# ECSE 325 Take-Home Exam Report

Pipelined Complex Multiplier and Design Closure

*Allan Reuben (260783324)*

*Noah Zwack (260769910)*

## Table of Contents

## 1 Introduction

For this assignment, we were asked to design and implement multiple versions of a complex squaring algorithm. We were asked to use different pipelining techniques and hardware features to increase the performance of our implementation, with a goal of reaching a clock speed of 200 MHz. In this report, we have documented our design choices and test results, which we will go over in detail.

## 2 Description of the Circuit Function

The function of the circuit that was given to us was to compute the square of a complex number. This complex number, call it $Z$, has a real and an imaginary part, $X$ and $Y$ respectively. To represent the real and imaginary parts of $Z$, we use two 32-bit input vectors. In addition to these inputs, we have an active low asynchronous reset input and the clock. There are also two 65-bit output vectors, which represent the real and imaginary parts of the result.

To calculate the real and imaginary parts, we use $Z^2 = (X + iY)(X + iY) = (X^2 - Y^2) + i(2XY)$. Therefore, the output vector for the real part will contain $X^2 - Y^2$, while the one for the imaginary part will contain $2XY$.

## 3 Basic Implementation

### 3.1 No Pipelining

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

entity g10_complex_square is
	port (
		i_clk : in std_logic;
		i_rstb : in std_logic;
		i_x : in std_logic_vector(31 downto 0);
		i_y : in std_logic_vector(31 downto 0);
		o_xx, o_yy : out std_logic_vector(64 downto 0)
```

```vhdl
	);
end g10_complex_square;

architecture rtl of g10_complex_square is

signal r_x, r_y : signed(31 downto 0);

begin

	p_mult : process(i_clk,i_rstb)
	begin
		if(i_rstb='0') then
			o_xx <= (others => '0');
			o_yy <= (others => '0');
			r_x <= (others => '0');
			r_y <= (others => '0');
		elsif(rising_edge(i_clk)) then
			r_x <= signed(i_x);
			r_y <= signed(i_y);
			o_xx <= std_logic_vector(('0' & (r_x * r_x)) - r_y * r_y);
			o_yy <= std_logic_vector(r_x * r_y & '0');
		end if;
	end process p_mult;

end rtl;
```

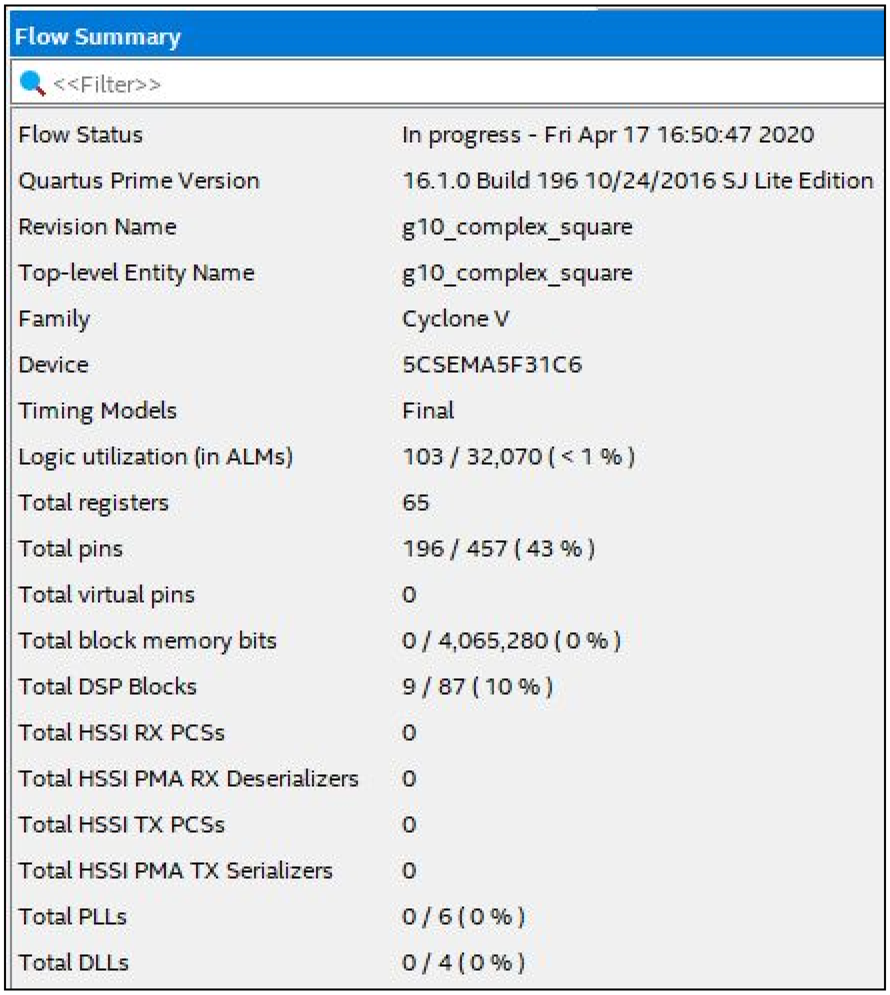
Figure 1: The non-pipelined basic implementation of the circuit function

For the first implementation of the circuit function, there is not much pipelining going on. The multiplication of the input signals and the subtraction of these signals is all happening in one clock cycle. The only pipelining that is present in this description is the assignment of the input signals to r_x and r_y.

Figure 2 shows the code used in the .sdc file for the TimeQuest analysis. It mandates that the clock frequency be at least 200 MHz. Below that, figures 3-5 show the results from compilation and the timing analysis. While our resource utilization is quite low, using only 65 registers, our maximum attainable frequency is only 110.83 MHz, nearly half of what we want.

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

Figure 2: Contents of the timing constraints (.sdc) file

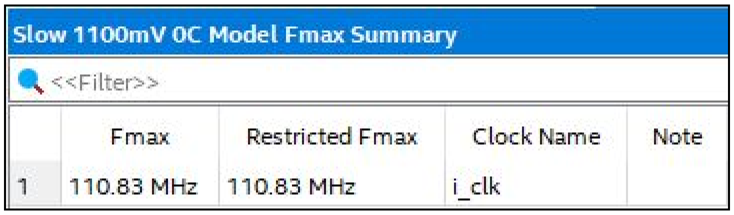


**Flow Summary**

<<Filter>>

| | |
|---|---|
| Flow Status | In progress - Fri Apr 17 16:50:47 2020 |
| Quartus Prime Version | 16.1.0 Build 196 10/24/2016 SJ Lite Edition |
| Revision Name | g10_complex_square |
| Top-level Entity Name | g10_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 % ) |

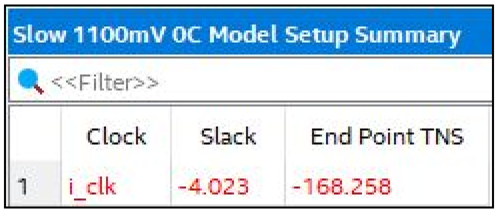
(3)



**Slow 1100mV 0C Model Fmax Summary**

<<Filter>>

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



**Slow 1100mV 0C Model Setup Summary**

<<Filter>>

| | Clock | Slack | End Point TNS |
|---|---|---|---|
| 1 | i_clk | -4.023 | -168.258 |

(4) (5)

Figures 3-5: The resource utilization (3), Fmax (4), and slack (5) for the basic, non-pipelined implementation

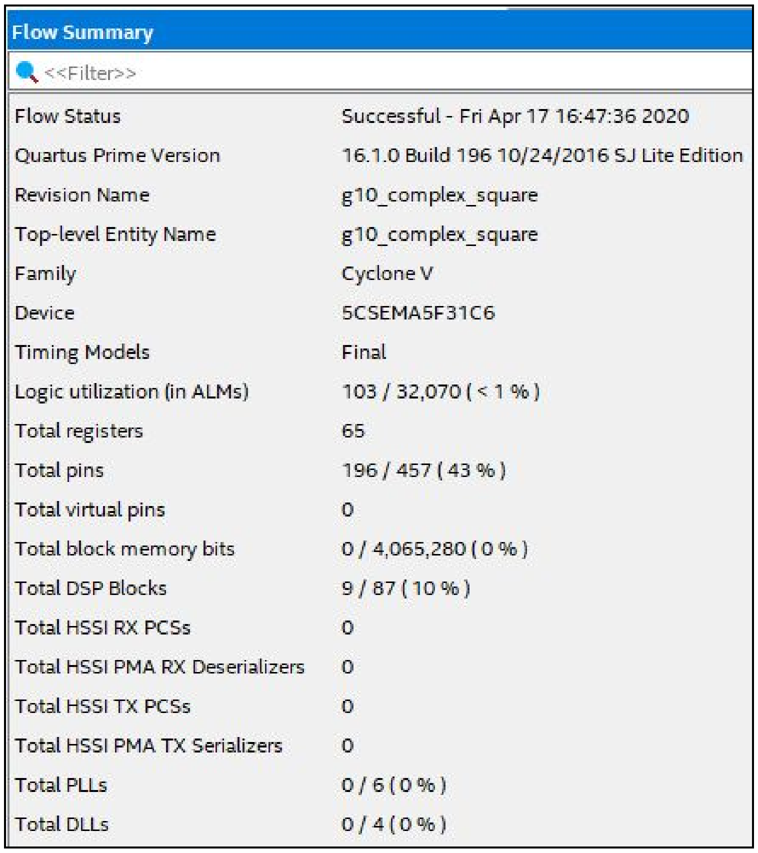
## 3.2 Pipelining x * x and y * y

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

entity g10_complex_square is
	port (
		i_clk : in std_logic;
		i_rstb : in std_logic;
		i_x : in std_logic_vector(31 downto 0);
		i_y : in std_logic_vector(31 downto 0);
		o_xx, o_yy : out std_logic_vector(64 downto 0)
	);
end g10_complex_square;

architecture rtl of g10_complex_square is

signal r_x, r_y     : signed(31 downto 0);
signal r_xx, r_yy   : signed(63 downto 0);

begin

	p_mult : process(i_clk,i_rstb)
	begin
		if(i_rstb='0') then
			o_xx <= (others => '0');
			o_yy <= (others => '0');
			r_x <= (others => '0');
			r_y <= (others => '0');
			r_xx <= (others => '0');
			r_yy <= (others => '0');
		elsif(rising_edge(i_clk)) then
			r_x <= signed(i_x);
			r_y <= signed(i_y);
			r_xx <= r_x * r_x;
			r_yy <= r_y * r_y;
			o_xx <= std_logic_vector(('0' & r_xx) - r_yy);
			o_yy <= std_logic_vector((r_x * r_y) & '0');
		end if;
	end process p_mult;

end rtl;
```

Figure 6: The pipelined version of the basic implementation of the circuit function

In the second implementation of the circuit function, we were asked to pipeline the multiplication between the subtraction operation. So, we added two registers that contain the $X^2$ and $Y^2$ operations. This pipelining should allow us to run the clock faster than the previous implementation.

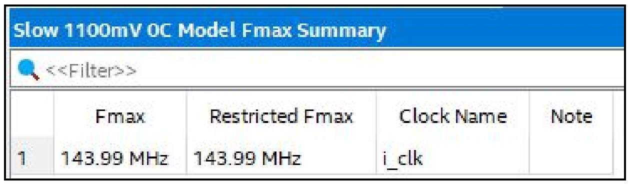
Figures 7 through 9 show the results of the compilation and timing analysis. As expected, there is a rather large increase in the maximum frequency, however, our resource utilization has not changed from the non-pipelined version. This is very surprising, as we would expect additional registers to be present in order to reflect the changes in the design. Upon examining the RTL Viewer, we saw the number of registers we were expecting (i.e. two additional stacks of registers in the pipelined version). After some investigation, it appears that, before the Fitter, the pipelined design is indeed using more registers. However, the Fitter is optimizing the number of registers used in the final design, and  has found a way to compress the design down to 65 registers, while still maintaining a faster clock speed.


Flow Summary

<<Filter>>

| | |
|---|---|
| Flow Status | Successful - Fri Apr 17 16:47:36 2020 |
| Quartus Prime Version | 16.1.0 Build 196 10/24/2016 SJ Lite Edition |
| Revision Name | g10_complex_square |
| Top-level Entity Name | g10_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 % ) |

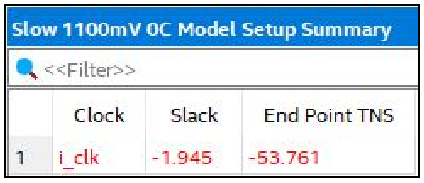
(7)



Slow 1100mV 0C Model Fmax Summary

| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 143.99 MHz | 143.99 MHz | i_clk | |

(8)



Slow 1100mV 0C Model Setup Summary

| | Clock | Slack | End Point TNS |
|---|---|---|---|
| 1 | i_clk | -1.945 | -53.761 |

(9)

Figures 7-9: The resource utilization (7), Fmax (8), and slack (9) for the basic, pipelined implementation

## 4 LPM Implementation

```
library IEEE;
library lpm;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use lpm.lpm_components.all;

entity g10_complex_square is
	port (
		i_clk : in std_logic;
		i_rstb : in std_logic;
		i_x : in std_logic_vector(31 downto 0);
		i_y : in std_logic_vector(31 downto 0);
		o_xx, o_yy : out std_logic_vector(64 downto 0)
	);
end g10_complex_square;

architecture rtl of g10_complex_square is
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 std_logic_vector(LPM_WIDTHP-1 downto 0)
	);
```

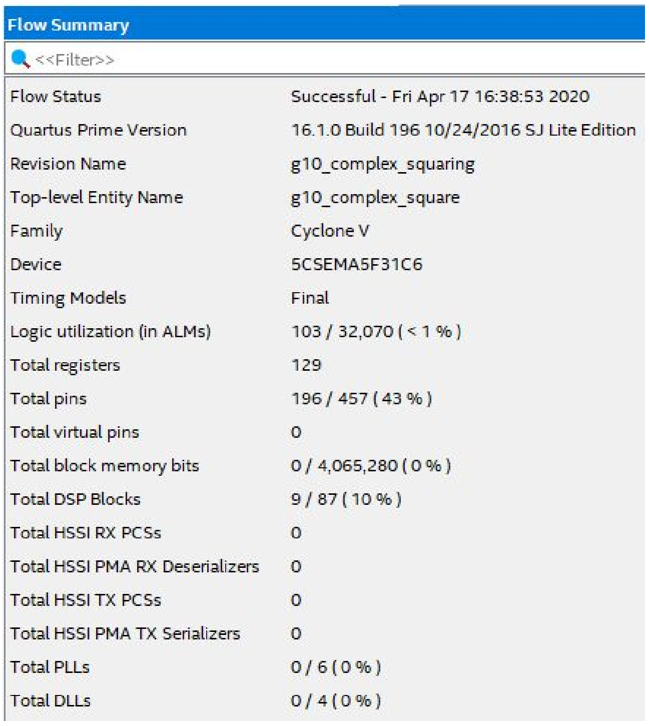
```
end component;

signal r_xx, r_yy, r_xy : std_logic_vector(63 downto 0);
constant pipeline_value   : integer := 2;

begin
       mult1 : LPM_MULT generic map (
              LPM_WIDTHA => 32,
              LPM_WIDTHB => 32,
              LPM_WIDTHP => 64,
              LPM_REPRESENTATION => "SIGNED",
              LPM_PIPELINE => pipeline_value
       )
       port map ( DATAA => i_x, DATAB => i_x, CLOCK => i_clk, RESULT => r_xx );

       mult2 : LPM_MULT generic map (
              LPM_WIDTHA => 32,
              LPM_WIDTHB => 32,
              LPM_WIDTHP => 64,
              LPM_REPRESENTATION => "SIGNED",
              LPM_PIPELINE => pipeline_value
       )
       port map ( DATAA => i_y, DATAB => i_y, CLOCK => i_clk, RESULT => r_yy );

       mult3 : LPM_MULT generic map (
              LPM_WIDTHA => 32,
              LPM_WIDTHB => 32,
              LPM_WIDTHP => 64,
              LPM_REPRESENTATION => "SIGNED",
              LPM_PIPELINE => pipeline_value
       )
       port map ( DATAA => i_x, DATAB => i_y, CLOCK => i_clk, RESULT => r_xy );

       p_mult : process(i_clk,i_rstb)
       begin
              if(i_rstb='0') then
                     o_xx <= (others => '0');
                     o_yy <= (others => '0');
              elsif(rising_edge(i_clk)) then
                     o_xx <= std_logic_vector(signed('0' & r_xx) - signed(r_yy));
                     o_yy <= (r_xy & '0');
              end if;
       end process p_mult;
end rtl;
```

Figure 10: The LPM module implementation of the circuit function

In the final implementation of the circuit function, we used the LPM_MULT components instead of the * operator for multiplying the two signals. This is because the board we are using has dedicated DSP blocks designed for fast multiplication. Therefore, we should be able to pipeline our design even further and achieve an even faster clock speed.
In our VHDL description, we have created an instance of the LPM_MULT component for each multiplication that needs to be calculated, namely $X^2$, $Y^2$, and $XY$. We have also created a variable called pipeline_value, which we incremented after each compilation. This increases the level of pipelining that's happening in the multiplier, thereby increasing the overall clock speed.

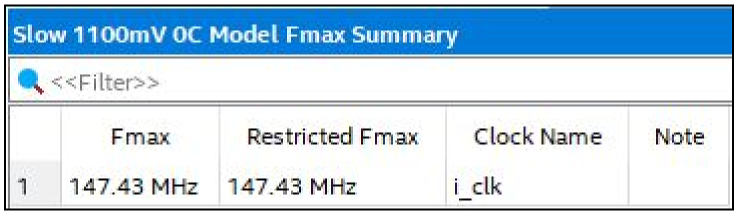
## 4.1 Pipeline Level 2

Figures 11 through 13 show the compilation report and timing analysis of the LPM implementation with the pipeline level set to 2. We can see from Figure 11 that the number of registers has almost doubled from the basic pipelined implementation. In addition, the maximum frequency has gone up slightly, but is still far from our target clock frequency of 200 MHz.



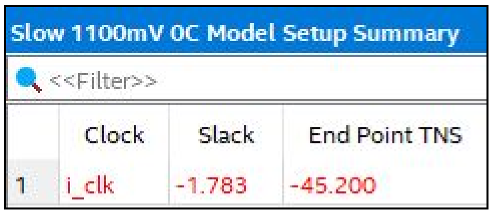
| | |
|---|---|
| Flow Status | Successful - Fri Apr 17 16:38:53 2020 |
| Quartus Prime Version | 16.1.0 Build 196 10/24/2016 SJ Lite Edition |
| Revision Name | g10_complex_squaring |
| Top-level Entity Name | g10_complex_square |
| Family | Cyclone V |
| Device | 5CSEMA5F31C6 |
| Timing Models | Final |
| 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 % ) |

(11)



Slow 1100mV 0C Model Fmax Summary

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



Slow 1100mV 0C Model Setup Summary

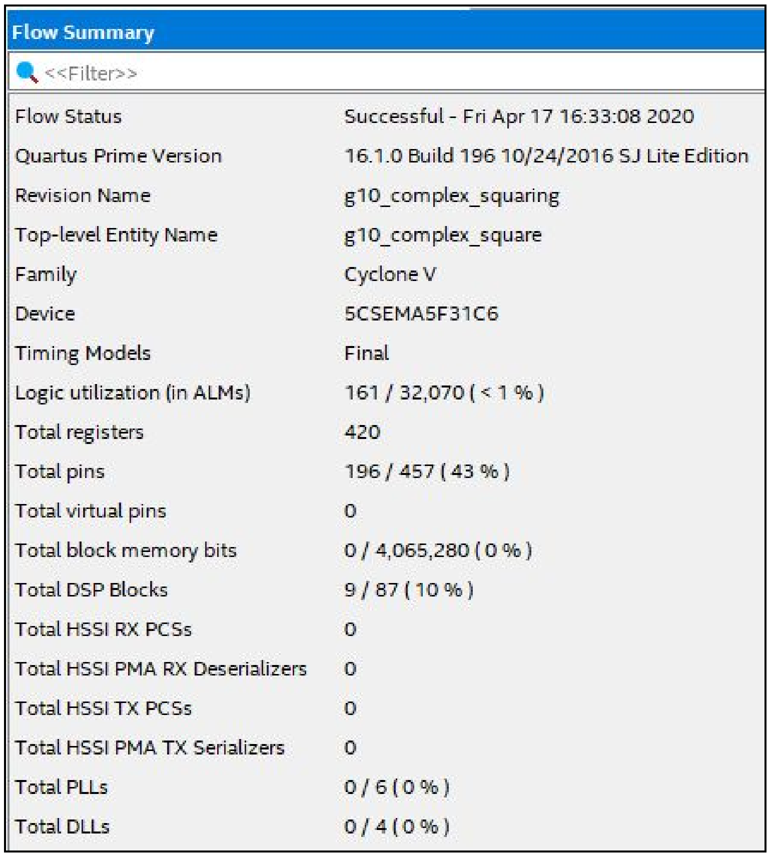
| | Clock | Slack | End Point TNS |
|---|---|---|---|
| 1 | i_clk | -1.783 | -45.200 |

(12) (13)

Figures 11-13: The resource utilization (11), Fmax (12), and slack (13) for the LPM implementation with the pipeline at level 2

## 4.2 Pipeline Level 3

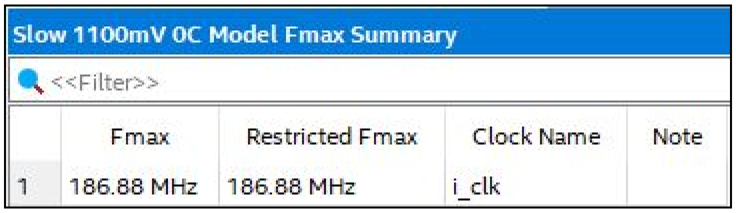
Figures 14-16 show the results for setting the pipeline level to 3. Our register count has increased significantly, as well as the number of ALM's. In addition, our maximum clock frequency has increased dramatically, although it's still below our desired frequency of 200 MHz. Clearly, when we increase the level of pipelining, our resource utilization increases significantly, but our performance also increases.



Flow Summary

| | |
|---|---|
| Flow Status | Successful - Fri Apr 17 16:33:08 2020 |
| Quartus Prime Version | 16.1.0 Build 196 10/24/2016 SJ Lite Edition |
| Revision Name | g10_complex_squaring |
| Top-level Entity Name | g10_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 % ) |

(14)

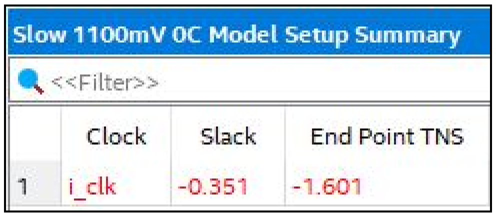


Slow 1100mV 0C Model Fmax Summary

<<Filter>>

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

(15)



Slow 1100mV 0C Model Setup Summary

<<Filter>>

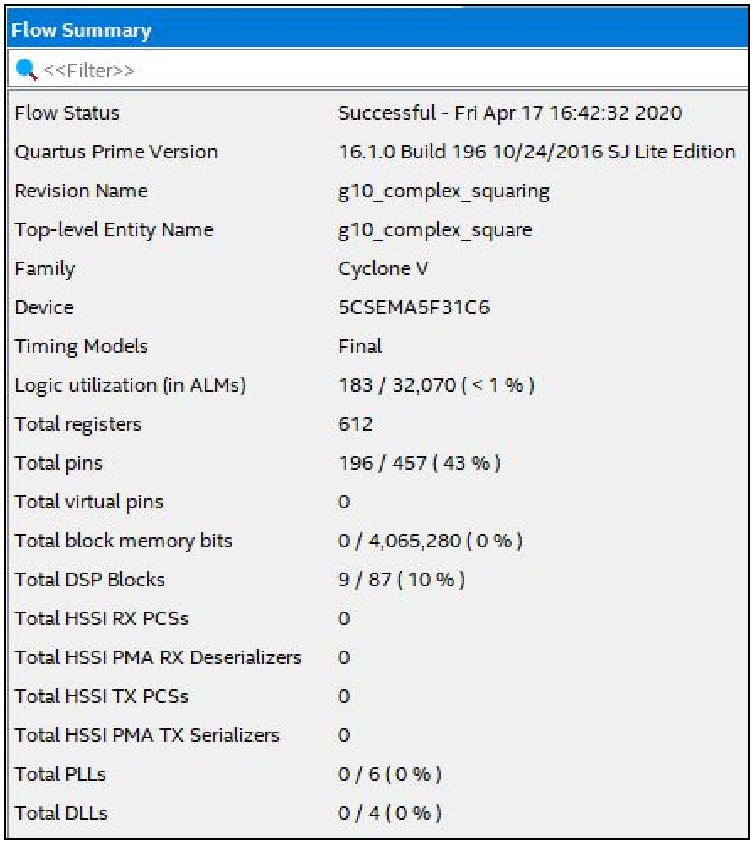
| | Clock | Slack | End Point TNS |
|---|---|---|---|
| 1 | i_clk | -0.351 | -1.601 |

(16)

Figures 14-16: The resource utilization (14), Fmax (15), and slack (16) for the LPM implementation with the pipeline at level 3

## 4.3 Pipeline Level 4

Figures 17-19 show the results for setting the pipeline level to 4. Once again, our resource utilization has increased, though not by as much as when the pipeline level went from 2 to 3. However, our frequency is now 264.76 MHz (Figure 18), which is well above our desired frequency of 200 MHz. In addition, the slack of the setup time is positive for the first time, meaning we can have room to slow down the clock if we need to.

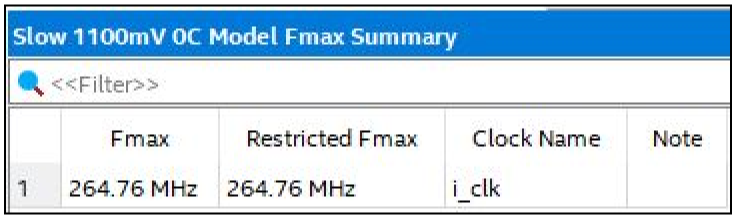


Flow Summary

<<Filter>>

| | |
|---|---|
| Flow Status | Successful - Fri Apr 17 16:42:32 2020 |
| Quartus Prime Version | 16.1.0 Build 196 10/24/2016 SJ Lite Edition |
| Revision Name | g10_complex_squaring |
| Top-level Entity Name | g10_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 % ) |

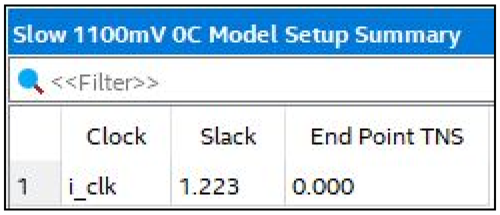
(17)



Slow 1100mV 0C Model Fmax Summary

<<Filter>>

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



Slow 1100mV 0C Model Setup Summary

<<Filter>>

| | Clock | Slack | End Point TNS |
|---|---|---|---|
| 1 | i_clk | 1.223 | 0.000 |

(18) (19)

Figures 17-19: The resource utilization (17), Fmax (18), and slack (19) for the LPM implementation with the pipeline at level 4

## 4.4 Bonus Attempt

To connect the FPGA circuitry as a complex number coprocessor with the ARM processor inside the chip, we understood that we could use two ways to attach the logic. The first would be to use the Avalon Streaming interface to realize a one way connection, with the processor providing the data to the complex number multiplier. For a more complex connection with both the inputs and outputs of the FPGA accessed by the processor, we could have used the Avalon Memory Mapped Interface. This would have allowed for us to implement the more complex Mandelbrot fractal generation that was shown in class.

## 5 Resource and Timing Table

| | **No Pipelining** | **Pipelined** | **LPM_MULT P = 2** | **LPM_MULT P = 3** | **LPM_MULT P = 4** |
|---|---|---|---|---|---|
| **# of ALM’s** | 103 | 103 | 103 | 161 | 183 |
| **# of Registers** | 65 | 65 | 129 | 420 | 612 |
| **# of DSP Blocks** | 9 | 9 | 9 | 9 | 9 |
| **Fmax (MHz)** | 110.83 | 143.99 | 147.43 | 186.88 | 264.76 |
| **Slack of setup time (ns)** | -4.023 | -1.945 | -1.783 | -0.351 | +1.223 |

The analysis of the contents of the table has been discussed throughout the report, but will be summarized here. Going from left to right, the resource utilization consistently increases from the second column onwards. This is best demonstrated by the register count, which increases from 65 to 612. This is not good as a designer, as we want to use as little resources as possible. However, the more resources we use, the better our performance is, as shown by the increasing maximum clock frequency and slack. This shows us the trade-off between resources and performance.

## 6 Conclusion

In conclusion, using the LPM_MULT component, with the pipeline level set to 4, it is possible to achieve a clock speed of 200 MHz with a positive setup slack. However, in doing so, we are using the most amount of resources when compared to the alternatives that were tried.