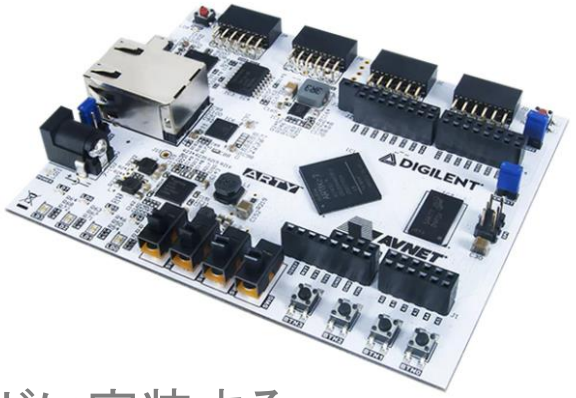
Department of Computer Science
Course number: CSC.T341

コンピュータ論理設計
Computer Logic Design

5. VLSIとリコンフィギャラブルシステム
VLSI and Reconfigurable Systems

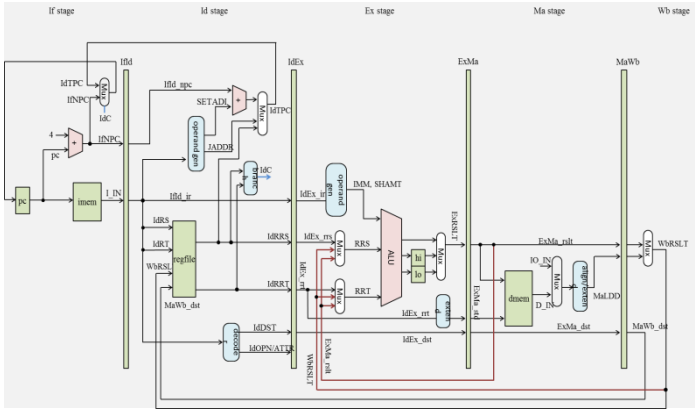
吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp　www.arch.cs.titech.ac.jp/lecture/CLD/
講義：月曜日 10:45-12:25, 木曜日 10:45-12:25

コンピュータ論理設計の特徴



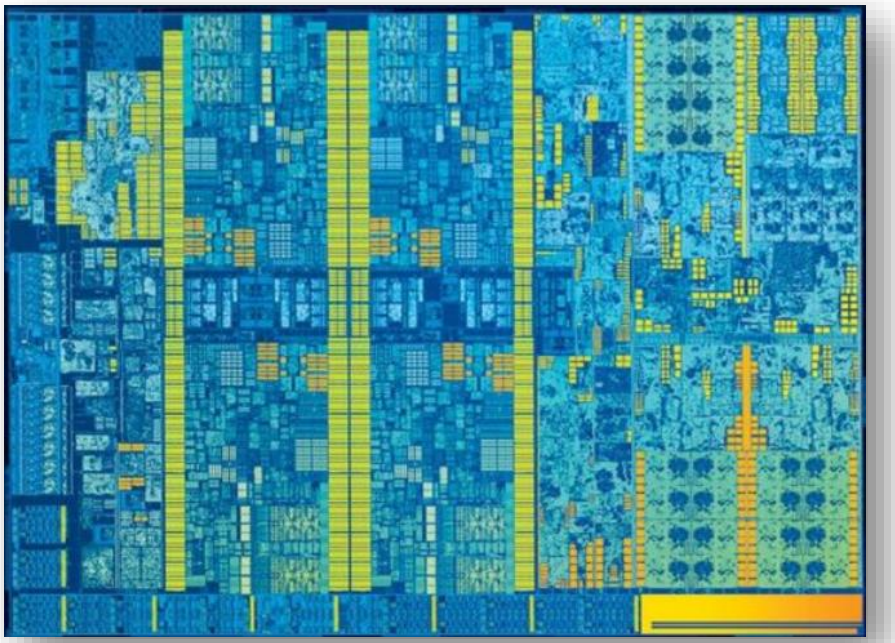
- 講義2単位，演習1単位．
- 1人1台のFPGA (Field-Programmable Gate Array) ボードを用いた演習．
- 4人程度を1グループとした共同作業と問題解決．
- 教科書で説明されるプロセッサのRISC-V版を ハードウェア記述言語Verilog HDLで記述し，FPGAボードに実装する．
- グループとしてプロセッサの高速化に取り組み，コンテスト形式で成果を競う．
- 3Q開講のコンピュータアーキテクチャ（CSC.T363）のための準備．

```
module main (clk, led);
  input  wire clk;
  output wire led;

  reg [26:0] cnt=0;
  always @(posedge clk) cnt <= cnt + 1;

  assign led = cnt[26];
endmodule
```

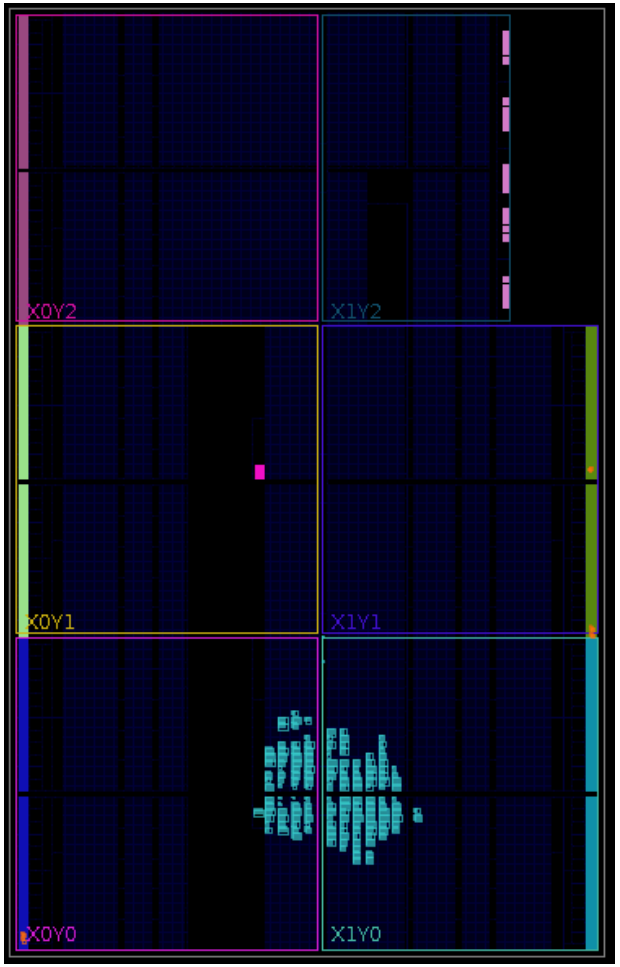


ASIC (Application Specific Integrated Circuit)

- ASIC（特定用途向け集積回路）
  - マイクロプロセッサとしてのASIC
  - FPGAとしてのASIC
- FPGAとしてのASICに実現されるマイクロプロセッサは？



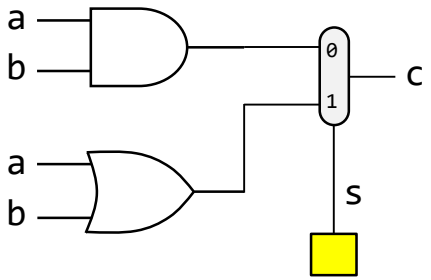
Intel Skylake microprocessor August 2015



Xilinx Artix-7 FPGA on Arty A7-35T

Reconfigurable Systems

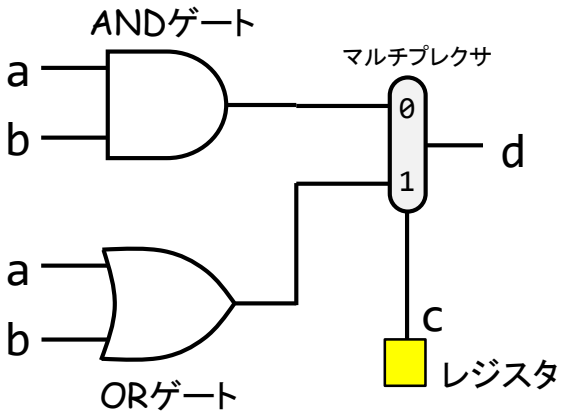
- 再構成可能(リコンフィギャラブル)システム
  - 問題の解法アルゴリズムをハードウェア化してユーザが書き換え可能なデバイス上で直接実行することにより，高い性能と柔軟性を実現するシステム
  - 書き換え可能なデバイスを活用するコンピューティングシステムをアダプティブコンピューティングと呼ぶ.
- FPGA (Field Programmable Gate Array)
  - AMD (Xilinx)
  - Intel (Altera)
  - Lattice
  - NanoBridge

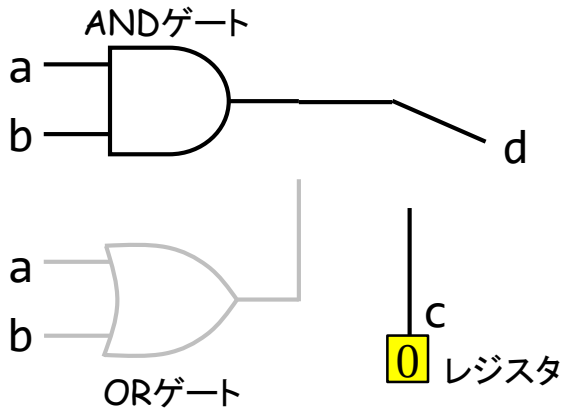


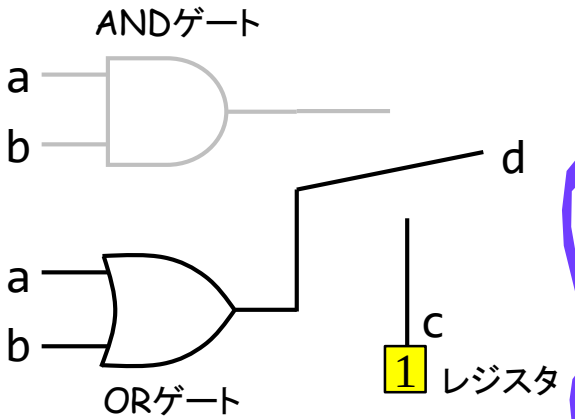
This circuit can change the role of AND or OR gate by the value stored in **s**.
Is this reconfigurable? No!

制御信号cによってANDゲートあるいはORゲートとなる回路

- ANDゲートとORゲートを切り替えることで性能が向上する例を考える.
- レジスタの値で出力を選択する回路
- この例は，再構成可能システムではない！

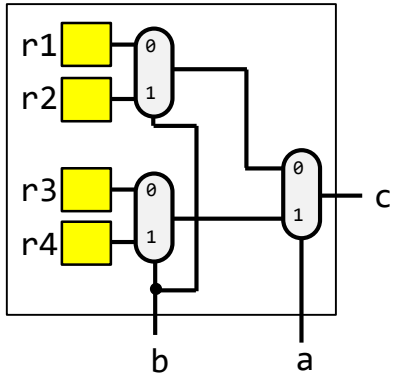




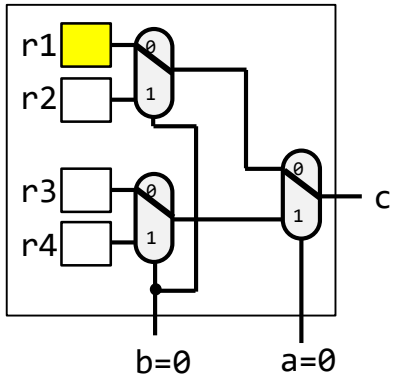


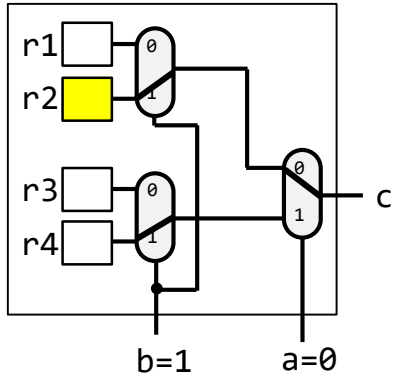
ルックアップテーブル (Lookup Table, LUT)

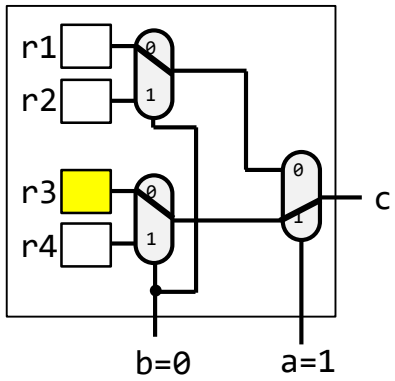
- a, b を入力として，c を出力とする LUT
- 値を保持するレジスタ（黄色）の値を選択する回路

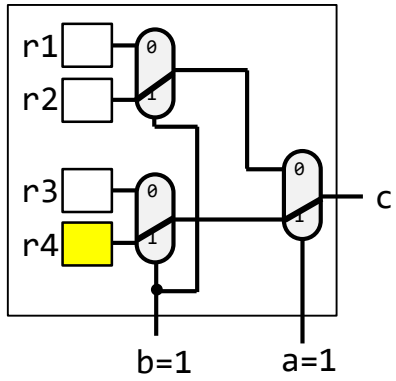


2入力のLUTの構成





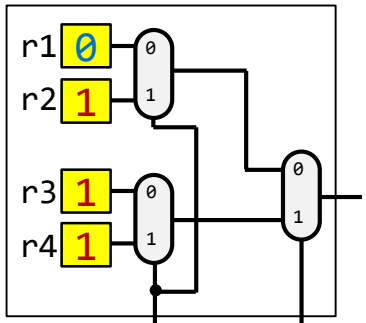




入力 a	入力 b	出力 c
0	0	r1
0	1	r2
1	0	r3
1	1	r4

ルックアップテーブル (Lookup Table, LUT)

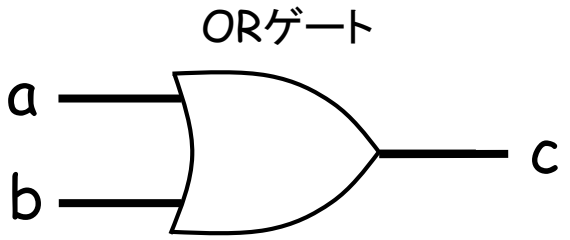
- レジスタの値を上から 0, 1, 1, 1 に設定すると、このLUTはORゲートと同じ動作をする.



2入力のLUT

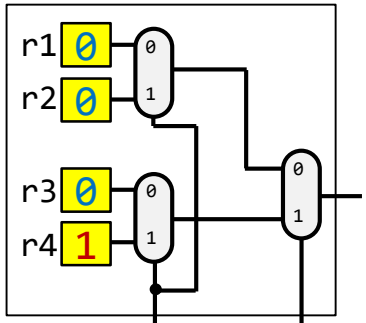
a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

真理値表



左の構成のLUTと等価な回路

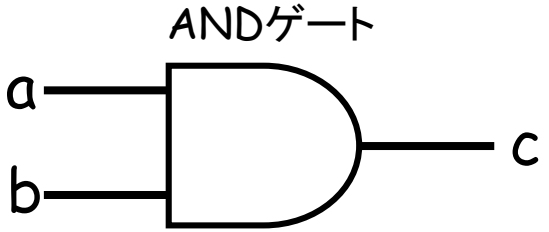
- レジスタの値を上から 0, 0, 0, 1 に設定すると、このLUTはANDゲートと同じ動作をする。



2入力のLUT

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

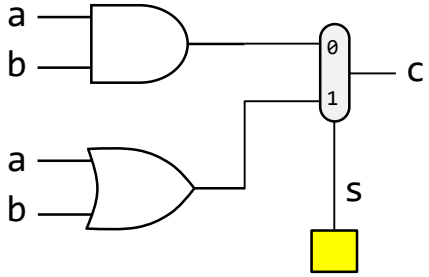
真理値表



左の構成のLUTと等価な回路

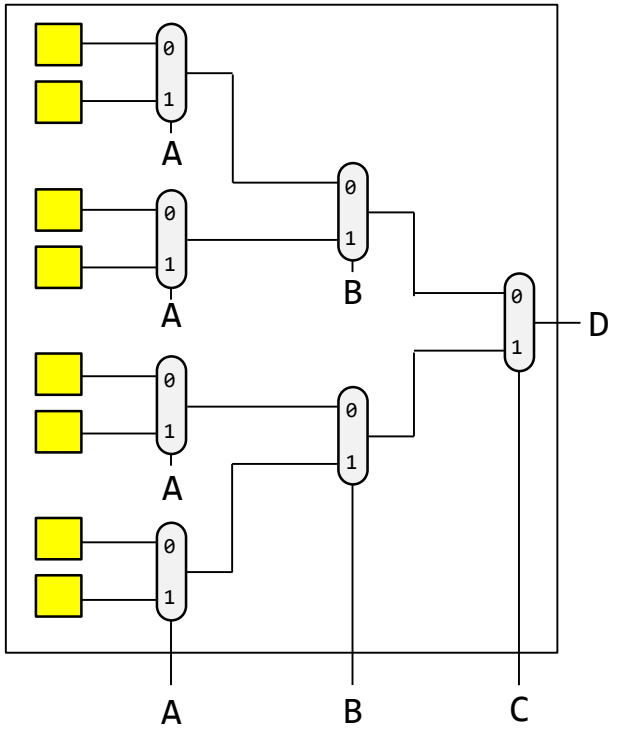
ルックアップテーブル (Lookup Table, LUT)

- Reconfigurable Logic



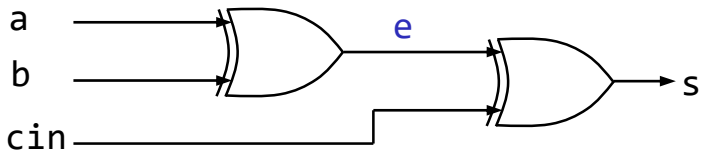
This circuit can change the role
of AND or OR gate by the value stored in **s**.
Is this reconfigurable?
No!

3-input LUT structure



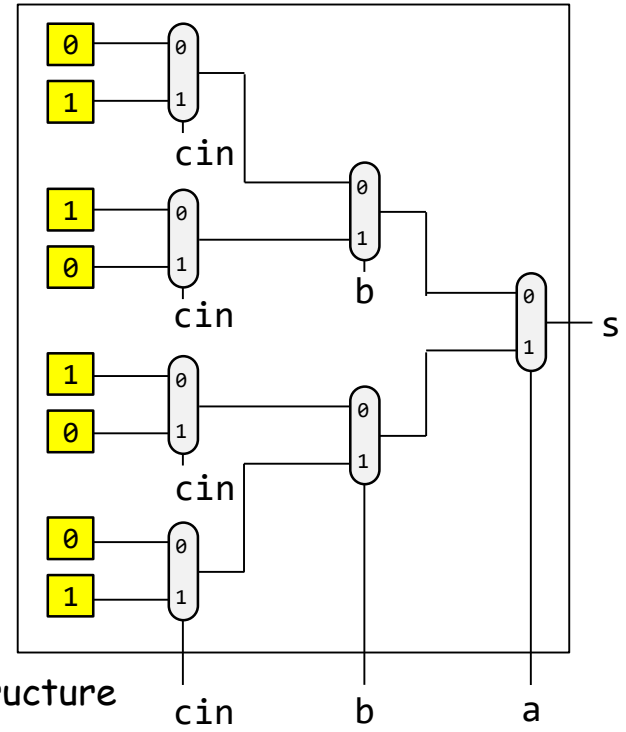
ルックアップテーブル (Lookup Table, LUT)

- Reconfigurable Logic
  - 3-input LUT has 8 configuration registers
  - 6-input LUT for Artix-7 FPGA

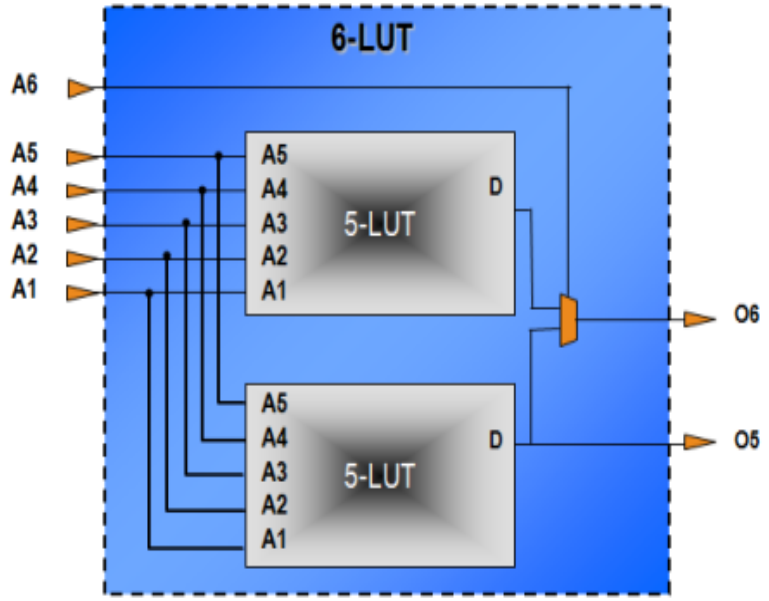


Truth table

a	b	cin	s
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



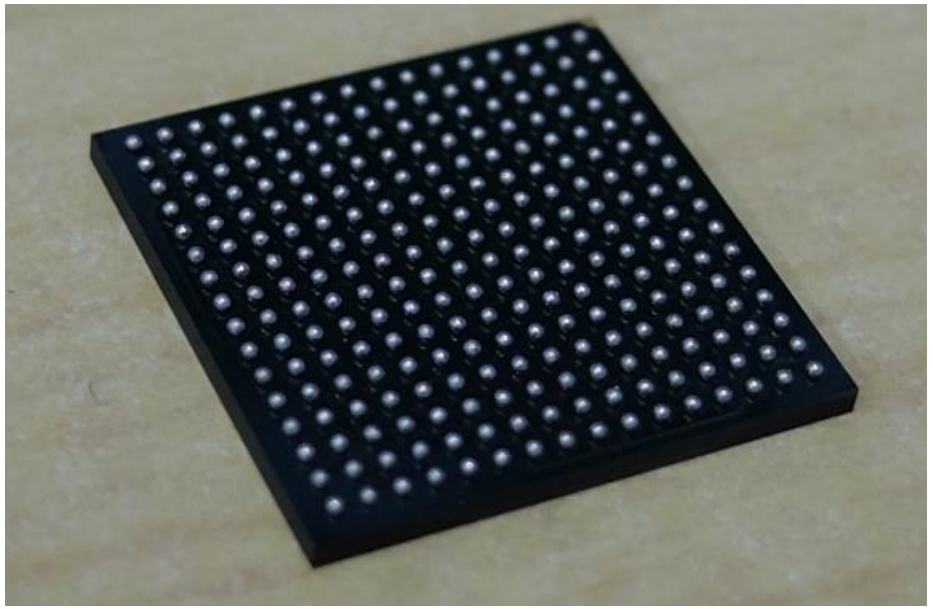
3-input LUT structure



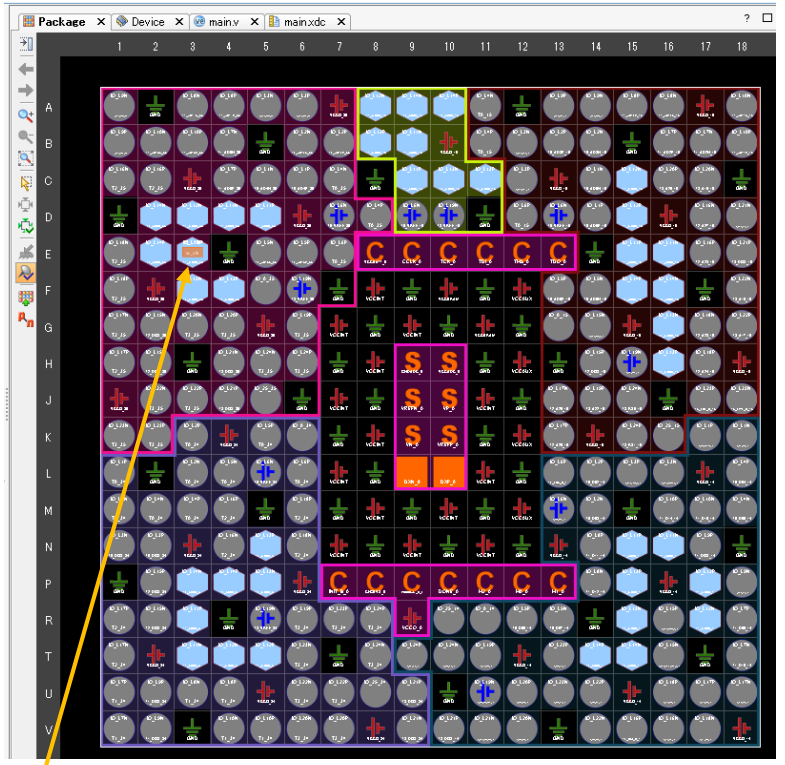
6-input LUT for Artix-7 FPGA

FPGA (Field Programmable Gate Array)

- Artix-7 FPGA (xc7a35tcsg324-1L)
- Vivado, open implemented design, and select Layout Menu, then I/O planning
  - 16 x 16 = 324 pins, max user pins of 250



FPGA chip photo

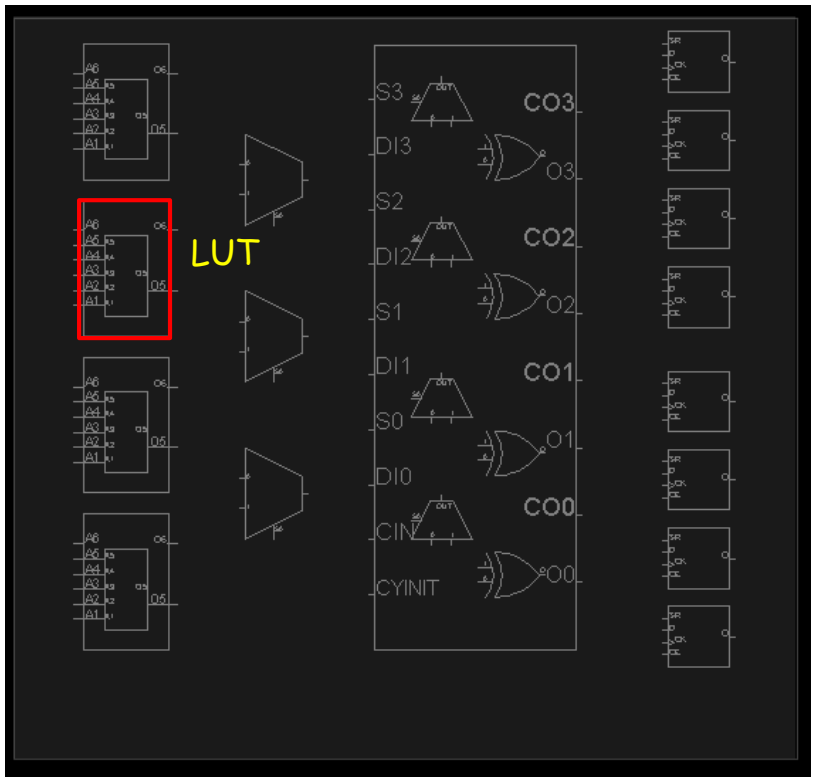


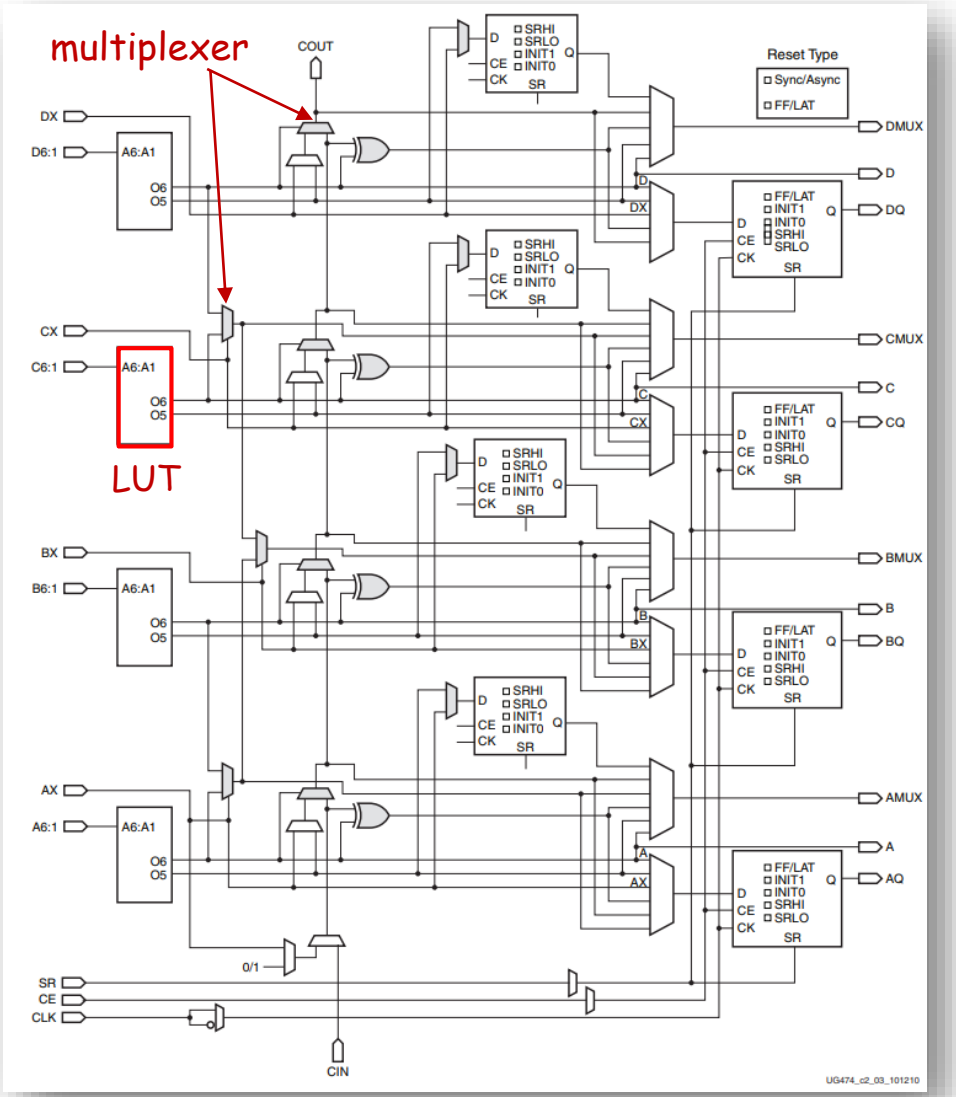
main11.xdcの最初の2行

```
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports { w_clk }];
create_clock -add -name sys_clk -period 10.00 [get_ports {w_clk}];
```

Slice and SLICEL

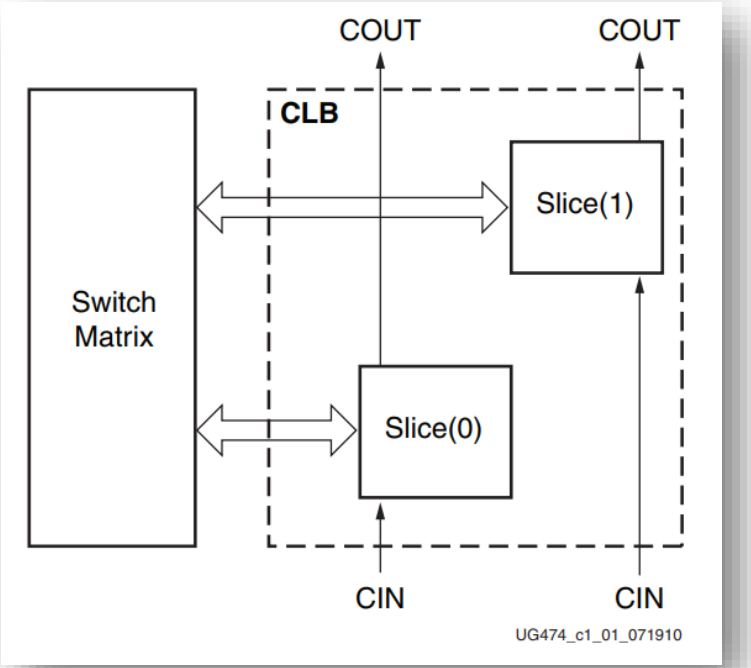
- There are two types of slices. They are SLICEL and SLICEM.
- Each slice has 4 LUTs and 8 FFs.





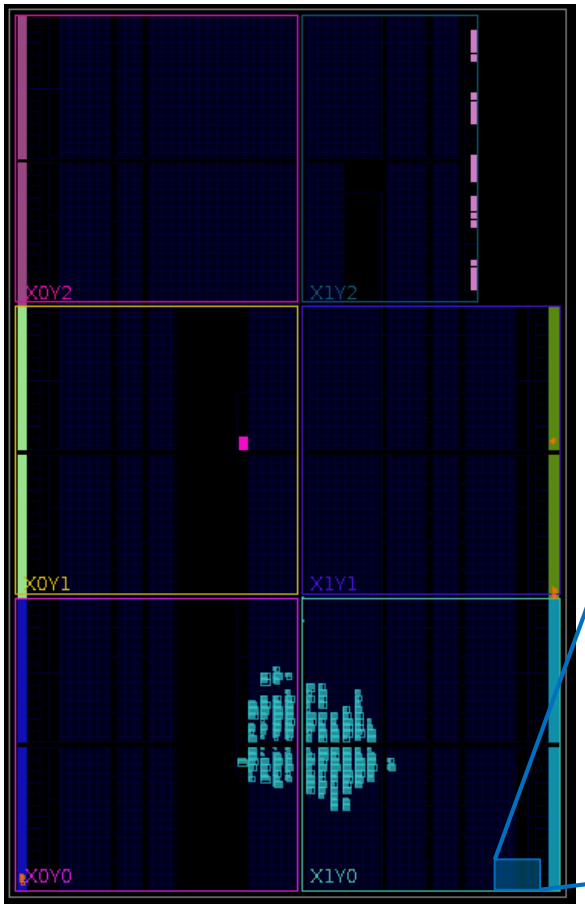
CLB (Configurable Logic Block) and Slice

- Each CLB has two slices
  - There are two types of slices. They are SLICEL and SLICEM.
  - Approximately two-thirds of the slices are SLICEL logic slices and the rest are SLICEM, which can also use their LUTs as distributed 64-bit RAM or as 32-bit shift registers (SRL32) or as two SRL16s.



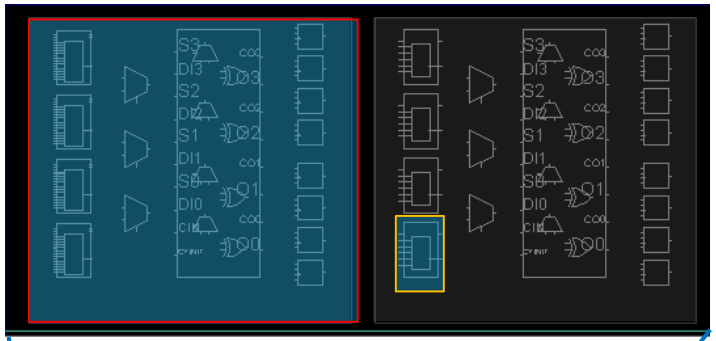
FPGA (Field Programmable Gate Array)

- Artix-7 FPGA (xc7a35tcsg324-1L)
  - 2,600 CLB = 5,200 slices



xc7a35tcsg324-1L FPGA die

Slice



LUT (Lookup Table)

CLB (Configurable Logic Block)

Xilinx 7 Series Configuration Logic Block (CLB)

7 Series FPGAs Configurable Logic Block

User Guide

UG474 (v1.8) September 27, 2016

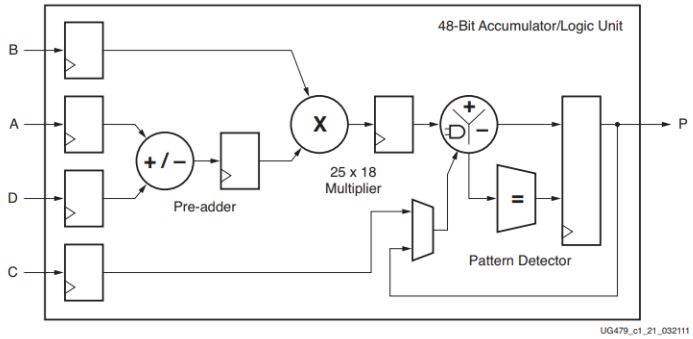
Slices = SLICEL + SLICEM
Distributed RAM (bit) = SLICEM x 256
Flip-Flops = LUTs x 2

Table 1-2: **Artix-7 FPGA CLB Resources**

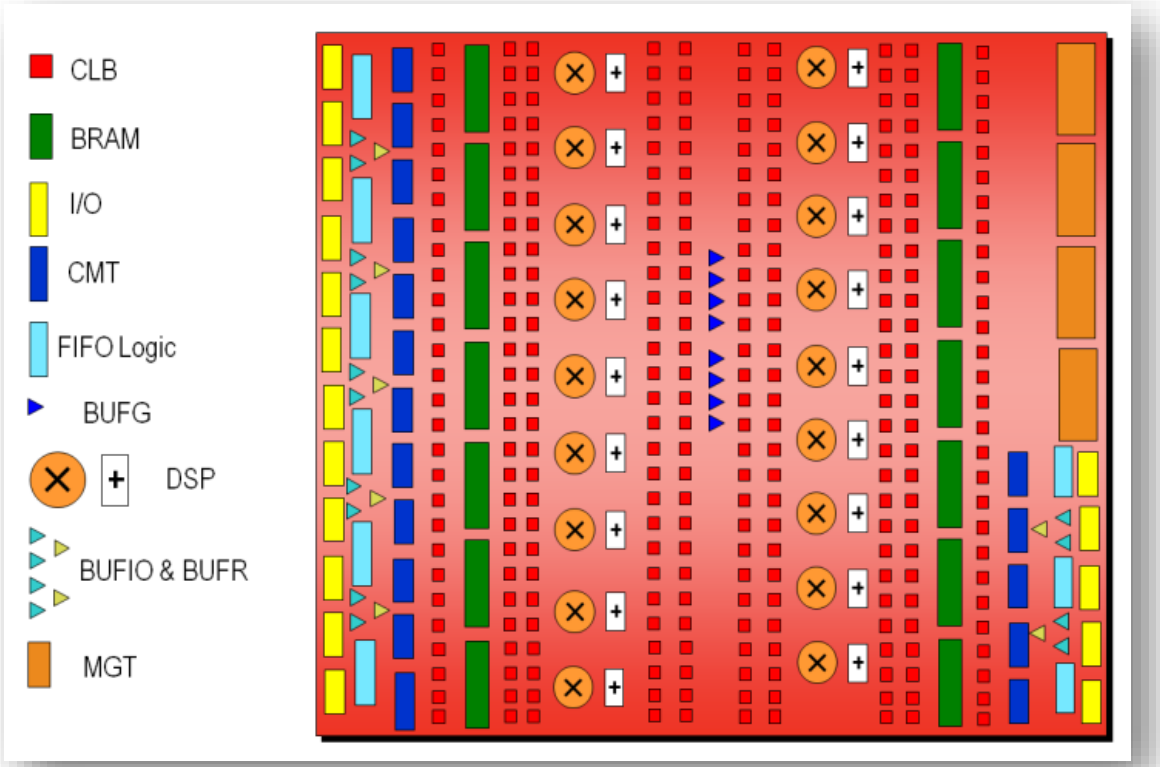
Device	Slices(1)	SLICEL	SLICEM	6-input LUTs	Distributed RAM (Kb)	Shift Register (Kb)	Flip-Flops
7A12T	2,000(2)	1,316	684	8,000	171	86	16,000
7A15T	2,600(2)	1,800	800	10,400	200	100	20,800
7A25T	3,650	2,400	1,250	14,600	313	156	29,200
7A35T	5,200(2)	3,600	1,600	20,800	400	200	41,600
7A50T	8,150	5,750	2,400	32,600	600	300	65,200
7A75T	11,800(2)	8,232	3,568	47,200	892	446	94,400
7A100T	15,850	11,100	4,750	63,400	1,188	594	126,800
7A200T	33,650	22,100	11,550	134,600	2,888	1,444	269,200

Artix-7 Architecture Overview

- CLB (Configurable Logic Block)
- BRAM (Block RAM, embedded memory)
- DSP (Digital Signal Processing)
- CMT (Clock Management Tile)
- Routing fabric

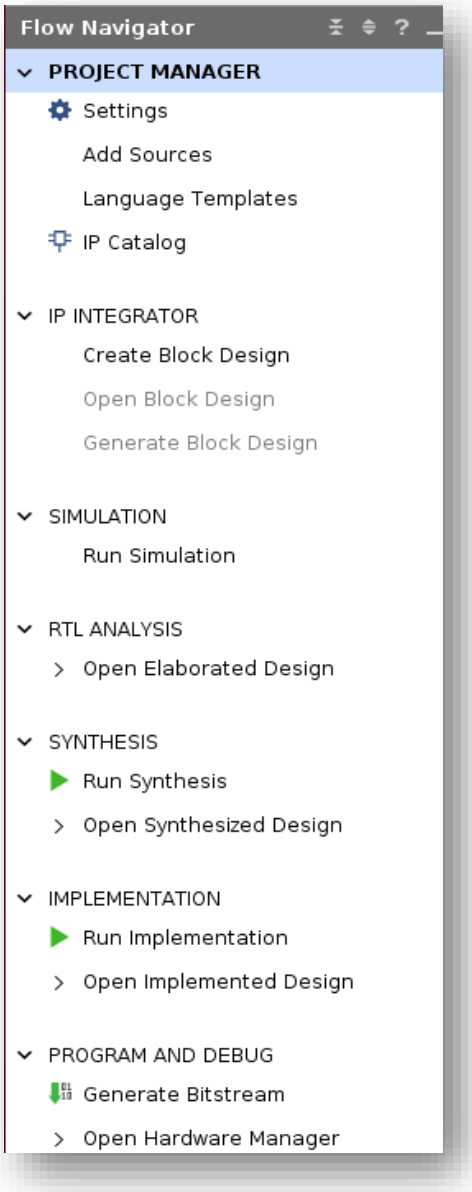


DSP Slice

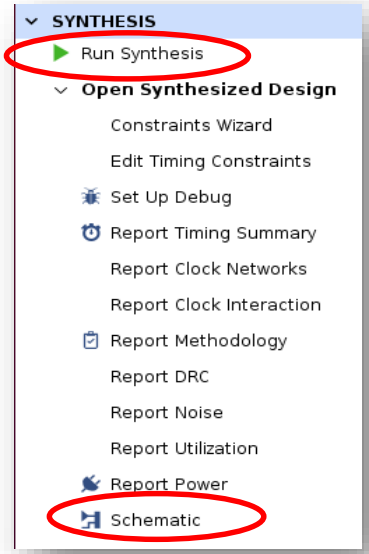


FPGA Design Flow

- IP Integrator
- Simulation
- RTL Analysis
- Synthesis
  - Logic Synthesis（論理合成），ネットリストの生成
- Implementation
  - Place and Route（配置・配線）
- Program and Debug
  - Bitstream generation
  - Program



Example Synthesis Result



code090.v

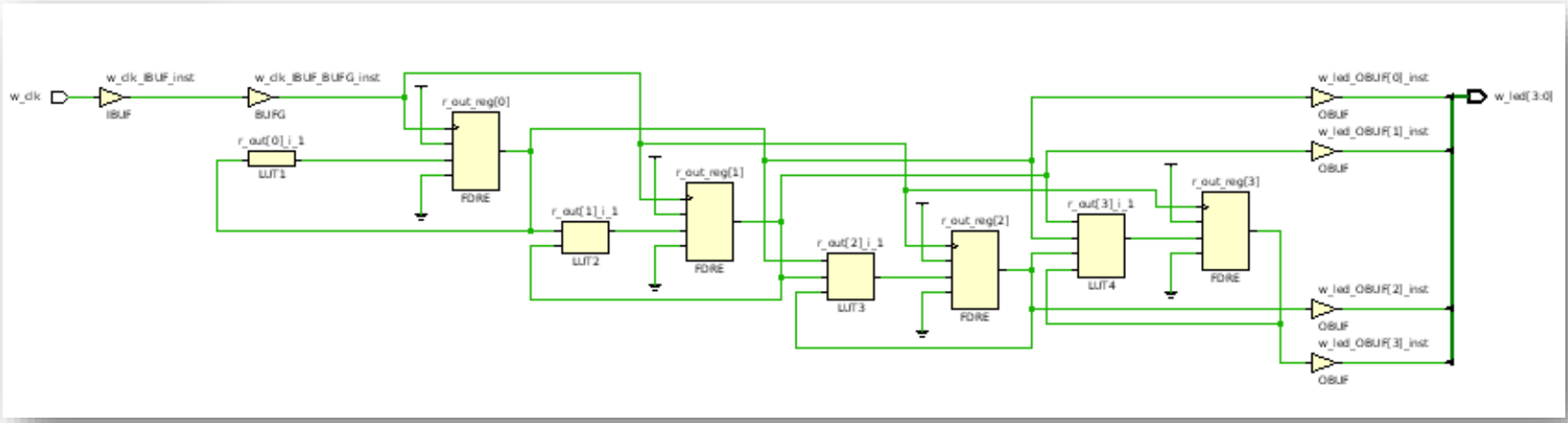
```
module m_main (w_clk, w_led);
  input  wire w_clk;
  output wire [3:0] w_led;

  reg [3:0] r_out = 0;
  always@(posedge w_clk) r_out <= r_out + 1;
  assign w_led = r_out;
endmodule
```

r_out[0]_i_1

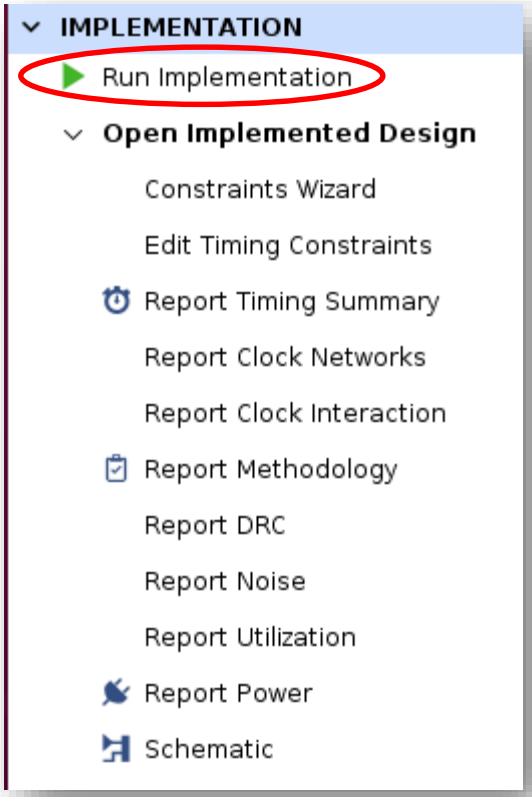
I0	O=!I0
0	1
1	0

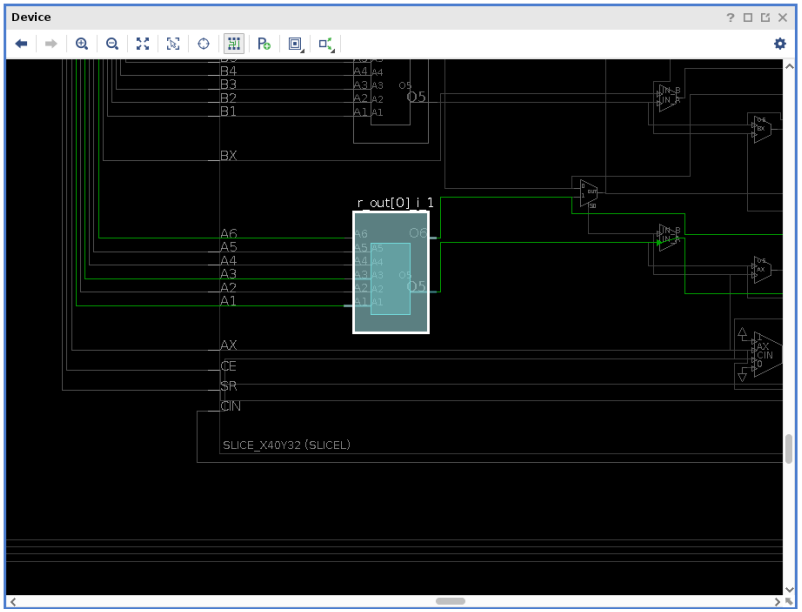
r_out[1]_i_1

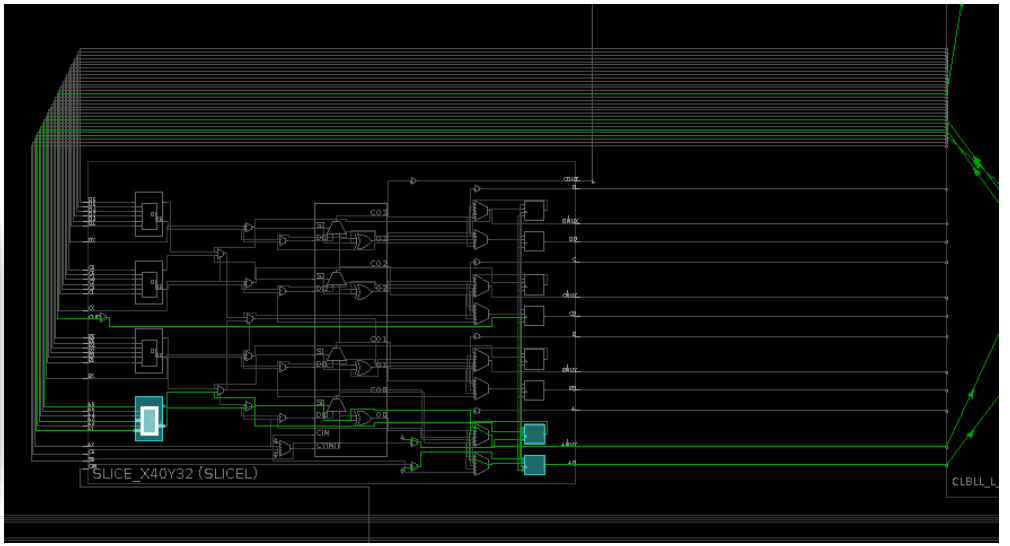
I1	I0	O=I0 & !I1 + !I0 & I1
0	0	0
0	1	1
1	0	1
1	1	0

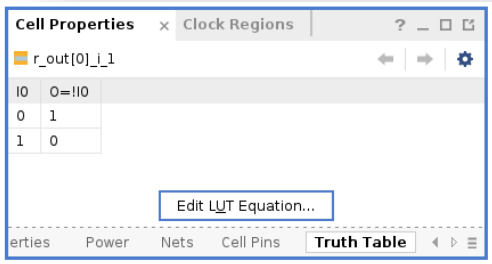


Example Implementation Result





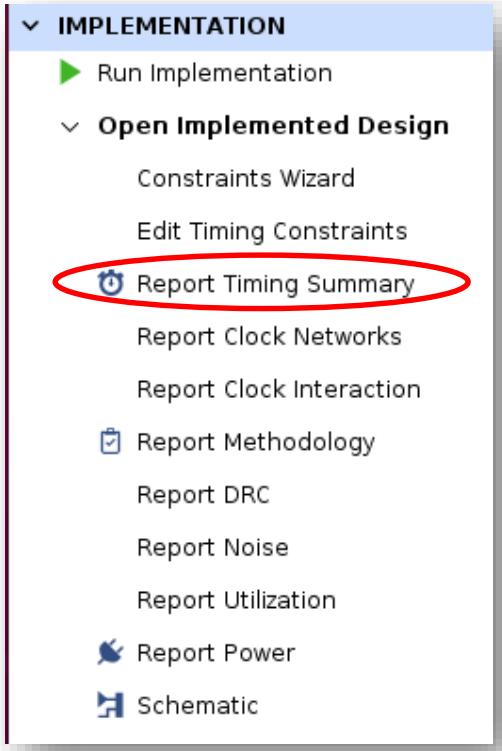


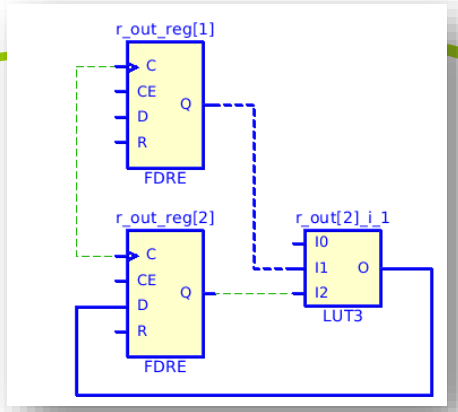


r_out[1]_i_1

I1	I0	O=I0 & !I1 + !I0 & I1
0	0	0
0	1	1
1	0	1
1	1	0

Example Implementation Result





Design Timing Summary

click here

Setup		Hold	
Worst Negative Slack (WNS):	6.674 ns	Worst Hold Slack (WHS):	0.465 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4	Total Number of Endpoints:	4

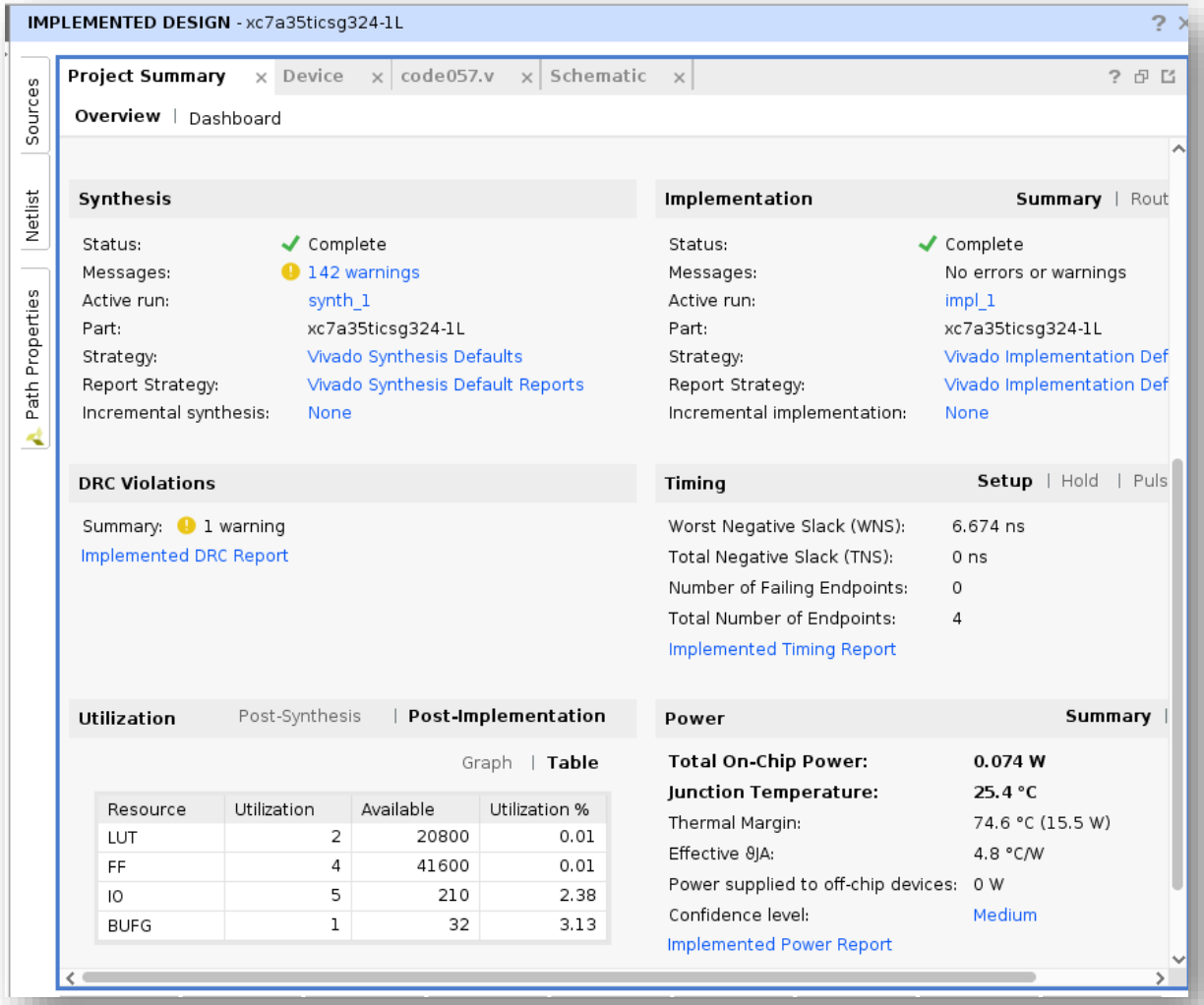
All user specified timing constraints are met.

Intra-Clock Paths - sys_clk - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	6.674	1	1	4	r_out_reg[1]/C	r_out_reg[2]/D	3.290	0.718	2.572	10.0
Path 2	6.694	1	1	4	r_out_reg[1]/C	r_out_reg[3]/D	3.316	0.744	2.572	10.0
Path 3	8.156	1	1	4	r_out_reg[1]/C	r_out_reg[1]/D	1.884	0.746	1.138	10.0
Path 4	8.172	1	1	5	r_out_reg[0]/C	r_out_reg[0]/D	1.822	0.580	1.242	10.0

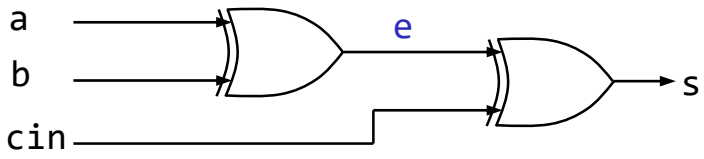
Example Implementation Result

- Project Summary Window
  - This window is useful to check the synthesis and implementation result at a glance.



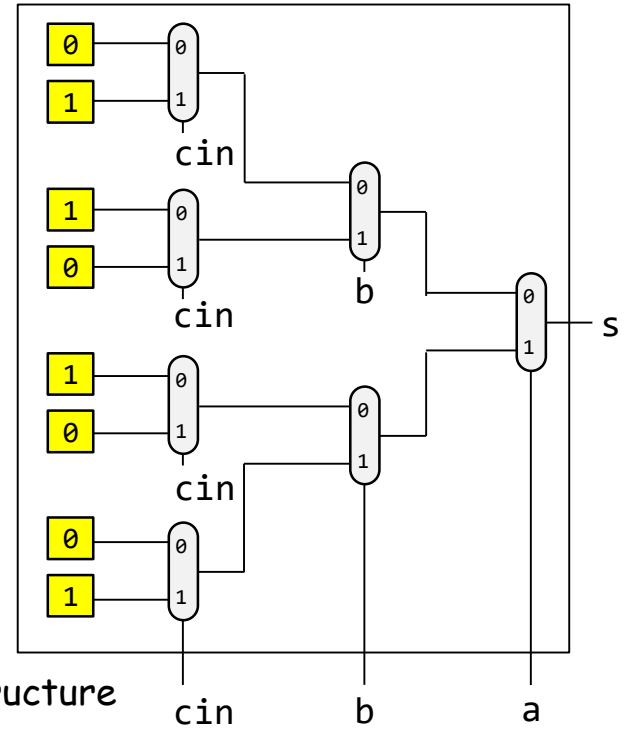
Distributed RAM としても利用できる LUT

- Reconfigurable Logic
  - 3-input LUT has 8 configuration registers
  - 6-input LUT for Artix-7 FPGA

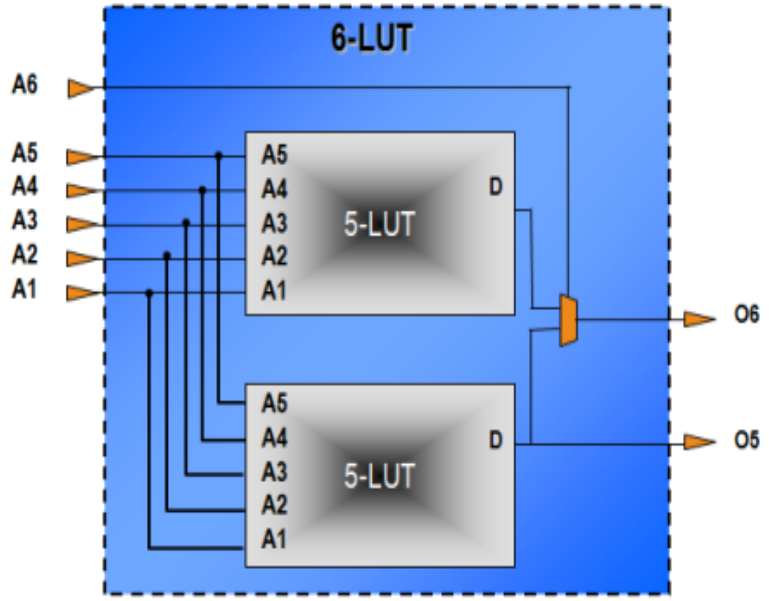


Truth table

a	b	cin	s
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



3-input LUT structure



6-input LUT for Artix-7 FPGA

Slice, SLICEL and SLICEM

Approximately two-thirds of the slices are SLICEL logic slices and the rest are SLICEM, which can also use their LUTs as distributed 64-bit RAM (LUTRAM) or as 32-bit shift registers (SRL32) or as two SRL16s.

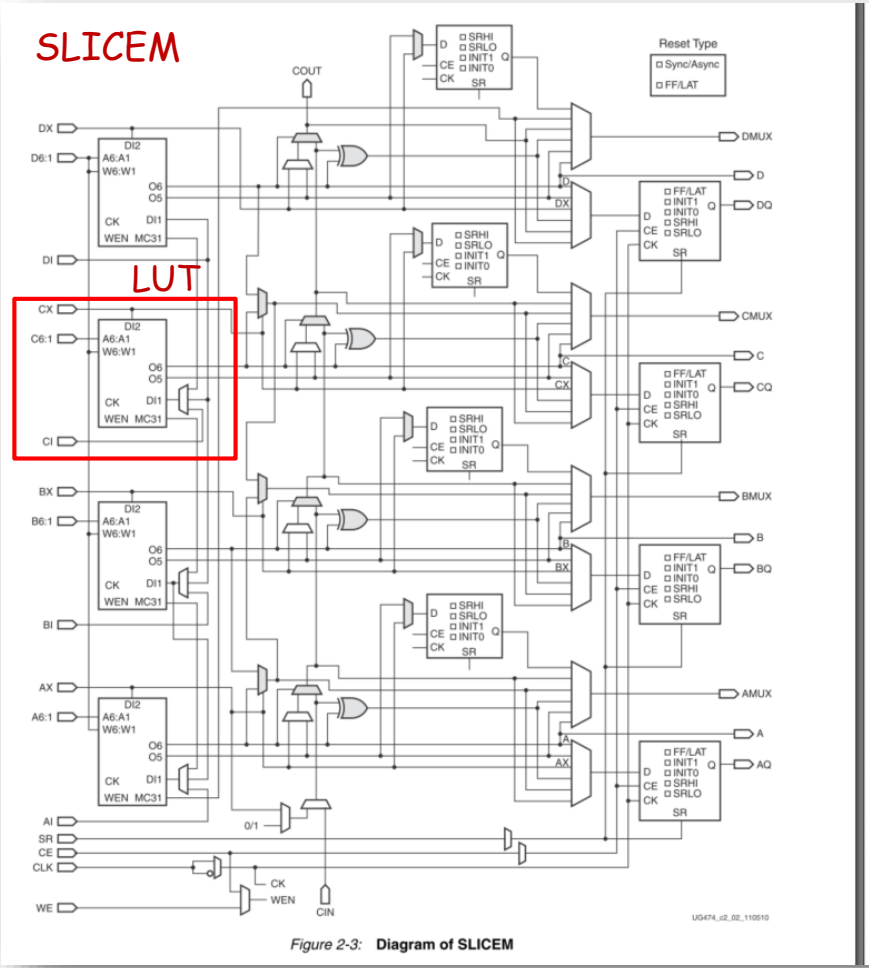


Figure 2-3: **Diagram of SLICEM**

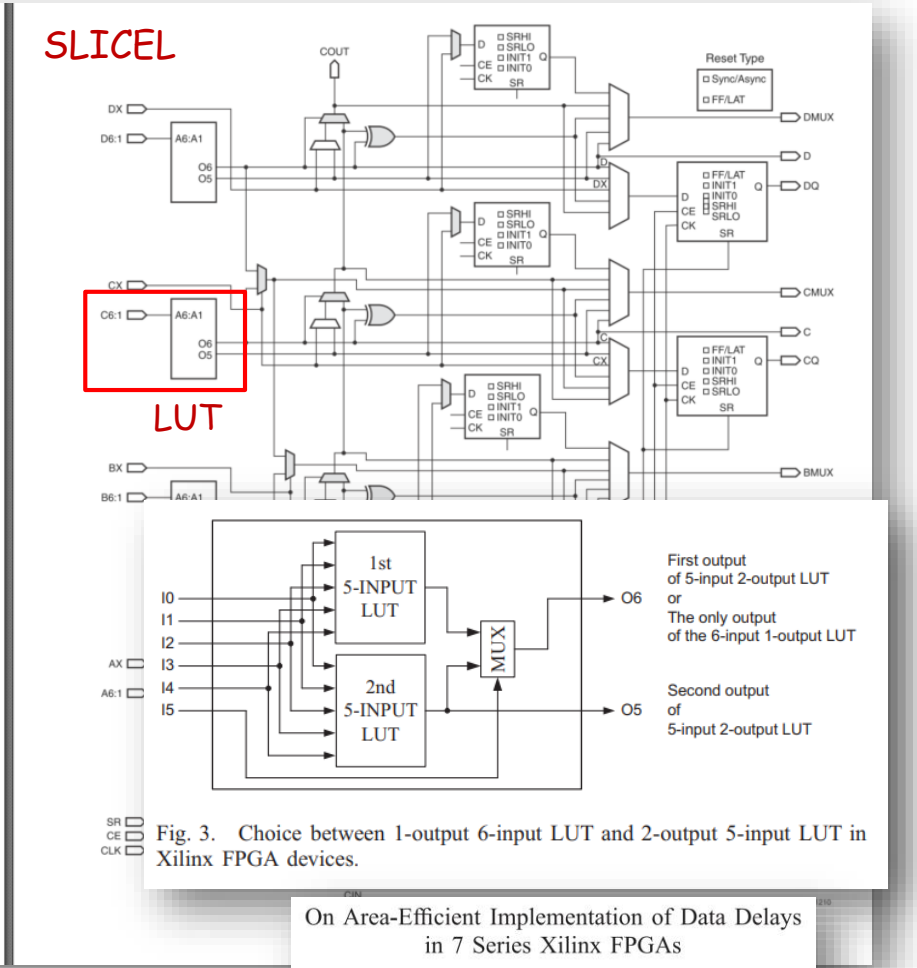


Fig. 3. Choice between 1-output 6-input LUT and 2-output 5-input LUT in Xilinx FPGA devices.

On Area-Efficient Implementation of Data Delays in 7 Series Xilinx FPGAs

Marek Parfieniuk
Department of Digital Media and Computer Graphics
Bialystok University of Technology
Wiejska 45A, 15-351 Bialystok, Poland
Email: m.parfieniuk@pb.edu.pl

Sang Yoon Park
Department of Electronic Engineering and MPEES-ARC
Myongji University
Yongin 449-728, Korea
Email: sypark@mju.ac.kr

Distributed RAM（分散メモリ）

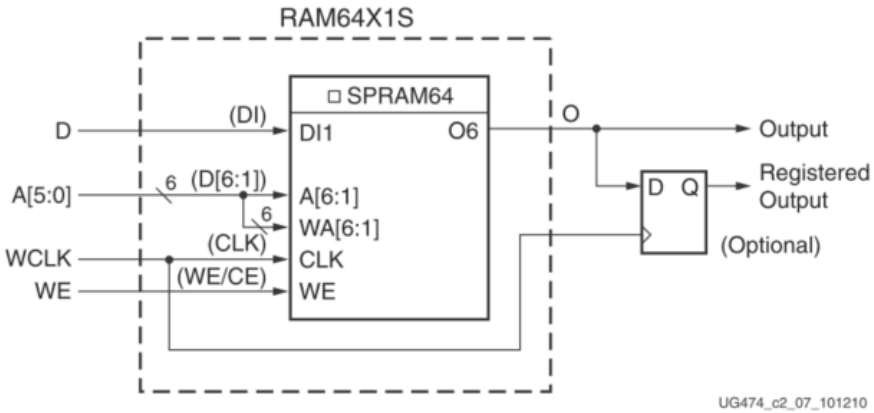


Figure 2-8: **64 X 1 Single Port Distributed RAM (RAM64X1S)**

```
module m_RAM64X1S (clk, a, d, we, dout);
  input wire clk;     // clock signal
  input wire [5:0] a; // address
  input wire d, we;   // data_in, write_enable
  output wire dout;   // data_out

  reg [0:0] mem [0:63];
  assign dout = mem[a];
  always @(posedge clk) if(we) mem[a] <= d;
endmodule
```

Table 2-3: **Distributed RAM Configuration**

RAM	Description	Primitive	Number of LUTs
32 x 1S	Single port	RAM32X1S	1
32 x 1D	Dual port	RAM32X1D	2
32 x 2Q	Quad port	RAM32M	4
32 x 6SDP	Simple dual port	RAM32M	4
64 x 1S	Single port	RAM64X1S	1
64 x 1D	Dual port	RAM64X1D	2
64 x 1Q	Quad port	RAM64M	4
64 x 3SDP	Simple dual port	RAM64M	4
128 x 1S	Single port	RAM128X1S	2
128 x 1D	Dual port	RAM128X1D	4
256 x 1S	Single port	RAM256X1S	4

- Single port
  - Common address port for synchronous writes and asynchronous reads
    - Read and write addresses share the same address bus

LUTRAM = 1

Distributed RAM（分散メモリ）

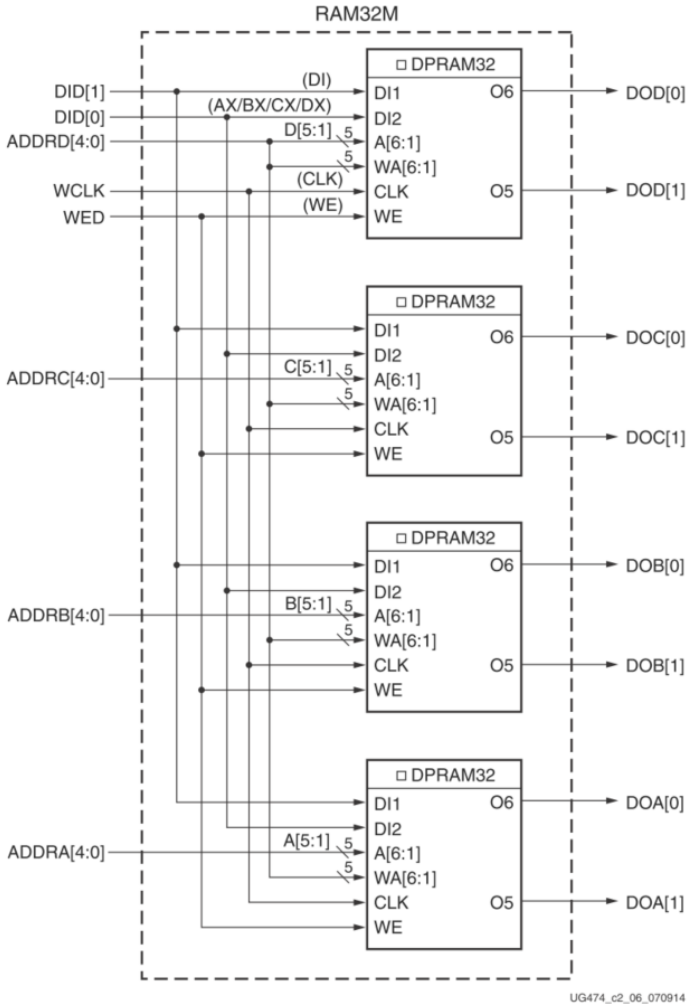


Figure 2-6: **32 X 2 Quad Port Distributed RAM (RAM32M)**

Table 2-3: **Distributed RAM Configuration**

RAM	Description	Primitive	Number of LUTs
32 x 1S	Single port	RAM32X1S	1
32 x 1D	Dual port	RAM32X1D	2
32 x 2Q	Quad port	RAM32M	4
32 x 6SDP	Simple dual port	RAM32M	4
64 x 1S	Single port	RAM64X1S	1
64 x 1D	Dual port	RAM64X1D	2
64 x 1Q	Quad port	RAM64M	4
64 x 3SDP	Simple dual port	RAM64M	4
128 x 1S	Single port	RAM128X1S	2
128 x 1D	Dual port	RAM128X1D	4
256 x 1S	Single port	RAM256X1S	4

- Quad port
  - One port for synchronous writes and asynchronous reads
  - Three ports for asynchronous reads

Distributed RAM（分散メモリ）

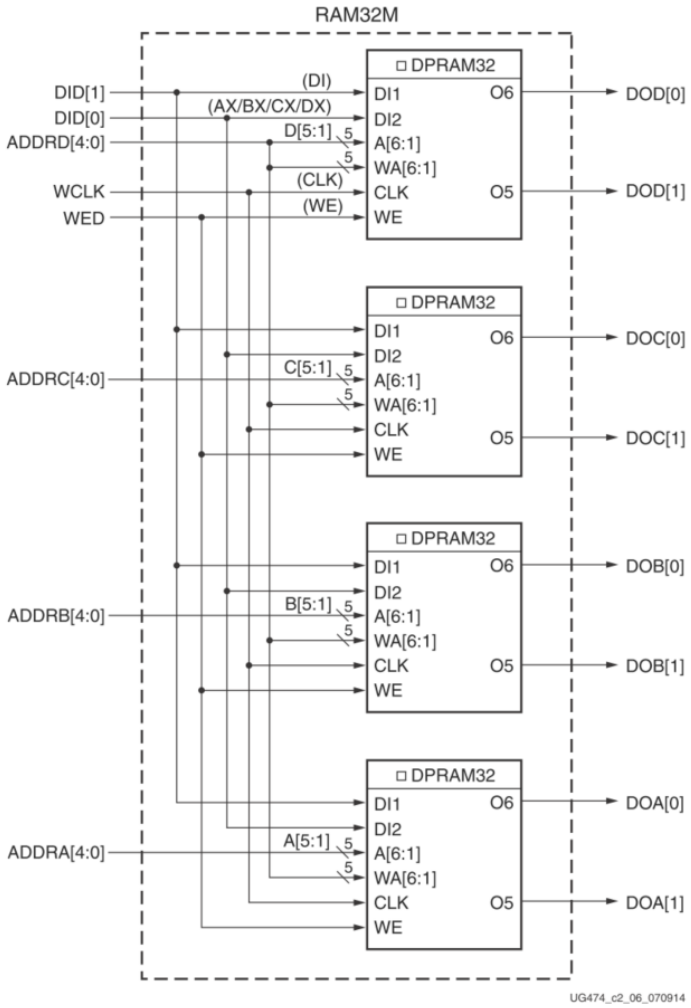
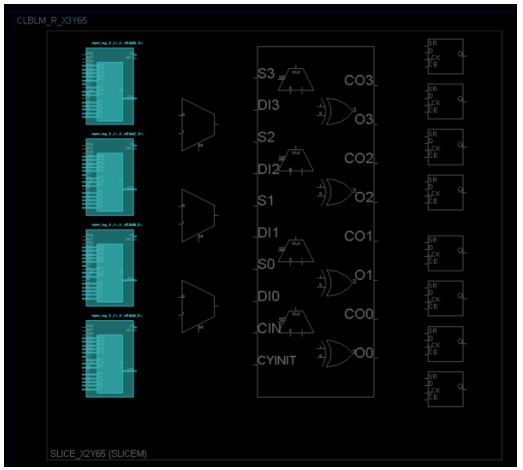


Figure 2-6: **32 X 2 Quad Port Distributed RAM (RAM32M)**

```
module m_RAM32M_Q (clk, a1, a2, a3, a4, d, we, dout1, dout2, dout3, dout4);
  input wire clk;
  input wire [4:0] a1, a2, a3, a4;
  input wire [1:0] d;
  input wire we;
  output wire [1:0] dout1, dout2, dout3, dout4;

  reg [1:0] mem [0:31];
  assign dout1 = mem[a1];
  assign dout2 = mem[a2];
  assign dout3 = mem[a3];
  assign dout4 = mem[a4];
  always @(posedge clk) if(we) mem[a1] <= d;
endmodule
```

Failed Routes	LUT	FF	BRAM	URAM	DSP
	4	0	0.0	0	0
0	4	0	0.0	0	0

LUTRAM = 4



Register file design of RISC-V soft processors

32x32 (32regs-32bit) single port register file

```
module m_RegFile_S (clk, a, d, we, dout);
  input wire clk;
  input wire [4:0] a;
  input wire [31:0] d;
  input wire we;
  output wire [31:0] dout;

  reg [31:0] mem [0:31];
  assign dout = mem[a];
  always @(posedge clk) if(we) mem[a] <= d;
endmodule
```

LUTRAM = 32

32x32 (32regs-32bit) dual port register file

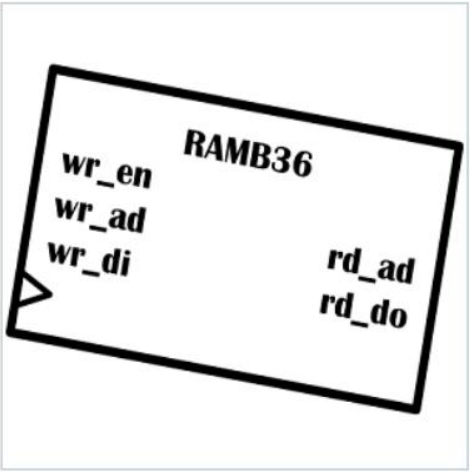
```
module m_RegFile_D (clk, a1, a2, d, we, dout1, dout2);
  input wire clk;
  input wire [4:0] a1, a2;
  input wire [31:0] d;
  input wire we;
  output wire [31:0] dout1, dout2;

  reg [31:0] mem [0:31];
  assign dout1 = mem[a1];
  assign dout2 = mem[a2];
  always @(posedge clk) if(we) mem[a1] <= d;
endmodule
```

Resource	Utilization	Available	Utilization %
LUT	32	63400	0.05
LUTRAM	32	19000	0.17
IO	108	210	51.43
BUFG	1	32	3.13

BRAM (Block RAM)

https://www.acri.c.titech.ac.jp/wordpress/archives/10236

BRAM 達人への道 (1) 構造と基本的な使い方



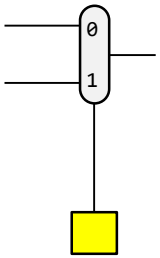
2021.01.21

FPGA を用いて AI のアクセラレーションのような高度な処理を行うことが当たり前の時代になりました。しかし、FPGA を利用する全てのエンジニアがそうしたアプリケーションに携わっているわけではなく、初心者、初級者レベルのエンジニアが多数おられるということも事実ではないかと思います。

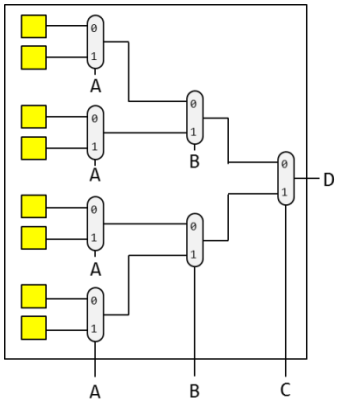
この記事では、現場レベルで使えるちょっとした工夫や考え方のヒントを紹介し、初級者のレベルアップや、現場の方々の手助けにつながればと考えています。

Configuration bitstream

- Bit vector of all configuration registers.
  - Program FPGA using a configuration chain of shift register like hardware.



Routing fabric



LUT

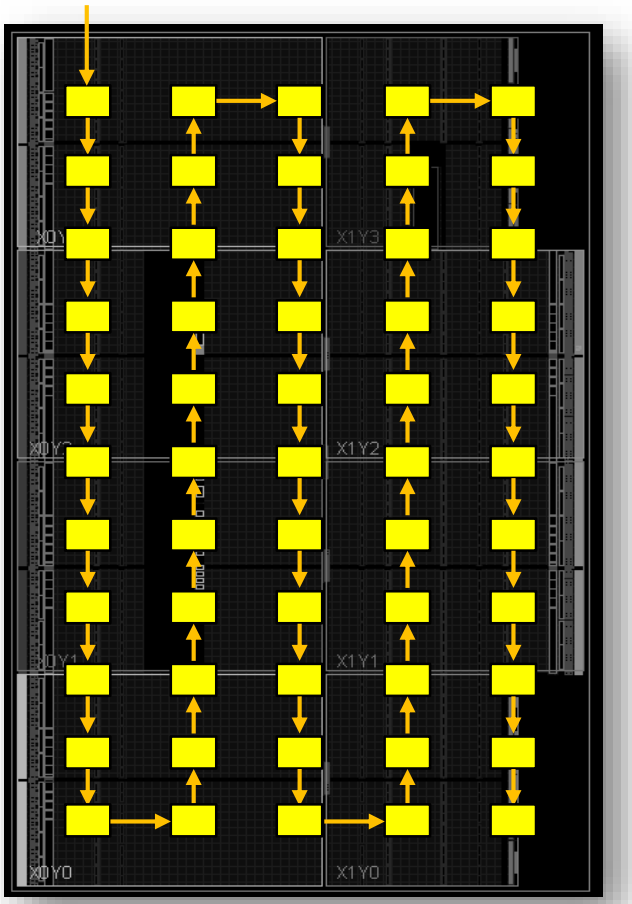
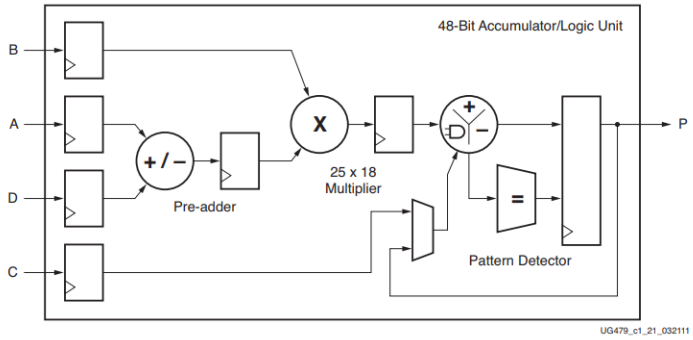


表 1-1：ビットストリームの長さ

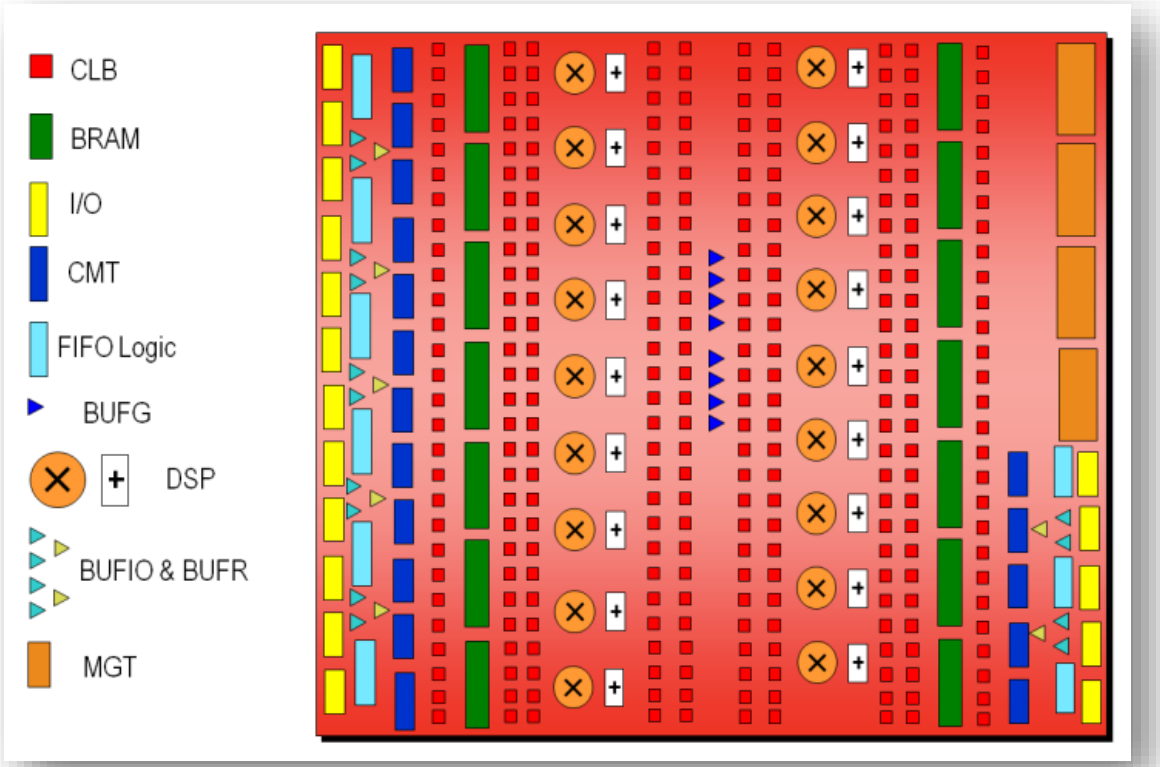
デバイス	コンフィギュレーション ビットストリームの長さ (ビット)
Artix-7 ファミリ	
7A15T	17,536,096
7A35T	17,536,096
7A50T	17,536,096
7A75T	30,606,304
7A100T	30,606,304
7A200T	77,845,216

Artix-7 Architecture Overview

- CLB (Configurable Logic Block)
- BRAM (Block RAM, embedded memory)
- DSP (Digital Signal Processing)
- CMT (Clock Management Tile)
- Routing fabric

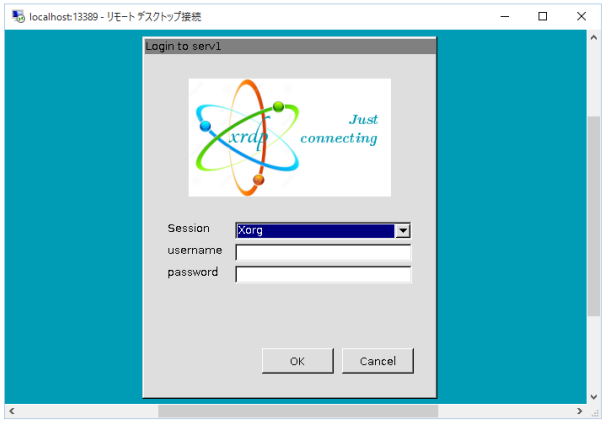


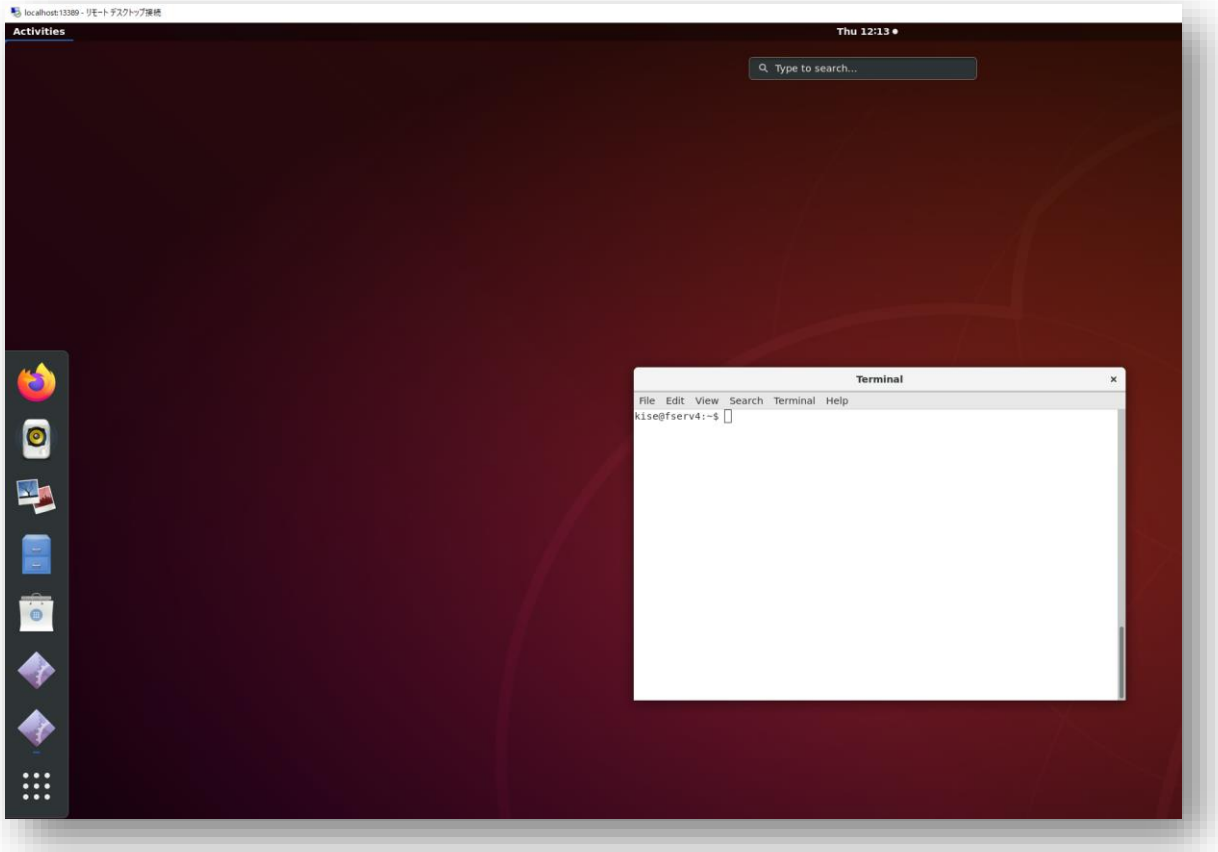
DSP Slice



ACRiルームのデモンストレーション

- LUT で実現される真理値表を確認する方法.
- Clocking Wizard を使って 20MHz のクロック信号を生成する方法.





Carry Cain in slice

electronics

MDPI

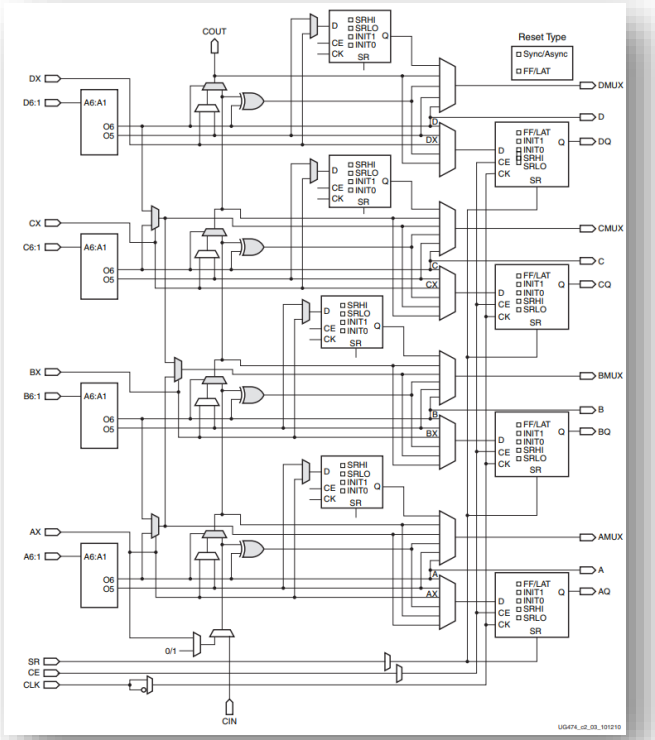
Article

Mapping Arbitrary Logic Functions onto Carry Chains in FPGAs

Raouf Senhadji-Navarro *,† and **Ignacio Garcia-Vargas** †

Department of Computer Architecture and Technology, University of Seville, 41012 Seville, Spain; iggv@us.es
* Correspondence: raouf@us.es
† These authors contributed equally to this work.

Abstract: Current Field Programmable Gate Arrays (FPGAs) provide fast routing links and special logic to perform carry operations; however, these resources can also be used to implement non-arithmetic circuits. In this paper, a new approach for mapping logic functions onto carry chains is presented. Unlike other approaches, the proposed technique can be applied to any logic function



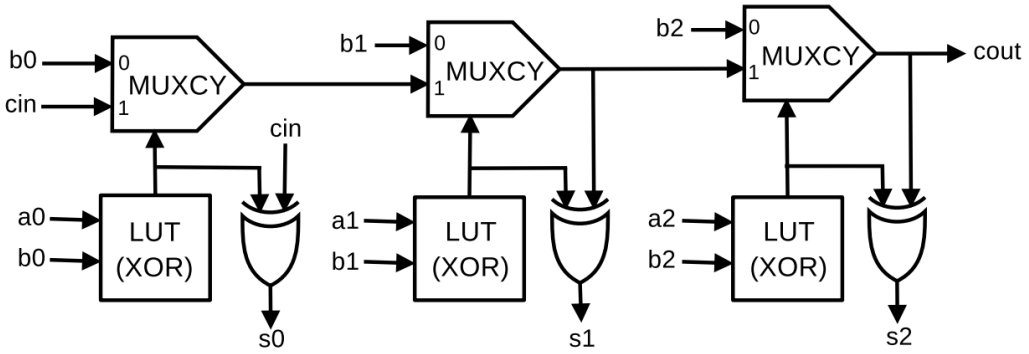


Figure 2. A 3-bit full adder implemented in a Xilinx FPGA device using carry chains.

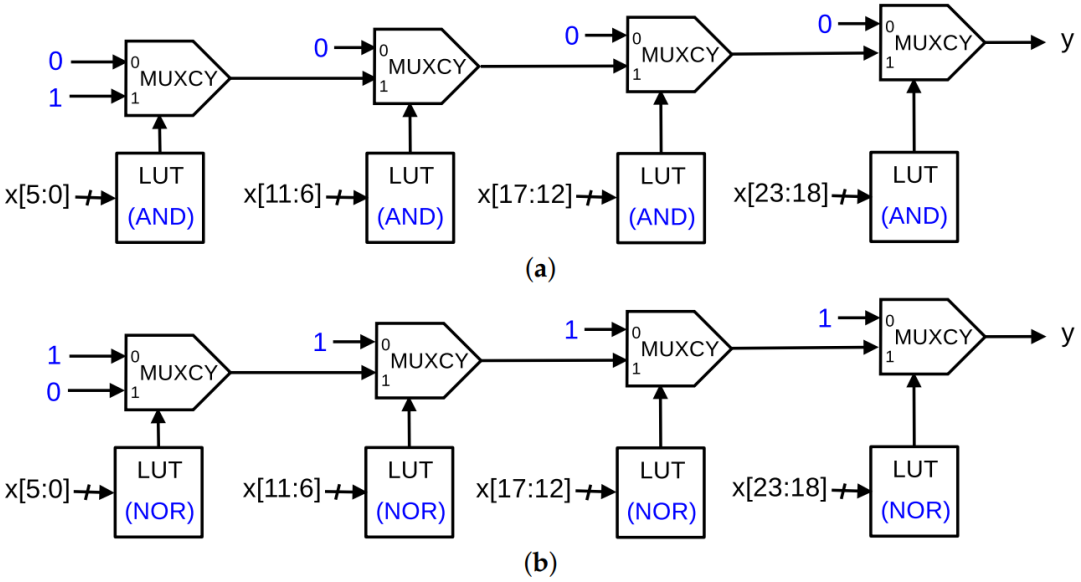


Figure 3. Examples of the implementation of wide input functions using carry chains: (**a**) 24-input AND and (**b**) 24-input OR.

References

- Computer Logic Design support page
  - https://www.arch.cs.titech.ac.jp/lecture/CLD/
- ACRi Room
  - https://gw.acri.c.titech.ac.jp
- ACRi Blog
  - https://www.acri.c.titech.ac.jp/wordpress/
- 情報工学系計算機室
  - http://www.csc.titech.ac.jp/
- Xilinx Vivado Design Suite
  - https://japan.xilinx.com/products/design-tools/vivado.html
- Digilent Arty A7-35T
  - https://reference.digilentinc.com/reference/programmable-logic/arty-a7/start
- Verilog HDL
  - https://ja.wikipedia.org/wiki/Verilog