ECE 2300
Digital Logic & Computer Organization
Spring 2021

More Sequential Logic
Verilog
Announcements

• Lab 2 due times are updated
  – Prelab 2A due today at 11:59PM
• HW3 will be posted tonight
• Prelim 1
  – Tuesday March 16, 1:00-2:15pm, over zoom
    • Inform 2300 staff asap if you have a schedule conflict
  – Coverage: Lectures 1~7
    • Binary number, Boolean algebra, CMOS, combinational logic, sequential logic, and Verilog
  – TA-led review session will be scheduled (& recorded) on Thursday March 11
  – An old prelim exam will be posted later this week
  – More information to be announced soon
S-R Latch

- **S-bar-R-bar latch**
  - Built from NAND gates
  - Inputs are active low rather than active high
  - When both inputs are 0, Q = QN = 1 (avoid!)

\[
\begin{array}{c|c|c|c}
S & R & Q & QN \\
\hline
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 1 & Last Q & Last QN \\
\end{array}
\]
D Latch and Flip-Flop

- **Latch: level sensitive**
  - Captures the input when enable signal asserted

- **Flip-Flop (FF): edge sensitive**
  - Captures the input at the triggering clock edges (e.g., L→H)
  - A single FF is also called a one-bit register
Recap: D Flip-Flop (DFF)

- Copies D to Q on the rising edge of the clock
DFF Timing Example

Circuit diagram

Waveform
(assuming Y & Z are initialized with 0s)
Another Example

Circuit diagram

Waveform (assuming Y & Z are initialized with 0s)
T (Toggle) Flip-Flop

- Output toggles only if T=1
- Output does not change if T=0
- Useful for building counters

Q: 0, 1, 0, 1, 0, 1, 0, ...

[Diagram of T Flip-Flop and D Flip-Flop connected]
T (Toggle) Flip-Flop

- Output toggles only if T=1
- Output does not change if T=0
- Useful for building counters

\[ Q_{\text{next}} = T \cdot Q' + T' \cdot Q \]

Q: 0, 1, 0, 1, 0, 1, 0, ...
## Binary Counters

- **Counts in binary in a particular sequence**
- **Advances at every tick of the clock**
- **Many types**

<table>
<thead>
<tr>
<th>Up</th>
<th>Down</th>
<th>Divide-by-n</th>
<th>n-to-m</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>1 1 1</td>
<td>0 0 0</td>
<td>n</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1 1 0</td>
<td>0 0 1</td>
<td>n+1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1 0 1</td>
<td>0 1 0</td>
<td>n+2</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1 0 0</td>
<td>0 1 1</td>
<td>.</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 1 1</td>
<td>1 0 0</td>
<td>m-1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 1 0</td>
<td>.</td>
<td>m</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>n</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>n-1</td>
<td>n+1</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>0 0 0</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>0 0 1</td>
<td>.</td>
</tr>
</tbody>
</table>
## Up Counter Sequence

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>Toggles every clock tick</td>
</tr>
<tr>
<td>0 0 1</td>
<td>Toggles every clock tick that two right bits = 11</td>
</tr>
<tr>
<td>0 1 0</td>
<td>Toggles every clock tick</td>
</tr>
<tr>
<td>0 1 1</td>
<td>Toggles every clock tick that right bit = 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>Toggles every clock tick</td>
</tr>
<tr>
<td>1 0 1</td>
<td>Toggles every clock tick</td>
</tr>
<tr>
<td>1 1 0</td>
<td>Toggles every clock tick</td>
</tr>
<tr>
<td>1 1 1</td>
<td>Toggles every clock tick</td>
</tr>
</tbody>
</table>

Lecture 7: 11
Building Binary Up Counter

Q0 toggles at every rising edge
Q1 toggles at the rising edge when Q0=1
Q2 toggles at the rising edge when Q0=Q1=1
Up Counter Timing Diagram

CLK
Q0
Q1
Q2

Count 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0
Hardware Description Languages

• **Hardware Description Language (HDL): a language for describing hardware**
  – Efficiently code large, complex designs
    • Programming at a more abstract level than schematics
  – CAD tools can automatically synthesize circuits

• **Industry standards:**
  – **Verilog:** We will use it from Lab 2
  – **SystemVerilog:** Successor to Verilog, gaining wide adoption
  – **VHDL (Very High Speed Integrated Circuit HDL)**
Verilog

• Developed in the early 1980s by Gateway Design Automation (later bought by Cadence)

• Supports modeling, simulation, and synthesis
  – We will use a (synthesizable) subset of the language features

• Major language features (in contrast to software programming languages)
  – Structure and instantiation
  – Concurrency
  – Bit-level behavior
Values

- **Verilog signals can take 4 values**
  - 0 Logical 0, or false
  - 1 Logical 1, or true
  - x Unknown logical value
  - z High impedance (Hi-Z), floating/non-connected

x might be a 0, 1, z, or in transition, or don’t cares
Sometimes useful debugging and often exploited by CAD tools during optimization
Bit Vectors

- Multi-bit values are represented by bit vectors (i.e., grouping of 1-bit signals)
  - Right-most bit is always least significant
  - Example
    - input[7:0] byte1, byte2, byte3; /* three 8-bit inputs */

- Constants
  - 4'b1001
    - Base format (b,d,h,o)
    - Decimal number representing bit width
  - Binary Constants
    - 8'b00000000
    - 8'b0xx01xx1
  - Decimal Constants
    - 4'd10
    - 32'd65536
Operators

- **Bitwise Boolean operators**
  - ~ NOT
  - & AND
  - ^ Exclusive OR
  - | OR

- **Arithmetic operators**
  - + Addition
  - – Subtraction
  - * Multiplication
  - / Division
  - % Modulus
  - << Shift left
  - >> Shift right
Verilog Program Structure

- **System is a collection of modules**
  - Module corresponds to a single piece of hardware

- **Declarations**
  - Describe names and types of inputs and outputs
  - Describe local signals, variables, constants, etc.

- **Statements** specify what the module does
module M_2_1 (x, y, sel, out);

input x, y;
input sel;
output out;
wire tx, ty;

AND and0 (x, ~sel, tx);
AND and1 (y, sel, ty);
OR  or0  (tx, ty, out);

endmodule
Verilog Hierarchy

A module can instantiate other modules forming a module hierarchy
Verilog Programming Styles

• **Structural**
  – Shows how a module is built from other modules via *instance* statements
  – Textual equivalent of drawing a schematic

• **Behavioral**
  – Specify what a module does in high-level description
  – Use procedural code (e.g., in *always* block) and continuous assignment (i.e., *assign*) constructs to indicate what actions to take

We can mix the structural and behavioral styles in a Verilog design
module M_2_1 (x, y, sel, out);
  input  x, y;
  input  sel;
  output out;

  wire tx, ty;

  AND and0 (x, ~sel, tx);
  AND and1 (y, sel, ty);
  OR  or0 (tx, ty, out);

endmodule

The order of the module instantiation does not matter
Essentially describing the schematic textually
Behavioral Style with Continuous Assignments

• An assign statement represents continuously executing combinational logic

```verilog
module MUX2_1 (x, y, sel, out);
    input  x, y;
    input  sel;
    output out;

    assign out = (~sel & x) | (sel & y);

endmodule
```

• Multiple continuous assignments happen in parallel; the order does not matter
(Behavioral Style) Combinational Logic with Always Blocks

module MUX2_1 (x, y, sel, out);
  input  x, y;
  input   sel;
  output reg  out;
  
  always @(x, y, sel)
  begin
    out <= (~sel & x) | (sel & y);
  end

endmodule

• An always block is reevaluated whenever a signal in its sensitivity list changes

• Formed by procedural assignment statements
  – reg needed on the LHS of a procedural assignment
Sequential Logic in Always Blocks

reg Q;

always @( clk, D )
begin
  if ( clk )
    Q <= D;
end

always @( posedge clk )
begin
  Q <= D;
end

• Sequential logic can only be modeled using always blocks

• Q must be declared as a “reg”
Blocking Assignments

- **Blocking assignments** (\( = \))
  - **Simulation behavior**: Right-hand side (RHS) evaluated sequentially; assignment to LHS is immediate

```verilog
class reg Y, Z;
always @ (posedge clk)
begin
    Y = A & B;
    Z = ~Y;
end
```

**Blocking assignments**

**Simulator interpretation**

\[
Y_{\text{next}} = A \& B \\
Z_{\text{next}} = \neg(A \& B)
\]

**Resulting circuit** (post synthesis)

When reg Y appears on RHS of a blocking assignment, use its **input signal** for connection.
Nonblocking Assignments

- **Nonblocking assignment** ( <= )
  - **Simulation behavior**: RHS evaluated in parallel (order doesn’t matter); Assignment to LHS is delayed until end of always block

```verilog
reg Y, Z;
always @(posedge clk)
begin
    Y <= A & B;
    Z <= ~Y;
end
```

**Simulator interpretation**
- \( Z_{\text{next}} = \sim Y \) // reading the old Y
- \( Y_{\text{next}} = A \& B \)

**Resulting circuit** (post synthesis)

When reg Y appears on RHS of a nonblocking assignment, use its **output signal** for connection
Assignments in Verilog

• Continuous assignments apply to combinational logic only

• *Always blocks* contain a set of procedural assignments (blocking or nonblocking)
  – Can be used to model either combinational or sequential logic
  – Always blocks execute concurrently with other always blocks, instance statements, and continuous assignment statements in a module
Before Next Class

- H&H 3.4, 4.6

Next Time

Finite State Machines