Lecture 1:
1) Introduction to Digital Systems
2) Digital circuit design review
What is a Digital System?

- **Structure:** A collection of interconnected digital modules designed to perform a particular function.

- **Function:** Takes a set of discrete information inputs and discrete internal information (system state) and generates a set of discrete information outputs.
Example: Odometer, a digital counter

- **Inputs:** Count up, Reset
- **Outputs:** Display
- **State:** Value of stored digits
Example: Washer and Dryer

- **Inputs:** Dirty (Wet) Clothes
- **Outputs:** Clean (Dry) Clothes
- **State:** Water Level, Temperature, Humidity
Example: Computers, Servers

- **Inputs:** Keyboard, Mouse, Ethernet Card, Microphone
- **Outputs:** Display, Ethernet Card, Speakers, Users
- **State:** ?
More Examples

- Alarm / security system
- Cruise control
- Heating/AC thermostat
- Microwave oven
- Vending machine
- Gas pump
- Sprinkler/Irrigation system controller
- Mars Rover
Digital System Modules

- Low level modules
  - Gates: AND, OR, XOR, BUFFER, etc
  - Blocks: Adder, subtractor, shifter, etc

- High level modules
  - PLD (Programmable Logic Device)
  - ASICS (Application Specific Integrated Circuit)
  - Microprocessors/Microcontrollers
Low Level Modules: Gates

Functions: done in class
Low Level Modules: Blocks

Temperature Averager

- 4 registers take temperature inputs
- Three adders
- Shifter (right by 2)

Counter

- Register, incrementer, N-input AND
High Level Modules: PLD

- **PLD (Programmable Logic Device)**
  - Provides specific functions
  - Examples: device-to-device interface, data communication, signal processing, data display, timing and control operations …
  - Quickly develop, simulate, and test designs
  - Prototype PLD is the exact same PLD that can be used in the final design of equipment like router, modem, DVD player, or navigation system
  - Internal logic gates connected by electronic links (‘fuses’) which can be connected (‘blown’) by programming to obtain different circuit configurations

Complex Digital logic designs \[\rightarrow\] Single Device (PLD)
PLD (continued…)

- **Types of PLDs**
  1) SPLDs (Simple Programmable Logic Devices)
     - ROM (Read-Only Memory)
     - PLA (Programmable Logic Array)
     - PAL (Programmable Array Logic)
     - GAL (Generic Array Logic)
  
  2) CPLD (Complex Programmable Logic Device)

  3) FPGA (Field-Programmable Gate Array)
Altera CPLD

Before Programming

After Programming
High Level Modules: ASIC

Layout of a design in 65 nm CMOS technology*

High Level Modules: Microcontroller/Microprocessor

PIC Microcontroller Unit by Microchip

ARDUINO Uno PCB (with Atmel ATmega328 8-bit Microcontroller)

PIC Microcontroller Kit

Qualcomm Snapdragon inside smartphone
Digital System Implementations

- PCB: Printed Circuit Board
- FPGA: Field Programmable Gate Array
- VLSI: Very Large Scale Integration
- SoC: System on Chip
Digital System: PCB

UEIDAQ 64-channel, 16-bit PCI digital I/O board
“FPGA is a digital duct tape”
- “It’s a do-all, catch-all, connect-anything-together, fix-the-last-minute-oversight, future-proofing, standards-hopping, jack-of-all-chips”
- 3 decades of evolution
Digital System: VLSI

AMD Deerhound (K8L) Core Die Photo
Qualcomm Snapdragon 800 processor

- **Krait 400 CPU** features 28HPm process technology superior 2GHz+ performance
- **Adreno 330** for advanced graphics
- **Hexagon QDSP6** for ultra low power applications and custom programmability
- **Integrated Gobi 4G LTE World Mode**, 802.11ac, USB 3.0 and BT 4.0 offers broad array of high speed connectivity

- Ultra HD Capture and Playback DTS-HD and Dolby Digital Plus audio Expanded Gestures
- **Low-power Snapdragon Sensor Core** increases sensor accuracy and efficiency
- **21MP with dual ISP**
- Support for up to 2560x2048 display Miracast 1080p HD support
- **IZat GNSS** with support for three GPS constellations
### Types of Digital System

- **Combinational Logic:**
  - Output = \( f \) (Input) ; output is some function of input

  ![Logic Circuit Diagram](image)

  Example:
  
  \[ y = f(a, b, c, d) \]

- **Boolean Algebra**
  - Use gates to perform logic operations
  - Boolean variables (e.g. a, b, c, d above) are not signals but are closely related
  - Operators: \( \cdot \) (AND), \( + \) (OR), \( \oplus \) (XOR), \( \bar{X} \) (denoted by \( X' \) in the text)
  - Truth tables: next page
Boolean Algebra

- Combinational Logic (text, p4):

  Unity Operators: \( A + 0 = A \) \quad A \cdot 1 = A

  Complement: \( A + \overline{A} = 1 \) \quad A \cdot \overline{A} = 0

  Commutativity: \( A + B = B + A \) \quad A \cdot B = B \cdot A

  Associativity: \( A + (B + C) = (A + B) + C \) \quad A \cdot (BC) = (AB) \cdot C

  Distributed Law: \( A \cdot (B + C) = AB + AC \)
  \( A + BC = (A + B) \cdot (A + C) \)

  Important: \( A + A = A \) \quad A \cdot A = A \quad A \cdot B + A \cdot \overline{B} = A

  Note: \( A \oplus B = AB + \overline{A}\overline{B} \)

  DeMorgan’s: \( (A + B + \ldots)’ = A’ \cdot B’ \cdot \ldots \) \quad (ABC)’ = A’ + B’ + \ldots
Logic Levels

- Active High (H) and Active Low (L)

3.3 V CMOS (Mosfet based) Technology
- 3.3 V
- 2.4 V
- 2 V
- 0.8 V
- 0.5 V
- 0 V

- 3.3 V Logic Families (based on 74LVT04 Hex Inverter)

5 V TTL (BJT based) Technology
- 5 V
- 2.7 V
- 2 V
- 0.8 V
- 0.4 V
- 0 V

- Standard 5 V TTL

### Active High (H)
- $V_{OH} = 1$
- $V_{OL} = 0$

### Active Low (L)
- $V_{OL} = 1$
- $V_{OH} = 0$

- If on the datasheet if the pin shows a bubble or bar, it is Active (L): you have to pull it low to enable that signal.
Types of Digital System

- **Combinational Logic:**
  - Rule: Active High (H) connects to non-bubble, Active Low (L) connects to bubble

  \[ C = A \cdot B \]

  \[ C = A + B \]

  - If active level is not correct, use identity
    \[ X(H) = \overline{X}(L) = X'(L), \ X(L) = \overline{X}(H) = X'(H) \]

  - If gate has wrong bubble, “push” the bubble

  - Examples: done in class
Example: Bubbles, Low, High

- What logic is produced by the following?

```
A(H)
B(L)
A(L)
B(H)
F(H)
```
Example: Bubbles, Active Low, Active High

- Change $A(H)$ to $A'(H)$
- Change $B(L)$ to $B'(H)$, $A(L)$ to $A'(H)$

DeMorgan’s:

$(A + B + \ldots)' = A' \cdot B' \ldots$

$(ABC)' = A' + B' + \ldots$
Types of Digital System

- Full Adder (done in class)

- Truth Table
- Outputs
- Simplification
Karnough Maps

- Simplification can be difficult; use K-Map
  - Full Adder using K-Map
  - More Examples with K-Maps
  - Variable Entered Map (for large number of variables)
Implementation of Logic Gates

- Logic Devices: NAND, NOR
### NAND Gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Vout</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>

- For $\overline{A \cdot B} = \overline{00}$, $V_{out} = V_{DD}$
- For $\overline{A \cdot B} = \overline{01}$, $V_{out} = V_{DD}$
- For $\overline{A \cdot B} = \overline{10}$, $V_{out} = V_{DD}$
- For $\overline{A \cdot B} = \overline{11}$, $V_{out} = 0 \text{V}$
Conversion to NAND and NOR

- Why?
- Because implementation (circuit) of NAND and NOR gates is easier than AND and OR gates

DeMorgan’s Law (1-16D, p.4)

- $C = (AB)' = A' + B'$
- $C = (A + B)' = A'B'$
Conversion to NAND

1) Convert AND to NAND by adding inversion (bubble) at the output
2) Convert OR to NAND by adding inversion (bubble) at the input
3) Two inversions cancel: no further conversion needed
4) Place inverters to correct bubble to non-bubble and vice versa
5) Compliment the variable that drives a bubble
More Logic Devices

- Multiplexer
- Decoder
- Adder
- Comparator
- Arithmetic Logic Unit (ALU)
## Multiplexer

- **2 input**

- **4 input**

### Truth Table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Switch Analogy (4PST)
Decoder

- 2 to 4-bit decoder

![Diagram of 2 to 4-bit decoder with truth table]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Q₀</th>
<th>Q₁</th>
<th>Q₂</th>
<th>Q₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
- 4-bit binary adder
1-bit comparator

Comparator

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Arithmetic Logic Unit (ALU)

- **ALU:**
  1) Addition, Subtraction, Multiplication, Division
  2) Logic Operations: OR, XOR, AND, NAND, NOR...

**A and B:** the inputs to the ALU

**R:** Output or Result

**F:** Code or Instruction from the Control Unit (aka as op-code)

**D:** Output status; it indicates cases such as:
  - carry-in
  - carry-out,
  - overflow,
  - division-by-zero
  - And . . .
Arithmetic Logic Unit (ALU)

- ALU Opcode and Functions
  - ALU Controller Required

<table>
<thead>
<tr>
<th>Class (Opcode)</th>
<th>Operation</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load/store</td>
<td>Addition (memory address)</td>
<td>010</td>
</tr>
<tr>
<td>Branch</td>
<td>Subtraction (comparison)</td>
<td>110</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>Depends on funct field:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100000        add</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>100010        subtract</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>100100        and</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td>100101        or</td>
<td>001</td>
</tr>
<tr>
<td></td>
<td>101010        set on less than</td>
<td>111</td>
</tr>
</tbody>
</table>

ALU Controller
Arithmetic Logic Unit (ALU)

- 1-bit ALU:
  - Logical AND and Logical OR operation
  - Result = a AND b when operation = 0
  - Result = a OR b when operation = 1
  - The operation line is the input of a MUX.
Arithmetic Logic Unit (ALU)

- Adding Full Adder
Arithmetic Logic Unit (ALU)

- 32 bit ALU
  - paralleling the one-bit ALUs