ECE 689

Computer Arithmetic Algorithms

N Bit AMM

Final Project

By
Bhaumik Mistry
NJID: 31363822
Email: bpm8@njit.edu

Abstract : *Task was to build and simulate the 4x2 AMM module , 8x8 AMM module with latches and N x N AMM using VHDL as the description language on the Altera design environment as the simulation tool and simulate its functionality. The task is successfully completed using varieties of modules and mathematical equations and mainly with 'generate' statements, which can be nested to create two-dimensional instance "arrays".*

1.0 Introduction.

The additive multiply modules (AMM) can receive additional addends and add them to the product of the input multiplicand and multiplier. A 4-by-2 AMM which can perform the arithmetic operation such as P = A x B + C + D. Here A and B are a 4-bit multiplicand and 2-bit multiplier respectively. The product P is of 6 bits since 4 + 2 = 6. C is of 4 bits, and D is of 2 bits.

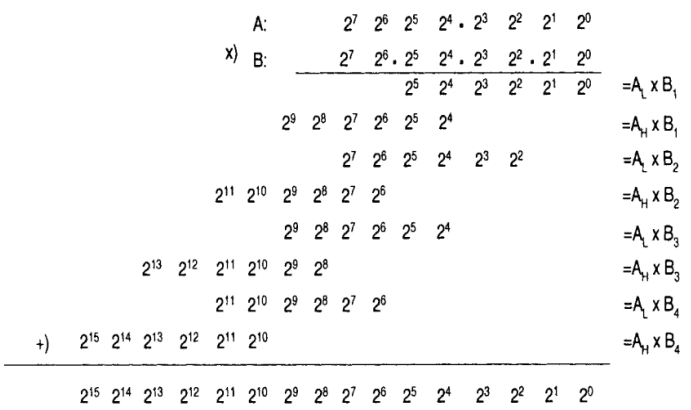

Figure 2 8*8 multiplication via 4by2 AMM

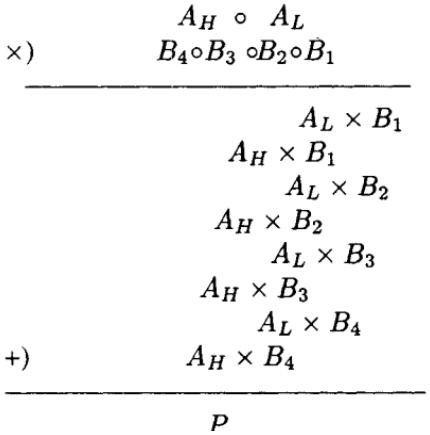

Figure 1 section vise multiplication

The array multiplier implemented by 4-by-2 AMMs is presented in Figure 3. As we can see from Figure 1, the worst case delay for an 4-by-2 AMM is

As we can see from Figure 5.17, the worst case delay for an 4-by-2 AMM is

$$\begin{aligned} \triangle_{AMM(4\times2)} &= 5\triangle_{FA} + \triangle_{AND} \\ &= (10+2)\triangle_g \\ &= 12\triangle_g. \end{aligned}$$

The total delay time of the modular multiplier completing 8-by-8 multiplication is:

$$\triangle_{AMM(8\times8)} = 12 + 18 + 18 + 18 + 12 = 78\triangle_g.$$

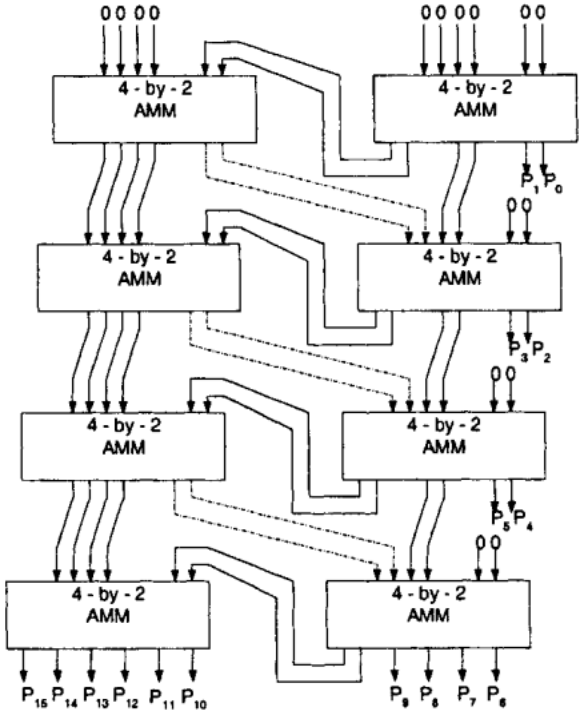

Figure 3 Logical construction of 8*8 multiplication using 4by2 AMM

2.0 Part I Implementation.

Task : Build and simulate the 4x2 AMM module shown in lecture 7 (and in Figure 6) using modular Verilog or VHDL as the description language and either the Altera or Xilinx WebPack design environment as the simulation tool. Then, using multiple instantiations of this module, build the 8x8 multiplier also shown in lecture 7 (and in Figure 3) and simulate its functionality.

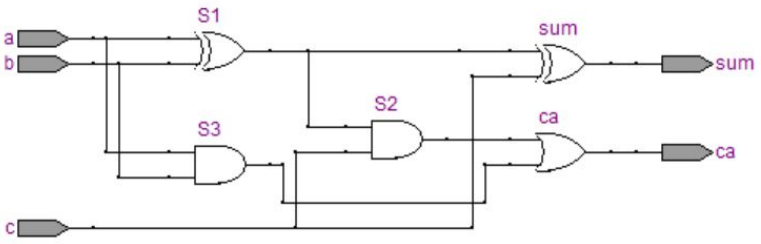

Figure 4 Full Adder RTL block

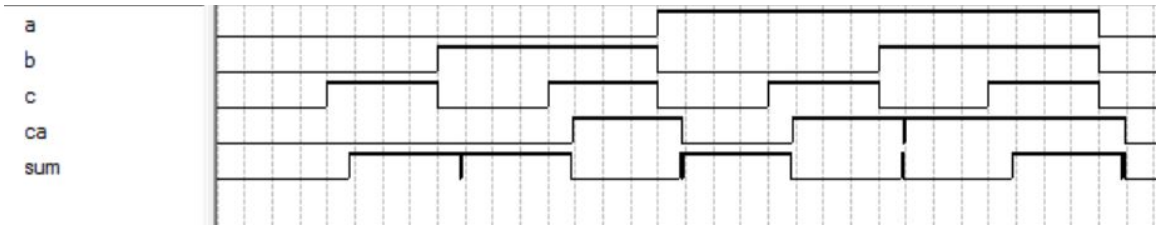

Figure 5 Full Adder simulation waveform

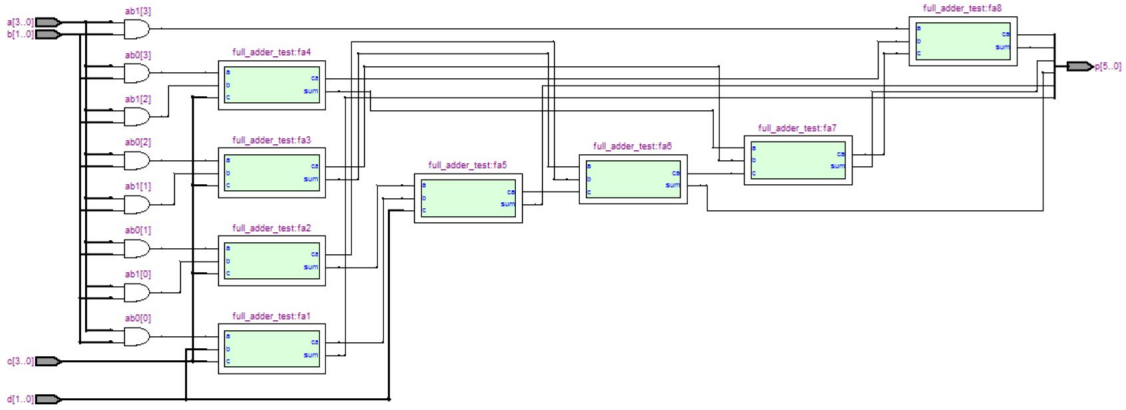

Figure 6 4by2 AMM using full adder RTL block.

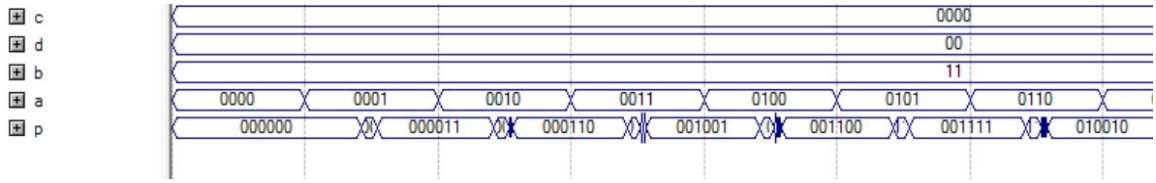

Figure 7 4by2 AMM waveform simulation.

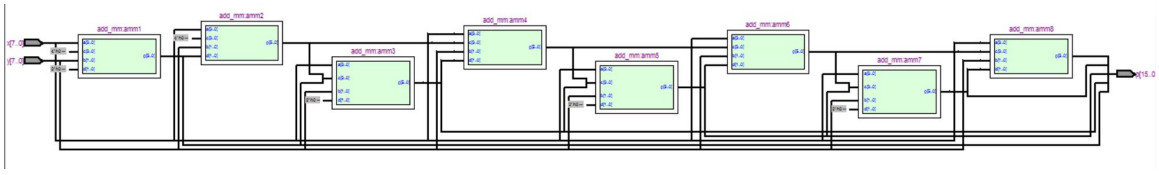

Figure 8 multiplication of 8*8 using 4by2 AMMs

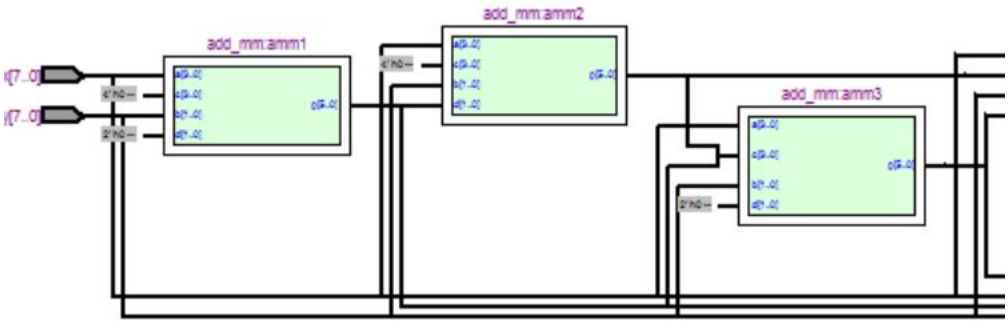

Figure 9 Zoom section I

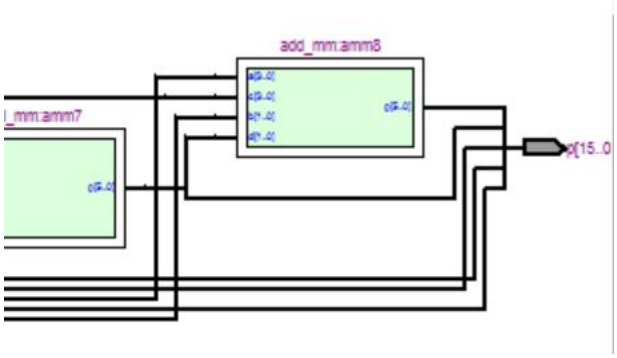

Figure 10 Zoom section II

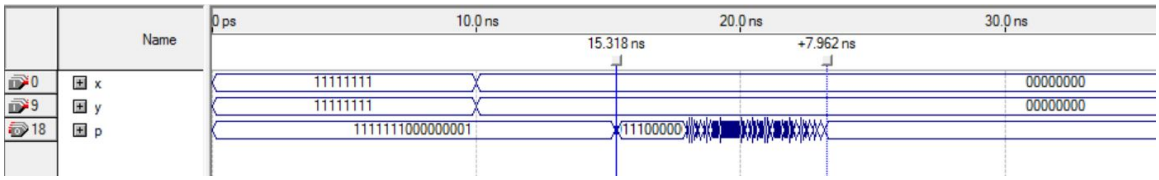

Figure 11 8*8 AMM simulation waveform.

- Above, set of images are presented to understand overall schematic and function simulation of part I. Which is to generate 8*8 AMM using 4by2 AMM.
- Figure 11 shows us the delayed o/p. Time bars are showing the delay in the output by an 8by8 AMM.
- This marks the successful completion of part 1 of the project.

3.0 Part 2 Implementation.

***Task**: Modify the design from part 1 to add the necessary latches for a maximum clock rate (minimum delay between latch stages). The latches should also be modular and are external to the 4x2 AMM. Show the pipelined nature of the latched operation using the simulation tool.*

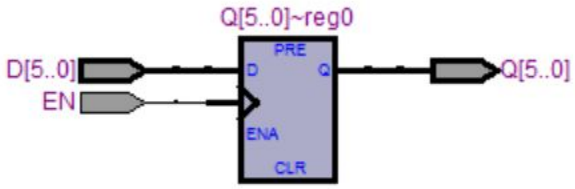

Figure 12 RTL block of latch to hold AMM output.

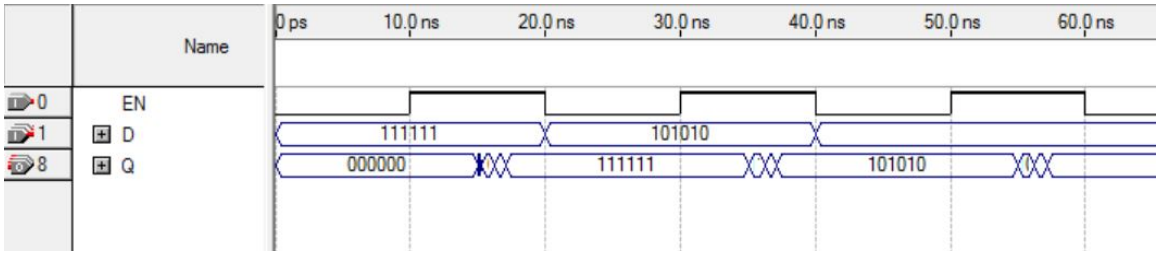

Figure 13 Latch output per cycle

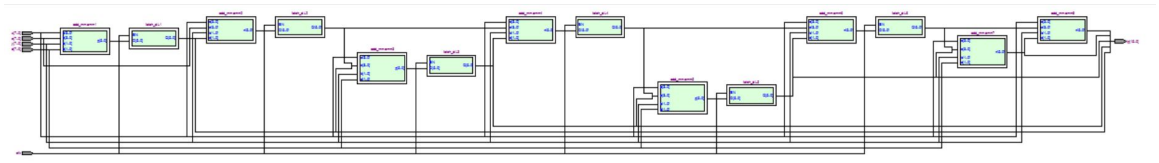

Figure 14 8by8 AMM having a latch after every AMM except last two AMMs.

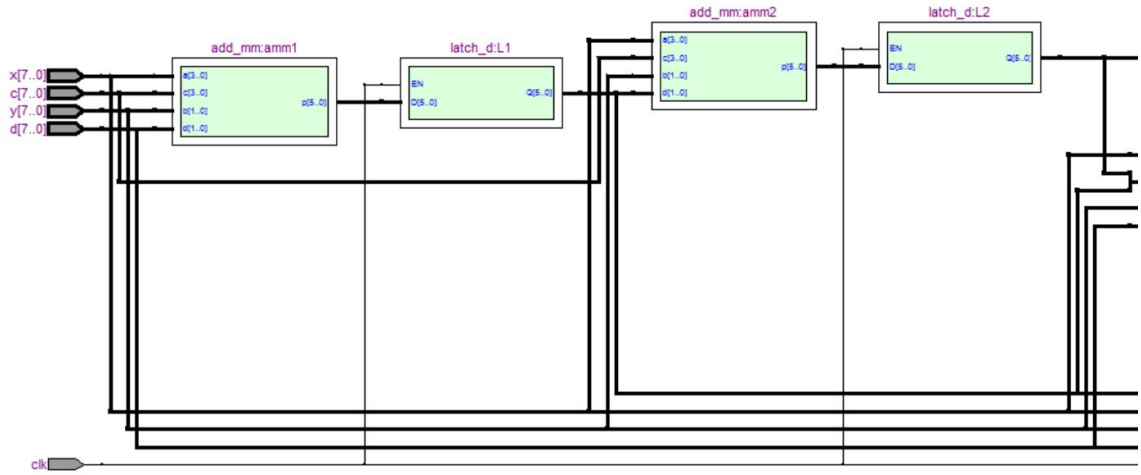

Figure 15 Zoom section of figure_14.

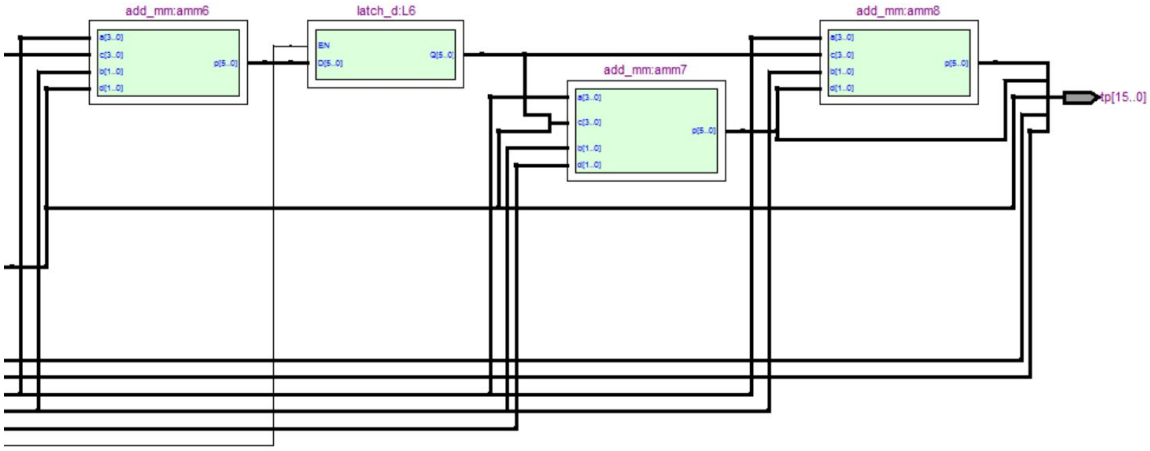

Figure 16 Zoom section of figure 14.

- As we see, Figure 15 and figure 16 gives us the zoomed view of 8by8 AMM using 4by2 AMM and latches attached to it.
- The last two AMMs will not have any latches attached to it.

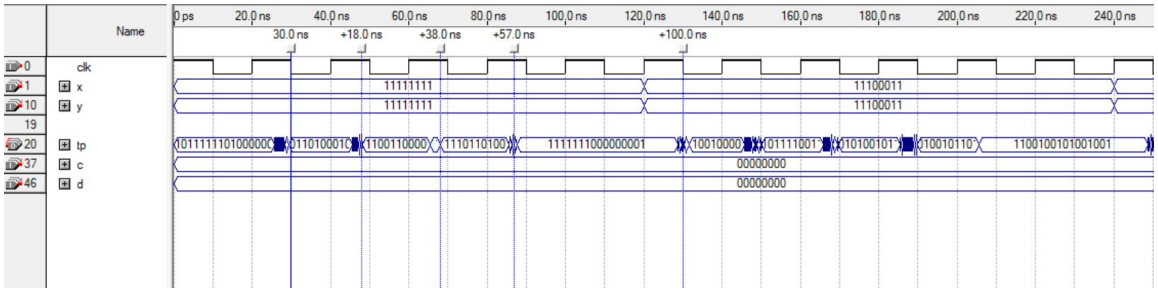

Figure 17 Waveform simulation of 8by8 AMM ith latches attached.

- As shown in figure 17, the pipelined view of the out put of the AMM can be easily determined.
- There are four Latches, which contribute to P (output), hence we can see the signal P changes its value after almost one cycle till the out put is collected at last AMM.
- The change in P values indicates that the latches are holding the output of the AMMs till the end of the current clock cycle.
- This proves the successful completion of part 2.

4.0 Part 3 Implimentation.

Task: *Modify the design from part 2 to allow the user (prior to synthesis – using a generic is allowable) to set the size of the multiplier, i.e. NxN instead of a fixed 8x8. This modified design should instantiate all of the necessary 4x2 AMMs and latches to successfully complete this operation. The user should only have to modify the one constant N and then run synthesis for the design to function.*

CASE I : GENERIC (N := 8)

```
generic(N       : integer :=8); --- Changing the N integer determines the NxN
port(   clk     : std_logic;
        x       : in std_logic_vector(N-1 downto 0);
        y       : in std_logic_vector(N-1 downto 0);
        p       : out std_logic_vector(N+N-1 downto 0)
        );
```

Figure 18 Section of code showing N:=8.

```
Info: Quartus II Fitter was successful. 0 errors, 4 warnings
```

- After multiple iteration, it came to my attention that falling in the trap to deal with one additional signal (AMM to latch) while iterating two for..generate loop, it will create additional use of signals in N bit loop which needs to be updated after every iteration.
- I can to a conclusion to make a different block with one AMM + Latch, which takes x y c d and a clock as input and gives o/p tp.

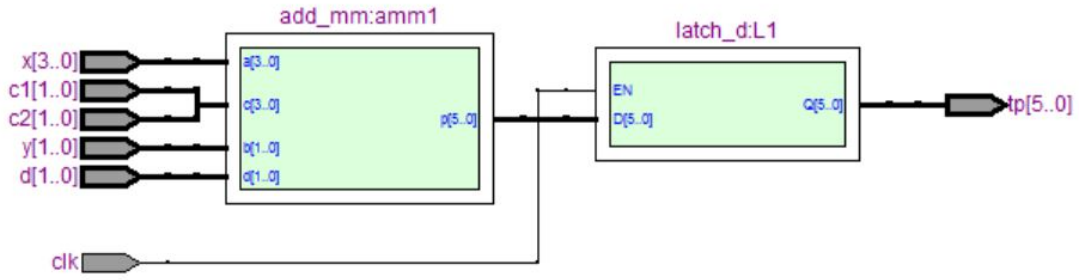

Figure 19 Newly generated RTl block.

- **Each above mentioned block is generated by nested for loops.**
- **The value of N can only be multiple of 4.**
  Math: (N>4 and ((N mod 4)=0))

Quartus II Version	9.1 Build 222 10/21/2009 SJ Web Edition
Revision Name	nbyn_amm
Top-level Entity Name	nbyn_amm
Family	Cyclone II
Met timing requirements	Yes
Total logic elements	145 / 4,608 (3 %)
Total combinational functions	145 / 4,608 (3 %)
Dedicated logic registers	48 / 4,608 (1 %)
Total registers	48
Total pins	33 / 89 (37 %)
Total virtual pins	0
Total memory bits	0 / 119,808 (0 %)
Embedded Multiplier 9-bit elements	0 / 26 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP2C5T144C6
Timing Models	Final

Figure 20 Report after N=8 synthesis

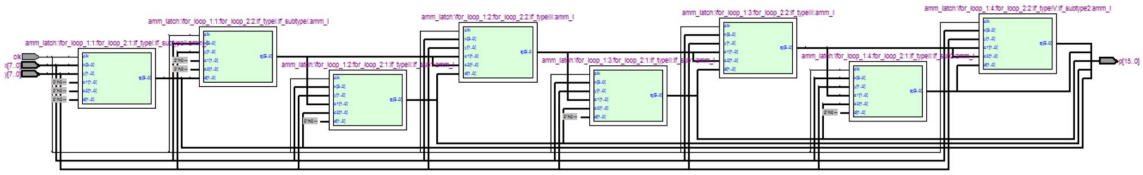

Figure 21 Automated Generation of 8 block by the algorithm after synthesis.

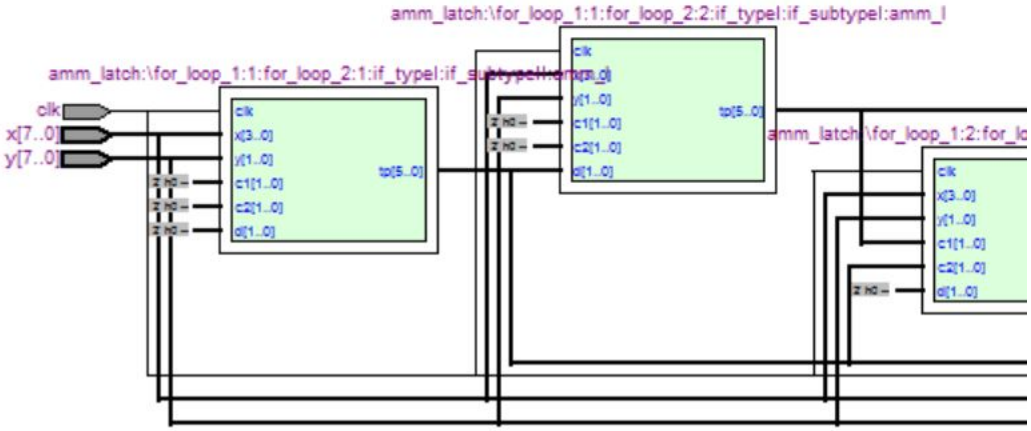

Figure 22 Zoom section of figure 21

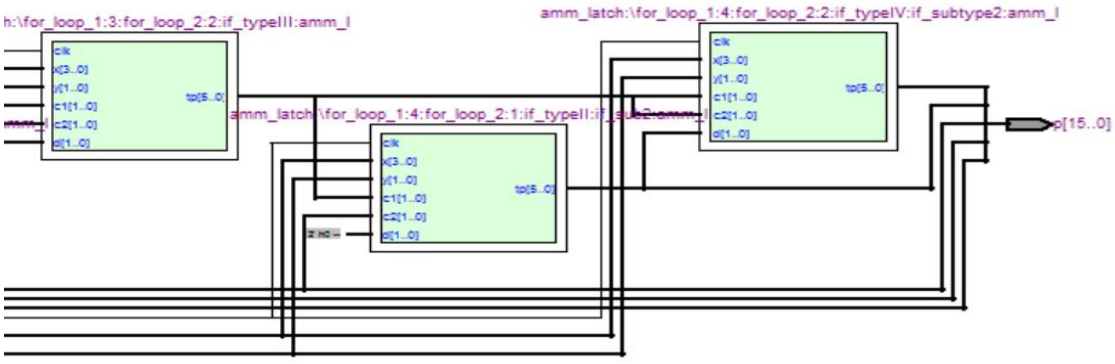

Figure 23 Zoom section of figure 21.

1	Worst-case tsu	N/A	None	7.414 ns	y[6]	amm_latch:\for_loop_1:4:for_loop_2:2:if_typeIV:if_subtype2:amm_lllatch_d:L1|Q[5]
2	Worst-case tco	N/A	None	7.113 ns	amm_latch:\for_loop_1:4:for_loop_2:2:if_typeIV:if_subtype2:amm_lllatch_d:L1|Q[4]	p[14]
3	Worst-case th	N/A	None	-3.532 ns	x[0]	amm_latch:\for_loop_1:1:for_loop_2:1:if_typeI:if_subtypeII:amm_lllatch_d:L1|Q[1]
4	Clock Setup: 'clk'	N/A	None	335.68 MHz (period = 2.979 ns)	amm_latch:\for_loop_1:3:for_loop_2:2:if_typeIII:amm_lllatch_d:L1|Q[3]	amm_latch:\for_loop_1:4:for_loop_2:2:if_typeIV:if_subtype2:amm_lllatch_d:L1|Q[5]
5	Total number of failed paths					

Figure 24 worst cases TCO

tco

	Slack	Required tco	Actual tco	From	To	From Clock
1	N/A	None	6.857 ns	amm_latch:\for_loop_1:1:for_loop_2:1:if_typeI:if_subtypeII:amm_lllatch_d:L1|Q[0]	p[0]	clk
2	N/A	None	5.955 ns	amm_latch:\for_loop_1:1:for_loop_2:1:if_typeI:if_subtypeII:amm_lllatch_d:L1|Q[1]	p[1]	clk
3	N/A	None	6.504 ns	amm_latch:\for_loop_1:2:for_loop_2:1:if_typeII:if_sub1:amm_lllatch_d:L1|Q[0]	p[2]	clk
4	N/A	None	6.669 ns	amm_latch:\for_loop_1:2:for_loop_2:1:if_typeII:if_sub1:amm_lllatch_d:L1|Q[1]	p[3]	clk
5	N/A	None	6.216 ns	amm_latch:\for_loop_1:3:for_loop_2:1:if_typeII:if_sub1:amm_lllatch_d:L1|Q[0]	p[4]	clk
6	N/A	None	6.161 ns	amm_latch:\for_loop_1:3:for_loop_2:1:if_typeII:if_sub1:amm_lllatch_d:L1|Q[1]	p[5]	clk
7	N/A	None	6.226 ns	amm_latch:\for_loop_1:4:for_loop_2:1:if_typeII:if_sub2:amm_lllatch_d:L1|Q[0]	p[6]	clk
8	N/A	None	6.695 ns	amm_latch:\for_loop_1:4:for_loop_2:1:if_typeII:if_sub2:amm_lllatch_d:L1|Q[1]	p[7]	clk
9	N/A	None	6.193 ns	amm_latch:\for_loop_1:4:for_loop_2:1:if_typeII:if_sub2:amm_lllatch_d:L1|Q[2]	p[8]	clk
10	N/A	None	6.776 ns	amm_latch:\for_loop_1:4:for_loop_2:1:if_typeII:if_sub2:amm_lllatch_d:L1|Q[3]	p[9]	clk
11	N/A	None	6.577 ns	amm_latch:\for_loop_1:4:for_loop_2:2:if_typeIV:if_subtype2:amm_lllatch_d:L1|Q[0]	p[10]	clk
12	N/A	None	6.548 ns	amm_latch:\for_loop_1:4:for_loop_2:2:if_typeIV:if_subtype2:amm_lllatch_d:L1|Q[1]	p[11]	clk
13	N/A	None	7.029 ns	amm_latch:\for_loop_1:4:for_loop_2:2:if_typeIV:if_subtype2:amm_lllatch_d:L1|Q[2]	p[12]	clk
14	N/A	None	6.601 ns	amm_latch:\for_loop_1:4:for_loop_2:2:if_typeIV:if_subtype2:amm_lllatch_d:L1|Q[3]	p[13]	clk
15	N/A	None	7.113 ns	amm_latch:\for_loop_1:4:for_loop_2:2:if_typeIV:if_subtype2:amm_lllatch_d:L1|Q[4]	p[14]	clk
16	N/A	None	6.461 ns	amm_latch:\for_loop_1:4:for_loop_2:2:if_typeIV:if_subtype2:amm_lllatch_d:L1|Q[5]	p[15]	clk

Figure 25 TCO from clock to each o/p bit.

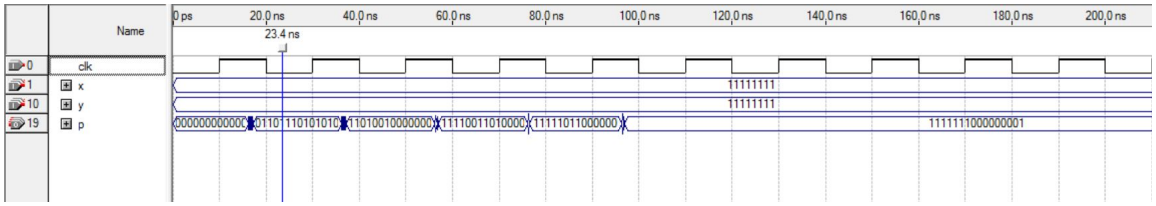

Figure 26 Waveform simulation of N=8 module.

- As we have seen in the part 2 of our project, which is manually generated block 8by8 AMM suing 4by2 AMMs and latches had similar delay and similar output pattern.
- There are four latches which contribute to the output directly and the pair of AMM gives out the o/p signal p.

CASE I : GENERIC (N := 16)

- **The value of N can only be multiple of 4.**
  Math: (N>4 and ((N mod 4)=0))

```
generic(N       : integer :=16);    --- Changing the N integer determines the NxN
port(   clk     : std_logic;
        x       : in std_logic_vector(N-1 downto 0);
        y       : in std_logic_vector(N-1 downto 0);
        p       : out std_logic_vector(N+N-1 downto 0)
        );

        end nbyn_amm;
```

Figure 27 Section of code showing N:=16.

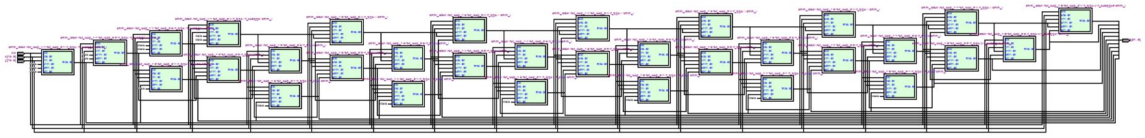

Figure 28 Automated generated RTL block for N=16.

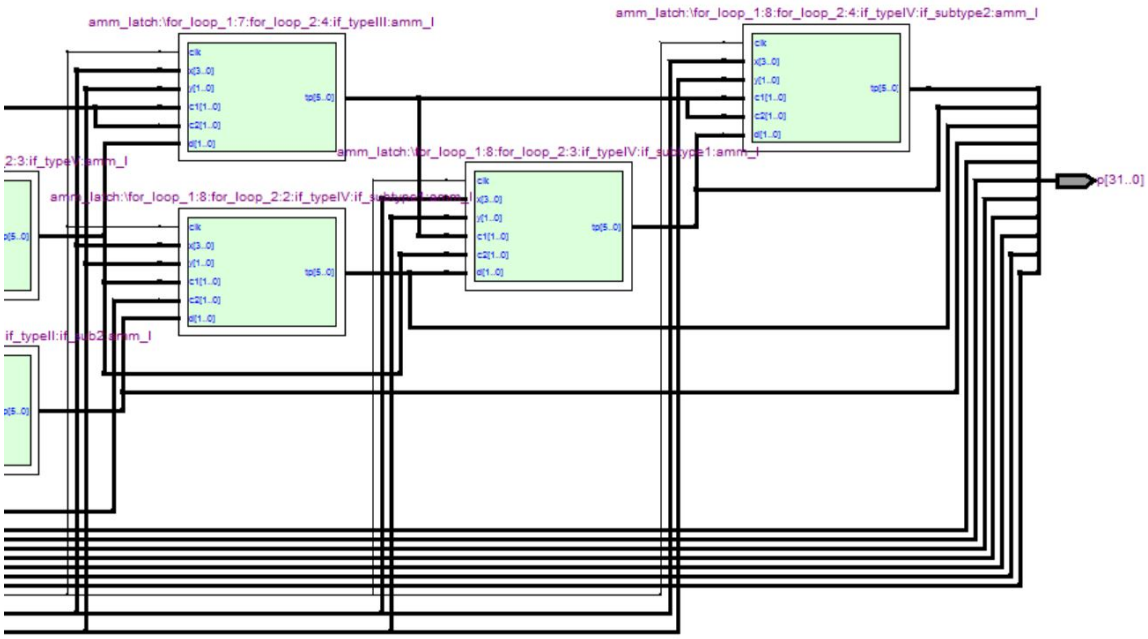

Figure 29 Zoomed section of o/p p.

Quartus II Version	9.1 Build 222 10/21/2009 SJ Web Edition
Revision Name	nbyn_amm
Top-level Entity Name	nbyn_amm
Family	Cyclone II
Met timing requirements	Yes
Total logic elements	601 / 4,608 (13 %)
Total combinational functions	601 / 4,608 (13 %)
Dedicated logic registers	192 / 4,608 (4 %)
Total registers	192
Total pins	65 / 158 (41 %)
Total virtual pins	0
Total memory bits	0 / 119,808 (0 %)
Embedded Multiplier 9-bit elements	0 / 26 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP2C5F256C6
Timing Models	Final

Figure 30 Report N=16

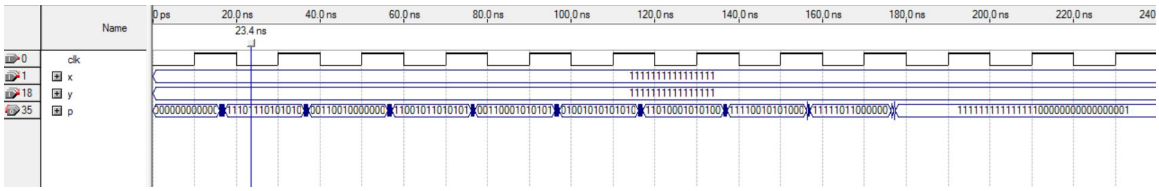

Figure 31 Waveform simulation with N=16.

CASE I : GENERIC (N := 32)

```
entity nbyn_amm is

    generic(N       : integer :=32);    --- Changing the N integer determines the NxN
    port(   clk     : std_logic;
            x       : in std_logic_vector(N-1 downto 0);
            y       : in std_logic_vector(N-1 downto 0);
            p       : out std_logic_vector(N+N-1 downto 0)
            );

            end nbyn_amm;
```

Figure 32 Section of code showing N:=8.

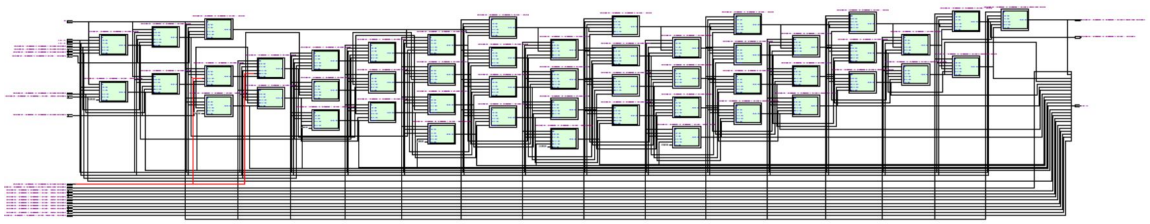

Figure 33 automated generated RTL block for N=32.

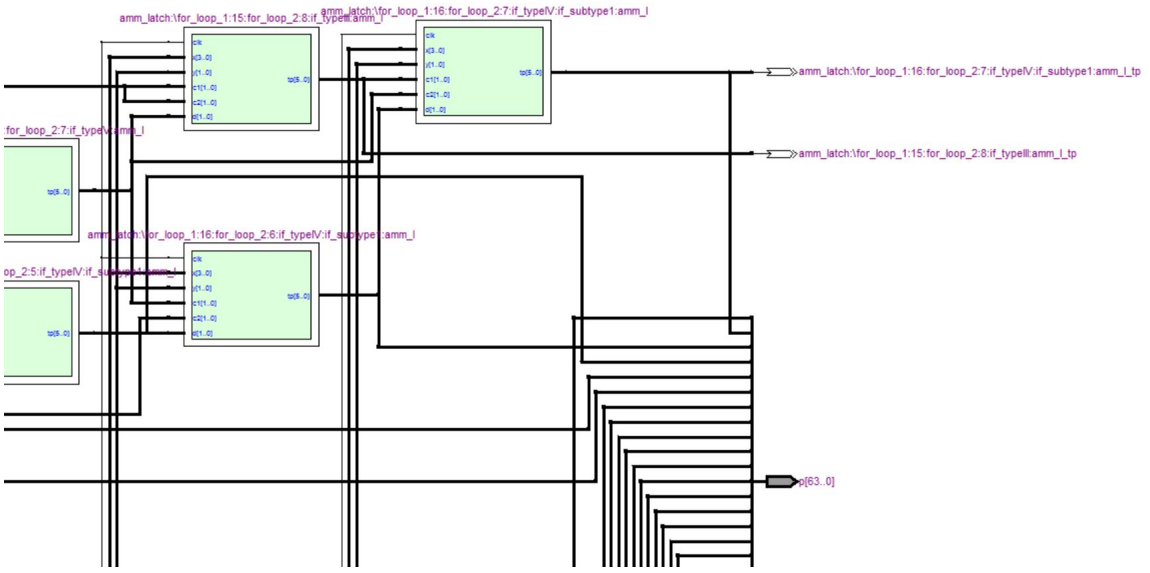

Figure 34 Zoomed section of above figure with o/p P with the range.

Quartus II Version	9.1 Build 222 10/21/2009 SJ Web Edition
Revision Name	nbyn_amm
Top-level Entity Name	nbyn_amm
Family	Cyclone II
Met timing requirements	Yes
Total logic elements	2,425 / 4,608 (53 %)
Total combinational functions	2,425 / 4,608 (53 %)
Dedicated logic registers	768 / 4,608 (17 %)
Total registers	768
Total pins	129 / 158 (82 %)
Total virtual pins	0
Total memory bits	0 / 119,808 (0 %)
Embedded Multiplier 9-bit elements	0 / 26 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP2C5F256C6
Timing Models	Final

Figure 35 report for N=32.

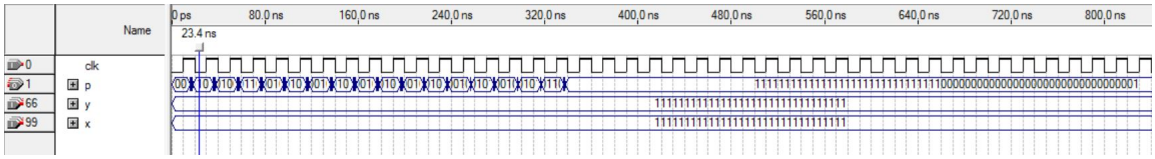

Figure 36 Waveform simulation for N=32.

----------Full_adder_test VHDL code----------

```
library ieee;
use ieee.std_logic_1164.all;

entity full_adder_test is
port     ( a,b,c: in std_logic;
           sum,ca: out std_logic
           );
end full_adder_test;

architecture rtl1 of full_adder_test is

signal S1, S2, S3: std_logic;

begin
                    s1     <= ( a XOR b );
                    s2     <= ( c AND s1 );
                    s3     <= ( a AND b );
                    Sum     <= ( s1 XOR c );
                    ca       <= ( s2 OR s3 );
end rtl1;
```

----------Latch_D VHDL code----------

-

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

entity latch_d is
   Port ( D  : in  STD_LOGIC_vector(5 downto 0);
          EN : in  STD_LOGIC;
          Q  : out STD_LOGIC_vector(5 downto 0));
end latch_d;

architecture Behavioral of latch_d is

   signal DATA : STD_LOGIC;
   signal zero : std_logic_vector(5 downto 0);

begin
process(D, EN)
   begin
      -- clock rising edge
          if (EN='1' and EN'event) then
             Q <= D;
          end if;
end process;

end Behavioral;
```

```
----------AMM 8by8 VHDL code----------
------------------------------------------------------------------------------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity add_mm is

 port            (a              : in std_logic_vector(3 downto 0);
                  c              : in std_logic_vector(3 downto 0);
                  b              : in std_logic_vector(1 downto 0);
                  d              : in std_logic_vector(1 downto 0);
                  p              : out std_logic_vector(5 downto 0));
 end add_mm;

architecture add_rtl of add_mm is

component full_adder_test is
 port   (         a,b,c    : in std_logic;
                  ca,sum : out std_logic
        );
 end component;

signal cat, st           : std_logic_vector (7 downto 0);
signal ab0, ab1          : std_logic_vector (3 downto 0);
begin

ab0(0) <= a(0) and b(0);
ab0(1) <= a(1) and b(0);
ab0(2) <= a(2) and b(0);
ab0(3) <= a(3) and b(0);
ab1(0) <= a(0) and b(1);
ab1(1) <= a(1) and b(1);
ab1(2) <= a(2) and b(1);
ab1(3) <= a(3) and b(1);

fa1: full_adder_test port map(ab0(0),d(0),  c(0),  cat(0),st(0));
fa2: full_adder_test port map(ab0(1),ab1(0),c(1),  cat(1),st(1));
fa3: full_adder_test port map(ab0(2),ab1(1),c(2),  cat(2),st(2));
fa4: full_adder_test port map(ab0(3),ab1(2),c(3),  cat(3),st(3));
fa5: full_adder_test port map(st(1) ,cat(0),d(1),  cat(4),st(4));
fa6: full_adder_test port map(st(2) ,cat(1),cat(4),cat(5),st(5));
fa7: full_adder_test port map(st(3), cat(2),cat(5),cat(6),st(6));
fa8: full_adder_test port map(ab1(3),cat(3),cat(6),cat(7),st(7));

P(0) <= st(0);
p(1) <= st(4);
p(2) <= st(5);
p(3) <= st(6);
p(4) <= st(7);
p(5) <= cat(7);

end add_rtl;

------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
----------AMM 4by2 VHDL code----------
library ieee;
use ieee.std_logic_1164.all;

entity e_e_amm is
port( --clk: in std_logic_vector;
          x       : in std_logic_vector(7  downto 0);
          y       : in std_logic_vector(7  downto 0);
          p :out std_logic_vector(15 downto 0));
end e_e_amm;

architecture amm_rtl of e_e_amm is

component add_mm is

port (          a                 : in std_logic_vector(3 downto 0);
                c                 : in std_logic_vector(3 downto 0);
                b                 : in std_logic_vector(1 downto 0);
                d                 : in std_logic_vector(1 downto 0);
                p                 :out std_logic_vector(5 downto 0));

 end component;

signal   pl,ph              : std_logic_vector(23 downto 0);

signal  tc                        : std_logic_vector(15 downto 0);

begin

amm1 : add_mm port map(x(3 downto 0), "0000", y(1 downto 0), "00", pl(5 downto 0));
tc(1 downto 0) <= pl(3 downto 2);

amm2 : add_mm port map(x(7 downto 4), "0000",     y(1 downto 0), pl(5 downto 4),
ph(5 downto 0));
tc(3 downto 2) <= ph(1 downto 0);

amm3 : add_mm port map(x(3 downto 0), tc(3 downto 0),y(3 downto 2), "00",
pl(11 downto 6));
tc(5 downto 4) <= pl(9 downto 8);

amm4 : add_mm port map(x(7 downto 4), ph(5 downto 2),y(3 downto 2), pl(11 downto
10), ph(11 downto 6));
tc(7 downto 6) <= ph(7 downto 6);

amm5 : add_mm port map(x(3 downto 0), tc(7 downto 4),y(5 downto 4), "00",
pl(17 downto 12));
tc(9 downto 8) <= pl(15 downto 14);

amm6 : add_mm port map(x(7 downto 4), ph(11 downto 8),y(5 downto 4), pl(17 downto
16),ph(17 downto 12));
tc(11 downto 10)<= ph(13 downto 12);

amm7 : add_mm port map(x(3 downto 0), tc(11 downto 8),y(7 downto 6),"00",
pl(23 downto 18));
tc(13 downto 12)<= pl(21 downto 20);
```

```
amm8 : add_mm port map(x(7 downto 4), ph(17 downto 14),y(7 downto 6),pl(23 downto
22),
ph(23 downto 18));
tc(15 downto 14) <= ph(19 downto 18);

p(1 downto 0) <=        pl(1 downto 0);

p(3 downto 2) <=        pl(7 downto 6);

p(5 downto 4) <=        pl(13 downto 12);

p(9 downto 6) <=        pl(21 downto 18);

p(15 downto 10)         <=  ph(23 downto 18);

end amm_rtl;
```

----------AMM 8by8 with latch VHDL code----------

```
-- using latch --

library ieee;
use ieee.std_logic_1164.all;

entity latch_88amm is

port( clk: in std_logic;
          x       : in std_logic_vector(7  downto 0);
          y       : in std_logic_vector(7  downto 0);
          c : in std_logic_vector(7  downto 0);
          d       : in std_logic_vector(7  downto 0);
          tp : out std_logic_vector(15 downto 0));
end latch_88amm;

architecture amm_rtl of latch_88amm is

component add_mm is

port (          a               : in std_logic_vector(3 downto 0);
                c               : in std_logic_vector(3 downto 0);
                b               : in std_logic_vector(1 downto 0);
                d               : in std_logic_vector(1 downto 0);
                p               :out std_logic_vector(5 downto 0));
 end component;

component latch_d is
port (  D  : in  STD_LOGIC_vector(5 downto 0);
     EN : in  STD_LOGIC;
     Q  : out STD_LOGIC_vector(5 downto 0));
end component;
```

```
signal  pl,ph           : std_logic_vector(23 downto 0);
signal  p                       : std_logic_vector(15  downto 0);
signal  tc              : std_logic_vector(15 downto 0);
signal  p1,p2,p3,p4,p5,p6: std_logic_vector(5 downto 0);
signal clk_p : STD_LOGIC;
signal zero : std_logic;

begin

amm1 : add_mm port map(x(3 downto 0), c(3 downto 0),            y(1 downto 0), d(1
downto 0),p1(5 downto 0));
L1      : latch_d port map(p1(5 downto 0),clk,pl(5 downto 0));

amm2 : add_mm port map(x(7 downto 4), c(7 downto 4),            y(1 downto 0), pl(5
downto 4),p2(5 downto 0));
L2      : latch_d port map(p2(5 downto 0),clk,ph(5 downto 0));
tc(3 downto 2) <= ph(1 downto 0);
tc(1 downto 0) <= pl(3 downto 2);

amm3 : add_mm port map(x(3 downto 0), tc(3 downto 0),y(3 downto 2), d(3 downto 2),
p3(5 downto 0));
L3      : latch_d port map(p3(5 downto 0),clk,pl(11 downto 6));

amm4 : add_mm port map(x(7 downto 4), ph(5 downto 2),y(3 downto 2), pl(11 downto
10),    p4(5 downto 0));
L4      : latch_d port map(p4(5 downto 0),clk,ph(11 downto 6));

tc(5 downto 4) <= pl(9 downto 8);
tc(7 downto 6) <= ph(7 downto 6);

amm5 : add_mm port map(x(3 downto 0), tc(7 downto 4),y(5 downto 4), d(5 downto
4),p5(5 downto 0));
L5 : latch_d port map(p5(5 downto 0),clk,pl(17 downto 12));

amm6 : add_mm port map(x(7 downto 4), ph(11 downto 8),y(5 downto 4), pl(17 downto
16),    p6(5 downto 0));
L6: latch_d port map(p6(5 downto 0),clk,ph(17 downto 12));

tc(9 downto 8) <= pl(15 downto 14);
tc(11 downto 10)<= ph(13 downto 12);

amm7 : add_mm port map(x(3 downto 0), tc(11 downto 8),y(7 downto 6), d(7 downto
6),pl(23 downto 18));

amm8 : add_mm port map(x(7 downto 4), ph(17 downto 14),y(7 downto 6),pl(23 downto
22),    ph(23 downto 18));

                tp(1 downto 0)          <=      pl(1 downto 0);
                tp(3 downto 2)          <=      pl(7 downto 6);
                tp(5 downto 4)          <=      pl(13 downto 12);
                tp(9 downto 6)          <=      pl(21 downto 18);
                tp(15 downto 10)        <= ph(23 downto 18);
end amm_rtl;
```

```
----------AMM 4by2 with latch VHDL code----------
------------------------------------------------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity amm_latch is
port( clk: in std_logic;
          x       : in std_logic_vector(3  downto 0);
          y       : in std_logic_vector(1  downto 0);
          c1 : in std_logic_vector(1  downto 0);--3downto2--
          c2 : in std_logic_vector(1  downto 0);--1downto0--
          d       : in std_logic_vector(1  downto 0);
          tp : out std_logic_vector(5 downto 0));
end amm_latch;

architecture amm_rtl of amm_latch is

component add_mm is
port (          a                 : in std_logic_vector(3 downto 0);
                c                 : in std_logic_vector(3 downto 0);
                b                 : in std_logic_vector(1 downto 0);
                d                 : in std_logic_vector(1 downto 0);
                p                 :out std_logic_vector(5 downto 0));
 end component;

component latch_d i
port (  D  : in  STD_LOGIC_vector(5 downto 0);
     EN : in  STD_LOGIC;
     Q  : out STD_LOGIC_vector(5 downto 0));
end component;

signal  p                         : std_logic_vector(15  downto 0);
signal cmerge           : std_logic_vector(3  downto 0);

begin

cmerge(1 downto 0)<= c2;
cmerge(3 downto 2)<= c1;

amm1 : add_mm port map(x(3 downto 0), cmerge(3 downto 0), y(1 downto 0), d(1
downto 0),p(5 downto 0));
L1       : latch_d port map(p(5 downto 0),clk,tp(5 downto 0));

end amm_rtl;

------------------------------------------------------------------------------------------------------------------------
```

```
----------AMM N by N with latch VHDL code----------
------------------------------------------------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.All;

entity nbyn_amm is
generic(N       : integer :=32); --- Changing the N integer determines the NxN
port(               clk             : std_logic;
                    x               : in std_logic_vector(N-1 downto 0);
                    y               : in std_logic_vector(N-1 downto 0);
                    p               : out std_logic_vector(N+N-1 downto 0)
);
end nbyn_amm;

architecture add_rtl of nbyn_amm is

constant r : natural := integer (N/4);   -- for row of the n by n matrix
constant c : natural := integer (N/2);   -- for col of the n by n matrix

----------Component Amm_Latch----------

-- It consists of one 4x2 amm and one latch which holds the amm output for the

-- maximam delay of an amm. The idea of merging the amm and a latch is to

-- avoid the number of inter-connection to be made.

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

component amm_latch is
port( clk: in std_logic;
          x       : in std_logic_vector(3  downto 0);
          y       : in std_logic_vector(1  downto 0);
          c1 : in std_logic_vector(1  downto 0);--3downto2--
          c2 : in std_logic_vector(1  downto 0);--1downto0--
          d       : in std_logic_vector(1  downto 0);
          tp : out std_logic_vector(5 downto 0));
end component;

--signal cd feeds zeros to initial row and initial col of the matrix--
signal cd : std_logic_vector(6 downto 0) := (0 => '0', others => '0');
-- Signal pt is wire which connects P to C and D of the next amm.
signal pt : std_logic_vector((r*c*6) downto 0);

begin
--- For loop with J changes the values of signal Y--

for_loop_1: for j in 1 to c generate
begin

--- For loop with I changes the va;ues of signal X—

for_loop_2: for i in 1 to r generate
begin
```

```
----------Type I-----------------------------------
        if_typeI: if (j=1) generate

                        ----------Sub-Types I,II and III----------

                        --first row and last block--
                        if_subtypeI: if (i=r) and (r>1) generate

                                amm_l: amm_latch port map (clk,
                                                        x(((4*i)-1) downto ((4*i)4)),
                                                        y((2*j)-1 downto (2*j)-2),
                                                        cd(5 downto 4),
                                                        cd(3 downto 2),
                                                        pt((((r*6*j)-(6*(r-i)))-5)-2
                                                        downto (((r*6*j)-(6*(r-i)))-
                                                        6)-2),
                                                        pt(((r*6*j)-(6*(r-i)))-1
                                                        downto ((r*6*j)-(6*(r-i)))-6)
                                                );
                        end generate if_subtypeI;

                        if_subtypeII:if (i=1) generate
                                amm_l: amm_latch port map (clk,
                                                        x(((4*i)-1) downto ((4*i)4)),
                                                        y((2*j)-1 downto (2*j)-2),
                                                        cd(5 downto 4),
                                                        cd(3 downto 2),
                                                        cd(1 downto 0),
                                                        pt(((r*6*j)-(6*(r-i)))-1
                                                        downto ((r*6*j)-(6*(r-i)))-6)
                                                );

                                p((2*j)-1 downto (2*j)-2) <= pt(((r*6*j)-(6*(r-i)))-5
                                downto ((r*6*j)-(6*(r-i)))-6);
                        end generate if_subtypeII;

                        if_subtypeIII: if ((i>1) and (i<r)) generate
                                amm_l: amm_latch port map (clk,
                                                        x(((4*i)-1) downto ((4*i)4)),
                                                        y((2*j)-1 downto (2*j)-2),
                                                        cd(5 downto 4),
                                                        cd(3 downto 2),
                                                        pt((((r*6*j)-(6*(r-i)))-5)-2
                                                        downto (((r*6*j)-(6*(r-i)))-
                                                        6)-2),
                                                        pt(((r*6*j)-(6*(r-i)))-1
                                                        downto ((r*6*j)-(6*(r-i)))-6)
                                                );
                        end generate if_subtypeIII;
                        ----------End of sub-types--------------------
        end generate if_typeI;


----------End of Type I-------------------------------
```

```
----------Type II-------------------------------------

        if_typeII: if ((i=1) and (j>1)) generate

                if_sub1:if (j<c) generate
                                amm_l: amm_latch port map (clk,
                                                        x(((4*i)-1) downto ((4*i)4)),
                                                        y((2*j)-1 downto (2*j)-2),
                                                        pt((((r*6*j)-(6*(r-i)))-1)-
                                                        ((r*6)-2) downto (((r*6*j)-
                                                        (6*(r-i)))-2)-((r*6)-2)),
                                                        pt((((r*6*j)-(6*(r-i)))-3)-
                                                        (r*6) downto (((r*6*j)-
                                                        (6*(r-i)))-4)-(r*6)),
                                                        cd(1 downto 0),
                                                        pt(((r*6*j)-(6*(r-i)))-1
                                                        downto ((r*6*j)-(6*(r-i)))-6)
                                                );

                                p((2*j)-1 downto (2*j)-2) <= pt(((r*6*j)-(6*(r-i)))-5
                                downto ((r*6*j)-(6*(r-i)))-6);
                end generate if_sub1;

                if_sub2:if (j=c) generate

                                amm_l: amm_latch port map (clk,
                                                        x(((4*i)-1) downto ((4*i)4)),
                                                        y((2*j)-1 downto (2*j)-2),
                                                        pt((((r*6*j)-(6*(r-i)))-1)-
                                                        ((r*6)-2) downto (((r*6*j)-
                                                        (6*(r-i)))-2)-((r*6)-2)),
                                                        pt((((r*6*j)-(6*(r-i)))-3)-
                                                        (r*6) downto (((r*6*j)-
                                                        (6*(r-i)))-4)-(r*6)),
                                                        cd(1 downto 0),
                                                        pt(((r*6*j)-(6*(r-i)))-1
                                                        downto ((r*6*j)-(6*(r-i)))-6)
                                                );

                                p((2*j)-1 downto (2*j)-2) <= pt(((r*6*j)-(6*(r-i)))-5
                                downto ((r*6*j)-(6*(r-i)))-6);
                                p((2*j)+1 downto (2*j)) <= pt(((r*6*j)-(6*(r-i)))-3 downto
                                ((r*6*j)-(6*(r-i)))-4);
                end generate if_sub2;

        end generate if_typeII;

----------End of Type II------------------------------
```

```
----------Type III-------------------------------------

        if_typeIII: if ((j>1)and (j<c) and (i=r)) generate
                                amm_l: amm_latch port map (clk,
                                                x(((4*i)-1) downto ((4*i)4)),
                                                y((2*j)-1 downto (2*j)-2),
                                                pt((((r*6*j)-(6*(r-i)))-1)-
                                                (r*6) downto (((r*6*j)-
                                                (6*(r-i)))-2)-(r*6)),
                                                pt((((r*6*j)-(6*(r-i)))-3)-
                                                (r*6) downto (((r*6*j)-
                                                (6*(r-i)))-4)-(r*6)),
                                                pt((((r*6*j)-(6*(r-i)))-5)-2
                                                downto (((r*6*j)-(6*(r-i)))-
                                                6)-2),
                                                pt(((r*6*j)-(6*(r-i)))-1
                                                downto ((r*6*j)-(6*(r-i)))-6)
                                        );

        end generate if_typeIII;

----------End of Type III------------------------------

----------Type IV--------------------------------------

        if_typeIV: if((i>1) and (j=c)) generate

                if_subtype1: if ((i<r) and (j=c)) generate
                                amm_l: amm_latch port map (clk,
                                                x(((4*i)-1) downto ((4*i)4)),
                                                y((2*j)-1 downto (2*j)-2),
                                                pt((((r*6*j)-(6*(r-i)))-1)-
                                                ((r*6)-2) downto (((r*6*j)-
                                                (6*(r-i)))-2)-((r*6)-2)),
                                                pt((((r*6*j)-(6*(r-i)))-3)-
                                                (r*6) downto (((r*6*j)-(
                                                6*(r-i)))-4)-(r*6)),
                                                pt((((r*6*j)-(6*(r-i)))-5)-2
                                                downto (((r*6*j)-(6*(r-i)))-
                                                6)-2),
                                                pt(((r*6*j)-(6*(r-i)))-1
                                                downto ((r*6*j)-(6*(r-i)))-6)
                                        );

                                p(((2*j)+(4*(i-1)))-1 downto ((2*j)+(4*(i-1)))-2) <=
                                pt(((r*6*j)-    (6*(r-i)))-5 downto ((r*6*j)-(6*(r-i)))-6);
                                p(((2*j)+(4*(i-1)))+1 downto ((2*j)+(4*(i-1)))) <=
                                pt(((r*6*j)-(6*(r-i)))-3 downto ((r*6*j)-(6*(r-i)))-4);
                end generate if_subtype1;
```

```
                if_subtype2: if ((i=r) and (j=c)) generate
                                amm_l: amm_latch port map (clk,
                                                x(((4*i)-1) downto ((4*i)4)),
                                                y((2*j)-1 downto (2*j)-2),
                                                pt((((r*6*j)-(6*(r-i)))-1)-
                                                (r*6) downto (((r*6*j)-
                                                (6*(r-i)))-2)-(r*6)),
                                                pt((((r*6*j)-(6*(r-i)))-3)-
                                                (r*6) downto (((r*6*j)-
                                                (6*(r-i)))-4)-(r*6)),
                                                pt((((r*6*j)-(6*(r-i)))-5)-2
                                                downto (((r*6*j)-(6*(r-i)))-
                                                6)-2),
                                                pt(((r*6*j)-(6*(r-i)))-1
                                                downto ((r*6*j)-(6*(r-i)))-6)
                                                );

                                p(((2*j)+(4*(i-1)))-1 downto ((2*j)+(4*(i-1)))-2) <=
                                pt(((r*6*j)-(6*(r-i)))-5 downto ((r*6*j)-(6*(r-i)))-6);
                                p(((2*j)+(4*(i-1)))+1 downto ((2*j)+(4*(i-1)))) <=
                                pt(((r*6*j)-(6*(r-i)))-3 downto ((r*6*j)-(6*(r-i)))-4);
                                p(((2*j)+(4*(i-1)))+3 downto ((2*j)+(4*(i-1)))+2) <=
                                pt(((r*6*j)-(6*(r-i)))-1 downto ((r*6*j)-(6*(r-i)))-2);

                end generate if_subtype2;
        end generate if_typeIV;

----------End of Type IV----------

----------Type V----------

        if_typeV: if((i>1) and (i<r) and (j>1) and (j<c)) generate
                                amm_l: amm_latch port map (clk,
                                                x(((4*i)-1) downto ((4*i)4)),
                                                y((2*j)-1 downto (2*j)-2),
                                                pt((((r*6*j)-(6*(r-i)))-1)-
                                                ((r*6)-2) downto (((r*6*j)-
                                                (6*(r-i)))-2)-((r*6)-2)),
                                                pt((((r*6*j)-(6*(r-i)))-3)-
                                                (r*6) downto (((r*6*j)-
                                                (6*(r-i)))-4)-(r*6)),
                                                pt((((r*6*j)-(6*(r-i)))-5)-2
                                                downto (((r*6*j)-(6*(r-i)))-
                                                6)-2),
                                                pt(((r*6*j)-(6*(r-i)))-1
                                                downto ((r*6*j)-(6*(r-i)))-6)
                                                );
        end generate if_typeV;

----------End of Type V----------

end generate for_loop_2;
end generate for_loop_1;
end add_rtl;
```