ECE 565: VLSI CAD Flow + Brief Intro to HLS, Logic Optimization & Technology Mapping

Shantanu Dutt
ECE Dept., UIC
VLSI CAD Flow: Overview

- High-Level Synthesis or HLS → network of interconnected modules: functional units [FUs], registers, muxes, demuxes, etc.

- Logic optimization (mainly of controller fsm’s and other “random” logic; other module designs such as arith. FUs, regs, muxes, etc. are well known).

- System Specification

- Architectural Design (HLS)

- Functional Design and Logic Design

- Circuit Design

- Physical Design

- Physical Verification and Signoff

- Fabrication

- Packaging and Testing

- Chip

- Design Rule Checking
- Layout Vs. Schematic Verification
- Electrical Rule Checking
- Electrical simulation and functional (logic) + metric (delay, power) verification.

- (E.g., Using a Hardware Description Language: Verilog, VHDL, SystemC or Schematic Capture)

- Iterate if simulation identifies metric problems (hot spots, timing violation, crosstalk)

Shantanu Dutt, UIC
System to Silicon Design

System Requirements

Algorithm

\[ X[k] = \sum x[n]e^{j2\pi k/N} \]
\[ x[n] = \sum X[k]e^{j2\pi k/N} \]

Hardware Architecture

Synthesis

System Integration

Fabricate and Test

Physical

Design For Test

[Source: MITRE]
VLSI CAD Flow: Overview

System Specification

Architectural Design (HLS)

Functional Design and Logic Design

Circuit Design

Physical Design

Physical Verification and Signoff

Fabrication

Packaging and Testing

Chip

Constrained optimization problem: Opt. metric at almost all stages: One of power, delay, chip-area, total wire length (WL) w/ constraints (upper bounds) on the others plus on temperature (approx. power density), crosstalk (routing stage), yield/variability (lower bound), etc. ➔ Very complex processes.

Disconnect between higher (above physical design) and lower stages (PD & lower): At higher stages some metrics such as those that are fully or partly interconnect-based (delay, area, dynamic power) cannot be v. accurately estimated ➔ incorrect decisions which percolate to lower stages (e.g. after routing between an adder and mult., interconnect length is much larger than anticipated due to say congestion and mis-routing) ➔ some specs (esp. delay) may not be met ➔ iterating back to higher stages to correct the approximations and re-design

Shantanu Dutt, UIC
VLSI CAD Flow: Overview

System Specification

- Architectural Design (HLS)
- Functional Design and Logic Design
- Circuit Design

Physical Design

Physical Verification and Signoff

- Fabrication
- Packaging and Testing
- Chip

Partitioning

Floorplanning

Placement

Clock Tree Synthesis

Signal Routing

Timing Closure

Iterate if simulation identifies metric problems (hot spots, timing violation, crosstalk)

Course Coverage: with particular focus on HLS and Physical Design

Shantanu Dutt, UIC
VLSI CAD Flow: Overview

Partitioning cuts up the system into multiple subsystems to minimize some metric (generally number of wires/nets cut—min-cut part.) either for multi-chip impl. or for recursive invocation for placement on a single chip.

Floorplanning places major/large polygon-shaped modules on a chip to minimize chip area, wirelength (WL), delay or dynamic power with constraints on the others.

Placement places smaller rectangular cells whose size/shape are well defined in sub-chip regions for the same goals as FP.

Routing interconnects cells/modules via horizontal/vertical or 45 deg. wires for similar goals as above.

Iterate if simulation identifies problems (hot spots, timing violation, crosstalk)
**Standard Cell Place and Route Flow**

- **Netlist**
- **Placement**
- **DEF**
- **Route**
- **DEF**
- **Parasitic Extract**
- **ESPF**
- **Timing Analysis**
- **Library SDF**
- **Timing OK**
- **Noise and Reliability**
- **OK**
- **Tool error**
- **Database to Manufacturer**

Some modules are custom designed & laid-out (e.g., RAMs), while others are automated using, say, standard cells.
Another complete chip showing RAMs (custom design) + standard-cell layout

Shantanu Dutt, UIC
What HLS Produces by Examples

Shantanu Dutt
UIC
Hardware Synthesis – Example 1

Programming language statement

\[ a := b + c; \]

Computer Code Generation

Hardware Synthesis

Execution in a CPU

Load r5 b
Load r7 c
ADD r2 r5 r7
(r2 <- r5 + r7)
Store r2 a

CPU

Register File

r2

r5

r7

Mux

Mux select

Reg r/w

Reg addr

ALU

ADD

32 Read Bus A

32 Read Bus B

32

Write Bus

Contol signals

Mux

Mux1 select

Mux2 select

Mux3 select

32

Contol signals

b

c

adder

a

ldb

ldc

lda

Shantanu Dutt
Hardware Synthesis – Example 2

Programming language construct

```plaintext
if (a <= b) then begin
  Block A of code;
end
else begin
  Block B of code;
end
```

Computer Code
Generation

Hardware Synthesis

<table>
<thead>
<tr>
<th>B:</th>
<th>Block B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load r2 a</td>
<td></td>
</tr>
<tr>
<td>Load r3 b</td>
<td></td>
</tr>
<tr>
<td>SUB r2 r2 r3</td>
<td></td>
</tr>
<tr>
<td>BZ A</td>
<td></td>
</tr>
<tr>
<td>BNEG A</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A:</th>
<th>Block A</th>
</tr>
</thead>
<tbody>
<tr>
<td>assmb code</td>
<td></td>
</tr>
<tr>
<td>JMP C</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C:</th>
<th>Block C</th>
</tr>
</thead>
<tbody>
<tr>
<td>assmb code</td>
<td></td>
</tr>
</tbody>
</table>

Execution in a computer

Computer

I/Ps to code block

Demux

Adder

Operations

(Mux may be reused for other operations)

Control signals

Control signals

Control signals

Hardware for Block B

Hardware for Block A

Recursively Synthesized

Mux

O/Ps of code block

Shantanu Dutt

UIC 18
Brief Intro to Logic Optimization & Technology Mapping

Extracted from Notes by:
Srinivas Devdas, MIT
Two-Level Logic Minimization

Can realize an arbitrary logic function in sum-of-products or two-level form

\[ F_1 = \overline{A} \overline{B} + \overline{A} B D + \overline{A} B \overline{C} \overline{D} + A B C \overline{D} + A \overline{B} + A B D \]

\[ F_1 = \overline{B} + D + \overline{A} \overline{C} + A C \]

Of great interest to find a minimum sum-of-products representation

– Solved problem even for functions with 100’s of inputs (variants of Quine-McCluskey)
Two-Level versus Multilevel

2-Level:

\[ f_1 = AB + AC + AD \]
\[ f_2 = \overline{AB} + \overline{AC} + \overline{AE} \]

6 product terms which cannot be shared.
24 transistors in static CMOS

Multi-level:

Note that \( B + C \) is a common term in \( f_1 \) and \( f_2 \)

\[ K = B + C \]
\[ f_1 = AK + AD \]
\[ f_2 = \overline{AK} + \overline{AE} \]

3 Levels
20 transistors in static CMOS
not counting inverters
Technologies

“Closed book”: gate-array, standard-cell

“Open book”: CMOS Domino, complex gate static CMOS

LOGIC EQUATIONS

TECHNOLOGY-INDEPENDENT OPTIMIZATION

Factoring
Commonality Extraction

TECH-DEPENDENT OPTIMIZATION
(MAPPING, TIMING)

LIBRARY

OPTIMIZED LOGIC NETWORK
Tech.-Independent Optimization

Involves:
Minimizing two-level logic functions.
Finding common subexpressions.
Substituting one expression into another.
Factoring single functions.

Factored versus Disjunctive forms

\[ f = ac + ad + bc + bd + a\bar{e} \]
sum-of-products or disjunctive form

\[ f = (a + b)(c + d) + a\bar{e} \]
factored form
multi-level or complex gate
Optimizations

\[ F = \begin{cases} 
    f_1 = AB + AC + AD + AE + A\overline{BCD}E \\
    f_2 = \overline{AB} + \overline{AC} + \overline{AD} + \overline{AF} + A\overline{BCD}F 
\end{cases} \]

**Factor** \( F \)

\[ F = \begin{cases} 
    f_1 = A(B + C + D + E) + ABCDE \\
    f_2 = \overline{A}(B + C + D + F) + \overline{ABCDF} 
\end{cases} \]

**Extract common expression**

\[ G = \begin{cases} 
    g_1 = B + C + D \\
    f_1 = A(g_1 + E) + \overline{A}Eg_1 \\
    f_2 = \overline{A}(g_1 + F) + \overline{A}Fg_1 
\end{cases} \]
What Does “Best” Mean?

Transistor count → AREA
Number of circuits → POWER
Number of levels → DELAY (Speed)

Need quick estimators of area, delay and power which are also accurate
Tech.-Dependent Optimization

Optimized Logic Equations

Technology Mapping

Gate Netlist

Area, delay and power dissipation cost functions
A standard cell technology or library is typically restricted to a few tens of gates. For example, the MSU library contains 31 cells.

Gates may be NAND, NOR, NOT, AOIs.
Mapping via DAG Covering

Represent network in canonical form
⇒ subject DAG

Represent each library gate with canonical forms for the logic function
⇒ primitive DAGs

Each primitive DAG has a cost

Goal: Find a minimum cost covering of the subject DAG by the primitive DAGs

Canonical form: 2-input NAND gates and inverters
# Sample Library

<table>
<thead>
<tr>
<th>Function</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVERTER</td>
<td>2</td>
</tr>
<tr>
<td>NAND2</td>
<td>3</td>
</tr>
<tr>
<td>NAND3</td>
<td>4</td>
</tr>
<tr>
<td>NAND4</td>
<td>5</td>
</tr>
</tbody>
</table>

![Sample Library Diagram](image)
Sample Library - 2

AOI21 4

AOI22 5
Trivial Covering

subject DAG

\[
\begin{align*}
7 & \quad \text{NAND2} = 21 \\
5 & \quad \text{INV} = 10 \\
31 & \\
\end{align*}
\]
Covering #1

2 INV  =  4
2 NAND2 =  6
1 NAND3 =  4
1 NAND4 =  5

19
Covering #2

1 INV = 2
1 NAND2 = 3
2 NAND3 = 8
1 AOI21 = 4

17
DAG Covering

Sound Algorithmic approach
NP-hard optimization problem

Tree covering heuristic: If subject and primitive DAGs are trees, efficient algorithm can find optimum cover in linear time
⇒ dynamic programming formulation
Partitioning a Graph
Resulting Trees

Break at multiple fanout points
Dynamic Programming

Principle of optimality: Optimal cover for a tree consists of a match at the root of the tree plus the optimal cover for the sub-trees starting at each input of the match.

Best cover for this match uses best covers for x, y, z

Best cover for this match uses best covers for p, z
Optimum Tree Covering

INV
11 + 2 = 13

NAND2
2 + 6 + 3 = 11

INV
3 + 3 = 6

AOI21
4 + 3 = 7
DP Algorithm for Tech. Mapping of Tree Circuits

Procedure $DP_{TM}(p: \text{gate output}, G: \text{circuit})$;

if $DP_{TM}(p, G)$ is “stored” then return(its cost, corresp. solution);

Going backward from $p$ to its fanins, produce all possible “cuts” (that separate the sub-ckt w/ $p$ as o/p from the rest of the circuit) so that the # of wires cut $\leq$ max # of i/ps available for cells/gates in the library;

cost = infinity;

For each such cut $C_i$ do

if the subckt containing $p$ is mappable to some gate $g_i$ in the library then begin

cost($C_i$) = cost($g_i$) + $\sum_{q_j \in C_i} (DP_{TM}(q_j, G)[1])$; /* if $DP_{TM}(q_j, G)$ has already been investigated, it will be “stored”, and there is no need to execute this procedure again */

if cost($C_i$) < cost then { cost = cost($C_i$); solution = corresponding soln. }

end if.

End for;

return(cost, solution);

end $DP_{TM}$;
Ci' is the set of gate o/p's that are in the “complement” (remainder) subckt. of G generated by cut Ci

Recursive Calls for DP_TM

Ci

Ci’ is the set of gate o/p's that are in the “complement” (remainder) subckt. of G generated by cut Ci

Example Cuts in DP_TM
Analysis:

• Optimality?

• Runtime analysis:
  – # of subsets that can be generated from a gate g’ o/p that includes g and includes fan-ins only up to a fan-in size of m (= max. number of i/ps among all cells in the library)
  – # of 1st time calls of DP_TM for each gate o/p
  – Total # of DP_TM calls

• What if the circuit is not a tree: how to process, optimal or non-optimal?

• How to obtain optimality for a general DAG and at the cost of how much increased runtime?