ECE 2300
Digital Logic & Computer Organization
Spring 2024

More Sequential Logic
Verilog
Announcements

• Prelab 2A due tomorrow
  – Form groups on CMS for Lab 2A

• HW 3 will be released tomorrow
**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→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
(assume both DFFs hold 0s initially)
Another DFF Timing Example

A DFF is often termed a “delay element” because it introduces a delay in data propagation. This delay results from updating the DFF state/output only on a clock edge. Beyond its important role as a storage element for holding states, the delay introduced by a DFF is also crucial for timing control and sequencing in digital systems.
T (Toggle) Flip-Flop

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

(when T=1)
Q: 0, 1, 0, 1, 0, 1, 0, ...

![Diagram of T (Toggle) Flip-Flop]
**T (Toggle) Flip-Flop**

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

(when $T=1$)

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

<table>
<thead>
<tr>
<th>$T$</th>
<th>$Q$</th>
<th>$Q_{next}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$Q_{next} = T \cdot Q' + T' \cdot Q$
**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-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>n</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>n+1</td>
</tr>
</tbody>
</table>

Lecture 7: 10
Up Counter Sequence

- Toggles every clock tick that right bit = 1
- Toggles every clock tick that two right bits = 11
- Toggles every clock tick

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

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

\[ \begin{array}{ccc}
Q_2 & Q_1 & Q_0 \\
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
1 & 1 & 1 \\
0 & 0 & 0 \\
0 & 0 & 1 \\
\end{array} \]
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
Evolution of Design Abstractions

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

[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 start using 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 means unknown/uninitialized (could be 0, 1, z, or in transition) or don't cares
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
  - 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
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 continuous assignment (`assign`) constructs and/or procedural code (in `always` blocks) to indicate what actions to take

We can mix the structural and behavioral styles in a Verilog design
Structural Style

module MUX_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 MUX_2_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)
  – An always block can be used to model either combinational or sequential logic
    • *Sequential logic can only be modeled using always blocks*
  – Always blocks execute concurrently with other always blocks, instance statements, and continuous assignment statements in a module
(Behavioral Style) Combinational Logic with Always Blocks

```verilog
define module MUX_2_1 (x, y, sel, out);
  define input x, y;
  define input sel;
  define output reg out;
  define 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
  - The left-hand side (LHS) of a procedural assignment must be declared as a “**reg**”
Sequential Logic in Always Blocks

Sequential logic can ONLY be modeled using always blocks

```verilog
reg Q;

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

D latch

Q is declared as a "reg" because it appears on the LHS of a procedural assignment

```verilog
reg Q;

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

DFF
Before Next Class

• H&H 3.4, 4.6

Next Time

More Verilog
Finite State Machines