product generation and reduction using conventional Booth recoding Radix-4 and Radix-8. In the delay analysis, the implementation at gate level and the delay on the critical paths in logic levels were taken into account. The proposed encoding scheme reduces the number of hard multiplies required for higher radix multiplication using conventional Booth recoding. The Radix-32 multiplication has one hard multiple calculations while Radix-

256 multiplication has two hard multiply. The partial products depending on these hard multiplies are known in advance reducing the fan-out requirements for these multiplies and providing greater design flexibility based on the specific implementation constraints. It was also known that the proposed designs also reduce the maximum fan-out of p-bit multiplicand to p/5 for Radix-32 and p/8 for Radix-256.

2.5 A 16x16 Bit Modified Radix-16 Booth Encoded Parallel Multiplier [20]

This technical paper written by Ahmed Elhossini, Ali Rashid and Mohammed K. Refai for the Al-Azhar Engineering Eight International Conference in 2004. This paper introduced a new multiplier structure based on 16x16 Radix-16 Booth algorithm and CSA addition. A pipeline version of the designs was also introduced. These designs were modeled using VHDL and synthesized using Xilinx FPGAs and ASIC standard cell library.

It started with an introduction of the parallel multipliers and the steps of the multiplication process. It proceeded with the explanation and the theory of the Modified Radix-16 Booth algorithm, the new architecture introduced in this paper and the implementation of the new architecture.

From the analysis and the findings, it was known that the new architecture proposed in this paper resulted in a multiplication time of 16.04ns on the Spartan II FPGA chip, 12.8ns for the ASIC standard cell library without final adder optimization. The comparison with other architecture was also analyzed and showed an improvement of 30% in speed.

CHAPTER 3

3. METHODOLOGY

3.1 Flow Chart

In order to achieve the project's objective, there are various steps that need to be executed. The procedures involved are as shown in the flow chart in Figure 3.1.

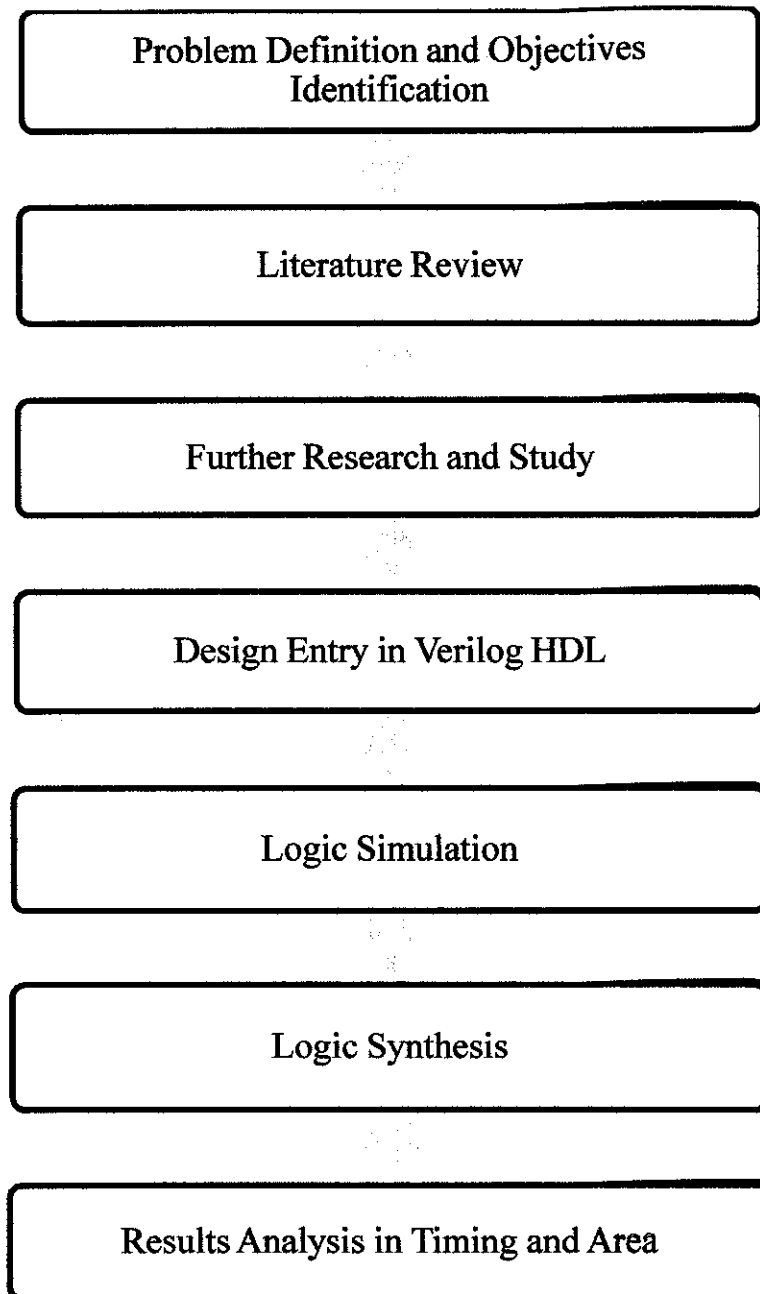


Figure 3.1: Project flow chart

3.2 Project Activities

3.2.1 Project Definition and Objectives Identification

First of all, we must clearly understand what needed to be done and the outcome of the project that we are doing. The project must be well-defined so that what was required for the project would be easier and we could concentrate more on the topics that were involved in the project. The main objectives of the project must be identified so that the objectives can be measured and achieved at the end of the project.

3.2.2 Literature Review

Next, technical papers that had been written by researchers were read to build up the knowledge on Radix-based multipliers to understand on the techniques and designs used and proposed by researchers and the outcomes of their project. This would be a tremendous help during the process of creating the multiplier designs and analyzing the performance of the multipliers.

3.2.3 Further Research and Study

Further research and study through articles, books or online materials were read and studied throughout the process of developing the project. A full understanding and basic on the project must be equipped in order to start drafting and designing the multiplier.

3.2.4 Design Entry in Verilog HDL

Radix-based Booth Encoding multipliers in this project were designed using Verilog Hardware Description Language. Testbench was also created in the design process to run logic simulation in Modelsim, software produced by Mentor Graphics. The design process will take some time to develop in order to obtain correct and effective designs of the Radix-based multipliers. The Verilog codes of the Radix-based multiplier are shown in Appendix I.

3.2.5 Logic Simulation

After the Radix-based Booth Encoding multipliers were designed using Verilog HDL, it is verified using Modelsim to check for any syntax errors and mistakes in the radix-based multipliers designs created in the HDL design entry process. When everything runs accordingly, we proceed with the synthesis. Some figures of the Radix-based Booth Encoding designs in logic simulation are shown in Appendix II.

3.2.6 Logic Synthesis

The designs were then synthesized using TSMC 0.35-micron of the ASIC standard cell library through Leonardo Spectrum, specially used to do logic synthesis on the Radix-based multiplier designs created in Verilog HDL. Standard cell library is used in this project instead of downloading it to an FPGA board because we need to obtain an accurate performance of the designs and when we download the designs to the board, the performance data might be skewed by the internal structure and available multiplier blocks in the FPGA. Appendix III showed some of the figures where the Radix-based Booth Encoding multipliers were being synthesized in Leonardo Spectrum.

3.2.7 Results Analysis in Timing and Area

The performance are then compared to each of the Radix-based multipliers designs created in this project as well as the Array, Wallace, Dadda and Reduced-Area multiplier designs created by Chris Lee. With this analysis, we were able to tell which type of multiplier gives us the lowest speed and the smallest area and vice versa. The synthesis reports of the timing and area of the Radix-based Booth Encoding multipliers were shown in Appendix IV.

3.3 Materials and Equipments

The equipments that were used throughout this project can be divided into two parts which includes the hardware and software as listed below.

3.3.1 Hardware

- Personal computer

- Verilog Hardware Description Language

3.3.2 Software

- Modelsim by Mentor Graphics (Logic Simulation)
- Leonardo Spectrum (Logic Synthesis)
- ASIC Design Kit Standard Cell Library, TSMC 0.35-micron (Logic Synthesis)

3.4 Gantt Chart

The Gantt Chart of this project can be seen attached in Appendix V. The Gantt Chart showed an overview of the project timeline over a span of two semesters.

CHAPTER 4

4. RESULTS AND DISCUSSIONS

4.1 Results of Area and Delay Comparison Performance among 32-bits Multiplier Designs

4.1.1 Results of Area Comparison among 32-bits Multiplier Designs

Based on the synthesis results of the Radix-based Booth Encoding multiplier displayed in Table 4.1, Area Optimization mode gave the best area savings compared to other modes of settings used for all the 32-bits Radix-based Booth Encoding multiplier designs.

The best-case Area Optimized results showed that Radix-4 Booth Encoding multiplier outperformed the other Radix-based Booth Encoding multiplier and has the most area saving compared to the others. However, the area saving becomes less significant as the higher the Radix-based Booth Encoding multipliers reached. The area reduced to about 44.95% from Radix-2 to Radix-4 Booth Encoding multiplier. The area then showed an increase of 49.10% from Radix-4 to the Radix-8 multiplier design, a hike of 99.15% from Radix-8 to Radix-16 and an increment of 78.06% from Radix-16 to Radix-32. Figure 4.1 shows the changes in the Radix-based area performances.

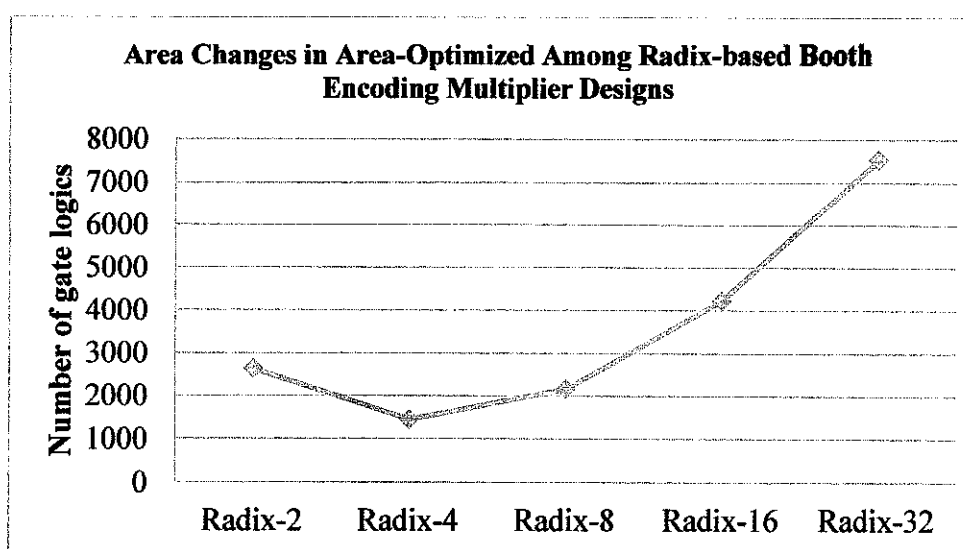


Figure 4.1: Area changes in Area-Optimized Among Radix-based Booth Encoding Multiplier Designs

For the Speed-Optimized mode, the logic gate usage increased significantly compared to the Area Optimized mode settings. From Radix-2 to Radix-4 Booth Encoding multiplier, it resulted in a decrement of 50.22%, a continual increment up to 49.13% from Radix-4 to Radix-8 multiplier, 101.33% of increase shown in Radix-8 to Radix-16 multiplier and finally an increase of 87.64% for Radix-16 to Radix-32 multiplier. The area gets bigger as the higher the Radix-based Booth Encoding multipliers go because the partial products being computed gets more complicated along the way. Figure 4.2 summarizes the changes of Radix-based Booth Encoding multiplier in terms of area performance.

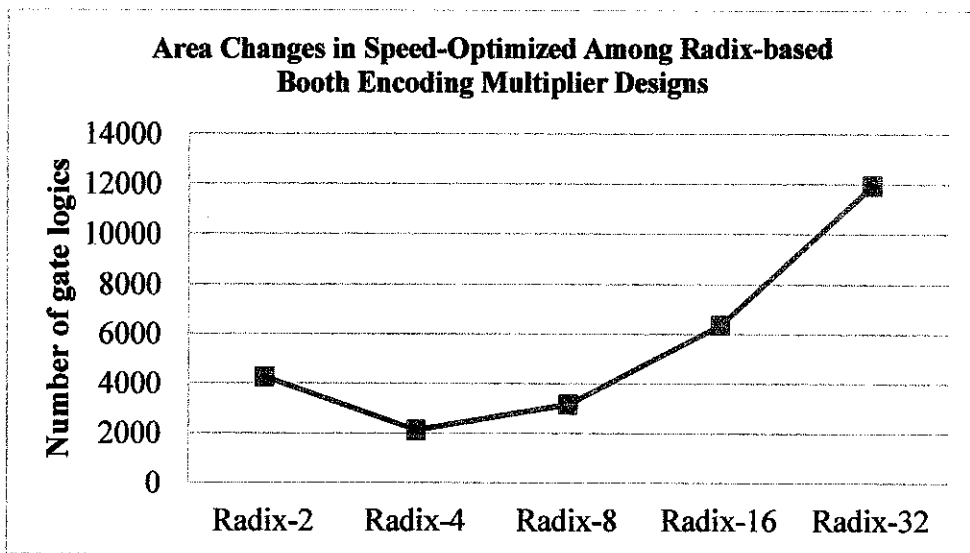


Figure 4.2: Area changes in Speed-Optimized Among Radix-based Booth Encoding Multiplier Designs

Table 4.1: Area comparison of Radix-based multipliers on Area-Optimized, Speed-Optimized and Auto-Optimized.

Area-Optimized	
Radix-2 Booth Encoding multiplier	2,634
Radix-4 Booth Encoding multiplier	1,450
Radix-8 Booth Encoding multiplier	2,162
Radix-16 Booth Encoding multiplier	4,234
Radix-32 Booth Encoding multiplier	7,539
Speed-Optimized	
Radix-2 Booth Encoding multiplier	4,261
Radix-4 Booth Encoding multiplier	2,121
Radix-8 Booth Encoding multiplier	3,163
Radix-16 Booth Encoding multiplier	6,368
Radix-32 Booth Encoding multiplier	11,949
Auto-Optimized	
Radix-2 Booth Encoding multiplier	2,634
Radix-4 Booth Encoding multiplier	1,450
Radix-8 Booth Encoding multiplier	2,162
Radix-16 Booth Encoding multiplier	4,232
Radix-32 Booth Encoding multiplier	7,468

In the Auto-Optimized mode, the time spent on synthesis effort is expected to be the least and it will produce a result identical in structure to the original HDL design. All the multiplier designs were comparatively very much the same to one another except for Radix-16 and Radix-32 Booth Encoding multiplier. From these findings, we were able to conclude that when a non-optimizing synthesis mode is used, it favors more towards an area-efficient output design. Figure 4.3 summarizes the area performance changes in the Radix-based Booth Encoding multiplier using the Auto-Optimized mode.

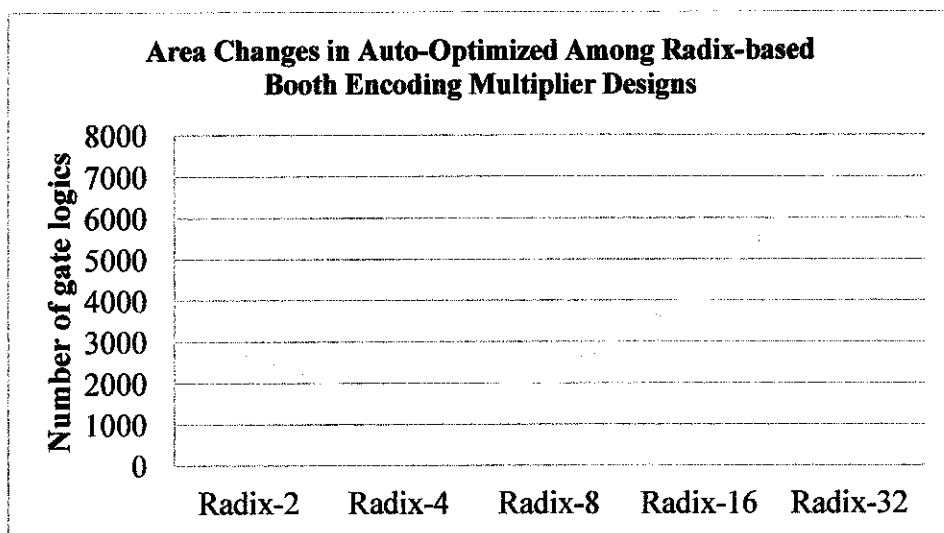


Figure 4.3: Area changes in Auto-Optimized Among Radix-based Booth Encoding Multiplier Designs

In my opinion, the analysis shows that the area performance increased as the higher the Radix-based multipliers goes is mainly due to the increasing number of the logic gates produced and the more complex structure of the multiplier designs as can be seen during the synthesis process as displayed in Appendix III. Based on the overall findings among the Radix-based Booth Encoding multiplier, Radix-4 Booth Encoding is the most compact in terms of area constraints and uses the least logic gates. This is true because when the Radix-based Booth Encoding multiplier increases, the number of partial product determined by the combinations of S_k and C_k gets more complicated and the number of bits in a collection of C_k also increases.

Table 4.2: Area performance on 32-bits multipliers in Area-Optimized, Speed-Optimized and Auto-Optimized modes.

Area-Optimized	
Array multiplier	1,744
Wallace multiplier	1,905
Dadda multiplier	1,782
Reduced-Area multiplier	1,802
Radix-4 Booth Encoding multiplier	1,450
Speed-Optimized	
Array multiplier	2,855
Wallace multiplier	2,732
Dadda multiplier	3,014
Reduced-Area multiplier	2,780
Radix-4 Booth Encoding multiplier	2,121
Auto-Optimized	
Array multiplier	1,738
Wallace multiplier	1,840
Dadda multiplier	1,738
Reduced-Area multiplier	1,739
Radix-4 Booth Encoding multiplier	1,450

When we compared the area performance with the other 32-bits multiplier designs, Table 4.2 displayed the results based in the Area-Optimized mode settings used during the synthesis process that shows the result of best savings for area if it is compared to the Speed-Optimized and Auto-Optimized modes. Here, we take the Radix-based Booth Encoding multiplier that has the best performance produced which is the Radix-4 Booth Encoding multiplier to compare with the other four multiplier designs.

In the Area-Optimized mode among all the 32-bits multiplier designs analyzed, Radix-4 Booth Encoding multiplier has the best area saving results compared to Array, Wallace, Dadda and Reduced-Area multiplier synthesized using the TSMC 0.35-micron ASIC standard cell library.

The next best result in area saving goes to Array multiplier which increased about 20.28% from the Radix-4 design. Next, the area hiked up to about 2.18% for Dadda multiplier and this is followed by an increment of 1.12% for Reduced-Area multiplier. Wallace multiplier design presents the highest area performance with 5.72% increase compared to the Reduced-Area design. Figure 4.4 display the changes in area performance in Area-Optimized mode among 32-bits multipliers.

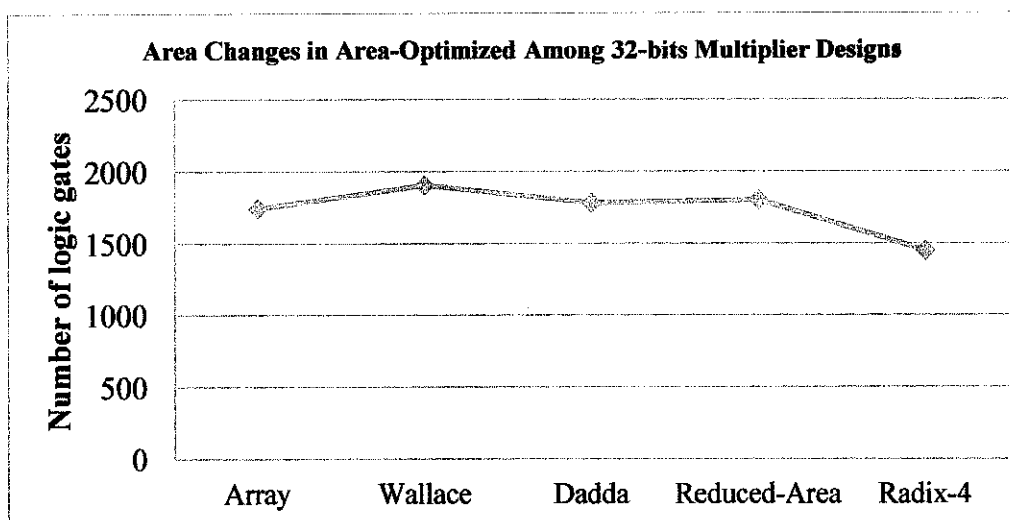


Figure 4.4: Area Changes in Area-Optimized Among 32-bits Multiplier Designs

When in the Speed-Optimized mode, the Dadda multiplier has the biggest area performance with 3,014 gate logic usage among all the five multipliers presented in this paper. Then, it follows with a decrease of 5.28% from the Array multiplier and further dropped to about 2.63% in the Reduced-Area design. Next, Wallace multiplier experienced a degradation of 1.73% and Radix-4 Booth Encoding multiplier continued to decrease around 22.36% which gives us the lowest in the gate logic usage among the designs studied. Figure 4.5 below summarizes the area changes among 32-bits multiplier designs in Speed-Optimized mode.

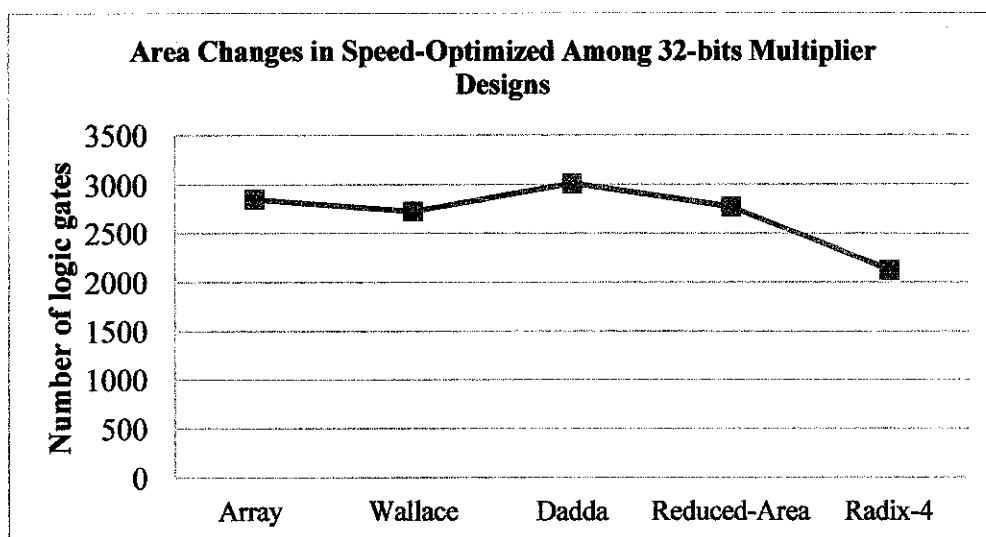


Figure 4.5 Area Changes in Speed-Optimized Among 32-bits Multiplier Designs

The number of the gate level logic produced by Array, Dadda, Reduced-Area and Radix-4 Booth Encoding multiplier in the Auto-Optimized mode is very similar to one another except for Wallace design. By using Auto-Optimized mode in Leonardo Spectrum during the synthesis, the time spent is expected to be the lowest and an identical result in structure to the original HDL design will be shown. The overall findings of the area performance conclude that Radix-4 Booth Encoding multiplier shows the best findings in the area performance among all three modes of Area-Optimized, Area-Optimized and Auto-Optimized. In addition, the results obtained from the Speed-Optimization mode were different from the results obtained from the Area-Optimization and Auto-Optimization mode where Wallace multiplier exhibited the largest area performance instead of Dadda multiplier as can be seen when being synthesized in Speed-Optimized mode. Figure 4.6 shows the changes in area among the 32-bits multipliers.

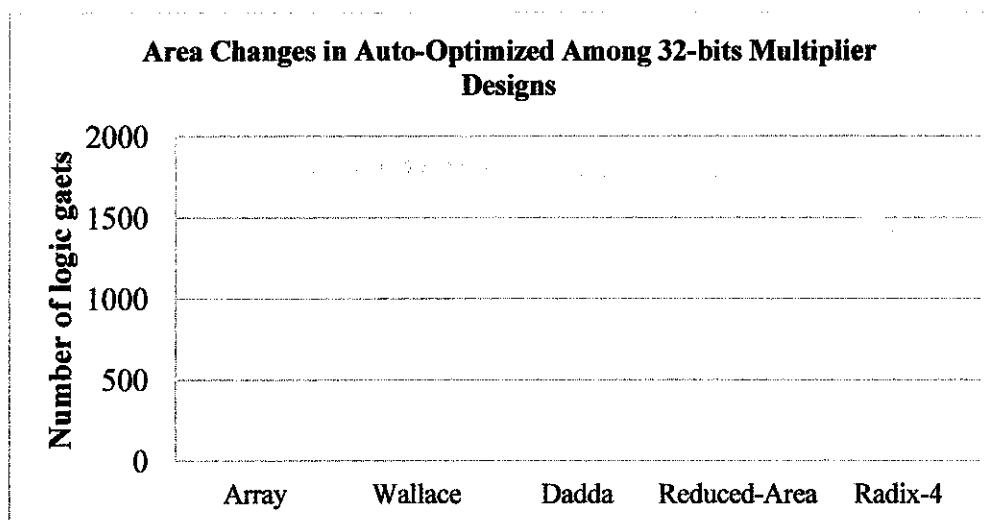


Figure 4.6: Area Changes in Auto-Optimized Among 32-bits Multiplier Designs

In my opinion, if we compare the five multiplier designs based on the number of logic gates produced, Radix-4 Booth Encoding multiplier is also the best multiplier in terms of area saving. When Radix-based Booth Encoding multiplier is used, it has the most compact area and this saves a lot of space and makes the performance faster when less number of logic gates is used.

4.1.2 Results of Delay Comparison among 32-bits Multiplier Designs

In the analysis among the Radix-based Booth Encoding multiplier designs, Speed-Optimized mode activated in the Leonardo Spectrum gives the best-case delay as shown in Table 4.3.

Table 4.3: Delay comparison of Radix-based multipliers in Area-Optimized, Speed-Optimized and Auto-Optimized.

Area-Optimized	
Radix-2 Booth Encoding multiplier	29.09ns
Radix-4 Booth Encoding multiplier	12.27ns
Radix-8 Booth Encoding multiplier	14.02ns
Radix-16 Booth Encoding multiplier	15.66ns
Radix-32 Booth Encoding multiplier	16.59ns
Speed-Optimized	
Radix-2 Booth Encoding multiplier	29.76ns
Radix-4 Booth Encoding multiplier	11.41ns
Radix-8 Booth Encoding multiplier	12.50ns
Radix-16 Booth Encoding multiplier	14.36ns
Radix-32 Booth Encoding multiplier	15.47ns
Auto-Optimized	
Radix-2 Booth Encoding multiplier	29.09ns
Radix-4 Booth Encoding multiplier	12.27ns
Radix-8 Booth Encoding multiplier	14.02ns
Radix-16 Booth Encoding multiplier	15.66ns
Radix-32 Booth Encoding multiplier	16.11ns

In the Speed-Optimized mode, a decreased of 18.35ns in the delay was seen during the transaction between Radix-2 and Radix-4 Booth Encoding designs when the multipliers were synthesized. The delay of the synthesized designs then slightly increased by 1.09ns from Radix-4 to Radix-8, experienced a hike up of 1.86ns from Radix-8 to Radix-16 and a raised of 1.11ns from Radix-16 to Radix-32 Booth Encoding multiplier. Figure 4.7 shows the delay changes in the Speed-Optimized mode.

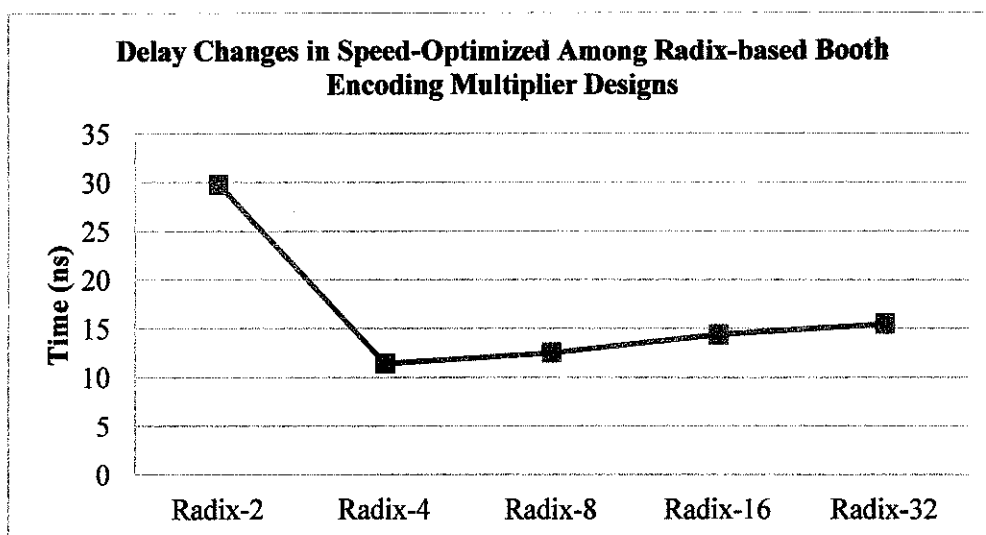


Figure 4.7: Delay changes summarized in the Speed-Optimized mode.

When the Area-Optimized mode was used, the parameters obtained from Table 4.3 are slightly different from the findings received when the designs were synthesized in the Speed-Optimized mode. It started off with a delay of 29.09ns in Radix-2 design and decreased by 16.82ns in the Radix-4 Booth Encoding multiplier. An increase of 1.75ns can be seen in the Radix-8 Booth Encoding multiplier and a continuous increment by 1.64ns in the Radix-16 designs and finally a hike of 0.93ns in the delay of the Radix-32 multiplier. The delay findings for the Area-Optimized mode were plotted in Figure 4.8 below.

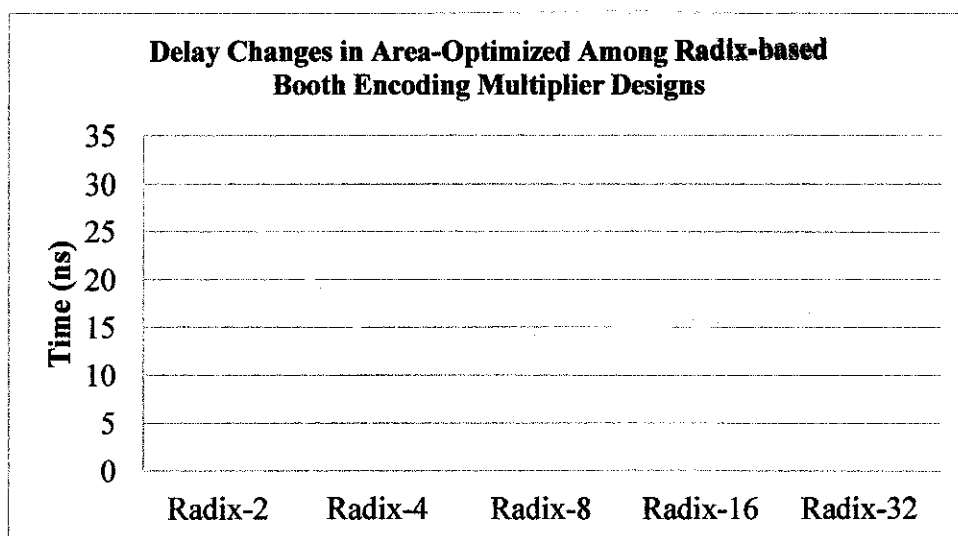


Figure 4.8: Delay changes as summarized in the Area-Optimized mode.

The synthesized findings in Auto-Optimization once again display a similar performance with the Area-Optimized mode. The only difference between both of the modes can be encountered in the Radix-32 Booth Encoding multiplier design where a decreased of 0.48ns was detected in the Radix-32 Auto-Optimized mode during the synthesis. The delay in the critical path became shorter in the higher Radix-based Booth Encoding multipliers compared to the Radix-2 Booth Encoding multiplier. Hence this proved that by using Booth Encoding technique, we are able to reduce the delay in the critical path tremendously. Figure 4.9 draws the changes seen in the delay performance in the Auto-Optimized mode.

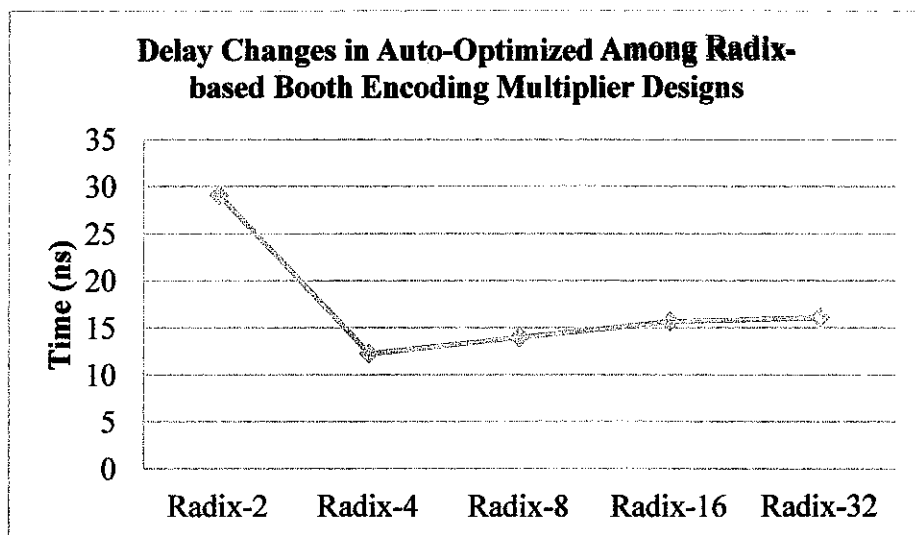


Figure 4.9: Delay changes as shown in the Auto-Optimized mode.

In my opinion, Radix-4 Booth Encoding multiplier can be used in high speed calculations because it produced the fastest performance among the other Radix-based designs. Radix-8, Radix-16 and Radix-32 multipliers are better compared to Radix-2 Booth Encoding multiplier in terms of speed but as the number of radix goes higher, the timing will increase due to the larger number of bits as determined by C_k in the multiplication process to produce a lower number of partial products.

The results and findings were further compared between the area and delay parameters by computing AD and AD^2 using the values obtained from the previous section. Here, A represents the Area and D represents the Delay. The AD and AD^2 parameters computed

will be a great help in assisting the selection of the right applications of the multiplier designs in terms of a particular high speed or a limited area. Table 4.4 below shows the parameter of AD and AD² tabulated for the Radix-based Booth Encoding multipliers.

Table 4.4: AD and AD² computed for all the Radix-based Booth Encoding designs in all three modes.

Area-Optimized					
	Radix-2 Booth Encoding multiplier	Radix-4 Booth Encoding multiplier	Radix-8 Booth Encoding multiplier	Radix-16 Booth Encoding multiplier	Radix-32 Booth Encoding multiplier
A	2,634	1,450	2,126	4,234	7,539
D	29.09	12.27	14.02	15.66	16.59
AD	76,623.06	17,791.50	29,806.52	66,304.44	125,072.01
AD ²	2,228,964.82	218,301.71	417,887.41	1,038,327.53	2,074,944.65
Speed-Optimized					
	Radix-2 Booth Encoding multiplier	Radix-4 Booth Encoding multiplier	Radix-8 Booth Encoding multiplier	Radix-16 Booth Encoding multiplier	Radix-32 Booth Encoding multiplier
A	4,261	2,121	3,163	6,368	11,949
D	29.76	11.41	12.50	14.36	15.47
AD	126,807.36	24,200.61	39,537.50	91,444.48	184,851.03
AD ²	3,773,787.03	276,128.96	493,750	1,313,142.73	2,859,645.43
Auto-Optimized					
	Radix-2 Booth Encoding multiplier	Radix-4 Booth Encoding multiplier	Radix-8 Booth Encoding multiplier	Radix-16 Booth Encoding multiplier	Radix-32 Booth Encoding multiplier
A	2,634	1,450	2,162	4,232	7,468
D	29.09	12.27	14.02	15.66	16.11
AD	76,623.06	17,791.50	30,311.24	66,273.12	120,309.48
AD ²	2,228,964.82	218,301.71	424,963.59	1,037,837.06	1,938,185.72

In the result comparison among Array, Dadda, Wallace, Reduced-Area and Radix-4 Booth Encoding multiplier, the best-case delay performance in terms of ns is given by Speed-Optimized mode as seen in Table 4.5.

Table 4.5: Delay performance of 32-bits multipliers in Area-Optimized, Speed-Optimized and Auto-Optimized modes.

Area-Optimized	
Array multiplier	20.64ns
Wallace multiplier	11.60ns
Dadda multiplier	10.97ns
Reduced-Area multiplier	12.03ns
Radix-4 Booth Encoding multiplier	12.27ns
Speed-Optimized	
Array multiplier	19.14ns
Wallace multiplier	9.82ns
Dadda multiplier	10.47ns
Reduced-Area multiplier	10.54ns
Radix-4 Booth Encoding multiplier	11.41ns
Auto-Optimized	
Array multiplier	22.55ns
Wallace multiplier	11.67ns
Dadda multiplier	11.50ns
Reduced-Area multiplier	11.80ns
Radix-4 Booth Encoding multiplier	12.27ns

When the synthesis process is in the Speed-Optimized mode, the best delay happens in Wallace multiplier with 9.82ns. It follows by an increment of 6.62% for the Dadda multiplier and continued with an increase in delay of 0.67% in the Reduced-Area design. Next, the delay of the synthesized designs then increased about 8.25% for Radix-4 Booth Encoding multiplier. The performance continued to hike up by 67.75% which gives us the longest delay time and this is shown by Array multiplier. From Table 4.5, we can conclude that Wallace multiplier produces the fastest time delay while Array multiplier shows the longest time in the performance delay. Figure 4.10 summarizes the delay changes when the synthesis is in the Speed-Optimization mode.

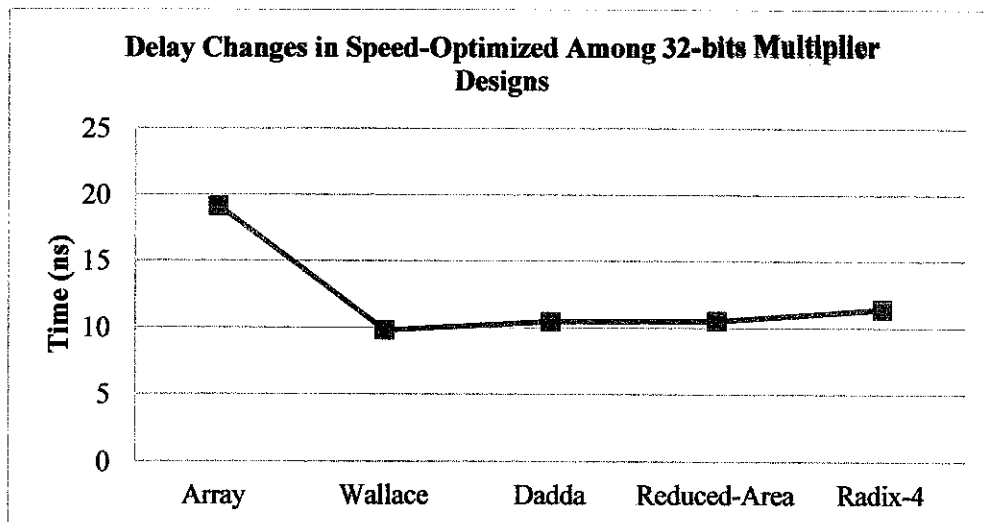


Figure 4.10: Delay changes summarized in the Speed-Optimized mode.

When the designs were synthesized in the Area-Optimization mode for the delay performance among Array, Dadda, Wallace, Reduced-Area and Radix-4 Booth Encoding multiplier designs, there are slight changes in the findings analyzed. The shortest delay was found to be Dadda multiplier instead of Wallace multiplier as given in the Speed-Optimized mode as mentioned previously. The parameter then hikes up to 5.74% as shown by Wallace multiplier and continued to show an increment of 3.71% in the Reduced-Area multiplier. A continuous increase of 2% in the delay performance of the Radix-4 booth encoding design is seen as shown in Table 4.5 and finally experienced a rise of 68.22% for Array multiplier which produced the longest delay among all the multiplier designs. The delay findings for the Area-Optimized mode are summarized in Figure 4.11 as shown below.

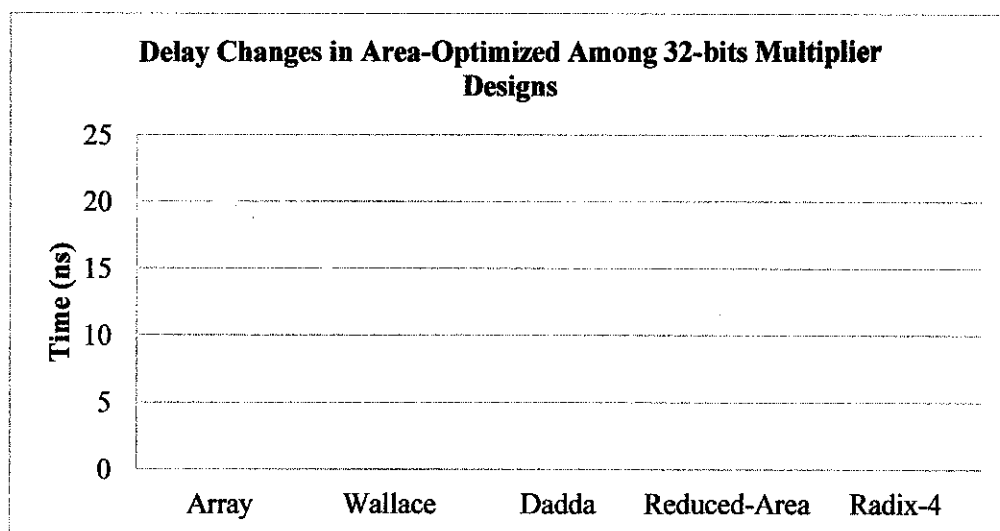


Figure 4.11: Delay changes summarized in the Area-Optimized mode.

The Auto-Optimized mode here once again displays a close performance analysis with the Area-Optimized mode. The only huge difference can be seen based on the results as displayed in Table 4.5 was that the Wallace multiplier delay values differ from each which displayed an increase of 4.61% in the Auto-Optimized mode compared to the Area-Optimized mode. From Table 4.5, we can conclude that the delay performance in the Area-Optimized and Auto-Optimized mode is different from the designs synthesized using the Speed-Optimized mode. In the Area-Optimized and Auto-Optimized mode, it was known that Array multiplier experienced the longest time delay performance while Dadda multiplier exhibits the shortest time delay in terms of speed. However, Speed-

Optimized mode displayed that Array multiplier has the longest delay while the fastest in speed is given by Wallace multiplier. Figure 4.12 draws the changes seen in the delay performance in the Auto-Optimized mode.

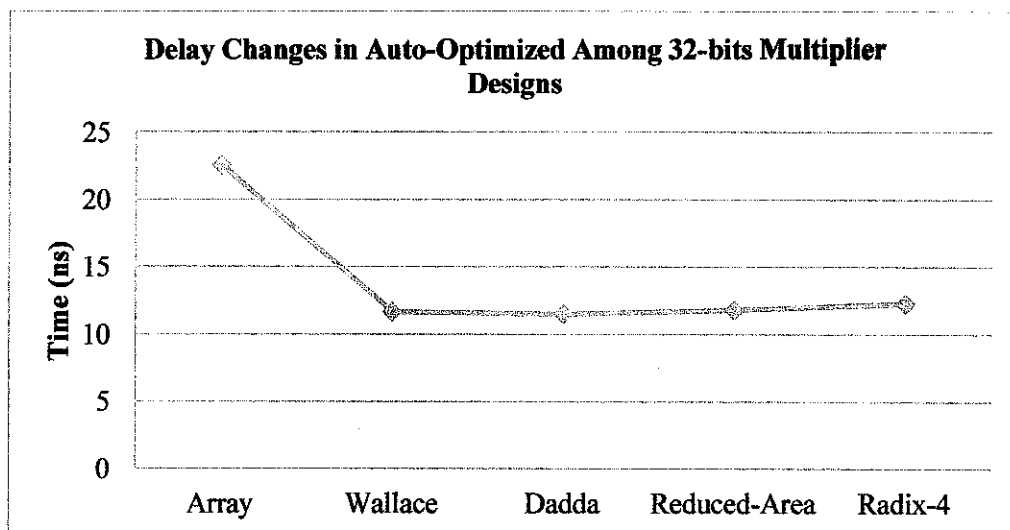


Figure 4.12: Delay changes summarized in the Auto-Optimized mode.

When the five multiplier designs are compared, Radix-4 Booth Encoding multiplier no longer is the fastest multiplier. Since the Speed-Optimized mode produced a smaller timing delay, Wallace multiplier is the best multiplier under the timing constraint among the five multipliers studied.

The results obtained were then computed to obtain values for AD and AD^2 where A represents the Area and D represents the Delay parameters analyzed from the previous section. The parameters of AD and AD^2 will be a good contribution in the selection of the suitable applications of the multiplier designs in terms of a high speed or limited area performance. Table 4.6 displayed the parameter computed for AD and AD^2 based on the five types of 32-bits multiplier designs mentioned in this paper.

Table 4.6: AD and AD² computed for the 32-bits multiplier designs

Area Optimized					
	Array multiplier	Wallace multiplier	Dadda multiplier	Reduced-Area multiplier	Radix-4 Booth Encoding multiplier
A	1,744	1,905	1,782	1,802	1,450
D	20.64	11.60	10.97	12.03	12.27
AD	35,996.16	22,098	19,548.54	21,678.06	17,791.50
AD ²	742,960.74	256,336.80	214,447.48	260,787.06	218,301.71
Speed Optimized					
	Array multiplier	Wallace multiplier	Dadda multiplier	Reduced-Area multiplier	Radix-4 Booth Encoding multiplier
A	2,855	2,732	3,014	2,780	2,121
D	19.14	9.82	10.47	10.54	11.41
AD	54,644.70	26,828.24	31,556.58	29,301.20	24,200.61
AD ²	1,045,899.56	263,453.32	330,397.39	308,834.65	27,6128.96
Auto Optimized					
	Array multiplier	Wallace multiplier	Dadda multiplier	Reduced-Area multiplier	Radix-4 Booth Encoding multiplier
A	1,738	1,840	1,738	1,739	1,450
D	22.55	11.67	11.50	11.80	12.27
AD	39,191.90	21,472.80	19,987	20,520.20	17,791.50
AD ²	883,777.35	250,587.58	229,850.50	242,138.36	218,301.71

4. Discussions

As the higher radix number of the Radix-based Booth Encoding multiplier, the partial products get more complicated. This is because the number of bits uses to determine the constant multiplied with the multiplicand becomes more complex as the higher the radix number of the Radix-based multipliers reached. The complexity of the number of partial products make the area of the Radix-8, Radix-16 and Radix-32 Booth Encoding multipliers bigger compared to Radix-4 Booth Encoding multiplier. Radix-4 Booth Encoding has the best performance either in terms of area or timing with the smallest area and fastest speed among the Radix-based Booth Encoding multiplier designs.

The number of partial products reduced significantly as the higher the Radix-based multipliers are. Radix-4 Booth Encoding multiplier, the number of partial product is decreased to half compared to the normal multiplier while with Radix-8 Booth Encoding, the number of partial product generated is one third compared to the basic multiplier. Radix-16 Booth Encoding, the number of partial product computed is one fourth compared with the normal multiplier. The number of partial product for Radix-32 Booth

Encoding multiplier reduces to one fifth compared to the normal multiplier. With this condition, the speed of the multipliers is assumed to be reduced significantly as the higher the Radix-based Booth Encoding multipliers become. That is the reason why the timing performance as shown in Radix-4, Radix-8, Radix-16 and Radix-32 Booth Encoding multipliers are faster compared to Radix-2 Booth Encoding when the designs are synthesized in Leonardo Spectrum to get the performance parameters.

The algorithm of the Radix-based Booth Encoding multipliers as discussed and designed in this project are the same for Radix-2, Radix-4, Radix-8, Radix-16 and Radix-32 Booth Encoding multipliers but the difference between each of the multipliers is the number of bits in a collection block as represented by C_k and S_k with one bit overlapping each other. Radix-2 uses 2-bits; Radix-4 uses 3-bits; radix-8 uses 4-bits; radix-16 uses 5-bits and radix-32 uses 6-bits when the encoding process is executed.

From the findings obtained, Radix-4 Booth Encoding multiplier is the best area-saving multiplier among the Radix-based Booth Encoding multipliers. This is suitable for complex calculations that require less number of logic gates and small area. The area of the multiplier designs increased in Radix-8, Radix-16 and Radix-32 due to the larger amount of logic gates needed to compute the partial product. If the area performance is compared among five 32-bits multiplier designs, Radix-4 Booth Encoding still produced the smallest area performance. So, Radix-4 Booth Encoding multiplier is the best radix multiplier and it is suitable to use in complex calculations but requires a small area performance.

In terms of timing performance, Radix-4 Booth Encoding multiplier has the fastest timing compared to the other number of radix multipliers. Among the Radix-based Booth Encoding multipliers, Radix-4 is the most suitable multiplier used for high speed in complex calculations. However, if the Radix-4 Booth Encoding multiplier designs were compared among Array, Wallace, Dadda and Reduced-Area, we found out that Wallace multiplier gives the fastest timing.

CHAPTER 5

5. CONCLUSION

Five types of Radix-based Booth Encoding multiplier consist of Radix-2, Radix-4, Radix-8, Radix-16 and Radix-32 Booth Encoding multipliers were designed using Verilog HDL, simulated in Modelsim and synthesized to compare the performance in terms of gate level logic and delay using 0.35-micron ASIC Design Kit standard cell library in Leonardo Spectrum using the Area-Optimized, Speed-Optimized and Auto-Optimized mode to determine the area and timing performance.

From the results computed, it is known that Radix-4 Booth Encoding multiplier has the best speed advantage among the 32-bits Radix-based Booth Encoding designs studied here and it is the best-suited for high speed applications that requires complex calculations. The other higher Radix-based Booth Encoding designs such as Radix-8, Radix-16 and Radix-32 shows a delay smaller than the Radix-2 but the delay increased when compared with the Radix-4 Booth Encoding design.

The gate level logic performance according to the synthesis results of the Radix-4 multiplier again has the lowest area constraints compared to Radix-2 Booth Encoding multiplier. However the gate level logic of Radix-8, Radix-16 and Radix-32 designs increased compared to Radix-4 Booth Encoding designs. Radix-4 and Radix-8 multiplier are the designs that show a lower area performance compared to the Radix-2 Booth Encoding multiplier. The multiplier designs of Array, Wallace, Dadda, Reduced-Area and Radix-4 Booth Encoding multipliers were designed in Verilog HDL, verified by simulation in Modelsim and synthesized in 0.35-micron ASIC Design Kit standard cell library in Leonardo Spectrum to obtain the performance in terms of gate level logic and delay.

The project was further developed where a further comparison was done among Array, Dadda, Wallace, Reduced-Area and Radix-4 Booth Encoding multipliers. In the Area-Optimized mode, Radix-4 Booth Encoding multiplier has the lowest gate level logic value in all three of the Area-Optimized, Speed-Optimized and Auto-Optimized mode.

However, there is a difference in the findings on which type of multiplier gives the largest gate level logic performance. Wallace multiplier exhibited the largest area in the Area-Optimized and Auto-Optimized modes while Dadda multiplier displays the lowest area performance in the Speed-Optimized mode.

Based on the results analyzed, Array multiplier has the longest delay performance in the Area-Optimized and Auto-Optimized modes and Dadda multiplier shows the shortest delay in terms of speed performance. However, when Speed-Optimized is used, Array multiplier produced the longest delay and Wallace multiplier is the fastest in speed.

REFERENCE

5.1 Main references

- [1] Behrooz Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford New York, Oxford University Press, New York, 2000.
- [2] William Stallings, Computer Organization and Architecture: Designing for performance, Fifth Edition, New Jersey: Prentice-Hall, Inc, 2000.
- [3] Booth Recoding, 27 July 2010, <<http://www.geoffknagge.com/fyp/booth.shtml>>
- [4] Booth's Multiplication Algorithm, 30 May 2010, <http://en.wikipedia.org/wiki/Booth's_multiplication_algorithm>
- [5] Radix-4 Booth Encoding, 2 January 2007, <<http://www.russinoff.com/libman/text/node35.html>>
- [6] Vivek Sagdeo, 1998, The Complete Verilog Book, Kluwer Academic Publishers, Boston.
- [7] Morris Mano and Michael Ciletti Digital Design, Pearson Education, Fourth Edition, New Jersey, 2006.

5.2 Other references

- [8] Weste, Neil H.E. and Eshraghian, Kamran, Principles of CMOS VLSI Design: A systems perspective, Addison-Wesley Publishing Company, Second Edition, 1993.
- [9] Samar Palnitkar, Verilog HDL: A Guide to Digital and Synthesis, SunSoft Press A Prentice Hall Title, 1996.
- [10] Thomas & Moorby's, The Verilog Hardware Description Language, Fourth Edition, Kluwer Academic Publishers, Boston, 1999.

5.3 Journal and Thesis

- [11] Mi Lu, Modular Structure of Large Multiplier in Arithmetic and Logic in Computer Systems, First Edition, New Jersey: John Wiley & Sons, Inc.
- [12] C.S. Wallace, A Suggestion for a Fast Multiplier, IEEE Transaction on Electronic Computers, 1964.

- [13] W.J. Townsend, E.E. Swartzlander, Jr. and J.A. Abraham, A Comparison of Dadda and Wallace Multiplier Delays, in SPIE Advance Signal Processing Algorithms, Architectures and Implementations XIII, 2003.
- [14] L. Dadda, Some Schemes for Parallel Multipliers, *Alta Frequenza*, 1965.
- [15] K.A.C. Bickerstaff, M. Schulte, and E.E. Swartzlander, Jr., Reduced Area Multipliers, International Conference on Application-Specific Array Processors, 1993.
- [16] S.Shah, A. J. Al-Khabb and D. Al-Khabb, Department of Electrical and Computer Engineering Concordia University, Montreal, Quebec, Comparison of 32-bit Multipliers for Various Performance Measures, The 12th International Conference on Microelectronics, 2000.
- [17] Ki-seon Cho, Jong-on Park, Jin-seok Hong and Goang-seog Choi, Storage Solution Group, DM R/D Center Samsung Electronics Co., Ltd., Korea, 54x54-bit Radix-4 Multiplier based on Modified Booth Algorithm, GLSVLSI'03, 2003.
- [18] Chris Y.H. Lee, Lo Hai Hiung, Sean W.F. Lee, Nor Hisham Hamid, Universiti Teknologi PETRONAS and Emerald Systems Sdn. Bhd., A Performance Comparison Study on Multiplier Designs, 2010.
- [19] Peter-Michael Seidal, Lee D McFearin and David W Matula, Department of Computer Science and Engineering, Southern Methodist University, Dallas, Texas, Binary Multiplication of Radix-32 and Radix-256, IEEE, 2001.
- [20] Ahmed Elhossini, Ali Rashid and Mohammed K. Refai, Al-Azhar Engineering Univeristy, A 16x16 Bit Modified Radix-16 Booth Encoded Parallel Multiplier, AEIC Eight International Conference, 2004.

APPENDICES

APPENDIX I: Verilog HDL codes on Radix-based Booth Encoding multipliers

```
/**
//          FINAL YEAR PROJECT          //
//          RADIX-2 BOOTH ENCODING MULTIPLIER          //
//          Designed by: Kelly Liew          //
//          32-bit multiplier          //
**/
module radix2(multiplicand,multiplier,product);

parameter width=16;

input [width-1:0] multiplicand;
input [width-1:0] multiplier;
output [width+width-1:0] product;

reg [width+width-1:0] product;
reg [width+width:0] PP,right;

integer i;

always @(multiplicand or multiplier)
begin
//Determine the base multiplicand value to be evaluated
PP [width+width:0]={ 16'b0,multiplicand,1'b0};

//Evaluate the 2-bit multiplicand
//Depend on the 3-bit multiplicand, (1,-1 or 0) x multiplicand
//to obtain the partial product
for (i=0; i<width; i=i+1)
begin
case (PP[1:0])

//When 10, partial product=-1xmultiplicand
//Then, shift right
2'b10:
begin
PP[width+width:width+1]=PP[width+width:width+1]-multiplier[width-1:0];
rightshift(PP,right);
end

//When 01, partial product=1xmultiplicand
//Then shift right
2'b01:
begin
PP[width+width:width+1]=PP[width+width:width+1]+multiplier[width-1:0];
rightshift(PP,right);
end

//When 00 or 11, partial product=0
//Then shift right
default:
rightshift(PP,right);

endcase

PP = right;
end
end
```

```

end

//The sum of the partial product produced
product[width+width-1:0] = PP[width+width:1];
end

//Right shift is being done here
task rightshift;
input [width+width:0] PP;
output [width+width:0] right;

//Shift by 1-bit
case (PP[width+width])
  1'b0 : right[width+width:0]={1'b0, PP[width+width:1]};
  1'b1 : right[width+width:0]={1'b1, PP[width+width:1]};
endcase

endtask

endmodule

//*****//
//          FINAL YEAR PROJECT          //
//          RADIX-4 BOOTH ENCODING MULTIPLIER //
//          Designed by: Kelly Liew      //
//          32-bit multiplier            //
//*****//
`define width 16
module radix4(multiplicand, multiplier, product);

parameter width = `width;
parameter N = `width/2;

input [width-1:0] multiplicand;
input [width-1:0] multiplier;
output [width+width-1:0] product;

reg [2:0] BP[N-1:0];
reg [width:0] PP[N-1:0];
reg [width+width-1:0] sign_p[N-1:0];
reg [width+width-1:0] P;

wire [width:0] neg_multiplicand;

integer i, j;

//Two's compliment for multiplicand
assign neg_multiplicand = {~multiplicand[width-1], ~multiplicand} + 1;

always @(multiplicand or multiplier or neg_multiplicand)
begin
//Generate the 3-bit multiplicand when it is 0
BP[0] = {multiplier[1], multiplier[0], 1'b0};
//Generate the 3-bit multiplicand when it is not 0
for (i=1; i<N; i = i+1)
  BP[i] = {multiplier[2*i+1], multiplier[2*i], multiplier[2*i-1]};

//Depend on the 3-bit multiplicand, (1,2,-1,-2 or 0) x multiplicand
//to obtain the partial product
for (i=0; i<N; i=i+1)
begin
case(BP[i])

```

```

//When 001 or 010, partial product=1xmultiPLICand
//3'b001, 3'b010 : PP[i] = {multiplicand[width-1],multiplicand};
3'b001, 3'b010 : PP[i] = {1*multiplicand};
//When 011 or 100, partial product=2xmultiPLICand
//3'b011 : PP[i] = {multiplicand, 1'b0};
3'b011 : PP[i] = {2*multiplicand};
//When 100, partial product=-2xmultiPLICand
//3'b100 : PP[i] = {neg_multiplicand[width-1:0], 1'b0};
3'b100 : PP[i] = {2*neg_multiplicand};
//When 101 or 110, partial product=-1xmultiPLICand
3'b101, 3'b110 : PP[i] = neg_multiplicand;
//When 000 or 111, partial product=0
default : PP[i] = 0;
endcase

//Sign extended
sign_p[i] = $signed(PP[i]);

//Shifting by 2-bit is done here
for (j=0; j<i; j=j+1)
    sign_p[i] = {sign_p[i], 2'b00};

end

//Partial product added together to get the final product value
P = sign_p[0];

for (i=1; i<N; i=i+1)
    P = P + sign_p[i];

end

assign product = P;

endmodule

/*****//
//                               FINAL YEAR PROJECT                               //
//                               RADIX-8 BOOTH ENCODING MULTIPLIER                               //
//                               Designed by: Kelly Liew                               //
//                               32-bit multiplier                               //
*****/

`define width 16
module radix8(multiplicand, multiplier, product);

    parameter width = `width;
    parameter N = `width/3;

    input [width-1:0] multiplicand;
    input [width-1:0] multiplier;
    output [width+width-1:0] product;

    reg [3:0] BP[N-1:0];
    reg [width:0] PP[N-1:0];
    reg [width+width-1:0] sign_p[N-1:0];
    reg [width+width-1:0] P;

    wire [width:0] neg_multiplicand;

    integer i, j;

```



```

//Two's compliment for multiplicand
assign neg_multiplicand = {~multiplicand[width-2], ~multiplicand[width-1], ~multiplicand} + 1;

always @(multiplicand or multiplier or neg_multiplicand)
begin
//Generate the 4-bit multiplicand when it is 0
BP[0] = {multiplier[2], multiplier[1], multiplier[0], 1'b0};
//Generate the 4-bit multiplicand when it is not 0
for (i=1; i<N; i = i+1)
BP[i] = {multiplier[3*i+2], multiplier[3*i+1], multiplier[3*i], multiplier[3*i-1]};
$display("The BP[%d] is equals to %b", i, BP[i]);
//Depend on the 4-bit multiplicand, (1,2,3,4,-1,-2,-3,-4 or 0) x multiplicand
//to obtain the partial product
for (i=0; i<N; i=i+1)
begin
case(BP[i])
//When 0001 or 0010, partial product=1xmultiplicand
4'b0001, 4'b0010 : PP[i] = {1*multiplicand};
//When 0011 or 0100, partial product=2xmultiplicand
4'b0011, 4'b0100 : PP[i] = {2*multiplicand};
//When 0101 or 0110, partial product=3xmultiplicand
4'b0101, 4'b0110 : PP[i] = {3*multiplicand};
//When 0111, partial product=4xmultiplicand
4'b0111 : PP[i] = {4*multiplicand};
//When 1000, partial product=-4xmultiplicand
//4'b1000 : PP[i] = {neg_multiplicand[width-1:0], 2'b00};
4'b1000 : PP[i] = {4*neg_multiplicand};
//When 1001 or 1010, partial product=-3xmultiplicand
4'b1001, 4'b1010 : PP[i] = {3*neg_multiplicand};
//When 1011 or 1100, partial product=-2xmultiplicand
4'b1011, 4'b1100 : PP[i] = {2*neg_multiplicand};
//When 1101 or 1110, partial product=-1xmultiplicand
4'b1101, 4'b1110 : PP[i] = neg_multiplicand;
//When 0000 or 1111, partial product=0
default : PP[i] = 0;
endcase

//Sign extended
sign_p[i] = $signed(PP[i]);

//Shifting by 3-bit is done here
for (j=0; j<i; j=j+1)
sign_p[j] = {sign_p[j], 3'b000};

end

//Partial product added together to get the final product value
P = sign_p[0];

for (i=1; i<N; i=i+1)
P = P + sign_p[i];

end

assign product = P;

endmodule

```

```

/******//
//          FINAL YEAR PROJECT          //
//          RADIX-16 BOOTH ENCODING MULTIPLIER          //
//          Designed by: Kelly Liew          //
//          32-bit multiplier          //
/******//

`define width 16
module radix16(multiplicand, multiplier, product);

    parameter width = `width;
    parameter N = `width/4;

    input [width-1:0] multiplicand;
    input [width-1:0] multiplier;
    output [width+width-1:0] product;

    reg [4:0] BP[N-1:0];
    reg [width:0] PP[N-1:0];
    reg [width+width-1:0] sign_p[N-1:0];
    reg [width+width-1:0] P;

    wire [width:0] neg_multiplicand;

    integer i, j;

    //Two's compliment for multiplicand
    assign neg_multiplicand = {~multiplicand[width-3], ~multiplicand[width-2], ~multiplicand[width-1], ~multiplicand}
        + 1;

    always @(multiplicand or multiplier or neg_multiplicand)
    begin
        //Generate the 5-bit multiplicand when it is 0
        BP[0] = {multiplier[3], multiplier[2], multiplier[1], multiplier[0], 1'b0};
        //Generate the 5-bit multiplicand when it is not 0
        for (i=1; i<N; i = i+1)
            BP[i] = {multiplier[4*i+3], multiplier[4*i+2], multiplier[4*i+1], multiplier[4*i], multiplier[4*i-1]};
            //Depend on the 5-bit multiplicand, (1,2,3,4,5,6,7,8,-1,-2,-3,-4,-5,-6,-7,-8 or 0) x multiplicand
            //to obtain the partial product
            for (i=0; i<N; i=i+1)
                begin
                    case(BP[i])
                        //When 00001 or 00010, partial product=1xmultiplicand
                        5'b00001, 5'b00010 : PP[i] = {1*multiplicand};
                        //When 00011 or 00100, partial product=2xmultiplicand
                        5'b00011, 5'b00100 : PP[i] = {2*multiplicand};
                        //When 00101 or 00110, partial product=3xmultiplicand
                        5'b00101, 5'b00110 : PP[i] = {3*multiplicand};
                        //When 00111 or 01000, partial product=4xmultiplicand
                        5'b00111, 5'b01000 : PP[i] = {4*multiplicand};
                        //When 01001 or 01010, partial product=5xmultiplicand
                        5'b01001, 5'b01010 : PP[i] = {5*multiplicand};
                        //When 01011 or 01100, partial product=6xmultiplicand
                        5'b01011, 5'b01100 : PP[i] = {6*multiplicand};
                        //When 01101 or 01110, partial product=7xmultiplicand
                        5'b01101, 5'b01110 : PP[i] = {7*multiplicand};
                        //When 01111, partial product=8xmultiplicand
                        5'b01111 : PP[i] = {8*multiplicand};
                        //When 10000, partial product=-8xmultiplicand
                        5'b10000 : PP[i] = {8*neg_multiplicand};
                        //When 10001 or 10010, partial product=-7xmultiplicand
                        5'b10001, 5'b10010 : PP[i] = {7*neg_multiplicand};
                    endcase
                end
    end
endmodule

```

```

//When 10011 or 10100, partial product=-6xmultiplicand
5'b10011, 5'b10100 : PP[i] = {6*neg_multiplicand};
//When 10101 or 10110, partial product=-5xmultiplicand
5'b10101, 5'b10110 : PP[i] = {5*neg_multiplicand};
//When 10111 or 11000, partial product=-4xmultiplicand
5'b10111, 5'b11000 : PP[i] = {4*neg_multiplicand};
//When 11001 or 11010, partial product=-3xmultiplicand
5'b11001, 5'b11010 : PP[i] = {3*neg_multiplicand};
//When 11011 or 11100, partial product=-2xmultiplicand
5'b11011, 5'b11100 : PP[i] = {2*neg_multiplicand};
//When 11101 or 11110, partial product=-1xmultiplicand
5'b11101, 5'b11110 : PP[i] = neg_multiplicand;
default : PP[i] = 0;
endcase

//Sign extended
sign_p[i] = $signed(PP[i]);

//Shifting by 4-bit is done here
for (j=0; j<i; j=j+1)
    sign_p[i] = {sign_p[i], 4'b0000};

end

//Partial product added together to get the final product value
P = sign_p[0];

for (i=1; i<N; i=i+1)
    P = P + sign_p[i];

end

assign product = P;

endmodule

/******//
//                               FINAL YEAR PROJECT                               //
//                               RADIX-32 BOOTH ENCODING MULTIPLIER                               //
//                               Designed by: Kelly Liew                               //
//                               32-bit multiplier                               //
/******//

`define width 16
module radix32(multiplicand, multiplier, product);

parameter width = `width;
parameter N = `width/5;

input [width-1:0] multiplicand;
input [width-1:0] multiplier;
output [width+width-1:0] product;

reg [5:0] BP[N-1:0];
reg [width:0] PP[N-1:0];
reg [width+width-1:0] sign_p[N-1:0];
reg [width+width-1:0] P;

wire [width:0] neg_multiplicand;

integer i, j;

```

```

//Two's compliment for multiplicand
assign neg_multiplicand = {~multiplicand[width-4], ~multiplicand[width-3], ~multiplicand[width-2],
    ~multiplicand[width-1], ~multiplicand} + 1;

always @(multiplicand or multiplier or neg_multiplicand)
begin
//Generate the 6-bit multiplicand when it is 0
BP[0] = {multiplier[4], multiplier[3], multiplier[2], multiplier[1], multiplier[0], 1'b0};
//Generate the 6-bit multiplicand when it is not 0
for (i=1; i<N; i = i+1)
    BP[i] = {multiplier[5*i+4], multiplier[5*i+3], multiplier[5*i+2], multiplier[5*i+1], multiplier[5*i] ,
        multiplier[5*i-1]};
//Depend on the 6-bit multiplicand,
//(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,-1,-2,-3,-4,-5,-6,-7,-8,-9,-10,-11,-12,-13,-14,-15,-16 or 0) x multiplicand
to obtain the partial product
for (i=0; i<N; i=i+1)
begin
case(BP[i])
//When 000001 or 000010, partial product=1xmultiplicand
6'b000001, 6'b000010 : PP[i] = {1*multiplicand};
//When 000011 or 000100, partial product=2xmultiplicand
6'b000011, 6'b000100 : PP[i] = {2*multiplicand};
//When 00101 or 00110, partial product=3xmultiplicand
6'b000101, 6'b000110 : PP[i] = {3*multiplicand};
//When 000111 or 001000, partial product=4xmultiplicand
6'b000111, 6'b001000 : PP[i] = {4*multiplicand};
//When 001001 or 001010, partial product=5xmultiplicand
6'b001001, 6'b001010 : PP[i] = {5*multiplicand};
//When 001011 or 001100, partial product=6xmultiplicand
6'b001011, 6'b001100 : PP[i] = {6*multiplicand};
//When 001101 or 001110, partial product=7xmultiplicand
6'b001101, 6'b001110 : PP[i] = {7*multiplicand};
//When 001111 or 010000 partial product=8xmultiplicand
6'b001111, 6'b010000 : PP[i] = {8*multiplicand};
//When 010001 or 010010 partial product=9xmultiplicand
6'b010001, 6'b010010 : PP[i] = {9*multiplicand};
//When 010011 or 010100, partial product=10xmultiplicand
6'b010011, 6'b010100 : PP[i] = {10*multiplicand};
//When 010101 or 010110, partial product=11xmultiplicand
6'b010101, 6'b010110 : PP[i] = {11*multiplicand};
//When 010111 or 011000, partial product=12xmultiplicand
6'b010111, 6'b011000 : PP[i] = {12*multiplicand};
//When 011001 or 011010, partial product=13xmultiplicand
6'b011001, 6'b011010 : PP[i] = {13*multiplicand};
//When 011011 or 011100, partial product=14xmultiplicand
6'b011011, 6'b011100 : PP[i] = {14*multiplicand};
//When 011101 or 011110, partial product=15xmultiplicand
6'b011101, 6'b011110 : PP[i] = {15*multiplicand};
//When 011111, partial product=16xmultiplicand
6'b011111 : PP[i]={16*multiplicand};
//When 100000, partial product=-16xmultiplicand
6'b100000 : PP[i] = {16*neg_multiplicand};
//When 100001 or 100010, partial product=-15xmultiplicand
6'b100001, 6'b100010 : PP[i] = {15*neg_multiplicand};
//When 100011 or 100100, partial product=-14xmultiplicand
6'b100011, 6'b100100 : PP[i] = {14*neg_multiplicand};
//When 100101 or 100110 , partial product=-13xmultiplicand
6'b100101, 6'b100110 : PP[i] = {13*neg_multiplicand};
//When 100111 or 101000, partial product=-12xmultiplicand
6'b100111, 6'b101000 : PP[i] = {12*neg_multiplicand};
//When 101001 or 101010, partial product=-11xmultiplicand
6'b101001, 6'b101010 : PP[i] = {11*neg_multiplicand};

```

```

//When 101011 or 101100, partial product=-10xmultiPLICand
6'b101011, 6'b101100 : PP[i] = {10*neg_multiplicand};
//When 101101 or 101110, partial product=-9xmultiPLICand
6'b101101, 6'b101110 : PP[i] = {9*neg_multiplicand};
//When 101111 or 110000, partial product=-8xmultiPLICand
6'b101111, 6'b110000 : PP[i] = {8*neg_multiplicand};
//When 110001 or 110010, partial product=-7xmultiPLICand
6'b110001, 6'b110010 : PP[i] = {7*neg_multiplicand};
//When 110011 or 110100, partial product=-6xmultiPLICand
6'b110011, 6'b110100 : PP[i] = {6*neg_multiplicand};
//When 110101 or 110110, partial product=-5xmultiPLICand
6'b110101, 6'b110110 : PP[i] = {5*neg_multiplicand};
//When 110111 or 111000, partial product=-4xmultiPLICand
6'b110111, 6'b111000 : PP[i] = {4*neg_multiplicand};
//When 111001 or 111010, partial product=-3xmultiPLICand
6'b111001, 6'b111010 : PP[i] = {3*neg_multiplicand};
//When 111011 or 111100, partial product=-2xmultiPLICand
6'b111011, 6'b111100 : PP[i] = {2*neg_multiplicand};
//When 111101 or 111110, partial product=-1xmultiPLICand
6'b111101, 6'b111110 : PP[i] = neg_multiplicand;
default : PP[i] = 0;
endcase

//Sign extended
sign_p[i] = $signed(PP[i]);

//Shifting by 5-bit is done here
for (j=0; j<i; j=j+1)
    sign_p[i] = {sign_p[i], 5'b00000};

end

//Partial product added together to get the final product value
P = sign_p[0];

for (i=1; i<N; i=i+1)
    P = P + sign_p[i];

end

assign product = P;

endmodule

//*****//
//          FINAL YEAR PROJECT          //
//          TESTBENCH USED IN BOOTH ENCODING MULTIPLIER          //
//          Designed by: Kelly Liew          //
//          32-bit multiplier          //
//*****//
module testbench();

    reg [15:0] multiplicand;
    reg [15:0] multiplier;

    wire [31:0] product, TestProduct;

    integer p, q;

//Depending on which type of Radix-based Booth Encoding multiplier called
radix2 testRunRadixTwo (multiplicand, multiplier, product);

```

```

//Test compare TestProduct if it is the same as Product result
assign TestProduct = multiplicand * multiplier;

//Display the multiplicand and multiplier value
initial
$monitor($time, "\t\tmultiplicand=%d \t\tmultiplier=%d \t\tproduct=%d", multiplicand,multiplier,product);

initial
begin

    $display("*****
    ***** #");
    $display("\t\tTime \t\tMultiplicand \t\tMultiplier \t\tProduct");

    $display("*****
    ***** #");
    for (p=1; p<=1; p=p+1)
        for (q=1; q<=20; q=q+1)
            #100 begin multiplicand = p; multiplier = q; end

    #100 $display("\n");

    for (p=2; p<=2; p=p+1)
        for (q=1; q<=20; q=q+1)
            #100 begin multiplicand = p; multiplier = q; end

    #100 $display("\n");

    for (p=3; p<=3; p=p+1)
        for (q=1; q<=20; q=q+1)
            #100 begin multiplicand = p; multiplier = q; end

    #100 $display("\n");

    for (p=4; p<=4; p=p+1)
        for (q=1; q<=20; q=q+1)
            #100 begin multiplicand = p; multiplier = q; end

    #100 $display("\n");

    for (p=5; p<=5; p=p+1)
        for (q=1; q<=20; q=q+1)
            #100 begin multiplicand = p; multiplier = q; end

    #100 $stop;

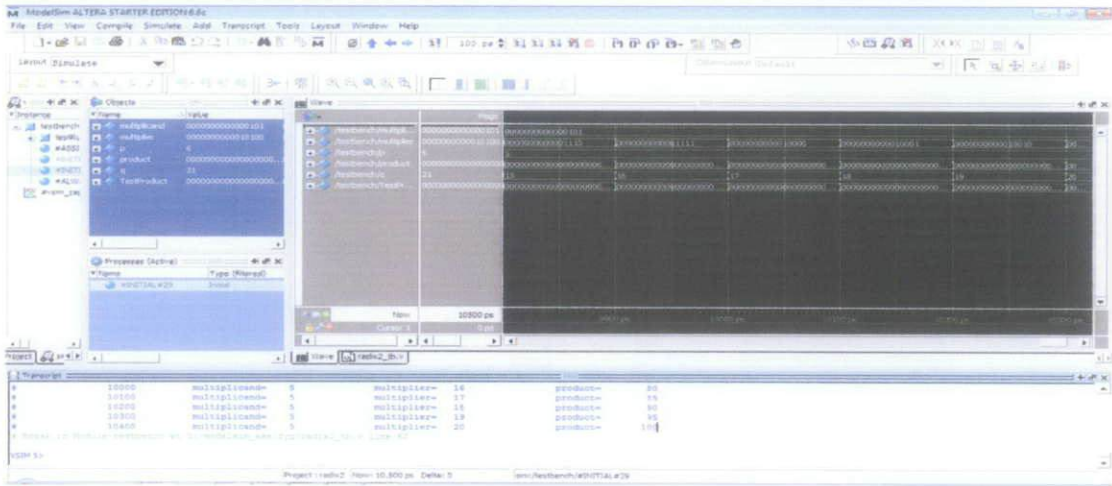
end

always @(product)
begin
    if (TestProduct != product)
        $display("Error!!! TestProduct: %d != Product: %d", product, TestProduct);
    end
endmodule

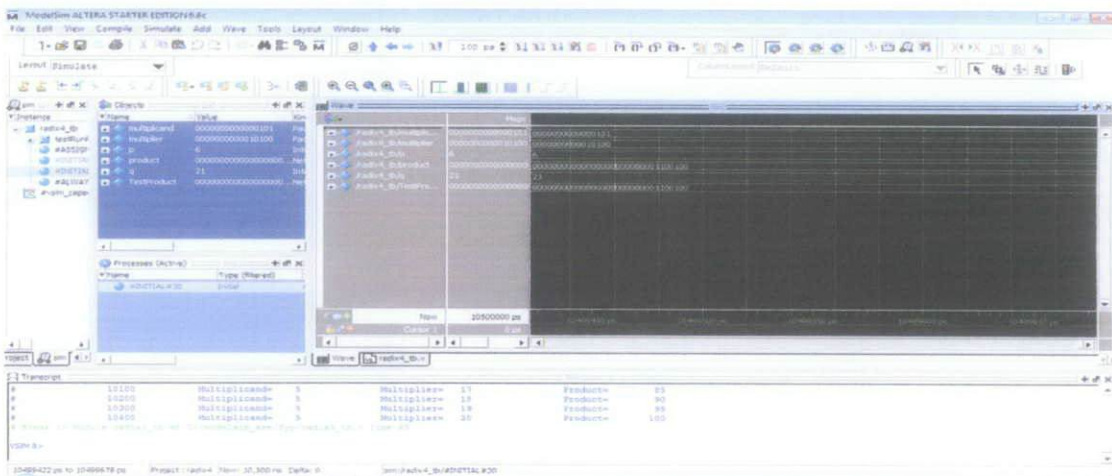
```

APPENDIX II: Logic Simulation in Modelsim

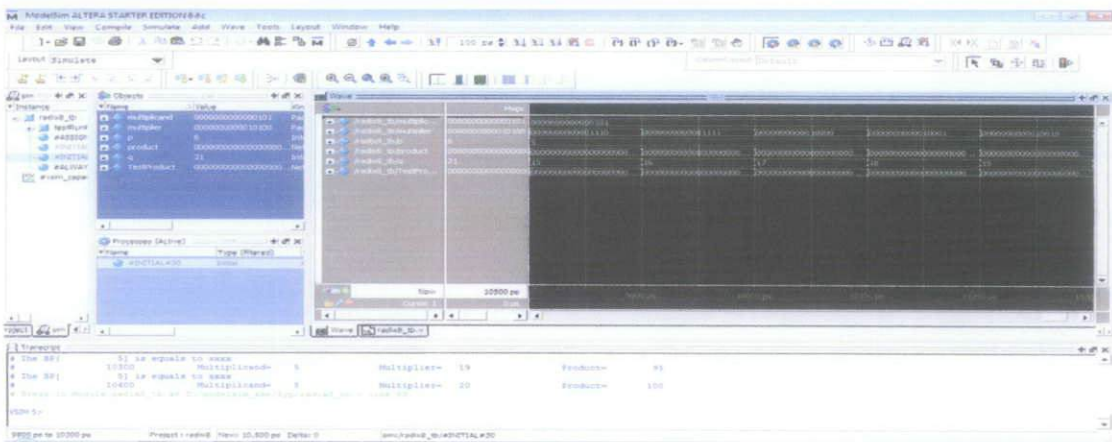
Simulation of Radix-2 Booth Encoding multiplier:



Simulation of Radix-4 Booth Encoding multiplier:

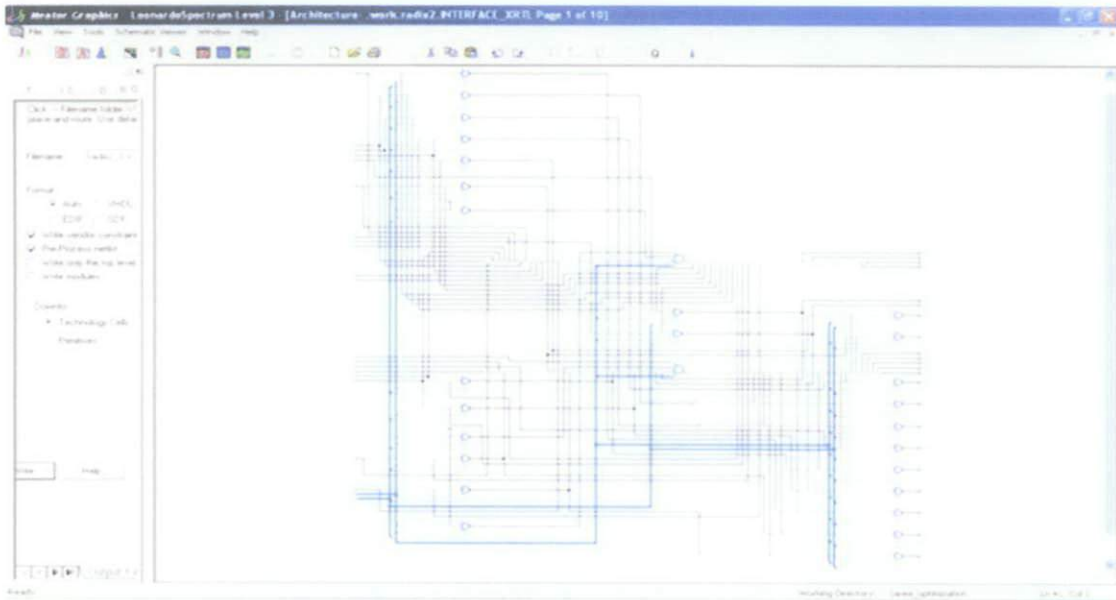


Simulation of Radix-8 Booth Encoding multiplier:

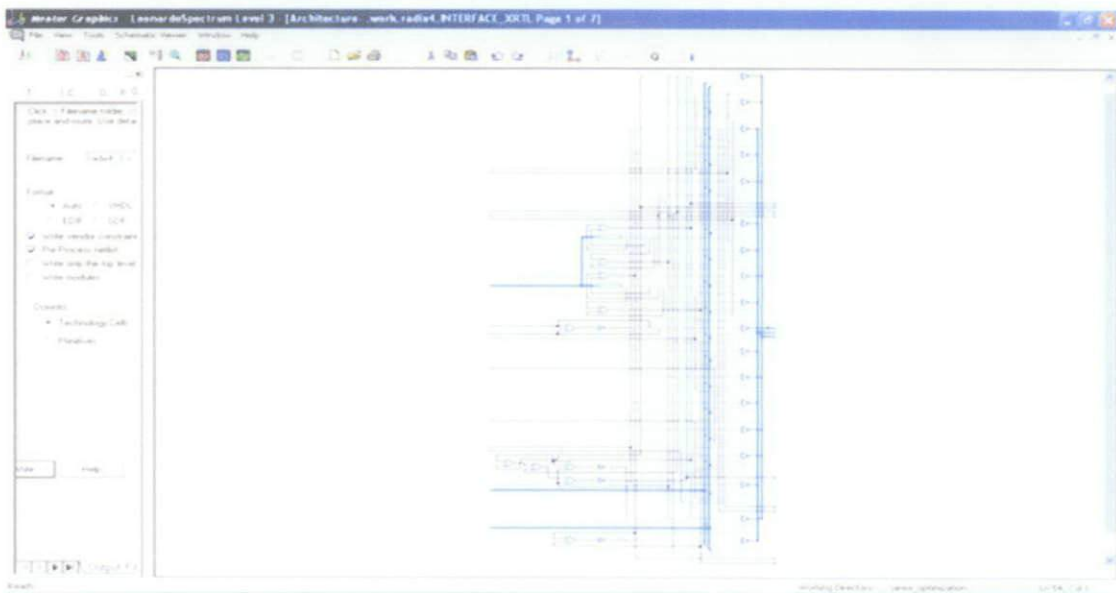


APPENDIX III: Logic Synthesis in Leonardo Spectrum

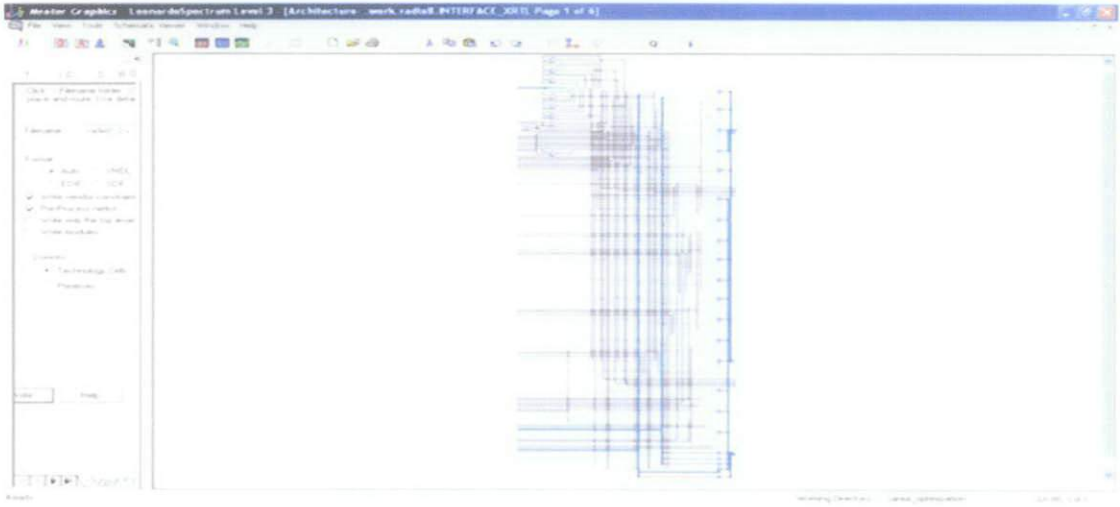
Radix-2 Booth Encoding multiplier logic gates representation:



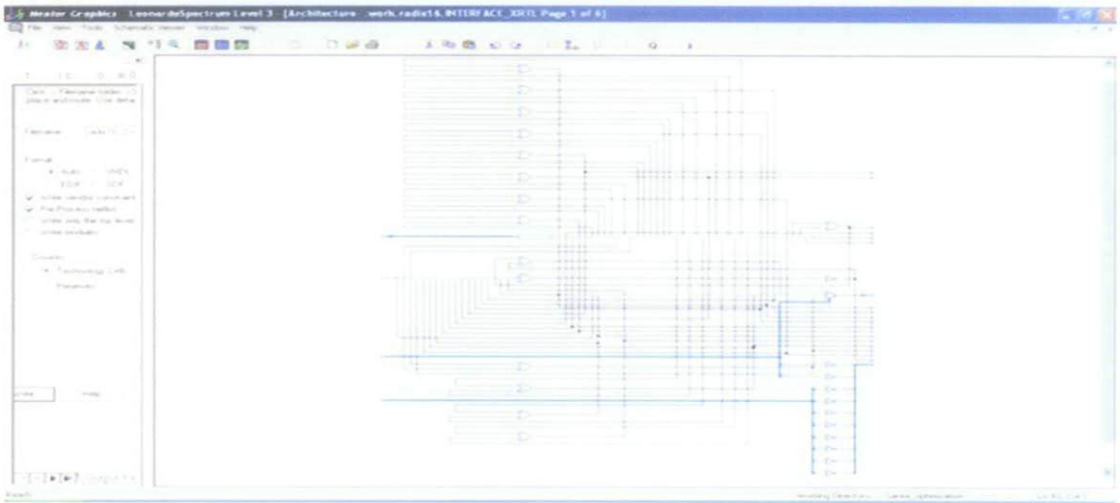
Radix-4 Booth Encoding multiplier logic gates representation:



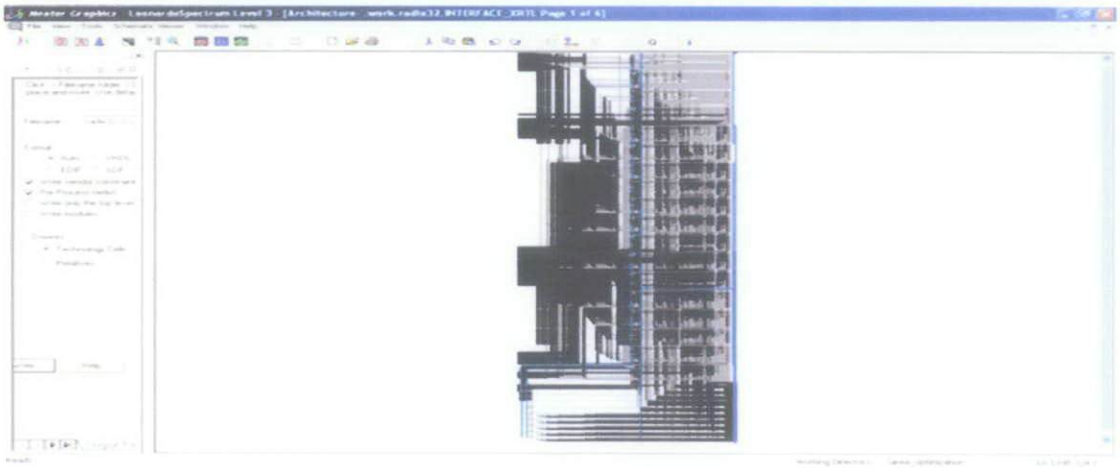
Radix-8 Booth Encoding multiplier logic gates representation:



Radix-16 Booth Encoding multiplier logic gates representation:



Radix-32 Booth Encoding multiplier logic gates representation:



APPENDIX IV: Reports Generated During Synthesis

Area Report for Area-Optimized mode

Radix-2 Booth Encoding multiplier:

```
*****  
Cell: radix2      View: INTERFACE      Library: work  
*****
```

Cell	Library	References	Total Area
and02	tsmc035_typ	27 x	1 34 gates
aoi21	tsmc035_typ	8 x	1 10 gates
aoi32	tsmc035_typ	1 x	2 2 gates
inv01	tsmc035_typ	306 x	1 233 gates
mux21	tsmc035_typ	464 x	2 849 gates
nand02	tsmc035_typ	23 x	1 23 gates
nand03	tsmc035_typ	6 x	1 7 gates
nor02	tsmc035_typ	6 x	1 6 gates
nor03	tsmc035_typ	2 x	1 2 gates
oai21	tsmc035_typ	14 x	1 17 gates
or02	tsmc035_typ	2 x	1 2 gates
xnor2	tsmc035_typ	383 x	2 732 gates
xor2	tsmc035_typ	338 x	2 717 gates

```
Number of ports :          64  
Number of nets :          1612  
Number of instances :      1580  
Number of references to this view : 0
```

```
Total accumulated area :  
Number of gates :          2634  
Number of accumulated instances : 1580
```

Radix-4 Booth Encoding multiplier:

```
*****  
Cell: radix4      View: INTERFACE      Library: work  
*****
```

Cell	Library	References	Total Area
and02	tsmc035_typ	2 x	1 3 gates
and03	tsmc035_typ	1 x	2 2 gates
ao21	tsmc035_typ	1 x	2 2 gates
ao22	tsmc035_typ	63 x	2 124 gates
ao221	tsmc035_typ	15 x	2 37 gates
aoi21	tsmc035_typ	7 x	1 9 gates
aoi22	tsmc035_typ	1 x	1 1 gates
aoi221	tsmc035_typ	83 x	2 164 gates
aoi222	tsmc035_typ	16 x	2 36 gates
fake_gnd	tsmc035_typ	1 x	1 1 fake_gnd
inv01	tsmc035_typ	313 x	1 238 gates
mux21	tsmc035_typ	101 x	2 185 gates
nand02	tsmc035_typ	17 x	1 17 gates
nand03	tsmc035_typ	6 x	1 7 gates
nor02	tsmc035_typ	46 x	1 46 gates
nor03	tsmc035_typ	7 x	1 9 gates
oai21	tsmc035_typ	6 x	1 7 gates
oai22	tsmc035_typ	35 x	1 52 gates
oai222	tsmc035_typ	13 x	2 29 gates
or02	tsmc035_typ	8 x	1 10 gates
or03	tsmc035_typ	6 x	2 9 gates
xnor2	tsmc035_typ	223 x	2 426 gates
xor2	tsmc035_typ	18 x	2 38 gates

```
Number of ports :          64  
Number of nets :          1021  
Number of instances :      989  
Number of references to this view : 0
```

```

Total accumulated area :
Number of fake_gnd :          1
Number of gates :             1450
Number of accumulated instances : 989

```

Radix-8 Booth Encoding multiplier:

```

*****
Cell: radix8      View: INTERFACE      Library: work
*****

```

Cell	Library	References	Total Area
and02	tsmc035_typ	7 x	1 9 gates
and03	tsmc035_typ	8 x	2 12 gates
and04	tsmc035_typ	6 x	2 10 gates
aoi21	tsmc035_typ	14 x	1 17 gates
aoi22	tsmc035_typ	251 x	1 371 gates
aoi222	tsmc035_typ	23 x	2 52 gates
aoi32	tsmc035_typ	14 x	2 24 gates
aoi33	tsmc035_typ	4 x	2 9 gates
fake_gnd	tsmc035_typ	1 x	1 1 fake_gnd
inv01	tsmc035_typ	457 x	1 347 gates
mux21	tsmc035_typ	53 x	2 97 gates
nand02	tsmc035_typ	174 x	1 174 gates
nand03	tsmc035_typ	46 x	1 57 gates
nand04	tsmc035_typ	69 x	1 102 gates
nor02	tsmc035_typ	63 x	1 63 gates
nor03	tsmc035_typ	6 x	1 7 gates
oai21	tsmc035_typ	44 x	1 55 gates
oai222	tsmc035_typ	1 x	2 2 gates
oai321	tsmc035_typ	5 x	2 11 gates
oai33	tsmc035_typ	4 x	2 8 gates
or02	tsmc035_typ	16 x	1 20 gates
or03	tsmc035_typ	13 x	2 20 gates
or04	tsmc035_typ	11 x	2 22 gates
xnor2	tsmc035_typ	334 x	2 638 gates
xor2	tsmc035_typ	16 x	2 34 gates

```

Number of ports :          64
Number of nets :          1671
Number of instances :      1640
Number of references to this view : 0

```

```

Total accumulated area :
Number of fake_gnd :          1
Number of gates :             2162
Number of accumulated instances : 1640

```

Radix-16 Booth Encoding multiplier:

```

*****
Cell: radix16     View: INTERFACE      Library: work
*****

```

Cell	Library	References	Total Area
and02	tsmc035_typ	15 x	1 19 gates
and03	tsmc035_typ	7 x	2 11 gates
and04	tsmc035_typ	11 x	2 19 gates
ao21	tsmc035_typ	7 x	2 12 gates
ao22	tsmc035_typ	68 x	2 134 gates
ao221	tsmc035_typ	13 x	2 32 gates
aoi21	tsmc035_typ	41 x	1 51 gates
aoi22	tsmc035_typ	74 x	1 110 gates
aoi221	tsmc035_typ	132 x	2 261 gates
aoi222	tsmc035_typ	31 x	2 69 gates
aoi32	tsmc035_typ	2 x	2 3 gates
aoi321	tsmc035_typ	1 x	2 2 gates
buf02	tsmc035_typ	12 x	1 12 gates
fake_gnd	tsmc035_typ	1 x	1 1 fake_gnd

inv01	tsmc035_typ	948 x	1	720 gates
mux21	tsmc035_typ	38 x	2	70 gates
nand02	tsmc035_typ	402 x	1	402 gates
nand03	tsmc035_typ	29 x	1	36 gates
nand04	tsmc035_typ	68 x	1	101 gates
nor02	tsmc035_typ	201 x	1	201 gates
nor03	tsmc035_typ	41 x	1	51 gates
nor04	tsmc035_typ	22 x	1	33 gates
oai21	tsmc035_typ	74 x	1	92 gates
oai22	tsmc035_typ	101 x	1	149 gates
oai221	tsmc035_typ	16 x	2	32 gates
oai222	tsmc035_typ	6 x	2	13 gates
oai32	tsmc035_typ	6 x	2	10 gates
oai322	tsmc035_typ	3 x	3	8 gates
oai33	tsmc035_typ	3 x	2	6 gates
or02	tsmc035_typ	21 x	1	26 gates
or03	tsmc035_typ	13 x	2	20 gates
or04	tsmc035_typ	10 x	2	20 gates
xnor2	tsmc035_typ	759 x	2	1450 gates
xor2	tsmc035_typ	28 x	2	59 gates

Number of ports : 64
Number of nets : 3236
Number of instances : 3204
Number of references to this view : 0

Total accumulated area :
Number of fake_gnd : 1
Number of gates : 4234
Number of accumulated instances : 3204

Radix-32 Booth Encoding multiplier:

Cell: radix32 View: INTERFACE Library: work

Cell	Library	References		Total Area
and02	tsmc035_typ	49 x	1	62 gates
and03	tsmc035_typ	12 x	2	18 gates
and04	tsmc035_typ	9 x	2	16 gates
ao21	tsmc035_typ	9 x	2	16 gates
ao22	tsmc035_typ	137 x	2	270 gates
ao221	tsmc035_typ	29 x	2	71 gates
aoi21	tsmc035_typ	98 x	1	122 gates
aoi22	tsmc035_typ	149 x	1	221 gates
aoi221	tsmc035_typ	87 x	2	172 gates
aoi222	tsmc035_typ	12 x	2	27 gates
aoi321	tsmc035_typ	6 x	2	13 gates
aoi322	tsmc035_typ	2 x	2	5 gates
fake_gnd	tsmc035_typ	1 x	1	1 fake_gnd
inv01	tsmc035_typ	1471 x	1	1118 gates
mux21	tsmc035_typ	28 x	2	51 gates
nand02	tsmc035_typ	667 x	1	667 gates
nand03	tsmc035_typ	41 x	1	51 gates
nand04	tsmc035_typ	69 x	1	102 gates
nor02	tsmc035_typ	593 x	1	593 gates
nor03	tsmc035_typ	32 x	1	40 gates
nor04	tsmc035_typ	79 x	1	117 gates
oai21	tsmc035_typ	129 x	1	160 gates
oai22	tsmc035_typ	171 x	1	253 gates
oai221	tsmc035_typ	73 x	2	145 gates
oai222	tsmc035_typ	19 x	2	42 gates
oai32	tsmc035_typ	4 x	2	7 gates
oai321	tsmc035_typ	3 x	2	7 gates
oai33	tsmc035_typ	2 x	2	4 gates
oai422	tsmc035_typ	3 x	3	8 gates
oai43	tsmc035_typ	1 x	2	2 gates
or02	tsmc035_typ	33 x	1	41 gates
or03	tsmc035_typ	6 x	2	9 gates
or04	tsmc035_typ	12 x	2	24 gates

```

xnor2      tsmc035_typ 1539 x    2  2939 gates
xor2       tsmc035_typ   70 x    2   148 gates

```

```

Number of ports :           64
Number of nets :          5676
Number of instances :      5645
Number of references to this view : 0

```

```

Total accumulated area :
Number of fake_gnd :       1
Number of gates :         7539
Number of accumulated instances : 5645

```

Delay Report for Area-Optimized mode

Radix-2 Booth Encoding multiplier:

Critical Path Report				
NAME	GATE	ARRIVAL		LOAD
-----	-----	-----	-----	-----
multiplicand(0)/		0.00	0.00 dn	0.17
ix2894/Y	inv01	0.23	0.23 up	0.04
ix67/Y	or02	0.21	0.44 up	0.01
ix6120/Y	inv01	0.14	0.59 dn	0.03
ix6122/Y	inv01	0.57	1.16 up	0.10
ix2910/Y	xor2	0.55	1.71 dn	0.03
ix2928/Y	aoi32	0.29	1.99 up	0.03
ix363/Y	inv01	0.13	2.12 dn	0.01
ix2966/Y	mux21	0.34	2.46 up	0.03
ix367/Y	inv01	0.12	2.57 dn	0.01
ix3016/Y	mux21	0.34	2.91 up	0.03
ix371/Y	inv01	0.12	3.03 dn	0.01
ix3072/Y	mux21	0.33	3.37 up	0.03
ix375/Y	inv01	0.12	3.48 dn	0.01
ix3144/Y	mux21	0.34	3.82 up	0.03
ix379/Y	inv01	0.12	3.94 dn	0.01
ix3222/Y	mux21	0.33	4.27 up	0.03
ix383/Y	inv01	0.12	4.39 dn	0.01
ix3318/Y	mux21	0.34	4.73 up	0.03
ix387/Y	inv01	0.12	4.85 dn	0.01
ix3418/Y	mux21	0.33	5.18 up	0.03
ix391/Y	inv01	0.12	5.29 dn	0.01
ix3536/Y	mux21	0.34	5.63 up	0.03
ix395/Y	inv01	0.12	5.75 dn	0.01
ix3658/Y	mux21	0.33	6.08 up	0.03
ix399/Y	inv01	0.12	6.20 dn	0.01
ix3798/Y	mux21	0.34	6.54 up	0.03
ix403/Y	inv01	0.12	6.66 dn	0.01
ix3942/Y	mux21	0.33	6.99 up	0.03
ix407/Y	inv01	0.12	7.11 dn	0.01
ix4102/Y	mux21	0.34	7.44 up	0.03
ix411/Y	inv01	0.12	7.57 dn	0.01
ix4272/Y	mux21	0.35	7.92 up	0.04
ix421/Y	xnor2	0.23	8.15 dn	0.01
ix4268/Y	mux21	0.37	8.52 up	0.04
ix4298/Y	xnor2	0.32	8.84 dn	0.04
ix757/Y	xnor2	0.18	9.02 up	0.01
ix4294/Y	inv01	0.11	9.13 dn	0.01
ix761/Y	mux21	0.35	9.47 up	0.04
ix1067/Y	xnor2	0.32	9.79 dn	0.04
ix3007/Y	mux21	0.36	10.16 up	0.04
ix4534/Y	xor2	0.46	10.61 dn	0.01
ix3017/Y	mux21	0.35	10.96 up	0.04
ix3035/Y	xnor2	0.32	11.28 dn	0.04
ix3367/Y	mux21	0.36	11.64 up	0.04
ix3373/Y	xor2	0.46	12.10 dn	0.01
ix4674/Y	mux21	0.35	12.44 up	0.04
ix4712/Y	xnor2	0.32	12.76 dn	0.04
ix3703/Y	mux21	0.36	13.13 up	0.04
ix4850/Y	xor2	0.46	13.58 dn	0.01

ix3713/Y	mux21	0.35	13.93	up	0.04
ix3731/Y	xnor2	0.32	14.25	dn	0.04
ix4015/Y	mux21	0.36	14.61	up	0.04
ix4021/Y	xor2	0.46	15.07	dn	0.01
ix4966/Y	mux21	0.35	15.42	up	0.04
ix5002/Y	xnor2	0.32	15.74	dn	0.04
ix4303/Y	mux21	0.36	16.10	up	0.04
ix5118/Y	xor2	0.46	16.56	dn	0.01
ix4313/Y	mux21	0.35	16.90	up	0.04
ix4331/Y	xnor2	0.32	17.22	dn	0.04
ix4567/Y	mux21	0.36	17.58	up	0.04
ix4573/Y	xor2	0.46	18.04	dn	0.01
ix5214/Y	mux21	0.35	18.39	up	0.04
ix5252/Y	xnor2	0.32	18.71	dn	0.04
ix4807/Y	mux21	0.36	19.07	up	0.04
ix5346/Y	xor2	0.46	19.53	dn	0.01
ix4817/Y	mux21	0.35	19.88	up	0.04
ix4835/Y	xnor2	0.32	20.20	dn	0.04
ix5023/Y	mux21	0.36	20.56	up	0.04
ix5029/Y	xor2	0.46	21.01	dn	0.01
ix5420/Y	mux21	0.35	21.36	up	0.04
ix5456/Y	xnor2	0.32	21.68	dn	0.04
ix5215/Y	mux21	0.36	22.04	up	0.04
ix5530/Y	xor2	0.46	22.50	dn	0.01
ix5225/Y	mux21	0.35	22.85	up	0.04
ix5243/Y	xnor2	0.32	23.17	dn	0.04
ix5383/Y	mux21	0.36	23.53	up	0.04
ix5389/Y	xor2	0.46	23.98	dn	0.01
ix5584/Y	mux21	0.35	24.33	up	0.04
ix5620/Y	xnor2	0.32	24.65	dn	0.04
ix5527/Y	mux21	0.36	25.02	up	0.04
ix5672/Y	xor2	0.46	25.47	dn	0.01
ix5537/Y	mux21	0.35	25.82	up	0.04
ix5555/Y	xnor2	0.32	26.14	dn	0.04
ix5647/Y	mux21	0.36	26.50	up	0.04
ix5653/Y	xor2	0.46	26.96	dn	0.01
ix5704/Y	mux21	0.35	27.31	up	0.04
ix5675/Y	xnor2	0.32	27.63	dn	0.04
ix5740/Y	mux21	0.34	27.97	up	0.03
ix5743/Y	inv01	0.12	28.08	dn	0.01
ix5776/Y	mux21	0.36	28.44	up	0.04
ix5774/Y	xor2	0.45	28.89	dn	0.01
ix5801/Y	mux21	0.20	29.09	up	0.00
product(29)/		0.00	29.09	up	0.00
data arrival time			29.09		
data required time			not specified		

data required time			not specified		
data arrival time			29.09		

Radix-4 Booth Encoding multiplier:

Critical Path Report					
NAME	GATE	ARRIVAL			LOAD

multiplicand(2)/		0.00	0.00	dn	0.15
ix21/Y	nor03	0.44	0.44	up	0.03
ix371/Y	ao21	0.39	0.83	up	0.04
ix3542/Y	inv01	0.25	1.08	dn	0.05
ix1998/Y	aoi222	0.39	1.47	up	0.03
ix2857/Y	nor03	0.34	1.81	dn	0.03
ix1804/Y	inv01	0.20	2.01	up	0.01
ix2861/Y	mux21	0.22	2.23	dn	0.04
ix1852/Y	mux21	0.39	2.62	up	0.04
ix3115/Y	xnor2	0.26	2.88	dn	0.02
ix3117/Y	xnor2	0.28	3.16	up	0.04
ix3133/Y	mux21	0.23	3.40	dn	0.04
ix2024/Y	mux21	0.37	3.76	up	0.03
ix3137/Y	inv01	0.12	3.89	dn	0.01

ix1722/Y	mux21	0.35	4.24	up	0.04
ix3145/Y	mux21	0.23	4.47	dn	0.04
ix2110/Y	mux21	0.38	4.85	up	0.04
ix3461/Y	xnor2	0.26	5.12	dn	0.02
ix2106/Y	xnor2	0.28	5.40	up	0.04
ix2154/Y	mux21	0.22	5.61	dn	0.03
ix3507/Y	inv01	0.19	5.80	up	0.01
ix2242/Y	mux21	0.23	6.03	dn	0.04
ix3515/Y	mux21	0.38	6.41	up	0.04
ix2424/Y	mux21	0.22	6.63	dn	0.03
ix3519/Y	inv01	0.19	6.81	up	0.01
ix2492/Y	mux21	0.23	7.04	dn	0.04
ix3527/Y	mux21	0.38	7.42	up	0.04
ix2696/Y	mux21	0.24	7.66	dn	0.04
ix3535/Y	mux21	0.38	8.04	up	0.04
ix2860/Y	mux21	0.24	8.28	dn	0.04
ix3543/Y	mux21	0.38	8.65	up	0.04
ix3004/Y	mux21	0.23	8.89	dn	0.04
ix3551/Y	mux21	0.38	9.27	up	0.04
ix3166/Y	mux21	0.22	9.49	dn	0.03
ix3555/Y	inv01	0.19	9.67	up	0.01
ix3234/Y	mux21	0.22	9.89	dn	0.03
ix3561/Y	nor02	0.34	10.23	up	0.03
ix3346/Y	nand02	0.19	10.42	dn	0.03
ix3743/Y	nor02	0.34	10.76	up	0.03
ix3416/Y	nand02	0.19	10.95	dn	0.03
ix3867/Y	nor02	0.34	11.29	up	0.03
ix3458/Y	inv01	0.10	11.39	dn	0.01
ix3911/Y	nor02	0.32	11.71	up	0.03
ix3473/Y	inv01	0.10	11.81	dn	0.01
ix3943/Y	nor02	0.28	12.09	up	0.02
ix3947/Y	xnor2	0.18	12.27	dn	0.00
product(31)/		0.00	12.27	dn	0.00
data arrival time			12.27		
data required time			not specified		

data required time			not specified		
data arrival time			12.27		

Radix-8 Booth Encoding multiplier:

Critical Path Report					
NAME	GATE		ARRIVAL		LOAD

multiplicand(2)/		0.00	0.00	dn	0.28
ix195/Y	nor03	0.44	0.44	up	0.03
ix2458/Y	nand02	0.15	0.58	dn	0.03
ix203/Y	nor02	0.32	0.91	up	0.03
ix2662/Y	nand02	0.15	1.06	dn	0.03
ix353/Y	nor02	0.32	1.38	up	0.03
ix2938/Y	nand02	0.15	1.53	dn	0.03
ix763/Y	nor02	0.32	1.85	up	0.03
ix3276/Y	nand02	0.15	2.00	dn	0.03
ix1565/Y	nor02	0.32	2.32	up	0.03
ix3678/Y	nand02	0.15	2.47	dn	0.03
ix2143/Y	nor02	0.32	2.80	up	0.03
ix4172/Y	nand02	0.15	2.94	dn	0.03
ix3079/Y	nor02	0.47	3.42	up	0.05
ix3087/Y	aoi21	0.30	3.72	dn	0.04
ix6098/Y	inv01	0.35	4.07	up	0.04
ix4346/Y	or02	0.34	4.41	up	0.03
ix4344/Y	oai21	0.17	4.58	dn	0.01
ix3103/Y	aoi21	0.36	4.94	up	0.03
ix3111/Y	xnor2	0.25	5.19	dn	0.04
ix4332/Y	xnor2	0.26	5.45	up	0.04
ix4324/Y	xnor2	0.23	5.68	dn	0.04
ix4318/Y	xnor2	0.26	5.94	up	0.04
ix4314/Y	xnor2	0.23	6.17	dn	0.04
ix3151/Y	xnor2	0.26	6.43	up	0.04
ix3159/Y	xnor2	0.23	6.66	dn	0.04

ix4296/Y	xnor2	0.26	6.92	up	0.04
ix4288/Y	xnor2	0.23	7.15	dn	0.04
ix4282/Y	xnor2	0.26	7.41	up	0.04
ix4278/Y	xnor2	0.23	7.64	dn	0.04
ix3199/Y	xnor2	0.29	7.92	up	0.04
ix4178/Y	aci22	0.12	8.04	dn	0.01
ix3969/Y	nand03	0.27	8.31	up	0.02
ix4164/Y	xnor2	0.29	8.60	dn	0.04
ix4424/Y	mux21	0.33	8.93	up	0.03
ix4851/Y	inv01	0.12	9.05	dn	0.01
ix4702/Y	mux21	0.33	9.38	up	0.03
ix4855/Y	inv01	0.12	9.50	dn	0.01
ix4976/Y	mux21	0.33	9.84	up	0.03
ix4859/Y	inv01	0.09	9.93	dn	0.01
ix5058/Y	nand02	0.21	10.13	up	0.03
ix4991/Y	xnor2	0.24	10.37	dn	0.02
ix5054/Y	xnor2	0.28	10.65	up	0.04
ix5128/Y	mux21	0.23	10.88	dn	0.04
ix5225/Y	mux21	0.38	11.26	up	0.04
ix5262/Y	mux21	0.22	11.48	dn	0.03
ix5357/Y	nor02	0.34	11.83	up	0.03
ix5362/Y	nand02	0.19	12.01	dn	0.03
ix5481/Y	inv01	0.14	12.15	up	0.01
ix5400/Y	nand02	0.16	12.31	dn	0.03
ix5669/Y	nor02	0.33	12.64	up	0.03
ix5482/Y	nand02	0.19	12.84	dn	0.03
ix5815/Y	nor02	0.34	13.17	up	0.03
ix5542/Y	nand02	0.19	13.36	dn	0.03
ix5867/Y	inv01	0.14	13.50	up	0.01
ix5568/Y	nand02	0.16	13.66	dn	0.03
ix5925/Y	nor02	0.20	13.86	up	0.01
ix5929/Y	or02	0.17	14.02	up	0.00
product(30) /		0.00	14.02	up	0.00
data arrival time			14.02		
data required time			not specified		

data required time			not specified		
data arrival time			14.02		

Radix-16 Booth Encoding multiplier:

Critical Path Report					
NAME	GATE	ARRIVAL			LOAD

multiplicand(2) /		0.00	0.00	dn	0.41
ix773/Y	nor03	0.44	0.44	up	0.03
ix4530/Y	nand02	0.15	0.58	dn	0.03
ix781/Y	nor02	0.32	0.91	up	0.03
ix4768/Y	nand02	0.15	1.06	dn	0.03
ix953/Y	nor02	0.32	1.38	up	0.03
ix5142/Y	nand02	0.15	1.52	dn	0.03
ix1077/Y	nor02	0.32	1.85	up	0.03
ix5836/Y	nand02	0.15	1.99	dn	0.03
ix2735/Y	nor02	0.32	2.31	up	0.03
ix6488/Y	nand02	0.15	2.46	dn	0.03
ix2923/Y	nor02	0.32	2.78	up	0.03
ix7428/Y	nand02	0.18	2.96	dn	0.04
ix7424/Y	oai21	0.24	3.20	up	0.01
ix5347/Y	inv01	0.15	3.35	dn	0.02
ix11868/Y	inv01	0.29	3.65	up	0.04
ix11684/Y	inv01	0.27	3.91	dn	0.05
ix11798/Y	inv01	0.32	4.24	up	0.04
ix7584/Y	or02	0.39	4.62	up	0.04
ix7580/Y	oai21	0.30	4.92	dn	0.05
ix7652/Y	xor2	0.62	5.54	up	0.03
ix7650/Y	xnor2	0.22	5.76	dn	0.04
ix5361/Y	nor02	0.35	6.11	up	0.03
ix8732/Y	inv01	0.10	6.21	dn	0.01
ix6809/Y	nor02	0.32	6.53	up	0.03
ix6945/Y	xnor2	0.30	6.83	dn	0.04
ix6953/Y	xnor2	0.27	7.10	up	0.04

ix8716/Y	xnor2	0.23	7.33	dn	0.04
ix8708/Y	xnor2	0.26	7.60	up	0.04
ix8702/Y	xnor2	0.23	7.82	dn	0.04
ix8698/Y	xnor2	0.26	8.08	up	0.04
ix6993/Y	xnor2	0.23	8.32	dn	0.04
ix7001/Y	xnor2	0.27	8.58	up	0.04
ix8680/Y	xnor2	0.23	8.81	dn	0.04
ix8672/Y	xnor2	0.26	9.08	up	0.04
ix8666/Y	xnor2	0.26	9.34	dn	0.05
ix7731/Y	oai22	0.20	9.54	up	0.01
ix8594/Y	aoi221	0.17	9.72	dn	0.01
ix7795/Y	nand04	0.28	9.99	up	0.02
ix8522/Y	xnor2	0.30	10.29	dn	0.04
ix9120/Y	mux21	0.33	10.62	up	0.03
ix9515/Y	inv01	0.12	10.74	dn	0.01
ix9714/Y	mux21	0.33	11.08	up	0.03
ix9843/Y	xnor2	0.26	11.33	dn	0.02
ix9845/Y	xnor2	0.28	11.61	up	0.04
ix9858/Y	mux21	0.23	11.84	dn	0.04
ix10125/Y	mux21	0.38	12.23	up	0.04
ix10084/Y	mux21	0.22	12.45	dn	0.03
ix10129/Y	inv01	0.19	12.64	up	0.01
ix10226/Y	mux21	0.21	12.85	dn	0.03
ix10133/Y	inv01	0.17	13.02	up	0.01
ix10344/Y	mux21	0.21	13.24	dn	0.03
ix10283/Y	nor02	0.34	13.58	up	0.03
ix10518/Y	inv01	0.10	13.68	dn	0.01
ix10429/Y	nor02	0.32	14.00	up	0.03
ix10614/Y	inv01	0.10	14.10	dn	0.01
ix10571/Y	nor02	0.32	14.42	up	0.03
ix10666/Y	inv01	0.10	14.53	dn	0.01
ix10655/Y	nor02	0.32	14.85	up	0.03
ix10696/Y	inv01	0.10	14.95	dn	0.01
ix10731/Y	nor02	0.32	15.27	up	0.03
ix10899/Y	xor2	0.39	15.66	dn	0.00
product(27)/		0.00	15.66	dn	0.00
data arrival time			15.66		
data required time			not specified		

data required time			not specified		
data arrival time			15.66		

Radix-32 Booth Encoding multiplier:

Critical Path Report					
NAME	GATE	ARRIVAL			LOAD

multiplicand(2)/		0.00	0.00	dn	0.50
ix3335/Y	nor03	0.44	0.44	up	0.03
ix7540/Y	nand02	0.15	0.58	dn	0.03
ix3343/Y	nor02	0.32	0.91	up	0.03
ix7534/Y	nand02	0.15	1.06	dn	0.03
ix3351/Y	nor02	0.32	1.38	up	0.03
ix7528/Y	nand02	0.15	1.52	dn	0.03
ix3359/Y	nor02	0.32	1.84	up	0.03
ix7522/Y	nand02	0.15	1.99	dn	0.03
ix3829/Y	nor02	0.32	2.31	up	0.03
ix7548/Y	nand02	0.15	2.46	dn	0.03
ix4025/Y	nor02	0.32	2.77	up	0.03
ix13326/Y	nand02	0.18	2.95	dn	0.04
ix9869/Y	nor02	0.48	3.43	up	0.05
ix9877/Y	aoi21	0.14	3.57	dn	0.01
ix14188/Y	inv01	0.38	3.95	up	0.06
ix19870/Y	inv01	0.25	4.19	dn	0.04
ix19752/Y	inv01	0.31	4.50	up	0.04
ix14264/Y	or02	0.38	4.88	up	0.04
ix14258/Y	oai21	0.25	5.13	dn	0.03
ix9887/Y	xor2	0.60	5.73	up	0.03
ix14402/Y	xnor2	0.33	6.06	dn	0.07
ix14928/Y	xor2	0.70	6.76	up	0.04

ix9893/Y	xnor2	0.23	6.99	dn	0.04
ix14920/Y	xnor2	0.28	7.27	up	0.04
ix9909/Y	xnor2	0.23	7.51	dn	0.04
ix14908/Y	xnor2	0.27	7.77	up	0.04
ix14900/Y	xnor2	0.23	8.00	dn	0.04
ix14892/Y	xnor2	0.26	8.26	up	0.04
ix14886/Y	xnor2	0.23	8.49	dn	0.04
ix14882/Y	xnor2	0.26	8.75	up	0.04
ix9957/Y	xnor2	0.23	8.98	dn	0.04
ix14870/Y	xnor2	0.26	9.24	up	0.04
ix14862/Y	xnor2	0.23	9.47	dn	0.04
ix14854/Y	xnor2	0.26	9.73	up	0.04
ix13041/Y	aoi22	0.16	9.90	dn	0.01
ix14798/Y	aoi221	0.22	10.12	up	0.01
ix13123/Y	nand04	0.17	10.29	dn	0.01
ix14184/Y	nor04	0.37	10.65	up	0.02
ix13215/Y	xnor2	0.36	11.01	dn	0.04
ix18155/Y	xnor2	0.24	11.25	up	0.02
ix14172/Y	xnor2	0.29	11.54	dn	0.04
ix18256/Y	mux21	0.33	11.87	up	0.03
ix18451/Y	inv01	0.12	11.99	dn	0.01
ix10944/Y	mux21	0.33	12.32	up	0.03
ix18455/Y	inv01	0.12	12.44	dn	0.01
ix18266/Y	mux21	0.33	12.78	up	0.03
ix18459/Y	inv01	0.12	12.90	dn	0.01
ix10940/Y	mux21	0.33	13.23	up	0.03
ix18463/Y	inv01	0.12	13.35	dn	0.01
ix18288/Y	mux21	0.33	13.68	up	0.03
ix18467/Y	inv01	0.12	13.80	dn	0.01
ix10936/Y	mux21	0.33	14.14	up	0.03
ix18471/Y	inv01	0.12	14.26	dn	0.01
ix18298/Y	mux21	0.33	14.59	up	0.03
ix18475/Y	inv01	0.12	14.71	dn	0.01
ix7508/Y	mux21	0.34	15.04	up	0.03
ix18479/Y	inv01	0.12	15.17	dn	0.01
ix18308/Y	mux21	0.33	15.50	up	0.03
ix18483/Y	inv01	0.09	15.59	dn	0.01
ix7504/Y	nand02	0.21	15.79	up	0.03
ix18615/Y	inv01	0.09	15.88	dn	0.01
ix18340/Y	nand02	0.20	16.08	up	0.03
ix18747/Y	inv01	0.09	16.17	dn	0.01
ix7500/Y	nand02	0.16	16.33	up	0.02
ix19007/Y	nor02	0.15	16.47	dn	0.01
ix19013/Y	aoi21	0.11	16.59	up	0.00
product(26)/		0.00	16.59	up	0.00
data arrival time			16.59		
data required time				not specified	

data required time				not specified	
data arrival time				16.59	

Area Report for Speed-Optimized mode

Radix-2 Booth Encoding multiplier:

Cell: radix2 View: INTERFACE Library: work

Cell	Library	References	Total Area
aoi21	tsmc035_typ	4 x	1 5 gates
aoi22	tsmc035_typ	106 x	1 157 gates
aoi32	tsmc035_typ	172 x	2 298 gates
inv01	tsmc035_typ	549 x	1 417 gates
mux21	tsmc035_typ	62 x	2 113 gates
nand02	tsmc035_typ	337 x	1 337 gates
nand03	tsmc035_typ	10 x	1 12 gates
nand04	tsmc035_typ	8 x	1 12 gates
nor02	tsmc035_typ	13 x	1 13 gates

```

nor03      tsmc035_typ    3 x    1    4 gates
nor04      tsmc035_typ    1 x    1    1 gates
oai21      tsmc035_typ   237 x    1   294 gates
oai22      tsmc035_typ   106 x    1   157 gates
xnor2      tsmc035_typ  1278 x    2  2441 gates

```

```

Number of ports :           64
Number of nets :           2918
Number of instances :       2886
Number of references to this view : 0

```

```

Total accumulated area :
Number of gates :           4261
Number of accumulated instances : 2886

```

Radix-4 Booth Encoding multiplier:

```

*****
Cell: radix4      View: INTERFACE      Library: work
*****

```

Cell	Library	References	Total Area
aoi21	tsmc035_typ	13 x	1 16 gates
aoi22	tsmc035_typ	223 x	1 330 gates
aoi221	tsmc035_typ	18 x	2 36 gates
buf02	tsmc035_typ	28 x	1 29 gates
fake_gnd	tsmc035_typ	1 x	1 1 fake_gnd
inv01	tsmc035_typ	278 x	1 211 gates
mux21	tsmc035_typ	13 x	2 24 gates
nand02	tsmc035_typ	213 x	1 213 gates
nand03	tsmc035_typ	63 x	1 78 gates
nand04	tsmc035_typ	37 x	1 55 gates
nor02	tsmc035_typ	81 x	1 81 gates
nor03	tsmc035_typ	6 x	1 7 gates
nor04	tsmc035_typ	3 x	1 4 gates
oai21	tsmc035_typ	98 x	1 122 gates
oai22	tsmc035_typ	68 x	1 101 gates
oai221	tsmc035_typ	25 x	2 50 gates
oai222	tsmc035_typ	3 x	2 7 gates
xnor2	tsmc035_typ	397 x	2 758 gates

```

Number of ports :           64
Number of nets :           1600
Number of instances :       1568
Number of references to this view : 0

```

```

Total accumulated area :
Number of fake_gnd :         1
Number of gates :           2121
Number of accumulated instances : 1568

```

Radix-8 Booth Encoding multiplier:

```

*****
Cell: radix8      View: INTERFACE      Library: work
*****

```

Cell	Library	References	Total Area
and04	tsmc035_typ	6 x	2 10 gates
aoi21	tsmc035_typ	7 x	1 9 gates
aoi22	tsmc035_typ	181 x	1 268 gates
aoi221	tsmc035_typ	10 x	2 20 gates
aoi222	tsmc035_typ	5 x	2 11 gates
aoi43	tsmc035_typ	4 x	2 9 gates
buf02	tsmc035_typ	23 x	1 23 gates
fake_gnd	tsmc035_typ	1 x	1 1 fake_gnd
inv01	tsmc035_typ	392 x	1 298 gates
mux21	tsmc035_typ	19 x	2 35 gates
nand02	tsmc035_typ	315 x	1 315 gates
nand03	tsmc035_typ	114 x	1 141 gates

```

buf02      tsmc035_typ    74 x    1    75 gates
fake_gnd   tsmc035_typ     1 x    1     1 fake_gnd
inv01      tsmc035_typ  1146 x    1   871 gates
mux21      tsmc035_typ   18 x    2    33 gates
nand02     tsmc035_typ  1286 x    1  1286 gates
nand03     tsmc035_typ   164 x    1   203 gates
nand04     tsmc035_typ   257 x    1   380 gates
nor02      tsmc035_typ  1253 x    1  1253 gates
nor03      tsmc035_typ   80 x    1    99 gates
nor04      tsmc035_typ   10 x    1    15 gates
oai21      tsmc035_typ   231 x    1   286 gates
oai22      tsmc035_typ   129 x    1   191 gates
oai221     tsmc035_typ   127 x    2   251 gates
oai222     tsmc035_typ   14 x    2    31 gates
xnor2      tsmc035_typ  3299 x    2  6301 gates

```

```

Number of ports :          64
Number of nets :         8554
Number of instances :     8523
Number of references to this view : 0

```

```

Total accumulated area :
Number of fake_gnd :      1
Number of gates :        11949
Number of accumulated instances : 8523

```

Delay Report for Speed-Optimized mode

Radix-2 Booth Encoding multiplier:

Critical Path Report				
NAME	GATE	ARRIVAL		LOAD
-----	-----	-----	-----	-----
multiplicand(1)/		0.00	0.00 dn	0.27
ix2894/Y	inv01	0.22	0.22 up	0.04
ix2892/Y	nand02	0.10	0.32 dn	0.01
ix7288/Y	inv01	0.37	0.68 up	0.06
ix8206/Y	inv01	0.16	0.85 dn	0.02
ix8086/Y	inv01	0.65	1.50 up	0.12
ix2984/Y	xnor2	0.28	1.77 dn	0.01
ix2982/Y	aoi22	0.46	2.23 up	0.04
ix235/Y	oai21	0.31	2.54 dn	0.05
ix3138/Y	inv01	0.19	2.73 up	0.01
ix245/Y	oai21	0.25	2.98 dn	0.05
ix3276/Y	aoi22	0.38	3.36 up	0.04
ix269/Y	oai22	0.29	3.65 dn	0.04
ix3714/Y	aoi22	0.29	3.94 up	0.01
ix287/Y	aoi22	0.30	4.24 dn	0.05
ix3910/Y	aoi22	0.38	4.62 up	0.04
ix311/Y	oai21	0.25	4.87 dn	0.04
ix4334/Y	aoi22	0.49	5.36 up	0.06
ix329/Y	oai22	0.34	5.70 dn	0.05
ix4852/Y	aoi22	0.39	6.08 up	0.04
ix353/Y	oai21	0.31	6.39 dn	0.05
ix5434/Y	aoi22	0.41	6.81 up	0.05
ix2757/Y	xnor2	0.24	7.04 dn	0.01
ix5450/Y	aoi32	0.56	7.61 up	0.09
ix2777/Y	xnor2	0.38	7.99 dn	0.04
ix5679/Y	aoi22	0.49	8.47 up	0.05
ix3095/Y	xnor2	0.23	8.71 dn	0.01
ix5690/Y	aoi32	0.56	9.27 up	0.09
ix5688/Y	xnor2	0.40	9.67 dn	0.04
ix5885/Y	mux21	0.44	10.12 up	0.05
ix3411/Y	xnor2	0.23	10.35 dn	0.01
ix5896/Y	aoi32	0.56	10.91 up	0.09
ix5894/Y	xnor2	0.40	11.31 dn	0.04
ix6081/Y	mux21	0.45	11.75 up	0.05
ix3705/Y	xnor2	0.23	11.98 dn	0.01
ix6092/Y	aoi32	0.56	12.54 up	0.09
ix6090/Y	xnor2	0.40	12.94 dn	0.04

ix6238/Y	mux21	0.45	13.39	up	0.05
ix3977/Y	xnor2	0.23	13.62	dn	0.01
ix6249/Y	aoi32	0.56	14.18	up	0.09
ix6247/Y	xnor2	0.40	14.58	dn	0.04
ix6405/Y	mux21	0.45	15.02	up	0.05
ix4225/Y	xnor2	0.23	15.25	dn	0.01
ix6416/Y	aoi32	0.56	15.81	up	0.09
ix6414/Y	xnor2	0.40	16.21	dn	0.04
ix6551/Y	mux21	0.45	16.66	up	0.05
ix4451/Y	xnor2	0.23	16.89	dn	0.01
ix6562/Y	aoi32	0.56	17.45	up	0.09
ix6560/Y	xnor2	0.40	17.85	dn	0.04
ix6680/Y	mux21	0.45	18.29	up	0.05
ix4655/Y	xnor2	0.23	18.52	dn	0.01
ix6691/Y	aoi32	0.56	19.08	up	0.09
ix6689/Y	xnor2	0.40	19.48	dn	0.04
ix6778/Y	mux21	0.45	19.93	up	0.05
ix4837/Y	xnor2	0.23	20.16	dn	0.01
ix6789/Y	aoi32	0.56	20.72	up	0.09
ix6787/Y	xnor2	0.40	21.12	dn	0.04
ix6884/Y	mux21	0.45	21.56	up	0.05
ix4995/Y	xnor2	0.23	21.79	dn	0.01
ix6895/Y	aoi32	0.56	22.35	up	0.09
ix6893/Y	xnor2	0.40	22.75	dn	0.04
ix6965/Y	mux21	0.45	23.20	up	0.05
ix5131/Y	xnor2	0.23	23.43	dn	0.01
ix6976/Y	aoi32	0.56	23.99	up	0.09
ix6974/Y	xnor2	0.40	24.39	dn	0.04
ix7044/Y	mux21	0.45	24.83	up	0.05
ix5245/Y	xnor2	0.23	25.06	dn	0.01
ix7055/Y	aoi32	0.56	25.62	up	0.09
ix7053/Y	xnor2	0.40	26.02	dn	0.04
ix7088/Y	mux21	0.45	26.47	up	0.05
ix5337/Y	xnor2	0.23	26.70	dn	0.01
ix7099/Y	aoi32	0.56	27.26	up	0.09
ix7097/Y	xnor2	0.40	27.66	dn	0.04
ix7129/Y	mux21	0.45	28.10	up	0.05
ix5405/Y	xnor2	0.23	28.33	dn	0.01
ix7126/Y	aoi32	0.47	28.80	up	0.07
ix7151/Y	xnor2	0.29	29.09	dn	0.02
ix7149/Y	mux21	0.31	29.40	up	0.02
ix7147/Y	xnor2	0.22	29.62	dn	0.01
ix5455/Y	oai21	0.14	29.76	up	0.00
product(30)/		0.00	29.76	up	0.00
data arrival time			29.76		
data required time				not specified	

data required time				not specified	
data arrival time				29.76	

Radix-4 Booth Encoding multiplier:

Critical Path Report					
NAME	GATE	ARRIVAL			LOAD
multiplier(1)/		0.00	0.00	dn	0.13
ix4485/Y	inv01	0.39	0.39	up	0.07
ix2599/Y	xnor2	0.34	0.73	dn	0.06
ix1778/Y	nand02	0.22	0.96	up	0.01
ix4499/Y	inv01	0.16	1.12	dn	0.03
ix4501/Y	inv01	0.39	1.51	up	0.06
ix2957/Y	oai22	0.23	1.74	dn	0.01
ix1890/Y	nor02	0.31	2.05	up	0.03
ix2969/Y	xnor2	0.24	2.29	dn	0.04
ix1820/Y	aoi22	0.40	2.69	up	0.03
ix3017/Y	oai21	0.35	3.04	dn	0.05
ix1792/Y	aoi22	0.38	3.43	up	0.04
ix3065/Y	oai21	0.27	3.70	dn	0.04
ix3353/Y	xnor2	0.34	4.04	up	0.06
ix3357/Y	xnor2	0.30	4.34	dn	0.04

ix2224/Y	aoi22	0.40	4.74	up	0.04
ix3369/Y	oai21	0.25	5.00	dn	0.04
ix3545/Y	xnor2	0.31	5.31	up	0.05
ix3553/Y	xnor2	0.22	5.53	dn	0.01
ix2376/Y	aoi22	0.36	5.89	up	0.04
ix3571/Y	oai22	0.32	6.21	dn	0.04
ix2900/Y	inv01	0.19	6.40	up	0.01
ix3587/Y	oai21	0.17	6.57	dn	0.01
ix2868/Y	aoi22	0.35	6.93	up	0.04
ix3599/Y	oai21	0.25	7.18	dn	0.04
ix3052/Y	mux21	0.41	7.58	up	0.04
ix3607/Y	inv01	0.10	7.68	dn	0.01
ix3346/Y	aoi22	0.31	7.99	up	0.04
ix3625/Y	oai21	0.25	8.24	dn	0.04
ix3524/Y	aoi22	0.38	8.63	up	0.04
ix3639/Y	mux21	0.15	8.78	dn	0.01
ix3896/Y	aoi22	0.21	8.99	up	0.01
ix3661/Y	aoi22	0.27	9.26	dn	0.04
ix4049/Y	aoi22	0.29	9.55	up	0.01
ix3675/Y	mux21	0.15	9.70	dn	0.01
ix4046/Y	nand02	0.24	9.94	up	0.04
ix3689/Y	nor02	0.24	10.18	dn	0.04
ix4157/Y	inv01	0.16	10.34	up	0.01
ix3691/Y	nor02	0.10	10.44	dn	0.01
ix4154/Y	nand02	0.24	10.68	up	0.04
ix3705/Y	nor02	0.21	10.89	dn	0.03
ix4242/Y	nand02	0.24	11.13	up	0.02
ix4240/Y	xnor2	0.19	11.32	dn	0.01
ix3711/Y	inv01	0.09	11.41	up	0.00
product(31)/		0.00	11.41	up	0.00
data arrival time			11.41		
data required time			not specified		

data required time			not specified		
data arrival time			11.41		

Radix-8 Booth Encoding multiplier:

Critical Path Report					
NAME	GATE	ARRIVAL			LOAD

multiplcand(1)/		0.00	0.00	dn	0.25
ix6540/Y	inv01	0.40	0.40	up	0.08
ix2426/Y	nand02	0.30	0.70	dn	0.08
ix997/Y	nor03	0.62	1.32	up	0.06
ix2690/Y	nand03	0.06	1.38	dn	0.01
ix6612/Y	inv01	0.29	1.68	up	0.04
ix1013/Y	and04	0.44	2.12	up	0.04
ix1505/Y	xnor2	0.15	2.27	dn	0.01
ix6372/Y	inv01	0.44	2.71	up	0.07
ix6374/Y	inv01	0.29	3.00	dn	0.05
ix1513/Y	xnor2	0.24	3.24	up	0.03
ix1521/Y	xnor2	0.20	3.44	dn	0.03
ix1529/Y	xnor2	0.26	3.70	up	0.04
ix1537/Y	xnor2	0.23	3.93	dn	0.04
ix1545/Y	xnor2	0.27	4.20	up	0.04
ix1553/Y	xnor2	0.23	4.43	dn	0.04
ix3976/Y	nand02	0.35	4.78	up	0.05
ix4260/Y	xnor2	0.29	5.07	dn	0.04
ix1665/Y	nor02	0.49	5.56	up	0.05
ix1771/Y	xnor2	0.33	5.89	dn	0.04
ix4556/Y	nand02	0.40	6.29	up	0.06
ix4866/Y	xnor2	0.30	6.58	dn	0.04
ix1887/Y	nor02	0.49	7.08	up	0.05
ix2009/Y	xnor2	0.30	7.38	dn	0.03
ix4836/Y	xnor2	0.25	7.63	up	0.03
ix5393/Y	oai221	0.17	7.80	dn	0.01
ix4826/Y	nor02	0.48	8.29	up	0.05
ix5447/Y	xnor2	0.25	8.53	dn	0.01
ix5414/Y	aoi22	0.39	8.92	up	0.04
ix5459/Y	oai21	0.26	9.19	dn	0.04

ix5622/Y	nand02	0.41	9.60	up	0.07
ix5537/Y	oai21	0.16	9.76	dn	0.01
ix5712/Y	nand02	0.36	10.12	up	0.05
ix5545/Y	xnor2	0.28	10.40	dn	0.03
ix5896/Y	aoi22	0.24	10.63	up	0.01
ix5767/Y	oai22	0.23	10.86	dn	0.03
ix5968/Y	nand02	0.29	11.15	up	0.04
ix5777/Y	nor02	0.24	11.39	dn	0.04
ix6174/Y	inv01	0.16	11.55	up	0.01
ix5779/Y	nor02	0.10	11.64	dn	0.01
ix6171/Y	nand02	0.24	11.89	up	0.04
ix5799/Y	nor02	0.21	12.10	dn	0.03
ix6246/Y	nand02	0.24	12.34	up	0.02
ix5827/Y	xnor2	0.16	12.50	dn	0.00
product(29)/		0.00	12.50	dn	0.00
data arrival time			12.50		
data required time			not specified		

data required time			not specified		
data arrival time			12.50		

Radix-16 Booth Encoding multiplier:

Critical Path Report					
NAME	GATE	ARRIVAL			LOAD

multiplicand(3)/		0.00	0.00	dn	0.49
ix12628/Y	inv01	0.36	0.36	up	0.07
ix4446/Y	nand04	0.18	0.54	dn	0.01
ix12626/Y	buf02	0.27	0.81	dn	0.02
ix2641/Y	nor04	0.24	1.05	up	0.01
ix5320/Y	inv01	0.28	1.33	dn	0.05
ix5780/Y	nor02	0.57	1.90	up	0.07
ix12826/Y	inv01	0.28	2.18	dn	0.04
ix2657/Y	nor03	0.57	2.74	up	0.05
ix7662/Y	nand02	0.22	2.96	dn	0.03
ix2665/Y	inv01	0.45	3.41	up	0.07
ix7660/Y	nor02	0.16	3.58	dn	0.01
ix12926/Y	inv01	0.36	3.93	up	0.05
ix13176/Y	inv01	0.35	4.29	dn	0.07
ix3297/Y	xnor2	0.27	4.56	up	0.03
ix13104/Y	inv01	0.17	4.73	dn	0.03
ix3299/Y	xnor2	0.20	4.93	up	0.03
ix3301/Y	xnor2	0.23	5.16	dn	0.04
ix3309/Y	xnor2	0.28	5.44	up	0.04
ix8434/Y	xnor2	0.23	5.67	dn	0.04
ix3319/Y	nor02	0.51	6.18	up	0.05
ix3427/Y	xnor2	0.33	6.51	dn	0.04
ix3435/Y	xnor2	0.28	6.79	up	0.04
ix8462/Y	xnor2	0.23	7.02	dn	0.04
ix3445/Y	nor02	0.51	7.53	up	0.05
ix3563/Y	xnor2	0.33	7.86	dn	0.04
ix10060/Y	xnor2	0.28	8.14	up	0.04
ix10058/Y	xnor2	0.23	8.37	dn	0.04
ix3581/Y	nor02	0.45	8.82	up	0.05
ix10820/Y	xnor2	0.32	9.14	dn	0.04
ix3713/Y	nor02	0.46	9.60	up	0.05
ix10806/Y	xnor2	0.24	9.84	dn	0.01
ix10199/Y	oai21	0.17	10.01	up	0.01
ix10684/Y	nor02	0.13	10.14	dn	0.01
ix10231/Y	nand03	0.32	10.46	up	0.06
ix10299/Y	xnor2	0.22	10.68	dn	0.01
ix11357/Y	aoi22	0.36	11.05	up	0.04
ix10307/Y	nor02	0.23	11.28	dn	0.03
ix11559/Y	nand02	0.33	11.61	up	0.03
ix10741/Y	oai21	0.15	11.76	dn	0.01
ix11640/Y	nand02	0.38	12.14	up	0.07
ix10749/Y	xnor2	0.30	12.44	dn	0.04

ix11871/Y	aoi22	0.29	12.73	up	0.01
ix10761/Y	oai22	0.26	12.99	dn	0.04
ix12011/Y	inv01	0.17	13.16	up	0.01
ix10763/Y	nor02	0.10	13.26	dn	0.01
ix12008/Y	nand02	0.24	13.50	up	0.04
ix10773/Y	nor02	0.21	13.72	dn	0.03
ix12127/Y	nand02	0.28	13.99	up	0.03
ix10797/Y	oai21	0.14	14.14	dn	0.01
ix12157/Y	nand02	0.17	14.31	up	0.01
ix10801/Y	inv01	0.05	14.36	dn	0.00
product(28)/		0.00	14.36	dn	0.00
data arrival time			14.36		
data required time			not specified		

data required time			not specified		
data arrival time			14.36		

Radix-32 Booth Encoding multiplier:

NAME	Critical Path Report			LOAD
	GATE	ARRIVAL		

multiplicand(3)/		0.00	0.00 dn	0.82
ix22296/Y	inv01	0.34	0.34 up	0.06
ix7582/Y	nand04	0.18	0.52 dn	0.01
ix22308/Y	buf02	0.28	0.80 dn	0.03
ix4627/Y	nor04	0.47	1.27 up	0.04
ix7700/Y	inv01	0.19	1.46 dn	0.02
ix21754/Y	inv01	0.43	1.89 up	0.06
ix7752/Y	nand02	0.20	2.09 dn	0.03
ix4635/Y	inv01	0.19	2.28 up	0.01
ix22384/Y	inv01	0.22	2.50 dn	0.05
ix4643/Y	nor03	0.64	3.14 up	0.07
ix5195/Y	xnor2	0.18	3.32 dn	0.01
ix21914/Y	inv01	0.63	3.95 up	0.11
ix23826/Y	inv01	0.25	4.20 dn	0.04
ix23776/Y	inv01	0.61	4.81 up	0.10
ix5313/Y	xnor2	0.34	5.15 dn	0.03
ix23170/Y	inv01	0.28	5.43 up	0.03
ix5315/Y	xnor2	0.20	5.63 dn	0.03
ix5317/Y	xnor2	0.26	5.90 up	0.03
ix23168/Y	inv01	0.26	6.16 dn	0.05
ix5319/Y	xnor2	0.27	6.42 up	0.04
ix23272/Y	inv01	0.18	6.60 dn	0.03
ix5321/Y	xnor2	0.23	6.83 up	0.04
ix10428/Y	xnor2	0.23	7.06 dn	0.04
ix5337/Y	xnor2	0.27	7.33 up	0.04
ix10424/Y	xnor2	0.23	7.56 dn	0.04
ix5347/Y	nor02	0.46	8.03 up	0.05
ix10466/Y	xnor2	0.32	8.35 dn	0.04
ix5473/Y	nor02	0.50	8.85 up	0.05
ix5595/Y	xnor2	0.33	9.18 dn	0.04
ix16628/Y	nand02	0.40	9.58 up	0.06
ix16626/Y	xnor2	0.30	9.88 dn	0.04
ix16622/Y	xnor2	0.26	10.14 up	0.04
ix16596/Y	xnor2	0.23	10.37 dn	0.04
ix11423/Y	nor02	0.31	10.68 up	0.02
ix19536/Y	xnor2	0.30	10.98 dn	0.04
ix11497/Y	oai21	0.18	11.16 up	0.01
ix19392/Y	nor02	0.15	11.31 dn	0.01
ix12477/Y	nand04	0.29	11.60 up	0.05
ix18810/Y	xnor2	0.30	11.90 dn	0.04
ix16733/Y	mux21	0.33	12.23 up	0.03
ix15450/Y	nand02	0.19	12.43 dn	0.03
ix16867/Y	nor02	0.53	12.96 up	0.06
ix20549/Y	xnor2	0.38	13.33 dn	0.05
ix20547/Y	xnor2	0.20	13.53 up	0.01
ix18527/Y	oai21	0.27	13.80 dn	0.04
ix20499/Y	aoi22	0.28	14.08 up	0.01
ix18541/Y	mux21	0.15	14.24 dn	0.01

ix14944/Y	aoi22	0.31	14.55	up	0.04
ix18681/Y	nor02	0.23	14.78	dn	0.03
ix7498/Y	nand02	0.32	15.10	up	0.04
ix18947/Y	oai21	0.15	15.25	dn	0.01
ix21623/Y	nand02	0.16	15.42	up	0.01
ix18951/Y	inv01	0.05	15.47	dn	0.00
product(26)/		0.00	15.47	dn	0.00
data arrival time			15.47		
data required time			not specified		

data required time			not specified		
data arrival time			15.47		

Area Report for Auto-Optimized mode

Radix-2 Booth Encoding multiplier:

 Cell: radix2 View: INTERFACE Library: work

Cell	Library	References	Total Area
and02	tsmc035_typ	27 x	1 34 gates
aoi21	tsmc035_typ	8 x	1 10 gates
aoi32	tsmc035_typ	1 x	2 2 gates
inv01	tsmc035_typ	306 x	1 233 gates
mux21	tsmc035_typ	464 x	2 849 gates
nand02	tsmc035_typ	23 x	1 23 gates
nand03	tsmc035_typ	6 x	1 7 gates
nor02	tsmc035_typ	6 x	1 6 gates
nor03	tsmc035_typ	2 x	1 2 gates
oai21	tsmc035_typ	14 x	1 17 gates
or02	tsmc035_typ	2 x	1 2 gates
xnor2	tsmc035_typ	383 x	2 732 gates
xor2	tsmc035_typ	338 x	2 717 gates

Number of ports : 64
 Number of nets : 1612
 Number of instances : 1580
 Number of references to this view : 0

Total accumulated area :
 Number of gates : 2634
 Number of accumulated instances : 1580

Radix-4 Booth Encoding multiplier:

 Cell: radix4 View: INTERFACE Library: work

Cell	Library	References	Total Area
and02	tsmc035_typ	2 x	1 3 gates
and03	tsmc035_typ	1 x	2 2 gates
ao21	tsmc035_typ	1 x	2 2 gates
ao22	tsmc035_typ	63 x	2 124 gates
ao221	tsmc035_typ	15 x	2 37 gates
aoi21	tsmc035_typ	7 x	1 9 gates
aoi22	tsmc035_typ	1 x	1 1 gates
aoi221	tsmc035_typ	83 x	2 164 gates
aoi222	tsmc035_typ	16 x	2 36 gates
fake_gnd	tsmc035_typ	1 x	1 1 fake_gnd
inv01	tsmc035_typ	313 x	1 238 gates
mux21	tsmc035_typ	101 x	2 185 gates
nand02	tsmc035_typ	17 x	1 17 gates
nand03	tsmc035_typ	6 x	1 7 gates
nor02	tsmc035_typ	46 x	1 46 gates
nor03	tsmc035_typ	7 x	1 9 gates
oai21	tsmc035_typ	6 x	1 7 gates

```

oai22      tsmc035_typ    35 x    1    52 gates
oai222     tsmc035_typ    13 x    2    29 gates
or02       tsmc035_typ     8 x    1    10 gates
or03       tsmc035_typ     6 x    2     9 gates
xnor2      tsmc035_typ   223 x    2   426 gates
xor2       tsmc035_typ    18 x    2    38 gates

```

```

Number of ports :           64
Number of nets :           1021
Number of instances :       989
Number of references to this view : 0

```

```

Total accumulated area :
Number of fake_gnd :       1
Number of gates :          1450
Number of accumulated instances : 989

```

Radix-8 Booth Encoding multiplier:

```

*****
Cell: radix8      View: INTERFACE      Library: work
*****

```

Cell	Library	References	Total Area
and02	tsmc035_typ	7 x	1 9 gates
and03	tsmc035_typ	8 x	2 12 gates
and04	tsmc035_typ	6 x	2 10 gates
aoi21	tsmc035_typ	14 x	1 17 gates
aoi22	tsmc035_typ	251 x	1 371 gates
aoi222	tsmc035_typ	23 x	2 52 gates
aoi32	tsmc035_typ	14 x	2 24 gates
aoi33	tsmc035_typ	4 x	2 9 gates
fake_gnd	tsmc035_typ	1 x	1 1 fake_gnd
inv01	tsmc035_typ	457 x	1 347 gates
mux21	tsmc035_typ	53 x	2 97 gates
nand02	tsmc035_typ	174 x	1 174 gates
nand03	tsmc035_typ	46 x	1 57 gates
nand04	tsmc035_typ	69 x	1 102 gates
nor02	tsmc035_typ	63 x	1 63 gates
nor03	tsmc035_typ	6 x	1 7 gates
oai21	tsmc035_typ	44 x	1 55 gates
oai222	tsmc035_typ	1 x	2 2 gates
oai321	tsmc035_typ	5 x	2 11 gates
oai33	tsmc035_typ	4 x	2 8 gates
or02	tsmc035_typ	16 x	1 20 gates
or03	tsmc035_typ	13 x	2 20 gates
or04	tsmc035_typ	11 x	2 22 gates
xnor2	tsmc035_typ	334 x	2 638 gates
xor2	tsmc035_typ	16 x	2 34 gates

```

Number of ports :           64
Number of nets :           1671
Number of instances :       1640
Number of references to this view : 0

```

```

Total accumulated area :
Number of fake_gnd :       1
Number of gates :          2162
Number of accumulated instances : 1640

```

Radix-16 Booth Encoding multiplier:

```

*****
Cell: radix16     View: INTERFACE      Library: work
*****

```

Cell	Library	References	Total Area
and02	tsmc035_typ	15 x	1 19 gates
and03	tsmc035_typ	7 x	2 11 gates
and04	tsmc035_typ	11 x	2 19 gates

ao21	tsmc035_typ	7 x	2	12 gates
ao22	tsmc035_typ	68 x	2	134 gates
ao221	tsmc035_typ	13 x	2	32 gates
aoi21	tsmc035_typ	41 x	1	51 gates
aoi22	tsmc035_typ	74 x	1	110 gates
aoi221	tsmc035_typ	132 x	2	261 gates
aoi222	tsmc035_typ	31 x	2	69 gates
aoi32	tsmc035_typ	2 x	2	3 gates
aoi321	tsmc035_typ	1 x	2	2 gates
buf02	tsmc035_typ	12 x	1	12 gates
fake_gnd	tsmc035_typ	1 x	1	1 fake_gnd
inv01	tsmc035_typ	948 x	1	720 gates
mux21	tsmc035_typ	38 x	2	70 gates
nand02	tsmc035_typ	402 x	1	402 gates
nand03	tsmc035_typ	29 x	1	36 gates
nand04	tsmc035_typ	68 x	1	101 gates
nor02	tsmc035_typ	201 x	1	201 gates
nor03	tsmc035_typ	41 x	1	51 gates
nor04	tsmc035_typ	22 x	1	33 gates
oai21	tsmc035_typ	74 x	1	92 gates
oai22	tsmc035_typ	101 x	1	149 gates
oai221	tsmc035_typ	16 x	2	32 gates
oai222	tsmc035_typ	6 x	2	13 gates
oai32	tsmc035_typ	6 x	2	10 gates
oai322	tsmc035_typ	3 x	3	8 gates
oai33	tsmc035_typ	3 x	2	6 gates
or02	tsmc035_typ	21 x	1	26 gates
or03	tsmc035_typ	13 x	2	20 gates
or04	tsmc035_typ	10 x	2	20 gates
xnor2	tsmc035_typ	759 x	2	1450 gates
xor2	tsmc035_typ	28 x	2	59 gates

Number of ports : 64
Number of nets : 3236
Number of instances : 3204
Number of references to this view : 0

Total accumulated area :
Number of fake_gnd : 1
Number of gates : 4234
Number of accumulated instances : 3204

Radix-32 Booth Encoding multiplier:

Cell: radix32 View: INTERFACE Library: work

Cell	Library	References	Total Area
and02	tsmc035_typ	19 x	1 24 gates
and03	tsmc035_typ	2 x	2 3 gates
and04	tsmc035_typ	3 x	2 5 gates
ao21	tsmc035_typ	4 x	2 7 gates
ao22	tsmc035_typ	34 x	2 67 gates
ao221	tsmc035_typ	6 x	2 15 gates
aoi21	tsmc035_typ	97 x	1 120 gates
aoi22	tsmc035_typ	171 x	1 253 gates
aoi221	tsmc035_typ	87 x	2 172 gates
aoi222	tsmc035_typ	12 x	2 27 gates
aoi321	tsmc035_typ	6 x	2 13 gates
aoi322	tsmc035_typ	2 x	2 5 gates
fake_gnd	tsmc035_typ	1 x	1 1 fake_gnd
inv01	tsmc035_typ	1502 x	1 1142 gates
mux21	tsmc035_typ	28 x	2 51 gates
nand02	tsmc035_typ	712 x	1 712 gates
nand03	tsmc035_typ	39 x	1 48 gates
nand04	tsmc035_typ	75 x	1 111 gates
nor02	tsmc035_typ	606 x	1 606 gates
nor03	tsmc035_typ	41 x	1 51 gates
nor04	tsmc035_typ	84 x	1 124 gates
oai21	tsmc035_typ	135 x	1 167 gates

oai22	tsmc035_typ	252 x	1	373 gates
oai221	tsmc035_typ	96 x	2	190 gates
oai222	tsmc035_typ	19 x	2	42 gates
oai32	tsmc035_typ	4 x	2	7 gates
oai321	tsmc035_typ	3 x	2	7 gates
oai33	tsmc035_typ	2 x	2	4 gates
oai422	tsmc035_typ	3 x	3	8 gates
oai43	tsmc035_typ	1 x	2	2 gates
or02	tsmc035_typ	5 x	1	6 gates
or03	tsmc035_typ	9 x	2	14 gates
or04	tsmc035_typ	7 x	2	14 gates
xnor2	tsmc035_typ	1593 x	2	3043 gates
xor2	tsmc035_typ	16 x	2	34 gates

Number of ports : 64
Number of nets : 5707
Number of instances : 5676
Number of references to this view : 0

Total accumulated area :
Number of fake_gnd : 1
Number of gates : 7468
Number of accumulated instances : 5676

Delay Report for Auto-Optimized mode

Radix-2 Booth Encoding multiplier:

NAME	Critical Path Report			LOAD
	GATE	ARRIVAL		

multiplicand(0)/		0.00 0.00 dn		0.17
ix2894/Y	inv01	0.23 0.23 up		0.04
ix67/Y	or02	0.21 0.44 up		0.01
ix6120/Y	inv01	0.14 0.59 dn		0.03
ix6122/Y	inv01	0.57 1.16 up		0.10
ix2910/Y	xor2	0.55 1.71 dn		0.03
ix2928/Y	aoi32	0.29 1.99 up		0.03
ix363/Y	inv01	0.13 2.12 dn		0.01
ix2966/Y	mux21	0.34 2.46 up		0.03
ix367/Y	inv01	0.12 2.57 dn		0.01
ix3016/Y	mux21	0.34 2.91 up		0.03
ix371/Y	inv01	0.12 3.03 dn		0.01
ix3072/Y	mux21	0.33 3.37 up		0.03
ix375/Y	inv01	0.12 3.48 dn		0.01
ix3144/Y	mux21	0.34 3.82 up		0.03
ix379/Y	inv01	0.12 3.94 dn		0.01
ix3222/Y	mux21	0.33 4.27 up		0.03
ix383/Y	inv01	0.12 4.39 dn		0.01
ix3318/Y	mux21	0.34 4.73 up		0.03
ix387/Y	inv01	0.12 4.85 dn		0.01
ix3418/Y	mux21	0.33 5.18 up		0.03
ix391/Y	inv01	0.12 5.29 dn		0.01
ix3536/Y	mux21	0.34 5.63 up		0.03
ix395/Y	inv01	0.12 5.75 dn		0.01
ix3658/Y	mux21	0.33 6.08 up		0.03
ix399/Y	inv01	0.12 6.20 dn		0.01
ix3798/Y	mux21	0.34 6.54 up		0.03
ix403/Y	inv01	0.12 6.66 dn		0.01
ix3942/Y	mux21	0.33 6.99 up		0.03
ix407/Y	inv01	0.12 7.11 dn		0.01
ix4102/Y	mux21	0.34 7.44 up		0.03
ix411/Y	inv01	0.12 7.57 dn		0.01
ix4272/Y	mux21	0.35 7.92 up		0.04
ix421/Y	xnor2	0.23 8.15 dn		0.01
ix4268/Y	mux21	0.37 8.52 up		0.04
ix4298/Y	xnor2	0.32 8.84 dn		0.04
ix757/Y	xnor2	0.18 9.02 up		0.01
ix4294/Y	inv01	0.11 9.13 dn		0.01
ix761/Y	mux21	0.35 9.47 up		0.04

ix1067/Y	xnor2	0.32	9.79	dn	0.04
ix3007/Y	mux21	0.36	10.16	up	0.04
ix4534/Y	xor2	0.46	10.61	dn	0.01
ix3017/Y	mux21	0.35	10.96	up	0.04
ix3035/Y	xnor2	0.32	11.28	dn	0.04
ix3367/Y	mux21	0.36	11.64	up	0.04
ix3373/Y	xor2	0.46	12.10	dn	0.01
ix4674/Y	mux21	0.35	12.44	up	0.04
ix4712/Y	xnor2	0.32	12.76	dn	0.04
ix3703/Y	mux21	0.36	13.13	up	0.04
ix4850/Y	xor2	0.46	13.58	dn	0.01
ix3713/Y	mux21	0.35	13.93	up	0.04
ix3731/Y	xnor2	0.32	14.25	dn	0.04
ix4015/Y	mux21	0.36	14.61	up	0.04
ix4021/Y	xor2	0.46	15.07	dn	0.01
ix4966/Y	mux21	0.35	15.42	up	0.04
ix5002/Y	xnor2	0.32	15.74	dn	0.04
ix4303/Y	mux21	0.36	16.10	up	0.04
ix5118/Y	xor2	0.46	16.56	dn	0.01
ix4313/Y	mux21	0.35	16.90	up	0.04
ix4331/Y	xnor2	0.32	17.22	dn	0.04
ix4567/Y	mux21	0.36	17.58	up	0.04
ix4573/Y	xor2	0.46	18.04	dn	0.01
ix5214/Y	mux21	0.35	18.39	up	0.04
ix5252/Y	xnor2	0.32	18.71	dn	0.04
ix4807/Y	mux21	0.36	19.07	up	0.04
ix5346/Y	xor2	0.46	19.53	dn	0.01
ix4817/Y	mux21	0.35	19.88	up	0.04
ix4835/Y	xnor2	0.32	20.20	dn	0.04
ix5023/Y	mux21	0.36	20.56	up	0.04
ix5029/Y	xor2	0.46	21.01	dn	0.01
ix5420/Y	mux21	0.35	21.36	up	0.04
ix5456/Y	xnor2	0.32	21.68	dn	0.04
ix5215/Y	mux21	0.36	22.04	up	0.04
ix5530/Y	xor2	0.46	22.50	dn	0.01
ix5225/Y	mux21	0.35	22.85	up	0.04
ix5243/Y	xnor2	0.32	23.17	dn	0.04
ix5383/Y	mux21	0.36	23.53	up	0.04
ix5389/Y	xor2	0.46	23.98	dn	0.01
ix5584/Y	mux21	0.35	24.33	up	0.04
ix5620/Y	xnor2	0.32	24.65	dn	0.04
ix5527/Y	mux21	0.36	25.02	up	0.04
ix5672/Y	xor2	0.46	25.47	dn	0.01
ix5537/Y	mux21	0.35	25.82	up	0.04
ix5555/Y	xnor2	0.32	26.14	dn	0.04
ix5647/Y	mux21	0.36	26.50	up	0.04
ix5653/Y	xor2	0.46	26.96	dn	0.01
ix5704/Y	mux21	0.35	27.31	up	0.04
ix5675/Y	xnor2	0.32	27.63	dn	0.04
ix5740/Y	mux21	0.34	27.97	up	0.03
ix5743/Y	inv01	0.12	28.08	dn	0.01
ix5776/Y	mux21	0.36	28.44	up	0.04
ix5774/Y	xor2	0.45	28.89	dn	0.01
ix5801/Y	mux21	0.20	29.09	up	0.00
product(29)/		0.00	29.09	up	0.00
data arrival time			29.09		
data required time			not specified		

data required time			not specified		
data arrival time			29.09		

Radix-4 Booth Encoding multiplier:

Critical Path Report					
NAME	GATE	ARRIVAL		LOAD	
-----	-----	-----	-----	-----	-----
multiplicand(2)/		0.00	0.00	dn	0.15
ix21/Y	nor03	0.44	0.44	up	0.03
ix371/Y	ao21	0.39	0.83	up	0.04
ix3542/Y	inv01	0.25	1.08	dn	0.05

ix1998/Y	aoi222	0.39	1.47	up	0.03
ix2857/Y	nor03	0.34	1.81	dn	0.03
ix1804/Y	inv01	0.20	2.01	up	0.01
ix2861/Y	mux21	0.22	2.23	dn	0.04
ix1852/Y	mux21	0.39	2.62	up	0.04
ix3115/Y	xnor2	0.26	2.88	dn	0.02
ix3117/Y	xnor2	0.28	3.16	up	0.04
ix3133/Y	mux21	0.23	3.40	dn	0.04
ix2024/Y	mux21	0.37	3.76	up	0.03
ix3137/Y	inv01	0.12	3.89	dn	0.01
ix1722/Y	mux21	0.35	4.24	up	0.04
ix3145/Y	mux21	0.23	4.47	dn	0.04
ix2110/Y	mux21	0.38	4.85	up	0.04
ix3461/Y	xnor2	0.26	5.12	dn	0.02
ix2106/Y	xnor2	0.28	5.40	up	0.04
ix2154/Y	mux21	0.22	5.61	dn	0.03
ix3507/Y	inv01	0.19	5.80	up	0.01
ix2242/Y	mux21	0.23	6.03	dn	0.04
ix3515/Y	mux21	0.38	6.41	up	0.04
ix2424/Y	mux21	0.22	6.63	dn	0.03
ix3519/Y	inv01	0.19	6.81	up	0.01
ix2492/Y	mux21	0.23	7.04	dn	0.04
ix3527/Y	mux21	0.38	7.42	up	0.04
ix2696/Y	mux21	0.24	7.66	dn	0.04
ix3535/Y	mux21	0.38	8.04	up	0.04
ix2860/Y	mux21	0.24	8.28	dn	0.04
ix3543/Y	mux21	0.38	8.65	up	0.04
ix3004/Y	mux21	0.23	8.89	dn	0.04
ix3551/Y	mux21	0.38	9.27	up	0.04
ix3166/Y	mux21	0.22	9.49	dn	0.03
ix3555/Y	inv01	0.19	9.67	up	0.01
ix3234/Y	mux21	0.22	9.89	dn	0.03
ix3561/Y	nor02	0.34	10.23	up	0.03
ix3346/Y	nand02	0.19	10.42	dn	0.03
ix3743/Y	nor02	0.34	10.76	up	0.03
ix3416/Y	nand02	0.19	10.95	dn	0.03
ix3867/Y	nor02	0.34	11.29	up	0.03
ix3458/Y	inv01	0.10	11.39	dn	0.01
ix3911/Y	nor02	0.32	11.71	up	0.03
ix3473/Y	inv01	0.10	11.81	dn	0.01
ix3943/Y	nor02	0.28	12.09	up	0.02
ix3947/Y	xnor2	0.18	12.27	dn	0.00
product(31)/		0.00	12.27	dn	0.00
data arrival time			12.27		
data required time			not specified		

data required time			not specified		
data arrival time			12.27		

Radix-8 Booth Encoding multiplier:

NAME	Critical Path Report			LOAD
	GATE	ARRIVAL		

multiplicand(2)/		0.00	0.00 dn	0.28
ix195/Y	nor03	0.44	0.44 up	0.03
ix2458/Y	nand02	0.15	0.58 dn	0.03
ix203/Y	nor02	0.32	0.91 up	0.03
ix2662/Y	nand02	0.15	1.06 dn	0.03
ix353/Y	nor02	0.32	1.38 up	0.03
ix2938/Y	nand02	0.15	1.53 dn	0.03
ix763/Y	nor02	0.32	1.85 up	0.03
ix3276/Y	nand02	0.15	2.00 dn	0.03
ix1565/Y	nor02	0.32	2.32 up	0.03
ix3678/Y	nand02	0.15	2.47 dn	0.03
ix2143/Y	nor02	0.32	2.80 up	0.03
ix4172/Y	nand02	0.15	2.94 dn	0.03
ix3079/Y	nor02	0.47	3.42 up	0.05
ix3087/Y	aoi21	0.30	3.72 dn	0.04
ix6098/Y	inv01	0.35	4.07 up	0.04

ix4346/Y	or02	0.34	4.41	up	0.03
ix4344/Y	oai21	0.17	4.58	dn	0.01
ix3103/Y	aoi21	0.36	4.94	up	0.03
ix3111/Y	xnor2	0.25	5.19	dn	0.04
ix4332/Y	xnor2	0.26	5.45	up	0.04
ix4324/Y	xnor2	0.23	5.68	dn	0.04
ix4318/Y	xnor2	0.26	5.94	up	0.04
ix4314/Y	xnor2	0.23	6.17	dn	0.04
ix3151/Y	xnor2	0.26	6.43	up	0.04
ix3159/Y	xnor2	0.23	6.66	dn	0.04
ix4296/Y	xnor2	0.26	6.92	up	0.04
ix4288/Y	xnor2	0.23	7.15	dn	0.04
ix4282/Y	xnor2	0.26	7.41	up	0.04
ix4278/Y	xnor2	0.23	7.64	dn	0.04
ix3199/Y	xnor2	0.29	7.92	up	0.04
ix4178/Y	aoi22	0.12	8.04	dn	0.01
ix3969/Y	nand03	0.27	8.31	up	0.02
ix4164/Y	xnor2	0.29	8.60	dn	0.04
ix4424/Y	mux21	0.33	8.93	up	0.03
ix4851/Y	inv01	0.12	9.05	dn	0.01
ix4702/Y	mux21	0.33	9.38	up	0.03
ix4855/Y	inv01	0.12	9.50	dn	0.01
ix4976/Y	mux21	0.33	9.84	up	0.03
ix4859/Y	inv01	0.09	9.93	dn	0.01
ix5058/Y	nand02	0.21	10.13	up	0.03
ix4991/Y	xnor2	0.24	10.37	dn	0.02
ix5054/Y	xnor2	0.28	10.65	up	0.04
ix5128/Y	mux21	0.23	10.88	dn	0.04
ix5225/Y	mux21	0.38	11.26	up	0.04
ix5262/Y	mux21	0.22	11.48	dn	0.03
ix5357/Y	nor02	0.34	11.83	up	0.03
ix5362/Y	nand02	0.19	12.01	dn	0.03
ix5481/Y	inv01	0.14	12.15	up	0.01
ix5400/Y	nand02	0.16	12.31	dn	0.03
ix5669/Y	nor02	0.33	12.64	up	0.03
ix5482/Y	nand02	0.19	12.84	dn	0.03
ix5815/Y	nor02	0.34	13.17	up	0.03
ix5542/Y	nand02	0.19	13.36	dn	0.03
ix5867/Y	inv01	0.14	13.50	up	0.01
ix5568/Y	nand02	0.16	13.66	dn	0.03
ix5925/Y	nor02	0.20	13.86	up	0.01
ix5929/Y	or02	0.17	14.02	up	0.00
product(30) /		0.00	14.02	up	0.00
data arrival time			14.02		
data required time			not specified		

data required time			not specified		
data arrival time			14.02		

Radix-16 Booth Encoding multiplier:

Critical Path Report					
NAME	GATE	ARRIVAL			LOAD
-----	-----	-----	-----	-----	-----
multiplicand(2) /		0.00	0.00	dn	0.41
ix773/Y	nor03	0.44	0.44	up	0.03
ix4530/Y	nand02	0.15	0.58	dn	0.03
ix781/Y	nor02	0.32	0.91	up	0.03
ix4768/Y	nand02	0.15	1.06	dn	0.03
ix953/Y	nor02	0.32	1.38	up	0.03
ix5142/Y	nand02	0.15	1.52	dn	0.03
ix1077/Y	nor02	0.32	1.85	up	0.03
ix5836/Y	nand02	0.15	1.99	dn	0.03
ix2735/Y	nor02	0.32	2.31	up	0.03
ix6488/Y	nand02	0.15	2.46	dn	0.03
ix2923/Y	nor02	0.32	2.78	up	0.03
ix7428/Y	nand02	0.18	2.96	dn	0.04
ix7424/Y	oai21	0.24	3.20	up	0.01
ix5347/Y	inv01	0.15	3.35	dn	0.02
ix11868/Y	inv01	0.29	3.65	up	0.04

ix11684/Y	inv01	0.27	3.91	dn	0.05
ix11798/Y	inv01	0.32	4.24	up	0.04
ix7584/Y	or02	0.39	4.62	up	0.04
ix7580/Y	oai21	0.30	4.92	dn	0.05
ix7652/Y	xor2	0.62	5.54	up	0.03
ix7650/Y	xnor2	0.22	5.76	dn	0.04
ix5361/Y	nor02	0.35	6.11	up	0.03
ix8732/Y	inv01	0.10	6.21	dn	0.01
ix6809/Y	nor02	0.32	6.53	up	0.03
ix6945/Y	xnor2	0.30	6.83	dn	0.04
ix6953/Y	xnor2	0.27	7.10	up	0.04
ix8716/Y	xnor2	0.23	7.33	dn	0.04
ix8708/Y	xnor2	0.26	7.60	up	0.04
ix8702/Y	xnor2	0.23	7.82	dn	0.04
ix8698/Y	xnor2	0.26	8.08	up	0.04
ix6993/Y	xnor2	0.23	8.32	dn	0.04
ix7001/Y	xnor2	0.27	8.58	up	0.04
ix8680/Y	xnor2	0.23	8.81	dn	0.04
ix8672/Y	xnor2	0.26	9.08	up	0.04
ix8666/Y	xnor2	0.26	9.34	dn	0.05
ix7731/Y	oai22	0.20	9.54	up	0.01
ix8594/Y	aoi221	0.17	9.72	dn	0.01
ix7795/Y	nand04	0.28	9.99	up	0.02
ix8522/Y	xnor2	0.30	10.29	dn	0.04
ix9120/Y	mux21	0.33	10.62	up	0.03
ix9515/Y	inv01	0.12	10.74	dn	0.01
ix9714/Y	mux21	0.33	11.08	up	0.03
ix9843/Y	xnor2	0.26	11.33	dn	0.02
ix9845/Y	xnor2	0.28	11.61	up	0.04
ix9858/Y	mux21	0.23	11.84	dn	0.04
ix10125/Y	mux21	0.38	12.23	up	0.04
ix10084/Y	mux21	0.22	12.45	dn	0.03
ix10129/Y	inv01	0.19	12.64	up	0.01
ix10226/Y	mux21	0.21	12.85	dn	0.03
ix10133/Y	inv01	0.17	13.02	up	0.01
ix10344/Y	mux21	0.21	13.24	dn	0.03
ix10283/Y	nor02	0.34	13.58	up	0.03
ix10518/Y	inv01	0.10	13.68	dn	0.01
ix10429/Y	nor02	0.32	14.00	up	0.03
ix10614/Y	inv01	0.10	14.10	dn	0.01
ix10571/Y	nor02	0.32	14.42	up	0.03
ix10666/Y	inv01	0.10	14.53	dn	0.01
ix10655/Y	nor02	0.32	14.85	up	0.03
ix10696/Y	inv01	0.10	14.95	dn	0.01
ix10731/Y	nor02	0.32	15.27	up	0.03
ix10899/Y	xor2	0.39	15.66	dn	0.00
product (27) /		0.00	15.66	dn	0.00
data arrival time			15.66		
data required time			not specified		

data required time			not specified		
data arrival time			15.66		

Radix-32 Booth Encoding multiplier:

NAME	Critical Path Report			LOAD
	GATE	ARRIVAL		

multiplicand (2) /		0.00	0.00 dn	0.48
ix3335/Y	nor03	0.44	0.44 up	0.03
ix7540/Y	nand02	0.15	0.58 dn	0.03
ix3343/Y	nor02	0.32	0.91 up	0.03
ix7534/Y	nand02	0.15	1.06 dn	0.03
ix3351/Y	nor02	0.32	1.38 up	0.03
ix7528/Y	nand02	0.15	1.52 dn	0.03
ix3359/Y	nor02	0.32	1.85 up	0.03
ix7522/Y	nand02	0.15	1.99 dn	0.03
ix3829/Y	nor02	0.32	2.31 up	0.03
ix7548/Y	nand02	0.15	2.46 dn	0.03
ix4025/Y	nor02	0.32	2.78 up	0.03

ix4033/Y	aoi21	0.19	2.97	dn	0.02
ix20168/Y	inv01	0.43	3.40	up	0.07
ix20062/Y	inv01	0.28	3.68	dn	0.05
ix19744/Y	inv01	0.33	4.01	up	0.04
ix20290/Y	inv01	0.27	4.28	dn	0.05
ix19808/Y	inv01	0.30	4.58	up	0.04
ix18446/Y	inv01	0.20	4.78	dn	0.03
ix12370/Y	oai21	0.26	5.04	up	0.03
ix5641/Y	aoi21	0.32	5.35	dn	0.06
ix6733/Y	xnor2	0.30	5.65	up	0.04
ix12360/Y	xnor2	0.23	5.89	dn	0.04
ix6743/Y	nor02	0.37	6.26	up	0.03
ix13302/Y	xnor2	0.30	6.56	dn	0.04
ix13298/Y	xnor2	0.26	6.82	up	0.04
ix10887/Y	xnor2	0.23	7.05	dn	0.04
ix10895/Y	xnor2	0.27	7.32	up	0.04
ix10903/Y	xnor2	0.23	7.56	dn	0.04
ix10911/Y	xnor2	0.26	7.82	up	0.04
ix13266/Y	xnor2	0.23	8.05	dn	0.04
ix15560/Y	xnor2	0.33	8.37	up	0.06
ix19692/Y	inv01	0.22	8.59	dn	0.04
ix13202/Y	aoi221	0.40	8.99	up	0.02
ix13663/Y	nand04	0.16	9.15	dn	0.01
ix13186/Y	nor04	0.57	9.72	up	0.02
ix13667/Y	xnor2	0.36	10.07	dn	0.04
ix18277/Y	xnor2	0.24	10.32	up	0.02
ix13174/Y	xnor2	0.29	10.61	dn	0.04
ix10948/Y	mux21	0.33	10.93	up	0.03
ix18447/Y	inv01	0.12	11.05	dn	0.01
ix18256/Y	mux21	0.33	11.39	up	0.03
ix18451/Y	inv01	0.12	11.51	dn	0.01
ix10944/Y	mux21	0.33	11.84	up	0.03
ix18455/Y	inv01	0.12	11.96	dn	0.01
ix18266/Y	mux21	0.33	12.29	up	0.03
ix18459/Y	inv01	0.12	12.42	dn	0.01
ix10940/Y	mux21	0.33	12.75	up	0.03
ix18463/Y	inv01	0.12	12.87	dn	0.01
ix18288/Y	mux21	0.33	13.20	up	0.03
ix18467/Y	inv01	0.12	13.32	dn	0.01
ix10936/Y	mux21	0.33	13.66	up	0.03
ix18471/Y	inv01	0.12	13.78	dn	0.01
ix18298/Y	mux21	0.33	14.11	up	0.03
ix18475/Y	inv01	0.12	14.23	dn	0.01
ix7508/Y	mux21	0.34	14.56	up	0.03
ix18479/Y	inv01	0.12	14.69	dn	0.01
ix18308/Y	mux21	0.33	15.02	up	0.03
ix18483/Y	inv01	0.09	15.11	dn	0.01
ix7504/Y	nand02	0.21	15.31	up	0.03
ix18615/Y	inv01	0.09	15.40	dn	0.01
ix18340/Y	nand02	0.20	15.60	up	0.03
ix18747/Y	inv01	0.09	15.69	dn	0.01
ix7500/Y	nand02	0.16	15.85	up	0.02
ix19007/Y	nor02	0.15	15.99	dn	0.01
ix19013/Y	aoi21	0.11	16.11	up	0.00
product (26) /		0.00	16.11	up	0.00
data arrival time			16.11		
data required time			not specified		

data required time			not specified		
data arrival time			16.11		
