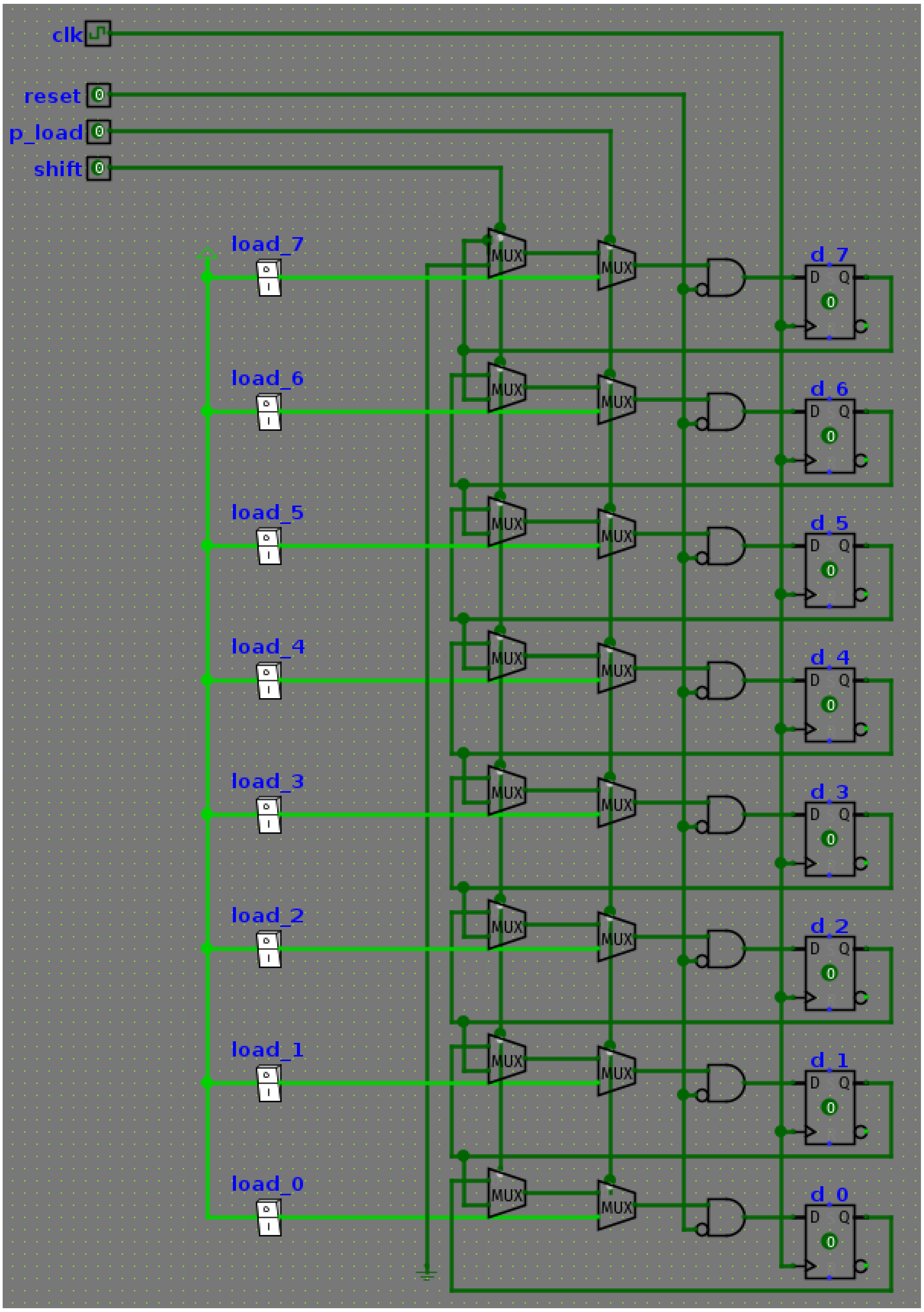
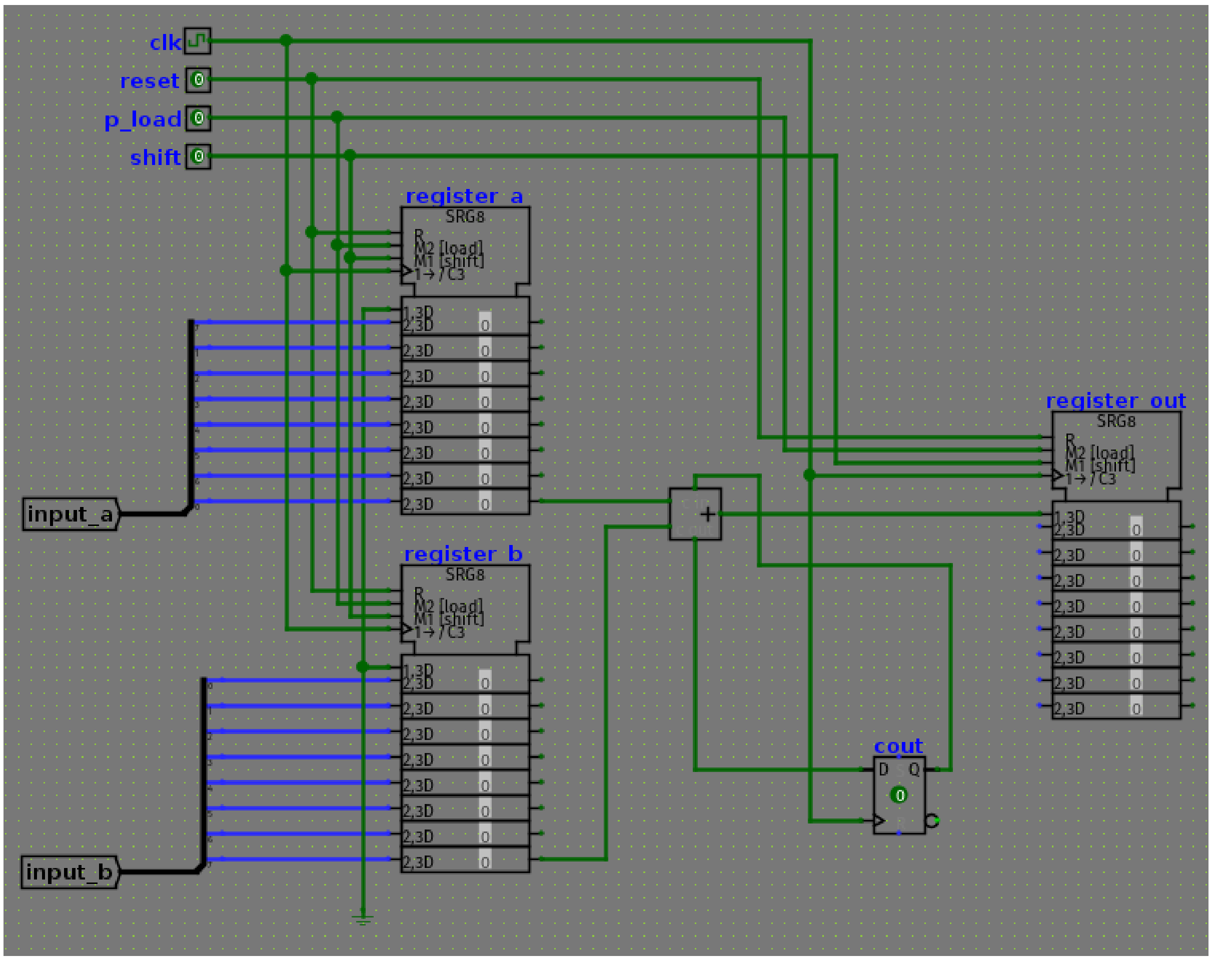
CS223
Digtal Design
Sec. 1
Lab 5
Erkin Aydın
ID: 22002956
5/12/2022

Circuit schematic for your shift register design using D flip-flops:



Circuit schematic for your serial adder using the shift registers, full adder, and D flip-flop:



SystemVerilog module for synchronously resettable D flip-flop:

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

        logic q_next;
        and a(q_next, d, ~reset);

        always @(posedge clk)
        q<=q_next;

endmodule
```

Structural SystemVerilog module for your shift register using the D flip-flop module along with the testbench:

Shift Register Module(with DFF):

```
module shift_reg(
        input logic shift, p_load, reset, clk,
        input logic s_bit,
        input logic[7:0] val_next,
        output logic[7:0] val);

        logic out_7_1, out_6_1, out_5_1, out_4_1, out_3_1, out_2_1, out_1_1, out_0_1;
        logic out_7_2, out_6_2, out_5_2, out_4_2, out_3_2, out_2_2, out_1_2, out_0_2;
        logic d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0;

        mux2 mux7_1(s_bit,d_7,shift, out_7_1);
        mux2 mux7_2(val_next[7], out_7_1, p_load, out_7_2);
        DFF reg7(out_7_2, clk, reset, d_7);

        mux2 mux6_1(d_7, d_6, shift, out_6_1);
        mux2 mux6_2(val_next[6], out_6_1, p_load, out_6_2);
        DFF reg6(out_6_2, clk, reset, d_6);

        mux2 mux5_1(d_6, d_5, shift, out_5_1);
        mux2 mux5_2(val_next[5], out_5_1, p_load, out_5_2);
        DFF reg5(out_5_2, clk, reset, d_5);

        mux2 mux4_1(d_5, d_4, shift, out_4_1);
        mux2 mux4_2(val_next[4], out_4_1, p_load, out_4_2);
        DFF reg4(out_4_2, clk, reset, d_4);

        mux2 mux3_1(d_4, d_3, shift, out_3_1);
        mux2 mux3_2(val_next[3], out_3_1, p_load, out_3_2);
        DFF reg3(out_3_2, clk, reset, d_3);

        mux2 mux2_1(d_3, d_2, shift, out_2_1);
        mux2 mux2_2(val_next[2], out_2_1, p_load, out_2_2);
        DFF reg2(out_2_2, clk, reset, d_2);

        mux2 mux1_1(d_2, d_1, shift, out_1_1);
        mux2 mux1_2(val_next[1], out_1_1, p_load, out_1_2);
        DFF reg1(out_1_2, clk, reset, d_1);

        mux2 mux0_1(d_1, d_0, shift, out_0_1);
        mux2 mux0_2(val_next[0], out_0_1, p_load, out_0_2);
        DFF reg0(out_0_2, clk, reset, d_0);
```

```
	assign val[7] = d_7;
	assign val[6] = d_6;
	assign val[5] = d_5;
	assign val[4] = d_4;
	assign val[3] = d_3;
	assign val[2] = d_2;
	assign val[1] = d_1;
	assign val[0] = d_0;

endmodule

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

	logic q_next;
	and a(q_next, d, ~reset);

	always @(posedge clk)
	q<=q_next;

endmodule

module mux2 (i1, i0, s, d);
	input i1, i0, s;
	output d;
	assign d = ~s&i0 | s&i1;
endmodule
```

Testbench for Shift Register:

```
module shift_reg_tb();

	logic shift, p_load, reset, clk;
	logic[7:0] val_next;
	logic[7:0] out;
	logic zero = 0;
	always begin
	clk <= 0; #10;
	clk <= 1; #10;
	end

	shift_reg rh_reg(shift, p_load, reset, clk, zero, val_next, out);

	initial begin
	assign shift = 0;
	assign p_load = 0;
```

```
        assign reset = 0;

        assign val_next = 8'b0000_0000;
        assign p_load = 1; #30;
        assign p_load = 0; #30;

        assign val_next = 8'b1101_0010; #5;
        assign p_load = 1; #60;

        assign p_load = 0; #60;

        assign shift = 1; #80;

        assign shift = 0;
        assign val_next = 8'b1111_1111; #5;
        assign p_load = 1; #20;
        assign p_load = 0; #15;
        assign reset = 1;

        end

endmodule
```

<u>Structural SystemVerilog module for your serial adder using the shift register, full adder, and D flip-flop modules along with the testbench:</u>

<u>Modules:</u>

```
module serial_adder(input logic shift, p_load, reset, clk,
                input logic[7:0] load_a,
                input logic [7:0] load_b,
                output logic[7:0] reg_out_val,
                output logic cout);

                logic s, cout_next;
                logic[7:0] reg_a_val;
                logic[7:0] reg_b_val;
                logic Clk;
                logic zero = 0;
                shift_reg reg_a(shift, p_load, reset, clk, zero, load_a, reg_a_val);
                shift_reg reg_b(shift, p_load, reset, clk, zero, load_b, reg_b_val);
                shift_reg reg_out(shift, p_load, reset, clk, s, 8'b0000_0000, reg_out_val);

                full_adder fa(reg_a_val[0], reg_b_val[0], cout, s, cout_next);

                DFF df(cout_next, clk, reset, cout);
```

```
			//Clock_divider clk_div(clk, Clk);

endmodule
/*
//Directly taken from lab_4
module Clock_divider(clock_in,clock_out
	);
	input clock_in; // input clock on FPGA
	output reg clock_out; // output clock after dividing the input clock by divisor
	reg[27:0] counter=28'd0;
	parameter DIVISOR = 28'd1_500_00000;
	always @(posedge clock_in)
	begin
	counter <= counter + 28'd1;
	if(counter>=(DIVISOR-1))
	counter <= 28'd0;

	clock_out <= (counter<DIVISOR/2)?1'b1:1'b0;

	end
endmodule
*/

//This part is directly taken from Lab2...
module full_adder(input a, b, cin, output s, cout);

	logic s_1, cout_1, s_2, cout_2;

	half_adder ha_1(a, b, s_1, cout_1);
	half_adder ha_2(cin, s_1, s_2, cout_2);

	assign s = s_2;
	or o( cout, cout_1, cout_2);

endmodule

//This part is directly taken from Lab2...
module half_adder(input a, b, output s, c);

	assign s = a ^ b;
	assign c = a & b;

endmodule

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

```
        logic q_next;
        and a(q_next, d, ~reset);

        always @(posedge clk)
        q<=q_next;

endmodule
```

Testbench:

```
module serial_adder_tb();

        logic shift, p_load, reset, clk;
        logic[7:0] load_a;
        logic[7:0] load_b;
        logic[7:0] reg_out_val;
        logic cout;

        serial_adder sa(shift, p_load, reset, clk, load_a, load_b, reg_out_val, cout);

        always begin

        assign clk = 0; #10;
        assign clk = 1; #10;

        end

        initial begin

        assign shift = 0;
        assign p_load = 0;
        assign reset = 0;
        assign reset = 1; #10;
        assign reset = 0; #10;
        assign load_a = 8'b0100_1010;
        assign load_b = 8'b0011_0011; #5;
        assign p_load = 1; #20;
        assign p_load = 0; #10;
        assign shift = 1; #300;
        assign shift = 0; #20;
        end

endmodule
```