**ECE 545—Digital System Design with VHDL**

**Lecture 1A**

**Digital Logic Refresher**

Part A – Combinational Logic Building Blocks

---

**Lecture Roadmap – Combinational Logic**

- Basic Logic Review
  - Basic Gates
  - De Morgan’s Law
- Combinational Logic Building Blocks
  - Multiplexers
  - Decoders, Demultiplexers
  - Encoders, Priority Encoders
  - Arithmetic circuits
  - ROM: Implementing combinational logic using ROM.
  - Tri-state buffers.

---

**Textbook References**

- Combinational Logic Review
  - Stephen Brown and Zvonko Vranesic,
    *Fundamentals of Digital Logic with VHDL Design, 2nd or 3rd Edition*
    - Chapter 2 Introduction to Logic Circuits (2.1–2.8 only)
    - Chapter 6 Combinational-Circuit Building Blocks (6.1–6.5 only)
  - OR your undergraduate digital logic textbook (chapters on combinational logic)

---

**Basic Logic Review**

*some slides modified from:
S. Dandamudi, “Fundamentals of Computer Organization and Design”*

---

**Rules**

- If you believe that you know a correct answer, please raise your hand
- I will select one or more students
  (independently whether an answer given by the first student is correct or incorrect)
- Please, identify yourself by first name and give an answer
- **Correct answer = 1 bonus point**

---

**Basic Logic Gates (2-input versions)**
Basic Logic Gates Generalized

- Simple logic gates
  - AND \( \rightarrow \) 0 if one or more inputs is 0
  - OR \( \rightarrow \) 1 if one or more inputs is 1
  - NAND = AND + NOT
    - 1 if one or more inputs is 0
  - NOR = OR + NOT
    - 0 if one or more input is 1
  - XOR \( \rightarrow \) 1 if an odd number of inputs is 1
  - XNOR \( \rightarrow \) 1 if an even number of inputs is 1
  - NAND and NOR gates require fewer transistors than AND and OR in standard CMOS
  - Functionality can be expressed by a truth table
    - A truth table lists output for each possible input combination

Number of Functions

- Number of functions
  - With \( N \) logical variables, we can define \( 2^N \) functions
  - Some of them are useful
    - AND, NAND, NOR, XOR, ...
  - Some are not useful:
    - Output is always 1
    - Output is always 0
  - “Number of functions” definition is useful in proving completeness property

Complete Set of Gates

- Complete sets
  - A set of gates is complete
    - if we can implement any logic function using only the type of gates in the set
  - Some example complete sets
    - \{AND, OR, NOT\} — Not a minimal complete set
    - \{AND, NOT\}
    - \{OR, NOT\}
    - \{NAND\}
    - \{NOR\}
  - Minimal complete set
    - A complete set with no redundant elements.

NAND as a Complete Set

- Proving NAND gate is universal

Logic Functions

- Logic functions can be expressed in several ways:
  - Truth table
  - Logical expressions
  - Graphical schematic form
  - HDL code
- Example:
  - Majority function
    - Output is one whenever majority of inputs is 1
    - We use 3-input majority function

Alternative Representations of Logic Function

<table>
<thead>
<tr>
<th>Truth table</th>
<th>Logical expression form</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Graphical schematic form

HDL code: \( F = (A \land B) \lor (B \land C) \lor (A \land C) \)
### Boolean Algebra

<table>
<thead>
<tr>
<th>Name</th>
<th>AND version</th>
<th>OR version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td>$x \cdot 1 = x$</td>
<td>$x + 0 = x$</td>
</tr>
<tr>
<td>Complement</td>
<td>$x \cdot x' = 0$</td>
<td>$x + x' = 1$</td>
</tr>
<tr>
<td>Commutative</td>
<td>$x \cdot y = y \cdot x$</td>
<td>$x + y = y + x$</td>
</tr>
<tr>
<td>Distribution</td>
<td>$x \cdot (y+z) = x \cdot y + x \cdot z$</td>
<td>$x + (y \cdot z) = (x + y) \cdot (x + z)$</td>
</tr>
<tr>
<td>Idempotent</td>
<td>$x \cdot x = x$</td>
<td>$x + x = x$</td>
</tr>
<tr>
<td>Null</td>
<td>$x \cdot 0 = 0$</td>
<td>$x + 1 = 1$</td>
</tr>
</tbody>
</table>

### Boolean Algebra (cont’d)

- **Boolean identities (cont’d)**

<table>
<thead>
<tr>
<th>Name</th>
<th>AND version</th>
<th>OR version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Involution</td>
<td>$x = (x')'$</td>
<td>---</td>
</tr>
<tr>
<td>Absorption</td>
<td>$x \cdot (x+y) = x$</td>
<td>$x + (x \cdot y) = x$</td>
</tr>
<tr>
<td>Associative</td>
<td>$x \cdot (y \cdot z) = (x \cdot y) \cdot z$</td>
<td>$x + (y + z) = (x + y) + z$</td>
</tr>
</tbody>
</table>

*de Morgan* $x \cdot y)' = x' + y'$ $x + y)' = x' \cdot y'$

(de Morgan’s law in particular is very useful)

### Alternative symbols for NAND and NOR

- NAND
- NOR

### Deriving Equivalent Expressions

- Using NAND gates
  - Get an equivalent expression
    $\overline{A \cdot B + C \cdot D} = (\overline{A \cdot B} + \overline{C \cdot D})'$
  - Using de Morgan’s law
    $A \cdot B + C \cdot D = (\overline{\overline{A} \cdot \overline{B}} \cdot \overline{\overline{C} \cdot \overline{D}})'$
  - Can be generalized
    - Example: Majority function
      $A \cdot B \cdot C + A \cdot C = ((A \cdot B)' \cdot (B \cdot C)' \cdot (A \cdot C)')'$

### Majority Function Using Other Gates

- Majority function

### Combinational Logic Building Blocks

*Some slides modified from:*
- S. Dandamudi, “Fundamentals of Computer Organization and Design”
**Multiplexers**

- multiplexer
  - n binary inputs (binary input = 1-bit input)
  - \( \log_2 n \) binary selection inputs
  - 1 binary output
  - Function: one of n inputs is placed onto output
  - Called n-to-1 multiplexer

**2-to-1 Multiplexer**

- When drawing schematics, can draw multi-bit multiplexers
- Example: 8-bit 4-to-1 multiplexer
  - 4 inputs (each 8 bits)
  - 1 output (8 bits)
  - 2 selection bits
  - Can also have multi-bit 2-to-1 muxes, 16-to-1 muxes, etc.

**Decoders**

- Decoder
  - n binary inputs
  - \( 2^n \) binary outputs
  - Function: decode encoded information
    - If enable=1, one output is asserted high, the other outputs are asserted low
    - If enable=0, all outputs asserted low
  - Often, enable pin is not needed (i.e. the decoder is always enabled)
  - Called n-to-2^n decoder
  - Can consider n binary inputs as a single n-bit input
  - Can consider 2^n binary outputs as a single 2^n-bit output
  - Decoders are often used for RAM/ROM addressing
Problem 8
Show how to implement a decoder that recognizes the following 4 ranges of a 16-bit address $A$, and generates the corresponding enable signals $e_0, e_1, e_2, e_3$:

For $A$ in: 
- $C000-\text{CFFF}$: $e_0$
- $D000-\text{DFFF}$: $e_1$
- $E000-\text{EFFF}$: $e_2$
- $F000-\text{FFFF}$: $e_3$

Demultiplexers

- Demultiplexer
  - 1 binary input
  - n binary outputs
  - $\log_2 n$ binary selection inputs
  - Function: places input onto one of n outputs, with the remaining outputs asserted low
  - Called 1-to-$n$ demultiplexer
- Closely related to decoder
  - Can build 1-to-$n$ demultiplexer from $\log_2 n$-to-$n$ decoder by using the decoder's enable signal as the demultiplexer's input signal, and using decoder's input signals as the demultiplexer's selection input signals.

Encoders

- Encoder
  - $2^n$ binary inputs
  - n binary outputs
  - Function: encodes information into an n-bit code
  - Called 2-to-n encoder
    - Can consider $2^n$ binary inputs as a single $2^n$-bit input
    - Can consider n binary output as a single n-bit output
  - Encoders only work when exactly one binary input is equal to 1
### Priority Encoders

- Priority Encoder
  - \(2^n\) binary inputs
  - \(n\) binary outputs
  - 1 binary "valid" output
  - Function: encodes information into an \(n\)-bit code based on priority of inputs
  - Called \(2^n\)-to-\(n\) priority encoder
  - Priority encoder allows for multiple inputs to have a value of '1', as it encodes the input with the highest priority (MSB = highest priority, LSB = lowest priority)
  - "Valid" output indicates when priority encoder output is valid
  - Priority encoder is more common than an encoder

### 4-to-2 MSB Priority Encoder

<table>
<thead>
<tr>
<th>(W_0)</th>
<th>(W_1)</th>
<th>(W_2)</th>
<th>(W_3)</th>
<th>(y_0)</th>
<th>(y_1)</th>
<th>(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### 4x4-bit Unsigned Multiplier

\[
\begin{array}{c}
\text{a} \\
\times \\
\text{b}
\end{array}
\Rightarrow
\begin{array}{c}
\text{c} \\
\Rightarrow \\
\text{u}
\end{array}
\]

### 4x4-bit Signed Multiplier

\[
\begin{array}{c}
\text{a} \\
\times \\
\text{b}
\end{array}
\Rightarrow
\begin{array}{c}
\text{c} \\
\Rightarrow \\
\text{s}
\end{array}
\]

### Unsigned vs. Signed Multiplication

<table>
<thead>
<tr>
<th></th>
<th>Unsigned</th>
<th>Signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1111 \times 1111)</td>
<td>15 \times 15</td>
<td>(-1) \times (-1)</td>
</tr>
<tr>
<td>(11100001)</td>
<td>225</td>
<td>(00000001)</td>
</tr>
</tbody>
</table>

### Logical Shift Right

\[
\begin{array}{c}
\text{A(3)} \\
\leftarrow \\
\text{A(2)} \\
\leftarrow \\
\text{A(1)} \\
\leftarrow \\
\text{A(0)} \\
\end{array}
\Rightarrow
\begin{array}{c}
\text{C} \\
\leftarrow \\
\text{L} \\
\leftarrow \\
\text{A} \\
\leftarrow \\
\end{array}
\]
Arithmetic Shift Right

\[
\begin{array}{c}
  \text{A} \ll \text{B} \\
  \text{C} \\
\end{array}
\]

Fixed Rotation

\[
\begin{array}{c}
  \text{A} \\
  \text{C} \\
\end{array}
\]

8-bit Variable Rotator Left

\[
\begin{array}{c}
  \text{B} \\
  \text{A} \ll \text{B} \\
  \text{C} \\
\end{array}
\]

Read Only Memory (ROM)

\[
\begin{array}{c}
  \text{ADDR} \\
  \text{DOUT} \\
\end{array}
\]

Implementing Arbitrary Combinational Logic Using ROM

\[
\begin{array}{c}
  \text{ADDR} \\
  \text{DOUT} \\
\end{array}
\]

Tri-state Buffer

(a) A tri-state buffer

\[
\begin{array}{c}
  x \\
  \text{f} \\
\end{array}
\]

(b) Equivalent circuit

\[
\begin{array}{c}
  x \\
  \text{f} \\
\end{array}
\]

(c) Truth table

\[
\begin{array}{c|c|c|c|c|c|c|c}
  x & f \\
  0 & 0 & Z & Z \\
  1 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{array}
\]
Four types of Tri-state Buffers

(a)  

(b)  

(c)  

(d)