Schedule

10. 04/27/17 Chapter 2 (CMP & Design Rule)
11. 05/04/17 Chapter 3 (Logic Gates, Noise Margin, power)
12. 05/11/17 Chapter 3 (Fan-out and loading, timing)
13. 05/18/17 Chapter 4 (Simulation, Cross Talk)
14. 05/25/17 Quiz 2, Chapter 4 (ATPG & DFT)
15. 06/01/17 Chapter 5 (Memory types, Set-up & hold time)
16. 06/08/17 Fin-FET
17. 06/15/17 Final Examination
18. 06/22/17 Exam review and more
Chapter 3: Logic Gates

Combinational logic functions
Static complementary logic gate structures
Combinational logic expressions

Combinational logic: function value (outputs) is a combination of inputs (and inputs only).
A logic gate implements a particular logic function.
Both specification (logic equations) and implementation (logic gate networks) are written in Boolean logic.
Gate design

Why designing gates for logic functions is non-trivial:

- may not have logic gates in the library for all logic expressions;
- a logic expression may map into gates that consume more area, delay, or power.
Boolean algebra terminology

Function:
\[ f = a'b + ab' \]

a is a variable; \( ab' \) is a term.

A function is irredundant if no term can be removed without changing its truth value.
Static complementary gates

Complementary: have complementary pullup (p-type) and pulldown (n-type) networks.
Static: do not rely on stored charge.
Simple, effective, reliable; hence ubiquitous.
Examples (1)

\[ f = (a+b) \times (a+b') \]
\[ = a \times a + a \times b' + b \times a + b \times b' \]
\[ = a + a \times b' + b \times a \]
\[ = a + a \times (b' + b) \]
\[ = a + a \]
\[ = a \]
Examples (2 & 3)

\[ f = a \cdot d + a \cdot e + b \cdot d + b \cdot e + c \cdot d + c \cdot e \]
\[ = (a+b+c) \cdot (d+e) \]

\[ f = a \cdot c \cdot e + a \cdot d \cdot e + b \cdot c \cdot e + b \cdot d \cdot e + a \cdot c \cdot f + a \cdot d \cdot f + b \cdot c \cdot f + b \cdot d \cdot f \]
\[ = (a \cdot c + a \cdot d + b \cdot c + b \cdot d) \cdot (e+f) \]
\[ = (a+b) \cdot (c+d) \cdot (e+f) \]
Examples (4)

\[ f = a + b \cdot a' + a' \cdot b' \]
\[ . \quad = a + a' \cdot (b + b') \]
\[ = a + a' \]
\[ = 1 \]
Completeness

A set of functions \( f_1, f_2, \ldots \) is complete iff every Boolean function can be generated by a combination of the functions.

NAND is a complete set; NOR is a complete set; \{AND, OR\} is not complete.

Transmission gates are not complete.

If your set of logic gates is not complete, you can’t design arbitrary logic.
Static complementary gate structure

Pullup and pulldown networks:

- Pullup network (V_DD)
- Pulldown network (V_SS)
- Inputs
- Outputs
- P trs
- N trs
Inverter

<table>
<thead>
<tr>
<th>a</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Inverter Layout

metal1

metal2

pdiff

metal1-diff via

GND

Out

In

metal1-poly via

copolysilicon

V_{DD}

PMOS (4/.24 = 16/1)

NMOS (2/.24 = 8/1)

diff

metal2-metal1 via

VLSI Design: Chapter 3
Inverter

\[ V_{OL} = 0 \]
\[ V_{OH} = V_{DD} \]
\[ V_M = f(R_n, R_p) \]

- \( V_{out} = V_{dd} \quad V_{in} = 0 \)
- \( V_{out} = GND \quad V_{in} = V_{DD} \)
NMOS transistor, 0.25um, $L_d = 0.25\text{um}$, $W/L = 1.5$, $V_{DD} = 2.5\text{V}$, $V_T = 0.4\text{V}$
Inverter Layout

**PLATE 8:** Layout examples of CMOS inverters and simple logic gates.
NAND gate

*a串*
+*並*

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>

\[ a \land b = \overline{\text{out}} \]
Layout of a NAND Gate
NOR gate

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ a + b = \text{out} \]
Layout of a NOR Gate
NOR & NAND

**PLATE 8:** Layout examples of CMOS inverters and simple logic gates.
AOI/OAI gates

AOI = and/or/inverter; OAI = or/and/inverter. Implement larger functions.

Pullup and pulldown networks are compact: smaller area, higher speed than NAND/NOR network equivalents.
out = \([ab+c]'\)

symbol

invert

or

and

AOI example

circuit
Pullup/pulldown dual network

Pullup and pulldown networks are duals. To design one gate, first design one network, then compute dual to get other network.
\[ O = A \cdot B \cdot C \cdot D \]
4-input NAND
Try this one

\[ O = D + A \cdot (B + C) \]
Try this one

$$O = D + A \cdot (B + C)$$
z = c \cdot (a + b)
Try again

\[ z = c \cdot (a + b) \]
# Gate to Logic K-Map

## Truth table

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Output</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>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
</tbody>
</table>

## Karnaugh Map

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

<table>
<thead>
<tr>
<th>X</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xy’</td>
<td>xy</td>
</tr>
<tr>
<td>1</td>
<td>x’y’</td>
<td>x’y</td>
</tr>
</tbody>
</table>
K-Map

\[ xyz' + xyz + x'y'z + xy'z \]

\[ = (xyz' + xyz) + (x'y'z + xy'z) + (xy'z + xyz) \]

\[ = (xy \cdot (z' + z)) + (y'z \cdot (x' + x)) + (xz \cdot (y' + y)) \]

\[ = (xy \cdot 1) + (y'z \cdot 1) + (xz \cdot 1) \]

\[ = xy + y'z + xz \]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Output</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</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>
x’y’z’w’ + x’y’z’w’ + xy’z’w’ + xy’z’w + ….. and statements

= wx + y’z’ + xz +x’z’
Logic Level (Gate Level)

MOS Level

Layout

Cross-section

Process
Chapter 3: Logic Gate

Electrical properties of static combinational gates.
Effects of parasitics on gate.
Driving large loads.
Transfer characteristics

Transfer curve shows static input/output relationship—hold input voltage, measure output voltage.
Inverter transfer curve
Choose threshold voltages between points where slope of transfer curve = -1.

Inverter has a high gain between $V_{IL}$ and $V_{IH}$ points, low gain at outer regions of transfer curve.

Note that logic 0 and 1 regions are not equal sized—in this case, high pullup resistance leads to smaller logic 1 range.
Noise margin

Noise margin = voltage difference between output of one gate and input of next. Noise must exceed noise margin to make second gate produce wrong output.

In static gates, voltages are $V_{OH} = V_{DD}$ and $V_{OL} = V_{SS}$, so noise margins are $V_{DD} - V_{IH}$ and $V_{IL} - V_{SS}$.
Solid logic 0/1 defined by $V_{SS}/V_{DD}$. Inner bounds of logic values $V_L/V_H$ are not directly determined by circuit properties, as in some other logic families.
Schedule

11. 05/04/17 Chapter 3  (Logic Gates, Noise Margin, power)
12. 05/11/17 Chapter 3  (Fan-out and loading, timing )
13. 05/18/17 Chapter 4  (Simulation, Cross Talk)
14. 05/25/17 Quiz 2, Chapter 4  (ATPG & DFT)
15. 06/01/17 Chapter 5  (Memory types, Set-up & hold time)
16. 06/08/17 Fin-FET
17. 06/15/17 Final Examination
18. 06/22/17 Exam review and more
Logic Level (Gate Level)
MOS Level
Layout
Cross-section
Process

Language
Truth table
Karnaugh Map
Inverter transfer curve

$$V_{out}$$

$$V_{in}$$

$$V_{IL}:$$ slope = -1

$$V_{IH}:$$ slope = -1
Inverter transfer curve

- Logic 1
- Logic 0

- $V_{IL}$: slope = -1
- $V_{IH}$: slope = -1

VLSI Design: Chapter 3
Power Domains
Logic levels
Logic level shifter

Levels at output of one gate must be sufficient to drive next gate.
Shifter
Power consumption analysis

Total Power = \( P_{\text{static}} + P_{\text{dynamic}} \)

\[ P_{\text{dynamic}} = P_{\text{switching}} + P_{\text{short}} \]

Most of power consumption comes from switching behavior.

Static power dissipation comes from leakage currents.
Dynamic power consumption (1)

Switching Power  \[ = IV \]
\[ = (Q/t) V = (CV/t) V \]
\[ = C V^2 f \]

Dynamic power consumption is independent of the p, n transistors’ size
Dynamic power consumption (2)

Short Circuit Power \[ = t_{sc} V_{DD} I_f \]
Static power consumption

Short Circuit Power = $I_{\text{sleakage}} V_{DD}$
Power Breakout List

250nm
Switching: 75%
Short: 20%
Leakage: 5%

90nm
Switching: 32%
Short: 10%
Leakage: 58%

*Source = Intel
Leakage Increased

![Graph showing leakage current and drawn gate length relationship](image)
Assume ideal input (step), RC load.
Delay assumptions

Assume that only one transistor group (p or n) is on at a time. This gives two cases:
rise time, p on, n off;
fall time, p off, n on.

Assume resistor model for transistor.
Ignores saturation region,
⇒ but results are acceptable!!
Inverter delay circuit

Load is resistor + capacitor, driver is resistor.
Inverter delay

\[ V_{\text{out}}(t) = V_{DD}(1-e^{-t/(R_n+R_L)C_L}) \]

\[ t_1 = 0.9 \ V_{DD}; \ t_2 = 0.1 \ V_{DD}; \ t_f = t_2 - t_1; \]
\[ t_f = 2.2 \ (R_n + R_L) \ C_L \]

For pullup time, use p transistor(s) resistance.
Quality of RC approximation

![Graph showing the comparison between transistor and RC responses over time.](image-url)
Parasitics and performance

VLSI Design: Chapter 3
Driving large loads

Sometimes, large loads must be driven:

off-chip;
long wires on-chip;
large fanout number.

Fanout = 3
All gates have the same drive current.

Slope is a function of “driving strength”
Effect of parasitics

Resistance slows down static gates, may cause function failure.

Increase transistor’s size to increase driving, but also increase input capacitance which reduces input slope.
Parasitics and performance

VDD

M1

Cgd12

Vin

Interconnect

Fanout

M2

Cdb2

Vout

Cdb1

M3

Cg3

M4

Cg4

Vout2

Simplified Model

Vin

Vout

CL

Cw
Chapter 3: Logic Gates

Delay
Switch logic.
Sizing up the driver transistors only pushes back the problem—large driver presents larger capacitance to earlier stage.

Use a series of buffers (inverters)
Buffer Sizing 2

\[ n = \frac{\ln\left(\frac{C_L}{C_{in}}\right)}{\ln a}. \]

Fig. 4.35 Chain of inverter stages driving a large load.
Buffer sizing

Use a chain of inverters, each stage has transistors larger than previous stage.

Optimal number of stage \( n_{\text{opt}} = \ln (C_{\text{big}}/C_g) \).
\( n_{\text{opt}} \) must be an even number (for inverter)
Driver sizes increased exponentially.
Buffer sizing

Narrower Width = Lower current through channel

Wider Width = Higher current through channel
Buffer sizing (N-MOS)

"1X" NMOS ($W/L = 6$)

"2X" NMOS ($W/L = 12$)

"2X" NMOS ($W/L = 6 + 6$)
Wire delay

Wires have parasitic resistance, capacitance. Parasitics start to dominate in deep-submicron wires. (70 ~ 80 % of delay comes from wires in deep submicron)

Distributed RC introduces time of flight along wire into gate-to-gate delay.
Wire Delay-2

Fig. 4.17 Diffusion capacitance.
Wire Delay in Deep Sub-Micron

Fig. 4.19 Capacitance between two wires: (a) Before sub-micron; (b) Deep sub-micron.
RC distribution line (L-model)

Assumes that dominant capacitive coupling is to ground, inductance can be ignored.
Elemental values are $r_i$, $c_i$. 

\[ V_{\text{in}} \rightarrow r_1 \rightarrow c_1 \rightarrow r_2 \rightarrow c_2 \rightarrow r_3 \rightarrow \ldots \rightarrow c_n \rightarrow V_{\text{out}} \]
Elmore defined delay through linear network as the first moment of the network impulse response.
Different Models

\[ \begin{array}{c}
\pi_2 \\
\pi_3 \\
\pi_4
\end{array} \]
\( \pi \)-model, 6 \( \pi \)-model

\[ V_{in} - \text{load} + V_{out} \]

\[ C_{load} \]
Delay Calculation

\[ \tau_{Di} = \sum_{k=1}^{N} C_k R_{ik} \]

\[ T_i = C_1 R_1 + C_2 R_1 + C_3 (R_1 + R_3) + C_4 (R_1 + R_3) + C_i (R_1 + R_3 + R_i) \]
RC Elmore delay

Can be computed as sum of sections.
Resistor $r_i$ must charge all downstream capacitors.

From $t_f = 2.2 \ (R_p + R_L) \ C_L$

Where both $R$ and $C$ proportional to the wire length,
$t_f$ proportional to the wire length square.

Delay grows as square of wire length.
Minimizing $rc$ product minimizes growth of delay with increasing wire length.
If the length is 10 unit, then the $t_f$ will be 100 times than a unit length delay.
## Example of a Timing Report

Startpoint: pt100_core/main_dp/src2_oprand_ff/DFFR_31_14/out_reg (rising edge-triggered flip-flop clocked by clk)

Endpoint: cp15/dr_index_ff/DFF_2_0/out_reg (rising edge-triggered flip-flop clocked by clk)

Path Group: clk  Path Type: max  Des/Clust/Port  Wire Load Model  Library

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock clk (rise edge)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>pt100_core/main_dp/src2_oprand_ff/DFFR_31_14/out_reg/CK (QDFZP)</td>
<td>0.00</td>
<td>#</td>
</tr>
<tr>
<td>pt100_core/main_dp/src2_oprand_ff/DFFR_31_14/out_reg/Q (QDFZP)</td>
<td>0.76</td>
<td>f</td>
</tr>
<tr>
<td>pt100_core/arithmetic_unit/src2_oprand[14] (main_dp_test_1)</td>
<td>0.00</td>
<td>0.76 f</td>
</tr>
<tr>
<td>pt100_core/U26/O (BUF4)</td>
<td>0.32</td>
<td>1.08 f</td>
</tr>
<tr>
<td>pt100_core/arithmetic_unit/src2_oprand[14] (arithmetic_unit_test_1)</td>
<td>0.00</td>
<td>1.08 f</td>
</tr>
<tr>
<td>pt100_core/arithmetic_unit/U459/O (INV4)</td>
<td>0.19</td>
<td>1.28 r</td>
</tr>
<tr>
<td>pt100_core/arithmetic_unit/arithmetic_adder/add_360/B[14]</td>
<td>0.00</td>
<td>1.28 r</td>
</tr>
<tr>
<td>(arithmetic_unit_DW01_add_33_3_test_1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pt100_core/arithmetic_unit/arithmetic_adder/add_360/U6/O (OR2)</td>
<td>0.32</td>
<td>1.60 r</td>
</tr>
<tr>
<td>pt100_core/arithmetic_unit/arithmetic_adder/add_360/U218/O (ND4T)</td>
<td>0.36</td>
<td>1.96 f</td>
</tr>
<tr>
<td>pt100_core/arithmetic_unit/arithmetic_adder/add_360/U408/O (INV4)</td>
<td>0.18</td>
<td>2.15 r</td>
</tr>
<tr>
<td>pt100_core/arithmetic_unit/arithmetic_adder/add_360/U464/O (ND2F)</td>
<td>0.21</td>
<td>2.36 f</td>
</tr>
<tr>
<td>pt100_core/arithmetic_unit/arithmetic_adder/add_360/U419/O (INV4)</td>
<td>0.11</td>
<td>2.47 r</td>
</tr>
<tr>
<td>pt100_core/arithmetic_unit/arithmetic_adder/add_360/U280/O (AOI12)</td>
<td>0.19</td>
<td>2.66 f</td>
</tr>
<tr>
<td>cp15/U1229/O (OA12P)</td>
<td>0.29</td>
<td>6.64 r</td>
</tr>
<tr>
<td>cp15/U1399/O (ND2P)</td>
<td>0.10</td>
<td>6.75 f</td>
</tr>
<tr>
<td>cp15/U903/O (INV2)</td>
<td>0.06</td>
<td>6.80 r</td>
</tr>
<tr>
<td>cp15/dr_index_ff/DFF_2_0/out_reg/RF (DFZCRBN)</td>
<td>0.00</td>
<td>6.80</td>
</tr>
<tr>
<td>data arrival time</td>
<td>6.80</td>
<td></td>
</tr>
<tr>
<td>clock clk (rise edge)</td>
<td>6.00</td>
<td>6.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>6.00</td>
</tr>
<tr>
<td>clock uncertainty</td>
<td>-1.00</td>
<td>5.00</td>
</tr>
<tr>
<td>cp15/dr_index_ff/DFF_2_0/out_reg/CK (DFZCRBN)</td>
<td>0.00</td>
<td>5.00</td>
</tr>
<tr>
<td>data required time</td>
<td>3.93</td>
<td></td>
</tr>
<tr>
<td>data required time</td>
<td>3.93</td>
<td></td>
</tr>
<tr>
<td>slack (VIOLATED)</td>
<td>-2.88</td>
<td></td>
</tr>
</tbody>
</table>

VLSI Design: Chapter 3 81
Buffer Tree
Max Transition/Cap

Maximum Transition Rule Violation

After Optimization

Before Optimization

Upsized Driver or Added Buffers

Maximum Transition Rule Met
Placement and wire capacitance

Unbalanced load

dvr

more balanced

dvr

VLSI Design: Chapter 3
Wire length is determined by layout architecture, but we can choose wire width to minimize delay.

Wire width can vary with distance from driver to adjust the resistance which drives downstream capacitance.
Different branches of tree can be set to different lengths to optimize delay. Optimal tapering improves delay by about 8%.
Speed up the circuit (1)

1. Buffer sizing
2. Buffer insertion, buffer tree
3. Bring critical signal closer to sink
4. Circuit replication
5. Cycle stealing, multi-cycle (memories, …)
6. Re-route
Speed up the circuit (2)

7. Re-placement (new floorplan, Chapter 7)
8. Logic re-write, Logic change (CKT change, re-synthesis, CKT replication, change libraries/cells/IPs…)
9. Wire sizing
10. Process retarget
11. Architecture change, algorithm change
12. Change spec…… and the last one…
Pray….😊
Switch logic

Can implement Boolean formulas as networks of switches.

Can build switches from MOS transistors—transmission gates.

Transmission gates do not amplify but have smaller layouts.
Types of switches

complementary  n-type
Behavior of n-type switch

n-type switch has source-drain voltage drop introduces threshold drop into logic 1. Switch under drives static gate, but gate restores logic levels.

V_{DD} \quad V_{DD} - V_t \quad \sim 0

V_{DD} \quad V_{DD} \sim 0
n-type switch driving switch logic

Voltage drop causes next stage to be turned on weakly.
Behavior of complementary switch

Complementary switch products full-supply voltages for both logic 0 and logic 1:

n-type transistor conducts logic 0;
p-type transistor conducts logic 1.
Home works assignment

Chapter 3:
3-4, 3-10, 3-11