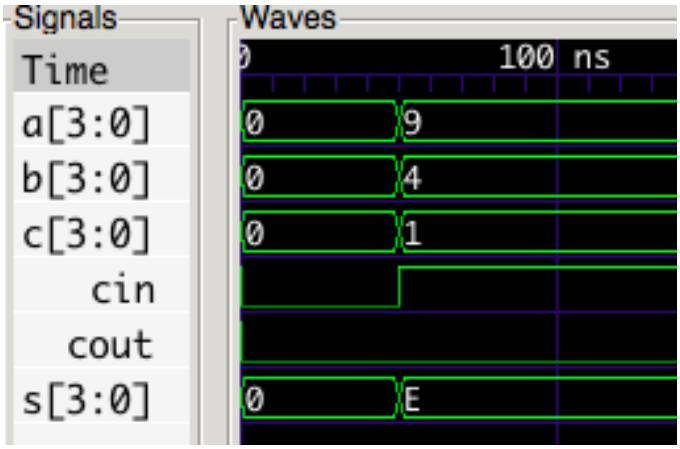
Hardware Synthesis Laboratory

2nd semester, Academic Year 2019

```
module fulladder (cout,s,a,b,cin);
    output cout;
    output s;
    input a,b,cin;

//assign {cout,s}=a+b+cin;
reg cout, s;
always @(a or b or cin)
begin
    {cout,s}=a+b+cin;
end

endmodule
```



Krerk Piromsopa, Ph. D.

This document is a part of the 2110363 Hardware Synthesis Lab I,

Department of Computer Engineering, Chulalongkorn University.

Preface

VerilogHDL is a hardware description language (HDL) for simulating and modeling electronic systems. It is one of standard languages used in the design and verification of digital circuits at the register-transfer level. The digital design and verification class and Hardware Synthesis Laboratory offered by the Department of Computer Engineering, Chulalongkorn University has been using VerilogHDL as a part of our laboratories for almost 20 years. We strongly believe this is a fundamental skill for Computer Engineering.

This document is created to ease the mastering of VerilogHDL as a vital tool. The goal is not only for the class, but also to familiar the students with the standard and tools in the digital design.

Hopefully, materials provided in this class will be useful for helping the students to jump start with digital design and verification and Hardware Synthesis.

Krerk Piromsopa, Ph. D.

Assistant Professor

Department of Computer Engineering

Chulalongkorn University

December 2018.

Laboratory 1: Introduction to VerilogHDL and Digital Simulation

Objectives

1. Get students to familiar with the simulation tool (Vivado)
2. Demonstrate the basic of Verilog simulation and waveform output
3. Able to explain structural model and behavioral model
4. Able to explain the differences between blocking and non-blocking assignments

Background

In this lab, you will learn to the fundamental of VerilogHDL and Digital Simulation using Vivado Design suite. Firstly, please download the free (webpack) version of Vivado Design Suite from Xilinx Web site[1]. The whole download is about 20GB. Alternatively, you may download it from the department server[2] . Should you have trouble finding a machine for installing the software, please contact the instructors. A (virtual) machine can be provided for you to remotely work with the tool. However, you may still have to install the Lab Edition for download the design to the FPGA board. For more information about the installation and Vivado IDE, please what the Xilinx tutorials' videos[3].

Please watch the demonstration video on how to use the simulation.

[1] https://www.xilinx.com/products/design-tools/vivado.html

[2] https://mis.cp.eng.chula.ac.th/krerk/teaching/2018s2-HWSynLab/ Username: student, password: HWSynLab

[3] https://www.xilinx.com/products/design-tools/vivado.html#video

Exercises

1. Complete the following 1-bit full adder and a test bench to validate such design by simulating all possible inputs. Use the Vivado tool to simulate and validate the design.

```
module fullAdder(cout, s, a, b, cin);

output cout;
output s;
input a;
input b;
input cin;

reg cout, s;

always @(       )
begin

end

endmodule
```

```
`timescale 1ns/1ns

module tester;

      reg a,b,cin;
      wire cout,s;

      fullAdder a1(cout,s,a,b,cin);

initial
begin
      //$dumpfile("time.dump");
      //$dumpvars(2,a1);
      $monitor("time %t: {%b %b} <-
{%d %d %d}", $time,cout,s,a,b,cin);
      #0;
      a=0;
      b=0;
      cin=0;

      //…...
      $finish;
end
endmodule
```

2. What would happen if we replace the always block of the full adder in question 1 with the following module? Would it give the same result? Please run the test bench and provide your analysis.

```
module fullAdder(cout, s, a, b, cin);

output cout;
output s;
input a;
input b;
input cin;

assign {cout,s} = a + b + cin;

endmodule
```

3. Please modify the following latch to be a (positive edge triggering) flipflop with asynchronous reset. Please also modify the test bench to validate your design.

```
`timescale 1ns/1ns

module DFlipFlop(q,clock,nreset,d);

output q;
input clock,nreset,d;

reg q;

always @(clock)
begin
      if (nreset==1)
            q=d;
      else
            q=0;
end
endmodule
```

```
module testDFlipFlop();
reg clock, nreset, d;
DFlipFlop D1(q,clock,nreset,d);
always
      #10   clock=~clock;

initial
begin
      //$dumpfile("testDFlipFlop.dump
");
      //$dumpvars(1,D1);
      #0 d=0;
      clock=0;
      nreset=0;
      #50 nreset=1;
      #1000 $finish;
end
always
      #8 d=~d;
endmodule
```

4. What is the differences between the 2 provided designs? Please write a test bench to show your analysis.

```
module shiftA(q,clock,d);
output [1:0] q;
input clock,d;

reg [1:0] q;

always @(posedge clock)
begin
      q[0]=d;
      q[1]=q[0];
end
endmodule

module shiftB(q,clock,d);
output [1:0] q;
input clock,d;

reg [1:0] q;

always @(posedge clock)
begin
      q[0]<=d;
      q[1]<=q[0];
end
endmodule
```

5. Please answer the following questions and submit (in PDF format) to CourseVille on Friday before 23:59 (midnight).

1. Please draw a schematic representing the logical blocks of both shiftA and shiftB in exercise 4.
2. What is the different between blocking and non-blocking assignment?
3. Is it possible to apply parameters to the design in exercise 4 to create shiftRegister with any number of bits? If Yes, please explain how.