Digital Systems Laboratory

Lecture #2

by

Dr. Choon Kim

CSE Department, UCSD
chk034@eng.ucsd.edu
Digital Technologies

CPU (Central Processing Unit)
GPU (Graphics Processing Unit)
DSP (Digital Signal Processor)
SoC (System on Chip)
FPGA (Field Programmable Gate Array)
ASIC (Application Specific Integrated Circuit)
Custom Designs
etc.
FPGAs (Field Programmable Gate Arrays)

- SRAM based (Flash memory)
- Antifuse

**Disadvantages:** Penalty on area, density, speed

**Advantages:** Flexibility, low startup costs, low risk, revisions without changing the hardware
Periodic table

Periodic Table
Transistors: Silicon

- Transistors are built out of silicon, a semiconductor
- Pure silicon is a poor conductor (no free charges)
- Doped silicon is a good conductor (free charges)
  - n-type (free negative charges, electrons)
  - p-type (free positive charges, holes)
MOS Transistors

- **Metal Oxide Semiconductor (MOS)** transistors:
  - Polysilicon (used to be *metal*) gate
  - Oxide (silicon dioxide) insulator
  - Doped **silicon**

![Diagram of MOS transistor](image-url)
Transistors: nMOS

**Gate = 0**, it is OFF (source and drain are disconnected)

**Gate = 1**, it is ON (channel between source and drain)

**Source = 0** => **Drain = 0**

**Source = 1** => **Drain = 0.8** (Poor one)
Transistors: pMOS

• pMOS transistor is just the opposite
  – ON when Gate = 0
    • Source = 0 => Drain = 0.2 (Poor zero)
    • Source = 1 => Drain = 1
  – OFF when Gate = 1
Transistor Function

nMOS

g = 0
OFF

s

nMOS

g = 1
ON

s
d

pMOS

g = 0
OFF

s
d

pMOS

g = 1
ON

s
d
Transistor Function

- nMOS transistors pass good 0’s, so connect source to GND
- pMOS transistors pass good 1’s, so connect source to $V_{DD}$
CMOS (Complementary MOS) Gate Structure

- **pMOS** pull-up network
- **nMOS** pull-down network

Inputs → pMOS pull-up network → Output

Inputs → nMOS pull-down network → Output
CMOS Gates: NOT Gate

\[ Y = \overline{A} \]

<table>
<thead>
<tr>
<th>A</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
V_{DD} \quad A \quad Y \quad \overline{N1} \quad \overline{GND}
\]

<table>
<thead>
<tr>
<th>A</th>
<th>P1</th>
<th>N1</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
CMOS Gates: NOT Gate

\[ Y = \overline{A} \]

<table>
<thead>
<tr>
<th>A</th>
<th>P1</th>
<th>N1</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ON</td>
<td>OFF</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>OFF</td>
<td>ON</td>
<td>0</td>
</tr>
</tbody>
</table>
CMOS Gates: NAND Gate

NAND

\[ Y = \overline{AB} \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>
CMOS Gates: **NAND Gate**

**NAND**

\[ Y = \overline{AB} \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>P1</th>
<th>P2</th>
<th>N1</th>
<th>N2</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>0</td>
</tr>
</tbody>
</table>
Q: How do you build a three-input NOR gate?
Homework: Build other gate using CMOS

Q: How do you build AND gate?
Q: How do you build OR gate?
Q: How do you build XOR gate?
Q: How do you build XNOR gate?

etc.
Transmission Gates

- nMOS pass 1’s poorly
- pMOS pass 0’s poorly
- Transmission gate is a better switch
  - passes both 0 and 1 well
- When $EN = 1$, the switch is ON:
  - $A$ is connected to $B$
- When $EN = 0$, the switch is OFF:
  - $A$ is not connected to $B
Noise

- Anything that degrades the signal
  - E.g., resistance, power supply noise, coupling to neighboring wires, etc.
- **Example:** a gate (driver) could output a 5 volt signal but, because of resistance in a long wire, the signal could arrive at the receiver with a degraded value, for example, 4.5 volts

![Diagram](attachment:diagram.png)
Logic Levels

Driver

Receiver

Output Characteristics

Input Characteristics

Logic High Output Range

Logic Low Output Range

Forbidden Zone

V_{OH}

V_{OL}

V_{DH}

V_{IL}

V_{IH}

V_{IL}

GND

V_{DD}

NM_{H}

NM_{L}
V_{DD} Scaling

- Chips in the 1970’s and 1980’s were designed using $V_{DD} = 5$ V
- As technology improved, $V_{DD}$ dropped
  - Avoid frying tiny transistors
  - Save power
- 3.3 V, 2.5 V, 1.8 V, 1.5 V, 1.2 V, 1.0 V, ...
- Be careful connecting chips with different supply voltages
# Logic Family Examples

<table>
<thead>
<tr>
<th>Logic Family</th>
<th>$V_{DD}$</th>
<th>$V_{IL}$</th>
<th>$V_{IH}$</th>
<th>$V_{OL}$</th>
<th>$V_{OH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTL</td>
<td>5 (4.75 - 5.25)</td>
<td>0.8</td>
<td>2.0</td>
<td>0.4</td>
<td>2.4</td>
</tr>
<tr>
<td>CMOS</td>
<td>5 (4.5 - 6)</td>
<td>1.35</td>
<td>3.15</td>
<td>0.33</td>
<td>3.84</td>
</tr>
<tr>
<td>LVTTL</td>
<td>3.3 (3 - 3.6)</td>
<td>0.8</td>
<td>2.0</td>
<td>0.4</td>
<td>2.4</td>
</tr>
<tr>
<td>LVCMOS</td>
<td>3.3 (3 - 3.6)</td>
<td>0.9</td>
<td>1.8</td>
<td>0.36</td>
<td>2.7</td>
</tr>
</tbody>
</table>
Power Consumption

- Power = Energy consumed per unit time
- Two types of power consumption:
  - Dynamic power consumption
  - Static power consumption
Dynamic Power Consumption

- Power to charge transistor gate capacitances
- The energy required to charge a capacitance, $C$, to $V_{DD}$ is $CV_{DD}^2$
- If the circuit is running at frequency $f$, and all transistors switch (from 1 to 0 or vice versa) at that frequency, the capacitor is charged $f/2$ times per second (discharging from 1 to 0 is free).

$$P_{dynamic} = \frac{1}{2}CV_{DD}^2f$$
Static Power Consumption

• Power consumed when no gates are switching
• It is caused by the quiescent supply current, $I_{DD}$, also called the leakage current
• Thus, the total static power consumption is:

$$P_{\text{static}} = I_{DD}V_{DD}$$
Power Consumption Example

- Estimate the power consumption of a wireless handheld computer
  - $V_{DD} = 1.2$ V
  - $C = 20$ nF
  - $f = 1$ GHz
  - $I_{DD} = 20$ mA
Power Consumption Example

- Estimate the power consumption of a wireless handheld computer
  - $V_{DD} = 1.2$ V
  - $C = 20$ nF
  - $f = 1$ GHz
  - $I_{DD} = 20$ mA

$$P = \frac{1}{2}CV_{DD}^2f + I_{DD}V_{DD}$$

$$= \frac{1}{2}(20 \text{ nF})(1.2 \text{ V})^2(1 \text{ GHz}) + (20 \text{ mA})(1.2 \text{ V})$$

$$= 14.4 \text{ W} + 0.024\text{W}$$

$$= 14.424\text{W}$$
Multiplexers

- If S is 0, then $I_0$ will pass and $I_1$ is blocked. Thus, $y = I_0$.
- Likewise, if S is 1, $y = I_1$. 

![Multiplexer Diagram](image)
More than 2-to-1 Multiplexers

- We can also make a 4:1 MUX using three 2:1 MUX

If \( S_1S_0 = 00 \), then \( S_1 \) will select MUX from A and B. Since \( S_0 = 0 \), \( Z = A \)

<table>
<thead>
<tr>
<th>( S_1 )</th>
<th>( S_0 )</th>
<th>( Z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
</tbody>
</table>
Multiplexers

- We can make 4:1 and above MUXes too.
- With 4 inputs, our selector needs to have two bits.
Mux example: Logical function unit

<table>
<thead>
<tr>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>Function</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>always 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A + B</td>
<td>logical OR</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(A \cdot B)'</td>
<td>logical NAND</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A \oplus B</td>
<td>logical xor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A \oplus \overline{B}</td>
<td>logical xnor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A \cdot B</td>
<td>logical AND</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(A + B)'</td>
<td>logical NOR</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>always 0</td>
</tr>
</tbody>
</table>

Question: What is the relation between number of inputs and the min. number of bits of selector?
Design example: 1-bit binary adder

- **Inputs:** A, B, Carry-in
- **Outputs:** Sum, Carry-out
  - Sum = $A \text{ xor } B \text{ xor } \text{Cin}$
  - Cout = $A \text{ B } + A \text{ Cin } + B \text{ Cin}$
    $= A \text{ B } + \text{Cin } (A \text{ xor } B)$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Cout</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
4-bit Ripple Carry Adders

- Chain 4 1-bit full adders together.
  \[ A_3A_2A_1A_0 + B_3B_2B_1B_0 = C_4 S_3S_2S_1S_0 \]
- Connect the carry-out of the previous adder and the carry-in of the next adder.
- Worst delay path (critical path): from \( A_0 \), \( B_0 \), or \( C_0 \) to \( S_3 \), or \( C_4 \)
Critical Path for worst delay

Propagation from $C_0$ to $C_4$
Welcome to Verilog

Some issues before starting:

SystemVerilog?

Synthesizable Subset
Big question when you **synthesize**!

**Question:**
Can I use **any construct** defined in Verilog HDL definition in my design if my goal is to **synthesize** my design into chip?

**Answer:**
**No.** All the constructs of Verilog are simulatable. However, not all the constructs of Verilog are synthesizable. Each CAD synthesizer **supports** its own **synthesizable subset** of the Verilog constructs only.

Then, which constructs are supported by our Altera Quartus II synthesizer for synthesis?
You have to review and use the following Altera's synthesizable subset support list,

(****Remember a warning, "Rule for handling possible incorrect information found on documents:" in later slide. You still need to verify any information by testing on Quartus II SW & DE1 board HW.****)

**Quartus II Verilog HDL Support**
**Quartus II Support for Verilog 2001**

Also, following information may help you when coding Verilog in general.
**Altera's Recommended HDL Coding Styles**
Verilog in one slide!
By Choon Kim

module module_name ( port specification... );

local wires, variables declaration;
task, function declaration;

continuous assignments ; // for mainly combinational circuit
procedural blocks; // for mainly sequential circuit
instantiation of modules ; // for hierarchical design
instantiation of primitives /UDP; // for built-in primitives

endmodule
Introduction to Verilog HDL

Rule for handling possible incorrect information found on documents:

Electronics, CAD Technology and Verilog HDL keep changing continuously. Therefore some information contained in the following documents (or any document in general) may be obsolete, incorrect or not working. In our CSE140L class, the way to verify the correctness of an information is to test it by simulating, compiling and testing the result on our FPGA DE1 board using our CAD SW (Altera Quartus II Web Edition Software v9.0 Service Pack 2).

1. [Quartus II example practice using Verilog]: (Note: You can do this example without understanding of Verilog.) Follow the instructions described in tut_quartus_intro_verilog_de1

2. [Verilog Tutorial for beginner]: Follow Verilog short tutorial to make yourself familiar with this new design entry methodology. Altera’s Introduction to Verilog, Altera’s Verilog HDL Basics

3. [References]: intro_verilog, Altera’s Recommended HDL Coding Styles

4. [Recommended Verilog books] Note that so many good Verilog books are available in the world and each book is different in various ways. In the end you yourself should find a book best suitable for your own unique situation:
   - A Verilog HDL Primer (Third Edition) -- 2005 by J. Bhasker
   - Verilog HDL (2nd Edition) -- 2003 by Samir Palnitkar

5. [Quick Reference Card]: Two-page Card, Multiple-page Card

6. In addition, there are numerous Verilog information & tutorials available on the web in doc, book, & video format. The Verilog on wiki is an excellent place to learn the language. It also lists many tutorial links that student may visit and learn.