ECE 351

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

04/11/19

* Figure 5-6: **1-bit Full adder**
  + As with any module, any nets not covered by the port list will need internal declarations.
    - As “wire” or perhaps “reg”.
* Figure 5-8: **Module D**
  + “#” represents the propagation delay.
  + Time delay base is 1ns by default in most simulators.
    - This can be modified.
* Figure 5-9: **Waveforms for Delay Simulation**
  + The “X” on the E and OUT signals are undefined during the propagation time for each logic gate with delays #4 and #5.
  + This would be the waveform result of using the “#” syntax.
  + Never use the “#” syntax.
  + Using the “#” syntax implies that a specific technology is used.
    - This is bad because:

1. **Not synthesizable**.
2. HDLs are supposed to be technology-independent.

* **Dataflow Level of Abstraction**
  + **reg** variables hold values.
  + **net** variables don’t.
* Methods of **Assigning net variable values**

1. Drive with a module output.
   1. Gate primitives are also considered modules.
2. With a “Continuous Assignment Statement.”

* **Continuous Assignment Statement**
  + Unique to the Dataflow level of abstraction.
    - Recognizing a continuous assignment statement means the description is at the Dataflow level of abstraction.
  + *Continuously* drives the net variable.
  + **LHS** updates whenever the **RHS** changes.
  + Syntax:

**assign** LHS = RHS;

* + - **assign** keyword (reserved).
    - LHS Variable Identifier for Net variable (Left-hand side)
    - RHS Some expression, arithmetic, or logical. (Right-hand side)
  + *Actual* Syntax:

**assign** [Strength Level] [delay] LHS = RHS;

* + - However, neither Strength Level, nor delay, are synthesizable.
* **Bit Select**
  + Suppose: wire[7:0]A;
  + A[2] is a bit select of bit 2 of wire a.
* **Expression:** A combination of operators and operands to get some result.
* **Operand**: a data element (a net variable or constant).
* **Operator**: anything that acts on an **operand.**
* **Arithmetic Operators: ‘**+’, ‘-‘, ‘/’, ‘\*’, ‘%’ (modulus), ‘\*\*’ (power)
  + Each requires only two **operands**.
  + The LHS size is determined by the size of the largest **operand**.
  + Example:

wire [3:0]A, B, C;

wire [5:0]D;

…

A = B + C; // 4 bit result. (largest operand is 4 bits)

D = B + C; // 6-bit result. (largest operand is 6 bits)

* + If any bit on the **RHS** is a ‘x’ the result is ‘x’ (don’t care).
* **Logical Operators:** ‘!’, ‘&&’ (and), and ‘||’(or)
  + Primary use will be in behavioral descriptions for if, then , else statements
  + Entire variable is considered as:
    - A logic 0
    - A logic 1
      * Is always the case if the operand doesn’t = 0 (or x or z).
    - A don’t care ‘x’
    - A high impedance ‘z’
* **Bitwise Logical Operators**: ‘&’, ‘|’, ‘~’ (negation), ‘^’(xor), ‘~^’ or ‘^~’ (xnor)
* **Concatenation Operator**: ‘{ }’
  + Suppose: assign D[7:4] = {D[0], D[1], D[3], D[6]};
  + Then D[7:4] = 4’b1011;
  + Suppose: assign D = {D[3:0], D[7:4]};
  + Then D = {D[3], D[2], D[1], D[0], D[7], D[6], D[5], D[4]}
* **Reduction Operators**: ‘&’, ‘~&’, ‘|’, ‘~|’, ‘^’, ‘^~’, ‘~^’
  + The compiler will use context by number of operands to determine if reduction operators or logical operators are used.
  + This is great for parity checking.
  + Applies the bitwise operator on pairs of bits from right to left
  + Suppose: y = 6’b011010

&y = 0110\_1&0

= 011\_0&0

= 01\_1&0

= 0\_1&0

= 0&0

= 0

* **Shift Operators:**
  + Syntax: LHS = variable << [Decimal Number]
* **Conditional Operators:**
  + Syntax: **assign** LHS = predicate ? value1 : value2;
    - The **predicate** is only true or false.
* **Replication Operator:**
  + Suppose: Y = {4{A}}; where A = 1’b1
  + Results: Y = 4’b1111
* Dataflow Description: **2x1 Mux** 
  + Inputs A, B, S.
  + Output Y.
  + **assign** Y = S ? B : A;
* Dataflow Description:  **Buffer**
  + Inputs A, C
  + Output B
  + **assign** B = C ? A : 1’bz;
* Dataflow Description: **8-bit Comparator**
  + Inputs (unsigned) A[7:0], B[7:0]
  + Outputs “A>B”, “A<B”, “A=B”
  + Example:

module comp(A, B, AGTB, ALTB, AEQB)

input [7:0]A, B;

output AGTB, AEQB, ALTB;

**assign** AGTB = A>B;

**assign** ALTB = A<B;

**assign** AEQB = A==B;

endmodule