Communications and Computer Engineering II

FPGA Basis

Hiroki Nakahara

Tokyo Institute of Technology

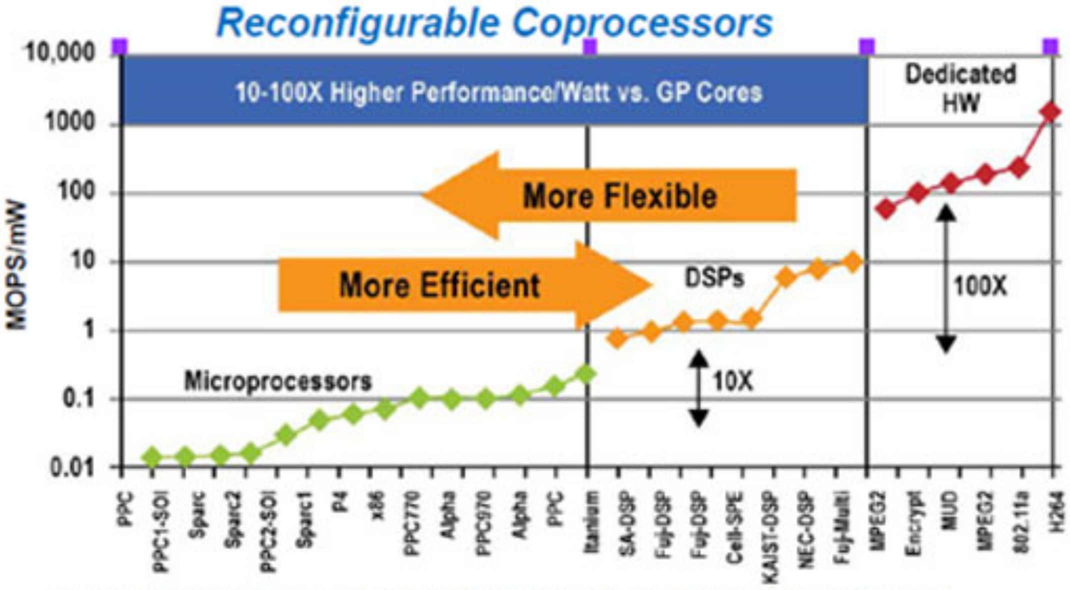
Outline

- What’s FPGA?
    - Recent trends
- FPGA Architecture
- Standard FPGA Design
    - RTL (Register Transfer Level)
    - High-level synthesis (HLS)
    - System-level design
- Summary

1.1 FPGA

The Dilemma: Flexibility vs. Efficiency

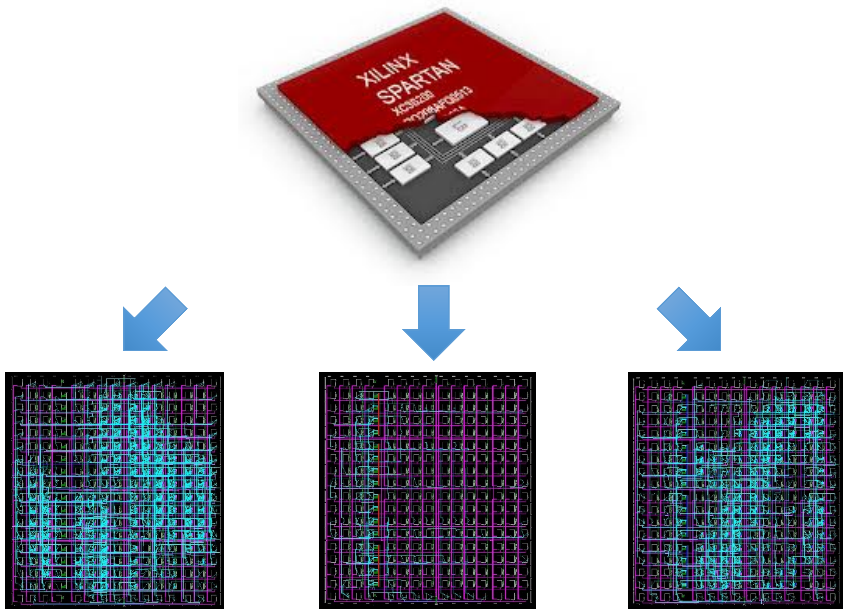
- FPGAs often offer the best of both worlds – replacing MPUs, DSPs, and dedicated ASSPs or ASICs
- Their on-the-fly reconfigurability helps them realize in-system logic functions that CPUs can’t



Source: "High-Performance Energy-Efficient Reconfigurable Accelerator Circuits for the Sub-45nm Era", July 2011
by Ram K. Krishnamurthy, Circuits Research Labs, Intel Corp.

FPGA

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



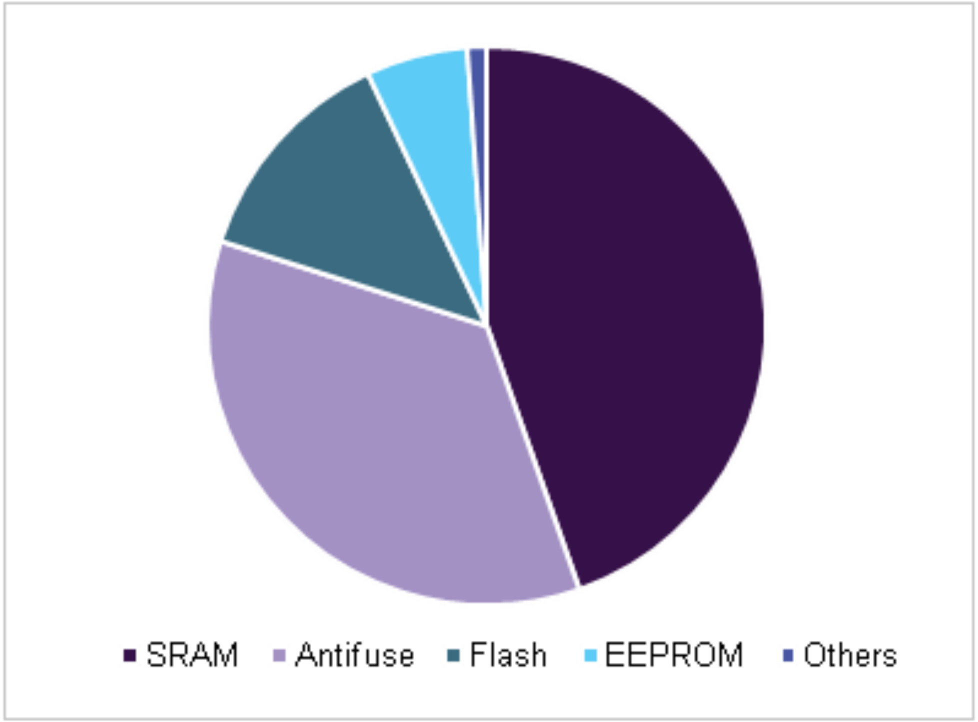
Pros. and Cons.

- 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

Product Type Segments

- SRAM
- Flash based
- Antifuse

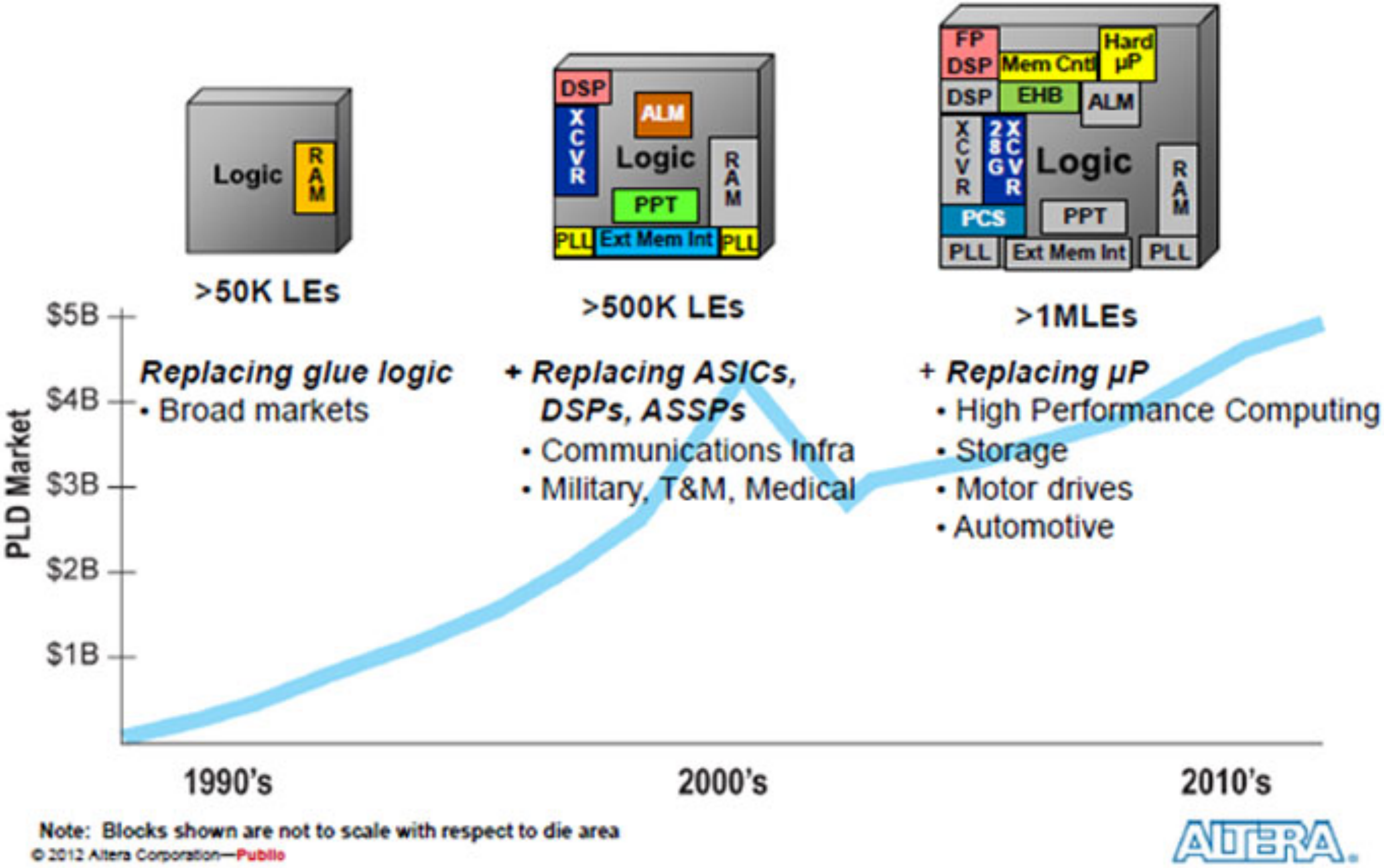
Global FPGA market share, by technology, 2015 (USD Million)



Source: https://www.grandviewresearch.com/industry-analysis/fpga-market

FPGA Growth Trend

- 20 Years FPGAs have been swallowing up system components (by Altera, now a part of Intel)



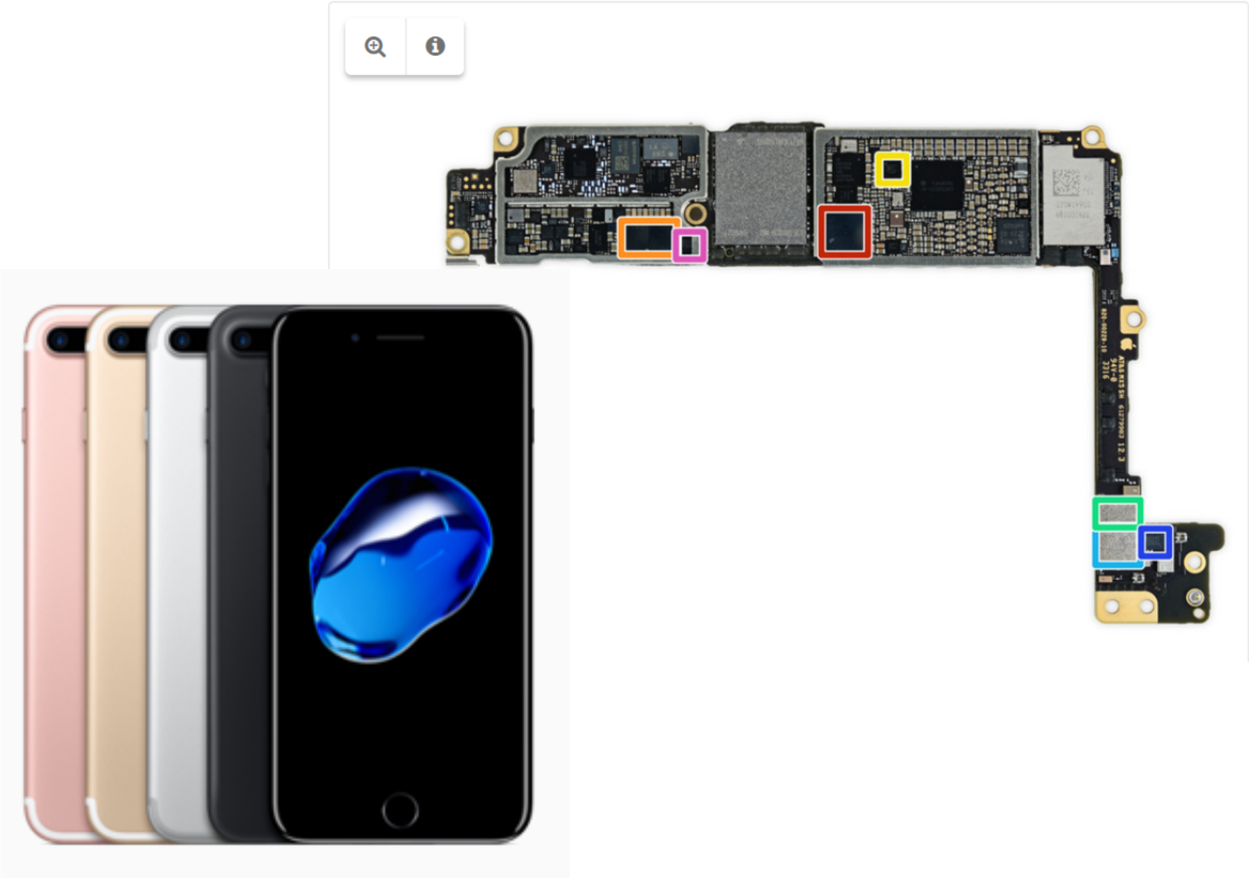
Note: Blocks shown are not to scale with respect to die area

© 2012 Altera Corporation—Public

Application Type Segments

- Industrial
- Automotive
- Consumer electronics
- Military & aerospace
- Telecom
- Data processing
- Others

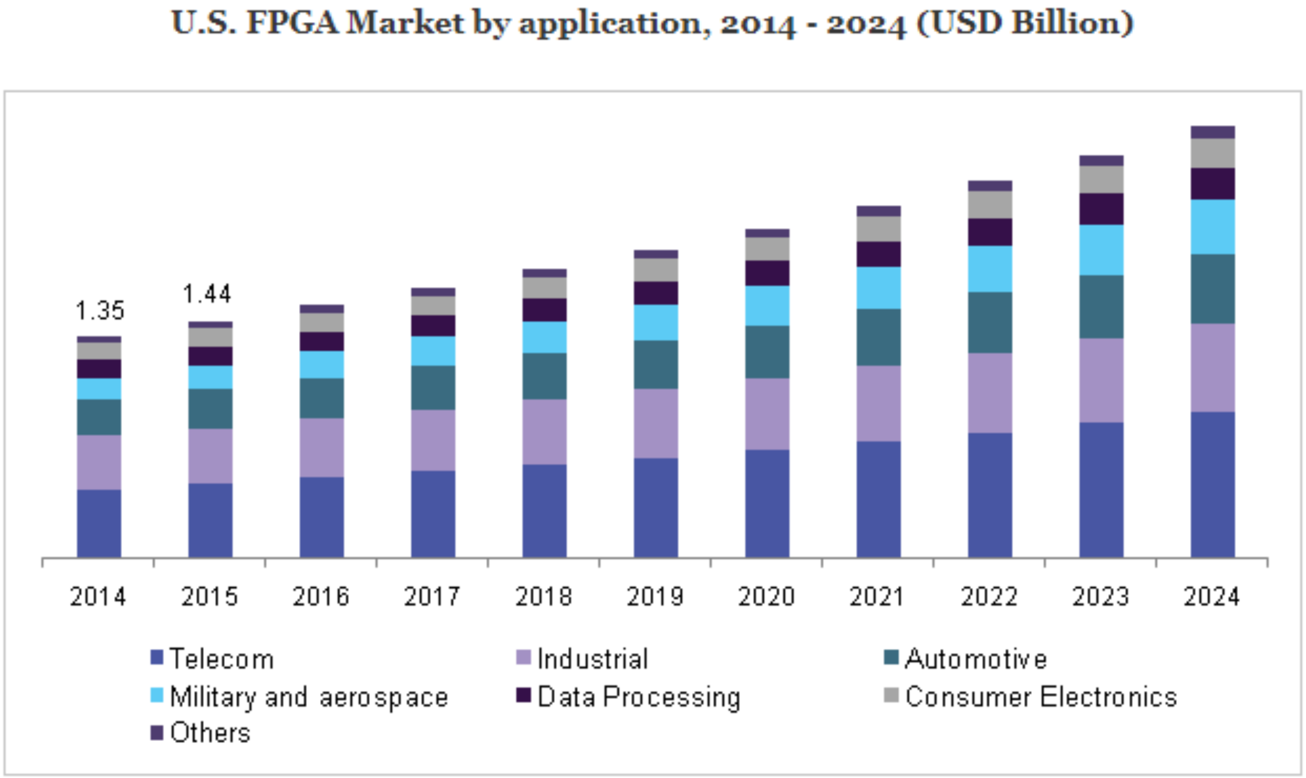
iPhone7 Plus



- 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

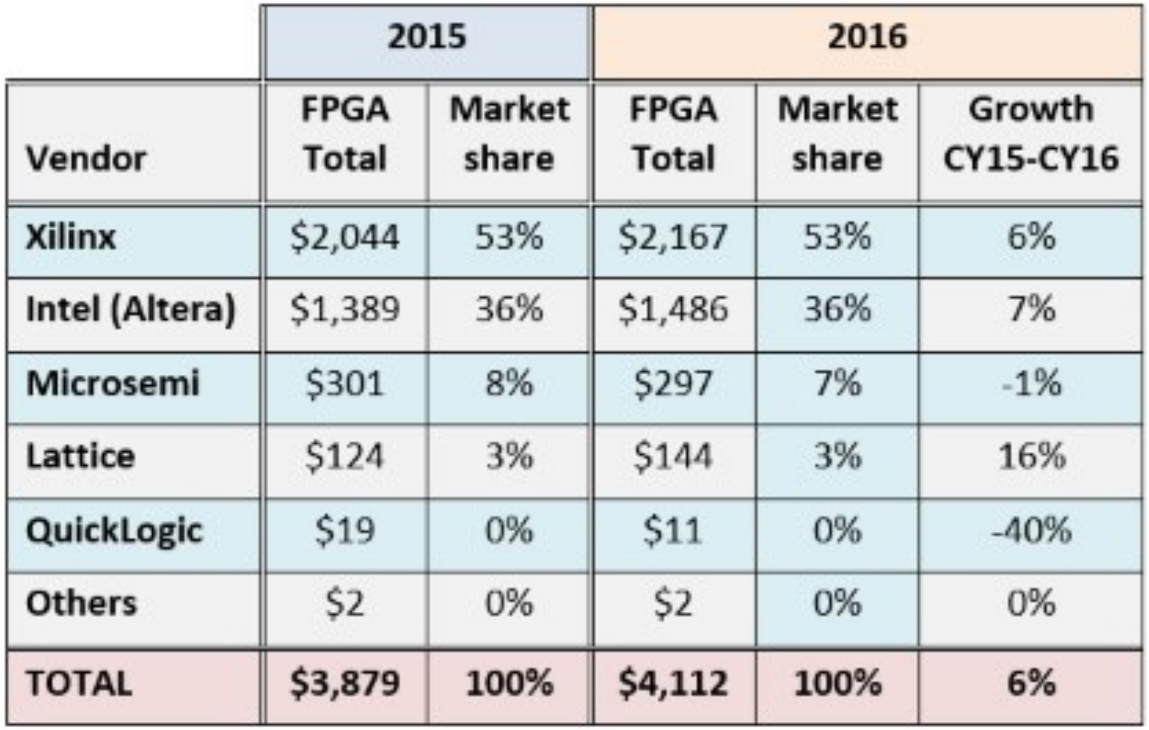
Source: https://www.ifixit.com/Teardown/iPhone+7+Plus+Teardown/67384

Market by Application



Source: https://www.grandviewresearch.com/industry-analysis/fpga-market

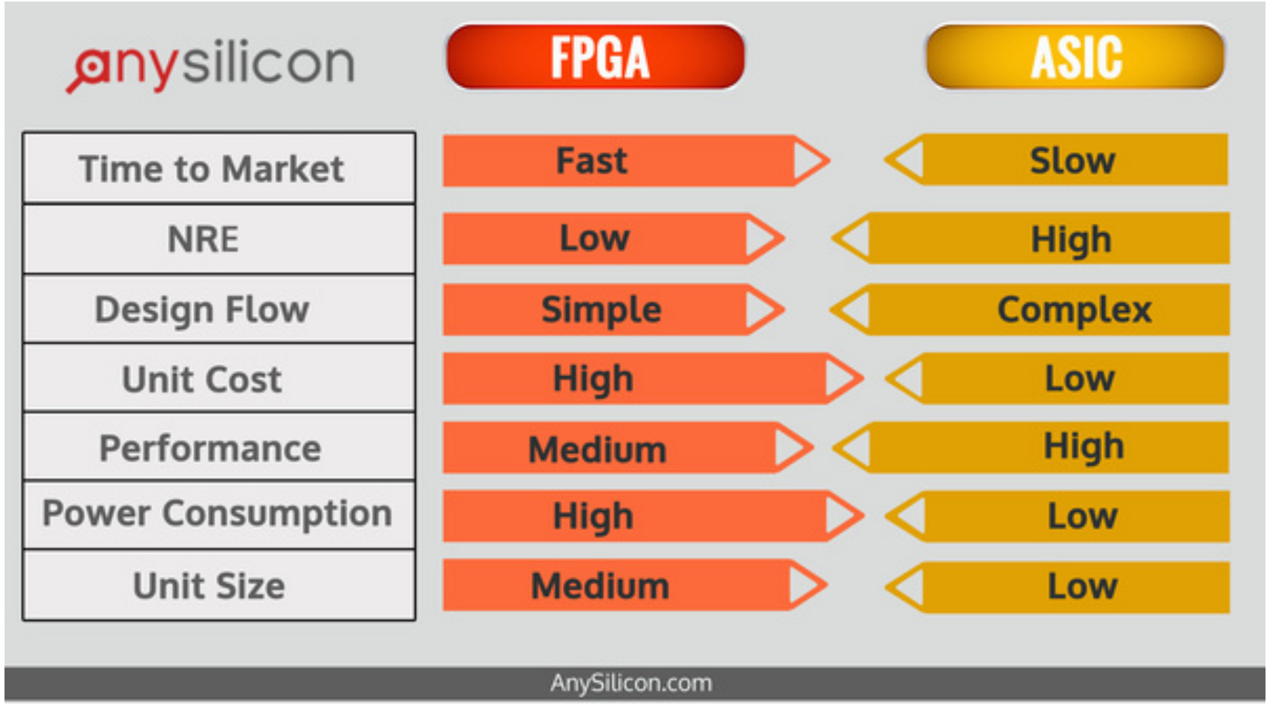
Market Share by Vendor



	2015		2016		
Vendor	FPGA Total	Market share	FPGA Total	Market share	Growth CY15-CY16
Xilinx	$2,044	53%	$2,167	53%	6%
Intel (Altera)	$1,389	36%	$1,486	36%	7%
Microsemi	$301	8%	$297	7%	-1%
Lattice	$124	3%	$144	3%	16%
QuickLogic	$19	0%	$11	0%	-40%
Others	$2	0%	$2	0%	0%
TOTAL	**$3,879**	**100%**	**$4,112**	**100%**	**6%**

Source EEtimes 3/5/2017

FPGA vs. ASIC: Which is Best?



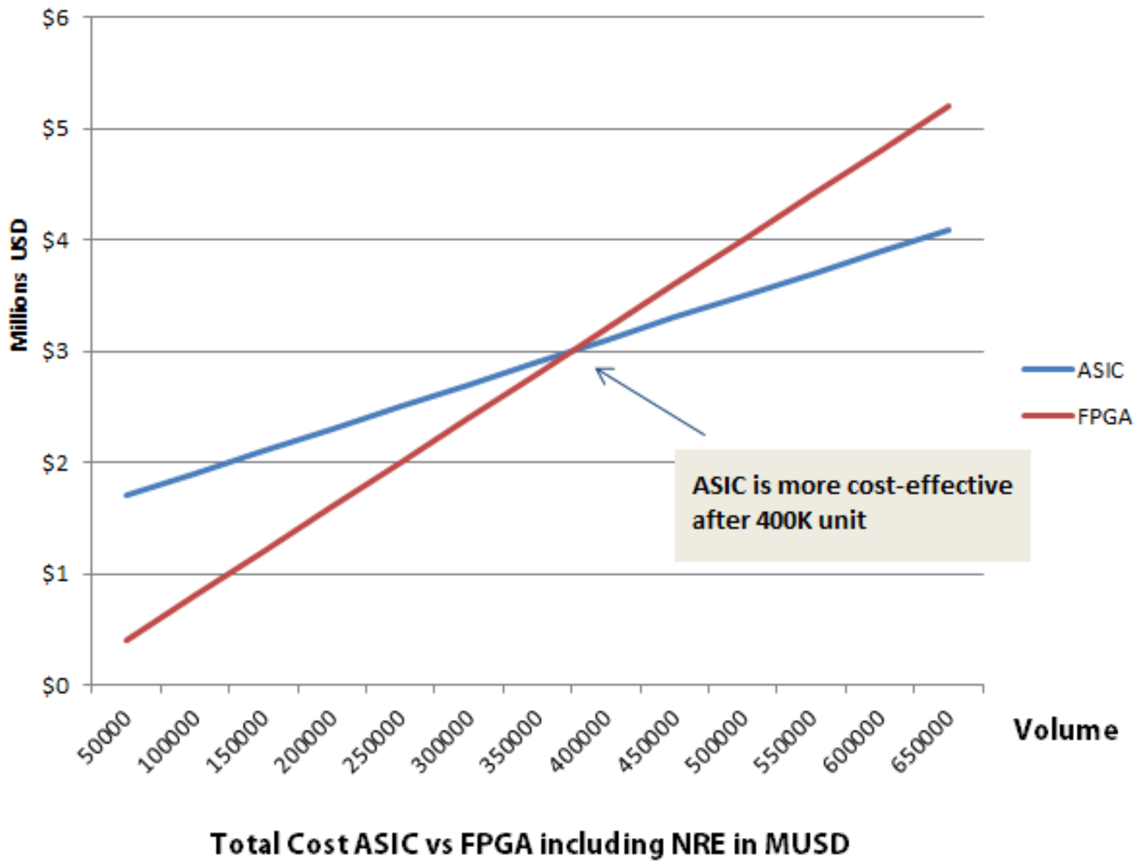
	FPGA	ASIC
Time to Market	Fast	Slow
NRE	Low	High
Design Flow	Simple	Complex
Unit Cost	High	Low
Performance	Medium	High
Power Consumption	High	Low
Unit Size	Medium	Low

AnySilicon.com

Source: https://anysilicon.com/fpga-vs-asic-choose/

Price Comparison

- ASIC→NRE: $1.5M, Unit cost: $4
- FPGA→NRE: $0, Unit cost: $8

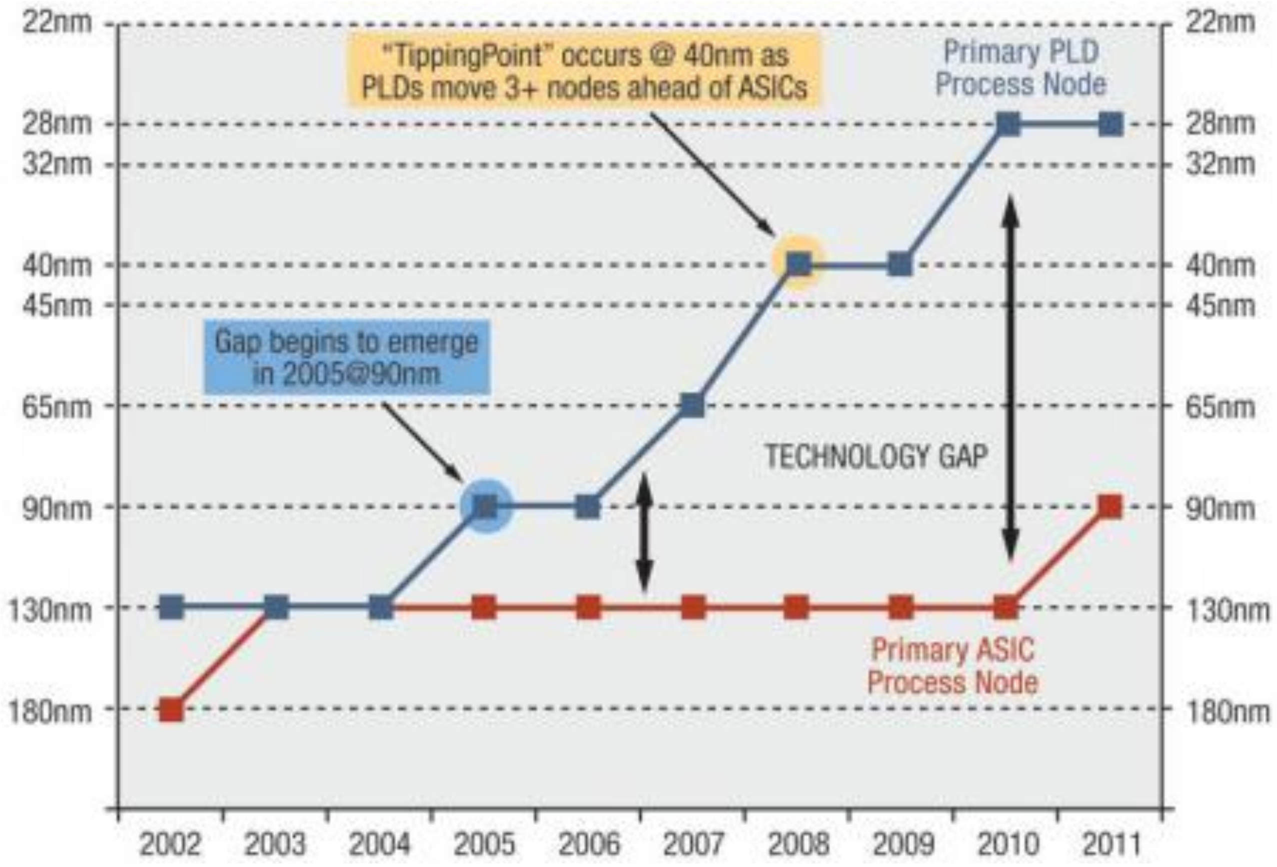


Total Cost ASIC vs FPGA including NRE in MUSD

anysilicon.com

Performance Comparison

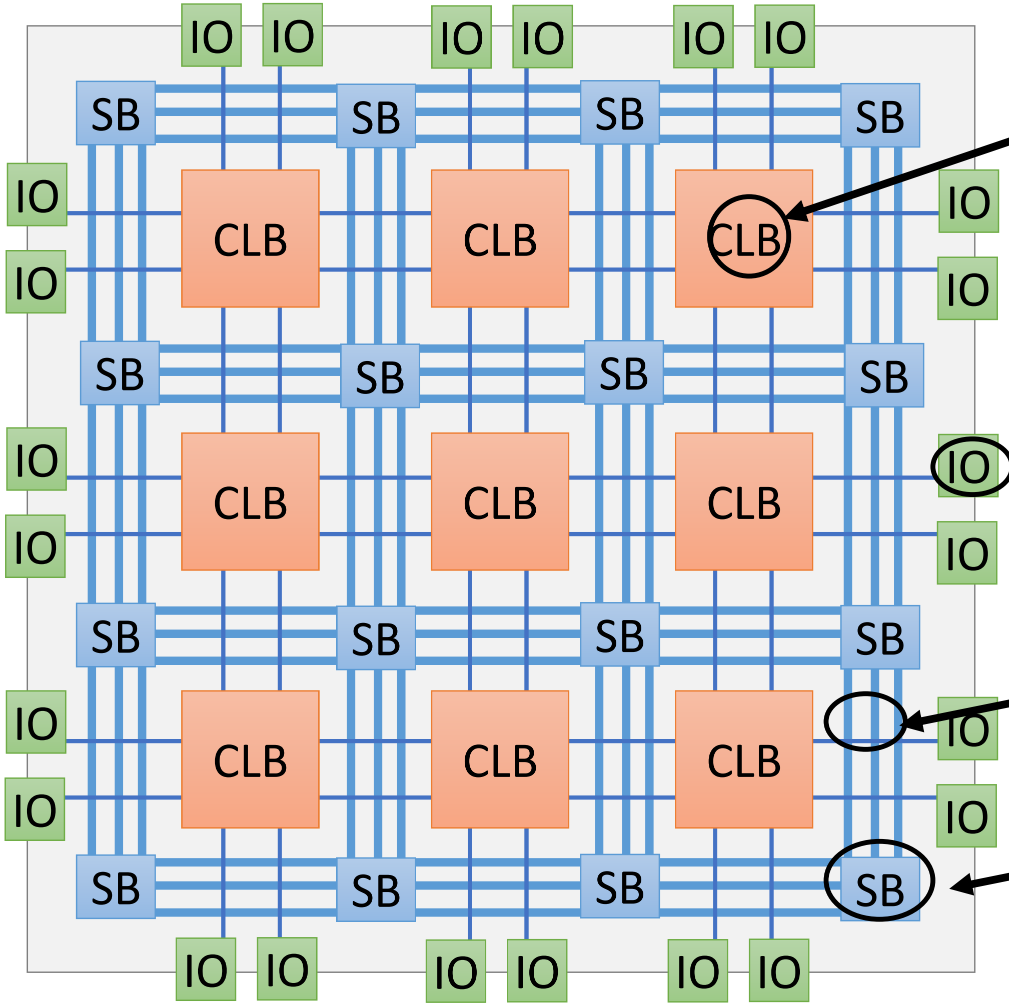
- The divergence between primary programmable logic device technology and that used for ASICs has continued to grow



Source: http://archive.rtcmagazine.com/articles/view/102503

1.2 FPGA Architecture

Island Style FPGA



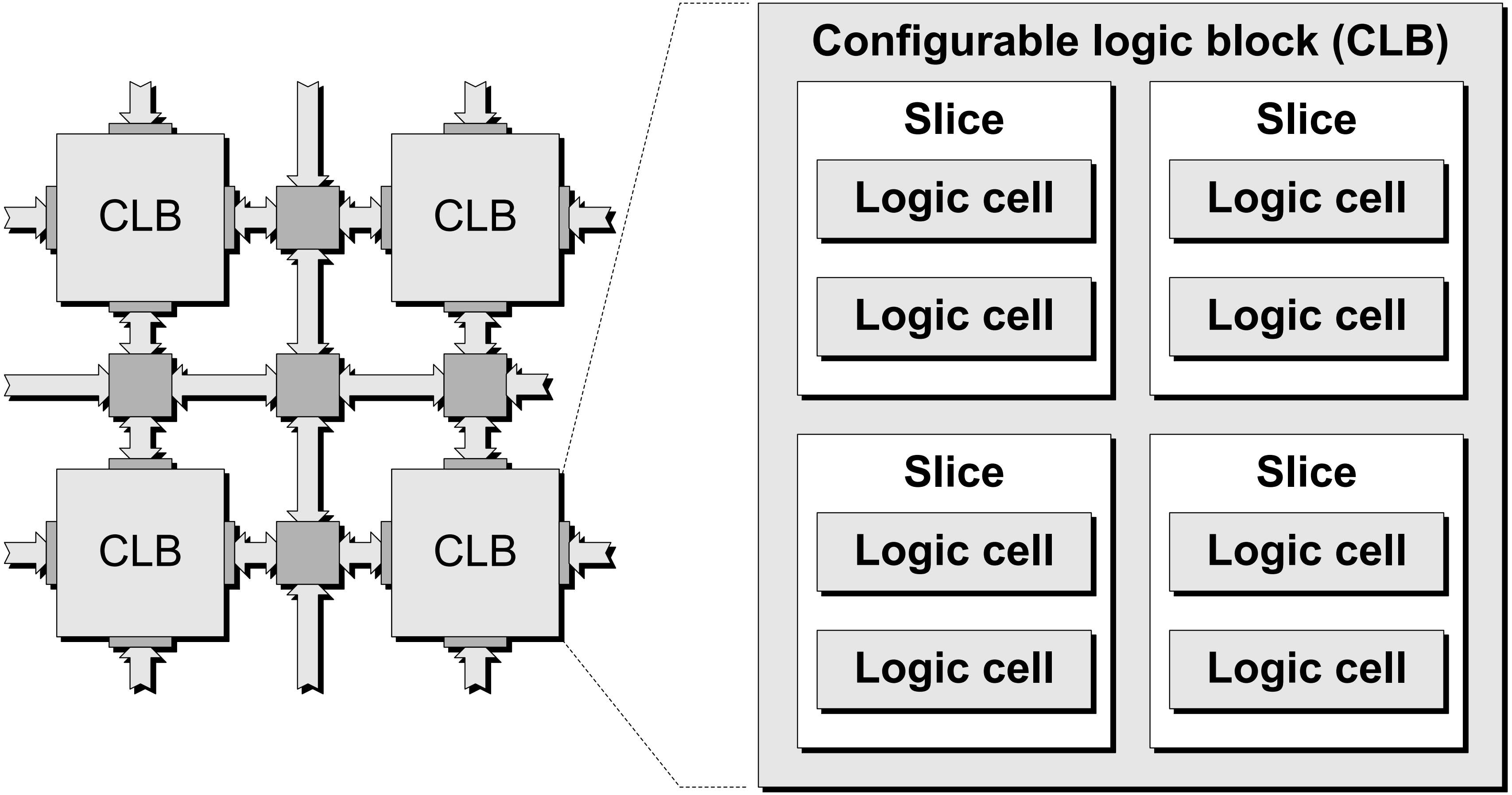
Configurable Logic Block (CLB)

I/O Block

Channels

Switch Blocks

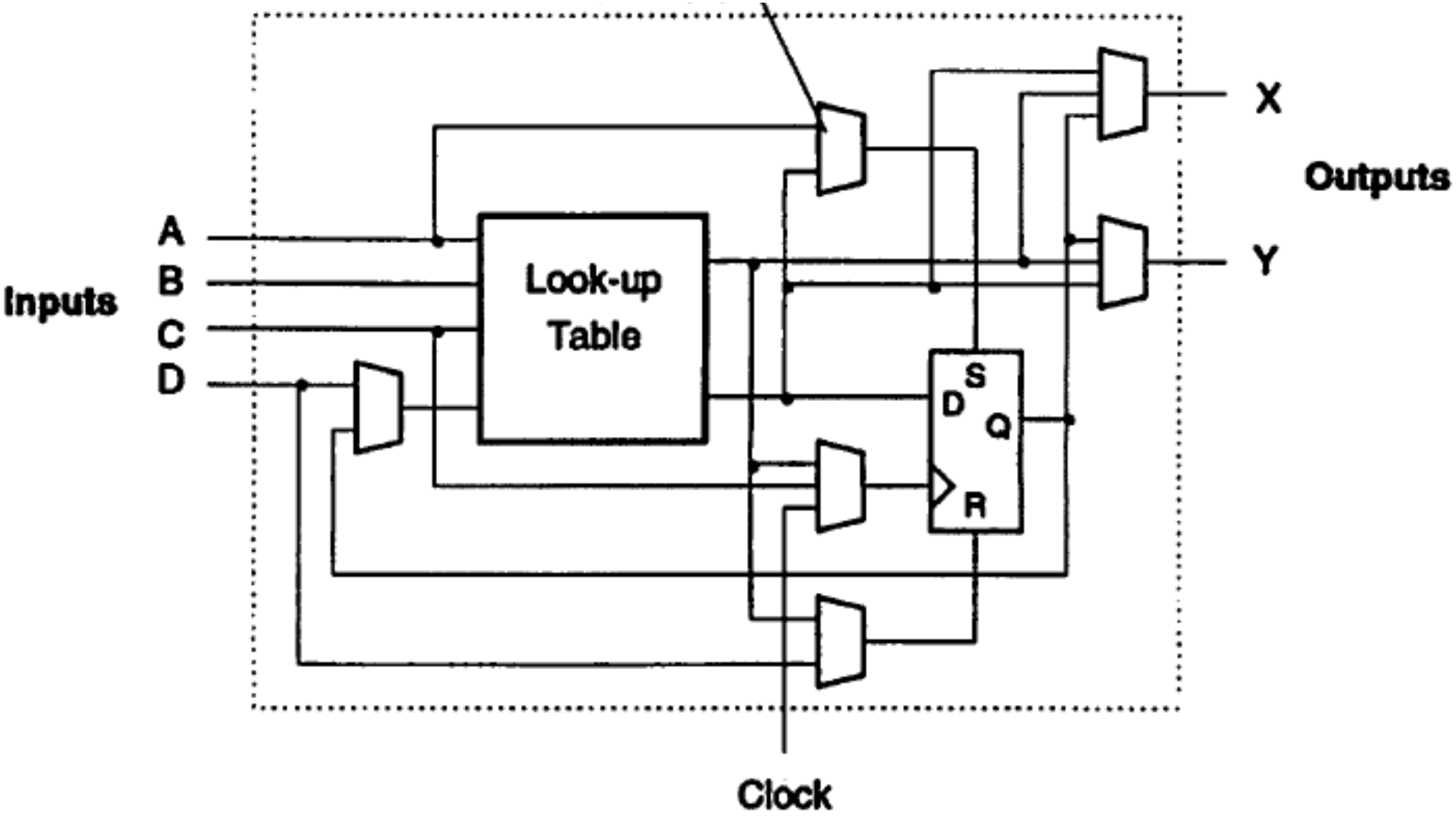
Xilinx CLB



The Design Warrior's Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043

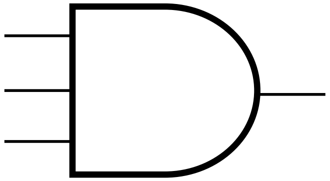
Logic Cell (Xilinx Inc. XC2000)

Programmable Multiplexer

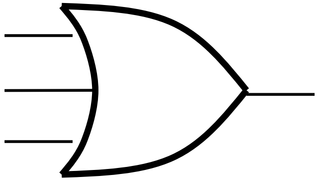


Realization of a Logic Function

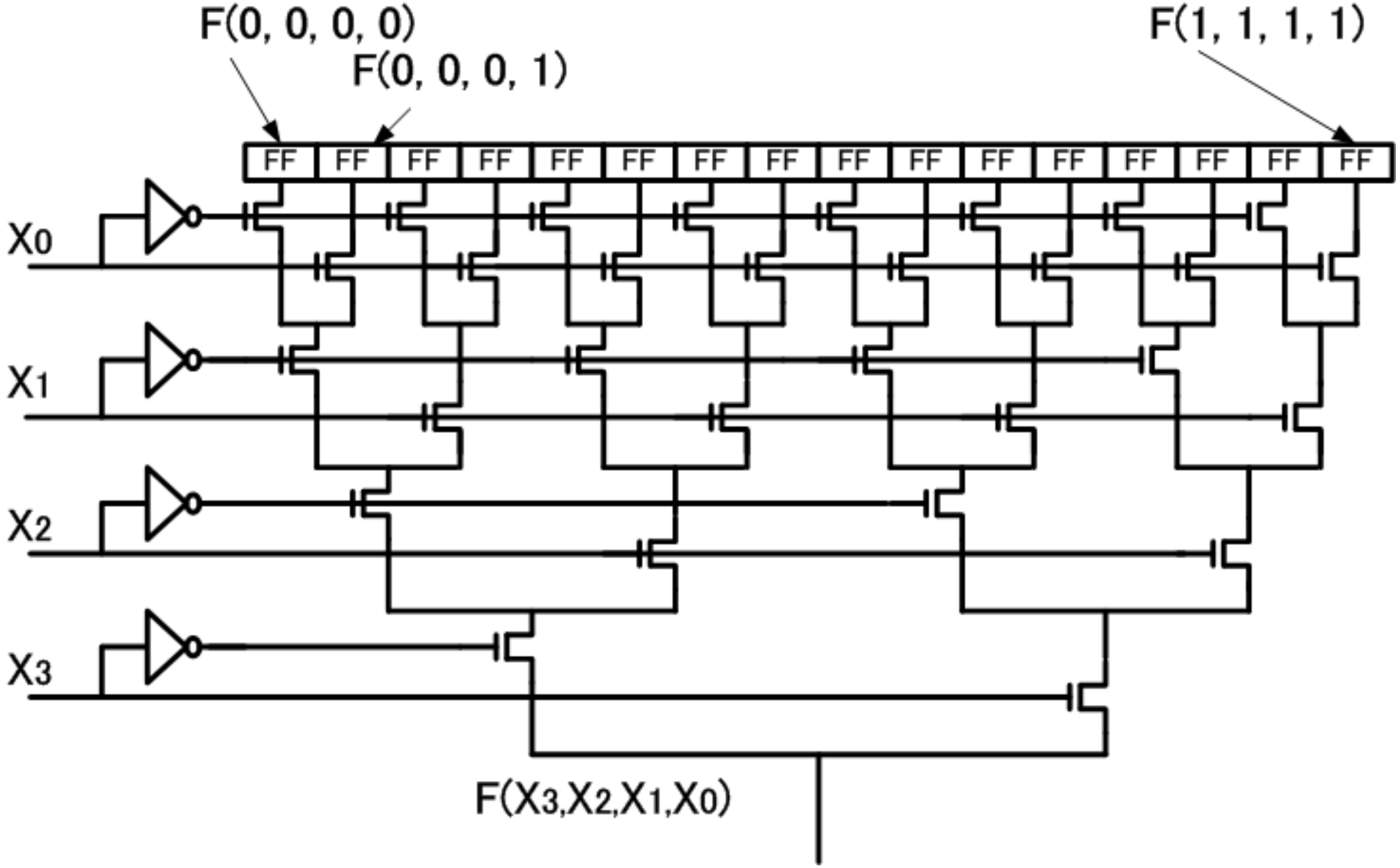
x0	x1	x2	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



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

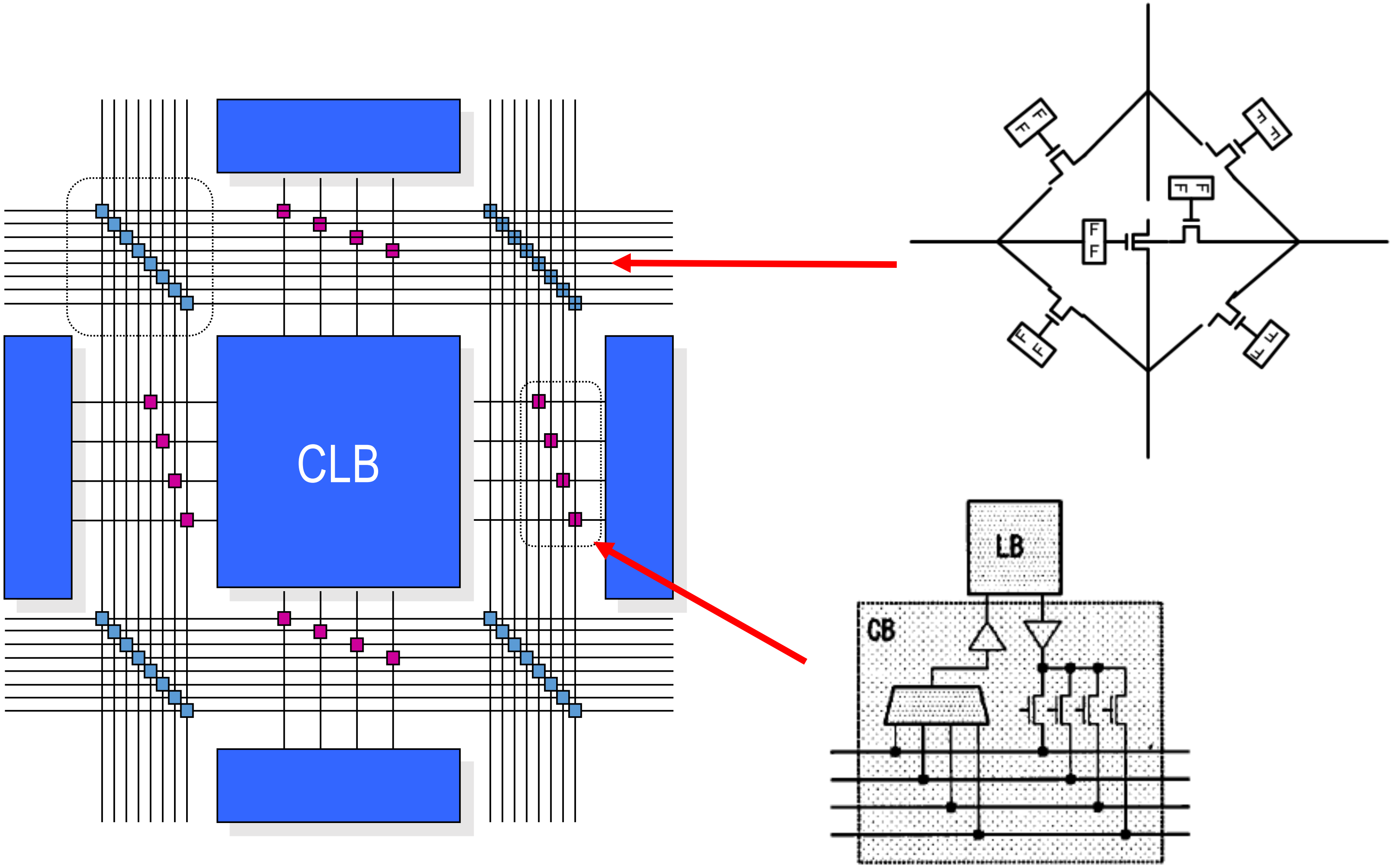


LUT Structure

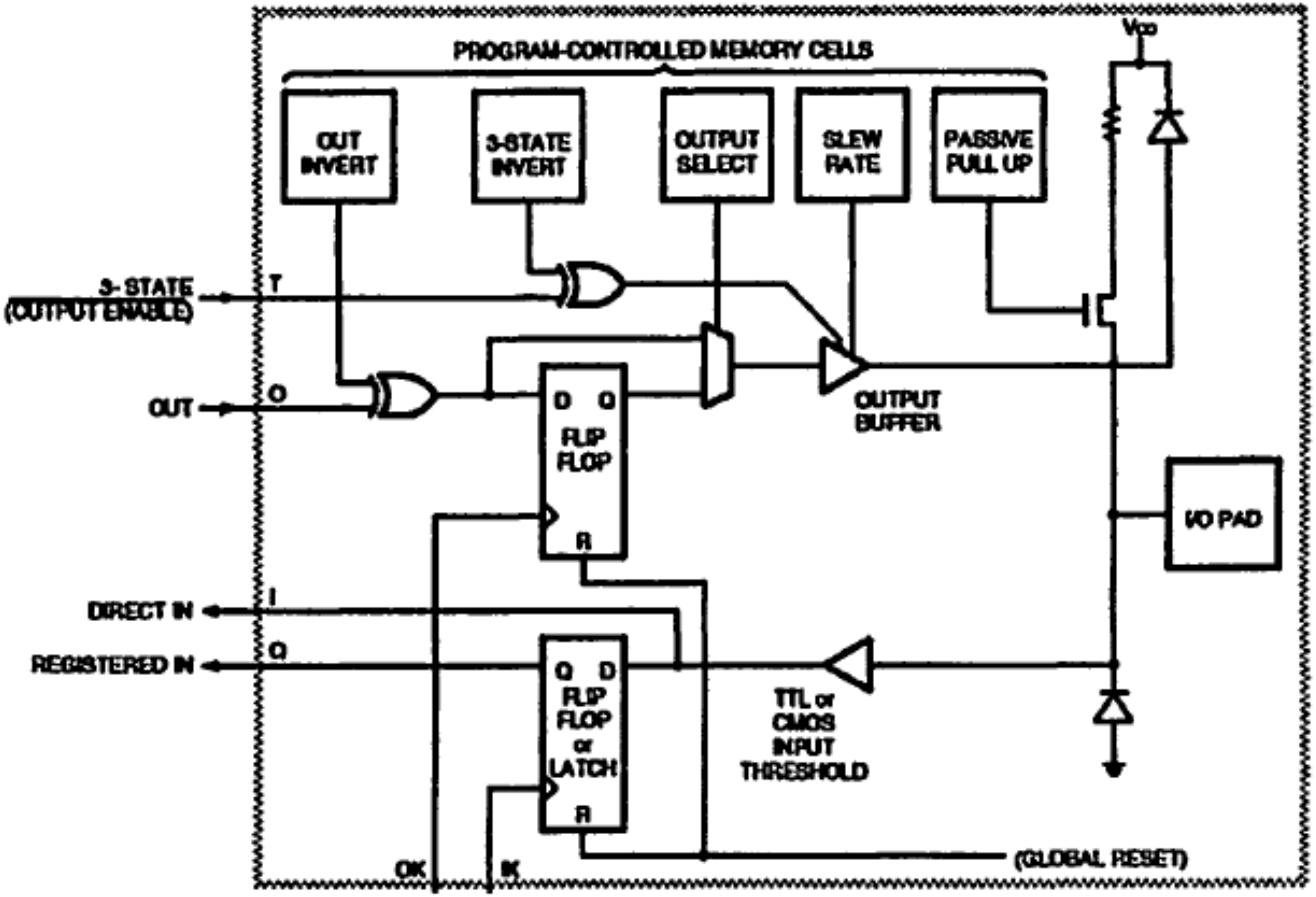


F(0, 0, 0, 0)
F(0, 0, 0, 1)
F(1, 1, 1, 1)
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
X_0
X_1
X_2
X_3
F(X_3,X_2,X_1,X_0)

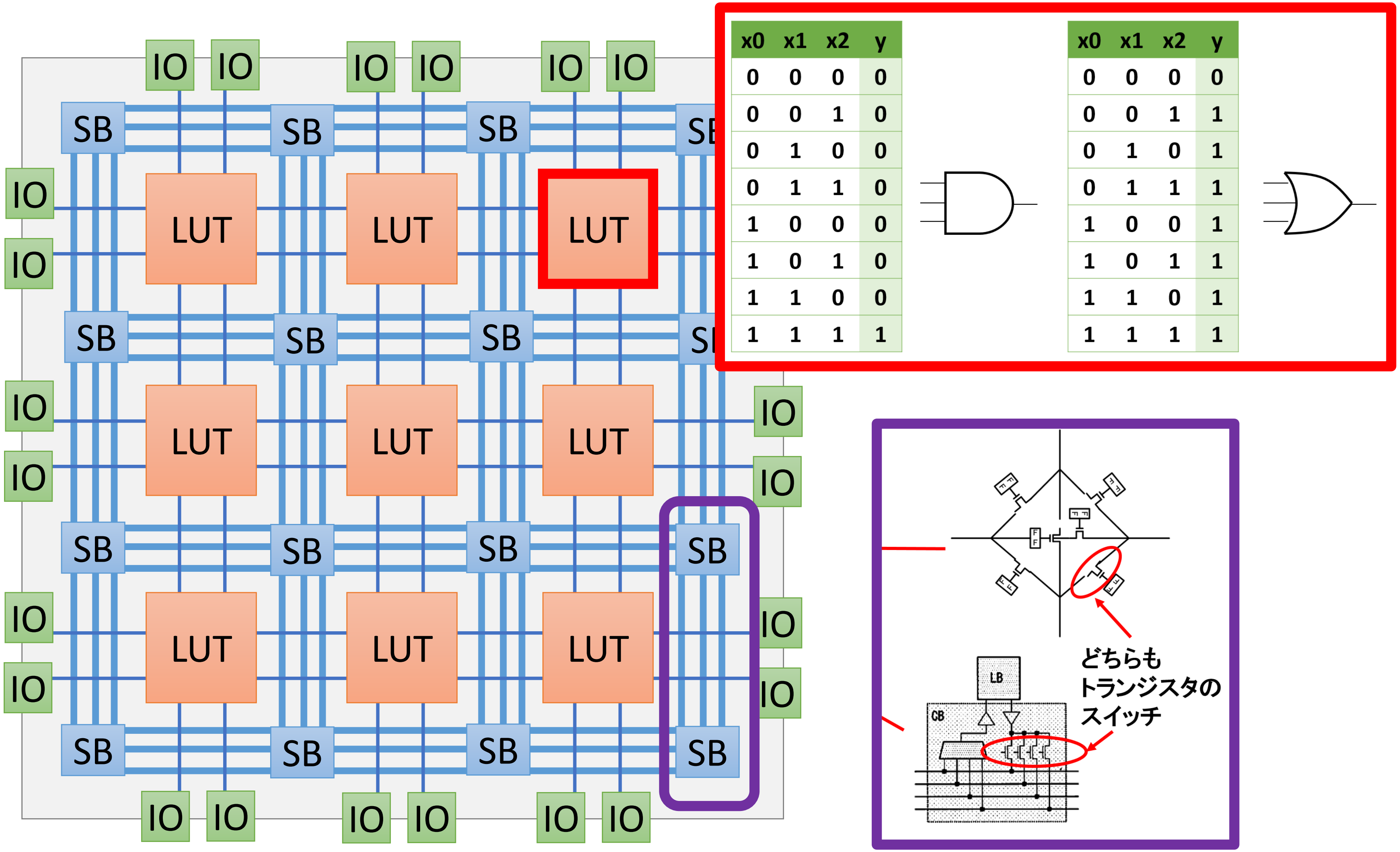
Channel and Switch Block



I/O Block


PROGRAM-CONTROLLED MEMORY CELLS
OUT INVERT
3-STATE INVERT
OUTPUT SELECT
SLEW RATE
PASSIVE PULL UP
Vcc
3- STATE (OUTPUT ENABLE)
T
OUT
O
D Q
FLIP FLOP
R
OUTPUT BUFFER
I/O PAD
DIRECT IN
I
REGISTERED IN
Q
Q D
FLIP FLOP or LATCH
R
TTL or CMOS INPUT THRESHOLD
(GLOBAL RESET)
OK
IK

Memory-based realizes “programmable”

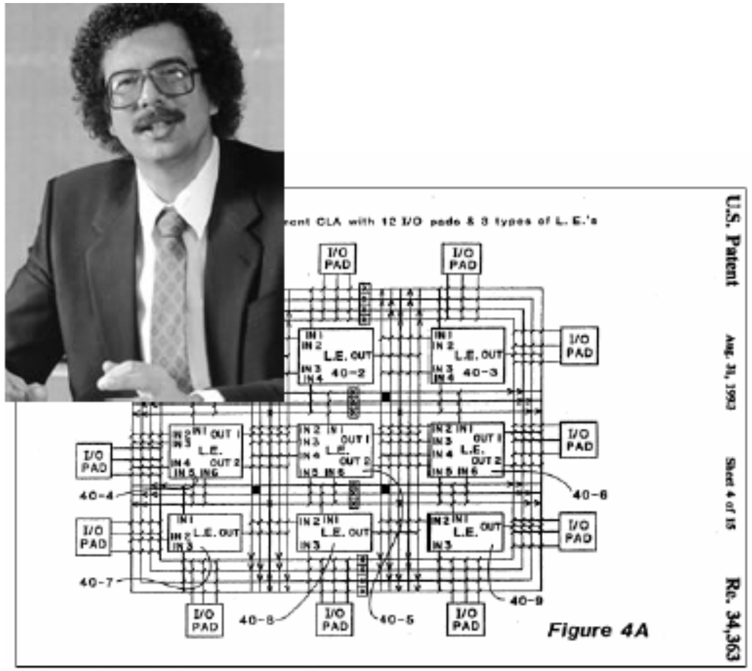


x0	x1	x2	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

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

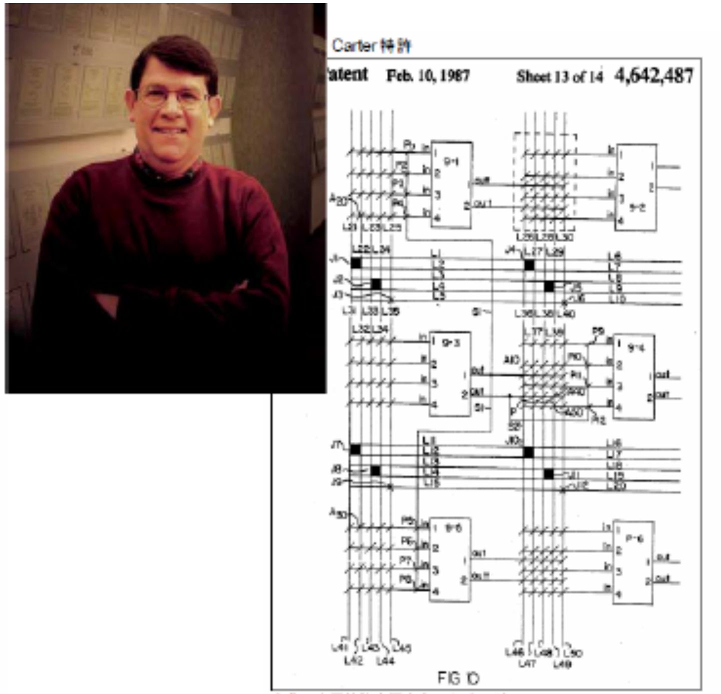
FPGA-world Famous Patents

US RE34363
by Ross H. Freeman
Island style architecture



出典：米国特許商標庁ホームページ

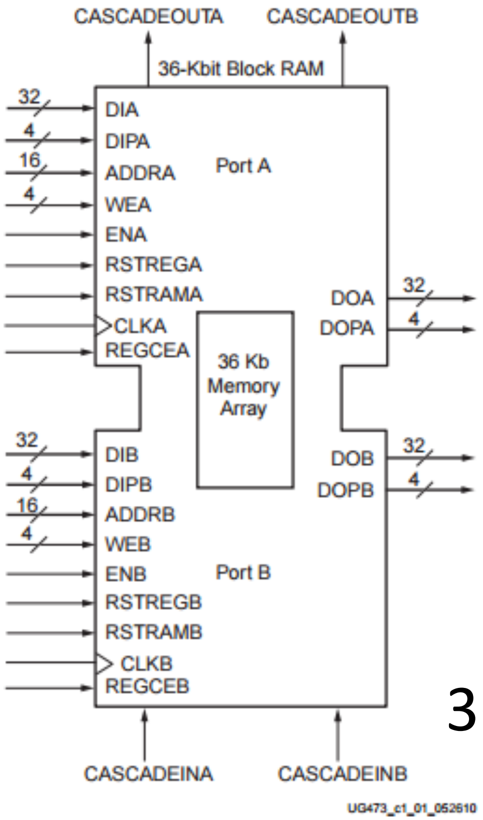
US 464248
by William S. Carter
Connection architecture

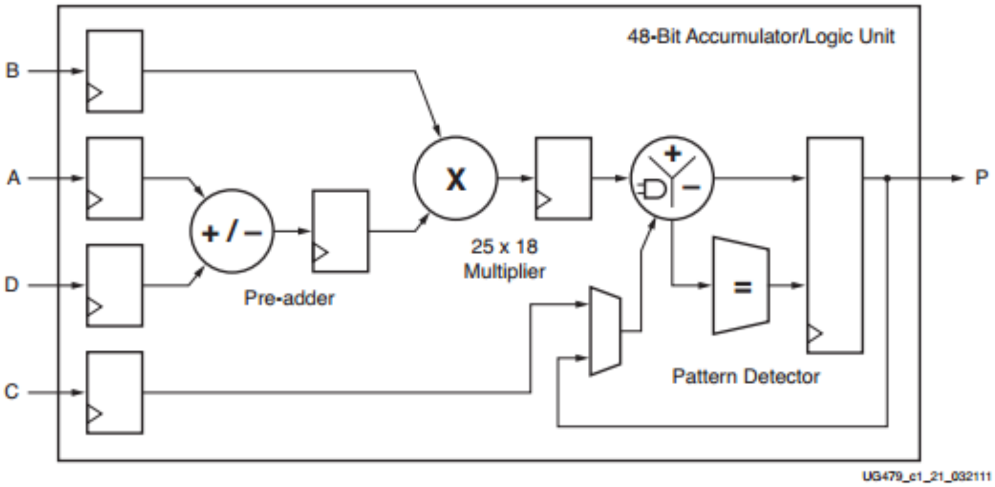


出典：米国特許商標庁ホームページ

Hard macro (Dedicated Circuit)

- Standard IP cores are "Pre-built" in the FPGA



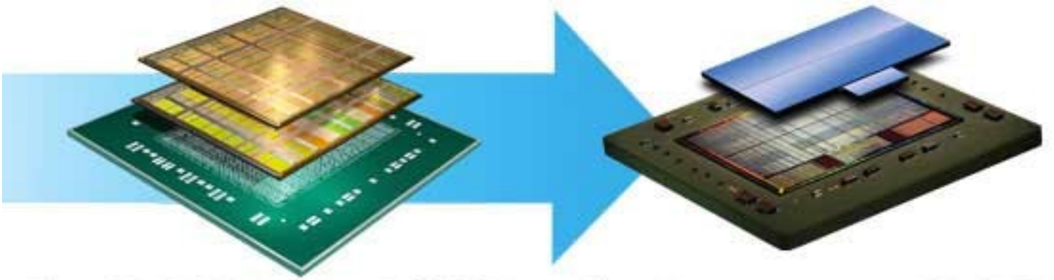


48bit DSP Block

36Kb Block Memory

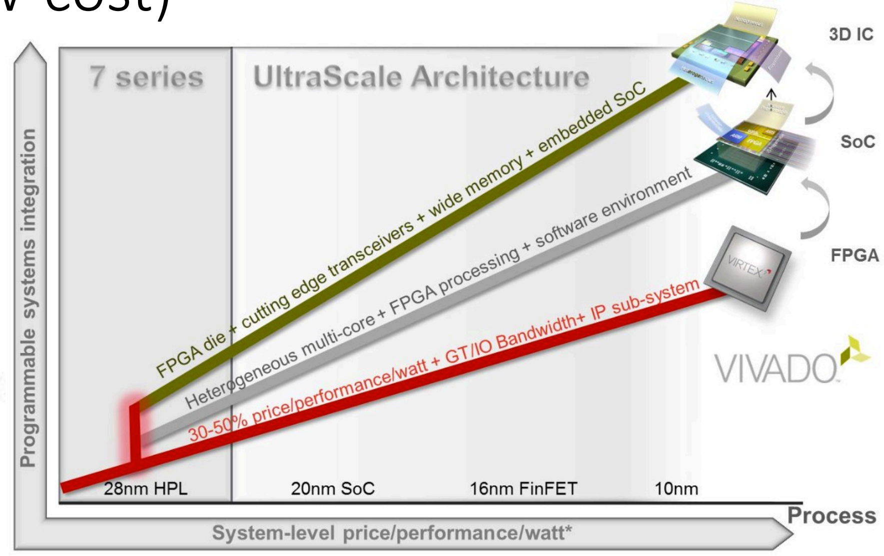
Xilinx FPGA Families

- 3D IC type
  - Virtex UltraScale+, Kintex UltraScle
- System on Chip (SoC) type
  - Zynq UltraSCale+
- Monolithic type
  - Artix(Low-cost), Spartan (Low-cost)



First 3D FPGA: Virtex-7 2000T
Based on Stacked Silicon Interconnect

First Heterogeneous 3D FPGA: Virtex-7 H580T
Based on Stacked Silicon Interconnect

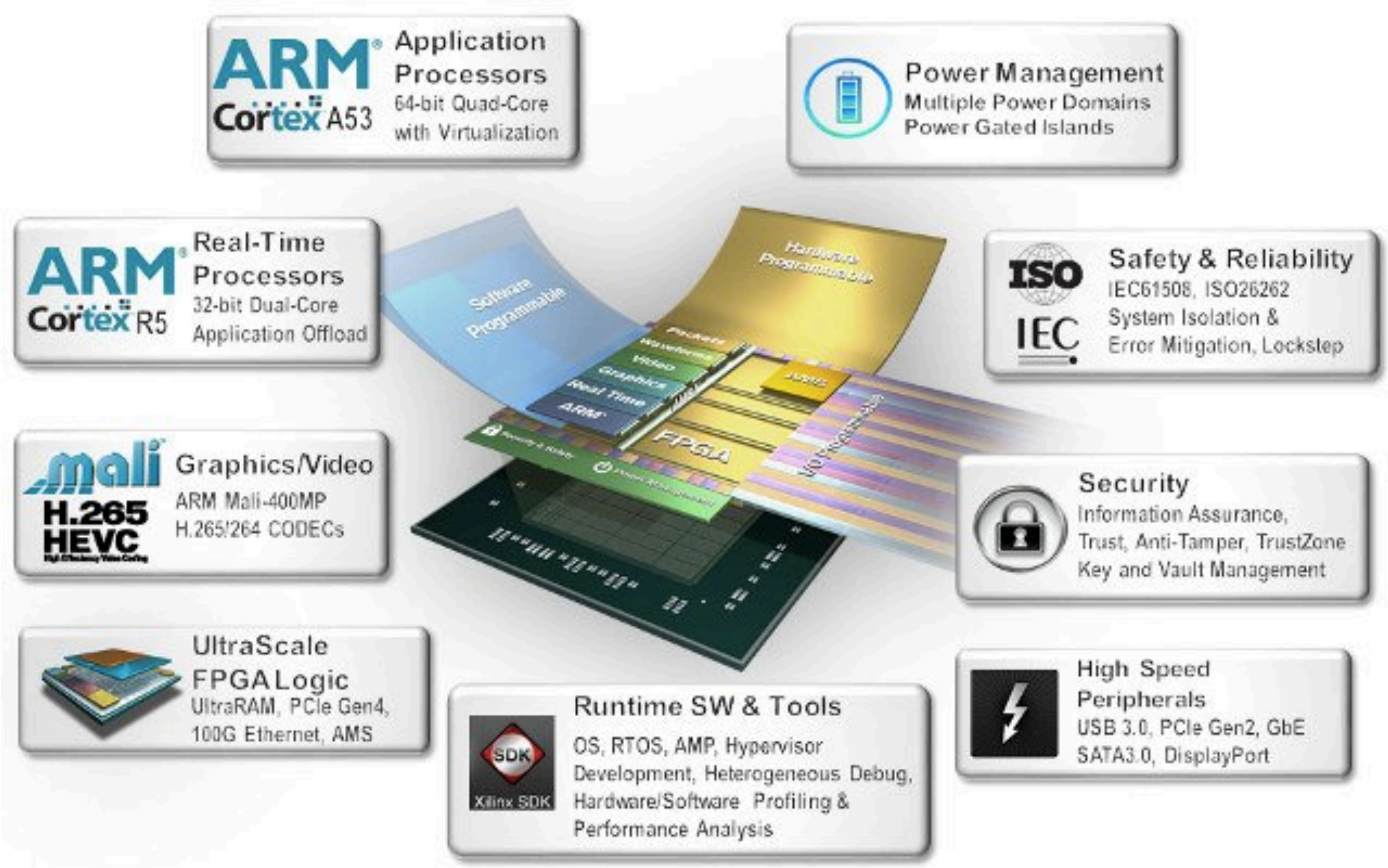


7 series
UltraScale Architecture
Programmable systems integration
FPGA die + cutting edge transceivers + wide memory + embedded SoC
Heterogeneous multi-core + FPGA processing + software environment
30-50% price/performance/watt + GT/IO Bandwidth+ IP sub-system
28nm HPL
20nm SoC
16nm FinFET
10nm
Process
System-level price/performance/watt*
3D IC
SoC
FPGA
VIVADO

* System level combines unit with BOM cost

Source: http://www.xilinx.com

Xilinx Zynq Ultra Scale+



Application Processors
ARM Cortex A53
64-bit Quad-Core with Virtualization

Power Management
Multiple Power Domains
Power Gated Islands

Real-Time Processors
ARM Cortex R5
32-bit Dual-Core Application Offload

Safety & Reliability
ISO IEC
IEC61508, ISO26262
System Isolation & Error Mitigation, Lockstep

Graphics/Video
mali H.265 HEVC
ARM Mali-400MP
H.265/264 CODECs

Security
Information Assurance, Trust, Anti-Tamper, TrustZone
Key and Vault Management

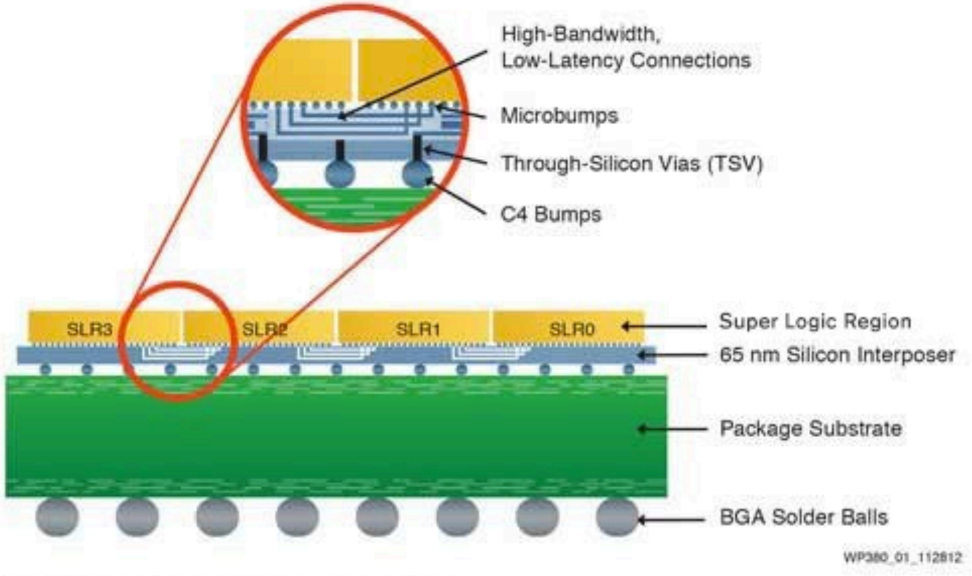
UltraScale FPGA Logic
UltraRAM, PCIe Gen4, 100G Ethernet, AMS

Runtime SW & Tools
SDK Xilinx SDK
OS, RTOS, AMP, Hypervisor
Development, Heterogeneous Debug, Hardware/Software Profiling & Performance Analysis

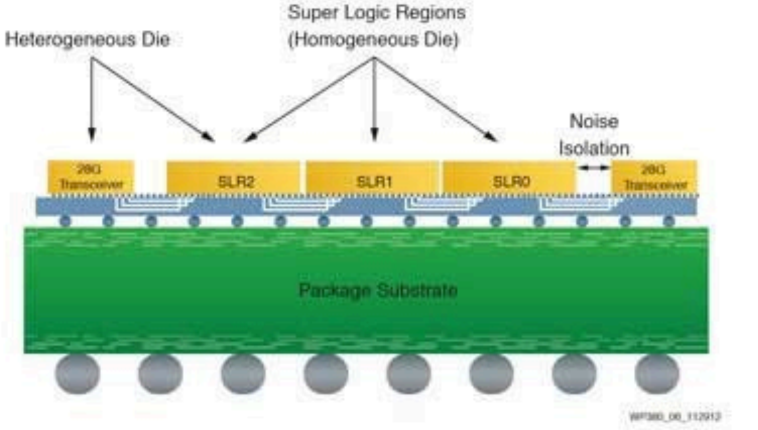
High Speed Peripherals
USB 3.0, PCIe Gen2, GbE
SATA3.0, DisplayPort

SSI(Stacked-Silicon Interconnect) Technology: Heterogeneous 3D FPGA

- High bandwidth connectivity across multiple dies
- Significantly increased die-to-die bandwidth per watt over multi-chip solutions
- Through-silicon via (TSV) with coarse pitch in passive silicon interposer (65nm) without transistor



FPGA Enabled by SSI Technology

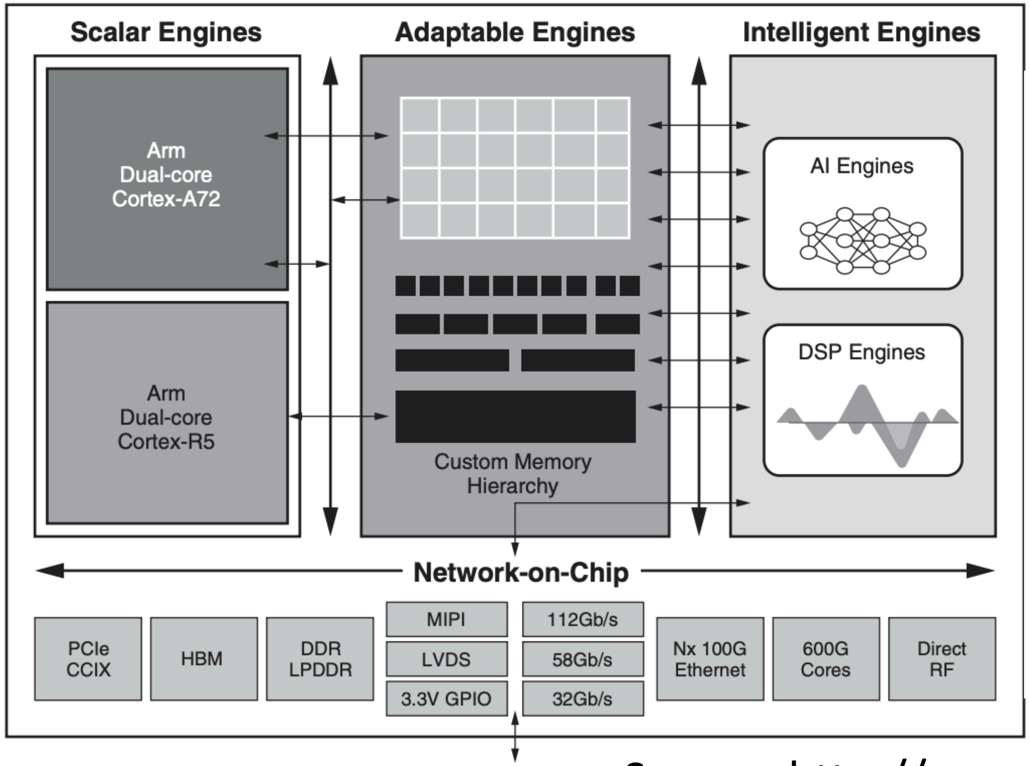


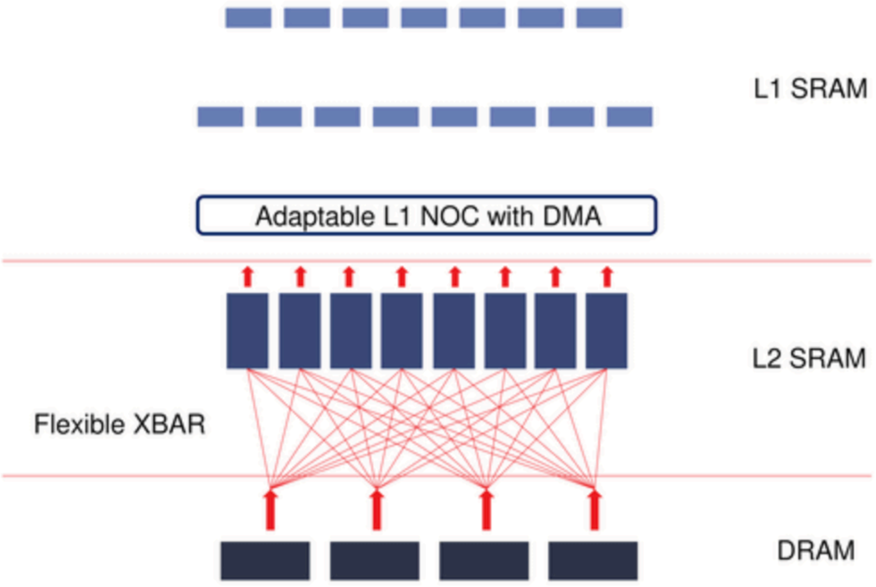
Heterogeneous 3D FPGA with Integrated 28G Transceivers

Source: http://www.xilinx.com

ACAP (Adaptive Compute Acceleration Platform)

- VERAL AI core series and prime series
  - 7nm technology
- Connected 3 elements with high-band NoC and Memory

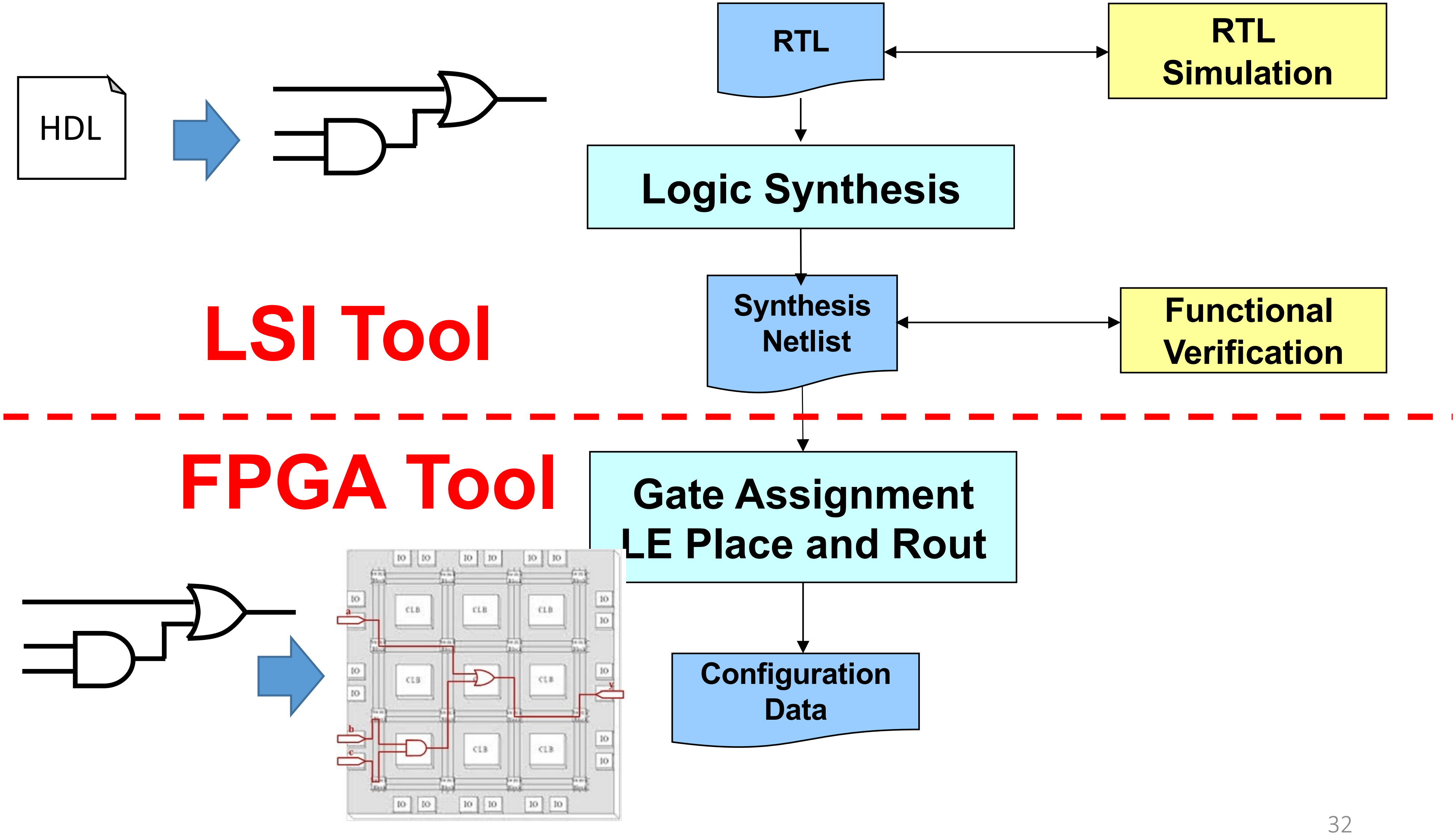




Source: http://www.xilinx.com

1.3 FPGA Design

Standard FPGA Design



RTL

RTL Simulation

Logic Synthesis

Synthesis Netlist

Functional Verification

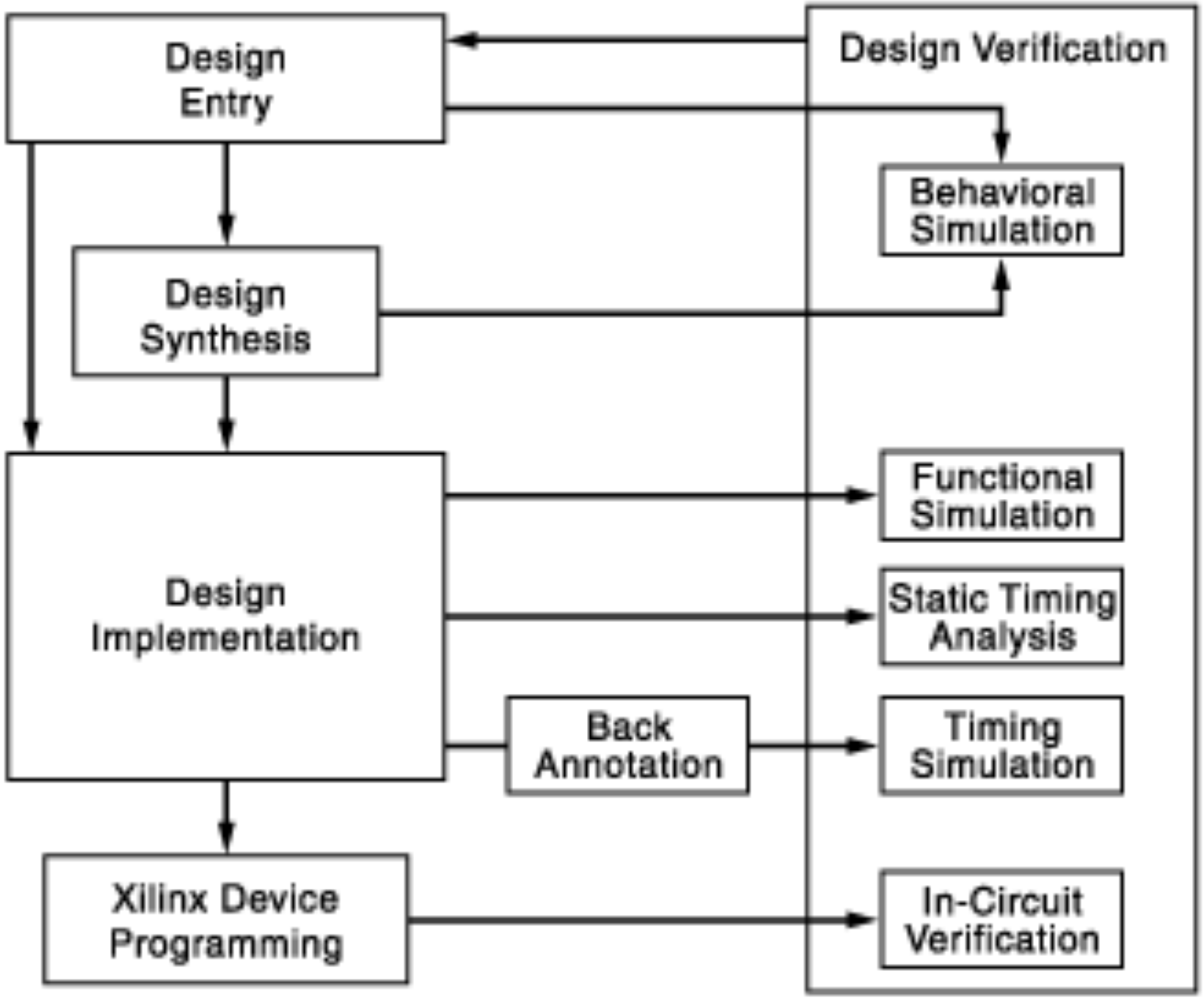
LSI Tool

FPGA Tool

Gate Assignment LE Place and Rout

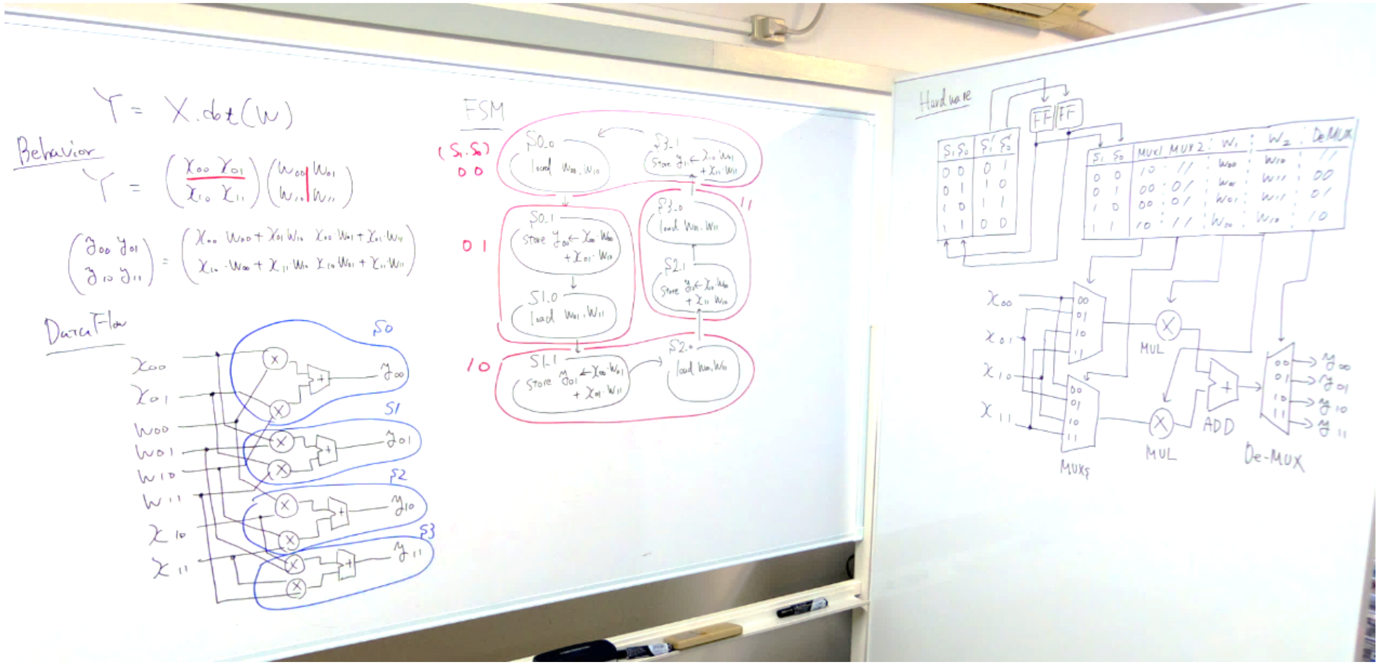
Configuration Data

FPGA Design Flow



How to write a HDL?

Y = X.dot(W)+B



Then, you can write your HDL!

Matrix Multiplication

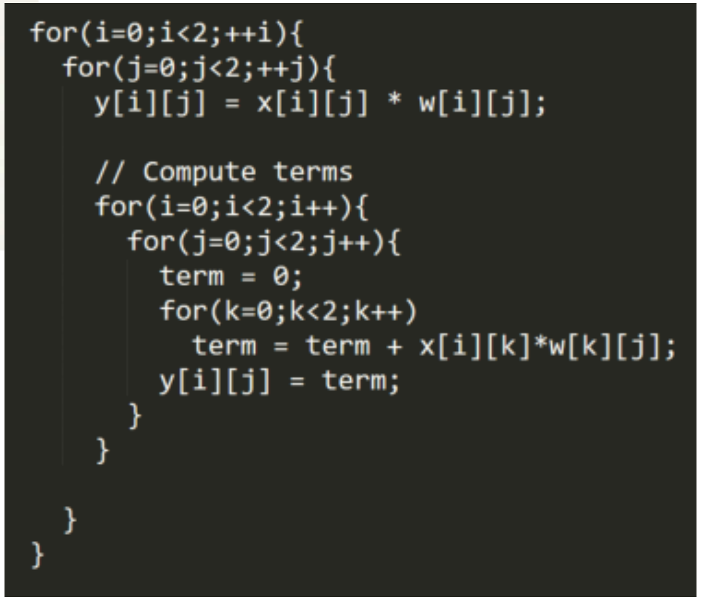
$$Y = X.\mathrm{dot}(W)$$

Behavior

$$Y = \begin{pmatrix} \underline{x_{00} \ x_{01}} \\ x_{10} \ x_{11} \end{pmatrix} \left(\begin{array}{c|c} w_{00} & w_{01} \\ w_{10} & w_{11} \end{array}\right)$$

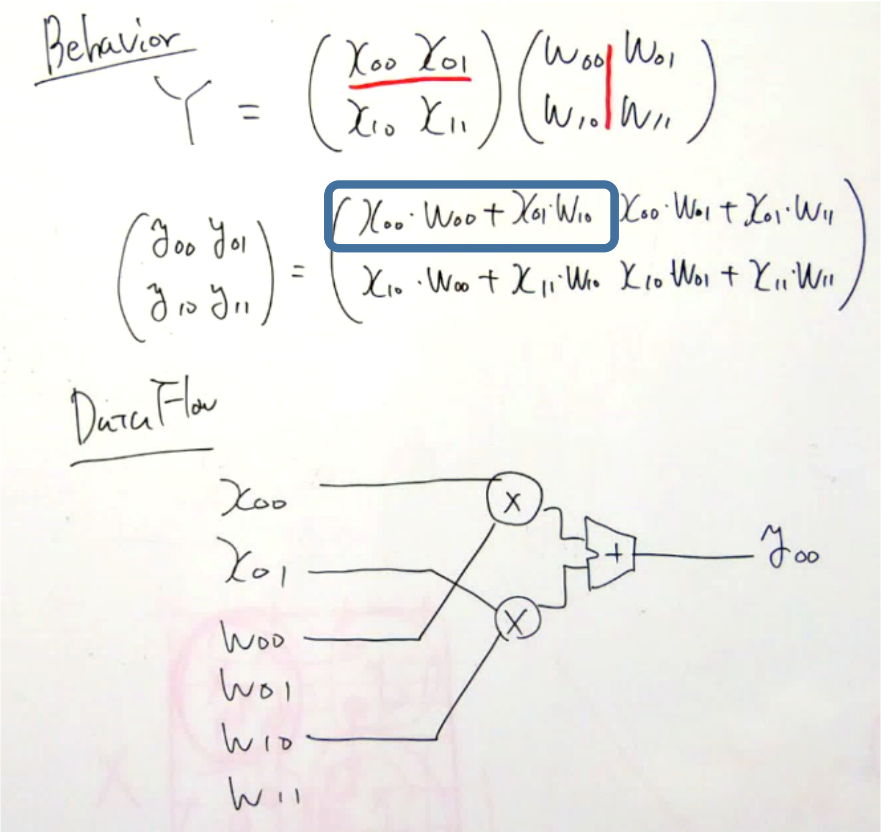
$$\begin{pmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \end{pmatrix} = \begin{pmatrix} x_{00} \cdot w_{00} + x_{01} \cdot w_{10} & x_{00} \cdot w_{01} + x_{01} \cdot w_{11} \\ x_{10} \cdot w_{00} + x_{11} \cdot w_{10} & x_{10} w_{01} + x_{11} \cdot w_{11} \end{pmatrix}$$

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

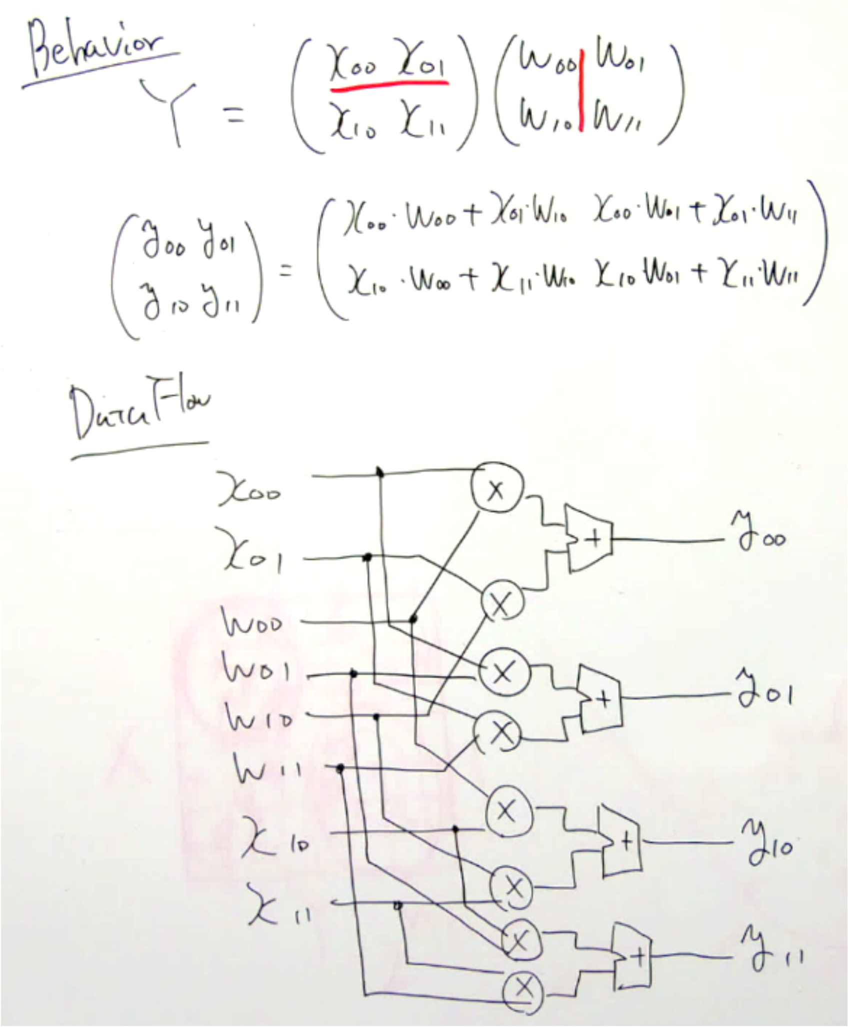


Behavior

$$Y = \begin{pmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{pmatrix} \begin{pmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{pmatrix}$$

$$\begin{pmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \end{pmatrix} = \begin{pmatrix} \boxed{x_{00} \cdot w_{00} + x_{01} \cdot w_{10}} & x_{00} \cdot w_{01} + x_{01} \cdot w_{11} \\ x_{10} \cdot w_{00} + x_{11} \cdot w_{10} & x_{10} w_{01} + x_{11} \cdot w_{11} \end{pmatrix}$$

DataFlow



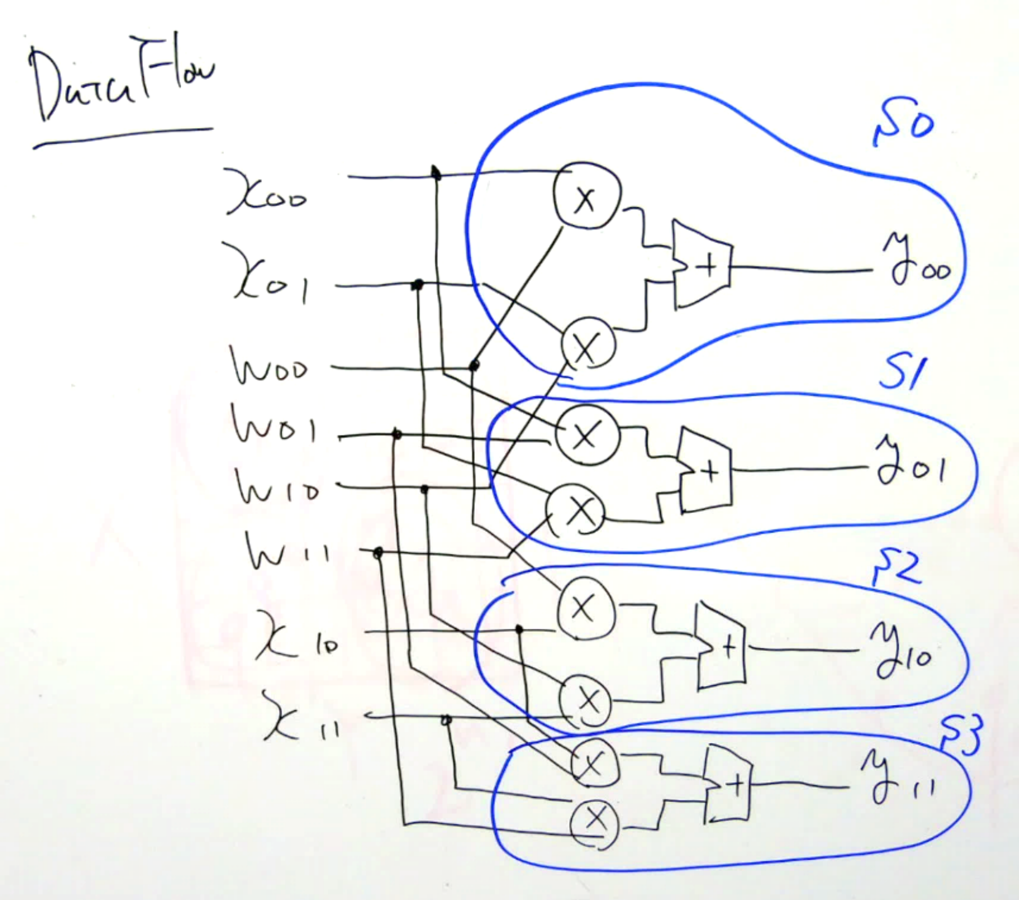
Behavior

$$Y = \begin{pmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{pmatrix} \begin{pmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{pmatrix}$$

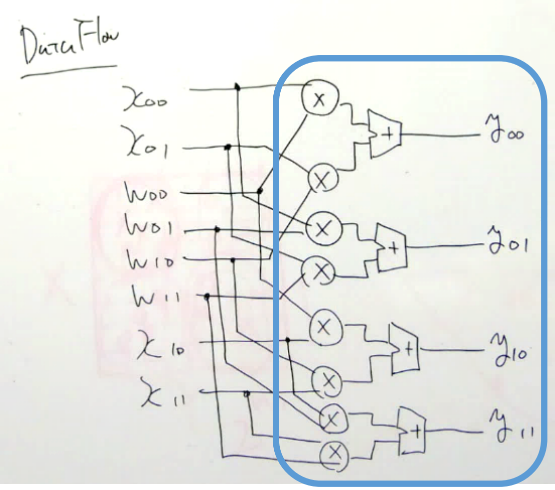
$$\begin{pmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \end{pmatrix} = \begin{pmatrix} x_{00} \cdot w_{00} + x_{01} \cdot w_{10} & x_{00} \cdot w_{01} + x_{01} \cdot w_{11} \\ x_{10} \cdot w_{00} + x_{11} \cdot w_{10} & x_{10} w_{01} + x_{11} \cdot w_{11} \end{pmatrix}$$

DataFlow

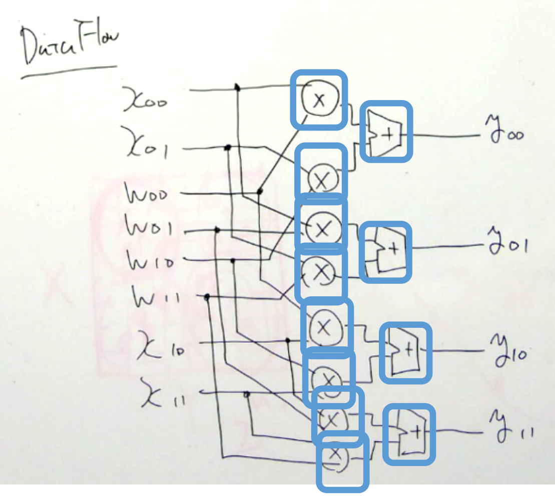
Scheduling



MACs (Multiply ACcumulations) realization

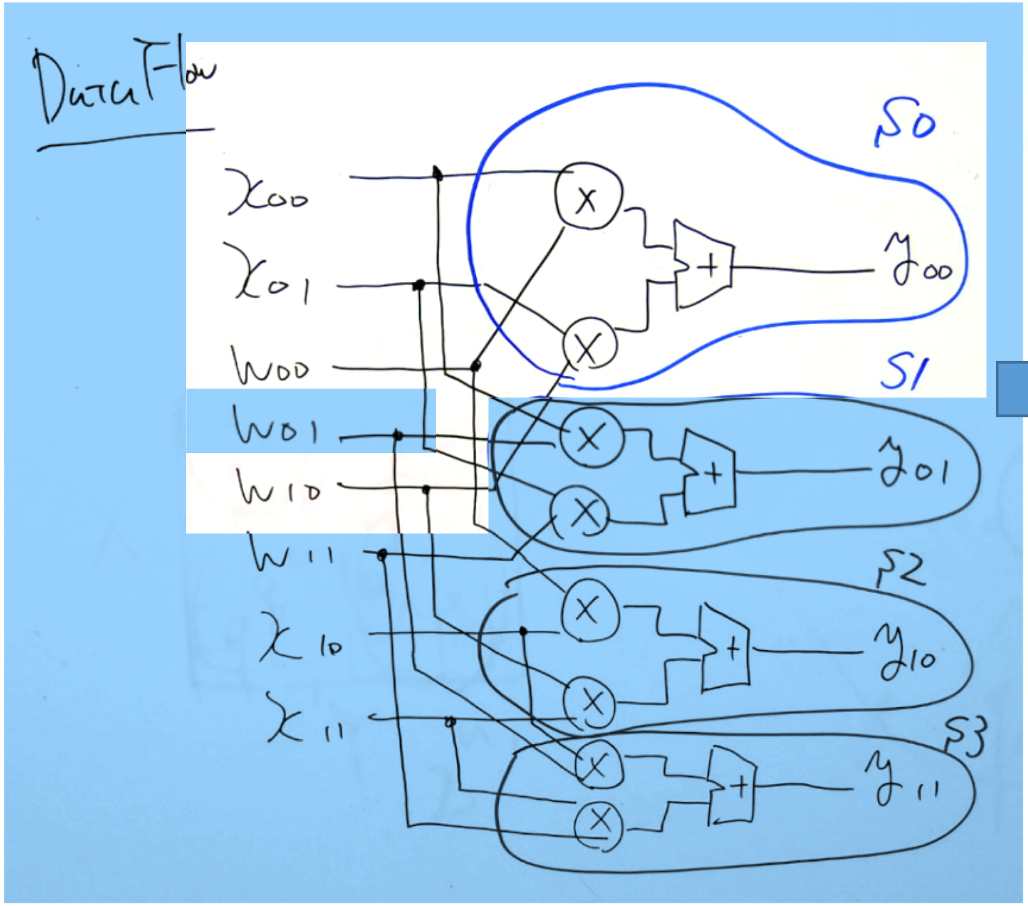


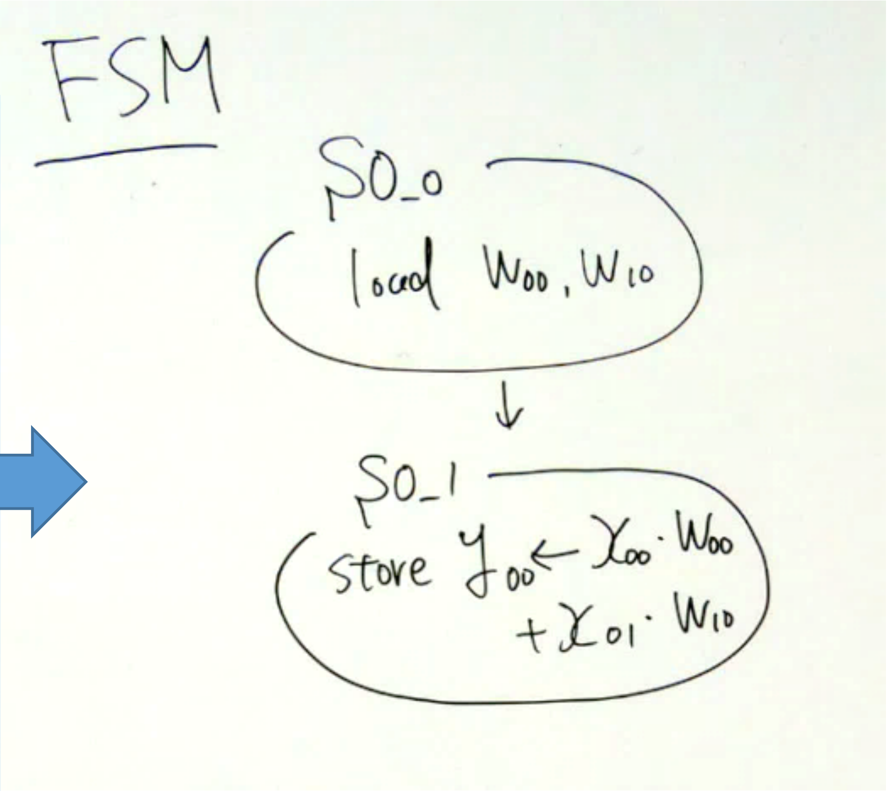
Fully parallel realization



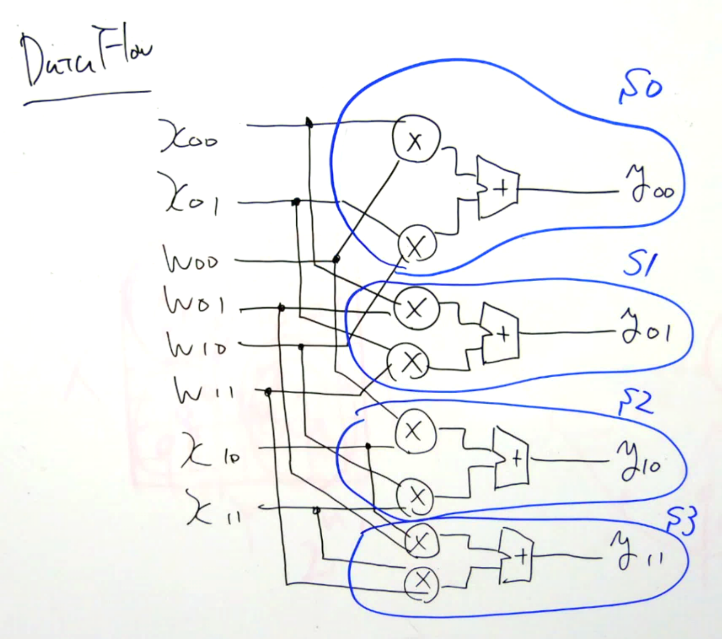
Sequential realization

Finite State Machine (FSM)

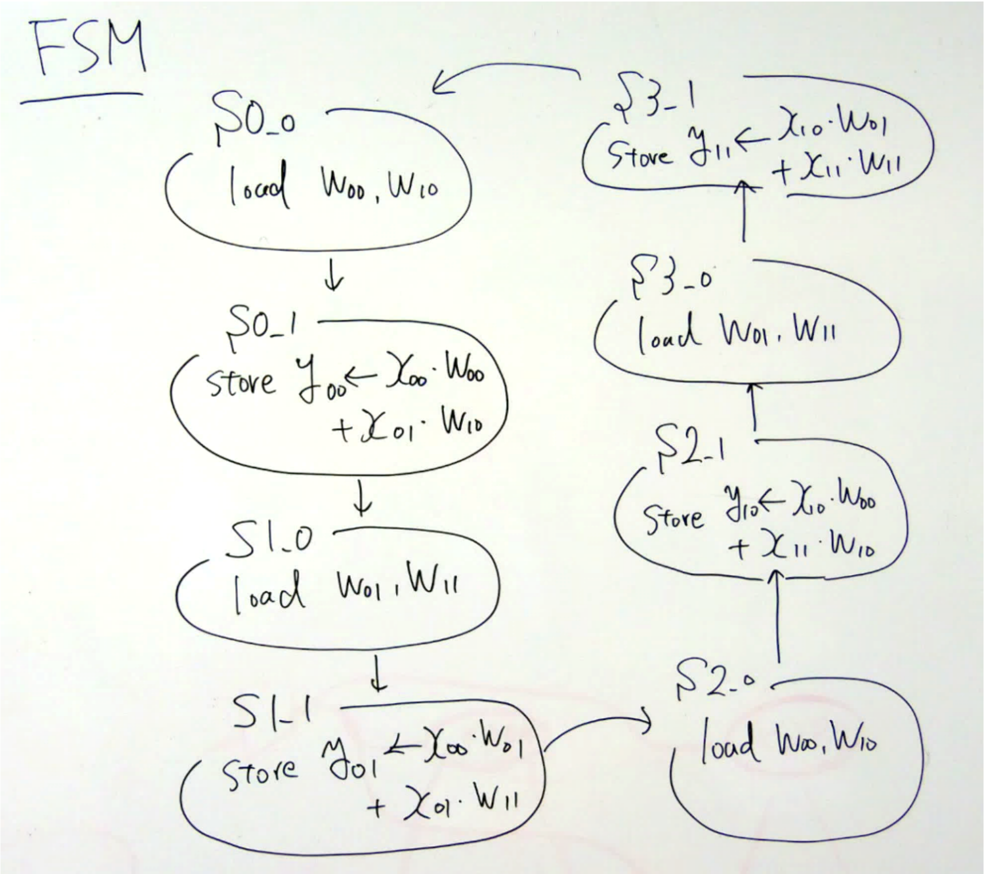




FSM (Cont.)

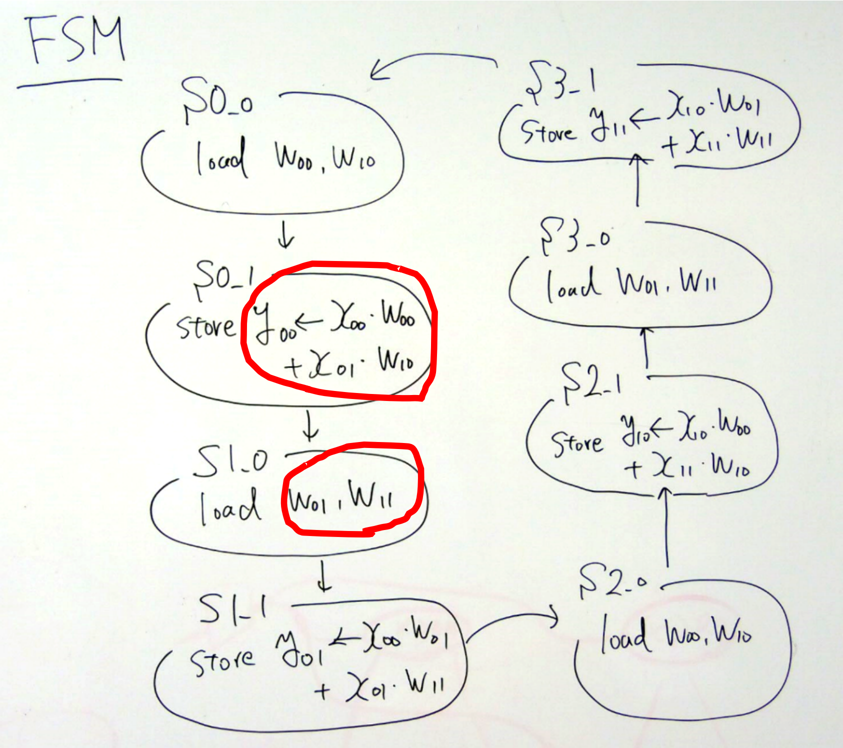


DataFlow
x_{00}
x_{01}
w_{00}
w_{01}
w_{10}
w_{11}
x_{10}
x_{11}
S0
S1
S2
S3
y_{00}
y_{01}
y_{10}
y_{11}

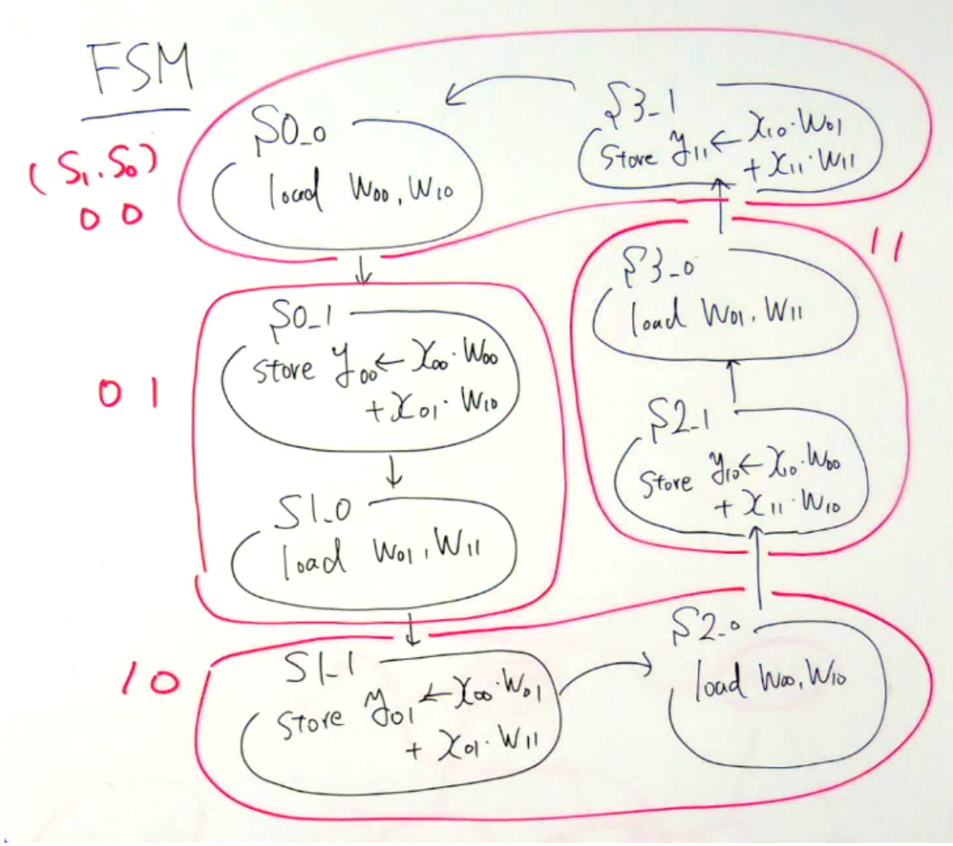


FSM
S0_0
load W_{00}, W_{10}
S0_1
store $y_{00} \leftarrow x_{00} \cdot W_{00} + x_{01} \cdot W_{10}$
S1_0
load W_{01}, W_{11}
S1_1
Store $y_{01} \leftarrow x_{00} \cdot W_{01} + x_{01} \cdot W_{11}$
S2_0
load W_{00}, W_{10}
S2_1
Store $y_{10} \leftarrow x_{10} \cdot W_{00} + x_{11} \cdot W_{10}$
S3_0
load W_{01}, W_{11}
S3_1
Store $y_{11} \leftarrow x_{10} \cdot W_{01} + x_{11} \cdot W_{11}$

FSM Optimization

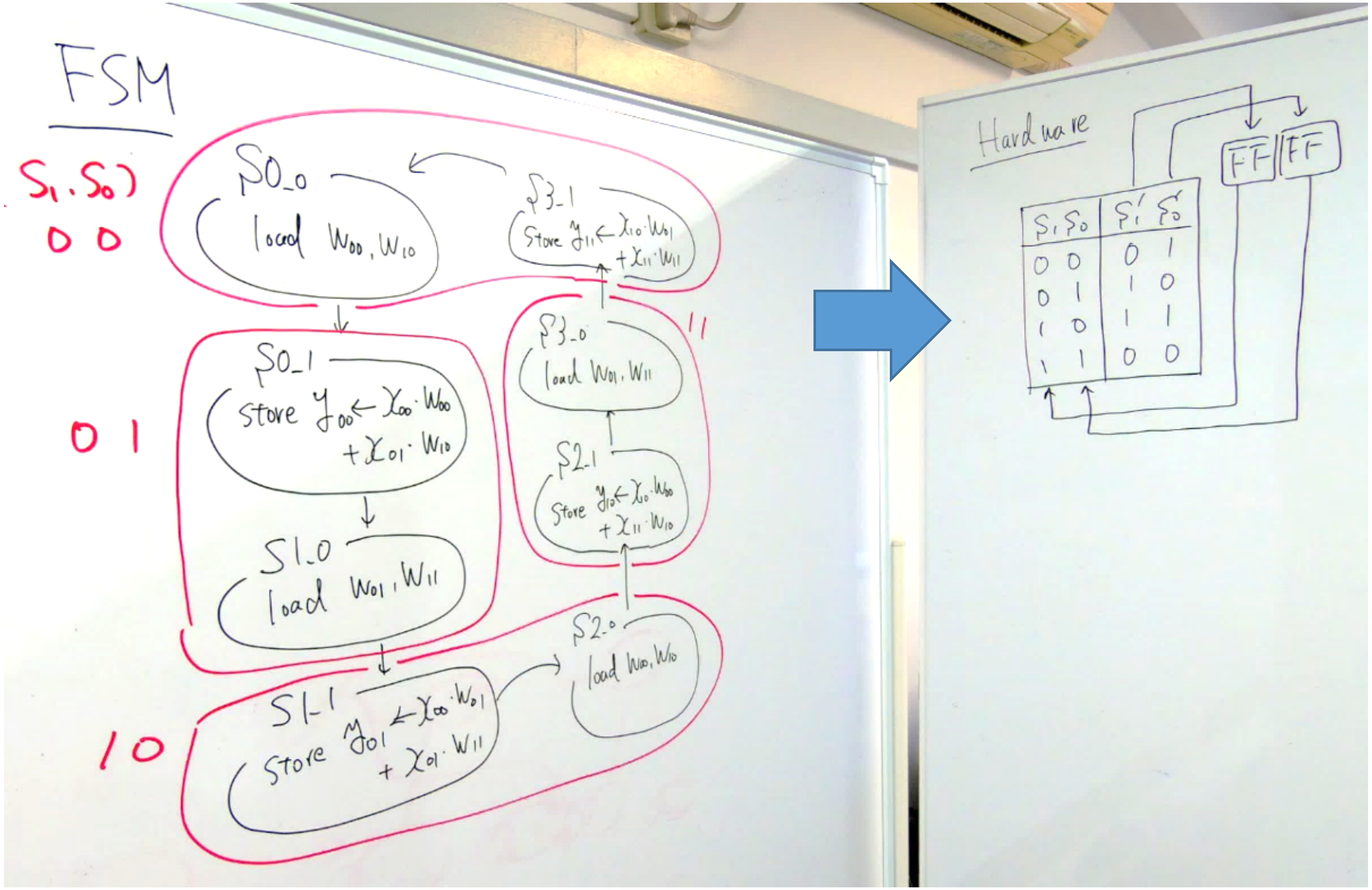

FSM
S0_0
load W_{00}, W_{10}
S0_1
store $y_{00} \leftarrow x_{00} \cdot W_{00} + x_{01} \cdot W_{10}$
S1_0
load W_{01}, W_{11}
S1_1
store $y_{01} \leftarrow x_{00} \cdot W_{01} + x_{01} \cdot W_{11}$
S2_0
load W_{00}, W_{10}
S2_1
store $y_{10} \leftarrow x_{10} \cdot W_{00} + x_{11} \cdot W_{10}$
S3_0
load W_{01}, W_{11}
S3_1
store $y_{11} \leftarrow x_{10} \cdot W_{01} + x_{11} \cdot W_{11}$

Load weights during the MAC operation


FSM
(S_1, S_0)
0 0
0 1
1 0
1 1
S0_0
load W_{00}, W_{10}
S0_1
store $y_{00} \leftarrow x_{00} \cdot W_{00} + x_{01} \cdot W_{10}$
S1_0
load W_{01}, W_{11}
S1_1
store $y_{01} \leftarrow x_{00} \cdot W_{01} + x_{01} \cdot W_{11}$
S2_0
load W_{00}, W_{10}
S2_1
store $y_{10} \leftarrow x_{10} \cdot W_{00} + x_{11} \cdot W_{10}$
S3_0
load W_{01}, W_{11}
S3_1
store $y_{11} \leftarrow x_{10} \cdot W_{01} + x_{11} \cdot W_{11}$

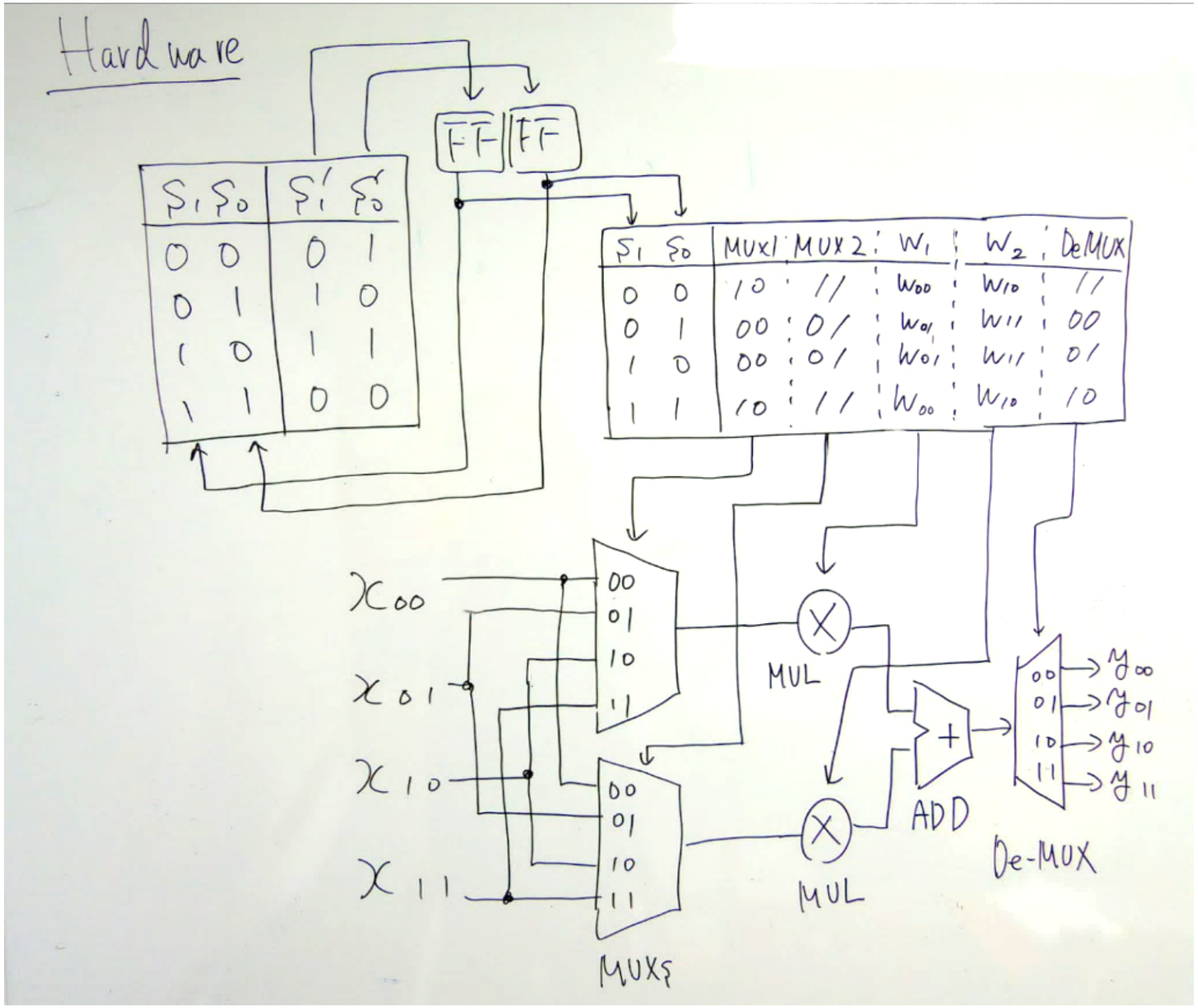
Assign state variables (1-hot code, gray one, natural binary one)

Realization of the FSM



S1	S0	S1'	S0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Assign to Primitives



S_1	S_0	S_1'	S_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

S_1	S_0	MUX1	MUX2	W_1	W_2	DeMUX
0	0	10	11	W_{00}	W_{10}	11
0	1	00	01	W_{01}	W_{11}	00
1	0	00	01	W_{01}	W_{11}	01
1	1	10	11	W_{00}	W_{10}	10

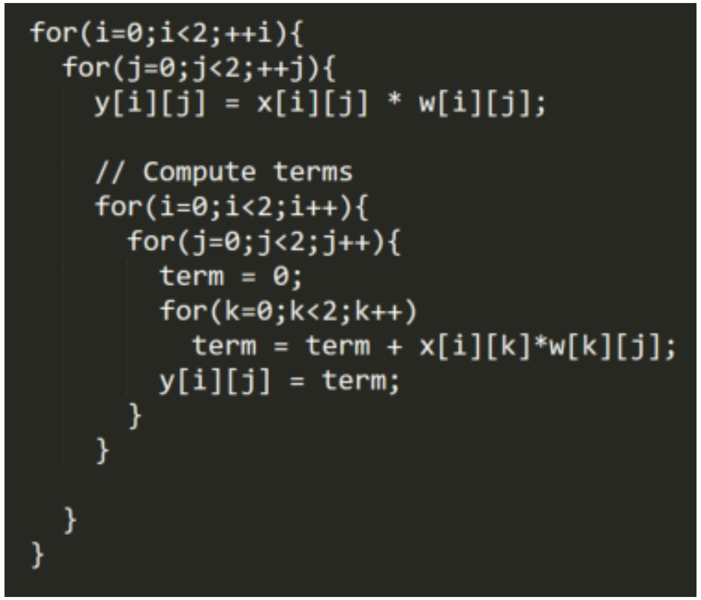
Control

Data path

Comparison of # Lines

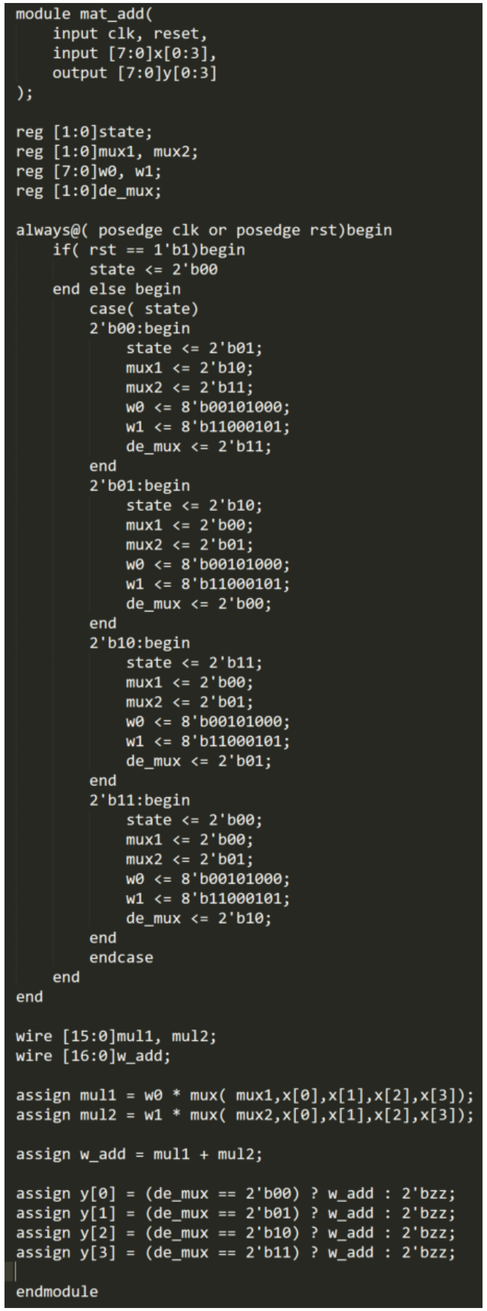
Y = X.dot(W)+B

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

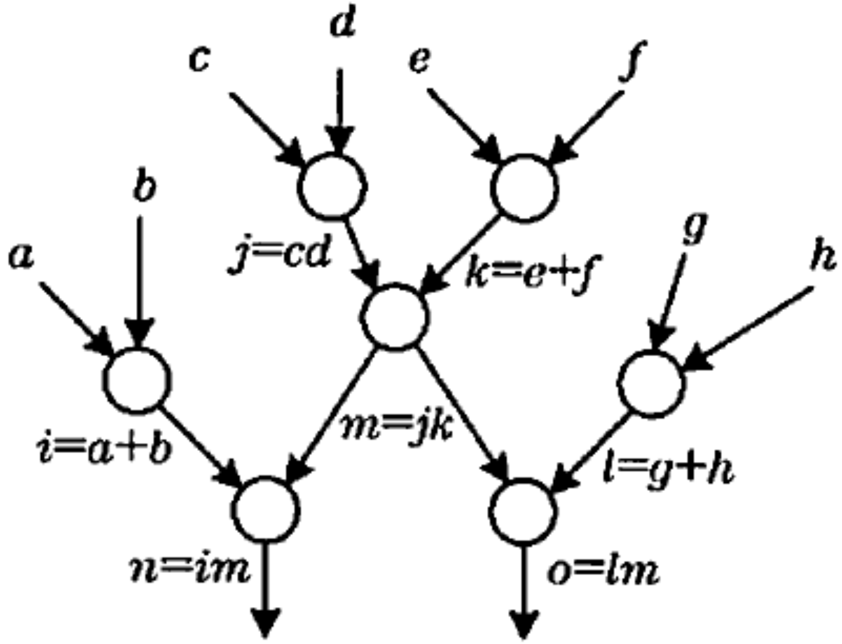


```
module mat_add(
    input clk, reset,
    input [7:0]x[0:3],
    output [7:0]y[0:3]
);

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
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

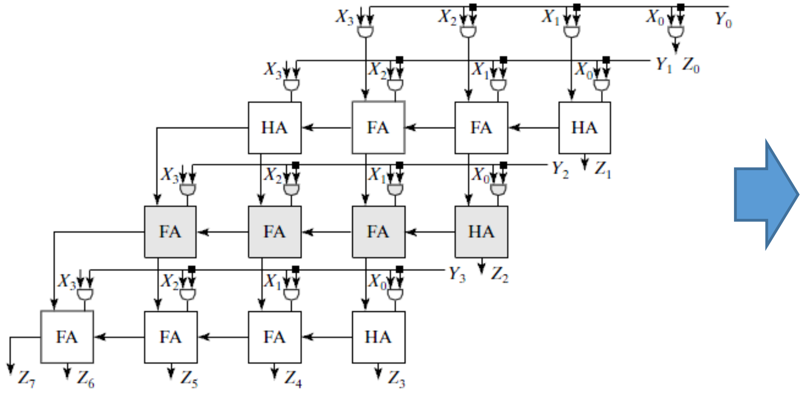
- Representation of a combinational logic circuit using a directed graph without a cycle
- Vertex：Logic gate, Edge：Input or output



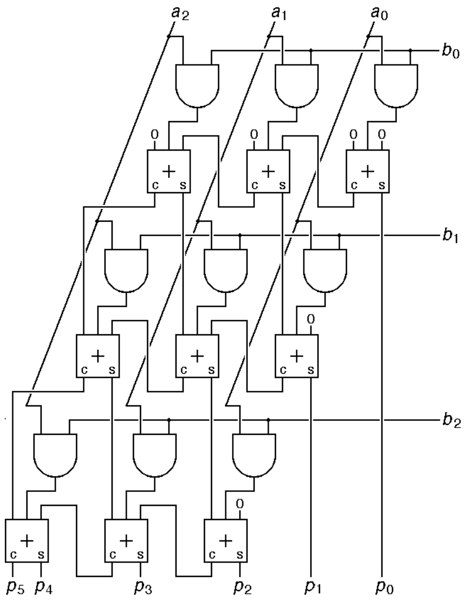
Logic Synthesis

- Synthesize from a given HDL specification to a Boolean network

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

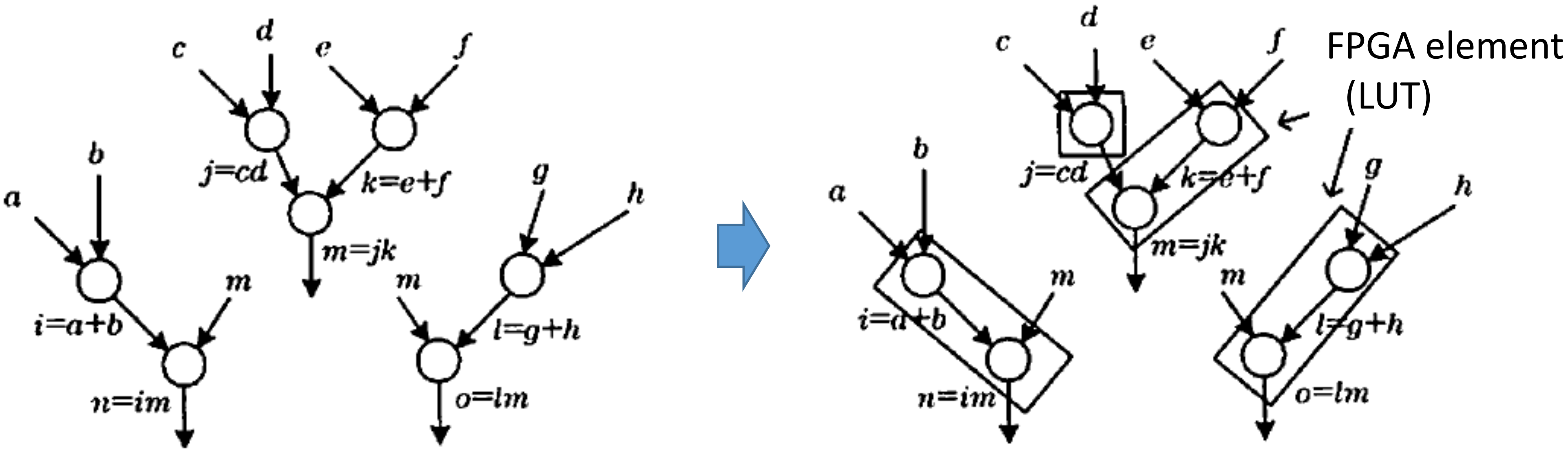


X_3 X_2 X_1 X_0 Y_0
X_3 X_2 X_1 X_0 Y_1 Z_0
HA FA FA HA
X_3 X_2 X_1 X_0 Y_2 Z_1
FA FA FA HA
X_3 X_2 X_1 X_0 Y_3 Z_2
FA FA FA HA
Z_7 Z_6 Z_5 Z_4 Z_3



Technology Mapping

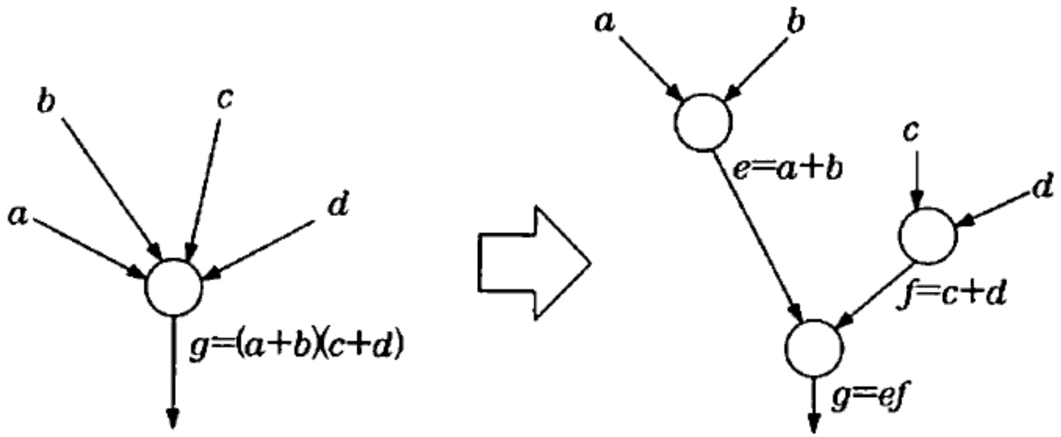
- A kind of a graph covering problem
- Goal: A depth optimized one by using a dynamic programming



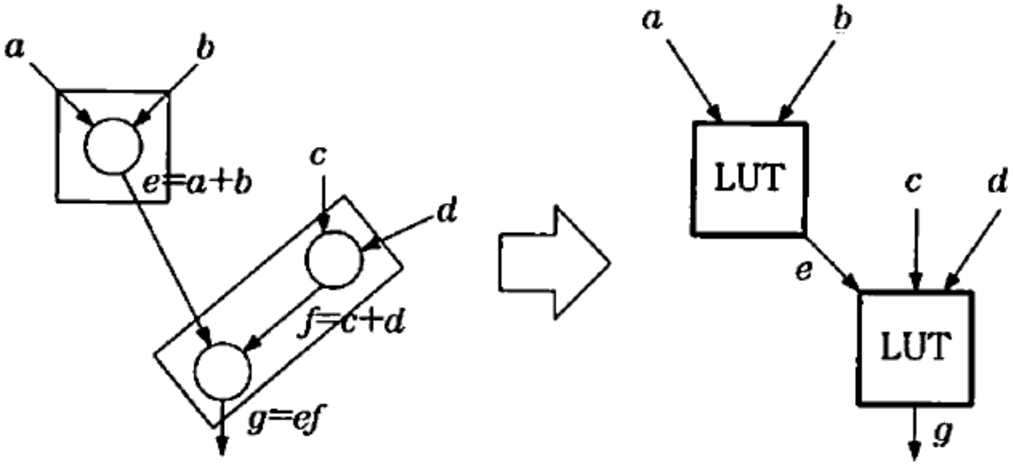
FPGA element (LUT)

Decomposition and Covering

Decomposition

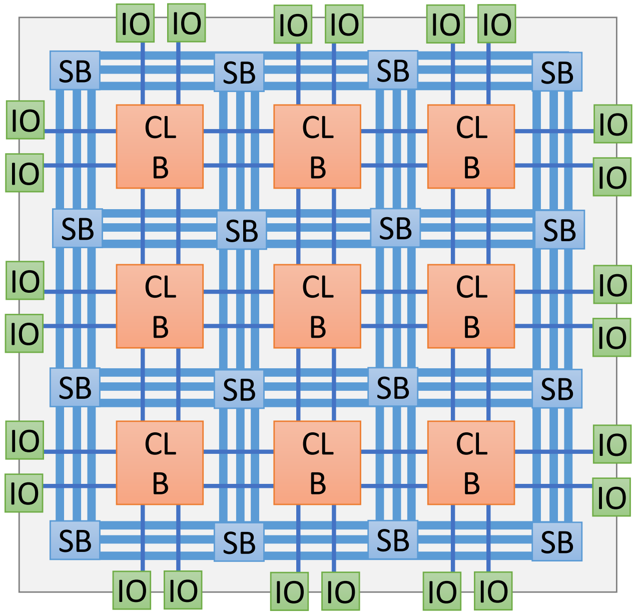


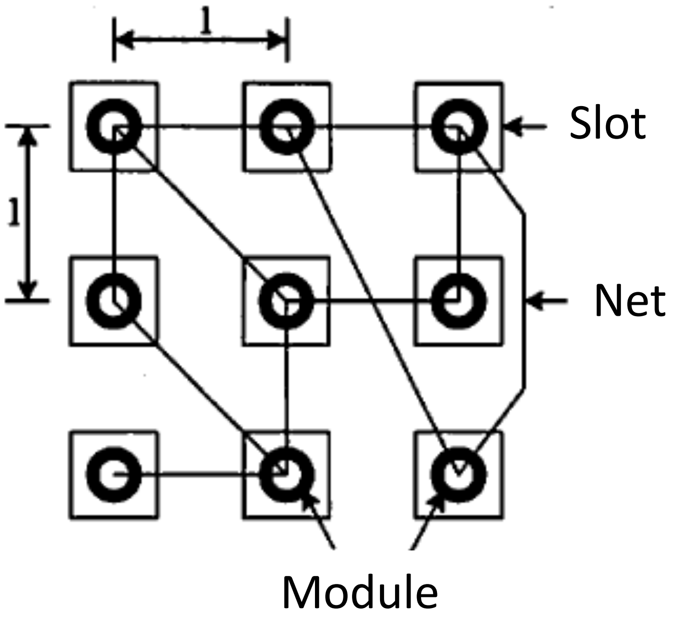
Covering



Placement

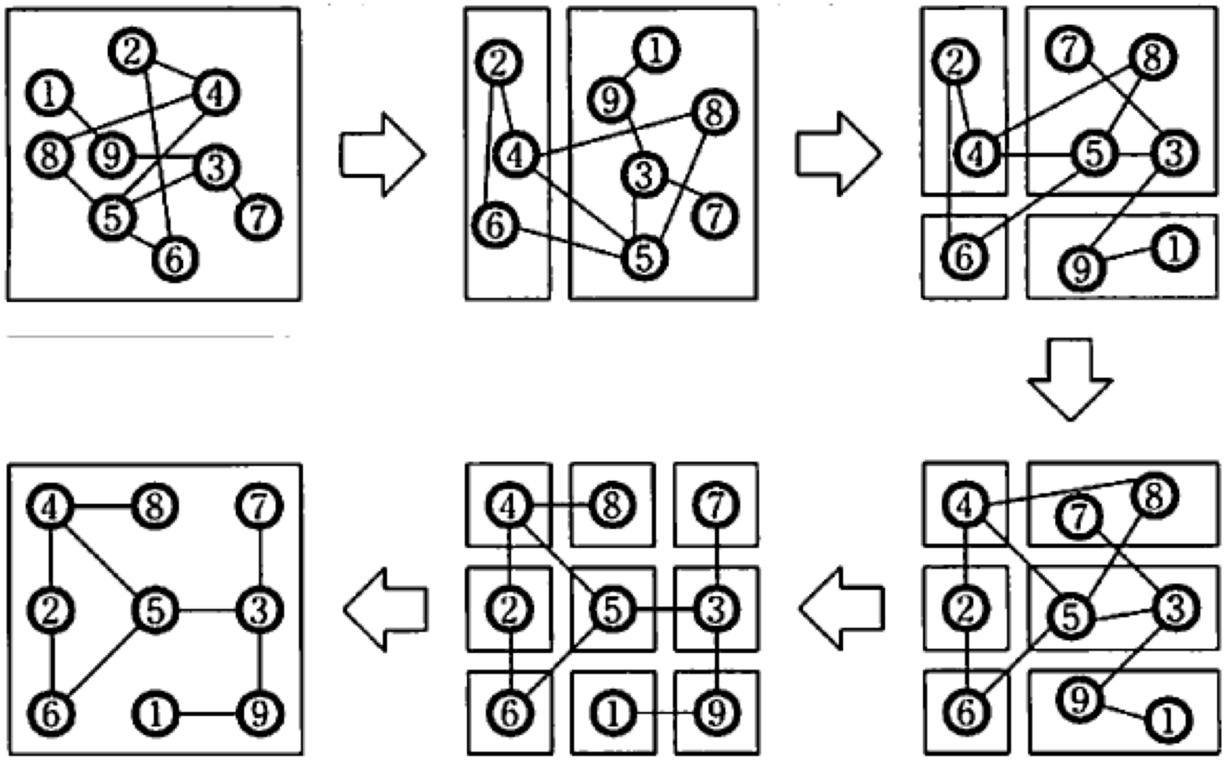
- Problem to place the module (logic gate) into the slot (location)
  - 2D allocation problem → NP-complete
  - Approximation (Simulated annealing, or min-cut tech.)





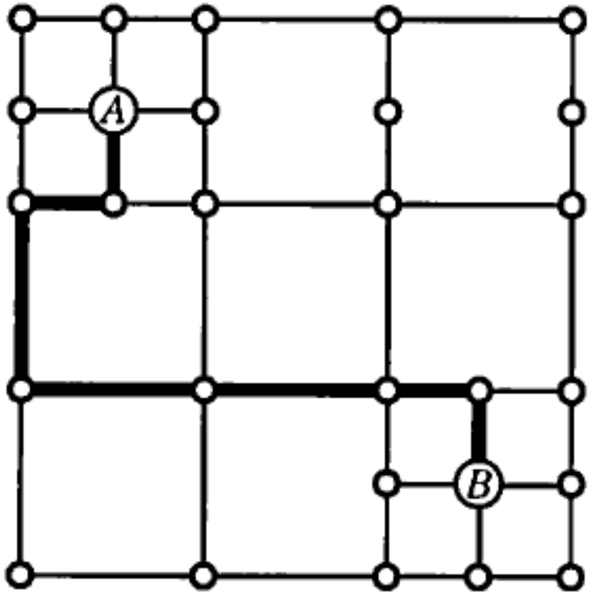
Min-cut Placement

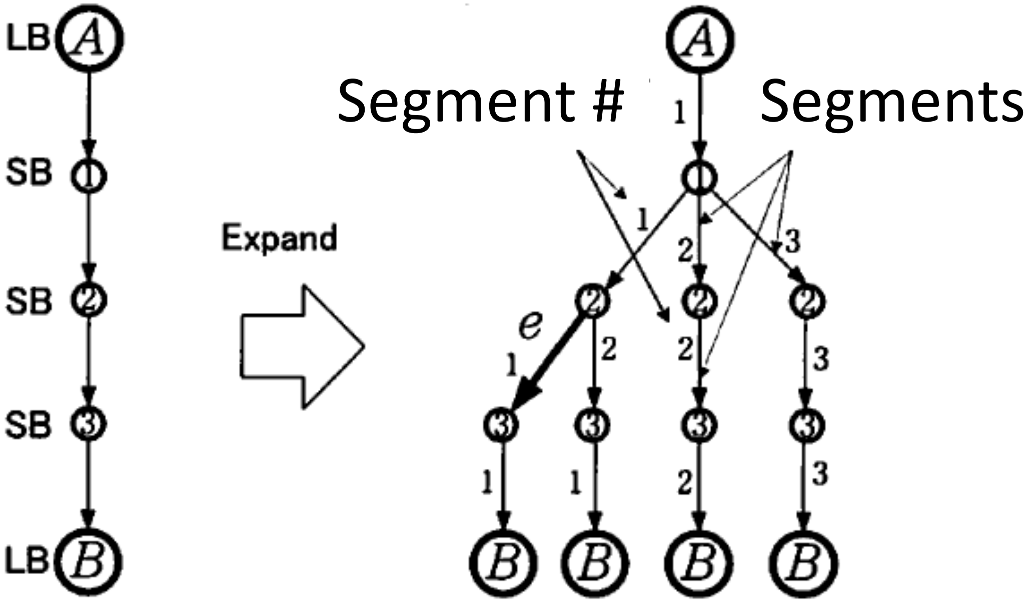
- Recursively divided a module set into two ones
    - Near-optimal solution can be obtained in the short time
- Used in a commercial FPGA design tool



Routing

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





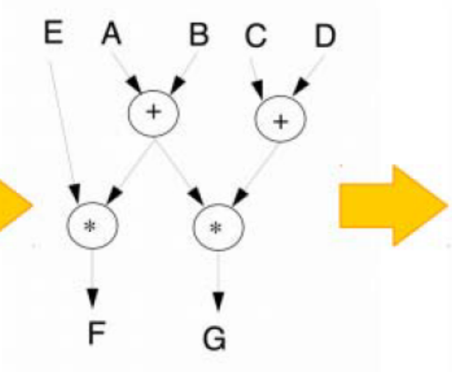
Global routing

Local routing

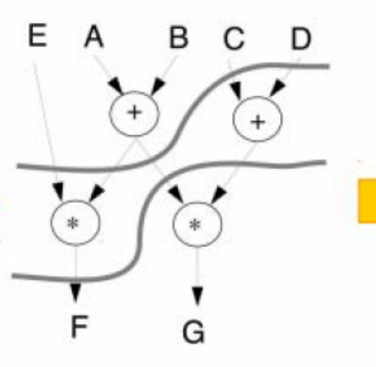
High-Level Synthesis (HLS)

```
PROCEDURE Test;
 VAR
A,B,C,D,E,F,G:integer;
 BEGIN
  Read(A,B,C,D,E);
  F := E*(A+B);
  G := (A+B)*(C+D);
  ...
 END;
```

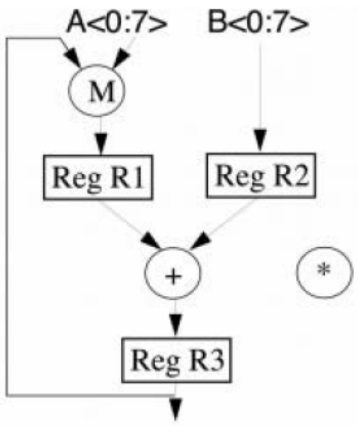
Input Behavioral Spec.



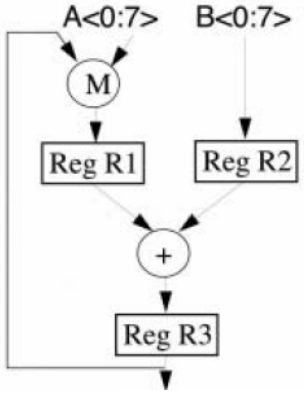
Dataflow



Scheduling



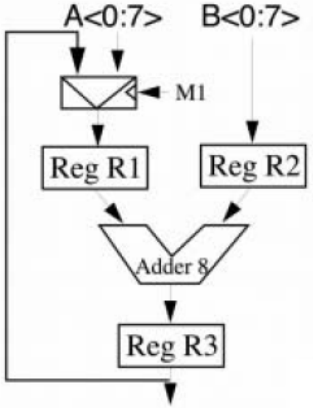
Data-path generation



Controller description:

```
S1: M1=1, Load R1 next S2;
S2: Load R2 next S3;
S3: Add, Load R3 next S4;
S4: M1=0, Load R1 next...
```

Controller (FSM) Generation



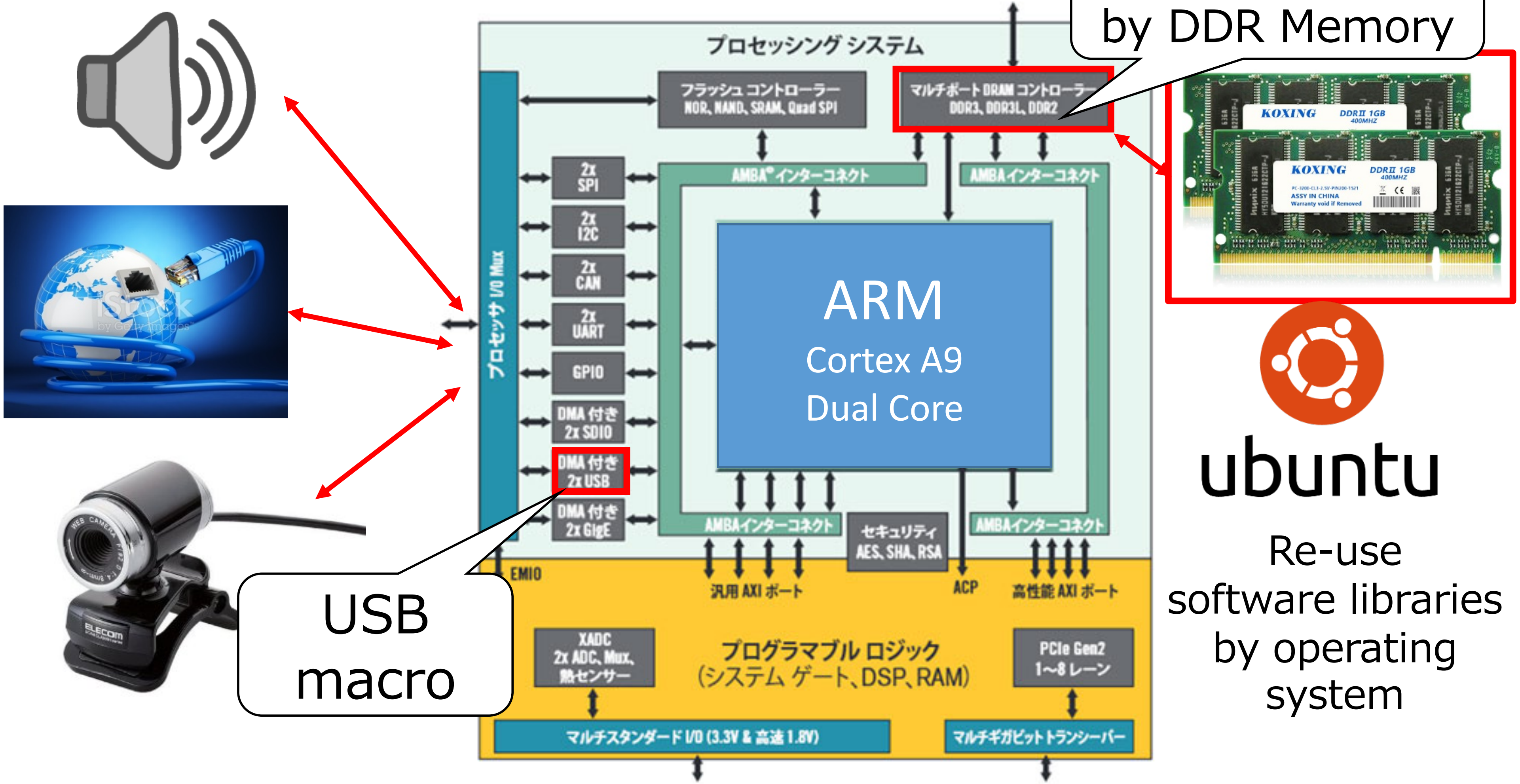
Controller ROM:

```
0000 : 11000000 0001
0001 : 00100000 0010
0010 : 00011000 0011
0011 : 01000000 0100
```

Mapping to resources (Binding)

System on Chip FPGA

- Xilinx: Zynq, Intel: Intel SoC



ARM
Cortex A9
Dual Core

Data Storage
by DDR Memory

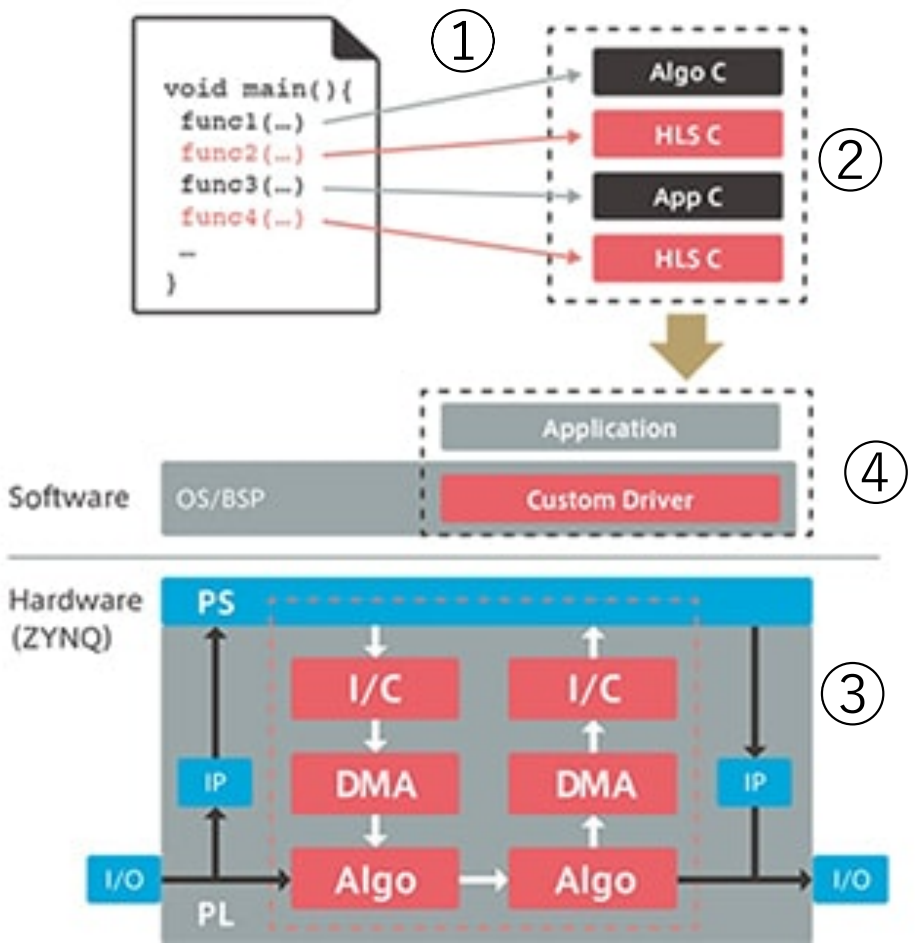
USB
macro

ubuntu

Re-use
software libraries
by operating
system

Source: Xilinx Inc. Zynq-7000 All Programmable SoC

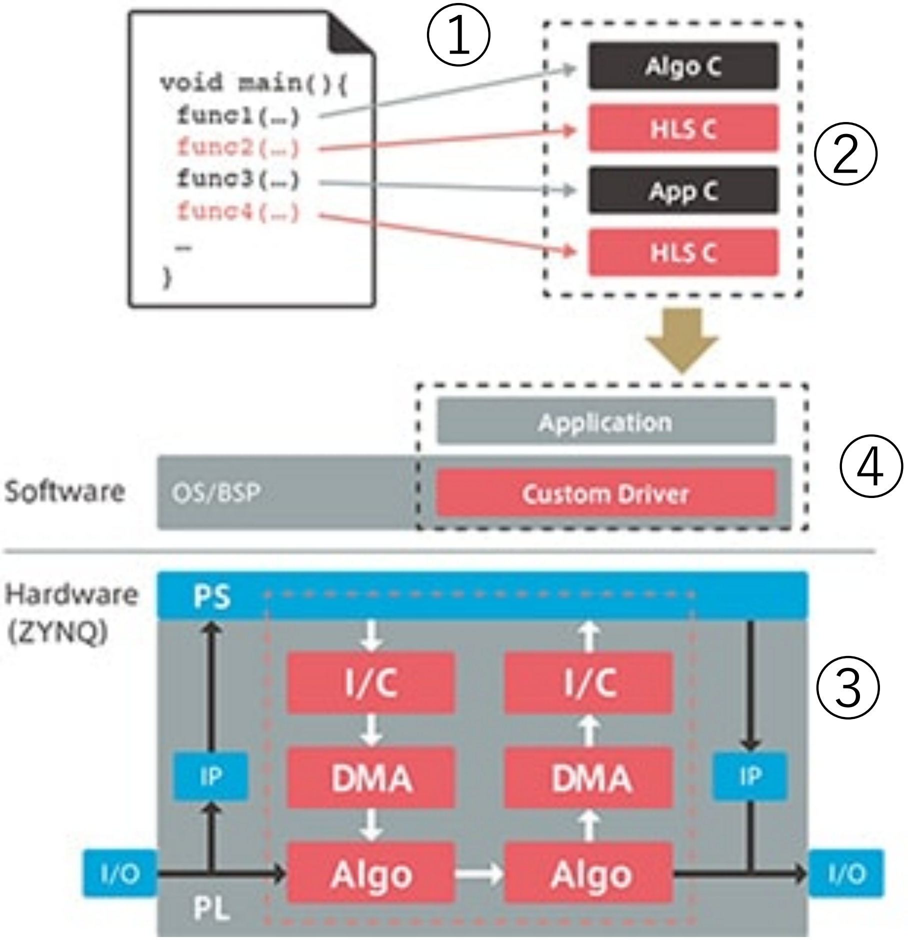
Conventional Design Flow for the SoC FPGA



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

Source: Xilinx Inc.

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

↓

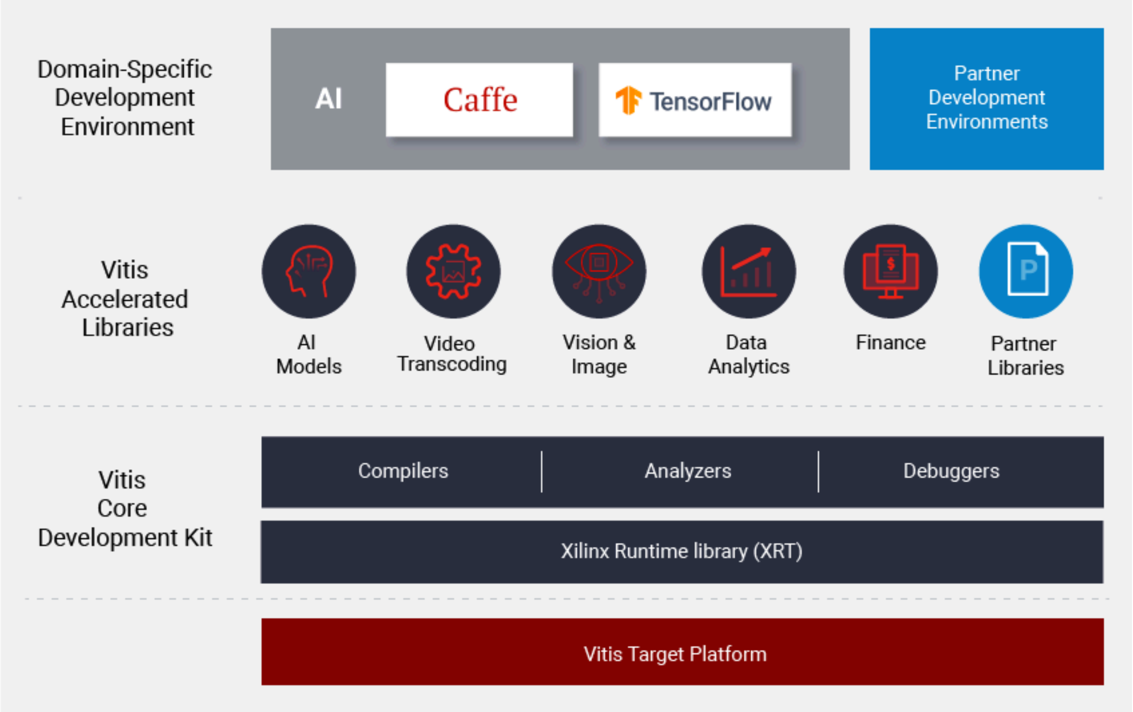
Automatically done

Source: Xilinx Inc.

Design Flow for Versal Architecture:

e.g. AI Accelerator

- Application specific framework (Caffe, TensorFlow)
- Convert pre-trained model with Vitis Libraries
- Conventional system-design tool-flow


Domain-Specific Development Environment
AI
Caffe
TensorFlow
Partner Development Environments
Vitis Accelerated Libraries
AI Models
Video Transcoding
Vision & Image
Data Analytics
Finance
Partner Libraries
Vitis Core Development Kit
Compilers
Analyzers
Debuggers
Xilinx Runtime library (XRT)
Vitis Target Platform

Source: Xilinx Inc.

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

Homework 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 vender adopts the anti-fuse-based FPGA than SRAM-based one. Why?

Deadline is 21st, Nov., 2019

(At the beginning of the next lecture)

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