ELEC6233 – FPGA Synthesis

ZIBO YAN
zy1u21
Electronic Engineering MSc
Terrence Mark

ABSTRACT: *A medium sized FFT butterfly RTL design was achieved using System Verilog. The hardware program was implemented on an Altera FPGA development system. The design generated from high-level synthesis and included controller and data path. The design supported two 8-bit signed input through and one user-defined coefficient of FFT butterfly using a simple hand-shaking protocol.*

1. 1. Introduction

The objective of the assignment is to synthesize a FFT butterfly from a specific pseudocode to an RTL-level code using System Verilog. The FFT butterfly has one complex number as 'twiddle' factor w, two complex number input, a, b, and two complex output y,z. The calculation is shown below.

$$Re\,y = Re\,a + (Re\,b \times Re\,w - Im\,b \times Im\,w)$$
$$Im\,y = Im\,a + (Re\,b \times Im\,w + Im\,b \times Re\,w)$$
$$Re\,z = Re\,a - (Re\,b \times Re\,w - Im\,b \times Im\,w)$$
$$Im\,z = Im\,a - (Re\,b \times Im\,w + Im\,b \times Re\,w)$$

The input values and 'twiddle' factor were all 8-bit input connecting to switch [7:0] on the FPGA. Switch [8] worked as handshaking input signal and Switch [9] worked as an active-low master reset. The output y,z were displayed through led[7:0].

The preparation works mainly included Control/data flow diagram, scheduling, and binding. The flow diagram describes the process of data loading, calculation and data display. Scheduling and binding gives constrains and allocates resource to each operation. The resource of this design was one full adder and one multiplication supporting for an 8-cycle FFT butterfly. The experimental work mainly included coding, simulation, synthesis and pin connection. According to the requirement, the code was divided into 'controller' and 'data path'. 'controller was described by a State machine of 10 states and 'data path 'contains registers, calculation resources and result display. The testbench for simulation work included three groups of input. The Synthesis was completed together with pin connection using Software Quartus Prime. The result was a medium sized program implemented on Altera FPGA, DE1-SoC.

In this report, the overall architecture of the design would be introduced firstly. Control Flow Graph, associated state machine design will be described with some code snippets. Modelsim testbench and results and RTL level diagrams will also be shown and explained in this section. Then, details of hardware blocks will be given in the next section. Finally, the methods of testing the design after programming the FPGA will be demonstrated.

1. 2. Overall architecture of the design

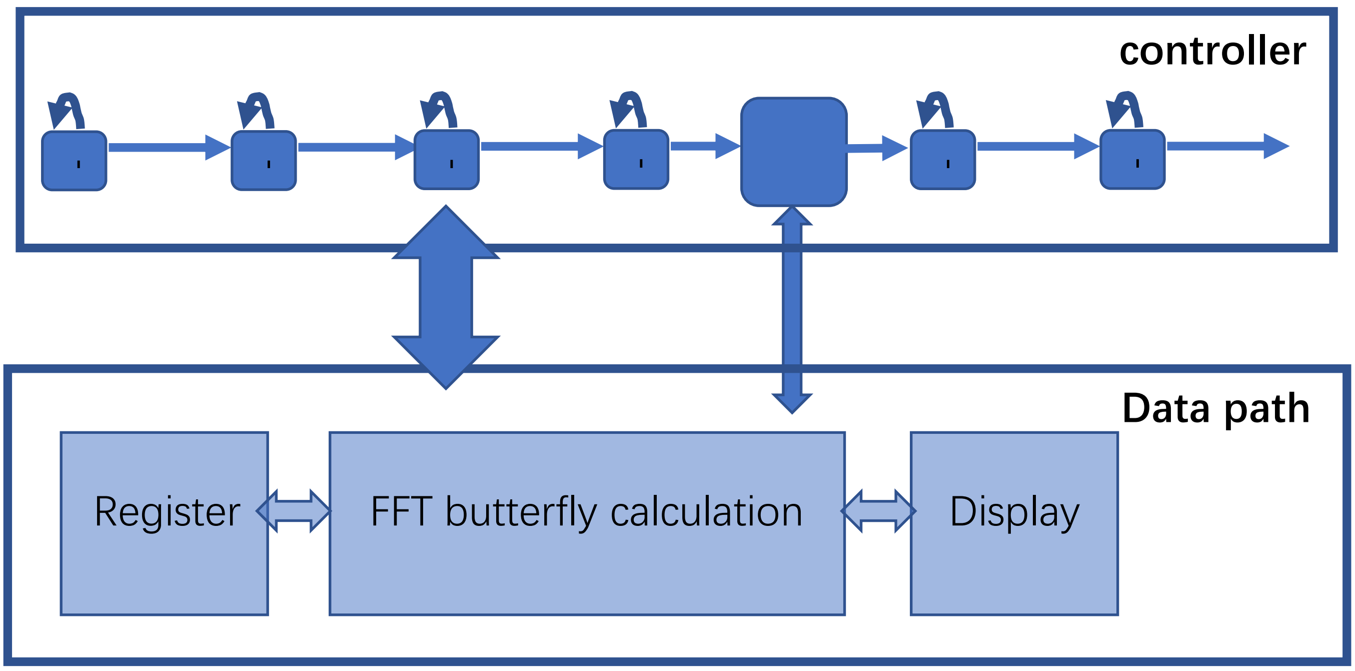


Figure 1 the diagram for architecture of the design

The architecture of the design has one controller and one data path. The controller can be described as Control Flow Graph. The Control Flow Graph can be divided into two parts. The first one is loading and displaying by handshaking signal SW[8]. The second one is to wait for calculation. The data path can be divided in three parts, register, FFT calculation and Display.

1.2.1 controller

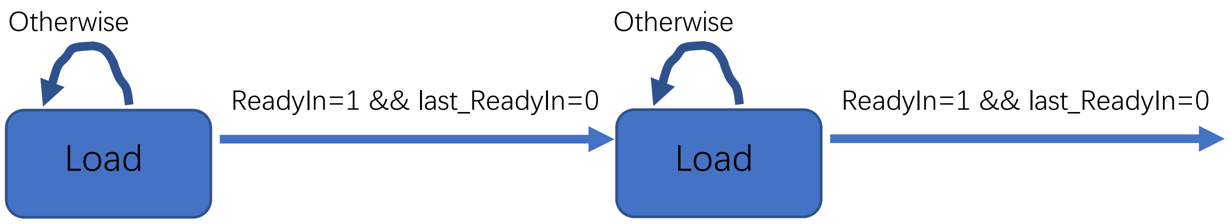


Figure 2 the loading and displaying part of controller

Where 'ReadyIn' is the handshaking signal and 'last_ReadyIn' refers to the value of 'ReadyIn' in last clock period. 'ReadyIn=1 && last_ReadyIn=0' becomes valid only when the handshaking signal converts from 0 to 1. Therefore, when SW[8] which is handshaking signal changed from 0 to1, the state steps forward.

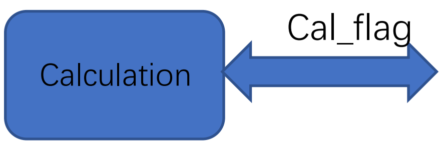


Figure 3 the calculation part of controller

The part of Controller waiting for calculating FFT butterfly. A flag was used for controlling calculation.

1.2.2 datapath

The data path contains three parts. The first one is a general register.

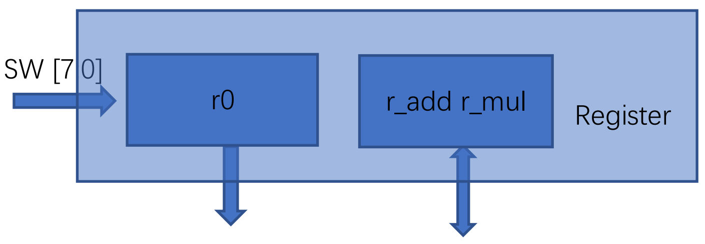


Figure 4 code for register

The general register contains two functions both supporting asynchronous reading and synchronous writing. The first function is to store SW[7:0] input. The second function is to store calculation results and ready for display output. For calculation, three states were considered including 'ready for next calculation', 'store addition and multiplication' and 'only store addition'.

The second part of data path is for calculation. The resource includes one synchronous multiplier and one full adder. By hand perform scheduling, a minimum-latency of 8-cycle calculation was design below.

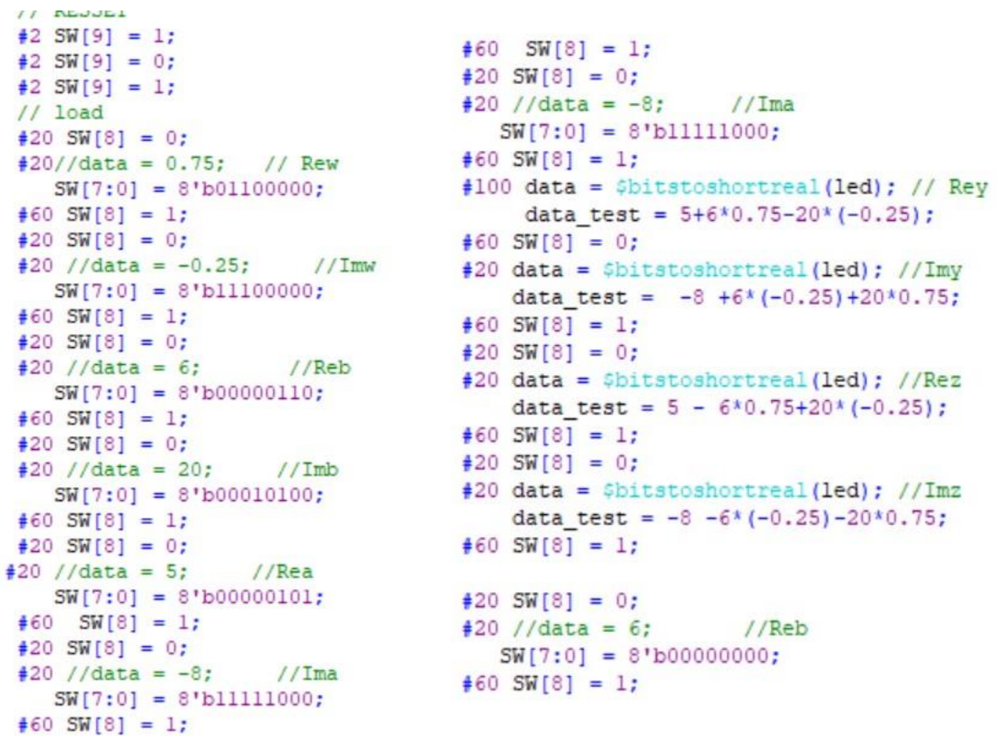
	adder	*multiplier*
Cycle 1		*Reb*Rew*
Cycle 2		*Imb*Imw*
Cycle 3	*(Reb*Rew) – (Imb*Imw)*	*Reb*Imw*
Cycle 4	*(Rea) + (Reb*Rew – Imb*Imw)*	*Imb*Rew*
Cycle 5	*(Rea) – (Reb*Rew – Imb*Imw)*	
Cycle 6	*(Reb*Imw) + (Imb*Rew)*	
Cycle 7	*(Ima) + (Reb*Imw + Imb*Rew)*	
Cycle 8	*(Ima) – (Reb*Imw + Imb*Rew)*	

Table 1 8-Cycle for FFT butterfly

In each cycle, the addition was completed directly while the multiplication was completed by a separate module. the reason for choosing synchronous multiplier is that the multiplier replies too much combinational logic which may leads to setup time or hold time validation.

1.2.3 testbench

Here is the testbench:



```
// RESSET
#2 SW[9] = 1;
#2 SW[9] = 0;
#2 SW[9] = 1;
// load
#20 SW[8] = 0;
#20//data = 0.75;   // Rew
   SW[7:0] = 8'b01100000;
#60 SW[8] = 1;
#20 SW[8] = 0;
#20 //data = -0.25;      //Imw
   SW[7:0] = 8'b11100000;
#60 SW[8] = 1;
#20 SW[8] = 0;
#20 //data = 6;       //Reb
   SW[7:0] = 8'b00000110;
#60 SW[8] = 1;
#20 SW[8] = 0;
#20 //data = 20;      //Imb
   SW[7:0] = 8'b00010100;
#60 SW[8] = 1;
#20 SW[8] = 0;
#20 //data = 5;      //Rea
   SW[7:0] = 8'b00000101;
#60  SW[8] = 1;
#20 SW[8] = 0;
#20 //data = -8;      //Ima
   SW[7:0] = 8'b11111000;
#60 SW[8] = 1;
```

```
#60  SW[8] = 1;
#20 SW[8] = 0;
#20 //data = -8;      //Ima
   SW[7:0] = 8'b11111000;
#60 SW[8] = 1;
#100 data = $bitstoshortreal(led); // Rey
     data_test = 5+6*0.75-20*(-0.25);
#60 SW[8] = 0;
#20 data = $bitstoshortreal(led); //Imy
    data_test =  -8 +6*(-0.25)+20*0.75;
#60 SW[8] = 1;
#20 SW[8] = 0;
#20 data = $bitstoshortreal(led); //Rez
    data_test = 5 - 6*0.75+20*(-0.25);
#60 SW[8] = 1;
#20 SW[8] = 0;
#20 data = $bitstoshortreal(led); //Imz
    data_test = -8 -6*(-0.25)-20*0.75;
#60 SW[8] = 1;

#20 SW[8] = 0;
#20 //data = 6;        //Reb
   SW[7:0] = 8'b00000000;
#60 SW[8] = 1;
```

Figure 5 testbench

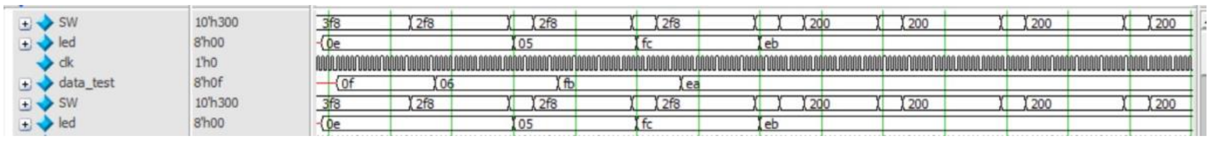


Figure 6 simulation of output

From the Figure4, we can see that the output ‘led’ is equal to ‘data_test’ which is the reference. Although there are slightly different. That is because during the calculation, 16 bit results has to be truncate to 8 bit where the number is around to nearest.

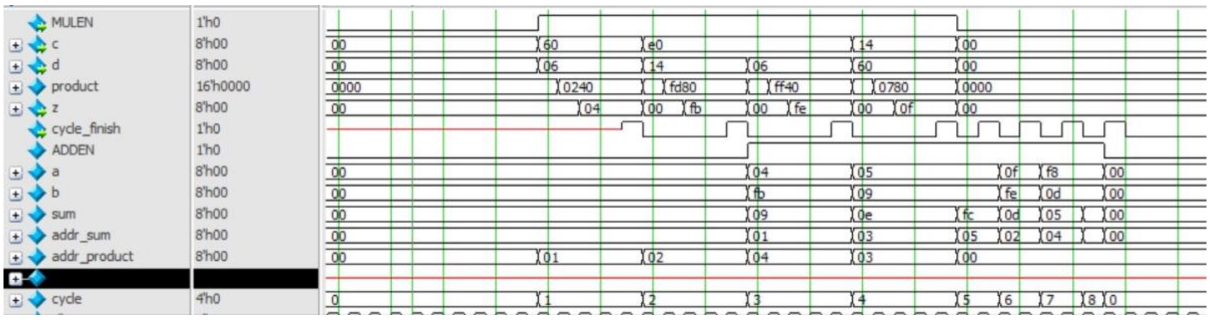


Figure 7 simulation of calculation

*From the figure 5 , the 8-cycle calculation operates. When ‘MULEN’ on, the multiplication works. ‘product’ calculates c*d and ‘z’ truncates ‘product’. When ‘ADDEN’ on, the addition works. At the end of cycle, all the parameter turns to 0 as default.*

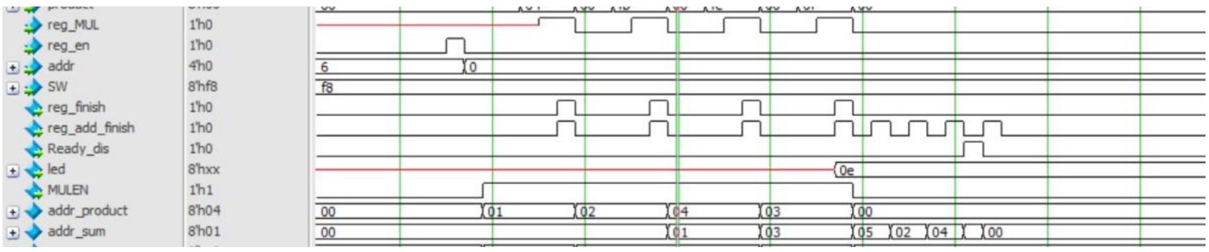


Figure 8 simulation of register

From the figure6, the register works when multiplication. When ‘reg_MUL’ on it starts to store result at with a proper address. When ‘reg_finish’ and ‘reg_add_finish’ on, it stops writing in and prepare for next storing。

1. 3. Details of hardware blocks (use appropriate section title to your design)

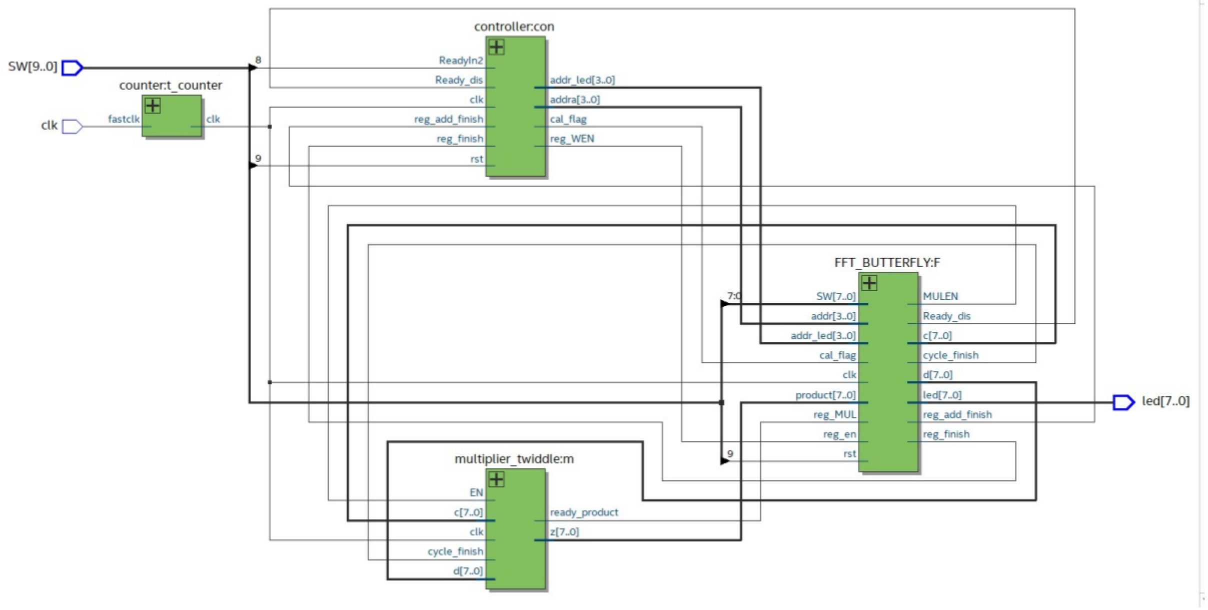


Figure 9 RTL diagram of the design

Here is the RTL diagram of the design. SW [9:0] is the input from switches on FPGA AND led[7:0] is connected to red LED light. Each of these four blocks, counter, controller, FFT_BUTTERFLY and multiplier, will be demonstrated detailed in following subsections.

1.3.1 counter

‘Counter’ works as a clock divider to generate a slow clock,2.5MHz for the design. The default clock frequency of FPGA is 50MHz. The frequency is too high especially for asynchronous input. It is of high possibility to generate Metastable states and jitter if by-hand switches were used as input under 50MHz clock.

1.3.2 controller

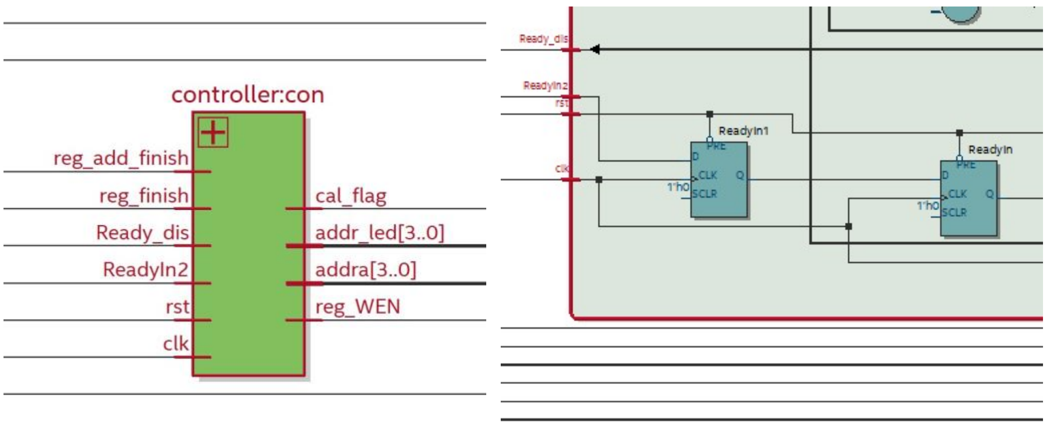


Figure 10 the left figure is controller box; the right figure is two register buffer for SW[8]

Controller is basically a machine state. In this design, the machine state has 10 state and each state steps to next state when ‘ReadyIn’ converts from 0 to 1. ‘ReadyIn’ is not directly connected to SW[8]. As an asynchronous input, it firstly triggered through two registers as *buffers. After clock, It can coincide with clock to reduce metastable. 'reg_add_finish', 'reg_finish' and 'Ready_dis' are flags that writes back to controller about states of data path. 'cal_reg' controls the start of calculation. 'addr_led' refers to output address while 'addra' refers to the address storing input from SW[7:0]. 'reg_WEN' is the enable signal to write SW[7:0] to register.*

1.3.3 multiplier_twiddle

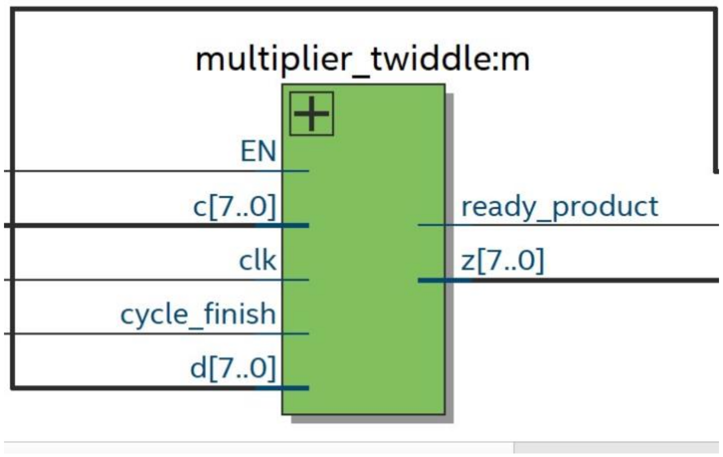
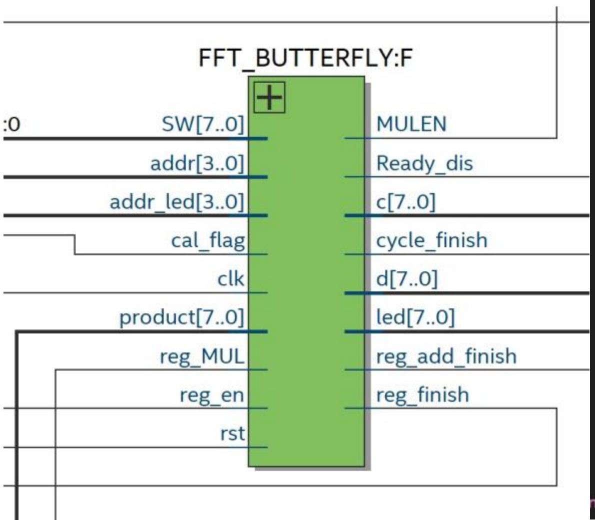
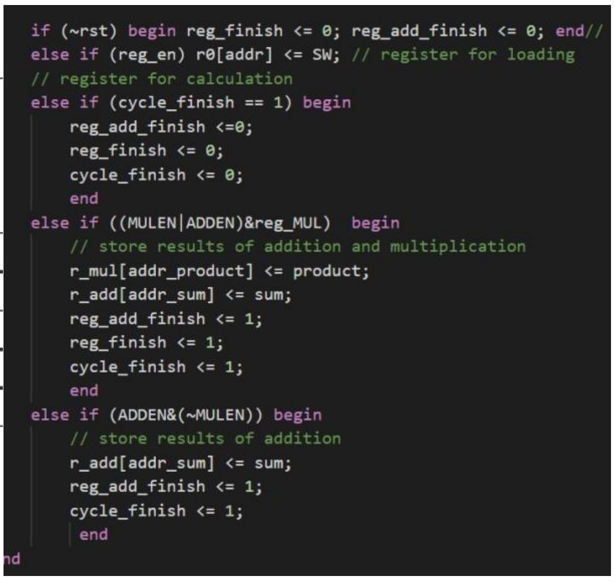


Figure 11 synchronous multiplier

This multiplier is a synchronous multiplier using three clock cycles to complete multiplication. One cycle is to store multiplicands. One cycle is to complete 8-bit multiplication with 16-bit product. One cycle is to truncate 16-bit product to 8-bit. Therefore, the multiplier has two 8-bit input 'c' and 'd' with one 8-bit output 'z'. 'EN' is the flag to start multiplication while 'cycle_finish' is the flag signal to stop multiplication and reset states. 'ready_product' rises to 1 when multiplication finishes.

1.3.4 FFT_butterfly



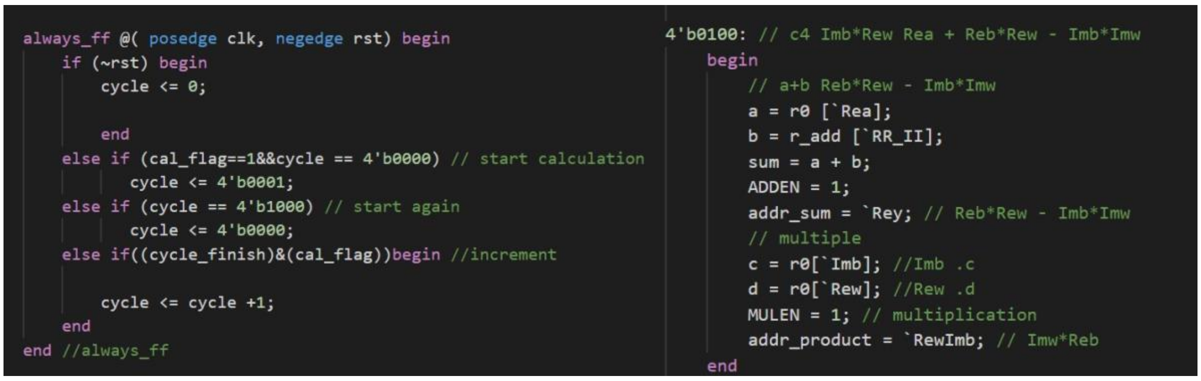


```
if (~rst) begin reg_finish <= 0; reg_add_finish <= 0; end//
else if (reg_en) r0[addr] <= SW; // register for loading
// register for calculation
else if (cycle_finish == 1) begin
    reg_add_finish <=0;
    reg_finish <= 0;
    cycle_finish <= 0;
    end
else if ((MULEN|ADDEN)&reg_MUL)  begin
    // store results of addition and multiplication
    r_mul[addr_product] <= product;
    r_add[addr_sum] <= sum;
    reg_add_finish <= 1;
    reg_finish <= 1;
    cycle_finish <= 1;
    end
else if (ADDEN&(~MULEN)) begin
    // store results of addition
    r_add[addr_sum] <= sum;
    reg_add_finish <= 1;
    cycle_finish <= 1;
     end
nd
```

Figure 12 FFT_BUTTERFLY module and the snapshot of register code

'FFT_BUTTERFLY' module contains main functions of data path including registers, calculation and display.

There are three registers declared, r0, r_mul, and r_add. 'r0'is the register for input SW[7:0] whose address is 'addr'. 'r_mul' is the register for input 'product', whose address is *‘addr_product’. ‘r_add’ is the register for addition whose address is ‘addr_sum’. The registers are all synchronous input and asynchronous output. When ‘reg_en’ is on, data can be written to r0. When ‘(MULEN|ADDEN)&reg_MUL’ is on which means addition and multiplication are all finished, data can be written to r_add and r_mul. When ‘ADDEN&(~MULEN)’ is on, which means only addition is finished, data can be written to r_add. ‘cycle_finish’ refers to a reset signal for register.*

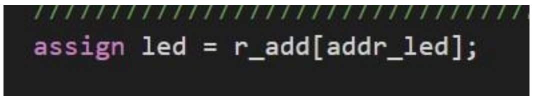


```
always_ff @( posedge clk, negedge rst) begin
    if (~rst) begin
        cycle <= 0;

        end
    else if (cal_flag==1&&cycle == 4'b0000) // start calculation
           cycle <= 4'b0001;
    else if (cycle == 4'b1000) // start again
           cycle <= 4'b0000;
    else if((cycle_finish)&(cal_flag))begin //increment

        cycle <= cycle +1;
    end
end //always_ff
```

```
4'b0100: // c4 Imb*Rew Rea + Reb*Rew - Imb*Imw
    begin
        // a+b Reb*Rew - Imb*Imw
        a = r0 [`Rea];
        b = r_add [`RR_II];
        sum = a + b;
        ADDEN = 1;
        addr_sum = `Rey; // Reb*Rew - Imb*Imw
        // multiple
        c = r0[`Imb]; //Imb .c
        d = r0[`Rew]; //Rew .d
        MULEN = 1; // multiplication
        addr_product = `RewImb; // Imw*Reb
    end
```

Figure 13 the left is the code for calculation increment and the right is the code for cycle 4 of 8 in calculation

Calculation part in this module consists of 8 cycles using one adder and one multiplier. When the start signal ‘cal_flag’ is on, the calculation starts from cycle 1. Each time ‘cycle_finish’ is on, which means addition, multiplication and register are all finished, the cycle increments. It will return to cycle 0 and wait start signal when all 8 cycles finish.



```
//////////////////////////////////
assign led = r_add[addr_led];
```

Figure 14 the code for led output

The display part of this module uses continuous assignment. Led is directly connected to register where output data is stored.

1. 5. FPGA Implementation

To test the design after programming the FPGA, three groups of tests had been completed. These tests were aimed at led hardware, ‘controller’ module and ‘register’. All input test data would not be change during three tests and keep the same with simulation. The LED output connection would change to show states of different modules in different tests.

Rew	*Imw*	*Reb*	*Imb*	*Rea*	*Ima*
0.75	*-0.25*	*6*	*20*	*5*	*-8*
0.75	*-0.25*	*0*	*0*	*0*	*0*
0.75	*-0.25*	*8’b11111111*	*8’b11111111*	*8’b11111111*	*8’b11111111*

Table 2 the table shows an example of test input data

LED was first connected to several fixed value. This was to test any hardware defect in LED.

```
assign led = 8’b11111111;
assign led = 8’b10101010;
……
```

Then LED was connected to address of registers which store the input from Switches (SW[7:0]). Each time SW[8] on the FPGA was changed from 0 to 1, the led changed to show the relative address.

```
assign led = addr;//addr increment from 0 to 5 firstly then 2 to 5 each cycle.
```

The final test was to connect LED to calculation results register as required. The FFT calculation would operate at least three times with different inputs and at least twice using

SW[9] to reset the program. Each time the results should be compared with right ones. The test inputs included positive numbers and negative numbers. The special cases 0, 8'b00000000 and 8'b11111111 were also test.

```
assign led = r_add[addr_led]
```

Noted that during the final test, led should keep showing last value Imz after four changes. Also, there should be slight jitter before first change in each calculation this was because Imz changed.

1. 6. Conclusion

A FFT butterfly system using System Verilog has been synthesized. It can deal with one 8-bit twiddle factor and two 8-bit input with a 8-bit output rounding to nearest. The FFT calculation can be completed within 40 clock cycles repeatedly. The system was designed by one controller and one data path. The logic resource contained 116 ALM, 182 registers, one multiplier and one adder. From the design, I learnt about how to complete a high-level synthesis using System Verilog. The preparation, CFD and scheduling are all necessary for design.

In the future, the FFT butterfly could be improved to accept more input and twiddle factors. Also the accuracy could be improved by expanding the data length.

1. 7. References

1. *DE1-SoC_User_manual_revf*