ECE 351

Professor Greenwood “somebody who doesn’t know the answer tell me…”

04/23/19

* **Repeat Loop**
  + Syntax

*repeat (num)*

*begin*

*…*

*end*

* + - num: is a decimal number.
* **Forever Loop**
  + Instead of:

*initial*

*clock = 1’b0;*

*always*

*#5 clock = ~clock; // Clock with 5ns period*.

* + Do:

*Initial*

*Begin*

*Clock = 1’b0*

*Forever #5 clock = ~clock;*

*End*

* **Synchronizing 2 reg values**

*reg clock;*

*reg A, B;*

*Initial*

*Forever @(posedge clock) A = B;*

* **Sequential Blocks**

*initial*

*begin*

*x = 1’b0;*

*y = 1’b1;*

*z = {x, y}; // z = 01*

*w = {y, z}; //w = 10*

*end*

* + ***With Delays:***

*initial*

*begin*

*x = 1’b0; //t = 0;*

*#5 y = 1’b1; // t = 5*

*#10 z = {x, y}; //t = 15*

*#20 w = {y, z}; //t = 35*

*End*

* **Parallel** **Blocks**: A block where all statements execute concurrently (subject to any # delays)
  + Unlike the sequential blocks we have been dealing with.
  + Statements execute concurrently even though blocking assignments are used.
  + Delays are with respect to the time the parallel block was entered.
  + “begin” and “end” are not used in parallel blocks.
    - “fork” and “join” take their place.

*initial*

*fork*

*x = 1’b0; //t = 0;*

*#5 y = 1’b1; // t = 5*

*#10 z = {x, y}; //t = 10*

*#20 w = {y, z}; //t = 20*

*Join*

* + Time delays become absolute with respect to the time the parallel block was entered.
* Example 7-36
  + In the always statement list priority goes to the first signal listed, then the next, etc.
* **Finite State Machines**
  + Mealy
    - State diagram
      * Inputs and resulting outputs are shown on the transition arcs from one state to the other.
  + Moore
    - State diagram
      * The state node indicates the state and (all) output(s).
      * Transitions display the inputs that will cause a change from one state to the other.
* **Moore FSM**
  + parameter one1 = 3’b001, two1s = 3’b100, zero = 3’b010
    - one hot encoding.
  + The simulator will allow for synthesizing either full encoding or one-hot encoding.
* **Tasks and Functions**
  + In Verilog we declare tasks and functions in the module for which they are use.
    - This is unlike C where functions can be declared and used externally.