DataLab. Internship Program (2023 Summer)

Week 3

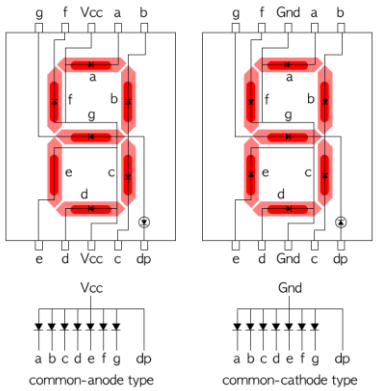
Tutor: 배은태

실습: FND Counter

이번 예제에서는 Nexys A7 보드에 장착된 FND(또는 7-세그먼트)의 동작을 이해하고 조합회로와 순차회로를 조합하여 이를 제어하는 컨트롤러를 설계합니다.

Nexys A7 보드에는 두 개의 4-digit common anode(공통 애노드 방식) seven-segment LED display가 내장되어 있습니다. seven-segment 디스플레이는 FND(Flexible Numeric Display)라고도 부릅니다.

FND에는 공통 애노드(common anode) 방식과 공통 캐소드(common cathode) 방식이 존재합니다. Nexys 보드에는 공통 애노드 방식의 FND를 사용합니다. Common anode 방식의 FND는 LED의 각 위치를 나타내는 A부터 G까지의 신호가 LOW일 때 LED가 켜지는 방식입니다. 반대로 common cathode 방식은 HIGH일 때 켜지는 방식입니다. 이에 따라 숫자를 표현하기 위한 A부터 G까지의 7비트 신호에 대한 진리표는 다음과 같습니다. 십진수 입력을 이용하여 FND 출력 신호로 변환하는 디코더 회로에 대해서는 뒤에서 자세히 다루도록 하겠습니다.



	a	b	c	d	e	f	g
0							1
1	1			1	1	1	1
2			1			1	
3					1	1	
4	1			1	1		
5		1			1		
6		1					
7				1	1	1	1
8							
9					1		

공통 애노드 타입

	a	b	c	d	e	f	g
0	1	1	1	1	1	1	
1		1	1				
2	1	1		1	1		1
3	1	1	1	1			1
4		1	1			1	1
5	1		1	1		1	1
6	1		1	1	1	1	1
7	1	1	1				
8	1	1	1	1	1	1	1
9	1	1	1	1		1	1

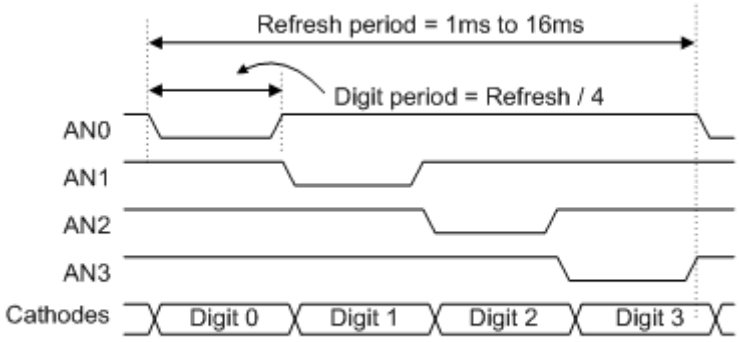
공통 캐소드 타입

FND의 입력신호

첫 번째 그림과 같이 Nexys 보드의 FND는 8비트의 AN이라는 신호와 7비트의 CA부터 CG까지의 신호를 입력으로 받습니다. 실제로 XDC 파일을 열어보면 다음과 같이 FND의 핀 정보가 정의되어 있는 것을 볼 수 있습니다.

```
55  ##7 segment display
56  #set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { CA }]; #IO_L24N_T3_A00_D16_14 Sch=ca
57  #set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { CB }]; #IO_25_14 Sch=cb
58  #set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { CC }]; #IO_25_15 Sch=cc
59  #set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { CD }]; #IO_L17P_T2_A26_15 Sch=cd
60  #set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { CE }]; #IO_L13P_T2_MRCC_14 Sch=ce
61  #set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { CF }]; #IO_L19P_T3_A10_D26_14 Sch=cf
62  #set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { CG }]; #IO_L4P_T0_D04_14 Sch=cg
63  #set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
64  #set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { AN[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
65  #set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { AN[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
66  #set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { AN[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
67  #set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { AN[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
68  #set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { AN[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
69  #set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { AN[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
70  #set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { AN[6] }]; #IO_L23P_T3_35 Sch=an[6]
71  #set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { AN[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
```

CA부터 CG까지의 7비트 신호는 각 자리의 FND에 "무엇이 표시될지"를 결정합니다. 이때 주의해야 할 점은 모든 8자리의 FND가 동시에 출력되는 것이 아니라는 사실입니다. CA부터 CG 신호선을 공유하기 때문에 한 시점에 "어떤 FND를 사용할지" 역시 결정해야 합니다. 이를 위해 사용하는 AN0부터 AN7까지의 8비트 선택 신호입니다. FND의 전환이 충분히 빠르게 이뤄진다면 8개 FND가 동시에 출력되는 듯한 착시를 제공할 수 있을 것입니다.



종합하면, 보드의 FND 소자에는 7비트의 CA-CG 신호와 8비트의 AN[7:0] 신호가 입력되어야 합니다. 따라서 예제에서 설계하고자 하는 FND 컨트롤러는 이 두 가지 신호에 대응하는 출력 포트 seg와 an을 가집니다. seg는 LED의 출력을, an은 FND의 선택을 제어합니다.

```
module FNDCounter(input clk, input rstn, output [6:0] seg, output reg [7:0] an);
```

그리고 이 두 신호는 FND의 핀과 매핑됩니다. DP는 LED의 점(.) 기호를 의미합니다. 이번 예제에서 DP 신호는 사용하지 않겠습니다.

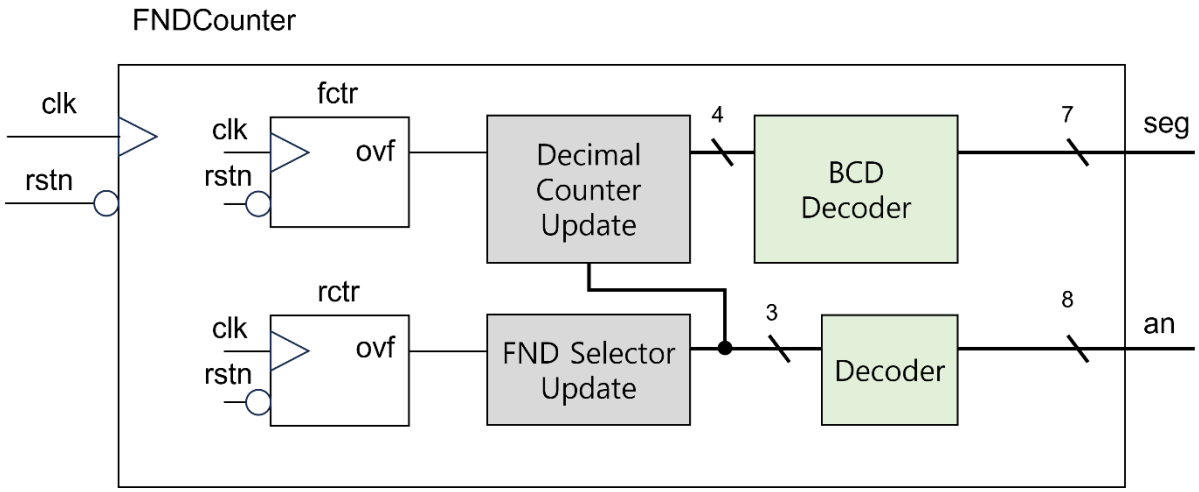
```
##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { seg[6] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { seg[5] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { seg[4] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { seg[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { seg[2] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { seg[1] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { seg[0] }]; #IO_L4P_T0_D04_14 Sch=cg
#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { an[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { an[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { an[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { an[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { an[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { an[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { an[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { an[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
```

clk와 rstn은 이전 예제와 마찬가지로 시스템 클럭과 CPU RESET을 의미합니다.

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk}];

##CPU Reset Button
set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { rstn }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
```

마지막으로, FND에 어떤 내용을 출력할지도 결정해야 합니다. 이번 예제에서는 편의상 일정 주기마다 카운트 업 되는 십진 카운터 값을 이용하겠습니다. 따라서 모듈 FNDCounter의 구성은 다음 그림과 같습니다. 각 컴포넌트의 자세한 내용은 다음 절에서 설명하겠습니다.



FND Selector Update

예제에서는 십진 카운터와 FND selection 신호(AN)을 일정한 주기로 제어하기 위해 각각 두 개의 서로 다른 오버플로우 주기를 가지는 카운터 fctr과 rctr을 사용합니다. 이 절에서는 현재 사용될 FND를 선택하는 신호인 AN[7:0]을 결정하는 rctr에 대하여 설명하겠습니다.

8개 FND가 한 바퀴 다 도는 데 걸리는 주기를 refresh period라고 말하고, $T_{refresh}$라고 표시하

기로 합시다. $T_{refresh}$가 16ms만 되어도 60Hz 이상의 주사율을 제공할 수 있습니다. 그리고 이를 만족하려면 AN 신호를 $\frac{T_{refresh}}{8}$ 주기로 갱신해주기만 하면 됩니다. 이 갱신 주기를 일정하게 제어하기 위해 사용되는 카운터가 rctr(Refresh Counter)입니다. 예제에서는 넉넉하게 10ms를 refresh period로 잡겠습니다. Nexys 보드의 클럭 속도가 100MHz이므로, 사이클 주기가 10ns인 점을 고려하여, rctr를 instantiate를 할 때 parameter인 RATE를 125000으로 설정합니다. rctr은 지정된 RATE에 따라 일정하게 오버플로우 신호인 refreshovf를 출력합니다.

[Counter24.v]

```
module Counter24 #(parameter RATE = 1000000)(sysclk, rstn, ovf);
    input sysclk;
    input rstn;
    output reg ovf;
    reg [23:0] cnt;

    always @(posedge sysclk or negedge rstn) begin
        if (!rstn) begin
            cnt <= 0;
            ovf <= 0;
        end
        else begin
            if (cnt == RATE) begin
                ovf <= 1;
                cnt <= 0;
            end
            else begin
                ovf <= 0;
                cnt <= cnt + 1;
            end
        end
    end
endmodule
```

[FNDCounter.v] (일부)

```
module FNDCounter(clk, rstn, seg, an);
    // ...
    reg [2:0] fndsel;
    wire refreshovf;
    Counter24 #(.RATE(125000)) rctr(clk, rstn, refreshovf); // 10/8ms
    // ...
endmodule
```

현재 출력할 FND는 refreshovf가 발생할 때마다 교체됩니다. 8개 FND 중 하나를 선택하기 위해 3비트 레지스터인 fndsel를 정의했습니다. 이 3비트 신호는 이후 8비트 신호인 an으로 디코딩 되어야 합니다.

```
    // FND select update
    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            fndsel <= 0;
        end
        else begin
            if (refreshovf)
                fndsel <= fndsel + 1;
        end
    end
```

Decimal Counter Update

이번 예제에서는 십진수 카운터를 화면에 띄울 정보로 사용합니다. 우선 카운터로 사용할 레지스터가 있어야 합니다. 예제에서는 넉넉하게 32비트 레지스터인 fndcnt를 정의했습니다. 그리고 카운터의 갱신 주기를 제어하는 또 다른 카운터가 있어야 합니다. 이 카운터가 fctr이라고 이전 절에서 언급한 바 있습니다. rctr과 마찬가지로 오버플로우 신호를 fndovf라는 wire로 연결하겠습니다. 십진 카운터를 업데이트 하는 코드도 FND를 선택하는 코드와 마찬가지로 그렇게 복잡하지 않습니다. 십진 카운터의 업데이트 주기는 편의상 10ms로 하겠습니다. rctr과 마찬가지로 Counter24 모듈을 사용합니다. parameter의 기본값이 1000000이므로, RATE 값을 별도로 새로 지정(override)하지는 않았습니다. BCDDecoder랑 bcd라고 하는 레지스터에 대해서는 바로 다음 절에서 이어서 설명하겠습니다.

```
module FNDCounter(clk, rstn, seg, an);
    // ...
    reg [31:0] fndcnt;  // Decimal up-counter
    reg [2:0]  fndsel;  // FND selection
    reg [3:0] bcd;

    wire fndovf;
    wire refreshovf;

    Counter24                   fctr(clk, rstn, fndovf);     // 10ms
    Counter24 #(.RATE(125000)) rctr(clk, rstn, refreshovf); // 10/8ms
    BCDDecoder dec(bcd, seg);

    // ...

    reg [31:0] fndcnt;
    // FND Decimal counter update
    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            fndcnt <= 0;
        end
```

```
        else begin
            if (fndovf)
                fndcnt <= fndcnt + 1;
        end
    end
```

BCD to 7-Segment Decoder (74LS47)

8자리의 FND에 8자리 십진수 값을 표현하기 위해 fndcnt 값을 그대로 사용할 수 없다는 점에 주의해야 합니다. 첫째는 각 자리에 해당하는 십진수 숫자를 알아야 한다는 점입니다. 디지털 시스템에서 내부적으로 사용하는 데이터는 이진수이기 때문에 2진수나 8진수 16진수와 같이 각 자리 숫자를 단순히 n개 비트로 묶어서 표현할 수 있는 경우에는 변환 과정이 복잡하지 않습니다. 원하는 자리 숫자를 구하려면 단지 거기에 해당하는 비트 영역을 긁어오면 되기 때문입니다. 하지만 10진수는 그렇지 않습니다. 각 자리 숫자를 얻어오려면 산술 연산이 동원되어야 이 과정은 앞서 언급한 8, 16진수보다 훨씬 복잡합니다.

둘째는 그렇게 얻어온 십진수 숫자를 다시 FND의 LED on/off 신호로 매핑해야 한다는 점입니다. 이를 위해 예제에서는 BCD to 7-segment decoder를 구현할 것입니다. 이 BCD 디코더는 4비트 BCD 코드를 입력으로 받아서 7비트의 FND 입력 신호를 출력하는 조합회로입니다. Common anode type의 FND용으로는 7447이라는 칩으로 생산되고 있습니다.

BCD란 십진수 문자를 컴퓨터 내부적으로(즉, 디지털 신호로) 표현하기 위해 고안된 인코딩(부호화) 방식을 의미합니다. BCD는 십진수 하나를 표현하기 위해 4비트의 고정 길이 데이터를 사용합니다. 예를 들어, 2945라는 수가 있다면, 16진수로는 0x0B81(0b0000101110000001)이 됩니다. fndcnt에는 이런 방식으로 숫자가 저장됩니다. 반면에 BCD 코드로 2945를 나타내면 00101001010001010이 됩니다.

Decimal	Binay (BCD) 8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

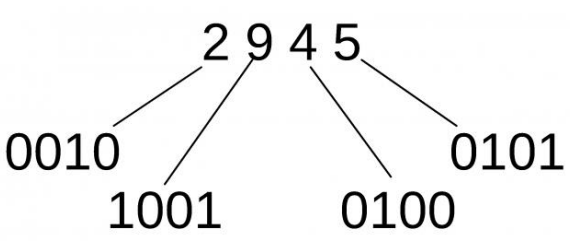
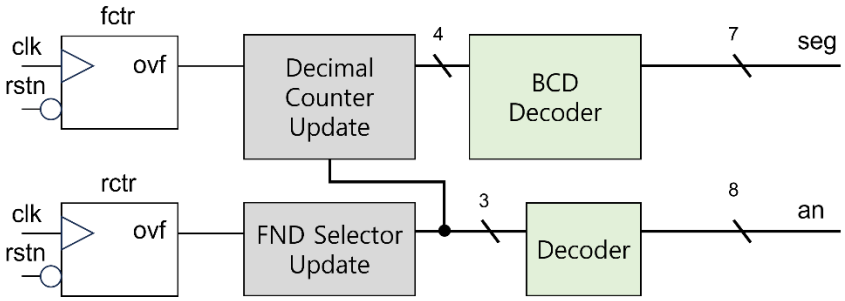


그림 출처: https://www.electroniclinic.com/codes-in-digital-electronics-bcd-excess-3-error-detection-ascii-gray-code/

예제에서는 원하는 자리의 BCD 코드를 얻은 다음, BCD Decoder를 거쳐 다시 7비트의 출력 신호인 seg를 얻어내는 방식으로 구현하겠습니다. 따라서, 이때 어떤 FND가 선택되었는지에 대한 정보가 필요하다는 점에 주의해야 합니다. 즉, Decimal Counter Update과정에는 카운터 값인 룽춧는 물론이고 fndsel의 값도 출력에 영향을 미칩니다.



다음 코드는 현재 자리의 BCD 코드 계산과 3비트 fndsel을 8비트의 AN으로 디코딩하는 기능을 모두 포함하고 있습니다.

```
    always @(*) begin
        case (fndsel)
            3'b000: begin
                an = 8'b1111_1110;
                bcd = fndcnt % 10;
            end
            3'b001: begin
                an = 8'b1111_1101;
                bcd = (fndcnt / 10) % 10;
            end
            3'b010: begin
                an = 8'b1111_1011;
                bcd = (fndcnt / 100) % 10;
            end
            3'b011: begin
                an = 8'b1111_0111;
                bcd = (fndcnt / 1000) % 10;
            end
            3'b100: begin
                an = 8'b1110_1111;
                bcd = (fndcnt / 10000) % 10;
            end
            3'b101: begin
                an = 8'b1101_1111;
                bcd = (fndcnt / 100000) % 10;
            end
            3'b110: begin
                an = 8'b1011_1111;
                bcd = (fndcnt / 1000000) % 10;
            end
            3'b111: begin
                an = 8'b0111_1111;
                bcd = (fndcnt / 10000000) % 10;
            end
```

```
            default: begin
                an = 8'b1111_1110;
                bcd = fndcnt % 10;
            end
        endcase
    end
```

- 지금까지 다룬 내용을 바탕으로 직접 Bitstream을 생성하고 결과를 확인해봅시다.
- 소스코드는 맨 뒤 **부록** 절에서 확인하실 수 있습니다.

추가: Mux

지금까지는 십진 카운터를 10ms의 고정된 단위로 업데이트 했습니다. 이번에는 Counter24 서브 모듈을 하나 더 instantiate하여 스위치에 따라 카운팅 속도를 제어할 수 있는 회로를 구현해봅시다. RATE가 절반인 500000으로 설정된 fctr2를 새로 추가했음에 주목하세요. 기존의 fctr은 fctr1로 구분하겠습니다.

```
    wire fndovf;
    wire fndovf1;
    wire fndovf2;
    wire refreshovf;

    Counter24                   fctr1(clk, rstn, fndovf1);   // 10ms
    Counter24 #(.RATE(500000)) fctr2(clk, rstn, fndovf2);   // 5ms
    Counter24 #(.RATE(125000)) rctr(clk, rstn, refreshovf); // 10/8ms
    BCDDecoder dec(bcd, seg);
```

십진 카운터 갱신을 제어하는 데에는 두 카운터의 출력인 fndovf1과 fndovf2 둘 중 하나의 신호만 사용됩니다. 둘 중 하나로 선택된 신호를 **fndovf**라고 두겠습니다.

```
  // FND Decimal counter update
    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            fndcnt <= 0;
        end
        else begin
            if (fndovf)
                fndcnt <= fndcnt + 1;
        end
    end
```

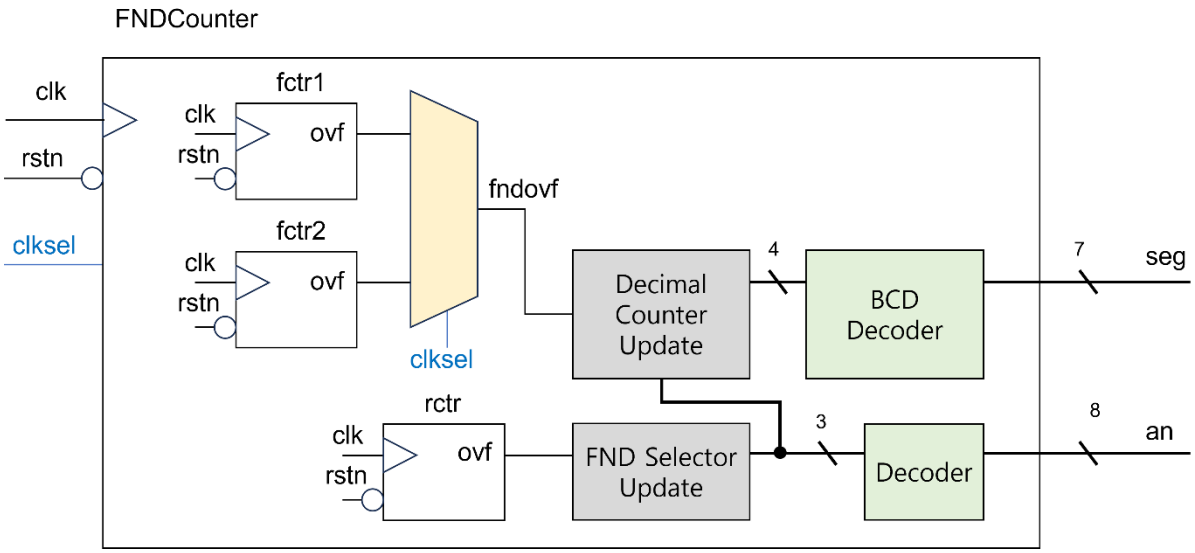
그리고 둘 중 하나를 선택하기 위해 제어신호인 clksel이라는 신호를 새로 도입하겠습니다.

```
module FNDCounter(clk, rstn, clksel, seg, an);
```

clksel은 스위치 SW[0]를 통해 제어할 수 있도록 XDC의 매핑 정보를 다음과 같이 수정합니다.

```
10
11  ##Switches
12  set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { clksel }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
```

이제 이 clksel을 통해 두 카운터 중 하나를 선택하도록 구현해야 합니다. 여기에 사용되는 회로가 멀티플렉서(Mux)입니다. Mux 역시 일종의 조합회로로 assign문을 통해 구현할 수 있습니다.




```
    assign fndovf = (clksel) ? fndovf2 : fndovf1;
```

부록: 소스코드

이번 예제는 week2/src/FNDCounter에 있습니다. clksel과 Mux를 추가하지 않은 버전은 FNDCounter_v1.v와 Nexys-A7-100T-Master_v1.xdc라는 별도의 소스 파일로 구분했습니다.

https://github.com/euntae-bae/2023-summer-internship/tree/main/week2/src/FNDCounter