Unit 7
Fundamental Digital Building Blocks:
Decoders & Multiplexers

Checkers / Decoders
- An AND gate only outputs ‘1’ for 1 combination
  - That combination can be changed by adding inverters to the inputs
  - We can think of the AND gate as “checking” or “decoding” a specific combination and outputting a ‘1’ when it matches.
Checkers / Decoders

- Place inverters at the input of the AND gates such that
  - F produces '1' only for input combination \(x,y,z = 010\)
  - G produces '1' only for input combination \(x,y,z = 110\)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</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>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>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>G</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>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>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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

AND gate decoding (checking for) combination 010

AND gate decoding (checking for) combination 110

Checkers / Decoders

- An OR gate only outputs '0' for 1 combination
  - That combination can be changed by adding inverters to the inputs

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</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>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>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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Add inverters to create an OR gate decoding (checking for) combination 010

Add inverters to create an OR gate decoding (checking for) combination 110

Decoder Exercise

- Compilers translate software to instructions that tell the processor to ADD, LOAD from Memory, Store to Memory, etc.
- These instructions are binary codes
- The processor must decode the instruction
- Create an AND gate decoder for each instruction type in the table that will produce '1' when that instruction is about to be executed

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>6-bit OPCODE OP[5:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>001000</td>
</tr>
<tr>
<td>LOAD</td>
<td>100011</td>
</tr>
<tr>
<td>STORE</td>
<td>101011</td>
</tr>
<tr>
<td>BRANCH</td>
<td>000100</td>
</tr>
</tbody>
</table>

Full Decoders

- A full decoder is a building block that:
  - Takes in an n-bit binary number as input
  - Decodes that binary number and activates the corresponding output
  - Individual outputs for _____________ input combinations

3-bit binary number

3-to-8 Decoder

There are gates inside to implement each output

1 output for each combination of the input number
Decoders

- A decoder is a building block that:
  - Takes a binary number as input
  - Decodes that binary number and activates the corresponding output
  - Put in 6=110, Output 6 activates ('1')
  - Put in 5=101, Output 5 activates ('1')

 Decoder Sizes

- A decoder w/ an n-bit input has $2^n$ outputs
  - 1 output for every combination of the n-bit input

Exercise

- Complete the design of a 2-to-4 decoder
Building Decoders

3-bit number (A2:A0)

Checker for 000
Checker for 001
Checker for 010
Checker for 011
Checker for 100
Checker for 101
Checker for 110
Checker for 111

Checker

O0 O1 O2 O3 O4 O5 O6 O7

A0
A1
A2

Consider any problems with this design.

Vending Machine Example

Assuming the keypad produces a 4-bit numeric output, add logic to produce the release signals for each of the 16 vending items.

1 2 3
4 5 6
7 8 9
0

12 13 14 15

Consider any problems with this design.

Enables

• In a normal decoder exactly one output is active at all times
• It may be undesirable to always have an active output
• We can add an extra input (called an enable) that can independently force all the outputs to their inactive values

2-to-4 Decoder

D0
D1
D2
D3

Y
X (MSB)

Enable

One output will always be active

When E=0, inputs is ignored

Since E=0, all outputs = 0

When E=1, inputs will cause the appropriate output to go active

Since E=1, outputs will function normally
Implementing Enables

- Original 2-to-4 decoder

When E = 0, force all outputs = 0
When E = 1, outputs operate as they did originally

Multiplexers

- Multiplexers are one of the most common digital circuits
- Anatomy: n data inputs, \( \log_2 n \) select bits, 1 output
- A multiplexer ("mux" for short) selects one data input and passes it to the output

<table>
<thead>
<tr>
<th>( S_1 )</th>
<th>( S_0 )</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( i_0 )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( i_1 )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( i_2 )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>( i_3 )</td>
</tr>
</tbody>
</table>

Multiplexers

Thus, input 2 = C is selected and passed to the output

Select bits = 10₂ = 2₁₀

Multiplexers

Thus, input 0 = A is selected and passed to the output

Select bits = 00₂ = 0₁₀
### Multiplexers

**2-to-1 Mux, 32-bit wide mux**

<table>
<thead>
<tr>
<th>S</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Thus, input 1 = B is selected and passed to the output.

\[ \text{Select bits } = 1 \oplus 1 = 1 \oplus 1 \]

### Recall Using T1/T2

- 1\textsuperscript{st} level of AND gates act as barriers only passing 1 channel
- OR gates combines 3 streams of 0's with the 1 channel that got passed (i.e. ICH1)
- 2\textsuperscript{nd} level of AND gates passes the channel to only the selected output

### Exercise: Build a 4-to-1 mux

- Complete the 4-to-1 mux to the right by drawing wires between the 2-to-4 decode and the AND gates

### Building a Mux

- To build a mux
  - Decode the select bits and include the corresponding data input.
  - Finally OR all the first level outputs together.
Building a Mux

- To build a mux
  - Decode the select bits and include the corresponding data input.
  - Finally OR all the first level outputs together.

S_1S_0 = \text{11}_2

\[ \begin{array}{c|c|c}
S_1 & S_0 & Y \\
\hline
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 2 \\
1 & 1 & 3 \\
\end{array} \]


Building Wide Muxes

- So far muxes only have single bit inputs...
  - I_0 is only 1-bit
  - I_1 is only 1-bit
- What if we still want to select between 2 inputs but now each input is a 4-bit number
- Use a 4-bit wide 2-to-1 mux

Use one mux per _________
- To build a 4-bit wide 2-to-1 mux, use ___ separate 2-to-1 muxes

Operation:
- When S=0, all muxes pass their I_0 inputs which means all the A bits get through
- When S=1, all muxes pass their I_1 inputs which means all the B bits get through

In general, to build an \text{m-bit wide (i.e. m-lane)} n-to-1 mux, use ___ individual _______ muxes
### Multiplexers

#### 4-to-1 Mux, 32-bit wide mux

1. **Select bits = 00₂ = 0₁₀**
2. Thus, input 0 = A[31:0] is selected and passed to the output

#### 2-to-1 Mux, 32-bit wide mux

1. **Select bits = 1₂ = 1₁₀**
2. Thus, input 1 = B[31:0] is selected and passed to the output

### Exercise

- How many 1-bit wide muxes and of what size would you need to build a **4-to-1, 8-bit wide mux** (i.e. there are 4 numbers: W[7:0], X[7:0], Y[7:0] and Z[7:0] and you must select one)
- How many 1-bit wide muxes and of what size would you need to build a **8-to-1, 2-bit wide mux**?

### Building Large Muxes

- Similar to a tournament of sports teams
  - Many teams enter and then are narrowed down to 1 winner
  - In each round winners play _________

Final output:
Design an 8-to-1 mux with 2-to-1 Muxes

Cascading Muxes
- Use several small muxes to build large ones
- Rules
  1. Arrange the muxes in stages (based on necessary number of inputs in 1st stage)
  2. Outputs of one stage feed to inputs of the next until only 1 final output
  3. All muxes in a stage connect to the same group of select bits
     - Usually, LSB connects to first stage
     - MSB connect to last stage

Building a 4-to-1 Mux

4-to-1 mux built w/ 2-to-1 muxes

Walk through an example:
\[ S_1 S_0 = 01 \]
Building a 4-to-1 Mux

Walk through an example:

\[ S_1 S_0 = 01 \]

Device vs. System Labels

- **Device Labels**: Signal names used **inside** the block
- **System Labels**: Signal names used **outside** the block

**Analogy**: Formal and Actual parameters in software function calls

1. \( a \) and \( b \) are like device labels and indicate the names used inside a block.
2. \( x \) and \( y \) are like system labels and represent the actual values to be used.

Exercise

- Sketch how you could build a 16-to-1 mux with 4-to-1 muxes? 8-to-1 and 2-to1 muxes?
Exercise

• Create a 3-to-1 mux using 2-to-1 muxes
  – Inputs: I0, I1, I2 and select bits S1,S0
  – Output: Y

<table>
<thead>
<tr>
<th>S1</th>
<th>S0</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>D0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>D1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>D2</td>
</tr>
</tbody>
</table>

Select-bit Ordering

• If we connect the select bits as shown to build an 8-to-1 mux, show how to label the inputs (i0-i7) so that the correct input is passed based on the binary value of S2:S0

<table>
<thead>
<tr>
<th>Select</th>
<th>S2</th>
<th>S1</th>
<th>S0</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>