

Parallel and Reconfigurable VLSI Computing (1)

FPGA Introduction

Hiroki Nakahara

Tokyo Institute of Technology

Outline

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

FY'20 Schedule

6/23 1 Tutorial & FPGA Basis

6/26 2 Hardware Preliminary

6/30 3 FPGA Architecture

7/ 3 4 FPGA Synthesis Flow

7/ 7 5 Walk Through FPGA Design

7/10 6 RTL Design Introduction

7/14 7 Practical RTL Design

7/17 8 RTL Design: Tiny Processor

7/21 9 High-Level Synthesis (HLS)

Design: Introduction

7/24 Cancel

7/28 10 HLS Optimizations

7/31 11 Practical HLS Design

8/ 4 12 Complexity of Logic Functions,
and its Decomposition:

Synthesis for an FPGA

8/ 7 13 AI on an FPGA

Hands-on design exercise using FPGA

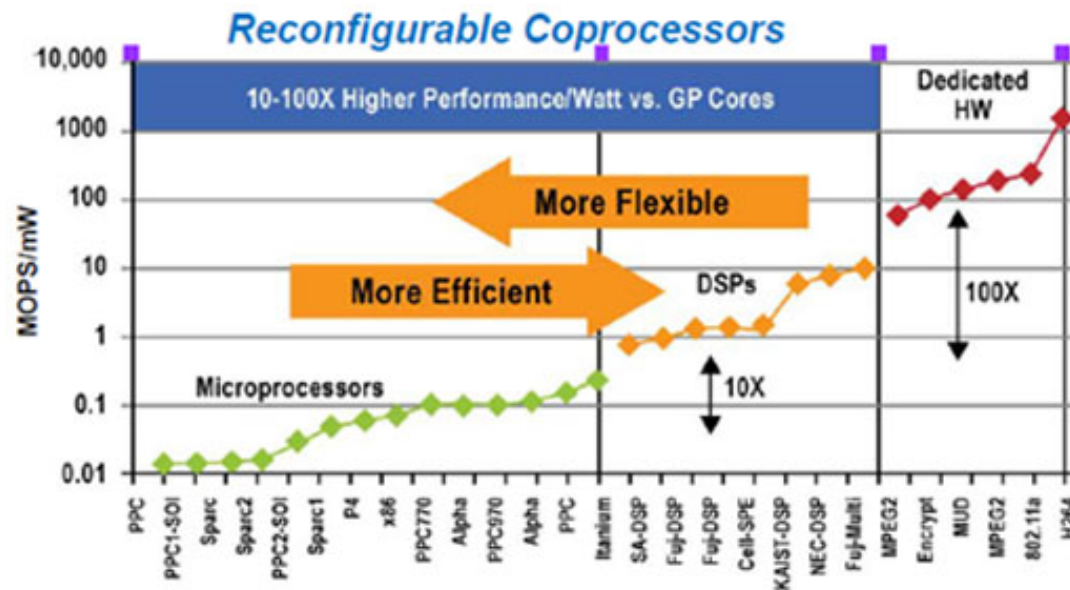
Evaluation

- Report: TBD
- Exercises
 - Submit by PDF file to OCW-i
- Lecture Slides:
 - > TOKYO TECH OCW

FPGA Basis

The Dilemma: Flexibility vs. Efficiency

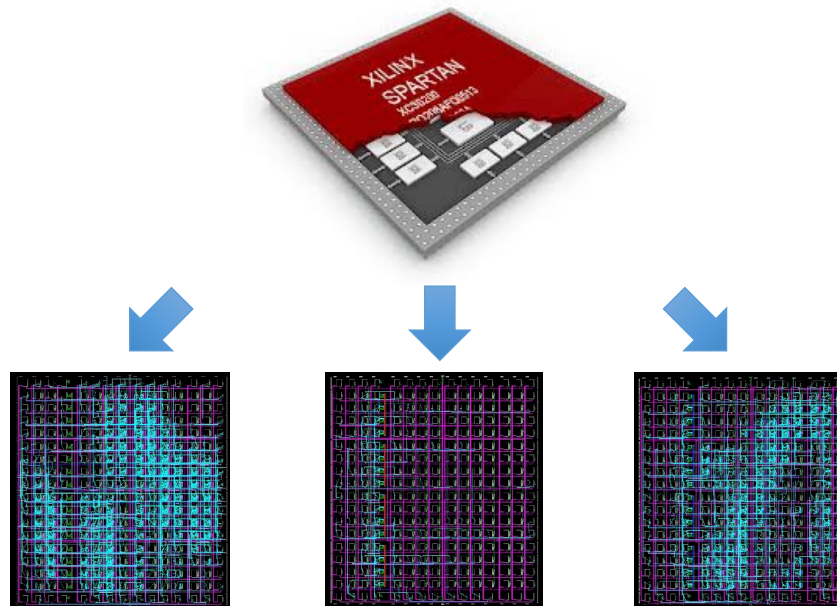
- FPGAs often offer the best of both worlds – replacing MPUs, DSPs, and dedicated ASSPs or ASICs
- Their on-the-fly reconfigurability helps them realize in-system logic functions that CPUs can't



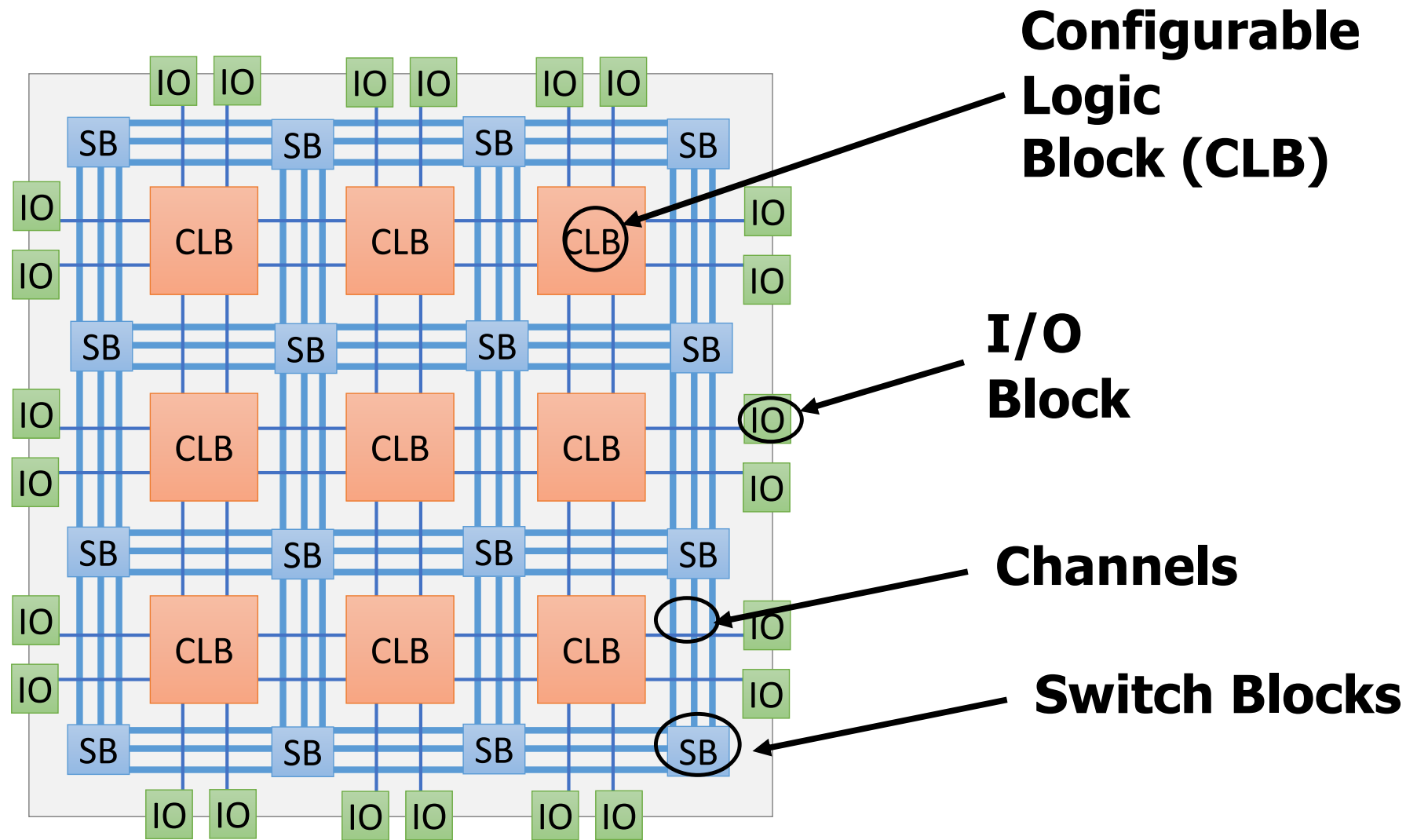
Source: "High-Performance Energy-Efficient Reconfigurable Accelerator Circuits for the Sub-45nm Era", July 2011
by Ram K. Krishnamurthy, Circuits Research Labs, Intel Corp.

FPGA

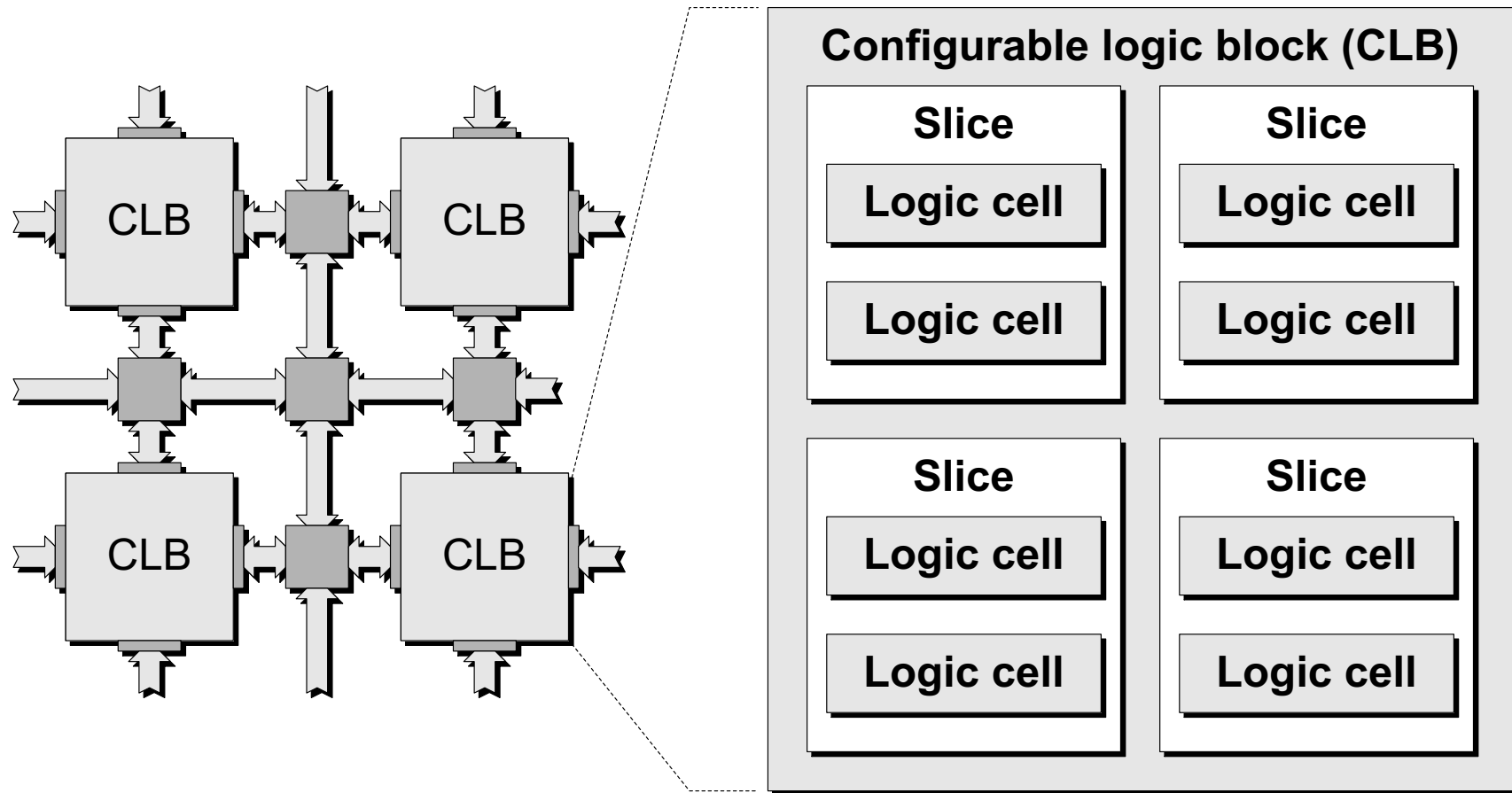
- Reconfigurable LSI or Programmable Hardware
- Programmable Logic Array and Programmable Interconnection
- Programmed by Reconfigurable Data



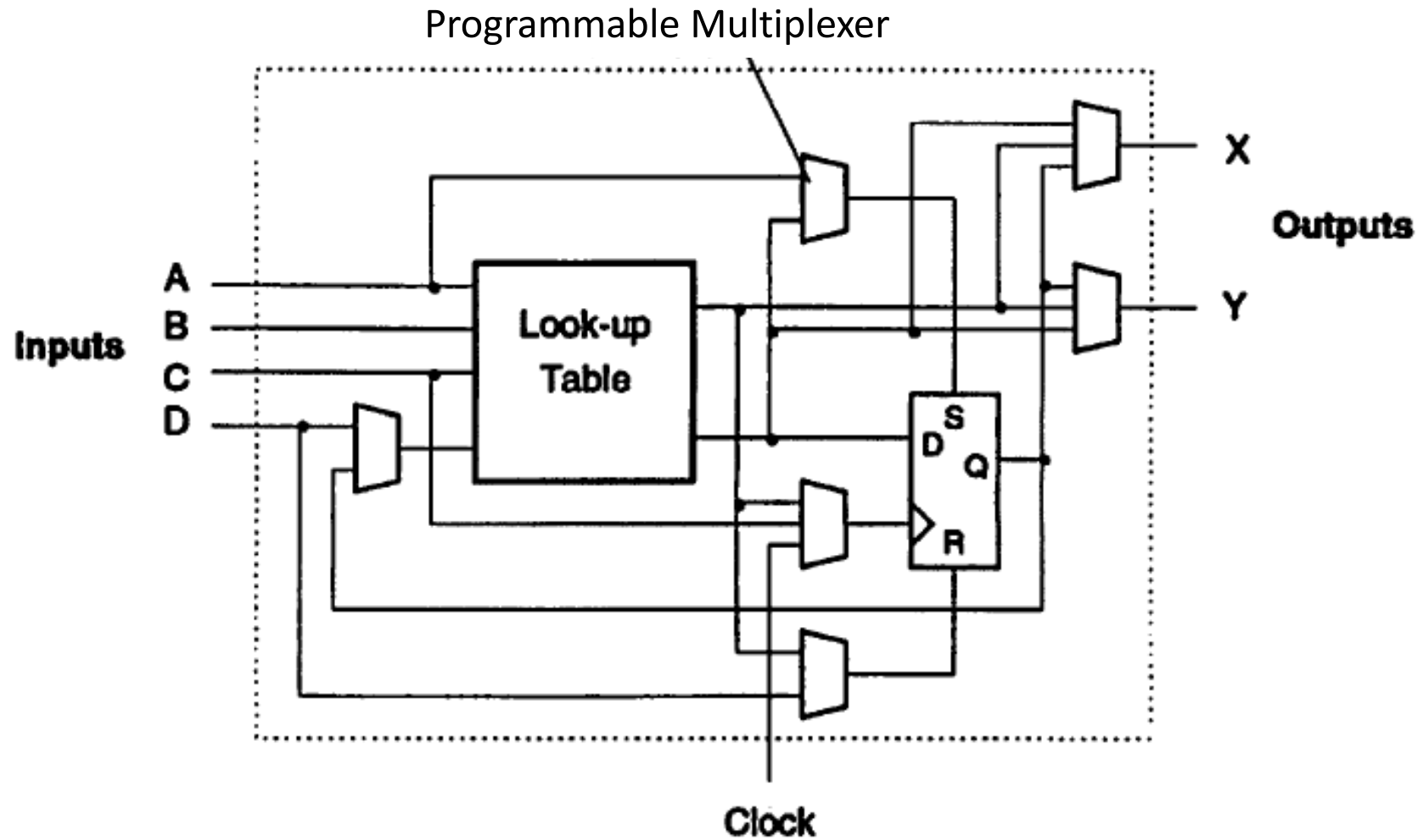
Island Style FPGA



Xilinx CLB

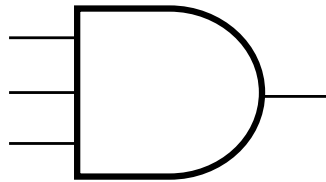


Logic Cell (Xilinx Inc. XC2000)

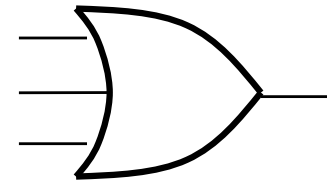


Realization of a Logic Function

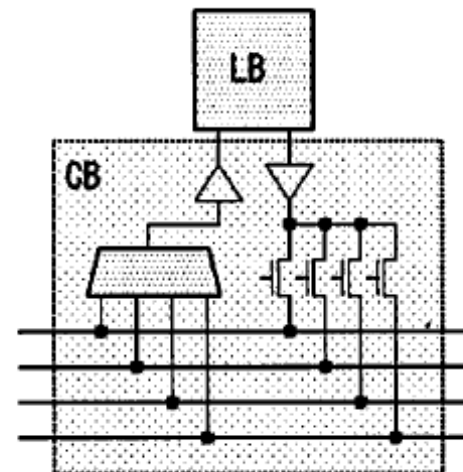
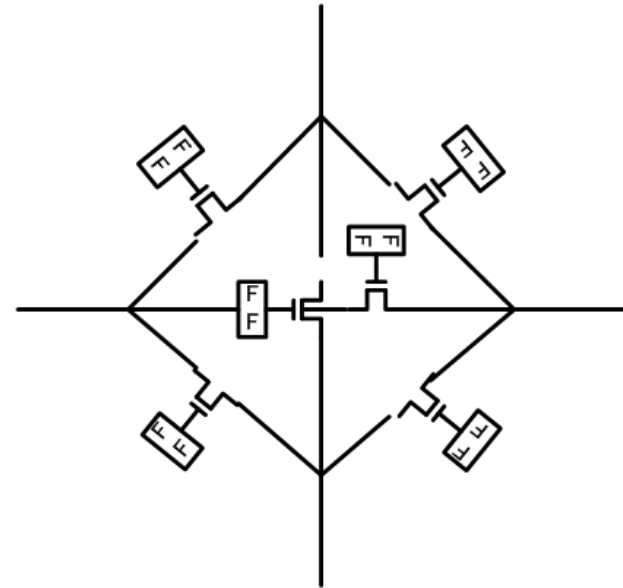
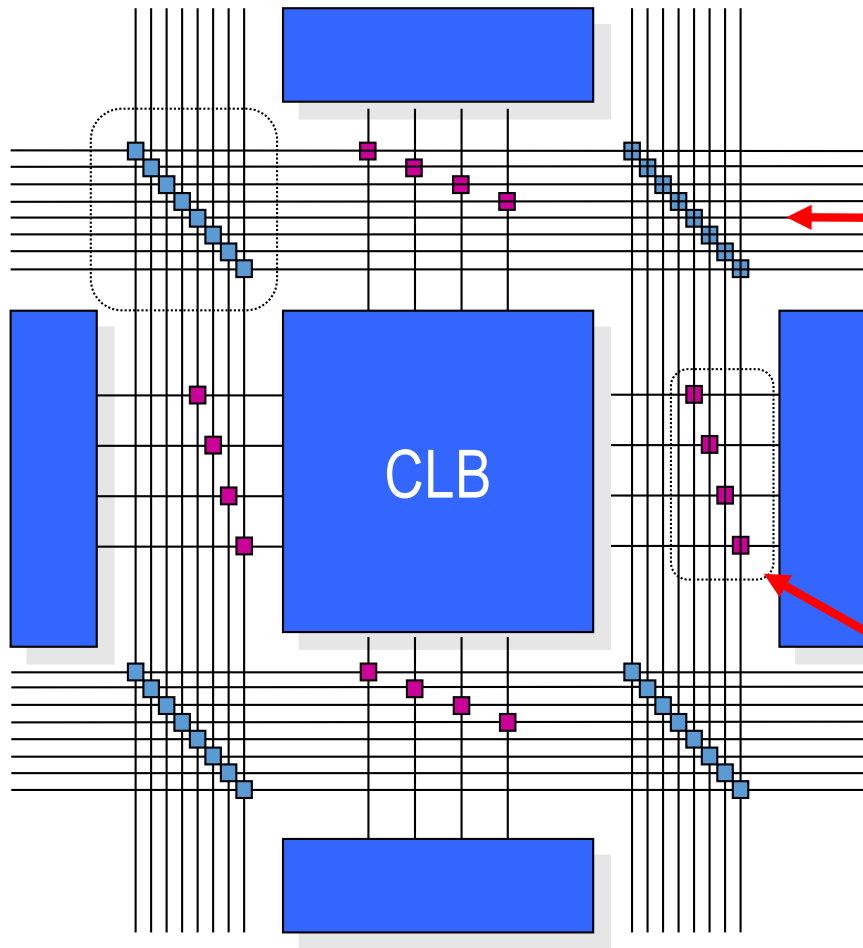
x0	x1	x2	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



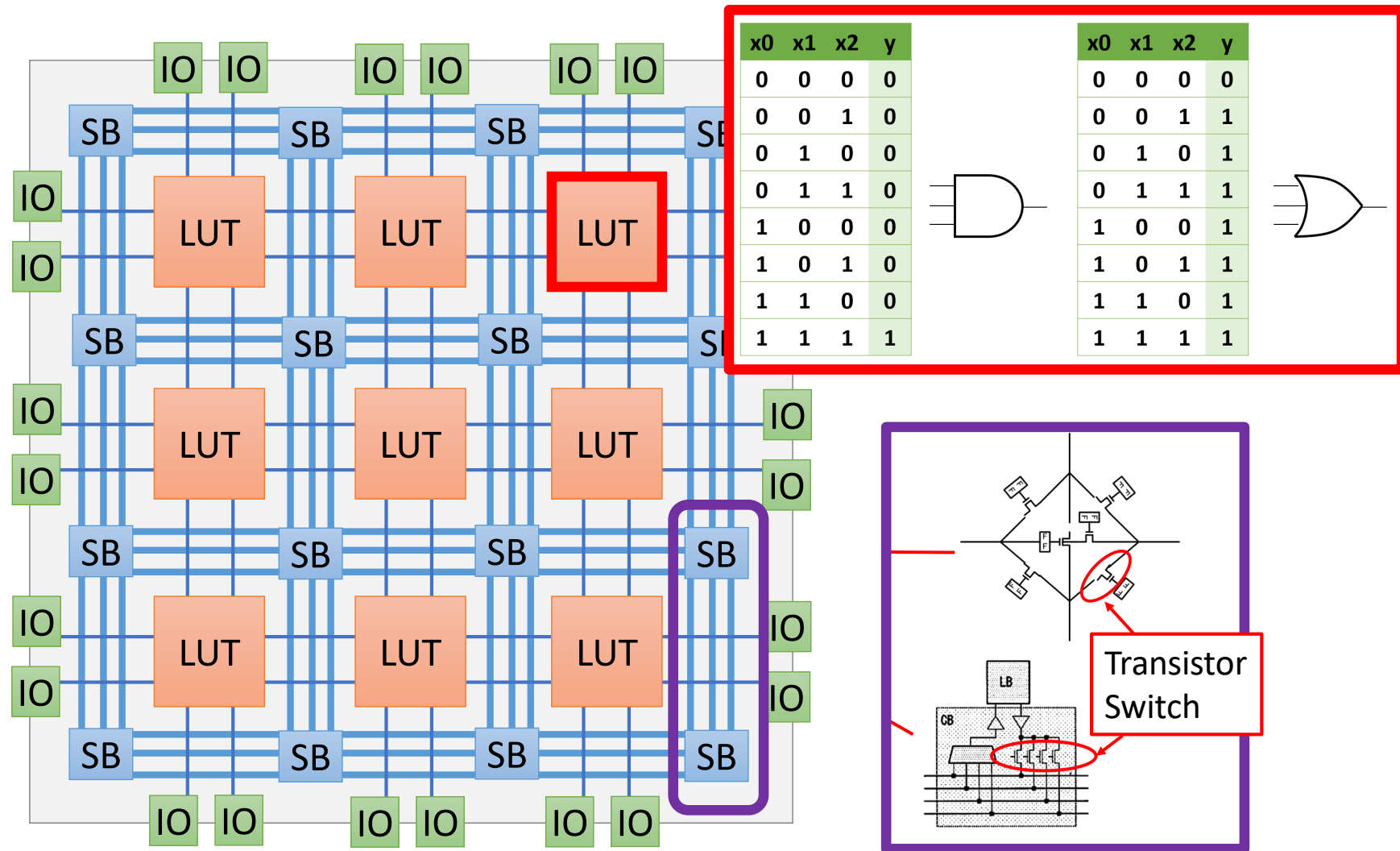
x0	x1	x2	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



Channel and Switch Block



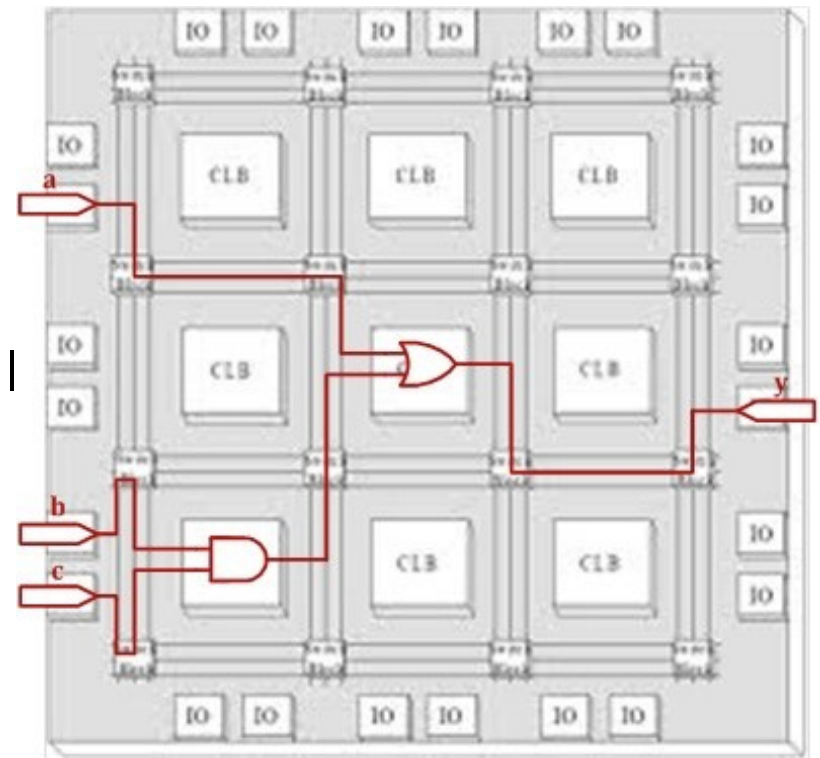
Memory-based realizes “programmable”



Realization of Logic Network

```
143 input  ap_clk;
144 input  ap_rst;
145 input  ap_start;
146 output ap_done;
147 output ap_idle;
148 output ap_ready;
149 input  [0:0] reset;
150 output [15:0] bmp_address0;
151 output  bmp_ce0;
152 output  bmp_we0;
153 output [11:0] bmp_d0;
154 input  [7:0] pad0;
155 input  [7:0] pad1;
156 output [31:0] ap_return;
157
158 reg ap_done;
159 reg ap_idle;
160 reg ap_ready;
161 reg bmp_ce0;
162 reg bmp_we0;
163 (* fsm_encoding = "none" *) reg [43:0] ap_CS_fsm = 44'b1;
164 reg  ap_sig_cseq_ST_st1_fsm_0;
165 reg  ap_sig_bdd_60;
166 reg [31:0] frame_num = 32'b00000000000000000000000000000000;
167 reg [11:0] ram_address0;
168 reg  ram_ce0;
169 reg  ram_we0;
170 reg [7:0] ram_d0;
171 wire [7:0] ram_q0;
172 reg [7:0] sprram_address0;
173 reg  sprram_ce0;
174 reg  sprram_we0;
175 reg [7:0] sprram_d0;
176 wire [7:0] sprram_q0;
177 reg [7:0] sprram_address1;
```

CAD Tool
Xilinx: Vivado
Intel: Quartus II

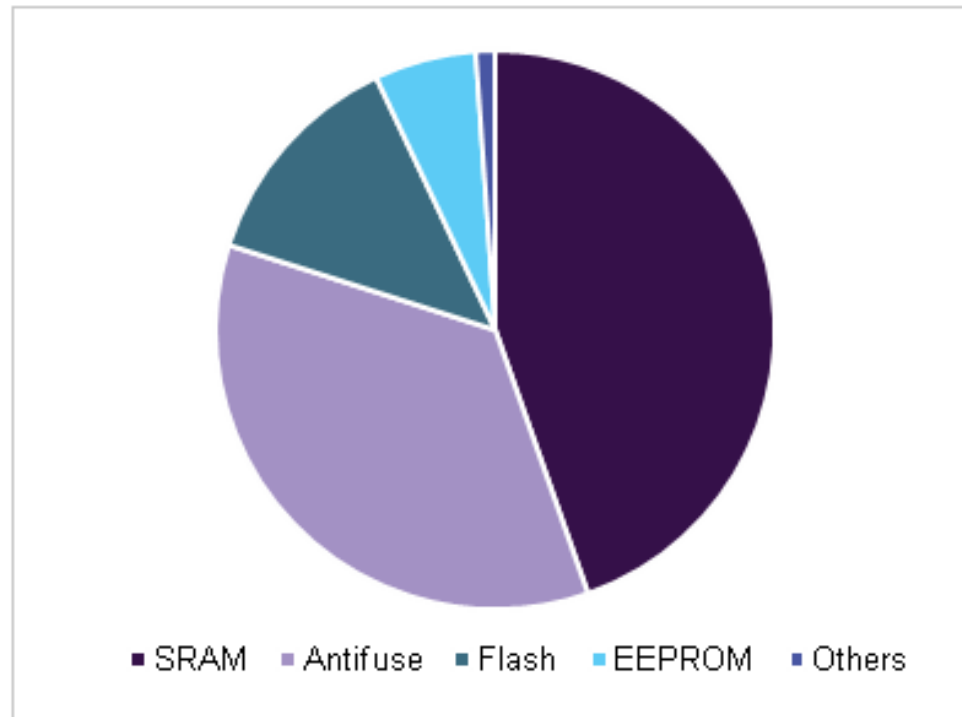


Describe a logic by
hardware description language
(VHDL/Verilog-HDL)

Product Type Segments

- SRAM
- Flash based
- Antifuse

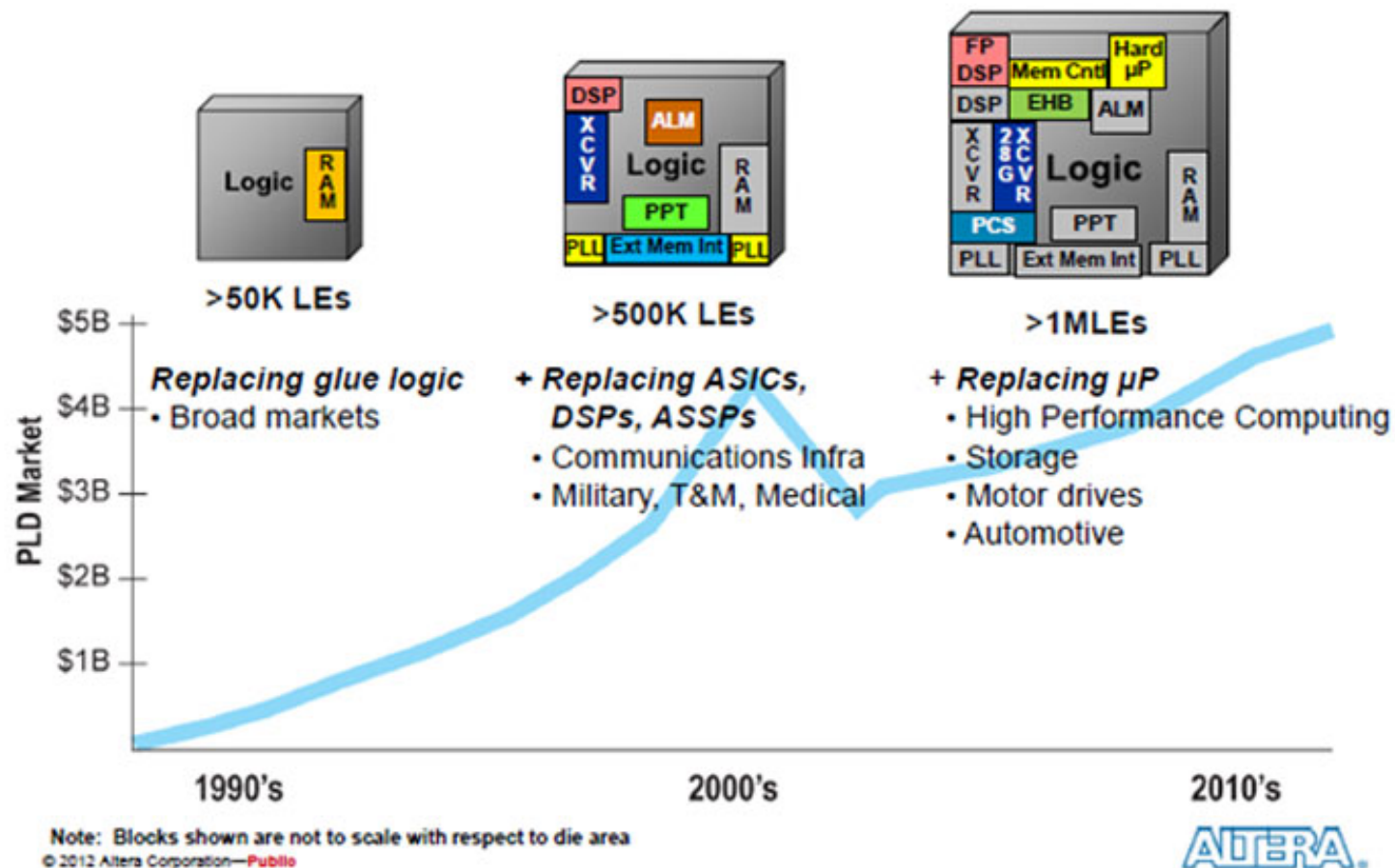
Global FPGA market share, by technology, 2015 (USD Million)



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

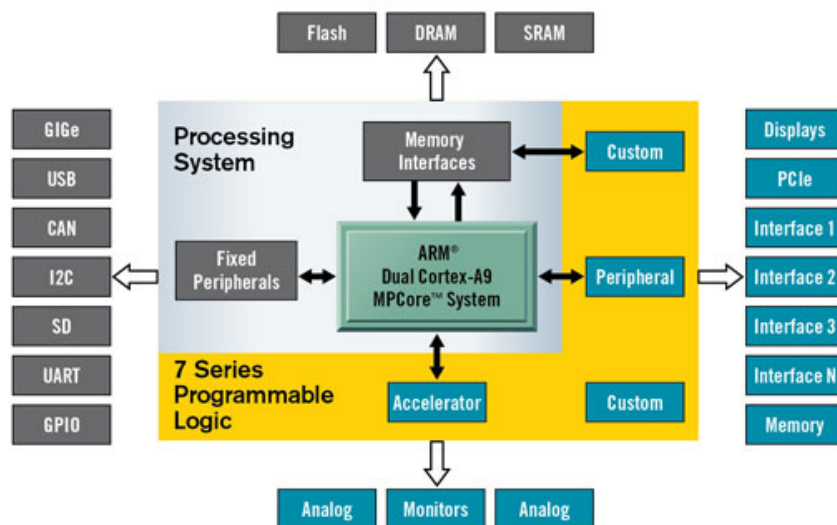
FPGA Growth Trend

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

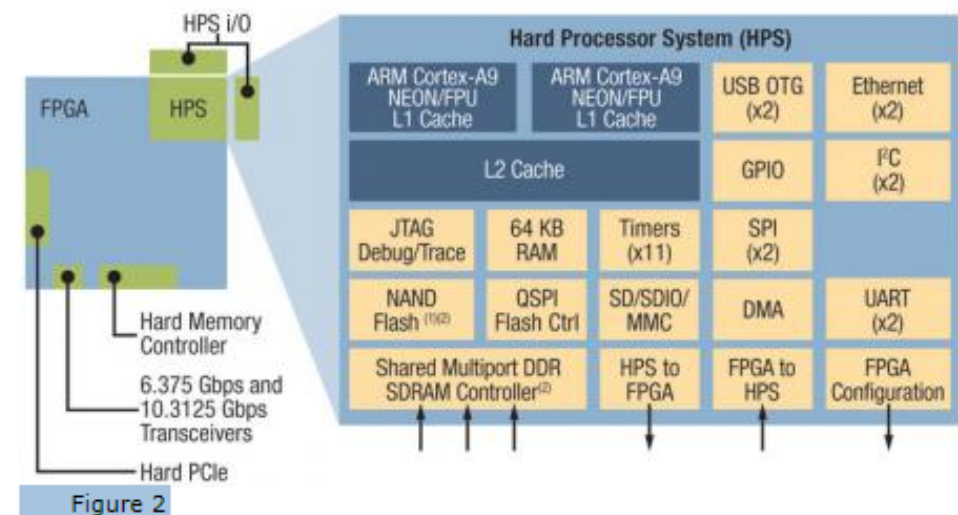


FPGA Mixed with GPUs: The Era of the Programmable SoC

- A generic example of an SoC FPGA, sometimes also known as an application services platform (ASP), shows a dual-core hard processor system with its complement of hard peripherals on the same die with an FPGA fabric



Xilinx ZYNQ Family

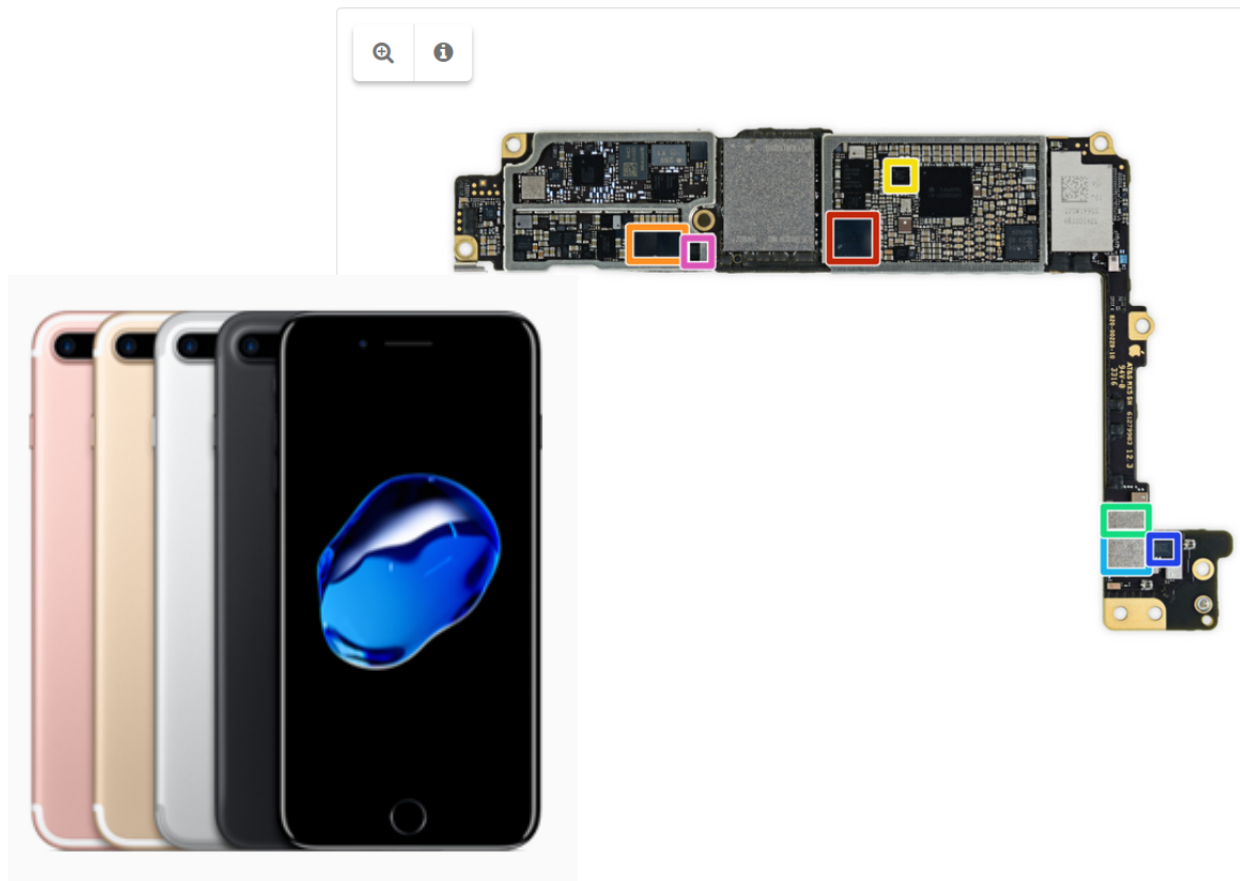


Intel SoC Series

Application Type Segments

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

iPhone7 Plus

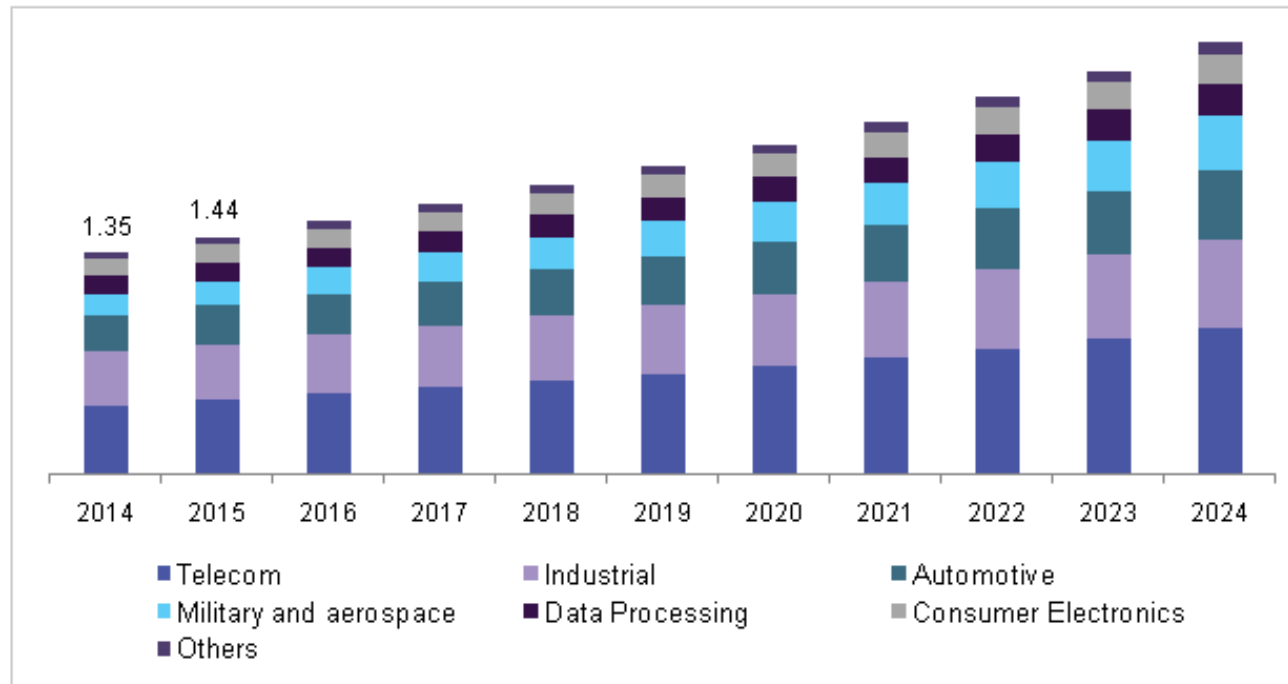


- But wait, there are even more ICs on the back!
- Apple/Cirrus Logic 338S00105 Audio Codec
- Cirrus Logic 338S00220 Audio Amplifier(x2)
- Lattice Semiconductor ICE5LP4K
- Skyworks 13702-20 Diversity Receive Module
- Skyworks 13703-21 Diversity Receive Module
- Avago LFI630 183439
- NXP 610A38

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

Market by Application

U.S. FPGA Market by application, 2014 - 2024 (USD Billion)



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?

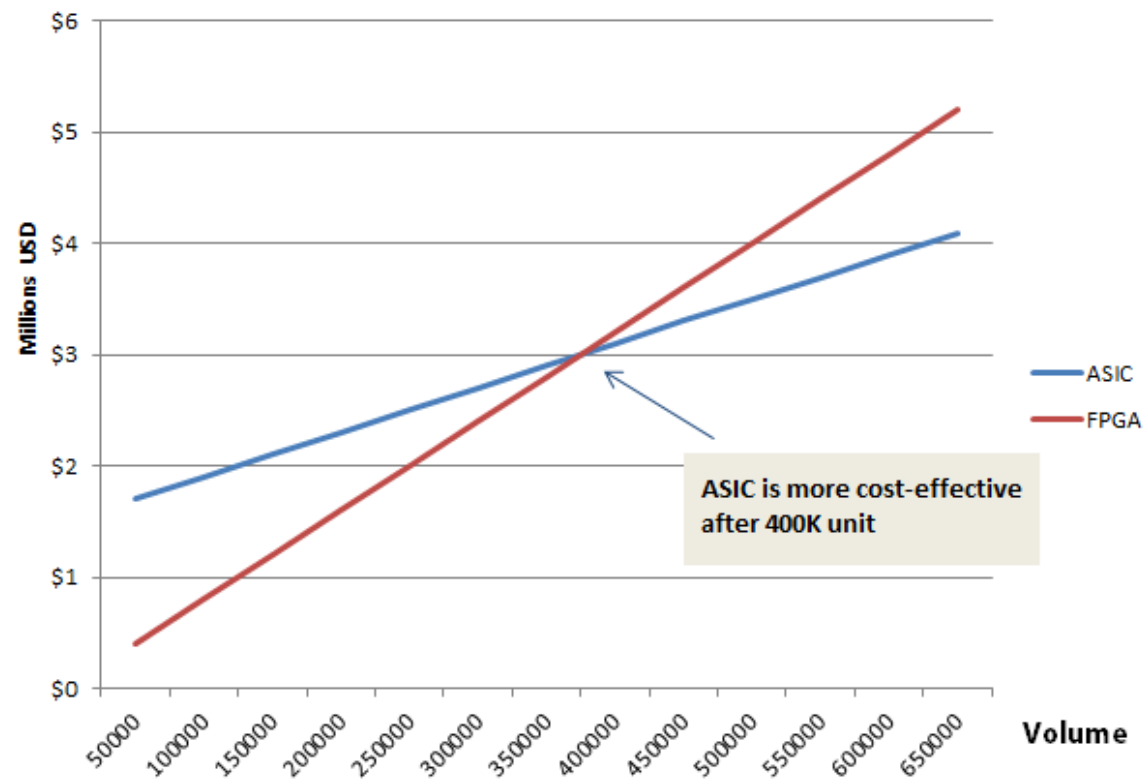
	FPGA	ASIC
Time to Market	Fast	Slow
NRE	Low	High
Design Flow	Simple	Complex
Unit Cost	High	Low
Performance	Medium	High
Power Consumption	High	Low
Unit Size	Medium	Low

AnySilicon.com

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

Price Comparison

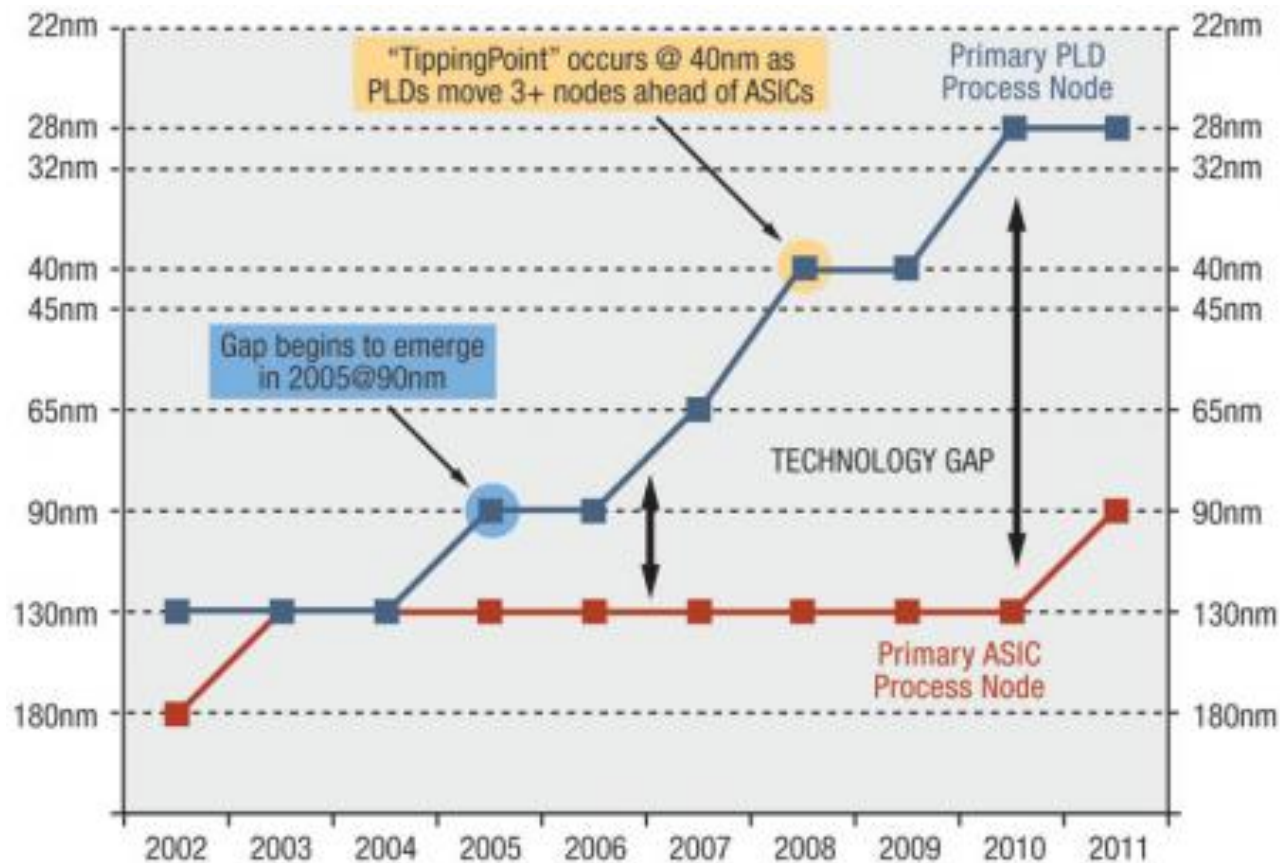
- ASIC→NRE: \$1.5M, Unit cost: \$4
- FPGA→NRE: \$0, Unit cost: \$8



Total Cost ASIC vs FPGA including NRE in MUSD

Performance Comparison

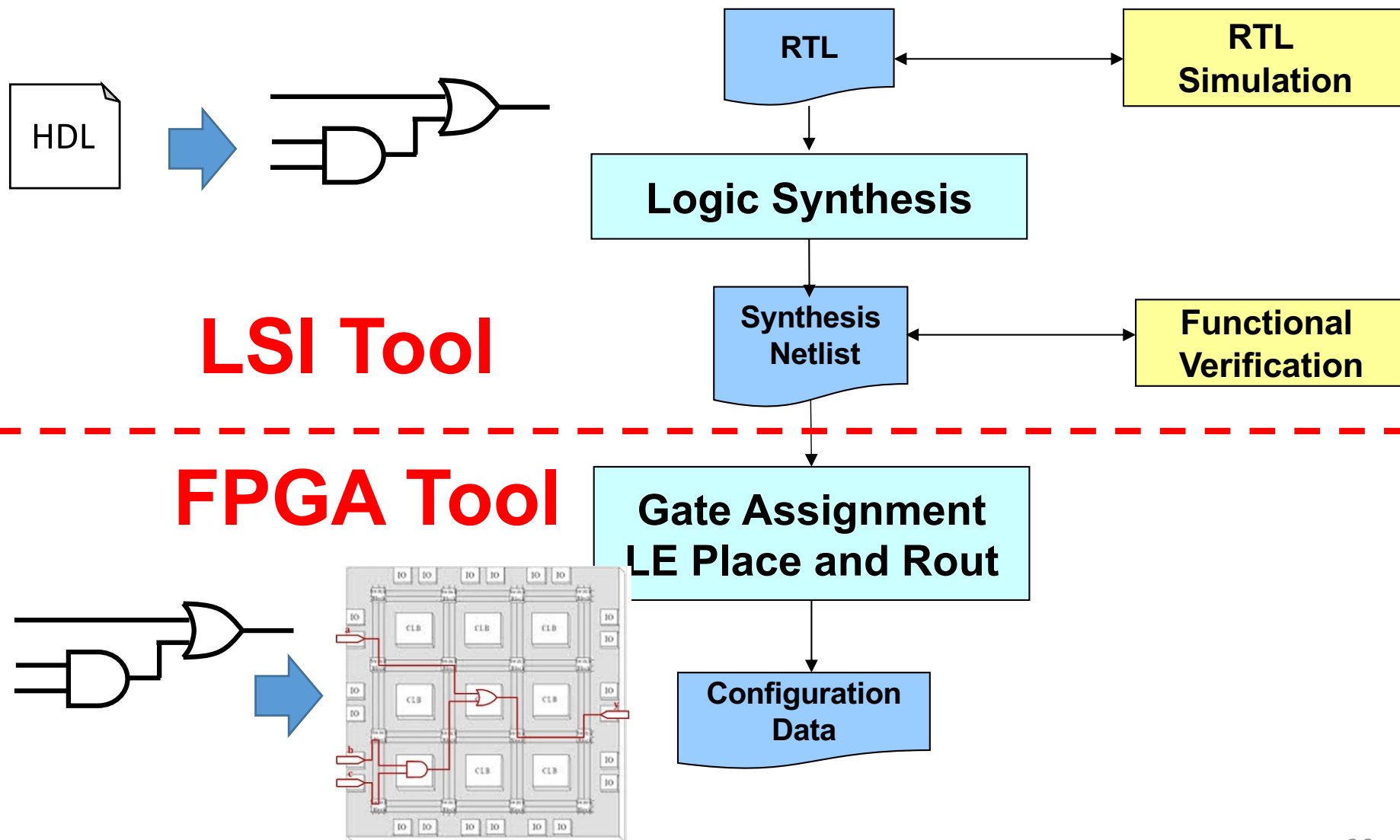
- The divergence between primary programmable logic device technology and that used for ASICs has continued to grow



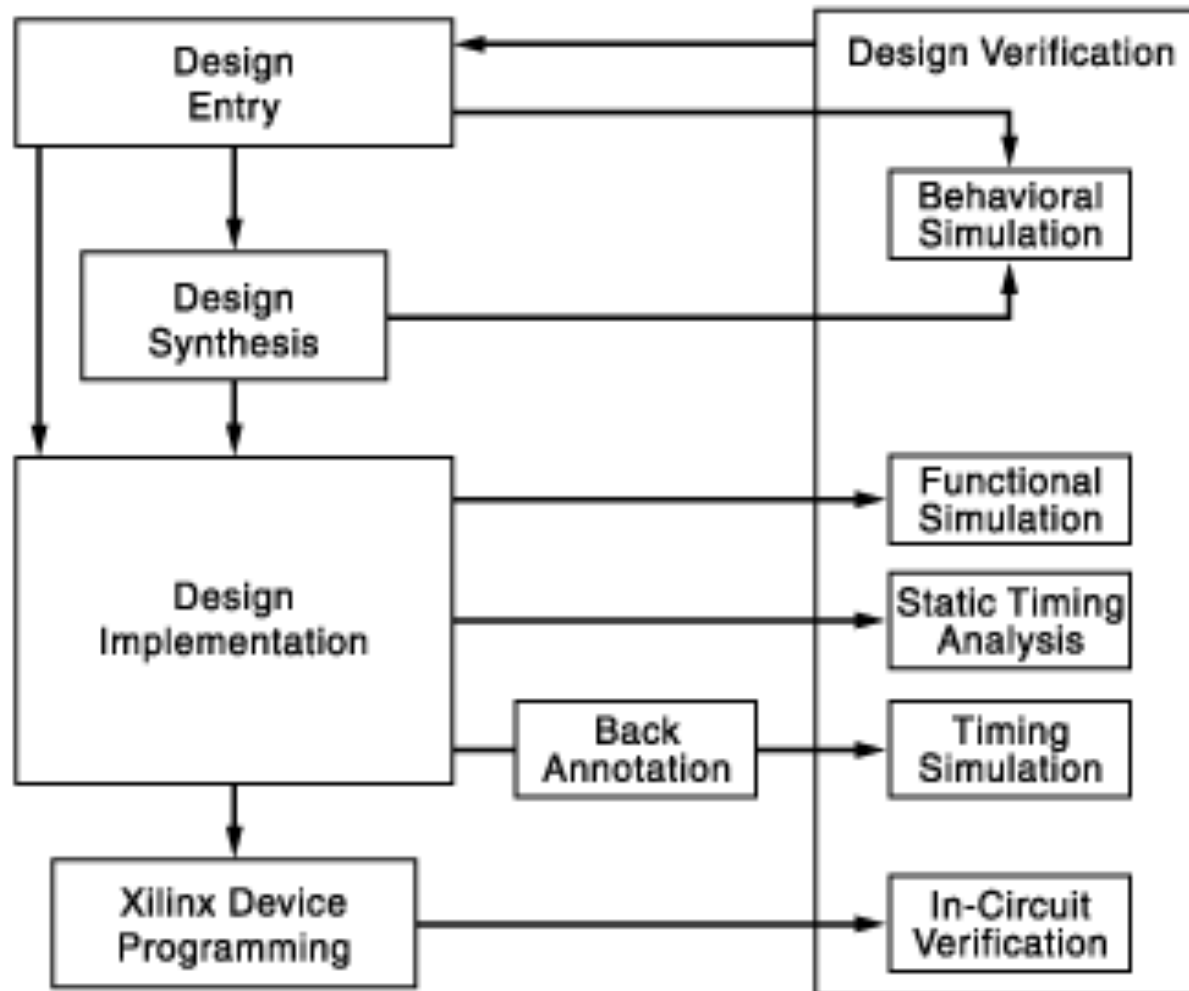
Source: <http://archive.rtcmagazine.com/articles/view/102503>

FPGA Programming

Standard FPGA Design

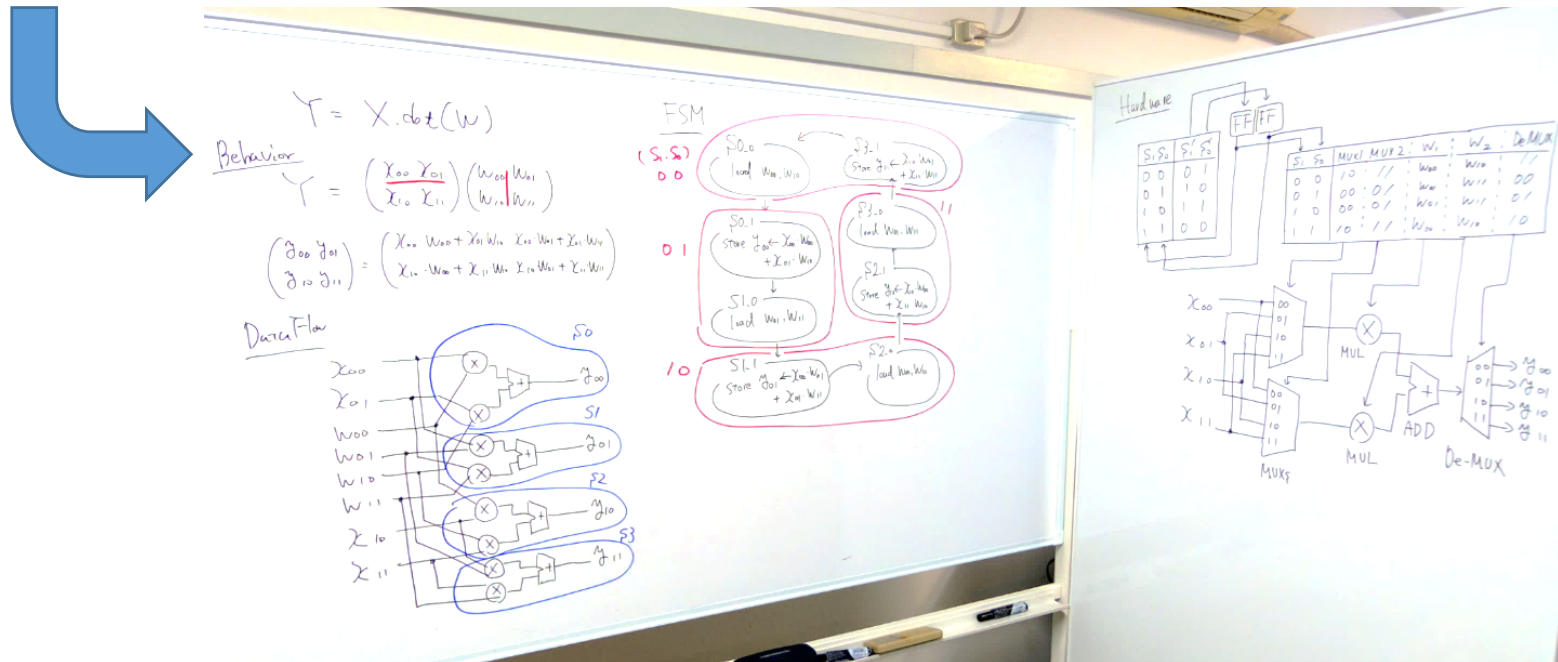


FPGA Design Flow



How to write a HDL?

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



Then, you can write your HDL!

Comparison of # Lines

$$Y = X \cdot W + B$$

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;

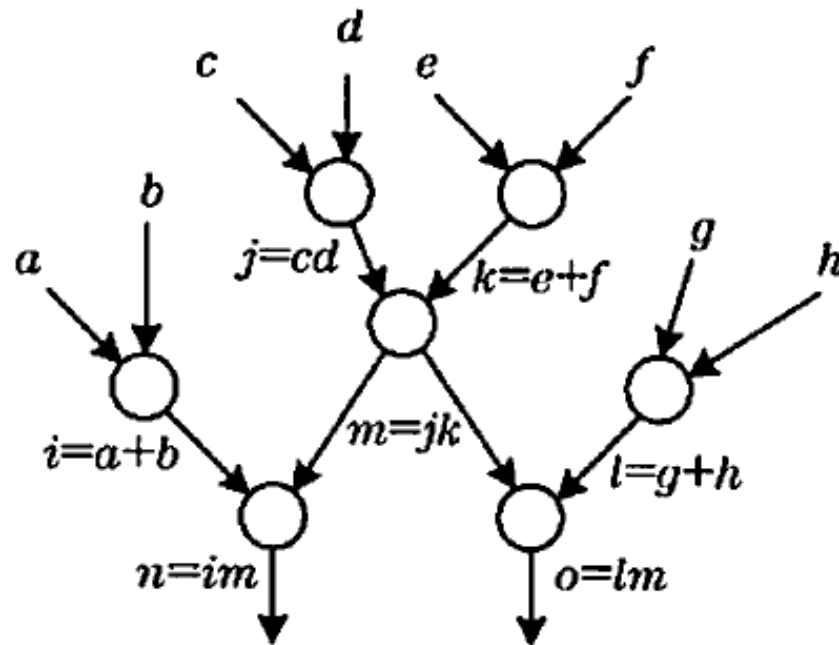
assign y[0] = (de_mux == 2'b00) ? w_add : 2'bzz;
assign y[1] = (de_mux == 2'b01) ? w_add : 2'bzz;
assign y[2] = (de_mux == 2'b10) ? w_add : 2'bzz;
assign y[3] = (de_mux == 2'b11) ? w_add : 2'bzz;

endmodule
```

Verilog-HDL: 66 lines

Boolean Network

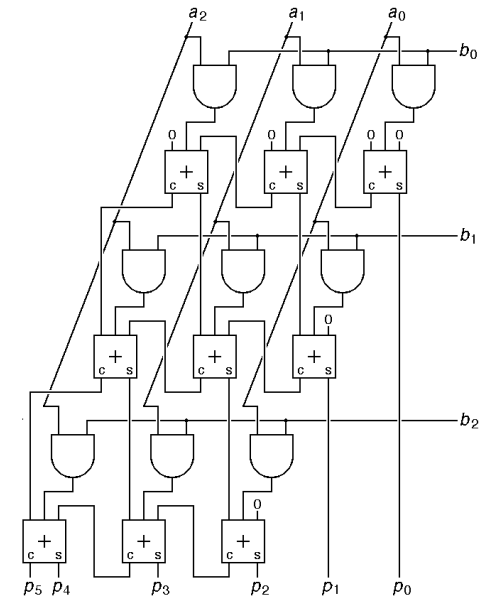
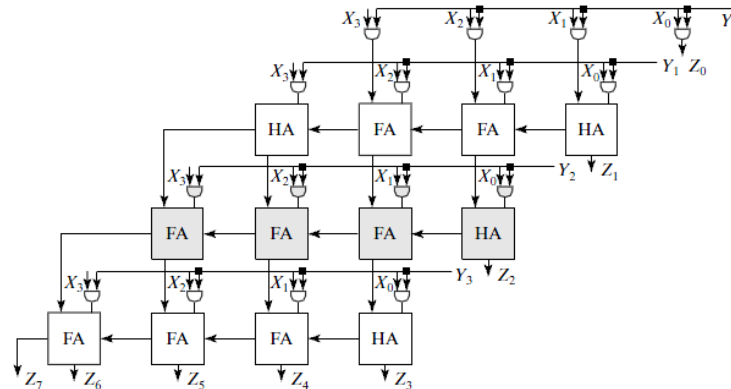
- Representation of a combinational logic circuit using a directed graph without a cycle
- Vertex : Logic gate, Edge : Input or output



Logic Synthesis

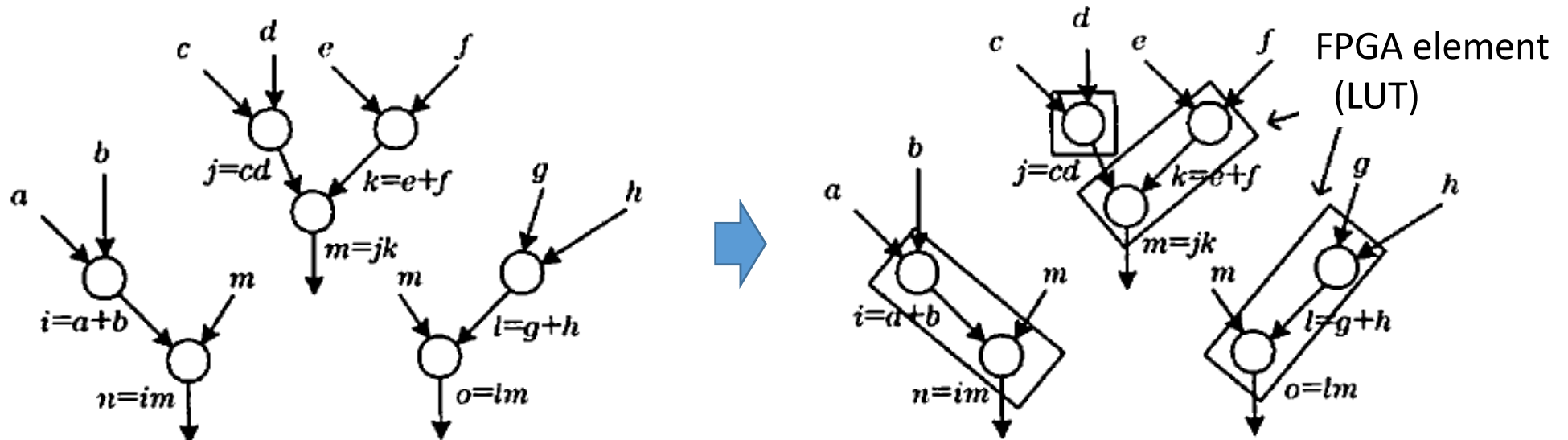
- Synthesize from a given HDL specification to a Boolean network

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



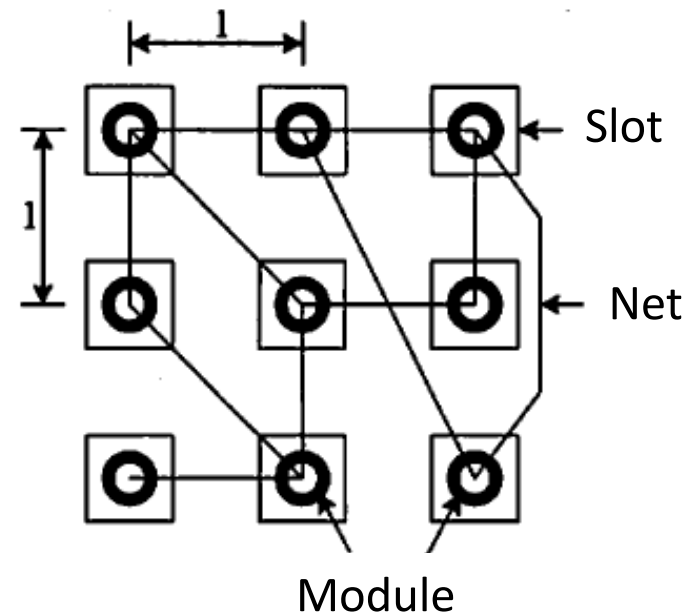
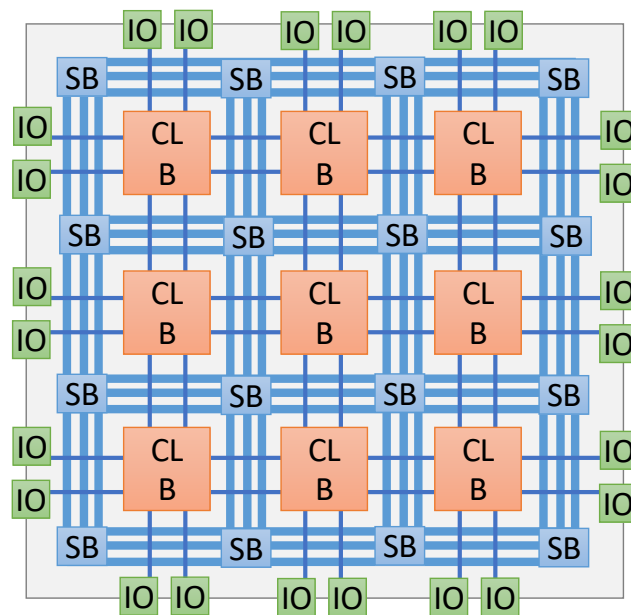
Technology Mapping

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



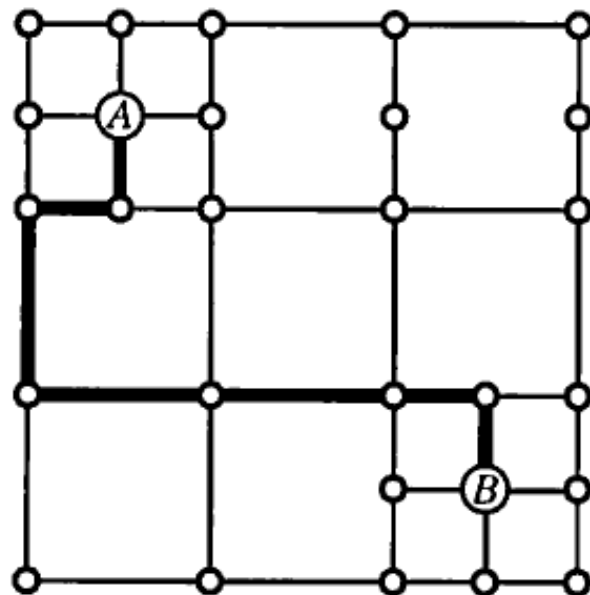
Placement

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

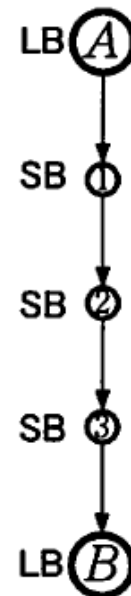


Routing

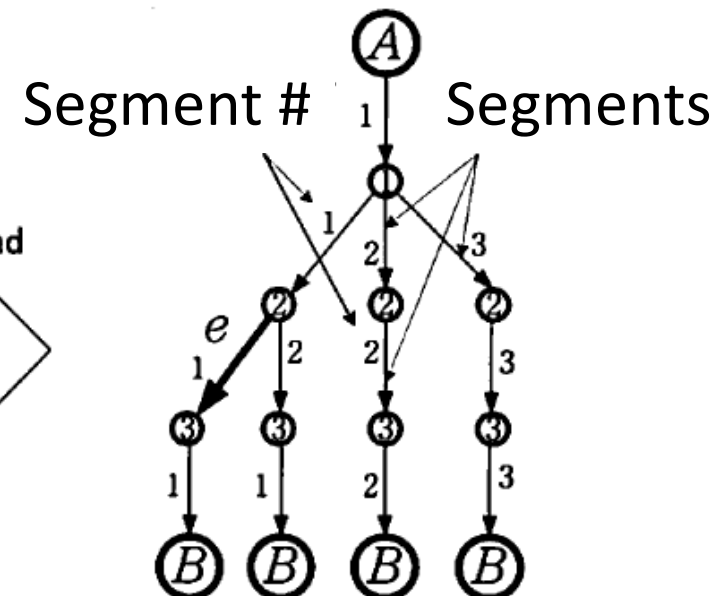
- Global routing: Determine the rough wiring path
- Local one: Determine the wiring segment and switch



Global routing



Expand
➡



Local routing

Pressor of Design Time

- Design Time = #lines \propto \$
- More higher-level description
 - High-level synthesis for C/C++

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

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
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
```

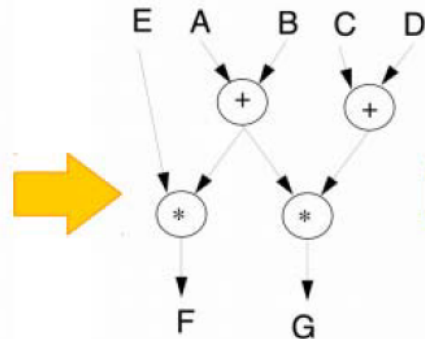
Verilog-HDL: 66 lines

High-Level Synthesis (HLS)

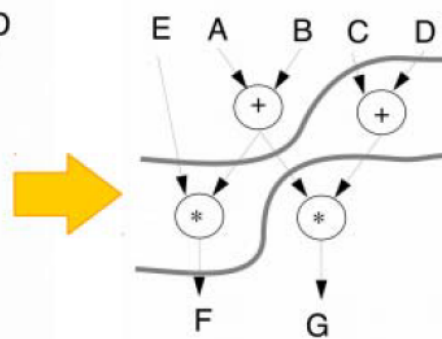
```

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;
    
```

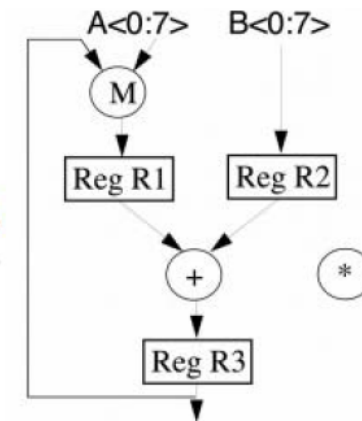
Input Behavioral Spec.



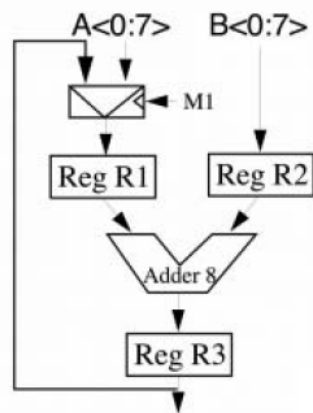
Dataflow



Scheduling



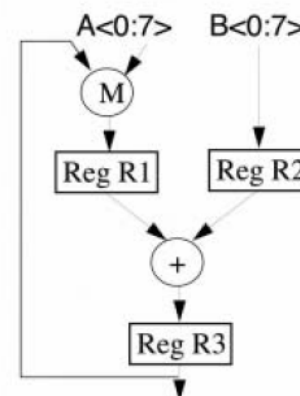
Data-path generation



Mapping to resources
(Binding)

Controller ROM:

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



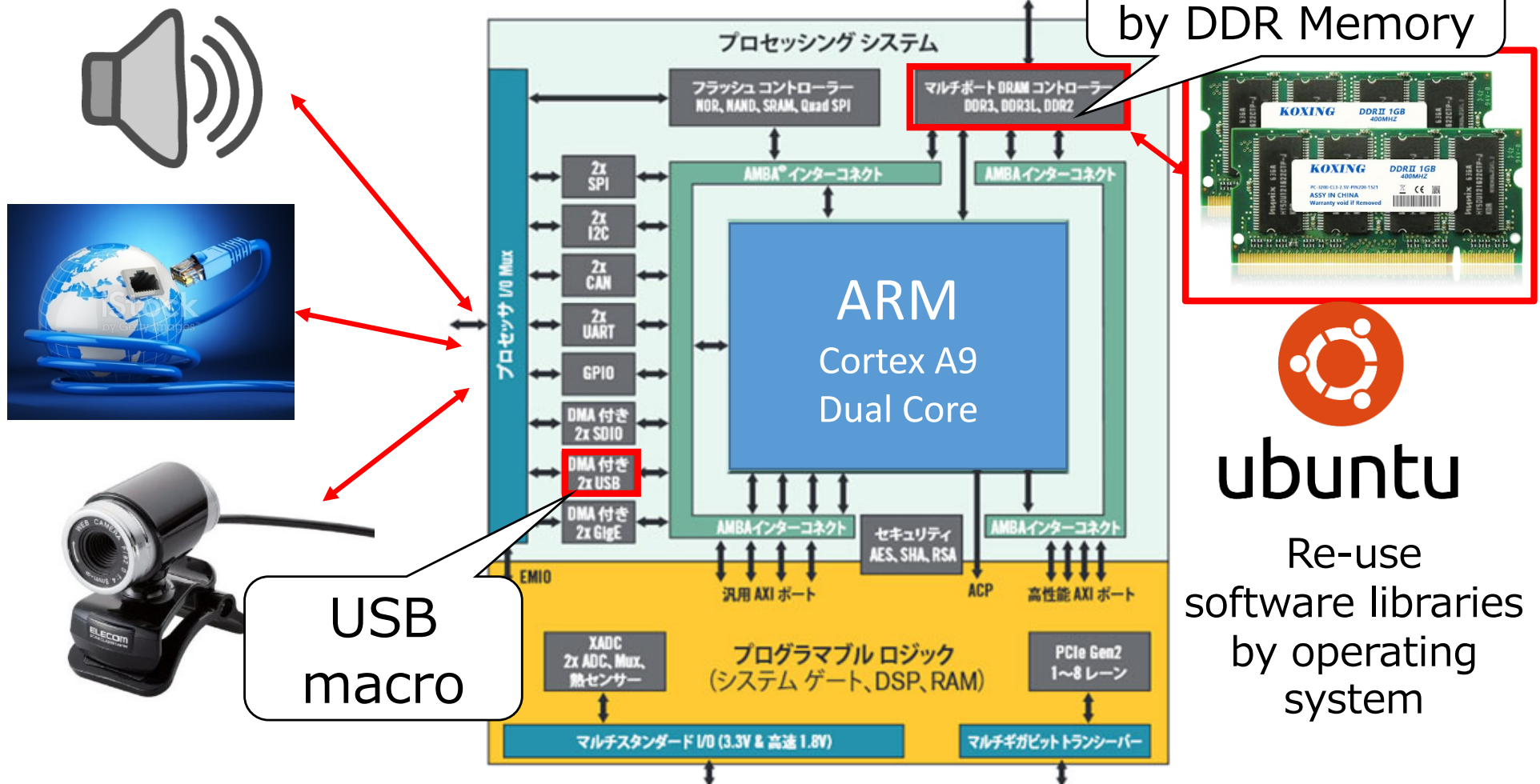
Controller (FSM) Generation

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...

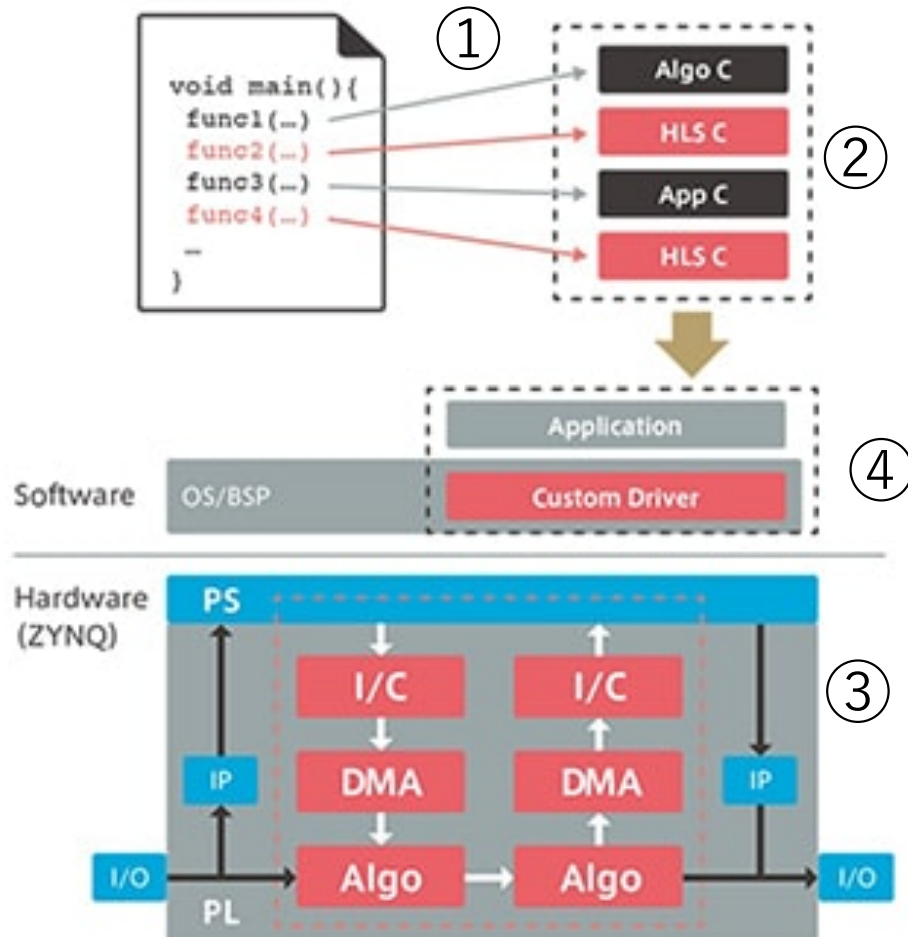
System on Chip FPGA

- Xilinx: Zynq, Intel: Intel SoC



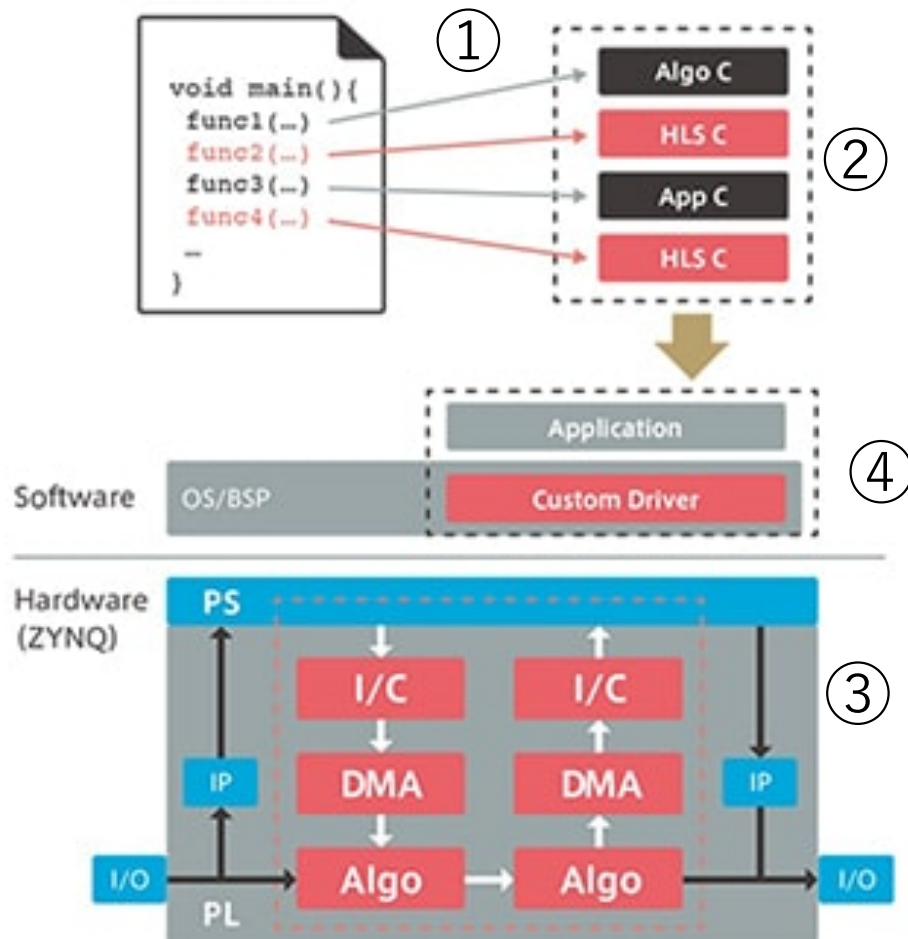
Source: Xilinx Inc. Zynq-7000 All Programmable SoC

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

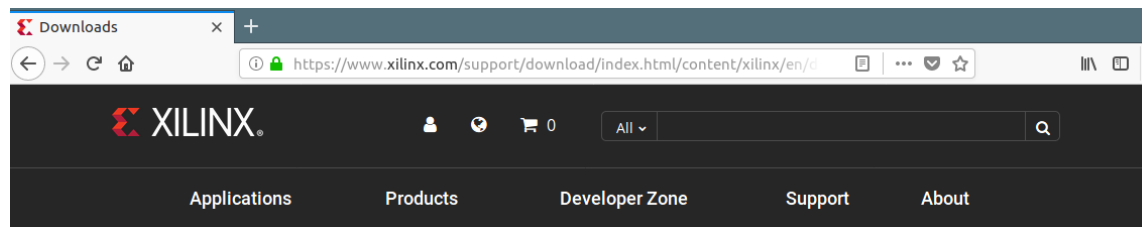
Summary

- FPGA: Reconfigurable LSI or Programmable Hardware
 - Consists of a programmable logic array and a programmable interconnection
 - Programmable (Memory-based) switch
- Standard FPGA design supports an RTL based one
 - Shifting to High-level (C/C++) design
- Benefits: Productivity, lower non-recurring engineering costs, maintainability, faster time to market

Exercise:

Install Vivado and SDK

Setup Your FPGA Development Environment for Ubuntu 16.04 LTS



Downloads

Installation Overview Video

Doc Navigator Video (5:28)

Licensing Help

Vivado

Embedded Development

SDSoC Development Environment

SDAccel Development Environment

Version

2018.1

2017.4

Archive

Vivado General Information - 2017.4

Important Information

Vivado 2017.3 and later versions require upgrading your license server tools to the Flex 11.14.1 versions listed below. Please note that Vivado 2017.3 is the last release that will support Solaris operating system. Xilinx will continue to support Windows and Linux operating systems.

Vivado Design Suite - HLx Editions: Unda

**Select 2017.4 version !!
(current version 2020.1)**

Vivado Design Suite - HLx Editions - 2017.4 Full Product Installation

Important

We strongly recommend to use the web installers as it reduces download time and saves significant disk space.

Please see [Installer Information](#) for details.

[Vivado HLx 2017.4: WebPACK and Editions - Windows Self Extracting Web Installer \(EXE - 51.3 MB\)](#)

MD5 SUM Value : 39677d35779915411487e5d89fe27ee8

[Vivado HLx 2017.4: WebPACK and Editions - Linux Self Extracting Web Installer \(BIN - 100.7 MB\)](#)

MD5 SUM Value : 915928e0f33af22f1370acc6ac32c2b5

[Vivado HLx 2017.4: All OS Installer Single-File](#)

Download Includes

Vivado Design Suite HLx Editions (All Editions)

Download Type

Full Product Installation

Last Updated

Dec 20, 2017

Answers

[2017.x - Vivado](#)

[Known Issues](#)

[2017.4 - Release](#)

[Notes](#)

[License Solution](#)

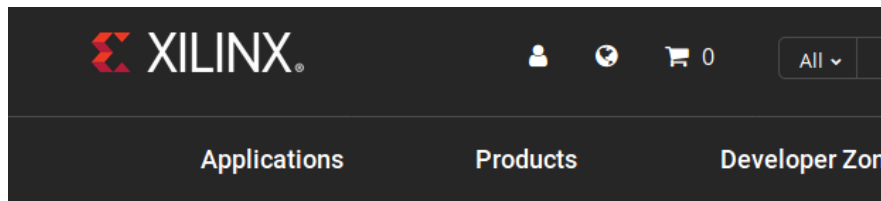
[Center](#)

Enablement

**For windows 7 or 10 user
(64-bit version)**

**CentOS 6.6-6.9, 7.2-7.3
Ubuntu 16.04 LTS
(64-bit version)**

Make Your Xilinx Account



[Xilinx - Adaptive. Intelligent.](#) > [Sign In](#)

Sign In

Username *

Password *

Forgot your [username](#) or [password](#)?

Sign In

New to Xilinx? [Create your account](#)

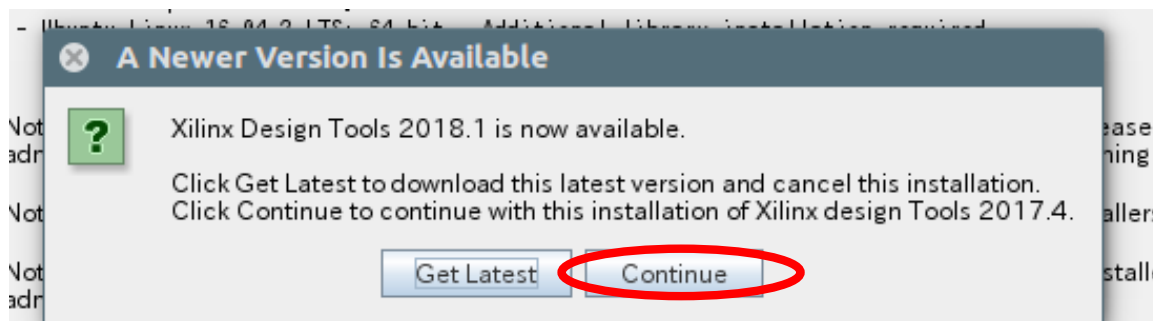
By signing in, you agree to the Xilinx [Terms of Use](#) and [Privacy Policy](#).

```
$cd /Download
```

```
$ chmod a+x Xilinx_Vivado_SDK_Web_2017.4_1216_1_Lin64.bin
```

```
$ sudo su (Change root user)
```

```
# ./Xilinx_Vivado_SDK_Web_2017.4_1216_1_Lin64.bin
```

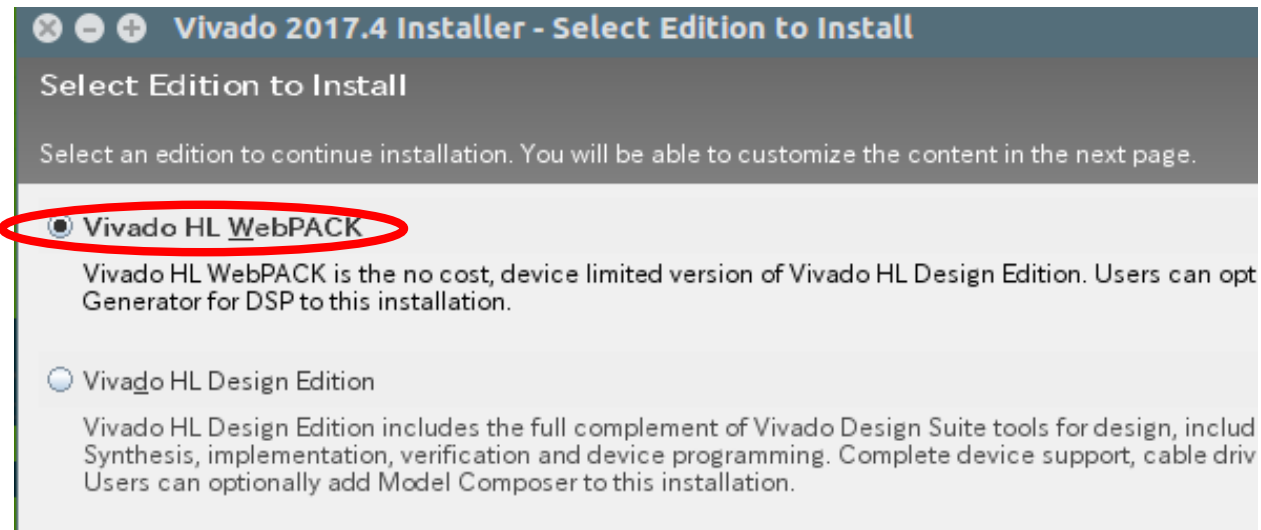


To reduce installation time, we recommend that you disable any anti-virus software before continuing.

Select "Continue" at "A Newer Version is Available" Dialog, then click "Next".

At "Select Install Type" window, enter your User ID and Password, and choose "Download and Install Now", then click "Next".

Next, check all "I Agree" for accept license agreements, then "Next".



Choose Vivado HL WebPACK version, which is a free version.

In WebPACK installation, you are not necessary customize "Design Tools", "Devices", and "Installation Options". Just click "Next".

Set your installation directory to "/opt/Xilinx".

Wait 3-4 hours...

Run Vivado

`$sudo su (change root user)`

`#source /opt/Xilinx/2017.4/Vivado/settings64.sh`

`#vivado &`

Run Xilinx SDK (Software Development Kit)

```
$sudo su (change root user)
```

```
#source /opt/Xilinx/2017.4/Vivado/settings64.sh
```

```
#xsdk &
```

Run Vivado HLS

```
$sudo su (change root user)
```

```
#source /opt/Xilinx/2017.4/Vivado/settings64.sh
```

```
#vivado_hls &
```


Exercise 1

0. I will send you an FPGA board. Please tell me your mailing address!

1. (Mandatory) Install Vivado HLx edition to your PC, and send e-mail screen-shot for startup windows (VIVADO, VIVADO_HLS, SDK). Send e-mail with PDF including screen shots.

If you meet any troubles, don't hesitate to contact me.

nakahara@ict.e.titech.ac.jp

2. (Mandatory) Why an FPGA can be applied to the high-end CMOS process?
3. (Mandatory) Why RTL based design is necessary to FPGA implementation?
4. (Mandatory) Investigate FPGA market for the past 10 years.

Deadline is 26th, June, 2020, JST PM 13:20 (At the beginning of the next lecture)