Today

• Fundamental concepts of digital systems (Mano Chapter 1)
• Binary codes, number systems, and arithmetic (Ch 1)
• Boolean algebra (Ch 2)
• Simplification of switching equations (Ch 3)
• Digital device characteristics (e.g., TTL, CMOS)/design considerations (Ch 10)
• Combinatoric logical design including LSI implementation (Chapter 4)
• Hazards, Races, and time related issues in digital design (Ch 9)
• Flip-flops and state memory elements (Ch 5)
• Sequential logic analysis and design (Ch 5)
• Synchronous vs. asynchronous design (Ch 9)
• Counters, shift register circuits (Ch 6)
• Memory and Programmable logic (Ch 7)
• Minimization of sequential systems
• Introduction to Finite Automata
### Standard Combinational Circuits in TTL

- All part number prefixed by 74, 74S, 74LS, etc.

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Description</th>
<th>Part Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'00, '01, '03, '10, '12, '13, '18, 20, '22, '26, '30, '37, '38, '39, '40</td>
<td>NAND</td>
<td>'70-'79</td>
<td>Flip-Flops</td>
</tr>
<tr>
<td>'02, '23, '24, '25, '27, '28, '33</td>
<td>NOR</td>
<td>'80, '82, '83,</td>
<td>Adders</td>
</tr>
<tr>
<td>'04, '05, '06, '14, '16</td>
<td>NOT</td>
<td>'81, '84, '89</td>
<td>RAM</td>
</tr>
<tr>
<td>'07, '17</td>
<td>Buffer</td>
<td>'85</td>
<td>Magnitude comparator</td>
</tr>
<tr>
<td>'08, '09, '11, '15, '21</td>
<td>AND</td>
<td>'86</td>
<td>XOR</td>
</tr>
<tr>
<td>'32</td>
<td>OR</td>
<td>'87</td>
<td>True/complement</td>
</tr>
<tr>
<td>'41, '42, '43, '44</td>
<td>4-line to 10-line decoder</td>
<td>'88</td>
<td>ROM</td>
</tr>
<tr>
<td>'45</td>
<td>BCD-to-decimal</td>
<td>'90-'116</td>
<td>Counters, S/R,</td>
</tr>
<tr>
<td>'46, '47, '48, '49</td>
<td>BCD-to-7-segment</td>
<td>'160-'179,</td>
<td>Latches</td>
</tr>
<tr>
<td>'52</td>
<td>AND-OR</td>
<td>'138, '139,</td>
<td>Decoder, MUX,</td>
</tr>
<tr>
<td>'60, '61</td>
<td>AND expander</td>
<td>'148-'159,</td>
<td>Encoders,</td>
</tr>
<tr>
<td>'62</td>
<td>AND-OR expander</td>
<td>'251-258,</td>
<td>Selectors, DeMUX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'348-'359</td>
<td></td>
</tr>
</tbody>
</table>
Generalized Combinational Circuits

• In general, $F(\ )$ will represent some high-level function needed for a system

\[
\bar{O} = (o_0, o_1, o_2, \ldots, o_{m-1}) = F(\bar{i}) = F(i_0, i_1, i_2, \ldots, i_{n-1})
\]
Generalized Combinational Circuits

- E.g., ASCII to 7-segment decoder needed for a display system

\[
\bar{O} = (a, b, c, d, e, f, g) = F(\bar{B}) = F(b_0, b_1, b_2, ..., b_7)
\]
Analysis vs. Design

• Analysis procedure:
  – Given a logic diagram, find $F(\ )$

$$F(x, y, z) = x \cdot y + x' \cdot z$$

• Design procedure:
  – Given $F(\ )$, design a logic circuit that implements $F(\ )$ with minimum number of gates

$$F(x, y, z) = xy + xz' + xy' + w = (w' x' y' + xyz)'$$
• Consider this logic diagram – what function does it perform?:

\[ F(a,b,c,d,e,f) \]
Analysis

• Consider this logic diagram – what function does it perform?:

1. Label first level gate outputs
Analysis

• Consider this logic diagram – what function does it perform?:

\[F(a,b,c,d,e,f)\]

1. Label first level gate outputs
2. Determine Boolean functions

\[T_1 = e'f'a\]
\[T_2 = e'fb\]
\[T_3 = ef'c\]
\[T_4 = efd\]
Analysis

• Consider this logic diagram – what function does it perform?:

\[
F(a,b,c,d,e,f) = T_1 + T_2 + T_3 + T_4
\]

1. Label first level gate outputs
2. Determine Boolean functions
3. Iterate until reaching output

\[
F(a,b,c,d,e,f) = e'f'a + e'fb + ef'c + efd
\]
Analysis

- Consider this logic diagram – what function does it perform?:

\[
\begin{align*}
F(a,b,c,d,e,f) &= T_1 + T_2 + T_3 + T_4 \\
T_1 &= e'f'a \\
T_2 &= e'fb \\
T_3 &= ef'c \\
T_4 &= efd
\end{align*}
\]

1. Label first level gate outputs
2. Determine Boolean functions
3. Iterate until reaching output

F(a,b,c,d,e,f) = e'f'a + e'fb + ef'c + efd

F( ) is a 4 input MUX
Design

• Design a 2-line to 4-line decoder with enable

1. Create a high-level function definition
Design

• Design a 2-line to 4-line decoder with enable

1. Create a high-level function definition, determine I/O requirements
Design

- Design a 2-line to 4-line decoder with enable

1. Create a high-level function definition, determine I/O requirements
2. Define truth table

<table>
<thead>
<tr>
<th>E</th>
<th>A_1</th>
<th>A_0</th>
<th>B_0</th>
<th>B_1</th>
<th>B_2</th>
<th>B_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</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>1</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</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>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Design

• Design a 2-line to 4-line decoder with enable

1. Create a high-level function definition, determine I/O requirements
2. Define truth table
3. Derive Boolean functions for each output

\[
\begin{align*}
B_0 &= EA_1'A_0' \\
B_1 &= EA_1'A_0 \\
B_2 &= EA_1A_0' \\
B_3 &= EA_1A_0 \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>E</th>
<th>A_1</th>
<th>A_0</th>
<th>B_0</th>
<th>B_1</th>
<th>B_2</th>
<th>B_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</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>1</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</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>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Design

• Design a 2-line to 4-line decoder with enable

1. Create a high-level function definition, determine I/O requirements
2. Define truth table
3. Derive Boolean functions for each output
4. Draw the logic diagram and verify correctness

\[
\begin{align*}
B_0 &= E A_1' A_0' \\
B_1 &= E A_1' A_0 \\
B_2 &= E A_1 A_0' \\
B_3 &= E A_1 A_0
\end{align*}
\]

<table>
<thead>
<tr>
<th>E</th>
<th>A_1</th>
<th>A_0</th>
<th>B_0</th>
<th>B_1</th>
<th>B_2</th>
<th>B_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</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>1</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</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>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Binary Addition Review

• 1-bit addition:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0*</td>
</tr>
</tbody>
</table>

* carry
## Binary Addition Review

- **1-bit addition:**

  \[
  \begin{array}{c|cc}
  + & 0 & 1 \\
  \hline
  0 & 0 & 1 \\
  1 & 1 & 0^* \\
  \end{array}
  \]

  * carry

- **As two Boolean functions:**

  \[
  \begin{array}{c|ccc}
  + & 0 & 1 & & + & 0 & 1 \\
  \hline
  0 & 0 & 1 & & 0 & 0 & 0 \\
  1 & 1 & 0 & & 1 & 0 & 1 \\
  \end{array}
  \]

  sum  carry
Binary Addition Review

• 1-bit addition:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0*</td>
</tr>
</tbody>
</table>

* carry

• As two Boolean functions:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

sum

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

carry

\[ \text{sum}(a, b) = a' b + ab' = a \oplus b \quad \text{carry}(a, b) = ab \]
Half Adder

\[ \text{sum}(a, b) = a' b + ab' = a \oplus b \quad \text{carry}(a, b) = ab \]
Full Adder

\[ \text{sum}_i = \text{FA}(\text{a}_i, \text{b}_i, \text{c}_{\text{in}_i}) \]

\[ \text{c}_{\text{out}_i} = \text{FA}(\text{a}_i, \text{b}_i, \text{c}_{\text{in}_i}) \]
Full Adder

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c_{in}</th>
<th>c_{out}</th>
<th>sum</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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</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>
</tr>
</tbody>
</table>
Full Adder

The Full Adder (FA) is a logic circuit that adds two binary numbers and an input carry to produce a sum and an output carry. The symbols for a Full Adder are:

- $a_i$: Input A
- $b_i$: Input B
- $c_{in}$: Input carry
- $c_{out}$: Output carry
- $\text{sum}_i$: Sum

The truth table for a Full Adder is:

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c_{in}$</th>
<th>$c_{out}$</th>
<th>$\text{sum}_i$</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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</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>
</tr>
</tbody>
</table>

The circuit diagram shows the inputs and outputs of the Full Adder, along with the truth tables for the sum and carry outputs.
Full Adder

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c_in</th>
<th>c_out</th>
<th>sum</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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</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>
</tr>
</tbody>
</table>

\[ \text{sum} = a \oplus b \oplus c_i \]

\[ c_o(a, b, c_i) = ac_i + bc_i + ab \]
Full Adder Implementation

\[
\begin{align*}
\text{FA} & \quad \begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
& \text{HA} & \\
& \text{s} & \text{c}
\end{array} \\
\text{cout} & \quad \begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
& \text{HA} & \\
\text{c_i} & \text{c} & \text{s}
\end{array}
\end{align*}
\]
Binary Adder

- Multiple Full Adders can be cascaded to create an arbitrary precision adder
Binary Adder

- Multiple Full Adders can be cascaded to create an arbitrary precision adder
- But there is an accumulation of delay through the carry stages

(1 AND + 1 OR Delays) x (N-1) stages
Binary Adder Carry Propagation Delay

- $A/B_{in}$
- Sum
- $Carry_i$
- $Carry_{i+1}$
- $Carry_{i+2}$
- $Carry_{i+3}$

1 XOR delay ~3 gate delays
1 AND delay ~2 gate delays
1 OR delay ~2 gate delays
Carries in Adders

• Consider the 8-bit sum:

\[
\begin{array}{c}
01111111 \\
+ 00000001 \\
\hline
10000000
\end{array}
\]

• There is a carry at each stage
• If the necessary carries could be scanned once in advance of the addition, incremental delays could be avoided
Carries in Adders
Review of Binary Subtraction

Define:  \( \bar{x} = r - x \)

\( -B \triangleq \bar{B} + 1 \)

\[
\begin{array}{c}
01001001 \\
-00110101 \\
\hline
11001010+1
\end{array}
\quad
\begin{array}{c}
01001001 \\
+11001011 \\
\hline
100010100
\end{array}

73
-53
20

• Is there an easy way to do the 2’s complement in one step?
Review of Binary Subtraction

Define: \( \bar{x} = r - x \)

\[-B \triangleq \bar{B} + 1\]

\[
\begin{array}{c}
01001001 \\
01001001 + 1 \\
10001010
\end{array}
\]

\[
\begin{array}{c}
-00110101 \\
11001011 + 1 \\
10001010
\end{array}
\]

\[73 - 53 = 20\]

• Is there an easy way to do the 2’s complement in one step?

\[\text{B = 1’s complement A}\]

\[\text{B+C}_0 = 2’s \text{ complement A}\]
Adder/Subtractor

If V asserted, overflow has occurred
Review of Multiplication

• Decimal

\[
\begin{array}{c}
123 \\
\times 45 \\
\hline
615 \\
492 \\
5535
\end{array}
\]

\[\text{carries}\]
Review of Multiplication

- Decimal

\[
\begin{array}{c}
123 \\
\times \ 45 \\
\hline
615 \\
492 \\
5535
\end{array}
\]

- Binary

\[
\begin{array}{c}
101101 \\
\times \ 11001 \\
\hline
101101 \\
000000 \\
000000 \\
101101 \\
10001100101
\end{array}
\]
2-bit By 2-bit Binary Multiplication

\[
\begin{array}{c|c|c}
\times & B_1 & B_0 \\
A_1 & A_1 & A_0 \\
A_0B_1 & A_0B_0 \\
\hline
A_1B_1 & A_1B_0 \\
C_3 & C_2 & C_1 & C_0
\end{array}
\]
4-Bit By 3-Bit Binary Multiplier

A₀
A₁
A₂
B₀ B₁ B₂ B₃

4-bit adder

C₀ C₃ C₂ C₁

S₀ S₁ S₂ S₃

4-bit adder

B₀ B₁ B₂ B₃

0
Uses for Decoders

- Memory address expansion

![Diagram](https://via.placeholder.com/150)

- Four 2\(^N\) location Memories

- 2-bit to 4-line

- E

- Bank 0

- Bank 1

- Bank 2

- Bank 3

- Data_out

- A\(_{0:N+2}\)

- A\(_{0:N-1}\)

- A\(_N\)

- A\(_{N+1}\)

- A\(_{N+2}\)
2-Line to 4-Line Decoder
3-Line to 8-Line Decoder

• A 2N to $2^N$ Decoder can be created from two N to $2^N$ Decoders
Encoder

- Encoder performs inverse operation of Decoder

F(A₀, A₁) = (A₀, A₁)
## Encoder Truth Table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>D₀ D₁ D₂ D₃ D₄ D₅ D₆ D₇</td>
<td>x y z</td>
</tr>
<tr>
<td>1 0 0 0 0 0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 0 0 1 0 0 0 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>0 0 0 0 1 0 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 1 0 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

\[
x = D₄ + D₅ + D₆ + D₇
\]
\[
y = D₂ + D₃ + D₆ + D₇
\]
\[
z = D₁ + D₃ + D₅ + D₇
\]

- What should output be for input (00000000)?
- What about (00100100)?
Priority Encoder

- Output encodes the largest (highest index) input that is 1.
- V indicates if there are any 1’s in the input

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_0</td>
<td>x</td>
</tr>
<tr>
<td>D_1</td>
<td>y</td>
</tr>
<tr>
<td>D_2</td>
<td>z</td>
</tr>
<tr>
<td>D_3</td>
<td>V</td>
</tr>
<tr>
<td>D_4</td>
<td></td>
</tr>
<tr>
<td>D_5</td>
<td></td>
</tr>
<tr>
<td>D_6</td>
<td></td>
</tr>
<tr>
<td>D_7</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>X X X 0</td>
</tr>
<tr>
<td>1 0 0 0 0 0 0</td>
<td>0 0 0 0 1</td>
</tr>
<tr>
<td>X 1 0 0 0 0 0</td>
<td>0 0 1 1 1</td>
</tr>
<tr>
<td>X X 1 0 0 0 0</td>
<td>0 1 0 1 1</td>
</tr>
<tr>
<td>X X X 1 0 0 0</td>
<td>0 1 1 1 1</td>
</tr>
<tr>
<td>X X X X 1 0 0</td>
<td>1 0 0 1 1</td>
</tr>
<tr>
<td>X X X X X 1 0</td>
<td>1 0 1 1 1</td>
</tr>
<tr>
<td>X X X X X X 1</td>
<td>1 1 0 1</td>
</tr>
<tr>
<td>X X X X X X X 1</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

Copyright ©2004
Stevens Institute of Technology
All rights reserved
Multiplexer

I_0
I_1
I_2
I_3
S_1
S_0

I_0
I_1
I_2
I_3

S_1S_0

I_0
I_1
I_2
I_3

2-bit to 4-line

A_0  A_1
B_0  B_1  B_2  B_3
Uses for MUXes

• Combining multiple information sources onto one channel
Function Implementation with MUX

• With static values I[0:15], this multiplexer implements the truth table shown.

<table>
<thead>
<tr>
<th>S_3</th>
<th>S_2</th>
<th>S_1</th>
<th>S_0</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I_0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>I_1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>I_2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>I_3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>I_4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>I_5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>I_6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>I_7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I_8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>I_9</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>I_{10}</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>I_{11}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>I_{12}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>I_{13}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>I_{14}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>I_{15}</td>
</tr>
</tbody>
</table>
DEMUX

- DEMUX performs inverse function of MUX
Multiplexing and Demultiplexing Data Streams

- Most speech signals in the telephone plant is carried on T1 transmission facility:
  - 24 voice channels, each sampled at 8 kHz with 8 bits/channel + synchronization = 1.544 Mb/s
- Multiple T1’s are combined to form a T3 line at ~45 Mb/s
Summary

• Fundamental concepts of digital systems (Mano Chapter 1)
• Binary codes, number systems, and arithmetic (Ch 1)
• Boolean algebra (Ch 2)
• Simplification of switching equations (Ch 3)
• Digital device characteristics (e.g., TTL, CMOS)/design considerations (Ch 10)
• Combinatoric logical design including LSI implementation (Chapter 4)
• Hazards, Races, and time related issues in digital design (Ch 9)
• Flip-flops and state memory elements (Ch 5)
• Sequential logic analysis and design (Ch 5)
• Synchronous vs. asynchronous design (Ch 9)
• Counters, shift register circuits (Ch 6)
• Memory and Programmable logic (Ch 7)
• Minimization of sequential systems
• Introduction to Finite Automata
Homework 5 – due in Class 7

• As always, show all work
• Problems 4-1, 4-5 (x=4, y=2, z=1), (A=4, B=2, C=1), 4-6, 4-7