LAB 2:

Structural Modeling of a JK Flip-Flop

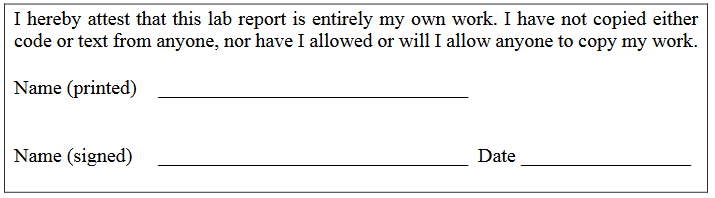
Professor Ronald Mehler

ECE 526L

Spring 2018

Garen Nikoyan

2/8/2018

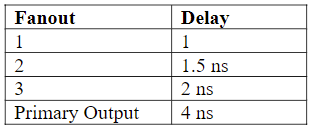
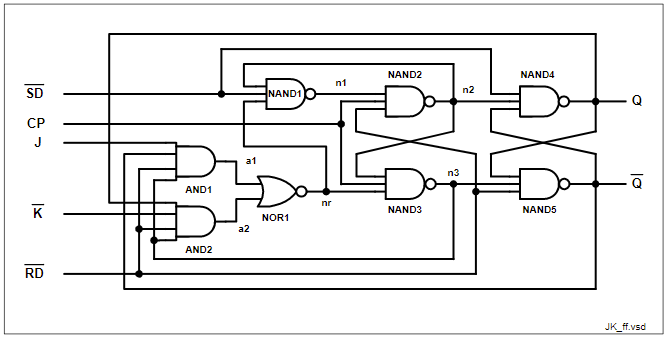


**Objective:**

The purpose of this lab is to become familiar with utilizing time delays of primitives and designing a JK flip-flop. The four Verilog system tasks that allow for outputting signal behavior to a log were also studied in this lab.

**Methodology:**

For this lab, the first step was to create the module for the JK flip-flop following the diagram found below. Gate time delays, as well as fanout delays, were incorporated into the design, and the tables for these can be found below. The testbench was then created, and would allow for testing of all possible modes of the JK flip-flop. The only state that was not tested was the Indeterminate mode, which is caused by having Set and Reset applied at the same time. Testing all modes required a clock, since some modes are synchronous. The module was tested with two different clocks to see the effect it would have on the circuit. One clock had a 50% duty cycle, with a period of 60 ns, and the second had a 10% duty cycle, with a period of 180 ns.



|  |  |
| --- | --- |
| **Gate** | **Delay (ns)** |
| Two input gates | 2 |
| Three input gates | 3 |
| Four input gates | 4 |

Figure 0. Schematic and Tables taken from ECE 526 Lab Manual, pg 27

**Analysis:**

The circuit was calculated to have a 39.5 ns propagation delay in the critical path, which means it would have a maximum operating frequency of 25.64 MHz. To be on the safe side, a much larger period was used when testing this module. The gate and fanout delays from above were used, and the critical path used to calculate the propagation is as follows: AND1-> NOR1-> NAND1-> NAND2-> NAND3-> NAND4-> NAND5.

Below, the waveforms from running the simulation can be found, along with the log, module code, and testbench code. With the clock running at a 50% duty cycle and 60 ns period, the flip-flop worked as intended. There was one issue found later in the testbench, which had set the flip-flop back into Load 1 mode after going through the other modes.

Using the same module and testbench vectors, the simulation was ran again using a 10% duty cycle clock, initially with a period of 100 ns. This gave similar results to the 50% duty cycle test. Since this used the same testbench as the first test, it has the same issue of being set back into Load 1 mode after through the other modes. The waveforms and logs can be found below.

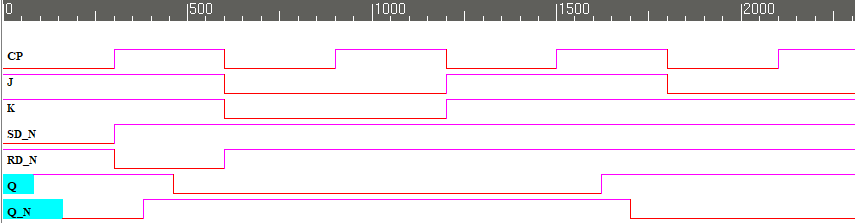


Figure 1. 50% Duty cycle, 60 ns period. Going through Async Set, Async Reset, Load 0, and Load 1 modes

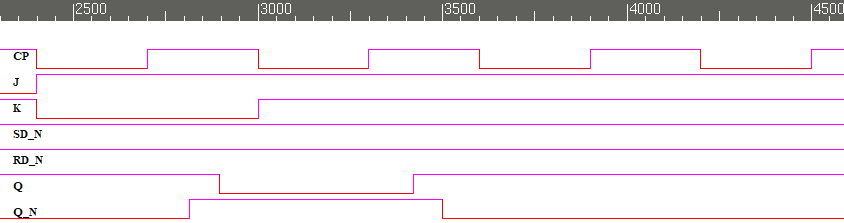


Figure 2. 50% Duty cycle, 60 ns period. Load 1 going into Hold, Toggle, Hold, and back to Load 1 modes

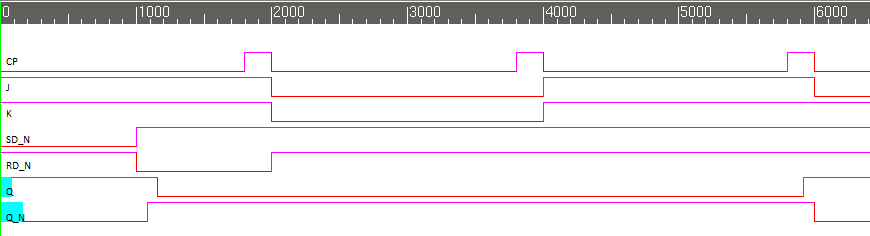


Figure 3. 10% Duty cycle, 100 ns period. Going through Async Set, Async Reset, Load 0, and Load 1 modes

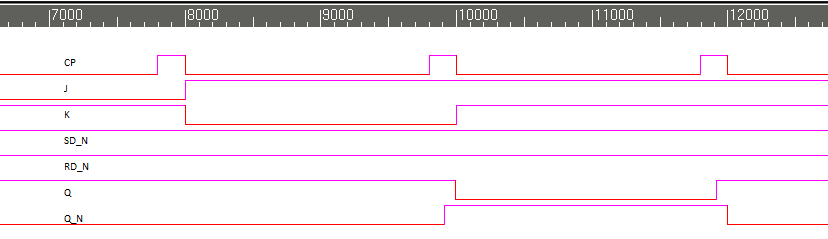


Figure 4. 10% Duty cycle, 100 ns period. Load 1 going into Hold, Toggle, Hold, and back to Load 1 modes

**Module:**

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

\*\*\* ECE526L Experiment #2 Garen Nikoyan, Spring 2018 \*\*\*

\*\*\* Structural Modeling of a JK Flip-Flop \*\*\*

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

\*\*\* Filename: JKFF.v Created by: Garen Nikoyan, 2/7/2018 \*\*\*

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

\*\*\* 2/7/2018: First draft \*\*\*

\*\*\* 2/8/2018: Corrected time delays

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

\*\*\* This module models a JK Flip-Flop \*\*\*

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

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

\*\*\* FO1 = 1ns; FO2 = 1.5ns; FO3 = 2ns; Primary Output = 4ns \*\*\*

\*\*\* 2 input gates = time\_delay\_1 = 2ns \*\*\*

\*\*\* 3 input gates = time\_delay\_2 = 3ns \*\*\*

\*\*\* 4 input gates = time\_delay\_3 = 4ns \*\*\*

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

\*\*\* The following results are expected from the JKFF \*\*\*

\*\*\* J K SD CP RD Q QN MODE

\*\*\* x x 0 x 1 1 0 Async Set \*\*\*

\*\*\* x x 1 x 0 0 1 Async Reset \*\*\*

\*\*\* x x 0 x 0 1-? 1-? Indeterminate \*\*\*

\*\*\* 0 0 1 p 1 0 1 Load 0 SYNC (reset) \*\*\*

\*\*\* 1 1 1 p 1 1 0 Load 1 SYNC (SET) \*\*\*

\*\*\* 0 1 1 p 1 q qb Hold SYNC \*\*\*

\*\*\* 1 0 1 p 1 qb q Toggle SYNC \*\*\*

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

`timescale 1 ns / 100 ps //100ps because precision of less than 1ns is needed

`define FO1 0.5 //0.5ns

`define FO2 1.5 //1.5ns

`define FO3 2 //2ns

`define PO 4 //4ns

`define time\_delay\_1 1 //1ns, 2 input gate delay

`define time\_delay\_2 2 //2ns, 3 input gate delay

`define time\_delay\_3 3 //3ns, 4 input gat delay

module JKFF(Q, Q\_N, SD\_N, CP, J, K\_N, RD\_N);

// Port Declarations

input SD\_N, CP, J, K\_N, RD\_N;

output Q, Q\_N;

// Internal variable declarations

wire a1, a2, nr, n1, n2, n3, Q, Q\_N;

// The netlist

and #(`time\_delay\_3+ `FO1) AND1(a1, J, Q\_N, RD\_N, n3);

and #(`time\_delay\_3+ `FO1) AND2(a2, Q, K\_N, RD\_N, n3);

nand #(`time\_delay\_2+ `FO1) NAND1(n1, nr, SD\_N, n2);

nor #(`time\_delay\_1+ `FO2) NOR2(nr, a1, a2);

nand #(`time\_delay\_2+ `FO3) NAND2(n2, n1, CP, RD\_N);

nand #(`time\_delay\_2+ `FO2) NAND3(n3, n2, CP, nr);

nand #(`time\_delay\_2+ `PO+ `FO3) NAND4(Q, SD\_N, n2, Q\_N);

nand #(`time\_delay\_2+ `PO+ `FO3) NAND5(Q\_N, Q, n3, RD\_N);

endmodule

**Testbench:**

`timescale 1 ns / 100 ps

`define period 1000 // 60ns for 50% duty cycle, 100ns for 10% duty cycle

module TB\_JKFF();

reg SD\_N, CP, J, K\_N, RD\_N; // inputs

wire Q, Q\_N; // outputs

JKFF UUT(Q, Q\_N, SD\_N, CP, J, K\_N, RD\_N); // UUT = unit under test

//always #(`period\*0.5) CP=~CP; // sets CP to 50% duty cycle, toggles CP every 30ns

always begin // used to set the duty cycle to 10%

CP=0; //CP set to 0 for 90% of the period

# ( `period\*0.9 );

CP=1; //CP set to 1 for 10% of the period

# (`period\*0.1);

end

initial begin // used initial begin because multiple $ statements are used

$write ("This shows the given inputs"); // good for writing text, but no built-in new line

$display (" that will be a part of all tests in this lab."); // good for writing text, built-in new line

$monitor ("%d ns J = %b, K\_N= %b, SD\_N = %b, RD\_N = %b",$time,J,K\_N,SD\_N,RD\_N);

end

initial begin

$vcdpluson;

// Asynchronous Tests

$display("Async Set Test");

J=1 ; K\_N=1 ; SD\_N=0 ; RD\_N=1;

#(`period/2)

$display("Q = %b Q\_N = %b", Q, Q\_N);

$display("Async Reset Test");

J=1 ; K\_N=1 ; SD\_N=1 ; RD\_N=0;

#(`period/2)

$display(" Q = %b Q\_N = %b", Q, Q\_N);

// Synchronous Tests

$display("Load 0 Test");

J=0 ; K\_N=0 ; SD\_N=1 ; RD\_N=1;

#`period

$display("Q = %b Q\_N = %b", Q, Q\_N);

$display("Load 1 Test");

J=1 ; K\_N=1 ; SD\_N=1 ; RD\_N=1;

#`period

$display("Q = %b Q\_N = %b", Q, Q\_N);

$display("Hold");

J=0; K\_N=1; SD\_N=1; RD\_N=1;

#`period

$display("Q = %b Q\_N = %b", Q, Q\_N);

$display("Toggle");

J=1; K\_N=0; SD\_N=1; RD\_N=1;

#`period

$display("Q = %b Q\_N = %b", Q, Q\_N);

$display("Hold");

J=1; K\_N=1; SD\_N=1; RD\_N=1;

#`period

$strobe("Q = %b Q\_N = %b", Q, Q\_N);

#100 $finish;

end

endmodule

**Log:**

Log for 50% duty cycle, 60ns 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 8 20:37 2018

This shows the given inputs that will be a part of all tests in this lab.

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

Async Set Test

0 ns J = 1, K\_N= 1, SD\_N = 0, RD\_N = 1

Q = 1 Q\_N = 0

Async Reset Test

30 ns J = 1, K\_N= 1, SD\_N = 1, RD\_N = 0

Q = 0 Q\_N = 1

Load 0 Test

60 ns J = 0, K\_N= 0, SD\_N = 1, RD\_N = 1

Q = 0 Q\_N = 1

Load 1 Test

120 ns J = 1, K\_N= 1, SD\_N = 1, RD\_N = 1

Q = 1 Q\_N = 0

Hold

180 ns J = 0, K\_N= 1, SD\_N = 1, RD\_N = 1

Q = 1 Q\_N = 0

Toggle

240 ns J = 1, K\_N= 0, SD\_N = 1, RD\_N = 1

Q = 0 Q\_N = 1

Hold

300 ns J = 1, K\_N= 1, SD\_N = 1, RD\_N = 1

Q = 0 Q\_N = 1

$finish called from file "TB\_JKFFV1.v", line 87.

$finish at simulation time 4600

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

Time: 460000 ps

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

Thu Feb 8 20:37:28 2018

Log for 10% duty cycle, 100ns 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 15 12:22 2018

This shows the given inputs that will be a part of all tests in this lab.

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

Async Set Test

0 ns J = 1, K\_N= 1, SD\_N = 0, RD\_N = 1

Q = 1 Q\_N = 0

Async Reset Test

100 ns J = 1, K\_N= 1, SD\_N = 1, RD\_N = 0

Q = 0 Q\_N = 1

Load 0 Test

200 ns J = 0, K\_N= 0, SD\_N = 1, RD\_N = 1

Q = 0 Q\_N = 1

Load 1 Test

400 ns J = 1, K\_N= 1, SD\_N = 1, RD\_N = 1

Q = 1 Q\_N = 1

Hold

600 ns J = 0, K\_N= 1, SD\_N = 1, RD\_N = 1

Q = 1 Q\_N = 0

Toggle

800 ns J = 1, K\_N= 0, SD\_N = 1, RD\_N = 1

Q = 0 Q\_N = 1

Hold

1000 ns J = 1, K\_N= 1, SD\_N = 1, RD\_N = 1

Q = 0 Q\_N = 1

$finish called from file "TB\_JKFFV1.v", line 87.

$finish at simulation time 13000

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

Time: 1300000 ps

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

Thu Feb 15 12:22:31 2018

**System Tasks:**

Four different system tasks were used in the testbench, $monitor, $display, $write, and $strobe. Each one has its use case, which was learned in this lab. $monitor is best used for watching all outputs, every time they change, so this was used to monitor all the inputs, as well as the time they changed at. $display and $write are very similar, with one big difference, which is that $display has a built in new line function and $write does not. $display was used to display some of the outputs the log, as $strobe was not always the best option. $strobe is special in that it is always the last thing executed in a single time step. Because of this, it was used to display the outputs for the last vector.

**Conclusion:**

The main lesson in this lab is how important it is to have proper time delays in sequential circuits, and how to work the testbench around it to properly test the module. Failure to do so will lead to incorrect test results, because the inputs would not have propagated throughout the circuit yet to give the correct output.