13주차 결과보고서

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

**1. 4-bit shift register**

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input clk, rst, in,*  *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 begin*  *out = out >> 1;*  *out[3] = in;*  *end*  *end*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg clk, rst, in;*  *wire [3:0] out;*  *boolean u\_boolean(*  *.clk(clk),*  *.rst(rst),*  *.in(in),*  *.out(out)*  *);*  *initial begin*  *clk = 1'b0;*  *forever #50 clk = ~clk;*  *end*  *initial begin*  *rst = 1'b1; // 초기 상태는 리셋 해제*  *in = 1'b0; // 초기 입력 값은 0*    *#100 rst = 1'b0; // 100ns 후 리셋 활성화 (active-low)*  *#50 rst = 1'b1; // 150ns에 리셋 해제*  *#50 in = 1'b1; // 200ns에 입력을 1로 변경*  *#200 rst = 1'b0; // 400ns에 리셋 활성화*  *#50 rst = 1'b1; // 450ns에 리셋 해제*  *#100 in = 1'b0; // 550ns에 입력을 다시 0으로 변경*  *#150 rst = 1'b0; // 700ns에 리셋 활성화*  *#50 rst = 1'b1; // 750ns에 리셋 해제*  *#50 in = 1'b1; // 800ns에 입력을 1로 변경*  *#100 in = 1'b0; // 900ns에 입력을 다시 0으로 변경t*  *end*  *initial begin*  *#1000*  *$finish;*  *end*  *endmodule* |

스크린샷, 소프트웨어, 멀티미디어 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명

4-bit Shift Register는 클록의 상승 엣지마다 데이터를 한 비트씩 오른쪽으로 이동시키며, 새로운 데이터는 입력 신호 'in'을 통해 레지스터의 가장 왼쪽 비트(out[3])로 직접 로드된다. 이 구현에서는 리셋 신호(rst)가 활성화되어 있을 때 레지스터가 4'b0000으로 초기화되며, 리셋 신호가 비활성화되면 입력된 데이터에 따라 비트가 오른쪽으로 이동한다.

시뮬레이션 과정을 상세히 살펴보면, 초기 상태에서 'out'은 4'b0000으로 설정되어 있다. 시뮬레이션 시작 시 리셋 신호는 먼저 활성화되어 'out'을 초기화하고, 이후 리셋이 해제된다. 200ns에 'in' 입력이 1로 변경되고, 다음 클록 펄스에 이 값이 'out[3]'으로 이동하며, 나머지 비트들은 오른쪽으로 이동한다. 이 과정은 리셋 신호가 여러 번 활성화될 때마다 반복되며, 각 활성화 시 'out'은 4'b0000으로 초기화된다.

시뮬레이션 결과를 통해, 리셋 신호와 입력값의 변화에 따라 'out' 레지스터의 비트들이 올바르게 이동하고 새로운 값이 적절히 로드되는 것을 확인할 수 있다. 입력값의 변경과 리셋의 활성화는 'out'의 상태에 직접적인 영향을 미치며, 이는 레지스터가 설계 의도와 동일하게 동작함을 보여준다.

**2. 4-bit ring counter**

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input clk, rst,*  *output reg [3:0] out*  *);*  *// 초기 상태*  *initial out = 4'b1000;*    *// 상승 엣지에서 동작*  *always @(posedge clk) begin*  *if (!rst) begin*  *out <= 4'b1000; // 리셋 시 출력 초기화*  *end else begin*  *out[0] <= out[1];*  *out[1] <= out[2];*  *out[2] <= out[3];*  *out[3] <= out[0];*  *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'b1; // 초기 상태는 리셋 해제*    *#100 rst = 1'b0; // 100ns에 리셋 활성화*  *#50 rst = 1'b1; // 150ns에 리셋 비활성화*    *#300 rst = 1'b0; // 450ns에 리셋 활성화*  *#50 rst = 1'b1; // 500ns에 리셋 비활성화*    *#200 rst = 1'b0; // 700ns에 리셋 활성화*  *#50 rst = 1'b1; // 750ns에 리셋 비활성화*    *#200 rst = 1'b0; // 950ns에 리셋 활성화*  *#50 rst = 1'b1; // 1000ns에 리셋 비활성화*  *end*  *initial begin*  *#1000*  *$finish;*  *end*  *endmodule* |

스크린샷, 멀티미디어 소프트웨어, 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명

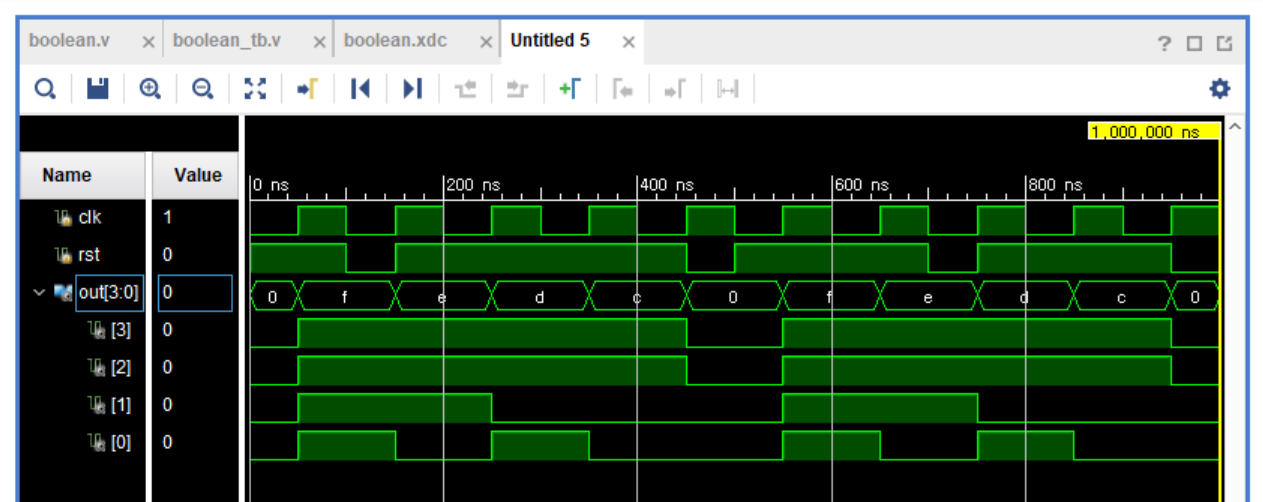
4-bit Ring Counter는 시계 신호의 각 상승 엣지에서 내부적으로 저장된 값을 순환시키는 구조이다. 초기화 상태에서는 'out'이 4'b1000으로 설정되어 있어, 가장 높은 비트만 활성화되어 있다. 이 후 클록의 각 상승 엣지마다 'out'의 비트들이 오른쪽으로 순환되며, 가장 낮은 비트(out[0])의 값이 가장 높은 비트(out[3])로 이동한다.

리셋 신호가 활성화될 때마다 카운터는 4'b1000으로 초기화된다. 이는 리셋 신호가 0으로 활성화되고, 1로 다시 비활성화될 때 발생한다. 예를 들어, 테스트 벤치에서는 100ns, 450ns, 700ns 및 950ns에 리셋이 활성화되어 각각 그 직후 150ns, 500ns, 750ns, 1000ns에 비활성화되어 순환 패턴이 다시 시작된다.

시뮬레이션 동안, 'out'은 4'b1000에서 시작하여 첫 번째 클록 후에는 4'b0001이 되고, 이후 순차적으로 4'b0010, 4'b0100과 같은 패턴으로 진행된다. 이와 같은 순환적인 비트 이동은 Ring Counter의 전형적인 동작을 나타내며, 데이터의 순환 저장과 반환에 이용될 수 있다. 각 리셋 신호 후에 'out'은 다시 4'b1000으로 초기화되어 순환 패턴이 재설정된다.

**3. 4-bit up/down counter**

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input clk, rst, state,*  *output reg [3:0] out,*  *output reg [6:0] seg,*  *output reg digit*  *);*  *// 초기 상태*  *initial out <= 4'b0000;*  *initial digit = 1;*    *// 상승 엣지에서 동작*  *always @(posedge clk) begin*  *if (!rst) begin*  *out <= 4'b0000; // 리셋 시 출력 초기화*  *end else begin*  *if(state == 1'b1) begin //up mode*  *if (out == 4'b1111) begin*  *out <= 4'b0000;*  *end*  *else begin*  *out <= out + 4'b0001;*  *end*  *seg <= 7'b0111110;*  *end*  *else begin*  *if (out == 4'b0000) begin //down mode*  *out <= 4'b1111;*  *end*  *else begin*  *out <= out - 4'b0001;*  *end*  *seg <= 7'b0111101;*  *end*  *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'b1; // 초기 상태는 리셋 해제*    *#100 rst = 1'b0; // 100ns에 리셋 활성화*  *#50 rst = 1'b1; // 150ns에 리셋 비활성화*    *#300 rst = 1'b0; // 450ns에 리셋 활성화*  *#50 rst = 1'b1; // 500ns에 리셋 비활성화*    *#200 rst = 1'b0; // 700ns에 리셋 활성화*  *#50 rst = 1'b1; // 750ns에 리셋 비활성화*    *#200 rst = 1'b0; // 950ns에 리셋 활성화*  *#50 rst = 1'b1; // 1000ns에 리셋 비활성화*  *end*  *initial begin*  *#1000*  *$finish;*  *end*  *endmodule* |

****

4-bit Up/Down Counter는 디지털 시스템에서 중요한 역할을 수행하는 구성 요소이다. 이 카운터는 클록 신호의 상승 엣지에 반응하여 카운트 값을 증가시키거나 감소시킨다. 리셋 신호가 활성화될 때마다 카운터는 0000으로 초기화된다. 카운터의 동작 모드는 state 입력 신호에 의해 결정된다. state가 1일 경우, 카운터는 증가 모드로 동작하며 0001부터 1111까지 순차적으로 증가한 후 다시 0000으로 롤오버한다. state가 0일 경우, 카운터는 감소 모드로 작동하여 0000에서 1111으로 감소하며, 0000에 도달하면 1111로 롤오버한다.

세그먼트 디스플레이(seg)는 카운터의 상태를 시각적으로 나타내기 위해 사용된다. 증가 모드에서는 0111110의 패턴을, 감소 모드에서는 0111101의 패턴을 표시하여 현재 모드를 나타낸다. 이러한 세그먼트 디스플레이는 사용자가 카운터의 현재 상태를 쉽게 인식할 수 있도록 돕는다.

테스트 벤치는 클록 신호를 반복적으로 토글하여 시뮬레이션 환경에서 카운터의 반응을 테스트한다. 리셋 신호는 시뮬레이션 도중 여러 번 활성화되어 카운터가 각 상태 전환 후 제대로 초기화되는지 확인한다. 시뮬레이션 과정은 카운터의 동작이 예상대로 진행되는지 검증하기 위해 설계되었으며, 각 상태에서 카운터가 적절하게 증가하거나 감소하는지를 확인한다.

**4. 결과 검토/논의 사항**

4-bit Shift Register는 입력 신호를 순차적으로 한 비트씩 오른쪽으로 이동시키는 기능을 수행한다. 클록 신호의 매 상승 엣지마다 데이터는 한 위치씩 이동하며, 왼쪽 끝 비트는 외부에서 주어진 입력 신호(in)로 채워진다. 이 회로는 데이터의 시리얼 전송이나 일시적인 데이터 저장에 활용될 수 있다. 시뮬레이션 결과는 리셋이 정상적으로 작동하여 회로가 초기화될 때마다 모든 비트가 0으로 설정됨을 보여준다. 이는 회로가 예상대로 리셋과 데이터 이동 기능을 정확히 수행함을 확인시켜 준다.

4-bit Ring Counter는 단일 '1'이 순환하며 나머지는 모두 '0'인 특수한 형태의 쉬프트 레지스터이다. 이 구조는 클록의 각 상승 엣지에서 '1'이 순차적으로 이동한다. 리셋 신호가 활성화될 때마다 카운터는 초기 상태 1000으로 설정된다. Ring Counter는 주로 시퀀스 생성에 사용되며, 시뮬레이션 결과는 카운터가 정상적으로 순환하는 것을 보여준다. 이는 순차적 로직 회로에서의 정확한 타이밍과 위치 추적에 유용하다.

4-bit Up/Down Counter는 사용자가 설정한 모드에 따라 카운트 값을 증가시키거나 감소시키는 기능을 한다. state 신호에 따라 카운터는 증가 모드 또는 감소 모드로 작동한다. 최대값(1111)에 도달한 후에는 최소값(0000)으로 롤오버하며, 그 반대의 경우도 마찬가지이다. 시뮬레이션 결과는 카운터가 상태 전환을 정확히 수행하고, 리셋 신호에 반응하여 적절히 초기화되는 것을 보여준다. 이 카운터는 복잡한 디지털 시스템에서 유연한 카운팅이 필요할 때 유용하다.

논의사항으로는, 각 카운터와 레지스터의 구현이 기대한 동작을 정확히 수행하는지 확인하는 것 외에도, 실제 하드웨어 환경에서의 성능과 타이밍 문제를 고려할 필요가 있다. 특히, 실제 응용에서는 전력 소비와 처리 속도도 중요한 고려사항이다. 따라서 설계를 최적화하여 하드웨어 자원을 효율적으로 사용하고, 필요에 따라 회로의 복잡성을 조절하는 것이 중요하다.

**5. 추가 이론**

파이프라이닝(Pipelining): Shift Register는 파이프라이닝 구조의 기본 요소이다. 파이프라이닝은 복잡한 연산을 여러 단계로 나누어 각 단계를 등시간 간격으로 처리함으로써 전체 처리 속도를 향상시키는 기법이다. 이 구조에서 Shift Register는 각 단계의 결과를 임시로 저장하고 다음 클록 사이클에서 다음 단계로 데이터를 전달하는 역할을 한다. 따라서, 더 많은 데이터 처리 요구 사항을 충족시킬 수 있는 병렬 처리 시스템의 구현이 가능하다.

모드 제어 로직(Mode Control Logic): Up/Down Counter의 동작 방식은 모드 제어 로직을 통해 더욱 다양화될 수 있다. 이 로직을 통해 카운터는 프로그래밍 가능한 방식으로 증가 또는 감소 기능을 수행하며, 사용자 입력에 따라 다이나믹하게 동작 모드를 전환할 수 있다. 이 기능은 사용자가 시스템 요구에 따라 카운터의 동작을 유연하게 변경할 수 있게 하여, 더욱 복잡한 디지털 시스템에서 요구하는 맞춤형 기능을 제공한다.