CS 223 Digital Design

Bilkent University

Fall 2024-25

Laboratory Assignment 4

Multifunction Register and 7-Segment Display

Nabeeha Khan

22301304

Section: 03

Date: 25th November 2024

Multifunction Register:

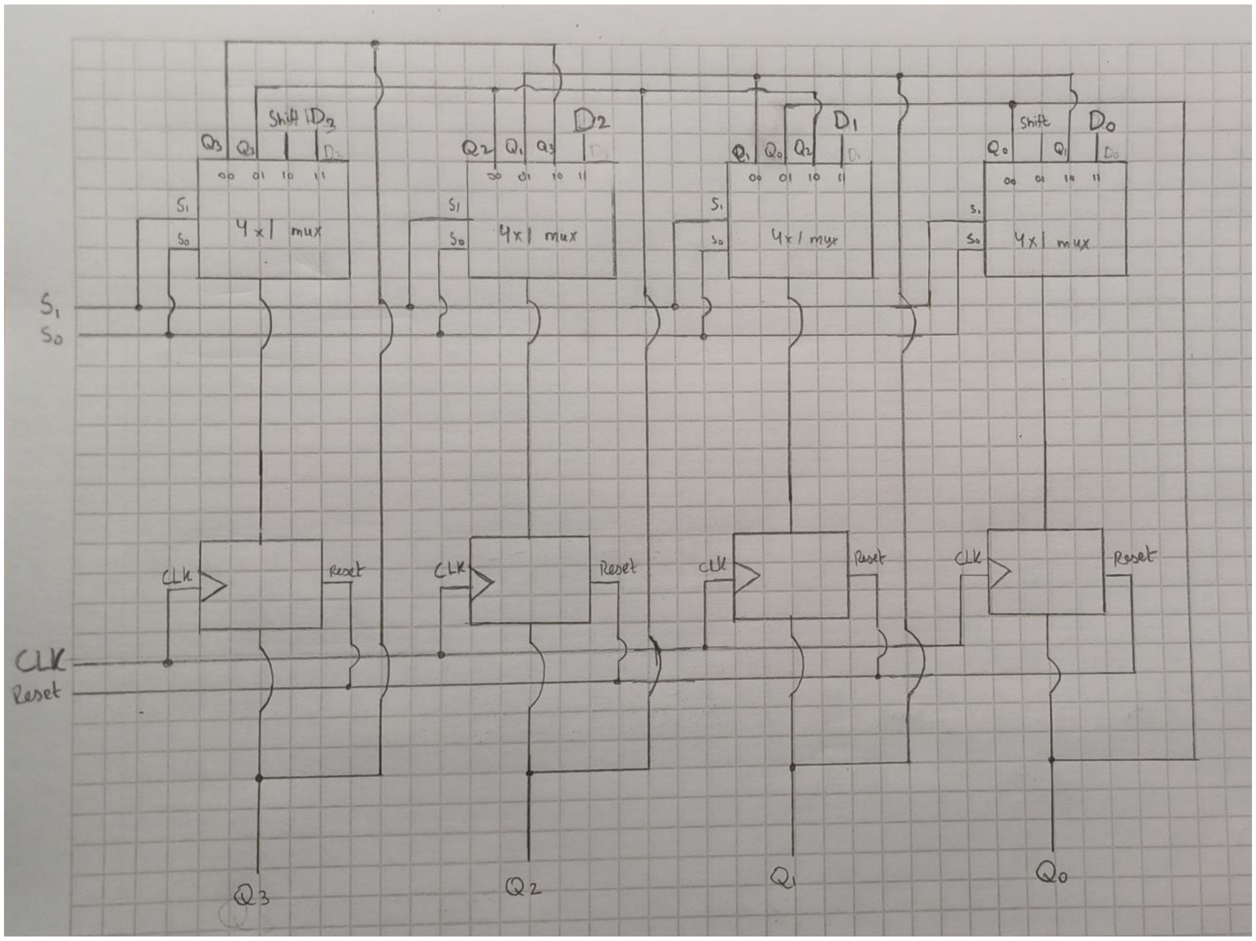
1. SystemVerilog Module for D flip-flop:

```
module dflipflop( input logic clk, reset, d, output logic q);

        always_ff @(posedge clk)
        begin
        if (reset)   q <= 1'b0;
        else          q <= d;
        end

endmodule
```

2. Circuit Schematic:



3. Structural SystemVerilog Module:

```
//4 to 1 mux module for register

module mux4to1(input logic I0, I1, I2, I3, S0, S1, output logic Y);

   logic l0, l1;

   assign l0 = (~S0 & I0) | (S0 & I1);

   assign l1 = (~S0 & I2) | (S0 & I3);

   assign Y = (~S1 & l0) | (S1 & l1);

endmodule


//Structural module for multiple register

module multifunctionReg(input logic d0, d1, d2, d3, S0, S1, shiftin, clk, reset, output logic q3,
q2, q1, q0);

         logic l0, l1, l2, l3;

         mux4to1 mux1(q3, q2, shiftin, d3, S0, S1, l3);

         mux4to1 mux2(q2, q1, q3, d2, S0, S1, l2);

         mux4to1 mux3(q1, q0, q2, d1, S0, S1, l1);

         mux4to1 mux4(q0, shiftin, q1, d0, S0, S1, l0);

         dflipflop dff1(clk, reset, l3, q3);

         dflipflop dff2(clk, reset, l2, q2);

         dflipflop dff3(clk, reset, l1, q1);

         dflipflop dff4(clk, reset, l0, q0);

endmodule


//Clock Divider

module clock_cycle( input clk, input rst, output logic clk_div );

         localparam constantNumber = 150000000;
         logic [31:0] count;

         always @(posedge(clk), posedge(rst))
         begin
            if (rst == 1'b1)
                count <= 32'b0;
            else if (count == constantNumber - 1)
                count <= 32'b0;
            else
                count <= count + 1;
         end

         always @ (posedge(clk), posedge(rst))
         begin
            if (rst == 1'b1)
                clk_div <= 1'b0;
            else if (count == constantNumber - 1)
                clk_div <= ~clk_div;
            else
                clk_div <= clk_div;
         end
endmodule
```

4. Testbench:

```
module multifunctionReg_TB();
        logic d0, d1, d2, d3, S0, S1, shiftin, clk, reset, q3, q2, q1, q0;
        multifunctionReg dut(d0, d1, d2, d3, S0, S1, shiftin, clk, reset, q3, q2, q1, q0);


        //Initial clock
        initial begin clk = 0;
                forever #5 clk = ~clk;
        end


        initial begin
                //Reseting the flip-flops
                reset = 1; S1 = 0; S0 = 0; shiftin = 0; d3 = 0; d2 = 0; d1 = 0; d0 = 0; #100;
                //Case 1: maintain current value
                reset = 0; S1 = 0; S0 = 0; shiftin = 0; d3 = 1; d2 = 1; d1 = 1; d0 = 1; #100;
                //Case 2: left shift
                reset = 0; S1 = 0; S0 = 1; shiftin = 0; d3 = 0; d2 = 0; d1 = 0; d0 = 0; #100;
                reset = 0; S1 = 0; S0 = 1; shiftin = 1; d3 = 0; d2 = 0; d1 = 0; d0 = 0; #100;
                //Case 3: right shift
                reset = 0; S1 = 1; S0 = 0; shiftin = 0; d3 = 0; d2 = 0; d1 = 0; d0 = 0; #100;
                reset = 0; S1 = 1; S0 = 0; shiftin = 1; d3 = 0; d2 = 0; d1 = 0; d0 = 0; #100;
                //Case 4: Parallel load
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 0; d2 = 0; d1 = 0; d0 = 0; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 0; d2 = 0; d1 = 0; d0 = 1; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 0; d2 = 0; d1 = 1; d0 = 0; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 0; d2 = 0; d1 = 1; d0 = 1; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 0; d2 = 1; d1 = 0; d0 = 0; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 0; d2 = 1; d1 = 0; d0 = 1; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 0; d2 = 1; d1 = 1; d0 = 0; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 0; d2 = 1; d1 = 1; d0 = 1; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 1; d2 = 0; d1 = 0; d0 = 0; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 1; d2 = 0; d1 = 0; d0 = 1; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 1; d2 = 0; d1 = 1; d0 = 0; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 1; d2 = 0; d1 = 1; d0 = 1; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 1; d2 = 1; d1 = 0; d0 = 0; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 1; d2 = 1; d1 = 0; d0 = 1; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 1; d2 = 1; d1 = 1; d0 = 0; #100;
                reset = 0; S1 = 1; S0 = 1; shiftin = 0; d3 = 1; d2 = 1; d1 = 1; d0 = 1; #100;
        end
endmodule
```

Hexadecimal to 7-Segment Decoder:

1. SystemVerilog Module:

```
module hexto7segment (input logic [3:0] hex, output logic [6:0] segments);
	always_comb begin
		 case (hex)
		4'h0: segments = 7'b0000001;
		4'h1: segments = 7'b1001111;
		4'h2: segments = 7'b0010010;
		4'h3: segments = 7'b0000110;
		4'h4: segments = 7'b1001100;
		4'h5: segments = 7'b0100100;
		4'h6: segments = 7'b0100000;
		4'h7: segments = 7'b0001111;
		4'h8: segments = 7'b0000000;
		4'h9: segments = 7'b0000100;
		4'hA: segments = 7'b0001000;
		4'hB: segments = 7'b1100000;
		4'hC: segments = 7'b0110001;
		4'hD: segments = 7'b1000010;
		4'hE: segments = 7'b0110000;
		4'hF: segments = 7'b0111000;
		default: segments = 7'b1111111; // Default all off
		endcase
	end
endmodule
```

2. Testbench:

```
module hexto7segment_TB ();
	logic [3:0] hex;
```

```
        logic [6:0] segments;
        hexto7segment dut(hex, segments);
        initial begin
        for(int i = 0; i < 16; i++) begin
        hex = 4’hi; #10;
end
end
endmodule
```

3. Advantages/Disadvantages of Hex number representation:

- Since the hex numbers require less digits, it makes input/ output handling easier [1].
- Since each hex digit corresponds to 4 bits, the conversion from hex to binary is easy, which is useful in computer programming [1].
- We cannot perform complex mathematical operations like multiplication and division [1].

4. Implementation on Basys3:

According to reference manual, the Basys 3 board's four-digit seven-segment display works by quickly lighting up one digit at a time in a repeating cycle, a technique called **multiplexing**. Each digit shares the same control lines for the segments (`CA` to `CG`), but only one digit is active at a time through its unique anode signal (`AN0` to `AN3`). The display refreshes rapidly (at least 60 times per second) to make all digits appear continuously lit to the human eye [2].

To drive this 4-digit 7-segment display, we would require a shift register to switch between the anodes, a 7-segment decoder to display each digit, and a refresh counter to control the speed of the cycle.

References:

1. What is hexadecimal numbering?
2. Basys 3 Reference Manual - Digilent Reference