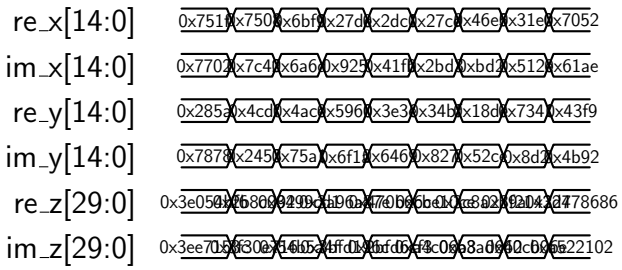
ELEC6233 FPGA Implementation of a Complex Number Multiplier

Joshua Tyler
jlt1e16
MSc Embedded Systems
Dr Jeff Reeve

Abstract

1 Introduction



2 Design

Include 'Circuit diagrams' and discussion of design

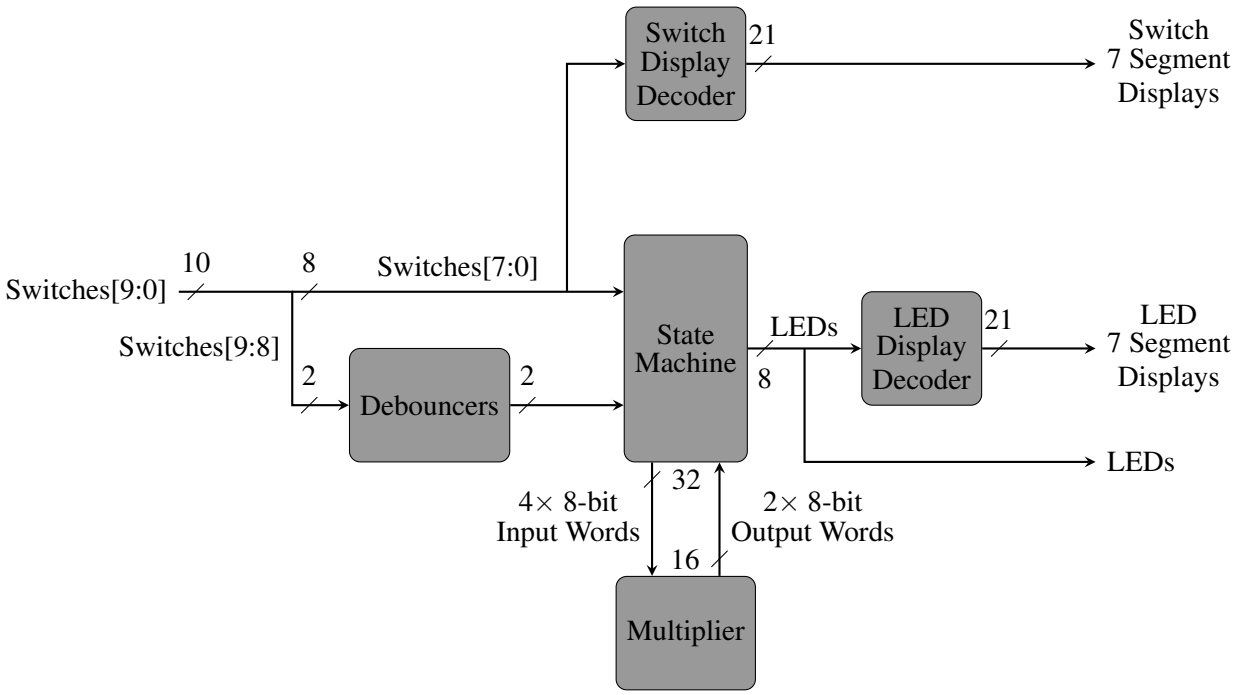


Figure 1: Overall Block Diagram

2.1 Complex Multiplier

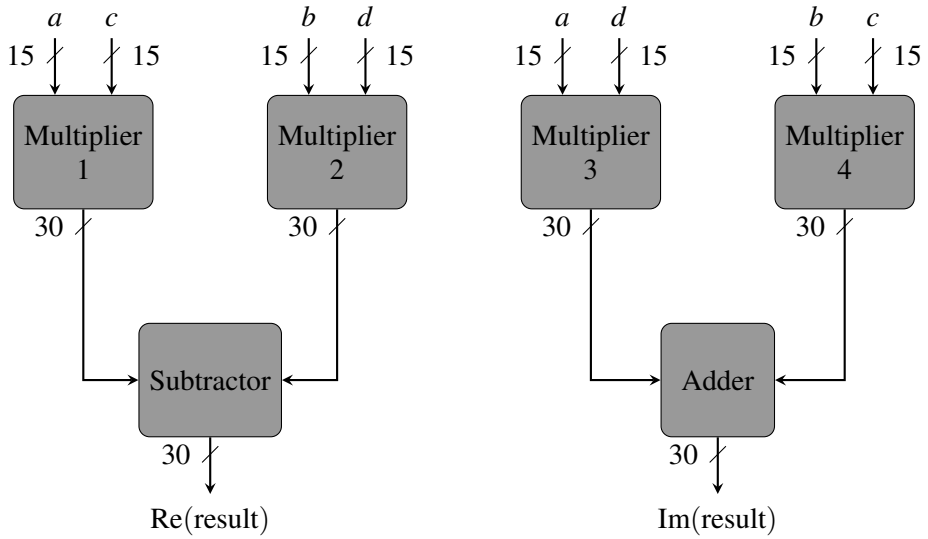


Figure 2: Multiplier architecture

2.2 State Machine

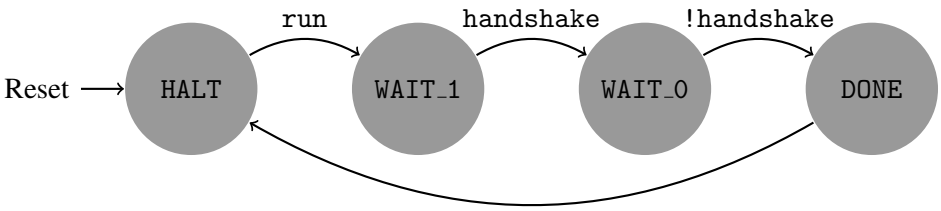


Figure 3: Read state machine state transition diagram

3 Verification

Include sample data and results

Listing 1: `test_bin_to_bcd.sv` Stimulus

```
17      $stop;
18  end
19
20  endmodule
```

Listing 2: `test_cmplx_mult.sv` Stimulus

```
17  logic 'LED_SIZE LED;
18  logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7;
19
20  cmplx_mult dut (.*);
21
22  //Assign switches to their fucntions
```

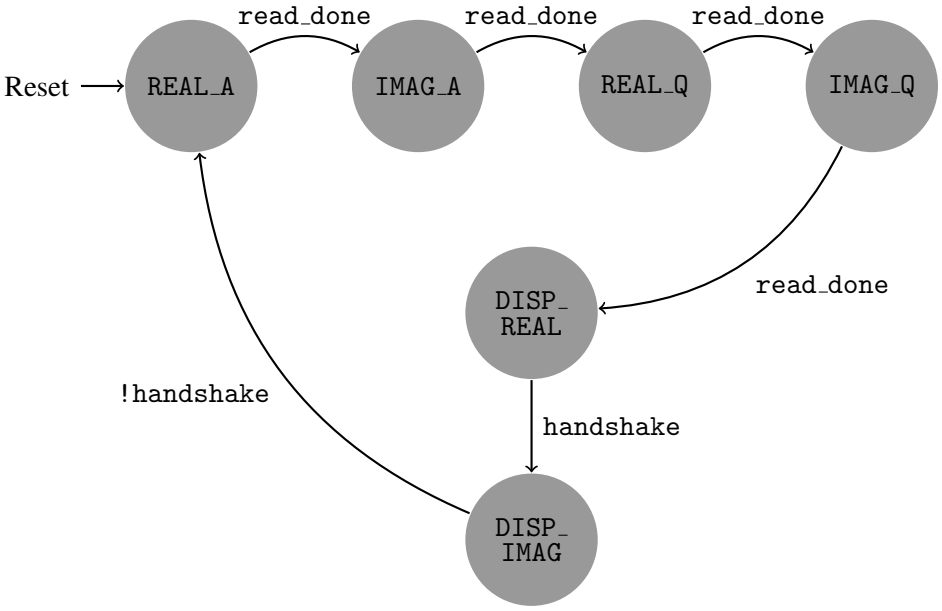


Figure 4: Main state machine state transition diagram

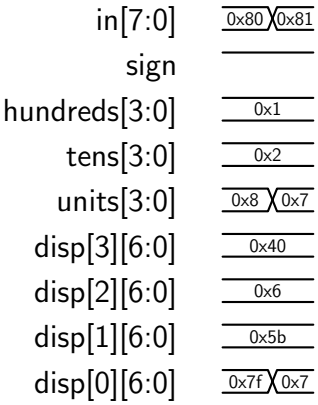


Figure 5: `test_bin_to_bcd.sv` Output

```
23 logic reset_n;
24 logic handshake;
25 logic `WORD_SIZE data_in;
26 always_comb
27 begin
28     switches[$high(switches)] = reset_n;
29     switches[$high(switches) -1] = handshake;
30     switches[$high(switches) -2 : $low(switches)] = data_in;
31 end
32
33 //Clock and reset
34 initial
35 begin
36     clk = 0;
37     reset_n = 1;
```

Listing 3: `test_debounce.sv` Stimulus

```
17
18
19 //Clock
```

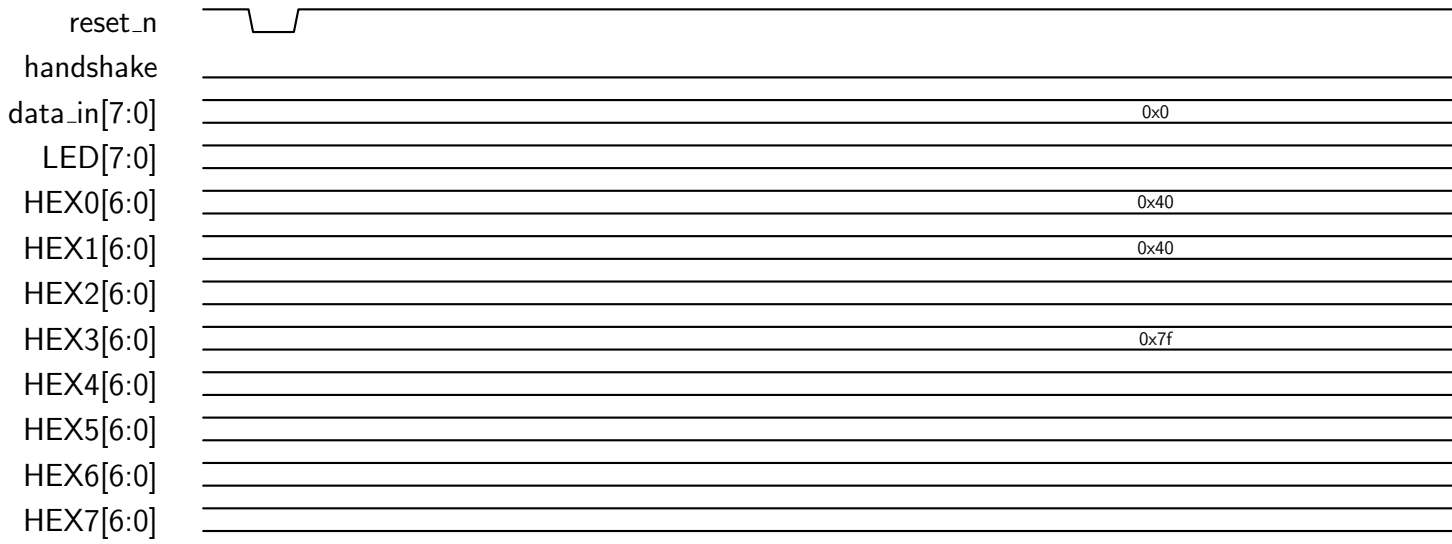


Figure 6: `test_cmplx_mult.sv` Output

```
20 initial
21 begin
22     clk = 0;
23
24     forever
25         #5ns clk = !clk;
26 end
27
28 // Stimulus
29 initial
30 begin
31     signal_in = 0;
32     #1.5us assert(signal_out == signal_in);
33
34     signal_in = 1;
35     #0.5us assert(signal_out != signal_in);
36     signal_in = 0;
37     #0.9us signal_in = 1;
```

Figure 7: `test_debounce.sv` Output

Listing 4: `test_mult.sv` Stimulus

```
17
18 mult dut (.*);
19
20 // Test a single calculation
21 task testNums;
22 input logic signed `MULT_IN_SIZE re_x_in, im_x_in, re_y_in, im_y_in;
23 input logic signed `MULT_OUT_SIZE re_z_in, im_z_in;
24 begin
25     re_x = re_x_in;
26     im_x = im_x_in;
27
```

```
28      re_y = re_y_in;
29      im_y = im_y_in;
30
31      #1ns;
32
33      assert(re_z == re_z_in) else $display("Error. Expected %x, got %x",
            re_z_in, re_z);
34      assert(im_z == im_z_in) else $display("Error. Expected %x, got %x",
            im_z_in, im_z);
35
36  end
37  endtask;
```

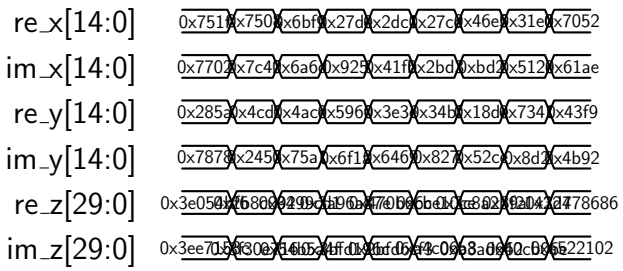


Figure 8: `test_mult.sv` Output

Listing 5: `test_read_sm.sv` Stimulus

```
17      reset_n = 1;
18      # 10ns reset_n = 0;
19      # 10ns reset_n = 1;
20
21      forever
22          #5ns clk = !clk;
23  end
24
25  //Stimulus
26  initial
27  begin
28      handshake = 0;
29      run = 0;
30      #32ns;
31
32      @(posedge clk);
33      run = 1;
34      @(posedge clk);
35      run = 0;
36      #30ns handshake = 1;
37      @(posedge clk) assert(read);
```

Listing 6: `test_sm.sv` Stimulus

```
17  logic `WORD_SIZE re_a, im_a, re_q, im_q;
18
19  sm dut (.*);
20
21  //Assign switches to their fucntions
```

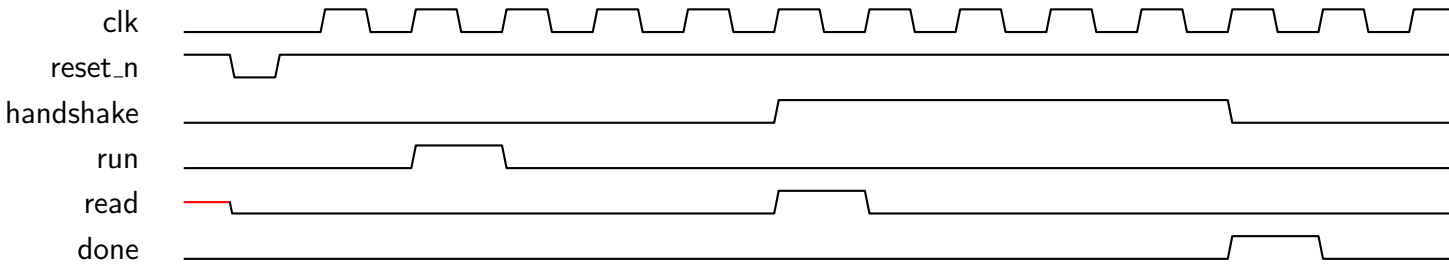


Figure 9: `test_read_sm.sv` Output

```
22 logic reset_n;
23 logic handshake;
24 logic `WORD_SIZE data_in;
25 always_comb
26 begin
27     SW[$high(SW)] = reset_n;
28     SW[$high(SW) -1] = handshake;
29     SW[$high(SW) -2 : $low(SW)] = data_in;
30 end
31
32 //Clock and reset
33 initial
34 begin
35     clk = 0;
36     reset_n = 1;
37     # 10ns reset_n = 0;
```

Figure 10: `test_sm.sv` Output

4 Synthesis

Include "Synthesis Result"

5 Conclusion

References