Project 2: Verilog Behavioral Modeling (10%)

ENEE 359a: Digital VLSI Design, Spring 2007
Assigned: Tuesday, February 20; Due: Tuesday, March 6

1. Purpose

The purpose of this assignment is to learn the rudiments of the Verilog hardware description language in the context of sequential circuits: you are going to build a simple state machine. Some of the most important concepts you will learn are those of *non-blocking assignments* and *concurrency.* Non-blocking assignments are specific to the Verilog language; concurrency is a powerful concept that shows up at all levels of digital circuit and digital system design. Your example will be a simple finite state machine that produces the n^{th} Fibonacci number given *n* as an input.

2. The Fibonacci State Machine

You are to design using Verilog a controller/datapath that accepts as input a number *n* on the port *input_number* and a clock to drive the controller FSM; the module should output *output_number*, the nth Fibonacci number, some number of clocks later.

The nth Fibonacci number is the sum of the (n-1) and (n-2) Fibonacci numbers, and the first two numbers in the sequence are 1 and 1. So the Fibonacci sequence is:

```
1, 1, 2, 3, 5, 8, 13, 21, ...
```

A *reset_n* signal is necessary to initialize your state machine in a reset state and initialize the two registers in your design that hold the (n-1) and (n-2) Fibonacci numbers of the sequence to 1 and 1 when *reset_n* is low. Your reset using the signal *reset_n* should be asynchronous. Your design should accept the input value of *n* on the port input_number when a start signal called go is high and *reset_n* is high. After *go* transitions from low to high, the design does not require any more inputs other than the clock *clk*. The output consists of the nth Fibonacci number on the port *output_number* and a *done* signal of 1 indicating that the output is valid. When the *done* signal is 0, then *output_number* can be anything. The *done* signal should only be high for one clock cycle.

The 23rd Fibonacci number is 46,368 (b520 in hex), which is the largest number your design is required to be able to generate. So the input is 5 bits wide (to accommodate the number 23), while the output is 16 bits wide (to accommodate 46,368).

You should use this module declaration (provided in the project directory):

```
module fibonacci (input_number, reset_n, go, clk, done, output_number);

       input [4:0] input_number ;
       input reset_n  ;
       input go ;
       input clk ;
       output done ;
       output [15:0] output_number;

endmodule
```

You must use the provided testbench to test your design (unmodified). The testbench tests your design with three different input numbers. Correctness and the total number of clock cycles required to calculate the three fibonacci output numbers is reported by the testbench.

Additional details:

1. Your Fibonacci state machine must *calculate* the *n*th Fibonacci number. Using a ROM with stored Fibonacci numbers, or a state machine with approximately 23 states, or similar approach is not permitted.
2. A couple of software-oriented approaches to stay away from since they are not synthesizable (you will later synthesize your design): There is no hardware analog to the Verilog *initial* block construct. The loop limit for a for loop or a while loop cannot be variable. In general, functions are synthesizable, but recursion is not synthesizable.
3. Some of the main hardware of a typical fibonacci calculator design include a controller (state machine), two registers to hold the (n-1) and (n-2) fibonacci numbers, an adder, and a counter to count up to the input_number.
4. The only type of register permitted in the design is rising-edge-triggered.
5. In a clocked always block you should use the nonblocking assignment operator, <=, when assigning to a signal that is synthesized as a register.

3. Running & Submitting Your Project

First, tap verilog to get access to the simulators; i.e., at the Unix prompt type the following:

```
tap cds-v
```

Then you can invoke your code this way:

```
verilog testbench.v yourcode.v
```

or

```
ncverilog testbench.v yourcode.v
```

The ncverilog simulator has a longer start-up time but executes much faster once it gets going.

When you submit your code to me via email, all I want is the fibonacci.v code … i.e., everything but the testbench.v file (I will use my own). Do not change any of the variable names within the fibonacci.v file, for hopefully obvious reasons.