Engr433: Digital Design

Review of Combinational Circuits

Dr. Curt Nelson

Combinational Review Topics

- Number Systems
  - Bases
    - Decimal
    - Binary
    - Hexadecimal
  - Conversions
    - Decimal <-> Binary
    - Decimal <-> Hexadecimal
    - Binary <-> Hexadecimal
- Basic gates
  - Inverters, And, Nand, Or, Nor, Xor, Xnor
Combinational Review Topics - Continued

- Boolean Algebra
  - Basic Laws
  - De'Morgans Theorems
- Truth Tables
  - Minterms, Sum of Products
  - Maxterms, Product of Sums
- Karnaugh Maps
  - Sum of Products - Minterms
  - Product of Sums - Maxterms
  - Minimization
    - Don't Cares
    - Entered Variable Mapping (EVM)

Combinational Review Topics - Continued

- Medium and Large Scale Integration (MSI and LSI)
  - Arithmetic: Adders / Subtractors
  - Data Format Converters: Encoders / Decoders
  - Data Selectors: Multiplexers/Demultiplexers
  - Other functions
Digital Abstraction

- Why do we use abstraction in the digital arena?
  - To manage complexity.

(b) Graphic symbol

c) Logic circuit

Digital Abstraction

...
Truth Tables

- All combinations of inputs on the left;
- Outputs on the right;
- 2-input AND and OR functions shown below.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_1 \cdot x_2$</th>
<th>$x_1 + x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

AND OR

Truth Table Proof of DeMorgan’s Theorem

$\overline{x \cdot y} = \overline{x} + \overline{y}$  
DeMorgan’s Theorem

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$x \cdot y$</th>
<th>$\overline{x \cdot y}$</th>
<th>$\overline{x}$</th>
<th>$\overline{y}$</th>
<th>$\overline{x} + \overline{y}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<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>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

I.HS RHS
DeMorgan’s Theorems in Terms of Logic Gates

(a) $\overline{x_1 x_2} = \overline{x_1} + \overline{x_2}$

(b) $\overline{x_1 + x_2} = \overline{x_1} \overline{x_2}$

- Function vs. Gate

SOP Implementation Using NAND Gates
POS Implementation Using NOR Gates

Timing Diagram
Logic Synthesis

- *Logic synthesis*, or *logic optimization*, is the process of translating a truth table, schematic, or VHDL code into a network of logic gates.
- Example:
  - Minterm form;
  - Maxterm form;
  - Minimum form.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</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>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

SOP Implementation – NAND Gates

(a) Sum-of-products realization
POS Implementation – NOR Gates

(b) Product-of-sums realization

Minimal Implementation
4-Variable Karnaugh Map (Kmap)

4-Variable Kmap Example

Four-variable function \( f = \Sigma m(2, 3, 5, 6, 7, 10, 11, 13, 14) \)
4-Variable Function with Don’t Cares

\[ f = \Sigma m(2, 4, 5, 6, 10) + d(12, 13, 14, 15) \]

Entered Variable (EV) Mapping

- Sometimes called Map Entered Variables (MEV’s);
- Allows many variables to be presented using a reduced size K-map;
- Occurs quite frequently in digital systems, especially state machines;
- May require K-map compression and expansion;
- References (entered-variable mapping or map-entered variables):
  - Tinder, Engineering Digital Design, Second Edition (Library);
  - Other digital logic textbooks;
  - Web – YouTube, etc.
Entered Variable (EV) Mapping Example

\[ f = \sum m(2, 5, 6, 7) \]

<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>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>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>

\[ f = a'bc' + ab'c + abc' + abc \]
\[ = bc' + ac \]

Entered Variable Truth Table Compression

\[
\begin{array}{c|c|c|c|c|}
\hline
a & b & c & f \\
\hline
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|}
\hline
a & b & f \\
\hline
0 & 0 & 0 \\
0 & 1 & c' \\
1 & 0 & c \\
1 & 1 & 1 \\
\hline
\end{array}
\]
Entered Variable Map Compression

\[ f = a'b'c' + ab'c + abc' + abc = bc' + ac \]

Three Variable Compression Example

\[ 1 = c + c' \]  
\[ 0 = cc' \]
The function \( f = \Sigma m(0, 2, 4, 5, 10, 11, 13, 15) \)

**Four Variable Compression Example**

\[
\begin{array}{cccc}
abcd & 00 & 01 & 11 & 10 \\
00 & 1 & 1 &   &   \\
01 & 1 & 1 &   &   \\
11 &   & 1 & 1 &   \\
10 & 1 &   & 1 &   \\
\end{array}
\]

The function \( f = \Sigma m(0, 2, 4, 5, 10, 11, 13, 15) \)

\[
\begin{array}{cccc}
\text{a} \text{b} & 00 & 01 & 11 & 10 \\
0 & d' & 1 & d & 0 \\
1 & d' & 0 & d & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{a} \text{b} & 0 & 1 \\
0 & c'd' + cd' = d' \\
1 & c'0 + c1 = c \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{a} \text{b} & 0 & 1 \\
0 & c'1 + c0 = c' \\
1 & c'd' + cd = d \\
\end{array}
\]

**Other Examples**

- Expansion;
- Compression and expansion using don’t cares.
MSI Circuits – Half Adder

(a) The four possible cases

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>c</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(b) Truth table

(c) Circuit

(d) Graphical symbol

MSI Circuits – Full Adder

(a) Truth table

<table>
<thead>
<tr>
<th>c_i</th>
<th>x_i</th>
<th>y_i</th>
<th>s_i</th>
<th>c_{i+1}</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>1</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(b) Karnaugh maps

(c) Circuit
N-bit Ripple Carry Adder

- Major issue – propagation delay.

Faster Adders – Carry Lookahead
Adder / Subtractor Circuit

Multiplexers

- Devices that select one of many inputs to be routed to one output based on the binary value of select lines
  - Enable – used to enable or disable the complete function;
  - Select – used to select which one of the inputs gets routed to the output.
Synthesis of Logic Functions Using Mux’s

• Mux’s can be used to synthesize logic functions as follows:
  – Create truth table;
  – Compress as necessary;
  – Implement.
• In general, an $N$ variable function can be implemented with one $N-1$ multiplexer and at most, one inverter.

Example: 3-Input Majority Function

(a) Truth table

(b) Circuit
De-Multiplexers

- A de-multiplexer is a circuit which places the value of a single data input onto one of a number of outputs.

![De-Multiplexer Diagram](image)

An \(n\)-to-\(2^n\) Decoder

- A decoder is a device that activates one output based on the binary value of the inputs;
- A decoder is a minterm generator;
- Enable – used to enable or disable the complete decoder function.

![Decoder Diagram](image)
A 2-to-4 Decoder

(a) Truth table

(b) Graphic symbol

(c) Logic circuit

A 2\(^n\)-to-\(n\) Binary Encoder

- An encoder is a device that outputs a binary code representing which one of many inputs is active.
- Priority encoder – assigns priority to certain inputs
  - Used in embedded computer systems to service interrupts.
**BCD to 7-segment Code Converter**

(a) Code converter

(b) 7-segment display

(c) Truth table

<table>
<thead>
<tr>
<th>$w_3$</th>
<th>$w_2$</th>
<th>$w_1$</th>
<th>$w_0$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$e$</th>
<th>$f$</th>
<th>$g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</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>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</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>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**A Four-bit Magnitude Comparator**
Arithmetic Logic Units (74HC381)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Inputs $s_2$ $s_1$ $s_0$</th>
<th>Outputs $F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear</td>
<td>0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>$B\leftarrow A$</td>
<td>0 0 1</td>
<td>$B\leftarrow A$</td>
</tr>
<tr>
<td>$A\Rightarrow B$</td>
<td>0 1 0</td>
<td>$A\Rightarrow B$</td>
</tr>
<tr>
<td>ADD</td>
<td>0 1 1</td>
<td>$A + B$</td>
</tr>
<tr>
<td>XOR</td>
<td>1 0 0</td>
<td>$A \oplus B$</td>
</tr>
<tr>
<td>OR</td>
<td>1 0 1</td>
<td>$A \lor B$</td>
</tr>
<tr>
<td>AND</td>
<td>1 1 0</td>
<td>$A \land B$</td>
</tr>
<tr>
<td>Reset</td>
<td>1 1 1</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

Simulation

- A *functional simulator* is used to determine if a designed circuit operates correctly from a *logic* perspective;
- Circuit *verification*:
  - User provides input values to the circuit;
  - Simulator determines the circuit response;
  - User checks responses against desired outputs;
- A *timing simulator* is used to check correctness by incorporating the *electrical* characteristics of a logic design in addition to the *logical* performance. This simulation requires:
  - Technology mapping;
  - Layout synthesis.
How are Logic Circuits Implemented?

- Logic implementations you have seen so far are good for small circuits:
  - 74HC00 (series in general), a few to 10’s of gates;
  - PLA and PAL, a few hundred gate equivalents;
- There is a need for larger programmable devices.
- Enter Field Programmable Gate Array’s - FPGA’s
  - Do not contain AND and OR planes, rather logic blocks;
  - 10,000 up to 1,000,000 gate equivalents and getting larger every year.

Structure of an FPGA
Combinational Logic Blocks - Lookup Tables

(a) Circuit for a two-input LUT

(b) \[ f_1 = \overline{s_1} s_2 + s_1 s_2 \]

(c) Storage cell contents in the LUT

Standard-Cell Integrated Circuits

- Gates are prebuilt and stored in a library;
- The gates needed for a design are selected and placed, and wires are routed between them;
- Standard-cell chips are one form of application specific integrated circuits (ASIC’s);
- CAD tools exist to place and route gates once a design has been synthesized.
Custom Chips

- Created from scratch;
- Designer selects number, placement, size, and connections for each and every transistor;
- These are the densest and highest speed devices;
- Requires a substantial and costly design effort;
- Used only when speed, area, power, or intellectual rights must be optimized:
  - Like processors, memories, DSP’s, etc.