ECE 5722 Homework 3

By Bruce Huynh

10/8/2021

Introduction:

In this assignment, we were tasked with creating SAYEH assembly code that communicates with the CPU. Alongside that, we were also tasked with creating RTL hardware for the IO interface that the CPU would communicate with.

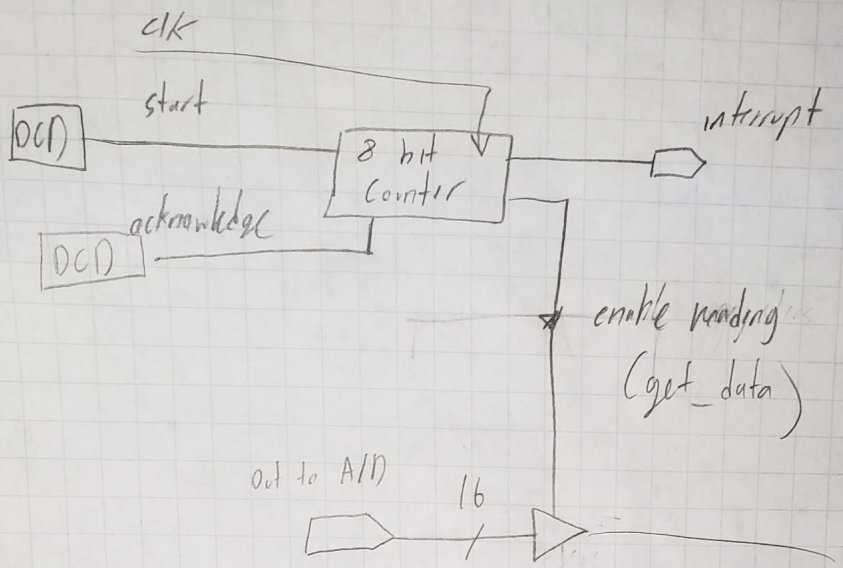
A.) Initialize the A/D interface

```
cwp
wpl
mil r0, 00
mih r0, 50 ; location 0x5000
mil r1, 01
mih r1, 00
mil r2, 00
mih r2, 02 ; location 0x0200 (User Program)
sta r0, r2
```

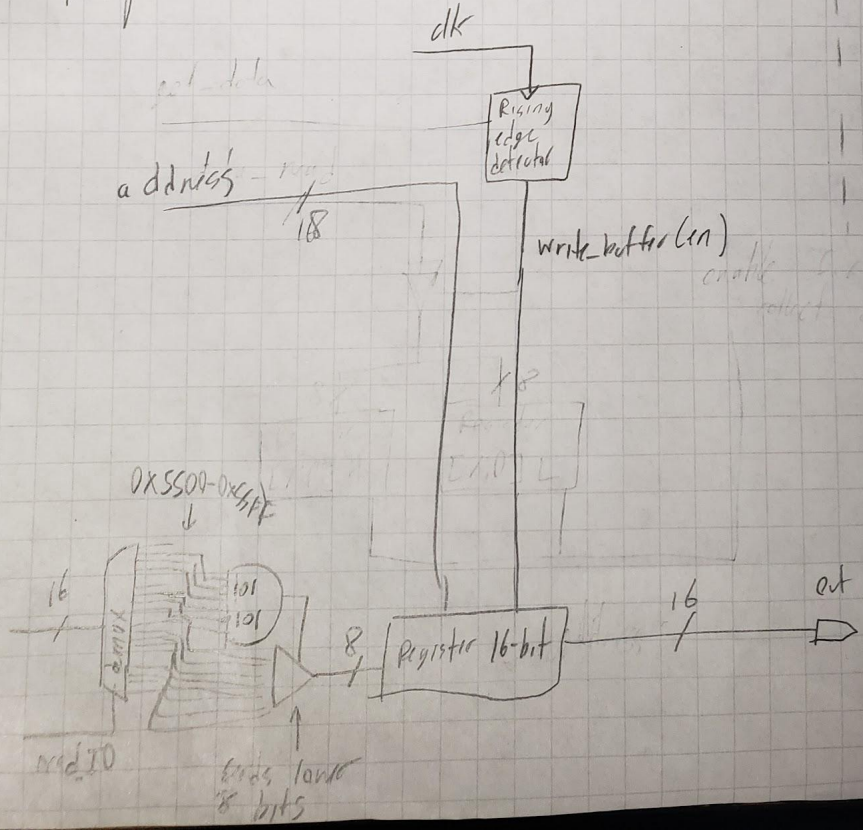
For this code, we write information from the user program into the location 0x5000.

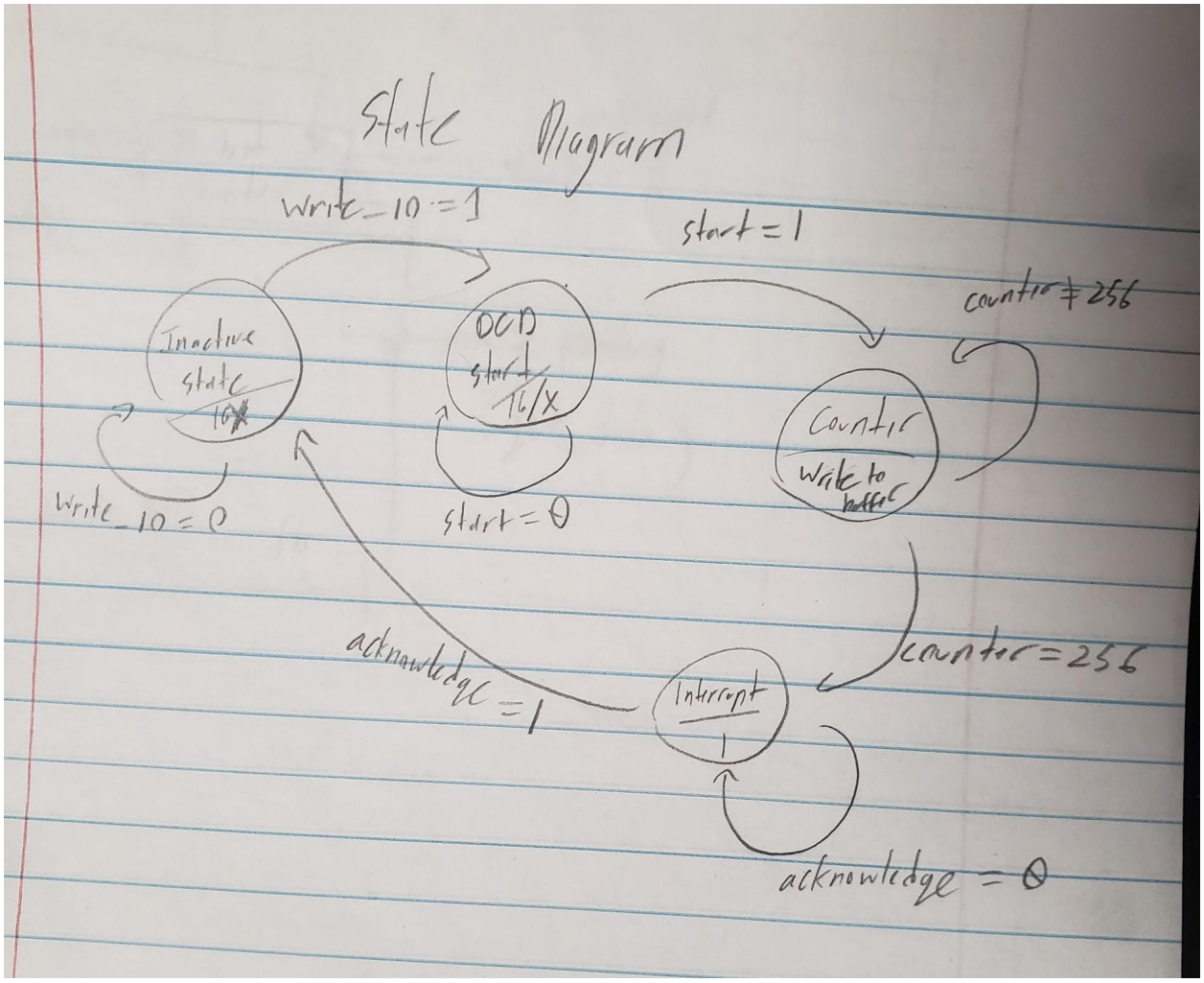
B.) Datapath and Controller

Controller:



Datapath





C.) RTL Verilog Code

```
module ece5722hw3 (input write_IO, read_IO, get_data, clk, rst, input
[15:0] data_in, input [15:0] data_in_DCD, output interrupt, output
[15:0] data_read_out);

wire startWire;
wire acknowledgeWire;
wire [7:0] dataAddOutWire;
wire [7:0] outputToBufferWire;
wire write_BufferW;


     DCD DCDstart(.address(data_in_DCD), .write_IO(write_IO),
.clk(clk), .rst(rst), .start_or_acknowledge_out(startWire));
     DCD DCDacknowledge(.address(data_in_DCD), .write_IO(write_IO),
.clk(clk), .rst(rst), .start_or_acknowledge_out(acknowledgeWire));
     IO_interface myIOI(.get_data(get_data), .start(startWire),
.acknowledged(acknowledgeWire), .clk(clk), .rst(rst),
.data_read(data_in), .dataAddOut(dataAddOutWire),
     .write_Buffer(write_BufferW), .interrupt(interrupt));
     addressDecoder myaddressDecoder(.address(data_in),
.read_IO(read_IO), .clk(clk), .outputToBuffer(outputToBufferWire));


assign data_read_out[15:0] = {outputToBufferWire[7:0],
dataAddOutWire[7:0]};



endmodule


module DCD (input [15:0] address, input write_IO, clk, rst, output
start_or_acknowledge_out);

     reg start_or_acknowledge_out_state;

always @ (posedge clk) begin
```

```
//              if ((address == 16'h5000 | address == 16'h5001) &
write_IO == 1)
                if ((address == 16'h5000 | address == 16'h5001) &
write_IO)
                        start_or_acknowledge_out_state <= 1;
                else
                        start_or_acknowledge_out_state <= 0;

end

assign start_or_acknowledge_out = start_or_acknowledge_out_state;


endmodule



module IO_interface (input start, get_data, acknowledged, clk, rst,
input [15:0] data_read, output interrupt, write_Buffer, output [7:0]
dataAddOut);
      reg [7:0] counter = 0;
      reg [7:0] dataH;
      reg [7:0] dataL;
      reg sendToInterrupt;
      reg write_Buffer_On;

always @ (posedge clk or posedge rst) begin

      if (rst) begin
            counter <= 0;
            dataH <= 8'b00000000;
            write_Buffer_On <= 0;
            sendToInterrupt <= 0;
      end
      else if (acknowledged == 1 & start != 1)
            counter <= 0;
      else if (start & get_data) begin
            counter <= counter + 1'b1;
                  if (counter != 8'b11111111) begin
```

```
                    dataH <= data_read[15:8];
                    write_Buffer_On <= 1;
                end
                else if (counter == 8'b11111111) begin
                    write_Buffer_On <= 0;
                    sendToInterrupt <= 1;
                    counter <= 8'b11111111;
                end



            end
        end


assign interrupt = sendToInterrupt;
assign dataAddOut = dataH;
assign write_Buffer = write_Buffer_On;


endmodule









module addressDecoder (input read_IO, clk, input [15:0] address,
output [7:0] outputToBuffer);
reg outState;
always @ (posedge clk) begin
    if ((address >= 16'h5500 & address <= 16'h55FF) & read_IO)
        outState <= 1;
    else
        outState <= 0;
end

assign outputToBuffer = outState ? address[7:0] : 8'h00;
```

```
endmodule
```

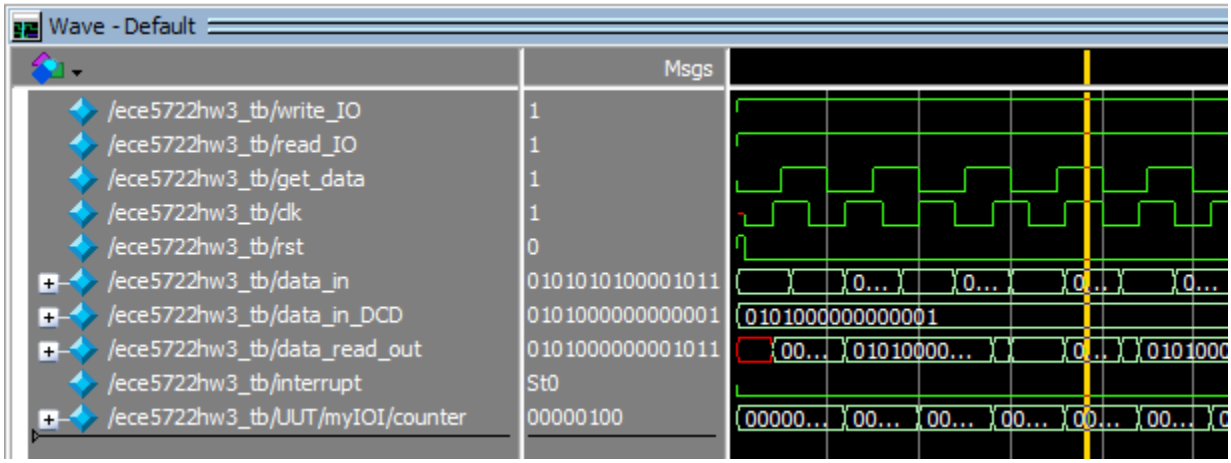
D.) Testbench

There is one testbench that was written for this circuit but it tests many different aspects of the design to make sure it functions correctly.

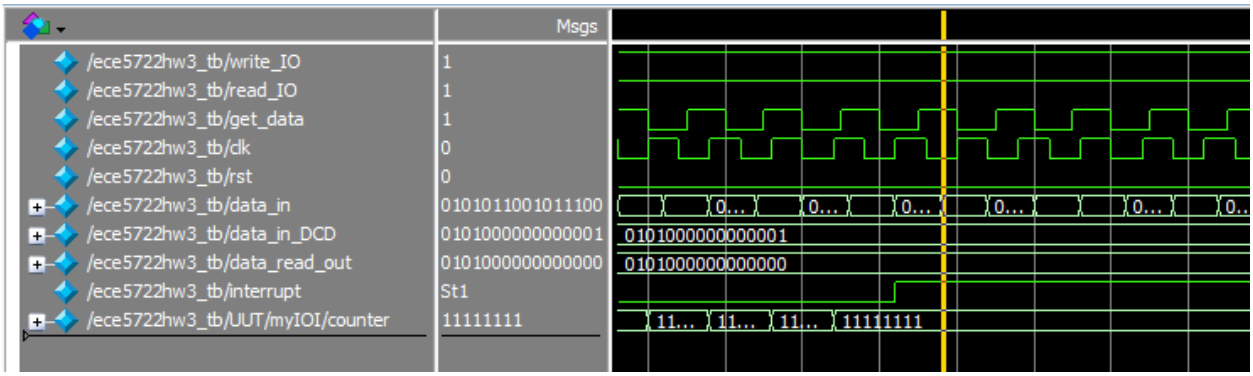
Testbench code:

```
module ece5722hw3_tb ();

reg write_IO, read_IO, get_data, clk, rst;
reg [15:0] data_in;
reg [15:0] data_in_DCD;
wire [15:0] data_read_out;
wire interrupt;



ece5722hw3 UUT (.write_IO(write_IO), .read_IO(read_IO),
.get_data(get_data), .clk(clk), .rst (rst), .data_in(data_in),
.data_in_DCD(data_in_DCD),
.data_read_out(data_read_out), .interrupt(interrupt));



initial begin

data_in <= 16'h5505;
data_in_DCD <= 16'h5001;
get_data <= 0;
```

```
write_IO <= 1;
read_IO <= 1;
rst <= 1;
#5 rst <= 0;
clk <= 0;

end

always #30 data_in <= data_in + 1;
always #20 clk = ~clk;
always #25 get_data = ~get_data;


endmodule
```



From this first image, you can see that data_read_out takes the lower 8 bits from the address decoder and it takes the upper 8 bits from the DCD. The information being output only gets done when there is a pulse on get_data.



In this image, when the counter reaches 256, the data will not be read anymore, even though data is still being sent in. Then, the interface will send out an interrupt signal. Once the CPU gets the interrupt, it will send an acknowledge signal to reset the circuit.

E.) ISR

```
cwp
wpl
mil r0, 00
mih r0, 00
mil r1, 01 ; constant 1 for wpl
mih r1, 00
mil r2, 00
mih r2, 01 ; 0x0100 = 256
wph
mil r4, 01
mih r4, 00 ; r4 = constant 1
mil r5, 00
mih r5, 09 ; write to address 0x0900
mil r6, FF
mih r6, 09 ; address 0x09FF
mil r7, 01
mih r7, 50 ; location 0x5001
L10

wph
sta r7, r4 ; store data from memory address 0x0001 to location 0x5001
```

```
sta r5, r7 ; store data from memory address 0x5001 to location 0x09XX

;L02
add r5, r4 ; add 1 to r5

wpl
add r0, r1 ; add 1 to r0
brc L2          ; break if carry
cmp r2, r0 ; compare stored to 256
inte
oup
awp 8      ; add an extra window pointer tab
awp 9      ; add an extra window pointer tab
awp 10          ; add an extra window pointer tab
mil r0, 00
mih r0, 55
mil r1, FF
mih r1, 55
mil r2, 01
mih r2, 00
add r0, r2
cmp r1, r2
oup r1
intd
brc L2          ; stop if done
jpr L10         ; otherwise jump back to L10

;L2
```

For the ISR, we keep adding data to 0x5500, and compare it to the max value of 0x55FF. If it has not reached this value and if the counter is not 256, we keep going.

Conclusion:

This assignment was the most difficult assignment to date as it tested our knowledge of SAYEH assembly and had us combine it with RTL design. It was good practice with working with addresses and the larger idea of system level design.