DESIGN OF MULTILEVEL NETWORKS

- Transformations to satisfy constraints
  - number of gate inputs
  - network size
  - network delay

- Design of networks with \texttt{XOR} and \texttt{XNOR} gates

- Design of networks with multiplexers (\texttt{MUXes})
Design more complex than for two-level networks

- No standard form
- Several requirements have to be met simultaneously
- Several outputs have to be considered
- CAD tools (logic synthesis) used
A design procedure

1. Obtain SP or PS expressions for the functions of the system

2. Transform the expressions (or the corresponding two-level networks) so that the requirements are met

3. Replace \texttt{AND} and \texttt{OR} gates by \texttt{NAND} and \texttt{NOR} when appropriate

Several iterations might be needed
Typical transformations to meet network requirements

- Size of network: number of gates and number of gate inputs
- Number of gates reduced by
  1. Factoring
  2. Subexpressions shared by several network outputs
Example 6.1: 1-bit comparator

**Inputs:** $x, y \in \{0, 1\}$  
$c \in \{\text{GREATER, EQUAL, LESS}\}$

**Output:** $z \in \{\text{GREATER, EQUAL, LESS}\}$

**Function:**  
\[
z = \begin{cases} 
\text{GREATER} & \text{if } x > y \text{ or } (x = y \text{ and } c = \text{GREATER}) \\
\text{EQUAL} & \text{if } x = y \text{ and } c = \text{EQUAL} \\
\text{LESS} & \text{if } x < y \text{ or } (x = y \text{ and } c = \text{LESS})
\end{cases}
\]
Example 6.1: Comparator (cont.)

Figure 6.1: Comparator
Example 6.1: Comparator (cont.)

Coding:

<table>
<thead>
<tr>
<th>$c$</th>
<th>$z$</th>
<th>$c_2$</th>
<th>$c_1$</th>
<th>$c_0$</th>
<th>$z_2$</th>
<th>$z_1$</th>
<th>$z_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_2$</td>
<td>$z_1$</td>
<td>$z_0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$GREATER$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$EQUAL$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$LESS$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x, y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$c$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>010</td>
<td>010</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
</tr>
</tbody>
</table>
Example 6.1: Comparator (cont.)

- Switching expressions:

\[
\begin{align*}
    z_2 &= xy' + xc_2 + y'c_2 & G \\
    z_1 &= (x' + y)(x + y')c_1 & E \\
    z_0 &= x'y + x'c_0 + yc_0 & S
\end{align*}
\]

- Resulting two-level network:
  - 7 AND and 4 OR gates
  - 22 equivalent gates
  - 25 gate inputs
Reducing network size (cont.)

Define:

\[
\begin{align*}
    t &= (x + y') \\
    w &= (x' + y) \\
    z_2 &= xy' + tc_2 \\
    z_1 &= twc_1 \\
    z_0 &= x'y + wc_0
\end{align*}
\]

- Size: 18 equivalent gates
- Further reduction: NAND network – 9 equivalent gates
Figure 6.2: 1-bit comparator implementations
Example 6.2: Modulo-64 Incrementer

- A two-level implementation:

\[
\begin{align*}
z_5 &= x_5x'_4 + x_5x'_3 + x_5x'_2 + x_5x'_1 + x_5x'_0 + x'_5x_4x_3x_2x_1x_0 \\
z_4 &= x_4x'_3 + x_4x'_2 + x_4x'_1 + x_4x'_0 + x'_4x_3x_2x_1x_0 \\
z_3 &= x_3x'_2 + x_3x'_1 + x_3x'_0 + x'_3x_2x_1x_0 \\
z_2 &= x_2x'_1 + x_2x'_0 + x'_2x_1x_0 \\
z_1 &= x_1x'_0 + x'_1x_0 \\
z_0 &= x'_0
\end{align*}
\]

- Two-level network:

7 \texttt{NOT} 20 \texttt{AND}, 5 \texttt{OR} gates, and 77 gate inputs
Factoring

\[ z_5 = x_5'x_4' + x_3' + x_2' + x_1' + x_0' + x_5'x_4x_3x_2x_1x_0 \]
\[ z_4 = x_4'x_3' + x_2' + x_1' + x_0' + x_4'x_3x_2x_1x_0 \]
\[ z_3 = x_3'x_2' + x_1' + x_0' + x_3'x_2x_1x_0 \]
\[ z_2 = x_2(x_1' + x_0') + x_2'x_1x_0 \]
\[ z_1 = x_1x_0' + x_1'x_0 \]
\[ z_0 = x_0' \]

- **Four-level network (NOT-OR-AND-OR):**

  7 NOT 10 AND and 9 OR gates, and 61 gate inputs
4-level implementation of modulo-64 incrementer

Figure 6.3: Four-level network for modulo-64 incrementer.
The fan-in of gates

- Fan-in of gates $\Leftrightarrow$ number of operands per operator
- Reduced by decomposing a large gate into several smaller gates
- AND and OR are associative,

$$a + b + c + d + e + f = (a + b + c) + (d + e + f)$$
Incrementer with max fan-in of 3

Terms to decompose:

\[
(x'_4 + x'_3 + x'_2 + x'_1 + x'_0) = (x'_4 + x'_3 + r_{210})
\]
\[
(x'_5x'_4x'_3x'_2x'_1x'_0) = x'_5a_{43}a_{210}
\]
\[
(x'_3 + x'_2 + x'_1 + x'_0) = x'_3 + r_{210}
\]
\[
(x'_4x'_3x'_2x'_1x'_0) = x'_4x'_3a_{210}
\]

\[
\begin{align*}
z_5 & = x_5(x'_4 + x'_3 + r_{210}) + x'_5a_{43}a_{210} \\
z_4 & = x_4(x'_3 + r_{210}) + x'_4x'_3a_{210} \\
z_3 & = x_3r_{210} + x'_3a_{210} \\
z_2 & = x_2(x'_1 + x'_0) + x'_2x'_1x_0 \\
z_1 & = x_1x'_0 + x'_1x_0 \\
z_0 & = x'_0
\end{align*}
\]

- More gates and more levels:
  6 NOT, 18 NAND, 3 NOR, size: 31 equivalent gates
Figure 6.4: Reducing the number of gate inputs
Example 6.4: Reducing output load of a gate (Buffering)

\[ z_i = w \cdot x \cdot y_i \quad 0 \leq i \leq 63 \]

Figure 6.5: Reducing the output load
- Output load of NAND producing $w \cdot x$: $64I$ ($I$ is load factor of NOR gate)

- Propagation delay (high to low) between $x$ and $z_i$ (load 5 at output):

$$
(0.05 + 0.038 \times 64) + (0.07 + 0.016 \times 5) = 2.63 \text{ns}
$$

- Use buffers

<table>
<thead>
<tr>
<th>Gate type</th>
<th>Fan-in</th>
<th>Propagation delays</th>
<th>Input factor</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$t_{pLH}[\text{ns}]$</td>
<td>$t_{pHL}[\text{ns}]$</td>
<td>[Standard loads]</td>
</tr>
<tr>
<td>Buffer</td>
<td>1</td>
<td>$0.15 + 0.006L$</td>
<td>$0.19 + 0.003L$</td>
<td>2</td>
</tr>
<tr>
<td>Inv. Buf.</td>
<td>1</td>
<td>$0.04 + 0.006L$</td>
<td>$0.05 + 0.006L$</td>
<td>4.7</td>
</tr>
</tbody>
</table>

- Delay:

$$
(0.05 + 0.038 \times 4) + (0.15 + 0.006 \times 32) + (0.07 + 0.016 \times 5) = 0.69 \text{ns}
$$
Example 6.5: Even parity circuit – alternatives

\[ x = (x_7, x_6, \ldots, x_0), \quad x_i \in \{0, 1\} \]

\[ z \in \{0, 1\} \]

Function:
\[
z = \begin{cases} 
1 & \text{if } \sum_{i=0}^{7} x_i \text{ is even} \\
0 & \text{otherwise}
\end{cases}
\]
Implementation 1: Two-level network.

CSP: 128 minterms – no reduction possible

Cost: 128 AND gates and one OR gate

Each AND gate 8 inputs, OR gate 128 inputs

Not practical: large number of gates, large fan-in
Implementation 2: Divide into two parts

\[ P(x) = P(x_l)P(x_r) + P'(x_l)P'(x_r) \]

Figure 6.6: Network with fan-in=4
Table 6.2: Characteristics of alternative implementations for the parity function

<table>
<thead>
<tr>
<th>Impl.</th>
<th>Network input load</th>
<th>Gates</th>
<th>No. levels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Type</td>
<td>Fan-in</td>
</tr>
<tr>
<td>1</td>
<td>64</td>
<td>AND</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OR</td>
<td>128</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>AND</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OR</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OR</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AND</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOT</td>
<td>1</td>
</tr>
</tbody>
</table>
Example 6.6: 8-input odd-parity checker

Input: \( \bar{x} = (x_7, \ldots, x_0), x_i \in \{0, 1\} \)

Output: \( z \in \{0, 1\} \)

Function: \( z = \begin{cases} 
0 & \text{if number of 1's in } \bar{x} \text{ is even} \\
1 & \text{if number of 1's in } \bar{x} \text{ is odd} 
\end{cases} \)

\[
\begin{align*}
& x_7 \\
& x_6 \\
& x_5 \\
& x_4 \\
& x_3 \\
& x_2 \\
& x_1 \\
& x_0 \\
\end{align*}
\]

Figure 6.7: Odd-parity checker

\[
z = x_7 \oplus x_6 \oplus x_5 \oplus x_4 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0
\]
Example: 32-bit equality comparator

Input: \( x = (x_{31}, \ldots, x_0), x_i \in \{0, 1\} \)
\( y = (y_{31}, \ldots, y_0), y_i \in \{0, 1\} \)

Output: \( z \in \{0, 1\} \)

Function: \( z = \begin{cases} 1 & \text{if } x_i = y_i \text{ for } 0 \leq i \leq 31 \\ 0 & \text{otherwise} \end{cases} \)

\[
z = AND(XNOR(x_{31}, y_{31}), \ldots, XNOR(x_i, y_i), \ldots, XNOR(x_0, y_0))
\]
Figure 6.8: 32-bit equality comparator
Networks with 2-input multiplexers

- 2-input multiplexer (MUX): \( z = MUX[x_1, x_0, s] = x_1 \cdot s \oplus x_0 s' \)

- Set \{MUX\} is universal (constants 0 and 1 available)

\[
\begin{align*}
\text{NOT}(x) & = MUX[0, 1, x] = 0 \cdot x \oplus 1 \cdot x' = x' \\
\text{AND}(x_1, x_0) & = MUX[x_1, 0, x_0] = x_1 x_0 \oplus 0 \cdot x'_0 = x_1 x_0
\end{align*}
\]

Figure 6.9: 2-input multiplexer and NOT and AND gates
Implementation of SFs with network of MUXes

- Shannon’s decomposition (SD)

\[ f(x_{n-1}, x_{n-2}, \ldots, x_0) = f(x_{n-1}, x_{n-2}, \ldots, 1) \cdot x_0 \]
\[ + f(x_{n-1}, x_{n-2}, \ldots, 0) \cdot x'_0 \]

\[ z = f(x_{n-1}, x_{n-2}, \ldots, x_0) \]
\[ = MUX[f(x_{n-1}, x_{n-2}, \ldots, x_1, 1), f(x_{n-1}, x_{n-2}, \ldots, x_1, 0), x_0] \]

Example:

\[ z = x_3(x_2 + x_0)x_1 = MUX[x_3x_1, x_3x_2x_1, x_0] \]
Design of networks with MUXes

- Obtain a tree of multiplexers by repeated use of SD

\[ f(x_{n-1}, \ldots, x_1, 1) \]
\[ f(x_{n-1}, \ldots, x_1, 0) \]
\[ z = f(x_{n-1}, \ldots, x_1, x_0) \]

\[ f(x_{n-1}, \ldots, x_2, 1, 1) \]
\[ f(x_{n-1}, \ldots, x_2, 0, 1) \]
\[ f(x_{n-1}, \ldots, x_2, 1, 0) \]
\[ f(x_{n-1}, \ldots, x_2, 0, 0) \]
\[ z = f(x_{n-1}, \ldots, x_1, x_0) \]

Figure 6.10: a) Realization of Shannon’s decomposition with multiplexer; b) Repeated decomposition.
Example 6.8

- Implement $f(x_3, x_2, x_1, x_0) = z = x_3(x_1 \oplus x_2 x_0)$ with MUX tree

- Decompose with respect to $x_2, x_1, x_0$

$$
\begin{align*}
    f(x_3, 0, 0, 0) &= 0 & f(x_3, 0, 0, 1) &= 0 \\
    f(x_3, 0, 1, 0) &= x_3 & f(x_3, 0, 1, 1) &= x_3 \\
    f(x_3, 1, 0, 0) &= 0 & f(x_3, 1, 0, 1) &= x_3 \\
    f(x_3, 1, 1, 0) &= x_3 & f(x_3, 1, 1, 1) &= x_3
\end{align*}
$$

- Eliminate redundant MUXes
Ordering of variables in subtrees affects the number of MUXes

![Diagram of digital system](image)

Figure 6.11: