McGill University

ECSE 325 - Final Project

Pipelined Complex Multiplier and Design Closure

Group 2

Section 005

Yi Zhu 260716006

Mai Zeng 260782174

Group 2
Yi Zhu 260716006
Mai Zeng 260782174

1. Introduction

In this final project, our main goal is to explore the effects of pipelining on the performance of synchronous digital systems. In order to do so, we will implement a circuit for a system to compute the square of a 32-bit complex number in VHDL and try to maximize speed of the circuit by using pipelining.

For the square computation system, a complex number *Z* can be represented by two 32-bit-signed signals: one signal *X* holds the real part of the complex number, and the other signal *Y* holds the imaginary part of the complex number. The complex number Z can then be represented by:

$$Z = X + iY$$

Thus, the square of a complex number *Z* can be represented by using two 32-bits-signals *X* and *Y*, where we take *X* and *Y* as our inputs and the 65-bits-signal Z^2 as our output:

$$Z^2 = (X + iY) \times (X + iY) = (X^2 - Y^2) + i(2XY)$$

To find the effects of pipelining on circuit speed, we first take the traditional method to implement the calculation, which uses "*" as multiplication. Then we add a pipeline register between the square operation and subtraction operation to improve the performance. Finally, we make a use of DSP Blocks, which include two 18x19 bit multipliers, to implement the multiplication operations directly. To enable the use of the pipeline registers in the DSP Blocks, we add the LPM_MULT component into our code. We implement the LPM_MULT component with 2, 3 and 4 pipelines respectively to compare the results.

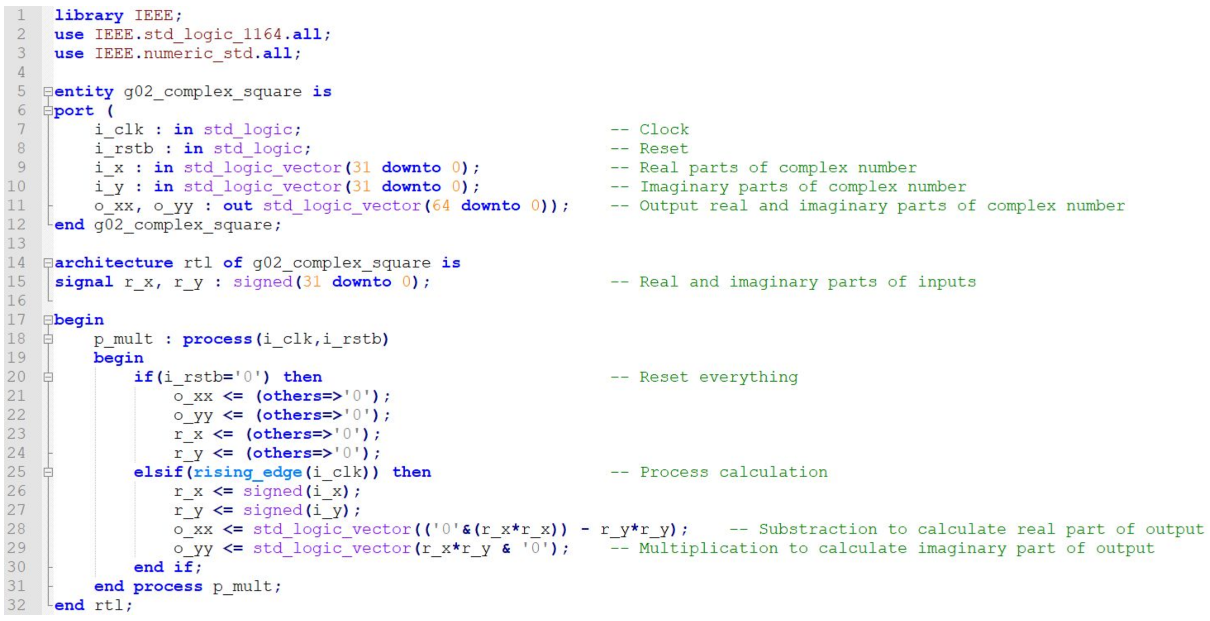
The VHDL code and timing analysis results of the original multiplication operation design are illustrated in part 2 of this report and the VHDL code and results of the design with pipeline register are shown in part 3 of this report. The VHDL code and test results by implementing the LPM_MULT component with three different numbers of pipelines can be found in part 4 of this report. All results are concluded in *Table 5* in part 5 for comparison and analysis.

In the last part of this report, we accomplish the bonus part by using our FPGA circuitry as a complex number coprocessor by the ARM processor inside the chip. In our design, both the inputs and outputs of the FPGA circuit are accessed by the processor.

2. Original Design

2.1 VHDL Code

The VHDL code below shows the original square computation system design with “*” operation. This VHDL code is taken from the lab manual.

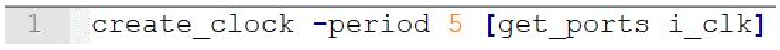


```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity g02_complex_square is
port (
    i_clk : in std_logic;                                  -- Clock
    i_rstb : in std_logic;                                 -- Reset
    i_x : in std_logic_vector(31 downto 0);                -- Real parts of complex number
    i_y : in std_logic_vector(31 downto 0);                -- Imaginary parts of complex number
    o_xx, o_yy : out std_logic_vector(64 downto 0));       -- Output real and imaginary parts of complex number
end g02_complex_square;

architecture rtl of g02_complex_square is
signal r_x, r_y : signed(31 downto 0);                     -- Real and imaginary parts of inputs

begin
    p_mult : process(i_clk,i_rstb)
    begin
        if(i_rstb='0') then                                -- Reset everything
            o_xx <= (others=>'0');
            o_yy <= (others=>'0');
            r_x <= (others=>'0');
            r_y <= (others=>'0');
        elsif(rising_edge(i_clk)) then                     -- Process calculation
            r_x <= signed(i_x);
            r_y <= signed(i_y);
            o_xx <= std_logic_vector(('0'&(r_x*r_x)) - r_y*r_y);    -- Substraction to calculate real part of output
            o_yy <= std_logic_vector(r_x*r_y & '0');       -- Multiplication to calculate imaginary part of output
        end if;
    end process p_mult;
end rtl;
```

Figure 2.1-1 VHDL Code for Original Design

As we can see in the entity part, there are four inputs: i_clk, i_rstb, i_x and i_y. The input i_clk is used to achieve the time-sensitive property of this square computation system while i_rstb is used to accomplish the reset function. Two 32-bits-registers i_x and i_y are used to hold the real and imaginary parts of the input complex number, ie. *X* and *Y* as we mentioned in the introduction part. There are also two outputs: o_xx and o_yy, which are 65-bits-registers to hold the real and imaginary parts of the complex number result, ie. *X^2-Y^2* and *2XY* respectively. In the architecture section, there are two 32-bits-signals r_x, r_y which are used to hold the signed-type number of the real and imaginary parts of the input complex number. In the process block, since the square computation system is time-sensitive and has the reset function, we have i_clk and i_rstb in the sensitivity list. Thus, all computations are done in one clock cycle. For the reset function, we have one if statement that when the i_rstb signal is equal to ‘0’, we set all outputs and signals to be ‘0’. Otherwise, we follow the formula listed in the introduction part above to calculate the real part and imaginary part outputs for the result complex number.

We also create a timing constraints file with clock period to 5ns to achieve an equivalent 200MHz clock. The code for .sdc file is shown in *Figure 2.1-2* below:

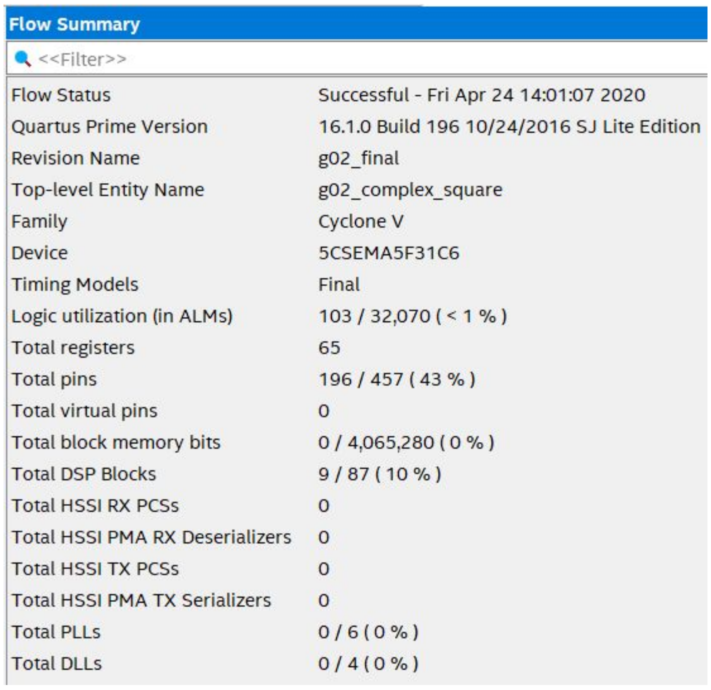


```
create_clock -period 5 [get_ports i_clk]
```

Figure 2.1-2 Timing Constraints with 5ns Clock Period

2.2 Resource Utilization

The flow summary of the original square computation system design with "*" operation is shown in *Figure 2.2* below.



Flow Summary

<<Filter>>

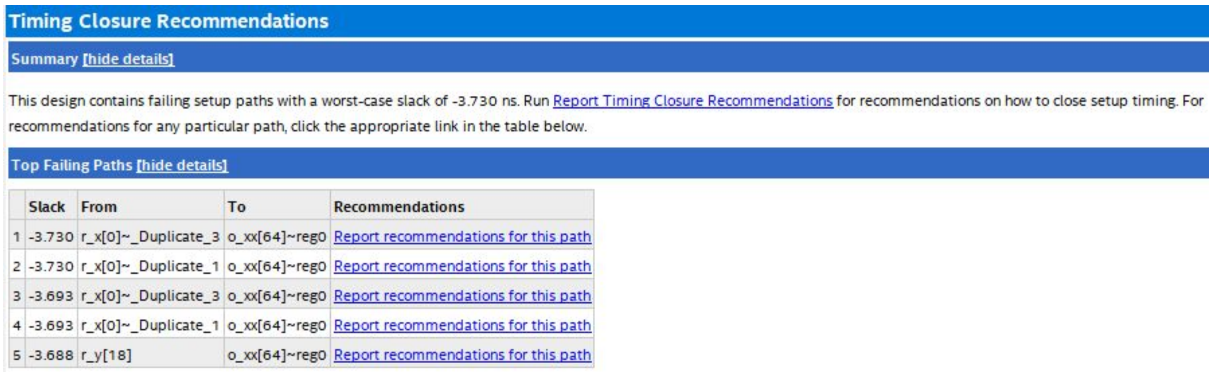
Flow Status	Successful - Fri Apr 24 14:01:07 2020
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	g02_final
Top-level Entity Name	g02_complex_square
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	103 / 32,070 (< 1 %)
Total registers	65
Total pins	196 / 457 (43 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	9 / 87 (10 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 2.2 Flow Summary for Original Design

We can see that there are 103 of 32070 ALMs used, which has the logic utilization less than 1%. Total 65 registers are used and 9 of 87 DSP Blocks are used, taking 10% of the total DSP Blocks.

2.3 Timing Analysis by TimeQuest

The following figures show the TimeQuest Timing Analyzer Report, Fmax summary and the Setup Summary in "Slow 1100mv 0C Model" for the original square computation system design with "*" operation.



Timing Closure Recommendations

Summary [hide details]

This design contains failing setup paths with a worst-case slack of -3.730 ns. Run Report Timing Closure Recommendations for recommendations on how to close setup timing. For recommendations for any particular path, click the appropriate link in the table below.

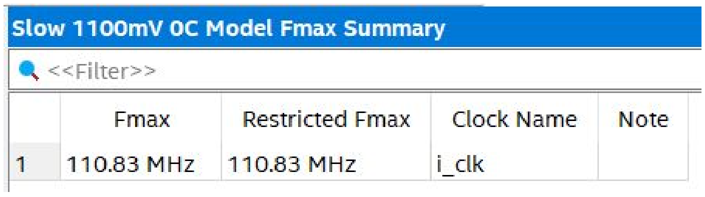
Top Failing Paths [hide details]

	Slack	From	To	Recommendations
1	-3.730	r_x[0]~_Duplicate_3	o_xx[64]~reg0	Report recommendations for this path
2	-3.730	r_x[0]~_Duplicate_1	o_xx[64]~reg0	Report recommendations for this path
3	-3.693	r_x[0]~_Duplicate_3	o_xx[64]~reg0	Report recommendations for this path
4	-3.693	r_x[0]~_Duplicate_1	o_xx[64]~reg0	Report recommendations for this path
5	-3.688	r_y[18]	o_xx[64]~reg0	Report recommendations for this path

Figure 2.3-1 TimeQuest Summary for Original Design

From the TimeQuest Summary above we can see that there are 5 timing violations in the timing analysis.

Group 2
Yi Zhu 260716006
Mai Zeng 260782174

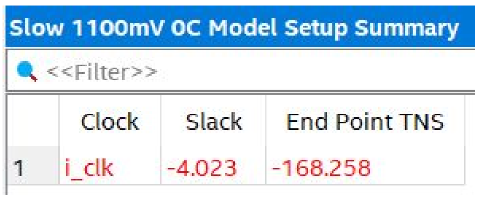


Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	110.83 MHz	110.83 MHz	i_clk	

Figure 2.3-2 Fmax Summary for Original Design



Slow 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	i_clk	-4.023	-168.258

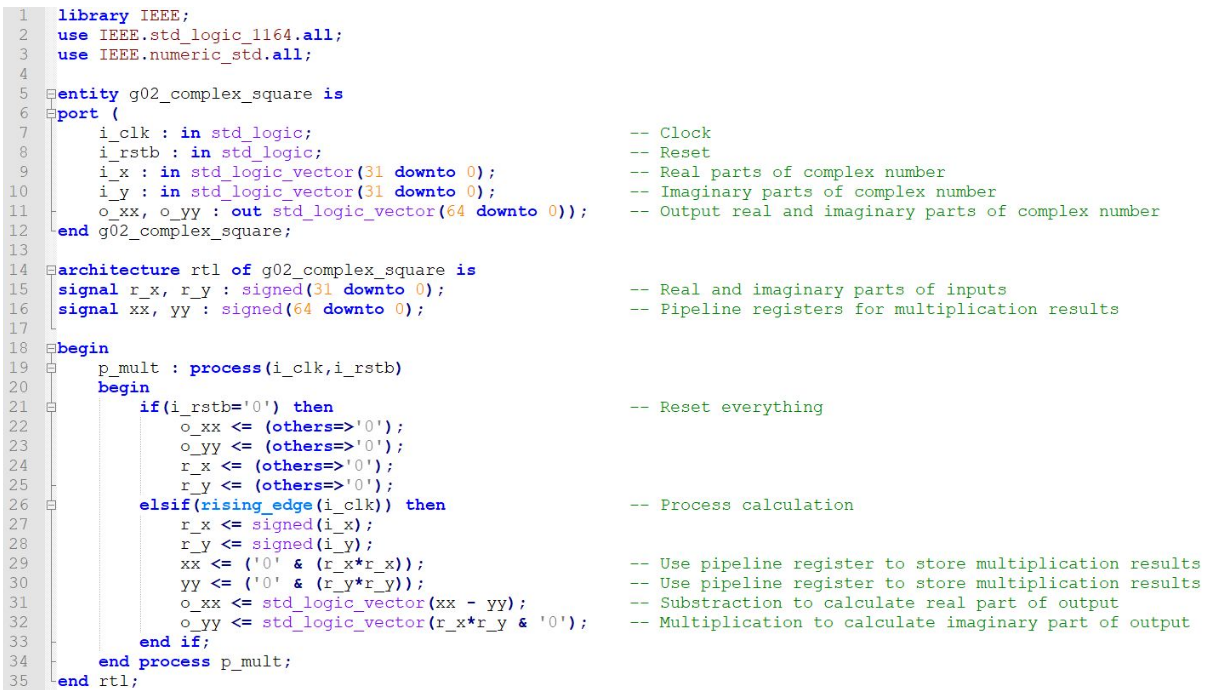
Figure 2.3-3 Setup Summary for Original Design

From two figures above we can see that for the "Slow 1100mv 0C Model", the maximum frequency is 110.83MHz while i_clk slack is -4.023. Thus, this design does not pass the timing test.

3. Design with a Pipeline

3.1 VHDL Code

The VHDL code below shows the square computation system design with pipeline register added between the multiplication and subtraction operations.



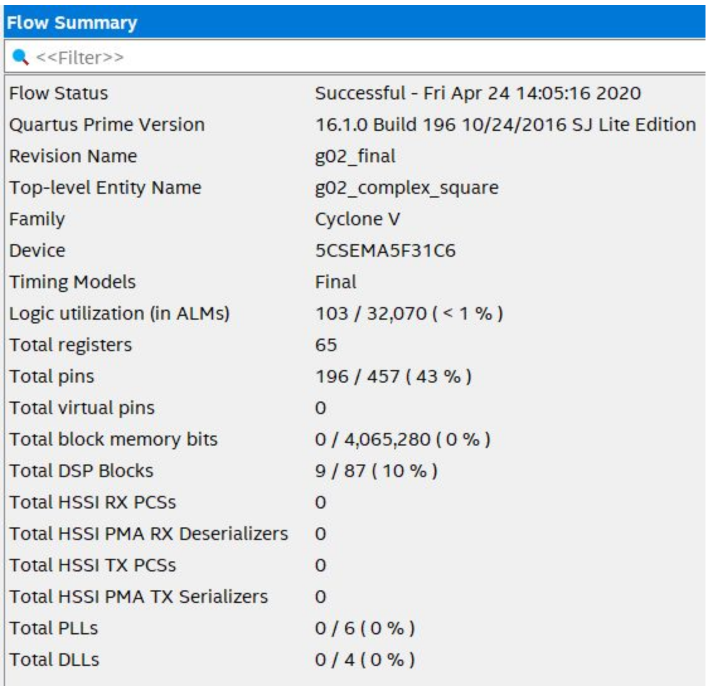
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity g02_complex_square is
port (
    i_clk : in std_logic;                               -- Clock
    i_rstb : in std_logic;                              -- Reset
    i_x : in std_logic_vector(31 downto 0);             -- Real parts of complex number
    i_y : in std_logic_vector(31 downto 0);             -- Imaginary parts of complex number
    o_xx, o_yy : out std_logic_vector(64 downto 0));    -- Output real and imaginary parts of complex number
end g02_complex_square;

architecture rtl of g02_complex_square is
signal r_x, r_y : signed(31 downto 0);                  -- Real and imaginary parts of inputs
signal xx, yy : signed(64 downto 0);                    -- Pipeline registers for multiplication results

begin
    p_mult : process(i_clk,i_rstb)
    begin
        if(i_rstb='0') then                             -- Reset everything
            o_xx <= (others=>'0');
            o_yy <= (others=>'0');
            r_x <= (others=>'0');
            r_y <= (others=>'0');
        elsif(rising_edge(i_clk)) then                  -- Process calculation
            r_x <= signed(i_x);
            r_y <= signed(i_y);
            xx <= ('0' & (r_x*r_x));                    -- Use pipeline register to store multiplication results
            yy <= ('0' & (r_y*r_y));                    -- Use pipeline register to store multiplication results
            o_xx <= std_logic_vector(xx - yy);          -- Substraction to calculate real part of output
            o_yy <= std_logic_vector(r_x*r_y & '0');    -- Multiplication to calculate imaginary part of output
        end if;
    end process p_mult;
end rtl;
```

Figure 3.1 VHDL Code for Design with a Pipeline

In this design, we add two 65-bits-signals xx and yy as pipeline registers to store the square result of r_x and r_y respectively. We keep everything unchanged except instead of doing the subtraction and multiplication simultaneously, we do the multiplication first and process the subtraction to calculate the real part of the output.

3.2 Resource Utilization

The flow summary of the square computation system design with pipeline register implemented is shown in *Figure 3.2* below:



Flow Summary

<<Filter>>

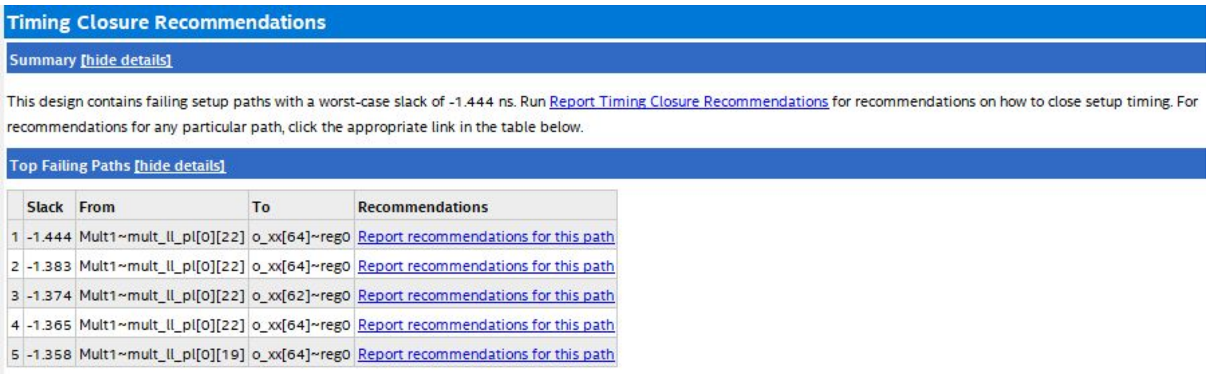
Flow Status	Successful - Fri Apr 24 14:05:16 2020
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	g02_final
Top-level Entity Name	g02_complex_square
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	103 / 32,070 (< 1 %)
Total registers	65
Total pins	196 / 457 (43 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	9 / 87 (10 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 3.2 Flow Summary for Design with a Pipeline

We can see that the utilization is unchanged. There are 103 of 32070 ALMs used, showing logic utilization less than 1%. Total 65 registers are used and 9 of 87 DSP Blocks are used, taking 10% of the total DSP Blocks.

3.3 Timing Analysis by TimeQuest

From the TimeQuest Summary below we can see that there are 5 timing violations in the timing analysis.



Timing Closure Recommendations

Summary [hide details]

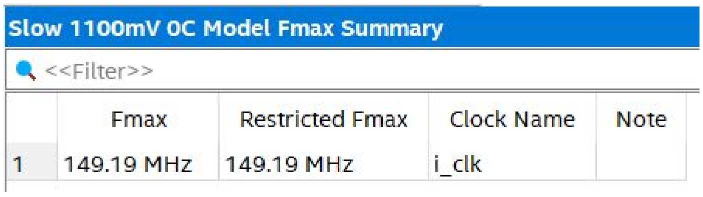
This design contains failing setup paths with a worst-case slack of -1.444 ns. Run Report Timing Closure Recommendations for recommendations on how to close setup timing. For recommendations for any particular path, click the appropriate link in the table below.

Top Failing Paths [hide details]

	Slack	From	To	Recommendations
1	-1.444	Mult1~mult_ll_pl[0][22]	o_xx[64]~reg0	Report recommendations for this path
2	-1.383	Mult1~mult_ll_pl[0][22]	o_xx[64]~reg0	Report recommendations for this path
3	-1.374	Mult1~mult_ll_pl[0][22]	o_xx[62]~reg0	Report recommendations for this path
4	-1.365	Mult1~mult_ll_pl[0][22]	o_xx[64]~reg0	Report recommendations for this path
5	-1.358	Mult1~mult_ll_pl[0][19]	o_xx[64]~reg0	Report recommendations for this path

Figure 3.3-1 TimeQuest Summary for Design with a Pipeline

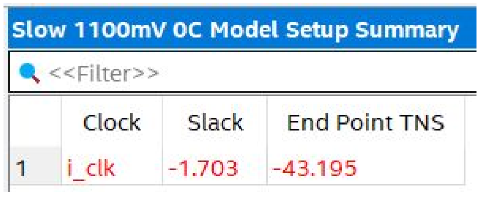
Group 2
Yi Zhu 260716006
Mai Zeng 260782174



Slow 1100mV 0C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	149.19 MHz	149.19 MHz	i_clk	

Figure 3.3-2 Fmax Summary for Design with a Pipeline



Slow 1100mV 0C Model Setup Summary

	Clock	Slack	End Point TNS
1	i_clk	-1.703	-43.195

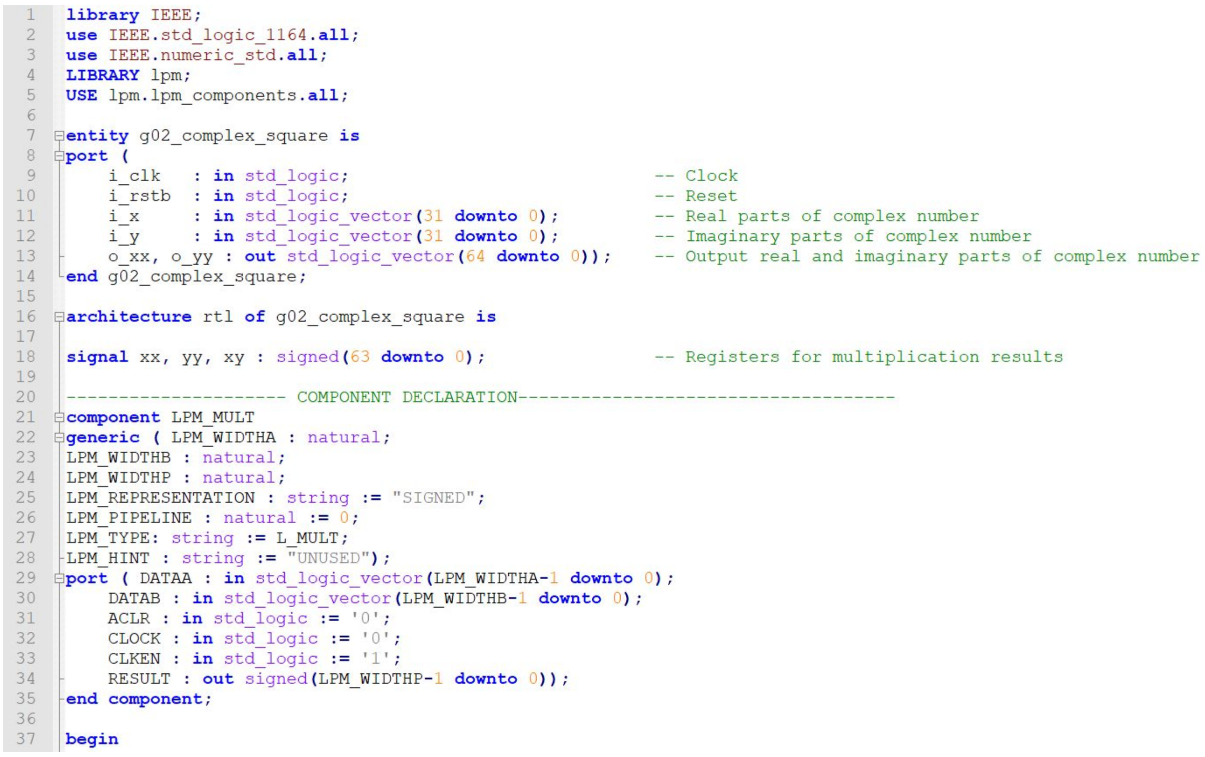
Figure 3.3-3 Setup Summary for Design with a Pipeline

From two figures above we can see that for the “Slow 1100mv 0C Model”, the maximum frequency is 149.19MHz while i_clk slack is -1.703. Thus, this design does not pass the timing test.

4. LPM_MULT Design with Pipelines

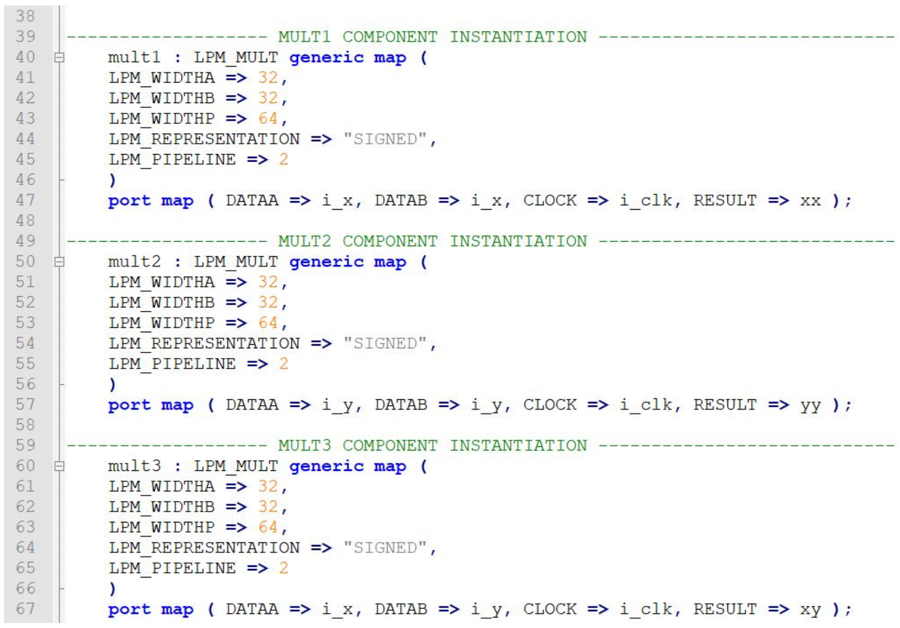
4.1 VHDL Code

The VHDL code below shows the square computation system design with using pipeline registers in the DSP Blocks. This VHDL code is taken from the lab manual with modification.



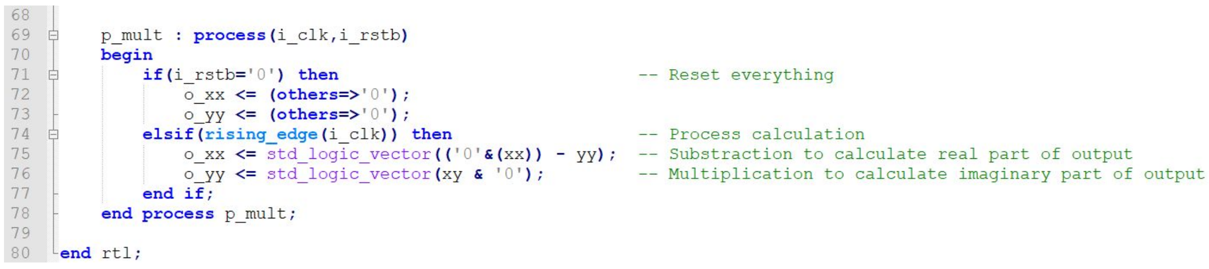
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
LIBRARY lpm;
USE lpm.lpm_components.all;

entity g02_complex_square is
port (
    i_clk   : in std_logic;                              -- Clock
    i_rstb  : in std_logic;                              -- Reset
    i_x     : in std_logic_vector(31 downto 0);          -- Real parts of complex number
    i_y     : in std_logic_vector(31 downto 0);          -- Imaginary parts of complex number
    o_xx, o_yy : out std_logic_vector(64 downto 0));     -- Output real and imaginary parts of complex number
end g02_complex_square;

architecture rtl of g02_complex_square is

signal xx, yy, xy : signed(63 downto 0);                 -- Registers for multiplication results

--------------------- COMPONENT DECLARATION-----------------------------------
component LPM_MULT
generic ( LPM_WIDTHA : natural;
LPM_WIDTHB : natural;
LPM_WIDTHP : natural;
LPM_REPRESENTATION : string := "SIGNED";
LPM_PIPELINE : natural := 0;
LPM_TYPE: string := L_MULT;
LPM_HINT : string := "UNUSED");
port ( DATAA : in std_logic_vector(LPM_WIDTHA-1 downto 0);
    DATAB : in std_logic_vector(LPM_WIDTHB-1 downto 0);
    ACLR : in std_logic := '0';
    CLOCK : in std_logic := '0';
    CLKEN : in std_logic := '1';
    RESULT : out signed(LPM_WIDTHP-1 downto 0));
end component;

begin
```

Figure 4.1-1 VHDL Code for LPM_MULT Design with 2 Pipelines Part 1



```vhdl

------------------- MULT1 COMPONENT INSTANTIATION ----------------------------
    mult1 : LPM_MULT generic map (
    LPM_WIDTHA => 32,
    LPM_WIDTHB => 32,
    LPM_WIDTHP => 64,
    LPM_REPRESENTATION => "SIGNED",
    LPM_PIPELINE => 2
    )
    port map ( DATAA => i_x, DATAB => i_x, CLOCK => i_clk, RESULT => xx );

------------------- MULT2 COMPONENT INSTANTIATION ----------------------------
    mult2 : LPM_MULT generic map (
    LPM_WIDTHA => 32,
    LPM_WIDTHB => 32,
    LPM_WIDTHP => 64,
    LPM_REPRESENTATION => "SIGNED",
    LPM_PIPELINE => 2
    )
    port map ( DATAA => i_y, DATAB => i_y, CLOCK => i_clk, RESULT => yy );

------------------- MULT3 COMPONENT INSTANTIATION ----------------------------
    mult3 : LPM_MULT generic map (
    LPM_WIDTHA => 32,
    LPM_WIDTHB => 32,
    LPM_WIDTHP => 64,
    LPM_REPRESENTATION => "SIGNED",
    LPM_PIPELINE => 2
    )
    port map ( DATAA => i_x, DATAB => i_y, CLOCK => i_clk, RESULT => xy );
```

Figure 4.1-2 VHDL Code for LPM_MULT Design with 2 Pipelines Part 2



```vhdl

    p_mult : process(i_clk,i_rstb)
    begin
        if(i_rstb='0') then                            -- Reset everything
            o_xx <= (others=>'0');
            o_yy <= (others=>'0');
        elsif(rising_edge(i_clk)) then                 -- Process calculation
            o_xx <= std_logic_vector(('0'&(xx)) - yy);  -- Substraction to calculate real part of output
            o_yy <= std_logic_vector(xy & '0');        -- Multiplication to calculate imaginary part of output
        end if;
    end process p_mult;

end rtl;
```

Figure 4.1-3 VHDL Code for LPM_MULT Design with 2 Pipelines Part 3

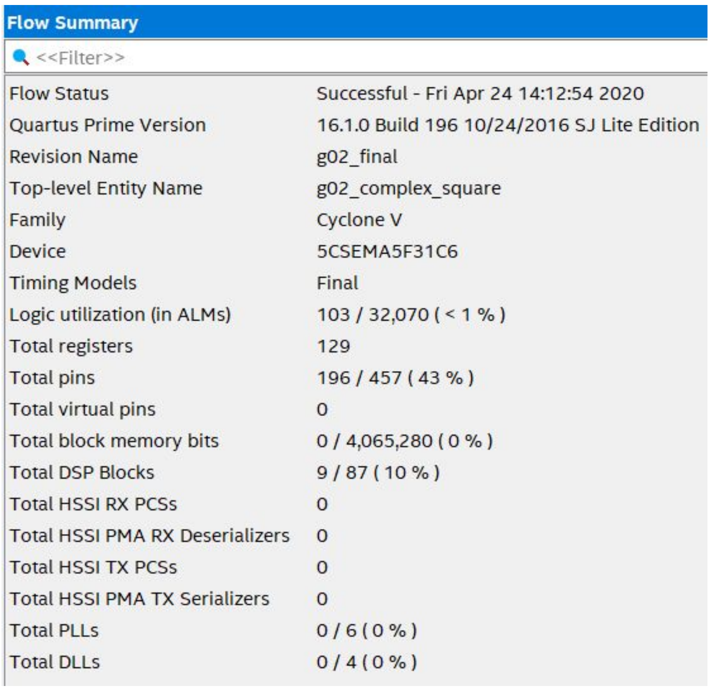
As we can see from figures above, we use the LPM_MULT component to enable the use of the pipeline registers in the DSP Blocks. In the library section, we load the LPM library and use all in the lpm.components. In the architecture section, we create three new 64-bits-signals to store the value calculated by multiplication while remove r_x and r_y since we will use i_x and i_y directly. Then, we declare the LPM_MULT component by following the code given in the lab manual while changing the output type for RESULT from std_logicc_vector to signed. We instantiate three LPM_MULT components in total for three multiplication operations we use in the computation system, naming mult1, mult2 and mult3 respectively. Finally, in the process part, we remove all statements related to r_x and r_y and replace multiplication operation by the result we get from the LPM multiplication component.

The VHDL code above is shown with 2 LPM pipelines implemented, which we run firstly. In the following parts we also run the code with the number of LPM pipelines modified to 3 and 4 by changing the number in the generic statements to see the circuit speed result.

4.2 LPM_MULT Design with 2 Pipelines

4.2.1 Resource Utilization

The flow summary of the square computation system design with 2 LPM pipelines implemented is shown in *Figure 4.2.1* below:



Flow Summary

<<Filter>>

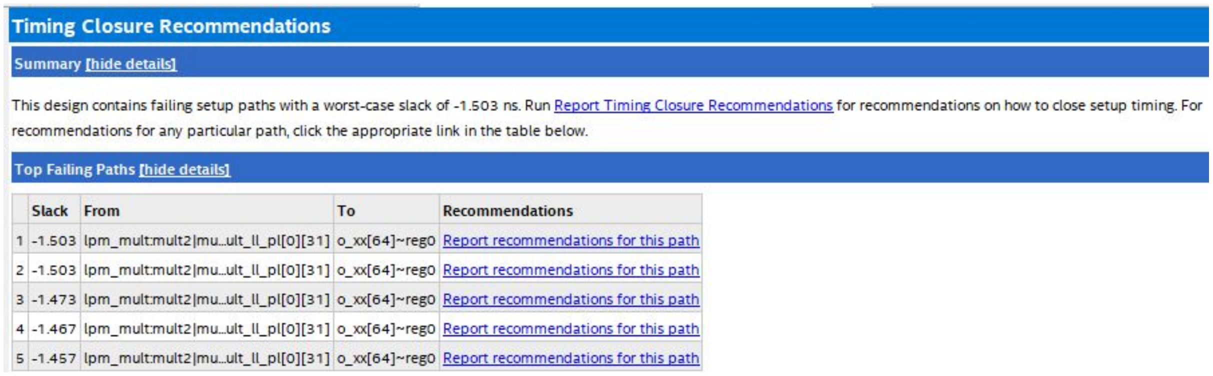
Flow Status	Successful - Fri Apr 24 14:12:54 2020
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	g02_final
Top-level Entity Name	g02_complex_square
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	103 / 32,070 (< 1 %)
Total registers	129
Total pins	196 / 457 (43 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	9 / 87 (10 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 4.2.1 Flow Summary for LPM_MULT Design with 2 Pipelines

We can see that there are 103 of 32070 ALMs used, showing logic utilization less than 1%. Total 129 registers are used and 9 of 87 DSP Blocks are used, taking 10% of the total DSP Blocks.

4.2.2 Timing Analysis by TimeQuest

From the TimeQuest Summary below we can see that for the square computation system design with 2 LPM pipelines, there are 5 timing violations in the timing analysis.



Timing Closure Recommendations

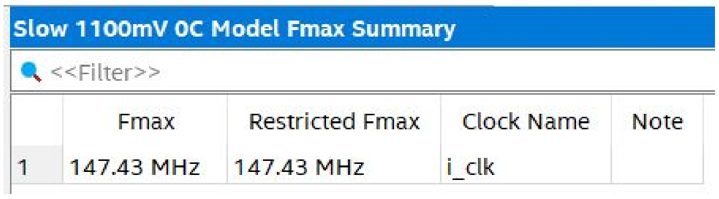
Summary [hide details]

This design contains failing setup paths with a worst-case slack of -1.503 ns. Run Report Timing Closure Recommendations for recommendations on how to close setup timing. For recommendations for any particular path, click the appropriate link in the table below.

Top Failing Paths [hide details]

	Slack	From	To	Recommendations
1	-1.503	lpm_mult:mult2|mu...ult_ll_pl[0][31]	o_xx[64]~reg0	Report recommendations for this path
2	-1.503	lpm_mult:mult2|mu...ult_ll_pl[0][31]	o_xx[64]~reg0	Report recommendations for this path
3	-1.473	lpm_mult:mult2|mu...ult_ll_pl[0][31]	o_xx[64]~reg0	Report recommendations for this path
4	-1.467	lpm_mult:mult2|mu...ult_ll_pl[0][31]	o_xx[64]~reg0	Report recommendations for this path
5	-1.457	lpm_mult:mult2|mu...ult_ll_pl[0][31]	o_xx[64]~reg0	Report recommendations for this path

Figure 4.2.2-1 TimeQuest Summary for LPM_MULT Design with 2 Pipelines

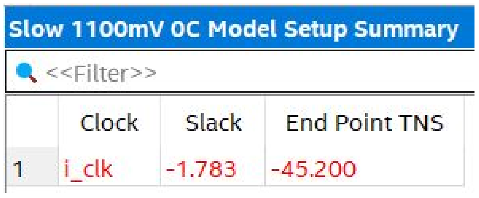


Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	147.43 MHz	147.43 MHz	i_clk	

Figure 4.2.2-2 Fmax Summary for LPM_MULT Design with 2 Pipelines



Slow 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	i_clk	-1.783	-45.200

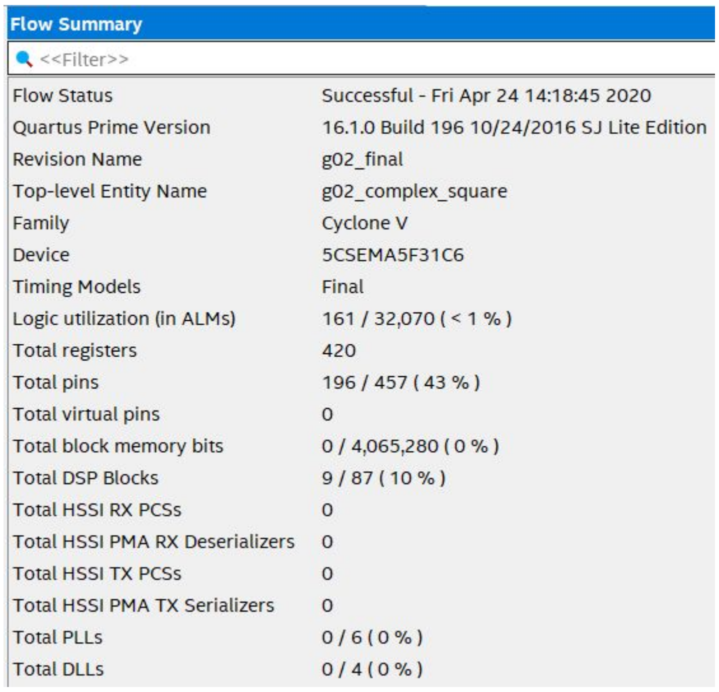
Figure 4.2.2-3 Setup Summary for LPM_MULT Design with 2 Pipelines

From two figures above we can see that for the “Slow 1100mv 0C Model” of the square computation system design with 2 LPM pipelines, the maximum frequency is 147.43MHz while i_clk slack is -1.783. Thus, this design does not pass the timing test.

4.3 LPM_MULT Design with 3 Pipelines

4.3.1 Resource Utilization

The flow summary of the square computation system design with 3 LPM pipelines implemented is shown in *Figure 4.3.1* below:



Flow Summary

<<Filter>>

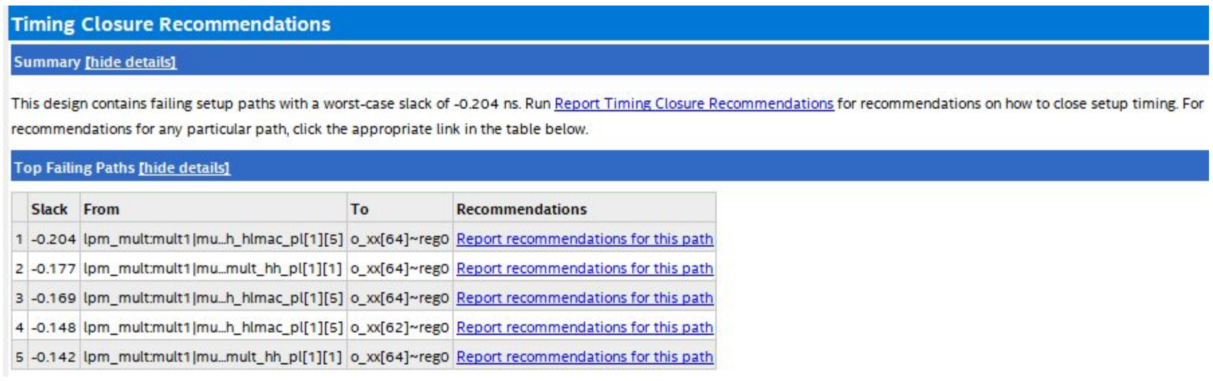
Flow Status	Successful - Fri Apr 24 14:18:45 2020
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	g02_final
Top-level Entity Name	g02_complex_square
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	161 / 32,070 (< 1 %)
Total registers	420
Total pins	196 / 457 (43 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	9 / 87 (10 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 4.3.1 Flow Summary for LPM_MULT Design with 3 Pipelines

We can see that there are 161 of 32070 ALMs used, showing logic utilization less than 1%. Total 420 registers are used and 9 of 87 DSP Blocks are used, taking 10% of the total DSP Blocks.

4.3.2 Timing Analysis by TimeQuest

From the TimeQuest Summary below we can see that for the square computation system design with 3 LPM pipelines, there are 5 timing violations in the timing analysis.



Timing Closure Recommendations

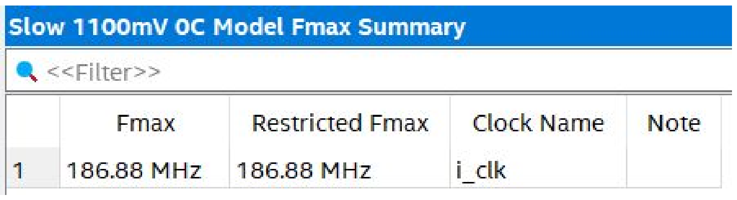
Summary [hide details]

This design contains failing setup paths with a worst-case slack of -0.204 ns. Run Report Timing Closure Recommendations for recommendations on how to close setup timing. For recommendations for any particular path, click the appropriate link in the table below.

Top Failing Paths [hide details]

	Slack	From	To	Recommendations
1	-0.204	lpm_mult:mult1|mu...h_hlmac_pl[1][5]	o_xx[64]~reg0	Report recommendations for this path
2	-0.177	lpm_mult:mult1|mu...mult_hh_pl[1][1]	o_xx[64]~reg0	Report recommendations for this path
3	-0.169	lpm_mult:mult1|mu...h_hlmac_pl[1][5]	o_xx[64]~reg0	Report recommendations for this path
4	-0.148	lpm_mult:mult1|mu...h_hlmac_pl[1][5]	o_xx[62]~reg0	Report recommendations for this path
5	-0.142	lpm_mult:mult1|mu...mult_hh_pl[1][1]	o_xx[64]~reg0	Report recommendations for this path

Figure 4.3.2-1 TimeQuest Summary for LPM_MULT Design with 3 Pipelines

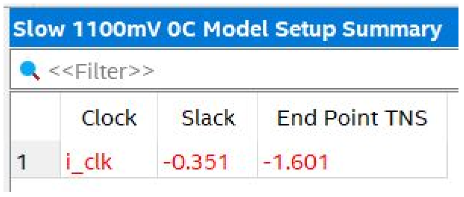


Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	186.88 MHz	186.88 MHz	i_clk	

Figure 4.3.2-2 Fmax Summary for LPM_MULT Design with 3 Pipelines



Slow 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	i_clk	-0.351	-1.601

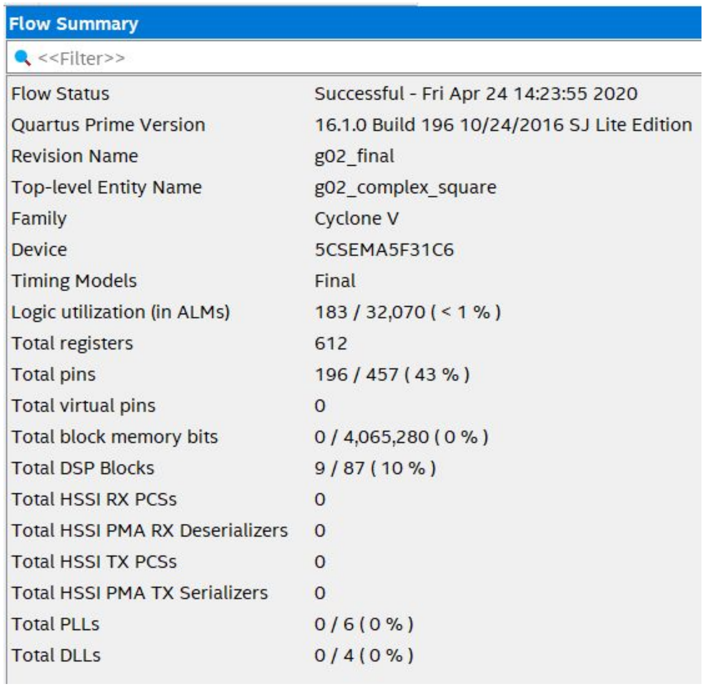
Figure 4.3.2-3 Setup Summary for LPM_MULT Design with 3 Pipelines

From two figures above we can see that for the “Slow 1100mv 0C Model” of the square computation system design with 3 LPM pipelines, the maximum frequency is 186.88MHz while i_clk slack is -0.351. Thus, this design nearly reaches our timing design requirements but still does not pass the timing test.

4.4 LPM_MULT Design with 4 Pipelines

4.4.1 Resource Utilization

The flow summary of the square computation system design with 4 LPM pipelines implemented is shown in *Figure 4.4.1* in the next page.



Flow Summary

<<Filter>>

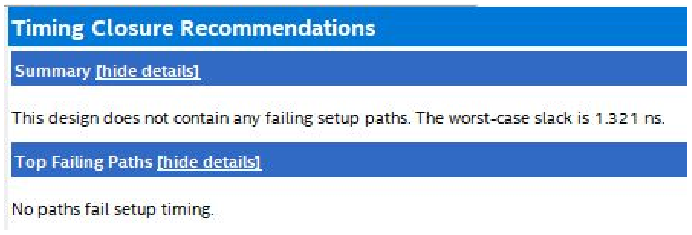
Flow Status	Successful - Fri Apr 24 14:23:55 2020
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	g02_final
Top-level Entity Name	g02_complex_square
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	183 / 32,070 (< 1 %)
Total registers	612
Total pins	196 / 457 (43 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	9 / 87 (10 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 4.4.1 Flow Summary for LPM_MULT Design with 4 Pipelines

We can see that there are 183 of 32070 ALMs used, showing logic utilization still less than 1%. Total 612 registers are used and 9 of 87 DSP Blocks are used, taking 10% of the total DSP Blocks.

4.4.2 Timing Analysis by TimeQuest

From the TimeQuest Summary below we can see that for the square computation system design with 4 LPM pipelines, there is no timing violations in the timing analysis.



Timing Closure Recommendations

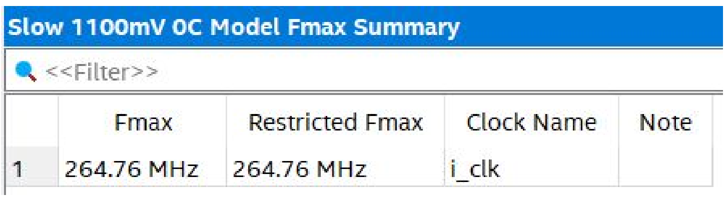
Summary [hide details]

This design does not contain any failing setup paths. The worst-case slack is 1.321 ns.

Top Failing Paths [hide details]

No paths fail setup timing.

Figure 4.4.2-1 TimeQuest Summary for LPM_MULT Design with 4 Pipelines

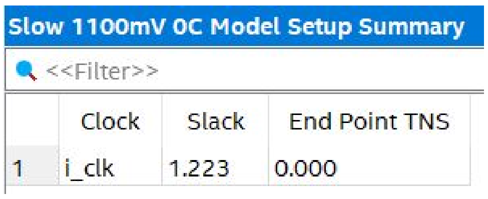


Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	264.76 MHz	264.76 MHz	i_clk	

Figure 4.4.2-2 Fmax Summary for LPM_MULT Design with 4 Pipelines



Slow 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	i_clk	1.223	0.000

Figure 4.4.2-3 Setup Summary for LPM_MULT Design with 4 Pipelines

From two figures above we can see that for the "Slow 1100mv 0C Model" of the square computation system design with 4 LPM pipelines, the maximum frequency reaches 264.76MHz, exceeding 200MHz, and the i_clk slack is 1.223. Thus, this design passes the timing test.

5. Resource Usage and Timing Analysis Summary

The FPGA resource usage and timing analysis summary for all designs listed above is concluded in the *Table 5* below:

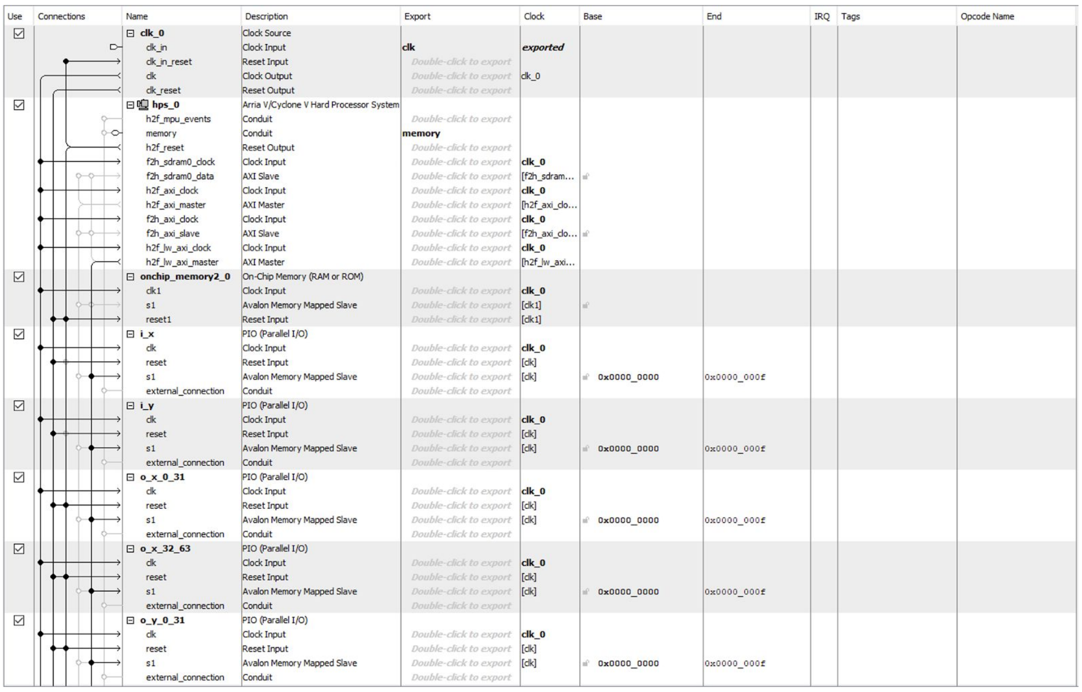
	No Pipelining	Pipelined	lpm_mult P=2	lpm_mult P=3	lpm_mult P=4
# ALMs	103	103	103	161	183
# Registers	65	65	129	420	612
# DSP Blocks	9	9	9	9	9
Fmax	110.83MHz	149.19MHz	147.43MHz	186.88MHz	264.76MHz
Slack	-4.023	-1.703	-1.783	-0.351	1.223

Table 5 FPGA Resource Usage and Timing Analysis Summary Table

From the table above we can see that without the implementation of the LPM pipeline, the maximum frequency increases from 110.83MHz to 149.19MHz with the use of the pipeline register. The slack in setup time also decreases from -4.023 to -1.703 after implementing the pipeline register. Meanwhile, the number of ALMs and the number of registers remain unchanged at 103 and 65 respectively. For the design with implementation of LPM pipelines, the resource usage increases significantly with the additional pipeline added that the number of ALMs grows from 103 to 183 and the number of registers grows from 129 to 612 with the number of pipelines set from 2 to 4. By the same time, the maximum frequency also reaches 147.43MHz, 186.88MHz and 264.76MHz with 2, 3 and 4 LPM pipelines implemented accordingly. The slack in setup time drops from -1.783 to -0.351 and reaches 1.223 with 4 LPM pipelines implemented. We see a growth trend in both resource usage and maximum frequency and downtrend in slack when more pipelines are used in the LPM_MULT component.

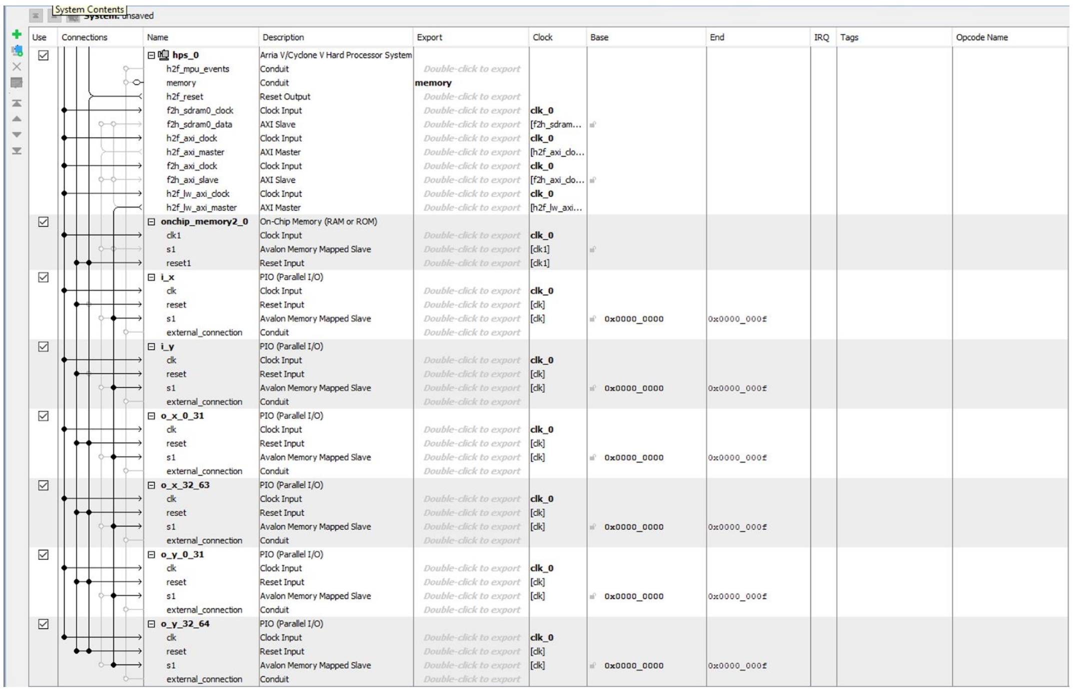
The result from the table above makes sense that with the implementation of pipeline, resource usage increases while the speed of the circuit also improves. By doing the pipelining using the LPM_MULT component with 4 pipelines, we achieve the timing goals of a 200MHz clock that the maximum frequency of our circuit reaches 264.76MHz.

6. Bonus: Connecting FPGA Logic to the Processor



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
☑		clk_0	Clock Source							
		clk_in	Clock Input	clk	exported					
		clk_in_reset	Reset Input	Double-click to export						
		clk	Clock Output	Double-click to export	clk_0					
		clk_reset	Reset Output	Double-click to export						
☑		hps_0	Arria V/Cyclone V Hard Processor System							
		h2f_mpu_events	Conduit	Double-click to export						
		memory	Conduit	memory						
		h2f_reset	Reset Output	Double-click to export						
		f2h_sdram0_clock	Clock Input	Double-click to export	clk_0					
		f2h_sdram0_data	AXI Slave	Double-click to export	[f2h_sdram...					
		h2f_axi_clock	Clock Input	Double-click to export	clk_0					
		h2f_axi_master	AXI Master	Double-click to export	[h2f_axi_clo...					
		f2h_axi_clock	Clock Input	Double-click to export	clk_0					
		f2h_axi_slave	AXI Slave	Double-click to export	[f2h_axi_clo...					
		h2f_lw_axi_clock	Clock Input	Double-click to export	clk_0					
		h2f_lw_axi_master	AXI Master	Double-click to export	[h2f_lw_axi...					
☑		onchip_memory2_0	On-Chip Memory (RAM or ROM)							
		clk1	Clock Input	Double-click to export	clk_0					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]					
		reset1	Reset Input	Double-click to export	[clk1]					
☑		i_x	PIO (Parallel I/O)							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f			
		external_connection	Conduit	Double-click to export						
☑		i_y	PIO (Parallel I/O)							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f			
		external_connection	Conduit	Double-click to export						
☑		o_x_0_31	PIO (Parallel I/O)							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f			
		external_connection	Conduit	Double-click to export						
☑		o_x_32_63	PIO (Parallel I/O)							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f			
		external_connection	Conduit	Double-click to export						
☑		o_y_0_31	PIO (Parallel I/O)							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f			
		external_connection	Conduit	Double-click to export						

Figure 6-1 System Connections part 1



System Contents
System: unsaved

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
☑		hps_0	Arria V/Cyclone V Hard Processor System							
		h2f_mpu_events	Conduit	Double-click to export						
		memory	Conduit	memory						
		h2f_reset	Reset Output	Double-click to export						
		f2h_sdram0_clock	Clock Input	Double-click to export	clk_0					
		f2h_sdram0_data	AXI Slave	Double-click to export	[f2h_sdram...					
		h2f_axi_clock	Clock Input	Double-click to export	clk_0					
		h2f_axi_master	AXI Master	Double-click to export	[h2f_axi_clo...					
		f2h_axi_clock	Clock Input	Double-click to export	clk_0					
		f2h_axi_slave	AXI Slave	Double-click to export	[f2h_axi_clo...					
		h2f_lw_axi_clock	Clock Input	Double-click to export	clk_0					
		h2f_lw_axi_master	AXI Master	Double-click to export	[h2f_lw_axi...					
☑		onchip_memory2_0	On-Chip Memory (RAM or ROM)							
		clk1	Clock Input	Double-click to export	clk_0					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]					
		reset1	Reset Input	Double-click to export	[clk1]					
☑		i_x	PIO (Parallel I/O)							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f			
		external_connection	Conduit	Double-click to export						
☑		i_y	PIO (Parallel I/O)							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f			
		external_connection	Conduit	Double-click to export						
☑		o_x_0_31	PIO (Parallel I/O)							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f			
		external_connection	Conduit	Double-click to export						
☑		o_x_32_63	PIO (Parallel I/O)							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f			
		external_connection	Conduit	Double-click to export						
☑		o_y_0_31	PIO (Parallel I/O)							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f			
		external_connection	Conduit	Double-click to export						
☑		o_y_32_64	PIO (Parallel I/O)							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f			
		external_connection	Conduit	Double-click to export						

Figure 6-2 System Connections part 2

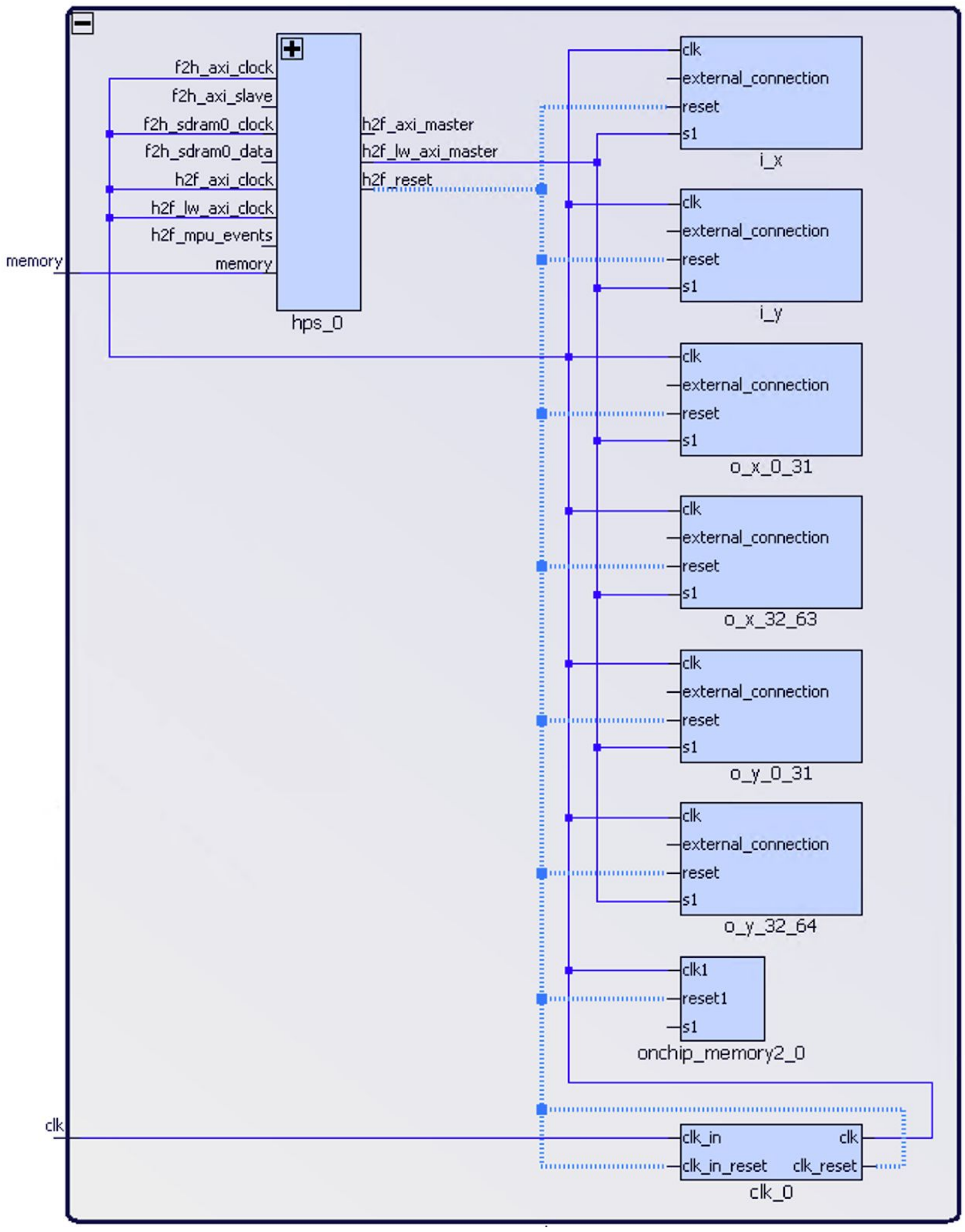


Figure 6-3 Schematic view

As shown in the *Figure 6-3* the schematic view. We can see that there are two inputs which are i_x and i_y each of them are 32 bits and we have 4 outputs o_y_0_31 means output y from 0 to 31 bits and o_y_32_63 means output y from 32 to 63 bits. We want both inputs and outputs of the FPGA circuit to be accessed by the processor so we need to add the clock and the s1 port of each one of them connected to the light weight master.

7. Conclusion

In this final project, we learned the basic knowledge of complex squaring computation in digital systems. We also explored the pipeline registers and the effects of the LPM_MULT component on the performance of synchronous digital systems. The TimeQuest timing output results from our computation system with the LPM_MULT component implemented using 4 pipelines comply with our estimation and satisfy the final project design requirements. Thus, we conclude that our circuit design for squaring a 32-bit complex number with 5ns timing constraint is successful.