ELEC6233 – FPGA complex multiplier synthesis

YEQIU TANG
yt2n16
MSc Embedded Systems
Dr Tom J Kazmierski

ABSTRACT: *A multiplier of two complex numbers with a simple sequential controller is presented. The design of the multiplier and sequential controller are discussed. The simulation results show this multiplier works properly to calculate the product of two complex numbers. And the multiplier is running properly on Altera DE0 development board. By this assignment, a clear understanding of complex multiplier is gained and what benefits the learning of Digital System Synthesis.*

1. Introduction

The objective of this assignment is to design, in combinational logic, a multiplier of two complex numbers:

$$p = \alpha \cdot q$$

Where $\alpha = Re\alpha + jIm\alpha$ with the real and imaginary part represented as signed 8-bit 2's complement numbers in the range -1..+1-2^{-8} and $q = Req + jImq$ with the real and imaginary part represented as signed 8-bit 2's complement integers in the range -128..127. The output from the multiplier $p = Rep + jImp$ is required to preserve only the 8-bit integer part of the calculation.

In addition, in the equation $p = Rep + jImp$, where $Rep = Re\alpha \times Req - Im\alpha \times Imq$ and $Imp = Re\alpha \times Imq + Im\alpha \times Req$, it is obviously there are four multiplications, one addition and one subtraction. Therefore, in order to implement the multiplier as combinational logic, four combinational multipliers are needed in this design.

Such calculations are frequently required in systems which implement Fourier transform algorithms and are used in many digital signal-processing applications.

2. Multiplier design, simulation and synthesis

In terms of the requirement, the multiplier must be combinational. Thus the multiplier is designed to use a multi-stage pipeline to add the result of the two adjacent products to the final output product, which is arranged in a binary tree structure. It requires the $\log_2 N$ level for the N-bit multiplier achieve, N is 8 in this design thus it contains 3-stage pipeline. The source code is in ZIP file.

The block diagram of this multiplier is shown below:

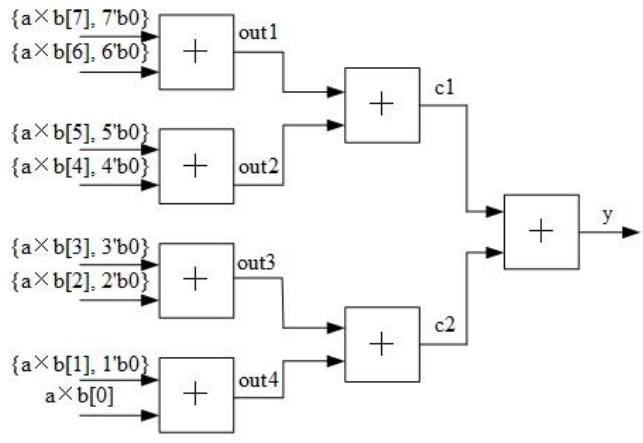

Figure 1 Block diagram of 8x8 multiplier

The multiplicand "a" is signed 8-bit 2's complement numbers in the range -1..+1-2^{-8} and the multipicator "b" is signed 8-bit 2's complement integers in the range -128..127 thus the product "y" should be 8-bit × 8-bit = 16-bit. According to the requirement that only the 8-bit integer of "y" is needed so the result is "y [14:7]".

The shortcoming of this design is that only positive (unsigned) number could be calculated. The solution is to use the absolute values of "a" and "b" to get the temporary result. Then analyse the sign of the result and output the correct value.

In other word, detect the bit 7 of both “a” and “b”. If the bit 7 is 1 which means the number is negative, convert it to its complement to make the number positive. By this way, both “a” and “b” would be positive in calculation and the result must be positive. Then convert the result to its complement only if the bit 7 (sign) of “a” and “b” are different (one of “a” and “b” is negative while the other is positive). In this method, the value and sign of result would always be correct.

In order to test the ability of this multiplier, a table of test numbers are designed, shown below:

a	b	y (theoretical)
-0.5 (8’b11000000)	7 (8’b00000111)	-3.5
0.5 (8’b01000000)	17 (8’b00010001)	8.5
-0.25 (8’b11100000)	-21 (8’b11101011)	5.25
0.5 (8’b01000000)	-18 (8’b11101110)	-9

Table 1 Test table for 8-bit multiplier

Then write the test bench according to this table and check the simulation waveform. The source code of test bench is in ZIP file.

The result waveform is shown below:

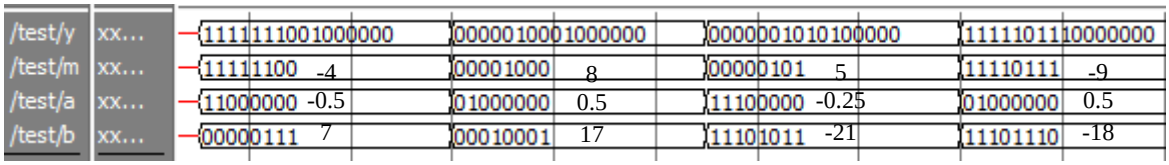

Figure 2 Result of multiplier simulation

In the simulation waveform, the data format of “a” is 2’s complement signed fix-point fraction, while “b” is 2’s complement signed integer. “m” and “y” are the results of calculation.

To make the simulation waveform correctly show the result needed, set “m” equal to “y [14:7]”. It is clearly shown that the four multiplication results “m” are -4 (8’b11111100), 8 (8’b00001000), 5 (8’b00000101) and -9 (8’b11110111), which are slightly smaller than the theoretical results. That is because when the full 16-bit result is truncated into 8-bit, the fraction part of result is entirely discarded.

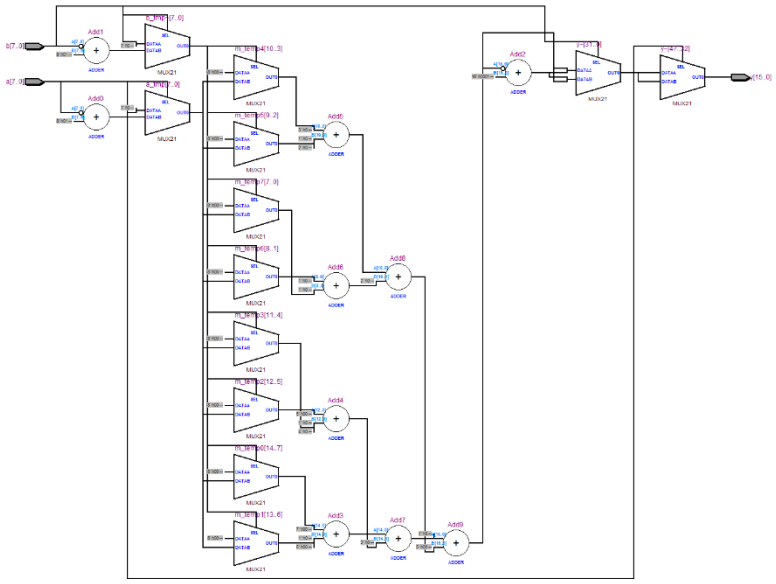

Figure 3 Synthesis of 8x8 multiplier

According to the simulation result, the multiplier works properly.

The figure at left is shown the synthesis result of this multiplier. The operation “a×b[n]” is implemented by a multiplexer.

Moreover, the input number “a” and “b” will be converted into absolute values before calculating, and also the sign of result “y” will be calibrated before outputting. The structure of synthesis result is nearly the same as the block diagram.

3. Complex number multiplier design, simulation and synthesis

According to the equation $p = \alpha \times q = Rep + jImp$, where $Rep = Re\alpha \times Req - Im\alpha \times Imq$ and $Imp = Re\alpha \times Imq + Im\alpha \times Req$, the product of "$\alpha$" and "$q$" is derived from four multiplications, one addition and one subtraction, while addition and subtraction are all performed by adder. The source code is in ZIP file.

The block diagram of this complex number multiplier design is shown below:

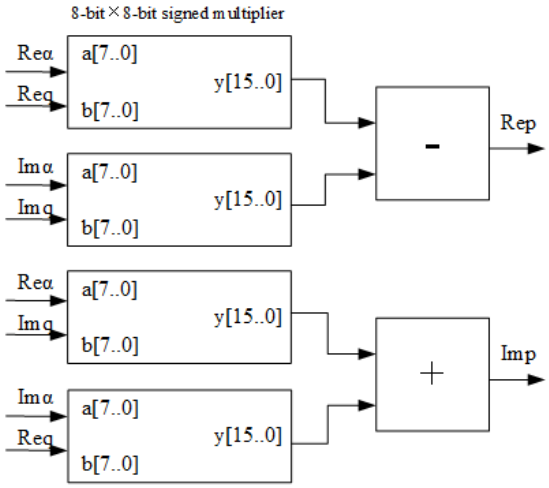

Figure 4 Block diagram of Complex number multiplier

This complex number multiplier module contains four multipliers and two adders (one of them work as subtractor), with four inputs and two outputs.

In addition, the four input number will be connected to the multiplier in pairs in order to make the whole design fully combinational.

As a result, by inputting all of the two complex number (four decimal numbers), the results of real part and imaginary part will be delivered directly. Moreover, the results "Rep" and "Imp" will be converted to 8-bit (bit 14 to bit 7).

To verify if this complex number multiplier works properly, a test table contains a list of complex numbers need to be used.

The table is shown below:

$Re\alpha$	$Im\alpha$	Req	Imq	Rep(theoretical)	Imp(theoretical)
0.5 (8'b01000000)	0.5 (8'b01000000)	17 (8'b00010001)	28 (8'b00011100)	-5.5	22.5
-0.5 (8'b11000000)	0.5 (8'b01000000)	10 (8'b00001010)	70 (8'b01000110)	-40	-30
0.25 (8'b00100000)	-0.5 (8'b11000000)	12 (8'b00001100)	-20 (8'b11101100)	-7	-11
-0.75 (8'b10100000)	0.25 (8'b00100000)	-100 (8'b10011100)	-55 (8'b11001001)	88.75	16.25

Table 2 Test table for complex number multiplier

Then write a test bench to input those numbers into the complex number multiplier. The source code is in ZIP file and the simulation results are shown below:

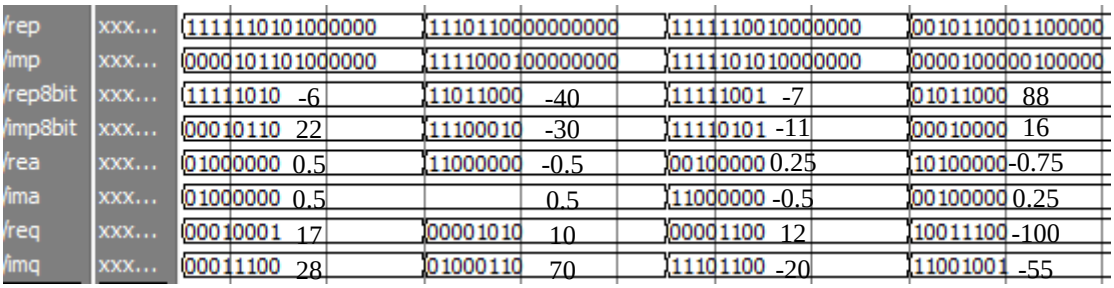

Figure 5 Result of complex number multiplier simulation

In the simulation, the "rea" and "ima" are the parts of "α", and the "req" and "imq" are the parts of "q". Moreover, the "rep" and "imp" are the parts of result p. Because only 8-bit integer results are needed, only bit 14 to bit 7 of the results are taken, represented as "rep8bit" and "imp8bit".

In addition, from the simulation waveform the results are slightly smaller than theoretical values, which is because the fraction parts are entirely discarded.

The synthesis of complex number multiplier is shown below:

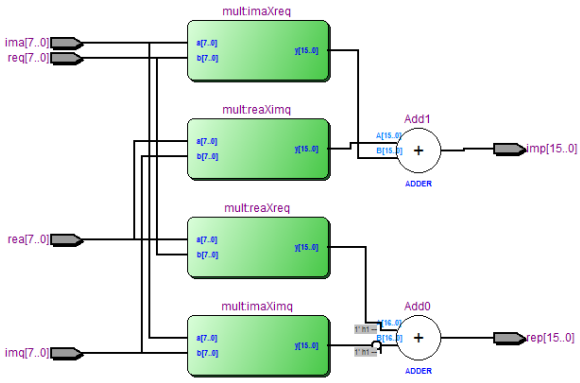

Figure 6 Synthesis of complex number multiplier

There are four 8x8 signed multipliers and two adders in this module. One of the adder performs subtraction.

Moreover, it is clearly shown in the figure that the output "imp" an "rep" are both 16-bit number that is in order to keep the maximum accuracy. They would be discarded the fraction part and converted into 8-bit format when finally outputting from LEDs.

4. Sequential logic controller design

A Finite State Machine is designed as the controller to read the operands from the switches and display the results on the LEDs. The 8-bit operands are read from SW7 – SW0 and the results are displayed on LED7 – LED0, while SW8 provides handshaking functionality and SW9 is active low reset of the controller. The source code is in ZIP file.

The block diagram is shown below:

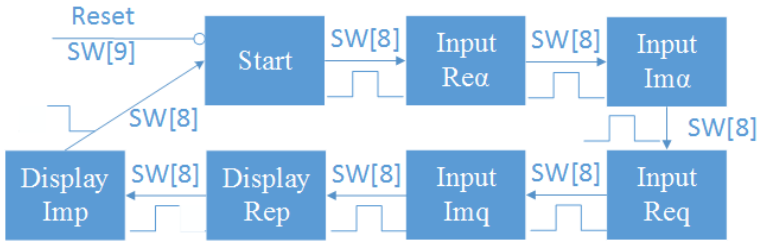

Figure 7 Block diagram of controller

The synthesis of the whole system is shown below:

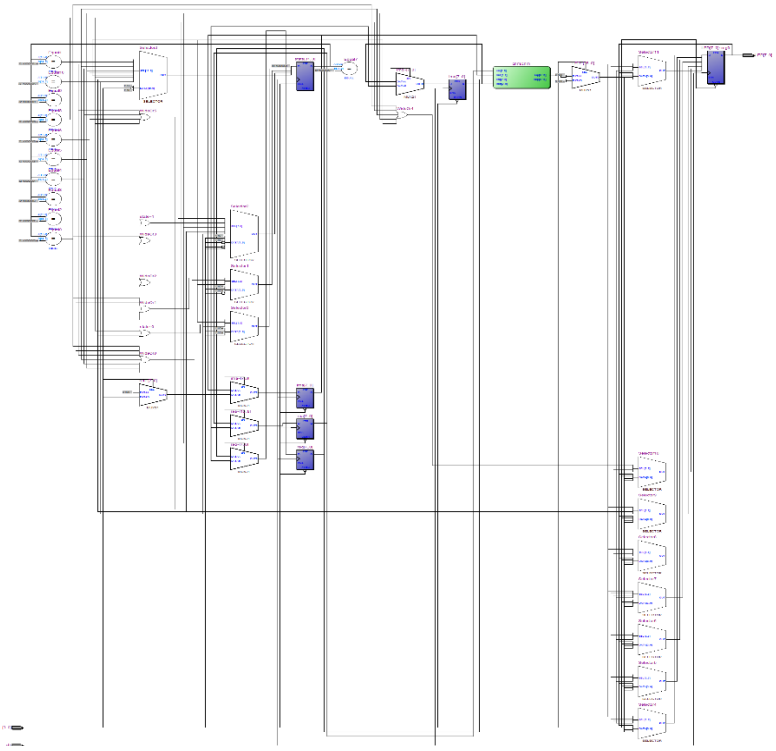

Figure 8 Synthesis of the whole system

It is shown in the schematic diagram there are four registers used to read and memorize the four input numbers and two registers are used to display the final result.

In addition, the LEDs will display each of the input numbers in case of the wrong input.

The performance of this sequential controller will be tested in the implementation on the Altera DE0 development board.

5. Altera DE0 implementation

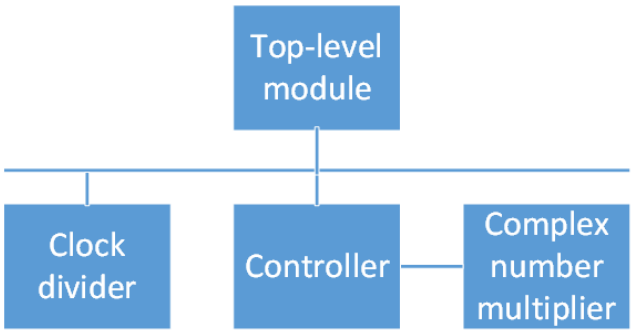

Figure 9 System structure

To implement on the Altera DE0 development board, in order to eliminate bouncing effects of the mechanical switches, an extra clock divider is added to supply a 10 Hz clock signal to sample the signal from SW [8]. The whole system structure is shown below:

After programming the FPGA, there should be a test to evaluate the performance and accuracy of this complex number multiplier in real device.

The Table 2 in section 3 can be used to test it. The table below is used to record the results of four groups of test numbers.

$Re\alpha$	$Im\alpha$	Req	Imq	Rep(actual)	Imp(actual)
0.5 (8'b01000000)	0.5 (8'b01000000)	17 (8'b00010001)	28 (8'b00011100)	8'b11111010	8'b00010110
-0.5 (8'b11000000)	0.5 (8'b01000000)	10 (8'b00001010)	70 (8'b01000110)	8'b11011000	8'b11100010
0.25 (8'b00100000)	-0.5 (8'b11000000)	12 (8'b00001100)	-20 (8'b11101100)	8'b11111001	8'b11110101
-0.75 (8'b10100000)	0.25 (8'b00100000)	-100 (8'b10011100)	-55 (8'b11001001)	8'b01011000	8'b00010000

Table 3 Experiment record table

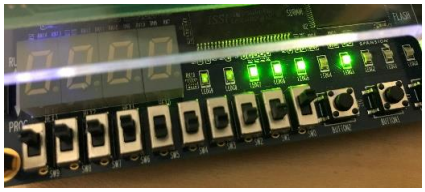

Figure 10 The experiment process

Then just need to input the values above one by one and note down the result into the table, the process is shown at left.

Compare the experiment results with the theoretical results and simulation results, the experiment results are exactly same as simulation. Therefore the test is successful and this complex number multiplier works properly.

6. Conclusion

To summarise, the objectives of this assignment have all been achieved and every required functions have been implemented. Moreover, the modules and algorithms in this design work properly. In other words, a complex number multiplier with a sequential controller has been designed and tested successfully.

Top-level Entity Name	cnmctrl
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	703 / 114,480 (< 1 %)
Total combinational functions	695 / 114,480 (< 1 %)
Dedicated logic registers	44 / 114,480 (< 1 %)
Total registers	44
Total pins	19 / 529 (4 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 11 Synthesis report

The synthesis report for Cyclone IV EP4CE115 is shown at left:

The total cost of logic elements is 703 which is kind of high for a multiplier. There must be some good way to implement this function more efficiency. This multiplier could continue to be optimized to investigate a better performance and efficiency.

Moreover, the total logic elements in EP4CE115 are 114480, and the complex number multiplier takes up less than 1% of it.

7. References

[1] Dr Tom J Kazmierski, "ELEC6233 Digital System Synthesis: Notes," University of Southampton, [Online]. Available: https://secure.ecs.soton.ac.uk/notes/elec6233/ [Accessed 16/04/2017]