

Communications and Computer Engineering II

# FPGA Basis

Hiroki Nakahara

Tokyo Institute of Technology

# Outline

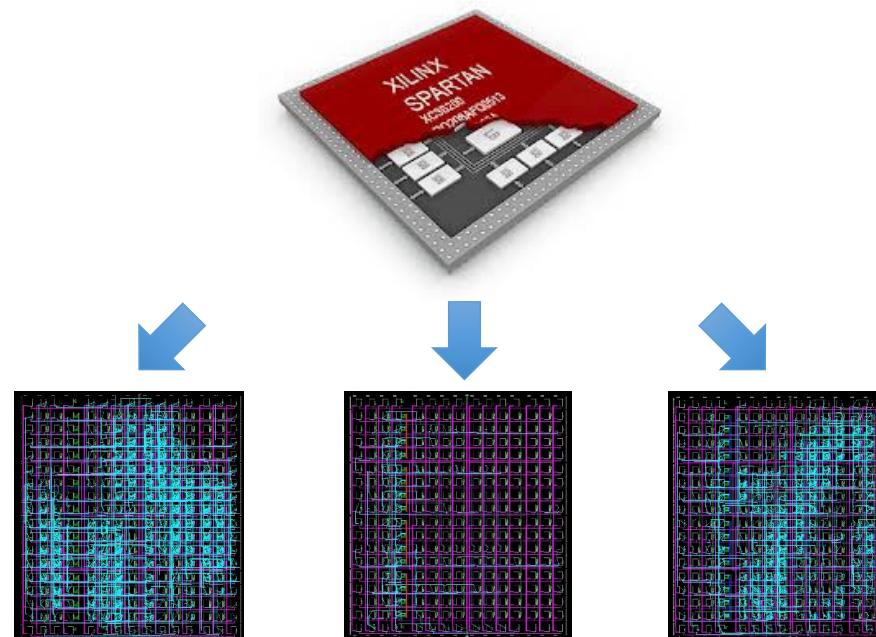
- FPGA Basis
  - FPGA Architecture
- Standard FPGA Design
  - RTL (Register Transfer Level)
- Summary

# FPGA Basis

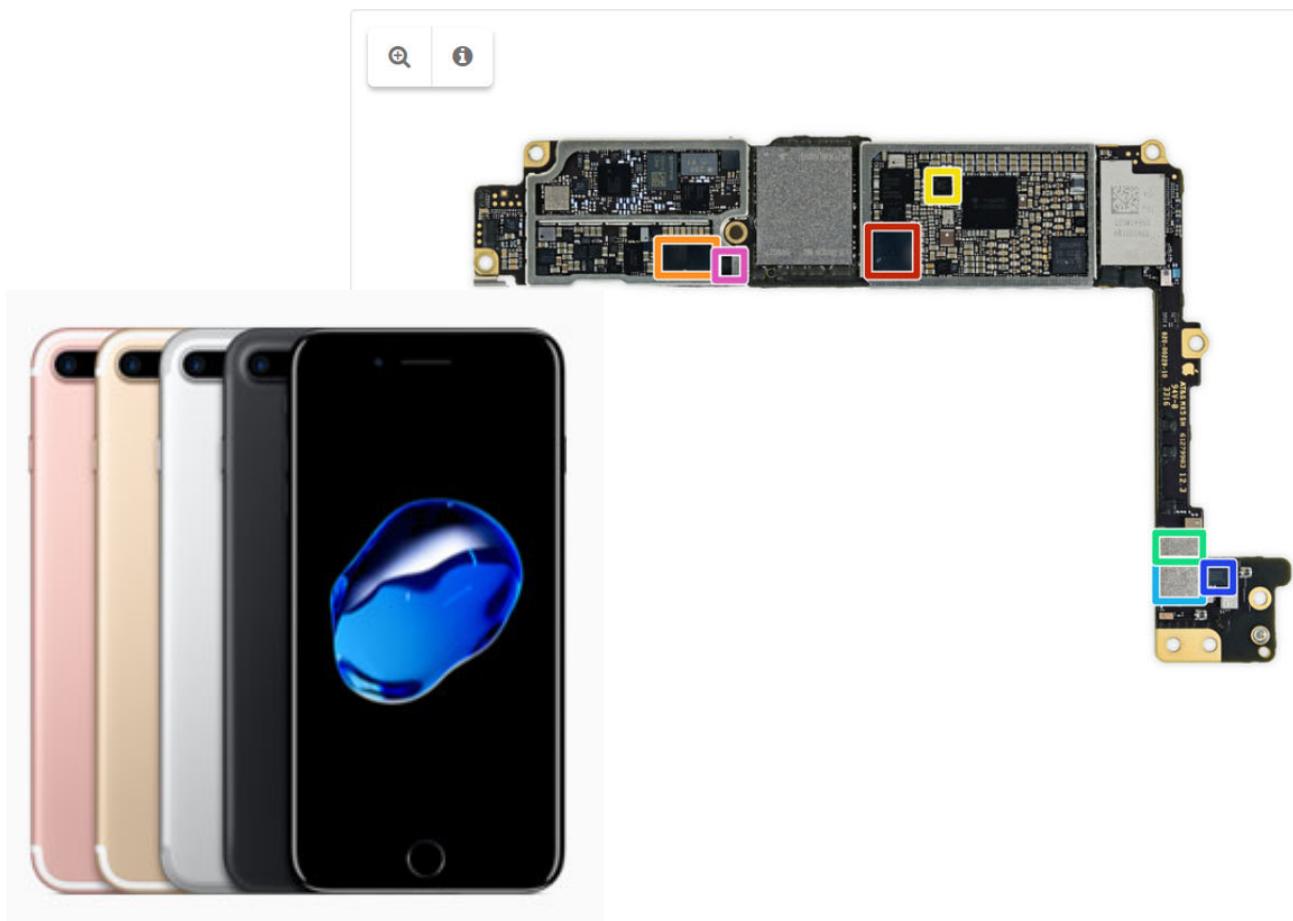
# 1.1 Programmable Device

# FPGA

- Reconfigurable LSI or Programmable Hardware
- Programmable Logic Array and Programmable Interconnection
- Programmed by Reconfigurable Data



# iPhone7 Plus



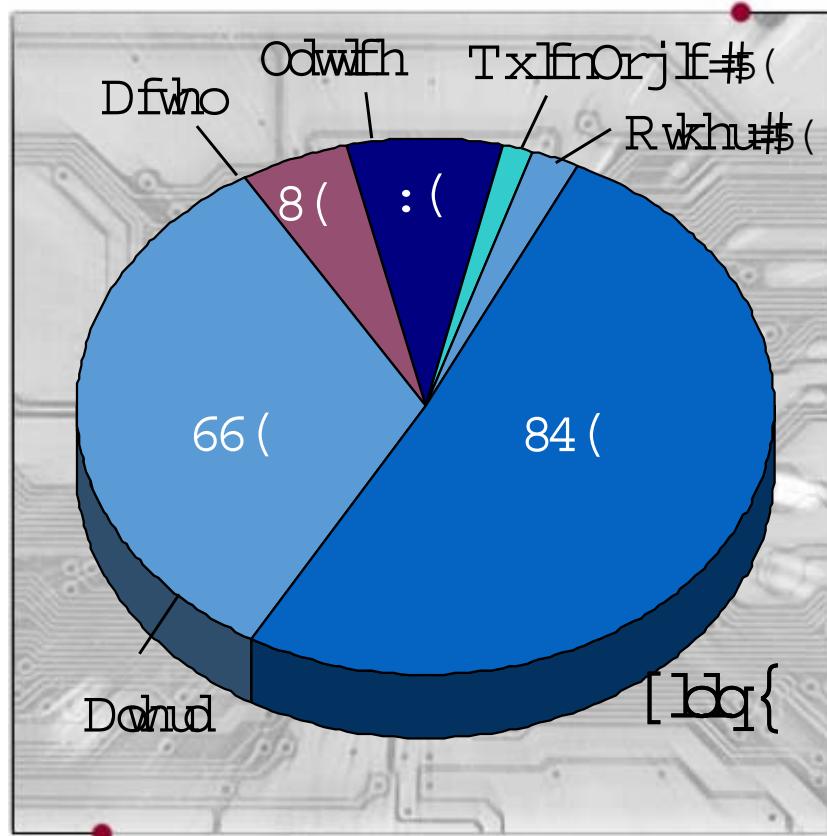
- But wait, there are even more ICs on the back!
  - Apple/Cirrus Logic 338S00105 Audio Codec
  - Cirrus Logic 338S00220 Audio Amplifier(x2)
  - Lattice Semiconductor ICE5LP4K
  - Skyworks 13702-20 Diversity Receive Module
  - Skyworks 13703-21 Diversity Receive Module
  - Avago LFI630 183439
  - NXP 610A38

Source: <https://www.ifixit.com/Teardown/iPhone+7+Plus+Teardown/67384>

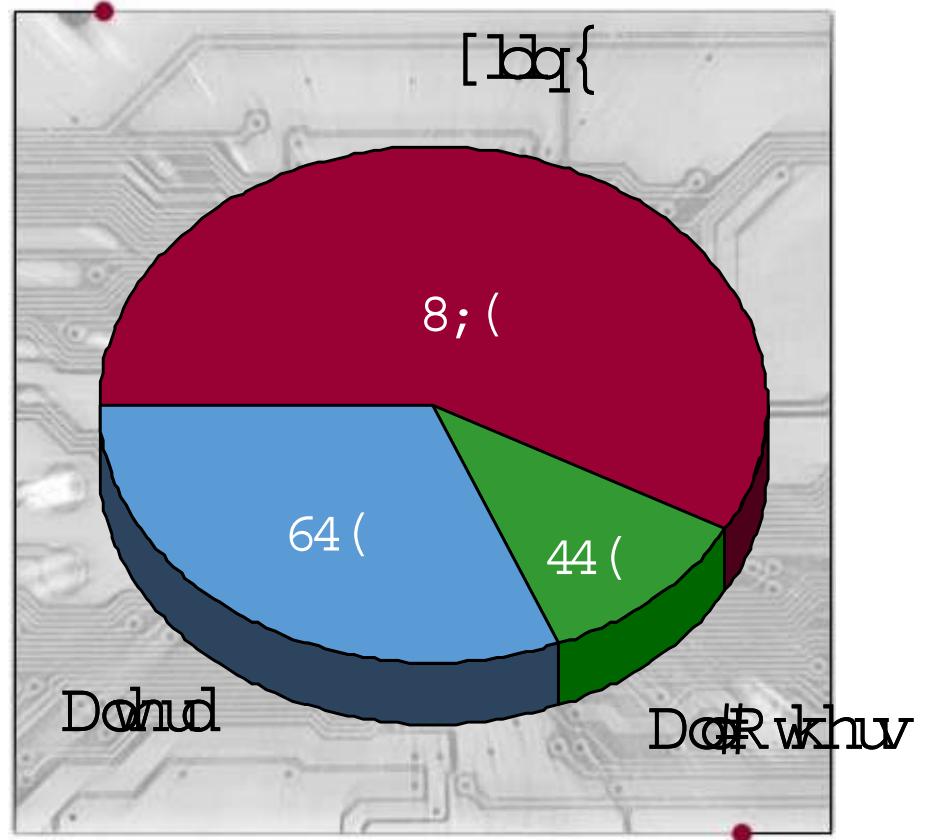
# The Programmable Marketplace

Q1 Calendar Year 2005

**PLD Segment**



**FPGA Sub-Segment**



Wz r#grp [1dq]#txssdhuv#gg1fdwqj#1# dwuibj#p dunhw

Source: Company reports

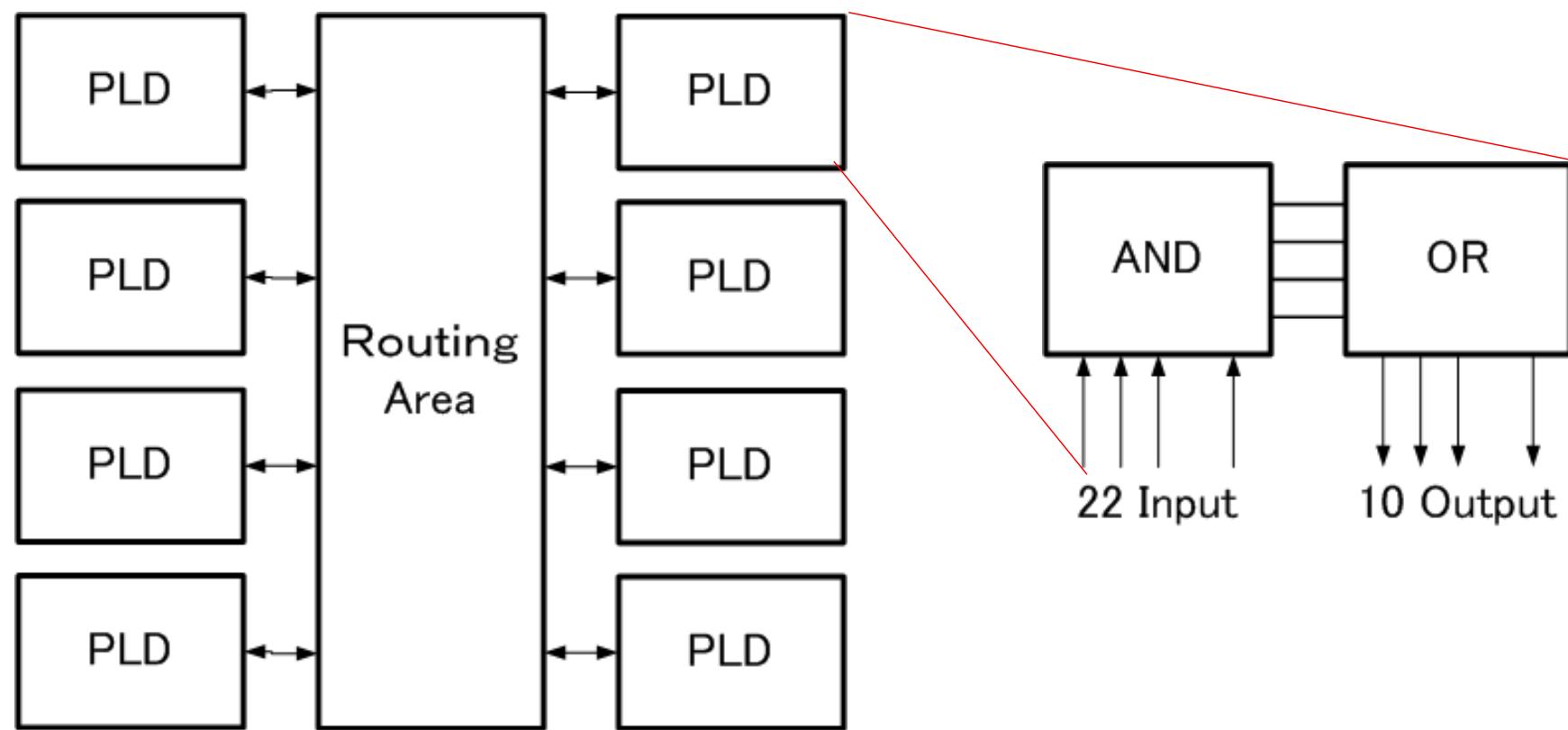
Latest information available; computed on a 4-quarter rolling basis

# FPGA Families

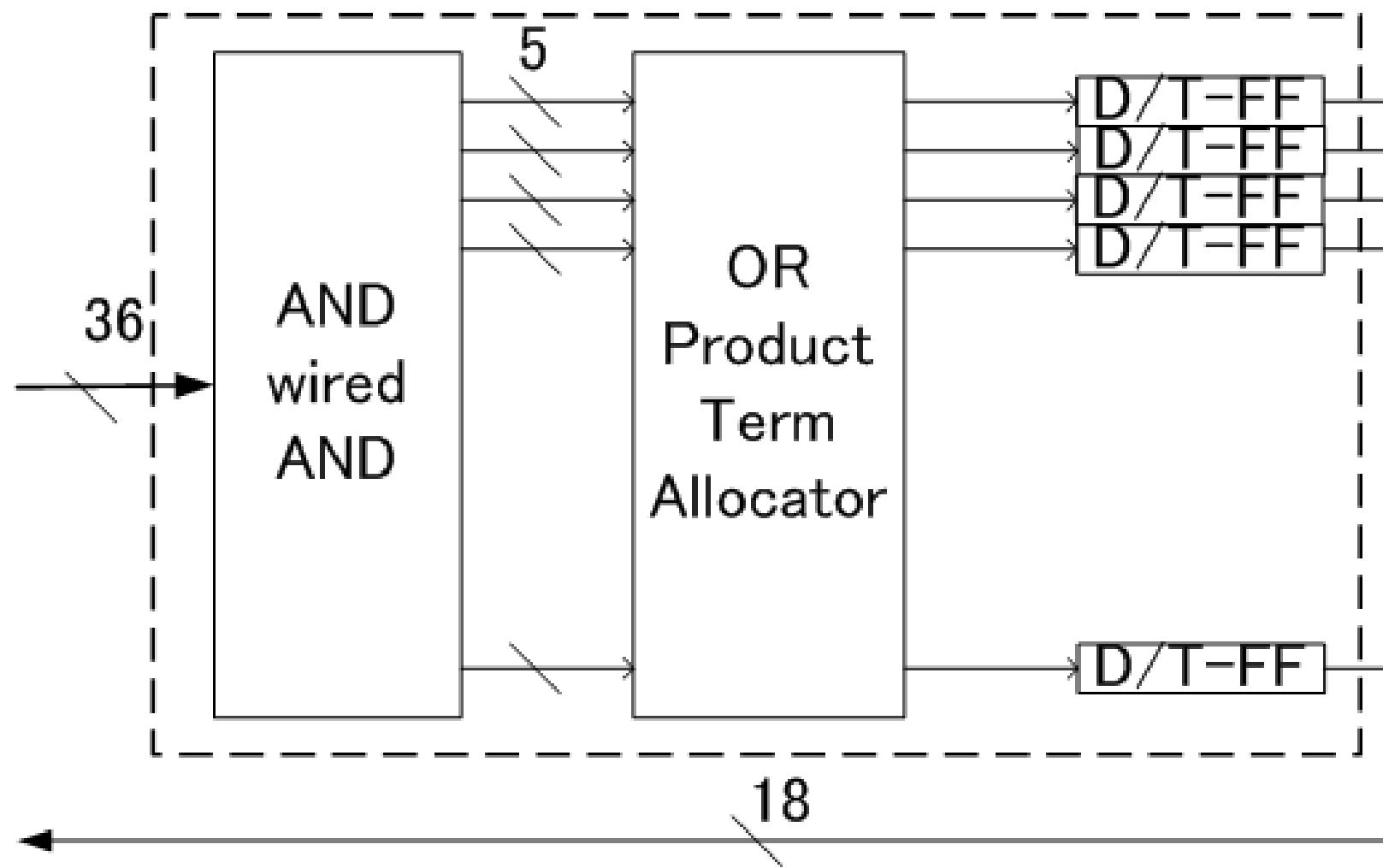
	Low-cost	Middle-end	High- Performance	Embedded
Xilinx Inc.	Artix-7 Spartan-7	Kintex-7 Kintex Ultra Scale	Virtex-7 Virtex Ultra Scale	Zynq
Altera Corp.	Cyclone V	Arria 10	Stratix V Stratix X	Cyclone SoC

## 1.2 Programmable Logic Device (PLD)

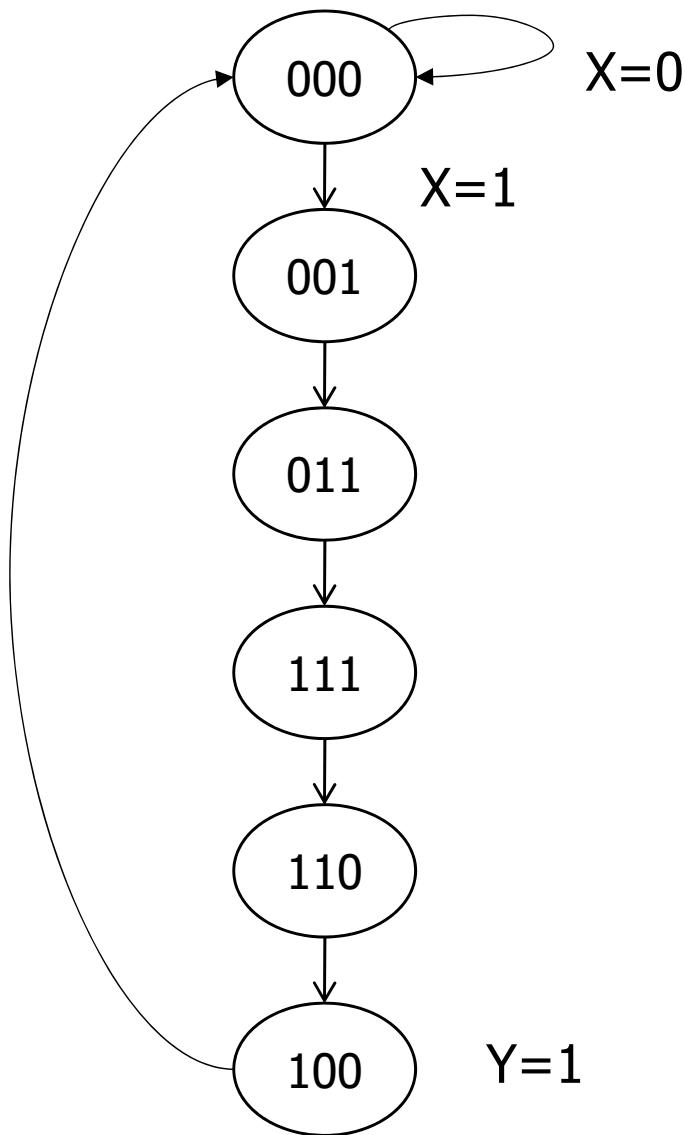
# Complex PLD (CPLD)



# Function Block in CPLD



# Example of PLD Design

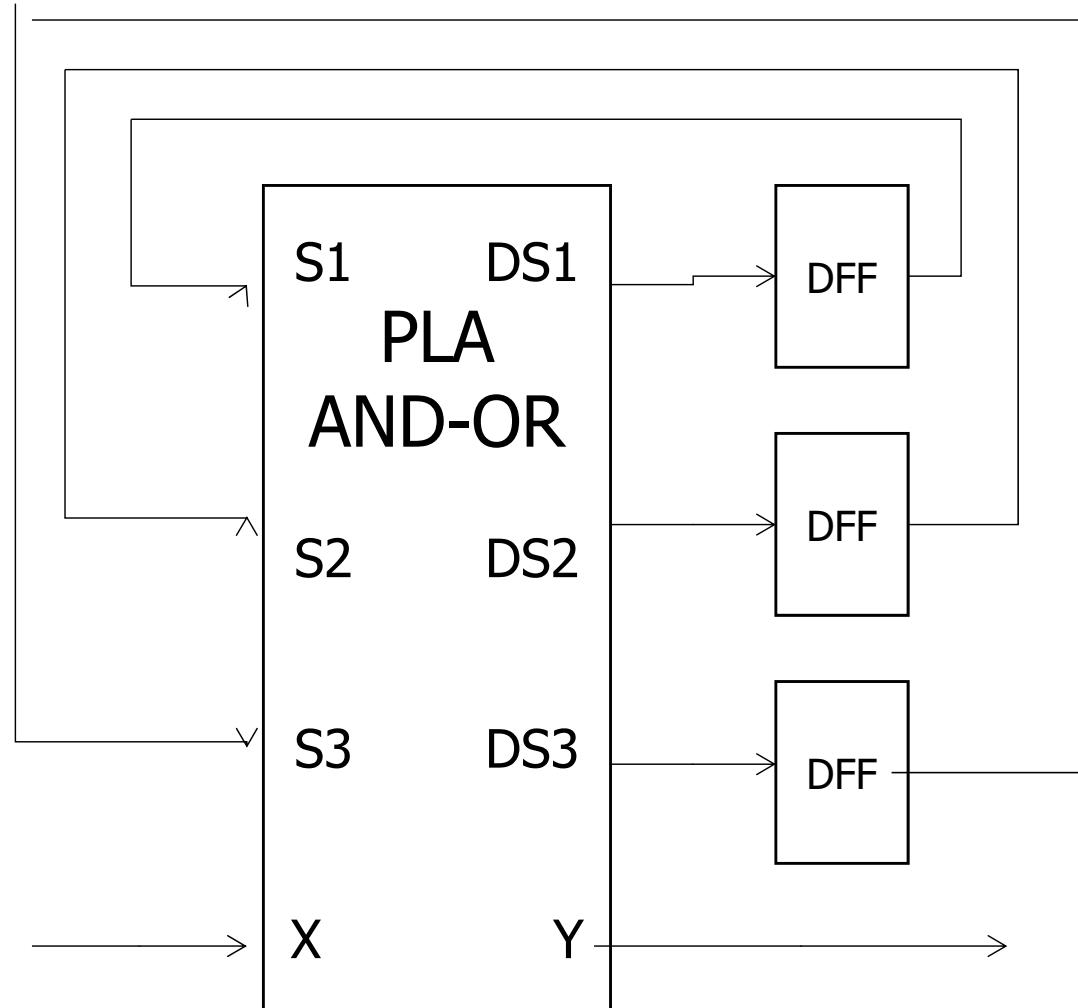


State Transition Table

S1	S2	S3	X	S1	S2	S3	Y
0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0
0	0	1	*	0	1	1	0
0	1	1	*	1	1	1	0
1	1	1	*	1	1	0	0
1	1	0	*	1	0	0	0
1	0	0	*	0	0	0	1

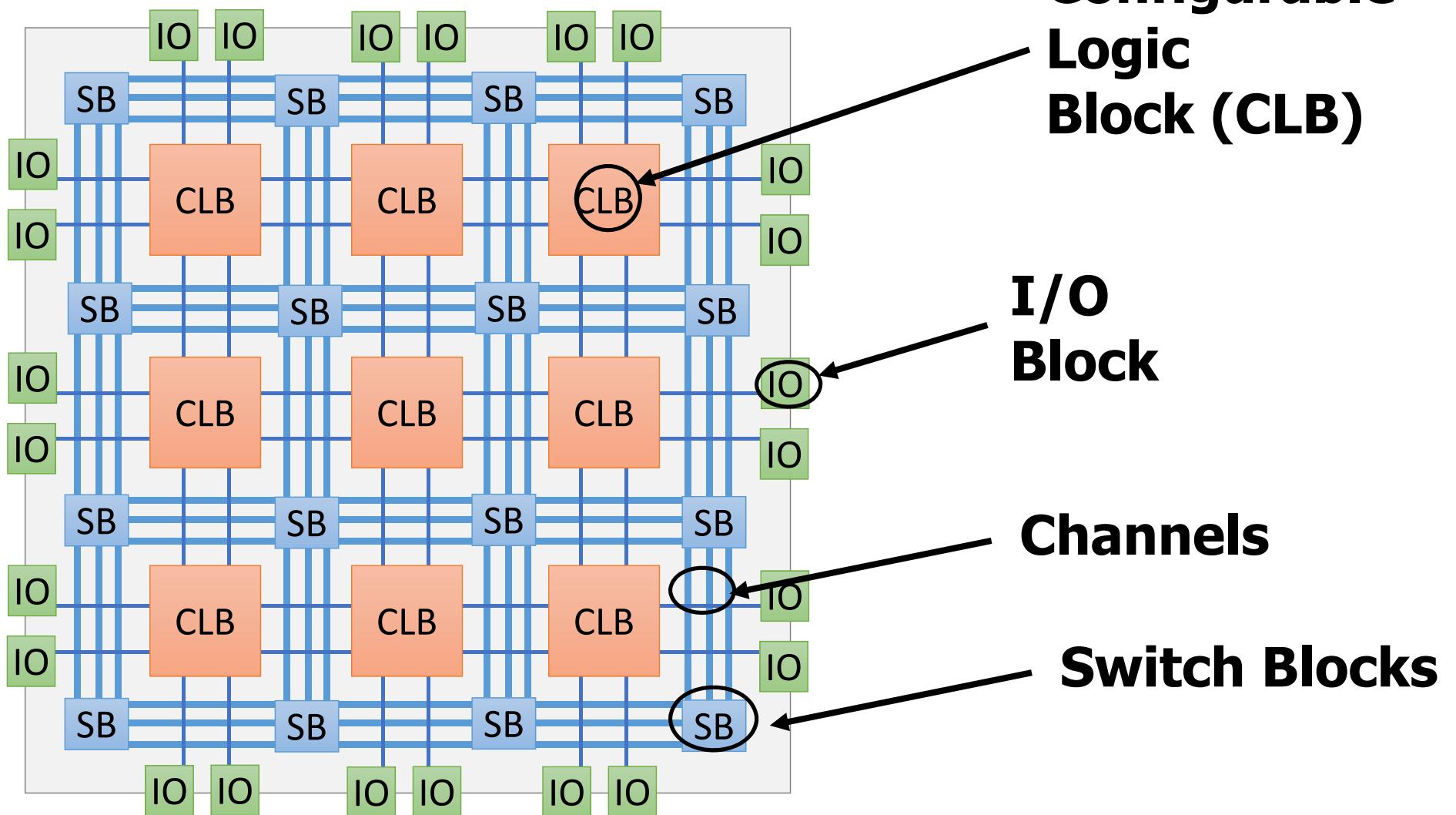
# PLA & FF Realization

S1	S2	S3	X	DS1	DS2	DS3	Y
0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0
0	0	1	*	0	1	1	0
0	1	1	*	1	1	1	0
1	1	1	*	1	1	0	0
1	1	0	*	1	0	0	0
1	0	0	*	0	0	0	1

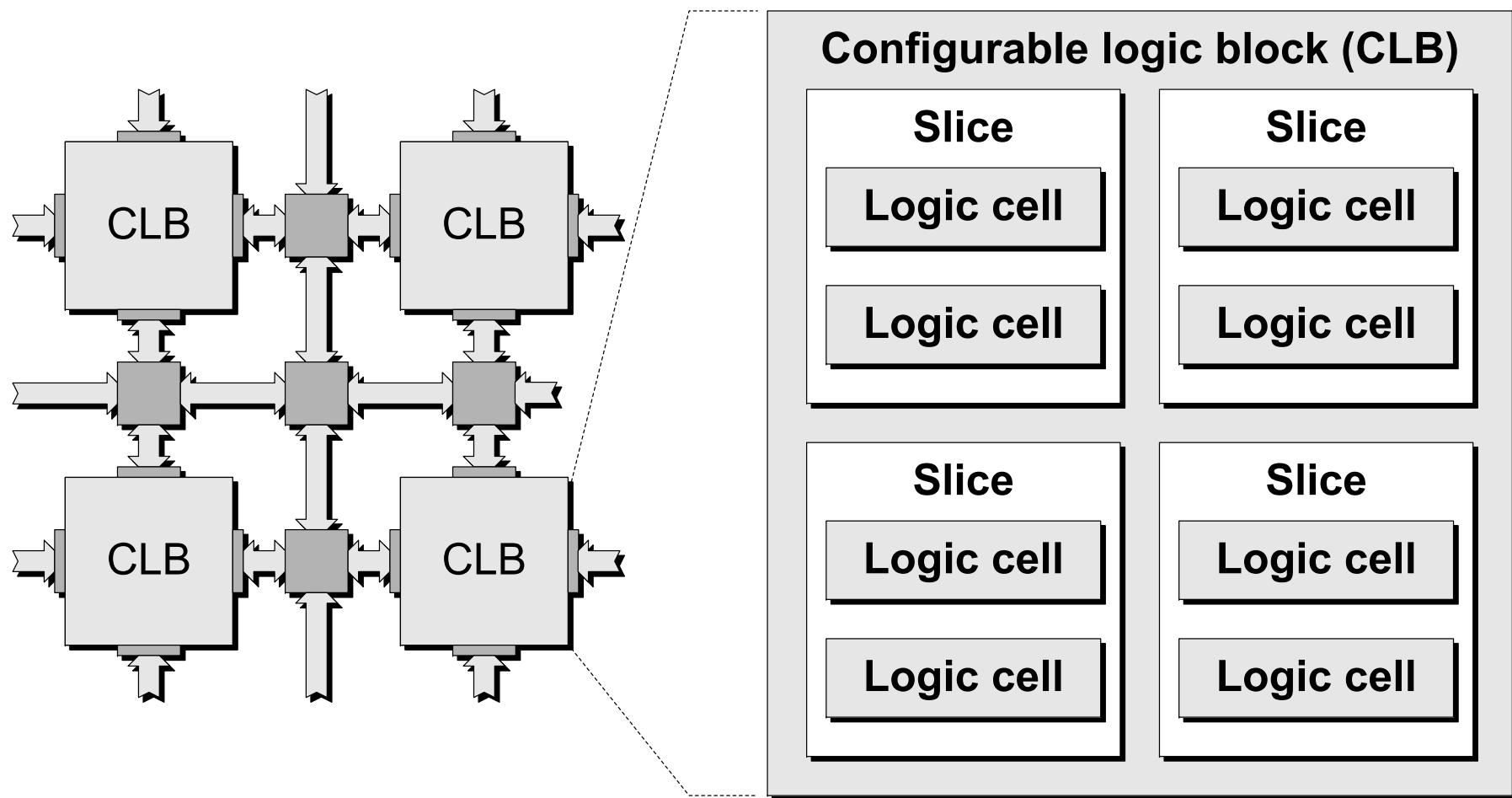


# 1.3 FPGA

# Island Style FPGA

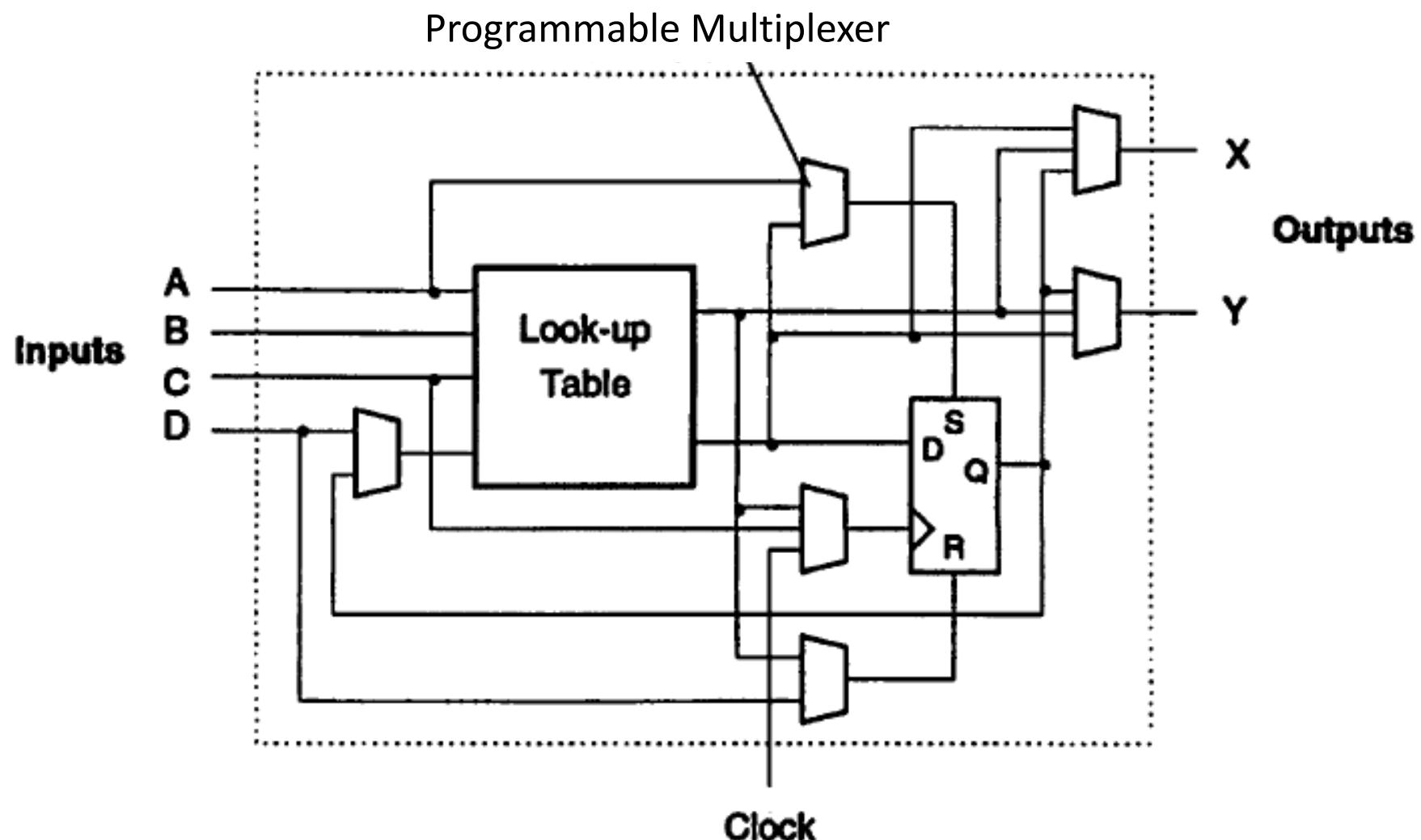


# Xilinx CLB



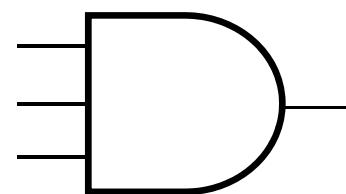
The Design Warrior's Guide to FPGAs  
Devices, Tools, and Flows. ISBN 0750676043  
Copyright © 2004 Mentor Graphics Corp. ([www.mentor.com](http://www.mentor.com))

# Logic Cell (Xilinx Inc. XC2000)

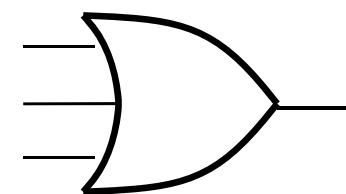


# Realization of a Logic Function

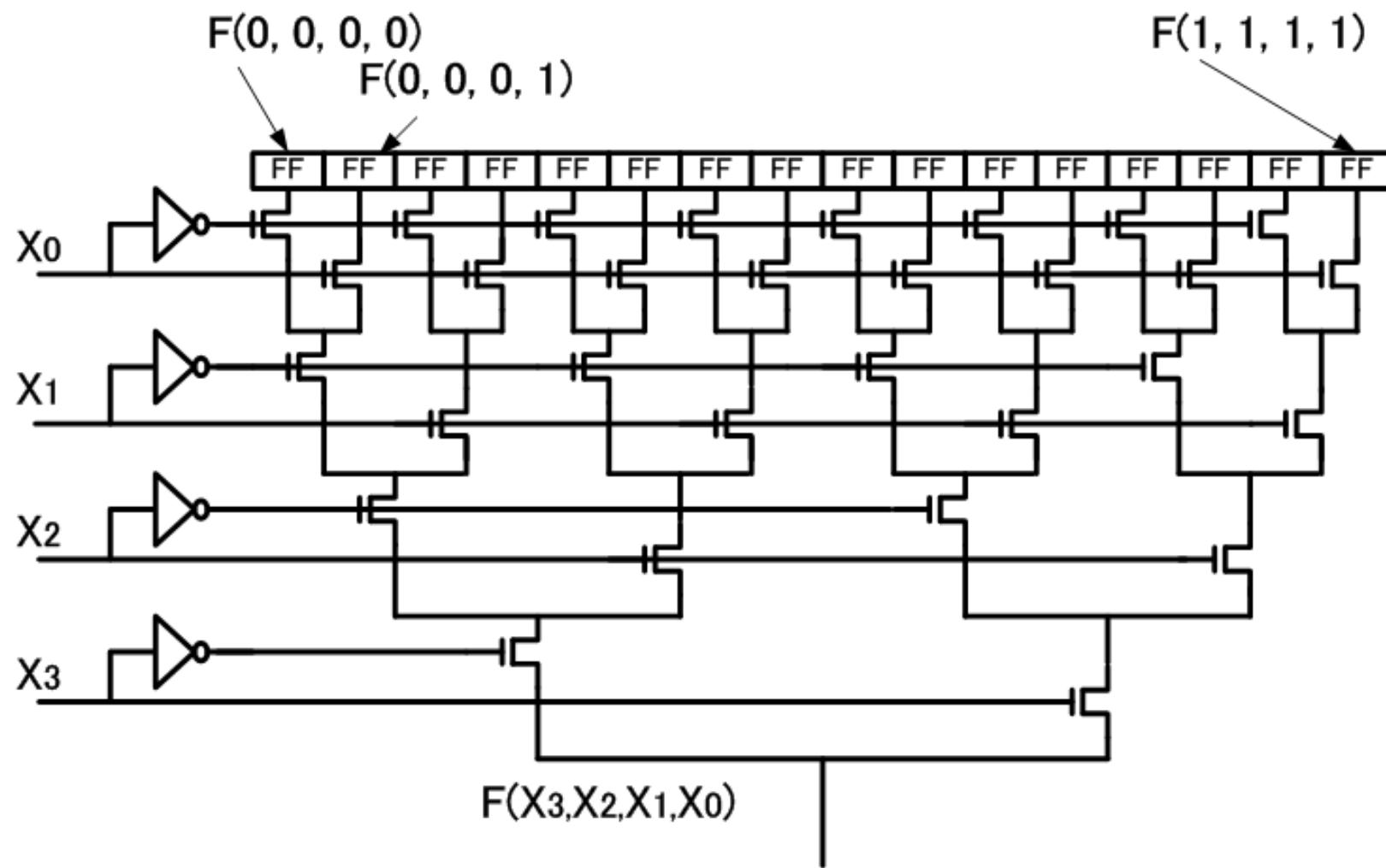
x0	x1	x2	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



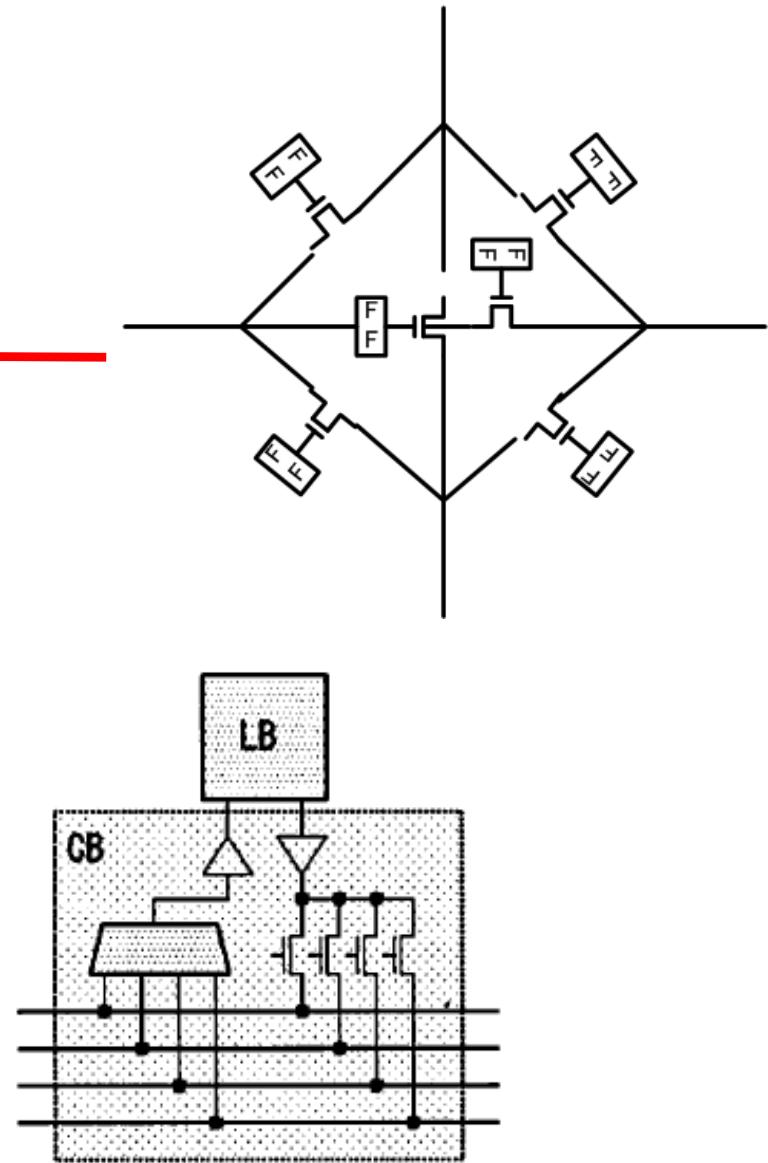
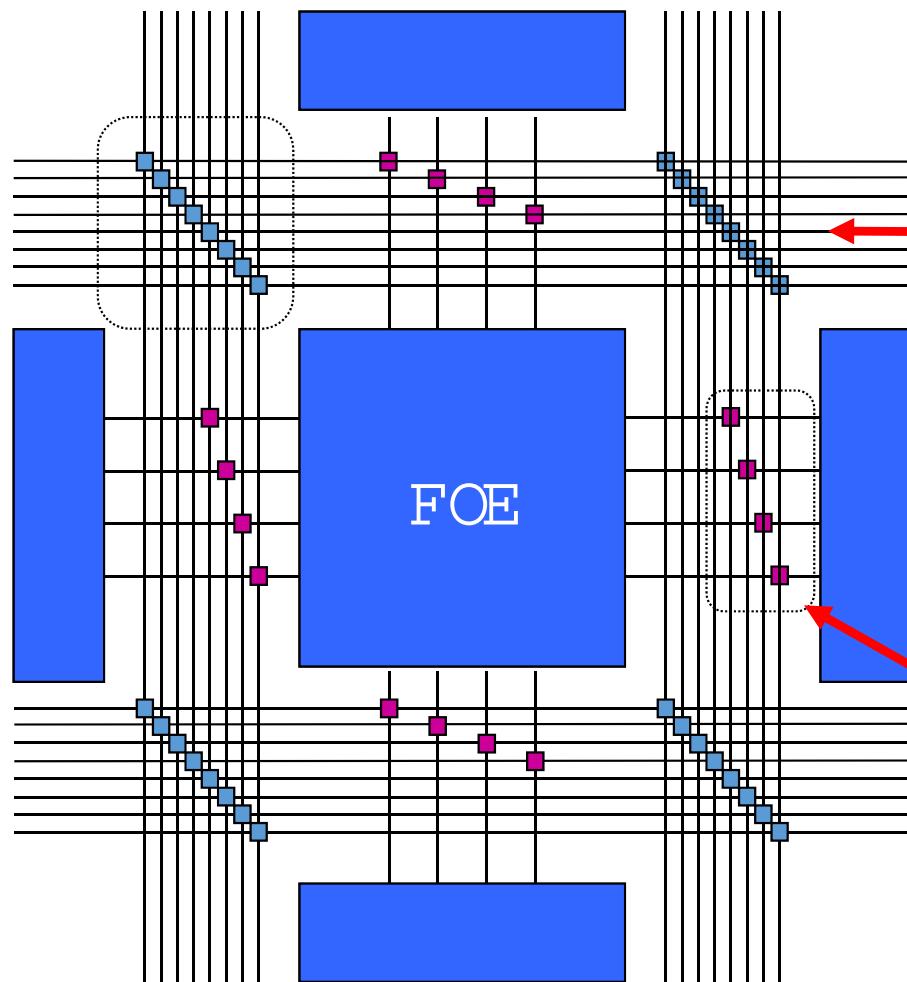
x0	x1	x2	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



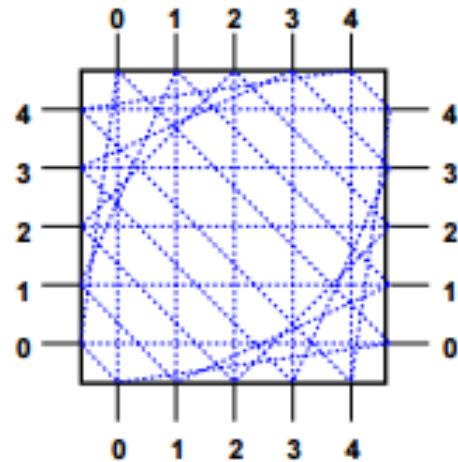
# LUT Structure



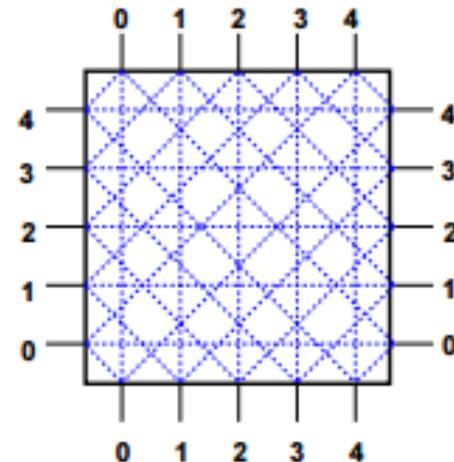
# Channel and Switch Block



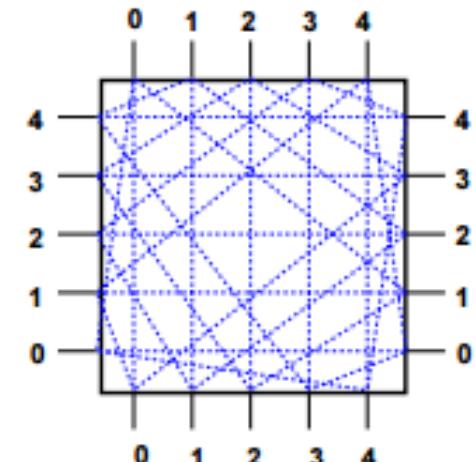
# Variation of a Switch Block



**Figure 3.2(a): Disjoint**



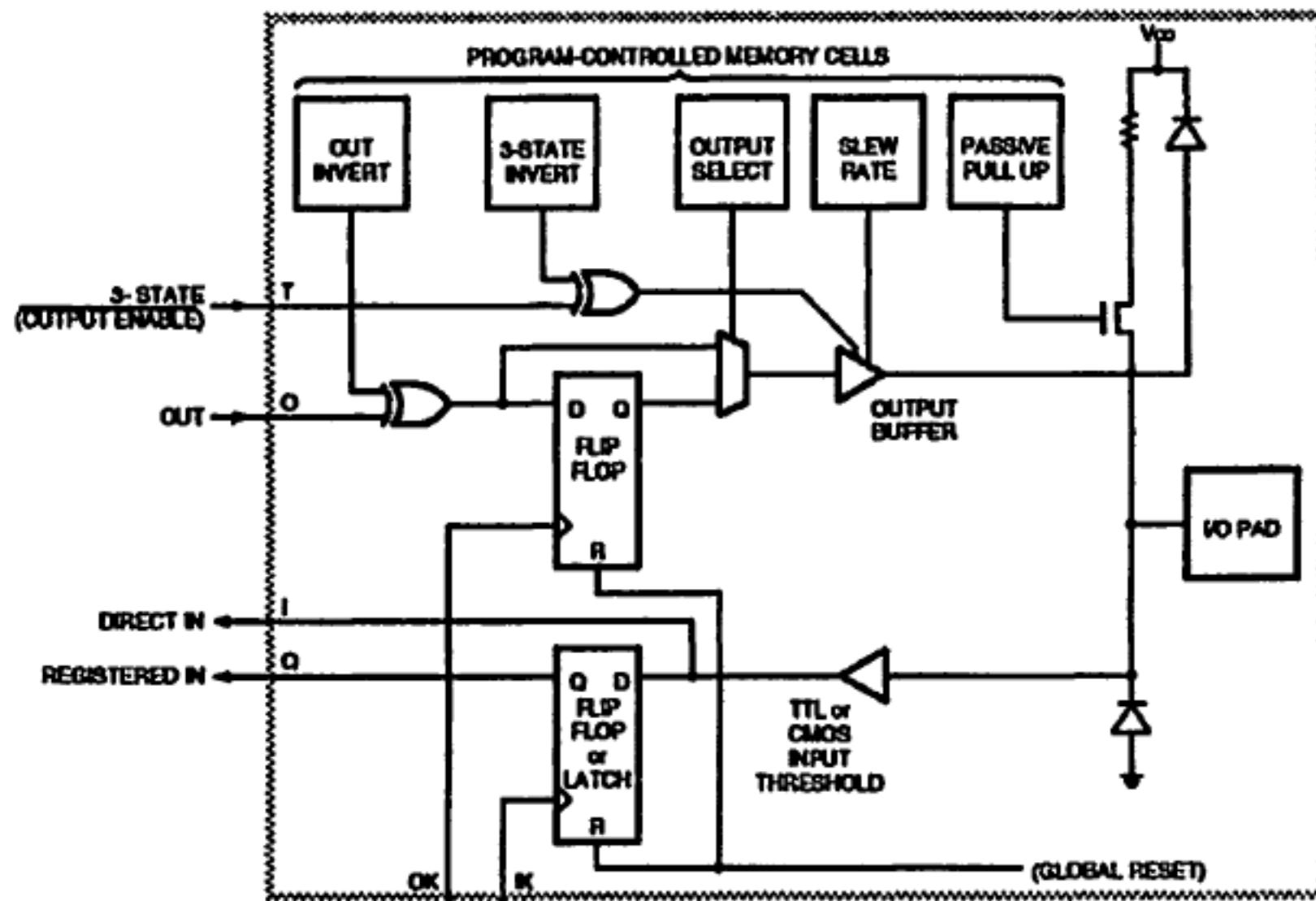
**Figure 3.2(b): Universal**



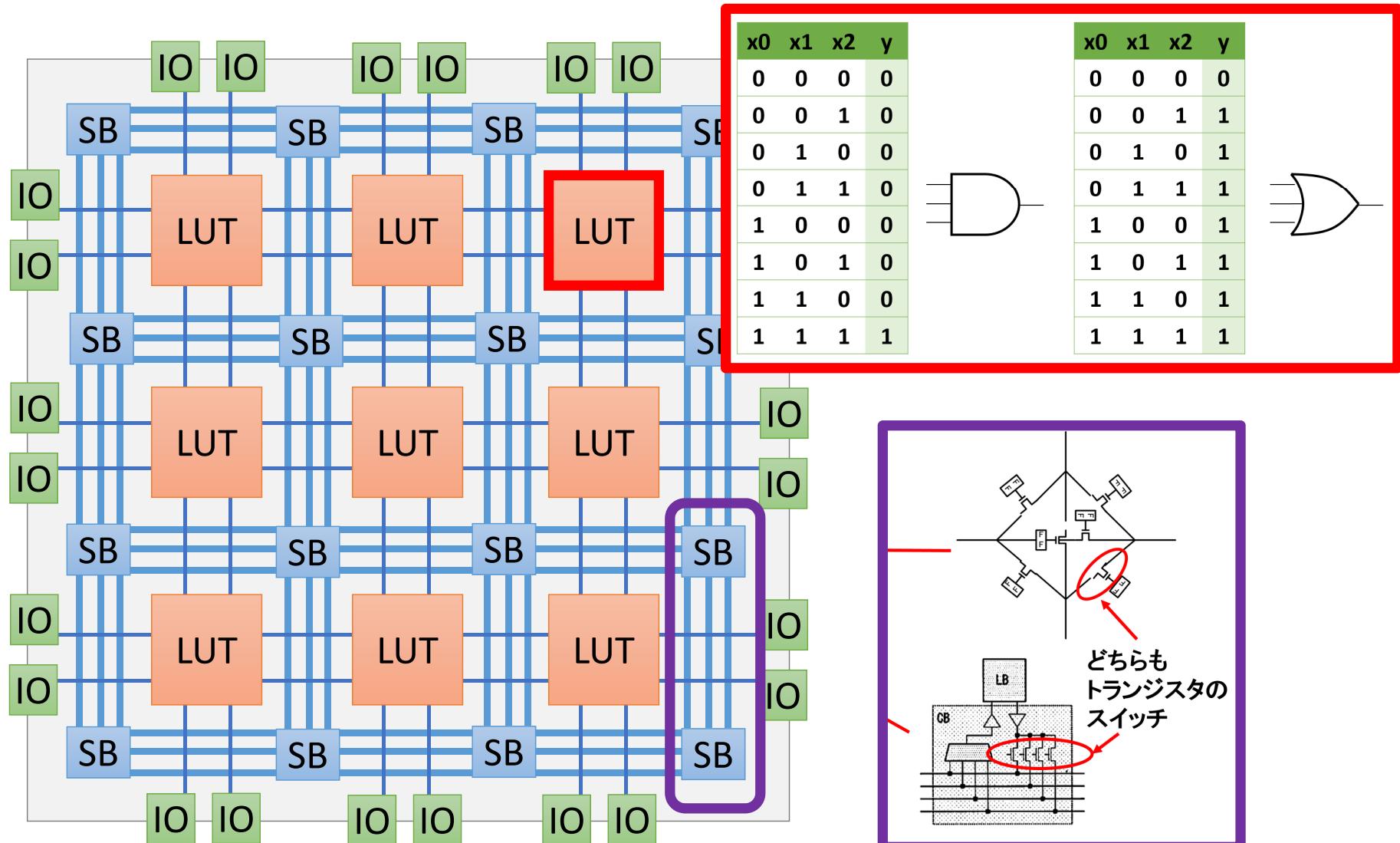
**Figure 3.2(c): Wilton**

Steven J. E. Wilton, "Architecture and Algorithms for Field Programmable Gate Arrays with Embedded Memory," Phd thesis, University of Toronto, 1997.

# I/O Block

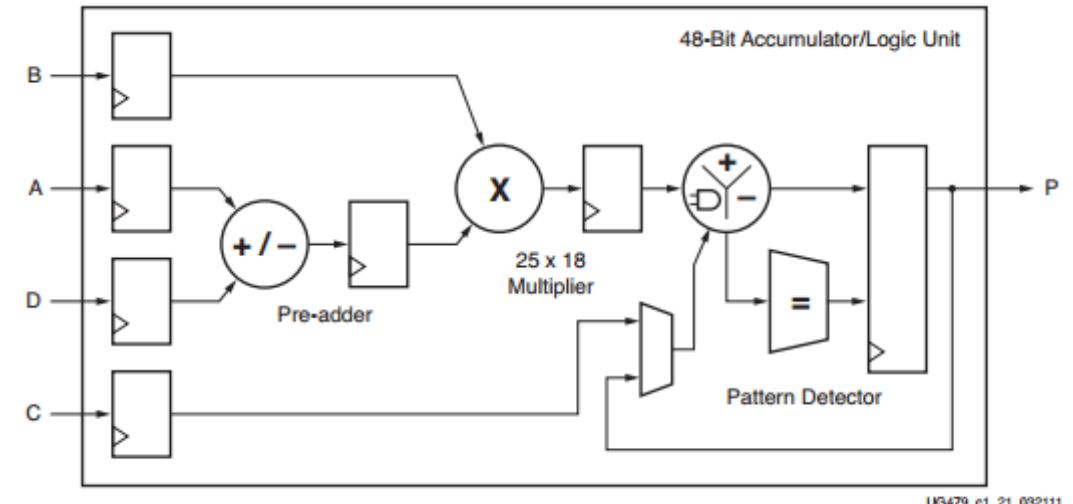
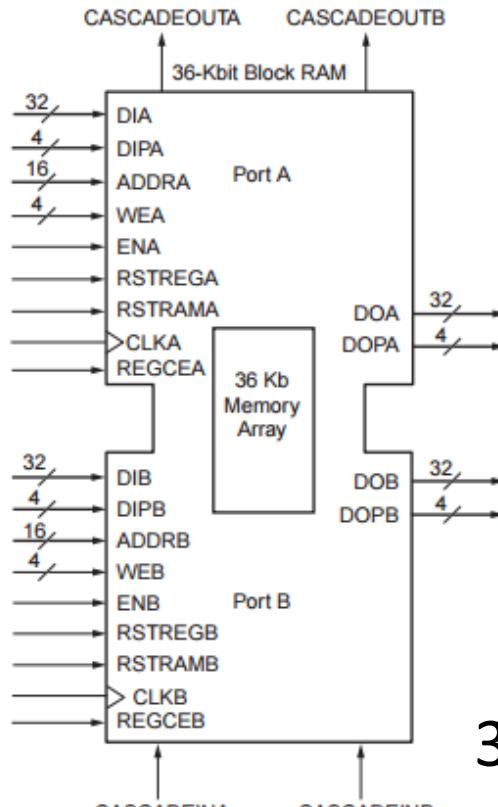


# Memory-based realizes “programmable”



# Hard macro (Dedicated Circuit)

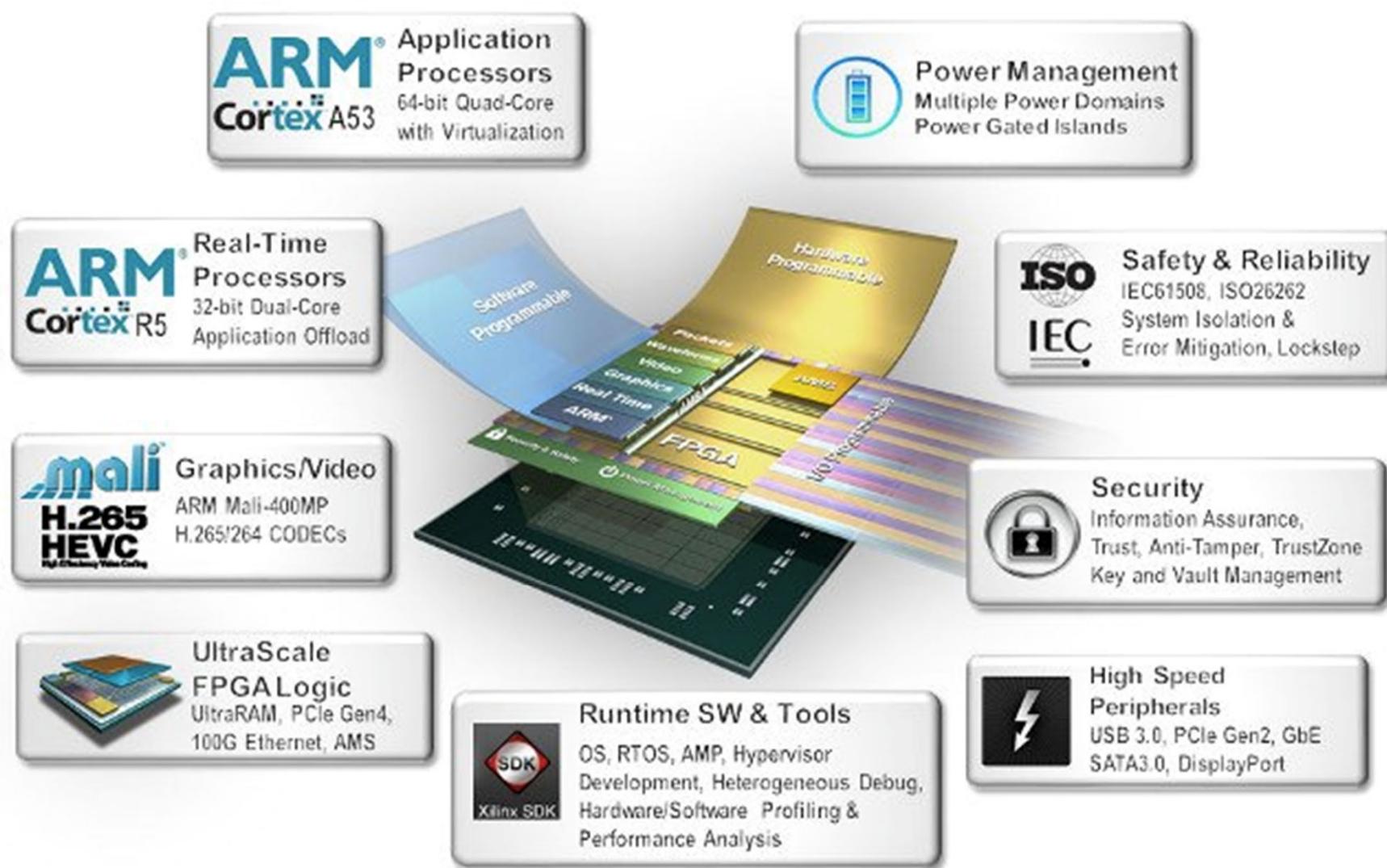
- Standard IP cores are “Pre-built” in the FPGA



48bit DSP Block

36Kb Block Memory

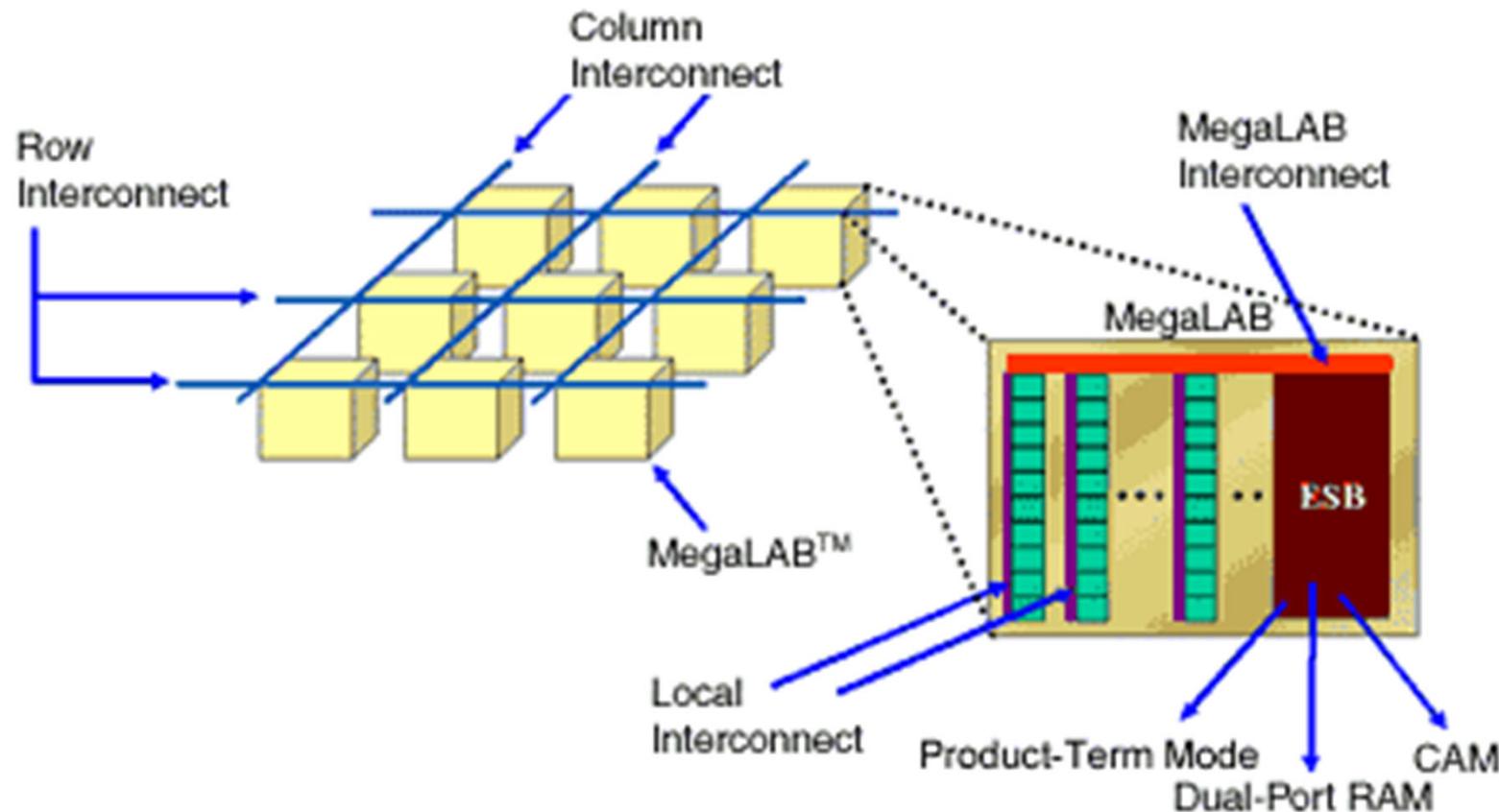
# Xilinx Zynq Ultra Scale+



# Pros. and Cons.

- Pros.
  - Short TAT(Turn-Around Time)
  - Small NRE (Non Recurrent Expense) Fee
  - Logic and Timing Design are required.
  - Full amount of IP (Intellectual Property)
- Cons.
  - Slow speed and Large Chip Area
  - High cost for volume manufacturing

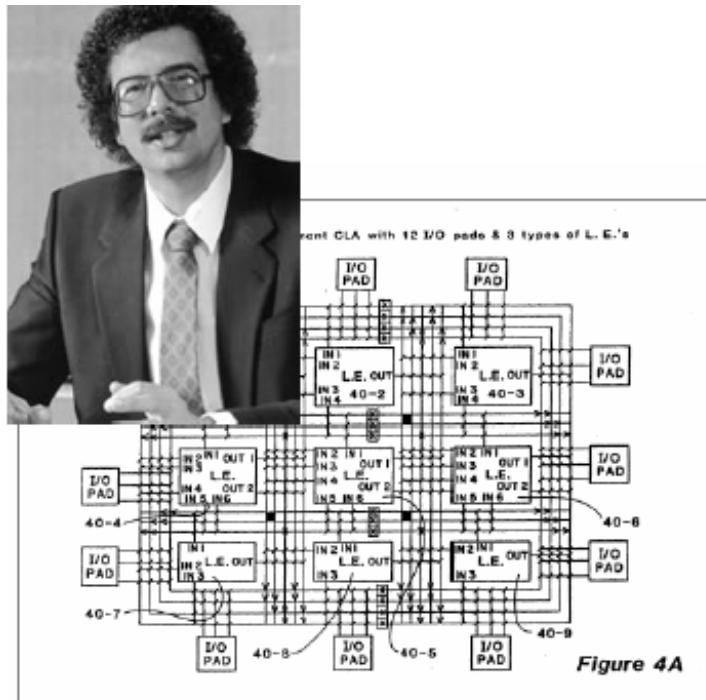
# Altera FPGA (Hierarchical Structure)



# FPGA-world Famous Patents

US RE34363

# by Ross H. Freeman Island style architecture

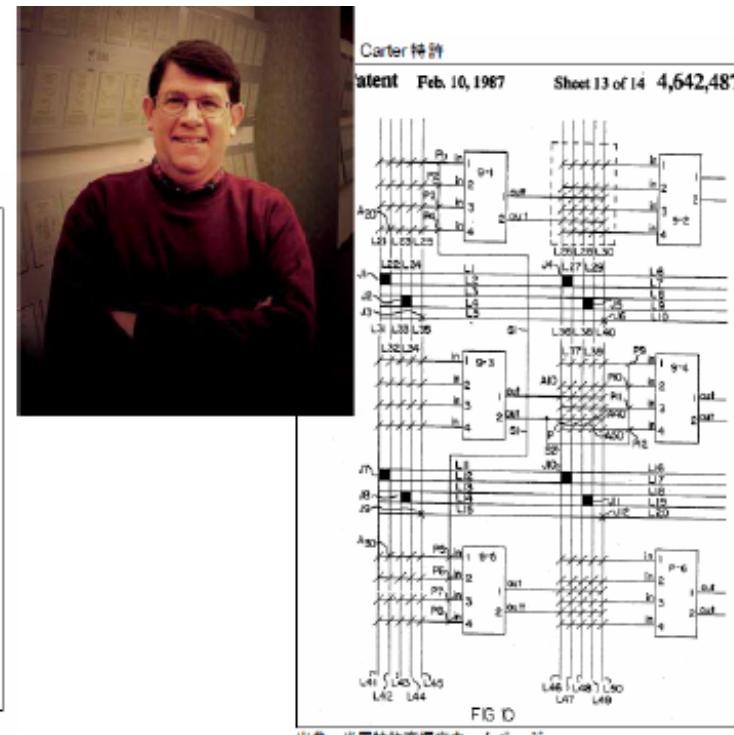


出典：米国特許商標庁ホームページ

US 464248

# by William S. Carter

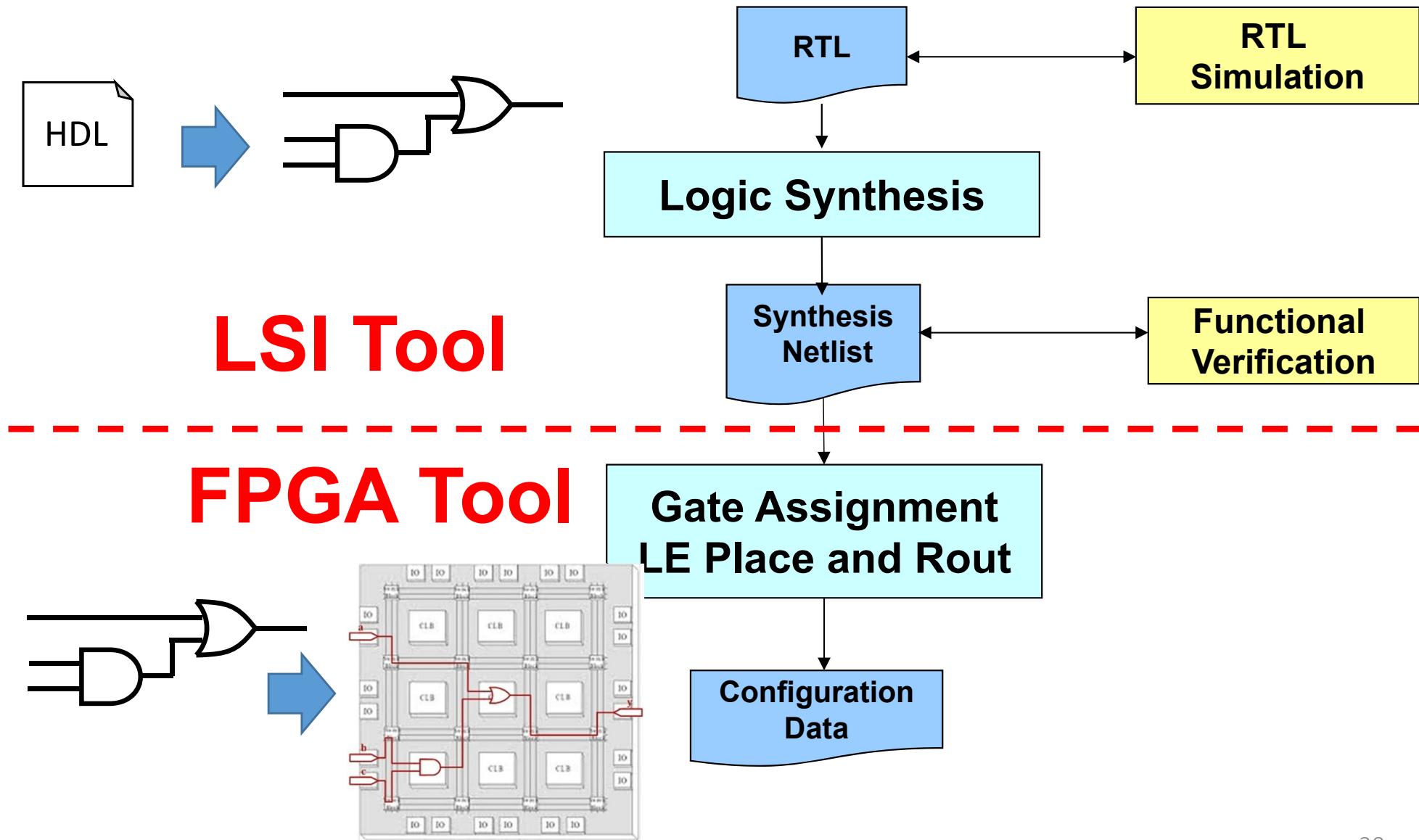
## Connection architecture



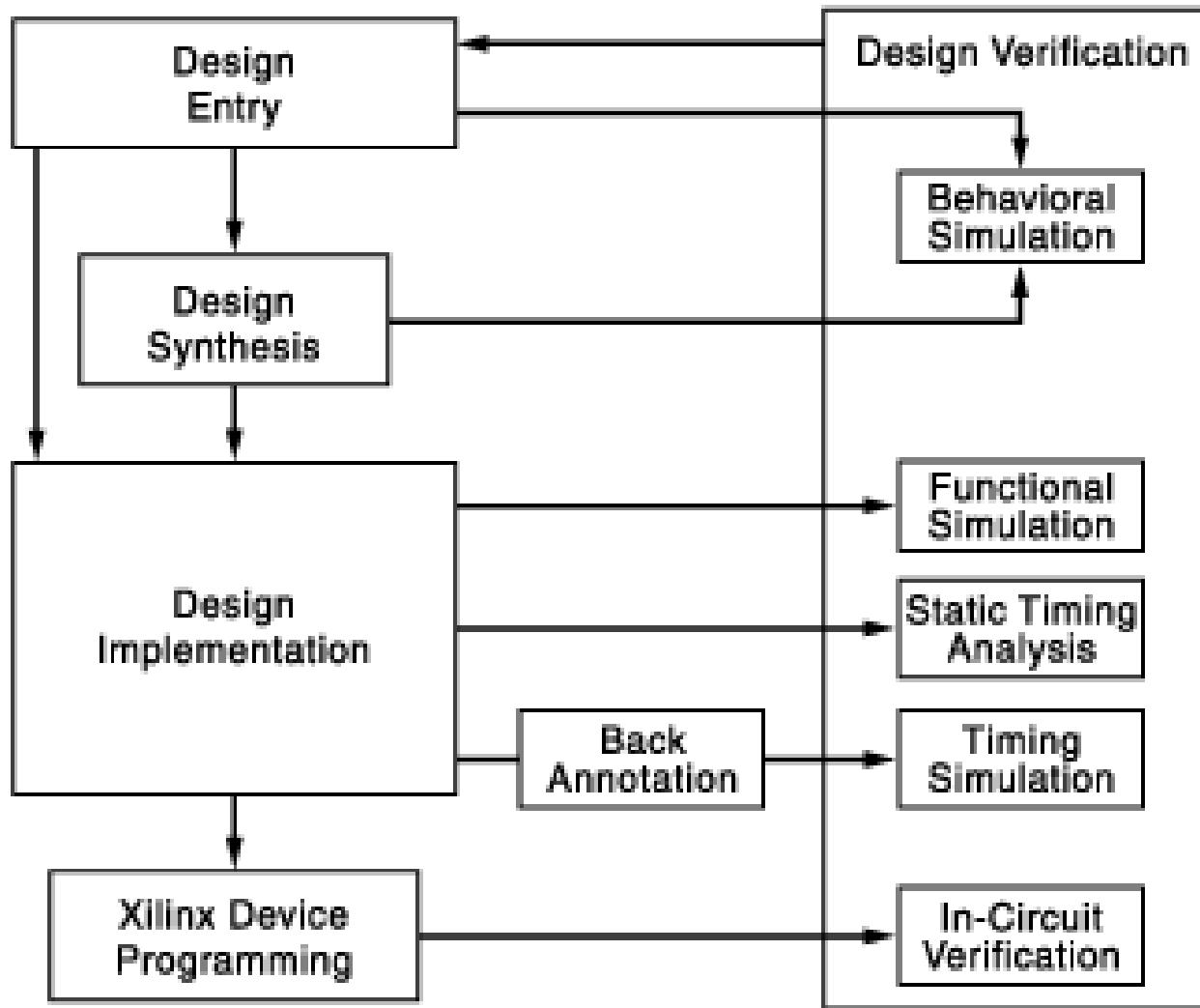
出典：米国特許商標庁ホームページ

## 2. Standard FPGA Design

# Standard FPGA Design



# FPGA Design Flow



# How to write a HDL?

$$Y = X \cdot \text{dot}(W) + B$$



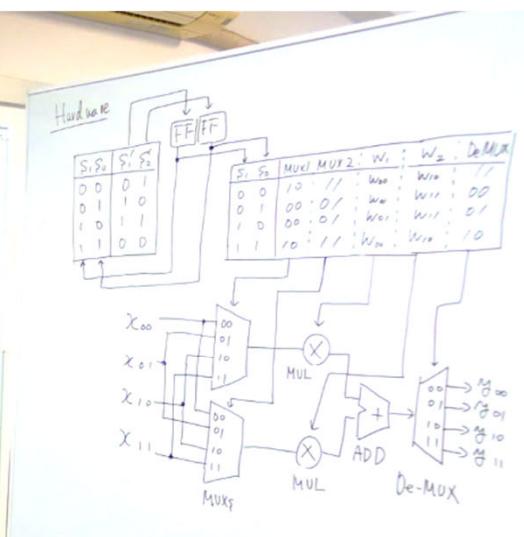
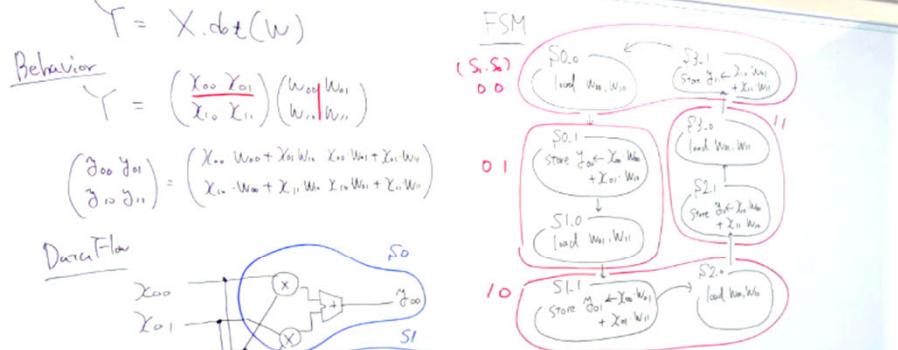
Behavior

$$Y = X \cdot \text{dot}(W)$$

$$Y = \begin{pmatrix} Y_{00} \\ Y_{01} \\ Y_{10} \\ Y_{11} \end{pmatrix} = \begin{pmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{pmatrix} \begin{pmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{pmatrix}$$

$$\begin{pmatrix} Y_{00} \\ Y_{01} \\ Y_{10} \\ Y_{11} \end{pmatrix} = \begin{pmatrix} X_{00} \cdot W_{00} + X_{01} \cdot W_{01} & X_{00} \cdot W_{01} + X_{01} \cdot W_{11} \\ X_{10} \cdot W_{00} + X_{11} \cdot W_{01} & X_{10} \cdot W_{01} + X_{11} \cdot W_{11} \end{pmatrix}$$

Data Flow



Then, you can write your HDL!



# Matrix Multiplication

$$Y = X \cdot \text{dot}(W)$$

Behavior

$$Y = \begin{pmatrix} \chi_{00} & \chi_{01} \\ \chi_{10} & \chi_{11} \end{pmatrix} \begin{pmatrix} W_{00} & | & W_{01} \\ W_{10} & | & W_{11} \end{pmatrix}$$

$$\begin{pmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \end{pmatrix} = \begin{pmatrix} \chi_{00} \cdot W_{00} + \chi_{01} \cdot W_{10} & \chi_{00} \cdot W_{01} + \chi_{01} \cdot W_{11} \\ \chi_{10} \cdot W_{00} + \chi_{11} \cdot W_{10} & \chi_{10} \cdot W_{01} + \chi_{11} \cdot W_{11} \end{pmatrix}$$

Alternatively,  
you can write a C-code

```
for(i=0;i<2;++i){  
    for(j=0;j<2;++j){  
        y[i][j] = x[i][j] * w[i][j];  
  
        // Compute terms  
        for(i=0;i<2;i++){  
            for(j=0;j<2;j++){  
                term = 0;  
                for(k=0;k<2;k++)  
                    term = term + x[i][k]*w[k][j];  
                y[i][j] = term;  
            }  
        }  
    }  
}
```

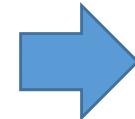
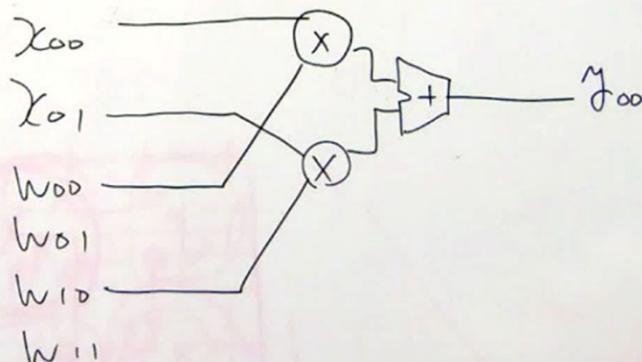
# Data path description

Behavior

$$Y = \begin{pmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{pmatrix} \begin{pmatrix} W_{00} & | & W_{01} \\ W_{10} & | & W_{11} \end{pmatrix}$$

$$\begin{pmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{pmatrix} = \begin{pmatrix} X_{00} \cdot W_{00} + X_{01} \cdot W_{10} & X_{00} \cdot W_{01} + X_{01} \cdot W_{11} \\ X_{10} \cdot W_{00} + X_{11} \cdot W_{10} & X_{10} \cdot W_{01} + X_{11} \cdot W_{11} \end{pmatrix}$$

Data Flow

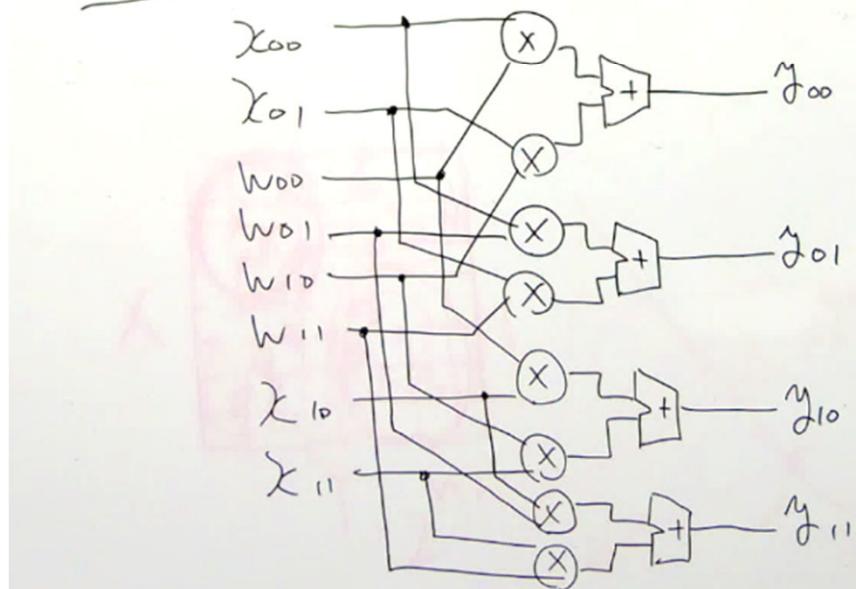


Behavior

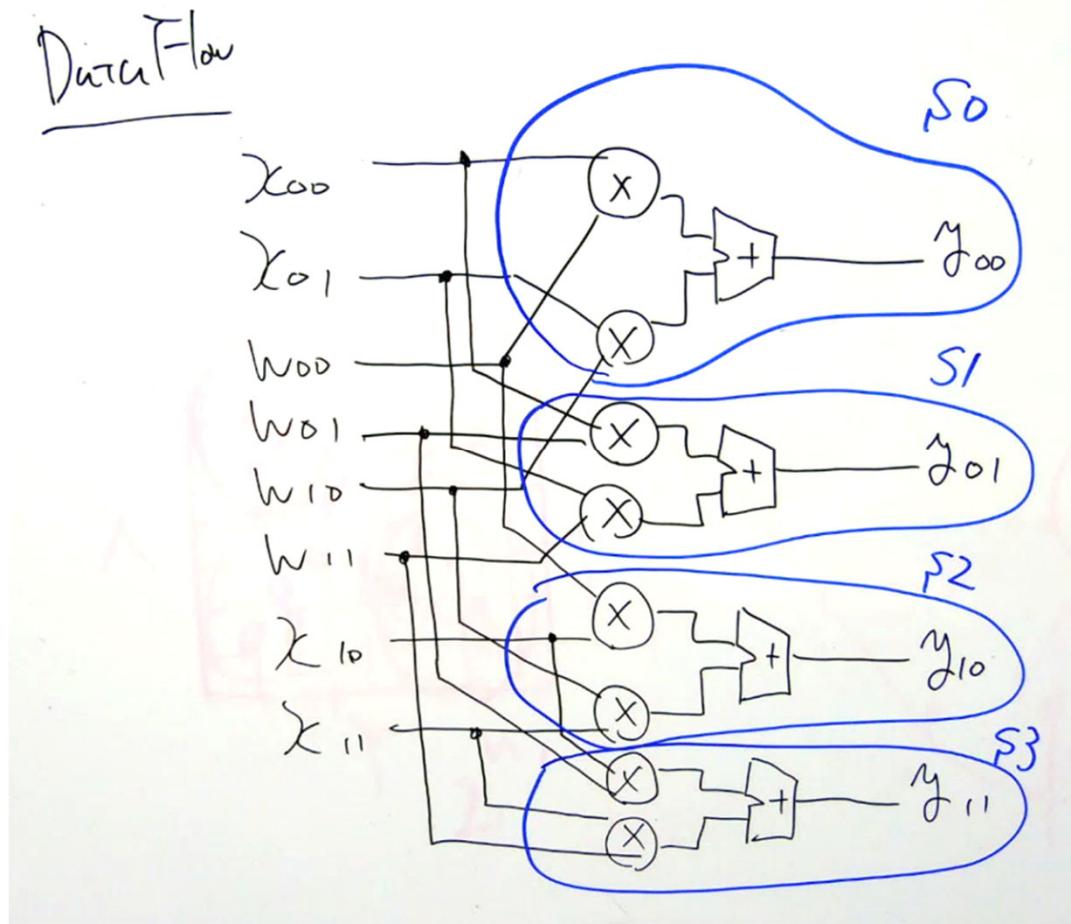
$$Y = \begin{pmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{pmatrix} \begin{pmatrix} W_{00} & | & W_{01} \\ W_{10} & | & W_{11} \end{pmatrix}$$

$$\begin{pmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{pmatrix} = \begin{pmatrix} X_{00} \cdot W_{00} + X_{01} \cdot W_{10} & X_{00} \cdot W_{01} + X_{01} \cdot W_{11} \\ X_{10} \cdot W_{00} + X_{11} \cdot W_{10} & X_{10} \cdot W_{01} + X_{11} \cdot W_{11} \end{pmatrix}$$

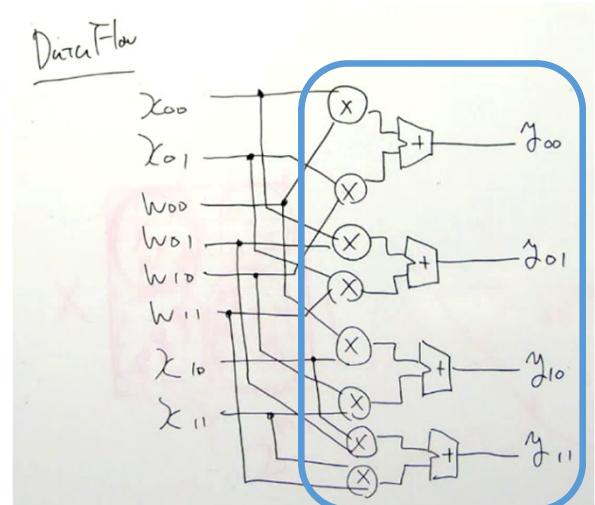
Data Flow



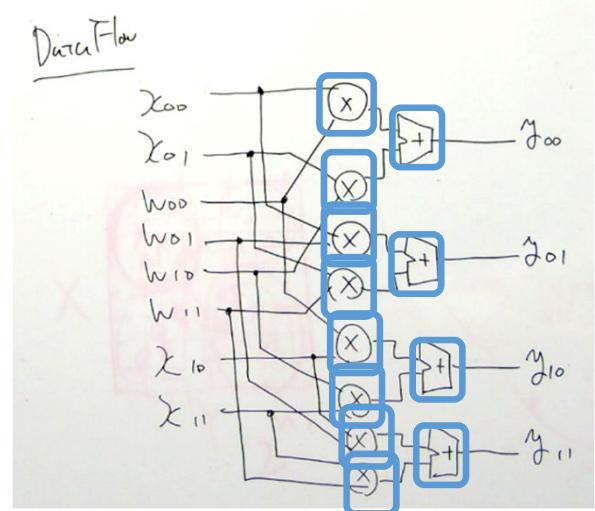
# Scheduling



MACs (Multiply ACCumulations)  
realization

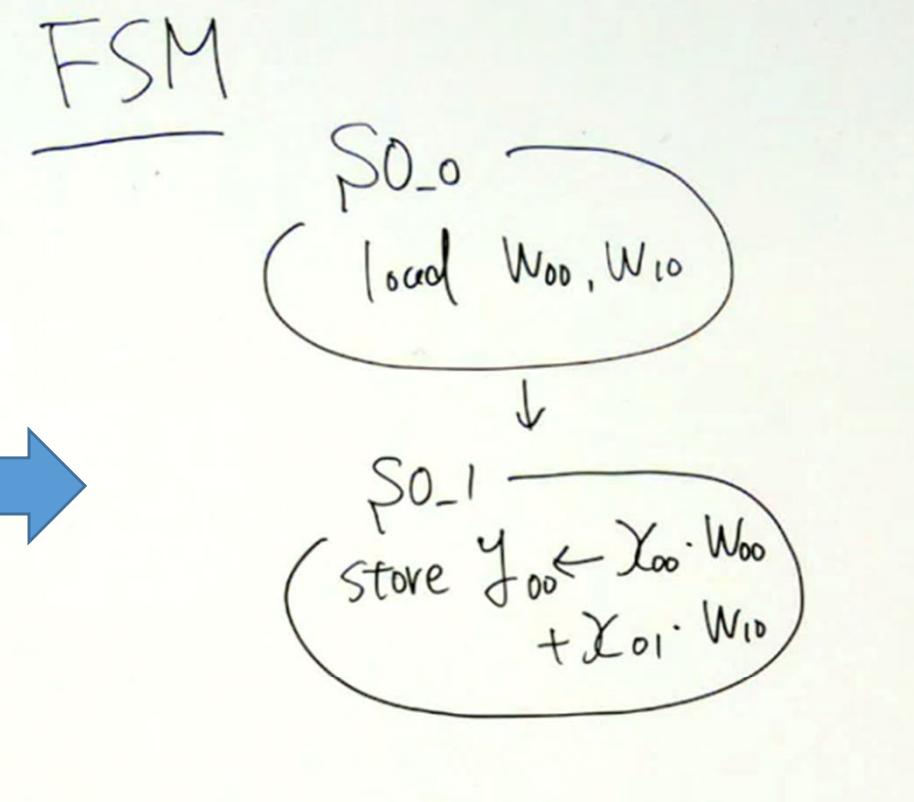
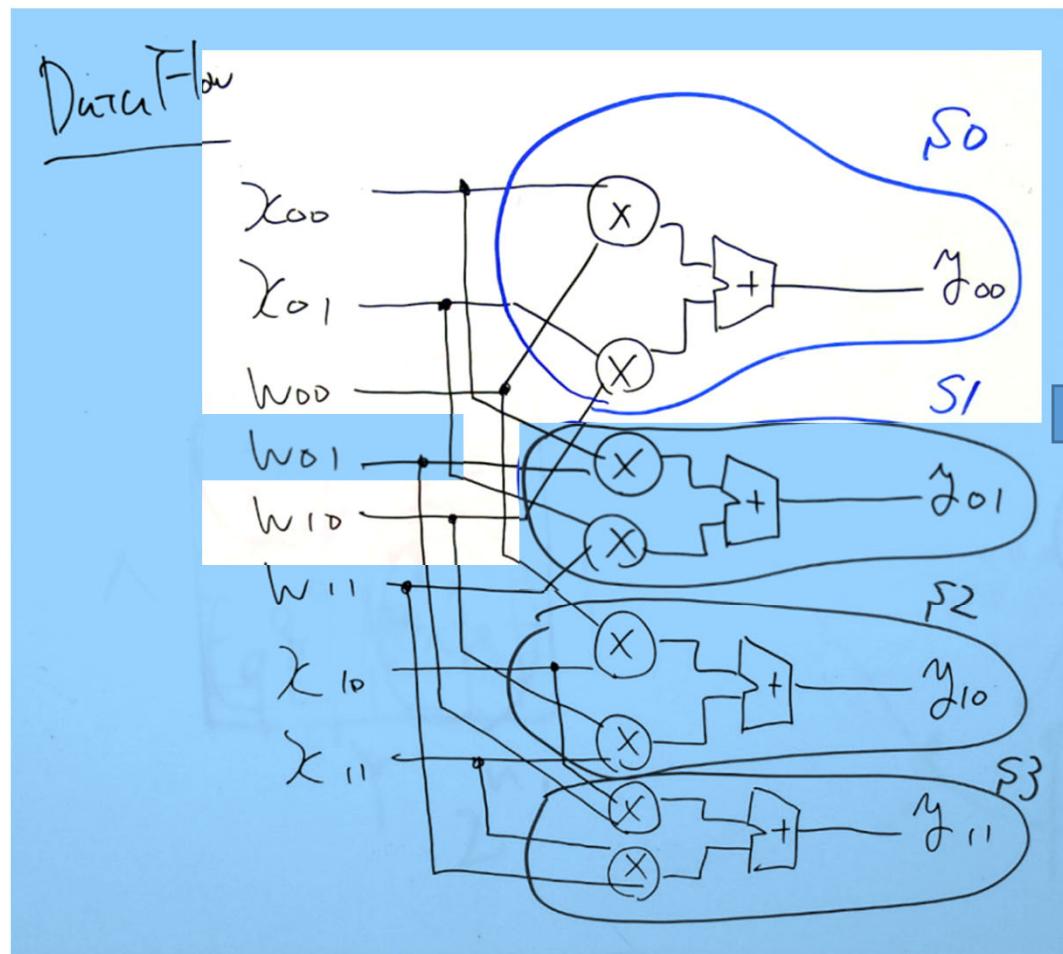


Fully parallel realization

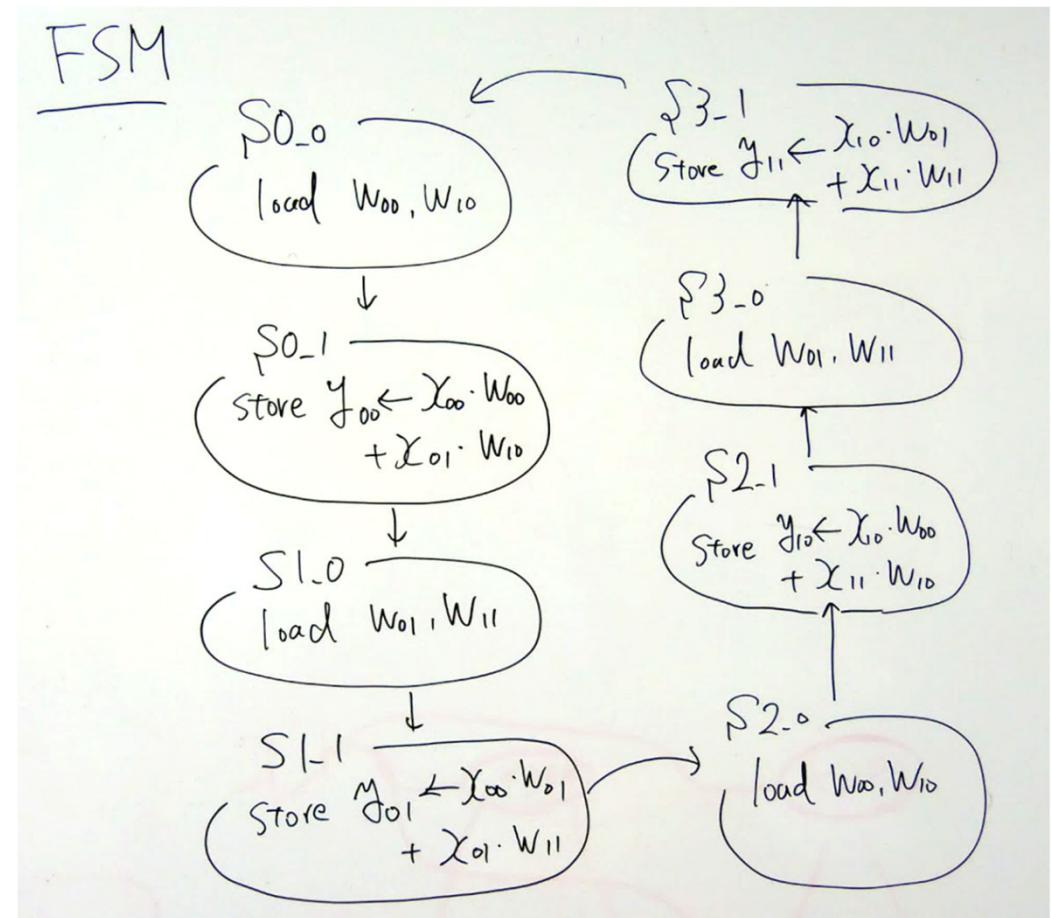
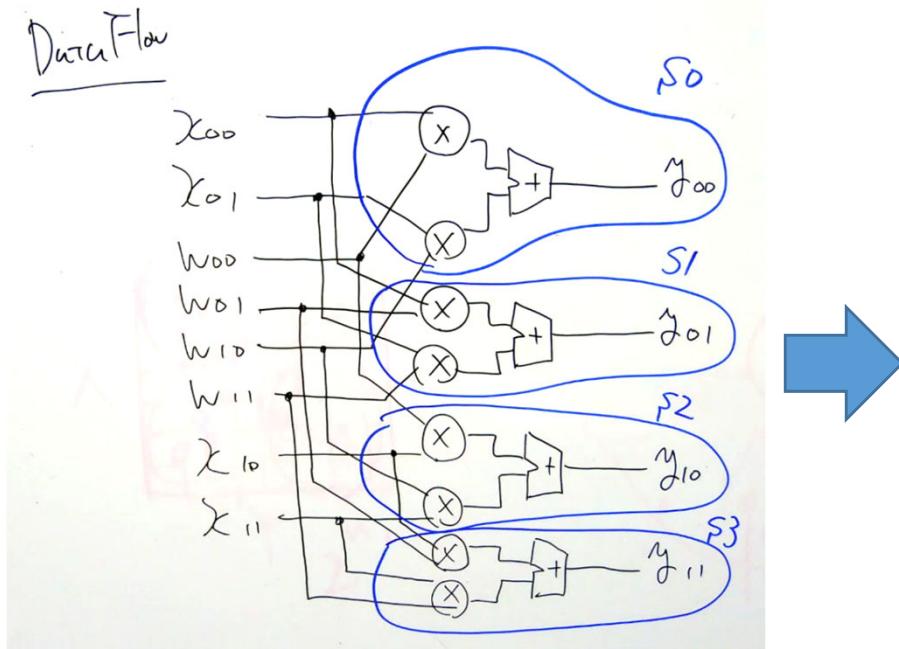


Sequential realization

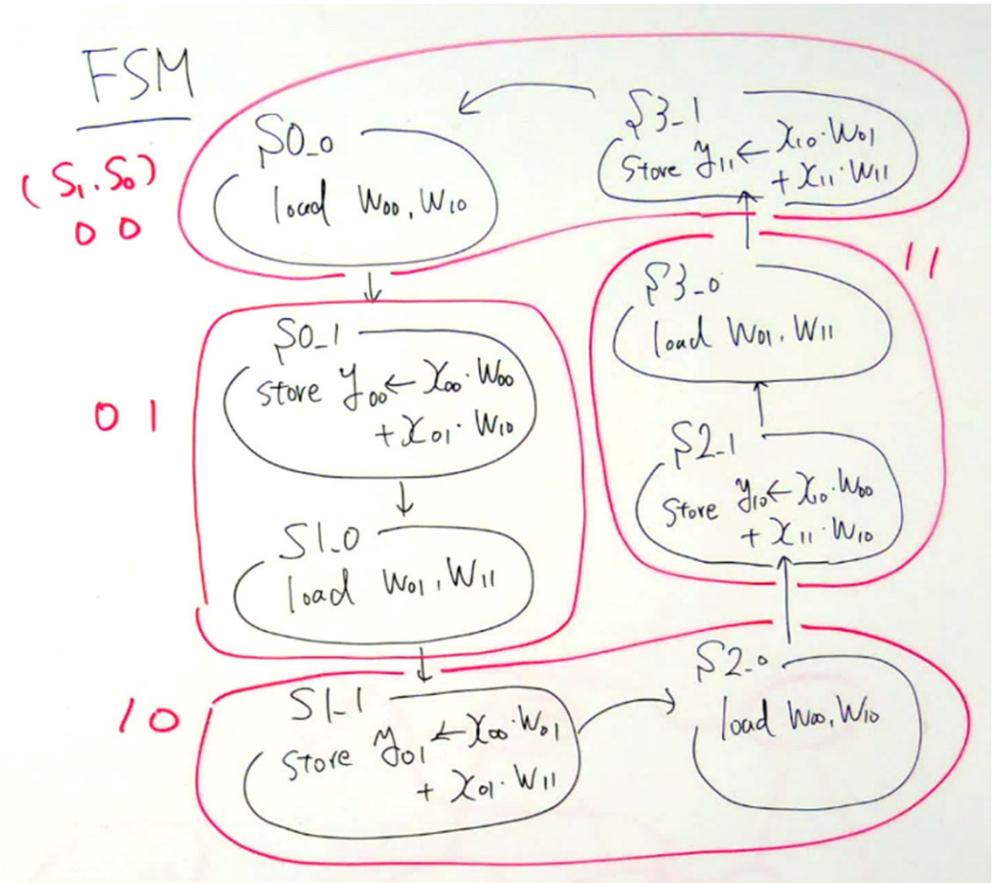
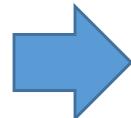
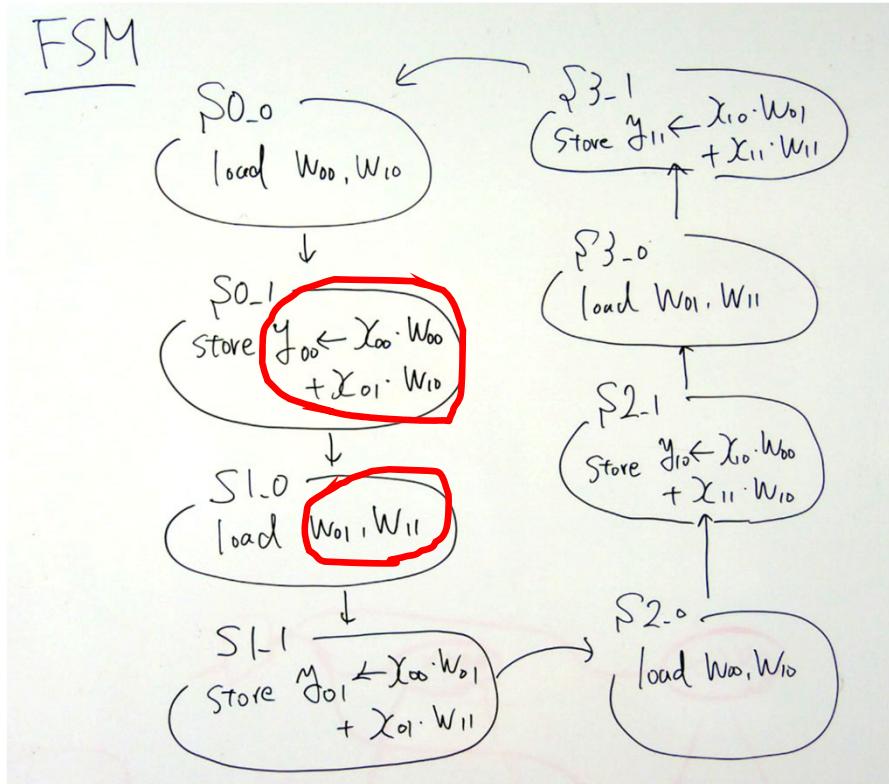
# Finite State Machine (FSM)



# FSM (Cont.)



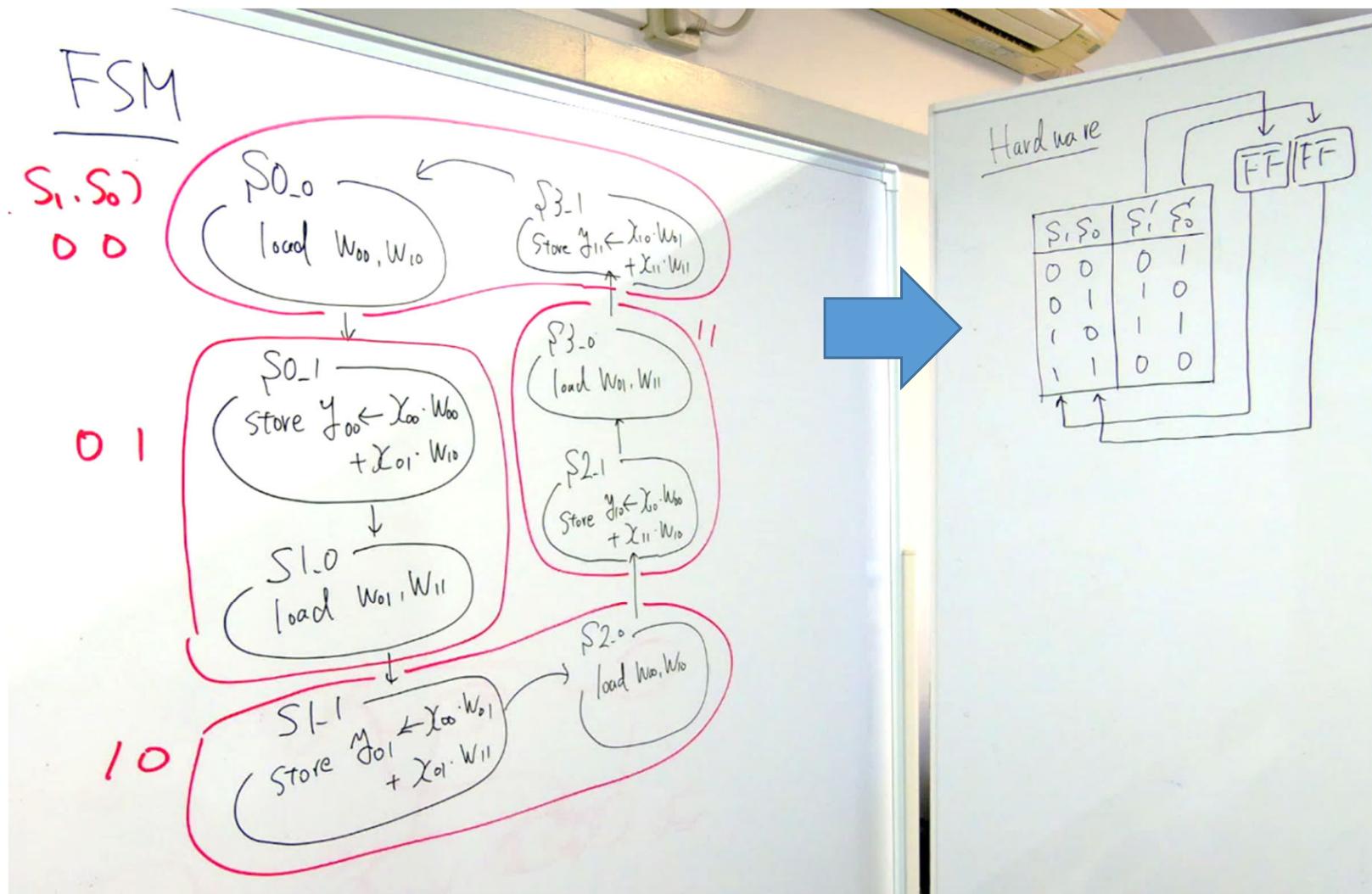
# FSM Optimization



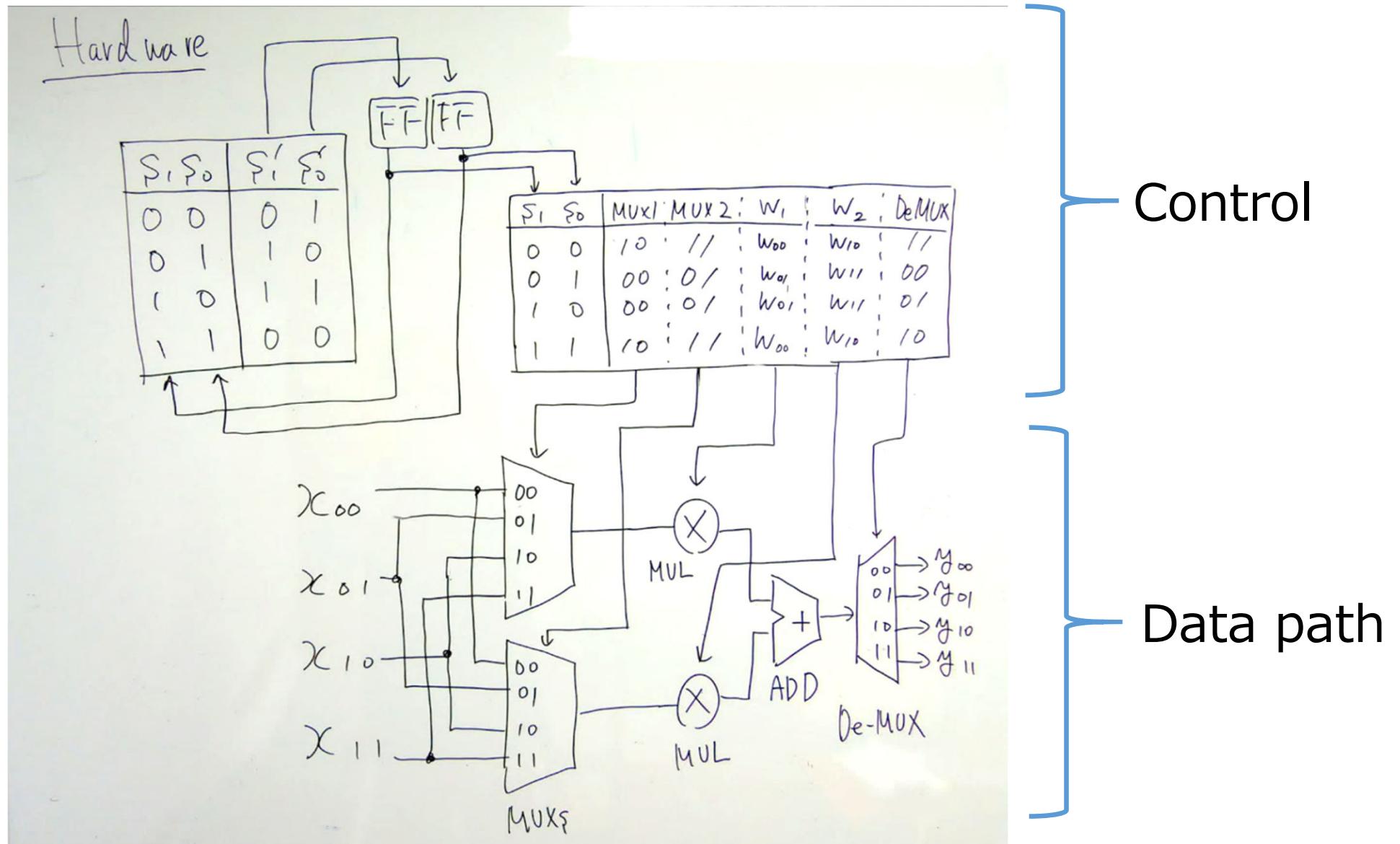
Load weights during the MAC operation

Assign state variables  
(1-hot code, gray one,  
natural binary one)

# Realization of the FSM



# Assign to Primitives



# Comparison of # Lines

$$Y = X \cdot \text{dot}(W) + B$$

Python: single line!

```
for(i=0;i<2;++i){  
    for(j=0;j<2;++j){  
        y[i][j] = x[i][j] * w[i][j];  
  
        // Compute terms  
        for(i=0;i<2;i++){  
            for(j=0;j<2;j++){  
                term = 0;  
                for(k=0;k<2;k++)  
                    term = term + x[i][k]*w[k][j];  
                y[i][j] = term;  
            }  
        }  
    }  
}
```

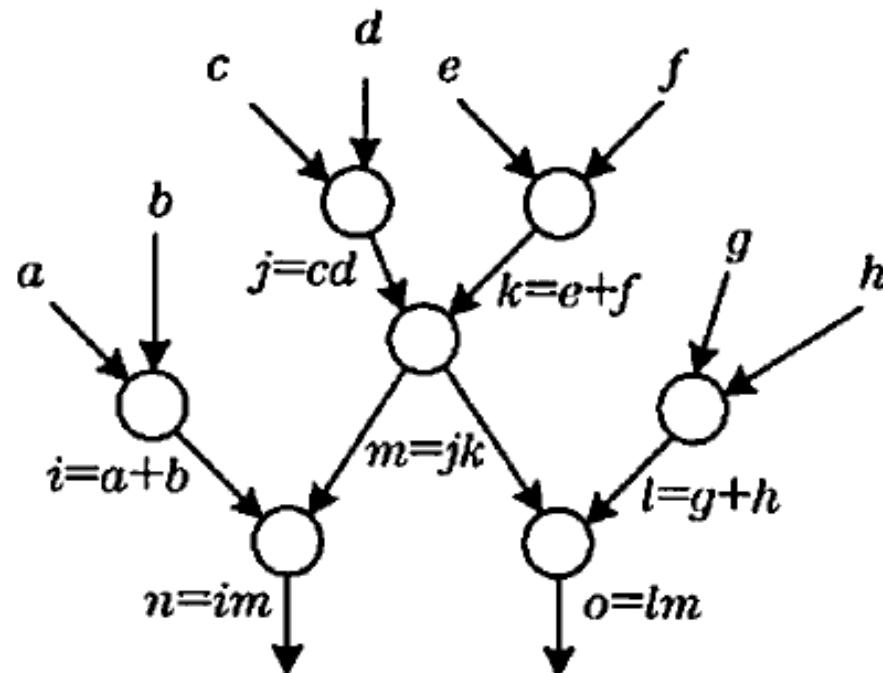
C/C++: ten lines

```
module mat_add(  
    input clk, reset,  
    input [7:0]x[0:3],  
    output [7:0]y[0:3]  
>;  
  
reg [1:0]state;  
reg [1:0]mux1, mux2;  
reg [7:0]w0, w1;  
reg [1:0]de_mux;  
  
always@(posedge clk or posedge rst)begin  
    if( rst == 1'b1)begin  
        state <= 2'b00  
    end else begin  
        case( state)  
            2'b00:begin  
                state <= 2'b01;  
                mux1 <= 2'b10;  
                mux2 <= 2'b11;  
                w0 <= 8'b00101000;  
                w1 <= 8'b11000101;  
                de_mux <= 2'b11;  
            end  
            2'b01:begin  
                state <= 2'b10;  
                mux1 <= 2'b00;  
                mux2 <= 2'b01;  
                w0 <= 8'b00101000;  
                w1 <= 8'b11000101;  
                de_mux <= 2'b00;  
            end  
            2'b10:begin  
                state <= 2'b11;  
                mux1 <= 2'b00;  
                mux2 <= 2'b01;  
                w0 <= 8'b00101000;  
                w1 <= 8'b11000101;  
                de_mux <= 2'b01;  
            end  
            2'b11:begin  
                state <= 2'b00;  
                mux1 <= 2'b00;  
                mux2 <= 2'b01;  
                w0 <= 8'b00101000;  
                w1 <= 8'b11000101;  
                de_mux <= 2'b10;  
            end  
        endcase  
    end  
  
wire [15:0]mul1, mul2;  
wire [16:0]w_add;  
  
assign mul1 = w0 * mux( mux1,x[0],x[1],x[2],x[3]);  
assign mul2 = w1 * mux( mux2,x[0],x[1],x[2],x[3]);  
  
assign w_add = mul1 + mul2;  
  
assign y[0] = (de_mux == 2'b00) ? w_add : 2'bzz;  
assign y[1] = (de_mux == 2'b01) ? w_add : 2'bzz;  
assign y[2] = (de_mux == 2'b10) ? w_add : 2'bzz;  
assign y[3] = (de_mux == 2'b11) ? w_add : 2'bzz;  
endmodule
```

Verilog-HDL: 66 lines

# Boolean Network

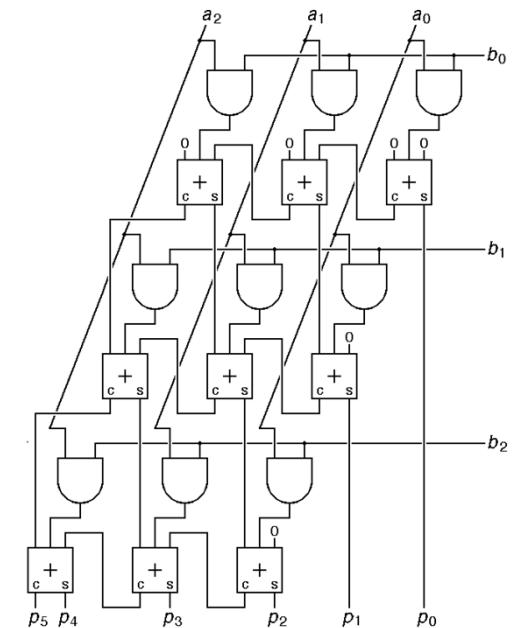
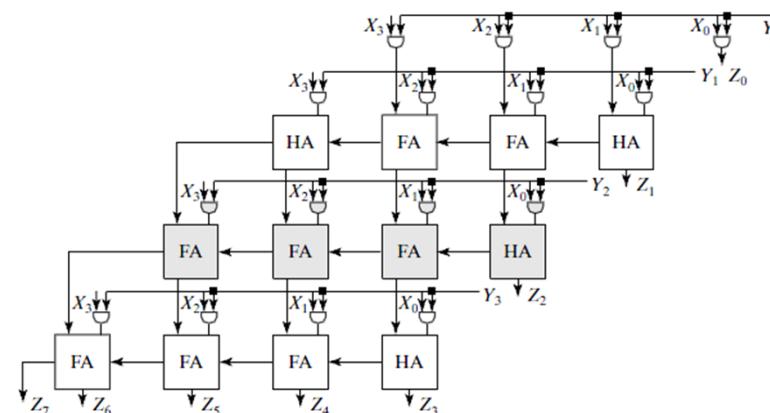
- Representation of a combinational logic circuit using a directed graph without a cycle
- Vertex : Logic gate, Edge : Input or output



# Logic Synthesis

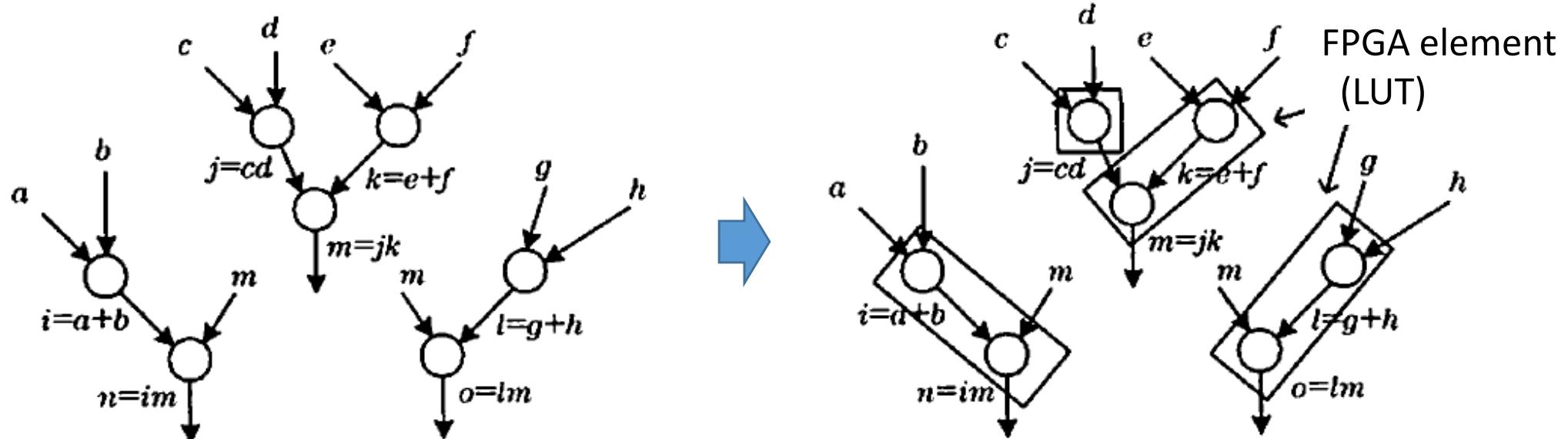
- Synthesize from a given HDL specification to a Boolean network

input [3:0]X,Y;  
output [7:0]Z;  
 $Z = X * Y$



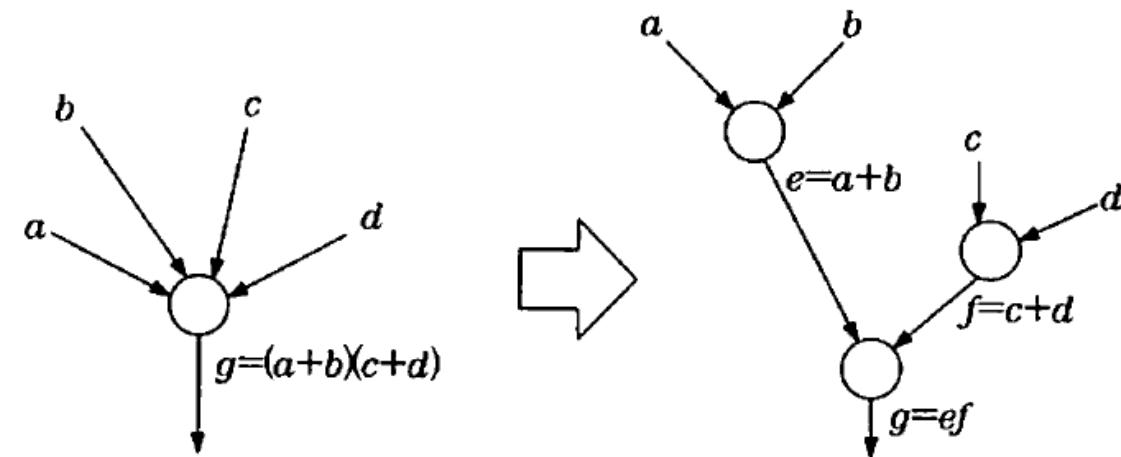
# Technology Mapping

- A kind of a graph covering problem
- Goal: A depth optimized one by using a dynamic programming

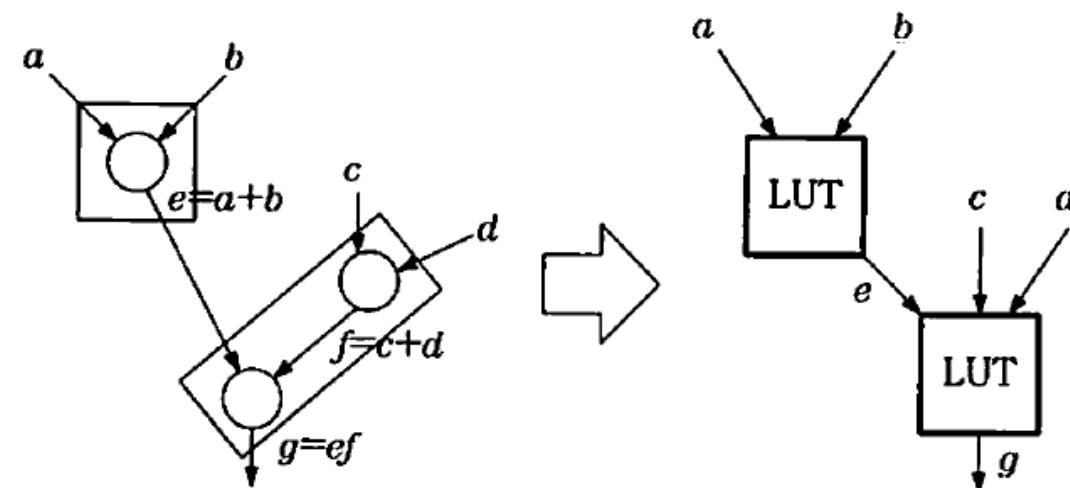


# Decomposition and Covering

Decomposition

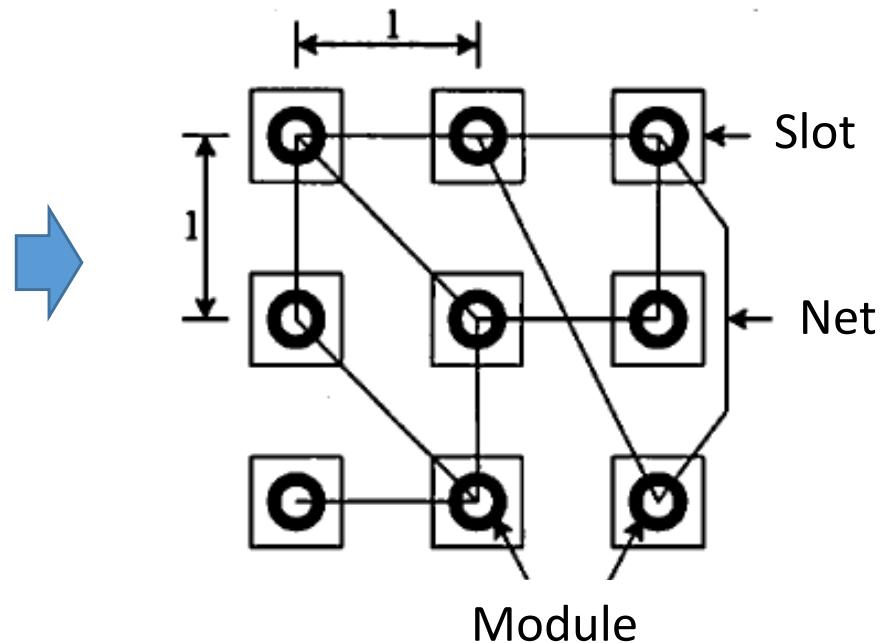
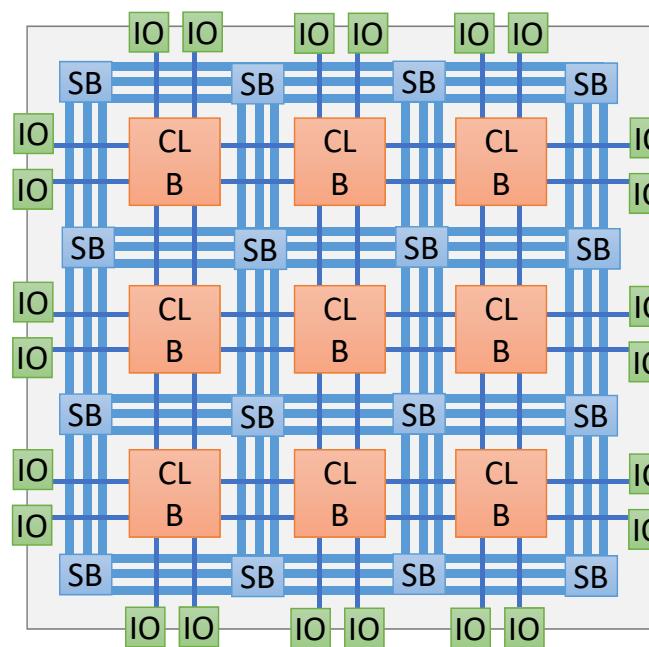


Covering



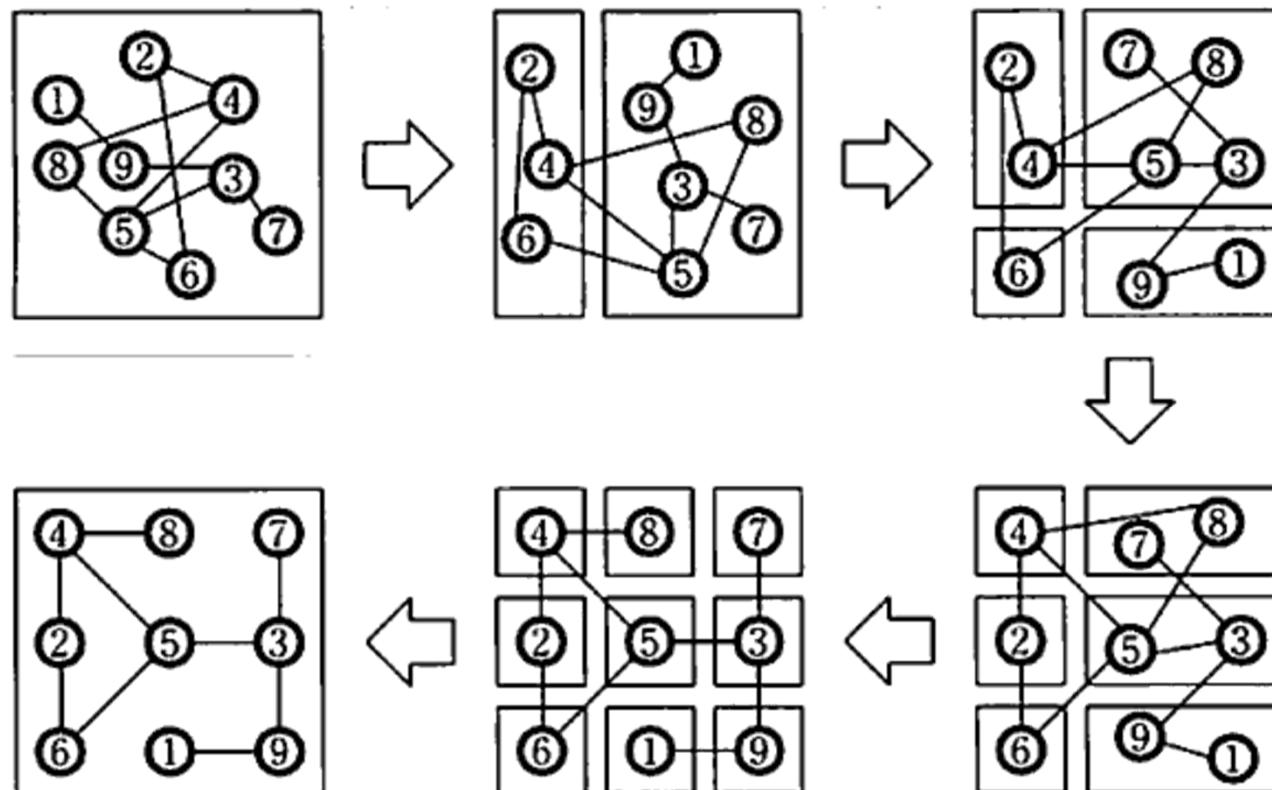
# Placement

- Problem to place the module (logic gate) into the slot (location)
  - 2D allocation problem → NP-complete
  - Approximation (Simulated annealing, or min-cut tech.)



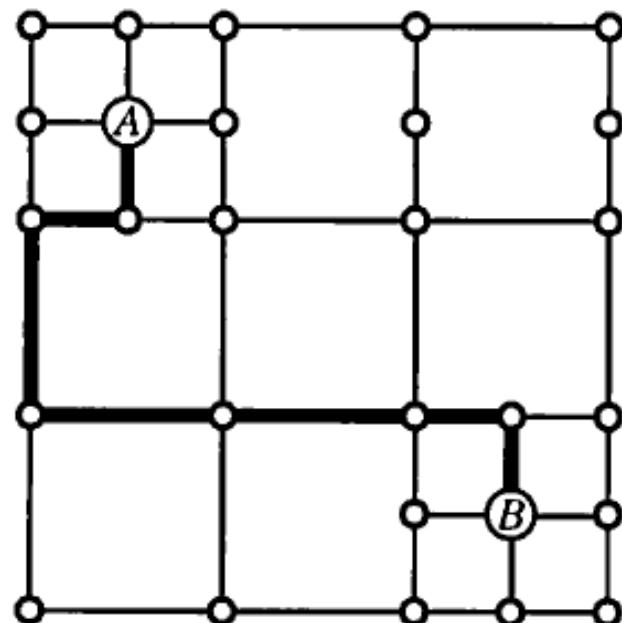
# Min-cut Placement

- Recursively divided a module set into two ones
  - Near-optimal solution can be obtained in the short time
- Used in a commercial FPGA design tool

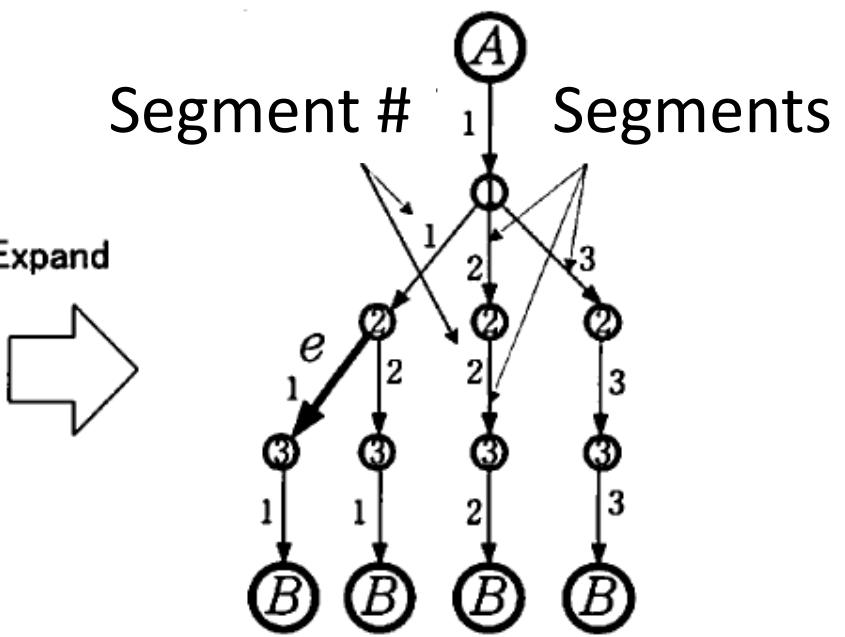
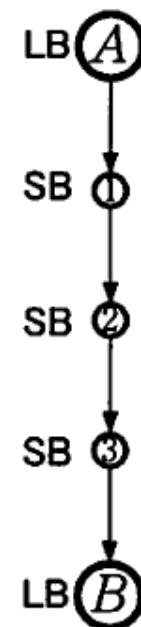


# Routing

- Global routing: Determine the rough wiring path
- Local one: Determine the wiring segment and switch

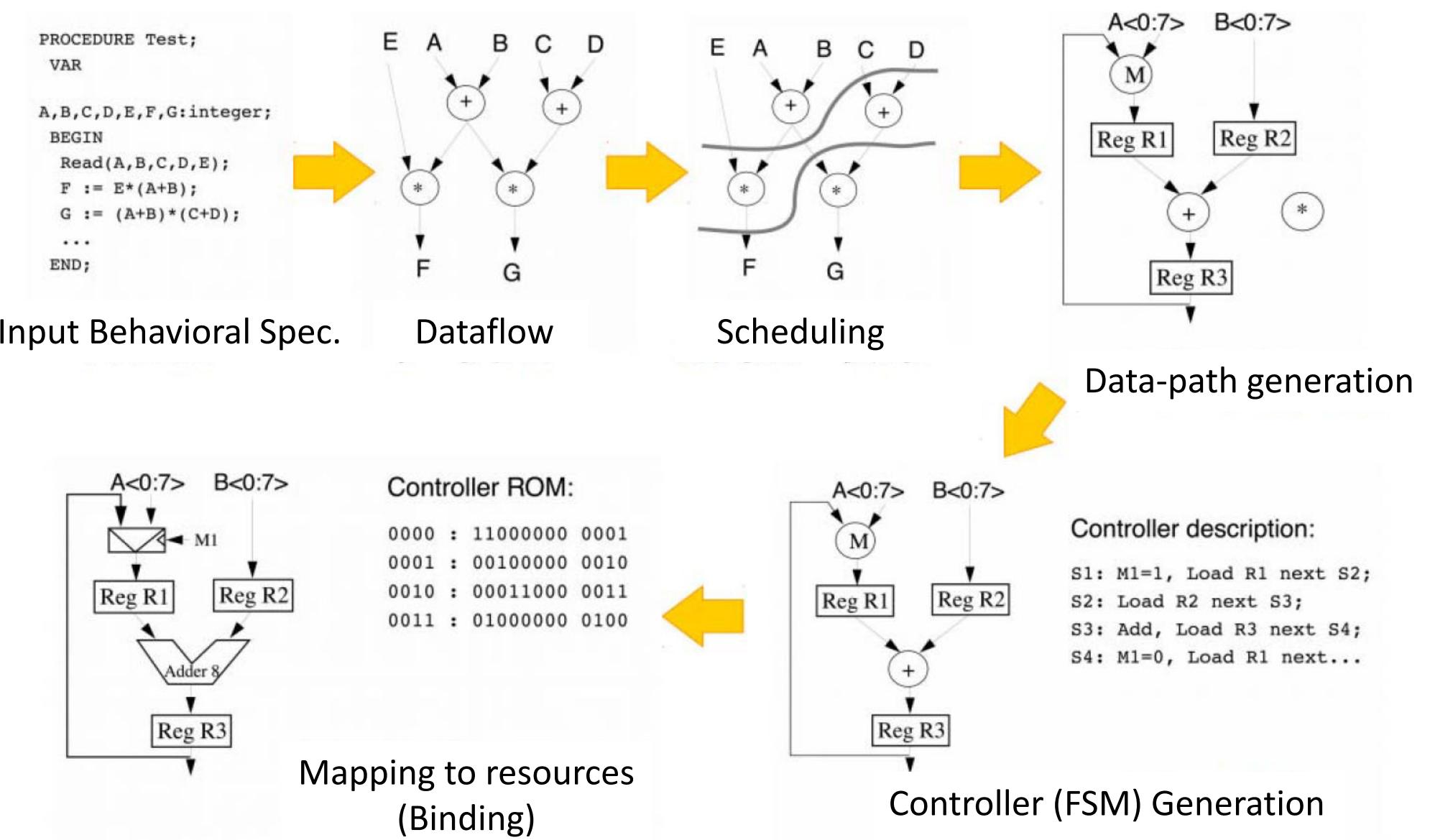


Global routing



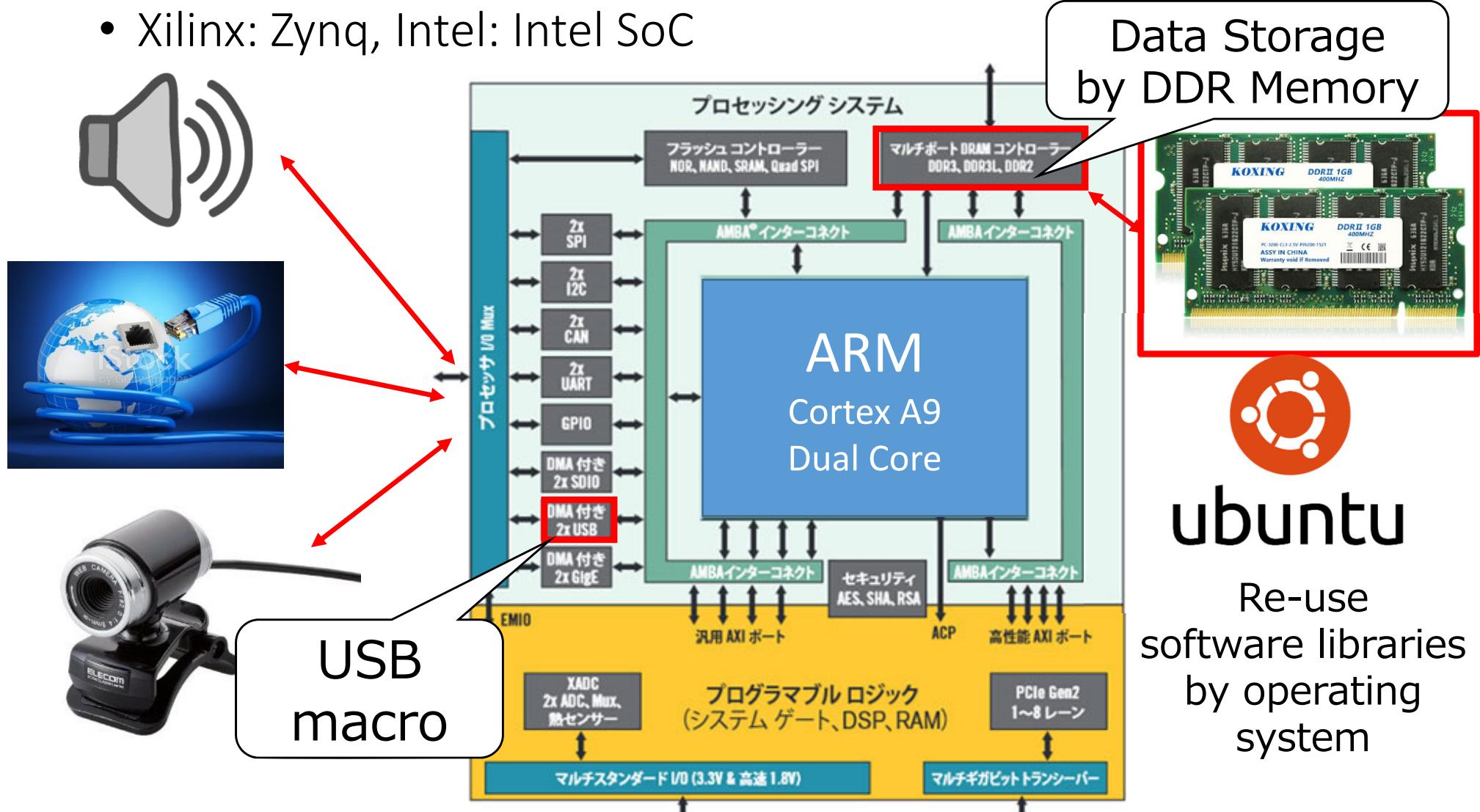
Local routing

# High-Level Synthesis (HLS)



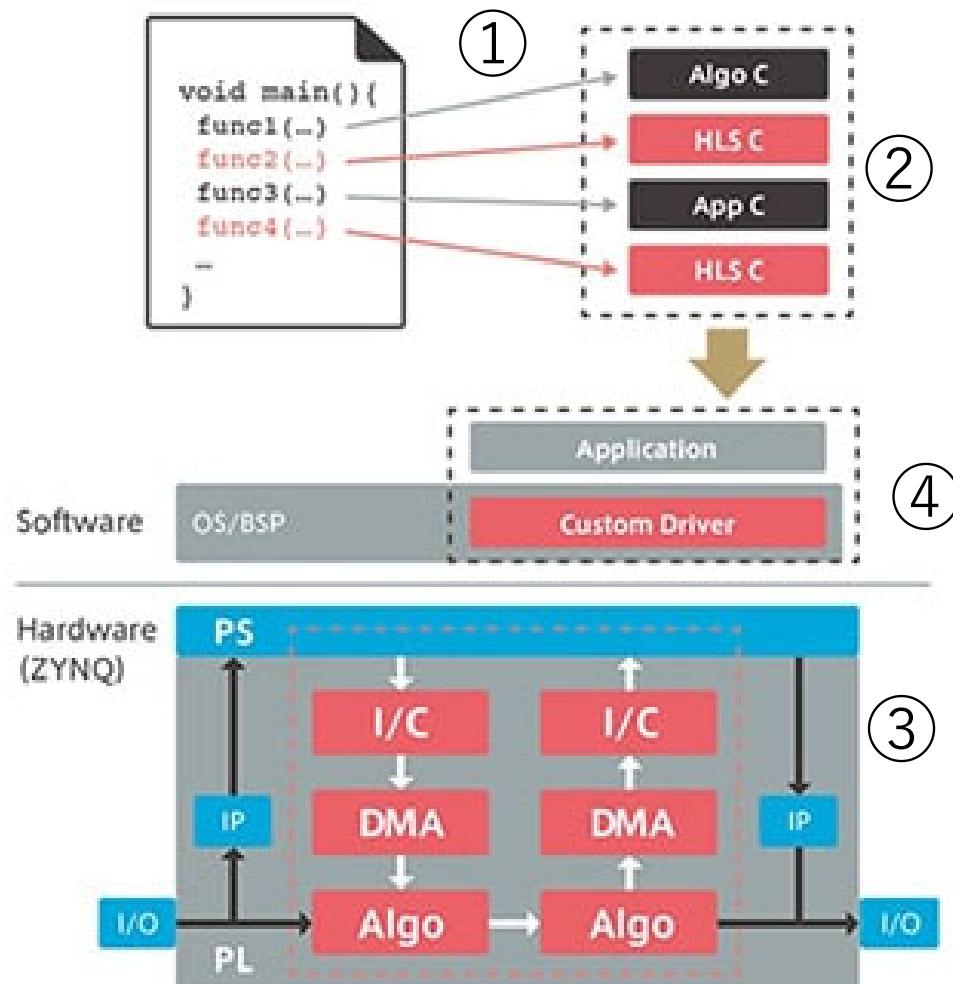
# System on Chip FPGA

- Xilinx: Zynq, Intel: Intel SoC



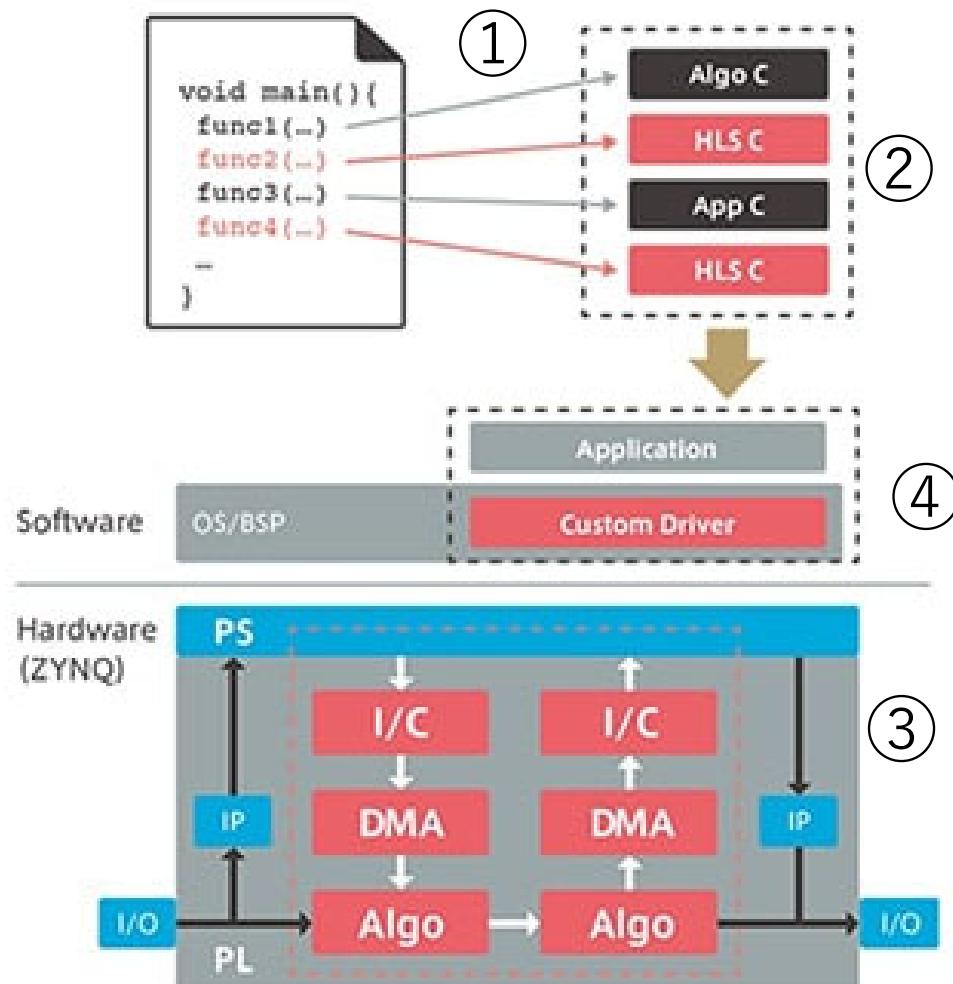
Source: Xilinx Inc. Zynq-7000 All Programmable SoC

# Conventional Design Flow for the SoC FPGA



1. Behavior design
2. Profile analysis
3. IP core generation by HLS
4. Bitstream generation by FPGA CAD tool
5. Middle ware generation

# System Design Tool for the SoC FPGA



1. Behavior design + pragmas
  2. Profile analysis
  3. IP core generation by HLS
  4. Bitstream generation by FPGA CAD tool
  5. Middle ware generation
- ↓
- Automatically done

# Summary

- FPGA: Reconfigurable LSI or Programmable Hardware
- It consists of a programmable logic array and a programmable interconnection
- Standard FPGA design supports an RTL based one
- Benefits: Productivity, lower non-recurring engineering costs, maintainability, faster time to market

# Exercise 1

1. (Mandatory) Show an application using a FPGA
2. (Selective) The modern FPGA has several configuration devices as follows:
  - Static RAM (SRAM)
  - Flash memory
  - Anti-fuse

In that case, both the SRAM-based FPGA and Flash-based one supports “multiple-time” configurations, while an anti-fuse only “one-time” configuration. Actually, some vendor adopts the anti-fuse-based FPGA than SRAM-based one. Why?

Deadline is 18<sup>th</sup>, Oct., 2018 (At the beginning of the next lecture)

Send a PDF file to nakahara{at}ict.e.titech.ac.jp