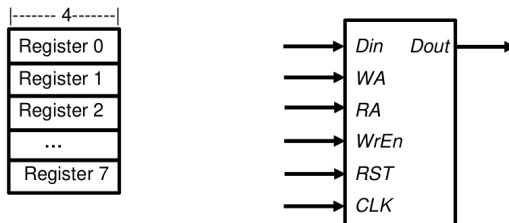


Abiria Placide
Course: CSE 4356-001
Prof. Bill Carroll
Sept 14, 2021

Design requirements

The goal for this lab is to design a 4x8 register file. This means that each register will have a width of 4 and a total of 8 registers. The end result is that we are able to read and write to and from those registers.



The inputs include:

Din: input data.

WA: write address.

RA: read address.

WE: write enable.

RST: reset register to zero.

CLK: read on negedge of the clock.

The outputs include:

Dout = LEDR 0-4 and HEX0 outputs.

Required Pin assignments on FPGA:

Din – SW3, SW2, SW1, SW0

Dout – LEDR3, LEDR2, LEDR1, LEDR0, HEX0

*WA*₂₋₀ – SW6, SW5, SW4

*RA*₂₋₀ – SW9, SW8, SW7

WrEn – Key3

RST – Key0

CLK – Key1

```
//Verilog code

module RegisterFile #(parameter WIDTH = 4, DEPTH = 8) (

    input [WIDTH-1:0] Din,
    input [2:0] WA, RA,
    input RST, CLK, WR_ENABLE,
    output [WIDTH-1:0] Dout,
    output [0:6] Hex0

);

/* wires and registers holding data being read */

//wire Decode0, Decode1, Decode3;
wire [7:0] SelectReg;
wire [3:0] Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7; //register outputs if selected

/* Instantiations */
Decoder3by8 decoder(
    .usr_input(WA),
    .ENABLE(WR_ENABLE),
    .Dout(SelectReg) //wrong?
);

NbitRegister Reg0(
    .Data_in(Din),
    .Clock(CLK),
    .Clear(RST),
    .WR_ENABLE(SelectReg[0]),
    .Data_out(Q0)
);

NbitRegister Reg1(
    .Data_in(Din),
    .Clock(CLK),
```

```
        .Clear(RST),  
        .WR_ENABLE(SelectReg[1]),  
        .Data_out(Q1)  
    );
```

```
NbitRegister Reg2(  
    .Data_in(Din),  
    .Clock(CLK),  
    .Clear(RST),  
    .WR_ENABLE(SelectReg[2]),  
    .Data_out(Q2)  
);
```

```
NbitRegister Reg3(  
    .Data_in(Din),  
    .Clock(CLK),  
    .Clear(RST),  
    .WR_ENABLE(SelectReg[3]),  
    .Data_out(Q3)  
);
```

```
NbitRegister Reg4(  
    .Data_in(Din),  
    .Clock(CLK),  
    .Clear(RST),  
    .WR_ENABLE(SelectReg[4]),  
    .Data_out(Q4)  
);
```

```
NbitRegister Reg5(  
    .Data_in(Din),  
    .Clock(CLK),  
    .Clear(RST),  
    .WR_ENABLE(SelectReg[5]),  
    .Data_out(Q5)  
);
```

```
NbitRegister Reg6(  
    .Data_in(Din),  
    .Clock(CLK),  
    .Clear(RST),  
    .WR_ENABLE(SelectReg[6]),  
    .Data_out(Q6)  
);
```

```

        .Data_in(Din),
        .Clock(CLK),
        .Clear(RST),
        .WR_ENABLE(SelectReg[6]),
        .Data_out(Q6)
    );
    NbitRegister Reg7(
        .Data_in(Din),
        .Clock(CLK),
        .Clear(RST),
        .WR_ENABLE(SelectReg[7]),
        .Data_out(Q7)
    );

    EightxOneMux Mux0(
        .Din0(Q0),
        .Din1(Q1),
        .Din2(Q2),
        .Din3(Q3),
        .Din4(Q4),
        .Din5(Q5),
        .Din6(Q6),
        .Din7(Q7),
        .RA(RA),
        .RDout(Dout)
    );

    binary2seven hex0(
        .BIN(Dout),
        .SEV(Hex0)
    );

//4 y
endmodule

```

3 X 8 Decoder

```
module Decoder3by8(  
    input  [2:0] usr_input,  
    input  ENABLE,  
    output reg [7:0] Dout  
);  
  
always @( * )  
begin  
    Dout = 8'd0; //will reset the Dout to 0 for a new selection  
    if(ENABLE)  
    begin  
        case(usr_input)  
            3'd0: begin Dout[0] = 1'd1; end  
            3'd1: begin Dout[1] = 1'd1; end  
            3'd2: begin Dout[2] = 1'd1; end  
            3'd3: begin Dout[3] = 1'd1; end  
            3'd4: begin Dout[4] = 1'd1; end  
            3'd5: begin Dout[5] = 1'd1; end  
            3'd6: begin Dout[6] = 1'd1; end  
            3'd7: begin Dout[7] = 1'd1; end  
  
            default : Dout = 8'd0;  
        endcase  
    end  
end  
  
endmodule
```

8 to 1 Mux

```
module EightxOneMux #(parameter WIDTH=2, DEPTH = 4)(
    input [2:0] RA,
    input [DEPTH-1:0] Din0,Din1,Din2,Din3,Din4,Din5,Din6,Din7, //4
    registers of size 8 bits
    output reg [DEPTH-1:0] RDout
);

always @( * )
begin
    RDout = 4'd0; //reset on every read select
    case (RA)
        3'd0: begin RDout = Din0; end
        3'd1: begin RDout = Din1; end
        3'd2: begin RDout = Din2; end
        3'd3: begin RDout = Din3; end
        3'd4: begin RDout = Din4; end
        3'd5: begin RDout = Din5; end
        3'd6: begin RDout = Din6; end
        3'd7: begin RDout = Din7; end

        default: RDout = 4'd0;
    endcase
end
endmodule
```

N-Bit Register

```
module NbitRegister #(parameter N = 4)
( input [N-1:0] Data_in,
  input Clock, Clear, WR_ENABLE,
  output reg [N-1:0] Data_out
);

always @(negedge Clock )
begin
    if(WR_ENABLE == 1'b1)
    begin
        if(Clear == 1'b0) Data_out <= 0;
        else if(Clock== 1'b0) Data_out <= Data_in;
        end
    end
end

endmodule
```

Binary-2-Seven Decoder

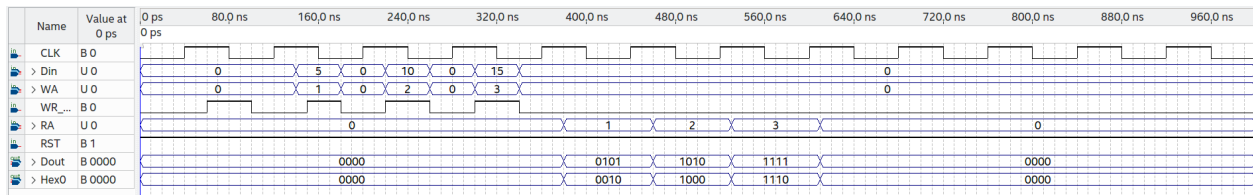
```
module binary2seven(input [3:0] BIN, output reg[0:6] SEV);

always @(BIN)
    case ({BIN[3:0]})
        4'b0000: {SEV[0:6]} = 7'b0000001; //0
        4'b0001: {SEV[0:6]} = 7'b1001111; //1
        4'b0010: {SEV[0:6]} = 7'b0010010; //2
        4'b0011: {SEV[0:6]} = 7'b0000110; //3
        4'b0100: {SEV[0:6]} = 7'b1001100; //4
        4'b0101: {SEV[0:6]} = 7'b0100100; //5
        4'b0110: {SEV[0:6]} = 7'b0100000; //6
        4'b0111: {SEV[0:6]} = 7'b0001111; //7
        4'b1000: {SEV[0:6]} = 7'b0000000; //8
        4'b1001: {SEV[0:6]} = 7'b0001100; //9
        4'b1010: {SEV[0:6]} = 7'b0001000; //A
        4'b1011: {SEV[0:6]} = 7'b1100000; //b
    endcase
end
```



























```
4'b1100: {SEV[0:6]} = 7'b0110001; //C
4'b1101: {SEV[0:6]} = 7'b1000010; //D
4'b1110: {SEV[0:6]} = 7'b0110000; //E
4'b1111: {SEV[0:6]} = 7'b0111000; //F
endcase
endmodule
```

Simulation results



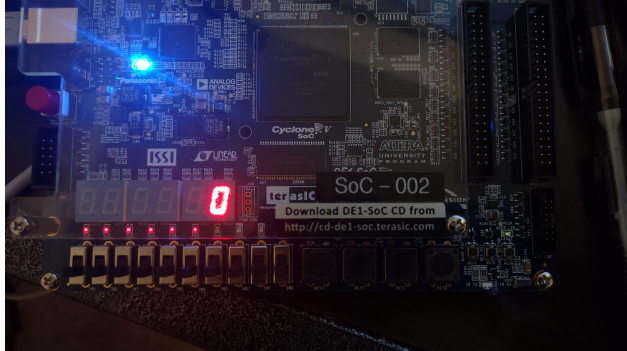
After writing to registers 0-3, we are able to observe the outputs on Dout and Hex0 being read from the registers depending on the address being selected.

Pin assignments

Node Name	Direction	Location	I/O Bank	VREF Group	Pin Location	I/O Standard	Reserved	Current Strength	Slew Rate
 CLK	Input	PIN_AA15	3B	B3B_NO	PIN_AA15	2.5 V		12mA...ult)	
 Din[3]	Input	PIN_AF10	3A	B3A_NO	PIN_AF10	2.5 V		12mA...ult)	
 Din[2]	Input	PIN_AF9	3A	B3A_NO	PIN_AF9	2.5 V		12mA...ult)	
 Din[1]	Input	PIN_AC12	3A	B3A_NO	PIN_AC12	2.5 V		12mA...ult)	
 Din[0]	Input	PIN_AB12	3A	B3A_NO	PIN_AB12	2.5 V		12mA...ult)	
 Dout[3]	Output	PIN_V18	4A	B4A_NO	PIN_V18	2.5 V		12mA...ult)	1 (default)
 Dout[2]	Output	PIN_V17	4A	B4A_NO	PIN_V17	2.5 V		12mA...ult)	1 (default)
 Dout[1]	Output	PIN_W16	4A	B4A_NO	PIN_W16	2.5 V		12mA...ult)	1 (default)
 Dout[0]	Output	PIN_V16	4A	B4A_NO	PIN_V16	2.5 V		12mA...ult)	1 (default)
 Hex0[0]	Output	PIN_AE26	5A	B5A_NO	PIN_AE26	2.5 V		12mA...ult)	1 (default)
 Hex0[1]	Output	PIN_AE27	5A	B5A_NO	PIN_AE27	2.5 V		12mA...ult)	1 (default)
 Hex0[2]	Output	PIN_AE28	5A	B5A_NO	PIN_AE28	2.5 V		12mA...ult)	1 (default)
 Hex0[3]	Output	PIN_AG27	5A	B5A_NO	PIN_AG27	2.5 V		12mA...ult)	1 (default)
 Hex0[4]	Output	PIN_AF28	5A	B5A_NO	PIN_AF28	2.5 V		12mA...ult)	1 (default)
 Hex0[5]	Output	PIN_AG28	5A	B5A_NO	PIN_AG28	2.5 V		12mA...ult)	1 (default)
 Hex0[6]	Output	PIN_AH28	5A	B5A_NO	PIN_AH28	2.5 V		12mA...ult)	1 (default)
 RA[2]	Input	PIN_AE12	3A	B3A_NO	PIN_AE12	2.5 V		12mA...ult)	
 RA[1]	Input	PIN_AD10	3A	B3A_NO	PIN_AD10	2.5 V		12mA...ult)	
 RA[0]	Input	PIN_AC9	3A	B3A_NO	PIN_AC9	2.5 V		12mA...ult)	
 RST	Input	PIN_AA14	3B	B3B_NO	PIN_AA14	2.5 V		12mA...ult)	
 WA[2]	Input	PIN_AE11	3A	B3A_NO	PIN_AE11	2.5 V		12mA...ult)	
 WA[1]	Input	PIN_AD12	3A	B3A_NO	PIN_AD12	2.5 V		12mA...ult)	
 WA[0]	Input	PIN_AD11	3A	B3A_NO	PIN_AD11	2.5 V		12mA...ult)	
 WR_ENABLE	Input	PIN_Y16	3B	B3B_NO	PIN_Y16	2.5 V		12mA...ult)	

DE1-SoC test results and photos/videos.

Using the input patterns 0000, 0101, 1010, 1111, we are able to write them to selected registers and select which register to read from. Data being read is reflected on Hex0 and LEDR0-3.



- Registers can be reset by picking an address, holding down reset(KEY0), and pressing clk (KEY1).

- Writing to a register can be done by picking an address through switches 6-4. Data can be written to a register using switches 0-3, then pressing clk (KEY1).

