LAB 4:

Behavioral Modeling of a Counter

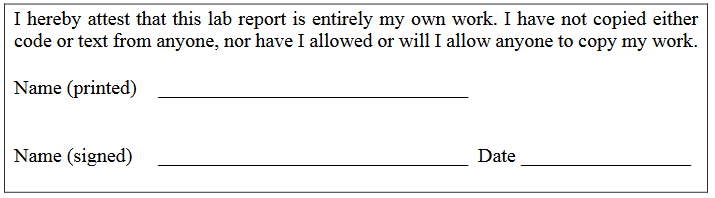
Professor Ronald Mehler

ECE 526L

Spring 2018

Garen Nikoyan

2/22/2018



**Objective:**

The purpose of this lab is to become familiar with utilizing behavioral code, as well an asynchronous assert, synchronous de-assert Reset. This is the first lab to not use primitive gates. The use of behavioral code is extremely beneficial to larger designs, and can greatly simplify smaller designs.

**Methodology:**

For this lab, the first step was to create the module for a six-bit counter. It is important to note that this, and all other modules in this lab, use System Verilog, and therefore end with a .sv extension. Also, when compiling, “-sverilog” had to be added in the instruction. Next, the AASD was modeled, following the function described in lecture. A new, top-level module was then created. The sole purpose of this module was to call the counter and AASD modules. Last was the creation of the test bench. Using a 10 ns period, 50% duty cycle clock, the test bench tested to see if the AASD worked properly. On top of this, it tested the enable and load features, and if the counter rolled over properly from 63d.

**Analysis:**

Looking at waveforms and logs, it can be seen that the AASD works properly, and overrides all other functions. It is tested once initially, and takes the Count line from xx to 00h. It is tested again later on, to ensure it overrides the load, and count functions. It takes 1 clock cycle for the input of reset to reach the output. One thing that is noticed is that when reset is de-asserted and set back to a value/input of 1, it takes about two clock cycles for this input to be registered and pushed to the output of the waveform which allows the counter to start up again. This is because the AASD file contains 2 FF’s. When a value of 0 is given to the RESET input it immediately takes that and pushed it to the output waveform, but when a value of 1 is given to the RESET input this needs to go through both flip flops in the AASD file which is the cause of the two clock cycle delay in order for the input to reach the output waveform.

In Figure 1, Data is loaded to have a value of 15h to make sure Count does not get loaded with a value, which it does not. An asynchronous reset is applied, which takes the Count from xx, to 00h. Reset is then asserted, and Enable is set high, which causes the Count to begin incrementing.

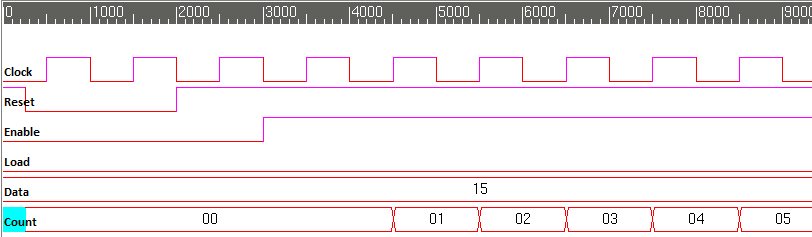


Figure 1. Testing Asynchronous reset, and checking if Enable high causes count to begin

In Figure 2, after Count reaches 07h, Load is set to high, and Data is set to 60d. The Count properly takes on this value, and begins incrementing when Load is set to low. Once it reaches 63d, the Count rolls over to 0, and begins incrementing again.

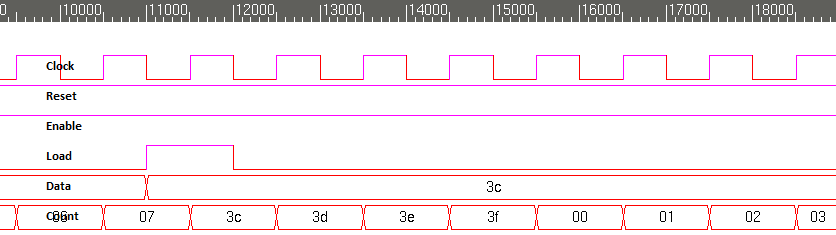


Figure 2. Letting count reach 7, then setting Load high and loading 60d, and checking if it rolls over properly

In Figure 3, Load is set to high, and it overrides the incrementing. Reset is then set to low, and it properly overrides the Load. Incrementing continues once Reset is set back to high, and Load to low.

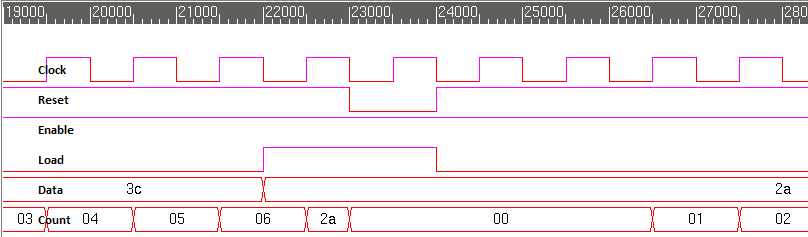


Figure 3. Testing if Load overrides Enable, and then if Reset overrides Load

In Figure 4, Reset is set to low while Enable is high, and Count is incrementing, and can be seen that Reset overrides it properly. Reset is set back to high. Lastly, Enable is set to low, and Load to high, to test if Load works with Enable low, which it does not, confirming correcting functioning of Enable.

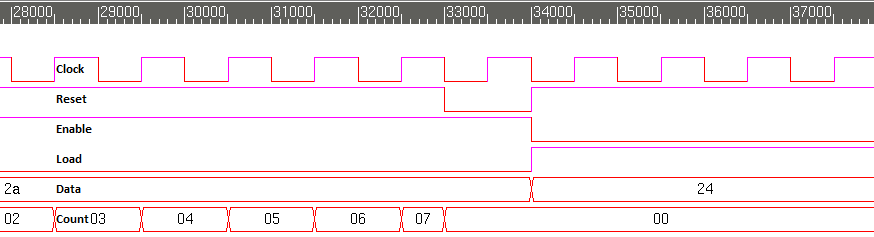


Figure 4. Testing count, and if Reset overrides Enable, and if Load works with Enable low

**Modules:**

Counter:

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

\*\*\* ECE526L Experiment #4 Garen Nikoyan, Spring 2018 \*\*\*

\*\*\* Behavioral Modeling of a Counter \*\*\*

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

\*\*\* Filename: counter.sv Created by: Garen Nikoyan, 2/22/2018 \*\*\*

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

\*\*\* 2/22/2018: First draft \*\*\*

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

\*\*\* This module models an counter \*\*\*

\*\*\* Asynchronous, active low Reset: sets Count to 0

\*\*\* Synchronous active high Enable: Count is incremented,

\*\*\* or loaded with new data if Load is high

\*\*\* Synchronous, active high Load: value on data pins is loaded

\*\*\* into counter after posedge of clock if Enable is high

\*\*\* If Load is low, Enable is high, and Reset is high, counter

\*\*\* advances on posedge of clock \*\*\*

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

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

`timescale 1 ns / 10 ps

module counter(Count, Clock, Reset, Enable, Load, Data);

output reg [5:0] Count;

input [5:0] Data;

input Clock, Enable, Reset, Load;

always @(posedge Clock or negedge Reset)

if(!Reset)

Count <= 6'b000000;

else if(Enable)

begin

if(Load)

Count <= Data;

else

Count <= Count+ 1;

end

endmodule

AASD:

`timescale 1ns/10ps

module AASD(AASD\_RST,CLOCK,RESET);

output reg AASD\_RST;

input CLOCK,RESET;

reg Out1; //this is the wire for the output of the first FF

always @(posedge CLOCK or negedge RESET) //allows 4 synchronous/asynchronous

if(!RESET)begin//asynchronous reset

Out1 <= 1'b0;

AASD\_RST <= 1'b0;

end

else begin

Out1 <= 1'b1;

AASD\_RST <= Out1;

end

endmodule

TLcount (top level module):

`timescale 1 ns / 10 ps

module TLcount(Count,Data,Clock,Reset,Enable,Load);

output [5:0] Count;

input [5:0] Data;

input Clock,Reset,Enable,Load;

AASD aasd1(RST, Clock, Reset);

counter counter1(Count, Clock, RST, Enable, Load, Data);

endmodule

**Testbench:**

`timescale 1 ns / 10 ps

`define period 10

module TBcount();

reg Clock, Enable, Reset,Load; // inputs

reg [5:0]Data; // inputs

wire [5:0]Count; // outputs

TLcount UUT(Count,Data,Clock,Reset,Enable,Load); // UUT = unit under test

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

initial begin

$vcdpluson;

$monitor("%d ns Data=%d Enable=%b Load =%b Reset=%b Count=%d",$time,Data,Enable,Load,Reset,Count);

Clock=0; // setting the clock

end

initial begin

Enable=0; Reset=1; Load=0; Data=6'b010101; // starting vector

#(`period\* 0.25) Reset=0; // checking if AASD works properly, demonstrating asynchronous reset

#(`period\* 1.75) Reset=1; // Asserting Reset asynchronously

#`period Enable=1; //

#(`period\* 8) Load=1; Data=6'b111100; // loading 60d after count reaches 7d

#`period Load=0; // Setting Load to 0 so counter will begin to increment

#(`period\* 10) Load=1; Enable=1; Data=6'b101010;

#`period Reset=0; // testing if reset overrides load

#`period Reset=1; Load=0; // incrementing continues, showing Enable works properly

#(`period\* 9) Reset=0; // testing if reset overrides increment

#(`period) Reset=1; Enable=0; Load=1; Data=6'd36; // checking if Load works while Enable is low

#(`period\* 4) $finish;

end

endmodule

**Log:**

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 20:03 2018

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

0 ns Data=21 Enable=0 Load =0 Reset=1 Count= x

3 ns Data=21 Enable=0 Load =0 Reset=0 Count= 0

20 ns Data=21 Enable=0 Load =0 Reset=1 Count= 0

30 ns Data=21 Enable=1 Load =0 Reset=1 Count= 0

45 ns Data=21 Enable=1 Load =0 Reset=1 Count= 1

55 ns Data=21 Enable=1 Load =0 Reset=1 Count= 2

65 ns Data=21 Enable=1 Load =0 Reset=1 Count= 3

75 ns Data=21 Enable=1 Load =0 Reset=1 Count= 4

85 ns Data=21 Enable=1 Load =0 Reset=1 Count= 5

95 ns Data=21 Enable=1 Load =0 Reset=1 Count= 6

105 ns Data=21 Enable=1 Load =0 Reset=1 Count= 7

110 ns Data=60 Enable=1 Load =1 Reset=1 Count= 7

115 ns Data=60 Enable=1 Load =1 Reset=1 Count=60

120 ns Data=60 Enable=1 Load =0 Reset=1 Count=60

125 ns Data=60 Enable=1 Load =0 Reset=1 Count=61

135 ns Data=60 Enable=1 Load =0 Reset=1 Count=62

145 ns Data=60 Enable=1 Load =0 Reset=1 Count=63

155 ns Data=60 Enable=1 Load =0 Reset=1 Count= 0

165 ns Data=60 Enable=1 Load =0 Reset=1 Count= 1

175 ns Data=60 Enable=1 Load =0 Reset=1 Count= 2

185 ns Data=60 Enable=1 Load =0 Reset=1 Count= 3

195 ns Data=60 Enable=1 Load =0 Reset=1 Count= 4

205 ns Data=60 Enable=1 Load =0 Reset=1 Count= 5

215 ns Data=60 Enable=1 Load =0 Reset=1 Count= 6

220 ns Data=42 Enable=1 Load =1 Reset=1 Count= 6

225 ns Data=42 Enable=1 Load =1 Reset=1 Count=42

230 ns Data=42 Enable=1 Load =1 Reset=0 Count= 0

240 ns Data=42 Enable=1 Load =0 Reset=1 Count= 0

265 ns Data=42 Enable=1 Load =0 Reset=1 Count= 1

275 ns Data=42 Enable=1 Load =0 Reset=1 Count= 2

285 ns Data=42 Enable=1 Load =0 Reset=1 Count= 3

295 ns Data=42 Enable=1 Load =0 Reset=1 Count= 4

305 ns Data=42 Enable=1 Load =0 Reset=1 Count= 5

315 ns Data=42 Enable=1 Load =0 Reset=1 Count= 6

325 ns Data=42 Enable=1 Load =0 Reset=1 Count= 7

330 ns Data=42 Enable=1 Load =0 Reset=0 Count= 0

340 ns Data=36 Enable=0 Load =1 Reset=1 Count= 0

$finish called from file "TBcount.sv", line 34.

$finish at simulation time 38000

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

Time: 380000 ps

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

Thu Feb 22 20:03:53 2018

**Conclusion:**

Everything in this lab worked as expected without running into any issues. One thing that was noted and as it should be in practically every lab is that one needs to give enough delay in order for the setup time and hold time to be enough to be able to register every input and be pushed to the output waveform. Another that was noted and which was new is that when it comes to the delay for the AASD file, when setting reset to 0, it takes one clock cycle for it to fully show and take action in the waveform, but two clock cycles after it is set to 1. A problem noticed too late, is that the testbench designed for this lab was repetitive for a few cases, and could have been completed with less test vectors. It is important to always have efficient designs, so greater care must be taken in later labs to avoid this