# 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**

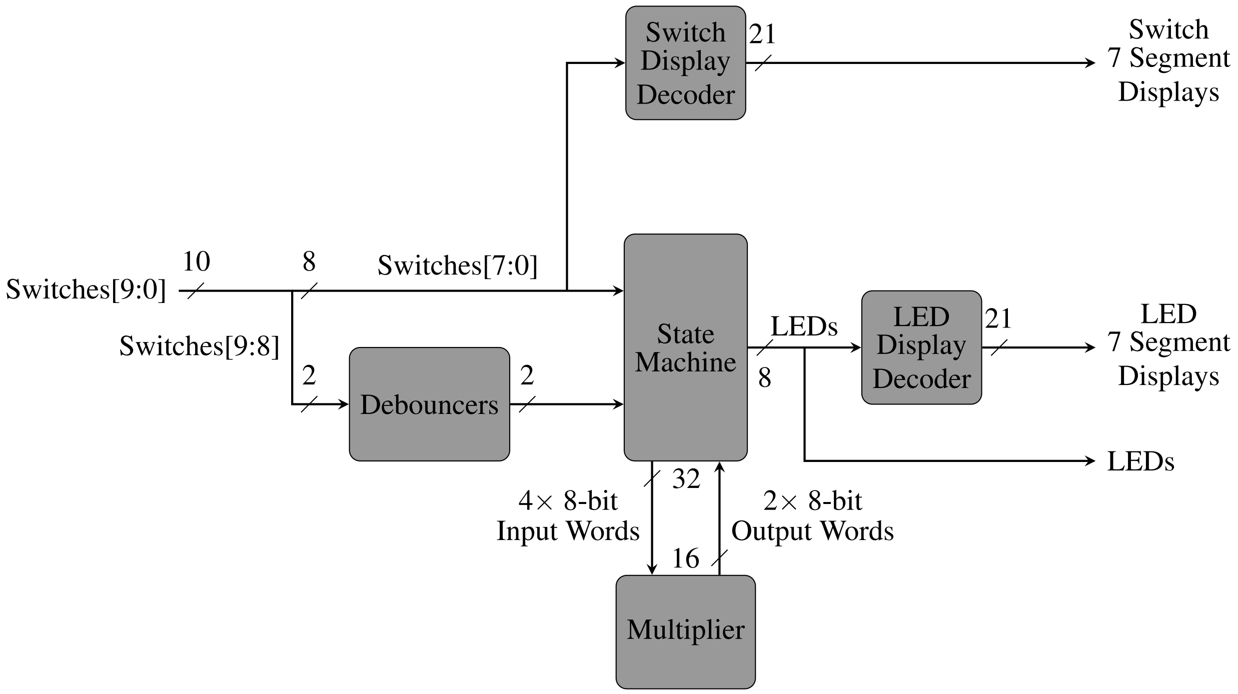

Figure 1: Overall Block Diagram

## 2.1 Complex Multiplier

## 2.2 State Machine

# 3 Verification

**Include sample data and results Add asterisked out states manually check waveforms Fix line numbers for stimulus**

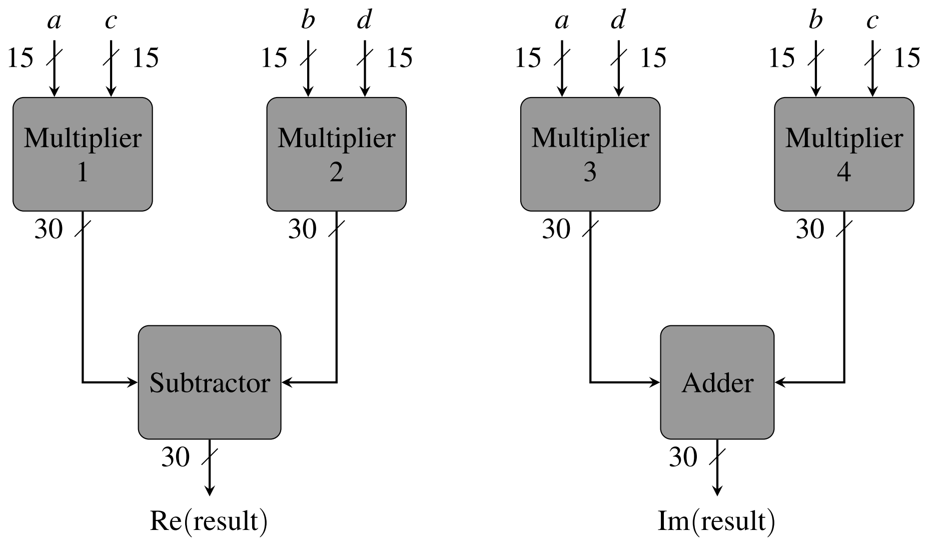

Figure 2: Multiplier architecture

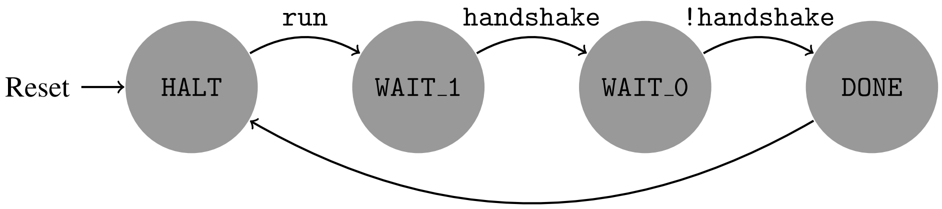

Figure 3: Read state machine state transition diagram

Listing 1: `test_bin_to_bcd.sv` Stimulus

```
    $stop;
end

endmodule
```

Listing 2: `test_cmplx_mult.sv` Stimulus

```
logic `LED_SIZE LED;
logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7;

cmplx_mult dut (.*);

//Assign switches to their fucntions
logic reset_n;
logic handshake;
logic `WORD_SIZE data_in;
always_comb
begin
    switches[$high(switches)] = reset_n;
    switches[$high(switches) -1] = handshake;
    switches[$high(switches) -2 : $low(switches)] = data_in;
end

//Clock and reset
initial
begin
```

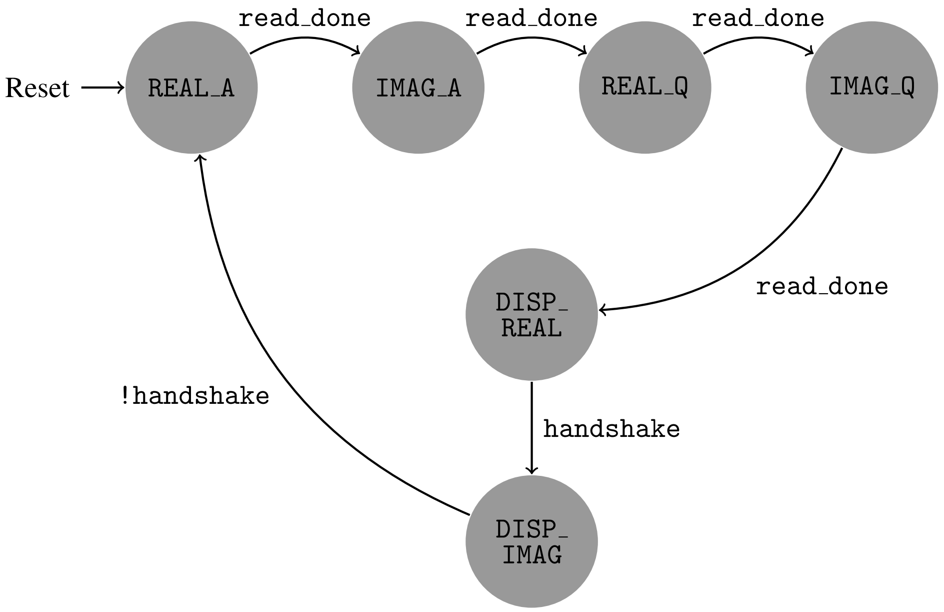

Figure 4: Main state machine state transition diagram

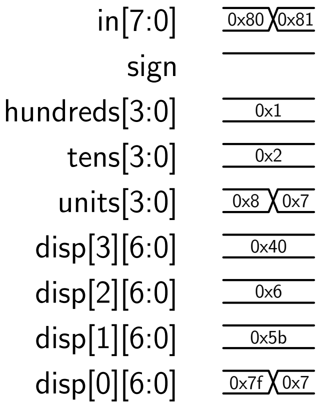

Figure 5: `test_bin_to_bcd.sv` Output

```
    clk = 0;
    reset_n = 1;
```

Listing 3: `test_debounce.sv` Stimulus

```


//Clock
initial
begin
    clk = 0;

    forever
        #5ns clk = !clk;
end

//Stimulus
initial
begin
    signal_in = 0;
    #110ns assert(signal_out == signal_in);
```

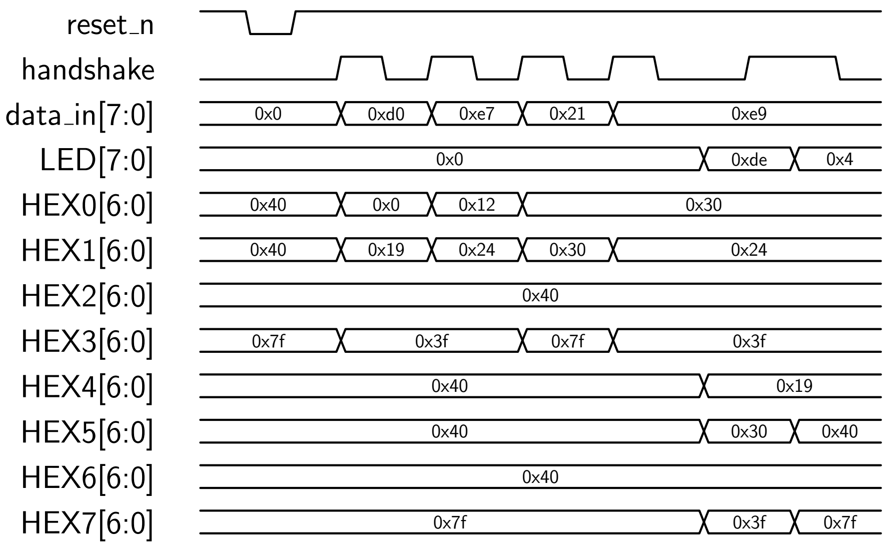

Figure 6: `test_cmplx_mult.sv` Output

```

    signal_in = 1;
    #20ns assert(signal_out != signal_in);
    signal_in = 0;
    #10ns signal_in = 1;
```

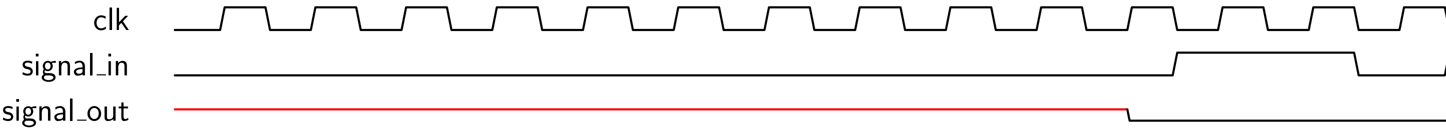

Figure 7: `test_debounce.sv` Output

Listing 4: `test_mult.sv` Stimulus

```

mult dut (.*);

//Test a single calculation
task testNums;
input logic signed ‘MULT_IN_SIZE re_x_in, im_x_in, re_y_in, im_y_in;
input logic signed ‘MULT_OUT_SIZE re_z_in, im_z_in;
begin
    re_x = re_x_in;
    im_x = im_x_in;

    re_y = re_y_in;
    im_y = im_y_in;

    #1ns;

    assert(re_z == re_z_in) else $display(”Error. Expected %x, got %x”, re_z_in, re_z);
    assert(im_z == im_z_in) else $display(”Error. Expected %x, got %x”, im_z_in, im_z);
```

```

end
endtask;
```

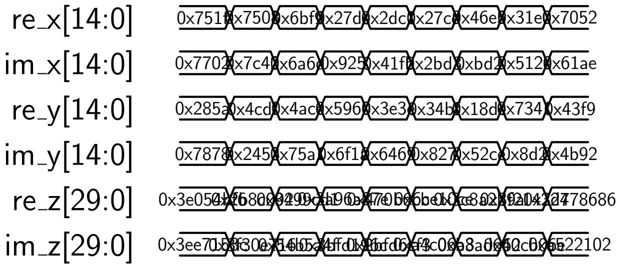

Figure 8: test_mult.sv Output

Listing 5: test_read_sm.sv Stimulus

```
    reset_n = 1;
    # 10ns reset_n = 0;
    # 10ns reset_n = 1;

    forever
        #5ns clk = !clk;
end

// Stimulus
initial
begin
    handshake = 0;
    run = 0;
    #32ns;

    @(posedge clk);
    run = 1;
    @(posedge clk);
    run = 0;
    #30ns handshake = 1;
    @(posedge clk) assert(read);
```

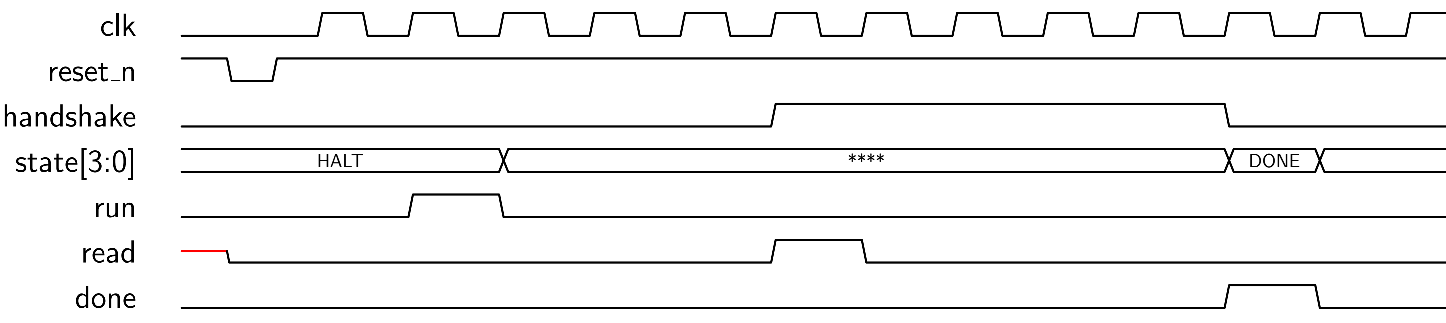

Figure 9: test_read_sm.sv Output

Listing 6: test_sm.sv Stimulus

```
logic ‘WORD_SIZE re_a , im_a , re_q , im_q;

sm dut (.*);

//Assign switches to their fucntions
logic reset_n;
logic handshake;
logic ‘WORD_SIZE data_in;
always_comb
begin
    SW[$high(SW)] = reset_n;
    SW[$high(SW) -1] = handshake;
    SW[$high(SW) -2 : $low(SW)] = data_in;
end

//Clock and reset
initial
begin
    clk = 0;
    reset_n = 1;
    # 10ns reset_n = 0;
```

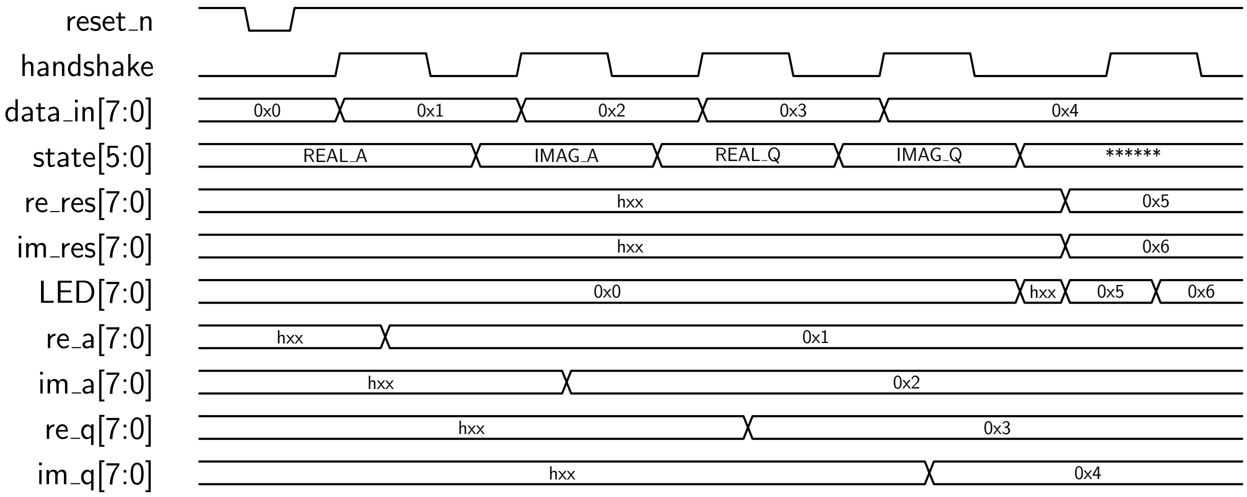

Figure 10: `test_sm.sv` Output

# 4 Synthesis

**Include “Synthesis Result”**

# 5 Conclusion

## References