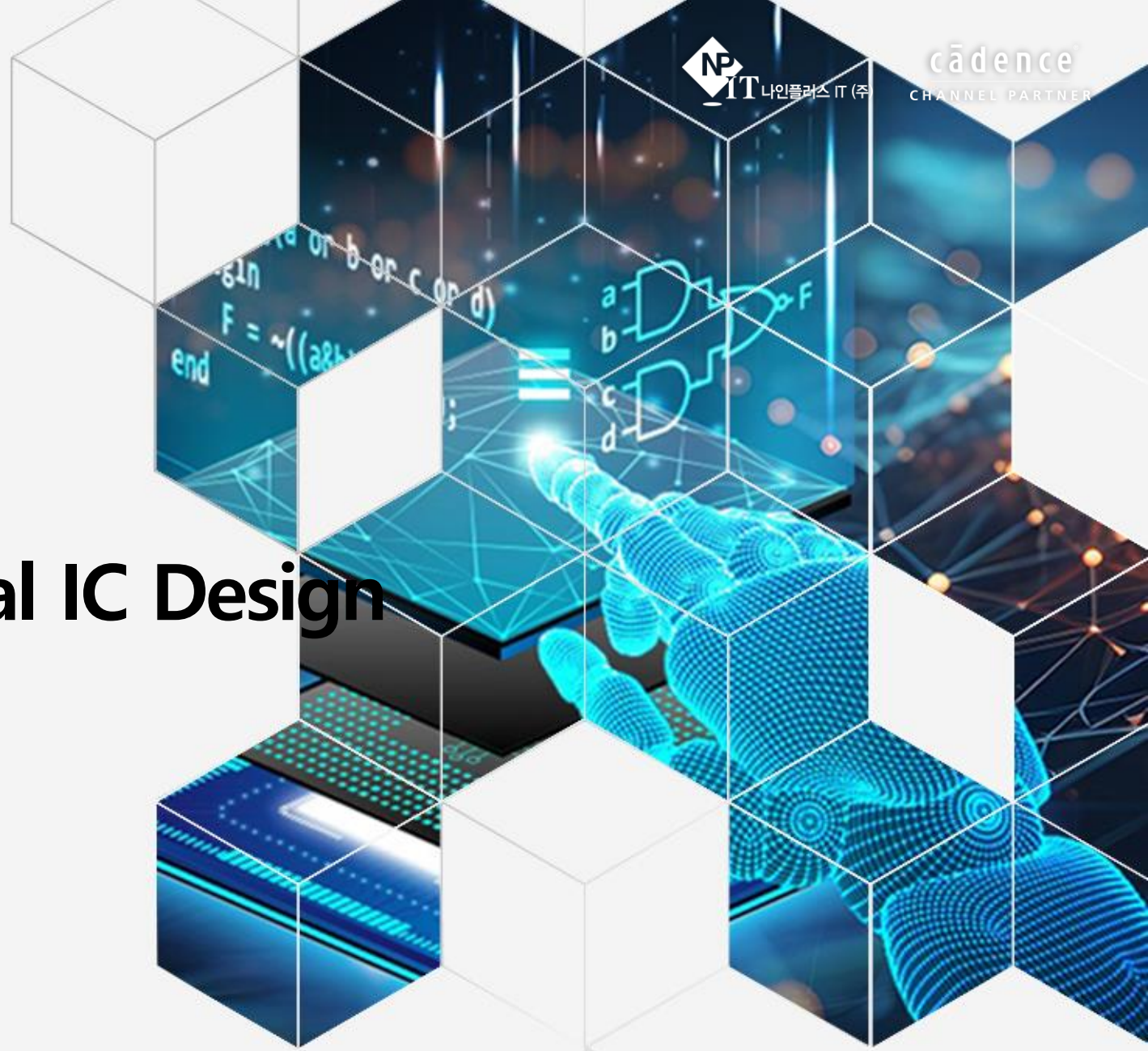
cadence



Fundamentals of Digital IC Design

Session2: Front End Design and Verification

Aug-2025 | NinePlus IT

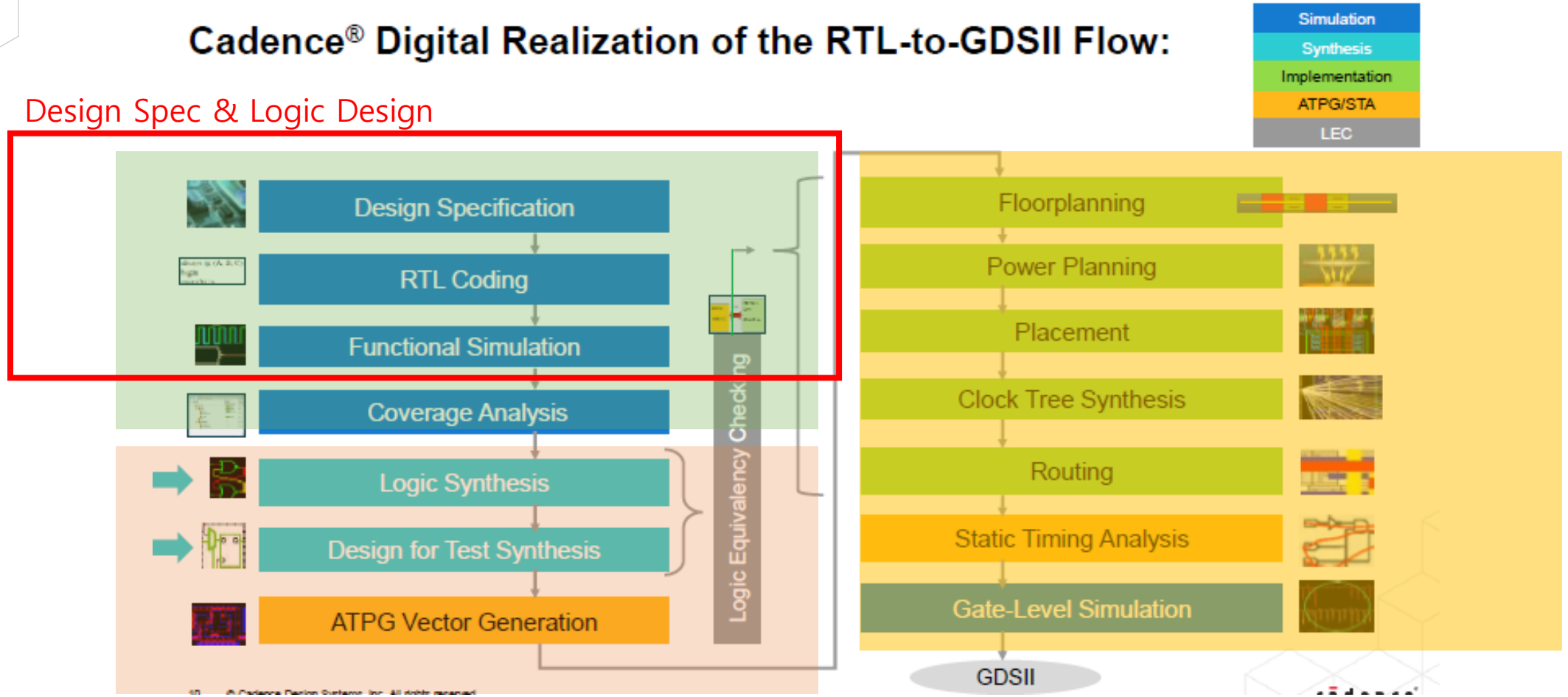
Research Engineer, Namhyun Cho

CONTENTS

Where are we?

Cadence® Digital Realization of the RTL-to-GDSII Flow:

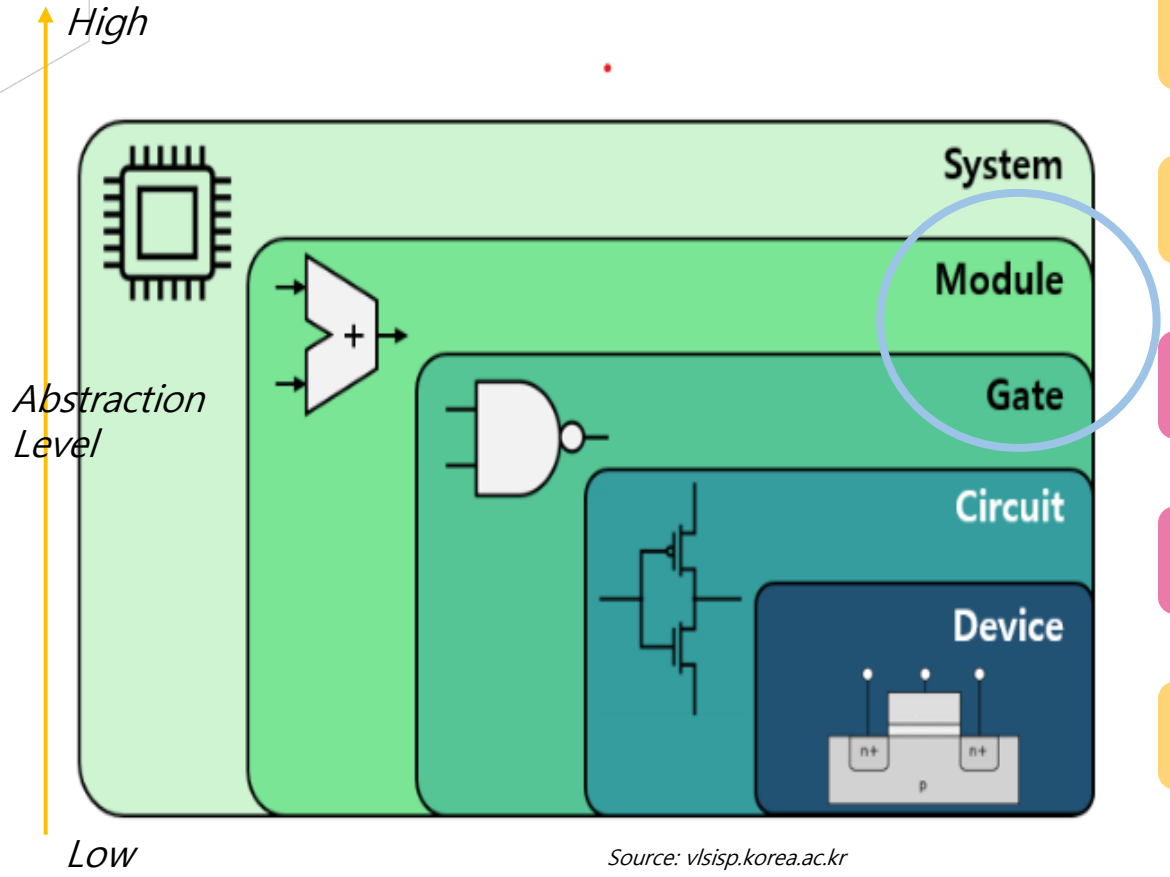
Design Spec & Logic Design



Front End

Back End

Diverse viewpoint to scrutinize chip designs



Source: vlsisp.korea.ac.kr

Device Level

- 소자의 설계

Circuit Level

- TR기반의 회로설계

Gate Level

- Logic Gate기반의 회로설계

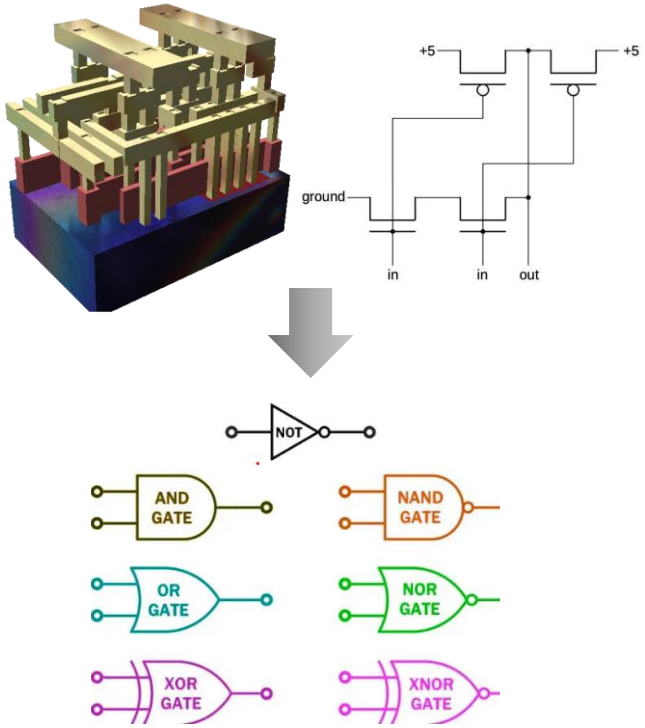
RTL(Module) Level

- 레지스터간 데이터 흐름 / 논리 연산 모델링 기반 회로설계

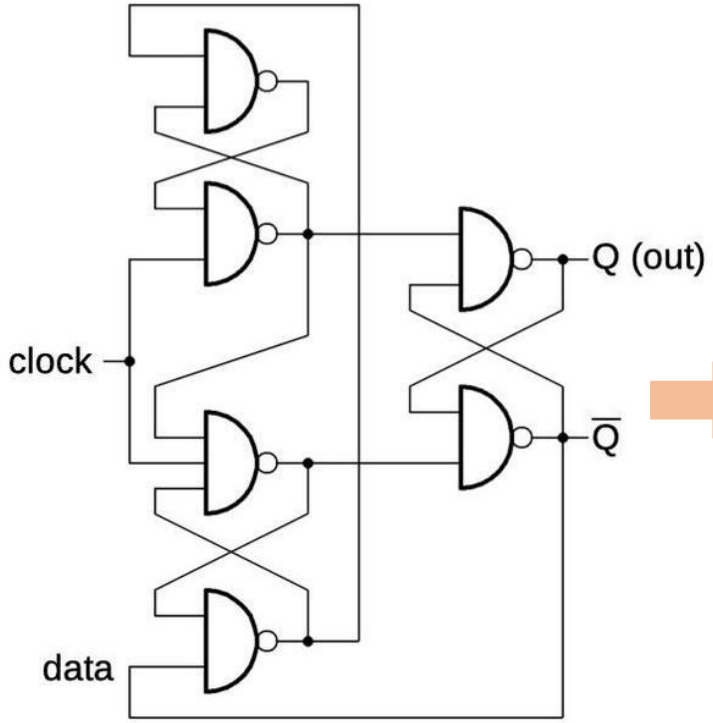
System Level

- 시스템 모듈 및 인터페이스를 통한 전체 시스템 설계

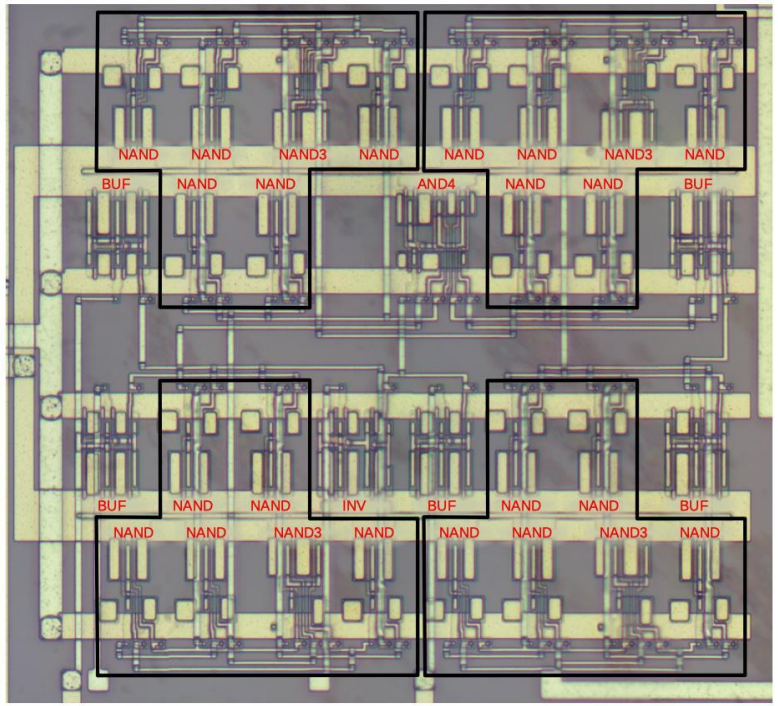
Semi-Custom IC Design



Standard Cell Library



Logic HW Design



Implementation: Place & Route

FPGA (참고)

◆FPGA(Field Programmable Gate Array)

- ◆ Gate Level 수준에서 설계 진행
- ◆ Standard Cell이 미리 구현되어 있음 (LUT)

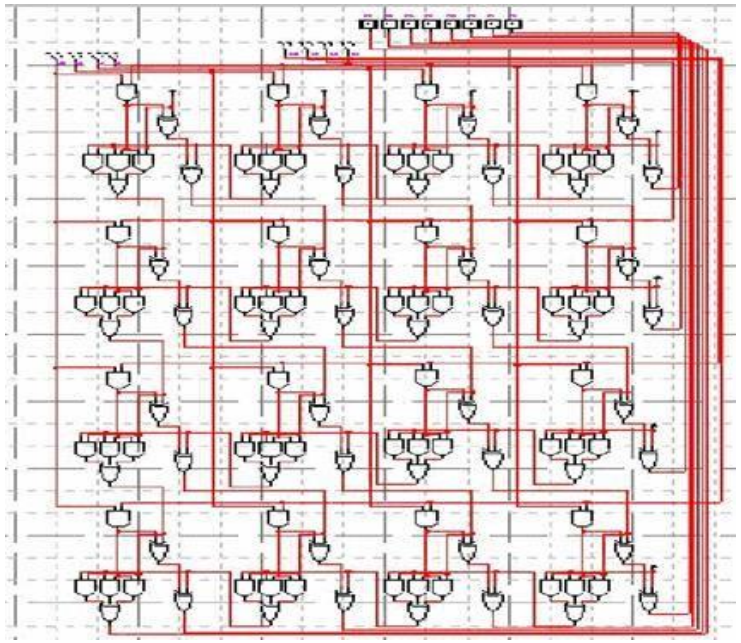
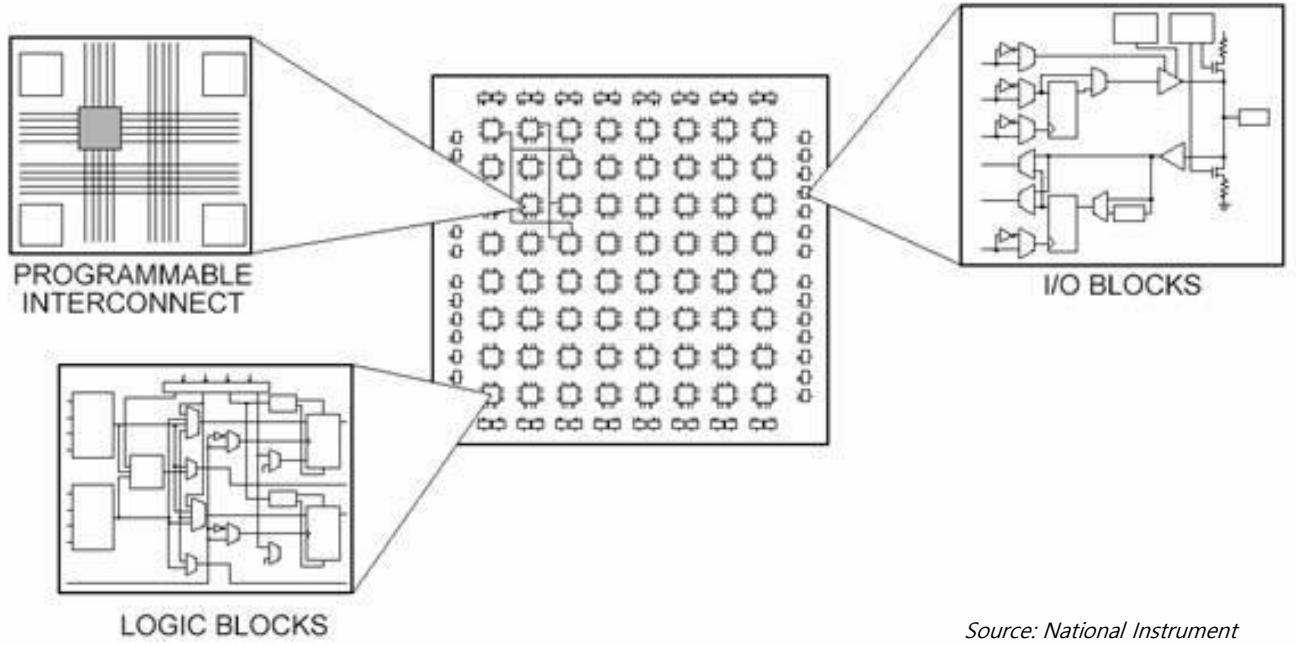


Fig. 4 X 4 multiplexer schematic



Source: National Instrument

Design Methodology

항목	DataFlow Modeling	Behavioral Modeling	Structural Modeling
기술 대상	데이터 흐름	동작 (제어 흐름)	게이트 / 모듈 연결 (인스턴스)
용도	간단한 조합 논리	FSM, 제어 로직 등	하드웨어 구조 묘사
추상화 수준	중간	높음	낮음

```
module mux2to1_dataflow (
    input  wire A, B,
    input  wire SEL,
    output wire Y
);
    assign Y = (SEL == 1'b0) ? A : B;
endmodule
```

```
module mux2to1_behavioral (
    input  wire A, B,
    input  wire SEL,
    output reg Y
);
   always @(*) begin
      if (SEL == 1'b0)
         Y = A;
      else
         Y = B;
   end
endmodule
```

```
module mux2to1_structural (
    input  wire A, B,
    input  wire SEL,
    output wire Y
);
    wire nSEL, outA, outB;

    not  (nSEL, SEL);
    and  (outA, A, nSEL);
    and  (outB, B, SEL);
    or   (Y, outA, outB);
endmodule
```

1. Dataflow Modeling

Lab1. 4-Bit Binary Full Adders With Fast Carry

1. Specification

TYPES SN54283, SN54LS283, SN54S283,
SN74283, SN74LS283, SN74S283
4-BIT BINARY FULL ADDERS WITH FAST CARRY
OCTOBER 1976—REVISED DECEMBER 1983

- **Full-Carry Look-Ahead Across the Four Bits**
- Systems Achieve Partial Look-Ahead Performance with the Economy of Ripple Carry
- Supply Voltage and Ground on Corner Pins to Simplify P-C Board Layout

TYPICAL ADD TIMES

TYPE	TWO 8-BIT WORDS	TWO 16-BIT WORDS	TYPICAL POWER DISSIPATION PER ADDER
'283	23ns	43ns	310 mW
'LS283	25ns	45ns	95 mW
'S283	15ns	30ns	510 mW

description

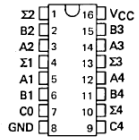
The '283 and 'LS283 adders are electrically and functionally identical to the '83A and 'LS283, respectively; only the arrangement of the terminals has been changed. The 'S283 high performance versions are also functionally identical.

These improved full adders perform the addition of two 4-bit binary words. The sum (Σ) outputs are provided for each bit and the resultant carry (C4) is obtained from the fourth bit. These adders feature full internal look-ahead across all four bits generating the carry term in ten nanoseconds, typically, for the '283 and 'LS283, and 7.5 nanoseconds for the 'S283. This capability provides the system designer with partial look-ahead performance at the economy and reduced package count of a ripple-carry implementation.

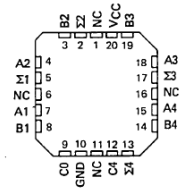
The adder logic, including the carry, is implemented in its true form. End around carry can be accomplished without the need for logic or level inversion.

Series 54, Series 54LS, and Series 54S circuits are characterized for operation over the full temperature range of −55°C to 125°C. Series 74, Series 74LS, and Series 74S circuits are characterized for 0°C to 70°C operation.

SN54283, SN54LS283 . . . J OR W PACKAGE
SN54S283 . . . J PACKAGE
SN74283 . . . J OR N PACKAGE
SN74LS283, SN74S283 . . . D, J OR N PACKAGE
(TOP VIEW)



SN54LS283, SN54S283 . . . FK PACKAGE
SN74LS283, SN74S283 . . . FN PACKAGE
(TOP VIEW)



NC – No internal connection

FUNCTION TABLE

INPUT						OUTPUT					
				WHEN C0 = L			WHEN C0 = H				
						WHEN C2 = L			WHEN C2 = H		
A1	B1	A2	B2	Σ1	Σ2	C2	Σ1	Σ2	C2		
A3	B3	A4	B4	Σ3	Σ4	C4	Σ3	Σ4	C4		
L	L	L	L	L	L	L	H	L	L		
H	L	L	L	H	L	L	L	H	L		
L	H	L	L	H	L	L	L	H	L		
H	H	L	L	L	H	L	H	H	L		
L	L	H	L	L	H	L	H	H	L		
H	L	H	L	H	H	L	L	L	H		
L	H	H	L	H	H	L	H	L	H		
H	H	H	L	L	L	H	H	L	H		
L	L	L	H	L	H	L	H	H	L		
H	L	L	H	H	H	L	L	L	H		
L	H	L	H	H	H	L	H	L	H		
H	H	L	H	L	L	H	H	L	H		
L	L	H	H	L	L	H	H	L	H		
H	L	H	H	H	L	H	L	H	H		
L	H	H	H	H	L	H	L	H	H		
H	H	H	H	L	H	H	H	H	H		

H = high level, L = low level

NOTE: Input conditions at A1, B1, A2, B2, and C0 are used to determine outputs Σ1 and Σ2 and the value of the internal carry C2. The values at C2, A3, B3, A4, and B4 are then used to determine outputs Σ3, Σ4, and C4.

3 TTL DEVICES

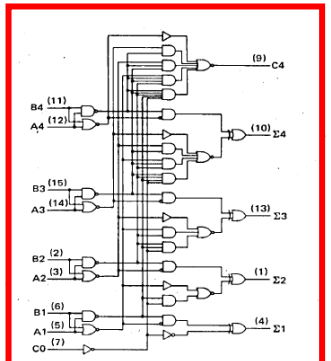
PRODUCTION DATA
This document contains information current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

3-917

TYPES SN54283, SN54LS283, SN54S283,
SN74283, SN74LS283, SN74S283
4-BIT BINARY FULL ADDERS WITH FAST CARRY

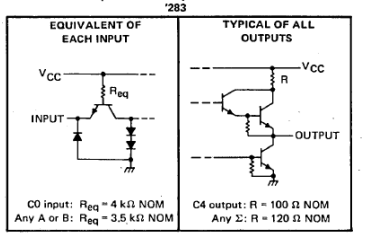
logic diagram



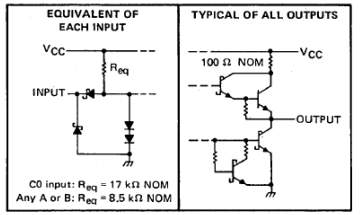
Pin numbers shown on logic notation are for D, J or N packages.

schematics of inputs and outputs

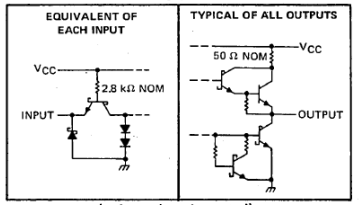
'283



'LS283



'S283



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) 7V
Input voltage: '283, 'S283 . . . . . 5.5V
'LS283 . . . . . 7V
Interemitter voltage (see Note 2) 5.5V
Operating free-air temperature range: SN54283, SN54LS283, SN54S283 . . . . . −55°C to 125°C
SN74283, SN74LS283, SN74S283 . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C

NOTES: 1. Voltage values, except interemitter voltage, are with respect to network ground terminal.
2. This is the voltage between two emitters of a multiple-emitter transistor. This rating applies for the '283 and 'S283 only between the following pairs: A1 and B1, A2 and B2, A3 and B3, A4 and B4.

3 TTL DEVICES

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

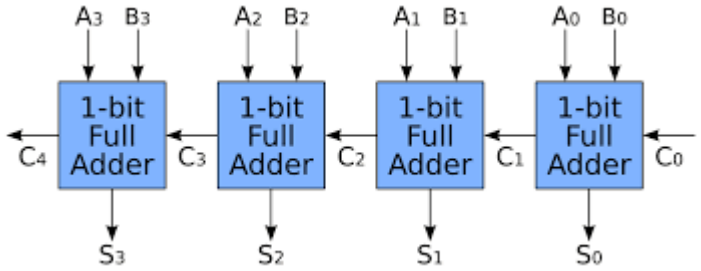
3-918

조합회로

- Gate-Level Modeling
- Data-Flow Modeling

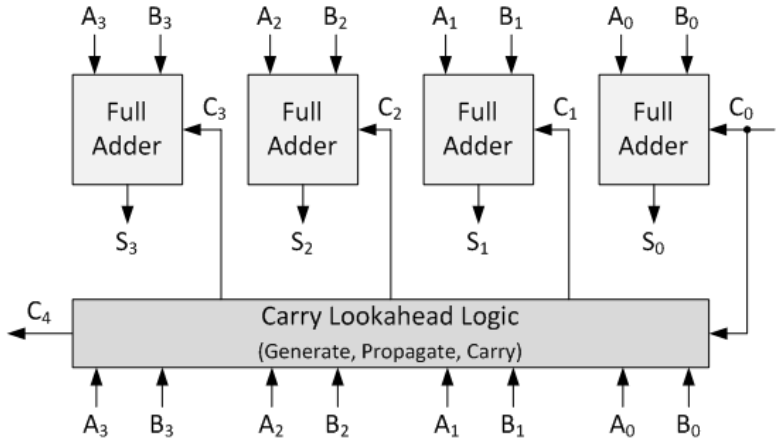
What is the CLA?

< Ripple Carry Adder >



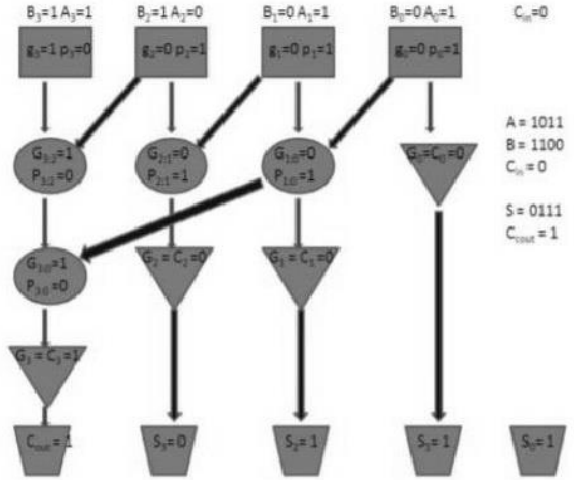
- 구조: 직렬구조
- 속도: 느림
- 면적: 작음

< Carry Lookahead Adder>



- 구조: 병렬계산
- 속도: 빠름
- 면적: 큰 편

< Kogge-Stone Adder >



- 구조: 트리구조
- 속도: 매우 빠름
- 면적: 큼

** RSA, CSA, CLA, CSLA, KSA 모두 처리방식이 달라 속도, 복잡도 다름

What is the CLA?

< Ripple Carry Adder >

$$S_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = (A_i \cdot B_i) + (C_i \cdot (A_i \oplus B_i))$$

- 구조: 직렬구조
- 속도: 느림
- 면적: 작음

< Carry Lookahead Adder>

Generate (G_i): $A_i \cdot B_i$

Propagate (P_i): $A_i \oplus B_i$

$$C_{i+1} = G_i + (P_i \cdot C_i)$$

$$S_i = P_i \oplus C_i$$

- 구조: 병렬계산
- 속도: 빠름
- 면적: 큰 편

< Kogge-Stone Adder >

$$G_i = A_i \cdot B_i$$

$$P_i = A_i \oplus B_i$$

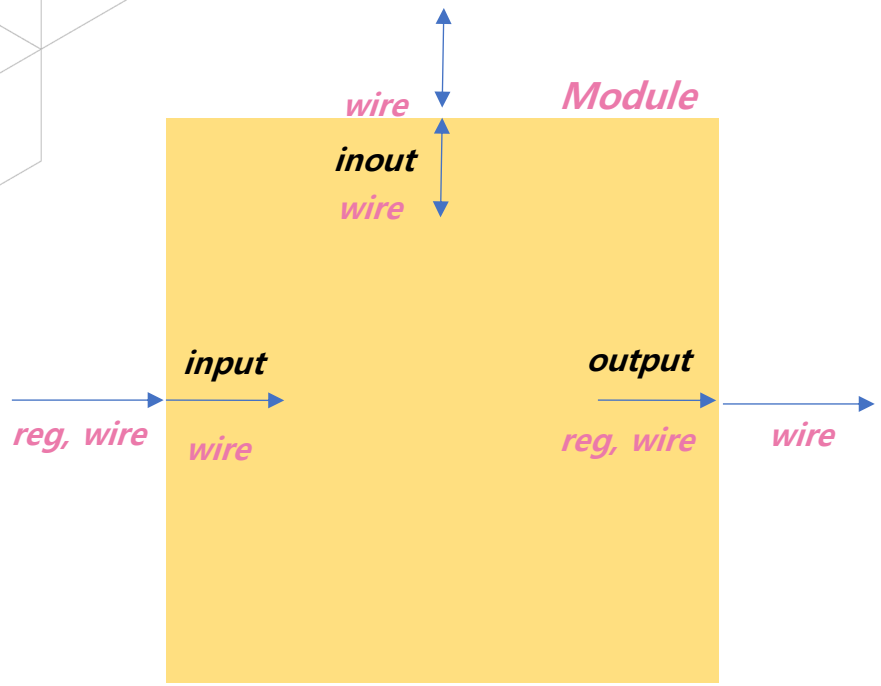
$$G_{i:j} = G_i + (P_i \cdot G_{j-1})$$

$$P_{i:j} = P_i \cdot P_{j-1}$$

$$C_{i+1} = G_{i:0}$$

- 구조: 트리구조
- 속도: 매우 빠름
- 면적: 큼

Port Declaration Principle



모듈을 기준으로,

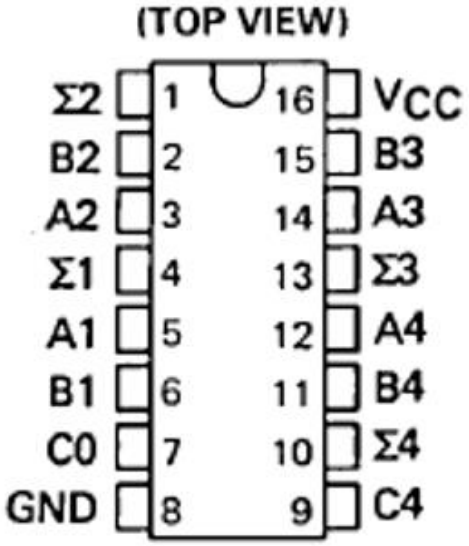
1. input port는 wire
2. output port에 연결되는 신호는 wire
3. inout port는 wire, inout에 연결된 신호도 wire

- **wire**
  - 다른 신호에 의해 지속적으로 구동
  - 하드웨어 요소 또는 모듈 간의 연결

- **reg**
  - 다른 값이 저장될 때까지 값을 유지

2. RTL Design

```
`timescale 1ns / 1ps
// SN74_283: 4-bit Binary Full Adder with Fast Carry

module SN74_283 (A, B, C0, S, C4);

    // Port declarations

    // Internal nets declarations

    // Internal signal Logic

    // Carry Lookahead Logic

    // Final Output Sum

endmodule
```



Generate (G_i): $A_i \cdot B_i$

Propagate (P_i): $A_i \oplus B_i$

$$C_{i+1} = G_i + (P_i \cdot C_i)$$

$$S_i = P_i \oplus C_i$$

모듈과 포트 선언 → 포트의 입출력 정의 + 내부신호 정의 → 메인 로직 기술 → Output 정의

2. RTL Design

Generate (G_i): $A_i \cdot B_i$

Propagate (P_i): $A_i \oplus B_i$

```
// Internal nets declarations
reg [3:0] G, P;

// Internal signal Logic
assign G = A & B;   // Generate
assign P = A ^ B;   // Propagate
```

VS

```
// Internal nets declarations
wire [3:0] G, P;

// Internal signal Logic
assign G = A & B;   // Generate
assign P = A ^ B;   // Propagate
```

둘 중 어떤 것이 맞을까요?

3. Testbench

```
`timescale 1ns / 1ps

module tb_SN74_283();

// Port declarations

// DUT

// stimulus

endmodule
```

Delay	A	B	C0	C4	S
0	0000	0000	0	0	0000
10	0101	0011	0	0	1000
20	1111	0001	0	1	0000
30	1111	1111	1	1	1111
40	1010	0101	1	1	0000

➔ Test case는 많은 영역을 Cover 해야함 (Coverage)

1. 일반 동작 테스트
2. 특이점 테스트

실제로는, 설계자가 직접 모든 Case 기술할 수 없으므로
System-Verilog 기반으로 작성하거나, UVM Framework를 사용

*UVM : Universal Verification Methodology

3. Testbench

```
`timescale 1ns / 1ps

module tb_SN74_283();

    // Port declarations
    reg  [3:0] A, B;
    reg        C0;
    wire [3:0] S;
    wire       C4;

    // DUT
    SN74_283 u0 (
        .A(A), .B(B), .C0(C0), .S(S), .C4(C4)
    );

    // stimulus
    initial begin
        A = 4'b0000; B = 4'b0000; C0 = 1'b0; #10;
        A = 4'b0101; B = 4'b0011; C0 = 1'b0; #10;
        A = 4'b1111; B = 4'b0001; C0 = 1'b0; #10;
        A = 4'b1111; B = 4'b1111; C0 = 1'b1; #10;
        A = 4'b1010; B = 4'b0101; C0 = 1'b1; #10;
        $finish;
    end

endmodule
```

➔ 값을 바꿔주려면 reg로 선언해야 한다!

가져올 모듈

가져온 인스턴스의 이름

포트 연결

Delay

Delay	A	B	C0	C4	S
0	0000	0000	0	0	0000
10	0101	0011	0	0	1000
20	1111	0001	0	1	0000
30	1111	1111	1	1	1111
40	1010	0101	1	1	0000

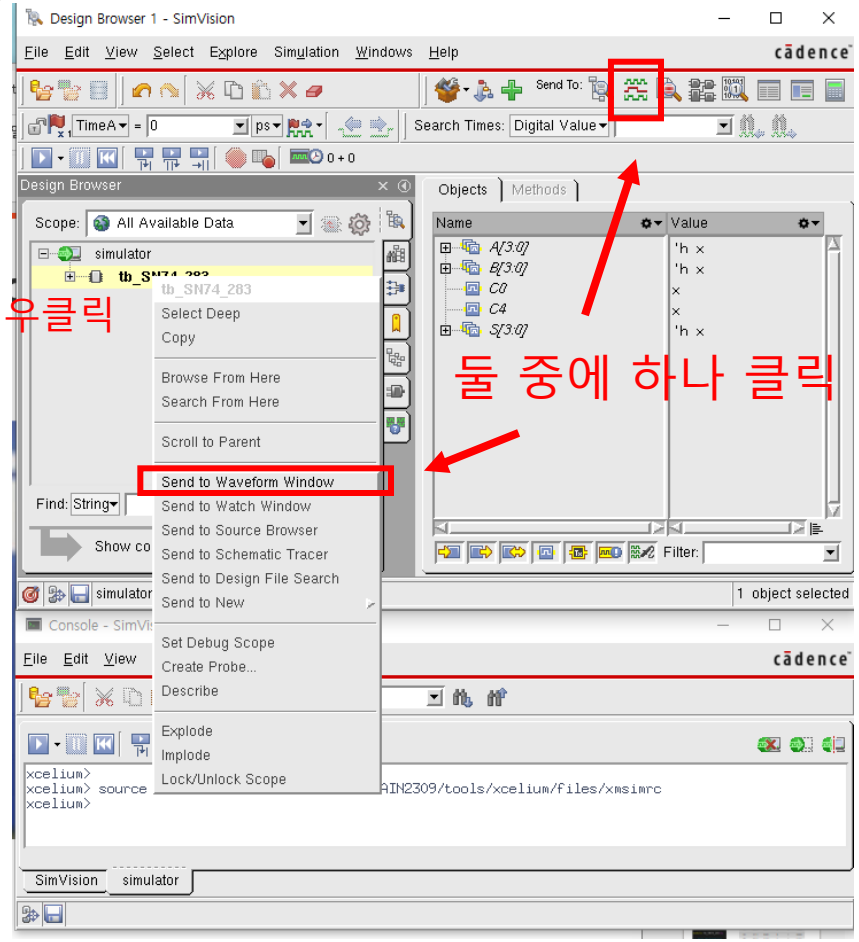
*UVM : Universal Verification Methodology

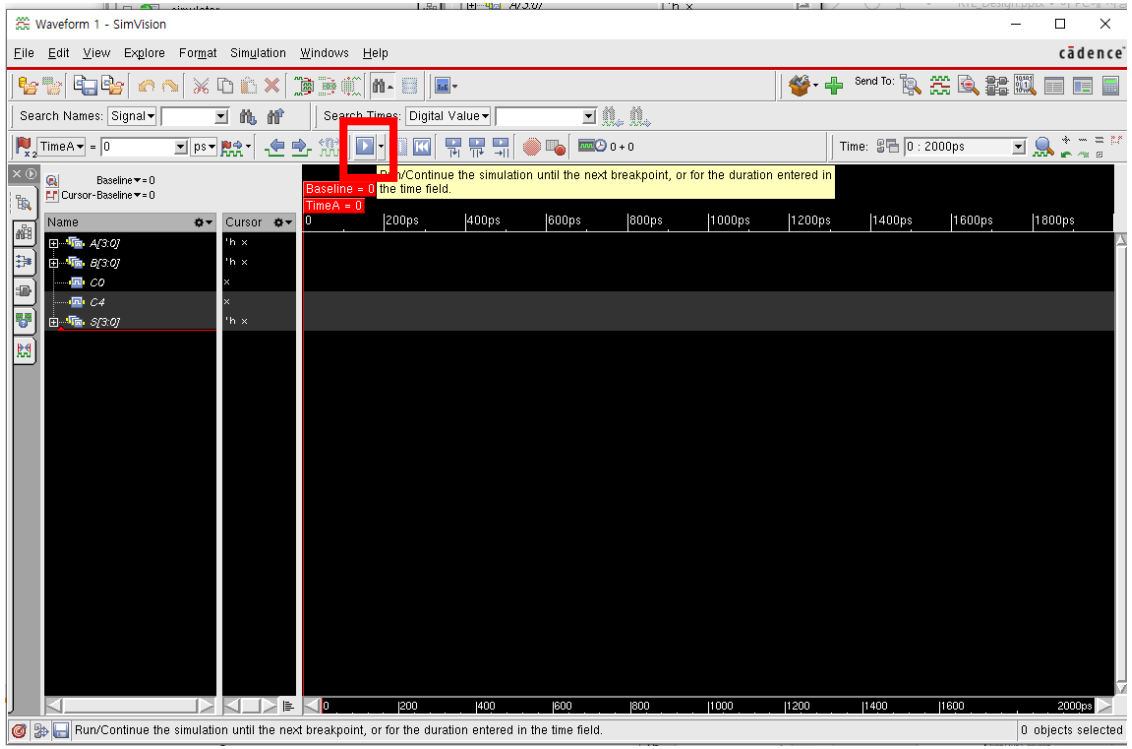
4. Simulation

실행 권한 부여

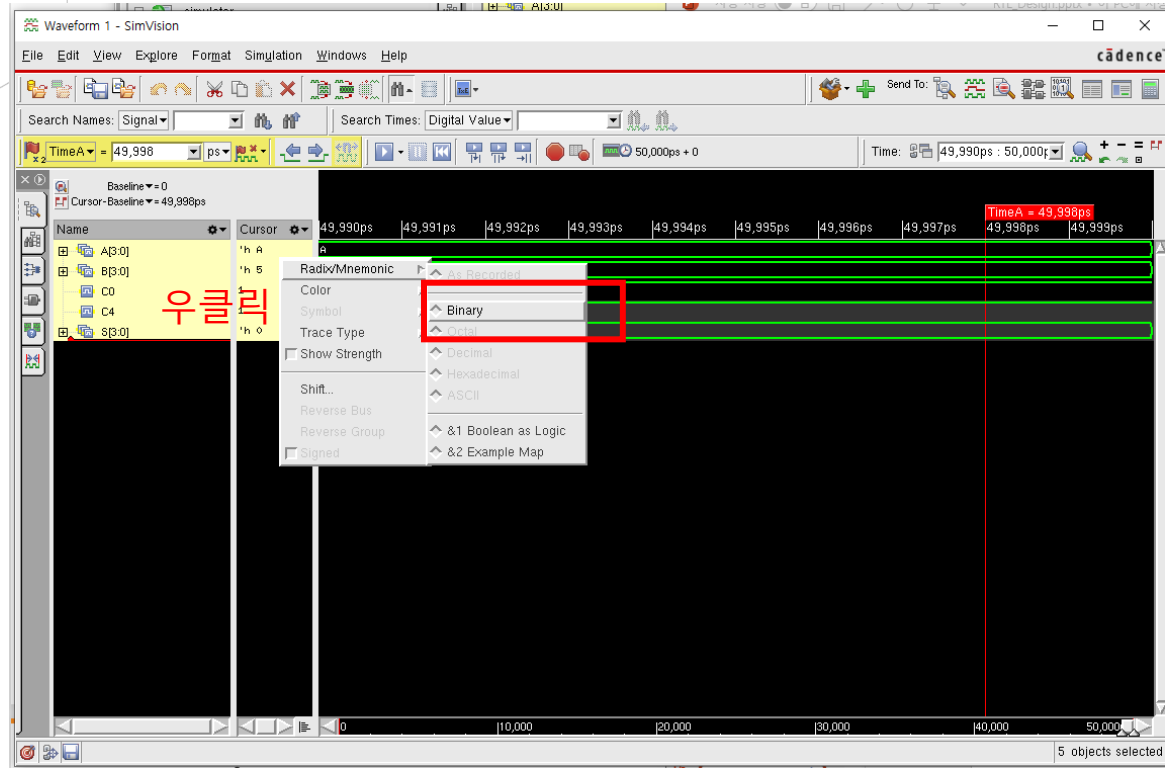
GUI 옵션. 없으면 시뮬레이션 후 종료

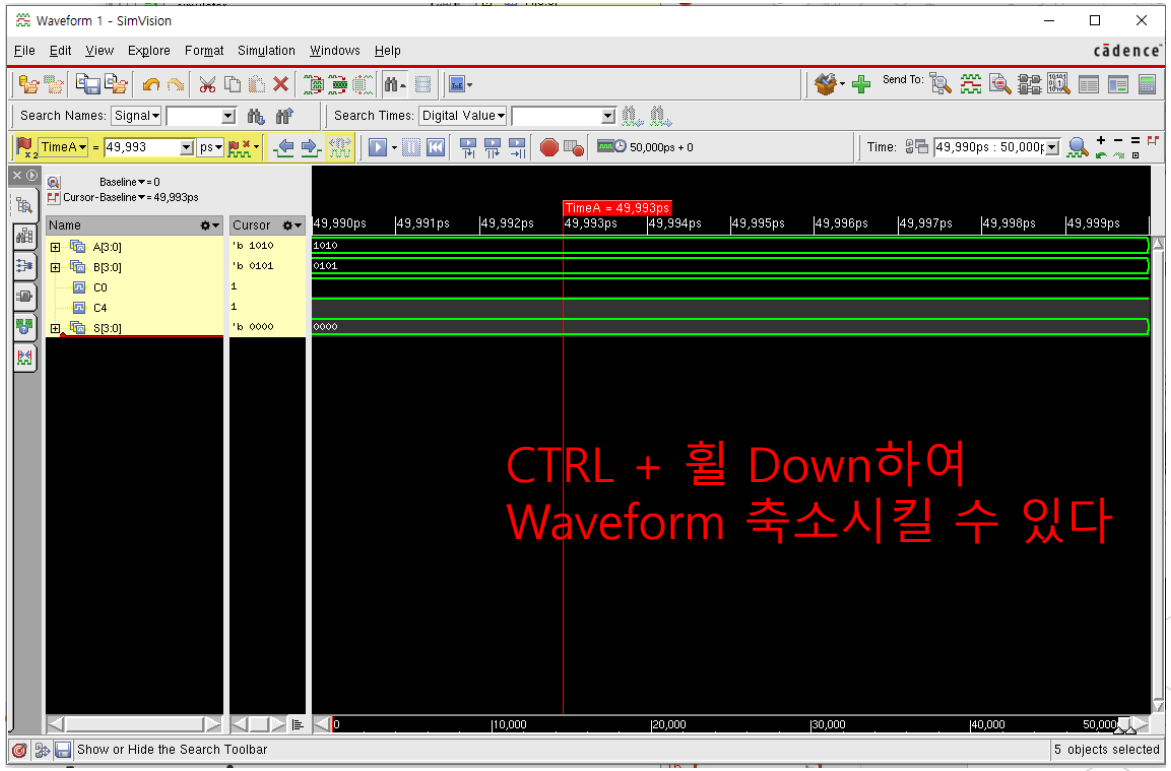
```
[nhcho@npedu-ic-edu LAB1]$ xrun -access +rwc SN74_283.v tb_SN74_283.v -gui
```



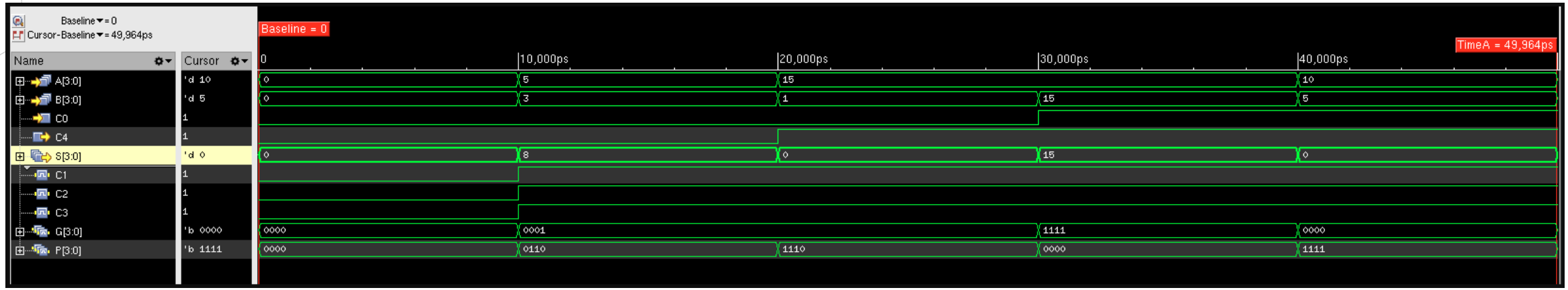


4. Simulation





4. Simulation

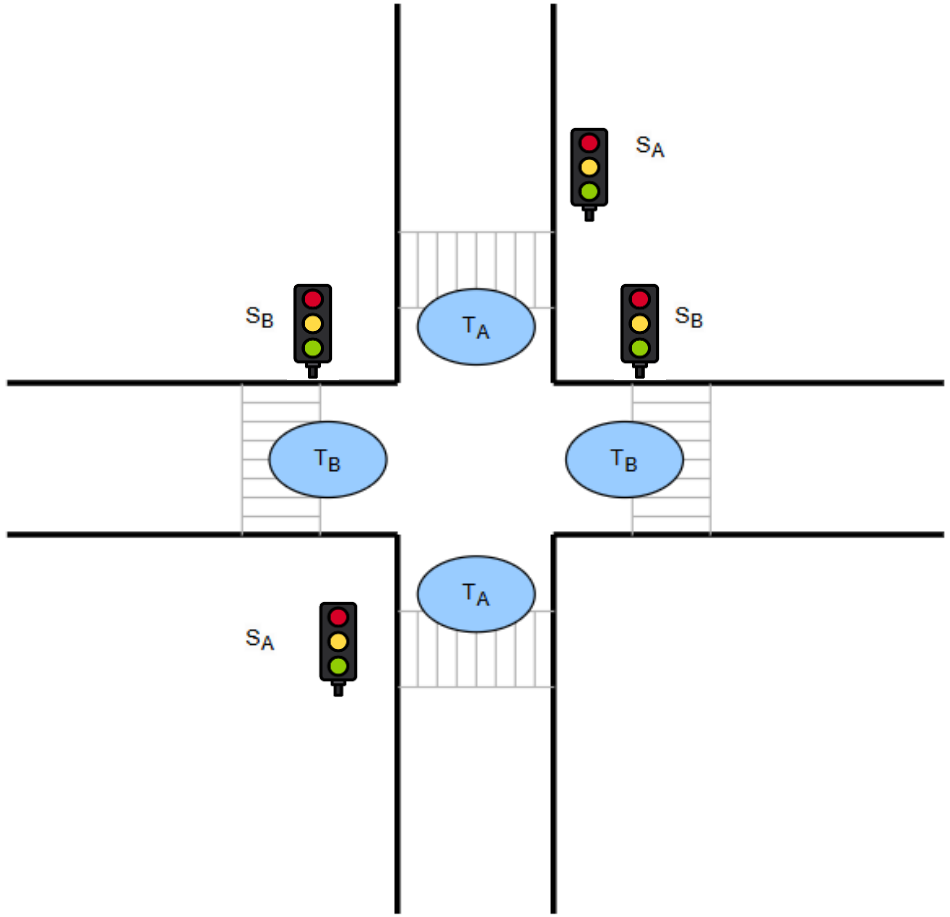


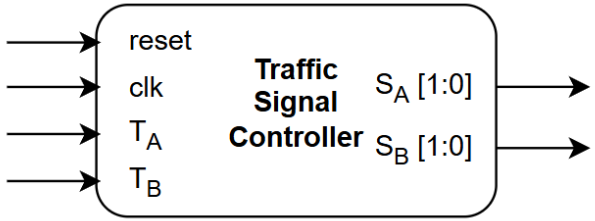
Delay	A	B	C0	C4	S
0	0000	0000	0	0	0000
10	0101	0011	0	0	1000
20	1111	0001	0	1	0000
30	1111	1111	1	1	1111
40	1010	0101	1	1	0000

2. Behavioral Modeling - FSM

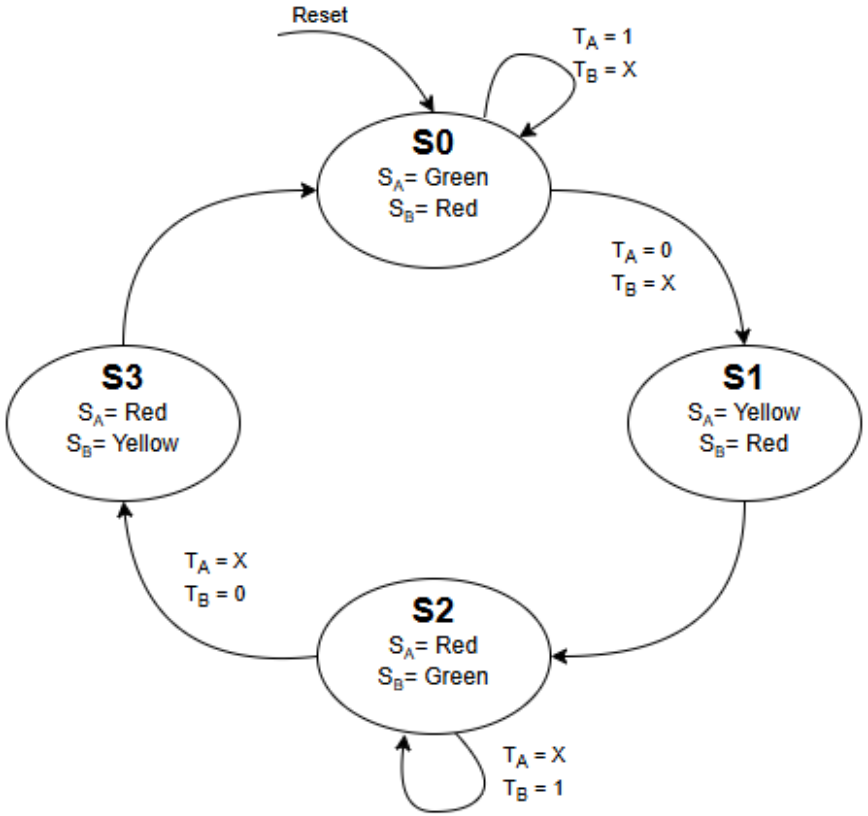
Lab2. Traffic Signal Controller

1. Specification





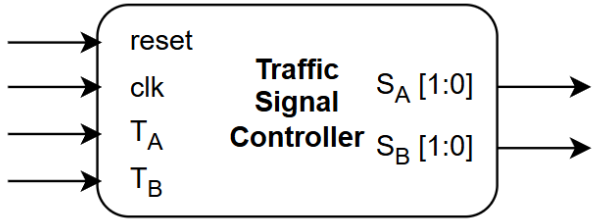
< Block Diagram >

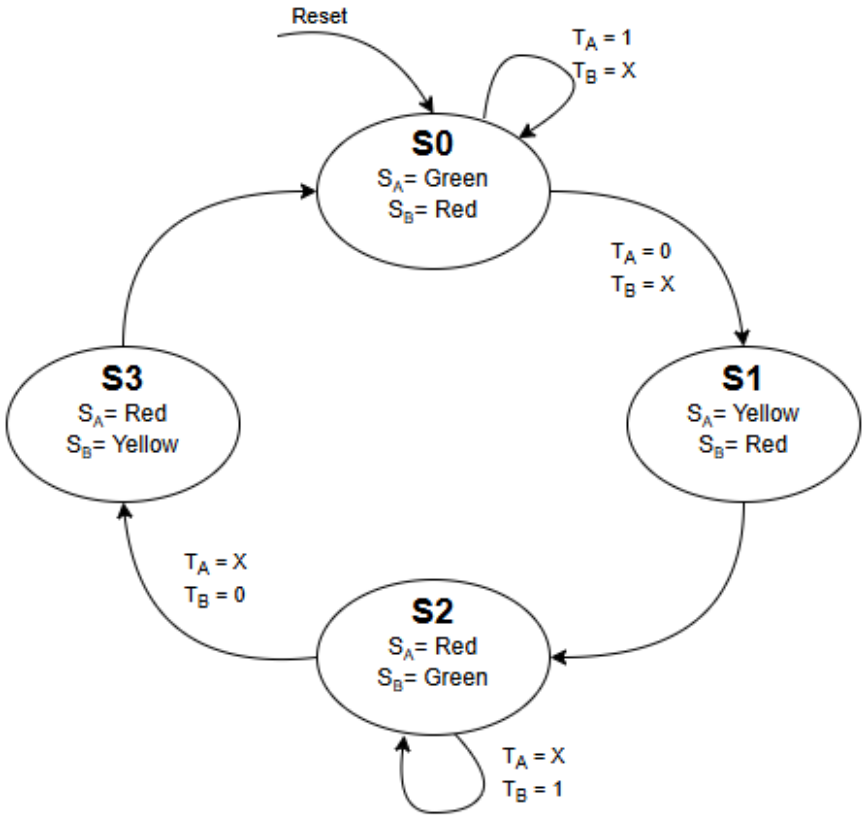


< State Diagram >

2. RTL Design

```
`timescale 1ns / 1ps

module trafficsignal_fsm(clk, reset, i_ta, i_tb, o_sa, o_sb);
        // Port declaration

        // internal variable & parameter
        reg [1:0] state, next_state;

        // state parameter
        parameter S0 = 2'b00;
        parameter S1 = 2'b01;
        parameter S2 = 2'b11;
        parameter S3 = 2'b10;

        // traffic light parameter
        parameter RED    = 2'b00;
        parameter YELLOW = 2'b01;
        parameter GREEN  = 2'b10;

        // status register
        always @(posedge clk or posedge reset) begin
                if (reset) state <= S0;
                else       state <= next_state;
        end

        // next state logic

        // output logic

endmodule
```





3-1. Testbench

```
`timescale 1ns / 1ps

module tb_traffic_signal_fsm( );
  // Port declaration

  // internal parameter
  parameter clk_period = 10;

  // DUT

  // Reset sequence

  // Clock generation

  // Stimulus

endmodule
```

시뮬레이션 결과 저장 파일

시뮬레이션 변수 값 저장

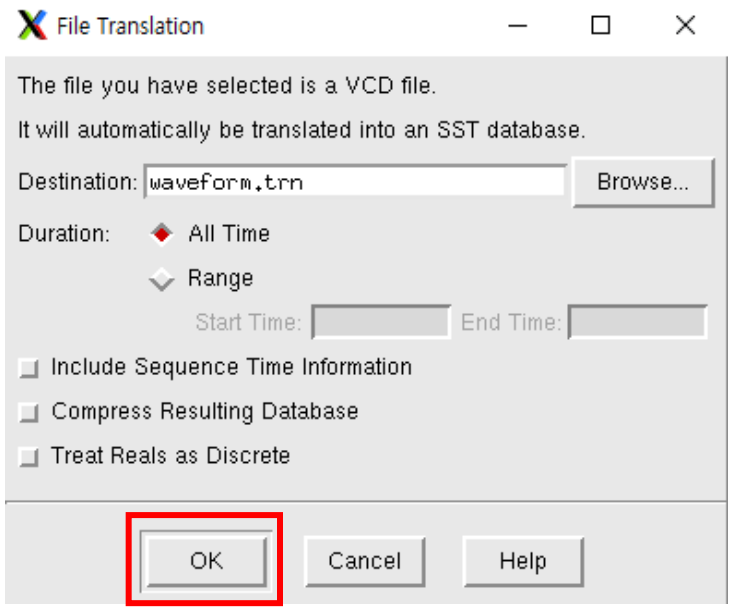
```
// Stimulus
initial begin
    $dumpfile("waveform.vcd"); // dump waveform
    $dumpvars(0, tb_traffic_signal_fsm); // dump variables
    i_ta = 0; i_tb = 0; #3;
    i_ta = 0; i_tb = 1; #(clk_period);
    i_ta = 1; i_tb = 0; #(clk_period);
    i_ta = 0;           #(2 * clk_period);
    i_ta = 0; i_tb = 1; #(5 * clk_period);
    i_ta = 0; i_tb = 0; #(5 * clk_period);
    $finish;
end
```

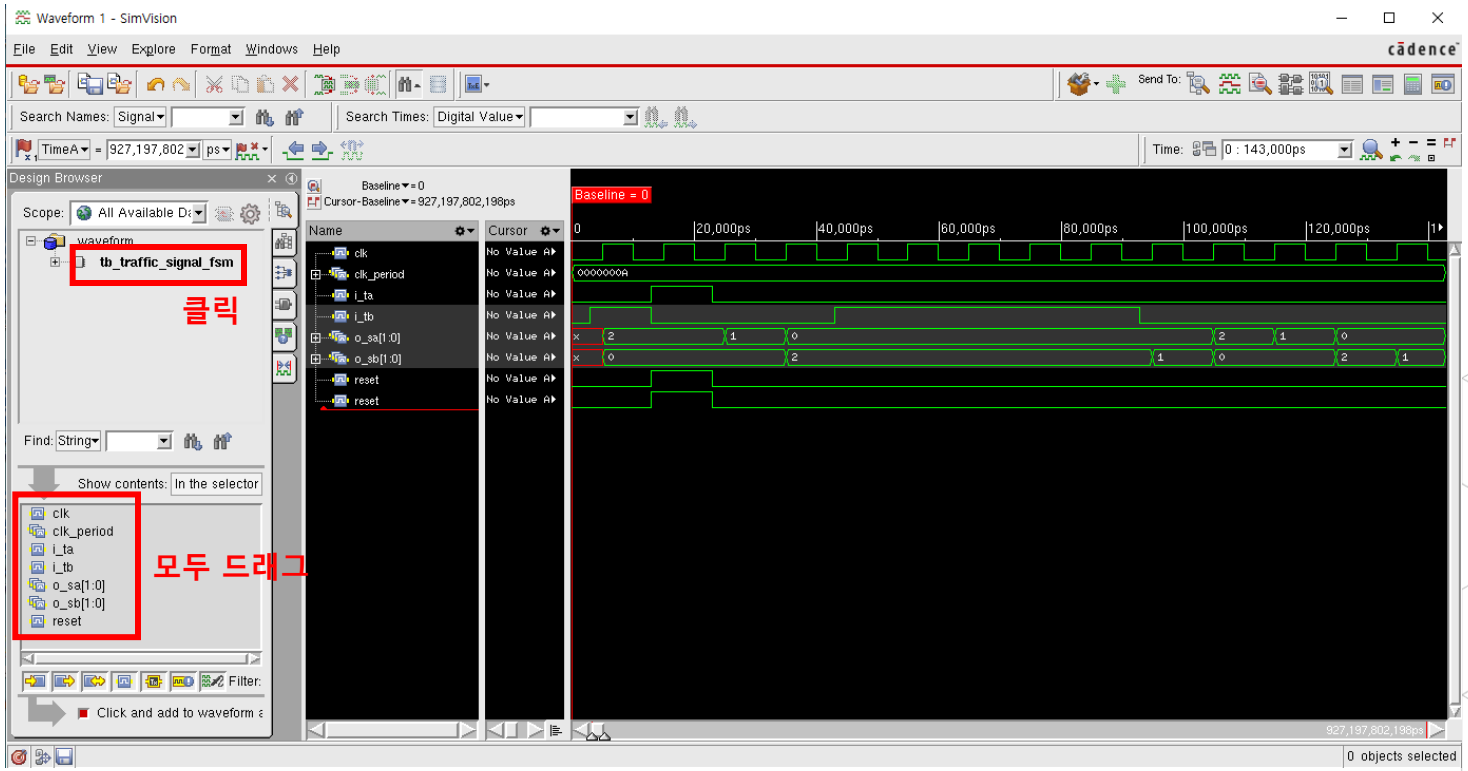
3-2. Simulation

GUI 옵션 없이 실행

```
[nhcho@npedu-ic-edu LAB2]$ xrun -access +rwc *.v
```

```
[nhcho@npedu-ic-edu LAB2]$ ls
tb_traffic_signal_fsm_solution.v  traffic_signal_fsm_solution.v  waveform.vcd  xcelium.d  xrun.history  xrun.key  xrun.log
[nhcho@npedu-ic-edu LAB2]$ simvision waveform.vcd &
```





3-2. Simulation

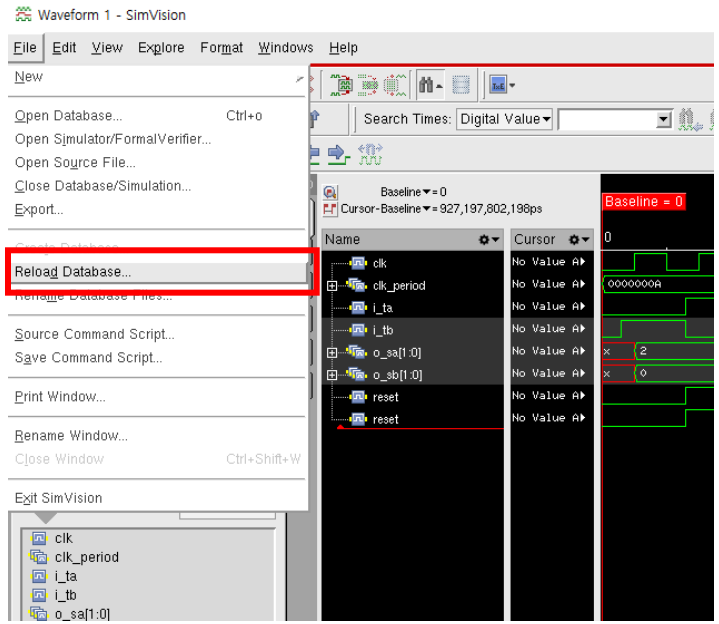
Simvision 백그라운드 실행중

```
[nhcho@npedu-ic-edu LAB2]$ simvision waveform.vcd &
[1] 45742
[nhcho@npedu-ic-edu LAB2]$ simvision(64): 23.09-s013: (c) Copyright 1995-2024 Cadence Design Systems, Inc.
txe(64): 23.09-s013: (c) Copyright 1995-2025 Cadence Design Systems, Inc.

[nhcho@npedu-ic-edu LAB2]$ ls
tb_traffic_signal_fsm_solution.v  traffic_signal_fsm_solution.v  waveform.vcd  xcelium.d  xrun.history  xrun.key  xrun.log
[nhcho@npedu-ic-edu LAB2]$ vim traffic_signal_fsm_solution.v
[nhcho@npedu-ic-edu LAB2]$ vim tb_traffic_signal_fsm_solution.v
```

만약, 코드를 수정했다?

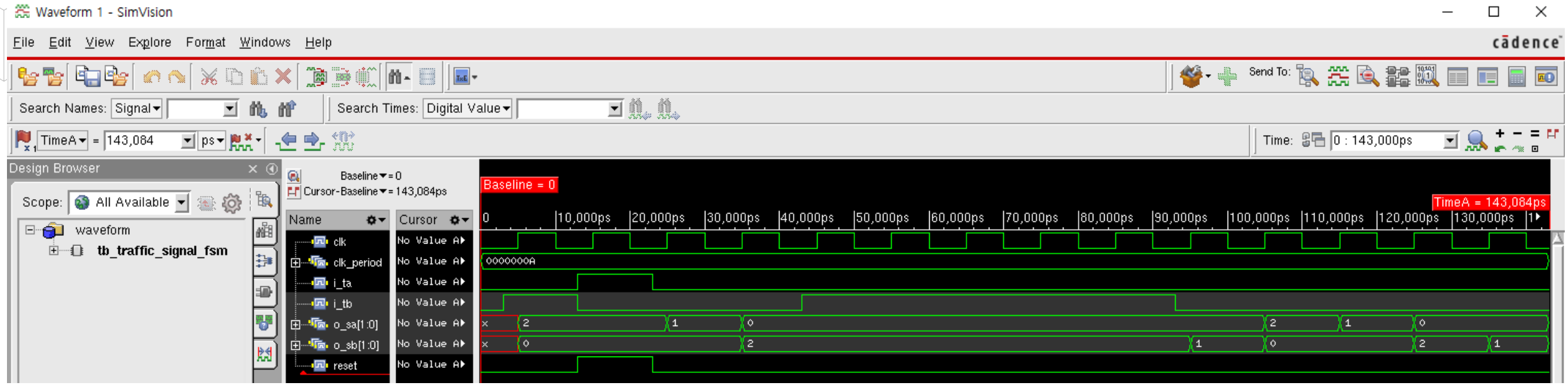
```
[nhcho@npedu-ic-edu LAB2]$ xrun -access +rwc *.v
```

시뮬레이션 재실행



Simvision 내에서
"File – Reload Database"

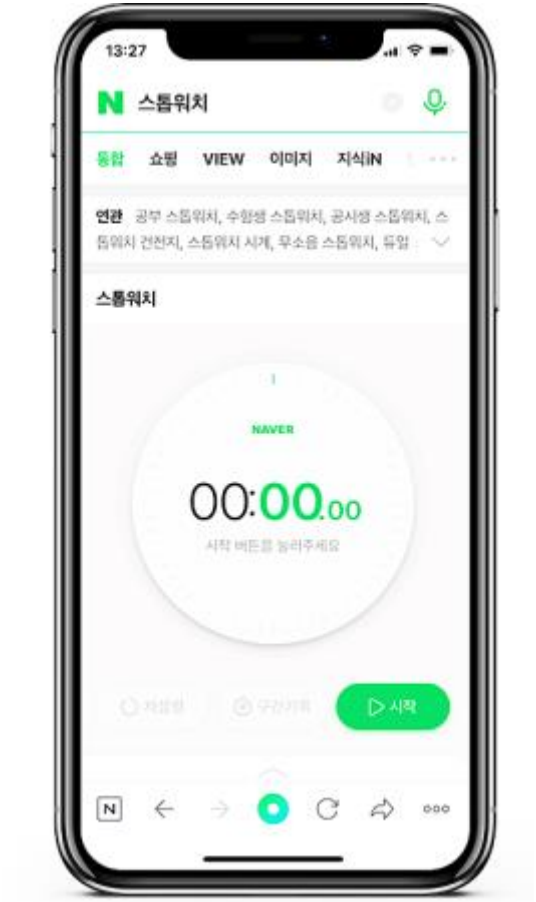
3-2. Simulation



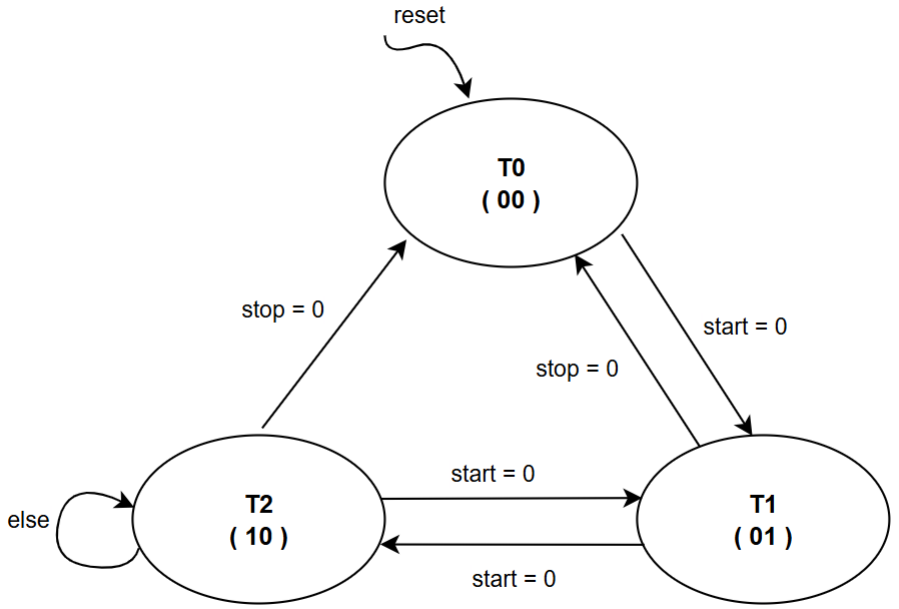
3. Structural Modeling (No Gate-Level Modeling)

Lab3. Stopwatch

1-1. Specification – Spec & State Diagram



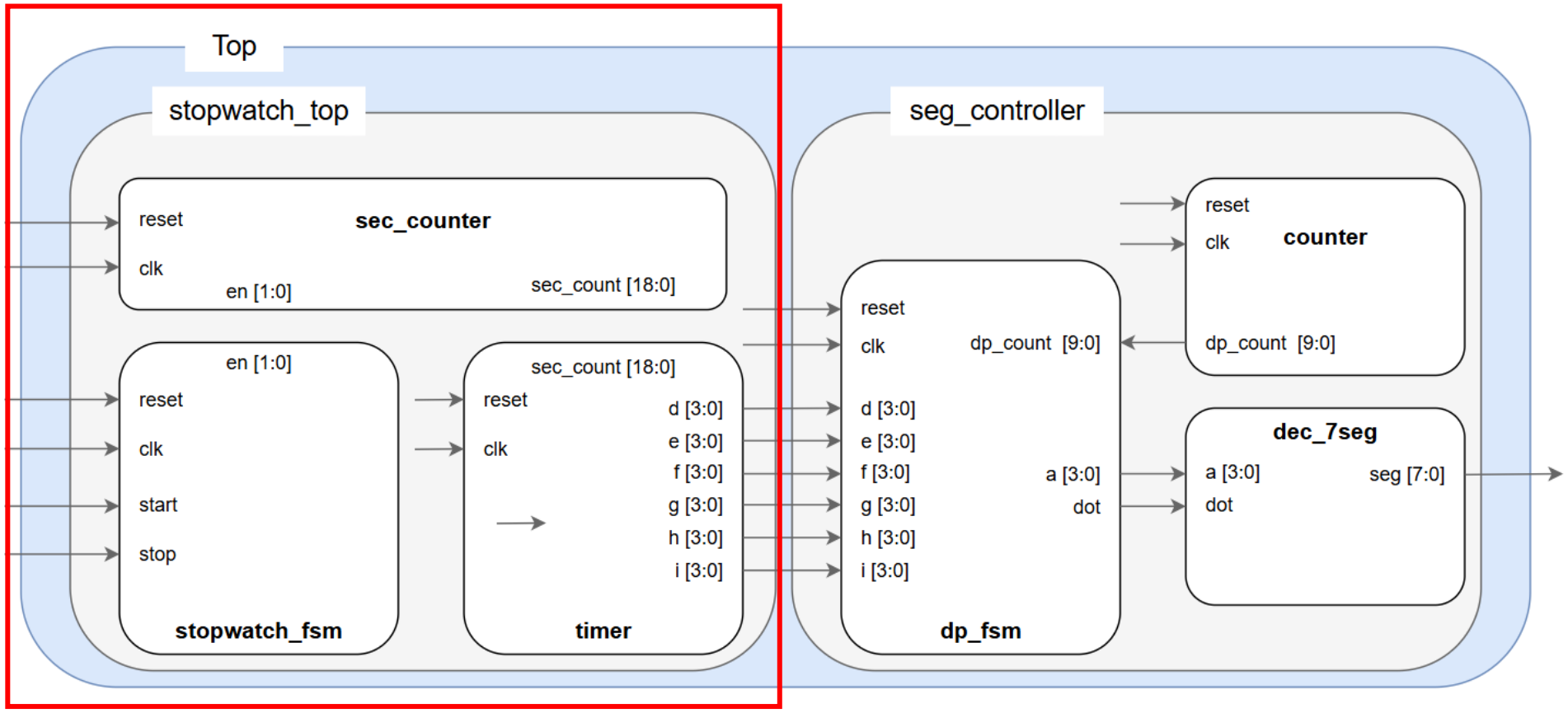
- 시작: 시간 증가
- (증가하던 중에) 시작: 시간 일시정지
- (일시정지 중에) 시작: 시간 증가
- 정지: 0으로 초기화



< State Diagram >

좌측 이미지 출처: https://blog.naver.com/bikebae/222619007852

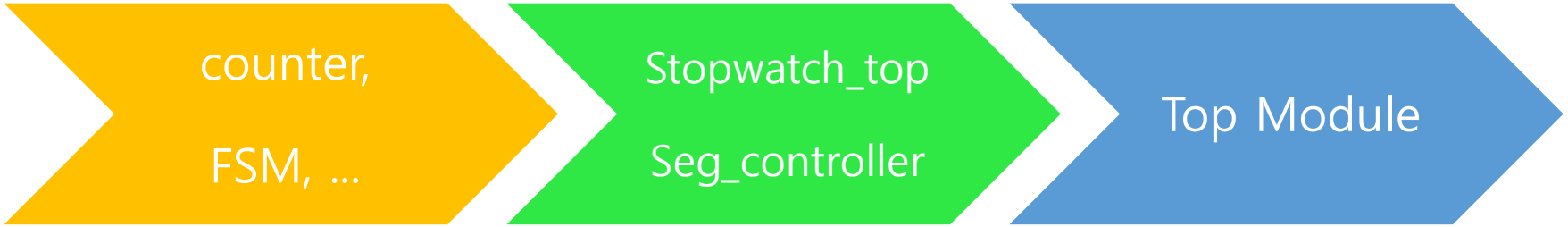
1-2. Specification – Block Diagram



Our Target

Design Methodology

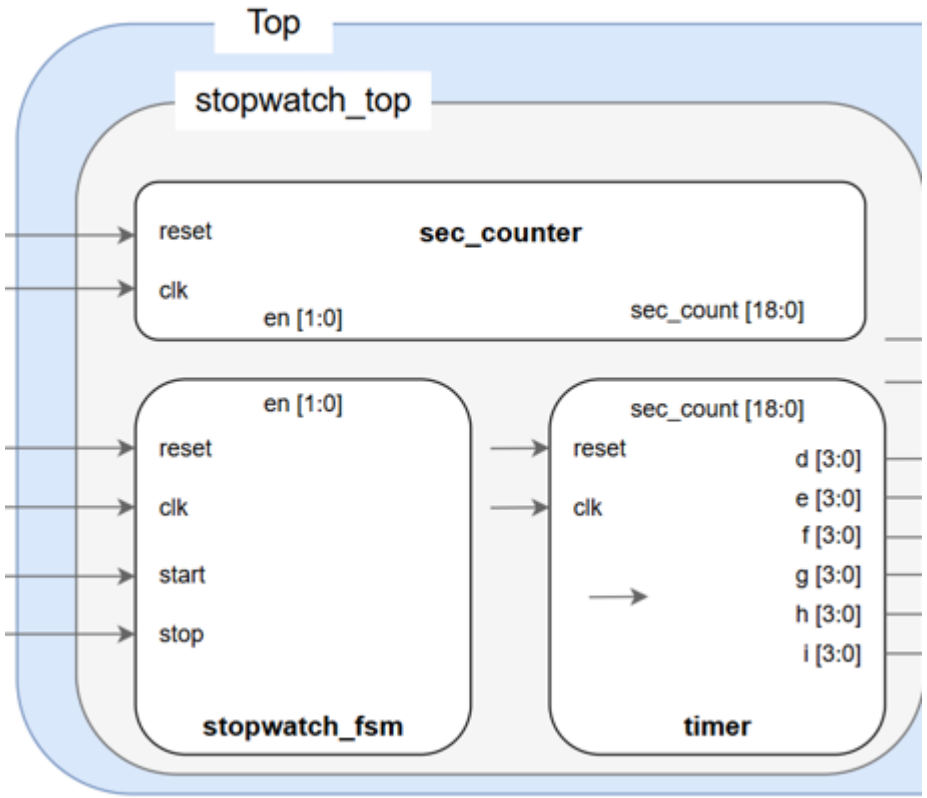
Top to Bottom Methodology



Bottom to Top Methodology

2. RTL Design

```
`timescale 1ns / 1ps

module sec_counter(clk, reset_n, en, sec_count);

    // port declaration

    // Logic
    // initialize counter at 10 clock cycle

endmodule

module timer(clk, reset_n, sec_count, stop, d, e, f, g, h, i);

    // port declaration

    // Logic
    // d~i, initialize g at 6

endmodule


module stopwatch_fsm(clk, reset_n, start, stop, en);

    //port declaration

    reg [1:0] state;
    reg [1:0] nextstate;

    parameter T0 = 2'b00; //base
    parameter T1 = 2'b01; //countup
    parameter T2 = 2'b10; //stop

    // state register
    always @(posedge clk or negedge reset_n) begin
        if (~reset_n) state <= T0;
        else          state <= nextstate;
    end

    // next state logic

    // Output logic


module stopwatch_top(clk, reset_n, start, stop, d, e, f, g, h, i);
    // port declaration

    // Instanciate

endmodule
```



3. Testbench

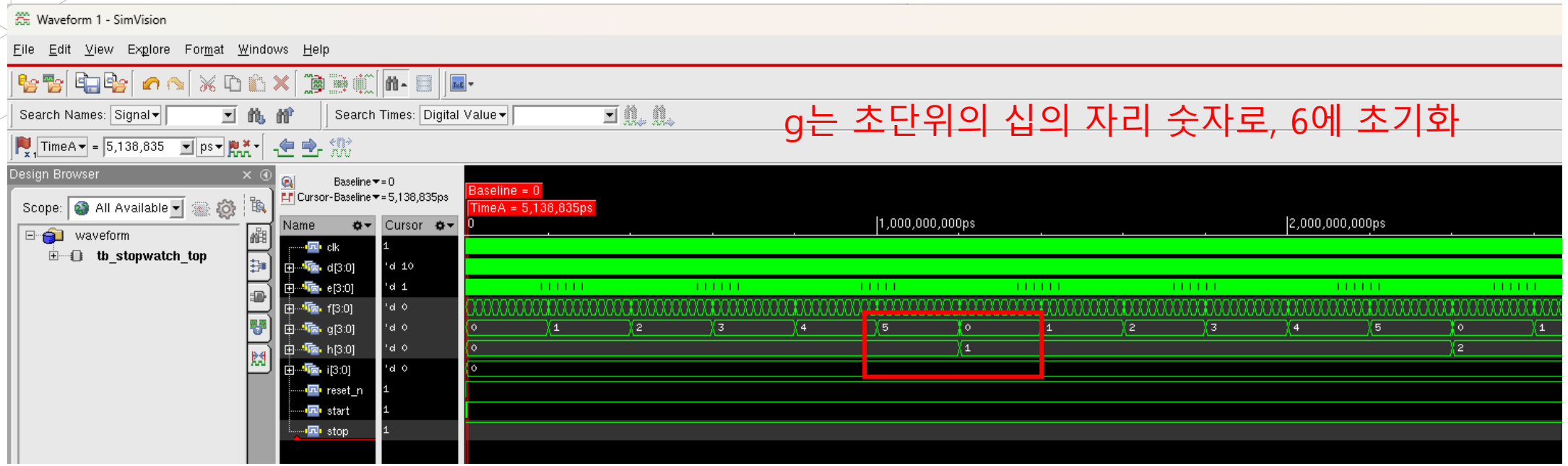
```
`timescale 1ns / 1ps

module tb_stopwatch_top;

    // Port declaration

    // DUT

    // 50MHz Clock Generation (20ns period)

    // Task: press "start" for 1 clock cycle
    task press_start;
    begin

    end
    endtask

    // Task: press "stop" for 1 clock cycle
    task press_stop;
    begin

    end
    endtask

    // Stimulus
    initial begin
        // initialize

        // [1] Press start (T0 -> T1)

        // [2] Press start (T1 -> T2)

        // [3] Press start (T2 -> T1)

        // [4] Press stop (T0)

    end

    // Monitoring
    initial begin
        // dump waveform
        // dump variables
        // monitoring
    end
endmodule
```

반복 작업을 함수로 작성

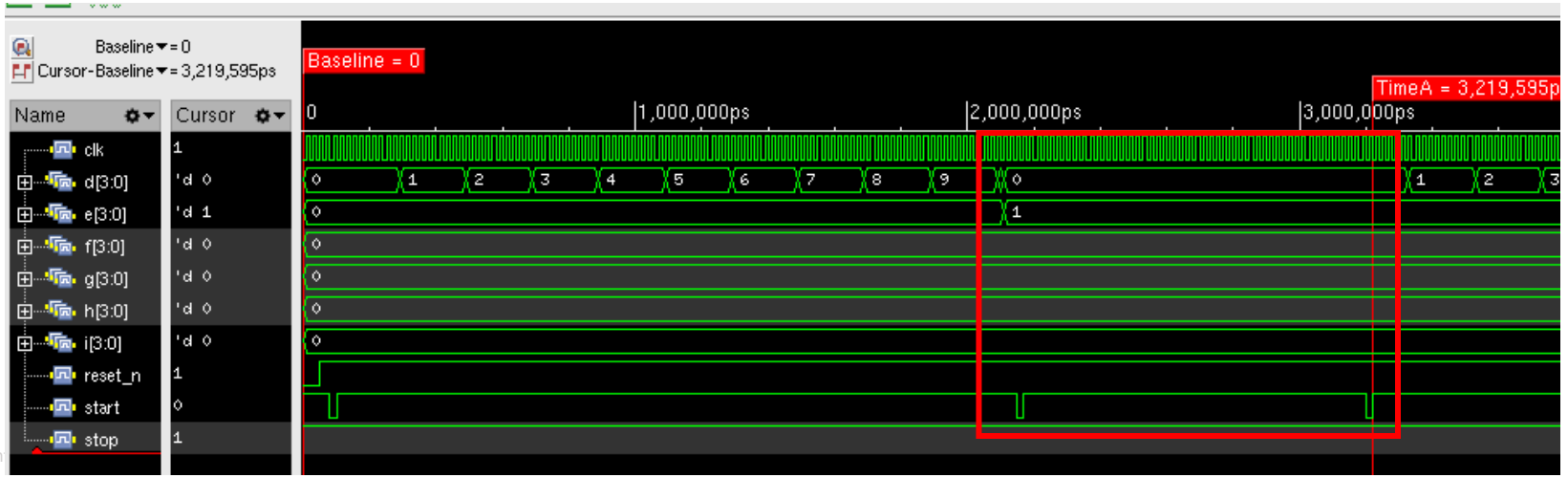
```
    // Monitoring
    initial begin
        $dumpfile("waveform.vcd"); // dump waveform
        $dumpvars(0, tb_stopwatch_top); // dump variables
        $display("    TIME     d e f g h i   start   stop"); // monitoring
        $monitor("%8d:   %d %d %d %d %d %d     %b     %b",
            $time, d, e, f, g, h, i, start, stop);
    end
```

```
xcelium> run
    TIME     d  e  f  g  h  i   start   stop
       0:    0  0  0  0  0  0     1     1
      80:    0  0  0  0  0  0     0     1
     100:    0  0  0  0  0  0     1     1
     290:    1  0  0  0  0  0     1     1
     490:    2  0  0  0  0  0     1     1
     690:    3  0  0  0  0  0     1     1
     890:    4  0  0  0  0  0     1     1
    1090:    5  0  0  0  0  0     1     1
    1290:    6  0  0  0  0  0     1     1
    1490:    7  0  0  0  0  0     1     1
    1690:    8  0  0  0  0  0     1     1
    1890:    9  0  0  0  0  0     1     1
    2090:   10  0  0  0  0  0     1     1
    2100:   10  0  0  0  0  0     0     1
    2110:    0  1  0  0  0  0     0     1
    2120:    0  1  0  0  0  0     1     1
    4150:    0  1  0  0  0  0     0     1
    4170:    0  1  0  0  0  0     1     1
    4350:    1  1  0  0  0  0     1     1
    4550:    2  1  0  0  0  0     1     1
    4750:    3  1  0  0  0  0     1     1
    4950:    4  1  0  0  0  0     1     1
    5150:    5  1  0  0  0  0     1     1
    5350:    6  1  0  0  0  0     1     1
    5550:    7  1  0  0  0  0     1     1
    5750:    8  1  0  0  0  0     1     1
    5950:    9  1  0  0  0  0     1     1
    6150:   10  1  0  0  0  0     1     1
    6170:    0  2  0  0  0  0     1     1
    6350:    1  2  0  0  0  0     1     1
    6550:    2  2  0  0  0  0     1     1
    6750:    3  2  0  0  0  0     1     1
```

4. Simulation



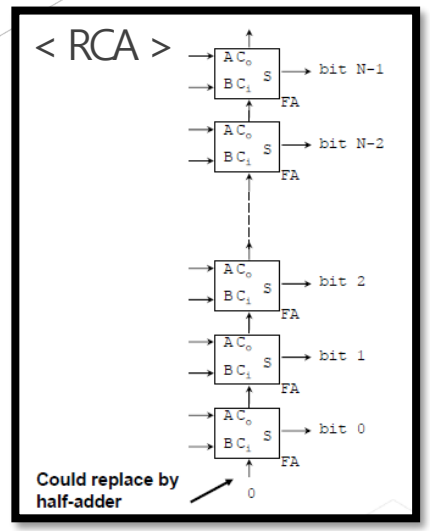
g는 초단위의 십의 자리 숫자로, 6에 초기화

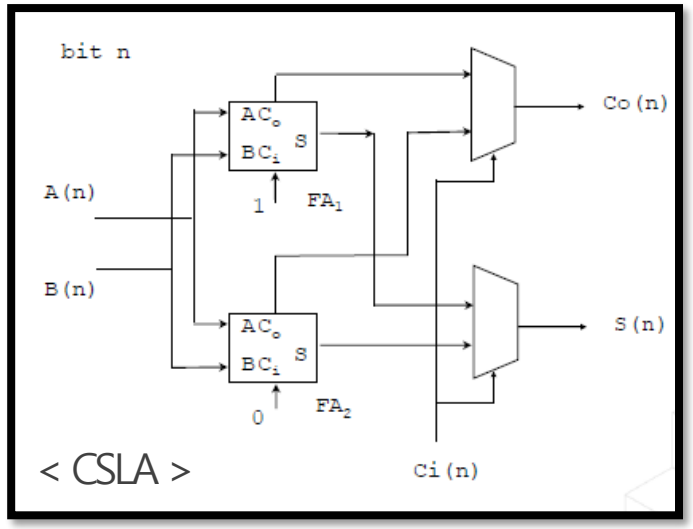


Start 다시 눌렸을 때 일시정지 확인

RTL 설계에서 Verilog Coding이 전부일까요?

Arithmetic Building Block - Adders






```
// Full Adder
module FA(output sum, cout, input a, b, cin);
  wire w0, w1, w2;

  xor (w0, a, b);
  xor (sum, w0, cin);

  and (w1, w0, cin);
  and (w2, a, b);
  or  (cout, w1, w2);
endmodule

// Ripple Carry Adder - 8 bits
module RCA8(output [7:0] sum, output cout, input [7:0] a, b, input clk);
  wire [7:1] c;

  FA fa0(sum[0], c[1], a[0], b[0], 1'b0);
  FA fa[6:1](sum[6:1], c[7:2], a[6:1], b[6:1], c[6:1]);
  FA fa7(sum[7], cout, a[7], b[7], c[7]);
endmodule
```

```
// Full Adder
module FA(output sum, cout, input a, b, cin);
  wire w0, w1, w2;
  xor (w0, a, b);
  xor (sum, w0, cin);
  and (w1, w0, cin);
  and (w2, a, b);
  or (cout, w1, w2);
endmodule

// 4-bit Ripple Carry Adder
module RCA4(output [3:0] sum, output cout, input [3:0] a, b, input cin);
  wire [2:0] c;
  FA fa0(sum[0], c[0], a[0], b[0], cin);
  FA fa1(sum[1], c[1], a[1], b[1], c[0]);
  FA fa2(sum[2], c[2], a[2], b[2], c[1]);
  FA fa3(sum[3], cout, a[3], b[3], c[2]);
endmodule

// 1-bit 2:1 MUX
module MUX2to1_w1(output y, input i0, i1, s);
  assign y = s ? i1 : i0;
endmodule

// 4-bit 2:1 MUX
module MUX2to1_w4(output [3:0] y, input [3:0] i0, i1, input s);
  assign y = s ? i1 : i0;
endmodule

// 8-bit Carry Select Adder
module CSelA8(output [7:0] sum, output cout, input [7:0] a, b, input clk);
  wire [3:0] sum0, sum1, sum4_0, sum4_1;
  wire cout0_0, cout0_1, cout1_0, cout1_1;
  wire c1;

  RCA4 rca0_0(sum0, cout0_0, a[3:0], b[3:0], 1'b0);
  RCA4 rca0_1(sum1, cout0_1, a[3:0], b[3:0], 1'b1);
  MUX2to1_w4 mux0_sum(sum[3:0], sum0, sum1, 1'b0);
  MUX2to1_w1 mux0_cout(c1, cout0_0, cout0_1, 1'b0);

  RCA4 rca1_0(sum4_0, cout1_0, a[7:4], b[7:4], 1'b0);
  RCA4 rca1_1(sum4_1, cout1_1, a[7:4], b[7:4], 1'b1);
  MUX2to1_w4 mux1_sum(sum[7:4], sum4_0, sum4_1, c1);
  MUX2to1_w1 mux1_cout(cout, cout1_0, cout1_1, c1);
endmodule
```

출처: https://github.com/ppashakhanloo/verilog-adders

Synthesis Report of Adders

< RCA >

```
Path 1: MET (3243 ps) Late Ext
          Group: clk
     Startpoint: (R) b[1]
          Clock: (R) clk
       Endpoint: (R) sum[7]
          Clock: (R) clk

                     Capture
       Clock Edge:+   10000
        Drv Adjust:+      0
       Src Latency:+      0
       Net Latency:+      0 (
           Arrival:=  10000

     Output Delay:-    2500
    Required Time:=    7500
     Launch Clock:-       0
      Input Delay:-    2500
        Data Path:-    1757
            Slack:=    3243
```

```
Cell-Area   Net-Area    Total-Area
-------------------------------------
  542.203      0.000       542.203
```

< CSLA >

```
Path 1: MET (3694 ps) Late Ext
          Group: clk
     Startpoint: (R) b[1]
          Clock: (R) clk
       Endpoint: (F) cout
          Clock: (R) clk

                     Capture
       Clock Edge:+   10000
        Drv Adjust:+      0
       Src Latency:+      0
       Net Latency:+      0 (
           Arrival:=  10000

     Output Delay:-    2500
    Required Time:=    7500
     Launch Clock:-       0
      Input Delay:-    2500
        Data Path:-    1306
            Slack:=    3694
```

```
Cell-Area   Net-Area    Total-Area
-------------------------------------
  728.482      0.000       728.482
```

< KSA >

```
Path 1: MET (3789 ps) Late Ext
          Group: clk
     Startpoint: (R) b[4]
          Clock: (R) clk
       Endpoint: (R) sum[7]
          Clock: (R) clk

                     Capture
       Clock Edge:+   10000
        Drv Adjust:+      0
       Src Latency:+      0
       Net Latency:+      0 (
           Arrival:=  10000

     Output Delay:-    2500
    Required Time:=    7500
     Launch Clock:-       0
      Input Delay:-    2500
        Data Path:-    1211
            Slack:=    3789
```

```
Cell-Area   Net-Area    Total-Area
-------------------------------------
  695.218      0.000       695.218
```

Path

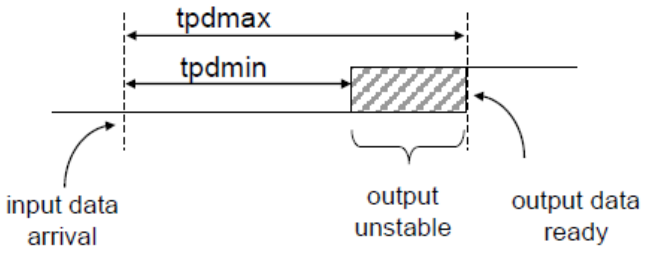
RCA > CSLA > KSA

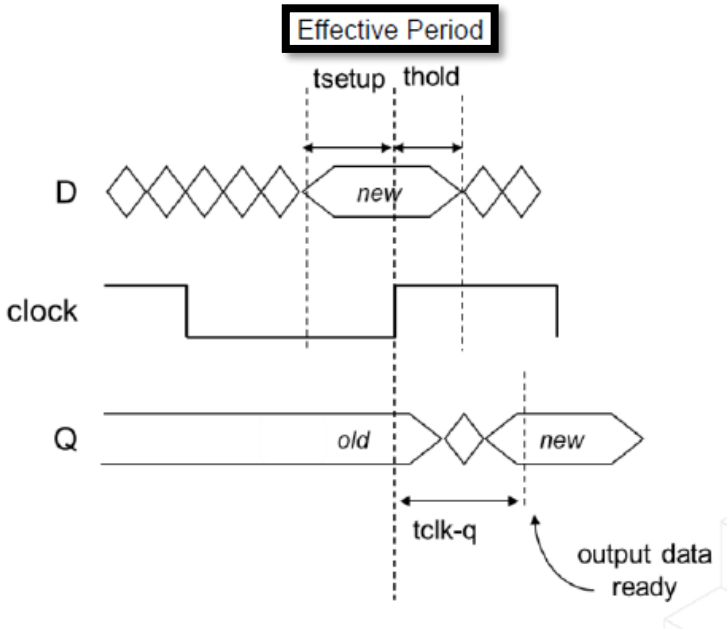
Area

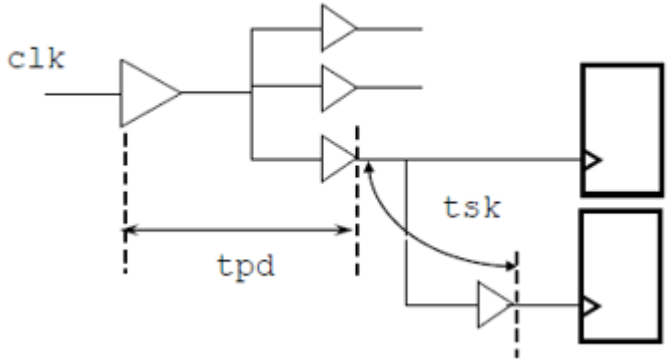
CSLA > KSA > RCA

그렇다면, 무조건 빠른 회로만 필요할까요?

Propagation delay & Timing





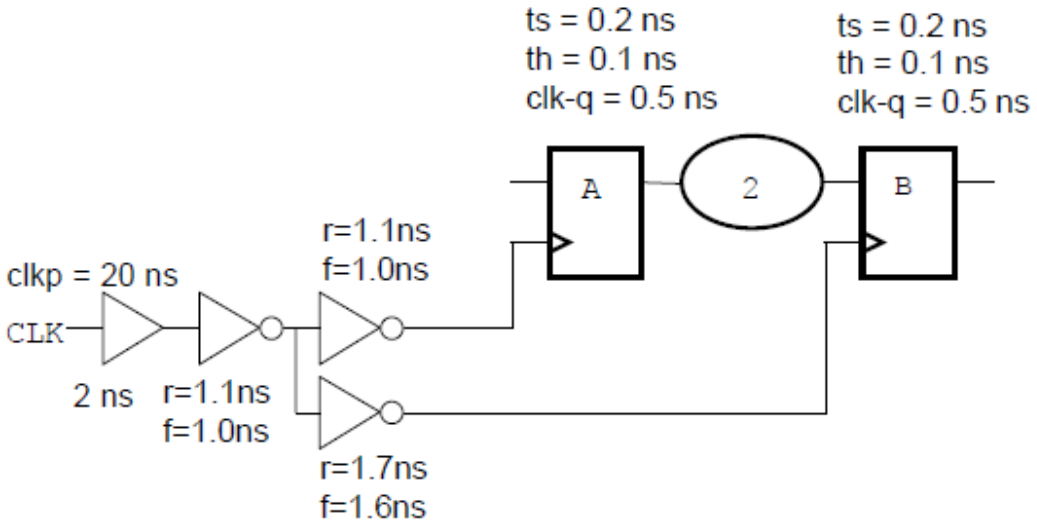


- 가장 짧은 경로 tpdmin
- 가장 긴 경로 tpdmax

- Edge부터 Data 지연 tclk - q
- Clock 기준으로 setup 전에 준비, hold 까지 유지 되어야 한다

- Clock tree에서 tpd, tsk 파라미터를 고려
- Positive skew는 hold time에, Negative skew는 setup time에 영향

Is delay always a bad thing?



Rising edge clock propagation delay at A = 2 + 1.1 + 1.0 = 4.1ns
Rising edge clock propagation delay at B = 2 + 1.1 + 1.6 = 4.7ns

Clock skew at B with respect to A = +0.6ns (Positive Skew)
Effective clock period = 20 + 0.6 = 20.6ns

Second Clock of B arrives at 24.7ns (4.7 + 20)
But, It must arrive at 24.5ns because of Setup time.

Data released time from A = 4.1 + 0.5 = 4.6ns

Maximum delay = 24.5 – 4.6 = 19.9ns

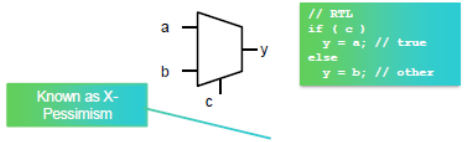
First Clock of B hold time = 4.7 + 0.1 = 4.8 ns
Data released time from A cannot arrive at B 4.8ns

Minimum delay = 4.8 – 4.6 = 0.2 ns

반드시 A,B 사이에 Logic 필요

Functional Design Challenges

X-Optimism Simulation Solution




```
// RTL
if ( c )
  y = a; // true
else
  y = b; // other
```

Known as X-Pessimism

Known as X-Optimism

Default

If the condition is unknown, assign the result as stated.

Only FOX guarantees that the simulation has no *new* unknowns.

Forward-Only-X (FOX)

If the condition is unknown, assign the result unknown.

Compute-As-Ternary (CAT)

If the condition is unknown, assign the result as the hardware would.

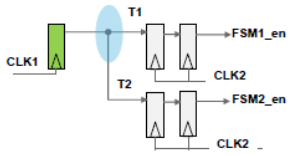
```
// Default
a b c : y
0 0 x : 0
0 1 x : 1
1 0 x : 0
1 1 x : 1
```

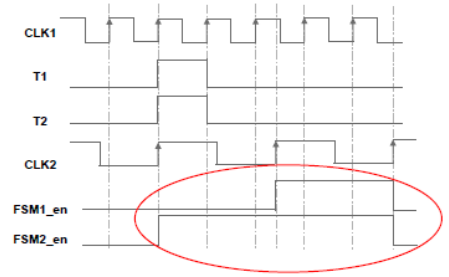
```
// FOX
a b c : y
0 0 x : x
0 1 x : x
1 0 x : x
1 1 x : x
```

```
// CAT
a b c : y
0 0 x : 0
0 1 x : x
1 0 x : x
1 1 x : 1
```


cadence

Divergence of CDC Signal





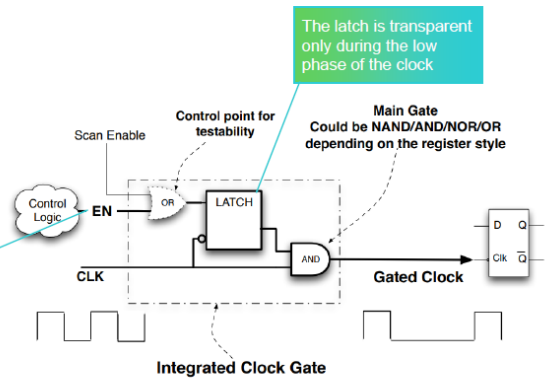
- A divergent logic style to multiple synchronization paths runs the risk of causing functional errors.
- Due to the propagation delay and different metastable settling times, the FSM1_en and FSM2_en could start at different times.
- This type of structure should be avoided by fanning out a single FSM enabled after synchronization to both FSMs.

cadence

What Is Clock Gating?

A technique used in synchronous circuits to reduce dynamic power by removing the clock when it is not essential to design intent functionality.

- Effectively, we prune the clock tree to save power.
- However, remember the PPA tradeoff?
- This is at the expense of more gates.



Design Block Enable Signal generated from our Control Logic

cadence

Power Reduction Technique	Leakage Power	Dynamic Power	Timing	Area Penalty	Methodology Impact	Methodology Change
Low-Power Optimization	10%	10%	0%	10%	None	None
Multi-Vt	6X	0%	0%	0%	Low	Multi-Vt library needed
Clock Gating	0%	20%	0%	<2%	Low	Clock-gating cells needed and extra overhead in STA
Multi-Supply Voltage (MSV)	2X	40-50%	0%	<10%	Medium	Micro-architecture and methodology needs to be domain aware; need voltage regulators and level shifters; verification and analysis challenge
Power Shut-Off (PSO)	10-50X	0%	4-8%	5-15%	Medium-High	Insertion of switch cells; retention flops; wake-up and shut-down time analysis; power shut off and restore verification
Dynamic Voltage Frequency Scaling (DVFS)	2-3X	40-70%	0%	<10%	High	Deterministic scheduling; multi-mode optimization and analysis flow needed; clock synchronization
Substrate Biasing	10X	0%	10%	<10%	High	Maintain well separation; multiple power rail distribution; static timing analysis

Thank you.

조남현 선임연구원

EDA | IC

Tel. 010-3219-3897 | **E-mail.** nhcho@npit.co.kr

Web. www.npit.co.kr