Previous Courses

- Digital Devices (EE 3713/EE3714)
  - Boolean Algebra
  - Simple Combinational, Sequential Networks (< 50 gates)
  - TTL, PLD implementation technologies
  - No CAD tools
- Microprocessors
  - Instruction sets, basic architecture
  - Assembly language programming
  - Microprocessor based solutions for Digital control

---

EE 4743/6743 : Digital Systems

- Complex Combinational and Sequential networks (up to thousands of gates)
  - Emphasis on combined datapath+Finite state machine designs for real time applications
- Modern CAD tool usage (schematic entry, simulation, technology mapping, timing analysis, synthesis)
- Logic Synthesis via VHDL
- Modern implementation technologies such Field Programmable Gate Arrays (FPGAs)

---

Course Philosophy

- The textbook in this course is more of a reference
  - Buy it, READ IT. Will help, especially with logic synthesis
- Material in class based on instructor notes
  - Many notes online, see both:
    - www.ece.msstate.edu/~mitch/class/ee4743
    - www.ece.msstate.edu/~reese/EE4743
  - Course notes from previous semesters still applicable, but will be supplemented
- You will need to stay caught up on lecture material.
  Falling behind is difficult to recover from.
Course Software

• We will use Altera Maxplus (runs on both PCs/Unix)
  – Runs great on any Pentium class machine that can run Win 98 (32 Mb RAM, 160 Mhz or better).
• Three student PC versions available (V 7.21, V 9.23, V10.1)
  – V9.23 is contained on the CDROM in the back of the textbook
  – Class WWW page has a link where you can download version V10.1 from the Altera WWW page.
  – Do not use V 7.21 – Use at least V9.23. Version 10.1 only needed if you are running Win2K or higher and want to download designs into the UP1 hardware board.
• Software also available on ECE Unix workstations also
  – See Class WWW page for instructions on using Maxplus on Unix workstations
  – Files are compatible between Unix workstation version and student versions.

Course Software (cont)

Software is the same as used by practicing engineers in industry.

• HIGHLY RECOMMEND that you install it on your home PC
  – No competition for seats in lab
  – Eat popcorn, watch DVD, listen to music while you do homework
  – If you don’t have a PC, get one. It is an indispensable tool for an engineer.

Lecture, Labs

• Lecture is MW. Will meet on Fridays also for ‘awhile’ to get a fast start on material
  – Will get these ‘extra’ lectures back near end of semester when concentrating on project
• Only need to attend your lab session (Simrall workstation lab, 1st floor) to hear TA explanation of lab assignment
  – Can complete assignment on Unix machines or home PC. Upload files from home for TA checkoff.
• Lab assignments due ONE WEEK from your assigned lab time unless otherwise noted.
Combinational Logic Review

Digital Devices was a LONG, LONG time ago in a galaxy FAR, FAR, AWAY for many of you.

We don’t expect you to remember everything you learned in Digital Devices, but you need to remember > 0%.

We will review some to help you remember. You also need to go back and look at old notes. After a couple of days of review, we will expect you to be up to speed, and then we will ZOOM along.

Ask QUESTIONS during CLASS to SLOW things down.

---

### 7404 Logic Gate

**Mixed Logic**

<table>
<thead>
<tr>
<th>A</th>
<th>Y</th>
<th>(A) (L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>1</td>
</tr>
</tbody>
</table>

Buffer that converts high true input to low true output

**Fixed Logic**

<table>
<thead>
<tr>
<th>A (L)</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Buffer that converts low true input to high true output

---

### Gate Summaries

- **7408**: AND
- **7432**: OR
- **7400**: NAND
- **7402**: NOR
Majority Gate (and-or) form

\[ F = AB + AC + BC \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Majority Gate (nand-nand) form

\[ F = ((AB)'(AC)'(BC)')' \]

\[ = AB + AC + BC \]

by De'Morgan's theorem

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Majority Gate (mixed logic) form

\[ F = AB + AC + BC \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Boolean Minimization

Reduce a boolean equation to fewer terms - hopefully, this will result in using less gates to implement the boolean equation.

Pencil-Paper: Algebraic techniques, K-maps
Automated: Many powerful algorithms exist

\[
\begin{array}{ccc|c}
A & B & C & F \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[F = A'BC + AB'C + ABC + ABC' + ABC\]

Find Boolean adjacencies to minimize equation; eliminate redundant term

K- maps

Graphical Aid for minimization - used to visualize boolean adjacencies

\[
\begin{array}{ccc}
A & B & C \\
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
BC & 00 & 01 & 11 & 10 \\
0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 \\
\end{array}
\]

\[F = BC + AC + AB\]

Technology Mapping

Technology mapping is takes a boolean equation and maps it onto a given technology. The technology can affect what constraints are used when doing minimization for the function.

- Discrete gates
- Programmable Logic
- Custom Integrated Circuits
Logic Synthesis

Logic Synthesis: convert a description of a digital system in a Hardware Description Language (HDL) to an implementation technology.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity majority is
port ( A, B, C : in std_logic;
      Y: out std_logic);
end majority;

ARCHITECTURE a of majority is
begin
  Y< = ( A a n dB ) o r ( Aa n dC ) o r ( Ba n dC ) ;
end a;
```

Combination Building Blocks

1 bit Multiplexer (MUX)

- If \( S = 0 \), then \( Y = I_0 \)
- If \( S = 1 \), then \( Y = I_1 \)

\[ Y = I_0 S' + I_1 S \]

Maxes often use to select groups of bits arranged in busses.
Memories

Memories are classified as K x N devices, K is the # of locations, N is the number bits per location (16 x 2 would be 16 locations, each storing 2 bits)

K locations require \( \log_2(K) \) address lines for selecting a location (i.e. a 16 location memory needs 4 address lines)

A memory that is \( K \times N \), can be used to implement \( N \) boolean equations, which use \( \log_2(K) \) variables (the \( N \) boolean equations must use the same variables).

One address line is used for each boolean variable, each bit of the output implements a different boolean equation.

The memory functions as a **LookUp Table (LUT)**.

Memory example

\[ F(A,B,C) = A \text{xor} B \text{xor} C \quad G = AB + AC + BC \]

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
<th>( F )</th>
<th>( G )</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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Recall that Exclusive OR (xor) is

\[ Y = A \oplus B = A \text{xor} B \]

8 x 2 Memory

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
<th>( D0 )</th>
<th>( D1 )</th>
<th>( F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

LookUp Table (LUT)

\( A[2:0] \) is 3 bit address bus, \( D[1:0] \) is 2 bit output bus.

Location 0 has “00”, Location 1 has “10”, etc.
Binary Adder

\[ F(A, B, C) = A \text{ xor } B \text{ xor } C \quad G = AB + AC + BC \]

These equations look familiar. Recall what a Full Adder is:

\[ \text{Sum} = A \text{ xor } B \text{ xor } Cin \]

\[ \text{Cout} = AB + Cin A + Cin B = AB + Cin (A + B) \]

4 Bit Ripple Carry Adder


Propagation Delay
Making a Design Run Fast

- Speed usually much more important than saving gates.
- The speed of a gate directly affects the maximum clock speed of digital system
- Gate speed is TECHNOLOGY dependent
  - 0.35u CMOS process has faster gates than 0.8u CMOS process
- Implementation choice will affect Design speed
  - A Custom integrated circuit will be faster than an FPGA implementation.
- Design approaches will affect clock speed of system
  - Smart designers can make a big difference

Summary

- Need to review your Digital Devices notes
  - Basic Gates, Boolean algebra (algebraic minimization, up to four variable K-maps), Combinational building blocks (muxes, decoders, memories, adders)
- We will discuss Hardware Description Languages
  - VHDL is the language used in the class
- We will discuss modern implementation technologies, primarily Field Programmable Gate Arrays (FPGAs)
- We will discuss design strategies for making designs run faster, not necessarily take less gates.