12주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20231609 이름: 정희선

**1. 2-bit counter**

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input clk, rst,*  *output reg [1:0] out*  *);*  *// 초기 상태*  *initial out = 2'b00;*    *// 상승 엣지에서 동작*  *always @(posedge clk) begin*  *if (!rst) begin*  *out <= 2'b00; // 리셋 시 출력 초기화*  *end else begin*  *out <= out + 1; // 카운터 증가*  *end*  *end*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg clk, rst;*  *wire [1:0] out;*  *boolean u\_boolean(*  *.clk(clk),*  *.rst(rst),*  *.out(out)*  *);*  *initial begin*  *clk = 1'b0;*  *forever #50 clk = ~clk;*  *end*  *initial begin*  *rst = 1'b0;*  *#100 rst = 1'b1; // 100ns 후 리셋 해제*  *#200 rst = 1'b0; // 300ns에 리셋 버튼 다시 눌림*  *#100 rst = 1'b1; // 400ns에 리셋 해제*  *#300 rst = 1'b0; // 700ns에 리셋 버튼 다시 눌림*  *#50 rst = 1'b1; // 750ns에 리셋 해제*  *end*  *initial begin*  *#1000*  *$finish;*  *end*  *endmodule* |

텍스트, 소프트웨어, 라인, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

도표, 평면도, 기술 도면, 텍스트이(가) 표시된 사진

자동 생성된 설명

2비트 카운터는 클럭 신호(clk)와 리셋 신호(rst)를 입력으로 받고, 2비트 출력(out)을 제공한다. initial 블록에서 카운터 출력 out을 초기화한다. 여기서 out의 초기값은 2'b00이다. always 블록은 클럭 신호의 상승 엣지(posedge)에서 실행된다. 리셋 신호 rst가 활성화되지 않았을 때(!rst), 출력 out은 2'b00으로 초기화된다. 리셋이 활성화된 상태(rst = 1)에서는 out 값을 1씩 증가시킨다. 카운터는 2비트 출력으로, 값이 3(2'b11)에서 다시 0(2'b00)으로 순환한다. 시뮬레이션 코드와 결과를 살펴보면 다음과 같다. 초기값으로 clk는 0으로 설정되며, 이후 #50 ns마다 신호가 반전(~clk)된다. 초기값으로 rst는 0으로 설정된다(리셋 상태). 초기 상태에는 리셋 신호(rst = 0)로 인해 카운터 출력 out은 항상 초기화 상태인 2'b00을 유지한다. 100ns에서 리셋이 해제되고(rst = 1), 이후 클럭 상승 엣지마다 카운터 값이 1씩 증가한다. (150ns: out = 2'b01, 250ns: out = 2'b10, 350ns: out = 2'b11) 이후에도 마찬가지로 리셋이 활성화 되는 경우 out을 1로 유지하고, 리셋이 해제되는 경우 클럭 상승 엣지마다 카운터 값이 1씩 증가하는 과정을 반복한다. 결론적으로, 리셋이 활성화되면 카운터 출력은 2'b00으로 초기화된다. 리셋이 비활성화된 상태에서 클럭 상승 엣지마다 카운터는 1씩 증가한다. 2비트 출력 특성으로 인해 카운터는 2'b00부터 2'b11까지 순환한다. 리셋 신호의 활성화와 비활성화 타이밍에 따라 출력 변화가 제어된다.

**2. 4-bit decade counter**

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input clk, rst,*  *output reg [3:0] out*  *);*  *// 초기 상태*  *initial out = 4'b0000;*    *// 상승 엣지에서 동작*  *always @(posedge clk) begin*  *if (!rst) begin*  *out <= 4'b0000; // 리셋 시 출력 초기화*  *end else if (out == 4'b1001) begin*  *out <= 4'b0000; // 9에서 0으로 초기화*  *end else begin*  *out <= out + 1; // 카운터 증가*  *end*  *end*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg clk, rst;*  *wire [3:0] out;*  *boolean u\_boolean(*  *.clk(clk),*  *.rst(rst),*  *.out(out)*  *);*  *initial begin*  *clk = 1'b0;*  *forever #50 clk = ~clk;*  *end*  *initial begin*  *rst = 1'b0;*  *#100 rst = 1'b1; // 리셋 해제 (100ns 후)*  *end*  *initial begin*  *#1500*  *$finish;*  *end*  *endmodule* |

스크린샷, 텍스트, 라인, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

도표, 텍스트, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명

4비트 Decade Counter(10진수 카운터)는 클럭 신호(clk)와 리셋 신호(rst)를 입력으로 받고, 4비트 출력(out)을 생성한다. Decade Counter는 값이 9(4'b1001)에서 다시 0(4'b0000)으로 초기화되며 0~9 사이에서 순환한다. 먼저 initial 블록에서 카운터 출력 out은 4'b0000으로 초기화된다. always 블록은 클럭 신호의 상승 엣지에서 동작한다. 리셋 신호(rst)가 비활성화된 상태(!rst)에서는 카운터가 4'b0000으로 초기화된다. 리셋이 비활성화되지 않았고 출력 out이 4'b1001(9)일 때, 카운터는 다시 4'b0000으로 초기화된다. 그렇지 않은 경우, 출력 out은 1씩 증가한다. 이 과정에서 카운터는 0~9 사이에서 순환하며 동작한다. 시뮬레이션 코드와 결과를 살펴보면 다음과 같다. 초기값으로 clk는 1'b0으로 설정되며, 이후 50ns마다 신호가 반전(~clk)된다. 초기값으로 rst는 0(리셋 상태)로 설정된다. 100ns 후 리셋 신호가 활성화(rst = 1)되어 카운터 동작이 시작된다. 이후 클럭 상승 엣지마다 카운터 값이 1씩 증가한다. (150ns: out = 4'b0001, 250ns: out = 4'b0010, 350ns: out = 4'b0011, … , 950ns: out = 4'b1001, 1050ns: out = 4'b0000) 카운터는 9(4'b1001)에 도달하면 다시 0(4'b0000)으로 초기화된다. 결론적으로, 리셋 신호(rst)가 비활성화되면 카운터는 초기화 상태를 유지하며 동작하지 않는다. 리셋 신호가 활성화되면 클럭 상승 엣지마다 카운터 값이 1씩 증가하며, 값이 9에 도달하면 다시 0으로 초기화된다.

**3. 4-bit 2421 decade counter**

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input clk, rst,*  *output reg [3:0] out*  *);*  *// 초기 상태*  *initial out = 4'b0000;*    *// 상승 엣지에서 동작*  *always @(posedge clk) begin*  *if (!rst) begin*  *out <= 4'b0000; // 리셋 시 출력 초기화*  *end else if (out == 4) begin*  *out = 4'b1011;*  *end else begin*  *out = out + 1;*  *end*  *end*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg clk, rst;*  *wire [3:0] out;*  *boolean u\_boolean(*  *.clk(clk),*  *.rst(rst),*  *.out(out)*  *);*  *initial begin*  *clk = 1'b0;*  *forever #50 clk = ~clk;*  *end*  *initial begin*  *rst = 1'b0;*  *#100 rst = 1'b1; // 리셋 해제 (100ns 후)*  *end*  *initial begin*  *#1500*  *$finish;*  *end*  *endmodule* |

**스크린샷, 소프트웨어, 멀티미디어 소프트웨어, 텍스트이(가) 표시된 사진

자동 생성된 설명**

**도표, 텍스트, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명**

2421 코드는 BCD(Binary Coded Decimal)의 변형으로, 숫자를 4비트로 표현할 때 각 비트에 2, 4, 2, 1의 가중치를 부여하는 방식을 사용한다. 일반적인 8421 BCD 코드와 달리, 2421 코드에서는 10진수 5부터의 표현이 달라지며, 이를 반영한 동작을 설계하였다. 따라서 이 회로는 0부터 9까지의 값을 2421 코드로 변환하여 출력한다. 먼저 initial 블록에서 카운터 출력 out은 4'b0000으로 초기화된다. always 블록은 클럭 신호의 상승 엣지에서 동작한다. 리셋 신호(rst)가 비활성화된 상태(!rst)에서는 출력 out이 초기화된 상태(4'b0000)로 유지된다. 리셋 신호가 활성화된 상태에서, 클럭 신호의 상승 엣지마다 카운터가 동작한다. 출력 값이 10진수 4에 도달하면, 일반적인 8421 표현이 아닌 2421 코드로 변환된 값인 4'b1011(10진수 5에 해당)을 출력한다. 이후 출력은 2421 코드 규칙에 따라 계속 증가한다. 시뮬레이션 코드와 결과를 살펴보면 다음과 같다. 초기값으로 clk는 1'b0으로 설정되며, 이후 50ns마다 신호가 반전된다. 초기값으로 rst는 0(리셋 상태)로 설정된다. 100ns 후 리셋 신호가 활성화(rst = 1)되며 카운터가 동작을 시작한다. (150ns: out = 4'b0001, 250ns: out = 4'b0010, 350ns: out = 4'b0011, 450ns: out = 4'b0100, 550ns: out = 4'b1011, … ) 출력 값은 2421 코드 규칙에 맞게 증가하며, 10진수 5부터의 변환 과정이 반영된다. 결론적으로 2421 코드에서 10진수 0~4는 일반적인 8421 코드와 동일하게 표현되고, 10진수 5부터는 2421 코드 변환 규칙에 따라 출력이 설정되며, 간단한 조건문(else if (out == 4))을 통해 구현하였다. 리셋 신호가 비활성화된 경우 출력 값은 초기화 상태(4'b0000)를 유지한다. 리셋 신호가 활성화된 후 카운터는 클럭 신호에 따라 증가하며, 0~9의 값을 2421 코드로 표현한다.

**4. 결과 검토/논의 사항**

2-bit Counter는 기본적인 상승 엣지 동작을 구현하여 2비트로 표현할 수 있는 범위(0~3)를 순차적으로 출력하는 설계였다. 리셋 신호(rst)가 비활성화되면 출력 값은 초기화 상태(0)로 유지되었고, 리셋이 활성화되면 카운터는 정상적으로 작동하며 출력 값이 0 → 1 → 2 → 3으로 순차적으로 증가하였다. 이 카운터의 장점은 간단한 구조로 동작하며, 기본적인 카운터의 동작 원리를 이해하는 데 적합하였다. 하지만 단점은 비트 수가 고정되어 있어 범위를 초과하는 값은 표현할 수 없다.

4-bit Decade Counter는 10진수 범위(0~9)를 반복적으로 출력하도록 설계되었다. 리셋 신호가 비활성화되면 출력 값은 0으로 초기화되었고, 리셋이 활성화되면 0에서 시작하여 9까지 순차적으로 증가한 후 0으로 돌아가는 동작을 확인하였다. 이 카운터의 장점은 추가적인 조건문으로 값의 범위를 제한하며, 실질적으로 사용 가능한 10진수 카운터로 확장되었다는 점이다.

4-bit 2421 Decade Counter는 일반적인 4-bit Decade Counter와 달리, 2421 코드 규칙을 사용하여 10진수를 표현하였다. 10진수 0~4는 일반적인 8421 BCD 표현과 동일했으나, 10진수 5부터는 2421 코드 규칙에 따라 출력 값이 변경되었다(예: 10진수 5는 1011, 6은 1100 등). 리셋 신호가 비활성화되면 출력 값은 0으로 초기화되었고, 리셋이 활성화되면 출력 값이 0~9를 2421 코드로 표현하며 순차적으로 증가하였다. 이 카운터의 장점은 2421 코드와 같은 비표준 표현 방식의 구현을 통해 코딩 논리를 확장할 수 있다는 점이다. 하지만 단점은 2421 코드 규칙을 위한 조건문이 추가되어 구현 복잡도가 증가한다.

카운터의 응용 가능성에 대해 논의해 볼 수 있다. 2-bit Counter는 기본적인 카운터 동작을 학습하고, 더 복잡한 카운터를 설계하기 위한 기초로 사용하기 적합하다. 4-bit Decade Counter는 디지털 시계, 키패드 입력 처리 등 10진수 기반의 응용에 활용 가능하다. 4-bit 2421 Decade Counter는 2421 코드 변환을 학습하는 데 적합하며, 특수한 데이터 표현 방식을 다룰 때 활용할 수 있다.

**5. 추가 이론**

조건문과 상태 머신에 대해 추가적으로 알아볼 수 있다. 2421 Decade Counter처럼 상태 전환 규칙이 복잡해지면, 단순한 조건문으로 구현하기 어려울 수 있다. 이 경우 FSM (Finite State Machine)을 사용하여 상태와 전이 조건을 체계적으로 설계할 수 있다. FSM은 세가지 요소로 구성된다. 상태(state): 현재 카운터가 나타내는 값(예: 10진수 0~9), 전이(transition): 특정 입력(클럭, 리셋 등)에 따라 상태가 변화하는 규칙, 출력(output): 현재 상태에 따라 생성되는 출력 값. 이러한 FSM의 장점은 복잡한 상태 전이를 명확히 정의하여 가독성과 유지보수성을 높일 수 있다는 점이다.