KLE Dr. M. S. SHESGIRI
COLLEGE OF ENGINEERING AND TECHNOLOGY
UDYAMBAGH, BELGAVI

DEPARTMENT OF
ELECTRONICS AND COMMUNICATION

Project on

Design and Development of High - speed Multiplier for DSP Applications

by

Vallabh Bhat	*[USN: 2KL19EC112]*
Praful Naikar	*[USN: 2KL19EC054]*
Sakshi Devale	*[USN: 2KL19EC120]*
Aditya Ingale	*[USN: 2KL19EC128]*

Guide

Prof. Kunjan D Shinde

CONTENTS

INTRODUCTION

- Multiplication is the key circuit block in modern day chips. This block is systems, controllers, filters, signal processors and various logic blocks as well.
- Since technology is updating every moment it is our prime need to implement fast multiplier in modern days VLSI chips used.
- In many Digital Signal Processing (DSP) applications, including convolution, Fast Fourier Transform (FFT), filtering, and in microprocessors in their arithmetic and logic unit, multiplication-based operations like multiply and Accumulate (MAC) and inner product are among the frequently used Computation-Intensive Arithmetic Functions (CIAF).

LITERATURE REVIEW

1.(Heidarpur and Mirhassani 2021)

Title: An Efficient and High-Speed Overlap-Free Karatsuba-Based Finite-Field Multiplier for FGPA Implementation.

Inferred: Proposes a novel hardware architecture for efficient field-programmable gate array (FPGA) implementation

2.(Behl, Gokhale, and Sharma 2020)

Title: Low-Delay FPGA-Based Implementation of Finite Field Multiplier

LITERATURE REVIEW (contd..)

3.(Valls and Boemo 2003)

Title: Efficient FPGAImplementation of Two's Complement Digit- Serial/Parallel Multiplier

Inferred: Efficient implementation of digit- serial/parallel multipliers on 4-input look-up table

4.(Ullah et al. 2022)

Title: Redundant Binary Signed Digit (RBSD) Booth Multiplier.

LITERATURE REVIEW (contd..)

5.(T et al. 2023)

Title: Flexible and Scalable FPGA- Oriented Design of Multipliers for Large Binary Polynomial

Inferred: work presented a flexible and scalable template architec ture for the hardware implementation of large binary poly nomial multipliers

6.(Rashidi, Farashahi, and Sayedi 2014)

Title: Throughput/Area Efficient ECC Processor using Montgomery Point Multiplication on FPGA

PROBLEM STATEMENT

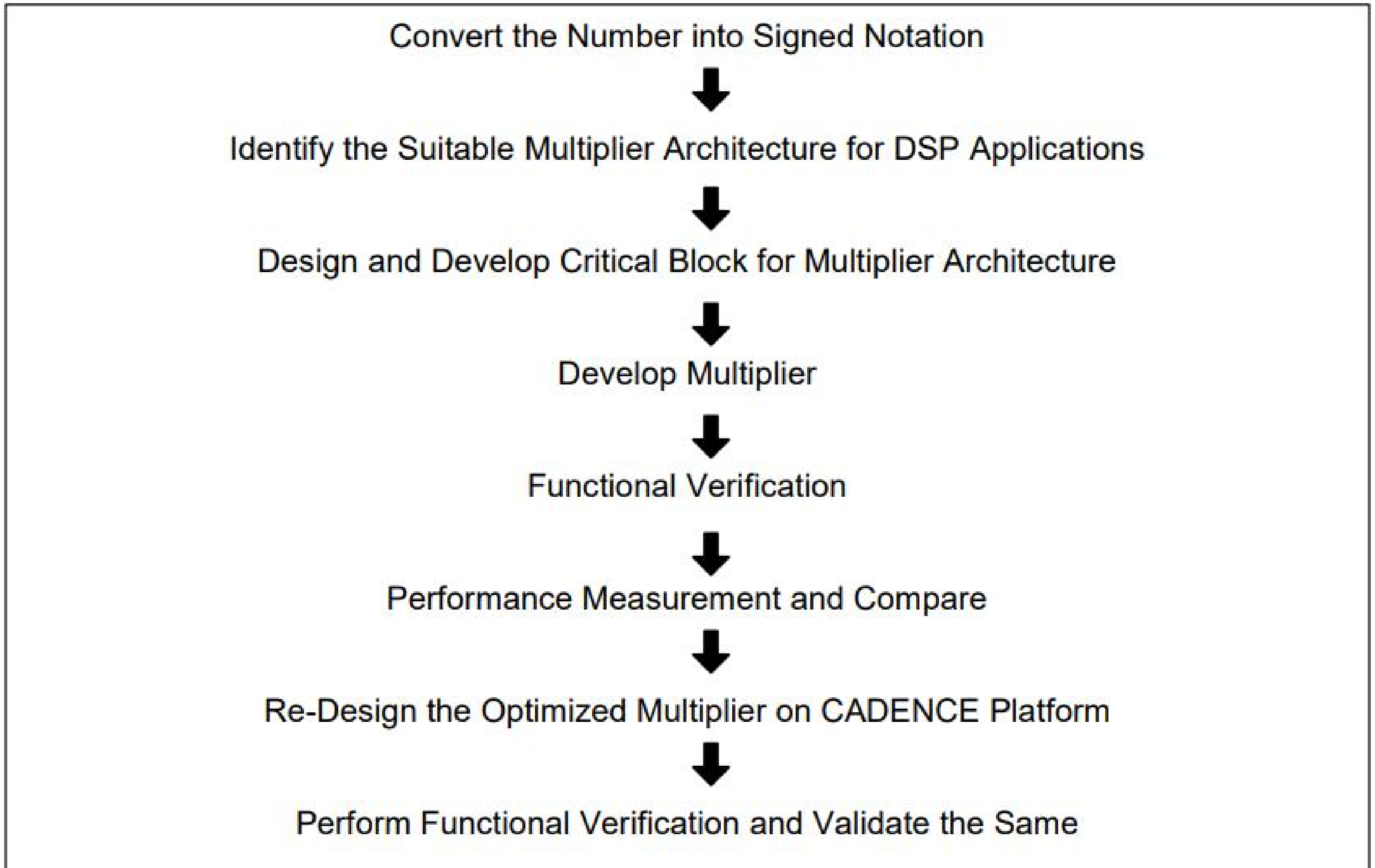
- ✓ Typical DSP algorithms like filtering or transformations involve multiplication as a primary process to generate a desired output.

- ✓ Multiplication being a critical block requires larger time to compute and produce results. Improving the performance of a multiplier improves the overall performance of a DSP block.

OBJECTIVES

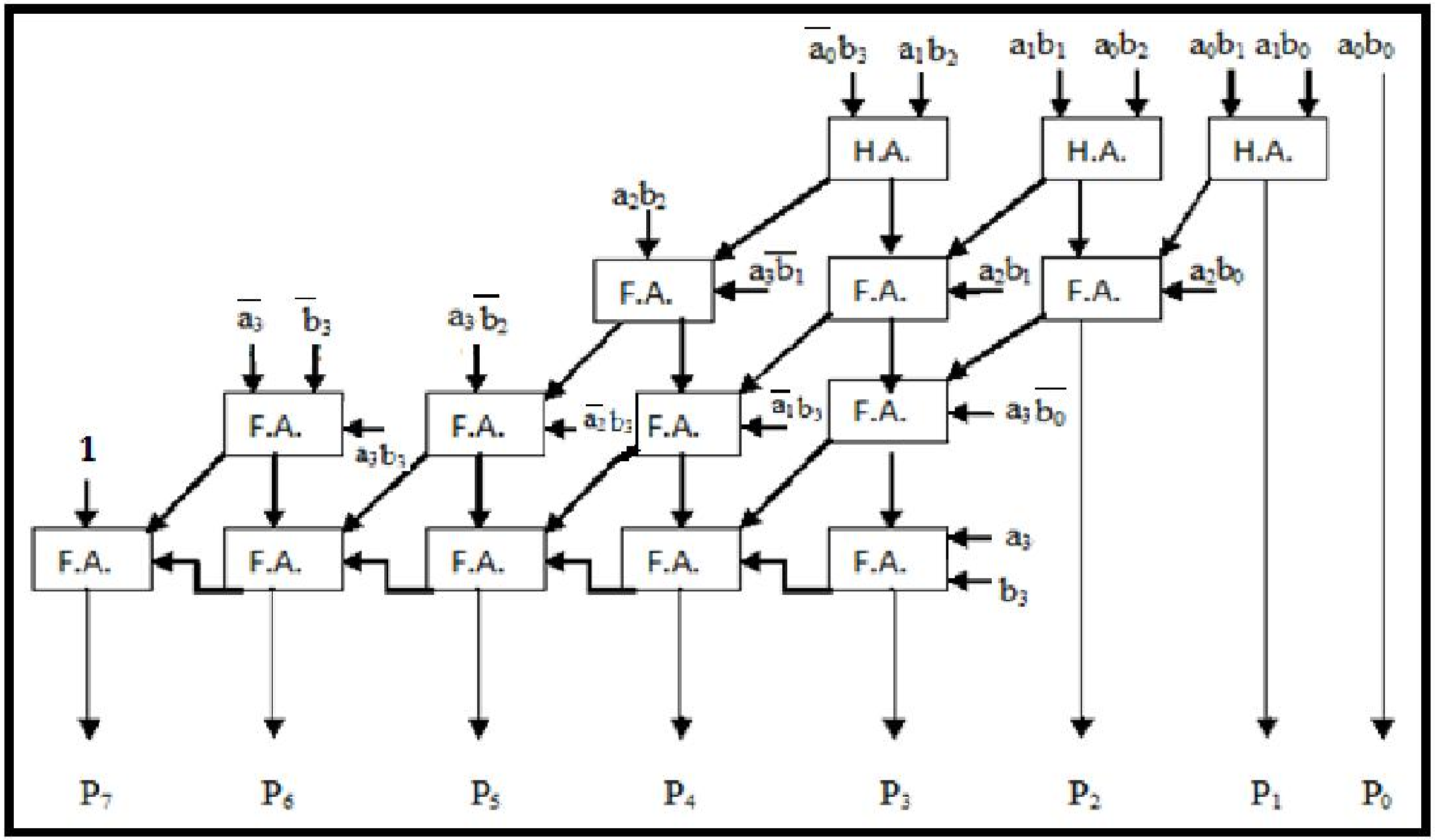
- ✓ To design and develop a high speed multiplier block for DSP application.
- ✓ To analyse the power and delay characteristics on FPGA level using Xilinx ISE/Vivado Design Suits.
- ✓ To design and analyse the schematic of optimized high speed multiplier on Cadence Virtuoso platform and validate the performance matric.

METHODOLOGY


Convert the Number into Signed Notation
Identify the Suitable Multiplier Architecture for DSP Applications
Design and Develop Critical Block for Multiplier Architecture
Develop Multiplier
Functional Verification
Performance Measurement and Compare
Re-Design the Optimized Multiplier on CADENCE Platform
Perform Functional Verification and Validate the Same

Fig> Explains the methodology of the project

ARCHITECTURE OF BAUGH-WOOLEY MULTIPLIER



ARCHITECTURE OF BAUGH-WOOLEY MULTIPLIER (CONTD..)

Multiplying two 2's compliment numbers

The Baugh-Wooley multiplication algorithm is an efficient way to handle the sign bits. This technique has been developed in order to design regular multipliers, suited for 2's-complement numbers. Let us consider two n-bit numbers, A and B, to be multiplied. A and B can be represented as

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \tag{1}$$

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \tag{2}$$

Where the a_i's and b_i's are the bits in A and B, respectively, and a_{n-1} and b_{n-1} are the sign bits.

The product, $P = A \times B$, is then given by the following equation:

$$\begin{aligned} P &= A \times B \\ &= \left(-a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i\right) \times \left(-b_{n-1}2^{n-1} + \sum_{j=0}^{n-2} b_j 2^j\right) \\ &= a_{n-1}b_{n-1}2^{2n-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} \\ &\quad -2^{n-1}\sum_{i=0}^{n-2} a_i b_{n-1} 2^i - 2^{n-1}\sum_{j=0}^{n-2} a_{n-1} b_j 2^j \end{aligned} \tag{3}$$

ARCHITECTURE OF BAUGH-WOOLEY MULTIPLIER (CONTD..)

Equation (3) indicates that the final product is obtained by subtracting the last two *positive terms* from the first two terms.

Baugh-Wooley multiplication algorithm

Rather than do a subtraction operation, we can obtain the 2's complement of the last two term and add all terms to get the final product.

The last two terms are $n-1$ bits each that extend in binary weight from position 2^{n-1} up to 2^{2n-3}. On the other hand, the final product is $2n$ bits and extends in binary weight from 2^0 up to 2^{2n-1}.

We pad each of the last two terms in Equation (3) with zeros to obtain a $2n$-bit number to be able to add them to the other terms. The padded terms extend in binary weight from 2^0 up to 2^{2n-1}.

Assuming X is one of the last two terms we can represent it with zero padding as

$$X = -0 \times 2^{2n-1} + 0 \times 2^{2n-2} + 2^{n-1} \sum_{i=0}^{n-2} x_i 2^i + \sum_{j=0}^{n-2} 0 \times 2^j \qquad (4)$$

ARCHITECTURE OF BAUGH-WOOLEY MULTIPLIER (CONTD..)

The above equation gives the *value* of X due to the fact that a negative value is associated with the MSB.

When we *store* X in a register, the negative sign at MSB is not used since X is stored as a *binary pattern*. Thus partial product X is, therefore, represented by

bit position	$2n-1$	$2n-2$	$2n-3$	$2n-4$	$\cdots$	n	$n-1$	$n-2$	$n-3$	$\cdots$	0
bit value	0	0	x_{n-2}	x_{n-3}	$\cdots$	x_1	x_0	0	0	$\cdots$	0

The two's complement of X is obtained by complimenting all bits in the above equation and adding '1' at the LSB:

bit position	$2n-1$	$2n-2$	$2n-3$	$2n-4$	$\cdots$	n	$n-1$	$n-2$	$n-3$	$\cdots$	0
bit value	1	1	x_{n-2}	x_{n-3}	$\cdots$	x_1	x_0	1	1	$\cdots$	$1+1$

Adding the '1' at LSB will result in the new pattern for $-X$ as

bit position	$2n-1$	$2n-2$	$2n-3$	$2n-4$	$\cdots$	n	$n-1$	$n-2$	$n-3$	$\cdots$	0
bit value	1	1	x_{n-2}	x_{n-3}	$\cdots$	x_1	x_0+1	0	0	$\cdots$	0

Assuming the last two terms are expressed as X and Y, then adding $-X$ to $-Y$ amounts to adding the following two bit patterns:

ARCHITECTURE OF BAUGH-WOOLEY MULTIPLIER (CONTD..)

bit position	$2n-1$	$2n-2$	$2n-3$	$2n-4$	$\cdots$	n	$n-1$	$n-2$	$n-3$	$\cdots$	0
$-X$	1	1	x_{n-2}	x_{n-3}	$\cdots$	x_1	x_0+1	0	0	$\cdots$	0
$+(-Y)$	1	1	y_{n-2}	y_{n-3}	$\cdots$	y_1	y_0+1	0	0	$\cdots$	0

The '1' pattern at most significant bits transforms into

bit position	$2n-1$	$2n-2$
	1	1
+	1	1
	1	0

Similarly, the '1' pattern at position $n-1$ becomes

bit position	n	$n-1$
		1
+		1
	1	0

ARCHITECTURE OF BAUGH-WOOLEY MULTIPLIER (CONTD..)

The final product $P = A \times B$ in Equation (3) becomes:

$$\begin{aligned} P \;=\; & a_{n-1}b_{n-1}2^{2n-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} + \\ & 2^{n-1}\sum_{i=0}^{n-2} \overline{b_{n-1}a_i}\,2^i + 2^{n-1}\sum_{j=0}^{n-2} \overline{a_{n-1}b_j}\,2^j + \\ & -2^{2n-1} + 2^n \end{aligned} \qquad (5)$$

Let us assume that A and B are 4-bit binary numbers, then the product $P = A \times B$ is 8-bits long and is given by

$$\begin{aligned} P \;=\; & a_3b_3 2^6 + \sum_{i=0}^{2}\sum_{j=0}^{2} a_i b_j 2^{i+j} + \\ & 2^3\sum_{i=0}^{2} \overline{b_3 a_i}\,2^i + 2^3\sum_{j=0}^{2} \overline{a_3 b_j}\,2^j + \\ & -2^7 + 2^4 \end{aligned} \qquad (6)$$

Figure 1 shows the implementation of the Baugh-Wooley multiplier.

The basic Baugh-Wooley cells are shown in Figure 2.

ARCHITECTURE OF BAUGH-WOOLEY MULTIPLIER (CONTD..)

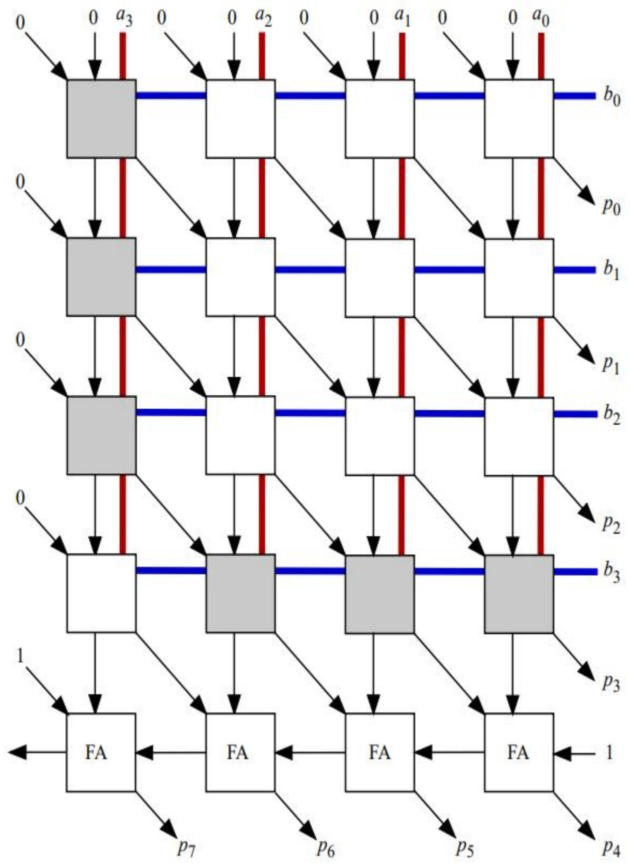


Fig. *Block Diagram of 4x4 Baugh-Wooley Multiplier*

SOFTWARE REQUIREMENTS

- ✓ Xilinx Vivado / ISE Design Suit.
- ✓ Cadence Virtuoso.

HARDWARE:

Family : Artix 7
Device : XC7A100T
Package : FGG676
Speed Grade : -3
Hardware : xc7a100t-3fgg676
Design Environment : ISE Design Suite 14.7

WORK CARRIED OUT

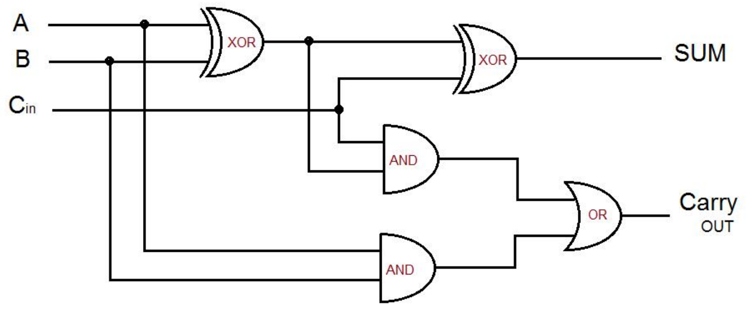


fig. Generic based full adder

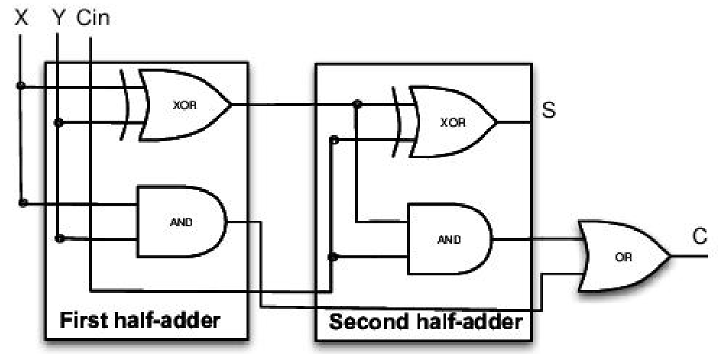


fig. Half adder based full adder

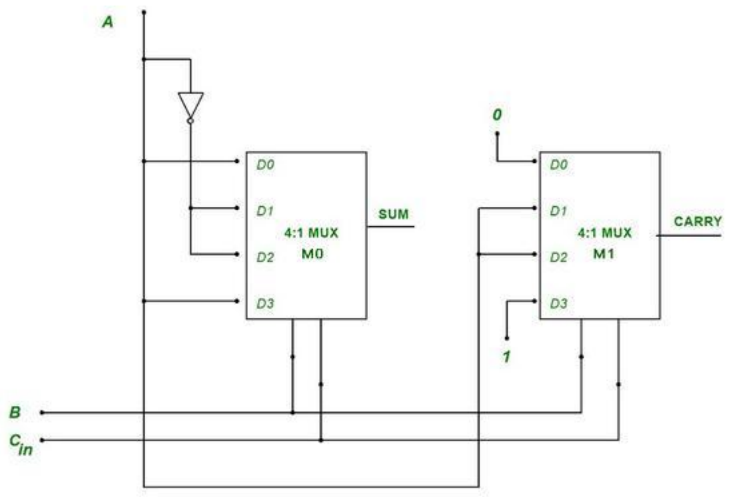


fig. MUX based full adder

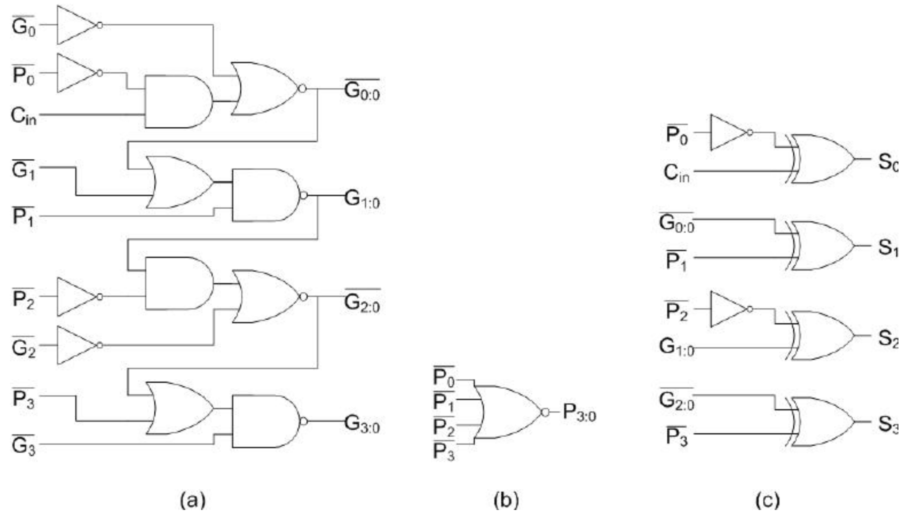


fig. PG based full adder

WORK CARRIED OUT (contd..)

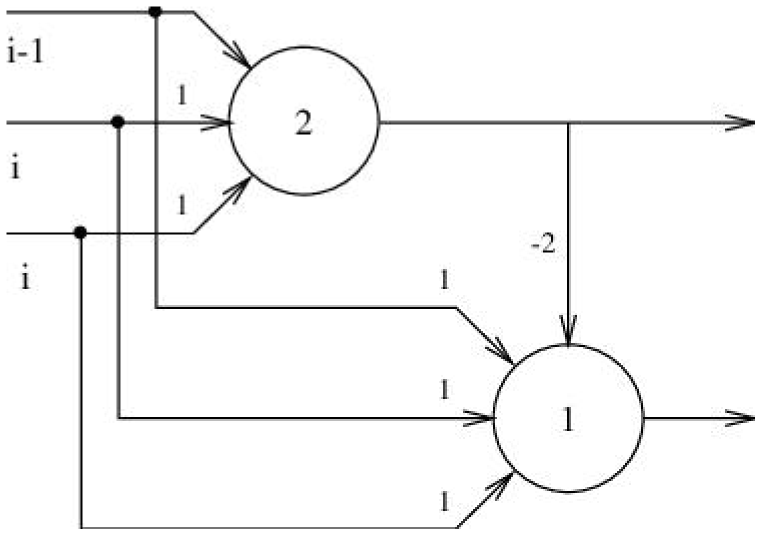


fig. Threshold logic based full adder

WORK CARRIED OUT (contd..)

	Generic Method	MUX based approach	Propagate and Generate	Half adder based approach
Slice LUT's	2	2	2	2
Occupied Slices	2	2	2	2
Combinational Path delay	3.813ns	3.813ns	3.813ns	3.813ns
Macros	1 bit 3 input xor -2	1 bit 2 input xor-2	1 bit 2 input xor-2	1 bit 2 input xor-2

Table1> Full adder design analysis tabulation

Design comparison post syntheses and implementation:

	Generic Addition	MUX based Addition	Propagate and Generate	Half adder based approach	Threshold Logic
Slice LUT's	16	13	14	14	114
Occupied Slices	9	5	5	5	37
Combinational Path delay	3.070ns(0.389ns logic, 2.681ns route) (12.7% logic, 87.3% route)	4.373ns(0.583ns logic, 3.790ns route) (13.3% logic, 86.7% route)	2.520ns(0.292ns logic, 2.228ns route) (11.6% logic, 88.4% route)	3.720ns(0.352ns logic, 3.368ns route) (12.2% logic, 87.8% route)	7.337ns(3.467ns logic, 3.870ns route) (47.3% logic, 52.7% route)
Macros	1-bit 2 input xor - 27	bit 2-to-1 multiplexer - 301-bit 2 input xor -15	1-bit 2-to-1 multiplexer-151-bit 2 input xor -30	1-bit 2 input xor -30	32-bit comparator lessequal-3032-bit 2 input xor-30

Table2> Computational Results of 4x4 Baugh-wooley Multiplier

WORK CARRIED OUT (contd..)

	Generic Addition	MUX based Addition	Propagate and Generate	Half adder based approach	Threshold Logic
Slice LUT's	78	77	78	78	468
Occupied Slices	27	24	34	42	151
Combinational Path delay	8.170ns(1.068ns logic, 7.102ns route) (13.1% logic, 86.9% route)	8.914ns(1.165ns logic, 7.749ns route) (13.1% logic, 86.9% route)	7.270ns(1.068ns logic, 6.202ns route) (14.3% logic, 85.7% route)	8.170ns(1.068ns logic, 7.102ns route) (13.1% logic, 86.9% route)	14.902ns(7.127ns logic, 7.776ns route) (47.8% logic, 52.2% route)
Macros	1-bit 2 input xor - 118	1-bit 2-to-1 multiplexer-1181-bit 2 input xor -59	1-bit 2-to-1 multiplexer-551-bit 2 input xor -118	1-bit 2 input xor - 118	32-bit comparator lessequal-11832-bit 2 input xor-118

Table3> Computational Results of 8x8 Baugh-wooley Multiplier

WORK CARRIED OUT (contd..)

	Generic Addition	MUX based Addition	Propagate and Generate	Half adder based approach	Threshold Logic
Slice LUT's	372	336	341	341	1970
Occupied Slices	201	185	184	182	644
Combinational Path delay	17.240ns(2.232ns logic, 15.008ns route) (12.9% logic, 87.1% route)	20.238ns(2.620ns logic, 17.618ns route) (12.9% logic, 87.1% route)	17.240ns(2.232ns logic, 15.008ns route) (12.9% logic, 87.1% route)	17.240ns(2.232ns logic, 15.008ns route) (12.9% logic, 87.1% route)	30.924ns(14.447ns logic, 16.477ns route) (46.7% logic, 53.3% route)
Macros	1-bit 2-to-1 multiplexer-486	1-bit 2-to-1 multiplexer-4861-bit 2 input xor -243	1-bit 2-to-1 multiplexer- 2461-bit 2 input xor -486	1-bit 2-to-1 multiplexer-486	32-bit comparator lessequal-48632-bit 2 input xor-486

Table4> Computational Results of 16x16 Baugh-wooley Multiplier

WORK CARRIED OUT (contd..)

Design comparison post syntheses and implementation for standardized adder designs

	4 bitKSA	8 bitKSA	16bitKSA
Slice LUT’s	4	10	16
Occupied Slices	2	5	9
Combinational Path delay	1.424ns(0.195ns logic, 1.229ns route) (13.7% logic, 86.3% route)	2.951ns(0.668ns logic, 2.283ns route) (22.6% logic, 77.4% route)	4.678ns(0.933ns logic, 3.748ns route) (23.2% logic, 76.8% route)
Macros	bit xor2 - 44-bit xor2 - 1	1-bit xor2 - 16	1-bit xor2 - 32

Table5> Performance Analysis of Kogge Stone Adder

	4bitBKA	8bitBKA	16bitBKA
Slice LUT’s	4	10	16
Occupied Slices	2	5	9
Combinational Path delay	1.567ns(0.122ns logic, 1.445ns route) (12.1% logic, 87.9% route)	2.987ns(0.560ns logic, 2.427ns route) (21.4% logic, 78.6% route)	4.753ns(0.832ns logic, 3.921ns route) (22.8% logic, 77.2% route)
Macros	bit xor2 - 44-bit xor2 - 1	1-bit xor2 - 16	1-bit xor2 - 32

Table6> Performance Analysis of Brent Kung Adder

WORK CARRIED OUT (contd..)

Implementation Details

	4bitBW	8bitBW	16bitBW
Slice LUT's	13	78	368
Occupied Slices	5	44	209
Combinational Path delay	4.231ns(0.679ns logic, 3.552ns route) (14.1% logic, 85.9% route)	8.390ns(1.045ns logic, 7.345ns route) (12.9% logic, 87.1% route)	17.245ns(2.351ns logic, 14.894ns route) (13.6% logic, 86.4% route)
Macros	1-bit 2 input xor - 15	1-bit 2 input xor - 118	1-bit 2-to-1 multiplexer- 486

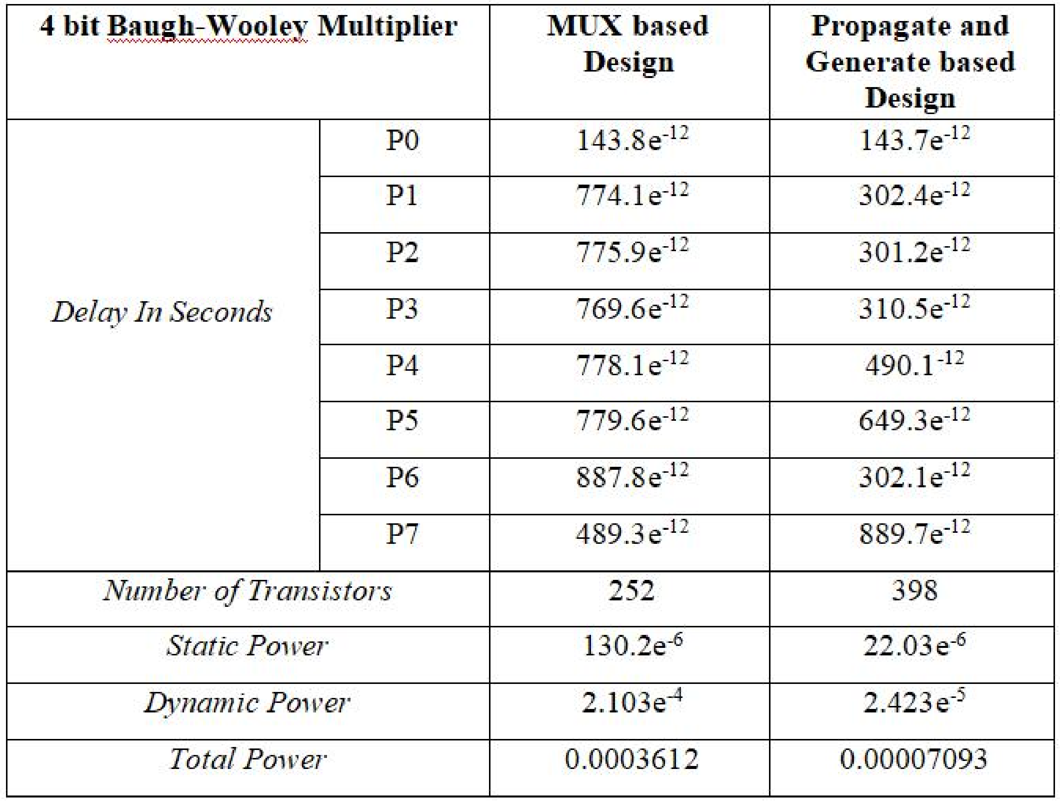
Table7> Computational Results of Baugh-Wooley Using Kogge Stone Adder (KSA)

	4bitBW	8bitBW	16bitBW
Slice LUT's	16	79	386
Occupied Slices	7	23	213
Combinational Path delay	3.103ns(0.283ns logic, 2.820ns route) (11.8% logic, 88.2% route)	8.039ns(1.257ns logic, 6.782ns route) (13.7% logic, 86.3% route)	17.340ns(2.371ns logic, 14.969ns route) (13.4% logic, 86.6% route)
Macros	1-bit 2 input xor - 27	1-bit 2 input xor -59	1-bit 2-to-1 multiplexer- 2761-bit 2 input xor -486

Table8> Computational Results of Baugh-Wooley Using Brent Kung Adder (BKA)

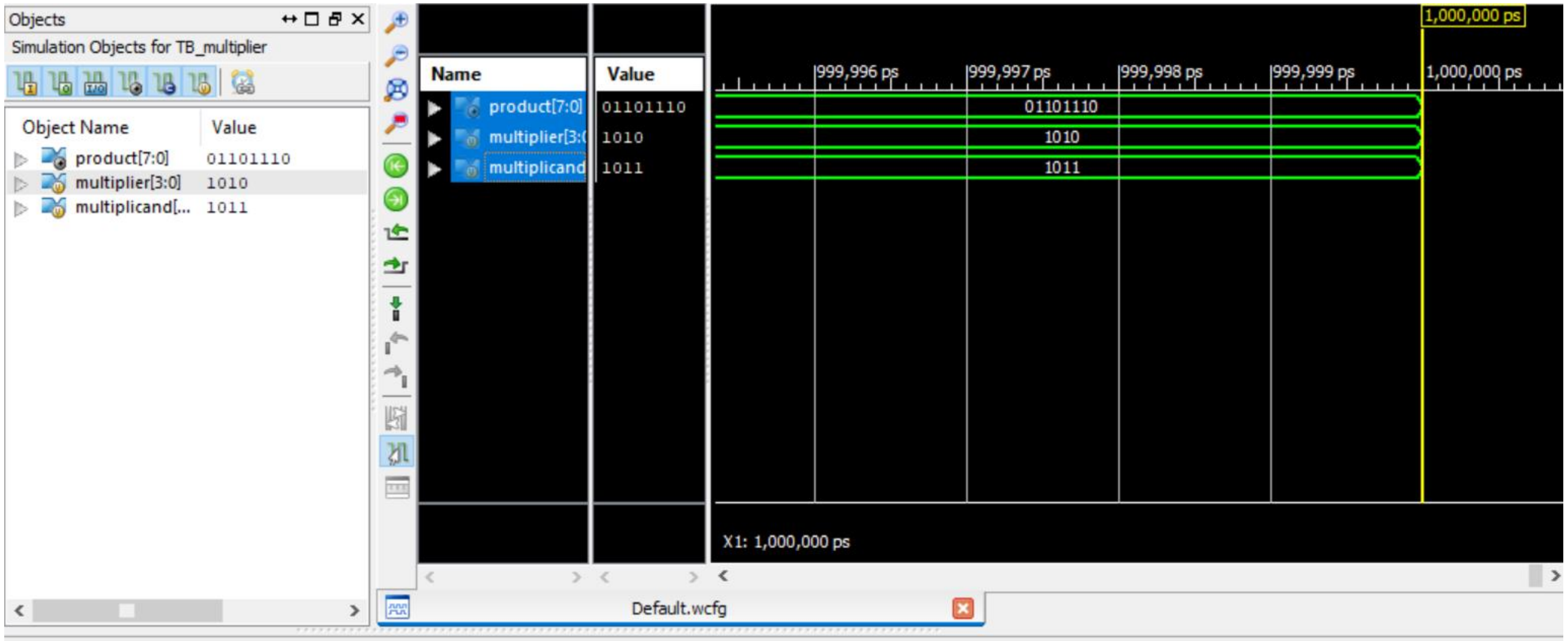
WORK CARRIED OUT (contd..)

Performance Tabulation on Cadence Virtuoso



4 bit Baugh-Wooley Multiplier		**MUX based Design**	**Propagate and Generate based Design**
Delay In Seconds	P0	$143.8e^{-12}$	$143.7e^{-12}$
	P1	$774.1e^{-12}$	$302.4e^{-12}$
	P2	$775.9e^{-12}$	$301.2e^{-12}$
	P3	$769.6e^{-12}$	$310.5e^{-12}$
	P4	$778.1e^{-12}$	490.1^{-12}
	P5	$779.6e^{-12}$	$649.3e^{-12}$
	P6	$887.8e^{-12}$	$302.1e^{-12}$
	P7	$489.3e^{-12}$	$889.7e^{-12}$
Number of Transistors		252	398
Static Power		$130.2e^{-6}$	$22.03e^{-6}$
Dynamic Power		$2.103e^{-4}$	$2.423e^{-5}$
Total Power		0.0003612	0.00007093

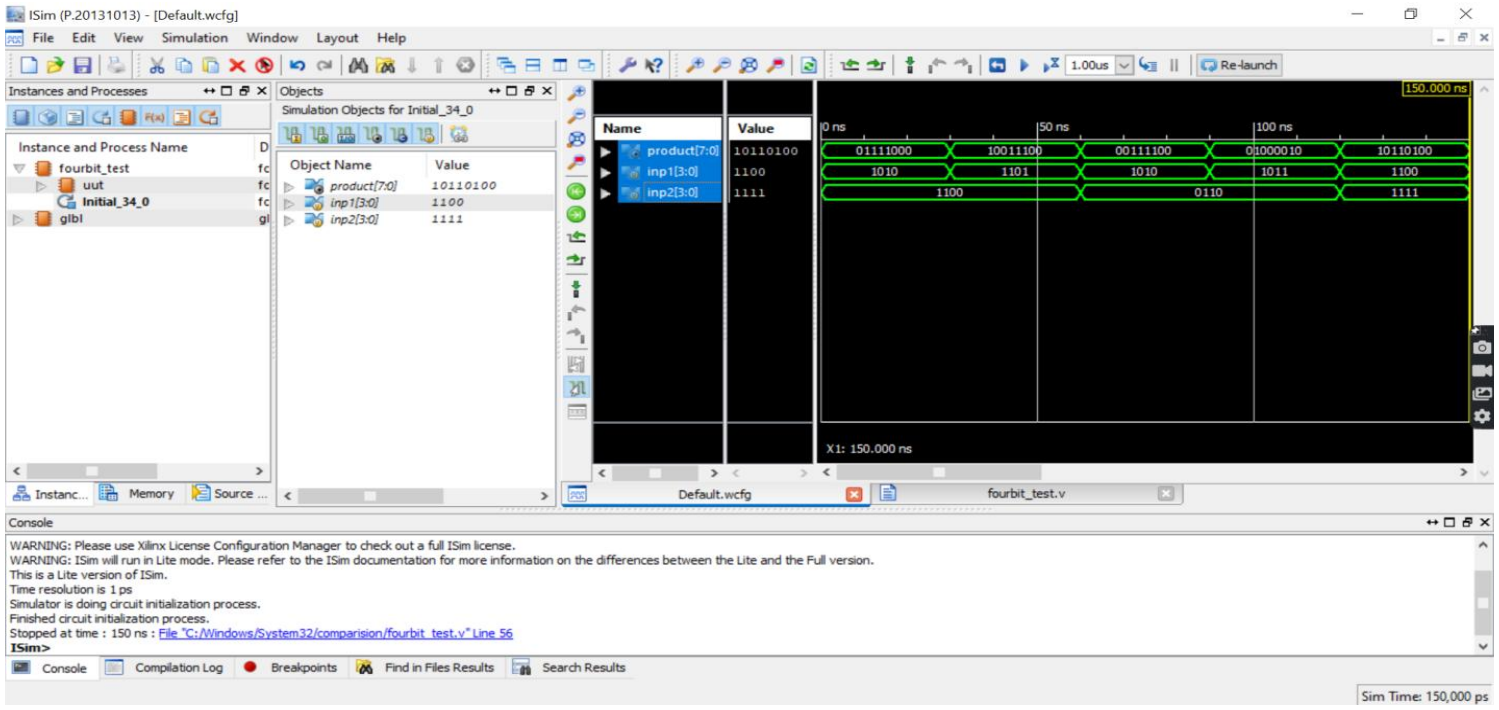
WORK CARRIED OUT (contd..)



Object Name	Value
product[7:0]	01101110
multiplier[3:0]	1010
multiplicand[...	1011

Fig. Output Waveform of Generic Multiplier

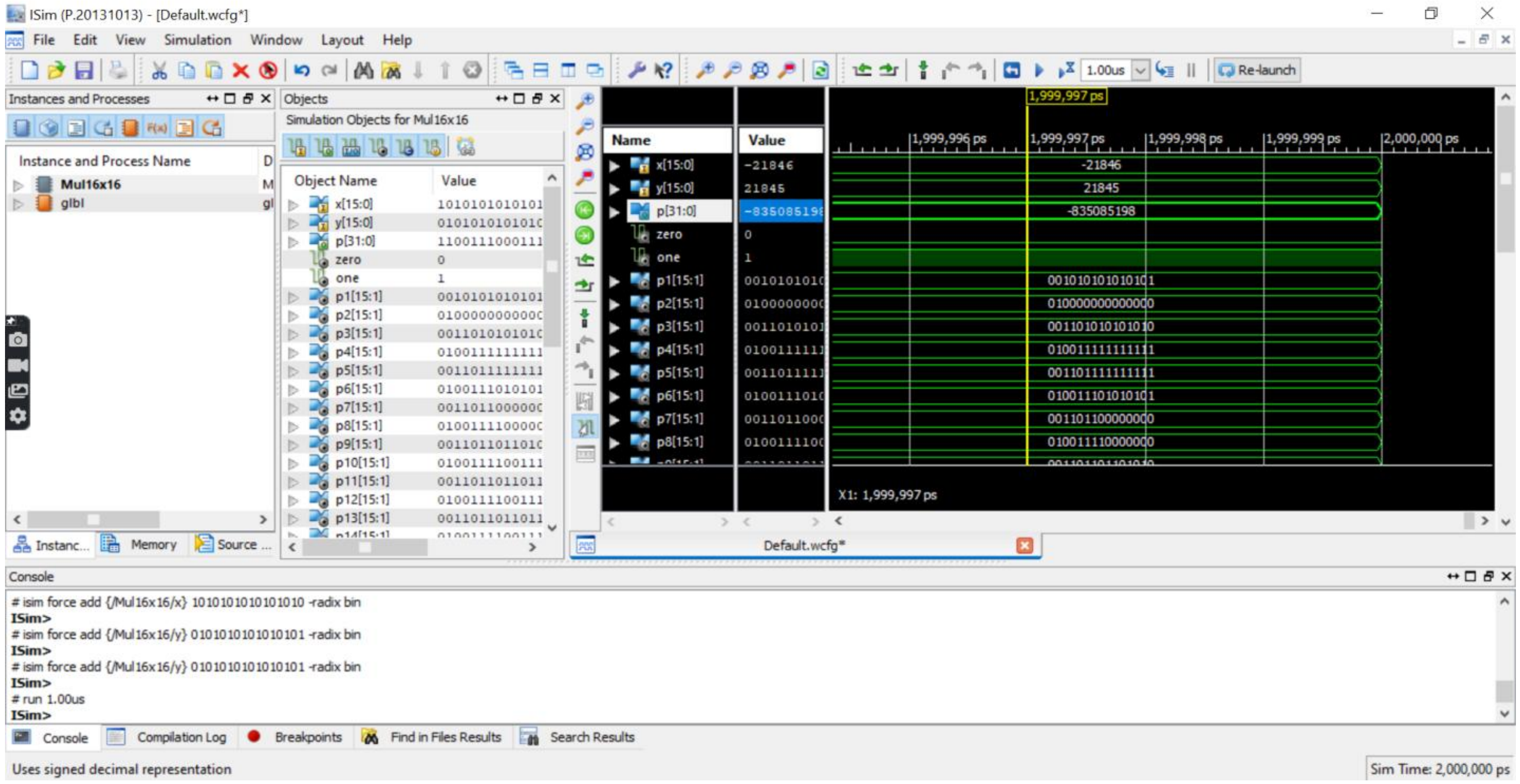
WORK CARRIED OUT (contd..)



```
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
Stopped at time : 150 ns : File "C:/Windows/System32/comparision/fourbit_test.v" Line 56
ISim>
```

Fig. Simulation Result of 4x4 Baugh-Wooley Multiplier using half adder based design

WORK CARRIED OUT (contd..)



```
Object Name   Value
x[15:0]       101010101010101
y[15:0]       010101010101010
p[31:0]       110011100011 1
zero          0
one           1
p1[15:1]      001010101010101
p2[15:1]      010000000000000
p3[15:1]      001101010101010
p4[15:1]      010011111111111
p5[15:1]      001101111111111
p6[15:1]      010011101010101
p7[15:1]      001101100000000
p8[15:1]      010011110000000
p9[15:1]      001101101101010
p10[15:1]     010011100111
p11[15:1]     001101101101 1
p12[15:1]     010011100111
p13[15:1]     001101101101 1
```

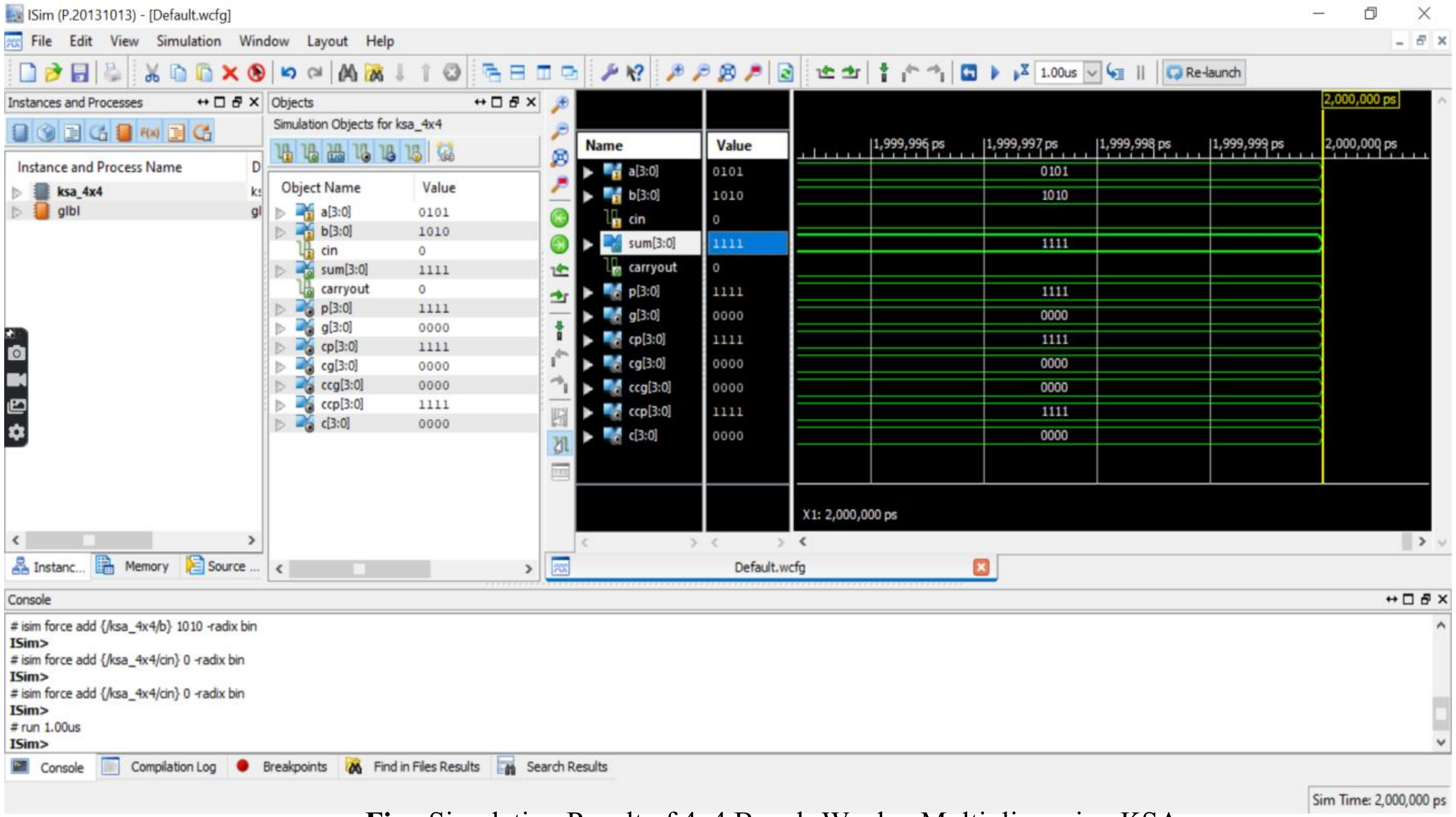
```
Name       Value                            1,999,996 ps   1,999,997 ps   1,999,998 ps   1,999,999 ps   2,000,000 ps
x[15:0]    -21846          -21846
y[15:0]    21845           21845
p[31:0]    -835085198      -835085198
zero       0
one        1
p1[15:1]   0010101010      001010101010101
p2[15:1]   0100000000      010000000000000
p3[15:1]   0011010101      001101010101010
p4[15:1]   0100111111      010011111111111
p5[15:1]   0011011111      001101111111111
p6[15:1]   0100111010      010011101010101
p7[15:1]   0011011000      001101100000000
p8[15:1]   0100111100      010011110000000

X1: 1,999,997 ps
```

```
# isim force add {/Mul16x16/x} 1010101010101010 -radix bin
ISim>
# isim force add {/Mul16x16/y} 0101010101010101 -radix bin
ISim>
# isim force add {/Mul16x16/y} 0101010101010101 -radix bin
ISim>
# run 1.00us
ISim>
```

Uses signed decimal representation   Sim Time: 2,000,000 ps

Fig. Simulation Result of 16x16 Baugh-Wooley Multiplier using conventional adders

WORK CARRIED OUT (contd..)

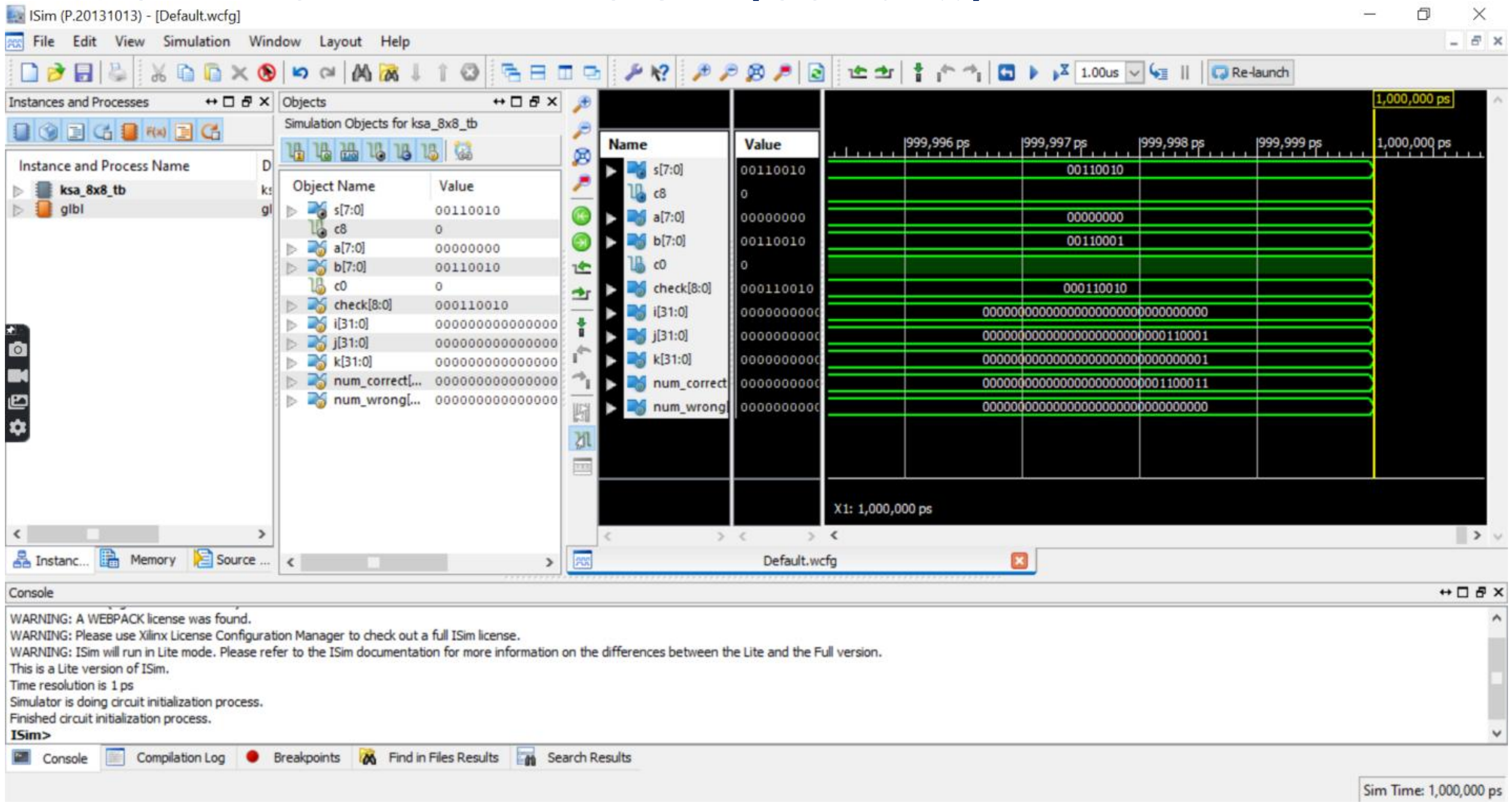


Object Name	Value
a[3:0]	0101
b[3:0]	1010
cin	0
sum[3:0]	1111
carryout	0
p[3:0]	1111
g[3:0]	0000
cp[3:0]	1111
cg[3:0]	0000
ccg[3:0]	0000
ccp[3:0]	1111
c[3:0]	0000

```
# isim force add {/ksa_4x4/b} 1010 -radix bin
ISim>
# isim force add {/ksa_4x4/cin} 0 -radix bin
ISim>
# isim force add {/ksa_4x4/cin} 0 -radix bin
ISim>
# run 1.00us
ISim>
```

Fig. Simulation Result of 4x4 Baugh-Wooley Multiplier using KSA

WORK CARRIED OUT (contd..)



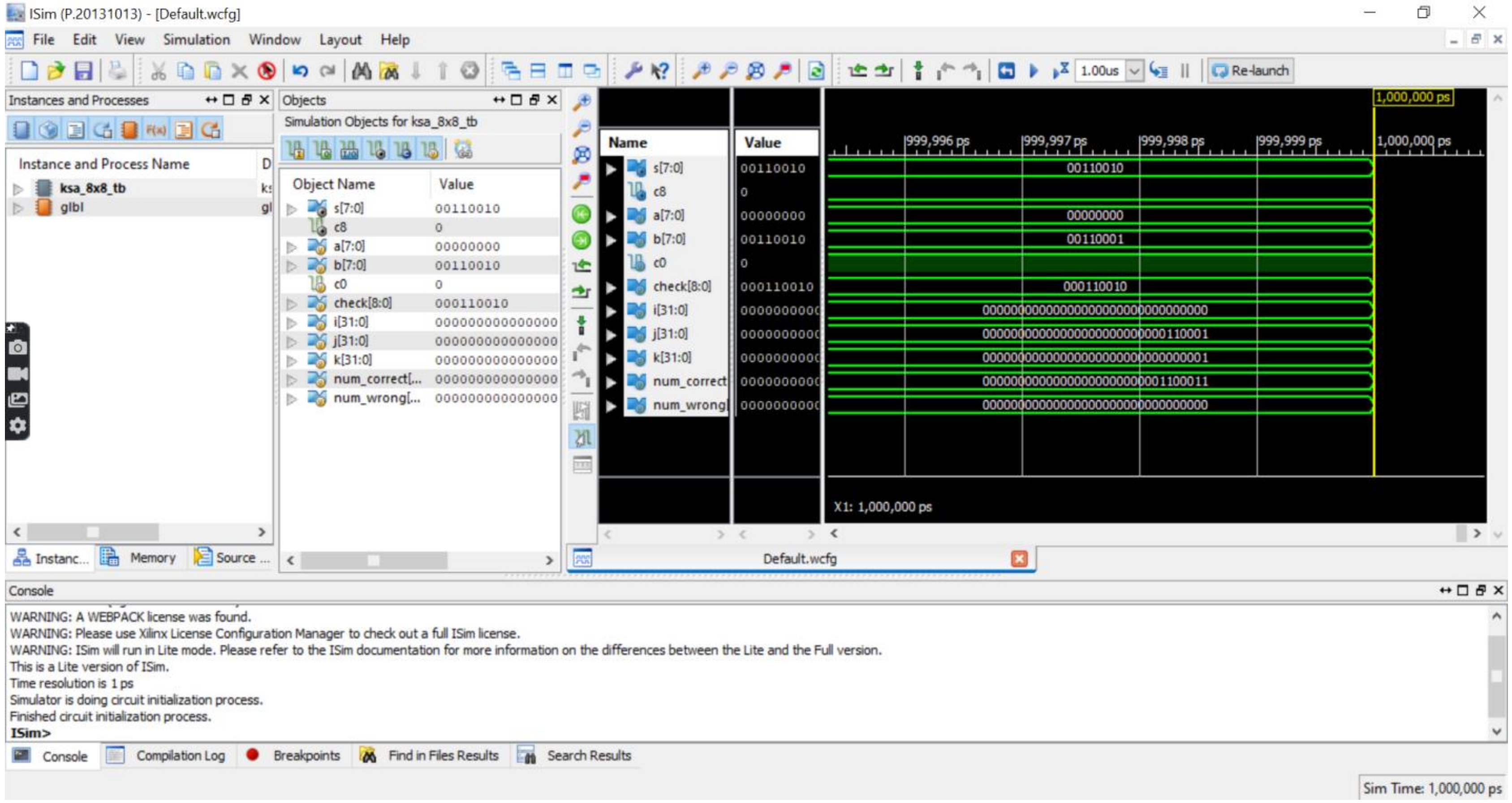
Object Name	Value
s[7:0]	00110010
c8	0
a[7:0]	00000000
b[7:0]	00110010
c0	0
check[8:0]	000110010
i[31:0]	000000000000000
j[31:0]	000000000000000
k[31:0]	000000000000000
num_correct[...	000000000000000
num_wrong[...	000000000000000

Name	Value	
s[7:0]	00110010	00110010
c8	0	
a[7:0]	00000000	00000000
b[7:0]	00110010	00110001
c0	0	
check[8:0]	000110010	000110010
i[31:0]	0000000000	00000000000000000000000000000000
j[31:0]	0000000000	00000000000000000000000000110001
k[31:0]	0000000000	00000000000000000000000000000001
num_correct	0000000000	00000000000000000000000000110011
num_wrong	0000000000	00000000000000000000000000000000

```
WARNING: A WEBPACK license was found.
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

Fig. Simulation Result of 8x8 Baugh-Wooley Multiplier sung BKA

WORK CARRIED OUT (contd..)



Instance and Process Name	D
ksa_8x8_tb	k
glbl	gl

Object Name	Value
s[7:0]	00110010
c8	0
a[7:0]	00000000
b[7:0]	00110010
c0	0
check[8:0]	000110010
i[31:0]	000000000000000
j[31:0]	000000000000000
k[31:0]	000000000000000
num_correct[...	000000000000000
num_wrong[...	000000000000000

Name	Value	
s[7:0]	00110010	00110010
c8	0	
a[7:0]	00000000	00000000
b[7:0]	00110010	00110001
c0	0	
check[8:0]	000110010	000110010
i[31:0]	0000000000	00000000000000000000000000000000
j[31:0]	0000000000	00000000000000000000000000110001
k[31:0]	0000000000	00000000000000000000000000000001
num_correct	0000000000	00000000000000000000000001100011
num_wrong	0000000000	00000000000000000000000000000000

```
WARNING: A WEBPACK license was found.
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

Fig. Simulation Result of 8x8 Baugh-Wooley Multiplier sung BKA

WORK CARRIED OUT (contd..)

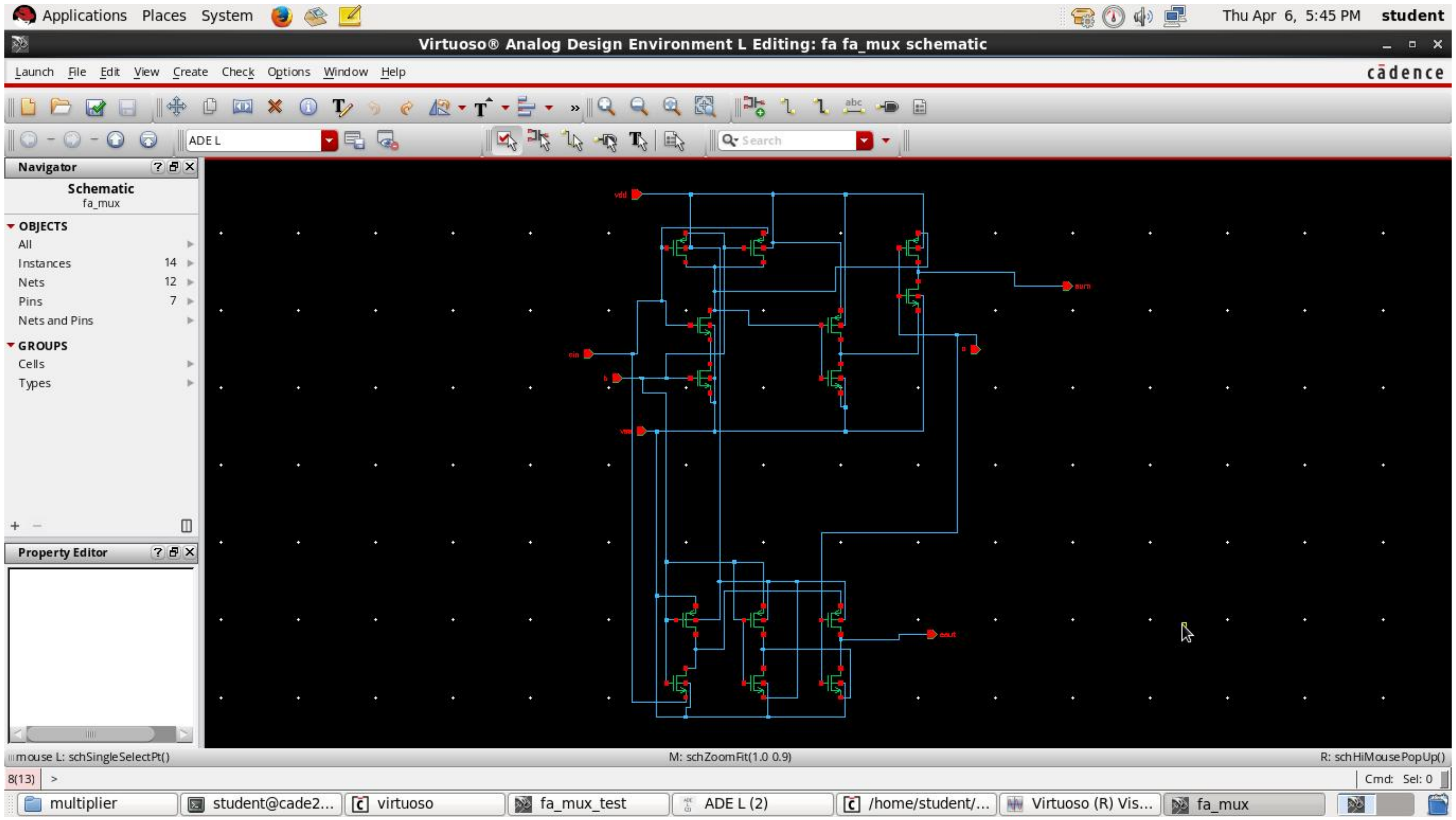


Fig. MUX Based Full Adder Design

WORK CARRIED OUT (contd..)

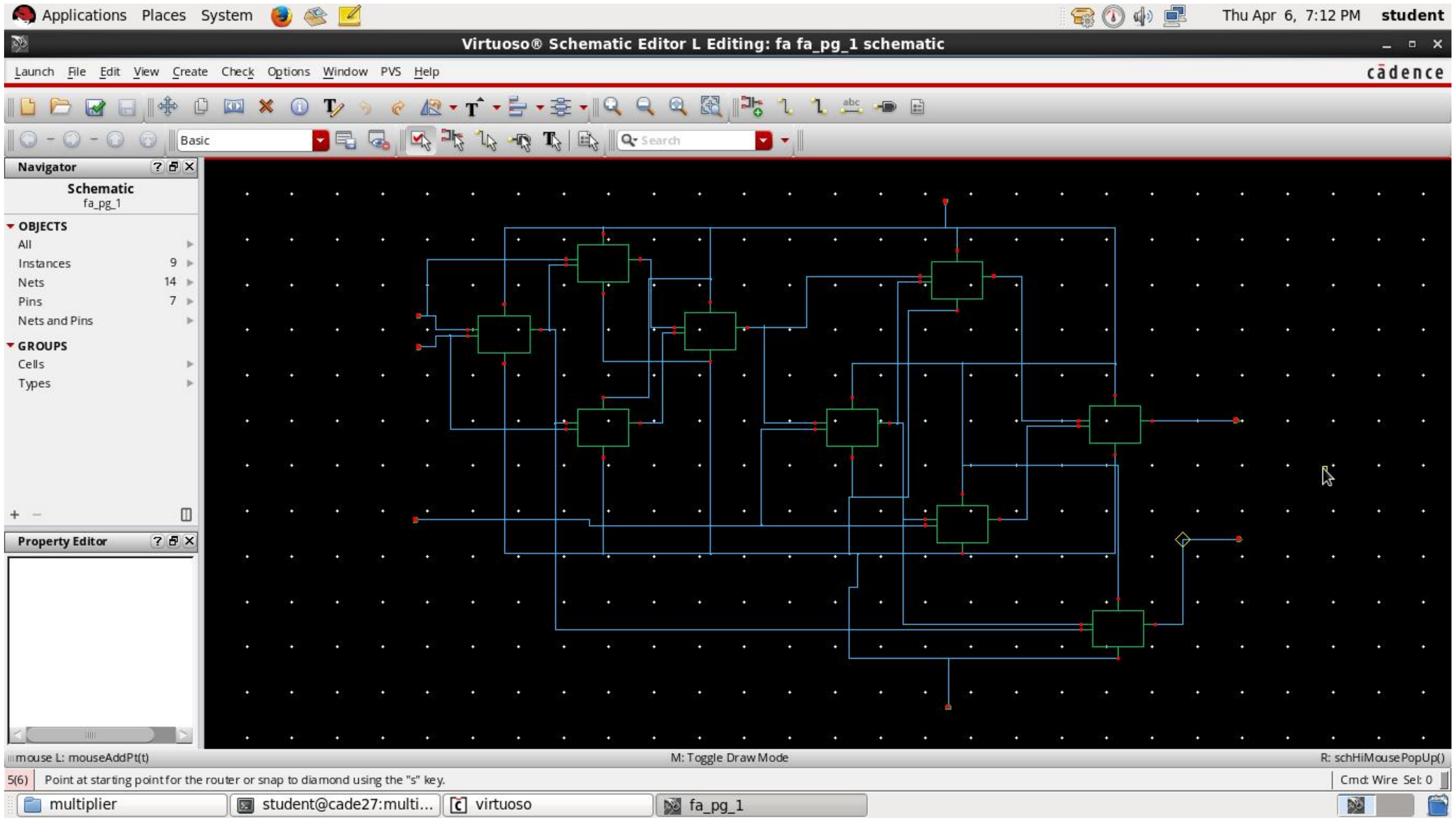


Fig. PG Based Full Adder Design

WORK CARRIED OUT (contd..)

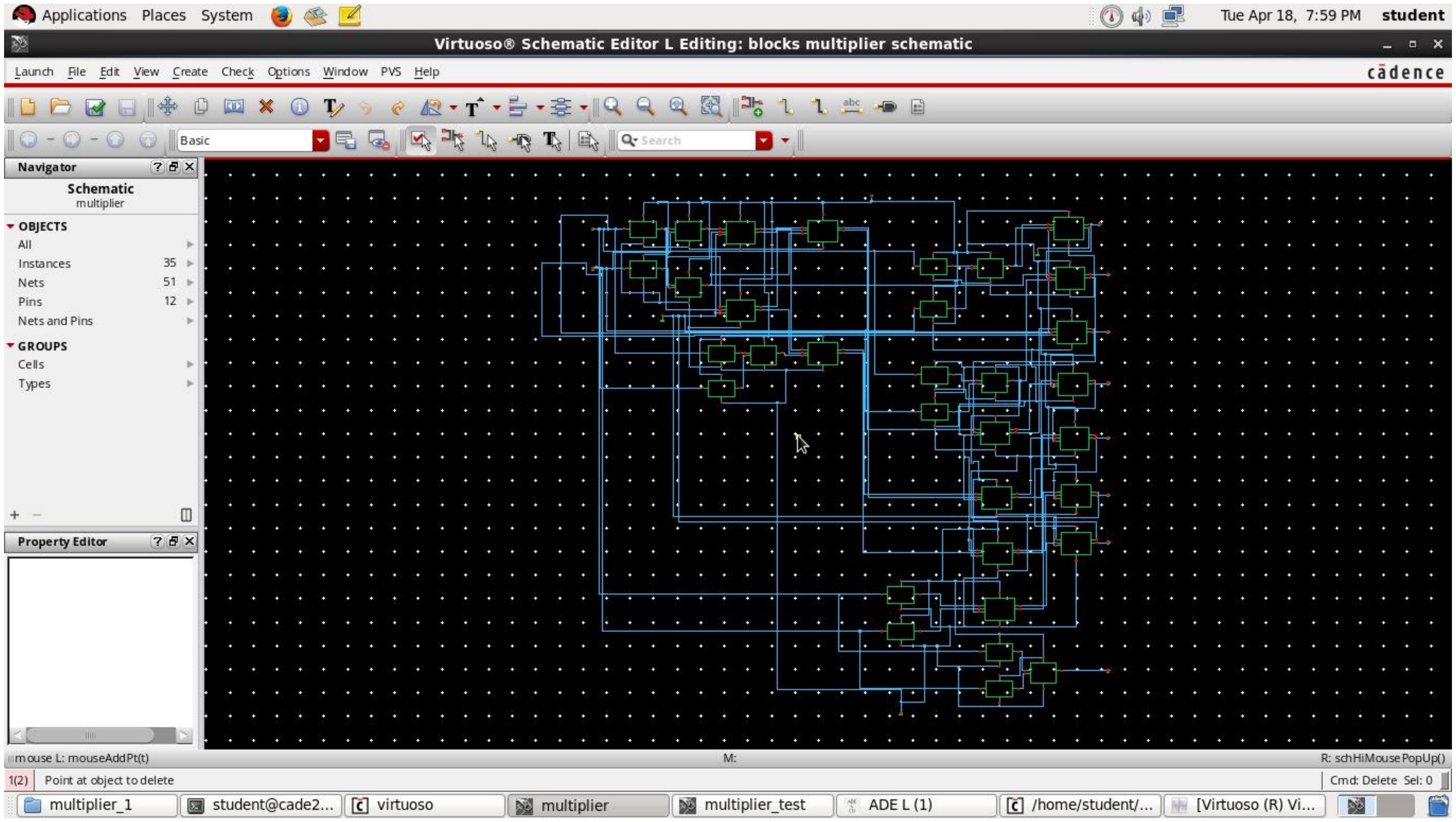
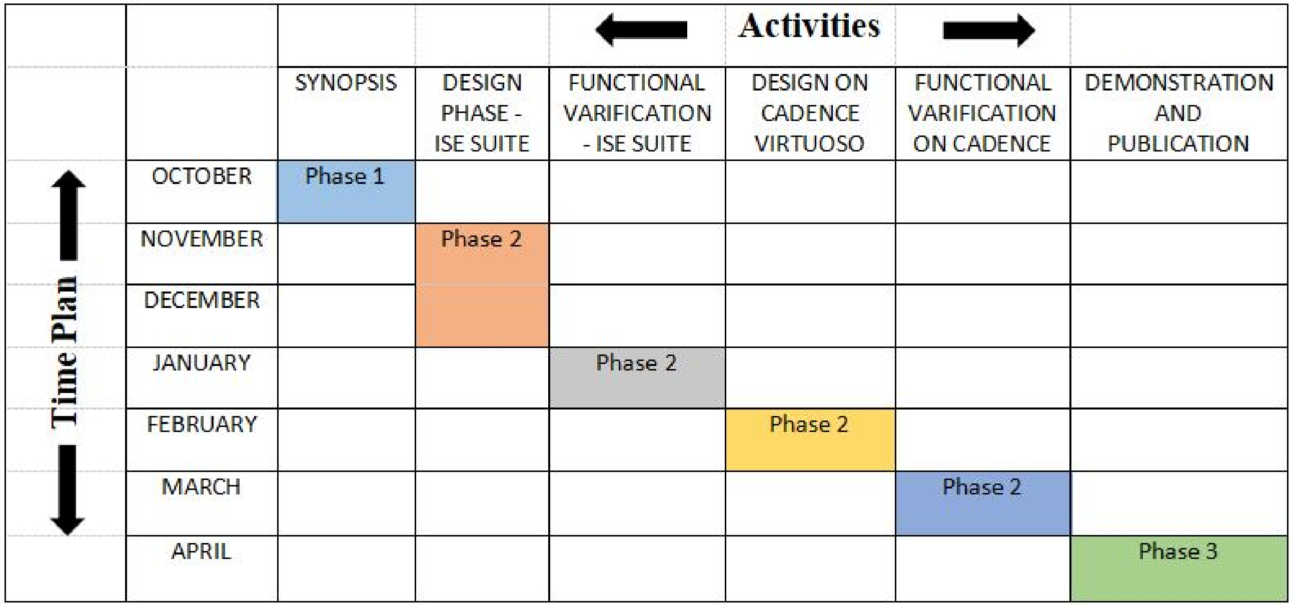


Fig. PG Based Full Adder Design

EXPECTED OUTCOMES

- ✓ Design and development of a high speed multiplier block for digital signal processing application
- ✓ Analysis its power and delay characteristics on FPGA level using Xilinx ISE/Vivado Design Suits.
- ✓ Analysis its power and delay characteristics on FPGA level using Xilinx ISE/Vivado Design Suits.

PERT CHART



					Activities		
		SYNOPSIS	DESIGN PHASE - ISE SUITE	FUNCTIONAL VARIFICATION - ISE SUITE	DESIGN ON CADENCE VIRTUOSO	FUNCTIONAL VARIFICATION ON CADENCE	DEMONSTRATION AND PUBLICATION
Time Plan	OCTOBER	Phase 1					
	NOVEMBER		Phase 2				
	DECEMBER						
	JANUARY			Phase 2			
	FEBRUARY				Phase 2		
	MARCH					Phase 2	
	APRIL						Phase 3

CONCLUSIONS & FUTURE SCOPE

- In conclusion, our study has provided a comprehensive analysis of the design and development of a high-speed multiplier for digital signal processing applications. We have explored various aspects of multiplier architecture and identified the Bough-Wooley multiplier as the most suitable for our targeted application.
- Our analysis of the architecture on FPGA level and implementation on Cadence Virtuoso has provided valuable insights into the computational results and resource usage. Based on our findings, we have determined that the propagate and generate based addition is the optimal approach for achieving high speed and optimized resource usage.
- Overall, this study will serve as a valuable resource for researchers and engineers working on digital signal processing applications that require high-speed multiplication.

FUTURE SCOPE

- The project can be extended by exploring the application of the Bough-Wooley multiplier architecture to other digital signal processing systems and analyzing its performance in comparison to other architectures.

- Additionally, further research can be conducted to optimize the design of the multiplier for specific applications, such as image or speech processing. The implementation of the architecture on different platforms, such as ASIC or SoC, can also be explored to improve its performance and efficiency.

- Furthermore, the study of addition techniques on the Bough-Wooley multiplier architecture can be extended by exploring the application of other techniques to further optimize resource usage and speed.

REFERENCES

[1] B. Shivaprasad, K. Shinde, V. Muddi, Design and implementation of parallel floating point matrix multiplier for quaternion computation, Control, Instrumentation, Communication and Computational Technologies (ICCICCT),2015, pp. 289–293.

[2] R Zimmermann and W Fichtner, Fellow, IEEELow-Power" Logic Styles: CMOS Versus Pass-Transistor Logic," IEEE Journal Of Solid-State Circuits, vol. 32, no. 7,pp.1079- 90,1997.

[3] D Markovic, B Nikolic, and V G Oklobdzija, "A general method in synthesis of pass-transistor circuits," Microelectr. J, vol. 31,pp. 991-8,2000.

[4] C H Chang, J Gu, and M Zhang, "A review of O.J8-mm full adder performances for tree structured arithmetic circuits," IEEE Trans. Very Large Scale Integr. (VLSI) Syst. Vol. 13 pp. 686-95, 2005.

THANK YOU