Lecture Roadmap – Combinational Logic

- Basic Logic Review
  - Basic Gates
  - DeMorgan’s Law
- Combinational Logic Blocks
  - Multiplexers
  - Decoders, Demultiplexers
  - Encoders, Priority Encoders
  - Half Adders, Full Adders
- Multi-Bit Combinational Logic Blocks
  - Multi-bit multiplexers
  - Multi-bit adders
  - Comparators

Lecture Roadmap – Sequential Logic

- Sequential Logic Building Blocks
  - Latches, Flip-Flops
- Sequential Logic Circuits
  - Registers, Shift Registers, Counters
  - Memory (RAM, ROM)

Textbook References

- Combinational Logic Review
  - 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)
- Sequential Logic Review
  - Chapter 7 Flip-flops, Registers, Counters, and a Simple Processors (7.3-7.4, 7.8-7.11 only)
  - OR your undergraduate digital logic textbook (chapters on sequential logic)

Basic Concepts

- Simple logic gates
  - \( \text{AND} \rightarrow 0 \) if one or more inputs is 0
  - \( \text{OR} \rightarrow 1 \) if one or more inputs is 1
  - \( \text{NOT} \)
  - \( \text{NAND} = \text{AND} + \text{NOT} \)
    - 1 if one or more inputs is 0
  - \( \text{NOR} = \text{OR} + \text{NOT} \)
    - 0 if one or more input is 1
  - XOR implements exclusive-OR function
  - 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
Basic Logic Gates

<table>
<thead>
<tr>
<th>Logic symbol</th>
<th>Truth table</th>
<th>Logic symbol</th>
<th>Truth table</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A B F</td>
<td>A</td>
<td>A B F</td>
</tr>
<tr>
<td></td>
<td>0 0 0</td>
<td>0</td>
<td>0 0 1</td>
</tr>
<tr>
<td></td>
<td>0 1 0</td>
<td>1</td>
<td>1 0 0</td>
</tr>
<tr>
<td></td>
<td>0 0 1</td>
<td>1</td>
<td>0 0 1</td>
</tr>
<tr>
<td></td>
<td>0 0 1</td>
<td>1</td>
<td>1 0 0</td>
</tr>
<tr>
<td></td>
<td>1 1 1</td>
<td>0</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

AND gate

OR gate

NOR gate

NOT gate

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 logical 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**

<table>
<thead>
<tr>
<th>Truth table</th>
<th>Logical expression form</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C F</td>
<td>$F = A B + B C + A C$</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>0 0 1 0</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>0 1 1 1</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td></td>
</tr>
<tr>
<td>1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

Logic Functions

- Logical functions can be expressed in several ways:
  - Truth table
  - Logical expressions
  - Graphical form
  - HDL code
- Example:
  - Majority function
    - Output is one whenever majority of inputs is 1
    - We use 3-input majority function
<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>$xx = x$</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>
<tr>
<td>de Morgan</td>
<td>$(x \cdot y)' = x' + y'$</td>
<td>$(x + y)' = x' \cdot y'$ (de Morgan’s law in particular is very useful)</td>
</tr>
</tbody>
</table>

### Majority Function Using Other Gates

- Using NAND gates
  - Get an equivalent expression
    $$A \cdot B + C \cdot D = (A \cdot B + C \cdot D)''$$
  - Using de Morgan’s law
    $$A \cdot B + C \cdot D = \left( (A \cdot B)' \cdot (C \cdot D)'ight)'$$
- 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 (cont’d)

- Majority function

### Combinational Logic Building Blocks

- Multiplexers
  - $n$ binary inputs (binary input = 1-bit input)
  - log$_n$ binary selection inputs
  - 1 binary output
  - Function: one of $n$ inputs is placed onto output
  - Called $n$-to-$1$ multiplexer
### 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

### Demultiplexers

- **Demultiplexer**
  - 1 binary input
  - n binary outputs
  - 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 $2^n$-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

**Single-Bit Adders**

- Half-adder
  - Adds two binary (i.e. 1-bit) inputs $A$ and $B$
  - Produces a sum and carryout
  - Problem: Cannot use it alone to build larger adders
- Full-adder
  - Adds three binary (i.e. 1-bit) inputs $A$, $B$, and carryin
  - Like half-adder, produces a sum and carryout
  - Allows building $M$-bit adders ($M > 1$)
    - Simple technique
      - Connect $C_{in}$ of one adder to $C_{in}$ of the next
    - These are called ripple-carry adders
    - Shown in next section

**4-to-2 Encoder**

- Truth table
  - $w_3$, $w_2$, $w_1$, $w_0$
  - $y_0$, $y_1$

- Circuit

**4-to-2 Priority Encoder**

- Truth table
  - $w_3$, $w_2$, $w_1$, $w_0$
  - $y_0$, $y_1$, $z$

- Circuit

**Half-Adder**

- Circuit
  - $x$, $y$, $c$, $s$
  - $x + y' (c + s)$

- Circuit
  - $x$, $y$, $c$, $s$
  - $x$, $y'$, $s$
Full-Adder

\[ x + y + c_{in} = s + 2c_{out} \]

\[
\begin{array}{cccc|c}
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

Multi-bit 4-to-1 Multiplexer

- When drawing schematics, can draw multi-bit multiplexers
- Example: 4-to-1 (8 bit) 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.

4-to-1 (8-bit) Multiplexer

A 4-to-1 (8-bit) multiplexer is composed of eight 4-to-1 (1-bit) multiplexers

16-bit Unsigned Adder

A 16-bit ripple-carry adder is composed of 16 (1-bit) full adders

- Inputs: 16-bit A, 16-bit B, 1-bit carryin (set to zero in the figure below)
- Outputs: 16-bit sum R, 1-bit overflow
Other multi-bit adder structures can be studied in ECE 645—Computer Arithmetic

Multi-Bit Combinational Logic Building Blocks

Multi-Bit Ripple-Carry Adder

Called a ripple-carry adder because carry ripples from one full-adder to the next.
Critical path is 16 full-adders.
Comparator

- Used to compare two M-bit numbers and produce a flag (M > 1)
- Inputs: M-bit input A, M-bit input B
- Output: 1-bit output flag
  - 1 indicates condition is met
  - 0 indicates condition is not met
- Can compare: >, ≥, <, ≤, =, etc.

\[
\begin{array}{ccc}
A & B & A > B? \\
M & M & 1 \text{ if } A > B \\
0 \text{ if } A \leq B
\end{array}
\]

Example: 4-bit comparator (A = B)

\[
\begin{array}{c}
A \\
B \\
B_3 \\
A_3 \\
B_2 \\
A_2 \\
B_1 \\
A_1 \\
B_0 \\
A_0
\end{array}
\]

\[
\begin{array}{c}
A = B? \\
A \\
B
\end{array}
\]

Tri-state Buffer

- (a) A tri-state buffer
- (b) Equivalent circuit
- (c) Truth table

\|
x | f | e
|---|---|---
| 0 | 0 | 0
| 0 | 1 | Z
| 1 | 0 | Z
| 1 | 1 | 1

Four types of Tri-state Buffers

- (a)
- (b)
- (c)
- (d)

Introduction to Sequential Logic

- Output depends on current as well as past inputs
- Depends on the history
- Have “memory” property
- Sequential circuit consists of
  - Combinational circuit
  - Feedback circuit
  - Past input is encoded into a set of state variables
  - Uses feedback (to feed the state variables)
    - Simple feedback
    - Uses flip flops

Sequential Logic Building Blocks
Introduction (cont’d)

Main components of a typical synchronous sequential circuit (synchronous = uses a clock to keep circuits in lock step)

- **COMBINATIONAL LOGIC**
- **INPUT**
- **PRESENT STATE S(t)**
- **CLOCK**
- **STATE-HOLDING ELEMENTS (i.e. FLIP-FLOPS)**
- **OUTPUT**
- **NEXT STATE S(t+1)**

State-Holding Memory Elements

- **Latch versus Flip Flop**
  - Latches are level-sensitive: whenever clock is high, latch is transparent
  - Flip-flops are edge-sensitive: data passes through (i.e. data is sampled) only on a rising (or falling) edge of the clock
  - Latches cheaper to implement than flip-flops
  - Flip-flops are easier to design with than latches
  - In this course, primarily use D flip-flops

D Latch vs. D Flip-Flop

- **Latch transparent when clock is high**
- **"Samples" D on rising edge of clock**

D Flip-Flop with Asynchronous Preset and Clear

- **Bubble on the symbol means "active-low"**
  - When preset = 0, preset Q to 1
  - When preset = 1, do nothing
  - When clear = 0, clear Q to 0
  - When clear = 1, do nothing
  - "Preset" and "Clear" also known as "Set" and "Reset" respectively
  - In this circuit, preset and clear are asynchronous
  - Q changes immediately when preset or clear are active, regardless of clock

D Flip-Flop with Synchronous Clear

- Asynchronous active-low clear: Q immediately clears to 0
- Synchronous active-low clear: Q clears to 0 on rising-edge of clock

Sequential Logic Circuits

Some slides modified from:
- S. Dandamudi, "Fundamentals of Computer Organization and Design"
In typical nomenclature, a register is a name for a collection of flip-flops used to hold a bus (i.e. std_logic_vector).

**Register**

- In typical nomenclature, a register is a name for a collection of flip-flops used to hold a bus (i.e. std_logic_vector).

**Shift Register**

- A sample sequence

**Parallel Access Shift Register**

- A circuit

**Synchronous Up Counter**

- Enable (synchronous): when high enables the counter, when low counter holds its value.
- Load (synchronous): when load = 1, load the desired value into the counter.
- Output carry: indicates when the counter “rolls over”.
- D3 downto D0, Q3 downto Q0 is how to interpret MSB to LSB.

**Read Only Memory (ROM)**

- Addressable read-only memory
- Can be synchronous (with clock) or asynchronous (no clock)
- Read signal is optional (can be always on)
Read-Only Memory (ROM)

- More efficient than registers for storing large amounts of data
- Can read and write to RAM
- Addressable memory
- Can be synchronous (with clock) or asynchronous (no clock)
- SRAM dimensions are:
  - (number of words) x (bits per word)
    - SRAM
  - Address is m bits, data is n bits
    - 2^m x n-bit RAM
  - Example: address is 5 bits, data is 8 bits
    - 32 x 8-bit RAM

Random Access Memory (RAM)

- Addressable memory
- Can be synchronous (with clock) or asynchronous (no clock)
- Address is m bits, data is n bits
- 2^m x n-bit RAM
- Example: address is 5 bits, data is 8 bits
  - 32 x 8-bit RAM

**Write**
- Data_in and address are stable
- Assert write signal (then de-assert)

**Read (optional)**
- Address is stable
- Assert read signal
- Data_out is valid