SoC를 이해하기 위해 알아야할 기본 구성 블럭

Agenda

- SoC 기본 조합논리 구성 블럭 이해하기
  - 멀티플렉서 디멀티플렉서
  - 인코더와 디코더
  - 비교기
  - 3상태 버스
- 기본 블럭 시뮬레이션 해보기 - 실습
- Q&A

SoC 기본 조합논리

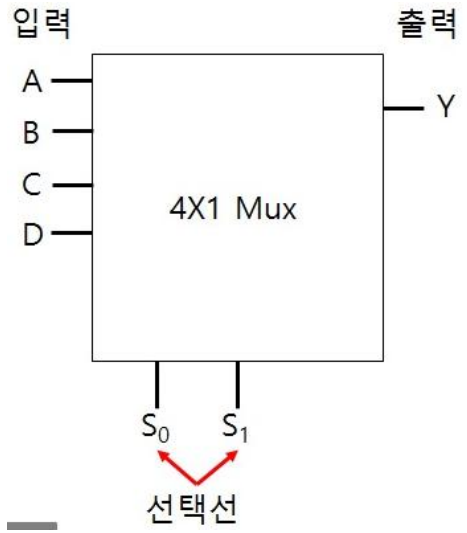
- 조합회로 (combinational logic)
  - 논리 게이트들의 집합으로 구성되며, 회로 구성과 현재의 입력에 의해 회로의 출력 값이 결정되는 회로
  - 저장소자 (래치, 플립플롭)이 포함되지 않은 회로

[표 10–1] 조합회로의 형태와 모델링에 사용되는 Verilog HDL 구문

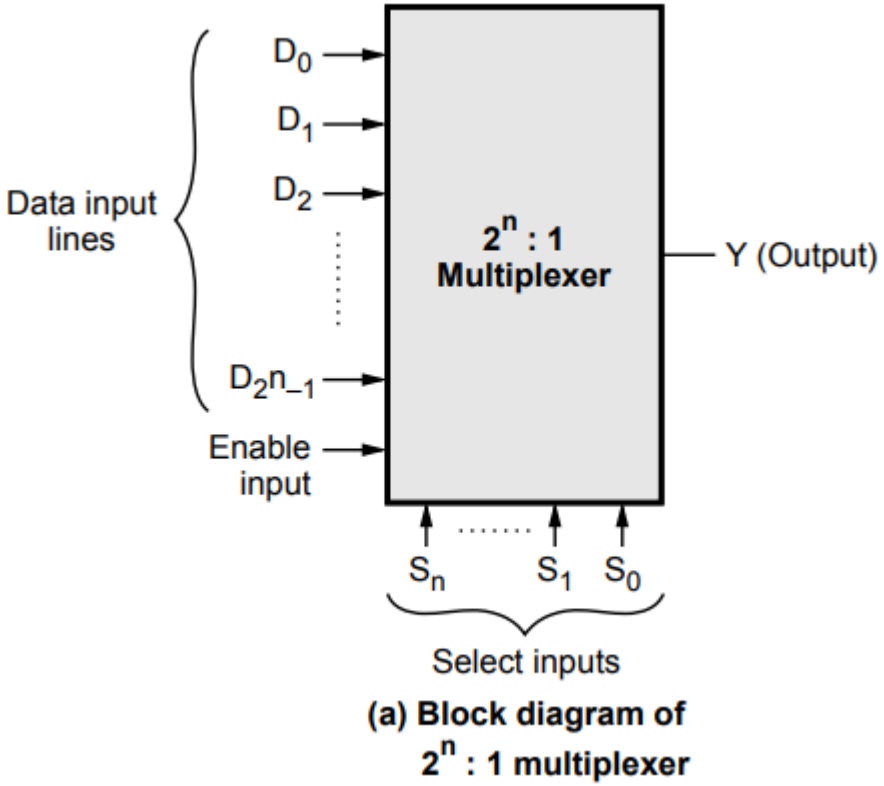
조합회로의 형태	조합회로 설계에 사용되는 Verilog HDL 구문
• 기본 논리 게이트 • 멀티플렉서 • 인코더, 디코더 • 랜덤 로직 • 가산기/감산기 • 비교기 • ALU • lookup table	• 게이트 프리미티브 • 연속 할당문(assign 문) • 행위수준 모델링 (if 문, case 문, for 문, repeat 문, while 문) • 함수 및 태스크 (시간 또는 이벤트 제어를 갖지 못함) • 모듈 인스턴스

SoC 기본 조합논리 구성 블럭 – Multiplexer

- 멀티플렉서 (Multiplexer, MUX)
  - 여러 개의 입력 중 하나를 선택하여 단일 출력으로 전달하는 장치
  - 데이터 선택기(Data Selector)
  - 데이터 입력: 2^n개의 데이터 입력 라인 (D0, D1, D2, ...).
  - 선택 라인: n개의 선택 라인 (S0, S1, ..., Sn-1).
  - 출력: 하나의 출력 라인 (Y).

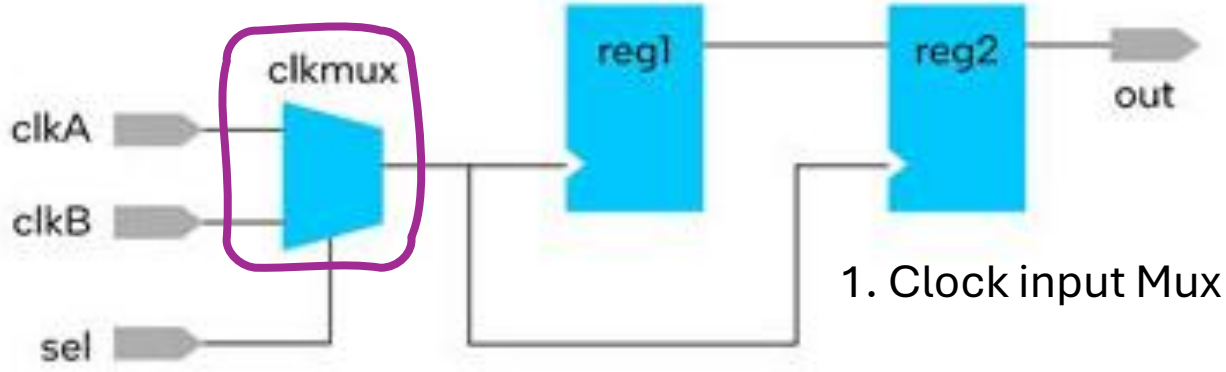


S_0	S_1	Y
0	0	A
0	1	B
1	0	C
1	1	D

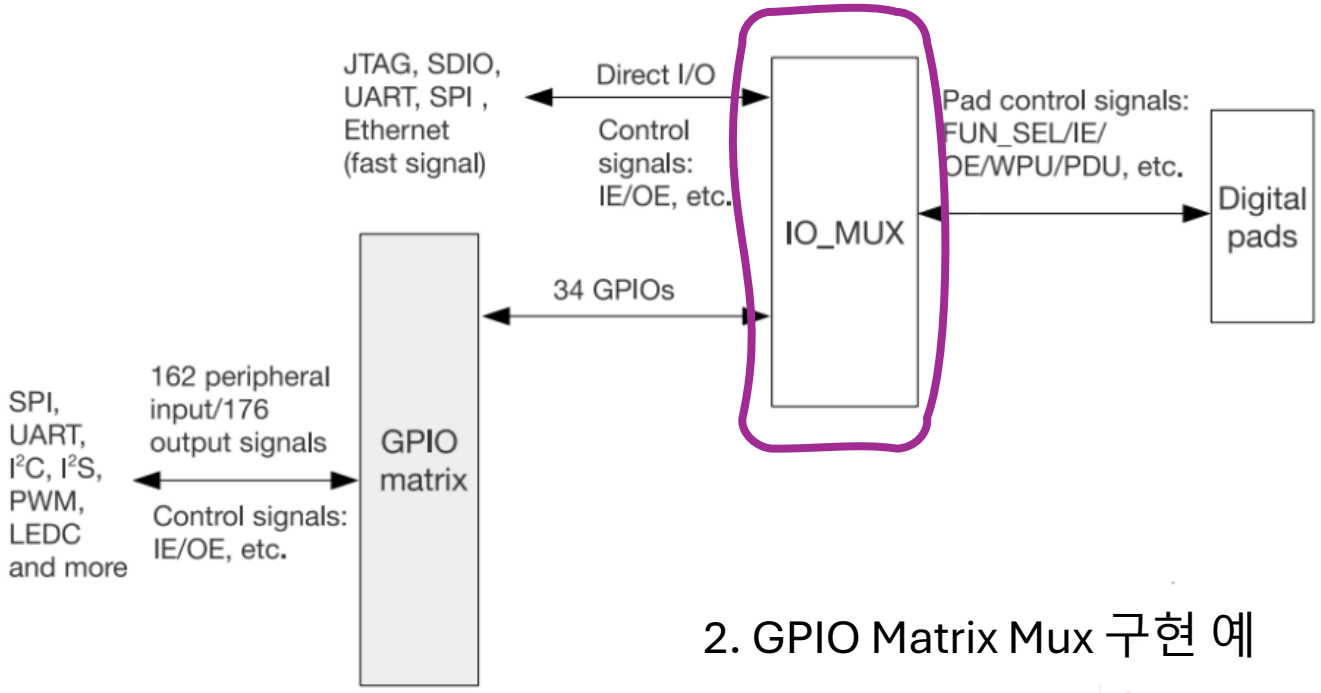


(a) Block diagram of 2^n : 1 multiplexer

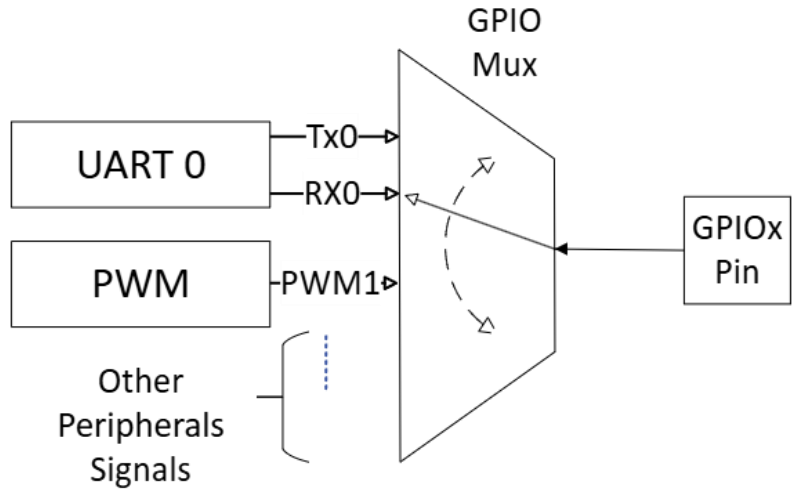
멀티플렉서를 활용한 핀 구성 사례



1. Clock input Mux



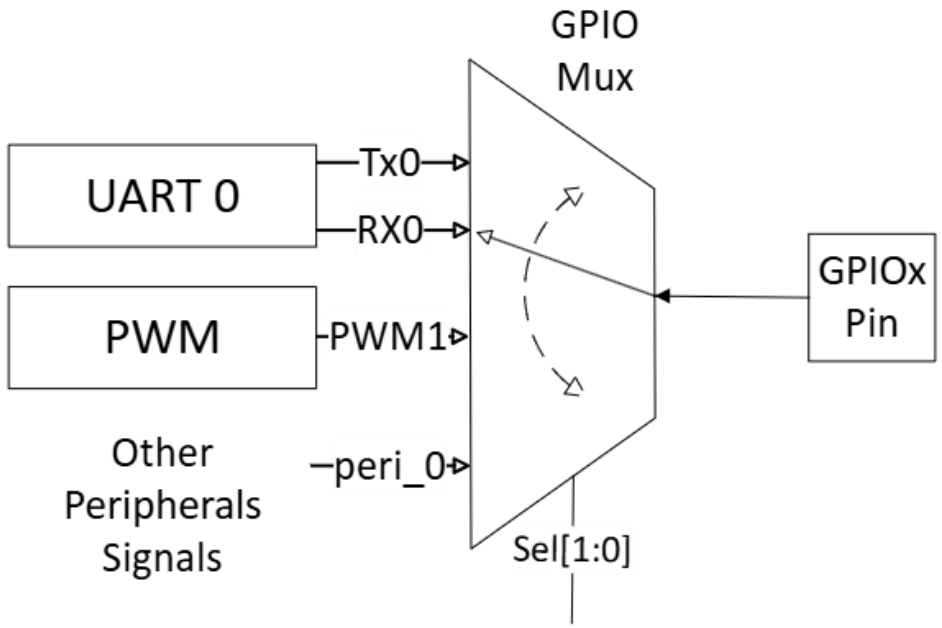
2. GPIO Matrix Mux 구현 예



간단한 GPIO Mux Model

멀티플렉서를 활용한 실습

- 아래 그림의 Mux 를 구현한 예입니다.
  - 코드를 작성해 하고 시뮬레이션 해보십시오.
  - 테스트 벡터는 다음장을 참조바랍니다




```
1   `timescale 1ns / 1ps
2   //peri_mux_io.v
3   module gpio_mux (
4       input wire [1:0] sel,       // 2비트 선택 신호
5       input wire tx0,             // UART0 Tx 신호
6       input wire rx0,             // UART0 Rx 신호
7       input wire pwm1,            // PWM 신호
8       input wire other_signal,    // 기타 주변 장치 신호
9       output reg gpio_pin         // GPIO 핀 출력
10  );
11
12      always @(*) begin
13          case (sel)
14              2'b00: gpio_pin = tx0;          // UART0 Tx 선택
15              2'b01: gpio_pin = rx0;          // UART0 Rx 선택
16              2'b10: gpio_pin = pwm1;         // PWM 신호 선택
17              2'b11: gpio_pin = other_signal; // 기타 신호 선택
18              default: gpio_pin = 1'bz;       // 기본값: High-Z 상태
19          endcase
20      end
21
22  endmodule
```

멀티플렉서를 활용한 실습 – 시뮬레이션 벡터

테스트 벡터는 sel입력에 따른 다양한 peri 선택의 예 입니다.

```
initial begin
    // 초기화
    tx0 = 1'b0; rx0 = 1'b1; pwm1 = 1'b0; other_signal = 1'b1;

    // 테스트 케이스 실행
    sel = 2'b00; #10; // tx0 선택 -> gpio_pin = tx0 (값: 0)
    sel = 2'b01; #10; // rx0 선택 -> gpio_pin = rx0 (값: 1)
    sel = 2'b10; #10; // pwm1 선택 -> gpio_pin = pwm1 (값: 0)
    sel = 2'b11; #10; // other_signal 선택 -> gpio_pin = other_signal (값: 1)

    $finish;
end
```

멀티플렉서를 활용한 실습 – testbench code결과

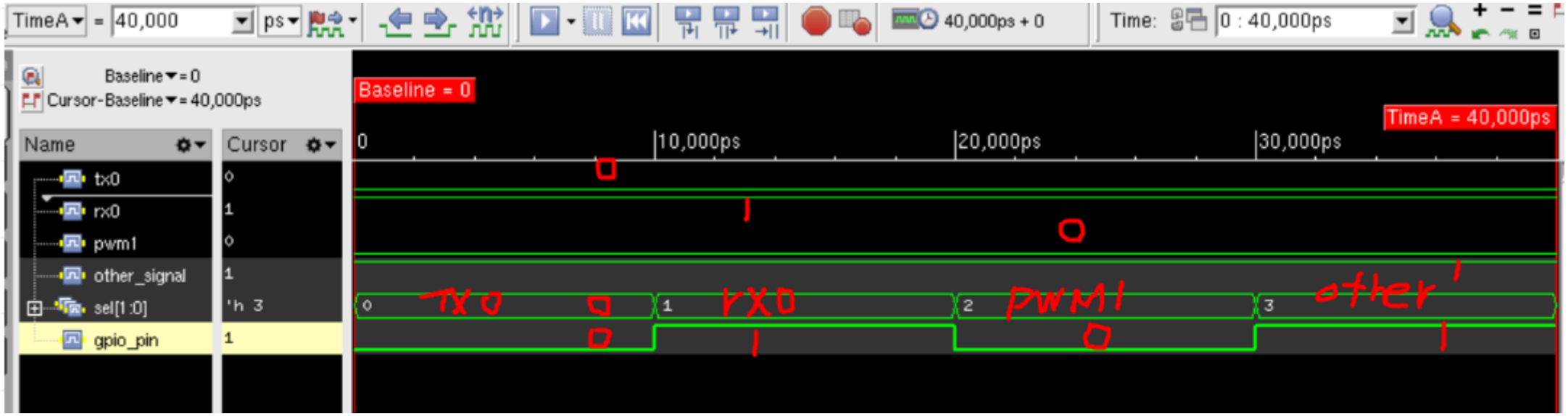
```
1   `timescale 1ns / 1ps
2   // tb_peri_mux_io.v
3   module gpio_mux_tb;
4
5       reg [1:0] sel;
6       reg tx0;
7       reg rx0;
8       reg pwm1;
9       reg other_signal;
10      wire gpio_pin;
11
12      // DUT (Device Under Test) 인스턴스화
13      gpio_mux dut (
14          .sel(sel),
15          .tx0(tx0),
16          .rx0(rx0),
17          .pwm1(pwm1),
18          .other_signal(other_signal),
19          .gpio_pin(gpio_pin)
20      );
```

```
22      initial begin
23          // 초기화
24          tx0 = 1'b0; rx0 = 1'b1; pwm1 = 1'b0; other_signal = 1'b1;
25
26          // 테스트 케이스 실행
27          sel = 2'b00; #10; // tx0 선택 -> gpio_pin = tx0 (값: 0)
28          sel = 2'b01; #10; // rx0 선택 -> gpio_pin = rx0 (값: 1)
29          sel = 2'b10; #10; // pwm1 선택 -> gpio_pin = pwm1 (값: 0)
30          sel = 2'b11; #10; // other_signal 선택 -> gpio_pin = other_signal (값: 1)
31
32          $finish;
33      end
34
35  endmodule
```

멀티플렉서를 활용한 실습 – 시뮬레이션 결과

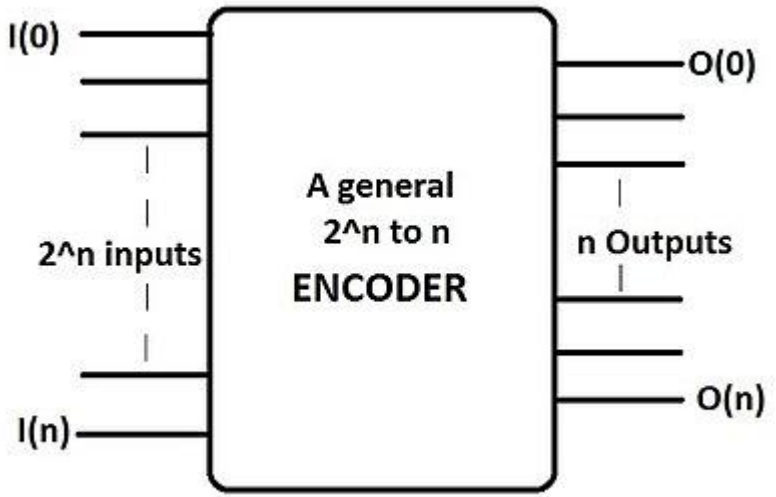
- 시뮬레이션 결과 확인

sel=00 -> gpio_pin=0 (tx0)
sel=01 -> gpio_pin=1 (rx0)
sel=10 -> gpio_pin=0 (pwm1)
sel=11 -> gpio_pin=1 (other_signal)

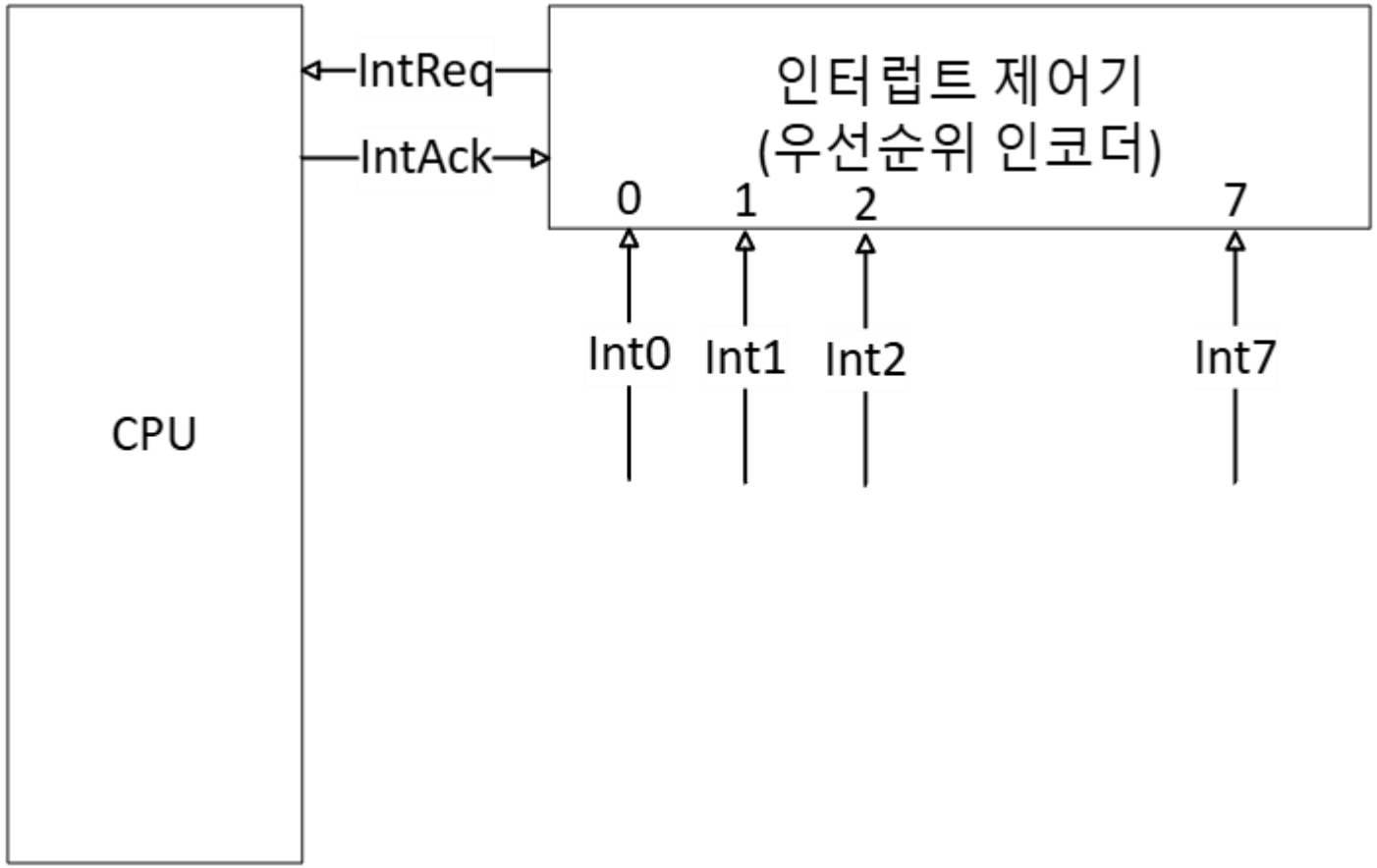


SoC 기본 조합논리 구성 블럭 – 인코더

- 인코더 (Encoder)
  - 인코더는 사람이 이해하기 쉬운 정보를 기계가 처리할 수 있는 코드화된 형식으로 변환하는 회로입니다.
  - 2^n개의 입력 라인을 n개의 출력 라인으로 변환합니다.
  - 일반적으로 한 번에 하나의 입력만 활성화됩니다.
  - 주요 유형:
    - 4-to-2 인코더
    - 8-to-3 인코더 (8진 인코더)
    - 10진수-to-BCD 인코더
  - 주요 사용 예
    - 키패드 스캔 로직
    - 인터럽트 제어기

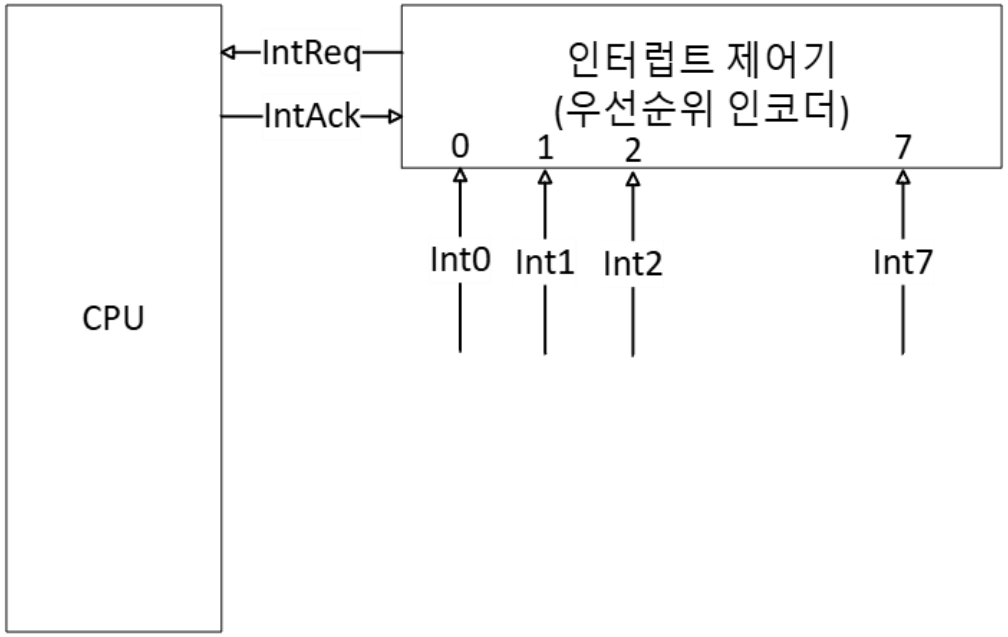


인코더 (Encoder) 를 활용한 예



인코더 (Encoder) 를 활용한 실습

- 아래 그림의 Encoder 를 이용한 인터럽트 컨트롤러를 구현한 예입니다.
  - 코드를 작성해 하고 시뮬레이션 해보십시오.
  - 테스트 벡터는 다음장을 참조바랍니다




```
1  `timescale 1ns / 1ps
2  //int_cntl_priority_encoder.v
3  module interrupt_controller (
4      input [7:0] Int,         // 8개의 인터럽트 입력
5      input IntAck,            // CPU로부터의 인터럽트 승인 신호
6      output reg [2:0] IntID, // 활성화된 인터럽트의 인덱스 출력
7      output reg IntReq        // 유효한 인터럽트 요청 신호
8  );
9
10 reg [7:0] active_int;        // 활성화된 인터럽트를 추적
11
12 always @(*) begin
13     IntReq = 1'b0;           // 기본값: 요청 없음
14     IntID = 3'b000;          // 기본값: 출력 없음
15     active_int = Int;        // 현재 활성화된 인터럽트를 저장
16
17     if (IntAck) begin
18         active_int = 8'b0000_0000; // 승인 시 모든 요청 초기화
19     end else begin
20         casex (active_int)
21             8'b1xxx_xxxx: begin IntID = 3'b111; IntReq = 1'b1; end // Int7
22             8'b01xx_xxxx: begin IntID = 3'b110; IntReq = 1'b1; end // Int6
23             8'b001x_xxxx: begin IntID = 3'b101; IntReq = 1'b1; end // Int5
24             8'b0001_xxxx: begin IntID = 3'b100; IntReq = 1'b1; end // Int4
25             8'b0000_1xxx: begin IntID = 3'b011; IntReq = 1'b1; end // Int3
26             8'b0000_01xx: begin IntID = 3'b010; IntReq = 1'b1; end // Int2
27             8'b0000_001x: begin IntID = 3'b001; IntReq = 1'b1; end // Int1
28             8'b0000_0001: begin IntID = 3'b000; IntReq = 1'b1; end // Int0
29             default: begin IntID = 3'b000; IntReq = 1'b0; end      // 요청 없음
30         endcase
31     end
32 end
33
34 endmodule
```

인코더 (Encoder) 를 활용한 실습– 시뮬레이션 벡터

테스트 벡터는 Int입력에 따른 interrupt 처리 예 입니다.

```
19      // 테스트 시뮬레이션
20      initial begin
21          // 모니터링: 시뮬레이션 결과를 출력
22          $monitor("Time=%0t | Int=%b | IntAck=%b | IntID=%b | IntReq=%b",
23                   $time, Int, IntAck, IntID, IntReq);
24
25          // 초기화
26          Int = 8'b0000_0000; // 모든 인터럽트 비활성화
27          IntAck = 1'b0;      // 승인 신호 비활성화
28
29          #10 Int = 8'b0000_0001;  // Int0 활성화 (우선순위 최하위)
30          #10 Int = 8'b0010_0000;  // Int5 활성화 (우선순위 중간)
31          #10 Int = 8'b1000_0000;  // Int7 활성화 (우선순위 최고)
32          #10 IntAck = 1'b1;       // CPU가 인터럽트를 수락 (모든 요청 초기화)
33          #10 IntAck = 1'b0;       // 승인 신호 비활성화 후 다음 요청 대기
34
35          #10 Int = 8'b0100_0010;  // 여러 입력 활성화 (Int6 선택)
36          #10 IntAck = 1'b1;       // CPU가 인터럽트를 수락
37          #10 IntAck = 1'b0;
38
39          #10 Int = 8'b0000_0000;  // 모든 입력 비활성화
40
41          #20 $finish;             // 시뮬레이션 종료
42      end
```

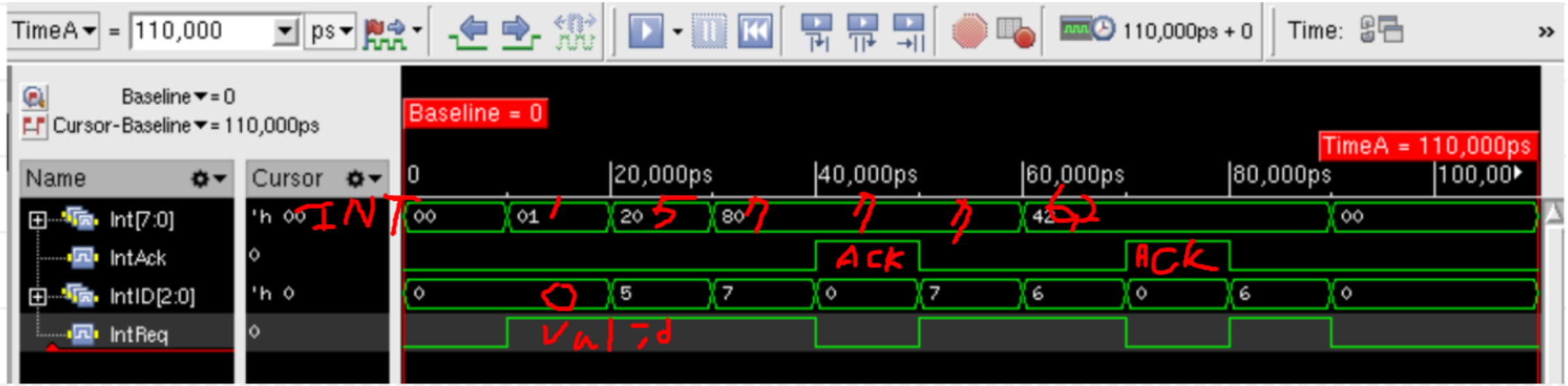
인코더 (Encoder) 를 활용한 실습– testbench code결과

```
1  `timescale 1ns / 1ps
2  //tb_int_cntl.v
3  module tb_interrupt_controller;
4
5      // 테스트 벤치에서 사용할 신호 선언
6      reg [7:0] Int;          // 8개의 인터럽트 입력
7      reg IntAck;             // CPU로부터의 인터럽트 승인 신호
8      wire [2:0] IntID;       // 활성화된 인터럽트의 인덱스 출력
9      wire IntReq;            // 유효한 인터럽트 요청 신호
10
11     // DUT (Device Under Test) 인스턴스화
12     interrupt_controller uut (
13         .Int(Int),
14         .IntAck(IntAck),
15         .IntID(IntID),
16         .IntReq(IntReq)
17     );
```

```
19     // 테스트 시뮬레이션
20     initial begin
21         // 모니터링: 시뮬레이션 결과를 출력
22         $monitor("Time=%0t | Int=%b | IntAck=%b | IntID=%b | IntReq=%b",
23                  $time, Int, IntAck, IntID, IntReq);
24
25         // 초기화
26         Int = 8'b0000_0000; // 모든 인터럽트 비활성화
27         IntAck = 1'b0;      // 승인 신호 비활성화
28
29         #10 Int = 8'b0000_0001;  // Int0 활성화 (우선순위 최하위)
30         #10 Int = 8'b0010_0000;  // Int5 활성화 (우선순위 중간)
31         #10 Int = 8'b1000_0000;  // Int7 활성화 (우선순위 최고)
32         #10 IntAck = 1'b1;       // CPU가 인터럽트를 수락 (모든 요청 초기화)
33         #10 IntAck = 1'b0;       // 승인 신호 비활성화 후 다음 요청 대기
34
35         #10 Int = 8'b0100_0010;  // 여러 입력 활성화 (Int6 선택)
36         #10 IntAck = 1'b1;       // CPU가 인터럽트를 수락
37         #10 IntAck = 1'b0;
38
39         #10 Int = 8'b0000_0000;  // 모든 입력 비활성화
40
41         #20 $finish;             // 시뮬레이션 종료
42     end
```

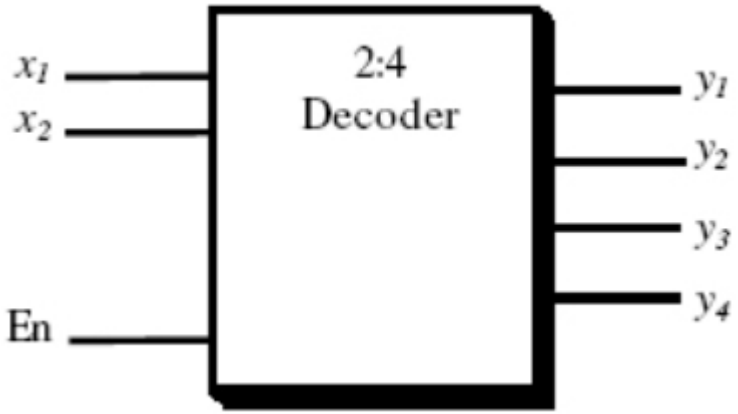
인코더 (Encoder) 를 활용한 실습– 시뮬레이션 결과

- 시뮬레이션 결과 확인

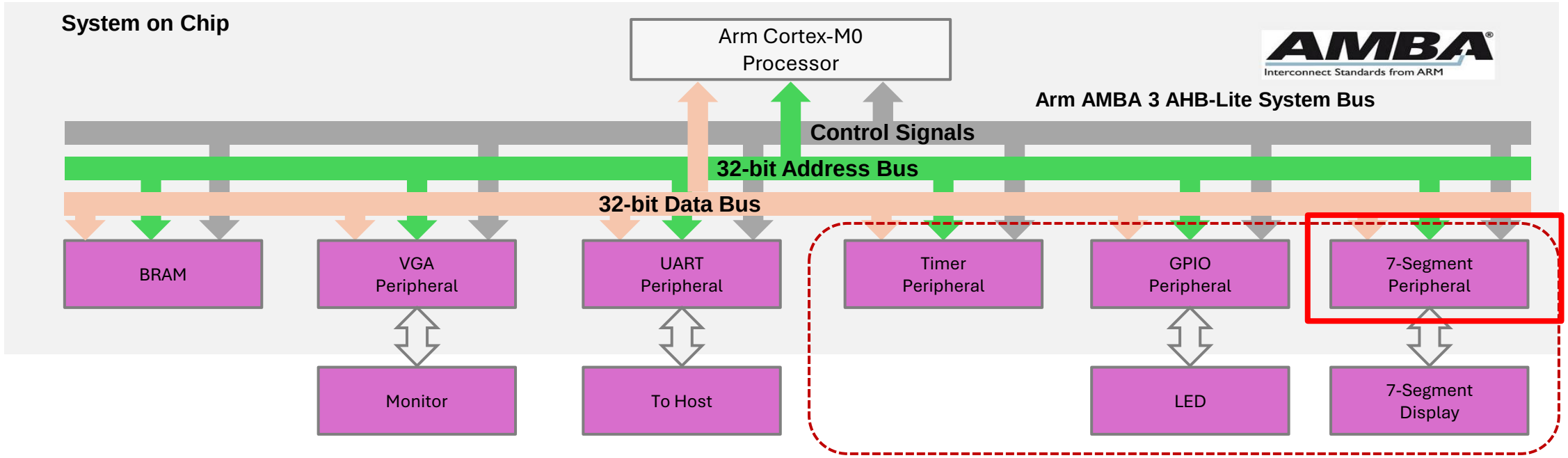


SoC 기본 조합논리 구성 블럭 – 디코더

- 디코더 (Decoder)
  - 디코더는 인코더의 반대 기능을 수행하며, 코드화된 입력을 원래의 형식으로 변환합니다.
  - n개의 입력 라인을 2^n개의 출력 라인으로 변환합니다.
  - 입력 코드에 따라 하나의 출력만 활성화됩니다.
  - 주요 유형:
    - 2-to-4 디코더
    - 3-to-8 디코더
    - 4-to-16 디코더
    - BCD-to-7세그먼트 디코더



디코더 (Decoder) 를 활용한 예



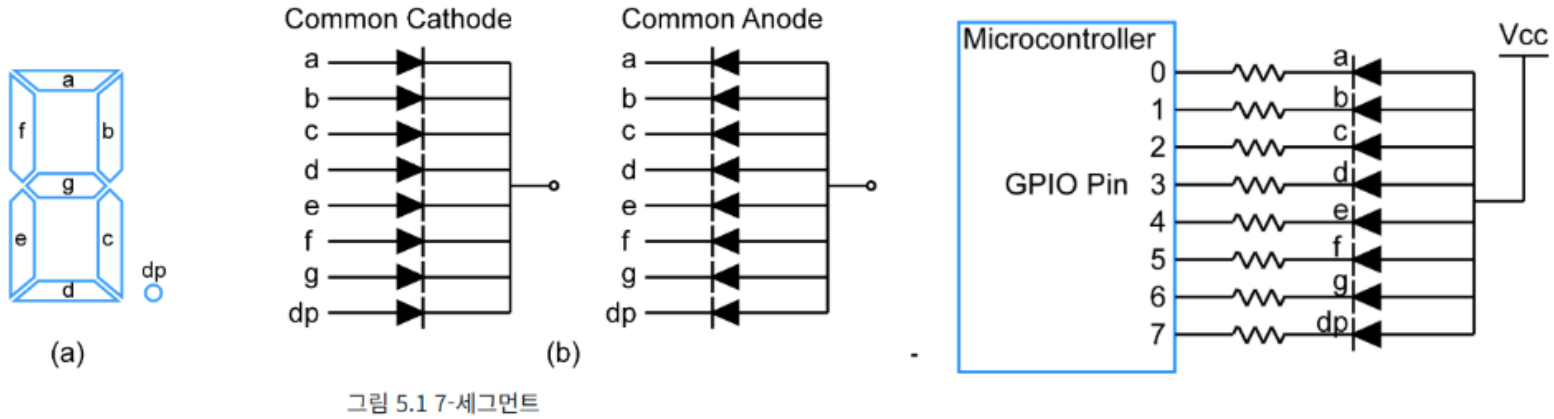
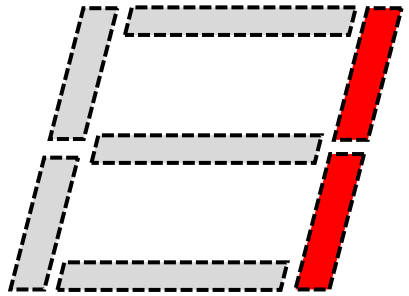


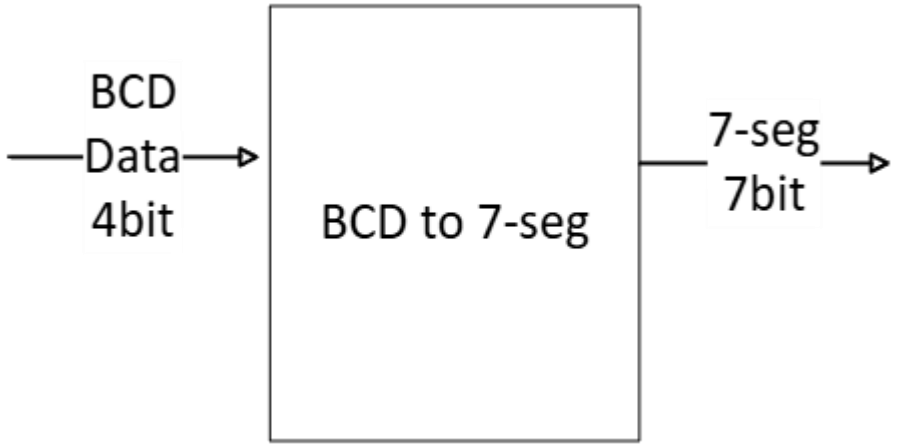
그림 5.1 7-세그먼트



Number '1'

디코더 (Decoder)를 활용한 실습

- 아래 그림의 BCD디코더는 7-세그먼트를 구현한 예입니다.
  - 코드를 작성해 하고 시뮬레이션 해보십시오.
  - 테스트 벡터는 다음장을 참조바랍니다




```
1   `timescale 1ns / 1ps
2   //bcdtoseg.v
3   module bcd_to_7seg (
4       input [3:0] bcd,        // 4비트 BCD 입력
5       output reg [6:0] seg    // 7-세그먼트 출력 (a-g)
6   );
7
8   always @(*) begin
9       case (bcd)
10          4'b0000: seg = 7'b1111110; // 0
11          4'b0001: seg = 7'b0110000; // 1
12          4'b0010: seg = 7'b1101101; // 2
13          4'b0011: seg = 7'b1111001; // 3
14          4'b0100: seg = 7'b0110011; // 4
15          4'b0101: seg = 7'b1011011; // 5
16          4'b0110: seg = 7'b1011111; // 6
17          4'b0111: seg = 7'b1110000; // 7
18          4'b1000: seg = 7'b1111111; // 8
19          4'b1001: seg = 7'b1111011; // 9
20          default: seg = 7'b0000000; // 비활성화 상태
21      endcase
22  end
23
24  endmodule
```

디코더 (Decoder)를 활용한 실습– 시뮬레이션 벡터

테스트 벡터는 Int입력에 따른 interrupt 처리 예 입니다.

```
14    initial begin
15        $monitor("Time=%0t | BCD Input=%b | Seven Segment Output=%b", $time, bcd, seg);
16
17        // 테스트 케이스들
18        bcd = 4'b0000; #10; // 입력: 0 -> 출력: a~g = "1111110"
19        bcd = 4'b0001; #10; // 입력: 1 -> 출력: a~g = "0110000"
20        bcd = 4'b0010; #10; // 입력: 2 -> 출력: a~g = "1101101"
21        bcd = 4'b0011; #10; // 입력: 3 -> 출력: a~g = "1111001"
22        bcd = 4'b0100; #10; // 입력: 4 -> 출력: a~g = "0110011"
23        bcd = 4'b0101; #10; // 입력: 5 -> 출력: a~g = "1011011"
24        bcd = 4'b0110; #10; // 입력: 6 -> 출력: a~g = "1011111"
25        bcd = 4'b0111; #10; // 입력: 7 -> 출력: a~g = "1110000"
26        bcd = 4'b1000; #10; // 입력: 8 -> 출력: a~g = "1111111"
27        bcd = 4'b1001; #10; // 입력: 9 -> 출력: a~g = "1111011"
28
29        $finish;
30    end
```

디코더 (Decoder)를 활용한 실습– testbench code결과

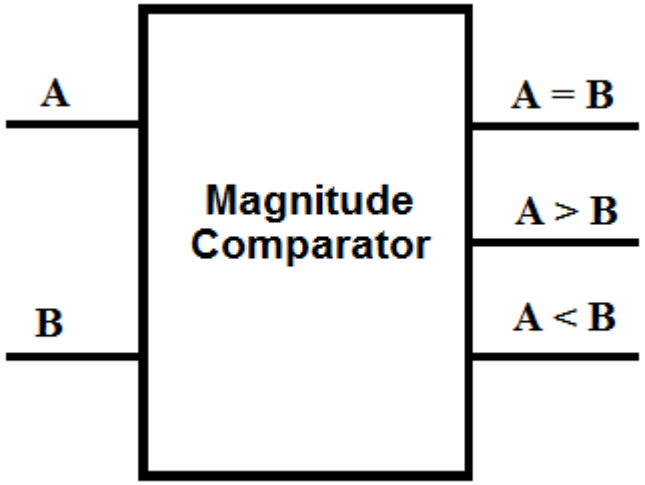
```
1  `timescale 1ns / 1ps
2  //tb_bcdtoseg.v
3  module tb_bcd_to_7seg;
4
5  reg [3:0] bcd;          // 테스트용 BCD 입력
6  wire [6:0] seg;         // 출력 (7-세그먼트 제어 신호)
7
8  // DUT(Design Under Test) 인스턴스 생성
9  bcd_to_7seg uut (
10     .bcd(bcd),
11     .seg(seg)
12 );
13
14 initial begin
15     $monitor("Time=%0t | BCD Input=%b | Seven Segment Output=%b", $time, bcd, seg);
16
17     // 테스트 케이스들
18     bcd = 4'b0000; #10; // 입력: 0 -> 출력: a~g = "1111110"
19     bcd = 4'b0001; #10; // 입력: 1 -> 출력: a~g = "0110000"
20     bcd = 4'b0010; #10; // 입력: 2 -> 출력: a~g = "1101101"
21     bcd = 4'b0011; #10; // 입력: 3 -> 출력: a~g = "1111001"
22     bcd = 4'b0100; #10; // 입력: 4 -> 출력: a~g = "0110011"
23     bcd = 4'b0101; #10; // 입력: 5 -> 출력: a~g = "1011011"
24     bcd = 4'b0110; #10; // 입력: 6 -> 출력: a~g = "1011111"
25     bcd = 4'b0111; #10; // 입력: 7 -> 출력: a~g = "1110000"
26     bcd = 4'b1000; #10; // 입력: 8 -> 출력: a~g = "1111111"
27     bcd = 4'b1001; #10; // 입력: 9 -> 출력: a~g = "1111011"
28
29     $finish;
30 end
31
32 endmodule
```

디코더 (Decoder)를 활용한 실습– 시뮬레이션 결과

- 시뮬레이션 결과 확인

SoC 기본 조합논리 구성 블럭 – 비교기

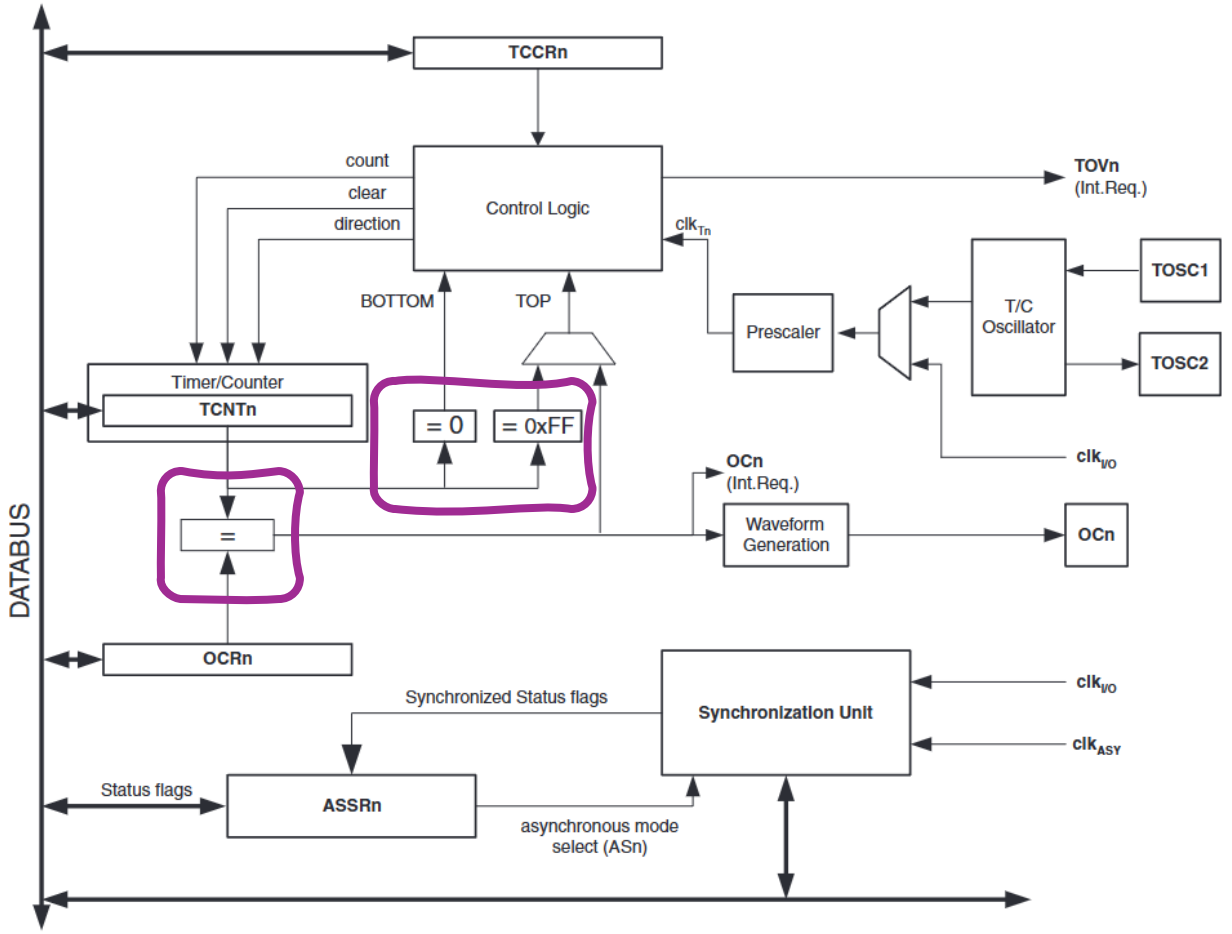
- 비교기 (comparator)
  - 두 개의 n비트 이진수 입력 (A와 B) 크기를 비교
  - 세 개의 출력: A > B, A = B, A < B
  - 동작 원리
    - 입력된 두 이진수를 비트 단위로 비교합니다.
    - 최상위 비트(MSB)부터 시작하여 하위 비트로 순차적으로 비교합니다.
    - 첫 번째로 다른 비트가 발견되면, 그 비트의 값에 따라 크기를 결정합니다.



비교기 (comparator) 를 활용한 예

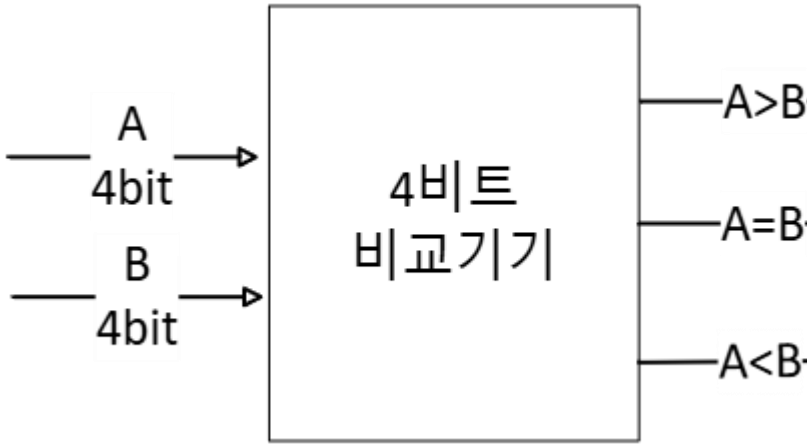
- 데이터 정렬 및 검색:
  - 메모리 관리 유닛(MMU)에서 주소 비교
  - 캐시 메모리의 태그 비교
- ADC 인터페이스:
  - ADC 출력값과 기준값 비교
- 타이머/카운터 모듈:
  - 카운트 값과 설정값 비교
- 전력 관리:
  - 전압 레벨 모니터링 및 비교

Figure 53. 8-bit Timer/Counter Block Diagram



비교기 (comparator) 를 활용한 실습

- 아래 그림의 4비트 비교기를 구현한 예입니다.
  - 코드를 작성해 하고 시뮬레이션 해보십시오.
  - 테스트 벡터는 다음장을 참조바랍니다




```
1   `timescale 1ns / 1ps
2   //comparator.v
3   module comparator #(parameter WIDTH = 4)(
4     input [WIDTH-1:0] a, b,
5     output equal, greater, less
6   );
7     assign equal = (a == b);
8     assign greater = (a > b);
9     assign less = (a < b);
10  endmodule
```

비교기 (comparator) 를 활용한 실습– 시뮬레이션 벡터

```
20  // 테스트 시나리오
21  initial begin
22    $monitor("%0t\t %b\t %b\t %b\t %b\t %b", $time, A, B, greater, equal, less);
23
24    // 테스트 케이스
25    a = 4'b0000; b = 4'b0000; #10; // 같음
26    a = 4'b0101; b = 4'b0011; #10; // A > B
27    a = 4'b0010; b = 4'b1000; #10; // A < B
28    a = 4'b1111; b = 4'b1111; #10; // 같음 (최대값)
29    a = 4'b1010; b = 4'b0101; #10; // A > B
30    a = 4'b0111; b = 4'b1001; #10; // A < B
31    a = 4'b1100; b = 4'b1100; #10; // 같음
32    a = 4'b0000; b = 4'b1111; #10; // A < B (최소값 vs 최대값)
33
34    // 시뮬레이션 종료
35    #10 $finish;
36  end
```

비교기 (comparator) 를 활용한 실습– testbench code결과

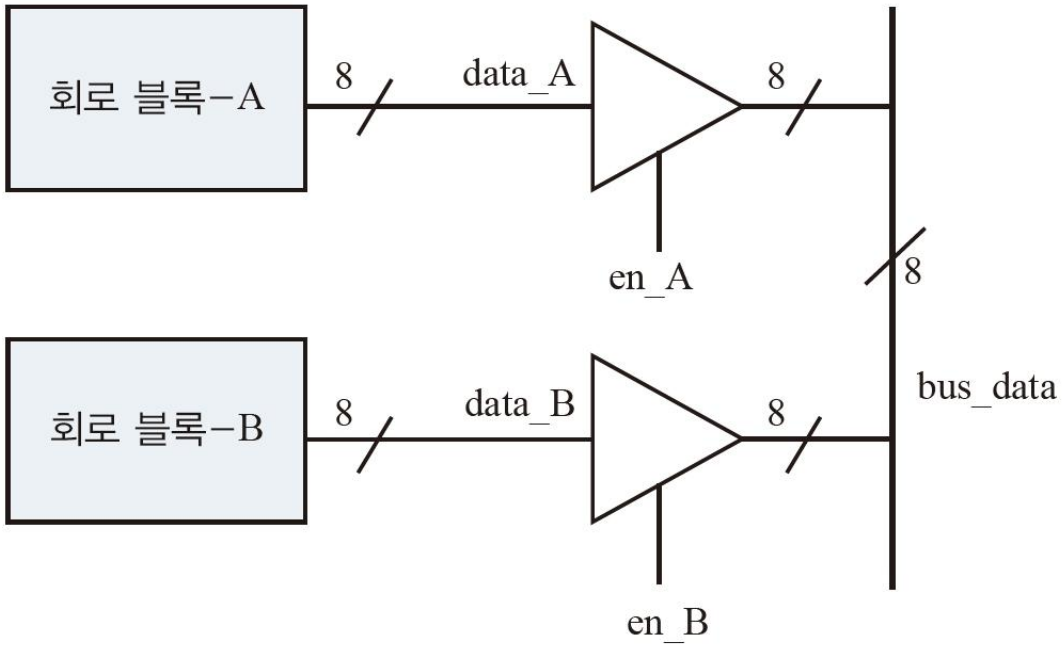
```
1  `timescale 1ns / 1ps
2  //tb_bcdtoseg.v
3  module tb_bcd_to_7seg;
4
5  reg [3:0] bcd;          // 테스트용 BCD 입력
6  wire [6:0] seg;         // 출력 (7-세그먼트 제어 신호)
7
8  // DUT(Design Under Test) 인스턴스 생성
9  bcd_to_7seg uut (
10     .bcd(bcd),
11     .seg(seg)
12 );
13
14 initial begin
15     $monitor("Time=%0t | BCD Input=%b | Seven Segment Output=%b", $time, bcd, seg);
16
17     // 테스트 케이스들
18     bcd = 4'b0000; #10; // 입력: 0 -> 출력: a~g = "1111110"
19     bcd = 4'b0001; #10; // 입력: 1 -> 출력: a~g = "0110000"
20     bcd = 4'b0010; #10; // 입력: 2 -> 출력: a~g = "1101101"
21     bcd = 4'b0011; #10; // 입력: 3 -> 출력: a~g = "1111001"
22     bcd = 4'b0100; #10; // 입력: 4 -> 출력: a~g = "0110011"
23     bcd = 4'b0101; #10; // 입력: 5 -> 출력: a~g = "1011011"
24     bcd = 4'b0110; #10; // 입력: 6 -> 출력: a~g = "1011111"
25     bcd = 4'b0111; #10; // 입력: 7 -> 출력: a~g = "1110000"
26     bcd = 4'b1000; #10; // 입력: 8 -> 출력: a~g = "1111111"
27     bcd = 4'b1001; #10; // 입력: 9 -> 출력: a~g = "1111011"
28
29     $finish;
30 end
31
32 endmodule
```

비교기 (comparator) 를 활용한 실습– 시뮬레이션 결과

- 시뮬레이션 결과 확인

SoC 기본 조합논리 구성 블럭 - 3상태 버스

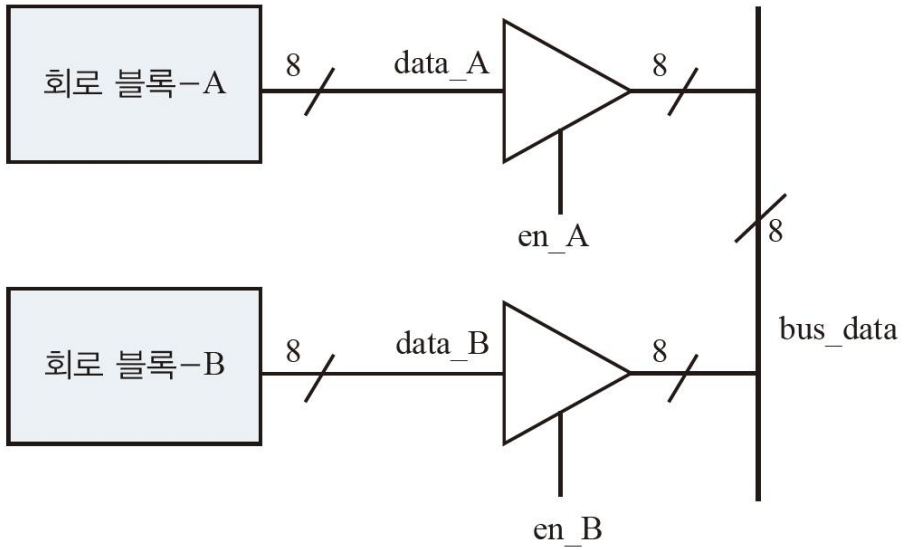
- 3상태 버스 드라이버
  - 다수의 신호원들이 공동으로 사용하는 버스 (bus)에 데이터를 보내거나 또는 신호원과 버스를 격리시켜 high-impedance 상태로 만드는 회로



[그림 10–14] 데이터 버스 드라이버

3상태 버스 를 활용한 실습

- 아래 그림의 3상태 버스를 구현한 예입니다.
  - 코드를 작성해 하고 시뮬레이션 해보십시오.
  - 테스트 벡터는 다음장을 참조바랍니다



[그림 10-14] 데이터 버스 드라이버

```
1   `timescale 1ns / 1ps
2   //3state_ex0.v
3   module data_bus_driver (
4       input [7:0] data_A,     // 회로 블록 A의 데이터
5       input [7:0] data_B,     // 회로 블록 B의 데이터
6       input en_A,             // 회로 블록 A의 활성화 신호
7       input en_B,             // 회로 블록 B의 활성화 신호
8       output [7:0] bus_data   // 공통 데이터 버스
9   );
10
11      // 3상태 버퍼 구현
12      assign bus_data = (en_A) ? data_A :
13                        (en_B) ? data_B :
14                        8'bz;  // 고임피던스 상태
15
16  endmodule
```

3상태 버스 를 활용한 실습– 시뮬레이션 벡터

테스트 벡터는 3상태 버스를 시뮬레이션 하기위한 벡터 입니다.

```
20      initial begin
21          $monitor("Time=%0t | en_A=%b, data_A=%b | en_B=%b, data_B=%b | bus_data=%b",
22                   $time, en_A, data_A, en_B, data_B, bus_data);
23
24          // 초기값 설정
25          data_A = 8'b10101010; data_B = 8'b11001100;
26          en_A = 0; en_B = 0; #10; // 둘 다 비활성화 -> bus_data는 고임피던스 상태
27
28          en_A = 1; en_B = 0; #10; // A 활성화 -> bus_data는 data_A 출력
29          en_A = 0; en_B = 1; #10; // B 활성화 -> bus_data는 data_B 출력
30
31          en_A = 1; en_B = 1; #10; // 둘 다 활성화 -> 충돌 상황 (테스트에서 확인)
32
33          en_A = 0; en_B = 0; #10; // 둘 다 비활성화 -> bus_data는 고임피던스 상태
34
35          $finish;
36      end
```

3상태 버스 를 활용한 실습– testbench code결과

```
1   `timescale 1ns / 1ps
2   //tb_3state_ex0.v
3   module tb_data_bus_driver();
4       reg [7:0] data_A;       // 테스트용 회로 블록 A 데이터 입력
5       reg [7:0] data_B;       // 테스트용 회로 블록 B 데이터 입력
6       reg en_A;               // 테스트용 회로 블록 A 활성화 신호
7       reg en_B;               // 테스트용 회로 블록 B 활성화 신호
8       wire [7:0] bus_data;    // 공통 데이터 버스 출력
9
10      // DUT(Design Under Test) 인스턴스 생성
11      data_bus_driver uut (
12          .data_A(data_A),
13          .data_B(data_B),
14          .en_A(en_A),
15          .en_B(en_B),
16          .bus_data(bus_data)
17      );
18      initial begin
19          $monitor("Time=%0t | en_A=%b, data_A=%b | en_B=%b, data_B=%b | bus_data=%b",
20                   $time, en_A, data_A, en_B, data_B, bus_data);
21          // 초기값 설정
22          data_A = 8'b10101010; data_B = 8'b11001100;
23          en_A = 0; en_B = 0; #10; // 둘 다 비활성화 -> bus_data는 고임피던스 상태
24          en_A = 1; en_B = 0; #10; // A 활성화 -> bus_data는 data_A 출력
25          en_A = 0; en_B = 1; #10; // B 활성화 -> bus_data는 data_B 출력
26          en_A = 1; en_B = 1; #10; // 둘 다 활성화 -> 충돌 상황 (테스트에서 확인)
27          en_A = 0; en_B = 0; #10; // 둘 다 비활성화 -> bus_data는 고임피던스 상태
28
29          $finish;
30      end
31  endmodule
```

3상태 버스 를 활용한 실습– 시뮬레이션 결과

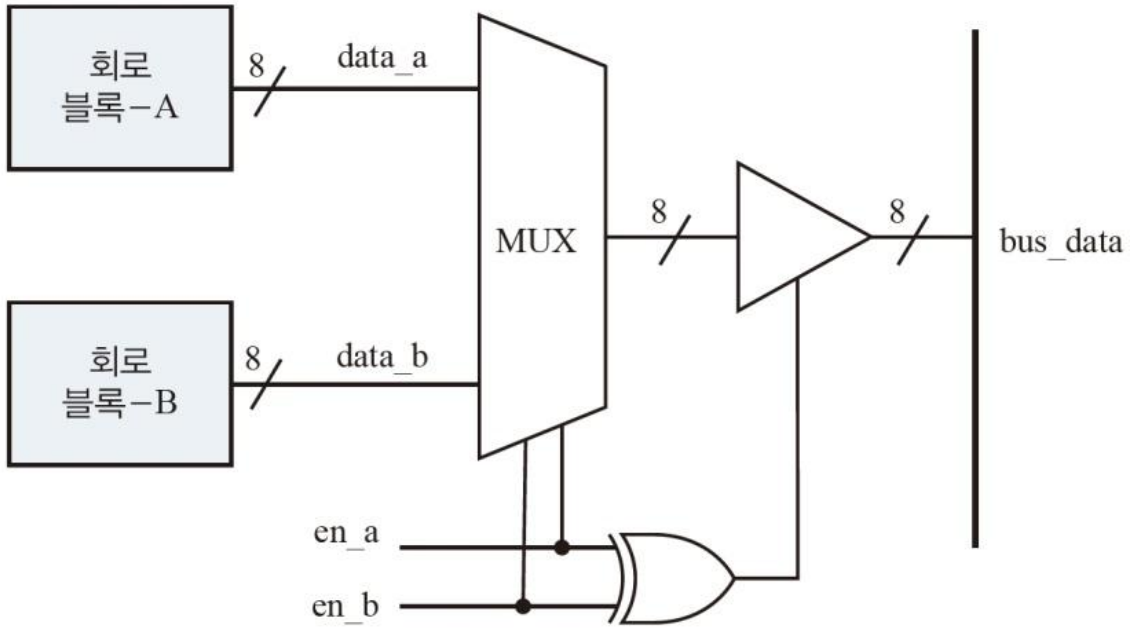
- 시뮬레이션 결과 확인

시뮬레이션 결과 예시

Time(ns)	en_A	data_A	en_B	data_B	bus_data
0	0	10101010	0	11001100	Z
10	1	10101010	0	11001100	10101010
20	0	10101010	1	11001100	11001100
30	1	10101010	1	11001100	X (충돌)
40	0	10101010	0	11001100	Z

SoC 기본 조합논리 구성 블럭 - 3상태 버스 사용예

- 오른쪽 그림은 3상태 버스의 충돌방지를 구현한 예입니다.
  - 코드를 작성해 하고 시뮬레이션 해보십시오.
  - 테스트 벡터는 다음장을 참조바랍니다



[그림 10-18] Multiplexed 버스 드라이버

```
1   `timescale 1ns / 1ps
2   //3state_ex1.v
3   module multiplexed_bus_driver (
4       input [7:0] data_a,     // 회로 블록 A의 데이터
5       input [7:0] data_b,     // 회로 블록 B의 데이터
6       input en_a,             // 회로 블록 A의 활성화 신호
7       input en_b,             // 회로 블록 B의 활성화 신호
8       output [7:0] bus_data   // 공통 데이터 버스
9   );
10
11      wire select;            // XOR 게이트 출력 (MUX 선택 신호)
12      wire [7:0] mux_out;     // MUX 출력
13
14      // XOR 게이트로 활성화 신호 결합
15      assign select = en_a ^ en_b;
16
17      // MUX 구현: 선택 신호에 따라 data_a 또는 data_b 선택
18      assign mux_out = (en_a) ? data_a :
19                       (en_b) ? data_b :
20                       8'bz;  // 고임피던스 상태
21
22      // 3상태 버퍼 구현: select 신호에 따라 MUX 출력 전달
23      assign bus_data = (select) ? mux_out : 8'bz;
24
25  endmodule
```

3상태 버스 를 활용한 실습– 시뮬레이션 벡터

- 아래 코드를 주석을 참조하여 각각의 경우에 맞는 벡터를 생성해 보시오.

```
1   `timescale 1ns / 1ps
2   //tb_3state_ex1.v
3   module tb_multiplexed_bus_driver();
4       reg [7:0] data_a;      // 테스트용 회로 블록 A 데이터 입력
5       reg [7:0] data_b;      // 테스트용 회로 블록 B 데이터 입력
6       reg en_a;              // 테스트용 회로 블록 A 활성화 신호
7       reg en_b;              // 테스트용 회로 블록 B 활성화 신호
8       wire [7:0] bus_data;   // 공통 데이터 버스 출력
9       // DUT(Design Under Test) 인스턴스 생성
10      multiplexed_bus_driver uut (
11          .data_a(data_a),
12          .data_b(data_b),
13          .en_a(en_a),
14          .en_b(en_b),
15          .bus_data(bus_data)
16      );
17      initial begin
18          $monitor("Time=%0t | en_a=%b, data_a=%b | en_b=%b, data_b=%b | bus_data=%b",
19                   $time, en_a, data_a, en_b, data_b, bus_data);
20          // 초기값 설정
21          data_a = 8'b10101010;
22          data_b = 8'b11001100;
23
24          // 테스트 케이스들
25          en_a = 1; en_b = 0; #10; // A 활성화 -> bus_data = data_a (10101010)
26          [redacted]#10; // B 활성화 -> bus_data = data_b (11001100)
27          #10; // XOR 조건 위반 -> bus_data는 고임피던스 상태(Z)
28          #10; // 둘 다 비활성화 -> bus_data는 고임피던스 상태(Z)
29          $finish;
30      end
31
32  endmodule
```

3상태 버스 를 활용한 실습– testbench code결과

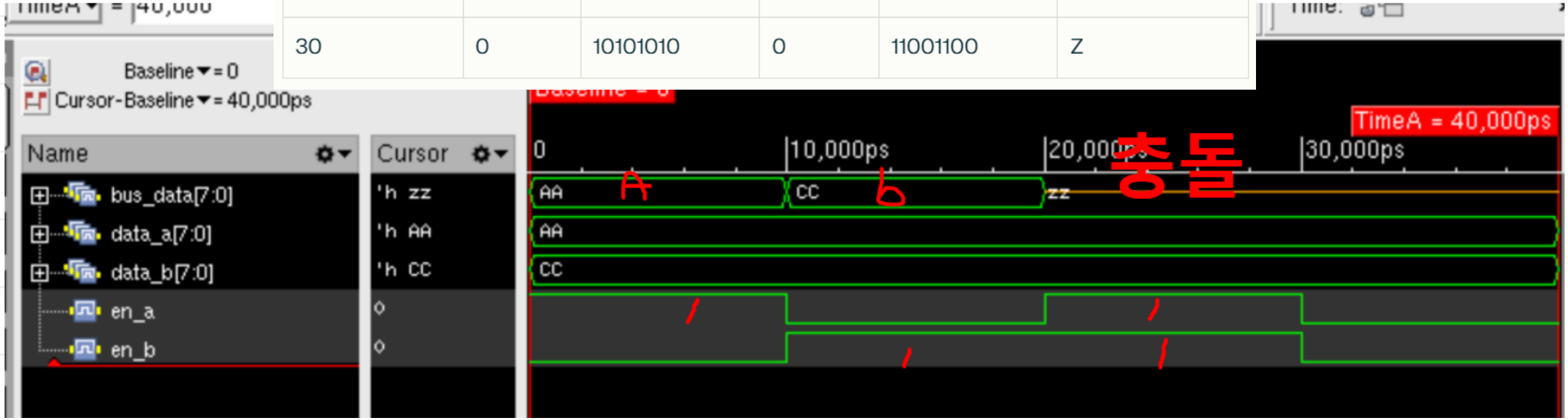
```
1  `timescale 1ns / 1ps
2  //tb_3state_ex1.v
3  module tb_multiplexed_bus_driver();
4      reg [7:0] data_a;      // 테스트용 회로 블록 A 데이터 입력
5      reg [7:0] data_b;      // 테스트용 회로 블록 B 데이터 입력
6      reg en_a;              // 테스트용 회로 블록 A 활성화 신호
7      reg en_b;              // 테스트용 회로 블록 B 활성화 신호
8      wire [7:0] bus_data;   // 공통 데이터 버스 출력
9      // DUT(Design Under Test) 인스턴스 생성
10     multiplexed_bus_driver uut (
11         .data_a(data_a),
12         .data_b(data_b),
13         .en_a(en_a),
14         .en_b(en_b),
15         .bus_data(bus_data)
16     );
17     initial begin
18         $monitor("Time=%0t | en_a=%b, data_a=%b | en_b=%b, data_b=%b | bus_data=%b",
19                  $time, en_a, data_a, en_b, data_b, bus_data);
20         // 초기값 설정
21         data_a = 8'b10101010;
22         data_b = 8'b11001100;
23
24         // 테스트 케이스들
25         en_a = 1; en_b = 0; #10; // A 활성화 -> bus_data = data_a (10101010)
26         en_a = 0; en_b = 1; #10; // B 활성화 -> bus_data = data_b (11001100)
27         en_a = 1; en_b = 1; #10; // XOR 조건 위반 -> bus_data는 고임피던스 상태(Z)
28         en_a = 0; en_b = 0; #10; // 둘 다 비활성화 -> bus_data는 고임피던스 상태(Z)
29         $finish;
30     end
31
32 endmodule
```

3상태 버스 를 활용한 실습– 시뮬레이션 결과

- 시뮬레이션 결과 확인

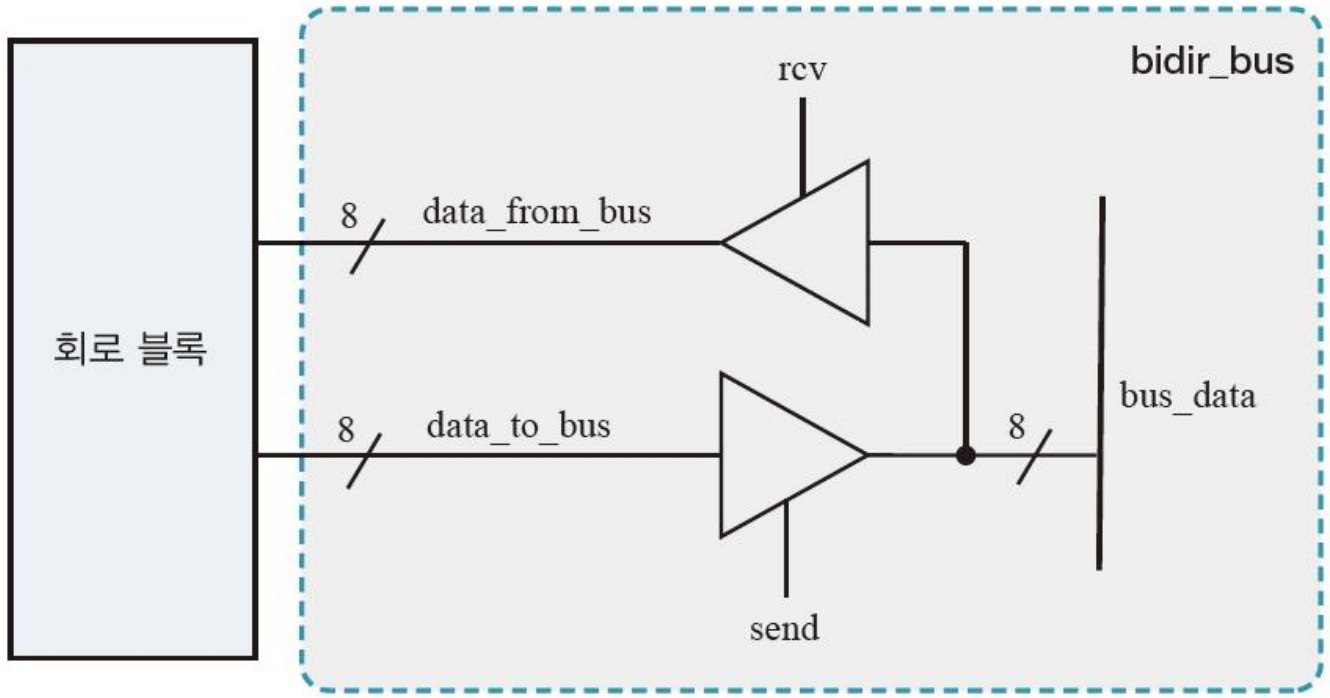
시뮬레이션 결과 예시

Time(ns)	en_a	data_a	en_b	data_b	bus_data
0	1	10101010	0	11001100	10101010
10	0	10101010	1	11001100	11001100
20	1	10101010	1	11001100	Z
30	0	10101010	0	11001100	Z



SoC 기본 조합논리 구성 블럭 - 3상태 버스

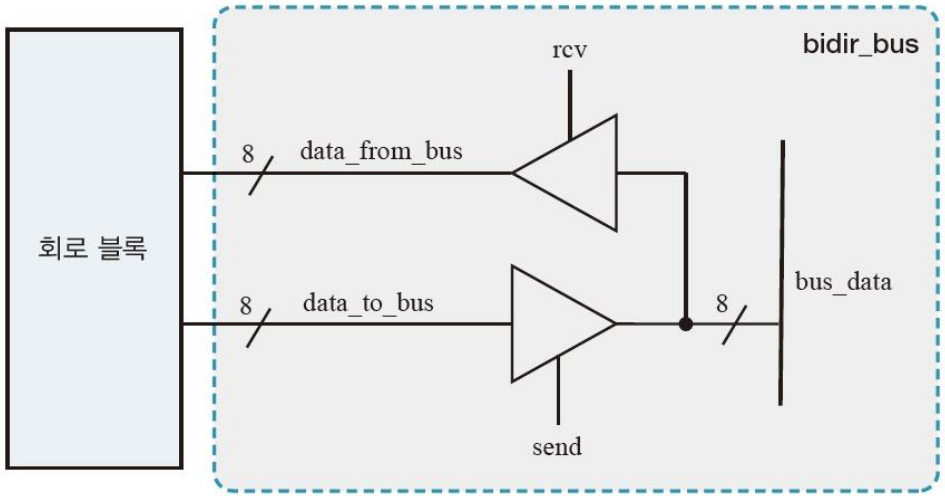
- 양방향 (bidirectional) 버스 드라이버
  - 보내기와 받기를 동시에 처리할 수 있는 회로
  - bus_data 포트는 **`inout`**으로 선언되어야 함



[그림 10–16] 8비트 양방향 버스 드라이버

3상태 버스 를 활용한 실습– 시뮬레이션 벡터

- 양방향 (bidirectional) 버스 드라이버를 구현한 예입니다.
  - 코드를 작성해 하고 시뮬레이션 해보십시오.
  - 테스트 벡터는 다음장을 참조바랍니다



[그림 10–16] 8비트 양방향 버스 드라이버

```
1   `timescale 1ns / 1ps
2   //3state_ex2.v
3   module bidirectional_bus_driver (
4       input [7:0] data_to_bus,     // 회로 블록에서 버스로 보낼 데이터
5       input send,                  // 데이터 전송 활성화 신호
6       input rcv,                   // 데이터 수신 활성화 신호
7       inout [7:0] bus_data,        // 양방향 데이터 버스
8       output reg [7:0] data_from_bus // 버스에서 읽어온 데이터
9   );
10
11      // 3상태 버퍼 구현: send 신호가 활성화되면 data_to_bus를 bus_data로 전달
12      assign bus_data = (send) ? data_to_bus : 8'bz;
13
14      // 데이터 수신 로직: rcv 신호가 활성화되면 bus_data를 data_from_bus로 읽어옴
15      always @(*) begin
16          if (rcv) begin
17              data_from_bus = bus_data;
18          end else begin
19              data_from_bus = 8'b0; // 기본값 설정
20          end
21      end
22
23  endmodule
```

3상태 버스 를 활용한 실습– 시뮬레이션 벡터

- 아래 코드를 주석을 참조하여 각각의 경우에 맞는 벡터를 생성해 보시오.

```
1   `timescale 1ns / 1ps
2   //tb_3state_ex2.v
3   module tb_bidirectional_bus_driver();
4       reg [7:0] data_to_bus;        // 송신 데이터 입력
5       reg send;                     // 송신 활성화 신호
6       reg rcv;                      // 수신 활성화 신호
7       wire [7:0] bus_data;          // 양방향 데이터 버스 (inout)
8       wire [7:0] data_from_bus;     // 수신 데이터 출력
9       // DUT(Design Under Test) 인스턴스 생성
10      bidirectional_bus_driver uut (
11          .data_to_bus(data_to_bus),
12          .send(send),
13          .rcv(rcv),
14          .bus_data(bus_data),
15          .data_from_bus(data_from_bus)
16      );
```

```
18      // 강제로 bus_data를 구동하기 위한 레지스터
        reg [7:0] bus_drive_data;
        assign bus_data = (rcv) ? bus_drive_data : 8'bz;  // rcv가 활성화되면 외부에서 값을 구동
        initial begin
            $monitor("Time=%0t | send=%b | rcv=%b | data_to_bus=%b | bus_data=%b | data_from_bus=%b",
                     $time, send, rcv, data_to_bus, bus_data, data_from_bus);
            // 초기 상태 설정
            data_to_bus = 8'b00000000;
            send = 0;
            rcv = 0;
            bus_drive_data = 8'b00000000;
            #10;
            // 송신 테스트 (send 활성화)
            [redacted]
            data_to_bus = 8'b10101010; #10;
            // 수신 테스트 (rcv 활성화)
            [redacted]
            bus_drive_data = 8'b11001100; #10;
            // 둘 다 비활성화된 상태 확인 (고임피던스 상태)
            send = 0;
            rcv = 0; #10;
            $finish;
        end
43  endmodule
```

3상태 버스 를 활용한 실습– testbench code결과

```
`timescale 1ns / 1ps
//tb_3state_ex2.v
module tb_bidirectional_bus_driver();
    reg [7:0] data_to_bus;       // 송신 데이터 입력
    reg send;                    // 송신 활성화 신호
    reg rcv;                     // 수신 활성화 신호
    wire [7:0] bus_data;         // 양방향 데이터 버스 (inout)
    wire [7:0] data_from_bus;    // 수신 데이터 출력
    // DUT(Design Under Test) 인스턴스 생성
    bidirectional_bus_driver uut (
        .data_to_bus(data_to_bus),
        .send(send),
        .rcv(rcv),
        .bus_data(bus_data),
        .data_from_bus(data_from_bus)
    );
```

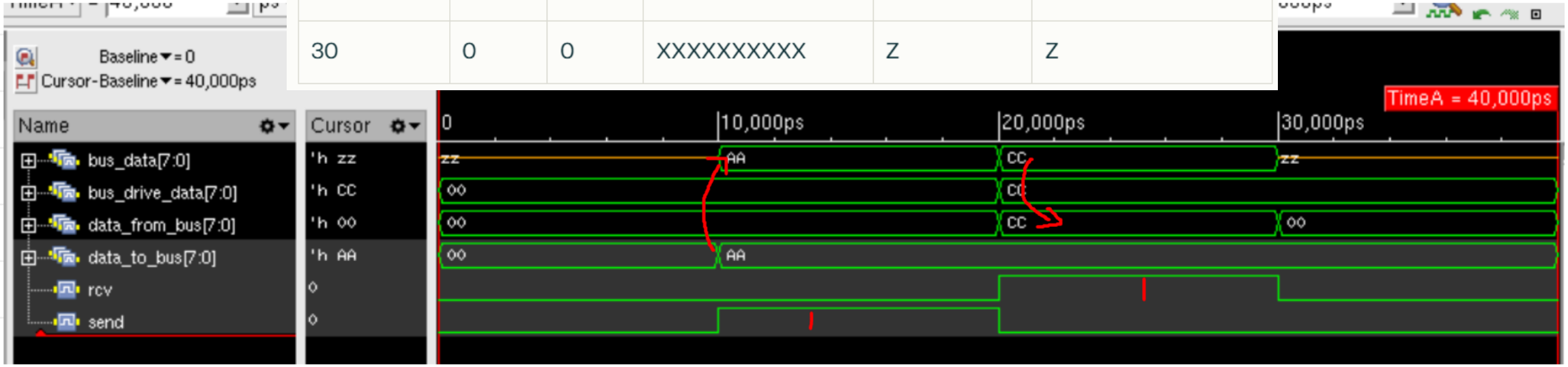
```
    // 강제로 bus_data를 구동하기 위한 레지스터
    reg [7:0] bus_drive_data;
    assign bus_data = (rcv) ? bus_drive_data : 8'bz;  // rcv가 활성화되면 외부에서 값을 구동
    initial begin
        $monitor("Time=%0t | send=%b | rcv=%b | data_to_bus=%b | bus_data=%b | data_from_bus=%b",
                 $time, send, rcv, data_to_bus, bus_data, data_from_bus);
        // 초기 상태 설정
        data_to_bus = 8'b00000000;
        send = 0;
        rcv = 0;
        bus_drive_data = 8'b00000000;
        #10;
        // 송신 테스트 (send 활성화)
        send = 1;
        rcv = 0;
        data_to_bus = 8'b10101010; #10;
        // 수신 테스트 (rcv 활성화)
        send = 0;
        rcv = 1;
        bus_drive_data = 8'b11001100; #10;
        // 둘 다 비활성화된 상태 확인 (고임피던스 상태)
        send = 0;
        rcv = 0; #10;
        $finish;
    end
endmodule
```

3상태 버스 를 활용한 실습– 시뮬레이션 결과

- 시뮬레이션 결과 확인

시뮬레이션 결과 예시

Time(ns)	send	rcv	data_to_bus	bus_data	data_from_bus
0	0	0	00000000	Z	00000000
10	1	0	10101010	10101010	Z
20	0	1	XXXXXXXXXX	11001100	11001100
30	0	0	XXXXXXXXXX	Z	Z



Thanks