CAD for semi-custom ASICs

- **ASIC** = application specific integrated circuit
- Semi-Custom = try to design reusing some already designed parts
- **CAD** = flow through a sequence of design steps and software tools.

Spectrum of design approaches

- **Fully custom** means everything done by hand, mostly at the transistor and layout level. Example: microprocessors.
- **Semi-custom** means try to design using existing parts. Example: ethernet chip, hard disk controller.
Example of modern system-on-a-chip IC

- Many big chunks

![Diagram of system-on-a-chip IC](image)

Useful Components in Semi-Custom

- Logic gates
  - Maximally useful components you can reuse
  - Can design without knowing exactly what gates (type, speed, power, size) you have: technology independent design.
  - Later, can map technology independent design onto specific gate library (technology): technology mapping problem.

- Memories
  - Module generator transforms specs on size (bits, words, speeds) into final layout.
  - Very structured designs.

- Datapaths
  - Well structured (adders, multipliers)
  - Often designed at gate and transistor level
  - Produced by module generators.
Semi-custom ASIC

- Made out of standard cells

Standard cell = one gate (complex)

ASIC CAD Tool Flow

- Behavioral synthesis
- Logic synthesis
- Technology mapping
- Verification, test
- Timing and power estimation
- Partitioning
- Row based layout
- Design rule checking and extraction
High level (behavioral synthesis)

- **Input:**
  - High level description of desired system function, usually as a program in a hardware description language (Verilog, VHDL).

- **Output:**
  - Register transfer level structure: FSMs, logic, ALUs, memory, busses.

Logic synthesis

- **Input:**
  - Boolean equations, state diagrams, etc.

- **Output:**
  - Gates and connections, called netlist, a structural design.
Technology mapping

- Input:
  - Technology independent gate level design (un-commited design)

- Output:
  - Gate level design using specific technology library.

Formal verification

- Input:
  - A specification for a design (Boolean eqns) and an implementation

- Output:
  - Decision yes/no: is specification == implementation
Timing estimation

- **Input:**
  - A gate level design, timing info about gates and wires

- **Output:**
  - Delay estimate – critical path length

---

Convergence problems between synthesis and layout

Gate network designed without real knowledge of wire delays

Design spec

Logic synthesis

Row-based layout

Failure

After layout timing violated due to wires
Incompletely specified functions

For incompletely specified function \( f \) we build 3 completely specified functions: \( f_{on} \), \( f_{dc} \), \( f_{off} \).

- \( f_{on} \) is the same as the on-set of \( f \).
- \( f_{off} \) is the same as the off-set of \( f \).
- \( f_{dc} \) is the same as the DC-set of \( f \).

\[ f \cup d \cup r \] is a tautology.

Motivation

- Commercial success - used almost everywhere VLSI is done.
- More general treatment of discrete functions of discrete value variables.
- Body of useful and general techniques - can be applied to other areas.
- Foundation for:
  - combinational and sequential synthesis
  - testing
  - timing and false paths
  - formal verification
  - optimal clocking schemes
  - power estimation
  - general combinatorics.
Outline of the class

- Introduction
- 2-level combinational circuits
- Binary decision diagrams
- Synthesis of multi-level circuits
- Technology mapping
- Delay in multi-level circuits
- Testability of multi-level circuits
- Boolean matching
- Automatic test pattern generation techniques in logic synthesis

Grading

- Homework assignments : 20%
- Final project : 70%
- Class presentation of the project : 10%

- You need to do one project for both 256b and 256d.
Texts

■ Suggested books:

Logic Synthesis

■ Goal:
  ■ Map a high level functional description of logic function into a set of primitives in a given technology.

■ Automation:
  ■ Predominantly for random logic

■ Automatic logic synthesis
  ■ Functional design (functional specification of the system, transformed into a logic description in terms of Boolean variables)
  ■ Logic design (manipulation of the logic representation without modification of functionality).
Physical design

- Custom (macro-cells)
  - high performance, highly optimized designs
- Standard cells
- Gate arrays
- Field programmable gate arrays
- Between macro cells and standard cell: algorithmically generated macros produced by module generators.
  - PLA: effective for designing combinational circuits
  - ROM: look-up table (large Si area)

2-level functions

- PLA are the most popular structures for implementation of 2-level logic functions.
Optimization steps for PLA

- **Logic optimization**: reduction of the number of product terms needed to implement the given function.
- **Topological**: elimination of unused space; folding and partitioning.
- **Layout and circuit optimization**: optimal sizing and placement of drivers, devices and lines.
- **Up to the definitions of the device and interconnect location, PLA is independent of implementation technology.**

**Advantages:**
- Regular structure, easy to automate
- Minimization is well understood

**Disadvantages:**
- No shape control
- Little control of speed
- Little control of I/O placement

---

PLA (2-level) vs Multi-level

<table>
<thead>
<tr>
<th>PLA (2-level)</th>
<th>Multi-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well developed</td>
<td>Relatively undeveloped</td>
</tr>
<tr>
<td>Technology independent</td>
<td>?</td>
</tr>
<tr>
<td>Multi-valued</td>
<td>?</td>
</tr>
<tr>
<td>Control logic</td>
<td>All (control and data flow)</td>
</tr>
<tr>
<td>Constrained layout</td>
<td>Flexible, no layout style</td>
</tr>
<tr>
<td>Automatic layout</td>
<td>+ Automatic layout</td>
</tr>
<tr>
<td>Mature</td>
<td>Not so</td>
</tr>
</tbody>
</table>
The Boolean $n$-cube

- $B = \{0, 1\}$
- $B^2 = \{0, 1\} \times \{0, 1\} = \{00, 01, 10, 11\}$

Basic definitions

- $B = \{0, 1\}$, $Y = \{0, 1, 2\}$, a logic function $f: B^m \rightarrow Y^n$
  - $x \in B^n$ is an input, $y \in Y^m$ is an output.
  - $2$ - don't care value
  - $f$ - incompletely specified function
  - $f$ - a completely specified function
  - $f = (f_1, f_2, \ldots, f_m)$

- $\forall i: 1 \leq i \leq m$:
  - **On-set:** $X_i^{on} \subseteq B^n$: such $x$ that $f_i(x) = 1$
  - **Off-set:** $X_i^{off} \subseteq B^n$: such $x$ that $f_i(x) = 0$
  - **Don't care set:** $X_i^{dc} \subseteq B^n$: such $x$ that $f_i(x) = 2$

- $m=1$: single output function
- $m>1$: multiple output function
Example

- **Tabular representation**

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y1</th>
<th>Y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</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>0</td>
<td>1</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>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ X^{on} = \{(0,0,0),(0,0,1),(1,0,0),(1,0,1),(1,1,0)\} \]

\[ X^{off} = \{(0,1,0),(0,1,1)\} \]

\[ X^{oc} = \{(1,1,1)\} \]

Boolean functions

\[ f(x) : B^n \rightarrow B \]

\[ B = \{0, 1\}, \ x = (x_1, x_2, \ldots, x_n) \]

- Each vertex of B^n is mapped to 0 or 1.
- The onset of f is \( \{x \mid f(x) = 1\} = f^1 = f^{-1}(1) \)
- The offset of f is \( \{x \mid f(x) = 0\} = f^0 = f^{-1}(0) \)
- If \( f^1 = B^n \), f is the tautology.
- If \( f^0 = B^n \), f is not satisfiable.
- If \( f(x) = g(x) \) for all \( x \) in B^n, then f and g are equivalent.
- \( x_1, x_2, \ldots \) are variables
- \( x_1, x_1', x_2, x_2' \ldots \) are literals
### Literals

- A literal is a variable or its negation: $y$, $y'$.
- It represents a logic function.
  - Literal $x_1$ represents the logic function $f$, where $f = \{ x | x_1 = 1 \}$
  - Literal $x'_1$ represents the logic function $g$, where $g = \{ x | x_1 = 0 \}$

### Boolean formulas

- Boolean functions can be represented by formulas defined as catenations of:
  - parentheses: $(x,y)$
  - literals: $x, y, z, x', y', z'$
  - Boolean operations: $+$ (or), $*$ (and)
  - complementations: $(x+y)'$

- Examples:
  - $f = x_1 \cdot x_2' + x_1' \cdot x_2 = (x_1+x_2) \cdot (x_1' + x_2')$
  - $h = a + b\cdot c = (a' \cdot (b' + c'))'$
  - We will usually replace $*$ by catenation, e.g. $a^*b \rightarrow ab$. 

---

256d 27

256d 28
Operations on Boolean functions

Multiple output functions: the usual Boolean operations are performed component-wise on the outputs.

A complement of \( f : B^n \rightarrow B^m \) is a function \( \overline{f} : B^n \rightarrow B^m \) such that \( \overline{f_1}, \overline{f_2}, \cdots, \overline{f_m} \) have their on-sets equal to the off-sets of \( f \).

\[
\begin{array}{c|c|c}
\text{on} & \text{off} & \text{on} \\
\hline
f & & T \\
\end{array}
\]

The intersection: \( h = f \cdot g \) \((f \cap g) : h_i \) has an on-set equal to the intersection of the on-sets of \( f_i \) and \( g_i \).
The difference: \( h = f - g \) \((f \# g) = f \cap \overline{g}\)

The union: \( h = f + g \) \((f \cup g)\)

The tautology: off set is empty.
Incompletely specified functions

For incompletely specified function \( ff \) we build 3 completely specified functions: \( ff_{on} \), \( ff_{dc} \), \( ff_{off} \).

\[
\begin{align*}
\text{On-set the same as On-set of } ff \\
\text{On-set the same as Off-set of } ff \\
\text{On-set the same as DC-set of } ff
\end{align*}
\]

\( f \cup d \cup r \) is a tautology

Algebraic representation

\( ff = (ff_1, ff_2, \ldots, ff_m) \). \( f_i \) is an algebraic representation of \( ff_i \) if it is a Boolean expression that evaluates to 1 for all inputs in \( X_i^{ON} \), to 0 for all inputs of \( X_i^{OFF} \), and either to 0 or 1 for all inputs in \( X_i^{DC} \).

Algebraic representation of \( ff \) is denoted by \( f, f(ff) \).
Example

\begin{align*}
\text{Can be simplified} \\
f_1 &= \bar{x}_2 + x_1 \bar{x}_3 \\
f_2 &= x_2 + x_1 \bar{x}_3 \\
(f_1 &+ f_2) &= x_1 \bar{x}_3 + \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3 + \bar{x}_1 x_2 x_3 + x_1 x_2 x_3 \\
\text{(Sum of products form)}
\end{align*}

Each product term in the sum of products algebraic representation of f determines a logic function.

\begin{align*}
\bar{x}_2 &\quad x_1 \quad x_2 \quad x_3 \\
x_1 \bar{x}_3 &\quad x_1 \quad x_2 \quad x_3 \\
\begin{array}{c|c|c|c|c|c|c|c|c|c}
\hline
x_2 & x_1 & x_2 & x_3 & x_1 \bar{x}_3 & x_1 & x_2 & x_3 & 2-D cube & 1-D cube \\
\hline
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
\end{align*}
Cubes

P - a product term in an algebraic sum of products expression of a logic function of n inputs and m outputs

A cube p is specified by \( C = [c_1, c_2, \cdots, c_{n+m}] \)

\[
C_i = \begin{cases} 
0 & \text{if } x_i \text{ appears complemented in } p \\
1 & \text{if } x_i \text{ appears not complemented in } p \\
2 & \text{if } x_i \text{ does not appear in } p \\
3 & \text{if } p \text{ is not present in algebraic representation of } f_{i-n} \\
4 & \text{if } p \text{ is present in the algebraic representation of } f_{i-n} 
\end{cases} 
\]

Example:

\[
f_1 = \overline{x_2} + x_1 \overline{x_3} \\
f_2 = x_2 + \overline{x_1} \overline{x_3} \\
P = x_2 \quad C = [2, 0, 2, 4, 3]
\]

Input cube = compact form of the coordinates of the vertices of the cube corresponding to the product term.

Example: \( I(c) = [2, 0, 2] \) represents \((1,0,1), (0,0,0), (1,0,0)\) and \((0,0,1)\).

\( O(c) = [4, 3] \) identifies the space where the cube belongs.

\[ C = \{c^1, c^2, \cdots, c^k\} \text{ is a cover of } f \text{ with } n \text{ inputs and } m \text{ outputs, if for } j=1,2,\ldots,m \text{ the set of input parts of the cubes that have a 4 in the } j\text{-th position contain all the vertices corresponding to the on-set of } f_{i-j} \text{ and none of the off-set of } f_{i-j}, \text{ i.e. a cover represents a union of the on-set and some arbitrary position of the don’t cares.} \]

There is a 1-1 correspondence between a cover and an algebraic representation of a function as a sum-of-products.
A matrix representation of a cover:

\[ M(C) = \begin{bmatrix} 2 & 0 & 2 & 4 & 3 \\ 1 & 2 & 0 & 4 & 3 \\ 2 & 1 & 2 & 3 & 4 \\ 0 & 2 & 0 & 3 & 4 \end{bmatrix} \]

Example: 

\[ f_1 = \overline{x}_2 + x_1 \overline{x}_3 \]
\[ f_2 = x_2 + x_1 \overline{x}_3 \]

\[ G = I(M(C)) \quad \text{input matrix} \]
\[ H = O(M(C)) \quad \text{output matrix} \]

Matrix representation and cover are used interchangeably.
If \( C \) is a cover of a single output function, then \( H = \emptyset \)

Let \( c = [c_1, c_2, \ldots, c_{n+m}] \) and \( d = [d_1, d_2, \ldots, d_{n+m}] \) be 2 cubes.

The cube \( c \) contains \( d \) if:

the cube represented by the input part of \( c \) contains all the vertices of \( d \);
and must be present in all Boolean spaces where \( d \) is present.

A minterm \( e^i \) is a cube whose input part does not contain any 2s and whose output part contains \((m-1)\) 3s and one 4 in position \( l \).

The input cube is a vertex and this vertex is present only in one, \( l \)-th Boolean n-space. A minterm \( e^i \) does not contain any other cube. If a cube contains a minterm \( e^i \) we say that \( e^i \) is an element of \( c \).

Example: \([1, 1, 4, 3]\) is a minterm and an element of \([2, 2, 1, 4, 4]\).

Each cube can be decomposed into a set of all minterms that are elements of the cube.
Example

c = [2, 2, 1, 4, 4]

\[ e_1^1 = [0, 0, 1, 4, 3] \]
\[ e_3^2 = [0, 0, 1, 3, 4] \]
\[ e_4^2 = [1, 0, 1, 4, 3] \]
\[ e_5^3 = [0, 1, 1, 4, 3] \]
\[ e_6^3 = [0, 1, 1, 3, 4] \]
\[ e_7^4 = [1, 1, 1, 4, 3] \]
\[ e_8^4 = [1, 1, 1, 3, 4] \]

A set of cubes \( C = [c^1, c^2, \cdots, c^{m+n}] \) covers a cube \( c \subseteq C \) if each of the minterms of \( c \) is covered by at least one cube of \( C \).

Special cubes:

- \( u^j \) - the universe of the \( j \)-th Boolean space

\[ u^j = \frac{1,2, \ldots, j, n, n+1, \ldots, n+j, \ldots, n+m}{2, \ldots, 2, 3, \ldots, 3, \ldots, 3} \]

\[ U = \frac{1,2, \ldots, j, n, n+1, \ldots, n+j, \ldots, n+m}{2, \ldots, 2, 4, \ldots, 4, \ldots, 4, 4} \]

- \( x^j \): the positive half-space of \( x^j \)

De Morgan’s law:

\[ C = [c^1, c^2, \cdots, c^{m+n}] \]

1. Express the cover in algebraic form

2. Exchange AND and OR

3. Change variables to complements

Example

\[ f = x_1 \overline{x}_3 + \overline{x}_2 x_4 + x_1 x_2 \overline{x}_4 \]
\[ \overline{f} = (\overline{x}_1 + x_3)(\overline{x}_2 + x_4)(\overline{x}_1 + \overline{x}_2 + x_4) \]

Multiply out using rules of Boolean algebra:

\[ x \overline{x} = 0 \]
\[ x x = x \]
\[ x + \overline{x} = 1 \]
Intersection or a product of 2 cubes

\[ c \cap d \]

<table>
<thead>
<tr>
<th>( i )</th>
<th>( d_i )</th>
<th>( c_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

\( \emptyset \) is an empty cube

If an output part of a cube has all 3 it is empty.

Intersection: input part corresponds to the vertices that are common to \( c \) and \( d \). Output part specifies that the cube is present in the Boolean \( n \)-spaces in which both \( c \) and \( d \) are present.

If 2 cubes have no common vertices or no common Boolean space:

\[ c \cap d = \emptyset, \text{c and d are orthogonal.} \]

\[ f \cap \bar{f} = \emptyset \]

The union of 2 cubes: \( c \cup d \) (c+d): the set of vertices covered by the input part of either \( c \) or \( d \) in the Boolean \( n \)-space where they are present.

In matrix representation: \( c \cup d \) is the matrix formed by 2 rows corresponding to \( c \) and \( d \), respectively.

The distance between 2 cubes: \((\# \text{ of conflicts})\)

\[ \delta(c, d) = \delta(I(c), I(d)) + \delta(O(c), O(d)) \]

where

\[ \delta(I(c), I(d)) = |\{ j | c_j \cap d_j = \emptyset \}| \]

\[ \delta(O(c), O(d)) = \begin{cases} 
0 & \text{if } c_j \cap d_j = 4 \quad \text{Some } j > n \\
1 & \text{otherwise}
\end{cases} \]
The consensus of 2 cubes: \( e = c \Theta d \)

If \( \delta(c,d) \neq 1 \) then \( e = \begin{cases} c \cap d & \text{if } \delta(c,d) = 0 \\ \emptyset & \text{if } \delta(c,d) \geq 2 \end{cases} \)

If \( \delta(I(c),I(d)) = 1 \land \delta(O(c),O(d)) = 0 \) then

\[
e_i = \begin{cases} c \cap d_i & \text{if } c \cap d_i \neq 0 \\ 2 & \text{otherwise} \end{cases}
\]

If \( \delta(I(c),I(d)) = 0 \land \delta(O(c),O(d)) = 1 \)

\[
e_i = \begin{cases} c \cap d_i & 1 \leq l \leq n \\ 4 & \text{if } c_i \text{ or } d_i = 4 \text{ for } n < l \leq n + m \\ 3 & \text{otherwise} \end{cases}
\]

Theorem: The consensus of 2 cubes \( a \) and \( b \), \( p = a \Theta b \) is contained in \( a \cup b \). If \( a \Theta b \neq \emptyset \), it contains minterms of both \( a \) and \( b \). \( p \) is the largest cube contained in \( a \cup b \).

\( \forall x, \exists y, x, y \in p, x \in a, y \in b. \)
The complement of a set of cubes $C$, $\overline{C}$ covers the complement of logic corresponding to $C$.

The difference: $C-H$ covers $C \cap \overline{H}$.

A cube is an implicant of $\text{ff}=(f,d,r)$ if it has an empty intersection with the cubes of a representation of $r$.

Example.

\[
F=M(C) = \begin{bmatrix} 2 & 0 & 2 & 4 & 3 \\ 1 & 2 & 0 & 4 & 3 \\ 2 & 1 & 2 & 3 & 4 \\ 0 & 2 & 0 & 3 & 4 \end{bmatrix}
\]

$(1,2,0,4,3)$ is an implicant of $\text{ff}$. $(0,2,1,3,4)$ is not since it contains $(0,0,1)$ in the Boolean space representing $\text{ff}$ that is in the off-set of $\text{ff2}$. 