DESIGN OF ARITHMATICAL LOGICAL UNIT

COEN 6501

Project report

Submitted BY

Zahidul Amin 40031489

Rokibul Hasan Bhuiyan 40091513

Submitted To

Dr. Asim Al-Khalili

FALL 2018 |CONCORDIA UNIVERSITY

Contents

Contribution

Project was a team effort, I Zahidul Amin worked on Designing, Coding and Testing and Synthesis of the circuit and report writing. I Rokibul Hasan Bhuiyan was responsible for Designing, Coding and Report writing.

Acknowledgement

We would like to thank Dr Asim Al-Khalili for his wonderful support and encouragement throughout the course. His advice gave us the confidence to learn the subject matter deeply, made us more confident and gave us guidelines to prepare ourselves in our carrier. We gained a beautiful insight on the field of Digital Design and Synthesis, through his wisdom. We learned how to think and design a problem and make it possible. It was both an honour and a privilege to attend lecture under him.

Abstract

In this project we apply different design methodology to implement arithmetical logical unit, which is one of the most used device in digital signal processing. This specific ALU can perform operation A^2-1 and more. We implement our design using the knowledge of this course Digital Design Synthesis by using VHDL (very high speed integrated circuit hardware description language) via IEEE approved packages in MODELSIM. In this design we use, different form of adders and multiplier, applying both structural and behavioral coding. We also apply designing technique by evaluating trade-off between area and speed. Our design is implemented using Hierarchical modeling technique, where we build structurally build every small component and merge them together. Then we develop test bench and perform synthesis on Xilinx.

Introduction

Digital design and synthesis is the idea which provides an approach to implement a complex system using digital logic into the simplest form. Arithmetical logical unit is a digital logic circuit mostly used in processors that performs mathematical operations as in integrated part of the processor. This component is an individual block, which may have multiple functions integrated into it. The basic input structure consists of instructions for which function to select, input for operands and input for execution of commands, the output structure consists of results of specific selected function and completion of the task. The methodology used falls within the realm of Digital design and Synthesis.

Design specification

A design engineers task at first is to sit with clients and visualize the idea which the clients wants, we must keep in mind that all that can be designed cannot be synthesized. There is limitation what is possible within the realm of digital logic with current available technology. In this project the design specification stated that arithmetical logical unit must be able to perform the mathematical operation of A^2-1, where it takes 8-bit **signed** binary input, and has an output of 16 bits. Other inputs associated with it are **reset**, **clock** and **load** and output also contain a **flag signal.** Upon transition of load from 1 to zero, the input signed bits will latched on to the input registers and then mathematical operations are performed, after its completion the results are loaded into registers and then flag value transitions to 1 showing the completion of the operation. The reset transition, clears everything to 0 and clock input corresponds to clock. Schematic diagram of the block is given below.

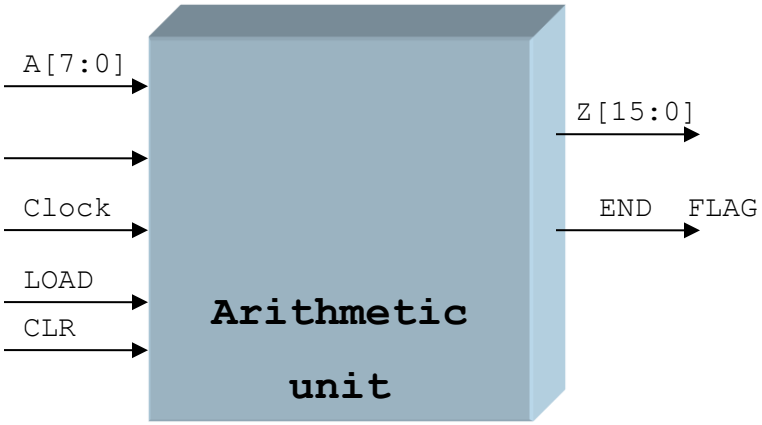

Figure 1 Design Specification

Design methodology

In this project we used combination of both structural and behavioral design. We started with smallest elements using simplest and shortest logical circuits, and then build individual modules of the total function and then merge them together also known as hierarchical modelling. Schematic description of typical hierarchical modelling is given below.

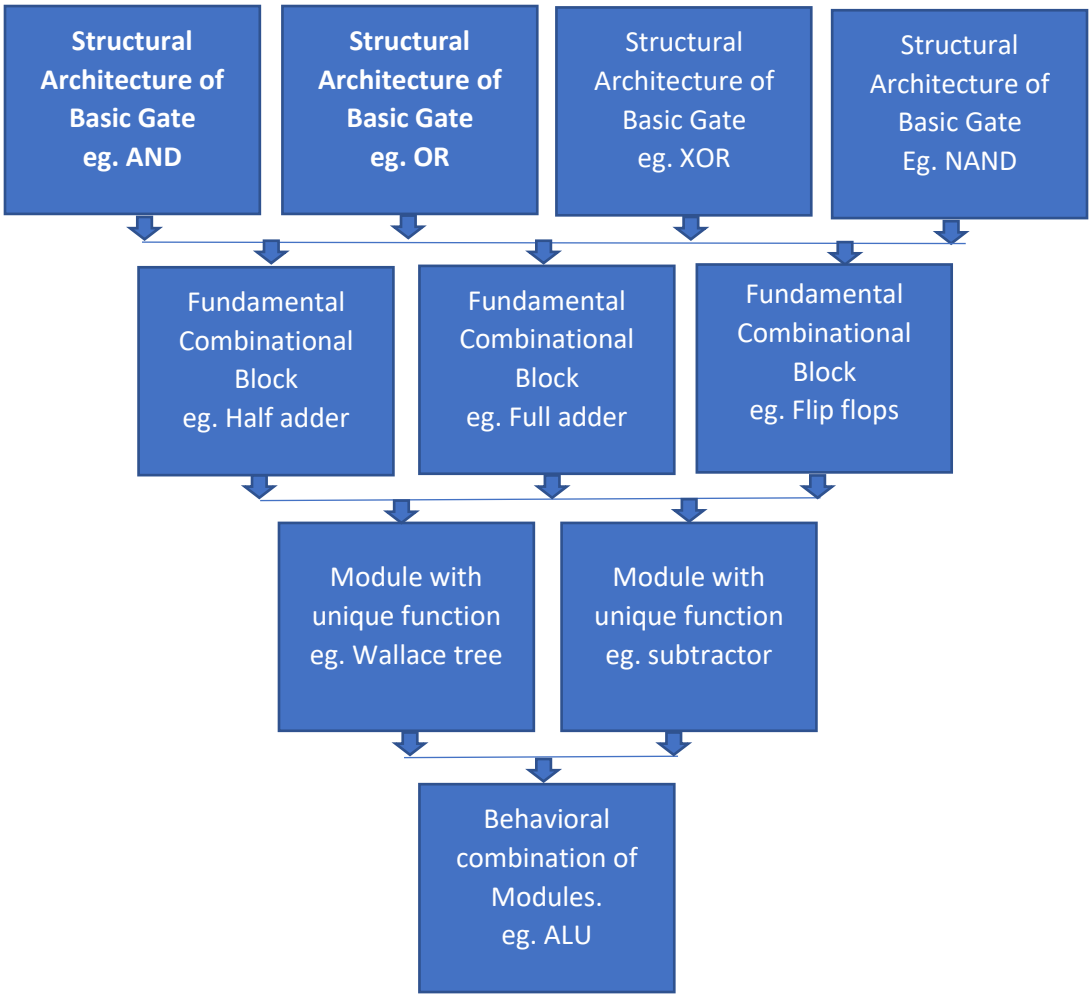

Figure 2 Hierarchical Design Approach

Specification flow chart

Based on the specification it is always wise to have a system flow chart and then proceed with hand drawing of the circuit.

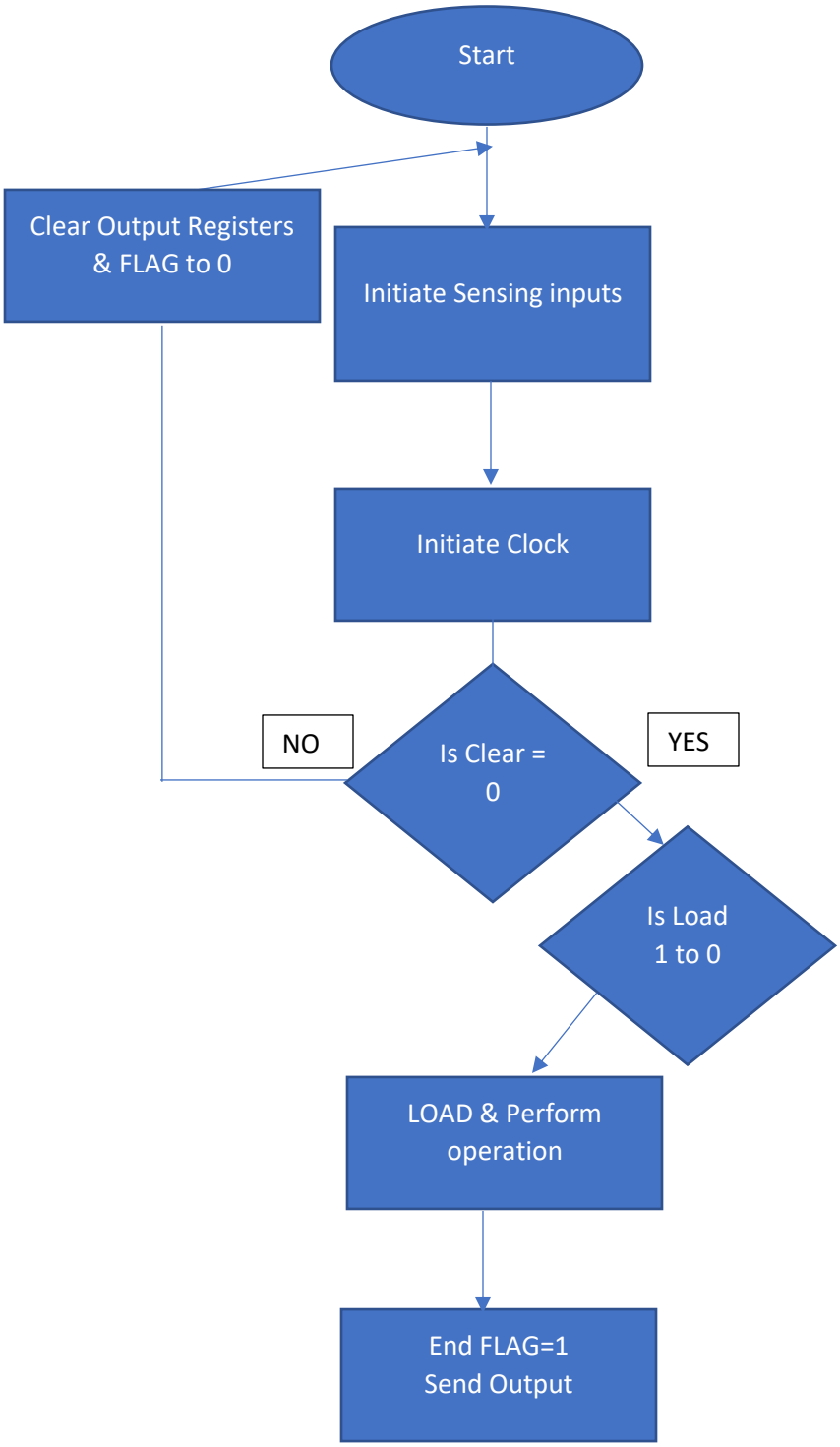

Figure 3 System Flow Chart

Design Flow

First the whole concept of ALU was hand drawn on a piece of paper just to get a hard copy of the visualization we had. Then we converted the concept into MODELSIM. After checking the design with the test bench, it is found that the ALU is working correctly. After that area and timing of the design have been calculated using Xilinx Design Manager.

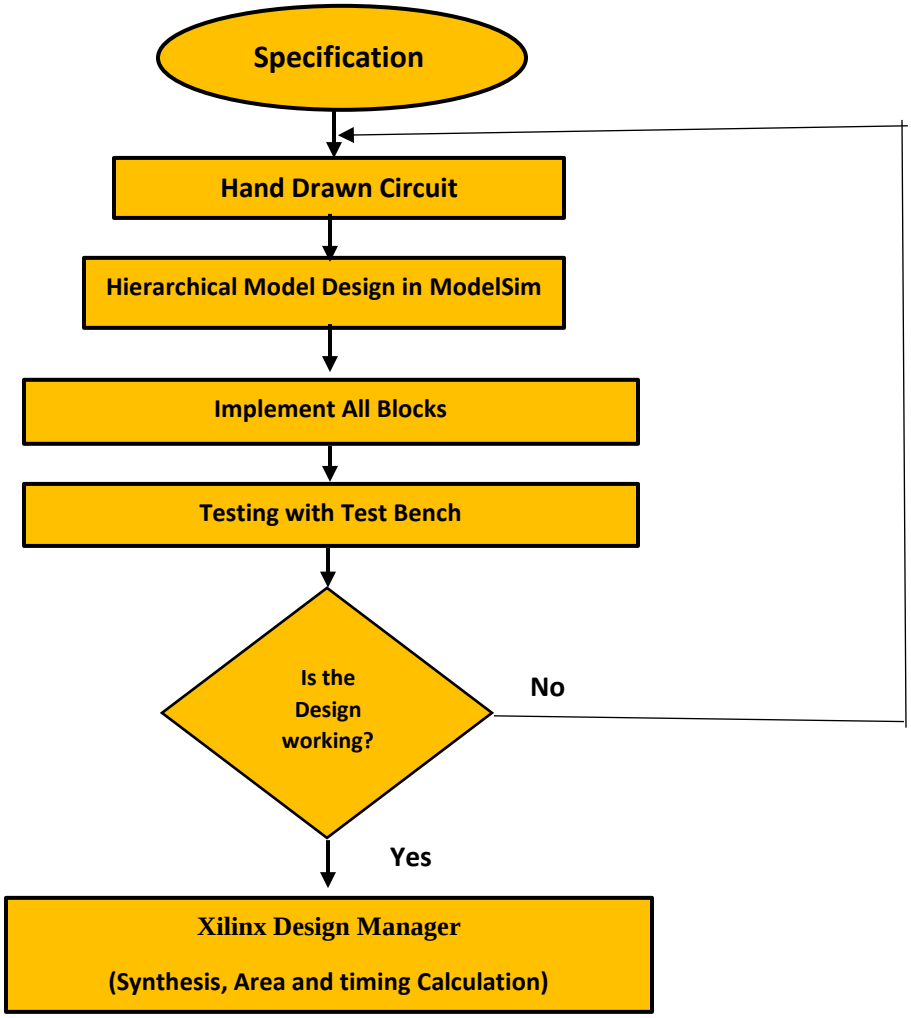

Figure 4 Design Flow Structure

TWO INPUT AND GATE BLOCK:

At first, we did structural coding of AND gate, the function of AND Gate is when both inputs are true it sends output as true.

INPUTS		OUTPUTS
0	0	0
0	1	0
1	0	0
1	1	1

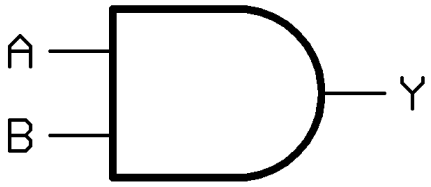

Figure 5 Hierarchical AND BLCOK

TWO INPUT OR GATE BLOCK

We did structural coding of OR GATE, the function of OR gate is that when one of the input is true the output is true.

INPUTS		OUTPUTS
0	0	0
0	1	1
1	0	1
1	1	1

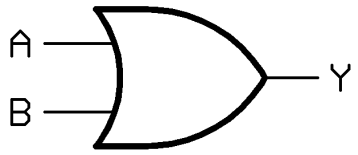

Figure 6 OR GATE

TWO INPUT XOR GATE BLOCK

We did structural coding of XOR GATE, the function of XOR gate is that when both inputs are compliment of each other the output becomes TRUE.

INPUTS		OUTPUTS
0	0	0
0	1	1
1	0	1
1	1	1

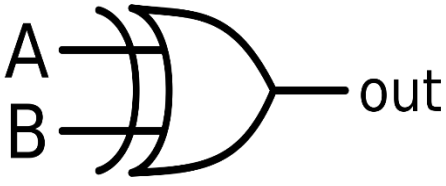

Figure 7 Two Input XOR GATE

TWO AND THREE INPUT NAND GATE BLOCK

We did structural implementation of two separate NAND gates one with two input another with three inputs, NAND gate perform opposite of AND GATE.

INPUTS		OUTPUTS
0	0	1
0	1	1
1	0	1
1	1	0

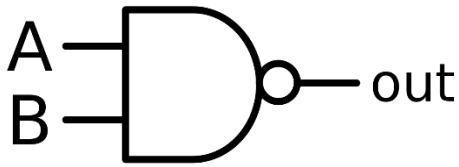

Figure 8 TWO INPUT NAND GATE

INPUTS			OUTPUTS
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

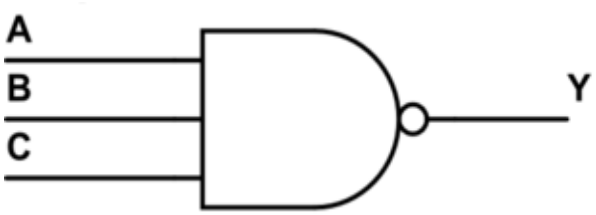

Figure 9 THREE INPUT NAND GATE

ONE BIT HALF ADDER BLOCK

Using previous block, we created one-bit half adder block, this block has two inputs which performs binary addiction and outputs are carry and sum.

A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

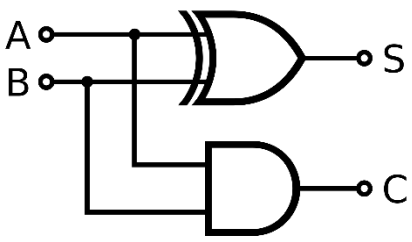

Figure 10 HALF ADDER CIRCUIT

ONE BIT FULL ADDER BLOCK

Similarly, as with half adder we have FULL ADDER with an extra input a carry bit from previous sum, it performs simple addition as half adder, but it takes in to account incoming carry bit.

A	B	CARRY	C-out	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

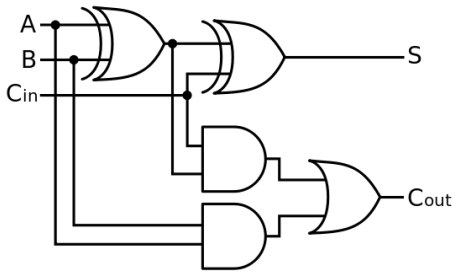

Figure 11 FULL ADDER CIRCUIT

D-FLIP FLOP BLOCK

Our design is based on Wallace tree architecture to perform multiplication, so there are not enough decencies of registers, in terms of speed and timing, so we choose D-Flip Flop, it can be imagined as a memory cell upon transition of clock it matches itself to the current input bit.

CLOCK	D	Q	QBAR
0	0	No CHANGE	No CHANGE
0	1	No CHANGE	No CHANGE
1	0	D	D-BAR
1	1	D	D-BAR

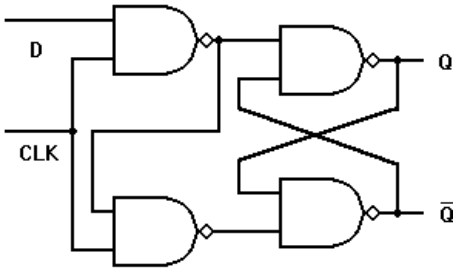

Figure 12 D FLIP FLOP CIRCUIT

2'S COMPLIMENT BLOCK

This block is implemented from the output of D-FLIP FLOP, All the inputs are twos complimented in this model as we are considering signed bit. At first first bit is Xored, which performs one's compliment, then we have a ripple carry adder of the output, in general we used half adders one after another, neglecting 0 input to save area.

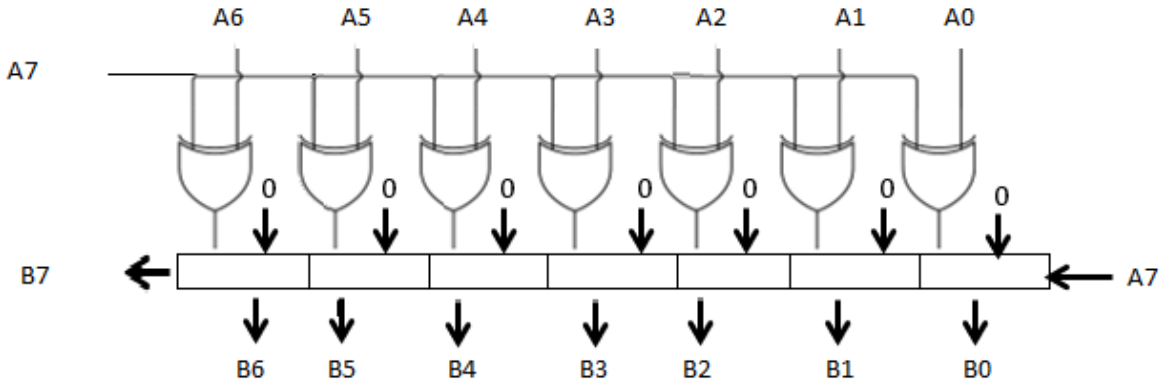

Figure 13 TWOS COMPLIMENT CIRCUIT

PARTIAL AND OPERATION BLOCK

In this hierarchical block we are taking eight bits and doing AND operation for Partial AND Product which is well illustrated in the figures below. It performs 36 unique and operations basically all the add operation that is involved for addition, to save space we did optimization, we performed AND only once of each combination.

									n7	n6	n5	n4	n3	n2	n1	n0
									n7	n6	n5	n4	n3	n2	n1	n0
								n7n0	n6n0	n5n0	n4n0	n3n0	n2n0	n1n0	n0n0	
							n7n1	n6n1	n5n1	n4n1	n3n1	n2n1	n1n1	n0n1		
						n7n2	n6n2	n5n2	n4n2	n3n2	n2n2	n1n2	n0n2			
					n7n3	n6n3	n5n3	n4n3	n3n3	n2n3	n1n3	n0n3				
				n7n4	n6n4	n5n4	n4n4	n3n4	n2n4	n1n	n0n4					
			n7n5	n6n5	n5n5	n4n5	n3n5	n2n5	n1n5	n0n5						
		n7n6	n6n6	n5n6	n4n6	n3n6	n2n6	n1n6	n0n6							
	n7n7	n6n7	n5n7	n4n7	n3n7	n2n7	n1n7	n0n7								
C_{out}	n7n6	n7n5	n7n4	n7n3	n7n2	n7n1	n7n0	n6n0	n5n0	n4n0	n3n0	n2n0	n1n0	0	n0	
	n7		n6n5	n6n4	n6n3	n6n2	n6n1	n5n1	n4n1	n3n1	n2n1		n1			
			n6		n5n4	n5n3	n5n2	n4n2	n3n2		n2					
					n5		n4n3		n3							
							n4									

Figure 14 8 BIT BINARY MULTIPLICATION

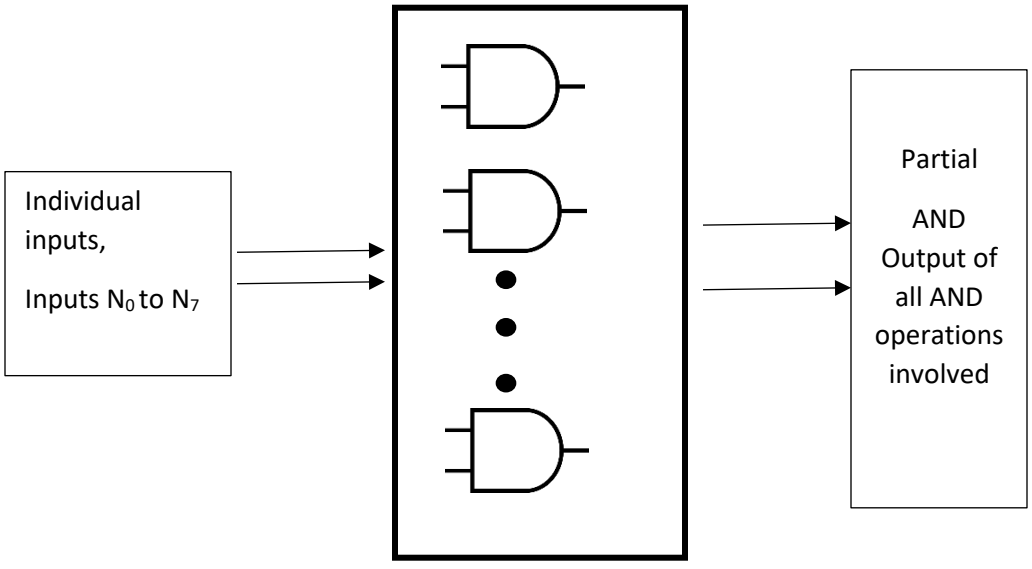

Figure 15 PARTIAL AND OPERATION CIRCUIT

A^2 BLOCK

In this block we have used Wallace tree for the A^2 Operation, it is very useful in implementing where we have large number of bits multiplication, it also very fast but consumes area more. In Wallace tree all the bits of the partial products which we have gained from AND Block would be sent for addition. We have used half adder where full adder is not needed to save area.

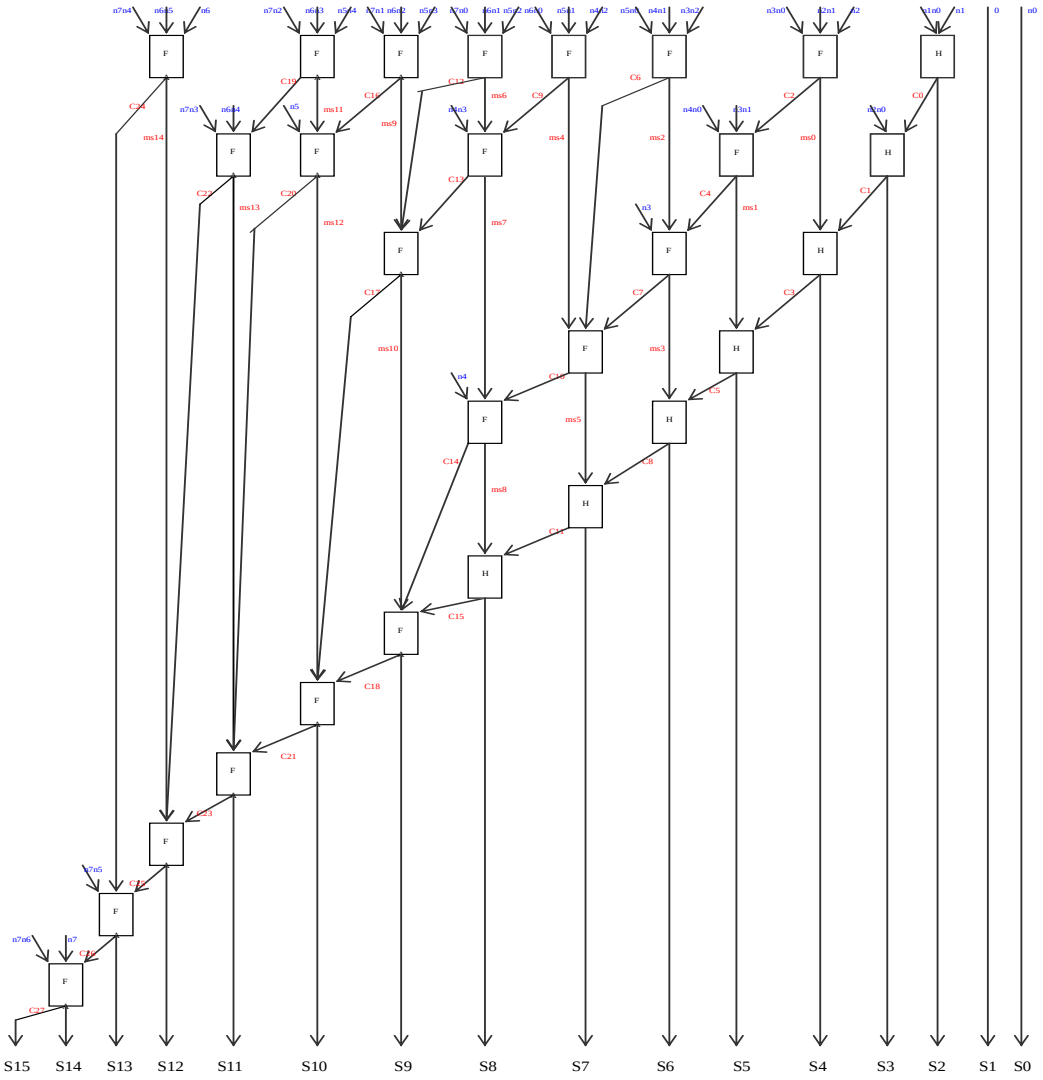

Figure 16 WALLACE TREE MULTIPLIER CIRCUIT

MINUS ONE BLOCK

This Hierarchical Block generates (-1). The output of A^2 Block is passed through this block where subtraction is done. The blocks below are ripple carry adder created by full adders.

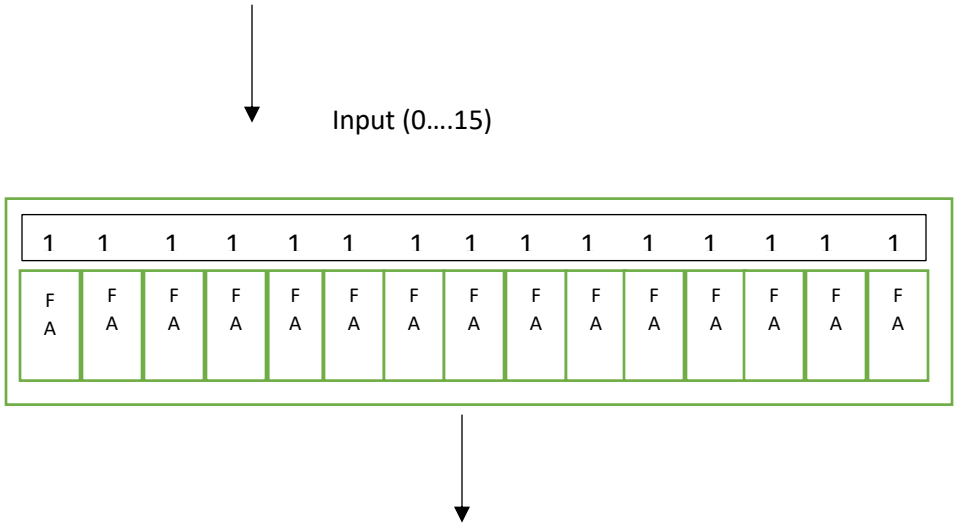

Figure 17 MINUS ONE CIRCUIT

A SQUARE MINUS ONE BLOCK

In this block we connect all the block together that performs the mathematical operation of A^2-1.

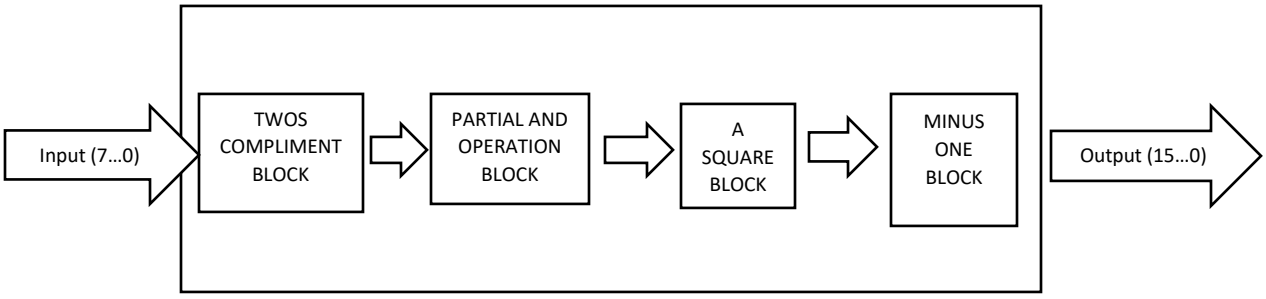

Figure 18 A SQUARE MINUS ONE CIRCUIT

ALU BLOCK

In block we implement the A square minus one block with the flow chart we defined earlier after that we, applied behavioral logic, to meet the specified design criteria, benefits of doing this allows us to debug easily, other modules are not affected by changes in another module. This block is implemented based on how the ALU should behave i of input, load, reset, output and flag.

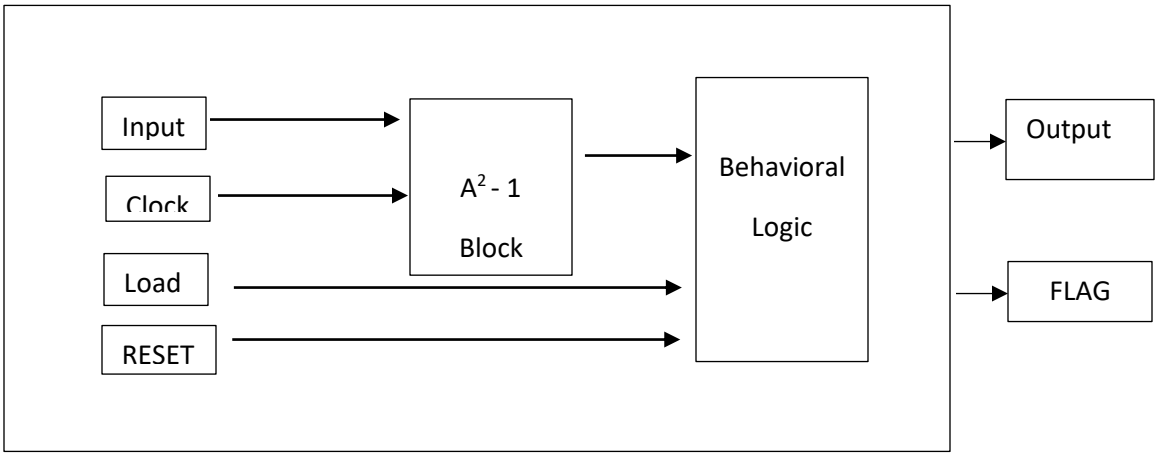

Figure 19 ALU CIRCUIT

Simulation and Results

TWO INPUT AND GATE:

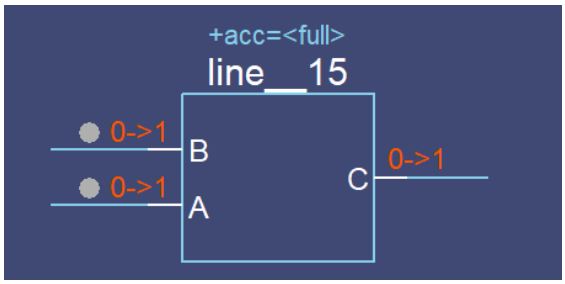

Figure 20 DATA FLOW CIRCUIT OF AND GATE

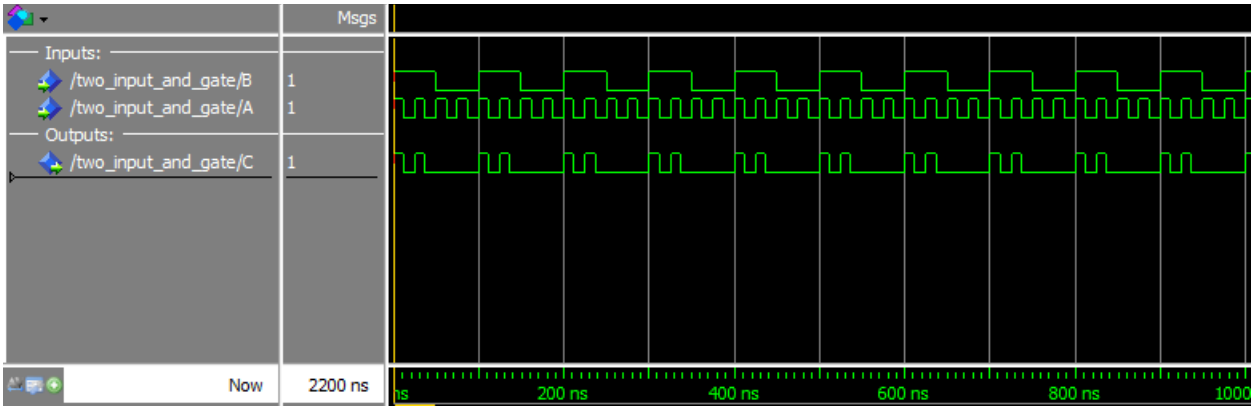

Figure 21 AND GATE WAVE FORM

Here we observe that the implementation of the AND gate is successful we get the result we expect.

TWO INPUT OR GATE

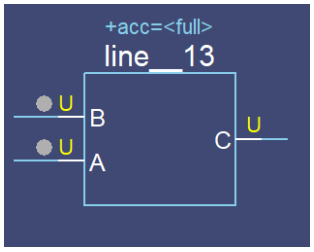

Figure 22 DATA FLOW CIRCUIT OF OR GATE

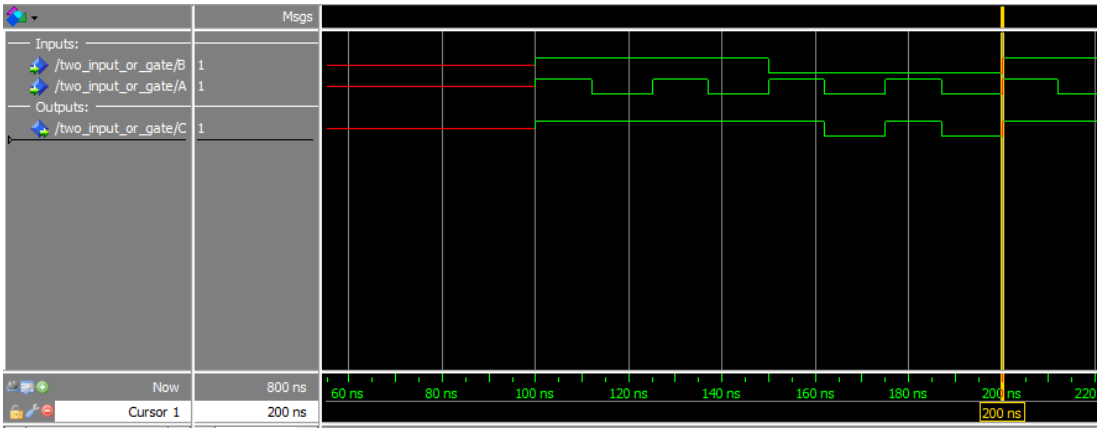

Figure 23 OR GATE WAVE FORM

As we observe from the result that, implementation of the OR gate is successful we get the result we expected from truth table.

TWO INPUT XOR GATE

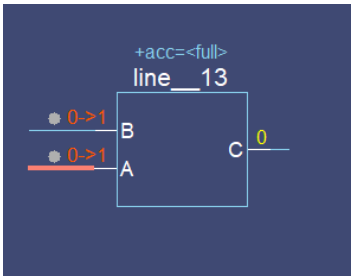

Figure 24 DATA FLOW CIRCUIT OF XOR GATE

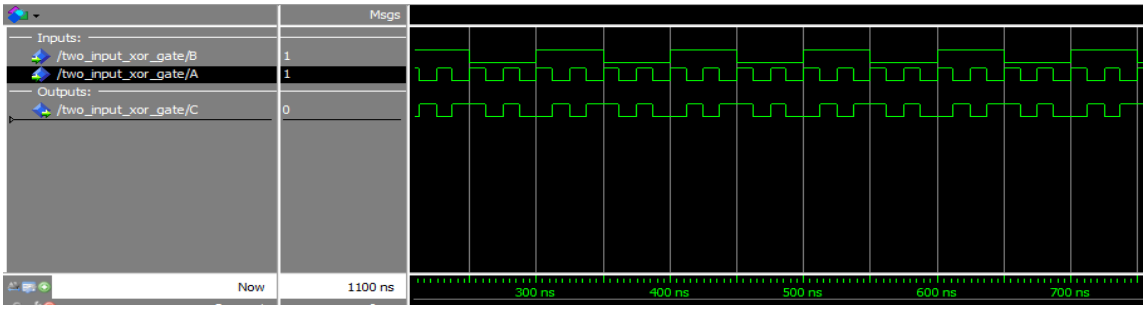

Figure 25 XOR GATE WAVE FORM

From the above wave form it is easily visible that the XOR implementation was a success.

TWO INPUT NAND GATE

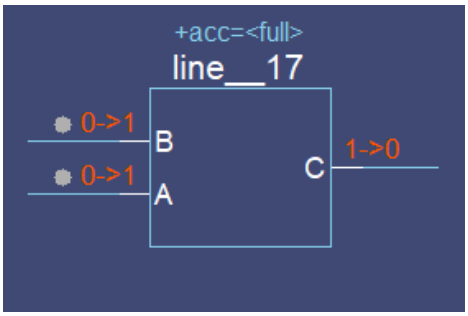

Figure 26 TWO INPUT NAND GATE

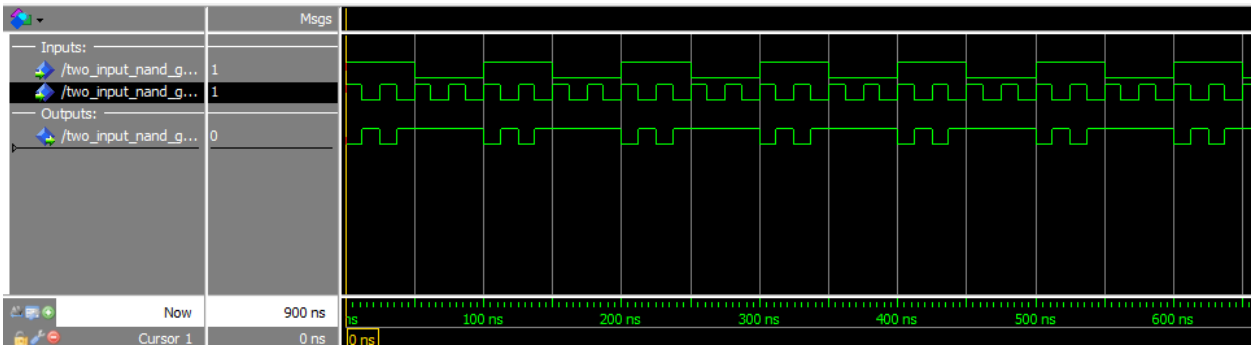

Figure 27 WAVE FORM OF TWO INPUT NAND GATE

From the wave form we confirm that the NAND gate is working according to the truth table.

THREE INPUT NAND GATE

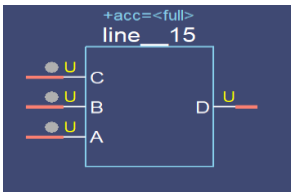

Figure 28 DATA FLOW CIRCUIT OF THREE INPUT NAND GATE

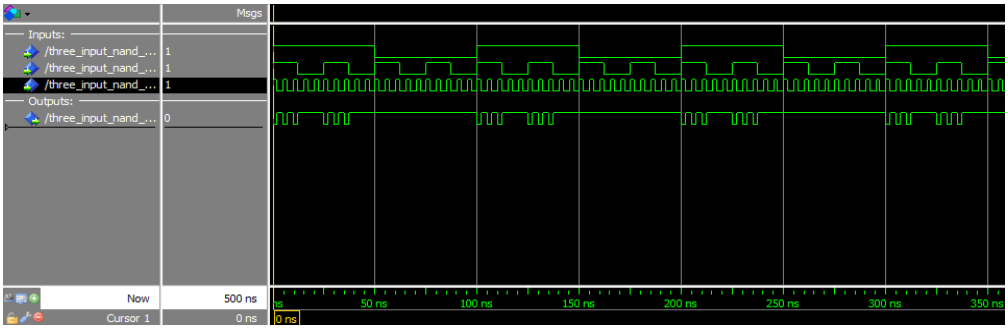

Figure 29 THREE INPUT NAND GATE WAVE FORM

The wave form shows that the designed NAND gate is working as intended.

D FLIP FLOP

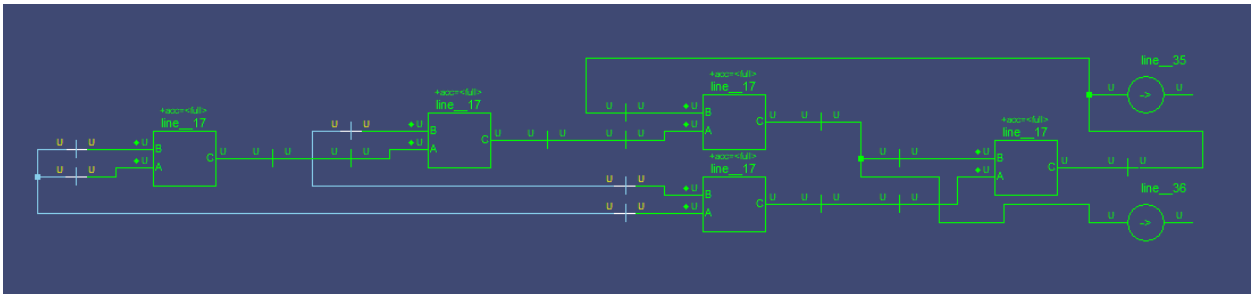

Figure 30 DATA FLOWCIRCUIT OF D FLIP FLOP

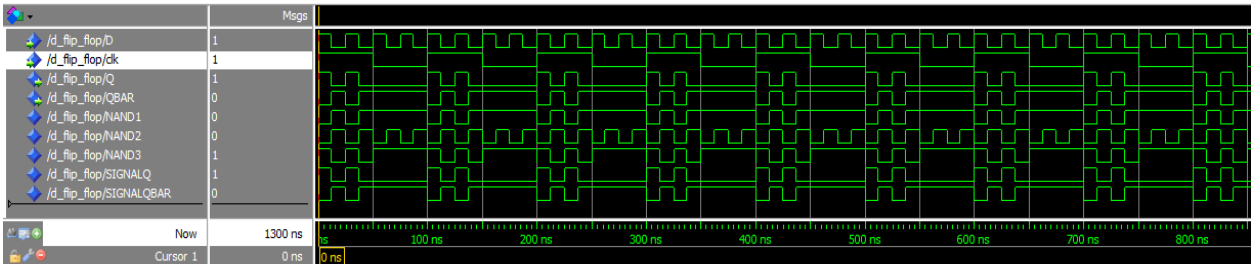

Figure 31 D FLIP FLOP WAVE FORM

The wave form shows that the implementation of the D FLIP FLOP was successful.

ONE BIT HALF ADDER

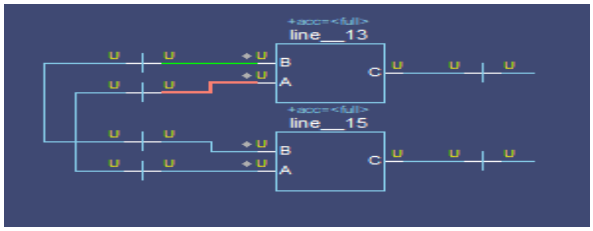

Figure 32 DATA FLOW CIRCUIT OF D FLIP FLOP

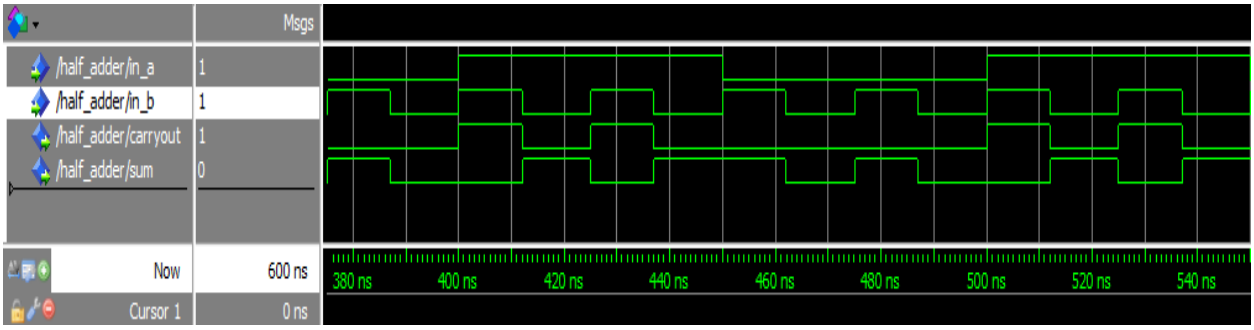

Figure 33 HALF ADDER WAVE FORM

As we observe from the waveform the half adder gives proper output in terms of sum and carry.

ONE BIT FULL ADDER

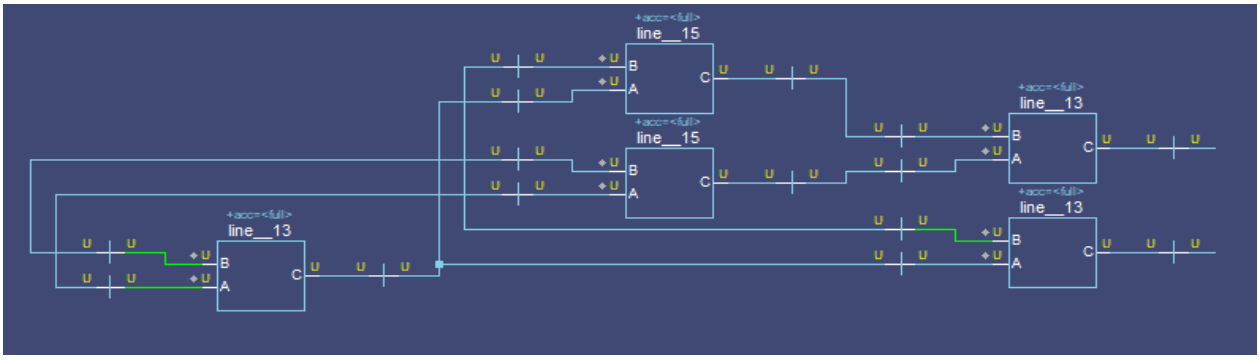

Figure 34 DATA FLOW CIRCUIT OF FULL ADDER

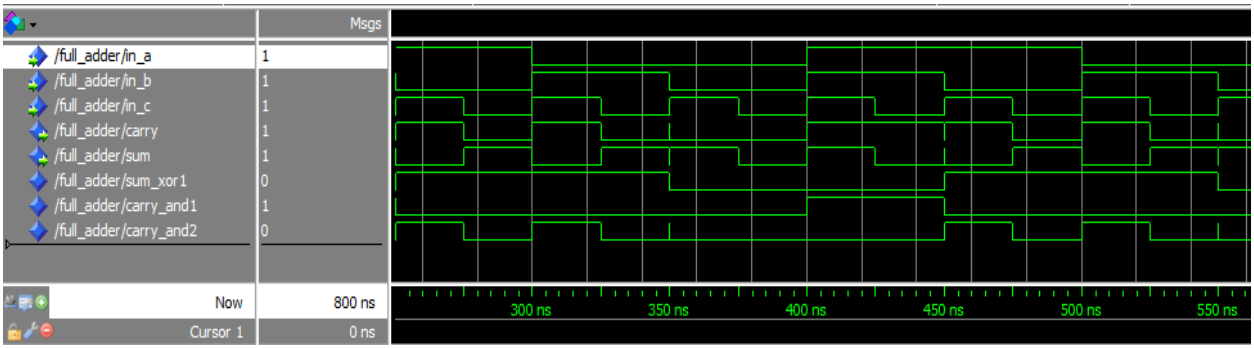

Figure 35 WAFE FORM OF FULL ADDER

The wave form confirms that the full adder circuit implementation is successful.

TWOS COMPLIMENT

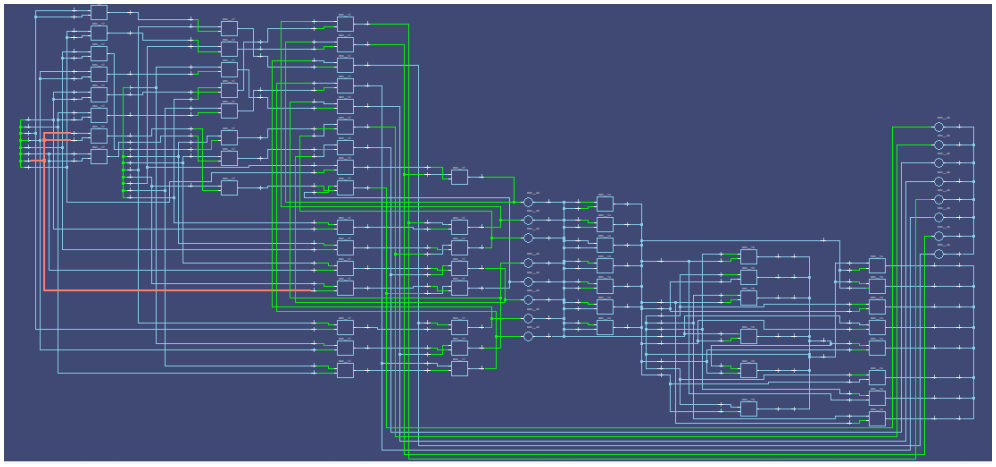

Figure 36 DATA FLOW CIRCUIT OF TWOS COMPLIMENT

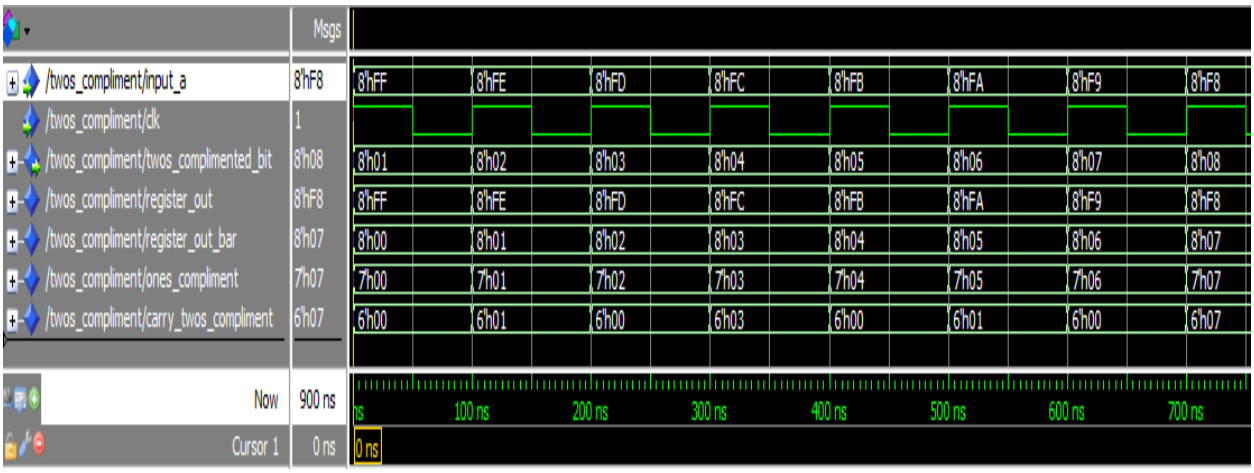

Figure 37 WAVE FORM OF TWOS COMPLIMENT

The wave form shows that this block is successful in performing negative integers to positive.

PARTIAL AND OPERATION BLOCK

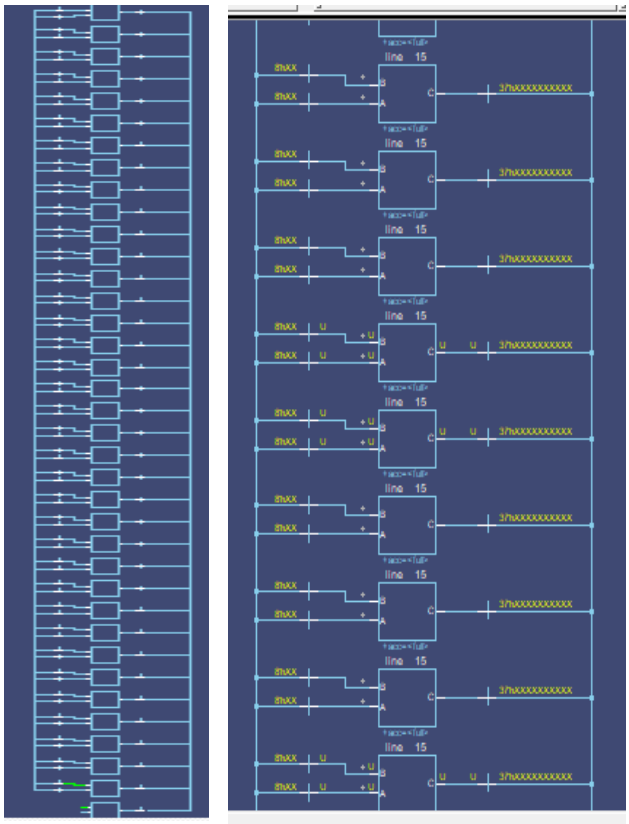

Figure 38 DATA FLOW CIRCUIT OF PARTIAL AND OPERATION

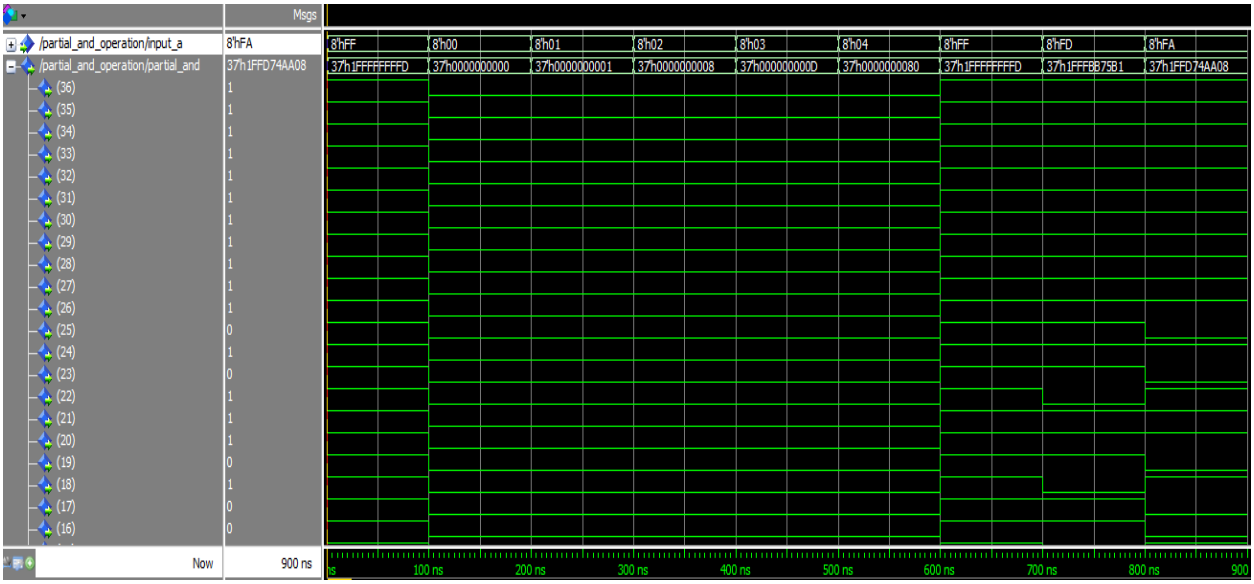

Figure 39 WAVE FORM OF PARTIAL AND OPERATION

From the wave form it is visible partial AND operation block give the output as expected.

A SQUARE BLOCK

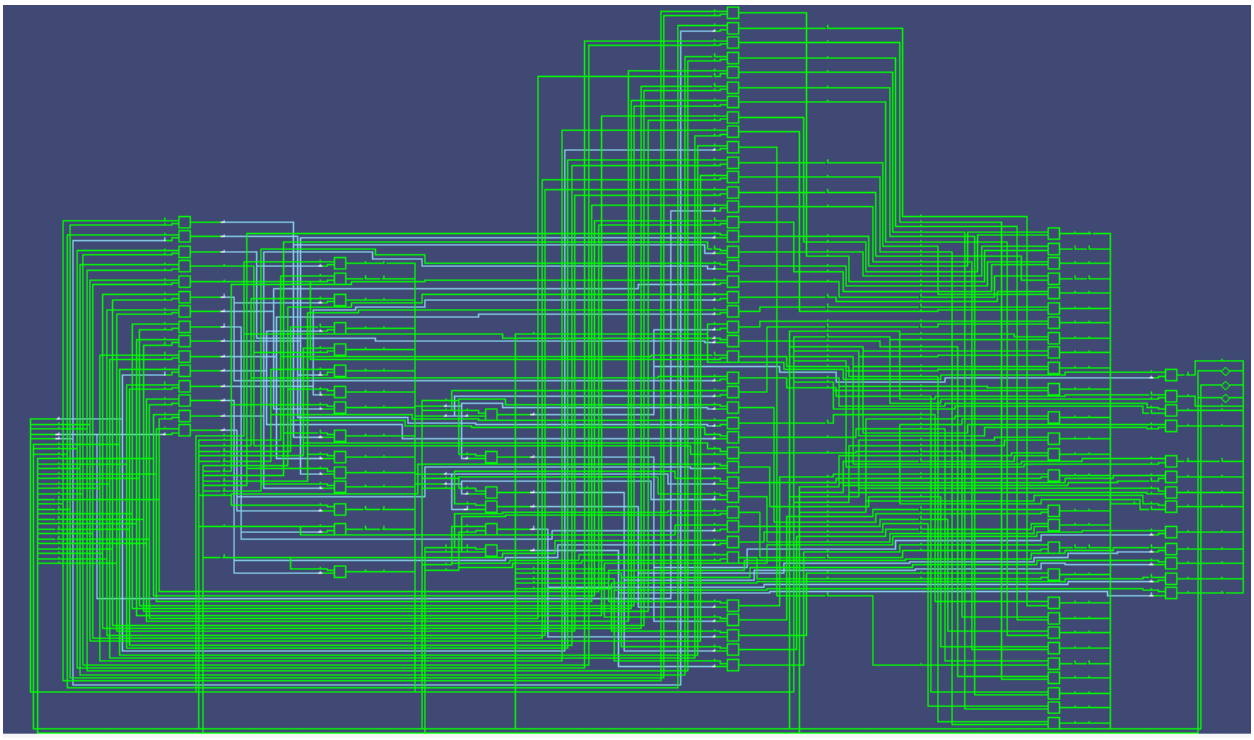

Figure 40 DATA FLOW CIRCUIT OF A SQUARE BLOCK

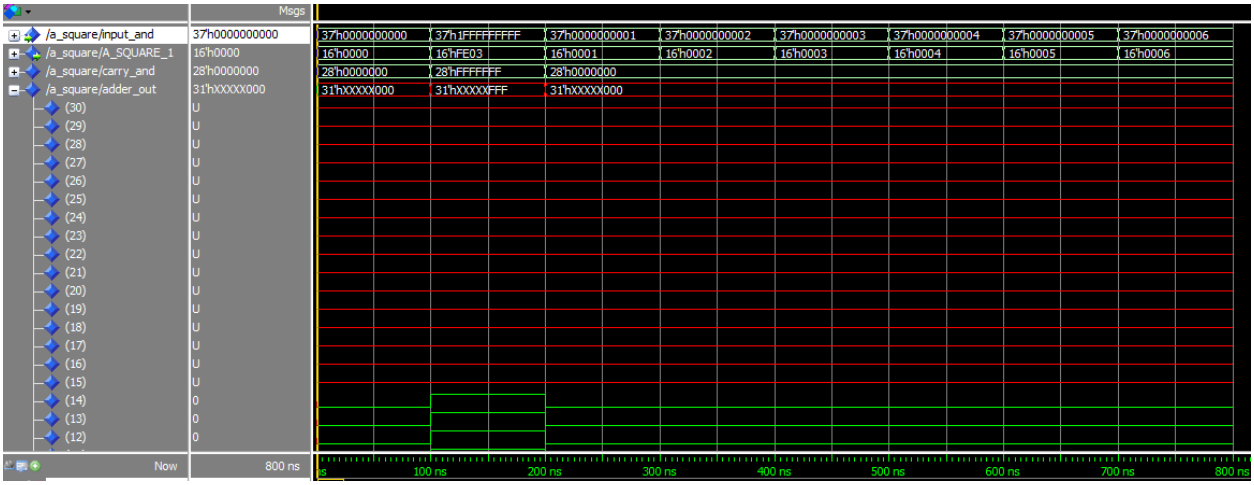

Figure 41 WAVE FORM OF WALLCE TREE ALSO A SQUARE BLOCK

The wave form is shows the output of Wallace tree without AND operation block, and the results confirm that the implementation is successful.

MINUS 1 BLOCK

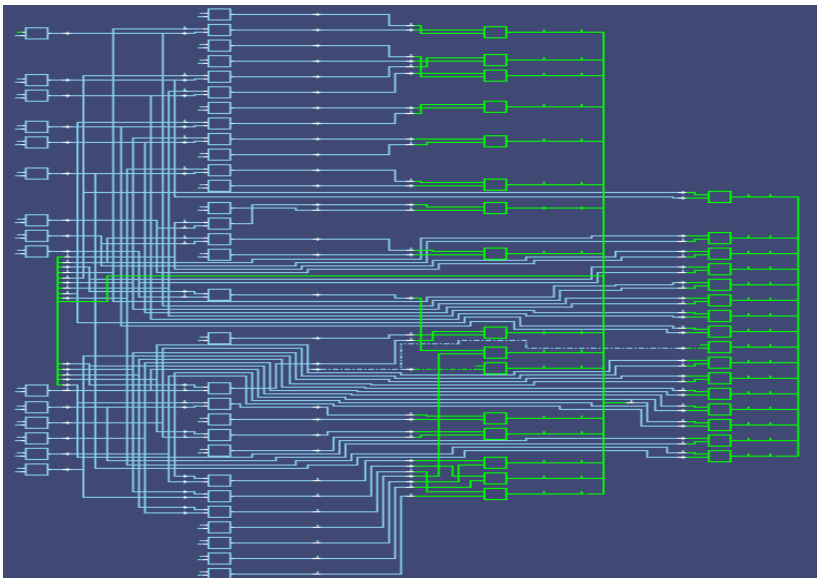

Figure 42 DATA FLOW CIRCUIT OF MINUS ONE BLOCK

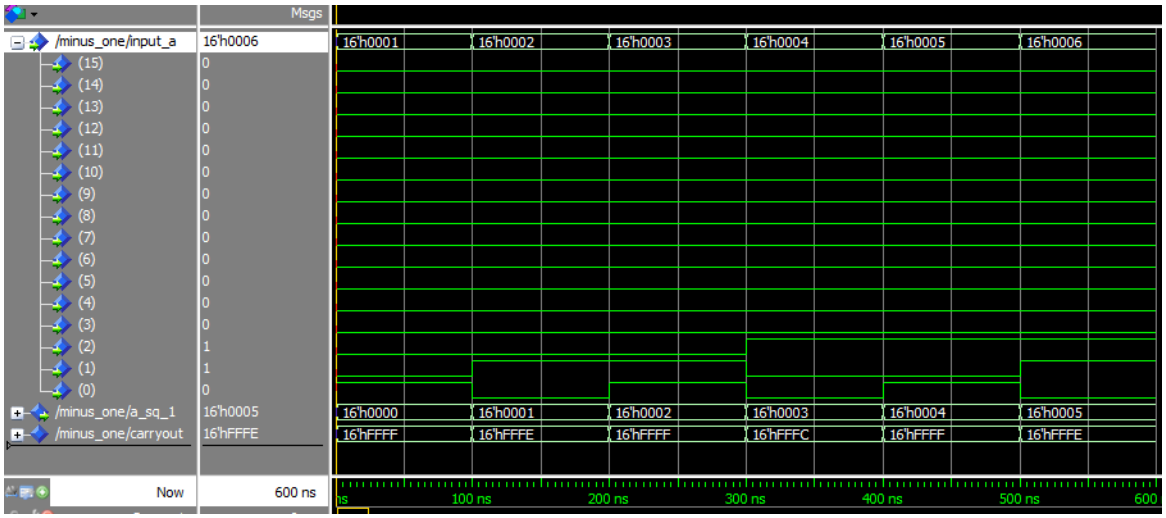

Figure 43 WAVE FORM OF MINUS ONE BLOCK

From the waveform output we see that the block is successful in subtracting minus one from integers.

A SQUARE MINUS 1 BLOCK

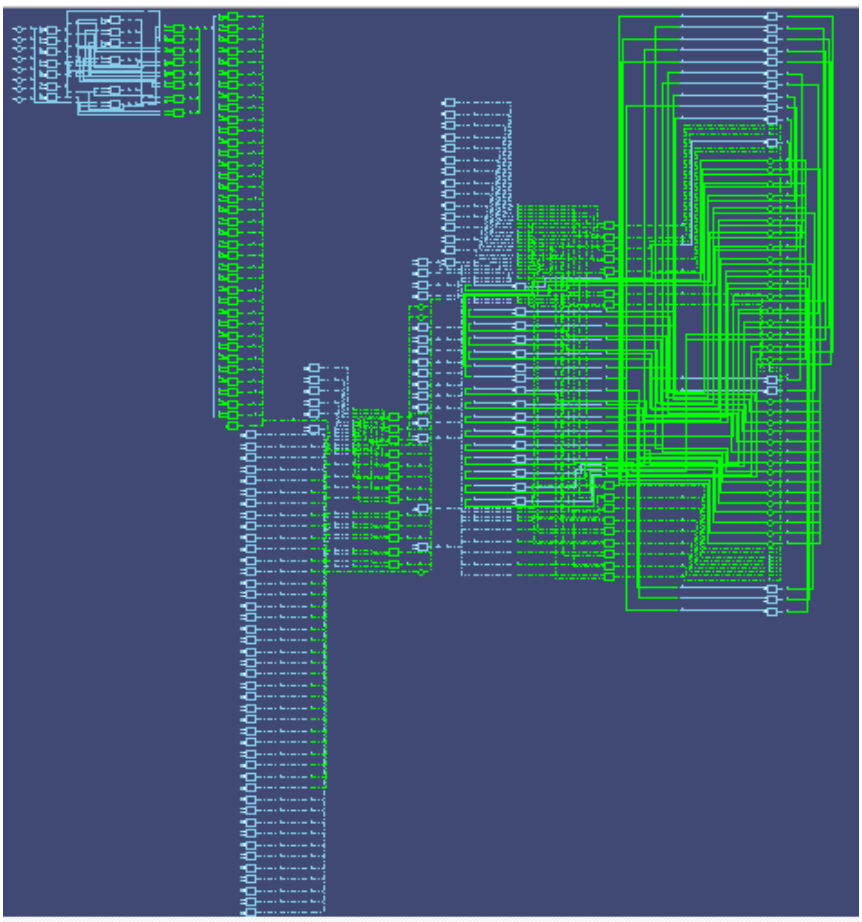

Figure 44 DATA FLOW CIRCUIT OF A SQUARE MINUS ONE BLOCK

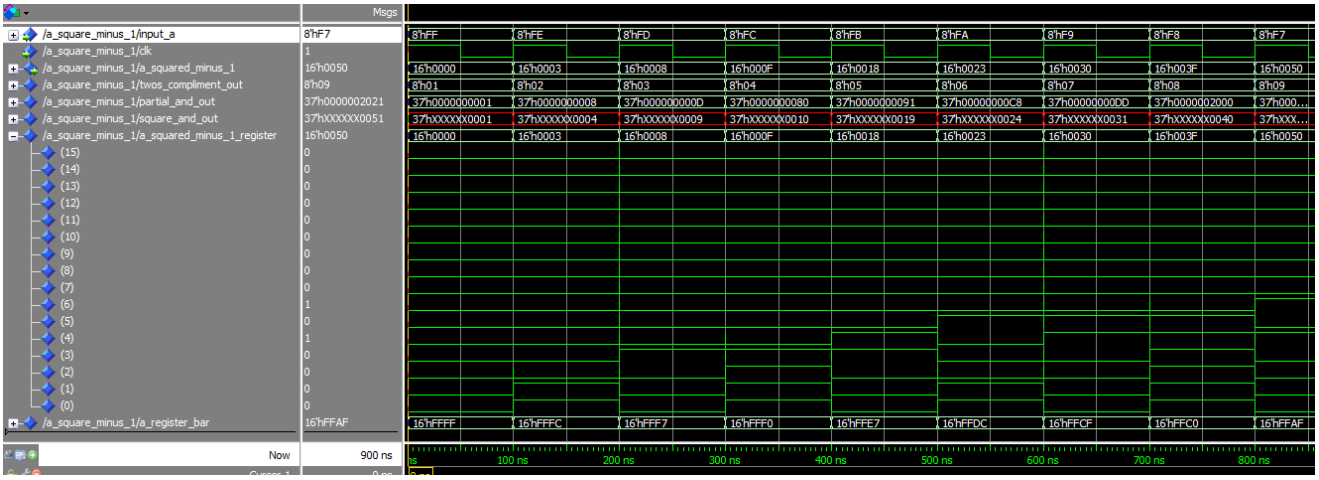

Figure 45 WAVE FORM OF A SQUARE MINUS ONE BLOCK

From the wave form we confirm that the implementation of the mathematical operation is successful.

ALU TEST BENCH

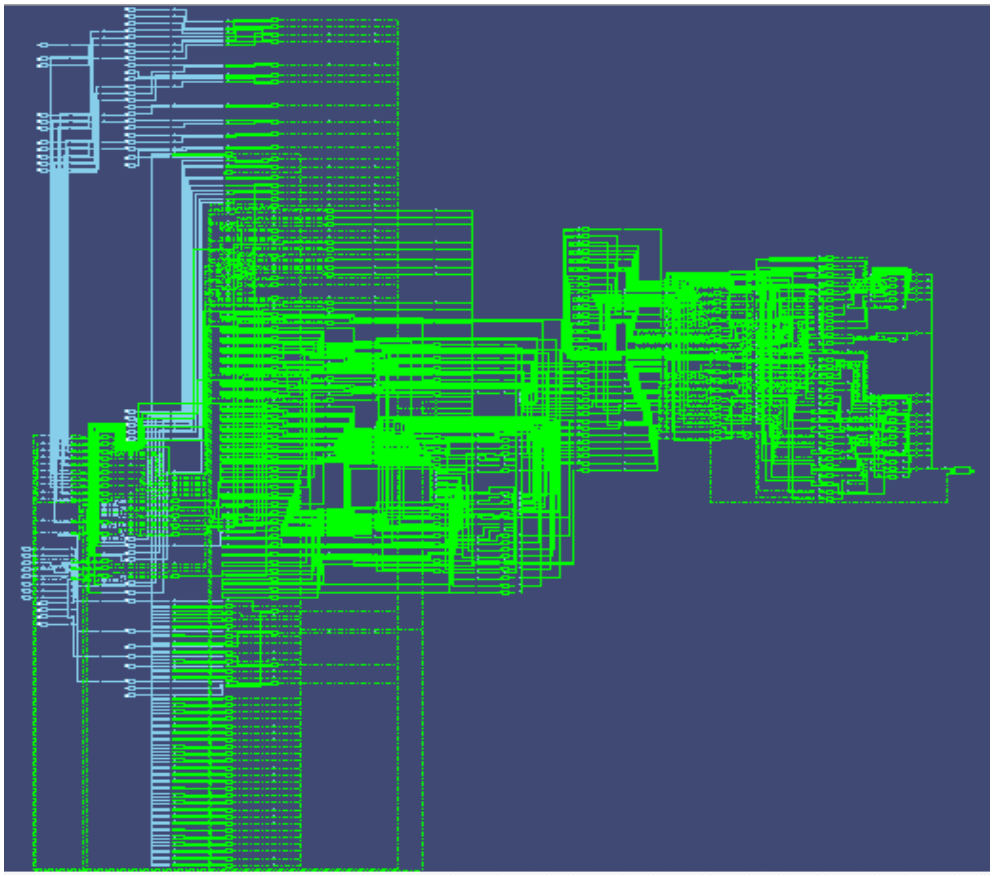

Figure 46 DATA FLOW CIRCUIT ALU BLOCK

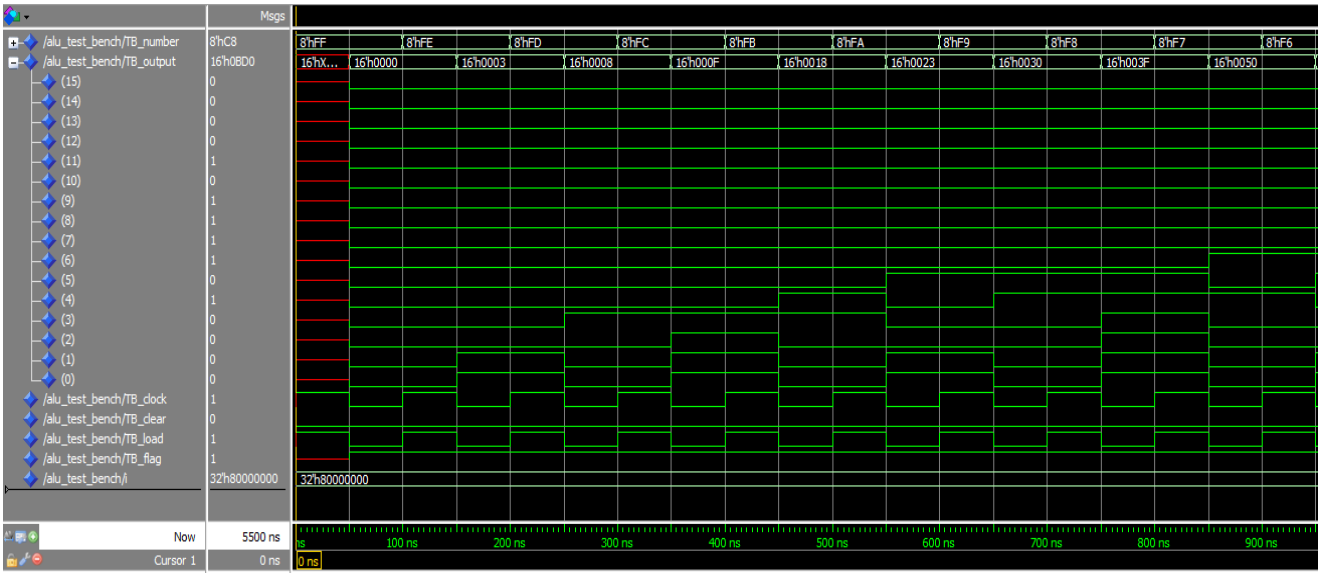

Figure 47 WAVE FORM OF FULL ALU TEST BENCH

Finally, the total ALU block was tested with test bench and we can confirm that the implementation was successful, it follows all the specified specifications, with the aid of structural and behavioral coding the circuit was complete, it works perfectly which we can observe from the wave form above.

Synthesis of ALU

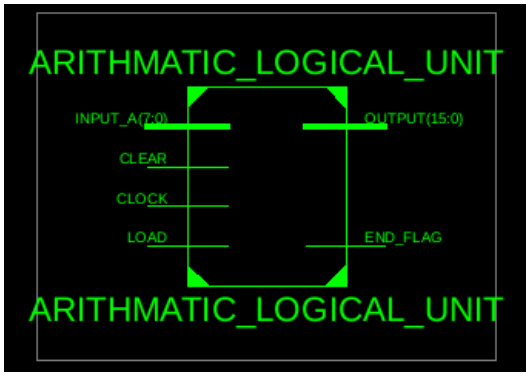

Figure 48 TOP LEVEL SYNHESIS OF ALU VIA XILNX

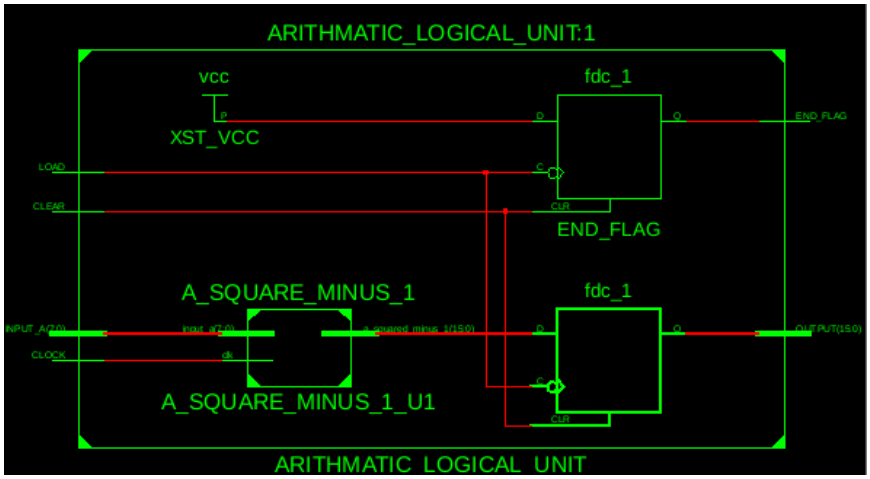

Figure 49 INSIDE OF ALU BLOCK

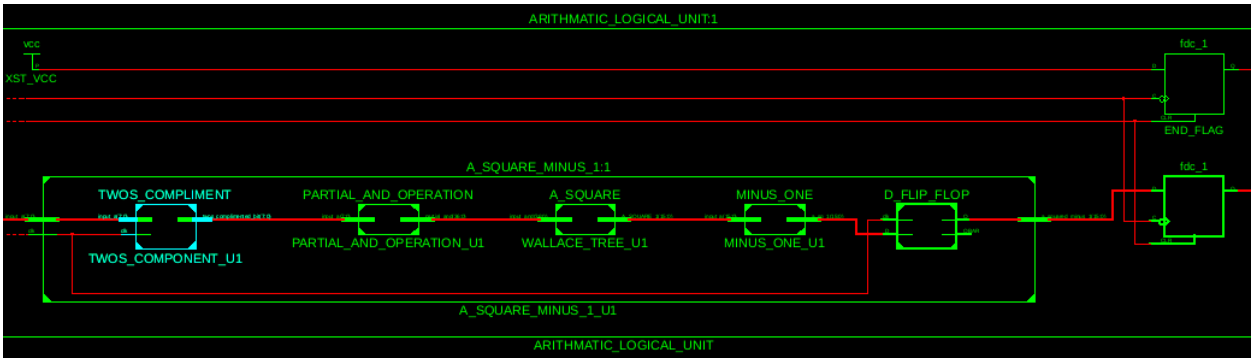

Figure 50 INTERNAL BLOCK OF TOP BLOCK

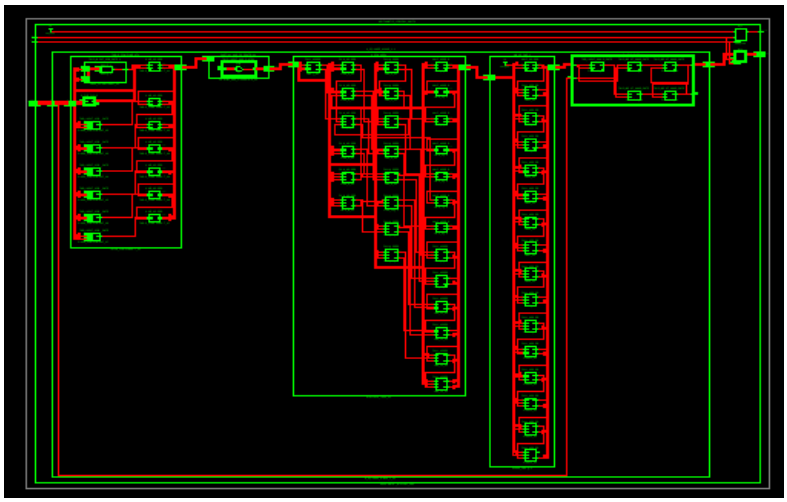

Figure 51 FULL ALU BLOCK EXPANDED

Synthesis Analysis

Area

ARITHMATIC_LOGICAL_UNIT Project Status (11/23/2018 - 21:57:45)			
Project File:	ALU.xise	**Parser Errors:**	No Errors
Module Name:	ARITHMATIC_LOGICAL_UNIT	**Implementation State:**	Synthesized
Target Device:	xa6slx4-3csg225	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	28 Warnings (28 new)
Design Goal:	Balanced	• **Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	
Environment:	System Settings	• **Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	107	2400	4%
Number of fully used LUT-FF pairs	0	107	0%
Number of bonded IOBs	28	132	21%
Number of BUFG/BUFGCTRLs	1	16	6%

Timing Report

==========

Timing Report

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.

FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT

GENERATED AFTER PLACE-and-ROUTE.

Clock Information:

Clock Signal | Clock buffer(FF name) | Load |

LOAD | BUFGP | 17 |

Asynchronous Control Signals Information:

No asynchronous control signals found in this design

Timing Summary:

Speed Grade: -3

Minimum period: No path found

Minimum input arrival time before clock: 17.658ns

Maximum output required time after clock: 3.597ns

Maximum combinational path delay: No path found

Timing Details:

All values displayed in nanoseconds (ns)

===

Timing constraint: Default OFFSET IN BEFORE for Clock 'LOAD'

Total number of paths / destination ports: 100376 / 33

Offset: 17.658ns (Levels of Logic = 15)

Source: CLOCK (PAD)

Destination: OUTPUT_15 (FF)

Destination Clock: LOAD falling

Data Path: CLOCK to OUTPUT_15

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
IBUF:I->O	34	1.222	1.549	CLOCK_IBUF (CLOCK_IBUF)

LUT3:I0->O 24 0.205 1.401
A_SQUARE_MINUS_1_U1/TWOS_COMPONENT_U1/TWOS_COMPLIMENT_U1/SUMMATION_U1/Mxor_C_xo<0>1 (A_SQUARE_MINUS_1_U1/twos_compliment_out<0>)

LUT4:I1->O 3 0.205 1.015
A_SQUARE_MINUS_1_U1/TWOS_COMPONENT_U1/TWOS_COMPLIMENT_U5/SUMMATION_U1/Mxor_C_xo<0>31
(A_SQUARE_MINUS_1_U1/TWOS_COMPONENT_U1/TWOS_COMPLIMENT_U5/SUMMATION_U1/Mxor_C_xo<0>3)

LUT6:I0->O 13 0.203 1.161
A_SQUARE_MINUS_1_U1/TWOS_COMPONENT_U1/TWOS_COMPLIMENT_U6/CARRY_U2/C
(A_SQUARE_MINUS_1_U1/TWOS_COMPONENT_U1/carry_twos_compliment<5>)

LUT4:I1->O 3 0.205 0.879
A_SQUARE_MINUS_1_U1/PARTIAL_AND_OPERATION_U1/product_u14/C1
(A_SQUARE_MINUS_1_U1/partial_and_out<14>)

LUT5:I2->O 2 0.205 0.961
A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/LINE_7_U1/CARRY_U5/C1
(A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/carry_and<9>)

LUT6:I1->O 3 0.203 0.879
A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/LINE_8_U2/SUMMATION_U2/Mxor_C_xo<0>1
(A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/adder_out<7>)

LUT3:I0->O 3 0.205 0.995
A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/LINE_8_U3/SUMMATION_U2/Mxor_C_xo<0>1
(A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/adder_out<8>)

LUT5:I0->O 2 0.203 0.864
A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/LINE_8_U4/CARRY_U2/C1
(A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/carry_and<15>)

LUT5:I1->O 4 0.203 0.788
A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/LINE_9_U3/CARRY_U5/C1
(A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/carry_and<18>)

LUT5:I3->O 4 0.203 0.931
A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/LINE_11_U2/CARRY_U5/C1
(A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/carry_and<23>)

LUT5:I1->O 2 0.203 0.617
A_SQUARE_MINUS_1_U1/WALLACE_TREE_U1/LINE_12_U2/SUMMATION_U2/Mxor_C_xo<0>1
(A_SQUARE_MINUS_1_U1/square_and_out<12>)

LUT6:I5->O 3 0.205 0.755
A_SQUARE_MINUS_1_U1/MINUS_ONE_U1/LINE13_U1/SUMMATION_U2/Mxor_C_xo<0>11
(A_SQUARE_MINUS_1_U1/MINUS_ONE_U1/LINE13_U1/SUMMATION_U2/Mxor_C_xo<0>1)

```
    LUT4:I2->O            1   0.203   0.684
A_SQUARE_MINUS_1_U1/MINUS_ONE_U1/LINE15_U1/SUMMATION_U2/Mxor_C_xo<0>1
(A_SQUARE_MINUS_1_U1/a_squared_minus_1_register<15>)

    LUT3:I1->O            2   0.203   0.000
A_SQUARE_MINUS_1_U1/OUTPUT_REGISTER_U15/FLIPFLOP_U3/C1 (output_results<15>)

    FDC_1:D                   0.102          OUTPUT_15

    ----------------------------------------

    Total                    17.658ns (4.178ns logic, 13.480ns route)

                               (23.7% logic, 76.3% route)


=========================================================================

Timing constraint: Default OFFSET OUT AFTER for Clock 'LOAD'

  Total number of paths / destination ports: 17 / 17

-------------------------------------------------------------------------

Offset:              3.597ns (Levels of Logic = 1)

  Source:            OUTPUT_15 (FF)

  Destination:       OUTPUT<15> (PAD)

  Source Clock:      LOAD falling


  Data Path: OUTPUT_15 to OUTPUT<15>

                            Gate     Net

    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)

    ----------------------------------------  ------------

    FDC_1:C->Q            1   0.447   0.579  OUTPUT_15 (OUTPUT_15)

    OBUF:I->O                 2.571          OUTPUT_15_OBUF (OUTPUT<15>)

    ----------------------------------------

    Total                    3.597ns (3.018ns logic, 0.579ns route)

                               (83.9% logic, 16.1% route)
```

Conclusion

From the specified design we have tried to achieve minimum delay and ensuring area minimization where possible by using half adders and full adders combination on the basis of Wallace tree. From our simulation in Modelsim, we have successfully implemented arithmetical logical unit as specified. Also, we have implemented our circuit virtually in an FPGA via Xilinx and synthesized the design without error.

From this project we have gained knowledge on designing digital circuits using structural and behavioral coding with hierarchical modelling

Codes

TWO INPUT AND GATE

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;

entity TWO_INPUT_XOR_GATE is
   port( A, B : in std_logic;
         C : out std_logic);
end TWO_INPUT_XOR_GATE;

architecture TWO_INPUT_XOR_GATE_ARCHITECTURE of TWO_INPUT_XOR_GATE is
begin
   C <= A xor B;
end TWO_INPUT_XOR_GATE_ARCHITECTURE;
```

TWO INPUT XOR GATE

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;

entity TWO_INPUT_XOR_GATE is
   port( A, B : in std_logic;
         C : out std_logic);
end TWO_INPUT_XOR_GATE;


architecture TWO_INPUT_XOR_GATE_ARCHITECTURE of TWO_INPUT_XOR_GATE is
```

```
begin
   C <= A xor B;
end TWO_INPUT_XOR_GATE_ARCHITECTURE;
```

TWO INPUT NAND GATE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity TWO_INPUT_NAND_GATE is
   port( A, B : in std_logic;
             C : out std_logic);
end TWO_INPUT_NAND_GATE;

architecture TWO_INPUT_NAND_GATE_ARCHITECTURE of TWO_INPUT_NAND_GATE is


begin

   C <= NOT(A AND B);
end TWO_INPUT_NAND_GATE_ARCHITECTURE;
```

THREE INPUT NAND GATE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity THREE_INPUT_NAND_GATE is
   port( A, B ,C : in std_logic;
             D : out std_logic);
end THREE_INPUT_NAND_GATE;

architecture THREE_INPUT_NAND_GATE_ARCHITECTURE of THREE_INPUT_NAND_GATE is


begin
   D <= NOT(A AND B AND C);
end THREE_INPUT_NAND_GATE_ARCHITECTURE;
```

TWO INPUT OR GATE

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;

entity TWO_INPUT_XOR_GATE is
   port( A, B : in std_logic;
             C : out std_logic);
end TWO_INPUT_XOR_GATE;
architecture TWO_INPUT_XOR_GATE_ARCHITECTURE of TWO_INPUT_XOR_GATE is
```

```
begin
   C <= A xor B;
end TWO_INPUT_XOR_GATE_ARCHITECTURE;
```

D FLIP FLOP

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

entity D_FLIP_FLOP is


Port ( D,clk : in  STD_LOGIC;


Q,QBAR : out STD_LOGIC);


end D_FLIP_FLOP;


architecture D_FLIP_FLOP_ARCHITECTURE of D_FLIP_FLOP is


component TWO_INPUT_NAND_GATE


port(a,b: in STD_LOGIC;


c:out STD_LOGIC);


end component;


signal NAND1,NAND2, NAND3 ,SIGNALQ,SIGNALQBAR:STD_LOGIC;
```

```
begin


FLIPFLOP_U0: TWO_INPUT_NAND_GATE port map(D,clk,NAND1);


FLIPFLOP_U1: TWO_INPUT_NAND_GATE port map(D,D,NAND2);


FLIPFLOP_U2: TWO_INPUT_NAND_GATE port map(NAND2,clk,NAND3);


FLIPFLOP_U3: TWO_INPUT_NAND_GATE port map(NAND1,SIGNALQBAR,SIGNALQ);


FLIPFLOP_U4: TWO_INPUT_NAND_GATE port map(NAND3,SIGNALQ,SIGNALQBAR);

Q<=SIGNALQ;
QBAR<=SIGNALQBAR;


end D_FLIP_FLOP_ARCHITECTURE;
```

ONE BIT HALF ADDER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity HALF_ADDER is
port
(
in_a : in std_logic;
in_b : in std_logic;
carryout : out std_logic;
sum : out std_logic
);
end HALF_ADDER;

architecture HALF_ADDER_ARCHITECTURE of HALF_ADDER is
```

```
component TWO_INPUT_XOR_GATE
port
(
 a : in std_logic;
 b : in std_logic;
 c : out std_logic
);
end component;

component TWO_INPUT_AND_GATE
port
(
a : in std_logic;
b : in std_logic;
c : out std_logic
);
end component;

begin
-- Implement circuit
SUMMATION_U1: TWO_INPUT_XOR_GATE port map (in_a, in_b, sum);
CARRY_U2: TWO_INPUT_AND_GATE port map (in_a, in_b, carryout);
end HALF_ADDER_ARCHITECTURE;
```

ONE BIT FULL ADDER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity FULL_ADDER is
port
(
in_a, in_b, in_c : in std_logic;
carry, sum : out std_logic
);
end FULL_ADDER;

architecture FULL_ADDER_ARCHITECTURE of FULL_ADDER is

signal sum_xor1, carry_and1, carry_and2 : std_logic;

component TWO_INPUT_XOR_GATE
port
(
a, b : in std_logic;
c : out std_logic
);
end component;

component TWO_INPUT_OR_GATE
port
(
a, b : in std_logic;
c : out std_logic
```

```
);
end component;
component TWO_INPUT_AND_GATE
port
(
a, b : in std_logic;
c : out std_logic
);
end component;
begin

SUMMATION_U1: TWO_INPUT_XOR_GATE port map(in_a, in_b, sum_xor1);
SUMMATION_U2: TWO_INPUT_XOR_GATE port map(sum_xor1, in_c, sum);

CARRY_U3: TWO_INPUT_AND_GATE port map (in_a, in_b, carry_and1);
CARRY_U4: TWO_INPUT_AND_GATE port map (sum_xor1, in_c, carry_and2);
CARRY_U5: TWO_INPUT_OR_GATE port map (carry_and1, carry_and2, carry);
end FULL_ADDER_ARCHITECTURE;
```

TWOS COMPLIMENT

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity TWOS_COMPLIMENT is
port

( input_a : in std_logic_vector (7 downto 0);
  clk : in std_logic;
  twos_complimented_bit : out std_logic_vector (7 downto 0)
);
end TWOS_COMPLIMENT;

architecture TWOS_COMPLIMENT_ARCHITECTURE of TWOS_COMPLIMENT is

component D_FLIP_FLOP
port(
D,clk : in  STD_LOGIC;

Q,QBAR : out STD_LOGIC);

end component;

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

component TWO_INPUT_XOR_GATE
```

```
port
(
 a : in std_logic;
 b : in std_logic;
 c : out std_logic
);
end component;
signal  register_out : std_logic_vector (7 downto 0);
signal  register_out_bar : std_logic_vector (7 downto 0);
signal  ones_compliment : std_logic_vector (6 downto 0);
signal  carry_twos_compliment : std_logic_vector (5 downto 0);

begin

--register_load
LOADING_REGISTER_U1 : D_FLIP_FLOP port map
(input_a(0),clk,register_out(0),register_out_bar(0));
LOADING_REGISTER_U2 : D_FLIP_FLOP port map
(input_a(1),clk,register_out(1),register_out_bar(1));
LOADING_REGISTER_U3 : D_FLIP_FLOP port map
(input_a(2),clk,register_out(2),register_out_bar(2));
LOADING_REGISTER_U4 : D_FLIP_FLOP port map
(input_a(3),clk,register_out(3),register_out_bar(3));
LOADING_REGISTER_U5 : D_FLIP_FLOP port map
(input_a(4),clk,register_out(4),register_out_bar(4));
LOADING_REGISTER_U6 : D_FLIP_FLOP port map
(input_a(5),clk,register_out(5),register_out_bar(5));
LOADING_REGISTER_U7 : D_FLIP_FLOP port map
(input_a(6),clk,register_out(6),register_out_bar(6));
LOADING_REGISTER_U8 : D_FLIP_FLOP port map
(input_a(7),clk,register_out(7),register_out_bar(7));

--ones compliment
ONES_COMPLIMENT_U1 : TWO_INPUT_XOR_GATE port map (register_out(7),
register_out(0), ones_compliment(0));

ONES_COMPLIMENT_U2 : TWO_INPUT_XOR_GATE port map (register_out(7),
register_out(1), ones_compliment(1));

ONES_COMPLIMENT_U3 : TWO_INPUT_XOR_GATE port map (register_out(7),
register_out(2), ones_compliment(2));

ONES_COMPLIMENT_U4 : TWO_INPUT_XOR_GATE port map (register_out(7),
register_out(3), ones_compliment(3));

ONES_COMPLIMENT_U5 : TWO_INPUT_XOR_GATE port map (register_out(7),
register_out(4), ones_compliment(4));

ONES_COMPLIMENT_U6 : TWO_INPUT_XOR_GATE port map (register_out(7),
register_out(5), ones_compliment(5));

ONES_COMPLIMENT_U7 : TWO_INPUT_XOR_GATE port map (register_out(7),
register_out(6), ones_compliment(6));
```

```
TWOS_COMPLIMENT_U1 : HALF_ADDER port map (
ones_compliment(0),register_out(7),carry_twos_compliment(0),twos_complimented
_bit(0));
TWOS_COMPLIMENT_U2 : HALF_ADDER port map (
ones_compliment(1),carry_twos_compliment(0),carry_twos_compliment(1),twos_com
plimented_bit(1));
TWOS_COMPLIMENT_U3 : HALF_ADDER port map (
ones_compliment(2),carry_twos_compliment(1),carry_twos_compliment(2),twos_com
plimented_bit(2));
TWOS_COMPLIMENT_U4 : HALF_ADDER port map (
ones_compliment(3),carry_twos_compliment(2),carry_twos_compliment(3),twos_com
plimented_bit(3));
TWOS_COMPLIMENT_U5 : HALF_ADDER port map (
ones_compliment(4),carry_twos_compliment(3),carry_twos_compliment(4),twos_com
plimented_bit(4));
TWOS_COMPLIMENT_U6 : HALF_ADDER port map (
ones_compliment(5),carry_twos_compliment(4),carry_twos_compliment(5),twos_com
plimented_bit(5));
TWOS_COMPLIMENT_U7 : HALF_ADDER port map (
ones_compliment(6),carry_twos_compliment(5),twos_complimented_bit(7),twos_com
plimented_bit(6));

end TWOS_COMPLIMENT_ARCHITECTURE;
```

PARTIAL AND OPERATION

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity PARTIAL_AND_OPERATION is
port

( input_a : in std_logic_vector (7 downto 0);
 partial_and : out std_logic_vector (36 downto 0)
);
end PARTIAL_AND_OPERATION;

architecture PARTIAL_AND_OPERATION_ARCHITECTURE of PARTIAL_AND_OPERATION is

component TWO_INPUT_AND_GATE
port
(
 a : in std_logic;
 b : in std_logic;
 c : out std_logic
);
end component;

begin

product_u0 : TWO_INPUT_AND_GATE port map (
input_a(0),input_a(0),partial_and(0));
product_u1 : TWO_INPUT_AND_GATE port map ( '0','0',partial_and(1));
product_u2 : TWO_INPUT_AND_GATE port map (
input_a(1),input_a(0),partial_and(2));
```

```
product_u3 : TWO_INPUT_AND_GATE port map (
input_a(1),input_a(1),partial_and(3));
product_u4 : TWO_INPUT_AND_GATE port map (
input_a(2),input_a(0),partial_and(4));
product_u5 : TWO_INPUT_AND_GATE port map (
input_a(3),input_a(0),partial_and(5));
product_u6 : TWO_INPUT_AND_GATE port map (
input_a(2),input_a(1),partial_and(6));
product_u7 : TWO_INPUT_AND_GATE port map (
input_a(2),input_a(2),partial_and(7));
product_u8 : TWO_INPUT_AND_GATE port map (
input_a(4),input_a(0),partial_and(8));
product_u9 : TWO_INPUT_AND_GATE port map (
input_a(3),input_a(1),partial_and(9));
product_u10 : TWO_INPUT_AND_GATE port map (
input_a(5),input_a(0),partial_and(10));
product_u11 : TWO_INPUT_AND_GATE port map (
input_a(4),input_a(1),partial_and(11));
product_u12 : TWO_INPUT_AND_GATE port map (
input_a(3),input_a(2),partial_and(12));
product_u13 : TWO_INPUT_AND_GATE port map (
input_a(3),input_a(3),partial_and(13));
product_u14 : TWO_INPUT_AND_GATE port map (
input_a(6),input_a(0),partial_and(14));
product_u15 : TWO_INPUT_AND_GATE port map (
input_a(5),input_a(1),partial_and(15));
product_u16 : TWO_INPUT_AND_GATE port map (
input_a(4),input_a(2),partial_and(16));
product_u17 : TWO_INPUT_AND_GATE port map (
input_a(7),input_a(0),partial_and(17));
product_u18 : TWO_INPUT_AND_GATE port map (
input_a(6),input_a(1),partial_and(18));
product_u19 : TWO_INPUT_AND_GATE port map (
input_a(5),input_a(2),partial_and(19));
product_u20 : TWO_INPUT_AND_GATE port map (
input_a(4),input_a(3),partial_and(20));
product_u21 : TWO_INPUT_AND_GATE port map (
input_a(4),input_a(4),partial_and(21));
product_u22 : TWO_INPUT_AND_GATE port map (
input_a(7),input_a(1),partial_and(22));
product_u23 : TWO_INPUT_AND_GATE port map (
input_a(6),input_a(2),partial_and(23));
product_u24 : TWO_INPUT_AND_GATE port map (
input_a(5),input_a(3),partial_and(24));
product_u25 : TWO_INPUT_AND_GATE port map (
input_a(7),input_a(2),partial_and(25));
product_u26 : TWO_INPUT_AND_GATE port map (
input_a(6),input_a(3),partial_and(26));
product_u27 : TWO_INPUT_AND_GATE port map (
input_a(5),input_a(4),partial_and(27));
product_u28 : TWO_INPUT_AND_GATE port map (
input_a(5),input_a(5),partial_and(28));
product_u29 : TWO_INPUT_AND_GATE port map (
input_a(7),input_a(3),partial_and(29));
product_u30 : TWO_INPUT_AND_GATE port map (
input_a(6),input_a(4),partial_and(30));
```

```
product_u31 : TWO_INPUT_AND_GATE port map (
input_a(7),input_a(4),partial_and(31));
product_u32 : TWO_INPUT_AND_GATE port map (
input_a(6),input_a(5),partial_and(32));
product_u33 : TWO_INPUT_AND_GATE port map (
input_a(6),input_a(6),partial_and(33));
product_u34 : TWO_INPUT_AND_GATE port map (
input_a(7),input_a(5),partial_and(34));
product_u35 : TWO_INPUT_AND_GATE port map (
input_a(7),input_a(6),partial_and(35));
product_u36 : TWO_INPUT_AND_GATE port map (
input_a(7),input_a(7),partial_and(36));

end PARTIAL_AND_OPERATION_ARCHITECTURE;
```

A SQUARE OPERATION

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity A_SQUARE is
port

( input_and: in std_logic_vector (36 downto 0);
 A_SQUARE_1 :out std_logic_vector (15 downto 0)
);
end A_SQUARE;

architecture A_SQUARE_ARCHITECTURE of A_SQUARE is


signal  carry_and : std_logic_vector (27 downto 0);
signal  adder_out : std_logic_vector (30 downto 0);

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

component FULL_ADDER
port
(
in_a : in std_logic;
in_b : in std_logic;
in_c : in std_logic;
carry : out std_logic;
```

```
sum : out std_logic
);
end component;


begin

LINE_0_U1 : A_SQUARE_1(0)<= input_and(0);

LINE_1_U1 : A_SQUARE_1(1)<= input_and(1);

LINE_2_U1 : HALF_ADDER port map
(input_and(2),input_and(3),carry_and(0),A_SQUARE_1(2));

LINE_3_U1 : HALF_ADDER port map
(carry_and(0),input_and(4),carry_and(1),A_SQUARE_1(3));

LINE_4_U1 : FULL_ADDER port map
(input_and(5),input_and(6),input_and(7),carry_and(2),adder_out(0));
LINE_4_U2 : HALF_ADDER port map
(adder_out(0),carry_and(1),carry_and(3),A_SQUARE_1(4));

LINE_5_U1 : FULL_ADDER port map
(input_and(8),input_and(9),carry_and(2),carry_and(4),adder_out(1));
LINE_5_U2 : HALF_ADDER port map
(adder_out(1),carry_and(3),carry_and(5),A_SQUARE_1(5));

LINE_6_U1 : FULL_ADDER port map
(input_and(10),input_and(11),input_and(12),carry_and(6),adder_out(2));
LINE_6_U2 : FULL_ADDER port map
(input_and(13),adder_out(2),carry_and(4),carry_and(7),adder_out(3));
LINE_6_U4 : HALF_ADDER port map
(adder_out(3),carry_and(5),carry_and(8),A_SQUARE_1(6));

LINE_7_U1 : FULL_ADDER port map
(input_and(14),input_and(15),input_and(16),carry_and(9),adder_out(4));
LINE_7_U2 : FULL_ADDER port map
(adder_out(4),carry_and(6),carry_and(7),carry_and(10),adder_out(5));
LINE_7_U3 : HALF_ADDER port map
(adder_out(5),carry_and(8),carry_and(11),A_SQUARE_1(7));

LINE_8_U1 : FULL_ADDER port map
(input_and(17),input_and(18),input_and(19),carry_and(12),adder_out(6));
LINE_8_U2 : FULL_ADDER port map
(input_and(20),carry_and(9),adder_out(6),carry_and(13),adder_out(7));
LINE_8_U3 : FULL_ADDER port map
(adder_out(7),input_and(21),carry_and(10),carry_and(14),adder_out(8));
LINE_8_U4 : HALF_ADDER port map
(adder_out(8),carry_and(11),carry_and(15),A_SQUARE_1(8));

LINE_9_U1 : FULL_ADDER port map
(input_and(22),input_and(23),input_and(24),carry_and(16),adder_out(9));
LINE_9_U2 : FULL_ADDER port map
(adder_out(9),carry_and(12),carry_and(13),carry_and(17),adder_out(10));
LINE_9_U3 : FULL_ADDER port map
(adder_out(10),carry_and(14),carry_and(15),carry_and(18),A_SQUARE_1(9));
```

```
LINE_10_U1 : FULL_ADDER port map
(input_and(25),input_and(26),input_and(27),carry_and(19),adder_out(11));
LINE_10_U2 : FULL_ADDER port map
(adder_out(11),input_and(28),carry_and(16),carry_and(20),adder_out(12));
LINE_10_U3 : FULL_ADDER port map
(adder_out(12),carry_and(17),carry_and(18),carry_and(21),A_SQUARE_1(10));

LINE_11_U1 : FULL_ADDER port map
(input_and(29),input_and(30),carry_and(19),carry_and(22),adder_out(13));
LINE_11_U2 : FULL_ADDER port map
(adder_out(13),carry_and(20),carry_and(21),carry_and(23),A_SQUARE_1(11));

LINE_12_U1 : FULL_ADDER port map
(input_and(31),input_and(32),input_and(33),carry_and(24),adder_out(14));
LINE_12_U2 : FULL_ADDER port map
(adder_out(14),carry_and(23),carry_and(22),carry_and(25),A_SQUARE_1(12));

LINE_13_U2 : FULL_ADDER port map
(input_and(34),carry_and(24),carry_and(25),carry_and(26),A_SQUARE_1(13));

LINE_14_U1 : FULL_ADDER port map
(input_and(35),input_and(36),carry_and(26),carry_and(27),A_SQUARE_1(14));

LINE_15_U1 : A_SQUARE_1(15)<=carry_and(27);


end A_SQUARE_ARCHITECTURE;
```

MINUS ONE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


entity MINUS_ONE is
port(

  input_a : in std_logic_vector (15 downto 0);
  a_sq_1 : out std_logic_vector (15 downto 0)
);
end MINUS_ONE;

architecture MINUS_ONE_ARCHITECTURE of MINUS_ONE is

component HALF_ADDER
port
(
in_a : in std_logic;
in_b : in std_logic;
carryout : out std_logic;
sum : out std_logic
```

```
);
end component;

component FULL_ADDER
port
(
in_a : in std_logic;
in_b : in std_logic;
in_c : in std_logic;
carry : out std_logic;
sum : out std_logic
);
end component;


signal carryout : std_logic_vector (15 downto 0);
begin

LINE0_U1 : HALF_ADDER port map (input_a(0),'1',carryout(0),a_sq_1(0));
LINE1_U1 : FULL_ADDER port map
(input_a(1),'1',carryout(0),carryout(1),a_sq_1(1));
LINE2_U1 : FULL_ADDER port map
(input_a(2),'1',carryout(1),carryout(2),a_sq_1(2));
LINE3_U1 : FULL_ADDER port map
(input_a(3),'1',carryout(2),carryout(3),a_sq_1(3));
LINE4_U1 : FULL_ADDER port map
(input_a(4),'1',carryout(3),carryout(4),a_sq_1(4));
LINE5_U1 : FULL_ADDER port map
(input_a(5),'1',carryout(4),carryout(5),a_sq_1(5));
LINE6_U1 : FULL_ADDER port map
(input_a(6),'1',carryout(5),carryout(6),a_sq_1(6));
LINE7_U1 : FULL_ADDER port map
(input_a(7),'1',carryout(6),carryout(7),a_sq_1(7));
LINE8_U1 : FULL_ADDER port map
(input_a(8),'1',carryout(7),carryout(8),a_sq_1(8));
LINE9_U1 : FULL_ADDER port map
(input_a(9),'1',carryout(8),carryout(9),a_sq_1(9));
LINE10_U1 : FULL_ADDER port map
(input_a(10),'1',carryout(9),carryout(10),a_sq_1(10));
LINE11_U1 : FULL_ADDER port map
(input_a(11),'1',carryout(10),carryout(11),a_sq_1(11));
LINE12_U1 : FULL_ADDER port map
(input_a(12),'1',carryout(11),carryout(12),a_sq_1(12));
LINE13_U1 : FULL_ADDER port map
(input_a(13),'1',carryout(12),carryout(13),a_sq_1(13));
LINE14_U1 : FULL_ADDER port map
(input_a(14),'1',carryout(13),carryout(14),a_sq_1(14));
LINE15_U1 : FULL_ADDER port map
(input_a(15),'1',carryout(14),carryout(15),a_sq_1(15));
end MINUS_ONE_ARCHITECTURE;
```

A SQUARE MINUS ONE

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity A_SQUARE_MINUS_1 is
port

( input_a : in std_logic_vector (7 downto 0);
  clk : in std_logic;
  a_squared_minus_1 : out std_logic_vector (15 downto 0)
);
end A_SQUARE_MINUS_1;

architecture A_SQUARE_MINUS_1_ARCHITECTURE of A_SQUARE_MINUS_1 is

component D_FLIP_FLOP
port(
D,clk : in  STD_LOGIC;

Q,QBAR : out STD_LOGIC);

end component;

component TWOS_COMPLIMENT
port
( input_a : in std_logic_vector (7 downto 0);
  clk : in std_logic;
  twos_complimented_bit : out std_logic_vector (7 downto 0)
);
end component;

signal twos_compliment_out : std_logic_vector (7 downto 0);

component PARTIAL_AND_OPERATION
port
( input_a : in std_logic_vector (7 downto 0);
 partial_and : out std_logic_vector (36 downto 0)
);
end component;
signal partial_and_out : std_logic_vector (36 downto 0);

component A_SQUARE
port

( input_and: in std_logic_vector (36 downto 0);
 A_SQUARE_1 :out std_logic_vector (15 downto 0)
);
end component;
signal square_and_out : std_logic_vector (36 downto 0);
signal a_squared_minus_1_register : std_logic_vector (15 downto 0);
signal a_register_bar : std_logic_vector (15 downto 0);
component MINUS_ONE is
port(

  input_a : in std_logic_vector (15 downto 0);
  a_sq_1 : out std_logic_vector (15 downto 0)
);
end component;
```

```
begin

TWOS_COMPONENT_U1 : TWOS_COMPLIMENT port map (input_a(7 downto
0),clk,twos_compliment_out(7 downto 0));
PARTIAL_AND_OPERATION_U1 : PARTIAL_AND_OPERATION port map
(twos_compliment_out(7 downto 0),partial_and_out(36 downto 0));
WALLACE_TREE_U1 : A_SQUARE port map (partial_and_out(36 downto
0),square_and_out(15 downto 0));
MINUS_ONE_U1 : MINUS_ONE port map (square_and_out(15 downto 0),
a_squared_minus_1_register(15 downto 0));
--- load to register
OUTPUT_REGISTER_U0 : D_FLIP_FLOP port map (
a_squared_minus_1_register(0),clk,a_squared_minus_1(0),a_register_bar(0));
OUTPUT_REGISTER_U1 : D_FLIP_FLOP port map (
a_squared_minus_1_register(1),clk,a_squared_minus_1(1),a_register_bar(1));
OUTPUT_REGISTER_U2 : D_FLIP_FLOP port map (
a_squared_minus_1_register(2),clk,a_squared_minus_1(2),a_register_bar(2));
OUTPUT_REGISTER_U3 : D_FLIP_FLOP port map (
a_squared_minus_1_register(3),clk,a_squared_minus_1(3),a_register_bar(3));
OUTPUT_REGISTER_U4 : D_FLIP_FLOP port map (
a_squared_minus_1_register(4),clk,a_squared_minus_1(4),a_register_bar(4));
OUTPUT_REGISTER_U5 : D_FLIP_FLOP port map (
a_squared_minus_1_register(5),clk,a_squared_minus_1(5),a_register_bar(5));
OUTPUT_REGISTER_U6 : D_FLIP_FLOP port map (
a_squared_minus_1_register(6),clk,a_squared_minus_1(6),a_register_bar(6));
OUTPUT_REGISTER_U7 : D_FLIP_FLOP port map (
a_squared_minus_1_register(7),clk,a_squared_minus_1(7),a_register_bar(7));
OUTPUT_REGISTER_U8 : D_FLIP_FLOP port map (
a_squared_minus_1_register(8),clk,a_squared_minus_1(8),a_register_bar(8));
OUTPUT_REGISTER_U9 : D_FLIP_FLOP port map (
a_squared_minus_1_register(9),clk,a_squared_minus_1(9),a_register_bar(9));
OUTPUT_REGISTER_U10 : D_FLIP_FLOP port map (
a_squared_minus_1_register(10),clk,a_squared_minus_1(10),a_register_bar(10));
OUTPUT_REGISTER_U11 : D_FLIP_FLOP port map (
a_squared_minus_1_register(11),clk,a_squared_minus_1(11),a_register_bar(11));
OUTPUT_REGISTER_U12 : D_FLIP_FLOP port map (
a_squared_minus_1_register(12),clk,a_squared_minus_1(12),a_register_bar(12));
OUTPUT_REGISTER_U13 : D_FLIP_FLOP port map (
a_squared_minus_1_register(13),clk,a_squared_minus_1(13),a_register_bar(13));
OUTPUT_REGISTER_U14 : D_FLIP_FLOP port map (
a_squared_minus_1_register(14),clk,a_squared_minus_1(14),a_register_bar(14));
OUTPUT_REGISTER_U15 : D_FLIP_FLOP port map (
a_squared_minus_1_register(15),clk,a_squared_minus_1(15),a_register_bar(15));



end A_SQUARE_MINUS_1_ARCHITECTURE;
```

ARITHMATICAL LOGICAL UNIT

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;

entity ARITHMATIC_LOGICAL_UNIT is

port(

INPUT_A : in std_logic_vector (7 downto 0);

CLOCK  : in std_logic ;

LOAD:    in std_logic ;

CLEAR:      in std_logic;

OUTPUT:   out std_logic_vector (15 downto 0);

END_FLAG:    out std_logic

);

end ARITHMATIC_LOGICAL_UNIT;
```

```
architecture ARITHMATIC_LOCICAL_UNIT_ARCHITECTURE of ARITHMATIC_LOGICAL_UNIT
is


component A_SQUARE_MINUS_1

port

( input_a : in std_logic_vector (7 downto 0);

  clk : in std_logic;

  a_squared_minus_1 : out std_logic_vector (15 downto 0)

);

end component;

signal  output_results : std_logic_vector (15 downto 0);

begin


A_SQUARE_MINUS_1_U1 : A_SQUARE_MINUS_1 port map
(INPUT_A,CLOCK,output_results(15 downto 0));


    process(CLOCK, CLEAR, LOAD)

    begin
```

```
        if CLEAR = '1' then
        OUTPUT <= "0000000000000000";
    END_FLAG<='0';
        else

                    if LOAD'event and LOAD='0' then
            OUTPUT<=output_results;
            END_FLAG <='1';

            end if;
        end if;
   end process;
end ARITHMATIC_LOCICAL_UNIT_ARCHITECTURE;
```

ALU TEST BENCH

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.all;


ENTITY ALU_TEST_BENCH IS
END ALU_TEST_BENCH;

ARCHITECTURE behavior OF ALU_TEST_BENCH IS

component ARITHMATIC_LOGICAL_UNIT
port(
INPUT_A : in std_logic_vector (7 downto 0);
CLOCK  : in std_logic ;
LOAD:    in std_logic ;
```

```
CLEAR:      in std_logic;
OUTPUT:   out std_logic_vector (15 downto 0);
END_FLAG:    out std_logic
);
end component;



  signal TB_number   : std_logic_vector (7 downto 0) := (others => '1');
  signal TB_output : std_logic_vector (15 downto 0)  := (others => '0');
  signal TB_clock   : std_logic := '0';
  signal TB_clear   : std_logic := '0';
   signal TB_load   : std_logic := '0';

  signal TB_flag : std_logic := '0';

  constant load_duration : time := 100 ns;

 signal i:integer;
BEGIN

    uut: ARITHMATIC_LOGICAL_UNIT PORT MAP (
          INPUT_A  => TB_number,
          CLOCK=> TB_clock,
          CLEAR => TB_clear,
          LOAD => TB_load,
          OUTPUT => TB_output,
      END_FLAG =>TB_flag
        );

   process
   begin
```

```
  for i in 0 to 255 loop
 tb_load <= '1';
tb_clock <= '1';
wait for load_duration/2;
tb_clock <= '0';
tb_load <= '0';

 wait for load_duration/2;
TB_number <= TB_number - x"1";

  end loop;
   end process;
END;
```

References

[1] NOTES, C. 6. (n.d.). *CONCORDIA UNIVERSITY.* Retrieved from COEN 6501: https://users.encs.concordia.ca/~asim/COEN_6501/Lecture_Notes/Lecture_Notes.htm

[2] OR GATE(n.d.). Retrieved from https://th.wikipedia.org/wiki/%E0%B9%84%E0%B8%9F%E0%B8%A5%E0%B9%8C:Logic-gate-or-us.png

[3] FULL ADDER, H. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Adder_(electronics)

[4] FLOP, D. F. (n.d.). Retrieved from http://worldclassprogramme.com/D-FlipFlop.php

[5] NAND GATE, N. (n.d.). Retrieved from https://simple.wikipedia.org/wiki/NAND_gate

[6] THREE INPUT NANDGATE, T. I. (n.d.). Retrieved from https://i.stack.imgur.com/Nh942.png

[7] XOR GATE, X. (n.d.). Retrieved from https://de.wikipedia.org/wiki/Datei:Logic-gate-xor-us.png

[8] *AND GATE WIKIPEDIA.* (n.d.). Retrieved from https://de.wikipedia.org/wiki/Datei:Logic-gate-and-us.png

[9] HALF ADDER, F. (n.d.). Retrieved from https://th.wikipedia.org/wiki/%E0%B8%A7%E0%B8%87%E0%B8%88%E0%B8%A3%E0%B8%9A%E0%B8%A7%E0%B8%81