



순차논리회로

# 논리회로 실습

부경대 컴퓨터·인공지능공학부 최필주

- 순차논리회로 개요
- D Flip-flop
- Counter
- Dot-matrix

# 순차논리회로 개요

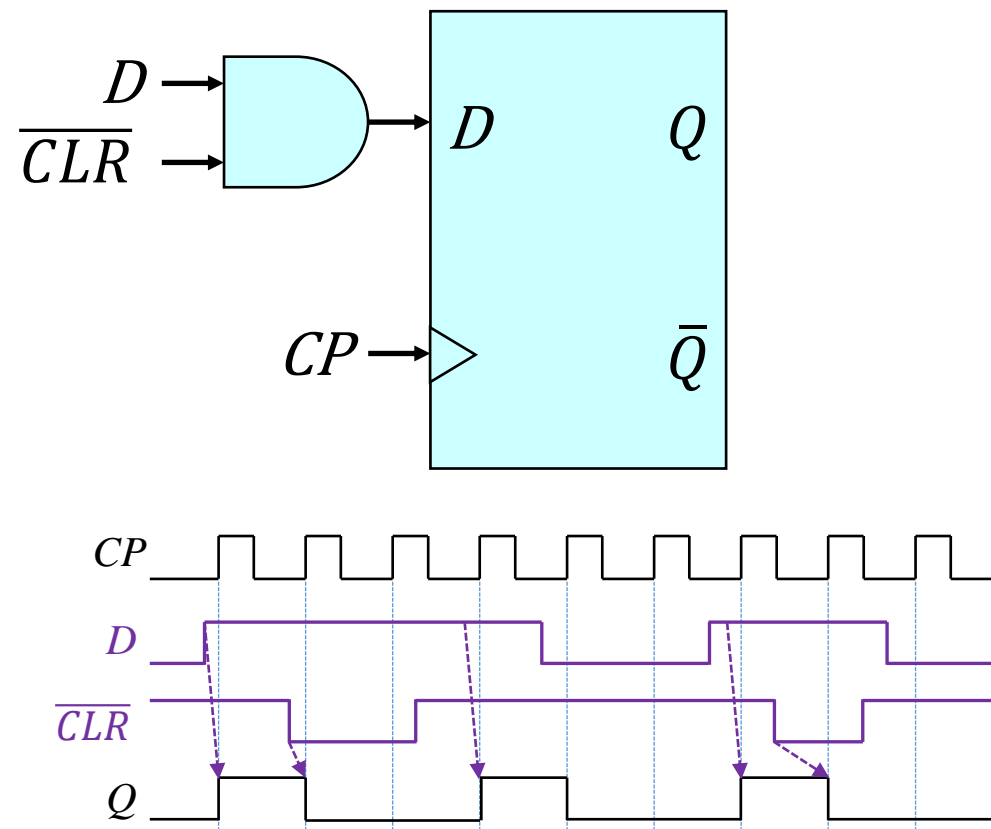
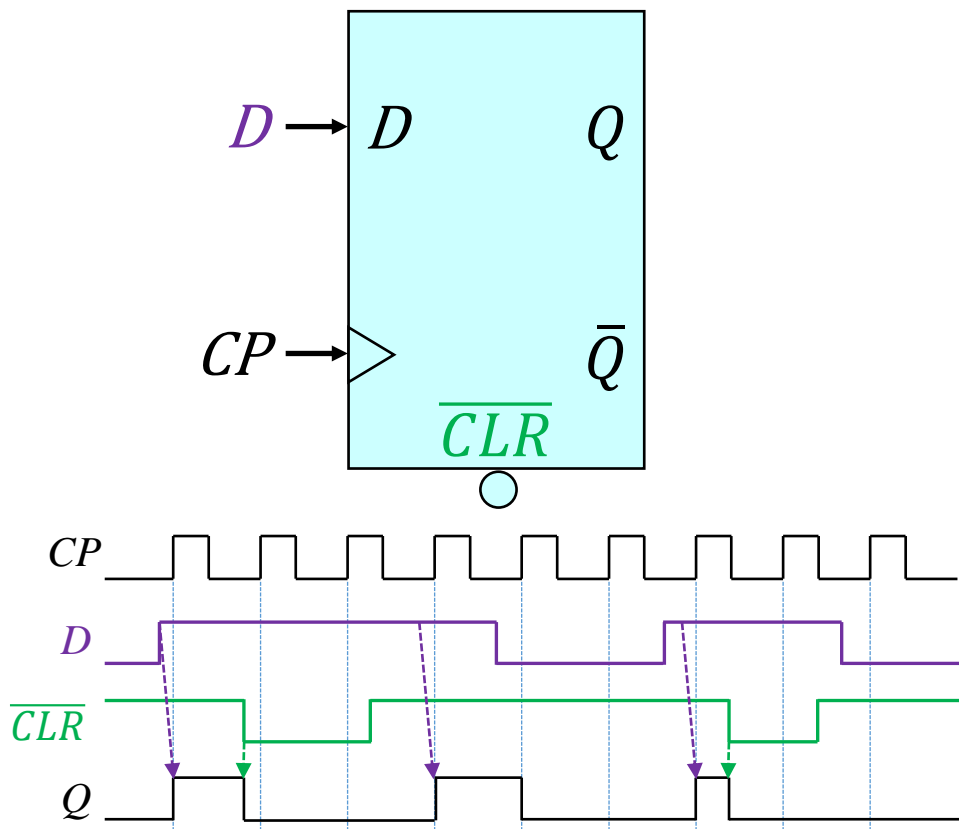
- 조합논리회로 vs. 순차논리회로
  - 조합논리회로: 입력 값이 항상 출력 값에 반영됨
  - 순차논리회로: trigger되었을 때에만 입력 값에 상태에 반영됨
- 순차논리회로 종류

Level triggered	Edge triggered	특성 방정식
Gated SR latch	SR flip-flop	$Q(t + 1) = \bar{R}(S + Q(t))$
<b>Gated D latch</b>	<b>D flip-flop</b>	$Q(t + 1) = D$
Gated JK latch	JK flip-flop	$Q(t + 1) = J\bar{Q}(t) + \bar{K}Q(t)$
Gated T latch	T flip-flop	$Q(t + 1) = \bar{Q}(t) + \bar{T}Q(t) = Q(t) \oplus T$

# 순차논리회로 개요

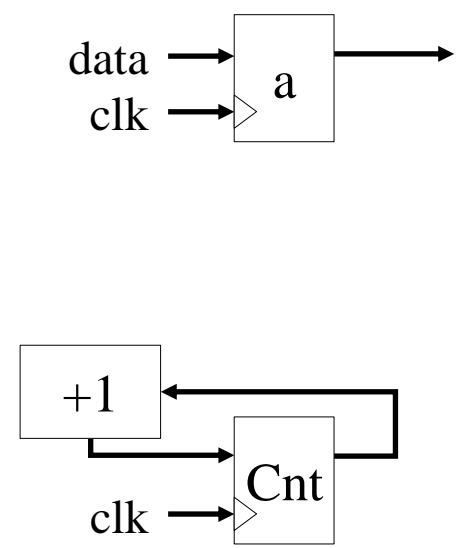
## ● 동기 vs. 비동기 입력 신호

- 동기 입력: edge에서 상태에 영향을 미치는 입력
- 비동기 입력: edge와 상관없이 영향을 미치는 입력



# D Flip-flop

- Verilog에서의 D FF의 표시

Verilog 표현 예시	생성되는 하드웨어 로직
<pre>reg [3:0] r_Cnt; reg [3:0] c_Cnt, n_Cnt;  always@(posedge clk)     a = data;  always@(posedge clk)     r_Cnt = r_Cnt + 1;  always@(posedge clk)     c_Cnt = n_Cnt;  always@*     n_Cnt = c_Cnt+1;</pre>	 <p>The hardware logic diagram illustrates the implementation of the Verilog code. It features a D flip-flop labeled 'a' and a counter labeled 'Cnt'. The flip-flop 'a' has two inputs: 'data' and 'clk', and one output. The counter 'Cnt' has a 'clk' input and a feedback loop where its output is incremented by 1 (via a '+1' block) and then fed back into the counter's input.</p>

# D Flip-flop

- Verilog에서의 D FF의 표시

Verilog 표현 예시	생성되는 하드웨어 로직
<pre>reg a, b, c_Data, n_Data;  always@(posedge clk)   if(!rst) a = 0; // synchronous reset   else a = data;  always@(posedge clk, negege de rst)   if(!rst) b = 0; // asynchronous reset   else b = data;  always@(posedge clk, posegde rst)   if(rst) c_Data = 1; // synchronous set   else c_Data = n_Data;</pre>	<p>The hardware logic consists of three D flip-flops labeled 'a', 'b', and 'c'. Flip-flop 'a' has inputs 'rst' (inverted) and 'data', and clock input 'clk'. Flip-flop 'b' has inputs 'data' and 'clk', and an asynchronous reset input 'rst' (inverted). Flip-flop 'c' has inputs 'n_Data' and 'clk', and a synchronous set input 'rst'.</p>



Counter

# 4-bit counter

- 구현하고자 하는 기능

- 버튼1을 누르면 증가, 버튼2를 누르면 감소하는 4비트 카운터
- 레지스터의 구현은 두 파트로 나누어 기술
  - Combinational logic part: 레지스터 입력 부분에 들어오는 데이터 기술
  - FF part: edge-triggered D FF 생성 + reset 설정

Combinational logic part	FF part
<pre>always@* begin   n_Cnt = fUp ? c_Cnt + 1 :            fDn ? c_Cnt - 1 : c_Cnt; end</pre>	<pre>always@(posedge i_Clk, posedge i_Rst)   if(i_Rst)    c_Cnt = 0;   else        c_Cnt = n_Cnt;</pre>



## ● 명세

### ■ 입력

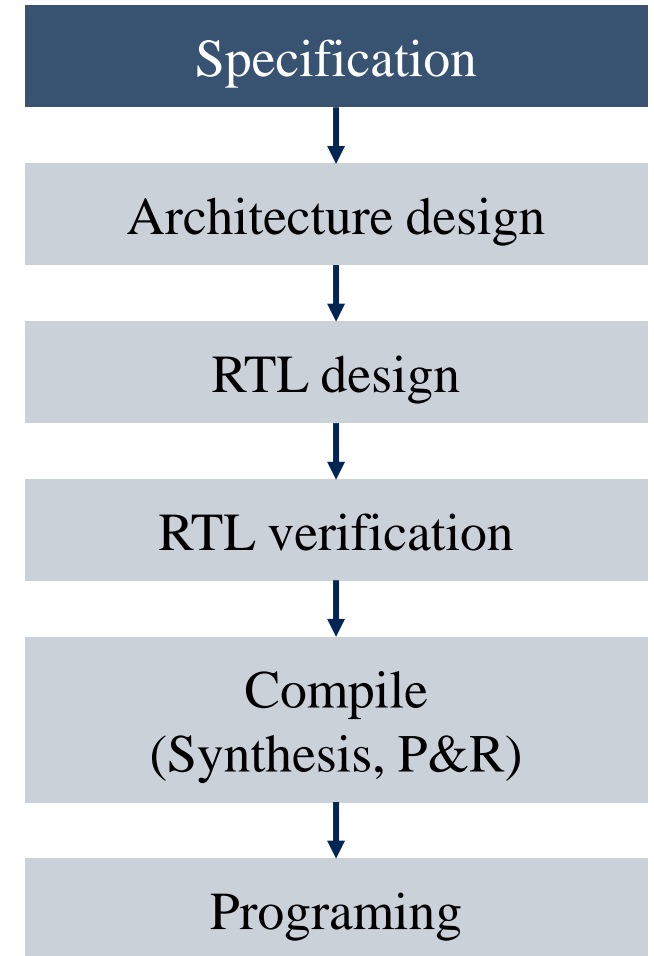
- i\_Push(2) ➔ Up & down
- i\_Clk
- i\_Rst

### ■ 출력: o\_FND(7), o\_LED(4)

### ■ 수행 기능

- Up/down 버튼에 따라 0~9 숫자를 증감

### ■ 모듈명: Counter



## ● RTL 설계

### ■ Counter.v

```
module Counter(i_Clk, i_Rst, i_Push, o_LED, o_FND);
input      i_Clk;      // 50MHz
input      i_Rst;
input      [1:0]      i_Push;
output     wire [3:0] o_LED;
output     wire [6:0] o_FND;
```

```
reg        [3:0]      c_Cnt, n_Cnt;
reg        [1:0]      c_UpDn, n_UpDn;
```

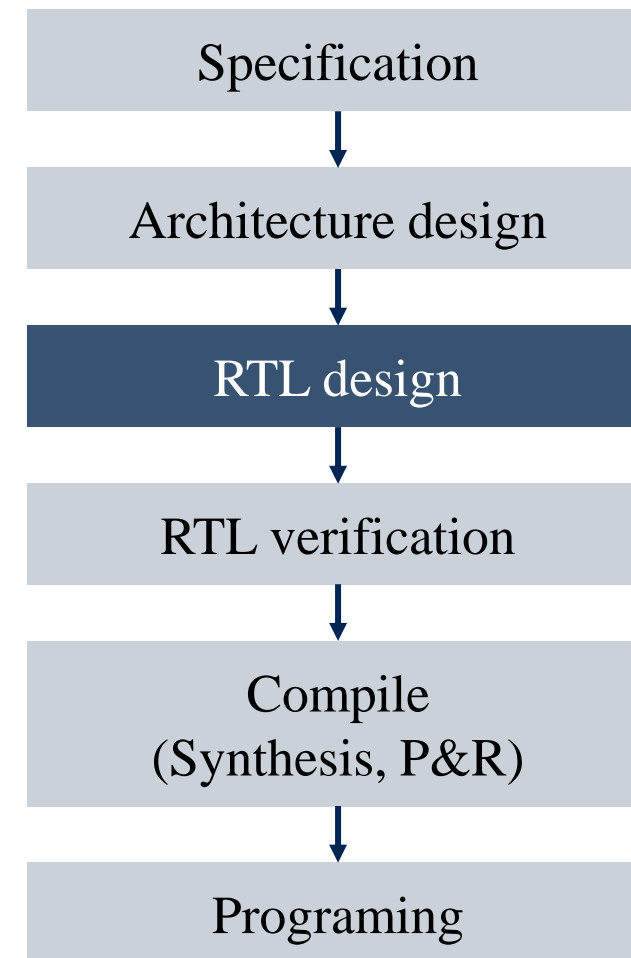
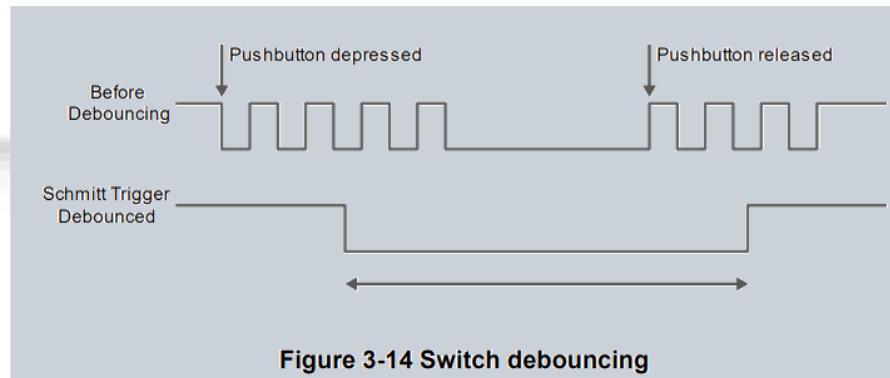
```
wire       fUp;
wire       fDn;
```

```
always@(posedge i_Clk, posedge i_Rst)
  if(i_Rst) begin
    c_Cnt = 0;
    c_UpDn = 2'b11;
  end else begin
    c_Cnt = n_Cnt;
    c_UpDn = n_UpDn;
  end
end
```

```
assign     {fUp, fDn} = ~i_Push & c_UpDn;
```

```
assign     o_LED = c_Cnt;
FND        FND0(c_Cnt, o_FND);
```

```
always@*
begin
  n_UpDn = i_Push;
  n_Cnt  = fUp ? c_Cnt + 1 :
           fDn ? c_Cnt - 1 : c_Cnt;
end
```



## ● RTL 검증 - testbench module

### ■ tb\_Counter.v

- Up/down에 따라 카운터 값이 변화 확인

```
`timescale 1 ns / 1ns
module tb_Cnt();
reg Clk;
reg Rst;
reg [1:0] Push;
wire[3:0] Cnt_o_LED;

Counter U0(Clk, Rst, Push, Cnt_o_LED,);

always
    #10 Clk = ~Clk;

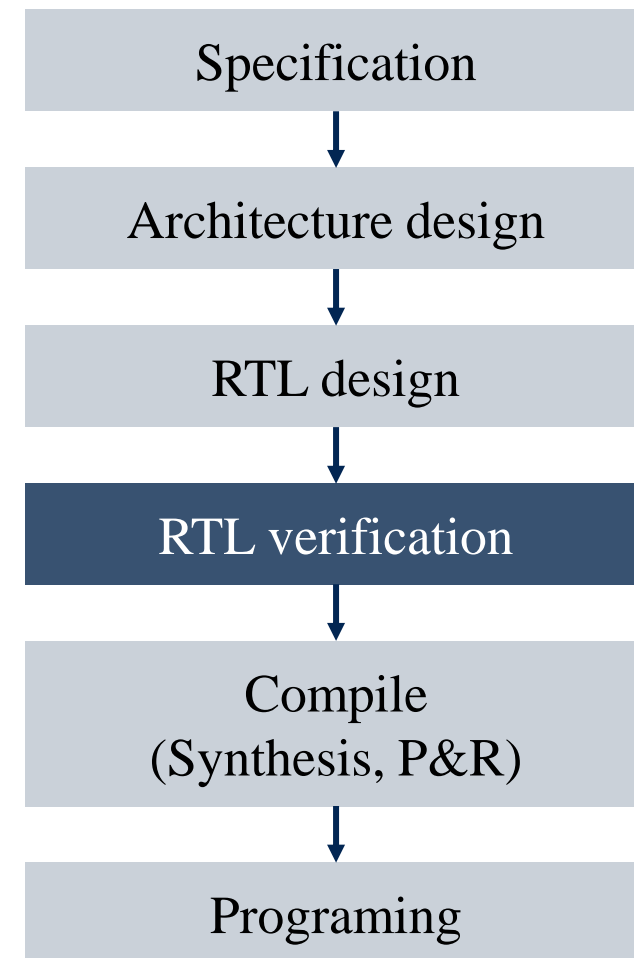
initial
begin
    Clk = 1;
    Rst = 1;
    Push = 2'b11;

    @(posedge Clk) Rst = 1;
    @(negedge Clk) Rst = 0;
```

```
#200 Push = 2'b01; #200 Push = 2'b11;
#200 Push = 2'b01; #200 Push = 2'b11;
#200 Push = 2'b01; #200 Push = 2'b11;
#200 Push = 2'b01; #200 Push = 2'b11;
#200 Push = 2'b01; #200 Push = 2'b11;
#200 Push = 2'b01; #200 Push = 2'b11;
#200 Push = 2'b01; #200 Push = 2'b11;
#200 Push = 2'b01; #200 Push = 2'b11;
#200 Push = 2'b01; #200 Push = 2'b11;
#200 Push = 2'b01; #200 Push = 2'b11;

#200 Push = 2'b10; #200 Push = 2'b11;
#200 Push = 2'b10; #200 Push = 2'b11;
#200 Push = 2'b10; #200 Push = 2'b11;
#200 Push = 2'b10; #200 Push = 2'b11;
#200 Push = 2'b10; #200 Push = 2'b11;
#200 Push = 2'b10; #200 Push = 2'b11;
#200 Push = 2'b10; #200 Push = 2'b11;
#200 Push = 2'b10; #200 Push = 2'b11;
#200 Push = 2'b10; #200 Push = 2'b11;
#200 Push = 2'b10; #200 Push = 2'b11;

end
endmodule
```

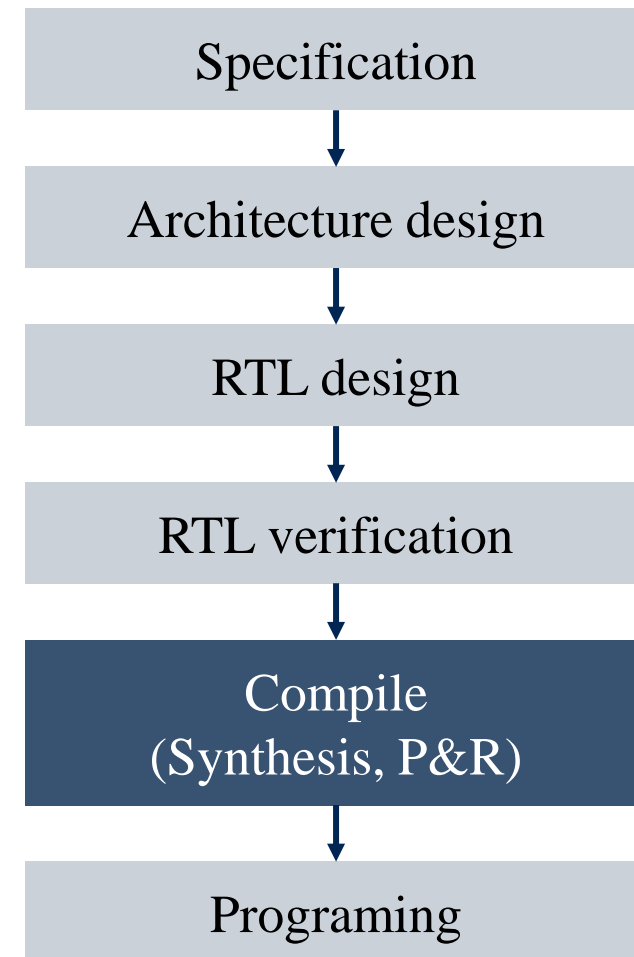


## ● FPGA 구현 – pin 설정

- 입력: i\_Clk, i\_Rst, i\_Push(2)
  - i\_Rst은 SW[9]로
- 출력: o\_FND(6), o\_LED(4)
  - FND와 LED에 대한 핀 설정(qsf 파일의 일부)

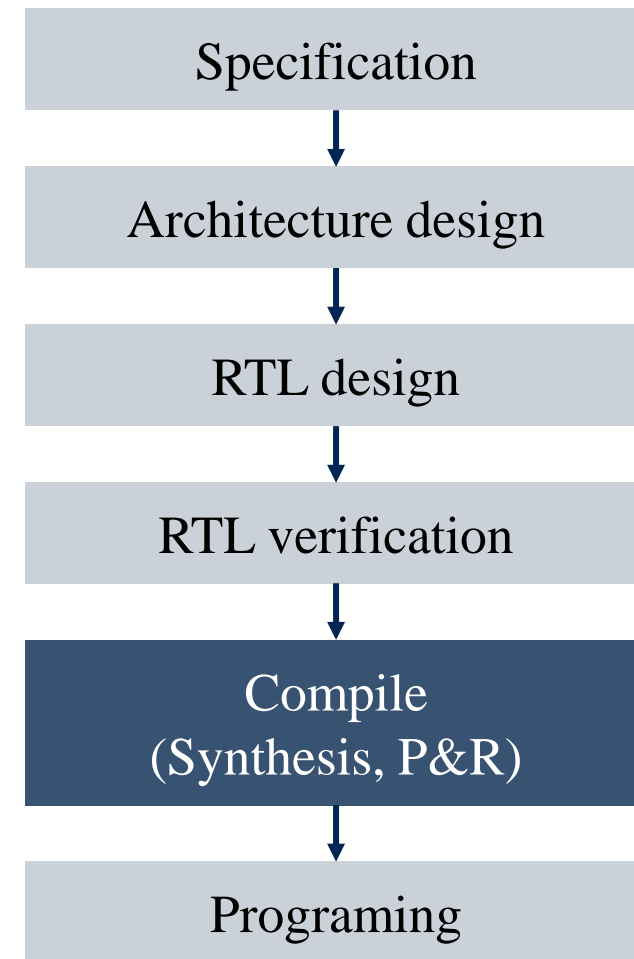
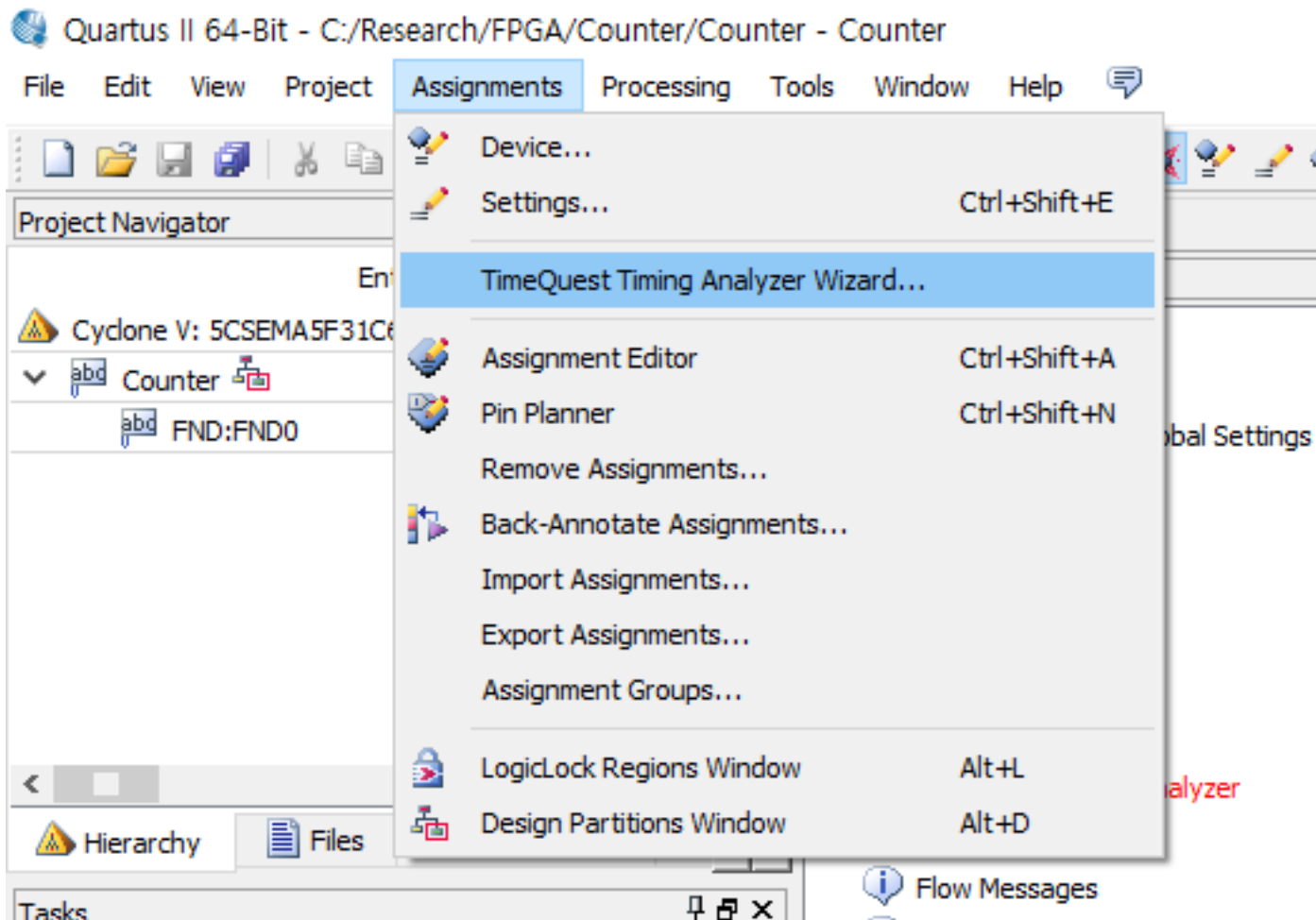
```
set_location_assignment PIN_AE26 -to o_FND[0]
set_location_assignment PIN_AE27 -to o_FND[1]
set_location_assignment PIN_AE28 -to o_FND[2]
set_location_assignment PIN_AG27 -to o_FND[3]
set_location_assignment PIN_AF28 -to o_FND[4]
set_location_assignment PIN_AG28 -to o_FND[5]
set_location_assignment PIN_AH28 -to o_FND[6]
```

```
set_location_assignment PIN_V16 -to o_LED[0]
set_location_assignment PIN_W16 -to o_LED[1]
set_location_assignment PIN_V17 -to o_LED[2]
set_location_assignment PIN_V18 -to o_LED[3]
```

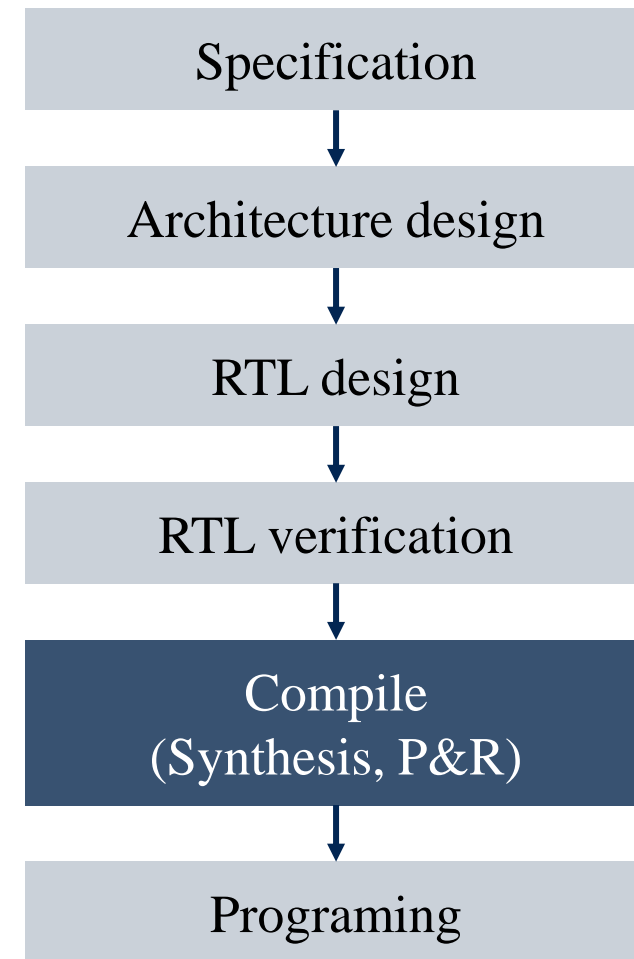
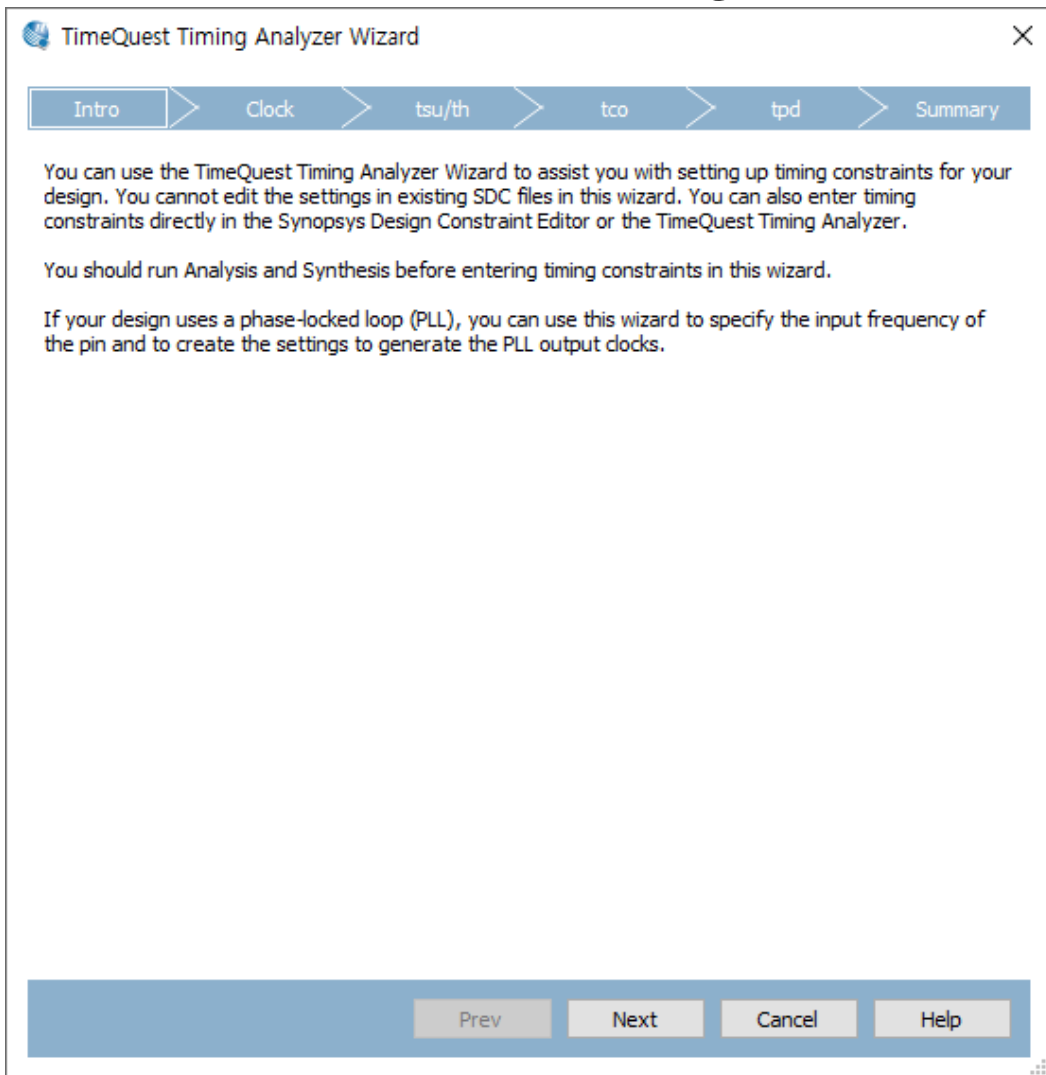


## ● FPGA 구현 – timing 설정

- Assignments - TimeQuest Timing Analyzer Wizard..



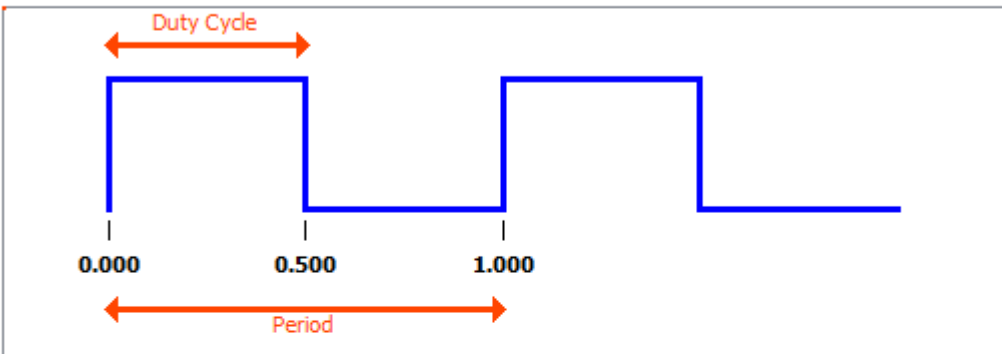
## ● FPGA 구현 – timing 설정



## ● FPGA 구현 – timing 설정: clock

TimeQuest Timing Analyzer Wizard

Intro > **Clock** > tsu/th > tco > tpd > Summary



Specify base clock settings:

	Clock Name	Input Pin	Period	Rising	Falling
1	CLK	i_Clk	20.000ns	0.000	10.000
2	<< New >>				

Equivalent SDC commands:

SDC Command
create_clock -name "CLK" -period 20.000ns [get_ports {i_Clk}] -waveform {0.000 10.000}

## ● FPGA 구현 – timing 설정: input/output delay (clock, reset 제외)

TimeQuest Timing Analyzer Wizard

Intro > Clock > **tsu/th** > tco > tpd > Summary

Note: If an I/O register is clocked by a PLL with a small positive phase shift, add one clock period to the tsu value and subtract one clock period from the th value.

Specify tsu/th settings:

	Port Name	tsu	th	Clock Name	Clock Inverted
1	i_Push[0] i_Push[1]	3.000ns		CLK	
2	<< New >>				

Equivalent SDC commands:

SDC Command
set_input_delay -clock "CLK" -max 7ns [get_ports {i_Push[0] i_Push[1]}]

TimeQuest Timing Analyzer Wizard

Intro > Clock > tsu/th > **tco** > tpd > Summary

Note: If an I/O register is clocked by a PLL with a small negative phase shift, add one clock period to both the tco and minimum tco values.

Specify tco settings:

	Port Name
1	o_FND[0] o_FND[1] o_FND[2] o_FND[3] o_FND[4] o_FND[5] o_FND[6] o_LED[0] o_LED[1] o_LED[2]
2	<< New >>

Equivalent SDC commands:


SDC Command
set_output_delay -clock "CLK" -max 15ns [get_ports {o_FND[0] o_FND[1] o_FND[2] o_FND[3] o_FND[4] o_FND[5] o_FND[6] o_LED[0] o_LED[1] o_LED[2] o_LED[3]}]



## ● FPGA 구현 – timing 설정

TimeQuest Timing Analyzer Wizard

Intro > Clock > tsu/th > tco > **tpd** > Summary



input port                      output port

Specify tpd settings:

	Input Port	Output Port	tpd	Minimum tpd
1	<< New >>	<< New >>		

Equivalent SDC commands:

SDC Command

TimeQuest Timing Analyzer Wizard

Intro > Clock > tsu/th > tco > tpd > **Summary**

Output SDC file

What name do you want for the output SDC file?

C:/Research/FPGA/Counter/Counter.sdc ...

☒ Add this output SDC file to the current project.

SDC Preview:

```
# Clock constraints
create_clock -name "CLK" -period 20.000ns [get_ports {i_Clk}] -waveform {0.000 10.000}

# Automatically constrain PLL and other generated clocks
derive_pll_clocks -create_base_clocks

# Automatically calculate clock uncertainty to jitter and other effects.
derive_clock_uncertainty

# tsu/th constraints
set_input_delay -clock "CLK" -max 17ns [get_ports {i_Push[0] i_Push[1]}]

# tco constraints
set_output_delay -clock "CLK" -max 15ns [get_ports {o_FND[0] o_FND[1] o_FND[2] o_FND[3]
o_FND[4] o_FND[5] o_FND[6] o_LED[0] o_LED[1] o_LED[2] o_LED[3]}]
```

Prev Finish Cancel Help

- 추가 연습1
  - 0에서 down 시 9로, 9에서 up 시 0으로 가도록 수정
- 추가 연습2
  - 기존 1자리에서 2자리로 수정
  - Up/down 버튼에 따라 00~99 숫자를 증감

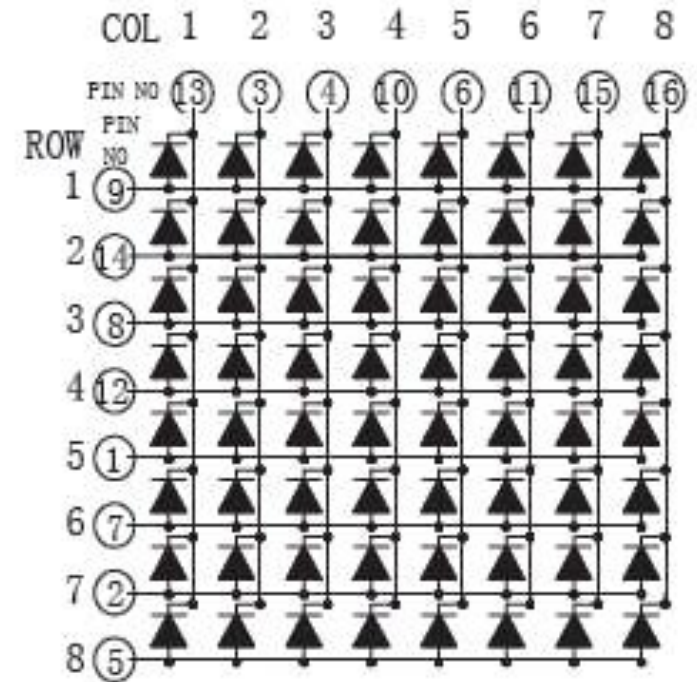
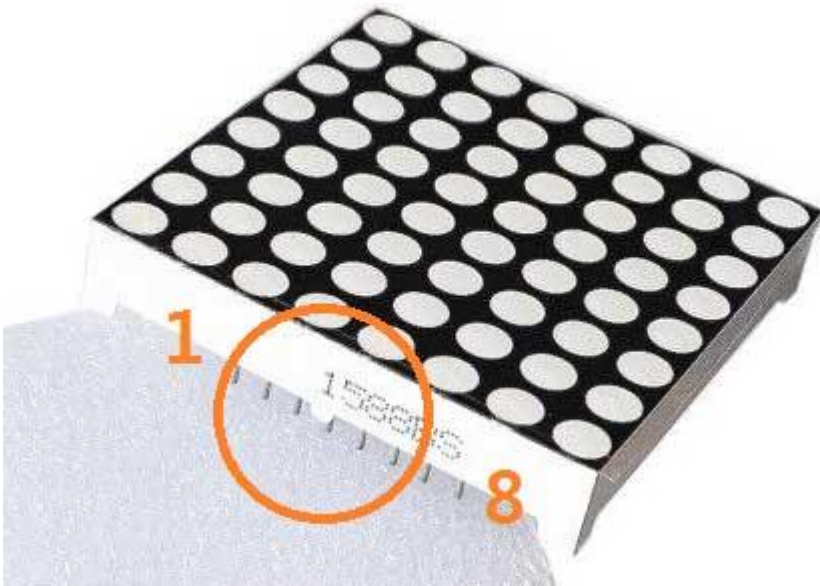


Dot matrix

# Dot matrix

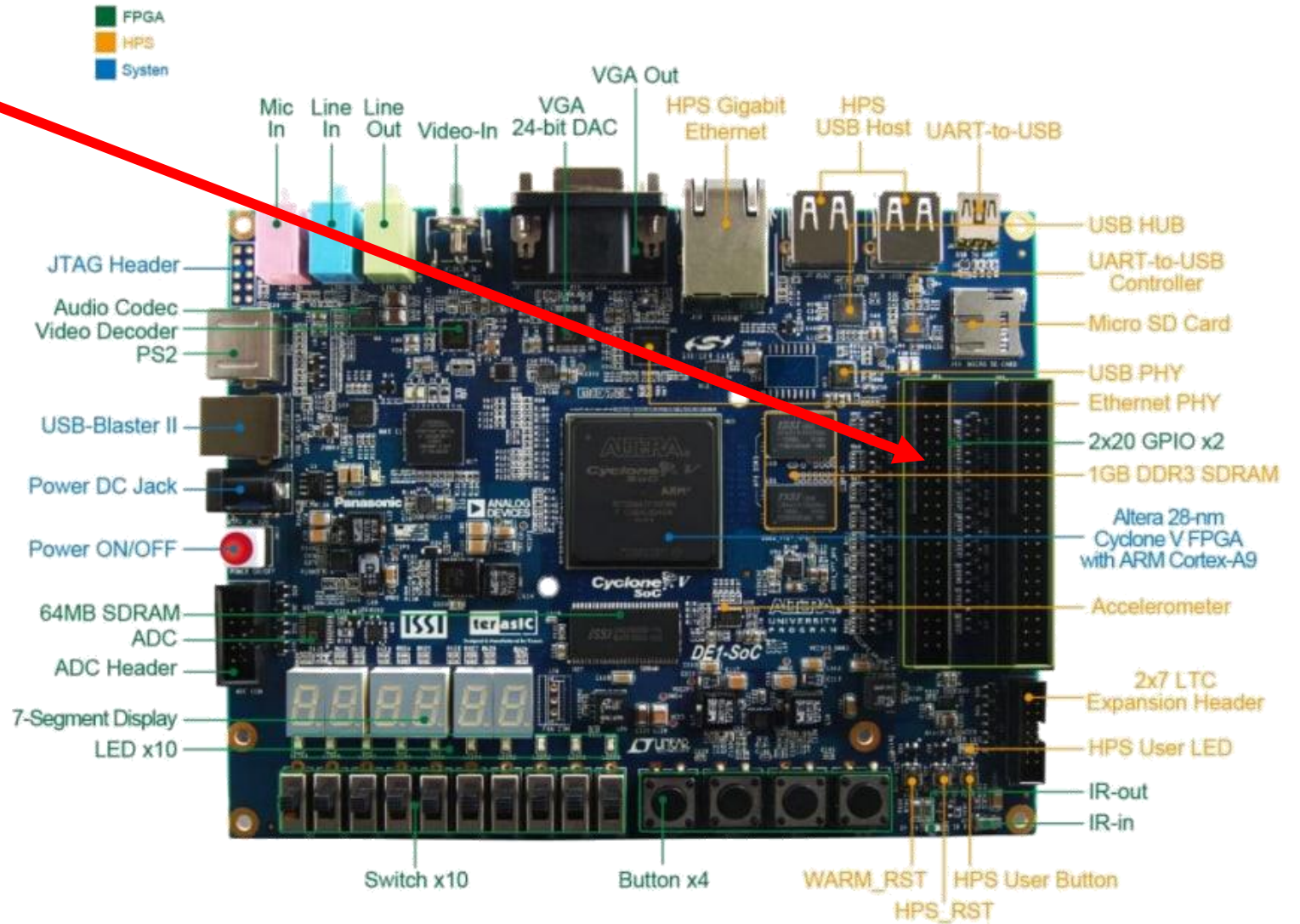
- 사용하려는 부품

- 8x8 형태로 LED가 2차원 배열 형태로 배치
- 불이 켜지는 조건: Row 쪽에 전원, Column 쪽에 GND



# Dot matrix

- DE1-SoC에 연결
  - GPIO 부분에 연결

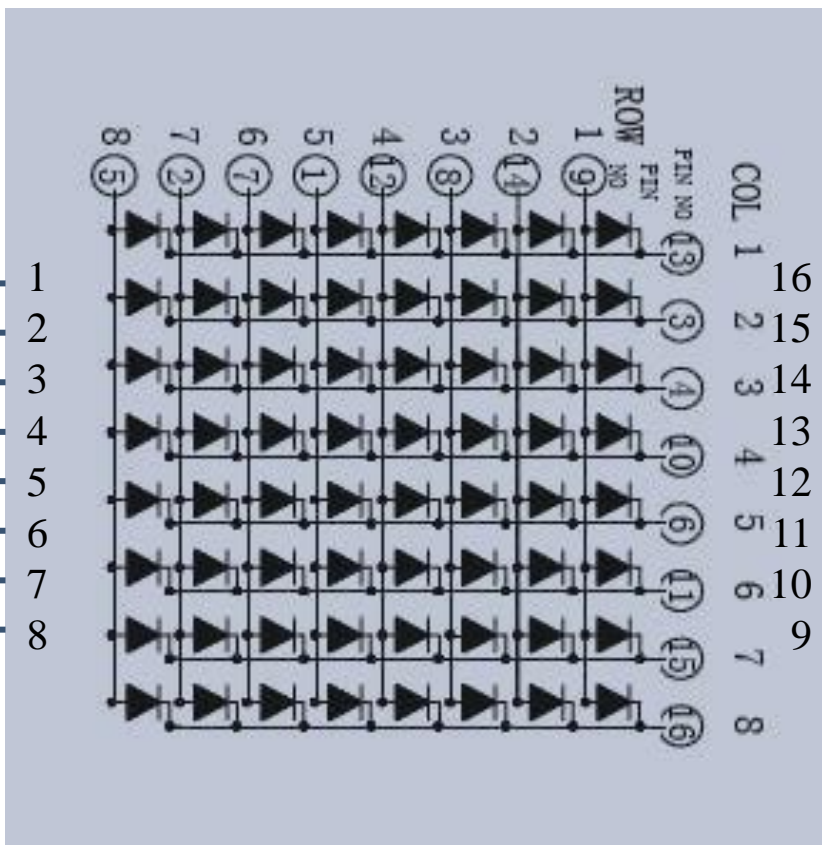


# Dot matrix

## ● DE1-SoC에 연결

GPIO0

0	1
2	3
4	5
6	7
8	9
VCC	GND
10	11
12	13
14	15
16	17
18	19
20	21
22	23
24	25
VCC	GND
26	27
28	29
30	31
32	33
34	35



GPIO1

0	1
2	3
4	5
6	7
8	9
VCC	GND
10	11
12	13
14	15
16	17
18	19
20	21
22	23
24	25
VCC	GND
26	27
28	29
30	31
32	33
34	35

Signal		PIN		
DotMat.	Verilog	DotMat.	Board	FPGA
Row1	Row[0]	9	GPIO1-25	PIN_AH23
Row2	Row[1]	14	GPIO1-15	PIN_AK28
Row3	Row[2]	8	GPIO0-25	PIN_AD20
Row4	Row[3]	12	GPIO1-19	PIN_AH25
Row5	Row[4]	1	GPIO0-11	PIN_AH17
Row6	Row[5]	7	GPIO0-23	PIN_AK21
Row7	Row[6]	2	GPIO0-13	PIN_AE16
Row8	Row[7]	5	GPIO0-19	PIN_AC20
Col1	Col[0]	13	GPIO1-17	PIN_AJ26
Col2	Col[1]	3	GPIO0-15	PIN_AG17
Col3	Col[2]	4	GPIO0-17	PIN_AA19
Col4	Col[3]	10	GPIO1-23	PIN_AG23
Col5	Col[4]	6	GPIO0-21	PIN_AJ20
Col6	Col[5]	11	GPIO1-21	PIN_AJ24
Col7	Col[6]	15	GPIO1-13	PIN_AJ27
Col8	Col[7]	16	GPIO1-11	PIN_AH24

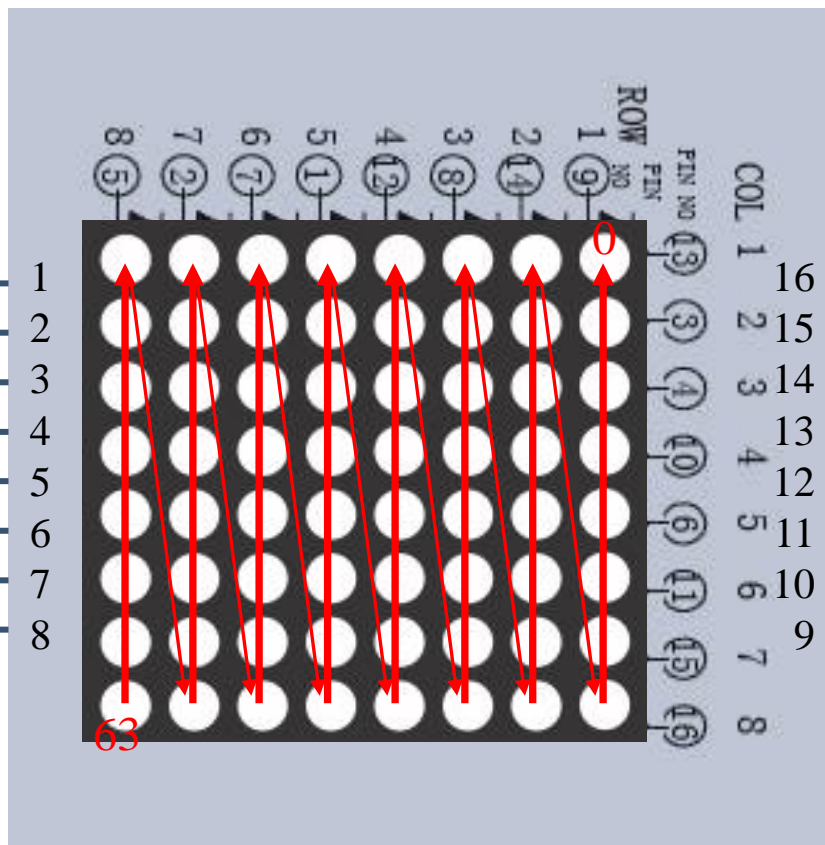


# Dot matrix

- DE1-SoC에 연결

GPIO0

0	1
2	3
4	5
6	7
8	9
VCC	GND
10	11
12	13
14	15
16	17
18	19
20	21
22	23
24	25
VCC	GND
26	27
28	29
30	31
32	33
34	35



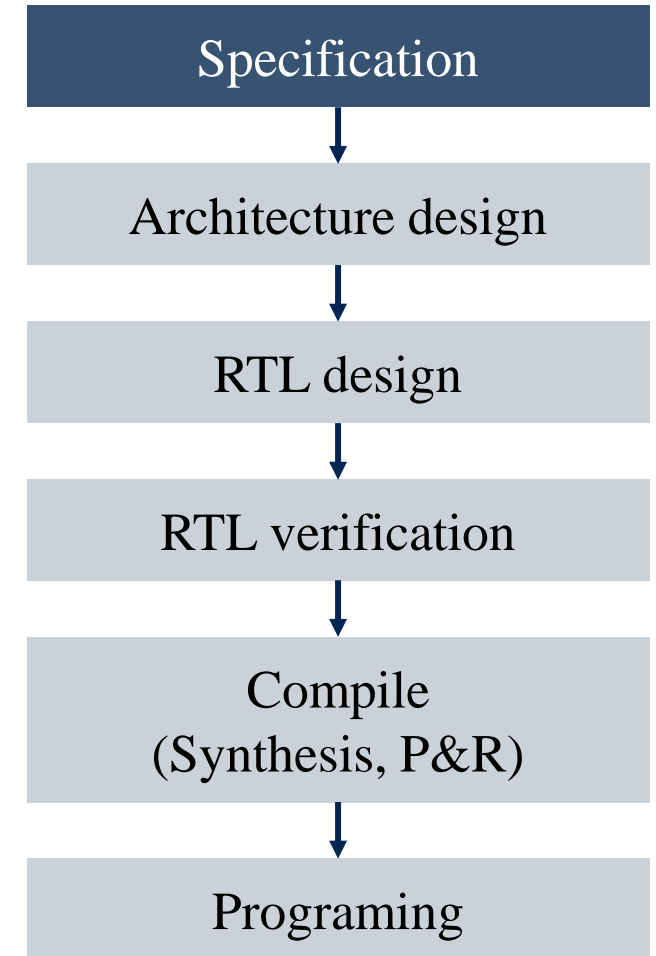
GPIO1

0	1
2	3
4	5
6	7
8	9
VCC	GND
10	11
12	13
14	15
16	17
18	19
20	21
22	23
24	25
VCC	GND
26	27
28	29
30	31
32	33
34	35

Signal		PIN		
DotMat.	Verilog	DotMat.	Board	FPGA
Row1	Row[0]	9	GPIO1-25	PIN_AH23
Row2	Row[1]	14	GPIO1-15	PIN_AK28
Row3	Row[2]	8	GPIO0-25	PIN_AD20
Row4	Row[3]	12	GPIO1-19	PIN_AH25
Row5	Row[4]	1	GPIO0-11	PIN_AH17
Row6	Row[5]	7	GPIO0-23	PIN_AK21
Row7	Row[6]	2	GPIO0-13	PIN_AE16
Row8	Row[7]	5	GPIO0-19	PIN_AC20
Col1	Col[0]	13	GPIO1-17	PIN_AJ26
Col2	Col[1]	3	GPIO0-15	PIN_AG17
Col3	Col[2]	4	GPIO0-17	PIN_AA19
Col4	Col[3]	10	GPIO1-23	PIN_AG23
Col5	Col[4]	6	GPIO0-21	PIN_AJ20
Col6	Col[5]	11	GPIO1-21	PIN_AJ24
Col7	Col[6]	15	GPIO1-13	PIN_AJ27
Col8	Col[7]	16	GPIO1-11	PIN_AH24

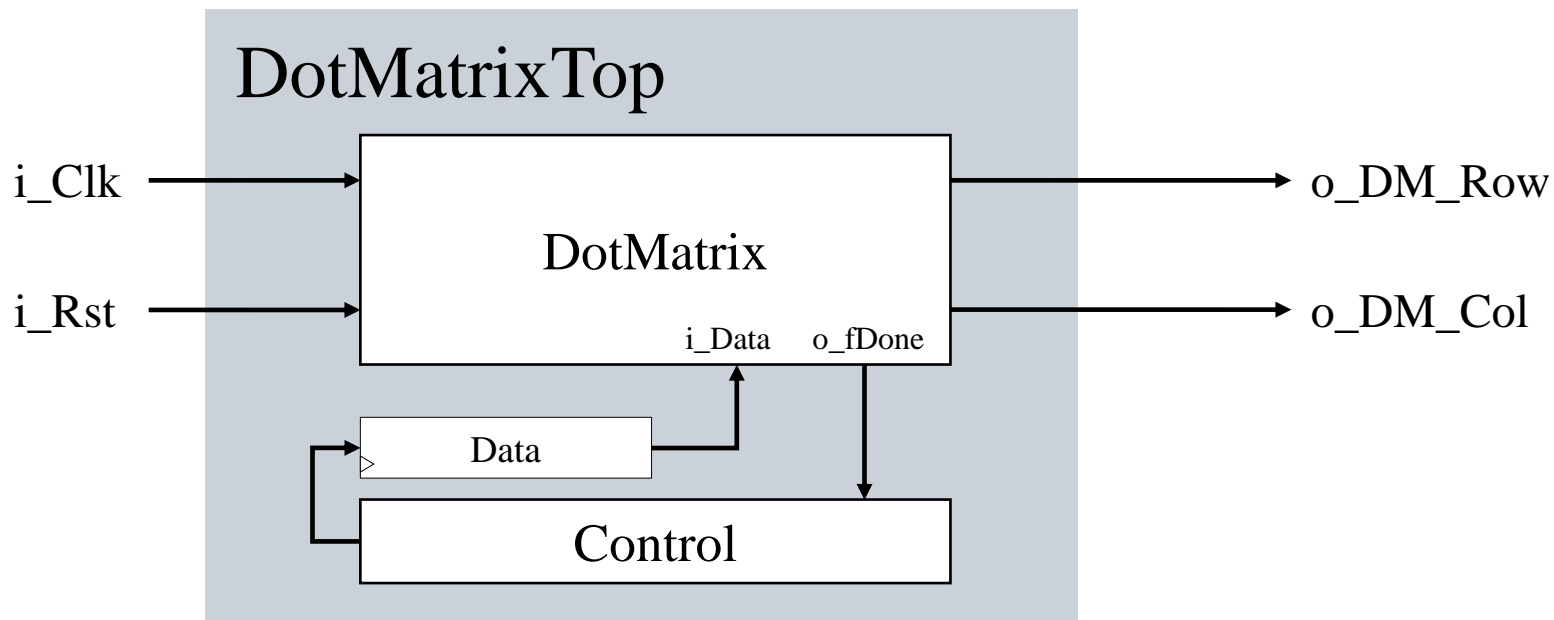
## ● 명세

- 출력: o\_DM\_Col(8), o\_DM\_Row(8)
- 수행 기능
  - 8x8 이미지를 출력
- 모듈명: DotMatrixTop





## ● 구조 설계

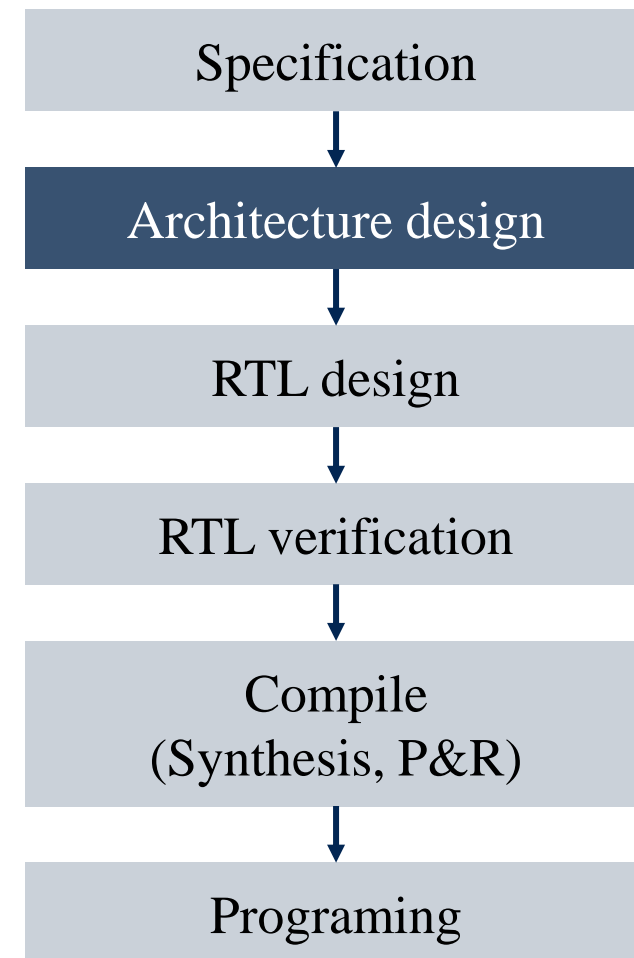


### ■ DotMatrix

- 64-bit 데이터(**i\_Data**)를 입력받아 dot matrix로 출력
- **o\_fDone**은 1회 출력 완료(16ms 간격)마다 1

### ■ DotMatrixTop

- 3가지 데이터를 1024ms마다 dot matrix에 전달



## ● RTL 설계

### ■ DotMatrix.v

```
module DotMatrix(i_Clk, i_Rst, i_Data, o_DM_Col, o_DM_Row, o_fDone);
```

```
input i_Clk; // 50MHz
```

```
input i_Rst;
```

```
input [63:0] i_Data;
```

```
output wire [7:0] o_DM_Col, o_DM_Row;
```

```
output wire o_fDone;
```

```
reg [7:0] c_Row, n_Row;
```

```
reg [16:0] c_Cnt, n_Cnt;
```

```
wire f2ms;
```

```
assign o_fDone = c_Row[7] && f2ms;
```

```
assign o_DM_Row = c_Row;
```

```
assign o_DM_Col =
```

```
(c_Row[7] ? i_Data[8*7+:8] : 0) |
```

```
(c_Row[6] ? i_Data[8*6+:8] : 0) |
```

```
(c_Row[5] ? i_Data[8*5+:8] : 0) |
```

```
(c_Row[4] ? i_Data[8*4+:8] : 0) |
```

```
(c_Row[3] ? i_Data[8*3+:8] : 0) |
```

```
(c_Row[2] ? i_Data[8*2+:8] : 0) |
```

```
(c_Row[1] ? i_Data[8*1+:8] : 0) |
```

```
(c_Row[0] ? i_Data[8*0+:8] : 0);
```

```
assign f2ms = c_Cnt == 100000 - 1;
```

```
always@(posedge i_Clk, posedge i_Rst)
```

```
if(i_Rst) begin
```

```
c_Row = 1;
```

```
c_Cnt = 0;
```

```
end else begin
```

```
c_Row = n_Row;
```

```
c_Cnt = n_Cnt;
```

```
end
```

```
always@*
```

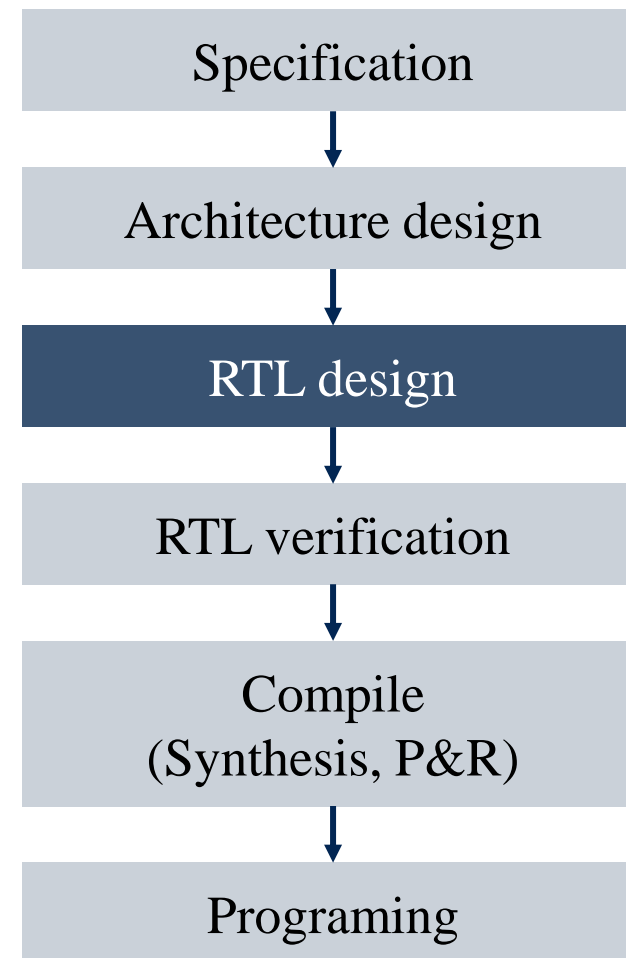
```
begin
```

```
n_Cnt = f2ms ? 0 : c_Cnt + 1;
```

```
n_Row = f2ms ? {c_Row[6:0], c_Row[7]} : c_Row;
```

```
end
```

```
endmodule
```



## ● RTL 설계

### ■ DotMatrixTop.v

```
module DotMatrixTop(i_Clk, i_Rst, o_DM_Col, o_DM_Row);
input      i_Clk;      // 50MHz
input      i_Rst;
output     wire [7:0] o_DM_Col, o_DM_Row;
```

```
reg [ 7:0]  c_Cnt, n_Cnt;
reg [63:0]  c_Data, n_Data;
```

```
wire DM_o_fDone;      // 16ms
```

```
DotMatrix  DM0(i_Clk, i_Rst, c_Data, o_DM_Col, o_DM_Row, DM_o_fDone);
```

```
always@(posedge i_Clk, posedge i_Rst)
    if(i_Rst) begin
        c_Cnt = 0;
        c_Data = 0;
    end else begin
        c_Cnt = n_Cnt;
        c_Data = n_Data;
    end
end
```

```
always@*
begin
    n_Cnt    = DM_o_fDone ? c_Cnt + 1 : c_Cnt;
    case(c_Cnt[7:6])
        2'h0    : n_Data = HEART;
        2'h1    : n_Data = SMILE;
        default : n_Data = ARROW;
    endcase
end
endmodule
```

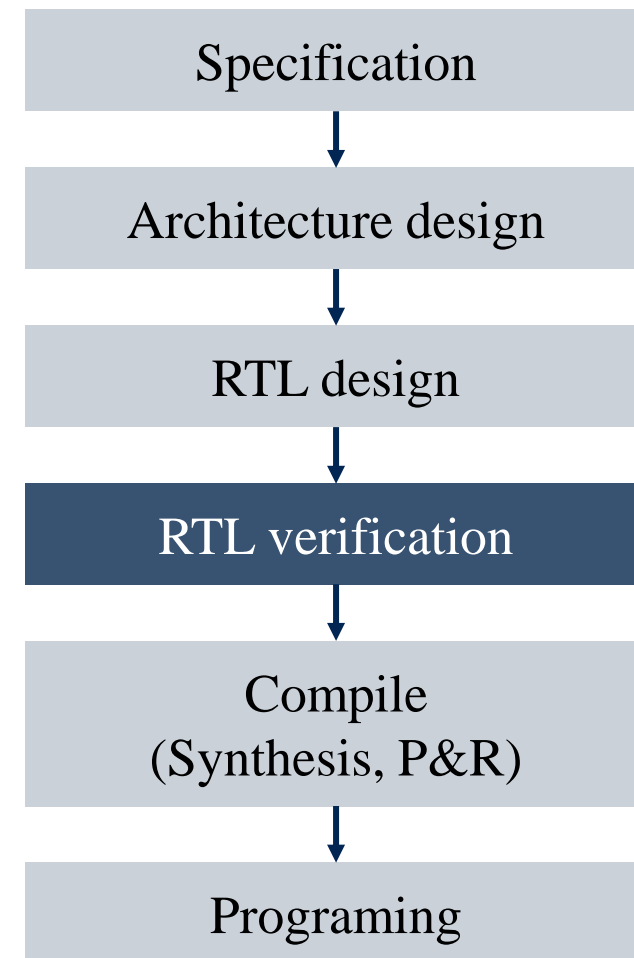
```
// <= counterclockwise rotation
parameter HEART    = {
    8'b11100011,
    8'b11011101,
    8'b10111101,
    8'b01111011,
    8'b00000011,
    8'b10000001,
    8'b11000001,
    8'b11100011};

parameter SMILE    = {
    8'b11000011,
    8'b10111101,
    8'b01101010,
    8'b01011110,
    8'b01011110,
    8'b01101010,
    8'b10111101,
    8'b11000011};

parameter ARROW    = {
    8'b11000011,
    8'b11000011,
    8'b11000011,
    8'b11000011,
    8'b00000000,
    8'b10000001,
    8'b11000011,
    8'b11100111};
```

## ● RTL 검증 - testbench module

- tb\_DotMatrix.v
  - DotMatrix 모듈 확인
  - 확인 내용
    - 2ms마다 o\_Row의 값이 변화하는지 확인
      - ✓ 8'b00000001 → 8'b00000010 → ...
    - o\_Row의 변화에 따른 o\_Col 확인
      - ✓ 예) o\_Row[7] = 1 일 때 o\_Col == i\_Data[63:56]
    - O\_fDone 확인



- RTL 검증 - testbench module

- tb\_DotMatrixTop.v
  - DotMatrixTop 모듈 확인
  - 확인 내용
    - 1024ms마다 DotMatrix의 입력데이터가 바뀌고 그에 따라 DotMatrix의 출력도 제대로 바뀌는지 확인

