0512.4490 Advanced Computer Architecture Lab 2022

Lab #3: Verilog introduction

In this lab we'll learn to use Verilog simulation tools using simple examples.

Technical notes:

A. In order to compile Verilog sources, use the following command within the directory that contains all the *.v files:

```
ncverilog +access+rw *.v
```

B. In order to see the wave-diagram of the previous Verilog simulation use:

```
simvision waves.vcd
```

Assignments:

For each exercise, submit the verilog source files, a log of the testbench run, and a wave screenshot if requested.

1. Mux

The file mux.v contains two implementations of a 2:1 mux: the first using continuous assignment syntax, and the second using an always block. Both have the following inputs/output:

```
module mux1(a,b,select,result);
  input a;
  input b;
  input select;
  output result;
```

If select is '1', b is selected, otherwise a is selected.

Complete the code, and test it using the provided mux_tb.v testbench.

2. Half adder

Design a 1-bit half adder using structural gate level Verilog. The adder accepts two binary inputs: a and b, and computes their output sum and carry:

```
module halfadder(a,b,sum,carry);
  input a,b;
  output sum, carry;
```

The files halfadder.v and halfadder_tb.v contain a partial implementation for the half adder and its testbench. Complete the code, verify that it's working well using the testbench. In addition to the log, submit a waveform screenshot showing correct operation.

3. Full adder

Design a 1-bit full adder taking three 1-bit inputs: a, b, and ci, and giving sum and co.

```
module fulladder( sum, co, a, b, ci);
 input   a, b, ci;
 output  sum, co;
```

Write a testbench and use it to simulate the adder, checking all input combinations.

4. Four bit unsigned adder

Design a 4-bit unsigned adder using instantiation of four 1-bit adders.

```
module  add4( sum, co, a, b, ci);
 input   [3:0] a, b;
 input   ci;
 output  [3:0] sum;
 output  co;
```

No need to write a testbench, since it will be tested as part of the next exercise.

5. Four bit add/sub unit

Use the 4-bit unsigned adder from the previous exercise to design a 4-bit signed add/sub unit, which can either add or sub based on a mode input (0 add, 1 sub).

```
module  addsub( result, operand_a, operand_b, mode );
  input   [3:0]  operand_a, operand_b;
  input   mode;
  output  [3:0]  result;
```

Test it with the provided testbench, and submit a wave screenshot of the simulation.

6. Parity

Design a state machine which will accept a stream of bits (one every clock), and will output '1' if the number of ones accepted so far is even, or '0' otherwise.

```
module iseven(clk, in, reset, out);
  input clk, in, reset;
  output out;
```

Fill in the provided parity.v and parity_tb.v and submit them along with the Verilog log file.

7. Bi-Modal N-bit Saturating Counter

Design an N-bit saturating counter circuit, which is widely used for branch prediction. The circuit maintains an N-bit counter, initialized to 0 at reset=1. The counter value is preserved when reset=0 and branch=0, whereas it is updated when reset=0 and branch=1 according to the following rule:

counter(t+1) = min{ counter(t) + 1, 2^N – 1} *if taken == 1*
counter(t+1) = max{ counter(t) – 1, 0 } *if taken == 0*

The output *prediction* equals 1 if and only if counter(t) >= 2^(N-1)

```
module sat_count(clk, reset, branch, taken, prediction);
  parameter N=2;
  input clk, reset, branch, taken;
  output prediction;
```

Submit a state diagram of sat_count(N=2). Fill the sat_count.v code, and write a testbench sat_count_tb.v. Your test-bench must perform a test of sat_count(N=2) and print a final answer such as "PASSED ALL TESTS".

8. fifo(N,W)

Design a first-in-first-out queue synchronous module which satisfies the specifications below:

```
module fifo(clk, reset, in, push, pop, out, full);
  parameter N=4; // determines the maximum number of words in queue.
  parameter M=2; // determines the bit-width of each word stored in the queue.

  input clk, reset, push, pop;
  input [M-1:0] in;
  output [M-1:0] out;
  output full;
```

The functionality:

Let $W(t)=\langle w_1,w_2,\ldots,w_n\rangle$ denote the words stored at the module at time t, where $1,\ldots,n$ corresponds to the chronological order of the words registered at the queue (n is the oldest). The outputs must respect the following behavior:

full(t) = 1 iff n=N

out(t) = w_n iff n>0 (otherwise out(t)=0^M)

W(t+1) = < > (empty queue) iff reset(t)=1

If reset(t)=0 then the functionality depends on n:

n=0:

- W(t+1) = <in> iff push(t)=1 and pop(t)=*
- W(t+1) = W(t) iff push(t)=0 and pop(t)=*

0<n<N:

- W(t+1) = $\langle in, w_1,w_2,\ldots,w_{n-1}\rangle$ iff push(t)=1 and pop(t)=1
- W(t+1) = $\langle in, w_1,w_2,\ldots,w_n\rangle$ iff push(t)=1 and pop(t)=0
- W(t+1) = $\langle w_1,w_2,\ldots,w_{n-1}\rangle$ iff push(t)=0 and pop(t)=1
- W(t+1) = W(t) iff push(t)=0 and pop(t)=0

n=N:

- W(t+1) = $\langle in, w_1,w_2,\ldots,w_{N-1}\rangle$ iff push(t)=1 and pop(t)=1
- W(t+1) = W(t) iff push(t)=* and pop(t)=0
- W(t+1) = $\langle w_1,w_2,\ldots,w_{N-1}\rangle$ iff push(t)=0 and pop(t)=1

Fill the fifo.v code, and write a testbench fifo_tb.v which instantiates fifo(4,2) module (set N,M to 4,2 from the test-bench using parameter overriding). Your test-bench must print a final answer such as "PASSED ALL TESTS".