Technology Mapping

Technology mapping transforms one logic circuit model into another one. *What is the difference between this and synthesis?*

- Discrete Components
- Programmable Logic
- Custom Integrated Circuits
- Standard-Cell Integrated Circuits

Compiler/Programmable Logic Comparison

(a) Mapping a computer program to implementations on different CPUs

- Computer Program in a high level language
- Compiler program speed, size optimizations
- CPU X (Intel Pentium)
- CPU Y (AMD Athlon)
- CPU Z (IBM PowerPC)

(b) Mapping a digital system to implementations using different FPGA technologies

- Digital system specified in an HDL
- Logic Synthesis Tool circuit speed, size optimizations
- FPGA Tech. X (Xilinx FPGA)
- FPGA Tech. Y (Altera FPGA)
- FPGA Tech. Z (Actel FPGA)
F = AB + BC + AC

Logic Synthesis

Logic Synthesis (designer definition): convert a description of a digital system in a Hardware Description Language (HDL) to an implementation technology.

Verilog HDL description

```
// Combinational Logic Circuit
module cmb_circ(Y, A, B, C);
    input A, B, C;
    output Y;
    assign Y = (A&B)|(A&C)|(B&C);
endmodule
```
Logic Synthesis

Logic Synthesis (designer definition): convert a description of a digital system in a Hardware Description Language (HDL) to an implementation technology.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity majority is
  port (   A, B, C :  in std_logic;
           Y:   out std_logic
  );
end majority;

ARCHITECTURE a of majority is
begin

  Y <= (A and B) or (A and C) or (B and C);
end a;
```

VHDL description

Gates

Logic Circuit Models

(a) Combinational logic

Boolean Equation: \[ Y = (B \land S) \lor (A \land \overline{S}) \]

Verilog:

```verilog
assign y = (b & s) | (a & !s);

always @(a or b or s)
  if (s) y = b; else y = a;
```

(b) Sequential logic

Boolean Equation: \[ Q_+ = Q \oplus A \]

Verilog:

```verilog
always @(posedge clk)
  q <= q ^ a;
```

Must be annotated with truth table that describes \( Q_+ \), \( Q \) dependence on \( D, \) \( CLK \)
## Basic Logic Gates

<table>
<thead>
<tr>
<th>Truth Table</th>
<th>Gate Symbol</th>
<th>Boolean</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>![Gate Symbol]</td>
<td>$Y = \overline{A}$</td>
<td>assign $y = !a$;</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

| AND         | ![Gate Symbol] | $Y = A \land B$ | assign $y = a \& b$; |
| A | B | Y |  |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| OR          | ![Gate Symbol] | $Y = A \lor B$ | assign $y = a \mid b$; |
| A | B | Y |  |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| NAND        | ![Gate Symbol] | $Y = \overline{A \land B}$ | assign $y = !(a \& b)$; |
| A | B | Y |  |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| NOR         | ![Gate Symbol] | $Y = \overline{A \lor B}$ | assign $y = !(a \mid b)$; |
| A | B | Y |  |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

| XOR         | ![Gate Symbol] | $Y = A \oplus B$ | assign $y = a \^ b$; |
| A | B | Y |  |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## 2:1 Multiplexer

<table>
<thead>
<tr>
<th>Truth Table</th>
<th>Gate Schematic</th>
<th>Internal net</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>B</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>x - don’t care</td>
<td>[Diagram]</td>
<td>$Y = (B \land S) \lor (A \land \overline{S})$</td>
<td>![Symbol]</td>
</tr>
</tbody>
</table>


Combinational Building Blocks

1 bit Multiplexer (2:1 MUX)

If \( S = 0 \), then \( Y = I_0 \)
If \( S = 1 \), then \( Y = I_1 \)

\[ Y = I_0 \overline{S} + I_1 S \]

Muxes are often used to select groups of bits arranged in busses.

How many wires in each bus?

Multiplexers

Shannon Expansion Theorem

- Original Function: \( f = x'y'z + xy'z + xyz' \)
- Cofactors:
  \[ f_x' = f(x=0) = y'z + yz = z \]
  \[ f_x = f(x=1) = y'z + yz' = y \oplus z \]
  \[ f = x'f_x' + xf_x \]
  \[ f = x'z + x(y \oplus z) \]

\[ \begin{array}{c|c|c|c|c}
 a & b & c & M \\
 \hline
 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 0 & 1 & 0 & 0 \\
 0 & 1 & 1 & 1 \\
 1 & 0 & 0 & 1 \\
 1 & 0 & 1 & 0 \\
 1 & 1 & 0 & 1 \\
 1 & 1 & 1 & 1 \\
\end{array} \]
Multiplexers

Memory Device Architecture

- $2^n \times m$ Device
  - $n$ inputs called “address lines”
  - $m$ outputs called “data lines”

- Contains 3 Main Subcircuits:
  1) Decoder (Address Decoder)
     - $1 : n \times 2^n$ Decoder Circuit
  2) Storage Array (Array of 1-bit Storage Cells)
     - $m \times 2^n : 1$-bit storage cell circuits
  3) Sense Amps (Amplifiers from Cells to Outputs)
     - $m : $Single-Ended OR Differential Amplifiers

Difference Between Memory Types (RAM, ROM, etc.) is Primarily Due to Storage Cell Implementation
Decoder (Review)

- $n \times 2^n$ Device
  - $n$ encoded inputs
  - $2^n$ decoded outputs

```
\begin{array}{lllll}
A_1 & A_0 & D_3 & D_2 & D_1 & D_0 \\
0 0 & 0 0 0 1 \\
0 1 & 0 0 1 0 \\
1 0 & 0 1 0 0 \\
1 1 & 1 0 0 0 \\
\end{array}
```

Decoders (with Enable)

```
\begin{array}{llll}
E & A_0 & C_1 & C_0 \\
1 0 & 0 1 \\
1 1 & 1 0 \\
0 X & 0 0 \\
\end{array}
```

```
\begin{array}{llllll}
E & A_0 & A_1 & C_3 & C_2 & C_1 & C_0 \\
1 0 0 & 0 0 0 1 \\
1 0 1 & 0 0 1 0 \\
1 1 0 & 0 1 0 0 \\
1 1 1 & 1 0 0 0 \\
0 X X & 0 0 0 0 \\
\end{array}
```
**Decoder**

<table>
<thead>
<tr>
<th>A[2:0]</th>
<th>Y0</th>
<th>Y1</th>
<th>Y2</th>
<th>Y3</th>
<th>Y4</th>
<th>Y5</th>
<th>Y6</th>
<th>Y7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>if A = 000 then Y0=1 else Y0=0;</td>
<td>if A = 001 then Y1=1 else Y1=0;</td>
<td>if A = 010 then Y2=1 else Y2=0;</td>
<td>if A = 011 then Y3=1 else Y3=0;</td>
<td>if A = 100 then Y4=1 else Y4=0;</td>
<td>if A = 101 then Y5=1 else Y5=0;</td>
<td>if A = 110 then Y6=1 else Y6=0;</td>
<td>if A = 111 then Y7=1 else Y7=0;</td>
</tr>
</tbody>
</table>

---

**Amplifiers (Review)**

- **Single-Ended Amplifier**
  - Gain: $A_v$
  - 1 input voltage, 1 output voltage referenced to common ground

  \[
  V_{in} \rightarrow A_v \rightarrow V_{out} = A_v V_{in}
  \]

- **Differential Amplifier**
  - Gain: $A_v$
  - 2 input voltages, 1 output voltage referenced to common ground

  \[
  V_1 \rightarrow \leftarrow A_v \rightarrow \leftarrow V_{out} = A_v (V_1 - V_2)
  \]

*Buffer Generally Refers to an Amplifier with Unity Gain ($A_v = 1$)*
Semiconductor Memory Device Architecture

- $2^n \times m$ Device
  - $n$ inputs called “address lines”
  - $2^n$ storage locations called “number of words”
  - $m$ outputs called “data lines”

![2x4 Decoder Diagram]

Storage Cell Array

Sense Amps

D4 D3 D2 D1 D0

Memory example

F (A,B,C) = A $\oplus$ B $\oplus$ C  \quad G = AB + AC + BC

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F</th>
<th>G</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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Recall that Exclusive OR ($\oplus$) is

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

Y = A $\oplus$ B  = A xor B

A[2:0] is 3 bit address bus, D[1:0] is 2 bit output bus.
Location 0 has “00”,
Location 1 has “10”,
Location 2 has “10”,
etc....
Logic Arrays

$F(A, B, C) = A \oplus B \oplus C$

$= \overline{AB}C + \overline{ABC} + \overline{A}BC + ABC$

$G(A, B, C) = AB + AC + BC$

What about these?

$F(A, B, C) = A \oplus B \oplus C$

$= \overline{AB}C + \overline{ABC} + \overline{A}BC + ABC$

$G(A, B, C) = AB + AC + BC$
What about these?

F(A, B, C) = A \oplus B \oplus C
= AB\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC

G(A, B, C) = AB + AC + BC

Binary Adder

F (A,B,C) = A \oplus B \oplus C \quad G = AB + AC + BC

These equations look familiar. Recall what a *Full Adder* is:

Sum = A \oplus B \oplus C_{in}

Cout = AB + C_{in}A + C_{in}B
= AB + C_{in}(A + B)

Full Adder (FA)
4 Bit Ripple Carry Adder

![Diagram of 4 Bit Ripple Carry Adder]

Full and Ripple Adders

### Truth Table for Full Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C&lt;sub&gt;i&lt;/sub&gt;</th>
<th>S</th>
<th>C&lt;sub&gt;o&lt;/sub&gt;</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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Truth Table**

- **Truth Table for Full Adder**
- **Truth Table for Ripple Carry Adder**

**Boolean Equations for Co**

\[ C_o = \text{Majority}(A, B, C_i) = (A \land B) \lor (A \land \neg C_i) \lor (B \land C_i) \]

**Diagram of 1-bit adder symbol**

**Diagram of 4-bit adder symbol and Ripple Carry Implementation**
Fixed-Point Multipliers

multiplicand \( r_2 \quad r_1 \quad r_0 \)
multiplier \( \times s_2 \quad s_1 \quad s_0 \)
partial product \( s_0 \cdot r_2 \quad s_0 \cdot r_1 \quad s_0 \cdot r_0 \quad s_1 \cdot r_2 \quad s_1 \cdot r_1 \quad s_1 \cdot r_0 \quad s_2 \cdot r_2 \quad s_2 \cdot r_1 \quad s_2 \cdot r_0 \)

\[ \begin{array}{ccc}
& 1 & 1 & 1 \\
\times & 1 & 0 & 1 \\
\hline
& 1 & 1 & 1 \\
\end{array} \quad \begin{array}{c}
0 \\
0 \\
0 \\
\end{array} \]

\[ \begin{array}{c}
+ 1 & 1 & 1 \\
\hline
\text{product} & 1 & 0 & 0 & 0 & 1 & 1 \\
\end{array} = 35 \]

Array Multiplier Structure

(a) An intuitive implementation of a 3x3 multiplier

(b) A Verilog implementation of a 3x3 multiplication using the `*` operator

```
module mult3x3 (a, b, y);
    input [2:0] a,b;
    output [5:0] y;
    //do 3x3 multiply
    assign y = a * b;
endmodule
```
Propagation Delay

Propagation Delay (non-inverting)
Busses

Here we require 16 wires and arbitration logic

Arbitration logic **Controls** the flow of data

---

Tri-State Buffer Types

- 3 states instead of 2 (0 and 1)
- 0, 1, z; z is "high impedance" state
- "high impedance" is open circuit

**Type of 3 state buffers**

If-statements to explain behavior only (Verilog HDL)
Tri-state Buffers

(a) Tristate buffer with high-true enable

(b) Half-duplex communication

(c) Bus multiplexing

Example

\[ R_n \rightarrow R_m \] designated by 4-Bit control word \( nm \), ie 0110 means \( R_1 \rightarrow R_2 \)
Making a Design Run Fast

• Speed is usually much more important than saving gates.
• The speed of a gate directly affects the maximum clock speed of a digital system
• Gate speed is TECHNOLOGY dependent
  – 90nm CMOS process has faster gates than 130nm CMOS process
• Implementation choice will affect Design speed
  – A Custom integrated circuit will be faster than an FPGA implementation.
• Design approaches will affect clock speed of system
  – Smart designers can make a big difference

Summary

• Need to review your Digital Logic Design notes
  – Basic Gates, Boolean algebra (algebraic minimization, up to four variable K-maps), Combinational building blocks (muxes, decoders, memories, adders)
• We will discuss Hardware Description Languages
  – Verilog is the language used in the class
• We will discuss modern implementation technologies, primarily Field Programmable Gate Arrays (FPGAs)
• We will discuss design strategies for making designs run faster, not necessarily take less gates.