Designing with MSI

Documentation Standards

- Block diagrams
  first step in hierarchical design
- Schematic diagrams
- Timing diagrams
- Circuit descriptions
Designing with MSI

Documentation standard

Block Diagram
Schematic diagrams

- Details of component inputs, outputs, and interconnections
- Reference designators
- Title blocks
- Names for all signals
- Page-to-page connectors
Designing with MSI

Block Diagram For Wake

Wake Period

Input keys

Digit1
Digit0

Error_Key1
Error_Key0

Select one of Mux

Display

D1
D0

To BCD calculator
From BCD calculator

To Error Display

(warmer + cooler)
Chapter : Design modules
I. Using MUX to Implement logic function

- A multiplex (Data selector) is a CLN module with:
  - \(2^n\) data inputs
  - n control inputs
  - 1 output

  Depending on the control inputs, the multiplexer connects one of the inputs to the output line.

- Block diagram of an 4-to-1 multiplexer:
I. Using MUX to Implement logic function

Circuit of an 4-to-1 multiplexer

May add an enable signal E
I. Using MUX to Implement logic function

Block diagram of an 8-to-1 multiplexer:
Designing with MSI

I. Using MUX to Implement logic function

Circuit of an 8-to-1 multiplexer
Designing with MSI

I. Using MUX to Implement logic function

Truth table of an 8-to-1 multiplexer

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN_L</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
I. Using MUX to Implement logic function

Example of design

A 2^n input lines and n selection lines MUX may be used to realize any function of (n+1) variables

Example

Use an 8-to-1 MUX to realize the following function of 4 variables

F( A,B,C,D) = Σ(0,2,4,5,6,8,10,13)
= A’B’C’D’ + A’B’CD’ + A’BC’D’ + A’BC’D + A’BCD’
+ AB’C’D’ + AB’CD’ + ABC’D

Solution

Use the variables A, B, C as the control (selection) inputs and use the remaining variable D to determine the input lines.

Rewrite F to to determine a factor for each input combination ABC:

F( A,B,C,D) = A’B’C’D’ + A’B’CD’ + A’BC’(D’ + D) + A’BCD’
+ AB’C’D’ + AB’CD’ + ABC’D + ABC (0)
I. Using 8-to-1 MUX to Implement logic function

Solution (Continued ....)

\[ F( A, B, C, D) = A'B'C'D' + A'B'CD' + A'BC'(D' + D) + A'BCD' \\
+ AB'C'D' + AB'CD' + ABC'D + ABC \ (0) \]

So the input to the 8-to-1 MUX are given by:
I0=D', I1=D', I2=1, I3=D', I4=D', I5=D', I6=D, I7=0
I. Using 8-to-1 MUX to Implement logic function

Example

Same function $F(A,B,C,D)$
Use BCD as the selection (control) lines

$$F(A,B,C,D) = \Sigma(0,2,4,5,6,8,10,13) = A'B'C'D' + A'B'CD' + A'BC'D' + A'BC'D + A'BCD' + AB'C'D' + AB'CD' + ABC'D$$

$$= B'C'D' + B'C'D + B'CD' + B'CD + BC'D' + BC'D + BCD' + BCD$$

$$= B'C'D'(A+A') + B'C'D(0) + B'CD'(A'+A) + B'CD(0) + BC'D'(A') + BC'D(A'+A) + BCD'(A') + BCD(0)$$
I. Using 8-to-1 MUX to Implement logic function

**Example**

```
<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>A'</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D0</td>
<td>D1</td>
<td>D2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A'</td>
<td>D6</td>
<td>D7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A'</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**F(A,B,C,D)**

**Exercise**
Repeat using as selection lines:
- A, C, D
- A, B, D
Designing with MSI

I. Using 4-to-1 MUX to Implement logic function

Example

\[ F(A,B,C,D) = \Sigma(3,4,8,9,10,13,14,15) = A'B'CD + A'BC'D' + AB'C'D' + AB'C'D + AB'CD' + ABC'D + ABCD' + ABCD \]

Use AB for Selection lines and factor out the various combinations of AB

\[ = A'B'(CD) + A'B(C'D') + AB'(C'D' + C'D + CD') + AB(C'D' + CD' + CD) \]

Implement the circuit of the input lines using NAND (or other) gates
I. Using 4-to-1 MUX to Implement logic function

Example
II. Decoders

- Depending on the control inputs, the multiplexer connects one of the inputs to the output line.
- **Block diagram of an 4-to-1 multiplexer:**
II. Decoders

- General decoder structure

- Map each input code to one of the output
- Typically \( n \) inputs, \( 2^n \) outputs
  - 2-to-4, 3-to-8, 4-to-16, etc.
II. Decoders

Binary 2-to-4 decoder

Note “x” (don’t care) notation.
II. Decoders

2-to-4-decoder logic diagram
II. Decoders

MSI 2-to-4 decoder

- Input buffering (less load)
- NAND gates (faster)
II. Decoders

Decoder Symbol

![Diagram of a 1/2 74x139 decoder](image)
II. Decoders

Complete 74x139 Decoder
II. Decoders

More decoder symbols
Designing with MSI

II. Decoders

3-to-8 Decoder

A (1)
B (2)
C (3)
G1 (6)
G2A_L (4)
G2B_L (5)
II. Decoders

74x138 3-to-8-decoder symbol
I. Using decoders to Implement logic function

Example: Given $F_1 = \sum_{X,Y,Z} (1,2,3)$ and $F_2 = \sum_{X,Y,Z} (3,5,6,7)$

Implementation using 3-to-8 Decoder
I. Using decoders to Implement logic function

Example: Given $F = \sum_{X,Y,Z} (0,2,3,4,6,7)$

Implementation using 3-to-8 Decoder

We will implement the complement of $F$ and “NOT” the result

$F' = m1 + m5$
III. Programmable Logic devices

Programmable Logic Arrays (PLAs)

- Any combinational logic function can be realized as a sum of products.
- Idea: Build a large AND-OR array with lots of inputs and product terms, and programmable connections.
  - \( n \) inputs
    - AND gates have \( 2n \) inputs -- true and complement of each variable.
  - \( m \) outputs, driven by large OR gates
    - Each AND gate is programmably connected to each output’s OR gate.
  - \( p \) AND gates (\( p << 2^n \))
Designing with MSI

III. Programmable Logic devices

Example: 4x3 PLA, 6 product terms
Designing with MSI

III. Programmable Logic devices
Compact representation

Input programming

Output programming
III. Programmable Logic devices

Some product terms

\[ O_1 = l_1 \cdot l_2 + l_1' \cdot l_2' \cdot l_3 \cdot l_4' \]
\[ O_2 = l_1 \cdot l_3' + l_1' \cdot l_3 \cdot l_4 + l_2 \]
\[ O_3 = l_1 \cdot l_2 + l_1 \cdot l_3' + l_1' \cdot l_2' \cdot l_4' \]
IV. Demultiplexer

General description

- **1-to-\(2^n\) Demultiplexer has:**
  - 1- input
  - Multiple outputs
  - \(n\) select lines

- **Function:** Route the single input to the selected output

\[
m = 2^n - 1
\]
IV. Demultiplexer

Function table of a 1-to-4 Demux

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>O₀</th>
<th>O₁</th>
<th>O₂</th>
<th>O₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>1</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

O₁ = x’y’I
O₂ = x’y I
O₃ = x’y’I
O₄ = x y I
Designing with MSI

V. Design by Cascading

Cascading MUXes
Design an 8-to-1 MUX using 4-to-1 MUX and other gates

Data inputs

<table>
<thead>
<tr>
<th>I0</th>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
<th>I6</th>
<th>I7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8-to-1 MUX

En

Data select

D0
D1
D2
D3
D4
D5
D6
D7

Output

C2
C1
C0
Cascading MUXes

Design an 8-to-1 MUX using 4-to-1 MUX and other gates
IV. Design by cascading Decoders

Cascading Decoders

Recall: Decoder with an enable signal En

Note “x” (don’t care) notation.
V. Design by cascading Decoders

Cascading Decoders

Recall: Decoder with an enable signal En

<table>
<thead>
<tr>
<th>En</th>
<th>i0</th>
<th>d0</th>
<th>d1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</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>1</td>
</tr>
</tbody>
</table>

Decoder

<table>
<thead>
<tr>
<th>En</th>
<th>i0</th>
<th>i1</th>
<th>d0</th>
<th>d1</th>
<th>d2</th>
<th>d3</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>1</td>
<td>0</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>0</td>
</tr>
</tbody>
</table>

38
Cascading Decoders

Design of a 2-to-4 decoder using 1-to-2 decoders
V. Design by cascading Decoders

Cascading Decoders

Design of a 2-to-4 decoder using 1-to-2 decoders

Operation

- En enables or disable the decoder
- i1=0 enables the top decoder
- i1=1 enables the lower decoder
V. Design by cascading Decoders

Cascading Decoders

Design of a 4-to-16 decoder using 2-to-4 decoders
VI. Modular Design

- **Goal**
  - Use existing components or design subcomponents as building blocks of higher circuit

- **Types of solutions**
  - Cascading components
  - Ripple design
    - Some output at level $i$ are used input at the next level $(i+1)$
    - Linear cascading of elements
VI. Modular Design

Motivations

Design of a 2 bits binary adder

\[ S = X + Y \]

\[ X = [X1X0], \quad Y = [Y1Y0], \quad S = [S2S1S0] \]

Two types of design possible:

- Brute force approach: Draw a truth table and derive the expressions of the output variable
- Iterative design
VI. Modular Design

Example: 2 bits binary adder

<table>
<thead>
<tr>
<th>X1</th>
<th>X0</th>
<th>Y1</th>
<th>Y0</th>
<th>S0</th>
<th>S1</th>
<th>S0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>0 1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>0 1 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>0 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>1 0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>1 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1 1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What if we want to do design a 5 bits adder:
- Truth table with 10 variables
- Not practical
VI. Modular Design

**Iterative design**
- Identify a basic component: 1-bit adder with carry in
- Use the basic component iteratively

**Basic component**

<table>
<thead>
<tr>
<th>Xi</th>
<th>1-bit Full adder</th>
<th>Yi</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Ci</td>
</tr>
<tr>
<td>Ci-1</td>
<td></td>
<td>Si</td>
</tr>
</tbody>
</table>

**Design of Basic component**
- Draw its truth table
- Design the corresponding circuit

**Iterative design of n-bit adder**

<table>
<thead>
<tr>
<th>Xn-1</th>
<th>Yn-1</th>
<th>1-bit Full adder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Sn-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Xn-2</th>
<th>Yn-2</th>
<th>1-bit Full adder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Sn-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cn-2</th>
<th>Cn-3</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>C0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X0</th>
<th>Y0</th>
<th>1-bit Full adder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>S0</td>
</tr>
</tbody>
</table>
VI. Modular Design

Another Example of Modular Design

Design an n-bit comparator to produce the following output F:

- F = 1 if X > Y
- F = 0 otherwise  X, Y are n-bit binaries

**Basic component C1**

- A 1-bit comparator with comparison result from preceding stage
- 3 inputs
  - Xi and Yi bit at stage i, and Fi-1 Result from stage i-1
- 1 output: Fi result of stage I

**Iterative comparison of two n-bits**

X = [Xn-1 ... X0] and Y = [Yn-1 ... Y0]
VI. Modular Design

Another Example of Modular Design

Design of the Basic 1-bit Comparator \( Ci \)

Truth Table

<table>
<thead>
<tr>
<th>Fi-1</th>
<th>Xi</th>
<th>Yi</th>
<th>Fi</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Simplified SOP Logic Expression

\[ Fi = XiFi-1 + XiYi' + Yi'Fi-1 \]