LAB 7. Counters

2017 Fall Logic Design LAB

Department of Computer Science and Engineering

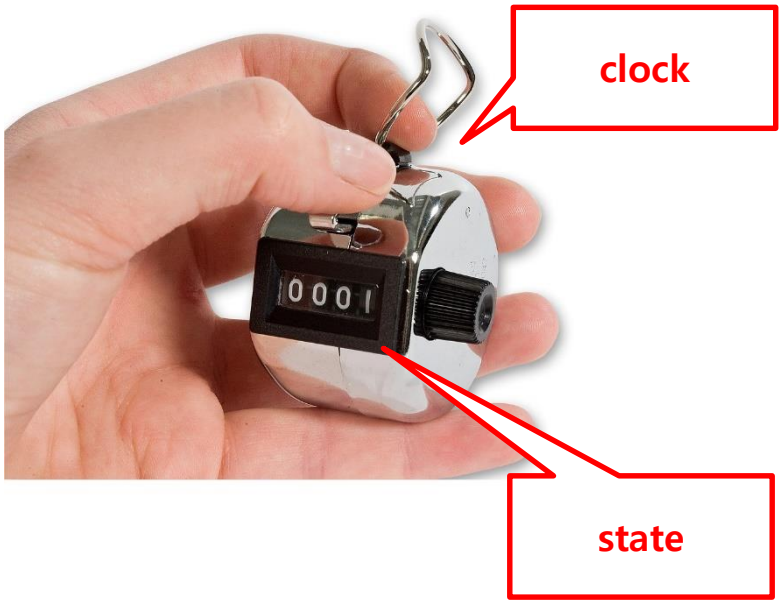
Seoul National University

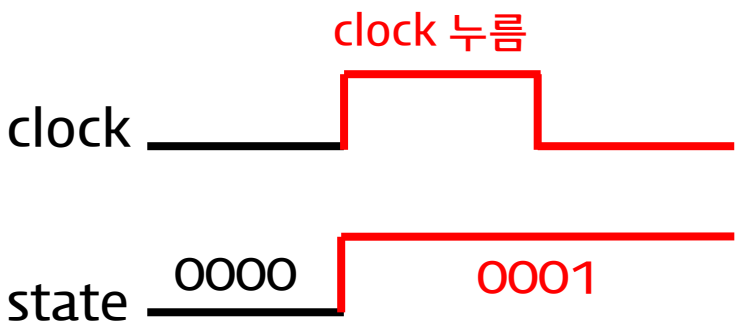
Outline

1. Counter의 기초
2. 4-bit Catalog Counter
3. Chattering 문제 및 해결 방법

Counter

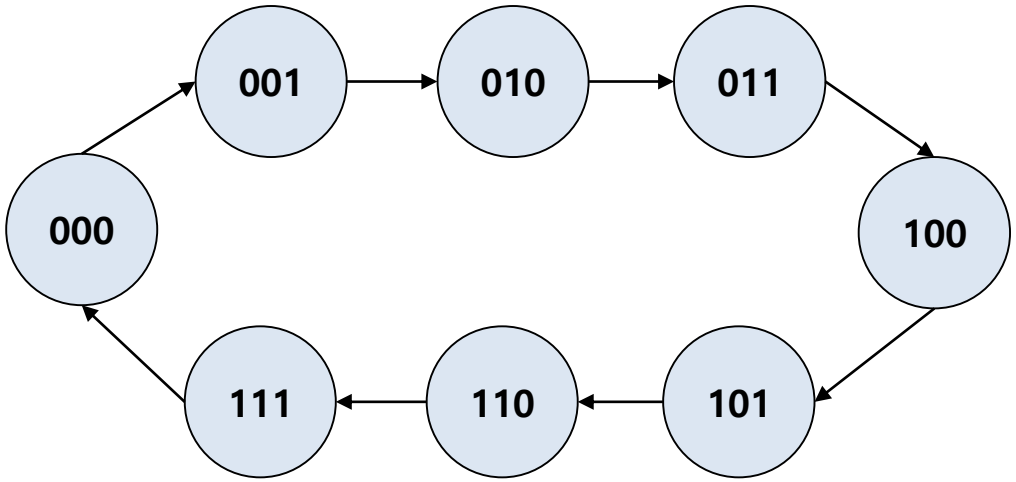
- 입력 신호에 따라 register의 상태(state)가 정해진 패턴대로 변화하는 회로
  - Finite state machine의 일종
  - Synchronous counter : 같은 clock 신호에 의해 동시에 모든 Flip-Flop trigger
  - Asynchronous counter : 전단 Flip-Flop 부터 순차적으로 변화





Counter의 종류

- Up/down counter
  - 가장 일반적인 형태의 카운터로 상태값이 순차적으로 증가/감소하는 카운터
  - **Binary counters**: 가장 일반적인 n-bit 이진수 사용하여 상태 변화 표현
  - Decade counters: 10진수를 표현하기 위해 0~9까지의 값만 표현
  - Gray-code counters: 상태 간 1 bit 단위로만 변화하는 gray code를 사용

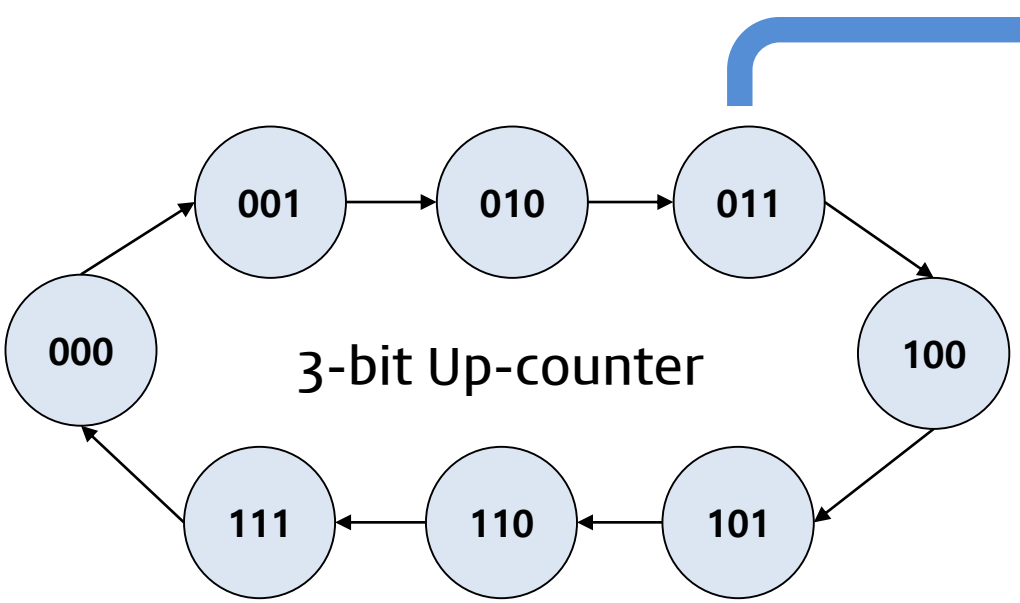


3-bit binary up-counter

Counter Logic Design

[실습1]

- 기본적인 Up/down 뿐 아니라 어떠한 상태 패턴도 counter를 통해 구현 가능
  - 현재 상태와 다음 상태의 관계를 나타내는 상태 변화도(State Transition Diagram) 작성
  - 상태 변화도를 상태 표(State Table)로 변환
  - 작성된 상태 표를 통해 next-state logic 표현



3-bit up-counter의 state transition diagram

3-bit up-counter의 state table

C	B	A	C+	B+	A+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

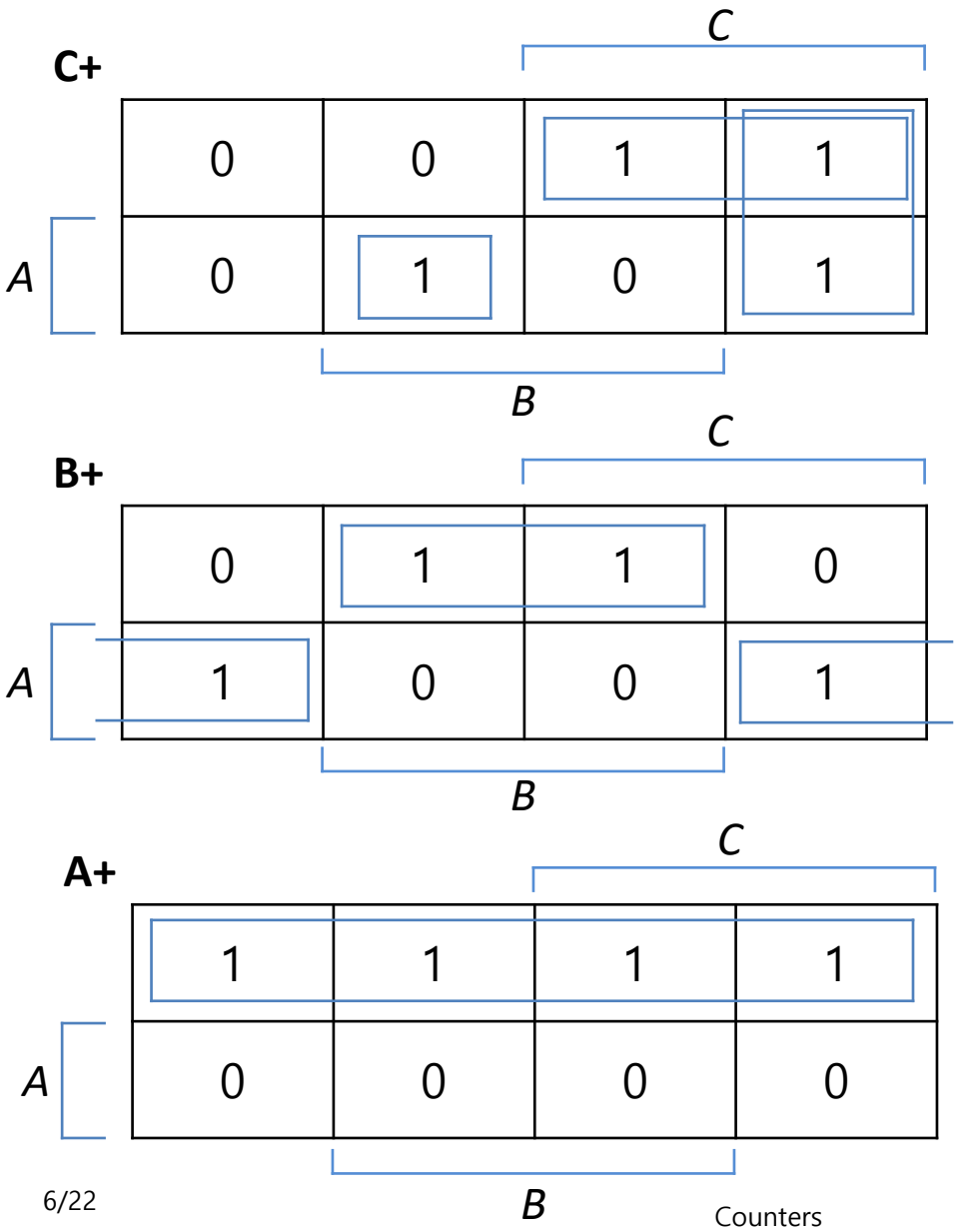
Expressing next-state logic

[실습1]

- 3-bit up-counter
  next-state에 대한 K-map

C	B	A	C+	B+	A+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

3-bit up-counter의 state table



Next-state logic implementation

[실습1]

- K-map 결과를 이용하여 최종적으로 3-bit up-counter의 구현

C	B	A	C+	B+	A+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

3-bit up-counter의 state table

$$C+ = \bar{A}C + \bar{B}C + AB\bar{C}$$

$$B+ = A\bar{B} + \bar{A}B$$

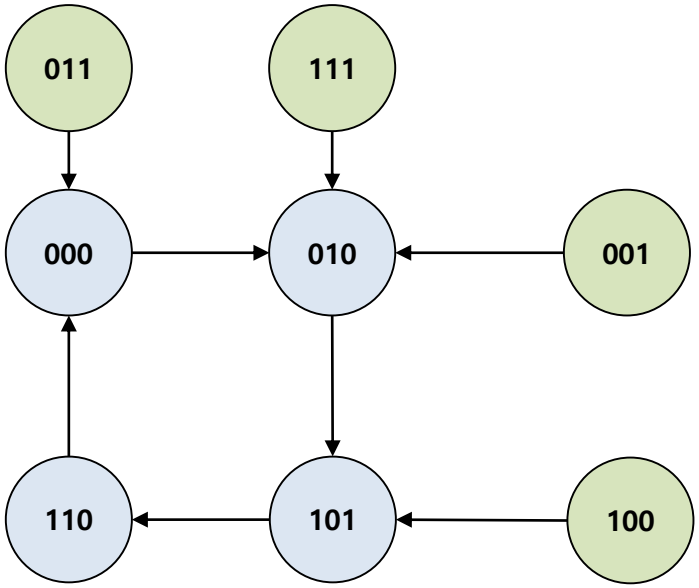
$$A+ = \bar{A}$$

3-bit up-counter의 boolean expression

Self-Starting Counter

[실습1]

- 표현할 수 있는 모든 상태 조합이 나타나지 않는 counter도 존재할 수 있음
  - 포함하지 않는 state의 경우 don't care 로 처리 후 K-map
  - 만약 counter가 포함하지 않은 state에서 시작할 경우는 정상동작을 보장하지 못함
    - **Don't care 상태들에 일정한 값을 부여하여 counter에 포함된 state로 진입할 수 있도록 한다.**



state transition diagram with self-starting

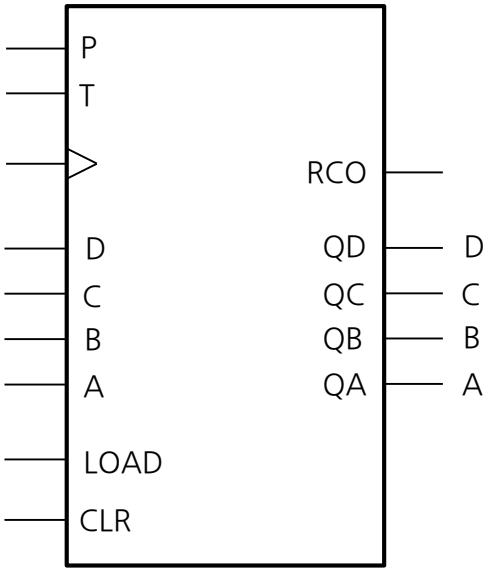
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	-	-	-
0	1	0	1	0	1
0	1	1	-	-	-
1	0	0	-	-	-
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	-	-	-

state table with don't care

4-bit Catalog Counter

[실습2]

- TTL catalog에 속한 회로 중 163 synchronous 4-bit counter
  - (P, T, $\overline{LOAD}$, $\overline{CLR}$)의 4 종류 컨트롤 신호 입력
  - (D, C, B, A)의 4-bit 데이터 신호 입력
  - RCO (Ripple-Carry output)와 (QD, QC, QB, QA) 4-bit 신호 출력
  - CLK 신호에 따른 synchronous한 load, clear 기능 수행 가능한 카운터



4-bit catalog counter

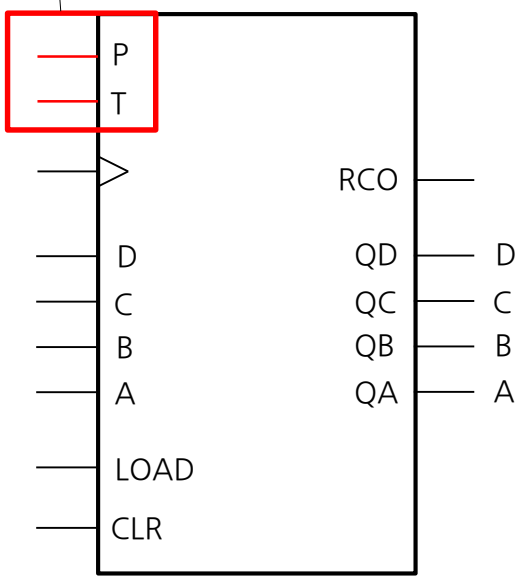
Counter with enable

[실습2]

```
module counter_with_enable(
    input clk,
    input enable,
    output [3:0] out_cnt
    );

    reg [3:0] cnt = 4'b0;
    assign out_cnt = cnt;

    always @(posedge clk)
    begin
        if (enable == 1)
            begin
                cnt = cnt + 1;
            end
    end
Endmodule
```

- Enable 신호가 1이 들어왔을 때만 레지스터의 값을 증가시킴

실제 catalog counter 구현 시에는 P AND T 값을 enable로 사용

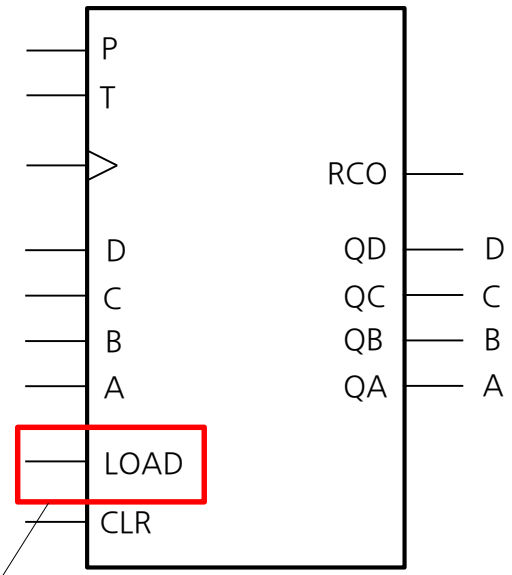


Counter with load

[실습2]

```
module counter_with_load(
    input clk,
    input load,
    input [3:0] load_value,
    output [3:0] out_cnt
    );

    reg [3:0] cnt = 4'b0;
    assign out_cnt = cnt;
    always @(posedge clk)
    begin
        if (load == 1)
            begin
                cnt = load_value;
            end
        else
            begin
                cnt = cnt + 1;
            end
    end
Endmodule
```

- load 신호가 1이 들어왔을 경우, load_value의 값으로 내부 레지스터를 업데이트



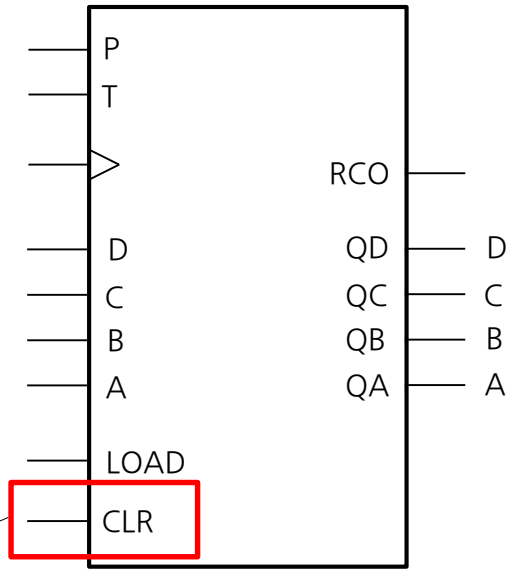
LOAD 신호가 입력되면 내부 레지스터를 {D, C, B, A} 값으로 업데이트

Counter with clear

[실습2]

```
module counter_with_clear(
    input clk,
    input not_clear,
    output [3:0] out_cnt
    );

    reg [3:0] cnt = 4'b0;
    assign out_cnt = cnt;

    always @(posedge clk)
    begin
        if (not_clear == 1)
            begin
                cnt = cnt + 1;
            end
        else
            begin
                cnt = 4'b0;
            end
    end
Endmodule
```

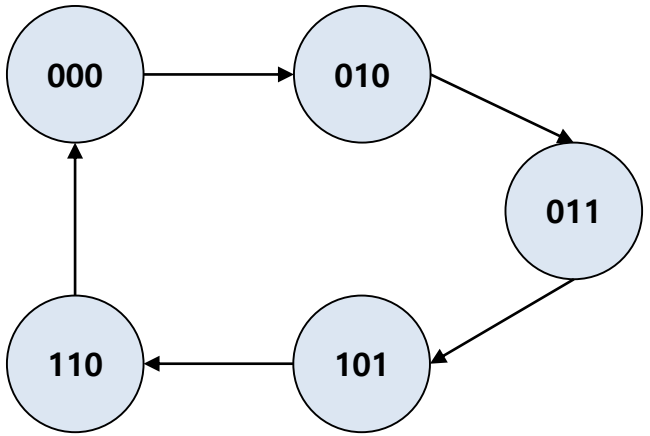
- not_clear 신호가 0이 들어왔을 경우, 내부 레지스터를 0으로 초기화
- Schemetic symbol 위에 bar가 존재하는 포트는 이와 같이 negative logic으로 동작함을 의미



CLR 신호가 입력되면 내부 레지스터를 0000으로 업데이트

실습 1. 임의의 Sequence를 가진 Counter 구현

- 목표
  - 일반적인 up/down counter가 아니라 임의의 수열을 표현하는 counter 구현
- 실습 내용
  - 000 -> 010 -> 011 -> 101 -> 110 -> 000 의 상태를 갖는 3-bit counter 구현
  - **앞서 설명한 counter logic design 방법론을 이용하여 structural 하게 구현**
  - Self-starting counter인지 확인하고, 아니라면 self-starting 되도록 구현할 것
  - 첫 state로 001, 100, 111을 입력한 다음 시뮬레이션을 통해 결과값 확인해본다.
- 제출 사항
  - 소스코드 및 시뮬레이션 결과 제출

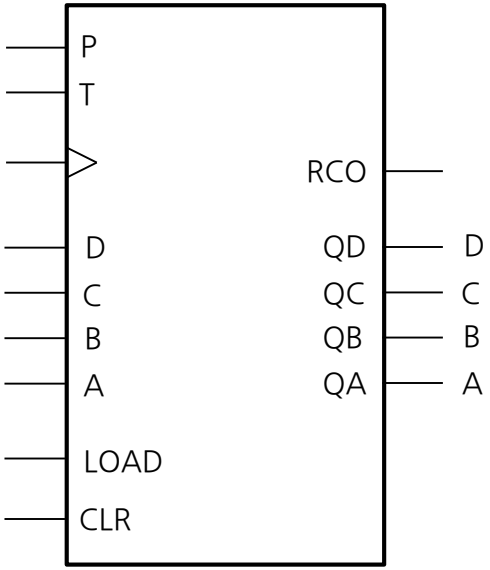


실습 2 – 4-bit Catalog Counter 구현

- 목표
  - Clear, load, enable 기능을 모두 가진 catalog counter 구현
- 실습 내용
  - 이전 슬라이드 내용을 참고하여 clear, load, enable 기능을 모두 가진 catalog counter를 구현해 본다.
  - 다음 슬라이드에 명시된 세부 사항을 만족시키도록 한다.
  - 시뮬레이션을 통해 동작이 예상한 결과와 같은 지 확인해본다.
- 제출 사항
  - 소스코드 및 시뮬레이션 결과 제출

실습 2 – 4-bit Catalog Counter block diagram

- 구현 세부 사항 및 4-bit Catalog Counter의 block diagram
  - P & T가 모두 1이어야 enable한다.
  - 1110 -> 1111 상태로 변화할 때 RCO (Ripple Carry Out)을 1로 내보낸다.
  - 1111 다음 상태는 0000 으로 초기화한다.
  - CLR (clear) 1일 때는 0000 초기화, LOAD 1일 때는 DCBA 값으로 초기화한다.
    - CLR 신호가 LOAD에 우선

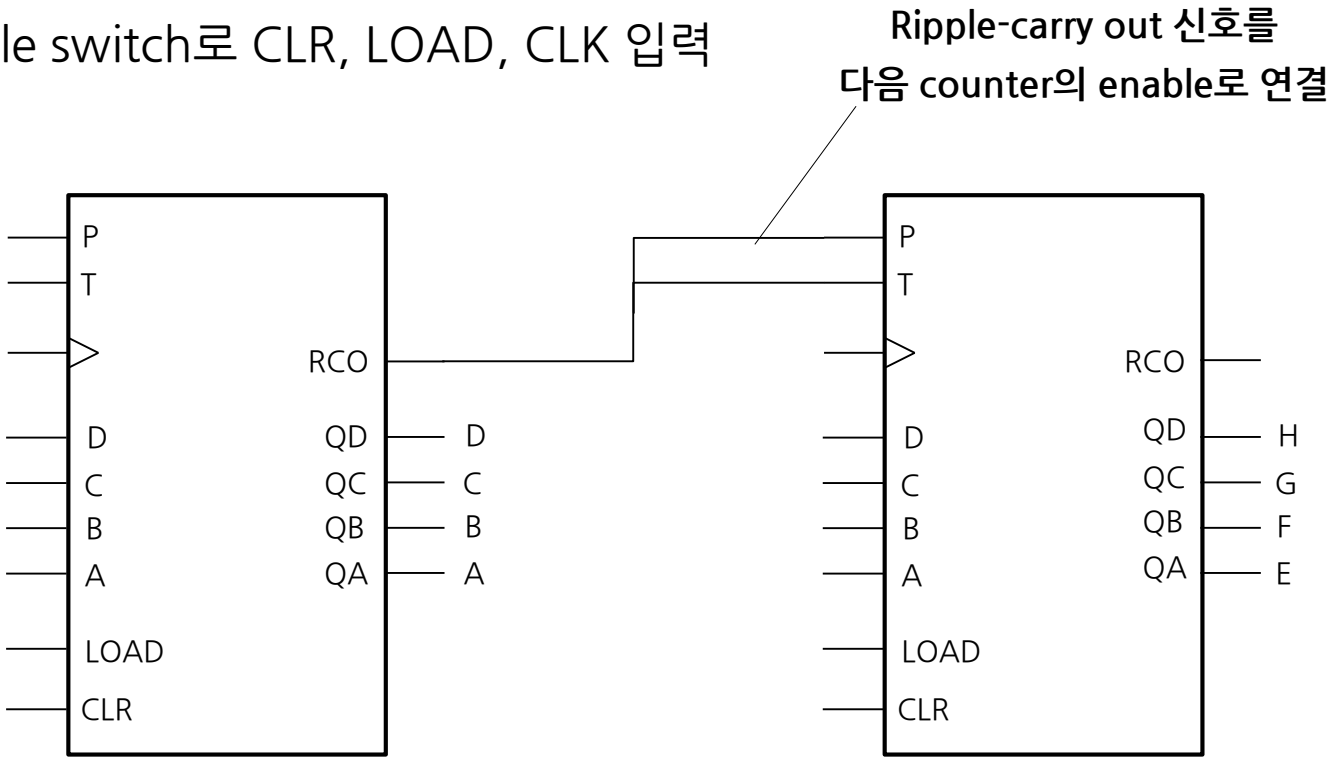


실습 3 – 8-bit Catalog Counter 구현

- 목표
  - 4-bit Catalog counter 2개를 이용하여 8-bit Catalog Counter 구현하기
- 실습 내용
  - 앞서 구현한 4-bit Catalog Counter 2개를 연결하여 8-bit Catalog Counter를 구현해본다.
  - 다음 슬라이드에 명시된 세부 사항을 만족시키도록 한다.
  - 먼저, 시뮬레이션을 통해 동작이 예상한 결과와 같은 지 확인해본다.
  - SNU Logic Design 보드를 사용하여 구현해본다.
- 제출 사항
  - **조교에게 보드 동작 검사**
  - 소스코드 및 시뮬레이션 결과 제출

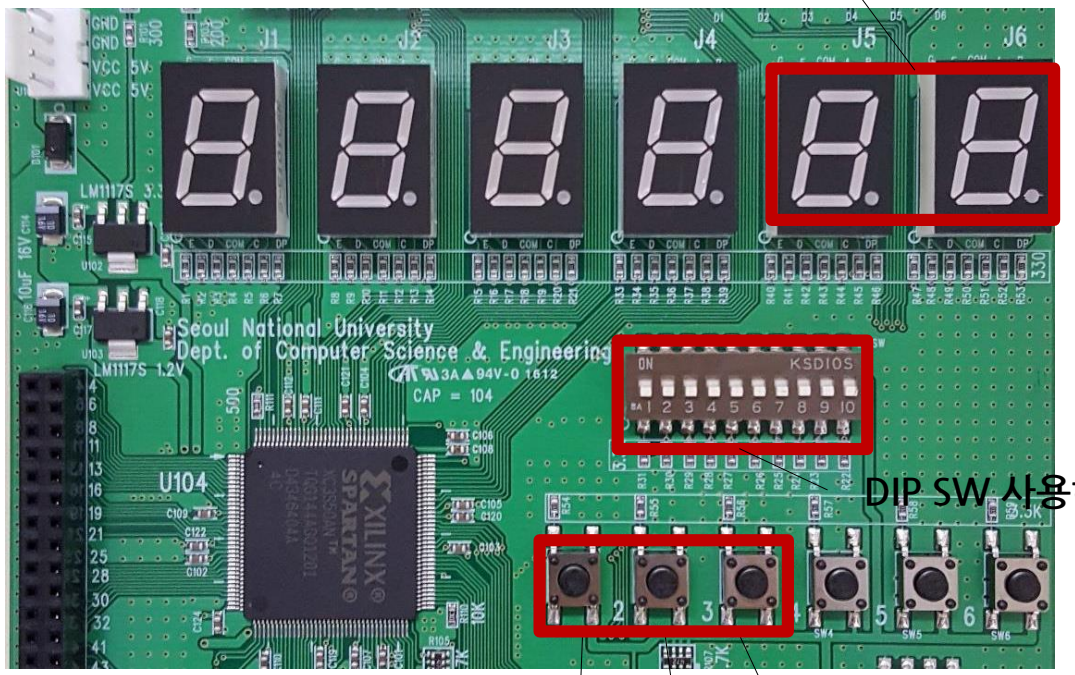
실습 3 – 8-bit Catalog Counter block diagram

- 구현 세부 사항 및 8-bit Catalog Counter의 block diagram
  - 첫번째 counter의 enable 신호(P & T)는 항상 1로 고정
  - 두번째 counter의 enable 신호는 아래 그림을 참고
  - DIP switch를 load input으로 사용
  - Tactile switch로 CLR, LOAD, CLK 입력



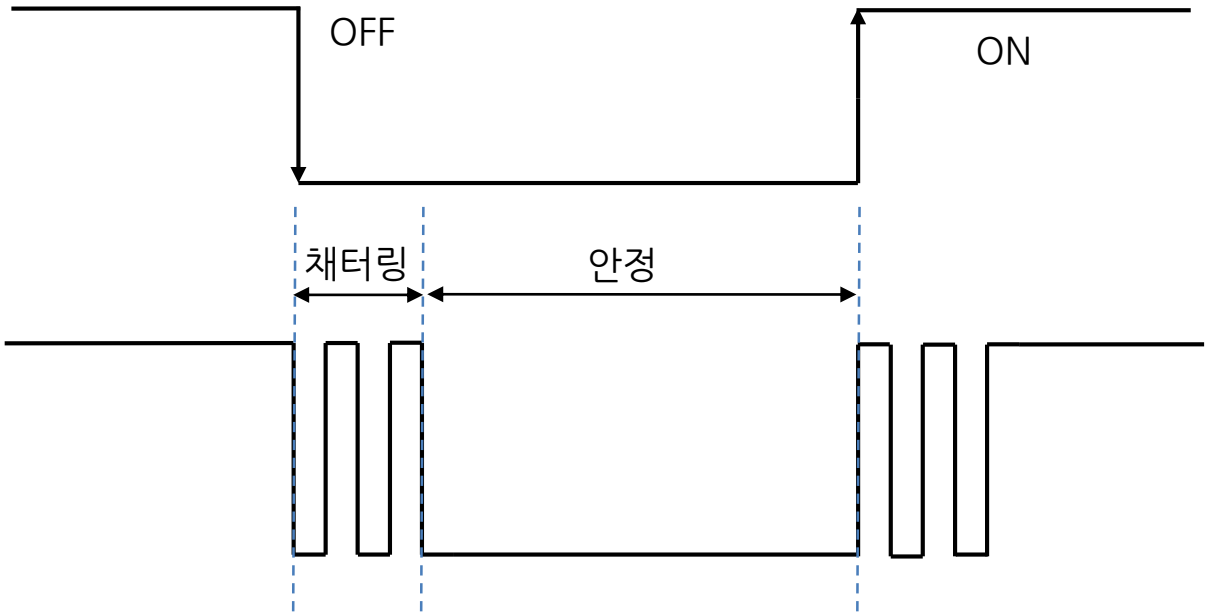
실습 3 – 8-bit Catalog Counter SNU 보드 구현

- CLK입력 시 배포하는 debouncer 모듈을 사용하여 채터링 현상 억제



채터링(chattering) 문제

- **채터링(chattering)**
  - 스위치가 열려있는 상태에서 닫거나 닫혀있는 상태에서 열 때, 즉, 스위치의 상태가 변하는 순간 발생
  - 10ms 이내의 열림과 닫힘이 수회 반복되는 현상
  - 입력된 SW 신호를 별도로 배포된 debouncer 모듈을 사용하여 채터링 방지



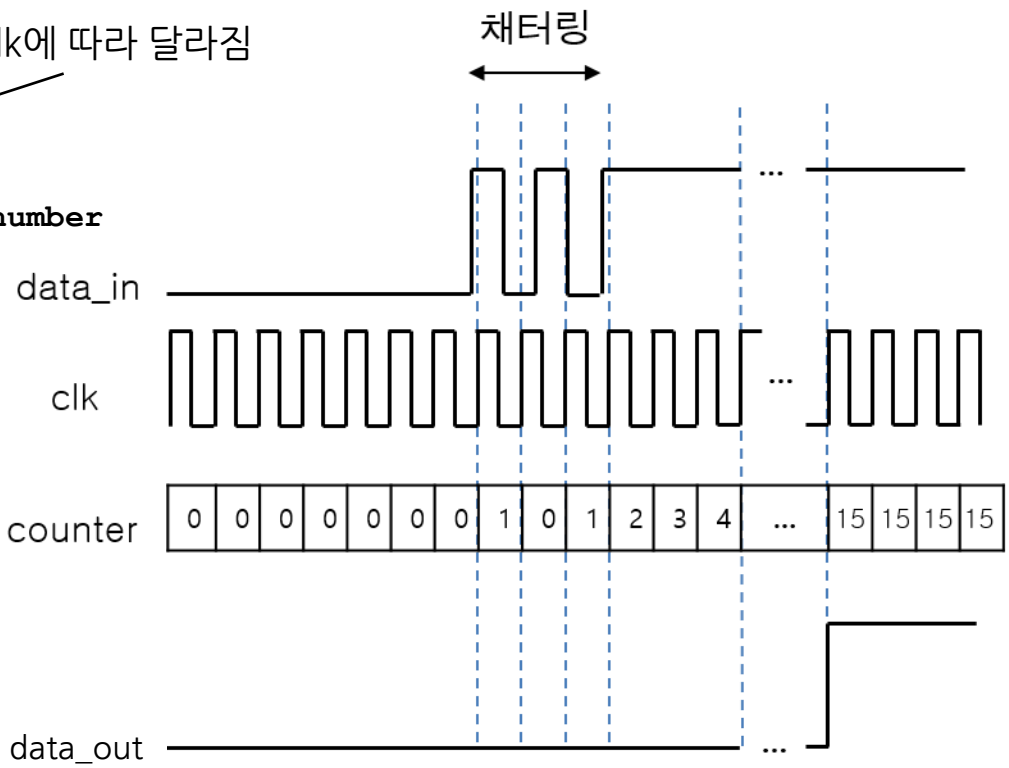
Chattering 문제의 해결

[실습3]

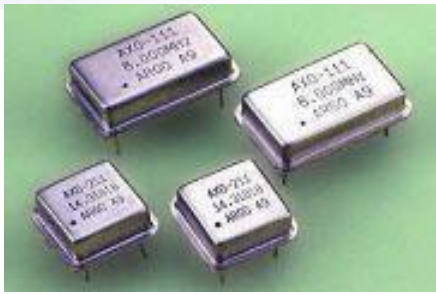
- Counter를 사용하여 chattering을 해결하는 방법

```
module Debouncer(
    input clk,
    input data_in,
    output reg data_out
);

reg [3:0] counter = 4'b0;

always @(posedge clk)
begin
    if (!data_in) counter = 0;
    else
    begin
        if (counter < N) // replace N to number
        begin
            counter = counter + 1;
        end
    end

    if (counter >= N ) data_out = 1; // N
    else               data_out = 0;
end

endmodule
```

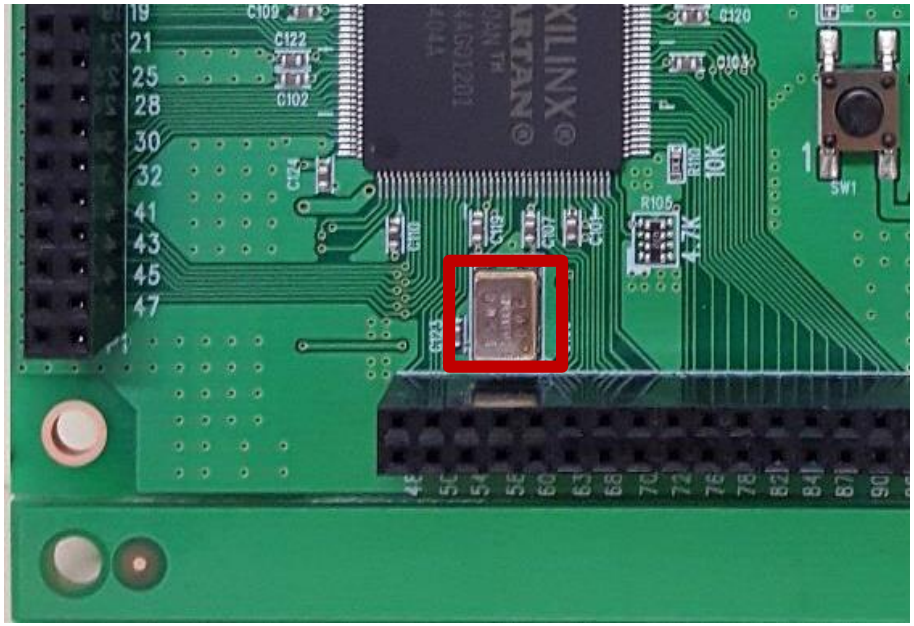
이 값은 clk에 따라 달라짐



보드에서의 clock 사용방법



- Clock 입력을 위해 오실레이터(oscillator) 사용
  - 전원을 공급받아 회로에 적합한 구형파를 출력해주는 소자
  - 다양한 주파수대의 제품이 존재
  - SNU Logic Design 보드의 경우 50MHz의 사용자용 오실레이터 존재 (P57 핀)



실습 제출 안내

- **제출 항목**
  - 실습지 참고하여 보고서 작성
  - 각 실습에서 작성한 Verilog 소스 코드

- **제출 방법 및 기한**
  - 작성한 보고서(PDF) 및 소스코드를 압축하여 하나의 파일로 제출
  - ETL 과제 게시판에 **팀 별로 제출**
  - **일요일 오후 6시**까지

실습 1. D Flip-Flop Verilog code

- 첫 state를 설정하기 위한 초기값이 0인 D Flip-Flop과 1인 D Flip-Flop을 이용

```
module D_FlipFlop0(
    input D,
    input CLK,
    output reg Q
    );
initial begin
    Q <= 1'b0;
end
always @(posedge CLK)
begin
    Q <= D;
end
endmodule
```

```
module D_FlipFlop1(
    input D,
    input CLK,
    output reg Q
    );
initial begin
    Q <= 1'b1;
end
always @(posedge CLK)
begin
    Q <= D;
end
endmodule
```

Backup slide : Counter Using Verilog

```
module up-counter(
    input clk,
    output [2:0] out_cnt
);
reg [2:0] state = 3'b000;
assign out_cnt = state;

always @(posedge clk)
begin
    case (state)
    begin
        3'b000: state = 3'b001;
        3'b001: state = 3'b010;
        3'b010: state = 3'b011;
        3'b011: state = 3'b100;
        3'b100: state = 3'b101;
        3'b101: state = 3'b110;
        3'b110: state = 3'b111;
        3'b111: state = 3'b000;
    end
end
endmodule
```

- Verilog의 case문을 활용하여 구현 가능
  - 상태 표(state table)를 토대로 case 문 사용

C	B	A	C+	B+	A+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

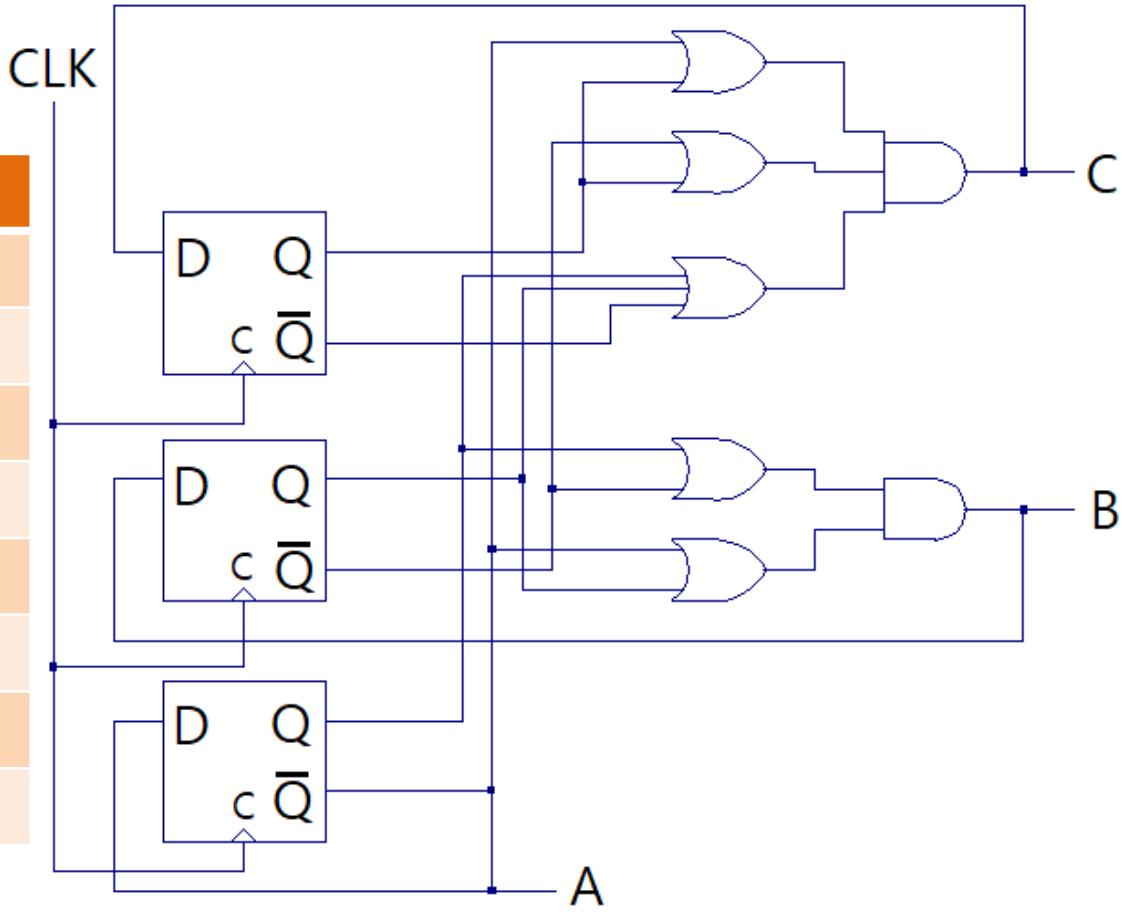
Next-state logic implementation

[실습1]

- K-map 결과를 이용하여 최종적으로 3-bit up-counter의 구현

C	B	A	C+	B+	A+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

3-bit up-counter의 state table



3-bit up-counter의 circuit diagram

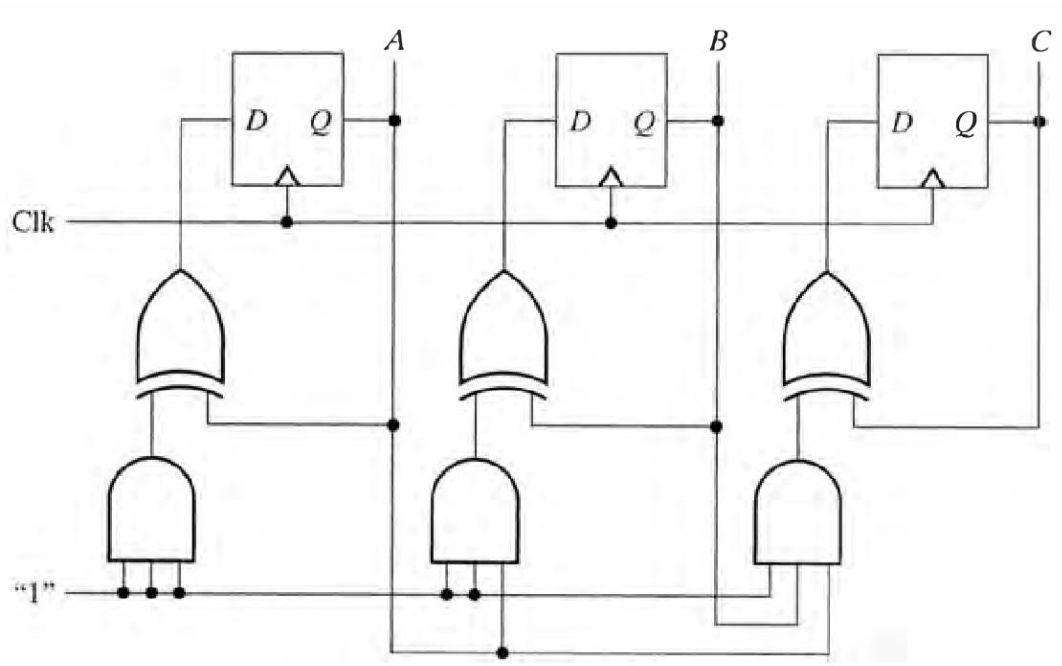
Next-state logic implementation

[실습1]

- K-map 결과를 이용하여 최종적으로 3-bit up-counter의 구현

C	B	A	C+	B+	A+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

3-bit up-counter의 state table



3-bit up-counter의 circuit diagram