ELEC6233 – High-Level Synthesis

Haoheng Huang
Hh2u22
MSc Microelectronics System Design
Tomasz Kazmierski

ABSTRACT:
This report presents the synthesis of an FFT butterfly system using System Verilog. The system can handle 8-bit twiddle factors and inputs, producing an 8-bit output. The picoMIPS architecture was used to design the control path, data path, and address path. The report includes details on the hardware blocks, Modelsim test benches and results, and the method for testing the FPGA. Future improvements include accepting more inputs and twiddle factors and increasing accuracy by expanding data length.

1. 1 Introduction

The objective of this assignment is to synthesize an FFT butterfly using SystemVerilog, starting from a specific pseudocode and converting it into an RTL-level code. [2]The FFT butterfly is composed of one complex number, known as the 'twiddle' factor, and two complex number inputs (a and b), producing two complex outputs (y and z). The calculation involved in this process can be represented by the following equation:

$$\begin{aligned}\text{Re } y &= \text{Re } a + (\text{Re } b \times \text{Re } w - \text{Im } b \times \text{Im } w)\\ \text{Im } y &= \text{Im } a + (\text{Re } b \times \text{Im } w + \text{Im } b \times \text{Re } w)\\ \text{Re } z &= \text{Re } a - (\text{Re } b \times \text{Re } w - \text{Im } b \times \text{Im } w)\\ \text{Im } z &= \text{Im } a - (\text{Re } b \times \text{Im } w + \text{Im } b \times \text{Re } w)\end{aligned}$$

It is essential to thoroughly analyze the functionality of the modules within the picoMIPS architecture before developing the behavioural code. An initial comprehensive system design was created, consisting of the control path, data path, and address path. Specific instruction types (ADD, ADDI, SUB, MULI, HOLD) and sequences were identified to estimate the total number of instructions required for computation.

This report first introduces the overall architecture of the design. The control flow diagram, related architectural designs, and some code snippets will be described. Modelsim test benches, results, and RTL-level charts will also be shown and explained in this section. Then, the next section will provide detailed information on the hardware blocks. Finally, the method for testing the design after programming the FPGA will be demonstrated.

1. 2 Overall architecture of the design

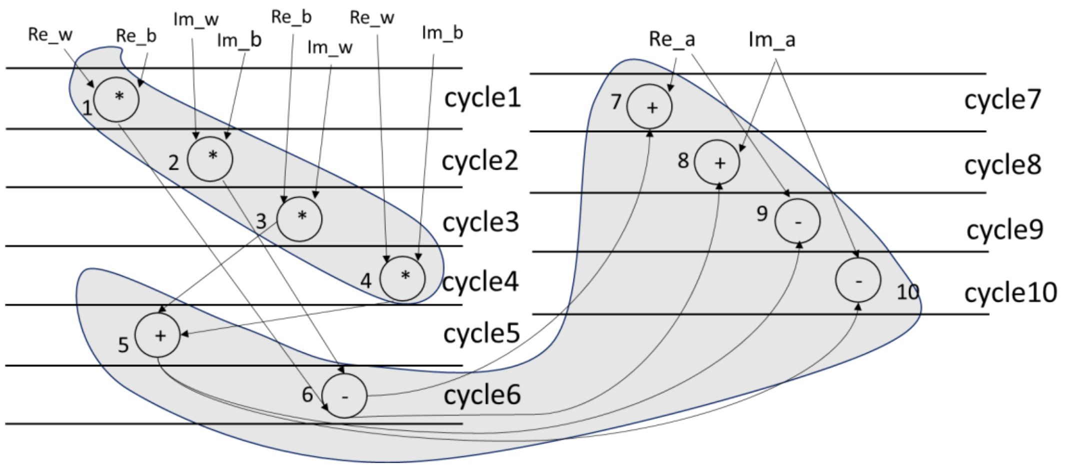


Figure 1.2.1 The Scheduling Diagram of my design.

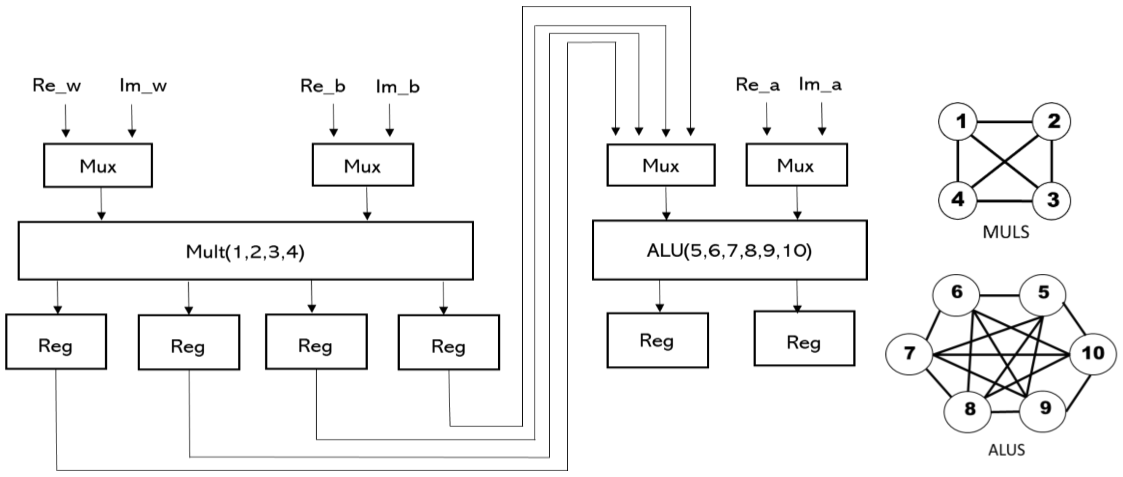


Figure 1.2.2 The structure of datapath and MULS' and ALUS' diagram.

In this project, the use of resources is minimized, which means that only one multiplier and one ALU are available due to resource constraints. As a result, Figure 1.2.1 shows the resulting scheduling diagram, which indicates that each cycle can only perform either multiplication or addition operations exclusively. Following the prescribed order of operations, the initial four cycles exclusively execute multiplication operations, while the final six cycles solely undertake addition or subtraction operations. As we can see from the above diagram, the addition and subtraction operations are further scheduled in pairs of add and subtract operations per clock cycle. These operations perform the butterfly part of the FFT transform.

Instruction Set

The design features a total of five distinct instructions, which have been implemented to ensure that all essential operations are adequately addressed within the system.

Details:

ADD -- Add the values stored in both the destination and source registers, and then store the resulting sum back into the destination register.

ADDI-- Add the immediate value to the contents of the destination register and then store the resulting sum back into the destination register.

MULI-- Multiply the immediate value by the contents of the destination register and subsequently store the resulting product back into the destination register.

HOLD-- Branch under specified conditions: execute the branch when the status bit Bstus (connected to SW[8]) is equal to 1 and matches the instruction bit [7].
SUB-- Subtract the value stored in the source register from the value stored in the destination register, and then store the resulting difference back into the destination register.

Instruction Format

Databus size = 8 bits, Instructions size = 20 bits,
Format: 4bits opcode, 4bits address for destination register (%d), 4bits address for souce register (%s), 8bits immediate or address.

ADD	%d	%s	%d = %d + %s
ADDI	%d	%0	%d = %d + Immediate
MULI	%d	%0	%d = %d * %s
HOLD	%d	%0	The branch under specified conditions
SUB	%d	%s	%d = %d + %s

Parts of the Hex Program

HEX	BINARY	ASSEMBLER
11000	20'b0001_0001_0000_00000000	ADDI 0 //CLEAR %1
12000	20'b0001_0010_0000_00000000	ADDI 0 //CLEAR %2
40089	20'b0100_0000_0000_10001001	HOLD 1
0118A	20'b0000_0001_0001_10001010	ADD %1; %1 = input Re_w
4000B	20'b0100_0000_0000_00001011	HOLD 0
4008C	20'b0100_0000_0000_10001100	HOLD 1
02280	20'b0000_0010_0010_10000000	ADD %2; %2 = input Im_w
4000E	20'b0100_0000_0000_00001110	HOLD 0
4008F	20'b0100_0000_0000_10001111	HOLD 1
06680	20'b0000_0110_0110_10000000	ADD %6; %6 = input Im_a
07300	20'b0000_0111_0011_00000000	ADD %7 %3 %7=Re_b
08400	20'b0000_1000_0100_00000000	ADD %8 %4 %8=Im_b
23180	20'b0010_0011_0001_10000000	MUL %3 %1 = Re_w*Re_b
24280	20'b0010_0100_0010_10000000	MUL %4 %2 = Im_w*Im_b
27280	20'b0010_0111_0010_10000000	MUL %7 %2 = Re_b*Im_w
28180	20'b0010_1000_0001_10000000	MUL %8 %1 = Re_w*Im_b
07800	20'b0000_0111_1000_00000000	ADD %7 %8 = Re_bIm_w+Re_wIm_b
33400	20'b0011_0011_0100_00000000	SUB %3 %4 = Re_wRe_b-Im_wIm_b
14000	20'b0001_0100_0000_00000000	ADDI 0 //CLEAR %4
04500	20'b0000_0100_0101_00000000	ADD %4 %5 = %4=Re_a
05300	20'b0000_0101_0011_00000000	ADD %5 %3 Re_y=Re_a+(Re_wRe_b-Im_wIm_b)
40025	20'b0100_0000_0000_00100101	HOLD 0
450A6	20'b0100_0101_0000_10100110	HOLD 1 %5
15000	20'b0001_0101_0000_00000000	ADDI 0 //CLEAR %5
05600	20'b0000_0101_0110_00000000	ADD %5 %6 = %5=Im_a
06700	20'b0000_0110_0111_00000000	ADD %6 %7 Im_y=Im_a+(Re_bIm_w+Re_wIm_b)
4602A	20'b0100_0110_0000_00101010	HOLD 0 %6
460AB	20'b0100_0110_0000_10101011	HOLD 1 %6

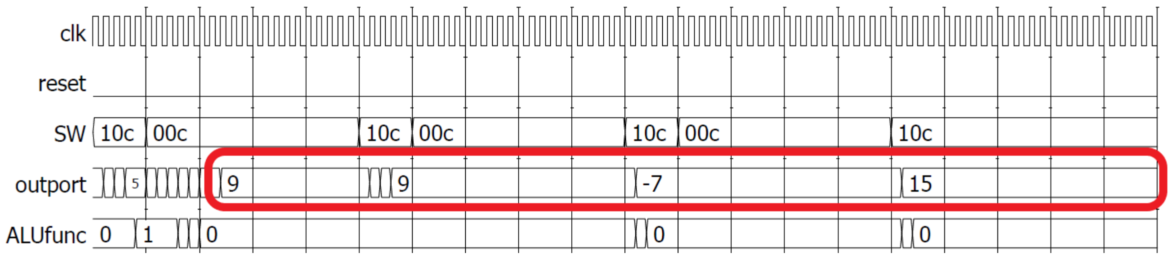


Figure 1.2.3 The simulation of the whole design.

To validate the functionality, I set the following inputs: Re_w=0.5, Im_w=-0.5, Re_b=10, Im_b=4, Re_a=1, and Im_a=8. The results obtained were correct and matched the expected outcomes within an error range of 1 since 8-bit inputs and outputs were used in this design.

1. 3 Details of hardware blocks

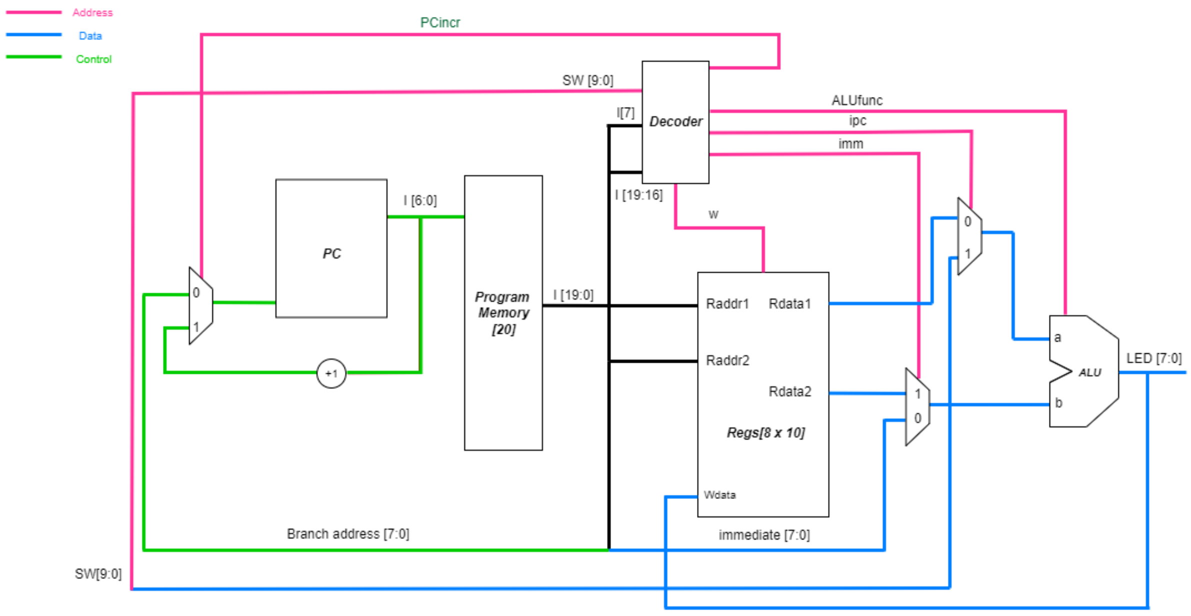


Figure 1.3.1 The structure of picoMIPS

In this project, our objective is to design an n-bit picoMIPS processor architecture. The architecture incorporates an arithmetic logic unit (ALU), a general-purpose register (GPR), a 20-bit instruction decoder, a program counter (PC), and a program memory that stores the machine code for the affine transformation algorithm. The interconnections between these components are illustrated in the accompanying block diagram.

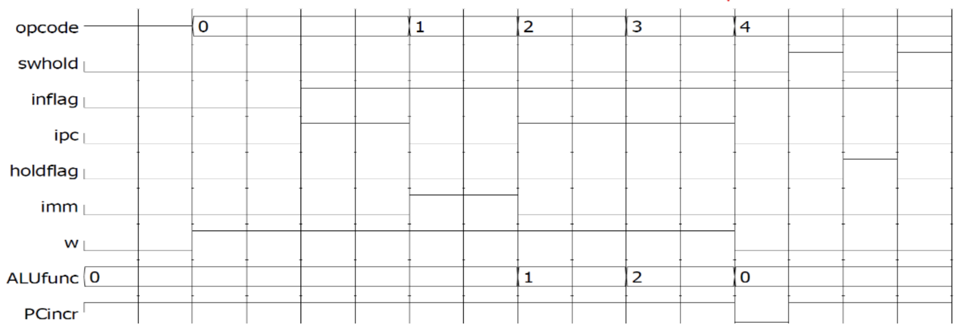


Figure 1.3.2 The structure of the decoder

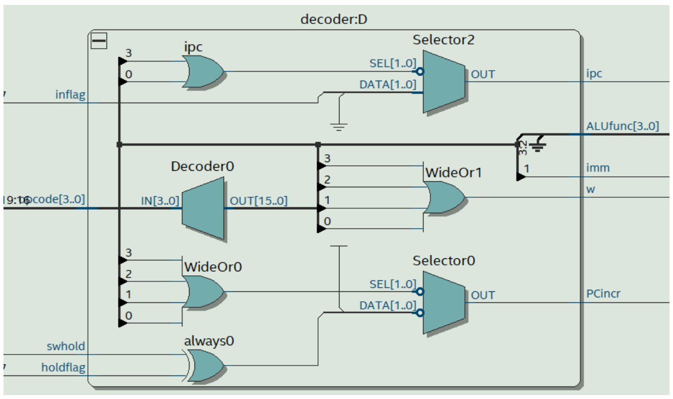


Figure 1.3.3 The simulation of the decoder

To validate the decoder module, this test bench has exercised all functionalities. The output of the decoder module affects both the PC and ALU. When they should = hold the flag and the function is ADD, ADDI, or HOLD, the value of alufunc should be RADD (4'b0000). When the function is SUB and MUL, the value of alufunc should be RSUB (4'b010) and RMUL (4'b001), respectively.

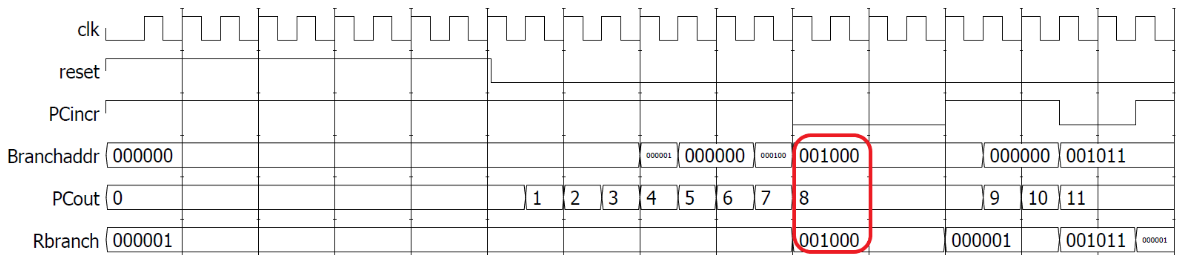
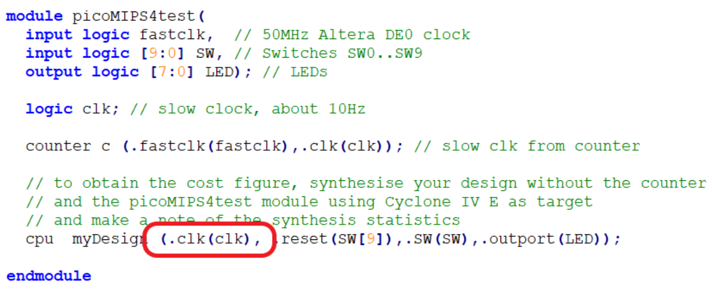


Figure 1.3.4 The simulation of Pcout

When the PCincr signal is high, the program counter (PCount) increments by one on each clock cycle. On the other hand, when the PCincr signal is low, the value of the program counter is set to the value of the Rbranch register, and it stays constant until the PCincr signal goes high again.

1. 4 FPGA implementation

In the previous simulation, I opted to use the fastclk for its efficiency. However, to demonstrate the functionality of the system on an FPGA, I need to integrate the counter module into the design.[1]



```
module picoMIPS4test(
  input logic fastclk,  // 50MHz Altera DE0 clock
  input logic [9:0] SW, // Switches SW0..SW9
  output logic [7:0] LED); // LEDs

  logic clk; // slow clock, about 10Hz

  counter c (.fastclk(fastclk),.clk(clk)); // slow clk from counter

  // to obtain the cost figure, synthesise your design without the counter
  // and the picoMIPS4test module using Cyclone IV E as target
  // and make a note of the synthesis statistics
  cpu  myDesign (.clk(clk), .reset(SW[9]),.SW(SW),.outport(LED));

endmodule
```

Figure 1.4.1 The code of picoMIPS4.

To ensure the proper functionality of the system, I carefully selected a set of test data to verify if the calculations yield the correct results.

Input Data	
Re_w=0.5	Im_w=-0.5
Re_b=10	Im_b=5
Re_a=1	Im_a=12

Output Data	
Re_y = 8.5	Im_y = 9.5
Re_z = -6.5	Im_z = 14.5

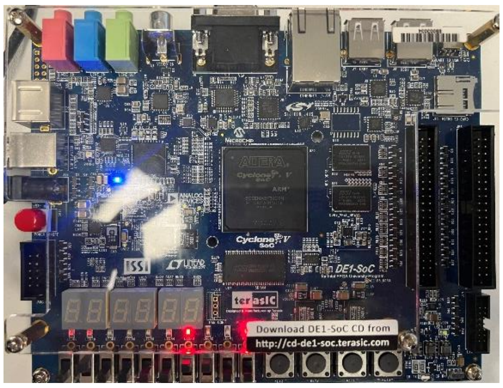


Figure 1.5.1 number 9

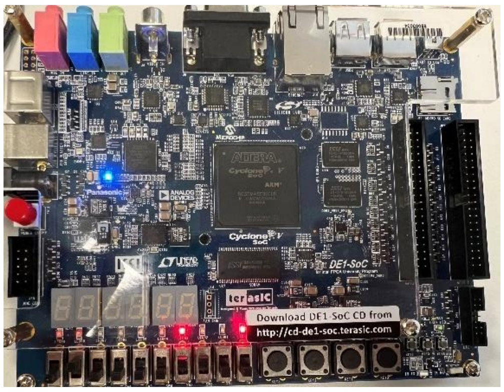


Figure 1.5.2 number 9

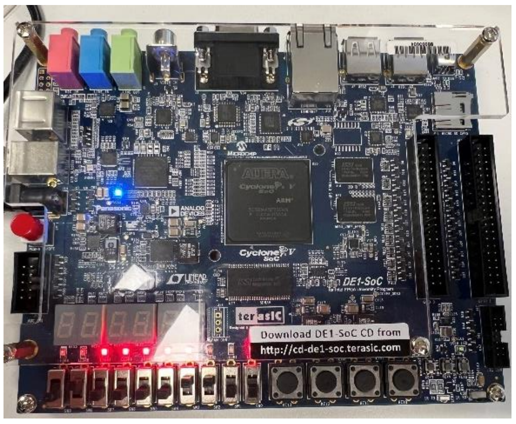


Figure 1.5.2 number -7

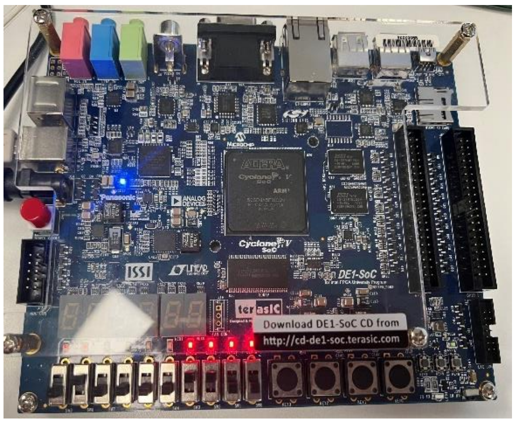


Figure 1.5.3 number 15

The results in the above figure match the expected outcomes. The results obtained were correct and matched the expected outcomes within an error range of 1 since 8-bit inputs and outputs were used in this design.

1. 5 Conclusion

An FFT butterfly system has been synthesized using System Verilog. It can handle one 8-bit twiddle factor and two 8-bit inputs, with an output rounded to the nearest 8-bit. The system is based on a picoMIPS architecture, further deepening my understanding of picoMIPS structure and teaching me how to complete high-level synthesis using System Verilog. Preparation, CFD, and scheduling are all necessary for the design. In the future, the FFT butterfly system can be improved to accept more inputs and twiddle factors. Additionally, accuracy can be improved by expanding the data length.

References

[1] ZwolińskiM., Digital system design with SystemVerilog. Upper Saddle River, Nj: Addison-Wesley, 2010.

[2] Tomasz Kazmierski, "ELEC6233 High-level synthesis slides," University of Southampton.