4X2 ADDITIVE MULTIPLY MODULE

Submitted by: Suhas Niranjan Yelluru [31371564]

PROF. TIMOTHY STEELE
NJIT
ECE 689

Table of Contents

Table of Figures

STRUCTURE OF PROJECT:

1. Designing a single 4x2 AMM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity amm_module is
    Port ( clk : in STD_LOGIC;
           a : in STD_LOGIC_VECTOR (3 downto 0);
           b : in STD_LOGIC_VECTOR (1 downto 0);
           c : in STD_LOGIC_VECTOR (3 downto 0);
           d : in STD_LOGIC_VECTOR (1 downto 0);
           p : out STD_LOGIC_VECTOR (5 downto 0));
end amm_module;

architecture Behavioral of amm_module is
--signal clk : STD_LOGIC;
signal and00 : STD_LOGIC;
signal and10 : STD_LOGIC;
signal and20 : STD_LOGIC;
signal and30 : STD_LOGIC;
signal and01 : STD_LOGIC;
signal and11 : STD_LOGIC;
signal and21 : STD_LOGIC;
signal and31 : STD_LOGIC;

signal fa1c : STD_LOGIC;
signal fa2s : STD_LOGIC;
signal fa2c : STD_LOGIC;
signal fa3s : STD_LOGIC;
signal fa3c : STD_LOGIC;
signal fa4s : STD_LOGIC;
signal fa4c : STD_LOGIC;
signal fa5c : STD_LOGIC;
signal fa6c : STD_LOGIC;
signal fa7c : STD_LOGIC;

begin
    process(clk)
        begin
        if rising_edge(clk) then

            --AND Products
            and00 <= a(0) AND b(0);
            and10 <= a(1) AND b(0);
            and20 <= a(2) AND b(0);
            and30 <= a(3) AND b(0);
            and01 <= a(0) AND b(1);
            and11 <= a(1) AND b(1);
            and21 <= a(2) AND b(1);
            and31 <= a(3) AND b(1);

            --FULLADDER-1
            fa1c <= (c(0) AND and00)OR(d(0) AND and00)OR(c(1) AND d(0));
            p(0) <= c(0) XOR and00 XOR d(0);

            --FULLADDER-2
            fa2s <= and10 XOR and01 XOR c(1);
            fa2c <= (and10 AND and01) OR (and10 AND c(1)) OR (c(1) AND and01);

            --FULLADDER-3
            fa3s <= and20 XOR and11 XOR c(2);
            fa3c <= (and20 AND and11) OR (and20 AND c(2)) OR (c(2) AND and11);
```

```
            --FULLADDER-4
            fa4s <= and30 XOR and21 XOR c(3);
            fa4c <= (and30 AND and21) OR (and30 AND c(3)) OR (c(3) AND and21);

            --FULLADDER-5
            p(1) <= fa1c XOR d(1) XOR fa2s;
            fa5c <= (fa1c AND d(1)) OR (fa1c AND fa2s) OR (d(1) AND fa2s);

            --FULLADDER-6
            p(2) <= fa5c XOR fa2c XOR fa3s;
            fa6c <= (fa5c AND fa2c) OR (fa5c AND fa3s) OR (fa3s AND fa2c);

            --FULLADDER-7
            p(3) <= fa3c XOR fa4s XOR fa6c;
            fa7c <= (fa3c AND fa4s) OR (fa3c AND fa6c) OR (fa4s AND fa6c);

            --FULLADDER-8
            p(4) <= and31 XOR fa4c XOR fa7c;
            p(5) <= (fa4c AND fa7c) OR (fa4c AND and31) OR (fa7c AND and31);

            end if;
            end process;
end Behavioral;
```

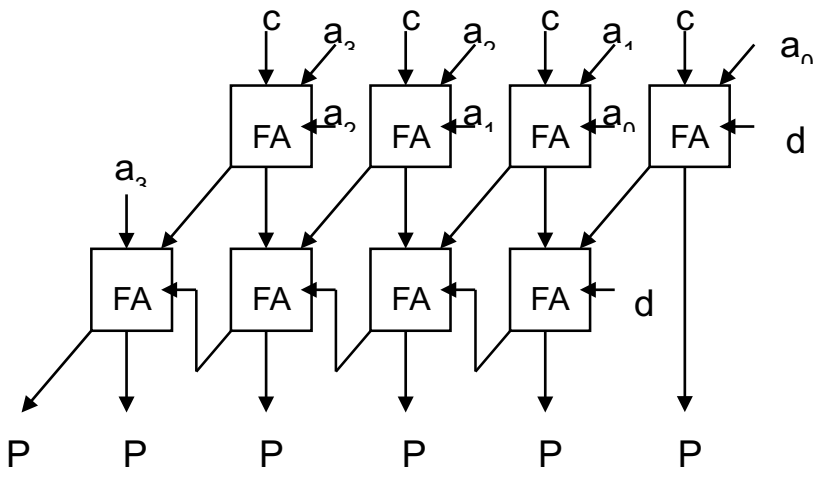

Figure 1: 4x2 AMM

In order to make the operation pipelined, we have to introduce latches for the outputs from AMM1 for the 3rd and 2nd bit, along with a latch for AMM2 for the 5th to 2nd bits.

This is because, these inputs need only one clock to be generated, however, the remaining inputs to the future AMMs would not have been generated.

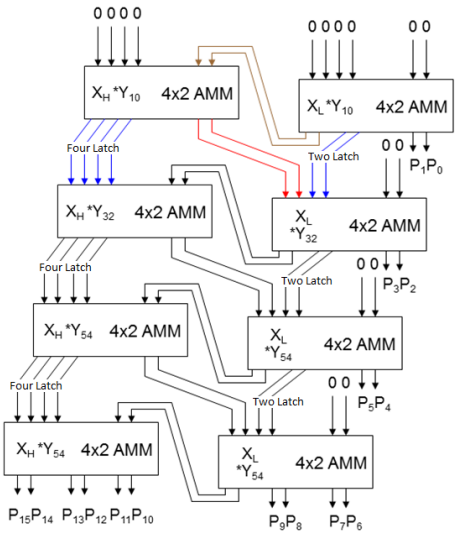

Figure 2: Using 4x2 AMM's for 8x8 multiplication

2. Two Latch: for all 3 downto 2 outputs on X_LxY_{10}, X_LxY_{32}, X_LxY_{54}, X_LxY_{76}

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

entity twolatch is
    Port ( clk : in std_logic;
           p23 : in std_logic_vector(1 downto 0) := "00";
           o23 : out std_logic_vector(1 downto 0) := "00"
    );
end twolatch;

architecture Behavioral of twolatch is
   type t_mem is array(natural range <>) of std_logic_vector(1 downto 0);
    signal mem : t_mem(0 to 1) := (others => "00");
begin
    process(clk)
    begin
        if rising_edge(Clk) then
            o23 <= mem(1);
            --mem(1) <= mem(0);
            mem(1) <= p23;
        end if;
    end process;
end Behavioral;
```

3.Four Latch: for all 5 downto 2 outputs on X_HxY_{10}, X_HxY_{32}, X_HxY_{54}, X_HxY_{76}

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

entity fourlatch is
    Port ( clk : in STD_LOGIC;
           p23 : in STD_LOGIC_VECTOR(3 downto 0):= "0000";
           o23 : out STD_LOGIC_VECTOR(3 downto 0):= "0000");
end fourlatch;

architecture Behavioral of fourlatch is
   type t_mem is array(natural range <>) of std_logic_vector(3 downto 0);
    signal mem : t_mem(0 to 1) := (others => "0000");
```

```
begin
 process(clk)
   begin
       if rising_edge(clk) then
           o23 <= mem(1);
           --mem(1) <= mem(0);
           mem(1) <= p23;
       end if;
   end process;
end Behavioral;
```

4. Calling all the above components to make an 8x8 multiplication:

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

entity ece689_proj is
   Port ( clk : in STD_LOGIC;
          a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          p : out STD_LOGIC_VECTOR (15 downto 0));
end ece689_proj;

architecture Behavioral of ece689_proj is

    signal a1 : STD_LOGIC_VECTOR (3 downto 0);
    signal a2 : STD_LOGIC_VECTOR (3 downto 0);

    signal b1 : STD_LOGIC_VECTOR (1 downto 0);
    signal b2 : STD_LOGIC_VECTOR (1 downto 0);
    signal b3 : STD_LOGIC_VECTOR (1 downto 0);
    signal b4 : STD_LOGIC_VECTOR (1 downto 0);

    signal c : STD_LOGIC_VECTOR (3 downto 0) := "0000";
    signal d : STD_LOGIC_VECTOR (1 downto 0) := "00";

    --two latch outputs
    signal q1 : STD_LOGIC_VECTOR (1 downto 0);
    signal q2 : STD_LOGIC_VECTOR (1 downto 0);
    signal q3 : STD_LOGIC_VECTOR (1 downto 0);

    --four latch outputs
    signal r1 : STD_LOGIC_VECTOR (3 downto 0);
    signal r2 : STD_LOGIC_VECTOR (3 downto 0);
    signal r3 : STD_LOGIC_VECTOR (3 downto 0);

    type s is array(7 downto 0) of std_logic_vector(5 downto 0);
    signal x : s := (others => "000000");

    component amm_module is
        Port(  clk : in STD_LOGIC;
               a : in STD_LOGIC_VECTOR (3 downto 0);
               b : in STD_LOGIC_VECTOR (1 downto 0);
               c : in STD_LOGIC_VECTOR (3 downto 0);
               d : in STD_LOGIC_VECTOR (1 downto 0);
               p : out STD_LOGIC_VECTOR (5 downto 0));
    end component;

      component twolatch is
        Port ( Clk : in std_logic;
               p23 : in std_logic_vector(1 downto 0) := "00";
               o23 : out std_logic_vector(1 downto 0) := "00"
        );
```

```
        end component;

        component fourlatch is
           Port ( Clk : in STD_LOGIC;
                  p23 : in STD_LOGIC_VECTOR(3 downto 0):= "0000";
                  o23 : out STD_LOGIC_VECTOR(3 downto 0):= "0000");
           end component;
begin

    a1 <= a(3 downto 0);
    a2 <= a(7 downto 4);

    b1 <= b(1 downto 0);
    b2 <= b(3 downto 2);
    b3 <= b(5 downto 4);
    b4 <= b(7 downto 6);

--XL*Y10 (4x2 AMM)
MODULE1: amm_module port map (clk,a1,b1,c,d,x(0));

--XH*Y10 (4x2 AMM)
MODULE2: amm_module port map (clk,a2,b1,c,x(0)(5 downto 4),x(1));
mod1: twolatch port map(clk, x(0)(3 downto 2), q1);

--XL*Y32 (4x2 AMM)
MODULE3: amm_module port map (clk,a1,b2,(x(1)(1 downto 0) & q1),d,x(2));
mod2: fourlatch port map(clk, x(1)(5 downto 2), r1);

--XH*Y32 (4x2 AMM)
MODULE4: amm_module port map(clk,a2,b2,r1,x(2)(5 downto 4), x(3));
mod3: twolatch port map(clk, x(2)(3 downto 2), q2);

--XL*Y54 (4x2 AMM)
MODULE5: amm_module port map(clk,a1,b3,(x(3)(1 downto 0) & q2),d,x(4));
mod4: fourlatch port map(clk,x(3)(5 downto 2), r2);

--XH*Y54 (4x2 AMM)
MODULE6: amm_module port map(clk,a2,b3,r2,x(4)(5 downto 4), x(5));
mod5: twolatch port map(clk, x(4)(3 downto 2), q3);

--XL*Y76 (4x2 AMM)
MODULE7: amm_module port map(clk,a1,b4,(x(5)(1 downto 0) & q3),d,x(6));
mod6: fourlatch port map(clk, x(5)(5 downto 2), r3);

--XH*Y76 (4x2 AMM)
MODULE8: amm_module port map(clk,a2,b4,r3,x(6)(5 downto 4), x(7));

p <= (x(7) & x(6)(3 downto 0) & x(4)(1 downto 0) & x(2)(1 downto 0) & x(0)(1 downto
0));

end Behavioral;
```

TESTBENCH

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity test is

end test;

architecture Behavioral of test is
    component ece689_proj is
            Port ( clk : in STD_LOGIC;
             a : in STD_LOGIC_VECTOR (7 downto 0);
             b : in STD_LOGIC_VECTOR (7 downto 0);
             p : out STD_LOGIC_VECTOR (15 downto 0));
     end component;


     signal a : STD_LOGIC_VECTOR (7 downto 0);
     signal b : STD_LOGIC_VECTOR (7 downto 0);
     signal clk : STD_LOGIC := '0' ;
     constant period : time:= 1ns;
     signal p : STD_LOGIC_VECTOR(15 downto 0);

begin

uut: ece689_proj port map(clk => clk,a => a, b => b,p => p);

-- GENERATE CLOCK OF PERIOD = 1ns
clk_process: process
                begin
                clk <= '0';
                wait for period/2;
                clk <= '1';
                wait for period/2;

             end process;

-- INCREMENT A and B FROM
test_process: process
                begin
                     a <= "00000000";

                for i in 0 to 255 loop
                     wait for 1ns;

                     a <= std_logic_vector(unsigned(a) + 1);
                      b<= "00000000";
                    for j in 1 to 9 loop

                        wait for 1ns;
                        b <= std_logic_vector(unsigned(b) + 1);
                     end loop;
                end loop;

                     wait;

                end process;
end Behavioral;
```

Results and Discussion:

1. Elaborated Design:

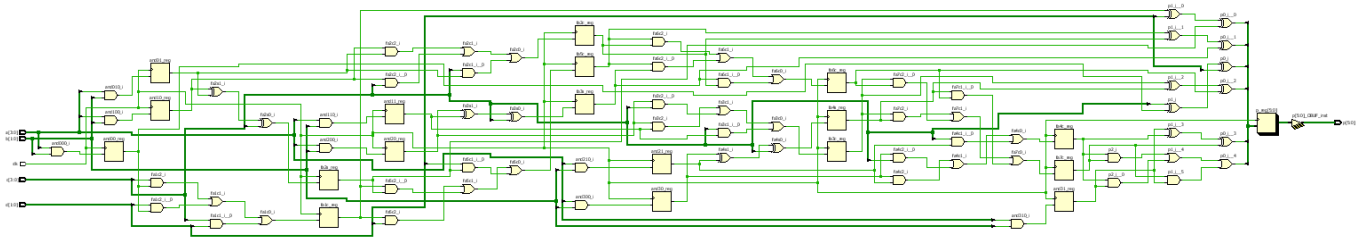

Figure 3: AMM

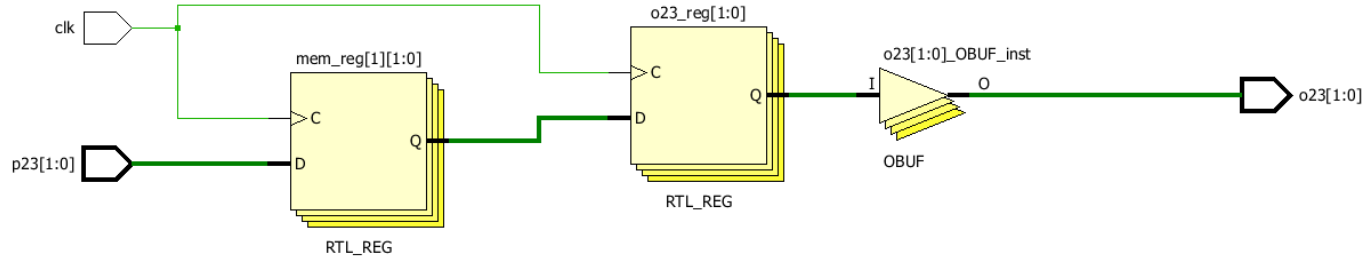

Figure 4: Two Latch

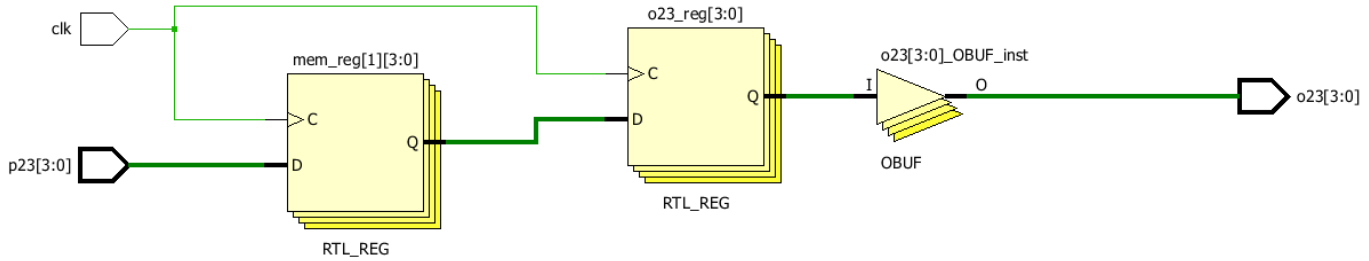

Figure 5: Four Latch

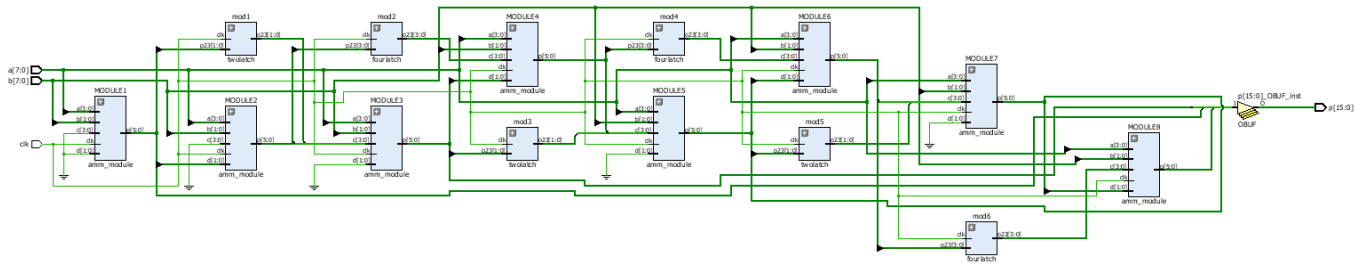

Figure 6: 8x8 Multiplication using 8, 4x2 AMM's

2. Project Summary:

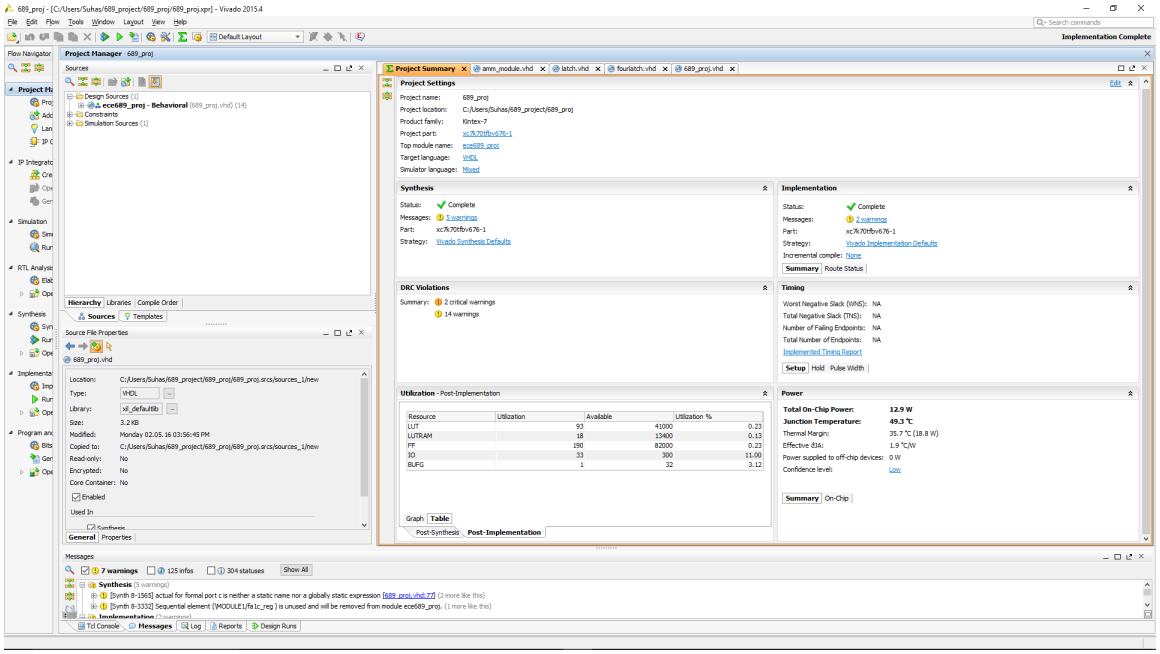

Figure 7: Project Summary

Resource	Utilization	Available	Utilization %
LUT	93	41000	0.23
LUTRAM	18	13400	0.13
FF	190	82000	0.23
IO	33	300	11.00
BUFG	1	32	3.12

Figure 8: Resource Utilization

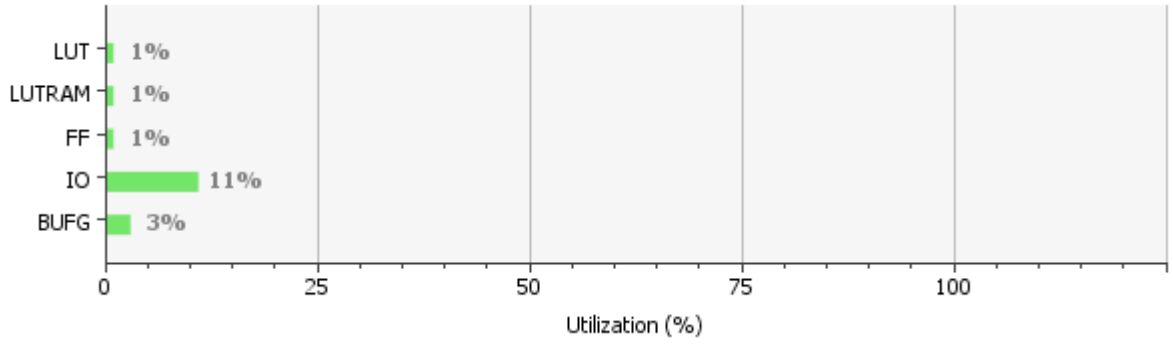

Figure 9: Number of LUT's, IO's FF's etc used

3. Outputs:

Example 1.

A = 0C, B = 11;

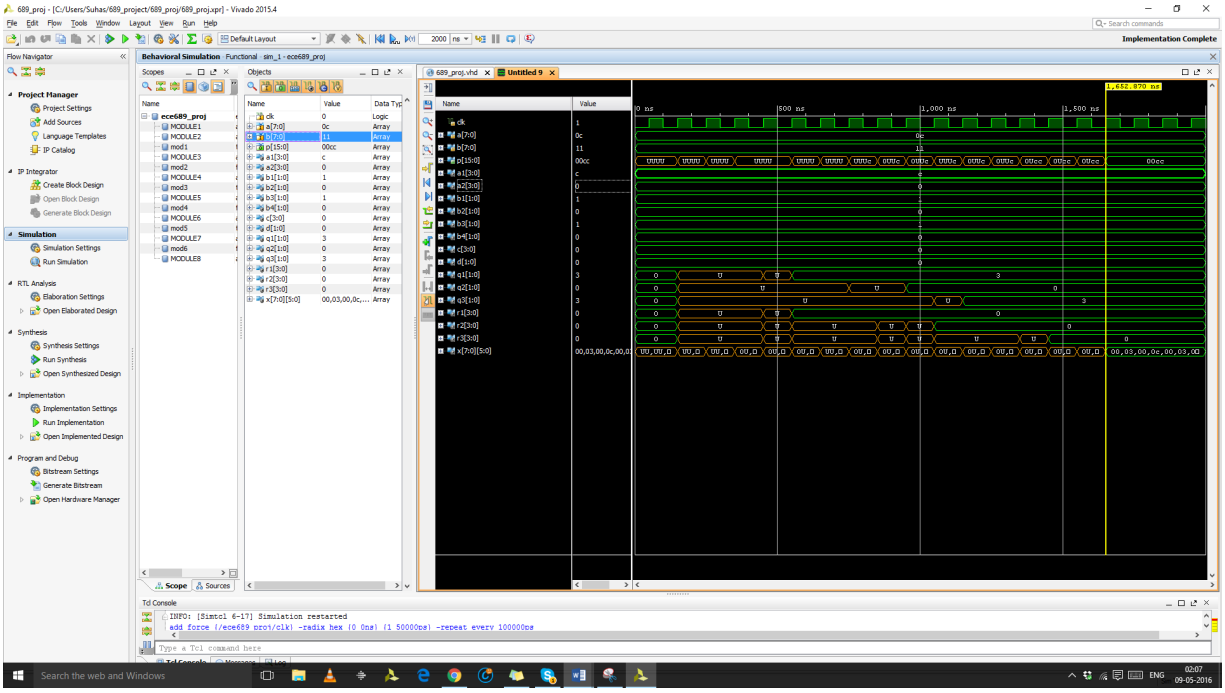

Figure 10: Example output#1

Example 2.

A = fa, B = 03;

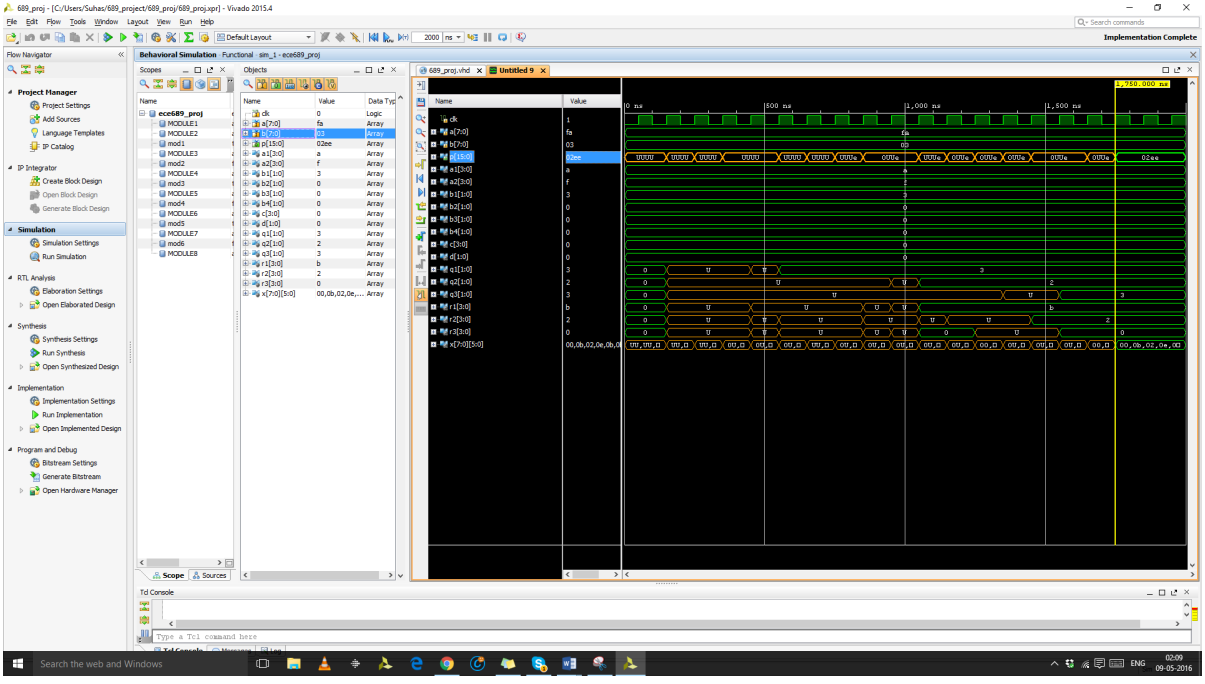

Figure 11: Example output#2

POST BEHAVIORAL SYNTHESIS:

1. AMM

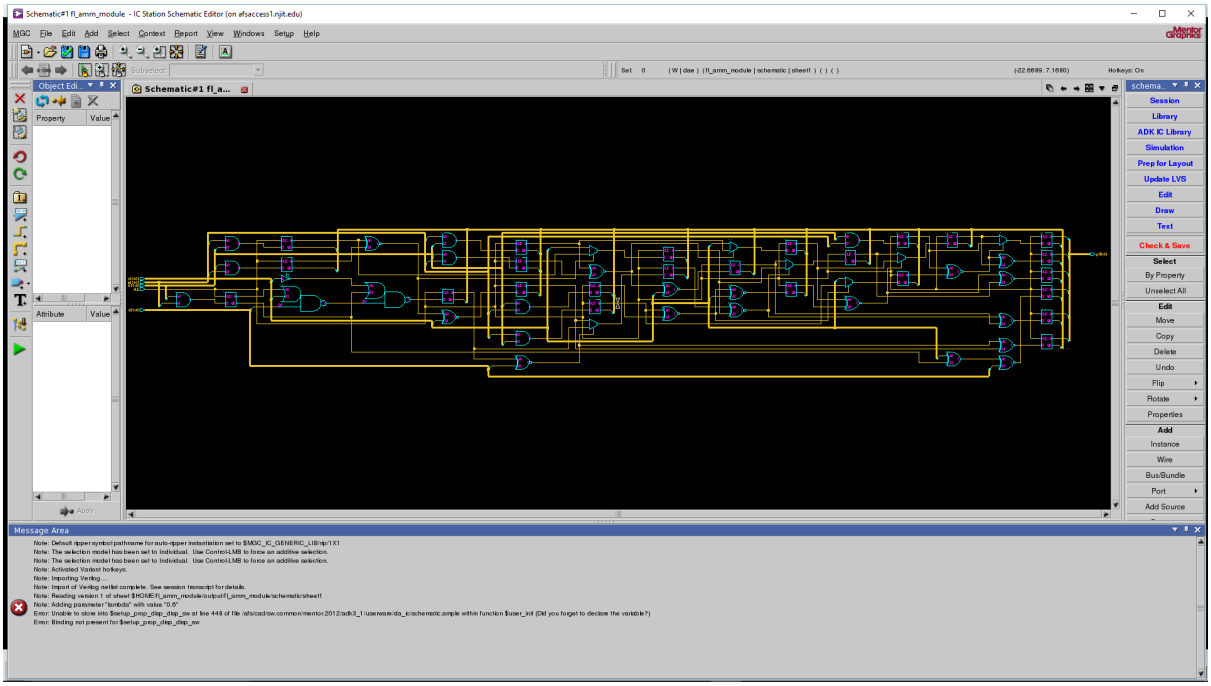

Figure 12: Schematic for AMM using Mentor Graphics toolkit

2. 8x8 multiplication

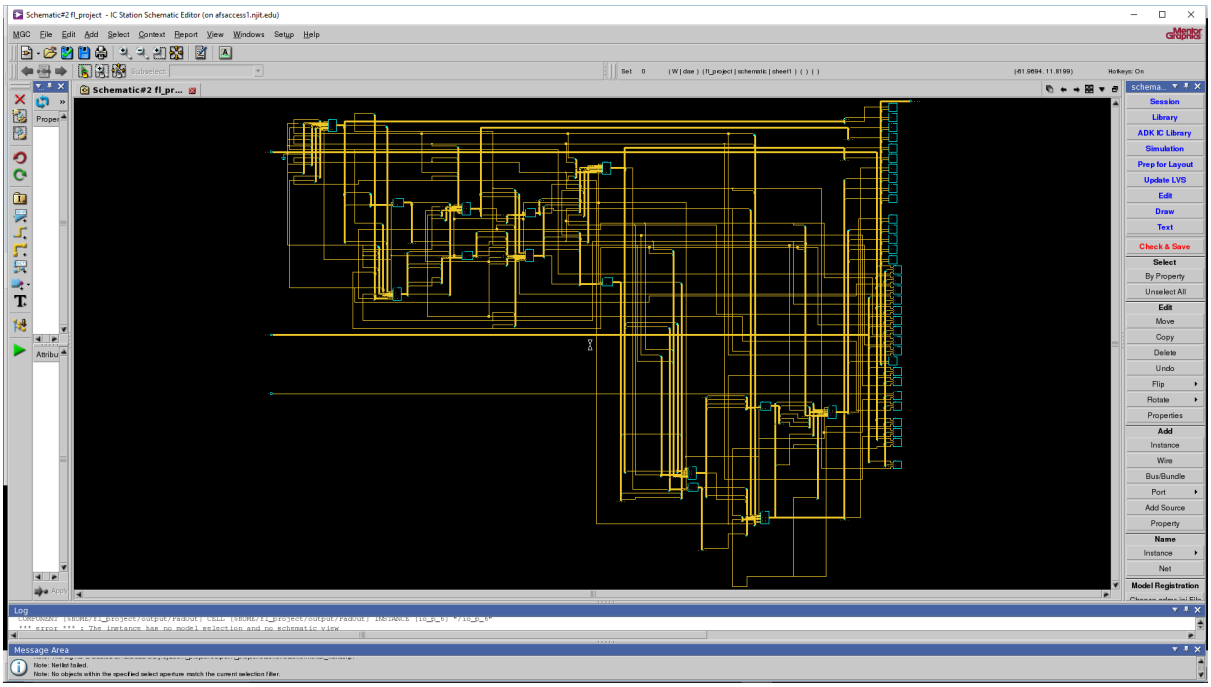

Figure 13: Schematic for 8x8 Pipelined Multiplier using 4x2 AMM's

3. Pad Frame

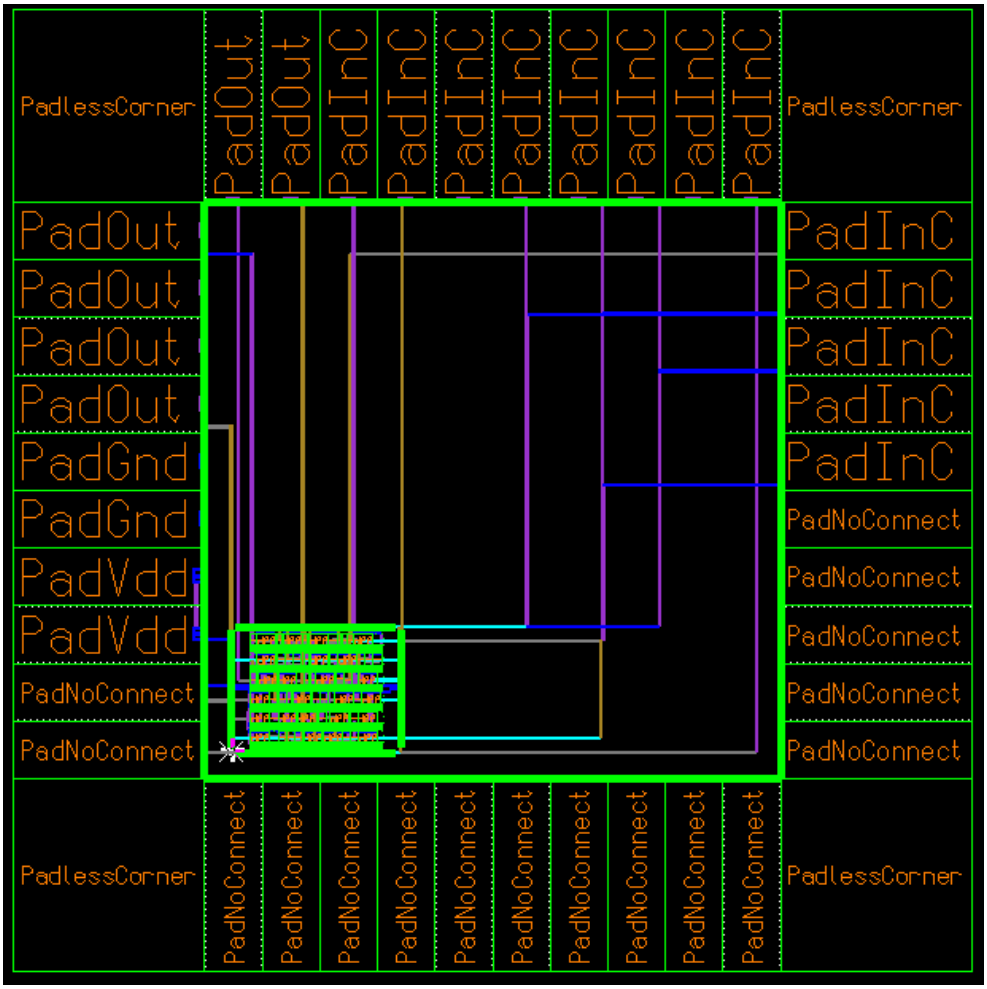

Figure 14: Pad frame for AMM

4. Layout for Two Latch

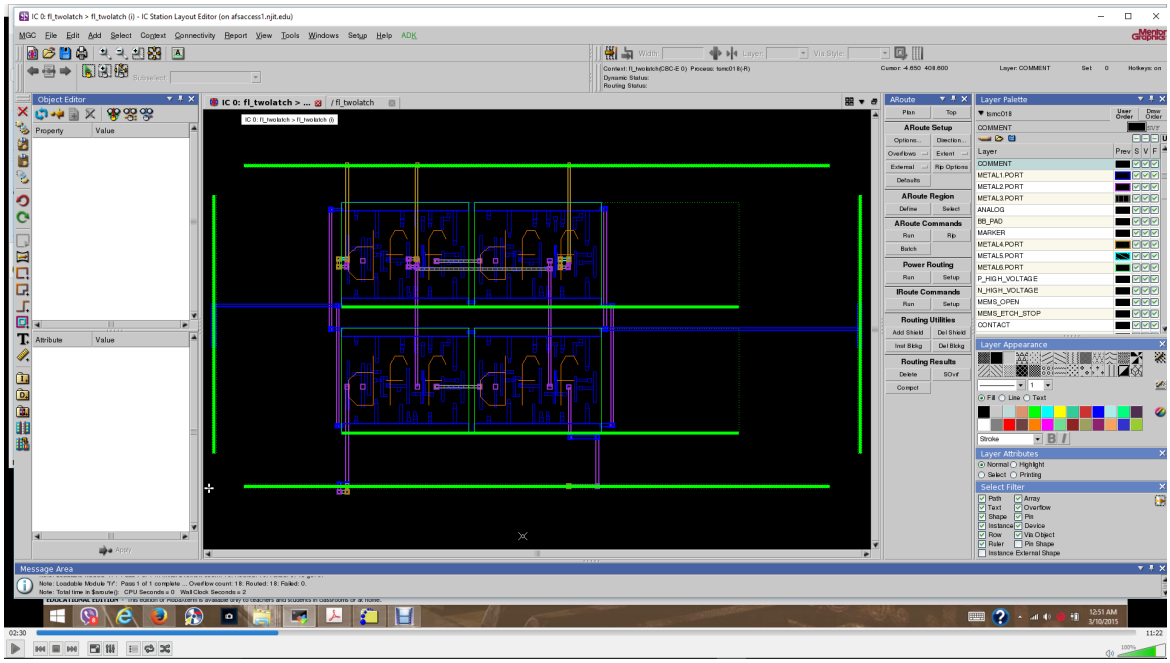

Figure 15: Layout of Two Latch

5. Schematic for Four Latch

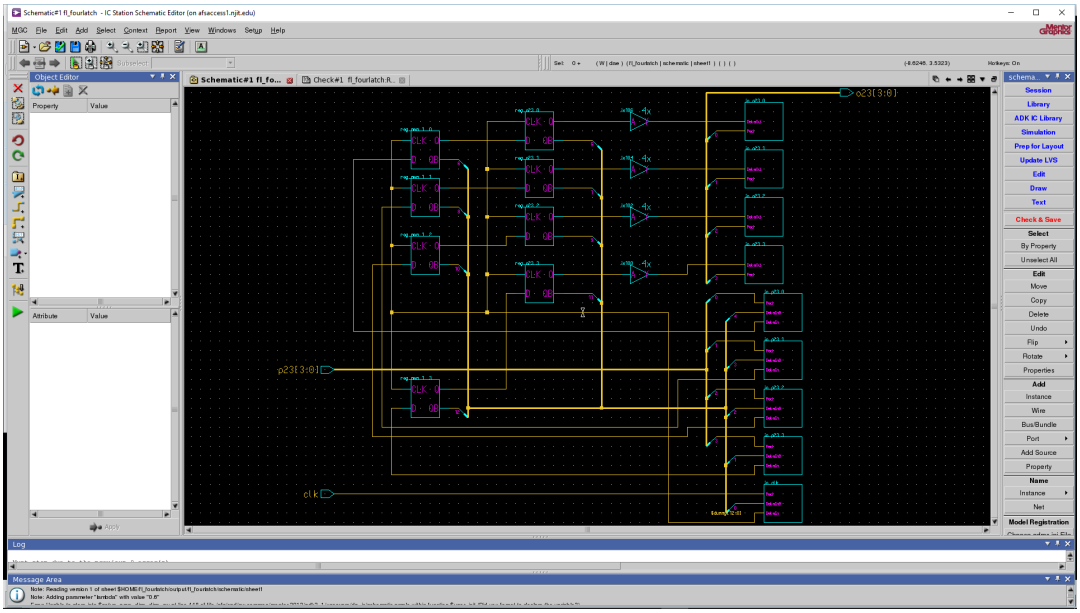

Figure 16: Schematic of Four Latch

6. Layout for Four Latch

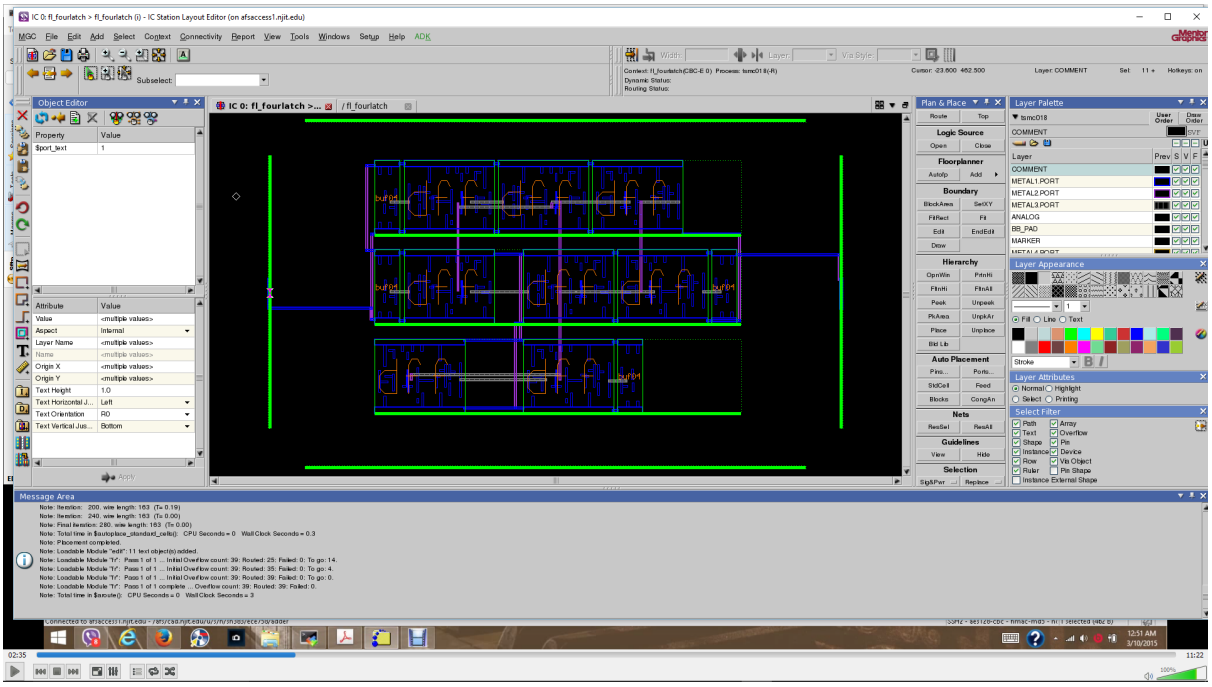

Figure 17: Layout of Four Latch