Embedded System Design
실습 8

Cho Yeongpil
Hanyang University

1. ISE를 이용한 FPGA 설계이해

1.FPGA 이해

- **FPGA (Field Programmable Gate Array)**
  - CLB (Configurable Logic Block으로 Slice, LUT, F/F으로 구성) 구성
  - 사용자의 목적에 맞게 사용하기 위한 DCM, 곱셈기, 메모리 등 구성
  - Schematic 또는 HDL로 디자인

1.FPGA 이해

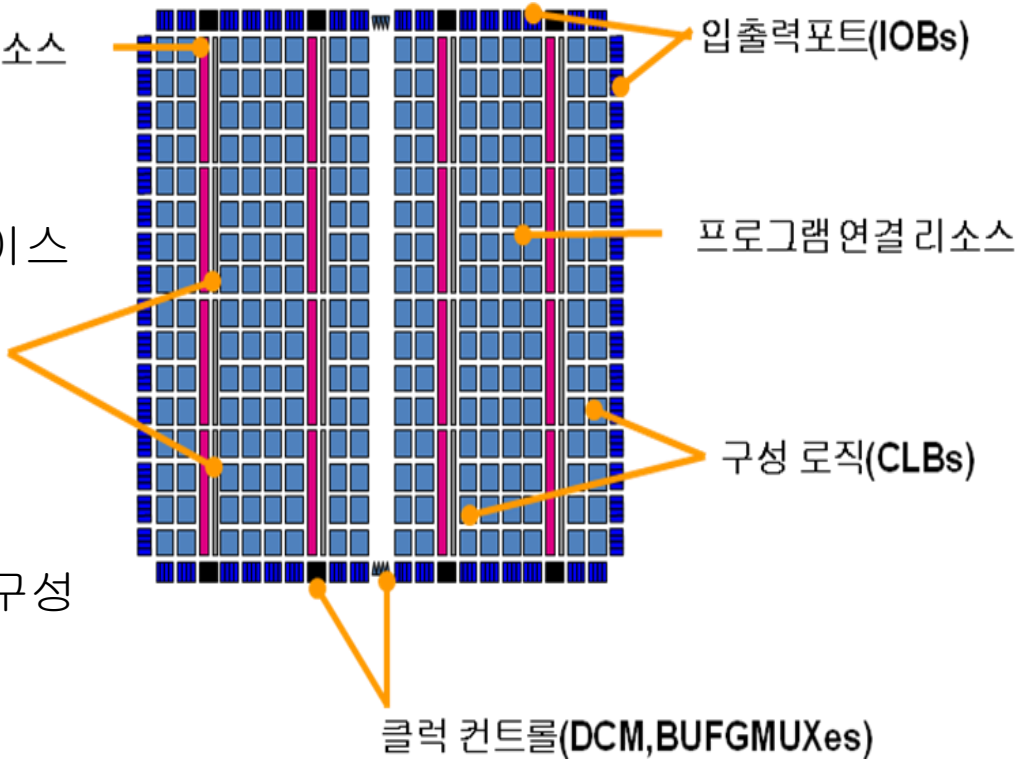
- **CPLD(Complex Programmable Logic Device)**
  - CPLD와 FPGA는 근본적인 구조가 다름
  - Xilinx의 CPLD는 기본 macro cell 단위로 구성 최종단 F/F 구성
  - CPLD의 로직은 macro cell 단위로 표시하며 그 숫자가 F/F 개수와 같다.
  - CPLD는 구조적으로 wafer 면적을 크게 사용하여 용량을 늘리는데 한계 있음
  - CPLD 이름은 macro cell 숫자 단위로 이름을 붙이는 경향.(예, XC9436, XC95288)
  - 디바이스가 간단한 구조이어서 Timing 예측이 쉽다.
  - FPGA에 비해 저전력

1.FPGA 이해

- **FPGA**
  - FPGA는 구조적으로 직접화하기 좋음.
  - CLB (Configurable Logic Block으로 Slice, LUT, F/F으로 구성) 구성
  - FPGA의 논리 표현은 LUT에서 저장기능은 F/F에서 한다.
  - 크게 Virtex의 고기능의 디바이스와 Spartan의 양산용 디바이스로 구분.
  - 사용자의 목적에 맞게 사용하기 위한 DCM, 곱셈기, 메모리, DSP, PPC 등 구성
  - 동작 Timing 의 예측이 어려워 Timing Simulation 필요.

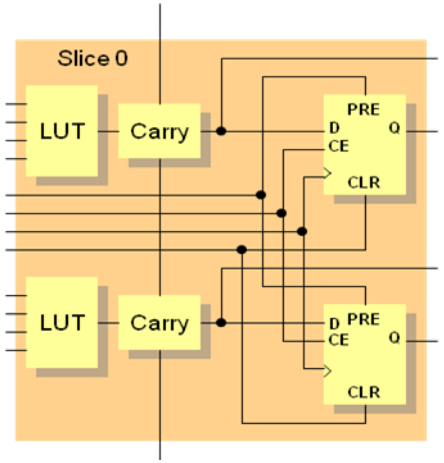
1.FPGA 이해

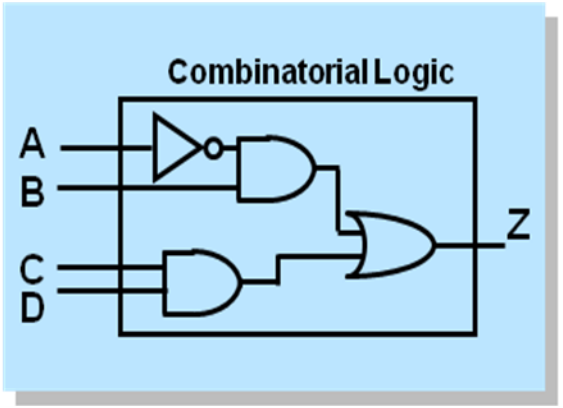
- **FPGA구성물**
  - CLB : 기본 로직 구성
  - 곱셈기 : 연산 블록
  - 입출력 I/O : 외부 디바이스와 연결
  - DCM : Clock의 변환
  - Memory : 데이터 저장
  - PPC, MGT, DSP Slice 등 구성



1.FPGA 이해

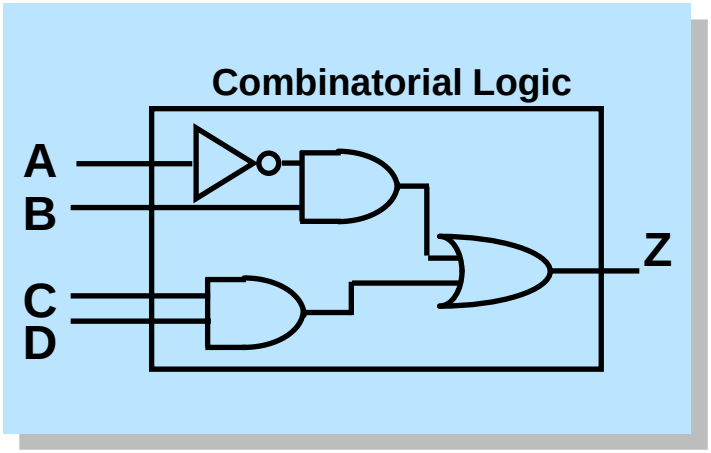
- CLB
  - Slice : Device에 따라 CLB당 2개(Virtex5) 또는4개 Slice
  - LUT : 논리 게이트 담당
    Slice당 2개(spartan 3),
    slice 당 4개 (Virtex5)
    Memory 기능
    Shift Register 기능
  - F/F : 저장 기능
    F/F 또는 latch로 동작





1.FPGA 이해

- **LUT(Look Up Table)**
  - 기능 생성기(Function Generators)라 불리움
  - 로직 소자가 들어가 있지 않다 ex) and or not gate
  - 용량이 크지 않아 입력수가 제한적이다.
  - LUT을 통과하는 딜레이는 일정.



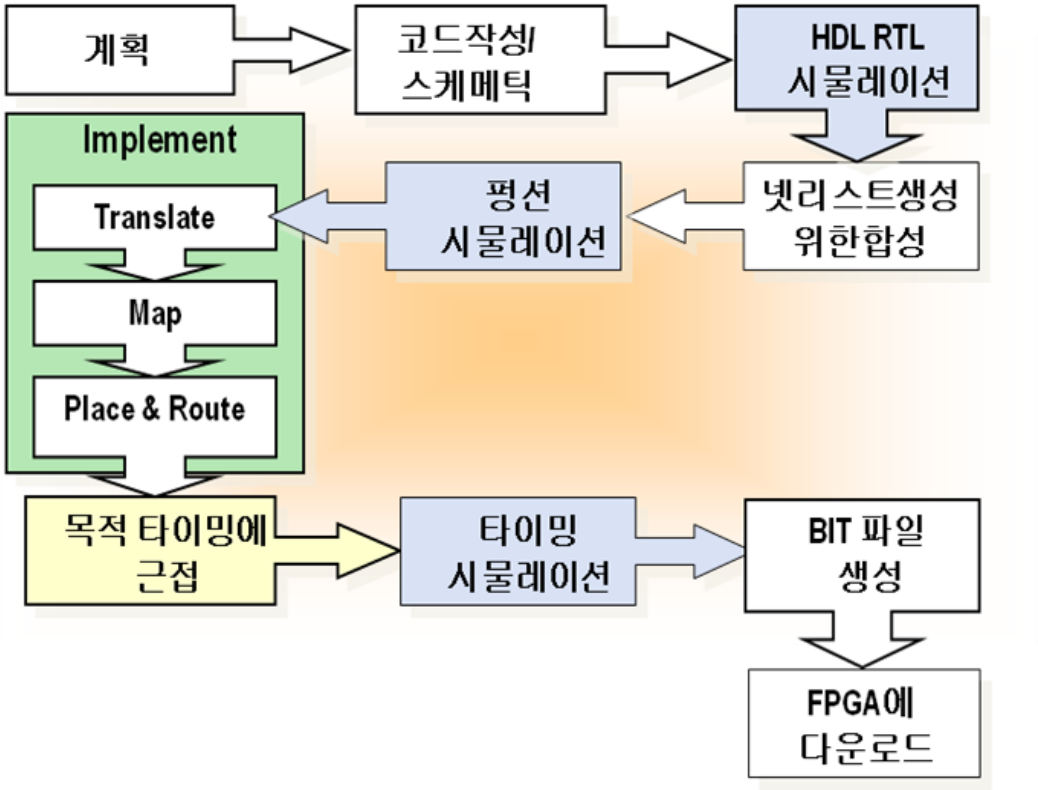
A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
	.	.	.	
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

1.FPGA 이해

- CLB
  - Memory : Block Memory 1블록당 18Kbits (spartan 3)
    36Kbits(Virtex 5)
    Distribute Memory : 16x1(1 LUT)
  - DCM :
    Multiplication 기능 - 입력 clock을 더 빠른 clock으로 변환
    Divide 기능 - 입력 clock을 더 느린 clock으로 변환
    Phase shift - clock의 위상 변환
  - DSP : MAC 구조를 이용한 빠른 연산 기능 제공
  - PPC : 프로세서 이용한 임베디드 S/W 기능
  - MGT : 고속 I/O 인터페이스

1. ISE 기능 이해

- **ISE(Integrated Software Environment)**
  - 디자인 Flow



1. ISE 기능 이해

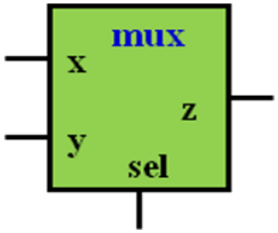
- **ISE(Integrated Software Environment)**
  - 계획 : 전체 디자인 환경이해와 Block Diagram 작성
  - 디자인작성(코드작성) : HDL이용 코드 작성
  - Synthesis(합성) : RTL 코드를 Netlist로 변환
  - Implementation : Netlist이용 FPGA에 로직 적용
    - ucf file(핀 정보파일) 필요
  - Bit File 생성 : FPGA 다운로드 위한 데이터 파일(bit file) 생성
  - FPGA 다운로드 : bit file FPGA에 다운로드 동작 체크.
  - Simulation : RTL, Gate, Timing Simulation으로 구분 디자인 동작 검증 위함.

2. Verilog 예제 디자인 실습

2.Multiplexer 설계

- **Multiplexer 란?**
  - 여러 개의 입력 신호 가운데 하나를 선택하는 기능
  - 선택 핀의 동작에 따라 테이터를 선택
  - 선택 핀에 따라 2n까지 가능

- 우측의 그림과 같이 **2x1 mux**는 **2**개의 입력을 선택적으로 출력을 한다.



입력(X, Y)	선택 핀(sel)	출력(z)
X	1	X
Y	0	Y

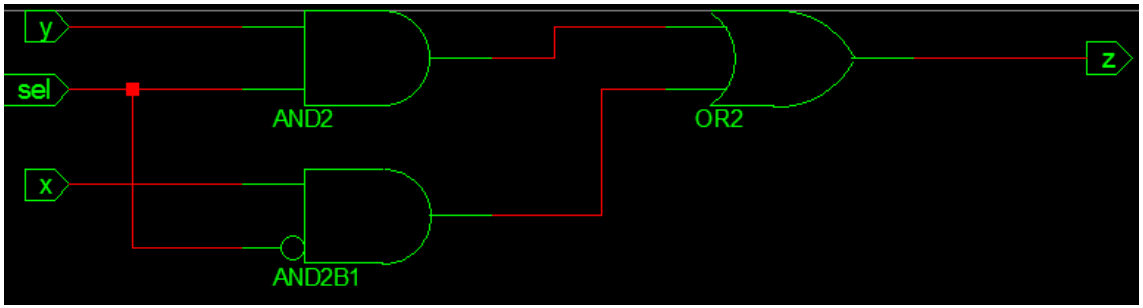
2.Multiplexer 설계

- **Multiplexer Verilog 디자인**

```
`timescale 1 ns / 1ps

module tr_mux  ( x, y, sel, z );
input   x, y ,  sel  ;
output  z ;

reg     z   ;

always @(sel or x or y) begin
    case (sel)
       1'b0    : z = x;
       2'b1    : z = y;
       default : z = 1'b0;
     endcase
 end

endmodule
```

2.Multiplexer 설계

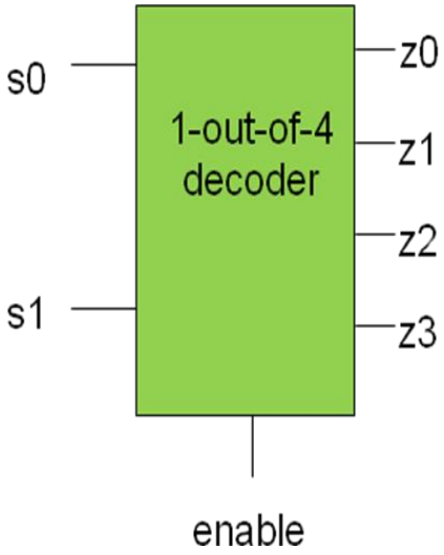
- **Multiplexer synthesis 결과**
  - 이전 로직 디자인
  - Sythesis 후 view RTL Schematic 실행



2.Decoder 설계

- **Decoder 란?**
  - n bit 이력 코드를 m 출력조합은 출력 중에 단 하나만
  - N의 입력은 '0' 또는 '1' 이므로 2n조합 코드로 입력
  - 출력 m은 출력중에 단하나만 '1'이 되고 나머지는 '0'

- 우측의 **2x4 Decoder**



2.Decoder 설계

- **Decoder 진리표**

입력(S0)	입력(S1)	출력(z0)	출력(z1)	출력(z2)	출력(z3)
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

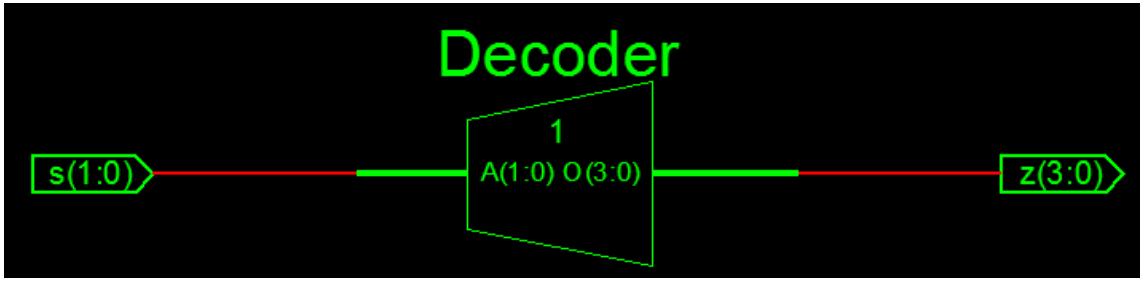
2.Decoder 설계

- **2x4 Decoder 동작 디자인**

```
`timescale 1 ns / 1ps

module tr_decoder ( s, z);
input   [1:0]   s    ;
output [3:0]   z    ;
reg      [3:0]   z   ;

always @(s) begin
   case (s)
        2'b00   : z = 4'b0001;
        2'b01   : z = 4'b0010;
        2'b10   : z = 4'b0100;
        2'b11   : z = 4'b1000;
        default : z = 4'b0000;
   endcase
end
endmodule
```

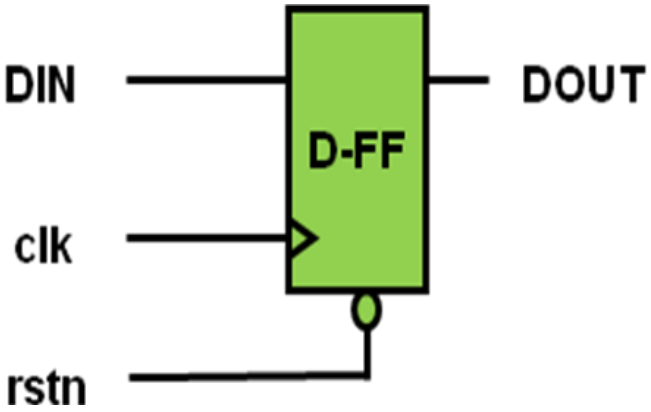
2.Decoder 설계

- **Synthesis 결과**
  - Synthesis 후 View RTL Schematic 실행



2.Register, F/F 설계

- **F/F 동작**
  - D F/F는 clock이 high에서 low로 동작할 때(하강 edge) 또는 low에서 high로 동작할 때(상승 edge)에 입력되는 데이터를 출력
- **F/F 블록도 및 진리표**

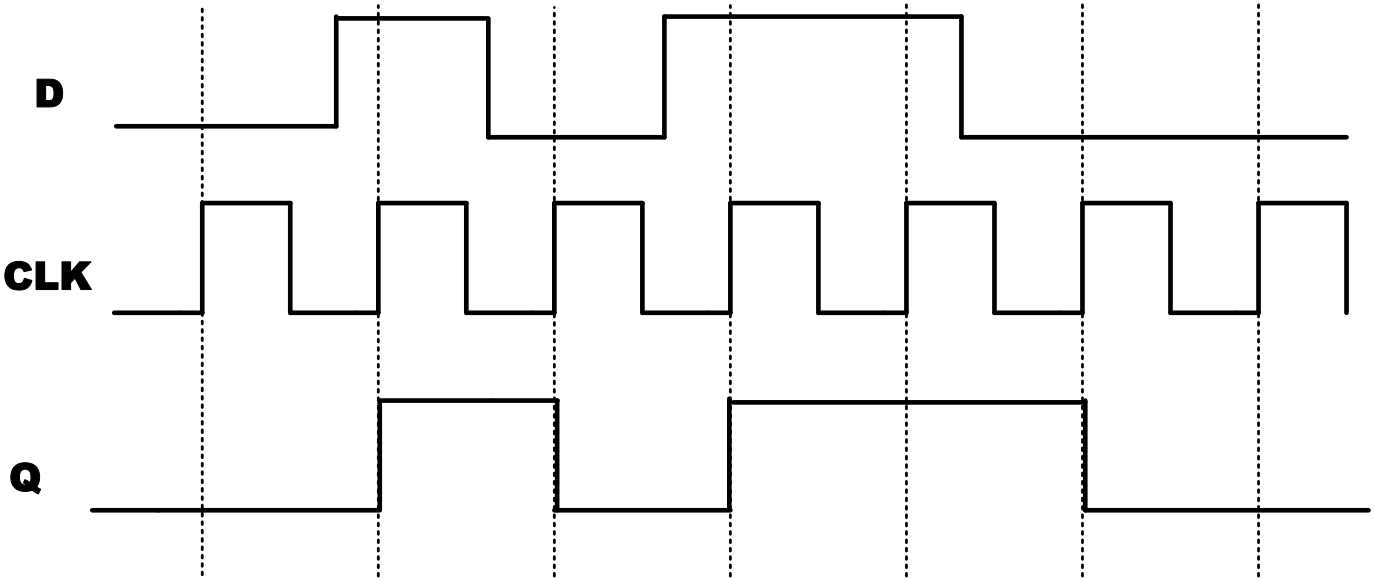


DIN	clk	rstn	DOUT
1	↑	1	1
0	↑	1	0
1	↑	0	0
0	↑	0	0

2.Register, F/F 설계

- **F/F** 타이밍도

- Clock이 **rising** 될 때 들어오는 데이터 **d**를 저장하여 **q**로 출력 시킴



2.Register, F/F 설계

- **F/F 동작 디자인**

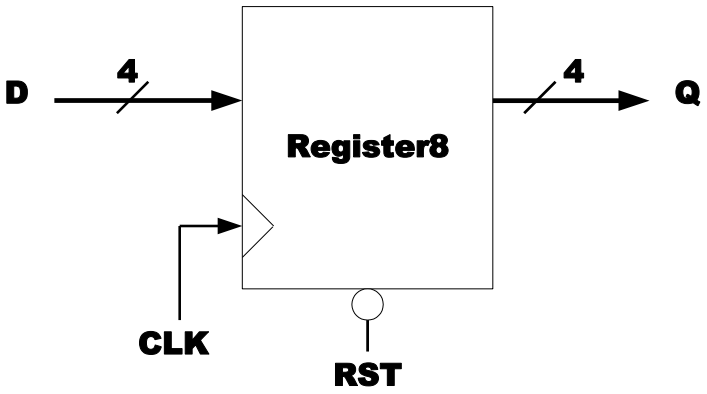
```
`timescale 1 ns / 1ps

module t_dff  ( clk,  rst, d , q );
input    clk , rst , d ;
output   q   ;
reg     q   ;

always @(posedge clk or negedge rst) begin
    if (~rst) begin
       q <= 1'b0;
    end    else begin
        q <= d;
                end
   end
endmodule
```

2.Register, F/F 설계

- **Register 란?**
  - F/F은 1bit를 저장하기 위한 기억소자
  - 2bit 이상 데이터를 저장하기 위해 데이터의 수치적 값으로 나타내는 것이 필요하며 이를 Register라는 F/F집합체에 의해 저장
  - 데이터의 동일한 초기 조건 및 clock 그리고 reset과 같은 활성화 신호에 의해 초기화 되거나 동작하는 기억소자

2.Register, F/F 설계

- **Register 동작**
  - F/F은 1bit를 저장하기 위한 기억소자
- **4bit Register 와 진리표**



CLK	RST	Q(3:0)
-	-	“0000”
↑	0	“0000”
↑	0	“0000”
↑	1	D
↑	0	Q

2.Register, F/F 설계

- **Register 동작 디자인**

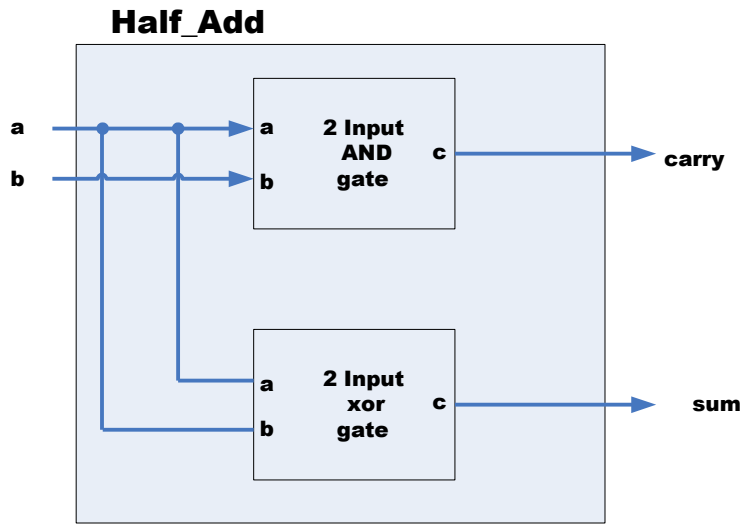
```
`timescale 1 ns / 1ps

module t_reg_4 ( clk, rst , d ,  q );
input   clk , rst ;
input   [3:0]  d  ;
output [3:0]  q  ;
reg      [3:0]   q   ;

always @(posedge clk or negedge rst) begin
    if (~rst) begin
       q <= 4'b0000;
    end
    else begin
        q <= d;
    end
 end
endmodule
```

2.Half/full Adder 설계

- **Half/Full Adder 이해**
  - Adder는 1bit의 가수를 더하여 합(Sum)과 자리 올림(Carry)을 발생 시키는 로직
  - 진리표를 참조하면 자릿수 올림은 가수나 피가수 중 하나만 1인 경우 1이 생성됨을 알 수 있음
  - Full Adder는 1bit 이상의 덧셈을 해야 하는 경우 사용
  - 가수, 피가수 그리고 자리 올림 수를 더하는 동작 수행

2.Half/full Adder 설계

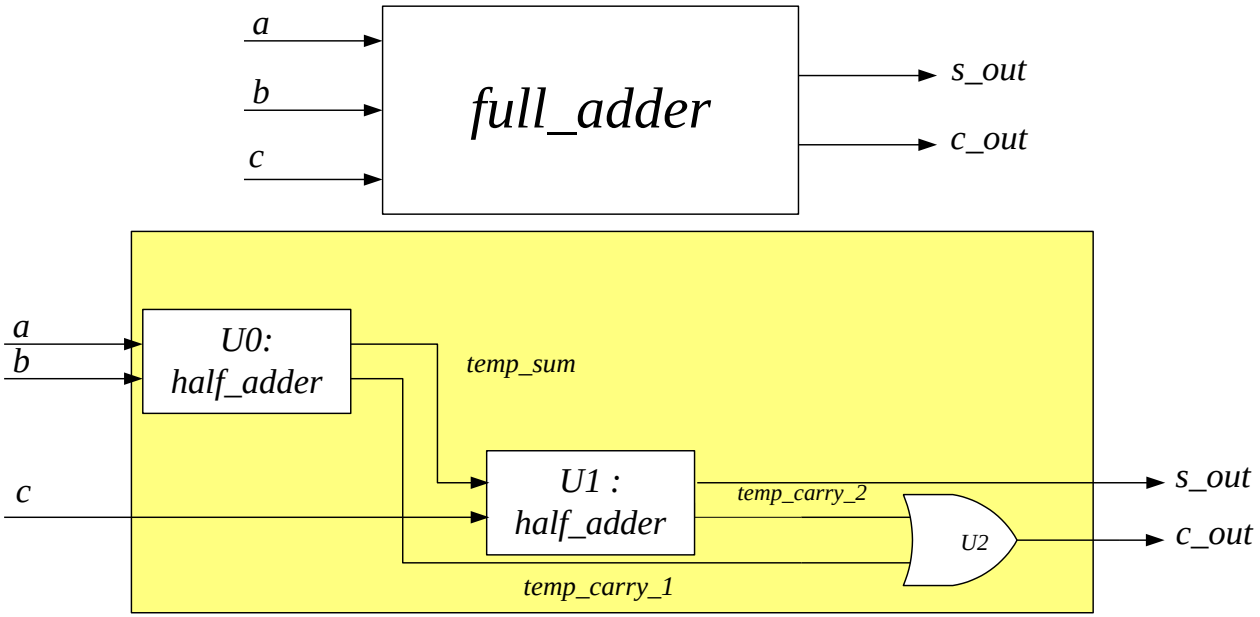
- **Half/Full Adder 이해**
  - Adder는 1bit의 가수를 더하여 합(Sum)과 자리 올림(Carry)을 발생 시키는 로직
- **Half Adder 회로도 및 진리표**



a	b	sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2.Half/full Adder 설계

- **Full Adder 회로도 및 진리표**



2.Half/full Adder 설계

- **Full Adder 회로도 및 진리표**

a	b	C	sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

2.Half/full Adder 설계

- **Half/ Full Adder 설계**

```
`timescale 1 ns / 1ps

module h_adder  (a , b , sum , carry);
input    a , b      ;
output   sum,  carry  ;

assign sum   = (a ^ b);assign carry = (a & b);

endmodule
```


2.Half/full Adder 설계

- **Half/ Full Adder 설계**

```
`timescale 1 ns / 1ps

module full_adder (a , b , c , sum_o , carry_o );
input    a, b, c ;
output   sum_o,  carry_o;
wire tmp_sum , tmp_carry_1, tmp_carry_2 ;
h_adder u0   (.a ( a   ),
              .b ( b   ),
              .sum  ( tmp_sum),
              .carry ( tmp_carry_1));
h_adder u1     (.a ( tmp_sum    ),
                .b     ( c  ),
                .sum   ( sum_o ),
                .carry ( tmp_carry_2));

assign carry_o = (tmp_carry_1 | tmp_carry_2);

endmodule
```

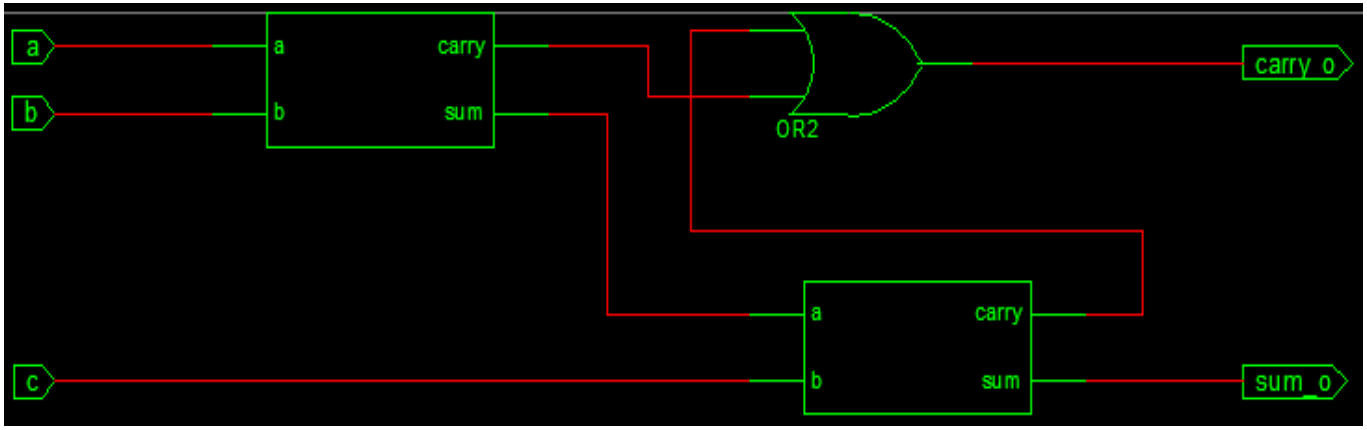
2.Half/full Adder 설계

- **Half/ Full Adder 설계**

```
Begin

u0 : h_adder
                port map ( a      => a,
                           b      => b,
                           sum    => tmp_sum,
                           carry  => tmp_carry_1);
u1 : h_adder
                port map ( a      => tmp_sum,
                           b      => c,
                           sum    => sum_o,
                           carry  => tmp_carry_2);

carry_o <= tmp_carry_1 or tmp_carry_2;

end Behavioral;
```

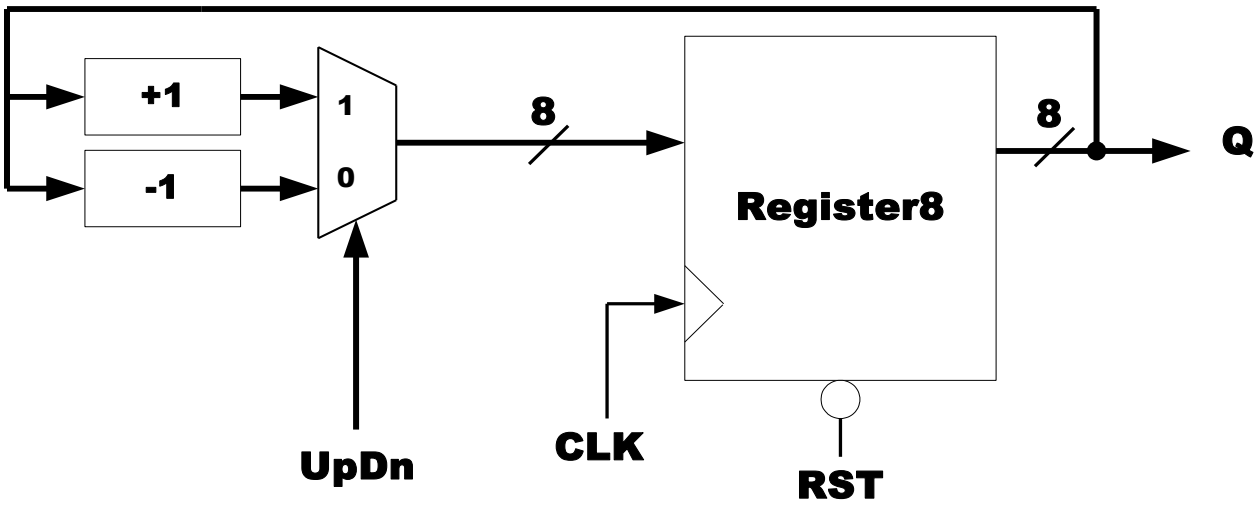
2.Half/full Adder 설계

- **Half/ Full Adder synthesis**



2. 8bit counter(up/down) 설계

- **Counter의 이해(동기식 카운터)**
  - 동기식은 입력되는 clock에 의해 동작하는 Counter
  - Up/down counter는 카운터 두개를 구성하여 외부 신호에 의해 둘 중 하나를 선택



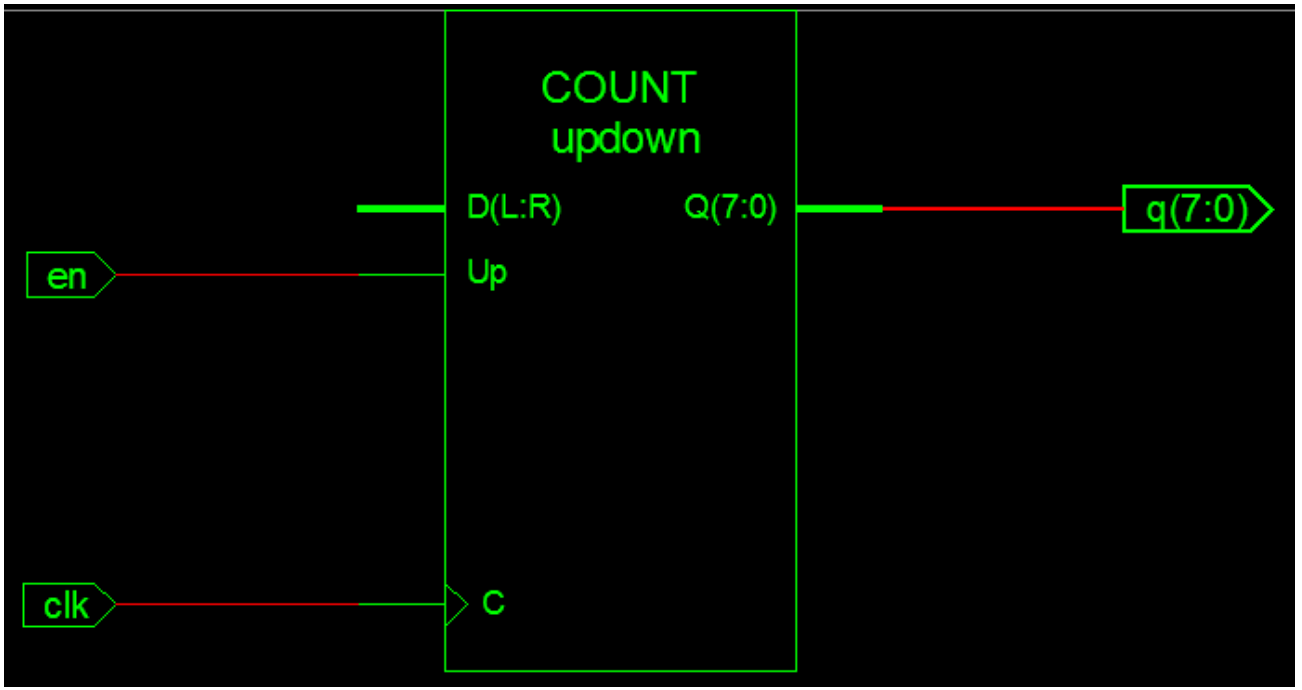
2. 8bit counter(up/down) 설계

- **Counter의 디자인**

```
`timescale 1 ns / 1ps

module cnt_8_ud (clk, rst, en, q);
input     clk, rst, en ;
output [7:0]   q    ;
reg       [7:0] tmp ;

always @(posedge clk or negedge rst) begin
    if (~rst) begin
       tmp <= 8'h00;
    end     else begin
         if (en) begin
                 tmp <= tmp +   1'b1
         end
         else begin
             tmp <= tmp -   1'b1;
         end
      end
   end
assign q = tmp;
endmodule
```

2. 8bit counter(up/down) 설계

- Counter의 synthesis 화면



3. Simulation의 이해

3.Simulation 동작 원리 이해

- **Simulation 동작 원리**
  - 로직을 검증하기 위해 실시
  - 크게 RTL, Gate, Timing Simulation으로 구분
  - 소스 로직과 보드에서 들어오는 입력 사항정리 파일 필요 (TestBench)
  - RTL, Gate 소스 와 TestBench 컴파일 후 결과 파형으로 검증

3.Simulation 동작 원리 이해

- **Simulation 종류**
  - RTL Simulation : RTL 소스 + TestBench
    (Behavioral Simulation)
  - Gate Simulation : Gate 소스 + TestBench
    (Post Translate Simulation)
  - Timing Simulation : Gate 소스 + TestBench + sdf
    (Post Route Simulation)

3.Simulation 동작 원리 이해

- **Library 이해**
  - Synthesis과정 후 RTL 소스가 Gate 소스로 변환
  - Gate 소스는 xilinx Lib로 로직 구성 되어 있음.
  - Gate소스 이후에 나타나는 로직 형태로 표현된 것이 xilinx Lib 형태로 로직 표현
  - Simulator는 이 lib의 동작 원리 및 형태를 알아야 Simulation 가능
  - ISE에서는 ModelSim의 Simulation Lib가 ISE버전 마다 차이가 있어 버전에 맞는 lib 다운로드 후 사용

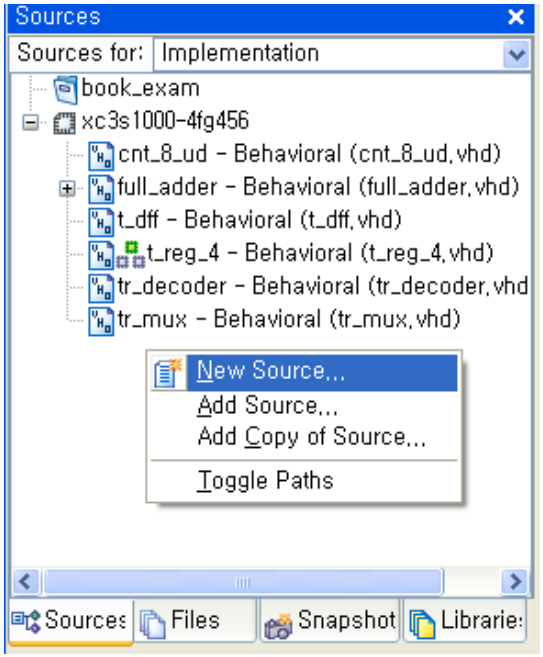
3.Simulation 동작 원리 이해

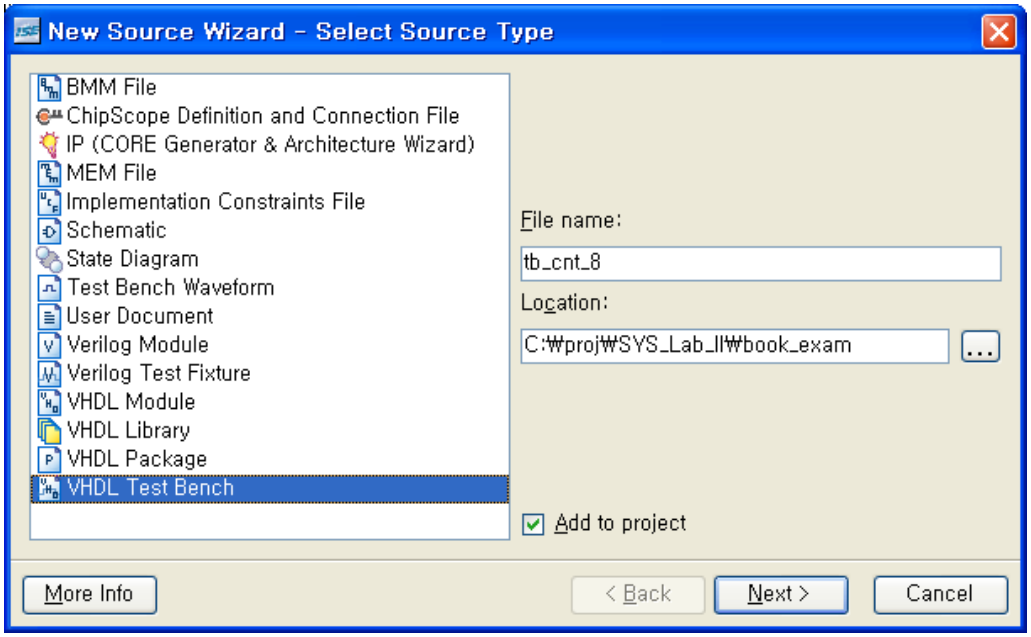
- 테스트벤치 작성법 이해
  - 보드에서 FPGA로 들어가는 데이터의 입력사항을 HDL코드로 정의 한것.

  - ISE에서는 테스트 벤치 이용 Behavioral , Post Translate , Post Map, Post Route Simulation으로 구분

3.Simulation 동작 원리 이해

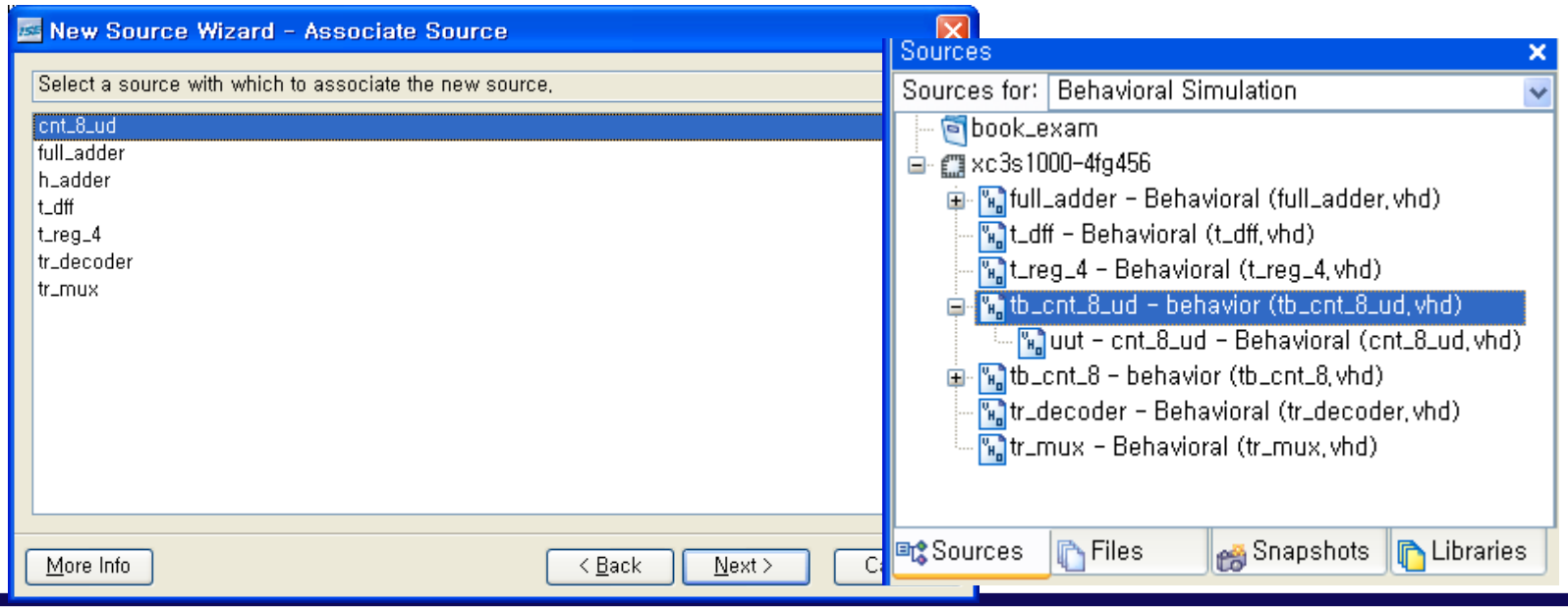
- 테스트벤치 작성법 이해
  - Source창 -> 마우스 오른쪽 버튼 -> New Source..
  - VHDL TestBench -> tb_cnt_8 입력





3.Simulation 동작 원리 이해

- 테스트벤치 작성법 이해
  - cnt_8_ud -> Next
  - Source for : Behavioral Simulation 선택 ->테스트 벤치 및 디자인 연결 확인



3.Simulation 동작 원리 이해

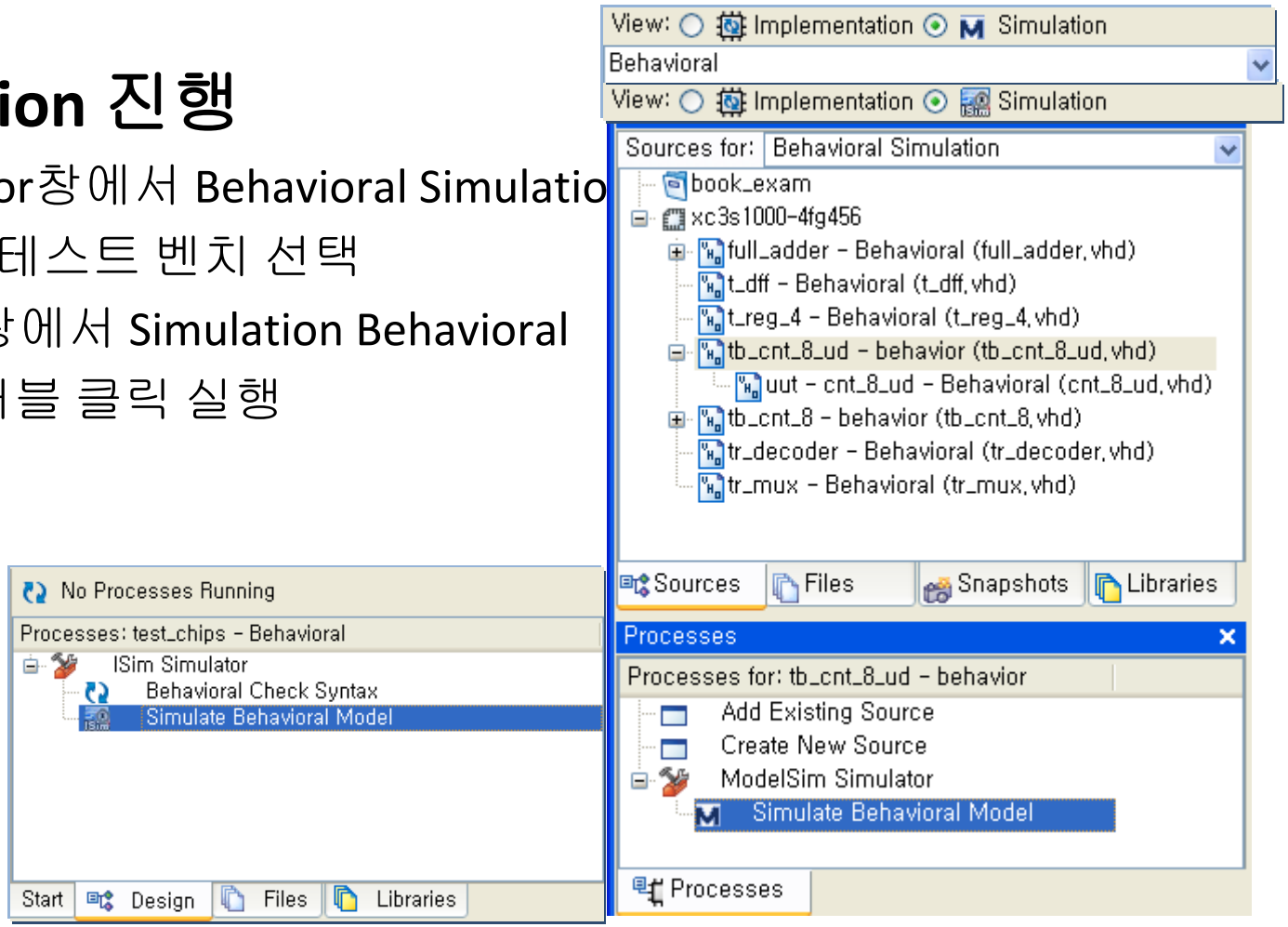
- 테스트벤치 전체 작성

```
`timescale 1 ns / 1ps

module tb_cnt_8_ud;
reg     clk ;
reg     rst ;
reg     en ;
wire  [7:0]  q ;

cnt_8_ud uut( .clk  ( clk  ),
              .rst  ( rst  ),
              .en   ( en   ),
              .q    ( q    ) );

parameter    thclk  = 5  ;  //100Mhz
parameter    tclk   = (thclk * 2);
parameter    trst   = (tclk   * 10 );
integer      count;

initial
begin  clk = 1'b0;
  forever #thclk clk = ~clk;
end
```

```
initial
begin
   rst         = 1'b0  ;
   en          = 1'b0  ;
   #trst rst   = 1'b1  ;
    for (count=0; count < 10; count=count+1) begin
       #(tclk * 256);
       en = (~en);
    end
     $finish;
endendmodule
```

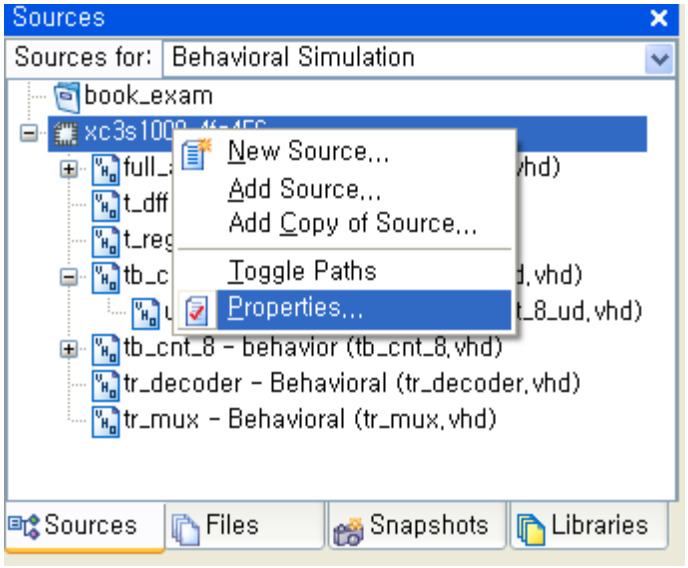
3.Simulation 동작 원리 이해

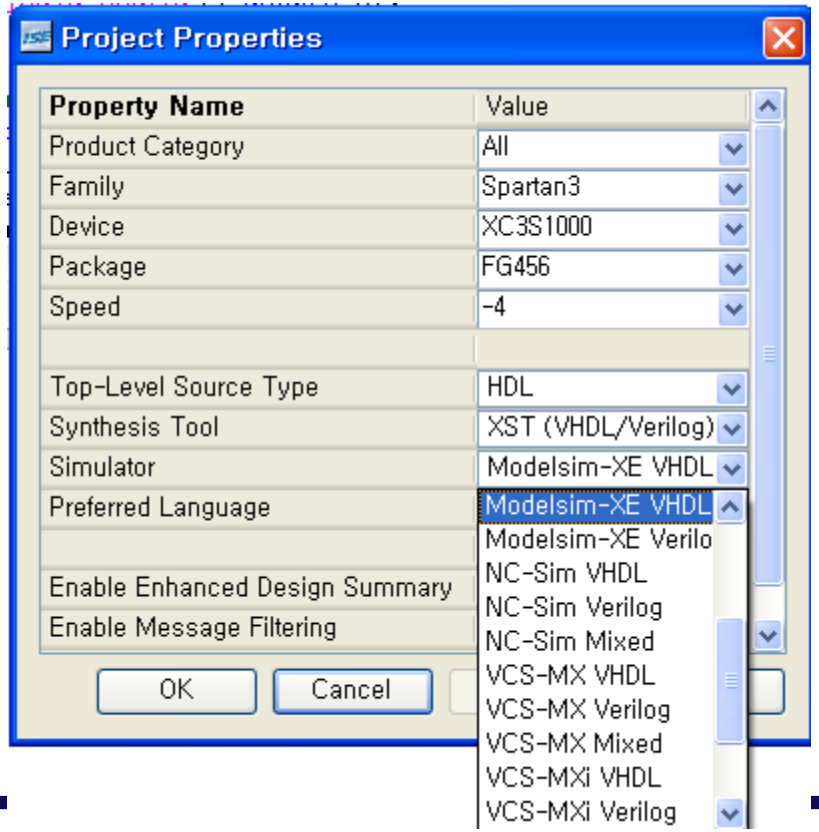
- **Simulation 진행**
  - Source for창에서 Behavioral Simulatio 선택 후 테스트 벤치 선택
  - Process창에서 Simulation Behavioral Model 더블 클릭 실행



3.Simulation 동작 원리 이해

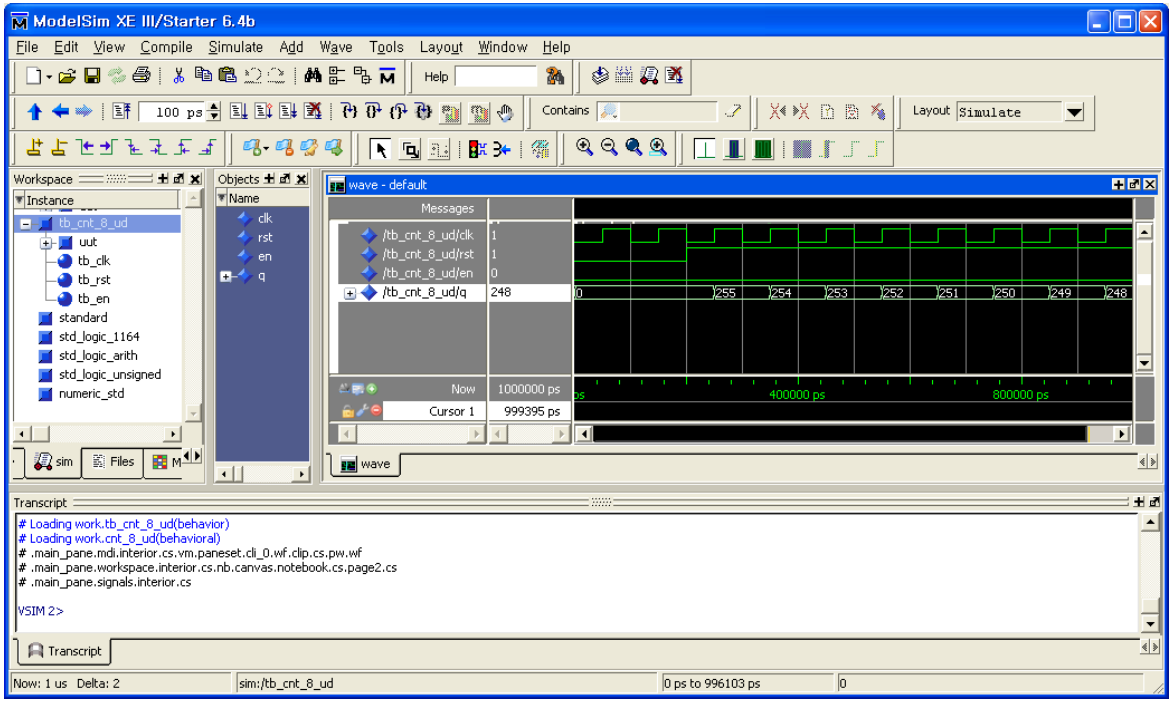
- **Simulation 진행**
  - 이전화면이 나타나지 않으면 아래내용 확인





3.Simulation 동작 원리 이해

- **Simulation 진행 (simulate – run all)**
  - ModelSim 창 Simulation 결과 확인
  - 자동 종료를 누르지 말것



4. LED, Buzzer, Push 및 DIP Swith 동작 실습

4.LED 디자인 설계

- **LED 동작 원리 및 설계**
  - SYS-Lab II의 LED는 Active Low이다. 즉 Low('0') 데이터가 들어가면 LED가 동작한다.
  - SYS-Lab II 의 24개 LED중 8개만 사용.
  - 8개의 LED에 10101010값을 입력하여 동작
  - 첫 번째 LED off, 두 번째 LED on, 세 번째 LED off 이런 식으로 동작 예상.

4.LED 디자인 설계

- **LED** 동작 디자인 및 핀정보

```
`timescale 1 ns / 1ps

module test_led     ( led );
output [7:0]  led      ;

assign led = 8'b10101010;

endmodule
```

```
NET "LED<0>" LOC = "N3";
NET "LED<1>" LOC = "N2";
NET "LED<2>" LOC = "N1";
NET "LED<3>" LOC = "M6";
NET "LED<4>" LOC = "M5";
NET "LED<5>" LOC = "M4";
NET "LED<6>" LOC = "M3";
NET "LED<7>" LOC = "M2";
```


4.Buzzer 설계

- **Buzzer 동작 원리 및 설계**
  - SYS-Lab II의 Buzzer는 Active High이다. 즉 High('1') 데이터가 들어가면 Buzzer가 동작한다.
  - rst가 '0'일 때 clk가 주기적으로 반복하므로 buzzer소리가 출력될 것이며, rst가 '1'일 때 buzzer는 소리가 나지 않는다.

```
`timescale 1 ns / 1ps

module test_buzzer ( clk, rst, q );
input  clk     ;
input  rst     ;
output q       ;
reg  [25:0] tmp      ;
always @(posedge clk or posedge rst) begin
    if (rst) begin
       tmp    <= 26'h0000000;
    end
```

```
else begin
        if (tmp == 26'b11000101101111011001)
                begin
                tmp    <= 26'h0000000;
                end
         else begin
                tmp    <= tmp + 1'b1;
               end
    end
end
```

```
NET "clk"      LOC = "ab12";
NET "rst"      LOC = "u14";
NET "q"        LOC = "ab15";
```

4.주파수 입력에 따른 Buzzer 출력 설계

- 주파수에 따른 Buzzer 동작 원리

  - Buzzer는 입력되는 파형의 주파수에 따라 음계가 변함.
  - 각 음계의 맞는 주파수를 정의함.
  - 예제 디자인은 참고 음계표를 이용 도레미파솔라시도(C, D,E,F,G,A,B) 음 출력
  - 음의 값을 음계표를 기준으로 보자
  - 입력 주파수를 음계 값에 맞게 분주하여 출력 동작

음계값(주파수 Hz)			
C(130.8128)	D(146.8324)	E(164.8138)	F(174.6141)
G(195.9977)	A(220.0000)	B(246.9417)	

4.주파수 입력에 따른 Buzzer 출력 설계

- 주파수에 따른 Buzzer 음계 표

옥타브 음계	1	2	3	4	5	6	7	8
C(도)	32.7032	65.4064	130.8128	261.6256	523.2511	1046.502	2093.005	4186.009
C#	34.6478	69.2957	138.5913	277.1826	554.3653	1108.731	2217.461	4434.922
D(레)	36.7081	73.4162	146.8324	293.6648	587.3295	1174.659	2349.318	4698.636
D#	38.8909	77.7817	155.5635	311.1270	622.2540	1244.508	2489.016	4978.032
E(미)	41.2034	82.4069	164.8138	329.6276	659.2551	1318.510	2637.020	5274.041
F(파)	43.6535	87.3071	174.6141	349.2282	698.4565	1396.913	2793.826	5587.652
F#	46.2493	92.4986	184.9972	369.9944	739.9888	1479.978	2959.955	5919.911
G(솔)	48.9994	97.9989	195.9977	391.9954	783.9909	1567.982	3135.963	6271.927
G#	51.9130	103.8262	207.6523	415.3047	830.6094	1661.219	3322.438	6644.875
A(라)	55.000	110.000	220.000	440.0000	880.000	1760.000	3520.000	7040.000
A#	58.2705	116.5409	233.0819	466.1638	932.3275	1864.655	3729.310	7458.620
B(시)	61.7354	123.4708	246.9417	493.8833	987.9666	1975.533	3951.066	7902.133

4.주파수 입력에 따른 Buzzer 출력 설계

- 주파수에 따른 **Buzzer** 출력 디자인**-1**

```
`timescale 1 ns / 1ps

module test_buzzer_scale   ( clk, rst, dip_sw, buzzer );

input         clk     ;
input         rst     ;
input [2:0] dip_sw  ;
output       buzzer  ;

reg  [19:0] tmp_0      ;
reg  [19:0] tmp_1      ;
reg  [19:0] tmp_2      ;
reg  [19:0] tmp_3      ;
reg  [19:0] tmp_4      ;
reg  [19:0] tmp_5      ;
reg  [19:0] tmp_6      ;
reg          buzzer     ;
```

4.주파수 입력에 따른 Buzzer 출력 설계

- 주파수에 따른 **Buzzer** 출력 디자인**-2**

```
always @(posedge clk or negedge rst) begin
    if (~rst) begin
       tmp_0    <= 20'h00000;
    end
    else begin
        if (tmp_0 == 20'b10111010101000100011) begin
                tmp_0    <= 20'h00000;
        end
        else begin
                tmp_0    <= tmp_0 + 1'b1;
        end
    end
end
```

```
always @(posedge clk or negedge rst) begin
    if (~rst) begin
       tmp_1    <= 20'h00000;
    end
    else begin
        if (tmp_1 == 20'b10100110010001011000) begin
                tmp_1    <= 20'h00000;
        end
        else begin
                tmp_1    <= tmp_1 + 1'b1;
        end
    end
end
```

4.주파수 입력에 따른 Buzzer 출력 설계

- 주파수에 따른 **Buzzer** 출력 디자인**-3**

```
always @(posedge clk or negedge rst) begin
    if (~rst) begin
       tmp_2    <= 20'h00000;
    end
    else begin
        if (tmp_2 == 20'b10010100001000011001) begin
                tmp_2    <= 20'h00000;
        end
        else begin
                tmp_2    <= tmp_2 + 1'b1;
        end
    end
end
```

```
always @(posedge clk or negedge rst) begin
    if (~rst) begin
       tmp_3    <= 20'h00000;
    end
    else begin
        if (tmp_3 == 20'b10001011110100010011) begin
                tmp_3    <= 20'h00000;
        end
        else begin
                tmp_3    <= tmp_3 + 1'b1;
        end
    end
end
```

4.주파수 입력에 따른 Buzzer 출력 설계

- 주파수에 따른 **Buzzer** 출력 디자인**-4**

```
always @(posedge clk or negedge rst) begin
    if (~rst) begin
       tmp_4    <= 20'h00000;
    end
    else begin
        if (tmp_4 == 20'b11111001001000000010) begin
                tmp_4    <= 20'h00000;
        end
        else begin
                tmp_4    <= tmp_4 + 1'b1;
        end
    end
end
```

```
always @(posedge clk or negedge rst) begin
    if (~rst) begin
       tmp_5    <= 20'h00000;
    end
    else begin
        if (tmp_5 == 20'b11011101111110010001) begin
                tmp_5    <= 20'h00000;
        end
        else begin
                tmp_5    <= tmp_5 + 1'b1;
        end
    end
end
```

4.주파수 입력에 따른 Buzzer 출력 설계

- 주파수에 따른 **Buzzer** 출력 디자인**-5**

```
always @(posedge clk or negedge rst) begin
    if (~rst) begin
       tmp_6    <= 20'h00000;
    end
    else begin
         if (tmp_6 == 20'b11000101101111011001) begin
                tmp_6    <= 20'h00000;
         end
         else begin
                tmp_6    <= tmp_6 + 1'b1;
         end
    end
end
```

```
always @(dip_sw or
         tmp_0  or
         tmp_1  or
         tmp_2  or
         tmp_3  or
         tmp_4  or
         tmp_5  or
         tmp_6   ) begin
    case (dip_sw)
        3'b000  : buzzer = tmp_0[19];
        3'b001  : buzzer = tmp_1[19];
        3'b010  : buzzer = tmp_2[19];
        3'b011  : buzzer = tmp_3[19];
        3'b100  : buzzer = tmp_4[19];
        3'b101  : buzzer = tmp_5[19];
        3'b110  : buzzer = tmp_6[19];
        3'b111  : buzzer = tmp_0[0];
        default : buzzer = tmp_0[0];
    endcase
end

endmodule
```

4.Counter 이용 LED 표시

- 동작 원리 및 설계
  - Counter는 clock의 개수를 세는 로직으로 clock의 count한 값을 표시하기 위해 모든 bit가 ‘0’,’1’을 반복함
  - Count값을 LED에 표시하기 위해서는 잔상이 없는 주파수로 분주 LED에 전달해야 함.
  - 형광등의 경우 60Hz로 동작하나 그 깜빡이는 현상을 못 느낌.
  - 설계 디자인의 가장 느린 출력은 2Hz로 하여 그 이후 분주 값을 LED에 연결하자.(2Hz., 4Hz, 8Hz,… 256Hz식으로 파형 발생.)
  - 느리게 분주된 값을 전달하기 위해 분주 된 값 중에 상위 8bit만 출력

4.Counter 이용 LED 표시

- 동작 디자인 설계

```
`timescale 1 ns / 1ps

module test_led_cnt  (clk, rst, led );
input            clk     ;
input            rst     ;
output  [7:0]  led     ;
reg      [25:0] tmp      ;
always @(posedge clk or negedge rst) begin
    if (~rst) begin
       tmp    <= 26'h0000000;
    end
    else begin
        if (tmp == 26'b10111110101111000010000000) begin
                tmp    <= 26'h0000000;
        end
        else begin
                tmp    <= tmp + 1'b1;
        end
    end
end
assign led = tmp[25:18];
endmodule
```


4.Counter 이용 LED 표시

- 동작 디자인 설계 및 핀 정보

```
NET "clk"        LOC = "ab12";
NET "rst"        LOC = "u14";
NET "led<0>"  LOC = "N3";
NET "led<1>"  LOC = "N2";
NET "led<2>"  LOC = "N1";
NET "led<3>"  LOC = "M6";
NET "led<4>"  LOC = "M5";
NET "led<5>"  LOC = "M4";
NET "led<6>"  LOC = "M3";
NET "led<7>"  LOC = "M2";
```

4. 2Hz LED 표시

- **2Hz로 LED 동작 및 이동 원리 설계**
  - 이전 로직 구성에서 2Hz ~ 256Hz까지 LED가 동작하도록 설계 디자인 참조
  - 2Hz로 이동하기 위해 Decoder를 이용(2주차 교육자료 참조) 디코딩 하여 2Hz로 동작 하는 3bit 시그널 생성 이동(2Hz) 하는 디자인 적용

4. 2Hz LED 표시

- 동작 디자인 설계-1

```
`timescale 1 ns / 1ps
module test_led_cnt  ( clk, rst, led );
input          clk     ;
input          rst     ;
output [7:0]  led     ;
reg     [2:0]   case_tmp ;
reg     [25:0] tmp      ;
reg              en       ;
reg     [7:0]   led      ;
always @(posedge clk or negedge rst) begin
    if (~rst) begin
       en     <= 1'b0;
       tmp    <= 26'h0000000;
    end
    else begin
         if (tmp == 26'b10111110101111000010000000) begin
                en     <= 1'b1;
                tmp    <= 26'h0000000;
         end
         else begin
                en     <= 1'b0;
                tmp    <= tmp + 1'b1;
         end
    end
end
```

4. 2Hz LED 표시

- 동작 디자인 설계-2

```
always @(posedge clk or negedge rst) begin
    if (~rst) begin
       case_tmp    <= 3'b000;
    end
    else begin
         if (en) begin
            if (case_tmp == 3'b111) begin
               case_tmp    <= 3'b000;
            end
            else begin
               case_tmp    <= case_tmp + 1'b1;
                 end
         end
         else begin
                 case_tmp    <= case_tmp;
         end
    end
end
```

```
always @(case_tmp) begin
    case (case_tmp)
       3'b000  : led = 8'b00000001;
       3'b001  : led = 8'b00000010;
       3'b010  : led = 8'b00000100;
       3'b011  : led = 8'b00001000;
       3'b100  : led = 8'b00010000;
       3'b101  : led = 8'b00100000;
       3'b110  : led = 8'b01000000;
       3'b111  : led = 8'b10000000;
       default : led = 8'b00000000;
      endcase
end

endmodule
```

4. 2Hz LED 표시

- 동작 디자인 핀 정보

```
NET "clk"        LOC = "ab12";
NET "rst"        LOC = "u14";
NET "led<0>"  LOC = "N3";
NET "led<1>"  LOC = "N2";
NET "led<2>"  LOC = "N1";
NET "led<3>"  LOC = "M6";
NET "led<4>"  LOC = "M5";
NET "led<5>"  LOC = "M4";
NET "led<6>"  LOC = "M3";
NET "led<7>"  LOC = "M2";
```

4.디자인 Simulation (p.57)

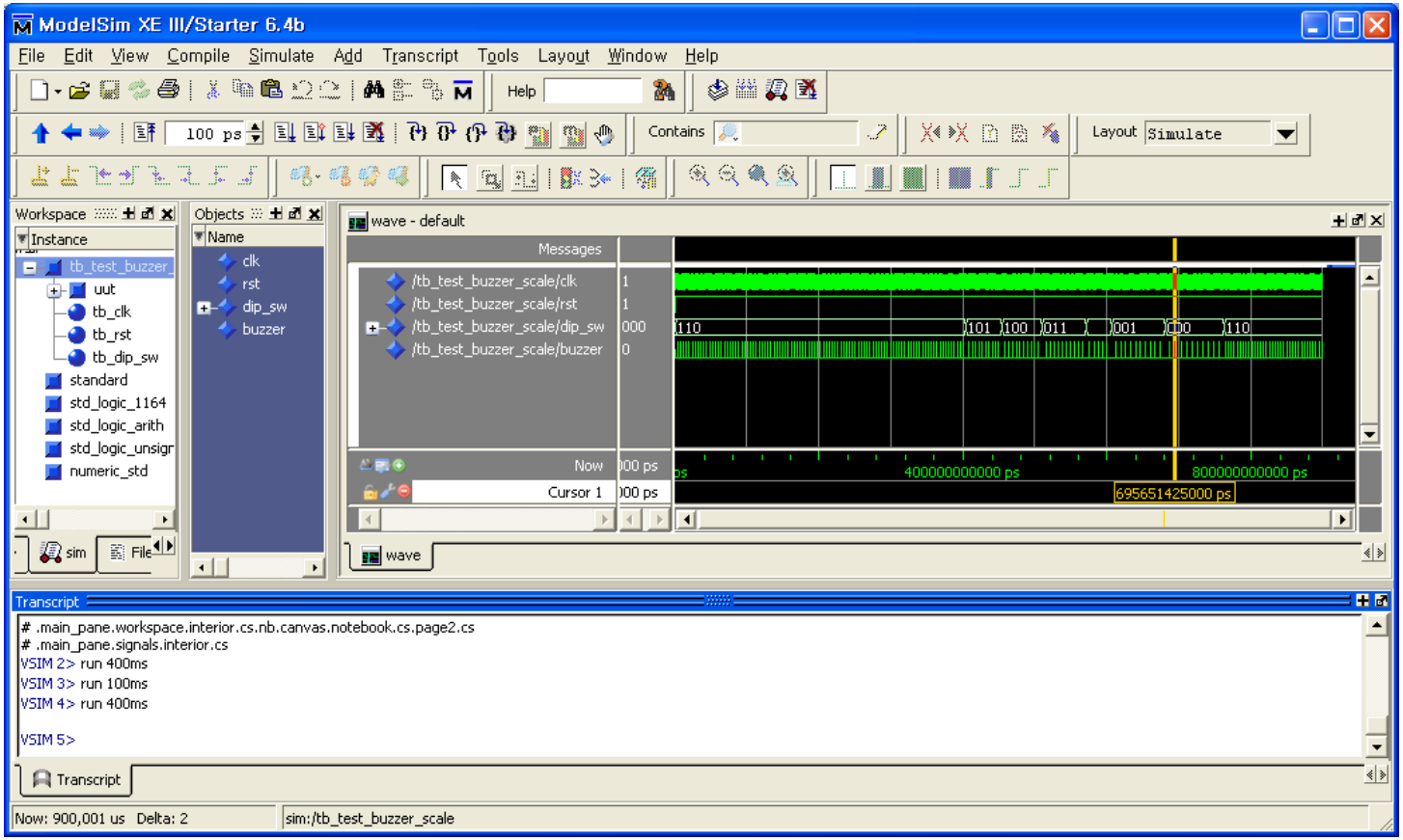
- **DIP Switch 이용 디자인 Simulation(TestBench)**

```
`timescale 1 ns / 1ps

module tb_test_buzzer_scale;
reg           clk   ;
reg           rst   ;
reg  [2:0]   dip_sw;
wire          buzzer;

test_buzzer_scale uut
     (.clk       ( clk    ),
      .rst       ( rst    ),
      .dip_sw    ( dip_sw ),
      .buzzer    ( buzzer ) );
parameter    thclk    =  5              ;
parameter    tclk     = (thclk *  2     );
parameter    trst     = (tclk  * 10     );
integer   count ;
integer   count1;

initial
begin
  clk = 1'b0;
  forever #thclk clk = ~clk;
end
```

```
begin
     rst           = 1'b0        ;
    #trst rst      = 1'b1        ;
   for (count1=0;count1 < 8;count1=count1+1) begin
        dip_sw = count1 ;
        for (count=0;count < 4096;count=count+1)
 begin
            #(tclk*256);
        end
    end
    $finish;
end

endmodule
```

4.디자인 Simulation

- **DIP Switch 이용 디자인 Simulation(wave 파형)**



4.디자인 Simulation (p.66)

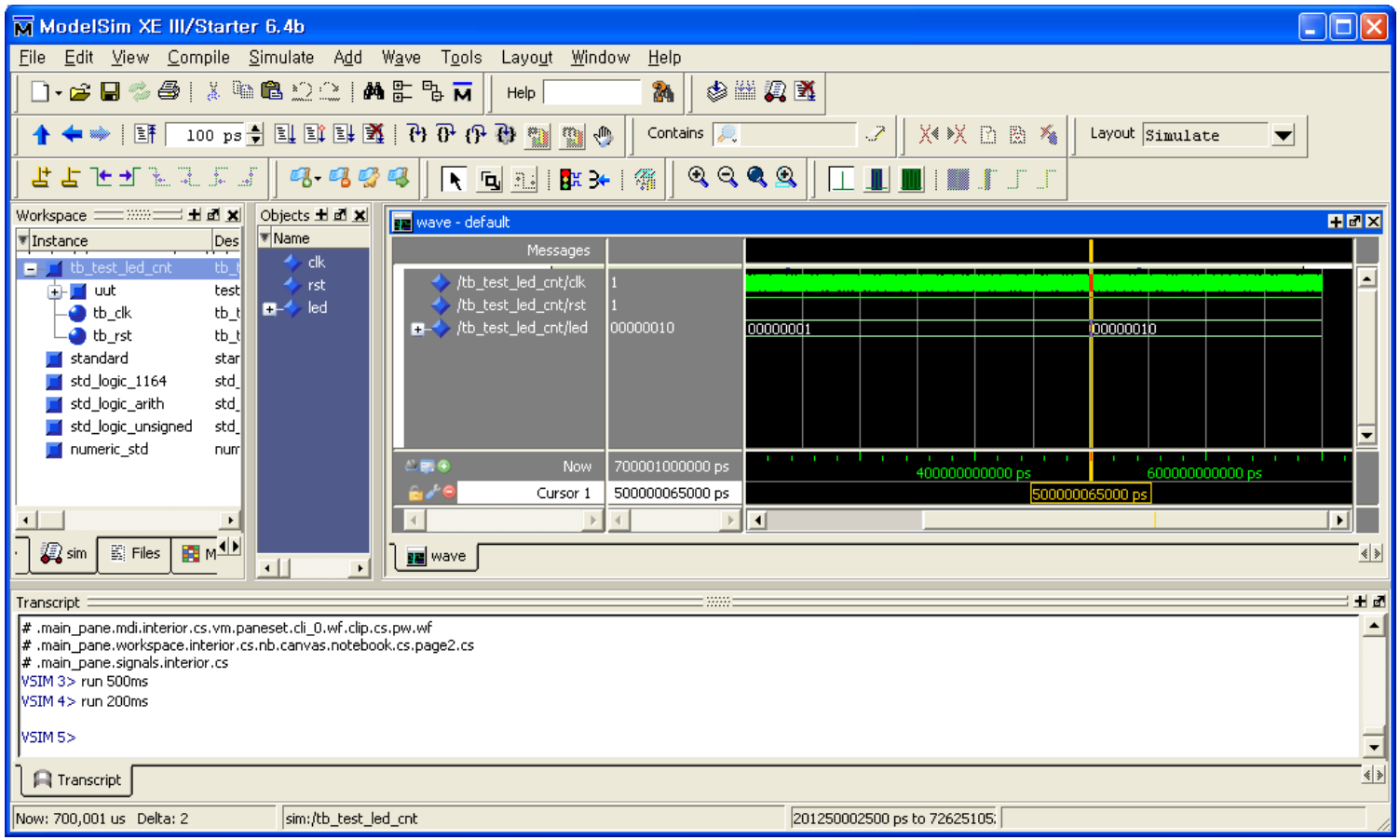
- **2Hz동작 LED 디자인 Simulation(TestBench)**

```
`timescale 1 ns / 1ps

module tb_test_led_cnt;
reg          clk   ;
reg          rst   ;
wire [7:0]   led   ;

test_led_cnt uut
      (.clk        ( clk     ),
       .rst        ( rst     ),
       .led        ( led     )   );
parameter     thclk    =  5              ;
parameter     tclk     = (thclk *  2     );
parameter     trst     = (tclk  * 10     );
integer    count ;
integer    count1 ;

initial
begin
  clk = 1'b0;
  forever #thclk clk = ~clk;
end
```

```
initial
begin
    rst            = 1'b0         ;
    #trst rst      = 1'b1         ;
    for (count1=0;count1 < 8;count1=count1+1) begin
        for (count=0;count < 5001;count=count+1) begin
          #(tclk*10000);
        end
    end
    $finish;
end

endmodule
```

4.디자인 Simulation

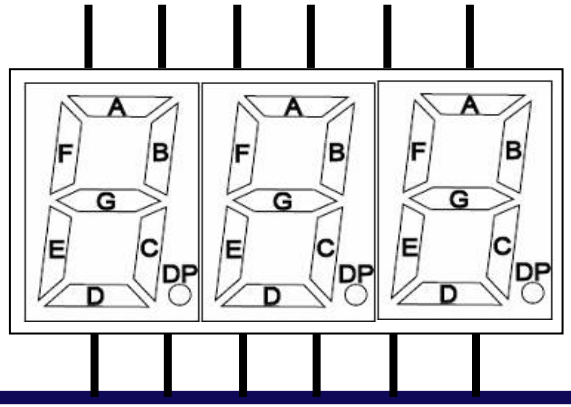
- **2Hz동작 LED 디자인 Simulation(Wave 파형)**



5. 7-segment 기본 동작 실습

5. 7-Segment 동작 이해

- **7-segment 기본 동작 이해**
  - SYS-Lab II 적용된 7-segment는 6개의 7-segment가 붙어 있는 형태
  - 각 segment에 I/O 핀을 연결하면 pin 낭비가 많아서 각 segment 별로 컨트롤하는 enable 시그널인 DIGIT를 연결하여 사용.
  - DIGIT가 enable 되어야 각 segment의 LED가 켜지는 형태
  - 각 segment는 아래의 그림과 같이 A ~ G, DP까지의 LED로 구성
  - 각 segment에 '1' 이 인가되면 LED가 동작하여 숫자 등 표시



5. 7-Segment 동작 구성

- **7-segment 동작 디자인**
  - 모든 DIGIT가 동작하도록 모든 DIGIT에 '1'값 인가
  - Seg값에는 원하는 값 출력 위한 값 설정

```
`timescale 1 ns / 1ps

module seg_test ( push, dip, digit, seg );
input  [3:0] push  ;
input  [7:0] dip   ;
output [5:0] digit ;
output [7:0] seg   ;
reg    [7:0] seg   ;
```

```
always @(push or dip) begin
    case ({push,dip})
        12'b100000000000 : seg = 8'b01111001;
        12'b010000000000 : seg = 8'b00100100;
        12'b001000000000 : seg = 8'b00110000;
        12'b000100000000 : seg = 8'b00011001;
        12'b000010000000 : seg = 8'b00010010;
        12'b000001000000 : seg = 8'b00000010;
        12'b000000100000 : seg = 8'b01111000;
        12'b000000010000 : seg = 8'b00000000;
        12'b000000001000 : seg = 8'b00010000;
        12'b000000000100 : seg = 8'b00001000;
        12'b000000000010 : seg = 8'b00000011;
        12'b000000000001 : seg = 8'b01000110;
        default          : seg = 8'b01000000;
    endcase
end
assign digit = 6'b111111;
endmodule
```

5. 7-Segment 동작 구성

- **7-segment 동작 디자인 핀 정보**

```
NET seg<0>          LOC = A7  ;
NET seg<1>          LOC = E8  ;
NET seg<2>          LOC = D8  ;
NET seg<3>          LOC = B8  ;
NET seg<4>          LOC = A8  ;
NET seg<5>          LOC = F9  ;
NET seg<6>          LOC = E9  ;
NET seg<7>          LOC = B9  ;
NET digit<0>        LOC = D6  ;
NET digit<1>        LOC = C6  ;
NET digit<2>        LOC = B6  ;
NET digit<3>        LOC = E7  ;
NET digit<4>        LOC = D7  ;
NET digit<5>        LOC = B7  ;
```

5.동작 디자인 Simulation

- 테스트 벤치 작성

```
`timescale 1 ns / 1ps

module tb_seg_test;
reg  [3:0]  push  ;
reg  [7:0]  dip   ;
wire [7:0]  seg   ;
wire [5:0]  digit ;

seg_test uut
     (.push      ( push  ),
      .dip       ( dip   ),
      .digit     ( digit ),
      .seg       ( seg   ) );

parameter    thclk =  5         ;
parameter    tclk  = (thclk *  2    );
parameter    trst  = (tclk  * 10    );
integer   count ;
integer   count1 ;
```

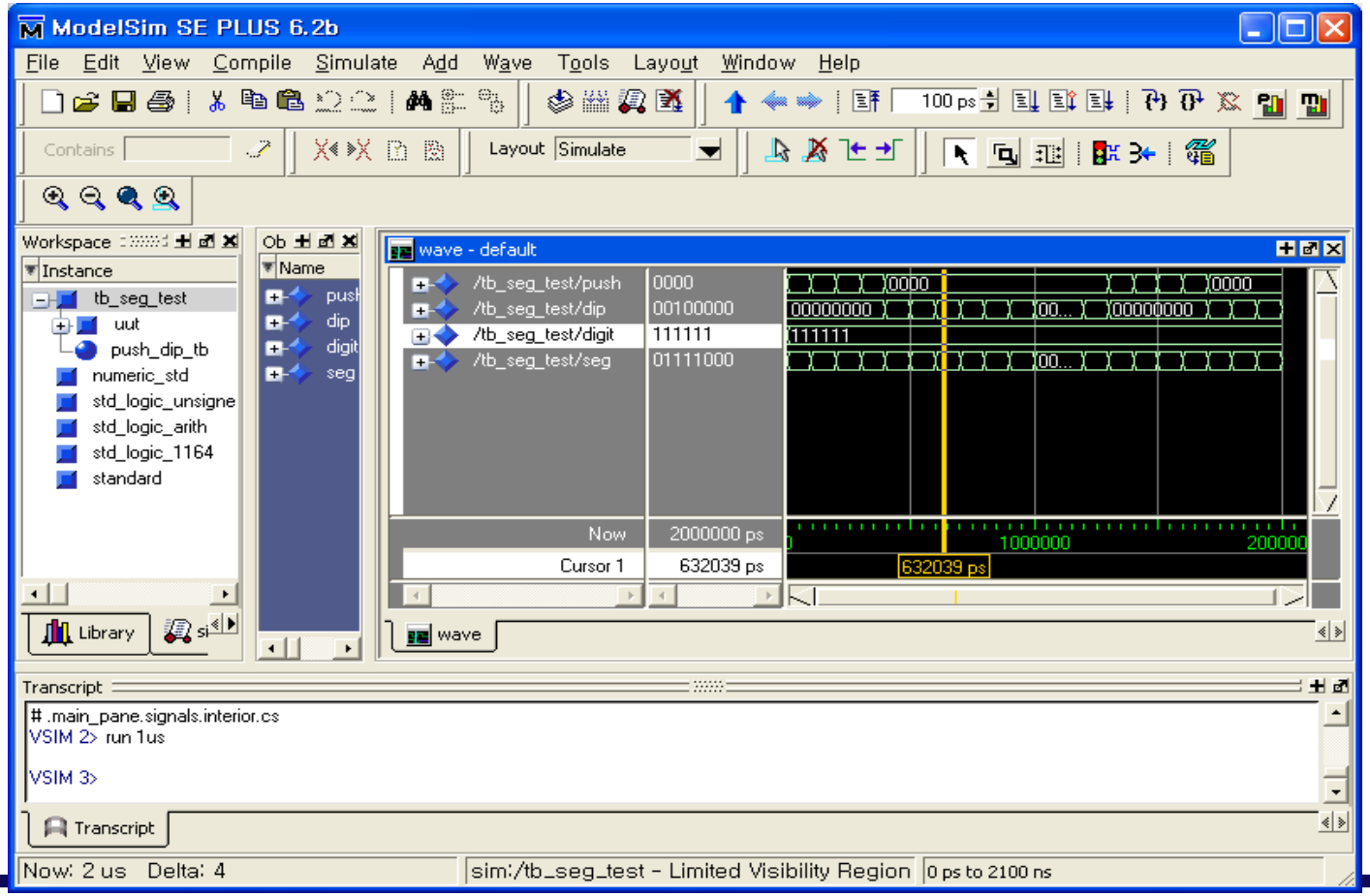
5.동작 디자인 Simulation

- 테스트 벤치 작성

```
initial
begin
    for (count1=0; count1 <10 ; count1 = count1+1) begin
        case (count1)
          0  : begin
               dip =8'b00000000;
               for (count=0;count < 4;count=count+1) begin
              case (count)
                  0   : push =4'b1000;
                  1   : push =4'b0100;
                  2   : push =4'b0010;
                  3   : push =4'b0001;
                 default : push =4'b0000;
                endcase
               #(tclk*100);
               end
             end
          1 : begin dip =8'b10000000;push =4'b0000;end
          2 : begin dip =8'b01000000;push =4'b0000;end
          3 : begin dip =8'b00100000;push =4'b0000;end
          4 : begin dip =8'b00010000;push =4'b0000;end
          5 : begin dip =8'b00001000;push =4'b0000;end
          6  : begin dip =8'b00000100;push =4'b0000;end
          7  : begin dip =8'b00000010;push =4'b0000;end
          8  : begin dip =8'b00000001;push =4'b0000;end
          default : begin dip =8'b00000000;push =4'b0000;
        end
          endcase
          push =4'b0000;
          #(tclk*100);
    end
    $finish;
end

endmodule
```

5.동작 디자인 Simulation

- **Simulation**



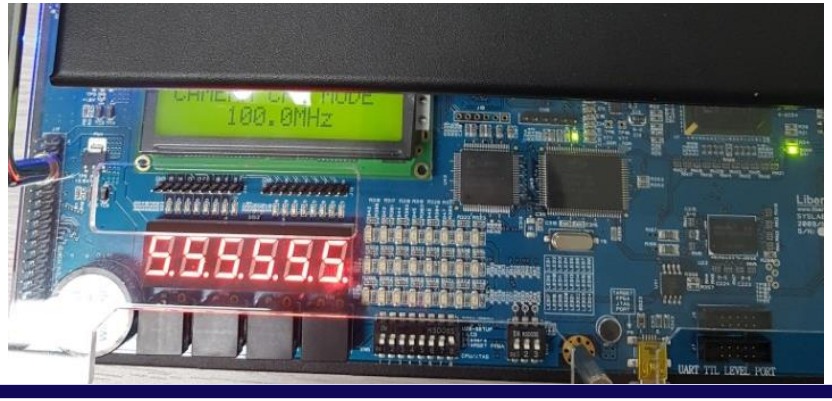
실습

- **seg_test.v 수정**
  - Seg_test 는 I/O 들의 polarity 때문에 제대로 동작하지 않는다.
  - 이를 수정하여 push switch와 dip switch가 눌러졌을 때 제대로 숫자를 나타내도록 수정하여라.
  - 동영상처럼 push button의 왼쪽부터 1, 2, 3, 4, 그리고 dip switch의 왼쪽부터 5, 6, …..이 나와야 한다.
  - 수정할 파일: Ch_5Segment -> segtest directory -> seg_test.v
    - 1. Pin Map에서 push button의 극성을 확인하여 버튼을 눌렀을 때 1이 되도록 Verilog code를 수정하여라. (현재: push button을 누르면 0, 아니면 1)
    - 2. Pin Map에서 dip switch의 극성을 확인하여 switch를 올렸을 때 1이 되도록 Verilog code를 수정하여라.
    - 3. 현재 7-segment 값이 제대로 나오지 않는 이유를 pin-map을 보고 극성을 체크하여 수정하여라.

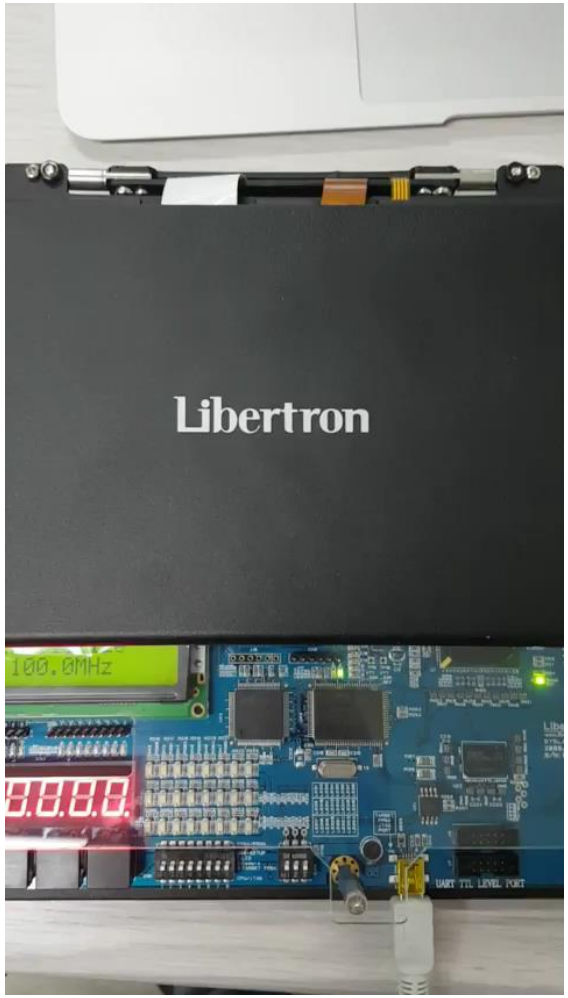
실습

- **seg_test.v 수정**
  - 현재 version

  - 수정 version



데모 영상



보고서 제출

- **보고서**
  - 학과, 학번, 이름
  - 코드를 캡쳐해서 보고서에 첨부
    - seg_test.v
      - 수정된 부분
  - 7-segment 동작 사진 촬영하여 첨부
    - 이상 있을 경우 보고서에 명시

제출 방법

- **제출 방법**
  - 워드나 한글로 작성하여 블랙보드에 제출
  - 문서 제목에 학번과 이름을 적을 것

- **마감일**
  - 6/8일 까지