CURIE Academy, Summer 2021
Lab 1: Computer Engineering
Hardware Perspective

Prof. Christopher Batten
School of Electrical and Computer Engineering
Cornell University
Materials Required for Lab 1

- Breadboard-Based Prototyping Platform
- Power Adapter
- 9V Battery
- Integrated Full-Adder Board
- Jumper Wires
- Pre-Cut Wire
- LEDs
- Resistors
Computer Systems Stack

Application

Algorithm

Programming Language

Operating System

Compiler

Instruction Set Architecture

Microarchitecture

Register-Transfer Level

Gate Level

Circuits

Devices

Technology

Smart Light

Flowchart

C++

Particle OS

Particle Development Environment

ARM Machine Instructions

Ripple Carry Adder

NOT, AND, OR, XOR

Inverter

Resistors, LEDs, Transistors
Various Electrical Devices

(a) LEDs
(b) Resistors
(c) Transistor

Figure 2: Various Devices
Figure 3: Symbols for Various Devices
Figure 4: Basic Electrical and Water Circuit
Basic Electrical and Water Circuit

(a) LEDS
(b) Resistors
(c) Transistor

Figure 2: Various Devices
Figure 3: Symbols for Various Devices
Figure 4: Basic Electrical and Water Circuit
Computer Systems Stack

Application
Algorithm
Programming Language
Operating System
Compiler
Instruction Set Architecture
Microarchitecture
Register-Transfer Level
Gate Level
Circuits
Devices
Technology

Smart Light
Flowchart
C++
Particle OS
Particle Development Environment
ARM Machine Instructions
Ripple Carry Adder
NOT, AND, OR, XOR
Inverter
Resistors, LEDs, Transistors

Computer Engineering

CURIE Lab 1
CURIE Lab 2
Simple LED Circuit
Breadboard Prototyping Platform

Digital Input Switches

Digital Output LEDs

Breadboard Power Supply
Connectivity Inside Breadboard
LED Circuit on Breadboard
Inverter Circuit

Figure 8 shows a more interesting circuit called an inverter. This circuit uses a PMOS transistor and an NMOS transistor. When the input is a logic one then the PMOS transistor is open and the NMOS transistor is closed; this essentially causes the output to be “pulled down” to a logic zero. When the input is a logic zero then the PMOS transistor is closed and the NMOS transistor is open; this essentially causes the output to be “pulled up” to a logic one. Figure 9 shows the implementation of the inverter using the prototyping platform provided for you in this lab. Notice how we have...
Figure 8 shows a more interesting circuit called an inverter. This circuit uses a PMOS transistor and an NMOS transistor. When the input is a logic one then the PMOS transistor is open and the NMOS transistor is closed; this essentially causes the output to be “pulled down” to a logic zero. When the input is a logic zero then the PMOS transistor is closed and the NMOS transistor is open; this essentially causes the output to be “pulled up” to a logic one. Figure 9 shows the implementation of the inverter using the prototyping platform provided for you in this lab. Notice how we have...
NOR, AND, OR, XOR Logic Gates

Figure 10: Symbol and Truth Table for NOT, AND, OR, XOR Gates

(a) Integrated Circuit
(b) AND Chip
(c) OR Chip
(d) XOR Chip

Figure 11: Four Logic Gates Implemented as an Integrated Circuit in a Single Chip

Attached the input of the inverter to one of the digital input switches and the output of the inverter to one of the digital output LEDs. Turning on the input will turn off the output; and turning off the input will turn on the output.

3. Gates: NOT, AND, OR, XOR

As computer engineers, we often use abstraction to hide implementation details and provide cleaner higher-level interfaces. Indeed digital signalling itself is an abstraction since we ignore the details of exact voltages and instead focus on logic one and logic zero values. To build more complicated circuits, we will create simple circuits and then abstract them into useful logic gates. For example, we can abstract the inverter discussed in the previous section into the NOT gate shown in Figure 10. If the input to a NOT gate is a logic one then the output is a logic zero; if the input to a NOT gate is a logic zero then the output is a logic one. Abstraction enables us to ignore the details of the specific implementation of a NOT gate using PMOS and NMOS transistors.

Figure 10 also shows three more useful logic gates. We can use a truth table to succinctly capture the functionality of each logic gate. The truth table shows what the output of the logic gate should be for every combination of inputs to the logic gate. For example, the output of an AND gate is only one when both of its inputs are one.
Logic Gates Implemented in Single Chip

(a) Integrated Circuit

(b) AND Chip

(c) OR Chip

(d) XOR Chip

attached the input of the inverter to one of the digital input switches and the output of the inverter to one of the digital output LEDs. Turning on the input will turn off the output; and turning off the input will turn on the output.

3. Gates: NOT, AND, OR, XOR

As computer engineers, we often use abstraction to hide implementation details and provide cleaner higher-level interfaces. Indeed digital signalling itself is an abstraction since we ignore the details of exact voltages and instead focus on logic one and logic zero values. To build more complicated circuits, we will create simple circuits and then abstract them into useful logic gates. For example, we can abstract the inverter discussed in the previous section into the NOT gate shown in Figure 10. If the input to a NOT gate is a logic one then the output is a logic zero; if the input to a NOT gate is a logic zero then the output is a logic one. Abstraction enables us to ignore the details of the specific implementation of a NOT gate using PMOS and NMOS transistors.

Figure 10 also shows three more useful logic gates. We can use a truth table to succinctly capture the functionality of each logic gate. The truth table shows what the output of the logic gate should be for every combination of inputs to the logic gate. For example, the output of an AND gate is only one when both of its inputs are one.
Aside: Binary Arithmetic

dec: 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
bin: 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

Figure 12: Binary and Decimal Representation

Step 1  Step 2  Step 3  Step 4

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>011</td>
<td>011</td>
<td>011</td>
</tr>
<tr>
<td>+110</td>
<td>+110</td>
<td>+110</td>
<td>+110</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>001</td>
<td>1001</td>
</tr>
</tbody>
</table>

Figure 13: Example Using Binary Addition for 3+6
Half-Adder Unit: Add Two 1b Numbers

<table>
<thead>
<tr>
<th>input A</th>
<th>input B</th>
<th>result base 10</th>
<th>result base 2</th>
<th>carry bit</th>
<th>sum bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 + 0</td>
<td>= 0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 + 1</td>
<td>= 1</td>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 + 0</td>
<td>= 1</td>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 + 1</td>
<td>= 2</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

As discussed in the previous section, if we add one plus one the answer is two which cannot be represented with a single bit. We must “overflow” from a one-bit result into a two-bit result. We call this half adder.

Figure 14: Four Possibilities when Adding Two One-Bit Numbers

Figure 15: Truth Tables for Sum Bit and Carry Bit
### Full-Adder: Add Three 1b Numbers

<table>
<thead>
<tr>
<th>input</th>
<th>input</th>
<th>input</th>
<th>result base 10</th>
<th>result base 2</th>
<th>carry bit</th>
<th>sum bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>+ 0</td>
<td>+ 0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>+ 0</td>
<td>+ 1</td>
<td>1</td>
<td>01</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>+ 1</td>
<td>+ 0</td>
<td>1</td>
<td>01</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>+ 1</td>
<td>+ 1</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>+ 0</td>
<td>+ 0</td>
<td>1</td>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>+ 0</td>
<td>+ 1</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>+ 1</td>
<td>+ 0</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>+ 1</td>
<td>+ 1</td>
<td>3</td>
<td>11</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 17: Eight Possibilities when Adding Three One-Bit Numbers
Full-Adder: Add Three 1b Numbers

Figure 17: Eight Possibilities when Adding Three One-Bit Numbers

Figure 18: Full-Adder

We can use a similar approach to build a “full adder” which can add three one-bit numbers to produce a two-bit result. Figure 17 shows the eight possibilities when adding three one-bit numbers. We now wish to implement a one-bit full adder using boolean logic gates. The full adder has three inputs corresponding to three one-bit numbers we wish to add together and two outputs corresponding to the sum and carry bits. As in the previous section, we could now create truth tables for both the sum bit and the carry bit and carefully construct a network of boolean logic gates which will always generate the desired outputs as a function of the inputs; note that we will need to use multiple stages of boolean logic.
Figure 20: Four-Bit Ripple-Carry Adder (FA = full-adder)

Column A produces a carry bit which we must include when calculating the second bit of the sum, and similarly column B produces a carry bit meaning the result overflows into three bits. If we focus on column B, we will realize that we actually need to add three one-bit numbers together (one bit from the first input, one bit from the second input, and the carry bit from column A) and that we will produce two outputs (a sum bit and a carry bit for the next column). A full adder provides the exact functionality needed to calculate each column of a multi-bit addition. Figure 20 illustrates how we can chain a series of full adders to compute a four-bit addition. The carry bit output from a full adder is connected to one of the three inputs of the full adder to the left. Note that the third input of the right-most full adder should be set to zero and that the carry bit output of the left-most full adder allows us to detect overflow (i.e., the result cannot be encoded in just four bits).

This section illustrates modular design, a powerful design concept which is critical for implementing complex systems. We first designed and evaluated a small module (half adder) and then reused this small module to implement a larger and more complex module (full adder). We then chained multiple full adders together into a ripple-carry adder to enable multi-bit addition.
Let’s wire up a simple LED and inverter circuit