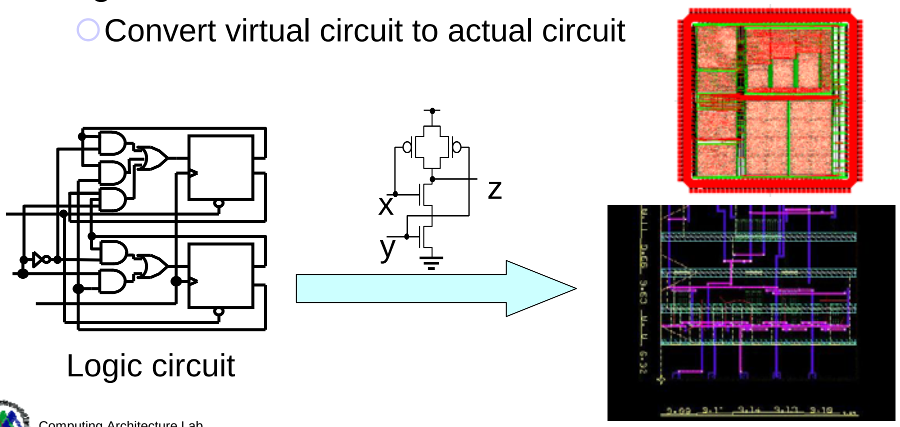
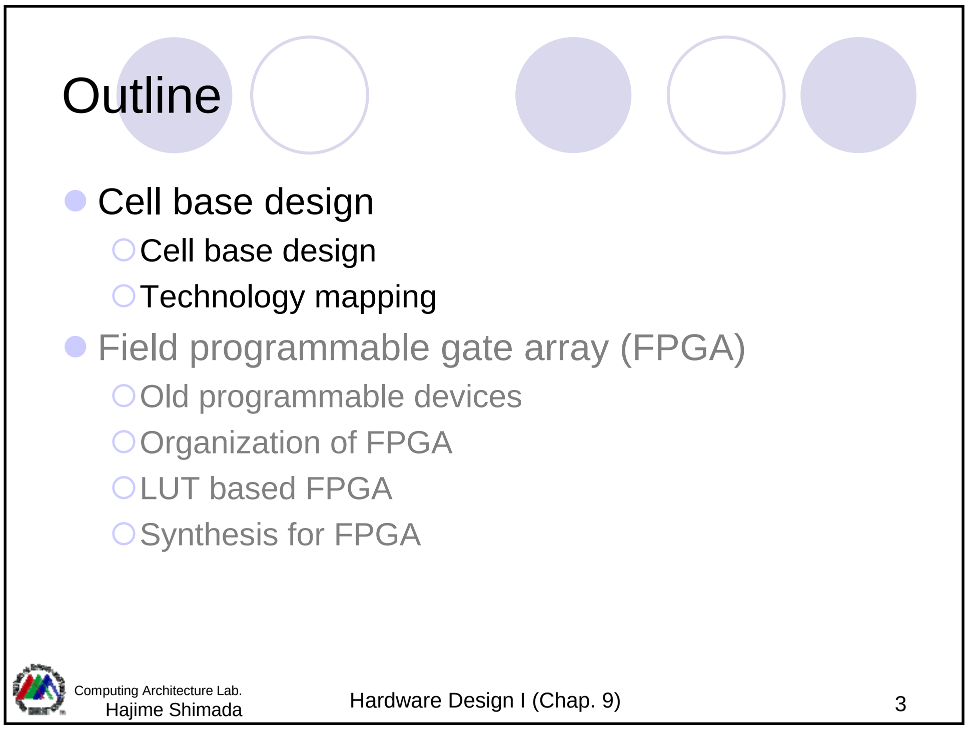
# Hardware Design I Chap. 9 Cell base design and programmable hardwares

Computing Architecture Lab.
Hajime Shimada
E-mail: shimada@is.naist.jp

## Abstract of this chapter

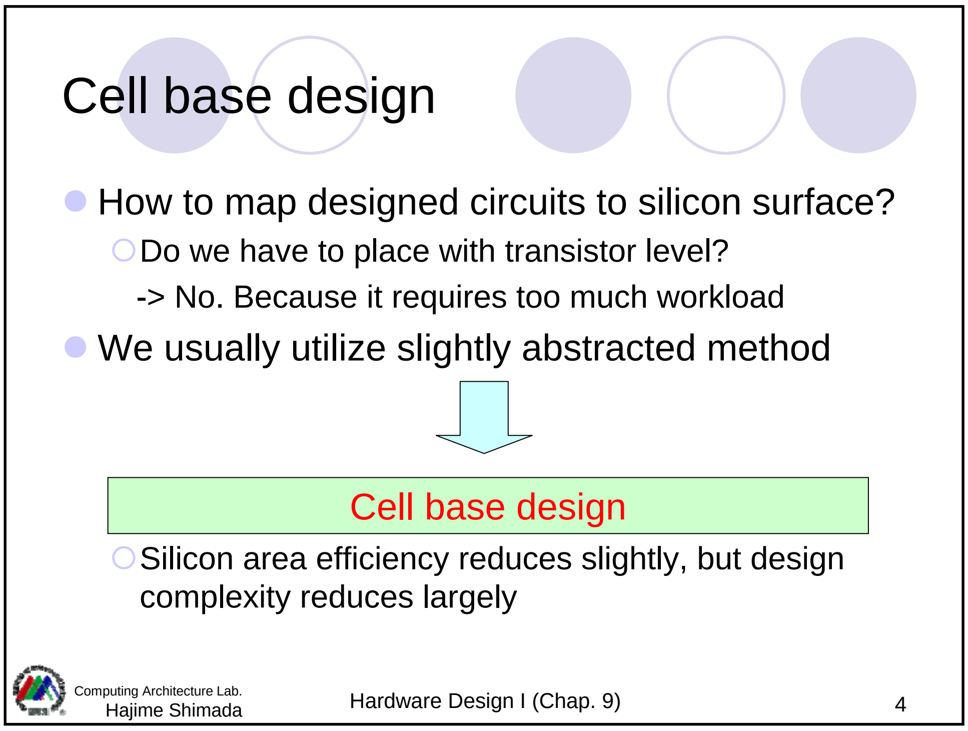
- This chapter treats procedure after we designed logic circuits
  - Convert virtual circuit to actual circuit





## Outline

- Cell base design
  - Cell base design
  - Technology mapping
- Field programmable gate array (FPGA)
  - Old programmable devices
  - Organization of FPGA
  - LUT based FPGA
  - Synthesis for FPGA



## Cell base design

- How to map designed circuits to silicon surface?
  - Do we have to place with transistor level?

    -> No. Because it requires too much workload
- We usually utilize slightly abstracted method

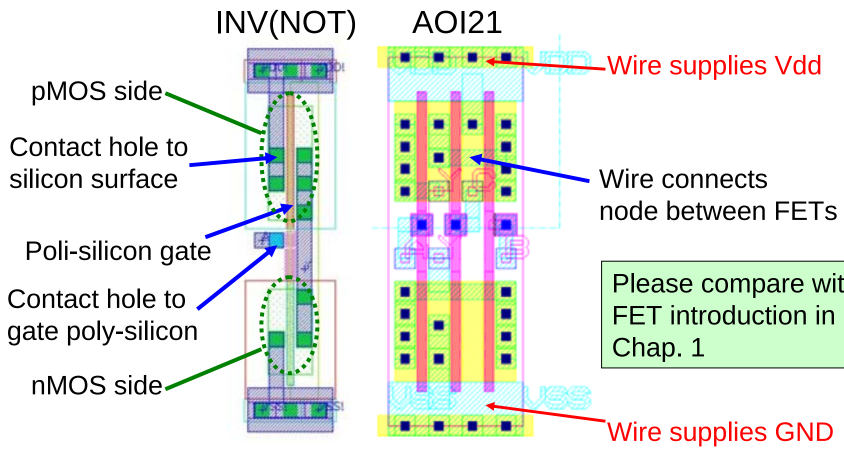
Cell base design

  - Silicon area efficiency reduces slightly, but design complexity reduces largely

## Outline of logic cell

- Widely used in current custom circuit design
- Place logic gate cell (or logic cell) which is constructed by placing FET in prior
  - Vertical size is constant
    - Increases horizontal side if we implement large logic gate
  - Each cell has already optimized
  - Widely used cell are provided as library
    - We can improve performance if library has updated
- Similar to logic cell, several pre-designed hardware is provided (e.g. RAM array macro)

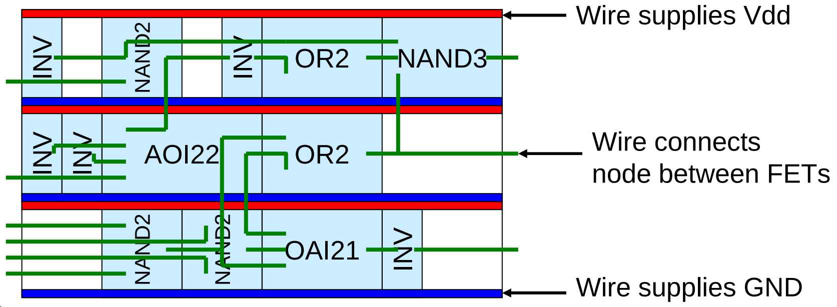
## Detail of logic cell


INV(NOT)
AOI21
pMOS side
Contact hole to silicon surface
Poli-silicon gate
Contact hole to gate poly-silicon
nMOS side
Wire supplies Vdd
Wire connects node between FETs
Wire supplies GND

Please compare with FET introduction in Chap. 1

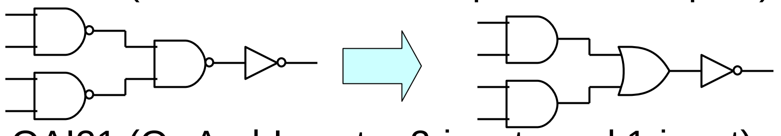
## Placing logic cells on the grid

- We don’t have to consider power supply network
  - We only have to place them to top and bottom of glid
- We can easily to estimate total area
  - Usually, we normalize with NAND2 area

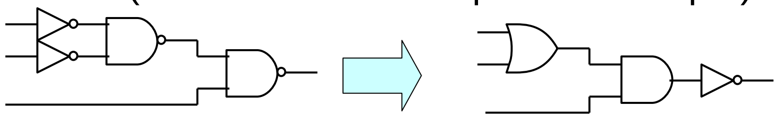


## Combinational devices

- Some cell library contains combinational devices for area efficiency implemenation
- AOI22 (And-Or-Inverter 2-inputs and 2-inputs)



- OAI21 (Or-And-Inverter 2-inputs and 1-input)

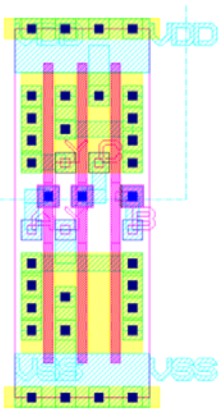


Combinational divice are smaller than combination of those gates

## Why combinational devices are small?

- By optimization
  - Don't have to consider fan-out and fan-in constraint in it
- By balanced organization
  - Total width of pMOS side and nMOS side are balanced

AOI21



## Technology mapping

- Mapping designed circuit to logic devices (cells)
- Characteristic of logic cells differ between semiconductor vendor or customize of library
  - List of usable logic cells (50-100 in usual)
  - Delay/area of each logic cell
    - Usually, delay is normalized by FO4 delay (fan-out 4 inverter delay)
    - Usually, area is normalized by $\lambda$ (half of minimum processing width)
  - Maximum current drive ability, ...

e.g. for high performance

| Type | Delay | Power |
|---|---|---|
| NAND2 | 2 | 6 |
| XOR2 | 5 | 14 |
| ... | ... | ... |

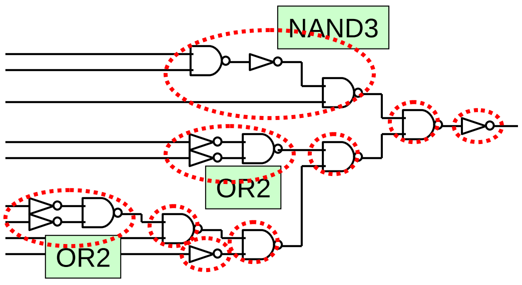
e.g. for low power

| Type | Delay | Power |
|---|---|---|
| NAND2 | 3 | 3 |
| XOR2 | 7 | 6 |
| ... | ... | ... |

## Example of technology mapping (1/2)

- The area of circuit differs between mapping
- Example 1: Total area is 19

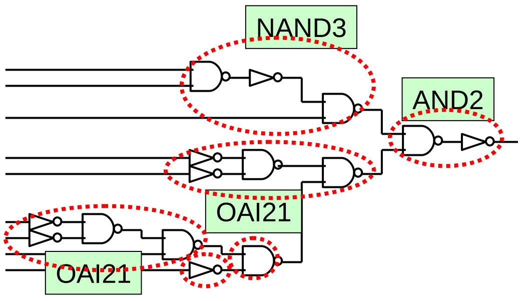
| Cell | Area | Num. | Total |
|---|---|---|---|
| INV | 1 | 2 | 2 |
| NAND2 | 2 | 4 | 8 |
| NAND3 | 3 | 1 | 3 |
| OR2 | 3 | 2 | 6 |
| Total | | 9 | 19 |



## Example of technology mapping (2/2)

- The area of circuit differs between mapping
- Example 2: Total area is 15

| Cell | Area | Num. | Total |
|---|---|---|---|
| INV | 1 | 1 | 1 |
| NAND2 | 2 | 1 | 2 |
| AND2 | 3 | 1 | 3 |
| NAND3 | 3 | 1 | 3 |
| OAI21 | 3 | 2 | 6 |
| Total | | 6 | 15 |

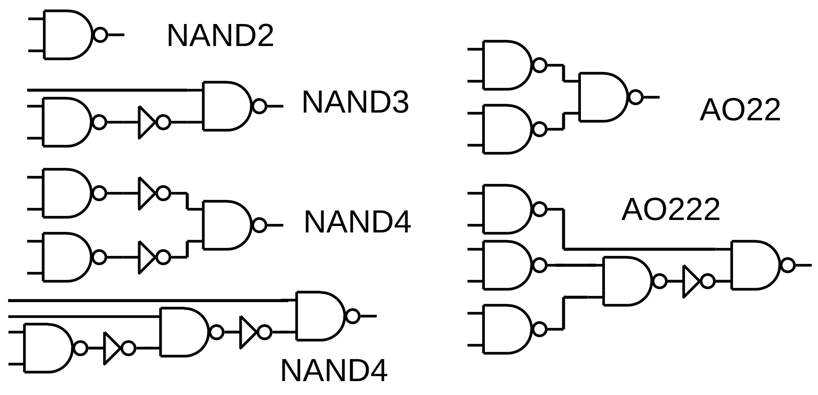


## Flow of technology mapping

1. Create pattern graph from library
2. Translate given circuit to NAND2 and INV
   - Called subject graph
3. Cover subject graph by pattern graph

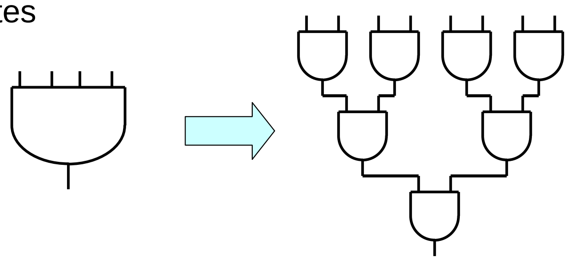
## Creating pattern graph

- Pattern graph: Normalized notation by NAND2 and INV

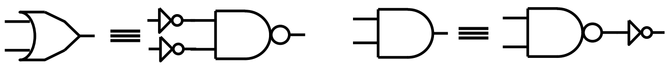


## Creating subject graph

- Translate multi level circuit to NAND2 and INV
  - Multiple input gate is decomposed to multiple 2-input gates

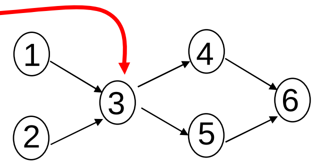


  - AND and OR are represented by NAND and INV

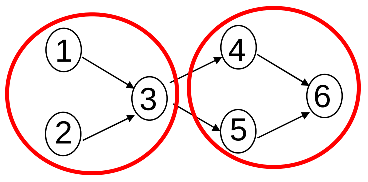


## Graph based covering

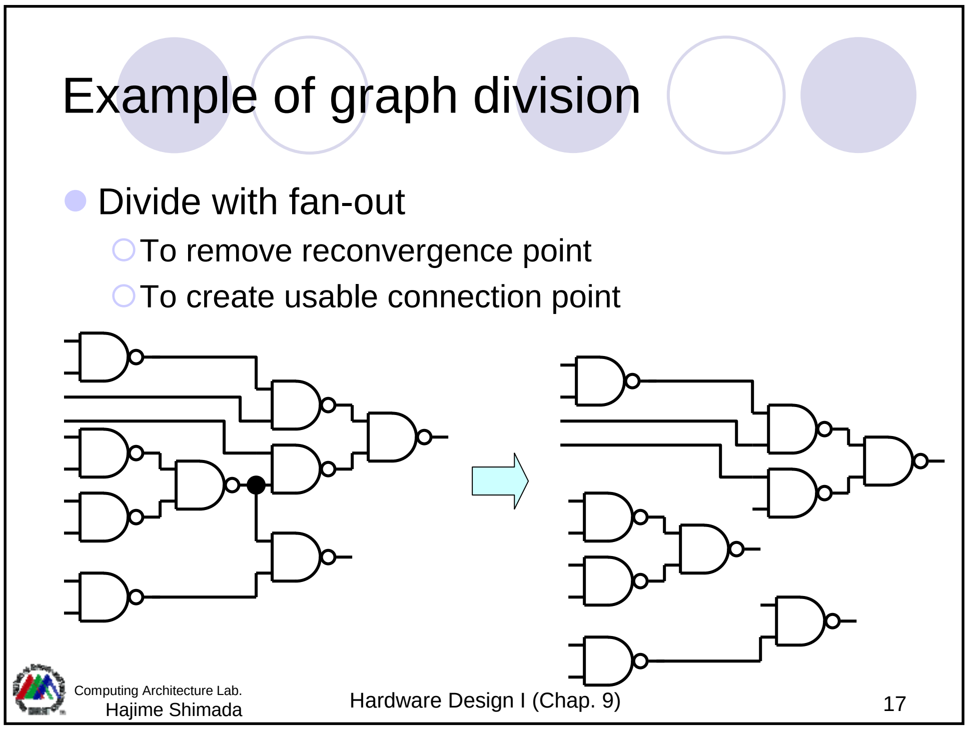
1. Divide graph with fan-out
   - Remove reconvergence point
2. Mapping trees independently
   - If there are several possible mapping, select best one
   - Consider to reduce delay of critical path



Graph

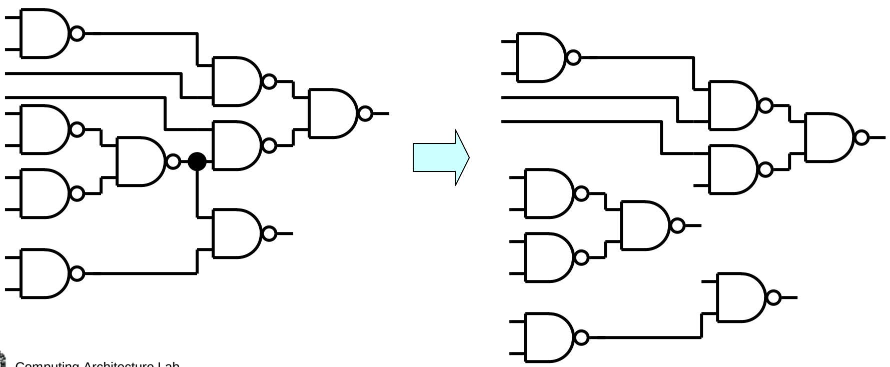


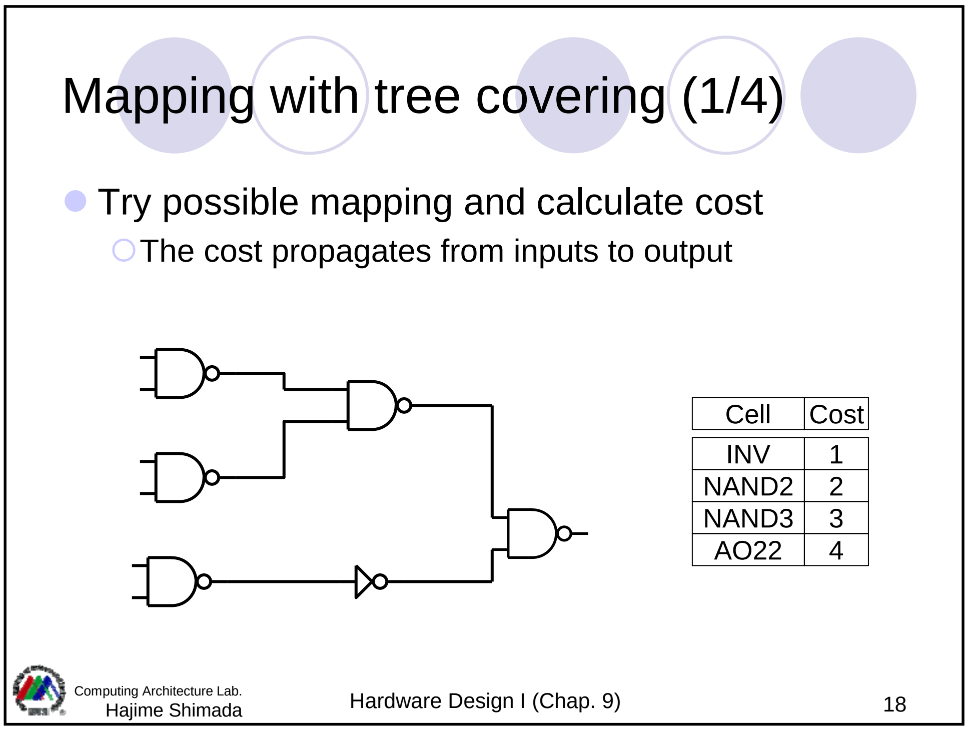
Group of trees



## Example of graph division

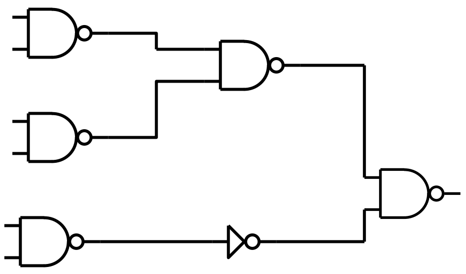
- Divide with fan-out
  - To remove reconvergence point
  - To create usable connection point





## Mapping with tree covering (1/4)

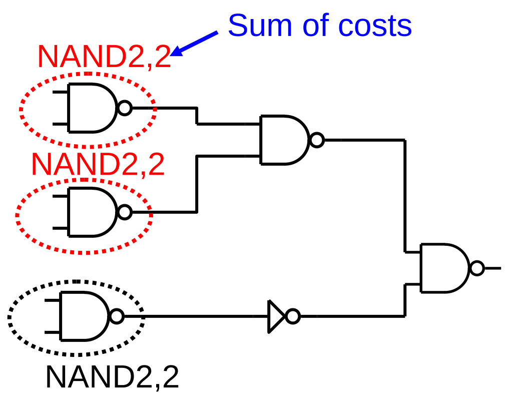
- Try possible mapping and calculate cost
  - The cost propagates from inputs to output



| Cell | Cost |
|---|---|
| INV | 1 |
| NAND2 | 2 |
| NAND3 | 3 |
| AO22 | 4 |

## Mapping with tree covering (2/4)

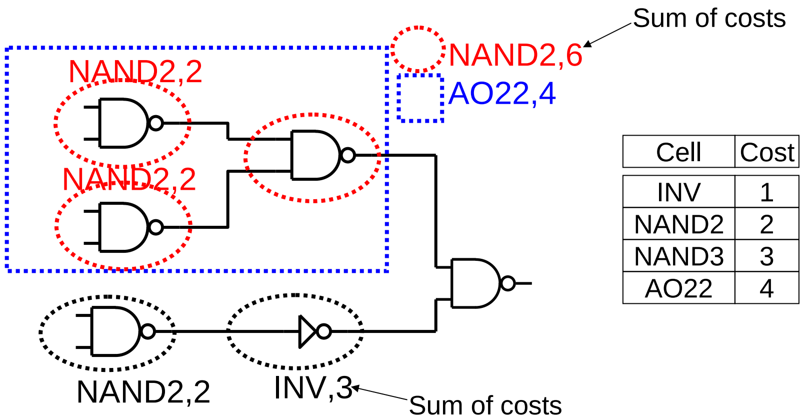
- Apply mapping to level 1 and calculate costs



| Cell | Cost |
|---|---|
| INV | 1 |
| NAND2 | 2 |
| NAND3 | 3 |
| AO22 | 4 |

## Mapping with tree covering (3/4)

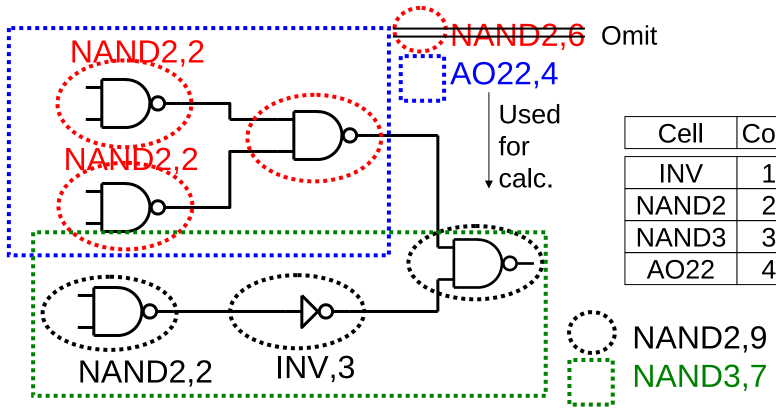
- Two possible mapping is shown in upper side
  - Cost differs between them



| Cell | Cost |
|---|---|
| INV | 1 |
| NAND2 | 2 |
| NAND3 | 3 |
| AO22 | 4 |

## Mapping with tree covering (4/4)

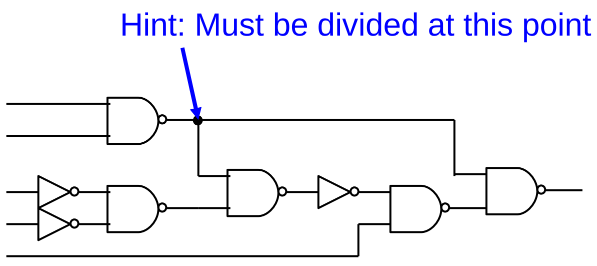
- Two possible mapping is shown in lower side
- Utilize AO22 and NAND3 becomes best



| Cell | Cost |
|---|---|
| INV | 1 |
| NAND2 | 2 |
| NAND3 | 3 |
| AO22 | 4 |

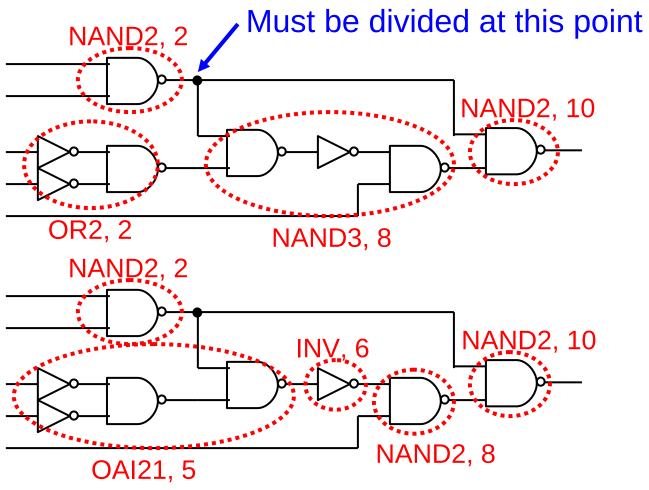
## Short exercise

- Apply tree covering to following circuit
  - Pattern graphs are shown on slide 8, 14, and 15



| Cell | Cost |
|---|---|
| INV | 1 |
| NAND2 | 2 |
| NAND3 | 3 |
| OR2 | 3 |
| AOI22 | 4 |
| AO22 | 4 |
| OAI21 | 3 |

## Answer



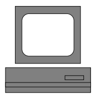
| Cell | Cost |
|---|---|
| INV | 1 |
| NAND2 | 2 |
| NAND3 | 3 |
| OR2 | 3 |
| AOI22 | 4 |
| AO22 | 4 |
| OAI21 | 3 |

## Outline

- Cell base design
  - Cell base design
  - Technology mapping
- Field programmable gate array (FPGA)
  - Old programmable devices
  - Organization of FPGA
  - LUT based FPGA
  - Synthesis for FPGA

## Hardware, software, and programmable hardware

- Hardware
  - High speed and low power consumption
  - Fixed function
    - Design is not so easy
- Software (on processor)
  - We can change operation easily
  - Low speed and high power consumption



Are there any hardware which we can design function easily?

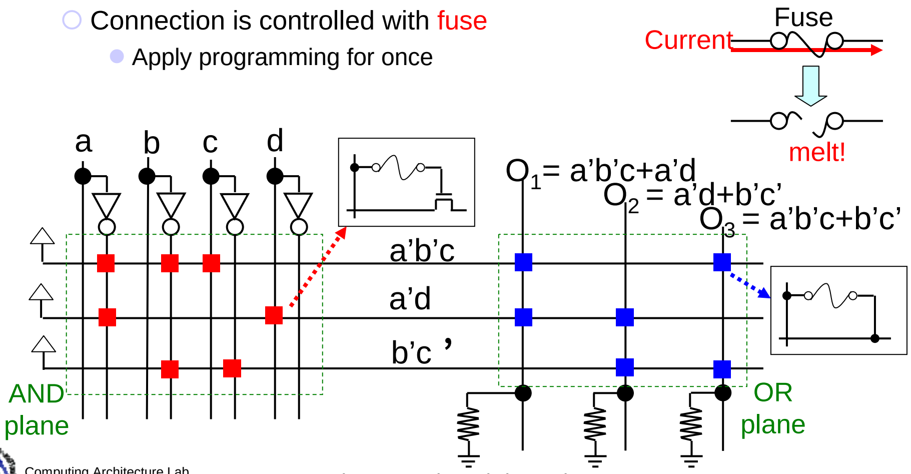
Programmable Hardware

## What's programmable hardware?

- LSI that user can define logic after semiconductor manufacture process
- There are several style programmable hardwares
  - Programmable Logic Array (PLA)
  - Complex Programmable Logic Device (CPLD)
  - Field Programmable Gate Array (FPGA) Widely used!

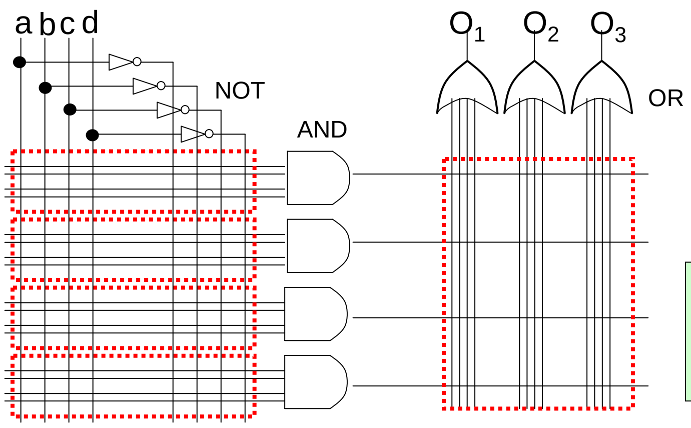
## Programmable Logic Array (PLA)

- Create AND-OR two level logic with wire connection
  - Connection is controlled with fuse
    - Apply programming for once



## Functionality of PLA

- Create arbitrary two level combinational logic with connection under AND plane and OR plane

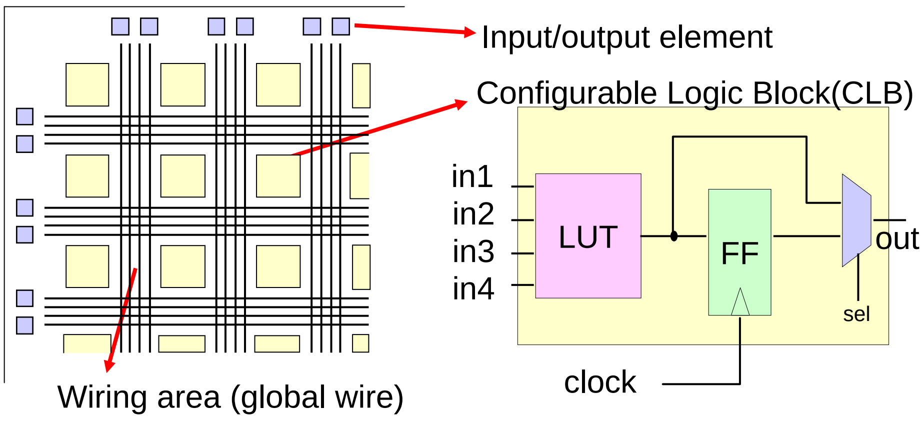


CPLD ->implement multiple PLA in one chip

## Field Programmable Gate Array (FPGA)

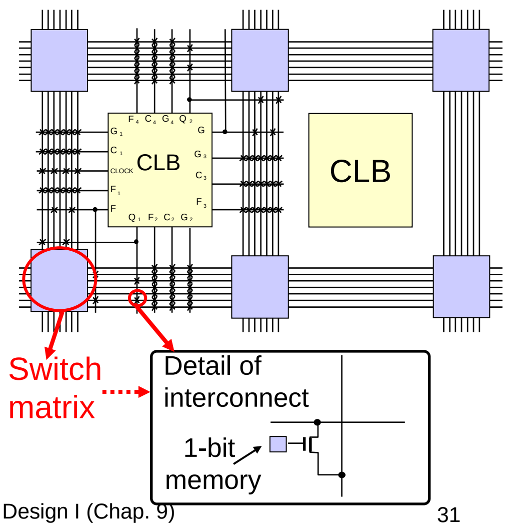
- Most widely used programmable hardware in recent years
- Utilizing Look Up Table (LUT) for unit of logic is current trend
  - LUT: A circuit which can realize arbitrary 3-5 inputs
- Widely used for prototyping
  - Or small-lot production
  - Or temporary use until ASIC comes
- In recent years, some of them can change organization of implemented hardware under processing data

## Outlined organization of FPGA using LUT



## Detail of network FPGA

- There are two connection points
  - Global wire and global wire (switch matrix)
  - Global wire and wire to CLB
- Wires are connected with path transistor
  - Connection is controlled by value of memory
  - By writing value to memory, we can change connection state

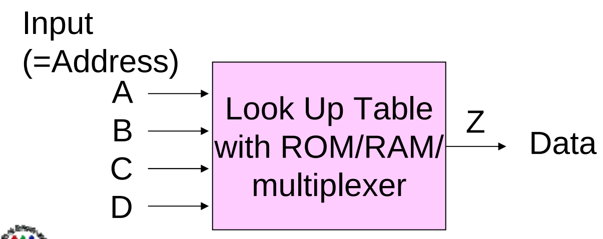


## Additional futures for FPGA

- Inner CLB
  - Implement 4-input LUT with two 3-input LUT (convenient for implementing adder)
  - Prepare high-speed carry line
- Specialized block
  - Prepare SRAM memory block for temporal data storage
  - Prepare specialized block (e.g. multiplier, high speed I/O, CPU, DSP, ...)
- Function of blocks
  - Some FPGA accept updating under operation
  - Some FPGA accept partial reconfigure

## Organization of LUT

- By assuming RAM based LUT organization, you can easily to estimate its function
  - e.g. It outputs 1-bit output from correspoinding 4-bit address
- LUT is also achieved by ROM or multiplexer

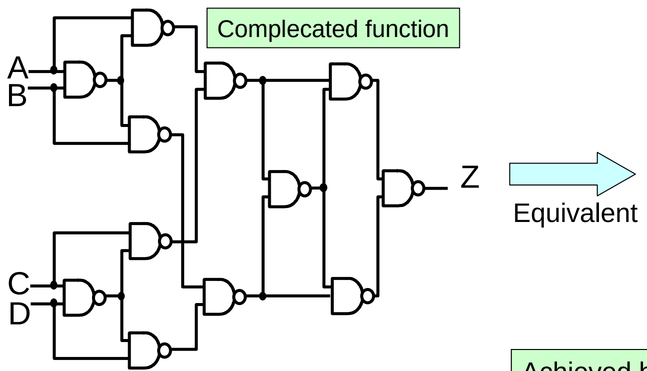


Content of RAM

| ABCD | Z |
|---|---|
| 0000 | 0 |
| 1000 | 1 |
| 0100 | 1 |
| 1100 | 0 |
| 0010 | 1 |
| 1010 | 1 |
| 0110 | 0 |
| 1110 | 1 |
| 0001 | 1 |
| 1001 | 0 |
| 0101 | 1 |
| 1101 | 1 |
| 0011 | 0 |
| 1011 | 1 |
| 0111 | 1 |
| 1111 | 0 |

## Represent logical expression with LUT

- We can implement arbitrary logical function with LUT
  - Achieve $2^{2^n}$ = 65536 function with 4-input LUT



| ABCD | Z |
|---|---|
| 0000 | 0 |
| 1000 | 1 |
| 0100 | 1 |
| 1100 | 0 |
| 0010 | 1 |
| 1010 | 1 |
| 0110 | 0 |
| 1110 | 1 |
| 0001 | 1 |
| 1001 | 0 |
| 0101 | 1 |
| 1101 | 1 |
| 0011 | 0 |
| 1011 | 1 |
| 0111 | 1 |
| 1111 | 0 |

Achieved by one 4-input LUT

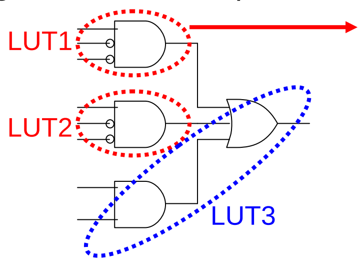
## How to map logic circuit to LUT based FPGA?

- Logic design for LUT based FPGA is defined as follows
  - Input: logical expressions
  - Output: Mapping information to LUT and their connections
- How to map logic circuit to LUT based FPGA?
  - Technology mapping based method
  - Function decomposition based method

## Technology mapping based method (1/2)

1. (Optimize logical circuit and translate to tree)
2. Decompose node to less than *n*-inputs
   - *n* equals to size of LUT (*n*-input LUT)
3. Cover tree with n-input partial circuit

e.g. cover with 4-input LUT

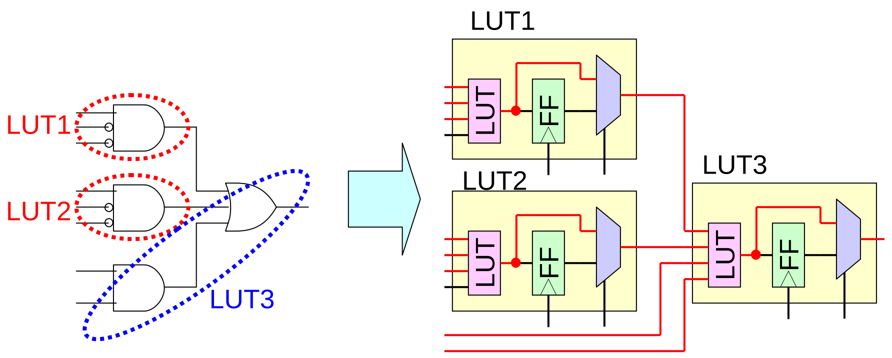


| ABC | f |
|---|---|
| 000 | 0 |
| 001 | 0 |
| : | |
| 100 | 1 |
| : | |
| 110 | 0 |
| 111 | 0 |

Content of LUT1
(Excluding D)

## Technology mapping based method (2/2)

- If we map two level logic, we utilize path through flip-flop in CLB



## Redundancy in technology mapping based method

- There's redundant point in technology mapping based method. Where is it?
  1. (Optimize logical circuit and translate to tree)
- Because there's no cost difference if the logical expression is represented by same number of LUT
  - e.g. There's no cost difference between following logical expression when we use 4-input LUT
    - f = a'bcd + ab'cd + abc'd + abcd'
    - f = abcd

## Bad example on technology mapping (1/2)

- Let’s consider following logical expression

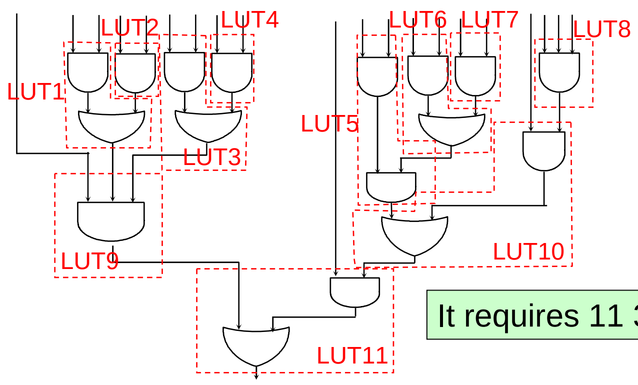
$$f = x_1\ x_2\ x_3'\ x_4'\ x_5' + x_1'\ x_2\ x_3'\ x_4'\ x_5 + x_1'\ x_2'\ x_3\ x_4'\ x_5' + x_1\ x_2'\ x_3\ x_4'\ x_5 + x_1\ x_2'\ x_3'\ x_4\ x_5' + x_1'\ x_2'\ x_3'\ x_4\ x_5 + x_1'\ x_2\ x_3\ x_4\ x_5'$$

- Reduce literals before technology mapping

$$f = x_3'\ (x_2\ x_4' + x_2'\ x_4)\ (x_1\ x_5' + x_1'\ x_5) + x_3\ (x_2'\ x_4'\ (x_1' x_5' + x_1\ x_5) + x_1'\ x_2\ x_4\ x_5')$$

## Bad example on technology mapping (2/2)

- Consider mapping them with 3-input LUT



It requires 11 3-input LUTs

## Function decomposition based method (1/2)

- Decompose function into function which has limited number of variables
  - f(X, Y) -> g(X, h(Y))
  - The number of variables must be smaller than inputs of LUT
- e.g. decompose function into less than 3 variables

$$f = x_1 x_2 x_3' x_4' x_5' + x_1' x_2 x_3' x_4' x_5 + x_1' x_2' x_3 x_4' x_5' + x_1 x_2' x_3 x_4' x_5 + x_1 x_2' x_3' x_4 x_5' + x_1' x_2' x_3' x_4 x_5 + x_1' x_2 x_3 x_4 x_5'$$

$$\Downarrow$$

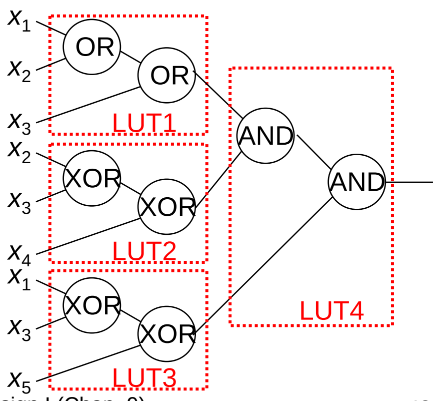
$$f = (\underbrace{x_1 + x_2 + x_3}_{\text{3 variables}}) \cdot (\underbrace{x_2 \oplus x_3 \oplus x_4}_{\text{3 variables}}) \cdot (\underbrace{x_1 \oplus x_3 \oplus x_5}_{\text{3 variables}})$$

## Function decomposition based method (2/2)

- By mapping decomposed function, we can reduce number of LUT
  - Prior example can be implemented with 4 3-input LUTs

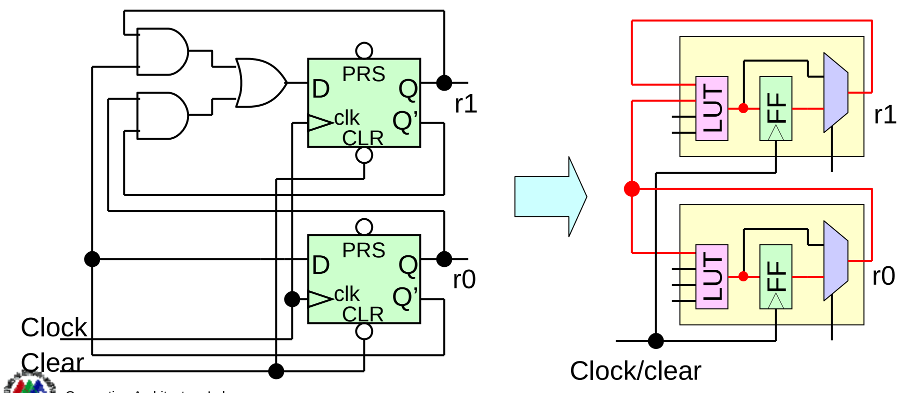
$$f = (x_1 + x_2 + x_3) \cdot (x_2 \oplus x_3 \oplus x_4) \cdot (x_1 \oplus x_3 \oplus x_5)$$

$$\Rightarrow$$



## Implement sequential circuit to FPGA

- By utilizing path through FF, we can implement sequential circuit



## Latest FPGAs

- Scale of circuit becomes millions gates order
  - Xilinx Virtex-7: 1955K CLB,  46.5M-bit SRAM block
  - Altera Stratix-V:  952K CLB,  52.8M-bit SRAM block
    - Altera utlizes 8-input LUT so that number of CLB becomes small
- Operation speed becomes hundreds of MHz
- Some of them implements large scale specialized block
  - Commercial level CPU/DSP, Ethernet MAC, DDR3-SDRAM interface, G-bit per second level interfaces, ...

Even if in practical use, it sometimes included in commercial appliances (not only for experimental production)