ECE 2300
Digital Logic & Computer Organization
Spring 2022

More Sequential Logic
Verilog
Announcements

• Updates to TA OH posted on Ed
• Prelab 2A due tomorrow
  – Form groups on CMS for Lab 2A

• Prelim 1
  – Thursday Feb 24, 1:00-2:15pm in class
    • closed book, closed notes, closed Internet
  – Coverage: Lectures 1~7
    • Binary number, Boolean algebra, CMOS, combinational logic, sequential logic, and Verilog
  – An old prelim exam will be posted tomorrow
  – More information to be announced soon
    • TA-led review session will be scheduled (& recorded)
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!)

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>QN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Last Q</td>
<td>Last QN</td>
</tr>
</tbody>
</table>
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., $L \rightarrow H$)
  - 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

<table>
<thead>
<tr>
<th>D</th>
<th>CLK</th>
<th>Q</th>
<th>QN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>Last Q</td>
<td>Last QN</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>Last Q</td>
<td>Last QN</td>
</tr>
</tbody>
</table>
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, ...

---

Lecture 7: 8
T (Toggle) Flip-Flop

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

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

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>n-1</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 0 0</td>
<td>n+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 0 1</td>
<td></td>
</tr>
</tbody>
</table>
Up Counter Sequence

0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

Toggles every clock tick that right bit = 1

Toggles every clock tick that two right bits = 11

Toggles every clock tick
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
Evolution of Design Abstractions

- Transistor-level entry
- Gate-level entry
- HDL (Verilog, VHDL)
- High-level programming language (?)

Design Productivity vs. CAD Tool Effort

[Figure credit: Kurt Keutzer]
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] a1, a2, a3; /* three 8-bit inputs */

- Constants
  - 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**

  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**
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
(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 (=)

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

Y and Z are inferred as FFs since the always block is sensitive to the clock edge.

Simulation behavior

```plaintext
Y_{next} = A & B
Z_{next} = ~(A & B) /* negating the new Y */
```

• Right-hand side (RHS) evaluated sequentially
• Assignment to LHS is immediate

Actual circuit

- When a reg appears on RHS of a blocking assignment (“Y” here), use its input for connection (“A&B” here)
Nonblocking Assignments

• **Nonblocking assignment (<=)**

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

  Y and Z are inferred as FFs since the always block is sensitive to the clock edge

  **Simulation behavior**

  \[
  Z_{next} = \sim Y \quad /* \text{negating the old } Y */ \\
  Y_{next} = A \& B
  \]

• RHS evaluated in parallel (order doesn’t matter)
• Assignment to LHS is delayed until end of always block

When a reg appears on RHS of a nonblocking assignment (Y here), use its output for connection
Before Next Class

• H&H 3.4, 4.6

Next Time

Finite State Machines