12주차 결과보고서

전공: 아트&테크놀로지학과 학년: 4학년 학번: 20191048 이름: 김도솔

**1.**

**2-bit counter**

* Verilog source code

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `timescale 1ns / 1ps  module binary\_counter(  clk, rst, out  );  input clk, rst;  output[1:0] out;  reg[1:0] out;  initial out = 2'b00;  always @(posedge clk) begin  if(!rst)  out <= 0;  else  out <= out + 1;  end  endmodule | `timescale 1ns / 1ps  module binary\_counter\_tb;  reg clk, rst;  wire[1:0] out;  binary\_counter u\_binary\_counter (  .clk (clk ),  .rst (rst ),  .out (out )  );  always #10 clk = ~clk;  initial begin  clk = 1'b0;  rst = 1'b0;  #20 rst <= 1'b1;  #80 rst <= 1'b0;  #50 rst <= 1'b1;  end  initial begin  #1000  $finish;  end  endmodule |

위는 Verilog 코드로 2-bit binary counter를 구현한 것이다. binary\_counter 모듈은 2개의 입력(clk, rst)과 1개의 출력(out)을 가지고 있다. 이때 clk는 클럭 신호, rst는 리셋 신호, out는 2비트의 출력 값을 의미한다. 초기에 out은 '00'으로 설정된다. 클럭의 상승 에지에서 리셋 신호가 활성화되면(rst가 0이면), out은 0으로 초기화된다. 그렇지 않으면 out은 현재 값에 1을 더한다. 이렇게 하면 카운터의 출력은 클럭의 상승 에지마다 1씩 증가하며, 리셋이 발생하면 0으로 초기화된다.

* Schematic Diagram

도표, 평면도, 텍스트, 기술 도면이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 알 수 있다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 두 입력 clk, rst의 값을 바꾸면 그에 따라 2 bit 출력 out의 값이 변화하는 것을 확인할 수 있다. 구현한 2-bit counter가 각 입력 값에 따라 어떻게 동작하는지 아래 출력 예시들을 통해 좀 더 자세히 살펴보겠다.

* 출력 예시

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk와 rst의 초기 값은 둘다 0이다. out도 design source code에서 초기 값을 00으로 설정했으므로 00을 출력한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고 rst가 0이다. 리셋 신호가 활성화되어 있으므로 out은 00을 출력한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 0이고 rst가 1이다. clk 신호가 0이므로 counter는 상태 값을 업데이트 하지 않는다. 따라서 이전 출력 값인 00을 유지한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고 rst가 1이다. 즉 clk 신호가 1로 활성화 되어 있고 리셋 신호는 비활성화 되어 있다. 따라서 counter가 클럭 신호의 상승 에지에서 출력 값인 out을 01로 업데이트하는 것을 관찰할 수 있다.

스크린샷이(가) 표시된 사진

자동 생성된 설명> 이후 리셋 신호는 계속해서 1을 유지하기에 매 클럭 주기마다 counter의 출력 값은 1씩 증가한다. 위 예시에서 최대 값인 3(11)을 넘어가면 다시 00을 출력하는 것을 알 수 있다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 0이므로 카운터는 이전 출력 값을 유지한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고, rst가 0이다. 클럭 신호와 리셋 신호가 둘다 활성화 되어 카운터는 00을 출력 한다. 이후 rst가 0인 동안 계속 출력 값을 00으로 유지한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고 rst가 1이다. 즉 클럭 신호가 활성화 되었고, 리셋 신호가 비활성화 되었다. 따라서 카운터는 00부터 다시 카운팅을 시작한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> 리셋 신호가 비활성화 되어 있는 동안 카운터는 계속해서 클럭의 상승 에지마다 출력 값을 1씩 증가시킨다. 최대 값인 11(3)에 도달하면 다음 클럭 주기에 다시 00을 출력하는 것을 확인할 수 있다. 이후 리셋 신호가 비활성화 된 동안 다시 00(0)에서 11(3)까지 카운팅을 반복한다.

**2.**

**4-bit decade counter**

* Verilog source code

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `timescale 1ns / 1ps  module decade\_counter(  clk, rst, out  );  input clk, rst;  output[3:0] out;  reg[3:0] out;  initial out = 4'b0000;  always @(posedge clk) begin  if(!rst)  out <= 4'b0000;  else if(out >= 4'b1001)  out <= 4'b0000;  else  out <= out + 1;  end  endmodule | `timescale 1ns / 1ps  module decade\_counter\_tb;  reg clk, rst;  wire[3:0] out;  decade\_counter u\_decade\_counter (  .clk (clk ),  .rst (rst ),  .out (out )  );  always #10 clk = ~clk;  initial begin  clk = 1'b0;  rst = 1'b0;  #20 rst <= 1'b1;  #80 rst <= 1'b0;  #50 rst <= 1'b1;  end  initial begin  #1000  $finish;  end  endmodule |

위는 Verilog 코드로 4-bit decade counter를 구현한 것이다. decade\_counter 모듈은 2개의 입력(clk, rst)과 1개의 출력(out)을 가지고 있다. 이때 clk는 클럭 신호, rst는 리셋 신호, out는 4비트의 출력 값을 의미한다. 초기에 out은 ‘0000’으로 설정된다. 클럭의 상승 에지에서 리셋 신호가 활성화되면(rst가 0이면), out은 ‘0000’으로 초기화된다. 그렇지 않고 만약 out이 4비트 이진수 ‘1001’ 이상이면, 다시 ‘0000’으로 초기화한다. 위의 두 경우 모두 해당되지 않으면 out은 현재 값에 1을 더한다. 이렇게 하면 카운터의 출력은 클럭의 상승 에지마다 1씩 증가하며 최대 값 10001을 초과하면 0000으로 초기화되고, 또한 리셋이 발생하면 0000으로 초기화된다.

* Schematic Diagram

텍스트, 도표, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 알 수 있다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 두 입력 clk, rst의 값을 바꾸면 그에 따라 4 bit 출력 out의 값이 변화하는 것을 확인할 수 있다. 구현한 4-bit decade counter가 각 입력 값에 따라 어떻게 동작하는지 아래 출력 예시들을 통해 좀 더 자세히 살펴보겠다.

* 출력 예시

스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

> clk와 rst의 초기 값은 둘다 0이다. out도 design source code에서 초기 값을 0000으로 설정했으므로 0000을 출력한다.

스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고 rst가 0이다. 리셋 신호가 활성화되어 있으므로 out은 0000을 출력한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 0이고 rst가 1이다. clk 신호가 0이므로 counter는 상태 값을 업데이트 하지 않는다. 따라서 이전 출력 값인 0000을 유지한다.

스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고 rst가 1이다. 즉 clk 신호가 1로 활성화 되어 있고 리셋 신호는 비활성화 되어 있다. 따라서 counter가 클럭 신호의 상승 에지에서 출력 값인 out을 0001(1)로 업데이트하는 것을 관찰할 수 있다.

스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

> 이후 리셋 신호는 계속해서 1을 유지하기에 매 클럭 주기마다 counter의 출력 값은 1씩 증가한다. 위 예시는 출력 값이 0100(4)까지 증가한 상태를 보여준다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 0이므로 카운터는 이전 출력 값을 유지한다.

스크린샷, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고, rst가 0이다. 클럭 신호와 리셋 신호가 둘다 활성화 되어 카운터는 0000을 출력 한다. 이후 rst가 0인 동안 계속 출력 값을 0000으로 유지한다.

스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고 rst가 1이다. 즉 클럭 신호가 활성화 되었고, 리셋 신호가 비활성화 되었다. 따라서 카운터는 0000부터 다시 카운팅을 시작한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> 리셋 신호가 비활성화 되어 있는 동안 카운터는 계속해서 클럭의 상승 에지마다 출력 값을 1씩 증가시킨다. 최대 값인 1001(9)에 도달하면 다음 클럭 주기에 0000을 출력하는 것을 확인할 수 있다. 이후 리셋 신호가 비활성화 된 동안 다시 0000(0)에서 1001(9)까지 카운팅을 반복한다.

**3.**

**4-bit 2421 decade counter**

* Verilog source code

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `timescale 1ns / 1ps  module decade\_counter(  clk, rst, out  );  input clk, rst;  output[3:0] out;  reg[3:0] out;  initial out = 4'b0000;  always @(posedge clk) begin  if(!rst)  out <= 4'b0000;  else if(out >= 4'b1001)  out <= 4'b0000;  else  out <= out + 1;  end  endmodule  module bcd\_to\_2421(  in, out  );  input[3:0] in;  output[3:0] out;  assign out[3] = in[3]|(in[2]&in[0])|(in[2]&in[1]);  assign out[2] = in[3]|(in[2]&in[1])|(in[2]&~in[0]);  assign out[1] = (in[2]&~in[1]&in[0])|in[3]|(~in[2]&in[1]);  assign out[0] = in[0];  endmodule  module decade\_counter\_2421(  clk, rst, out  );  input clk, rst;  output[3:0] out;  wire[3:0] conv;  decade\_counter dc(  .clk(clk),  .rst(rst),  .out(conv)  );  bcd\_to\_2421 bt2(  .in(conv),  .out(out)  );  endmodule | `timescale 1ns / 1ps  module decade\_counter\_2421\_tb;  reg clk, rst;  wire[3:0] out;  decade\_counter\_2421 u\_decade\_counter\_2421 (  .clk (clk ),  .rst (rst ),  .out (out )  );  always #10 clk = ~clk;  initial begin  clk = 1'b0;  rst = 1'b0;  #20 rst <= 1'b1;  #80 rst <= 1'b0;  #50 rst <= 1'b1;  end  initial begin  #1000  $finish;  end  endmodule |

위는 Verilog 코드로 4-bit 2421 decade counter를 구현한 것이다. decade\_counter \_2421 모듈은 이전에 구현한 decade\_counter 모듈과 새로 구현한 bcd\_to\_2421 모듈을 연결하여 4비트 decade 카운터의 출력을 4비트 2421 decade로 변환한다.

decade\_counter 모듈은 이전에 구현한 것을 그대로 가져온 것이라 설명을 생략하고, bcd\_to\_2421 모듈은 4비트 BCD 8421 코드를 4비트 BCD 2421 코드로 변환하는 모듈이다. in은 4비트 8421 입력, out은 4비트 2421 출력을 나타낸다. 출력의 각 자릿수마다 BCD 8421 코드를 2421 코드로 변환하는 논리식들이 사용되었다. decade\_counter\_2421 모듈은 2개의 입력(clk, rst)과 1개의 출력(out)을 가지고 있다. 이때 clk는 클럭 신호, rst는 리셋 신호, out는 4비트의 출력 값을 의미한다. clk와 rst는 decade\_counter로 전달되고, out은 bcd\_to\_2421의 출력으로 연결된다. 내부적으로 decade\_counter가 4비트 카운터를 생성하고, 그 출력이 bcd\_to\_2421의 입력으로 사용된다. bcd\_to\_2421 모듈을 통해 변환된 출력은 decade\_counter\_2421의 최종 출력인 out으로 사용된다.

* Schematic Diagram

도표, 스크린샷, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 알 수 있다.

* Simulation 결과

스크린샷, 텍스트, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 두 입력 clk, rst의 값을 바꾸면 그에 따라 4 bit 출력 out의 값이 변화하는 것을 확인할 수 있다. 구현한 4-bit 2421 decade counter가 각 입력 값에 따라 어떻게 동작하는지 아래 출력 예시들을 통해 좀 더 자세히 살펴보겠다.

* 출력 예시

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk와 rst의 초기 값은 둘다 0이다. out도 design source code에서 초기 값을 0000으로 설정했으므로 0000을 출력한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고 rst가 0이다. 리셋 신호가 활성화되어 있으므로 out은 0000을 출력한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 0이고 rst가 1이다. clk 신호가 0이므로 counter는 상태 값을 업데이트 하지 않는다. 따라서 이전 출력 값인 0000을 유지한다.

스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고 rst가 1이다. 즉 clk 신호가 1로 활성화 되어 있고 리셋 신호는 비활성화 되어 있다. 따라서 counter가 클럭 신호의 상승 에지에서 출력 값인 out을 0001(1)로 업데이트하는 것을 관찰할 수 있다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> 이후 리셋 신호는 계속해서 1을 유지하기에 매 클럭 주기마다 counter의 출력 값은 1씩 증가한다. 위 예시는 출력 값이 0100(4)까지 증가한 상태를 보여준다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 0이므로 카운터는 이전 출력 값을 유지한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고, rst가 0이다. 클럭 신호와 리셋 신호가 둘다 활성화 되어 카운터는 0000을 출력 한다. 이후 rst가 0인 동안 계속 출력 값을 0000으로 유지한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고 rst가 1이다. 즉 클럭 신호가 활성화 되었고, 리셋 신호가 비활성화 되었다. 따라서 카운터는 0000부터 다시 카운팅을 시작한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 0이므로 카운터는 이전 출력 값을 유지한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> clk가 1이고, rst가 1이다. 따라서 카운터는 클럭의 상승 에지에서 출력 값을 0001로 업데이트한다. 리셋 신호가 비활성화 되어 있는 동안 카운터는 계속해서 클럭의 상승 에지마다 출력 값을 1씩 증가시킨다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> decade counter와 달리 bcd\_to\_2421 모듈에 의해 클럭의 상승 에지에 십진수 5를 2421 코드로 나타낸 1011을 출력하는 것을 관찰할 수 있다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> 마찬가지로 클럭의 상승 에지에 십진수 6을 2421 코드로 나타낸 1100을 출력한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> 클럭의 상승 에지에 십진수 7을 2421 코드로 나타낸 1101을 출력한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> 클럭의 상승 에지에 십진수 8을 2421 코드로 나타낸 1110을 출력한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> 클럭의 상승 에지에 십진수 9를 2421 코드로 나타낸 1111을 출력한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

> 최대 값인 1111(9)에 도달하면 다음 클럭 주기에 0000을 출력하는 것을 확인할 수 있다. 이후 리셋 신호가 비활성화 된 동안 다시 0000(0)에서 1111(9)까지 카운팅을 반복한다.

**4.**

2-bit binary counter, 4-bit decade counter, 4-bit 2421 decade counter의 Verilog 코드를 작성하고 Simulation 결과와 FPGA의 동작을 통해 회로가 원하는 대로 잘 구현되었는지 확인하는 과정을 거쳤다. 먼저 클럭 신호의 상승 에지에서 카운터의 상태 값이 변경된다는 점, 다음으로 상태 값이 설정된 최대 값에 도달했을 때(ex. 2-bit counter는 11, 4-bit decade counter는 1001, 4-bit 2421 decade counter는 1111) 다음 클럭 신호에 초기 값인 0으로 상태가 되돌아간다는 점, 마지막으로 rst 값이 0이 되면 상태 값을 0으로 초기화하여 새로운 카운팅을 시작한다는 점에서 구현된 카운터들이 올바르게 동작한다는 것을 확인할 수 있었다.

**5.**

이번 실험에서는 constraint 파일에 “set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets clk\_IBUF]”라는 명령어를 추가해 오류를 해결했다. 이 명령은 Xilinx Vivado의 Tcl 스크립트에서 사용되는 것으로, 클럭 라우팅과 관련된 속성을 설정하는 데 사용된다. 간단히 말하면, 이 명령은 특정 네트의 클럭 라우팅에 대한 제약을 해제하는 역할을 한다. 이 명령을 사용함으로써, Vivado가 더 자유롭게 최적의 라우팅을 수행할 수 있게 된다. 명령어의 각 부분은 다음과 같은 의미를 가진다.

* set\_property: Tcl에서 속성을 설정하는 명령어이다.
* CLOCK\_DEDICATED\_ROUTE: 클럭에 대한 라우팅 제약을 나타내는 속성이다.
* FALSE: 해당 제약을 해제하고, 라우팅이 자유롭게 이루어질 수 있도록 허용한다.
* [get\_nets clk\_IBUF]: clk\_IBUF 라는 이름의 넷을 찾아 그에 대한 속성을 설정한다. 이는 일반적으로 클럭 신호에 사용되는 버퍼(IBUF)에 해당된다.