LAB 3:

Hierarchical Modeling

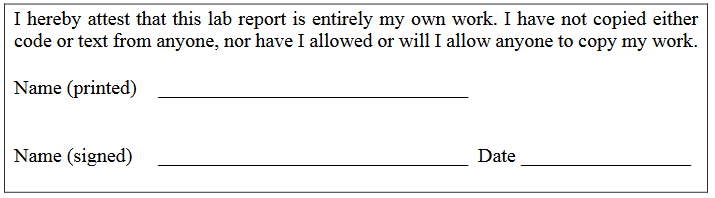
Professor Ronald Mehler

ECE 526L

Spring 2018

Garen Nikoyan

2/15/2018



**Objective:**

The purpose of this lab is to become familiar with utilizing multiple modules and integrating them into a single design, in this case, a register. An SR Latch module was used to create a D flip-flop module, which was then used alongside a 2 input 1 output multiplexer to create a register.

**Methodology:**

For this lab, the first step was to create the module for an SR latch, following the diagram found below. Gate time delays, as well as fanout delays, were incorporated into the design. One big difference that had to be made in order for the latch to work properly was that parameters had to be added into the code to make the SR Latch easily adaptable to other designs. This allowed the SR latch module to be applied and instantiated three times within the D flip flop module, with different input and output delays each time. The parameters allowed not only to change the delays for the primitives at the time of instantiation, but if the delays didn’t need to be changed, then they would result back to their original delay, which came from the original design/code. Once the SR Latches were taken care of, the next step was to create a 2 to 1 MUX along with a DFF. This allowed for the instantiation of the MUX and DFF 8 times to create an 8-bit register. Last was the creation of the test bench. The test bench tested to see if the register lines were able to hold a value if ENA was high, to check if the registers stayed unchanged if ENA is low, and to test RST.

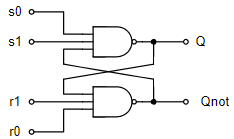


Figure 1. Schematic of SR Latch used for module, taken from ECE 526 Lab Manual, pg 31

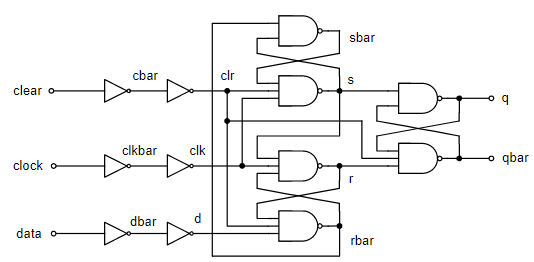


Figure 2. Schematic of D flipflop used for module, taken from ECE 526 Lab Manual, pg31

**Analysis:**

The circuit was calculated to have a 41.9 ns propagation delay in the critical path (calculation can be found below), which means it would have a maximum operating frequency of 23.87 MHz. During testing, the module was found to be stable with a period as low as 41.91ns, which gives a maximum operating frequency of 23.86MHz, and was metastable at a period of 41.9ns. At a period of 41.9ns or less, one period is not enough time between the ENA line being set high, and the register lines taking on the value of Data lines. The simulation was run again with a period of 38ns to demonstrate this. The simulation was also run with a period of 10 ns to show that none of the functions work with such a small period. Reset looks like it is working but this is due to a flaw in the testing which lets the final value stay for longer than 2 periods, which gives the reset enough time to propagate throughout the circuit (17.2 ns).

MUX propagation delay:

1.5ns + 2.8ns + 2.5ns = 6.8 ns

DFF propagation delay:

1.5ns + 1.5ns + 4 ns + 2.5 ns + 4ns + 3.8 ns +3.8 ns +7.5 ns +6.5 ns = 35.1 ns

Total propagation delay = 41.9 ns

Maximum Operating frequency = 1/ (total propagation delay) = 23.87 MHz

Below, the waveforms from running the simulation can be found, along with the log, module codes, and test bench code.

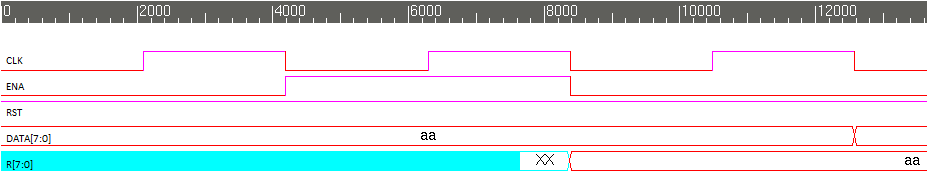


Figure 3. 41.91 ns period. Register takes on values of DATA line (1’hAA) after ENA goes high

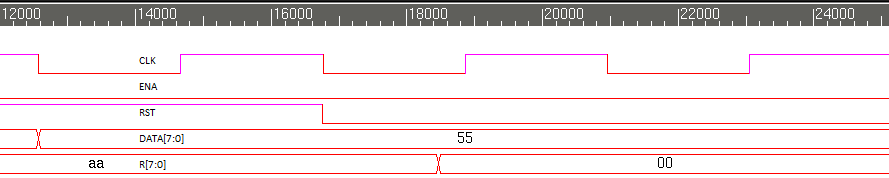


Figure 4. 41.91 ns period. ENA is deasserted and DATA lines changed, R lines does not change. RST line is set low, and R is reset to 00.

Below is the output wave forms when the period is 38ns. As can be seen here and in the Log, the register lines are set after enable is already set to low. There is one period of delay between Enable going high, and then back to low. This is problematic because the register lines should not be taking on values after the enable line is set to low. The propagation delay of the circuit causes this to happen, as the circuit is still trying to set the register lines from when enable was high, even after enable is set to low.

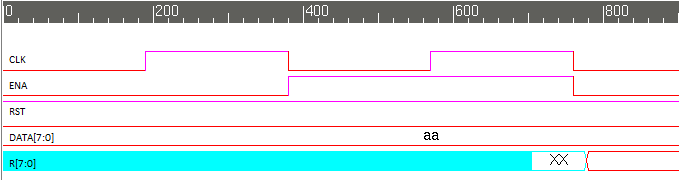


Figure 5. 38 ns period. Register takes on values of DATA line AFTER ENA goes low

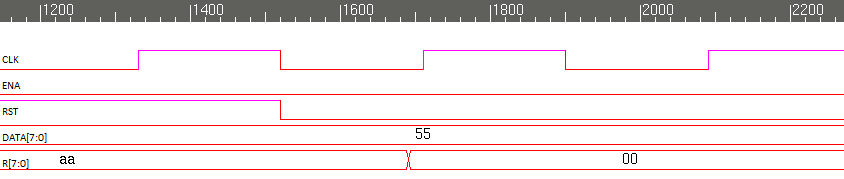
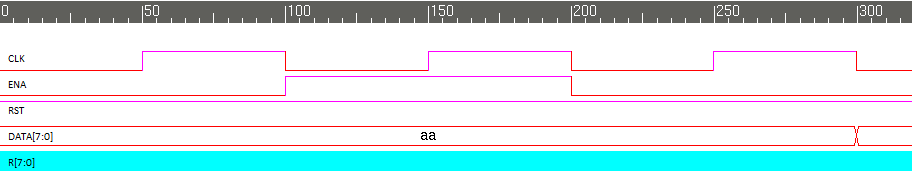


Figure 6. 38 ns period. Reset function works as expected because it has a much smaller propagation delay

Below are the results of the simulation when the period is 10 ns. This is nowhere near a large enough period for the propagation delay. As can be seen, the register lines never take on a value from the data line. The only reason reset looks to be working is because of the delay at the end of test bench, which adds another period of delay. As can be seen, reset takes only roughly 17 ns to propagate, compared to the 41.9 ns that it takes for the registers to change their values.



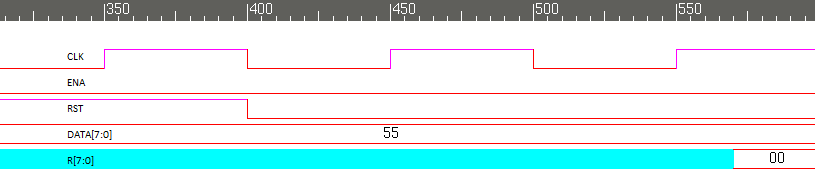


Figure 7. 10 ns period. This period is too short and none of the expected functionality is there

**Modules:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* ECE526L Experiment #3                Garen Nikoyan, Spring 2018 \*\*\*

\*\*\* Hierarchical Modeling                \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Filename: register.v     Created by: Garen Nikoyan, 2/15/2018 \*\*\*

\*\*\* -Revision History                           \*\*\*

\*\*\* 2/15/2018: First draft \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* This module models a register, using a multiplexer and D flip-flop\*\*\*

\*\*\*         \*\*\*

\*\*\* The gates of the flip flop have different input delays, \*\*\*

\*\*\* as well as fanout delays:          \*\*\*

\*\*\* FO1 = 0.5ns; FO2 = 0.8ns; FO3 = 1.0ns; Primary Output = 4ns \*\*\*

\*\*\* 1 input gates = IN1 = 1ns \*\*\*

\*\*\* 2 input gates = IN2 = 2ns \*\*\*

\*\*\* 3 input gates = IN3 = 3ns         \*\*\*

\*\*\* \*\*\*

\*\*\*                                     \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

`timescale 1 ns / 10 ps

`define PO 4 // time delay for primary output, out of dff module, is 4 ns

module register(R, CLK, DATA, ENA, RST);

    output [7:0] R;

    input [7:0] DATA;

    input CLK, ENA, RST;

    wire [7:0] MO;

    wire [7:0] RN;

MUX mux0(MO[0],R[0],DATA[0],ENA); // MO[ ] gets passed to dff as the data coming in

dff dff0(R[0],RN[0],CLK,MO[0],RST); // R[ ] is the primary output

MUX mux1(MO[1],R[1],DATA[1],ENA);

dff dff1(R[1],RN[1],CLK,MO[1],RST);

MUX mux2(MO[2],R[2],DATA[2],ENA);

dff dff2(R[2],RN[2],CLK,MO[2],RST);

MUX mux3(MO[3],R[3],DATA[3],ENA);

dff dff3(R[3],RN[3],CLK,MO[3],RST);

MUX mux4(MO[4],R[4],DATA[4],ENA);

dff dff4(R[4],RN[4],CLK,MO[4],RST);

MUX mux5(MO[5],R[5],DATA[5],ENA);

dff dff5(R[5],RN[5],CLK,MO[5],RST);

MUX mux6(MO[6],R[6],DATA[6],ENA);

dff dff6(R[6],RN[6],CLK,MO[6],RST);

MUX mux7(MO[7],R[7],DATA[7],ENA);

dff dff7(R[7],RN[7],CLK,MO[7],RST);

endmodule

module SR\_Latch2(Q, Qnot, s0, s1, r0, r1);

    // using parameter allows us to easily change these time delays when we call the module

    parameter   NAND1\_IN = 3,   // 3 ns delay for 3 input gate

            NAND1\_FO = 0.5, // 0.5 ns delay, for fanout 1 delay

            NAND2\_IN = 3,

            NAND2\_FO = 0.5;

    output Q, Qnot;

    input s0, s1, r0, r1;

    wire s0,s1,r0,r1;

    wire Q, Qnot;

    nand #(NAND1\_IN + NAND1\_FO) NAND1(Q,s0,s1,Qnot);

    nand #(NAND2\_IN + NAND2\_FO) NAND2(Qnot,r1,r0,Q);

endmodule

module dff(q, qbar, clock, data, clear);

    output q, qbar;

    input clock, data, clear;

    not #(`InverterIN+ `InverterOUT) CLR1(cbar,clear);

    not #(`InverterIN+ `InverterOUT) CLR2(clr,cbar);

    not #(`InverterIN+ `InverterOUT) CLK1(clkbar,clock);

    not #(`InverterIN+ `InverterOUT) CLK2(clk,clkbar);

    not #(`InverterIN+ `InverterOUT) D1(dbar,data);

    not #(`InverterIN+ `InverterOUT) D2(d,dbar);

    SR\_Latch2 #(2, 0.5, 3, 1.0) SR1(sbar,s,rbar,1'b1,clr,clk);

    SR\_Latch2 #(3, 0.8, 3, 0.8) SR2(r,rbar,s,clk,clr,d);

    SR\_Latch2 #(2, 4.5, 3, 4.5) SR3(q,qbar,s,1'b1,clr,r);

endmodule

module MUX(OUT, A, B, SEL);

    parameter NOT1in = 1,    // delays set according to # of inputs and fanout

         NOT1out = 0.5,

         AND1in = 2,

         AND1out = 0.8,

         AND2in = 2,

         AND2out = 0.8,

         OR1in = 2,

         OR1out = 0.5;

// Port Declarations

input A, B, SEL;

output OUT;

// Internal variable declarations

wire A1, B1, SEL\_N;

// The netlist

not #(NOT1in+NOT1out) NOT1(SEL\_N, SEL);

and #(AND1in+AND1out) AND1(A1, A, SEL\_N);

and #(AND2in+AND2out) AND2(B1, B, SEL);

or #(OR1in+OR1out) OR1(OUT, A1, B1);

endmodule

**Testbench:**

`timescale 1 ns / 10 ps

`define period 41.91

module reg\_test();

reg CLK, ENA, RST; // inputs

reg [7:0]DATA; // inputs

wire [7:0]R; // outputs

register UUT(R, CLK, DATA, ENA, RST); // UUT = unit under test

always #(`period\*0.5) CLK=~CLK; // sets CLK to 50% duty cycle

initial begin

$vcdpluson;

$monitor("%d ns DATA=%h ENA=%b RST=%b R=%h",$time,DATA,ENA,RST,R);

CLK=0; // setting the clock

end

initial begin

    ENA=0; RST=1; DATA=8'hAA; // starting vector, DATA Line is 10101010

    #`period ENA=1; // checking if register holds DATA values when ENA asserted

    #`period ENA=0; // deasserting ENA

    #`period DATA=8'h55; // changing DATA to 01010101 to see if R changes while ENA is off

    #`period RST=0; // checking if registers reset when RST is 0

#`period $finish;

end

endmodule

**Log:**

Log for 41.91ns period

Chronologic VCS simulator copyright 1991-2017

Contains Synopsys proprietary information.

Compiler version M-2017.03-SP1\_Full64; Runtime version M-2017.03-SP1\_Full64; Feb 22 12:35 2018

VCD+ Writer M-2017.03-SP1\_Full64 Copyright (c) 1991-2017 by Synopsys Inc.

0 ns DATA=aa ENA=0 RST=1 R=xx

42 ns DATA=aa ENA=1 RST=1 R=xx

76 ns DATA=aa ENA=1 RST=1 R=XX

84 ns DATA=aa ENA=1 RST=1 R=aa

84 ns DATA=aa ENA=0 RST=1 R=aa

126 ns DATA=55 ENA=0 RST=1 R=aa

168 ns DATA=55 ENA=0 RST=0 R=aa

185 ns DATA=55 ENA=0 RST=0 R=00

$finish called from file "reg\_test.v", line 42.

$finish at simulation time 25146

V C S S i m u l a t i o n R e p o r t

Time: 251460 ps

CPU Time: 0.220 seconds; Data structure size: 0.0Mb

Thu Feb 22 12:35:36 2018

Log for 38ns period

Chronologic VCS simulator copyright 1991-2017

Contains Synopsys proprietary information.

Compiler version M-2017.03-SP1\_Full64; Runtime version M-2017.03-SP1\_Full64; Feb 22 12:42 2018

VCD+ Writer M-2017.03-SP1\_Full64 Copyright (c) 1991-2017 by Synopsys Inc.

0 ns DATA=aa ENA=0 RST=1 R=xx

38 ns DATA=aa ENA=1 RST=1 R=xx

71 ns DATA=aa ENA=1 RST=1 R=XX

76 ns DATA=aa ENA=0 RST=1 R=XX

78 ns DATA=aa ENA=0 RST=1 R=aa

114 ns DATA=55 ENA=0 RST=1 R=aa

152 ns DATA=55 ENA=0 RST=0 R=aa

169 ns DATA=55 ENA=0 RST=0 R=00

$finish called from file "reg\_test.v", line 42.

$finish at simulation time 2280

V C S S i m u l a t i o n R e p o r t

Time: 228000 ps

CPU Time: 0.210 seconds; Data structure size: 0.0Mb

Thu Feb 22 12:42:51 2018

Log for 10ns period

Chronologic VCS simulator copyright 1991-2017

Contains Synopsys proprietary information.

Compiler version M-2017.03-SP1\_Full64; Runtime version M-2017.03-SP1\_Full64; Feb 22 12:44 2018

VCD+ Writer M-2017.03-SP1\_Full64 Copyright (c) 1991-2017 by Synopsys Inc.

0 ns DATA=aa ENA=0 RST=1 R=xx

10 ns DATA=aa ENA=1 RST=1 R=xx

20 ns DATA=aa ENA=0 RST=1 R=xx

30 ns DATA=55 ENA=0 RST=1 R=xx

40 ns DATA=55 ENA=0 RST=0 R=xx

57 ns DATA=55 ENA=0 RST=0 R=00

$finish called from file "reg\_test.v", line 42.

$finish at simulation time 600

V C S S i m u l a t i o n R e p o r t

Time: 60000 ps

CPU Time: 0.240 seconds; Data structure size: 0.0Mb

Thu Feb 22 12:44:46 2018

**Conclusion:**

One of the main lessons that can be taken away from this lab is the importance of the use of parameters and how it can not only make the use of modules rather flexible, but also reduce the number of modules one would have to make depending on the current situation. The force file is also a useful tool which can be utilized to shorten time between compilations, but to more importantly help reduce typographical errors.

It also important to note that different modes of a circuit have different propagation delays, as was seen in this lab. The critical path for reset was less than half of the critical path for the register lines. This needs to be kept in mind later on as this could impact test results, and make a circuit that doesn’t work, look like it is working.