전공: 컴퓨터공학 학년: 2학년 학번: 20161603 이름: 신민준

1. 2-bit counter의 결과 및 simulation 과정에 대해서 설명하시오.

2-bit binary counter는 4개의 state, 00, 01, 10, 11을 순차적으로 입력 pulse에 따라 바꾸어 가는 sequential circuit이다. 이 때, reset 입력을 추가적으로 구현해 reset 입력이 들어올 때 00의 state로 설정되도록 하고, 입력 pulse의 leading edge에서 값의 변화가 일어나도록 구현했다. 이 과정을 overflow bit 를 추가한 state diagram과 state table로 보이면 다음 그림과 같다.

|  |  |  |
| --- | --- | --- |
| **Present state** | **next state** | |
| **qi** | x = 0 | x = 1 |
| q0 | q0 | q1 |
| q1 | q1 | q2 |
| q2 | q2 | q3 |
| q3 | q3 | q0 |



이 때, 의 state를 사용한다.

보인 위의 자료드을 기반으로 구현한 2-bit counter의 Verilog code는 다음과 같았다.

|  |
| --- |
| `timescale 1ns / 1ps  module two\_bit\_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) begin  out <= 2'b00;  end  else begin  out <= out + 1;  end  end  endmodule |

위 코드에서 볼 수 있듯, clk 신호의 leading edge에서, rst 신호가 들어오지 않은 경우, 내부의 2-bit register인 out의 값에 1을 더하고, rst 신호가 들어온 경우에는 out 값을 00으로 초기화한다. 이 때, out 값이 11인 상황에서 clk 신호가 들어오면 자연스럽게 값이 overflow되어 00의 state로 돌아가게 된다.

작성한 코드의 회로도는 다음과 같은 다이어그램으로 나타났다.

스크린샷이(가) 표시된 사진

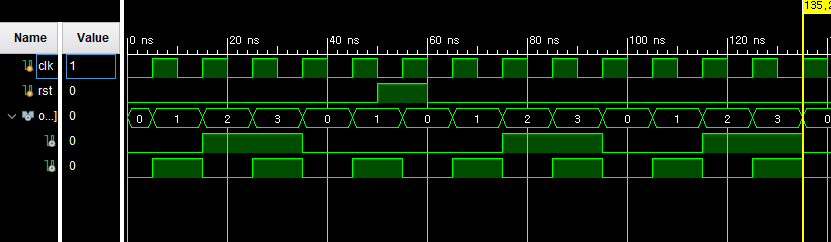
자동 생성된 설명

구현한 2-bit counter가 정상적으로 작동하는지 확인하기 위해, 다음과 같이 counter를 테스트 할 수 있는 simulation source를 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module twobit\_sim;  reg clk, rst;  wire[1:0] out;    two\_bit\_counter tbc(  .clk(clk),  .rst(rst),  .out(out)  );    initial clk = 0;  initial rst = 0;    always clk = #5 clk + 1;    initial begin  #50  assign rst = ~rst;  #10  assign rst = ~rst;  #1600  $finish;  end  endmodule |

위 코드에서는 10ns의 clock cycle을 가진 machine을 가정하고, 5개의 clock cycle을 돌린 이후 rst 값을 잠시 high로 두어 reset 기능이 정상작동 하는지 확인하고자 했다.

이 simulation source의 결과는 다음과 같았다.



Clk 값이 1로 변하는 leading edge에서 출력인 out 배열의 값이 00->01->10->11->00 ->… 순서대로 변하는것을 확인할 수 있고, 또한 50ns 타이밍에 rst 입력에 high 값을 넣었을 때 내부 register의 값이 00으로 reset되는 것도 볼 수 있다. 따라서, 구현하고자 한 2-bit counter의 두 가지의 기능들을 전부 다 올바르게 구현했다고 할 수 있다.

1. 4-bit decade counter의 결과 및 simulation 과정에 대해서 설명하시오.

4-bit counter는 기본적으로 개의 state를 사용할 수 있는데, 이 때 0~9의 state만을 사용하는 경우 decade counter가 완성된다. 이 decade counter의 state diagram과 state table은 다음과 같이 나타낼 수 있다.



위의 다이어그램에서 화살표는 입력이 1일 때를 나타낸다.

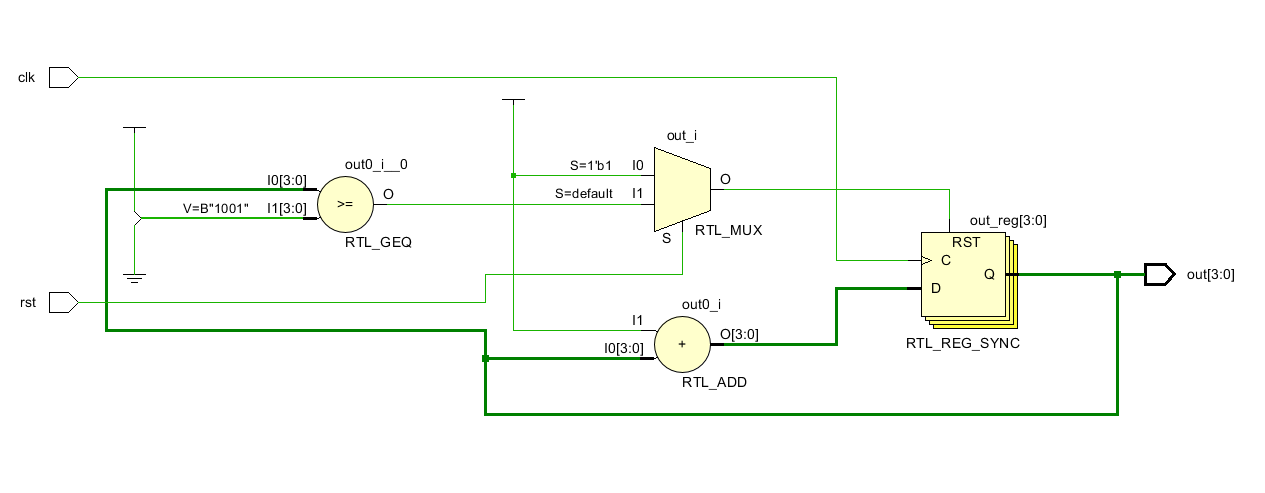
|  |  |  |
| --- | --- | --- |
| **Present state** | **next state** | |
| **qi** | x = 0 | x = 1 |
| q0 | q0 | q1 |
| q1 | q1 | q2 |
| q2 | q2 | q3 |
| q3 | q3 | q4 |
| q4 | q4 | q5 |
| q5 | q5 | q6 |
| q6 | q6 | q7 |
| q7 | q7 | q8 |
| q8 | q8 | q9 |
| q9 | q9 | q0 |

이 때, 의 state 값을 부여할 수 있다.

위에서 정리한 내용을 바탕으로 Verilog code를 사용해 decade counter의 모듈을 생성한 것은 다음과 같았다.

|  |
| --- |
| `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) begin  out <= 4'b0000;  end  else if(out >= 4'b1001) begin  out <= 4'b0000;  end  else begin  out <= out + 1;  end  end  endmodule |

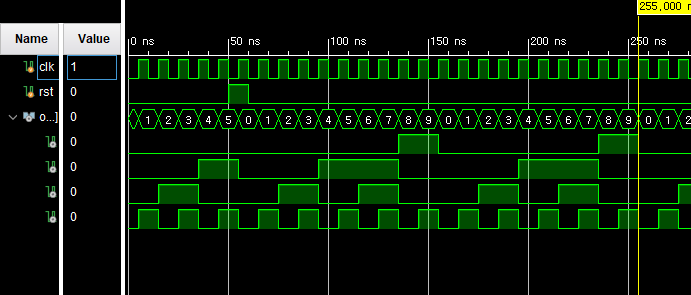
기본적으로 4-bit counter의 기능과 유사하나, out 값이 인 상태에서 clk 입력이 들어온 경우, 다시 out register의 값을 0000으로 초기화시키는 부분을 추가해 decade counter를 구현했다. 구현한 code를 다이어그램으로 나타내었을 때 다음과 같았다.



구현한 module이 정상적으로 의도한 기능을 수행하는지 확인하기 위해, 다음과 같이 simulation code를 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module decade\_sim;  reg clk, rst;  wire[3:0] out;    decade\_counter dc(  .clk(clk),  .rst(rst),  .out(out)  );    initial clk = 0;  initial rst = 0;    always clk = #5 clk + 1;    initial begin  #50  assign rst = ~rst;  #10  assign rst = ~rst;  #1600  $finish;  end  endmodule |

2-bit counter 때와 마찬가지로, 이 simulation으로 두 가지를 확인하고자 했다. 하나는 counter가 정상적으로 입력 pulse에 따라 counting을 실시하고, 9 다음에 0의 state로 정상적으로 변화하는지, 그리고 다른 하나는 reset 기능이 정상적으로 구현되었는지이다. 이를 확인하기 위해, 10ns 주기의 clock을 설정하고, 어느 정도 counter가 진행된 후, reset line에 high 값을 주어 reset 기능이 정상작동 하는지 테스트했다. 테스트한 결과는 다음과 같았다.



테스트 결과, 0~9까지의 state가 의도한 대로 clock의 leading edge에 맞춰 변화했고, reset signal 또한 정상적으로 state를 0으로 reset 시키는 것을 확인할 수 있어, 구현한 모듈이 정확하다는 것을 볼 수 있었다.

1. 4-bit 2421 decade counter의 결과 및 simulation 과정에 대해서 설명하시오.

2421 decade counter는 2.번에서 구현한 decade counter에 추가적으로 출력 값의 형식을 2421 code로 바꿔주는 counter이다. 따라서, BCD to 2421 code converter를 위에서 구현한 decade counter의 출력에 추가해 주어 해당 모듈을 간단하게 구현할 수 있다. 2421 decade counter의 진리표는 다음과 같이 나타낼 수 있는데, 이는 일반적인 decade counter의 state에 대응하는 binary 값만을 바꾼 모습을 보인다.

|  |  |  |
| --- | --- | --- |
| **Present state** | **next state** | |
| **qi** | x = 0 | x = 1 |
| q0 | q0 | q1 |
| q1 | q1 | q2 |
| q2 | q2 | q3 |
| q3 | q3 | q4 |
| q4 | q4 | q5 |
| q5 | q5 | q6 |
| q6 | q6 | q7 |
| q7 | q7 | q8 |
| q8 | q8 | q9 |
| q9 | q9 | q0 |

State diagram의 경우에도, 아래의 decade counter의 state diagram에서 대응하는 숫자를 2421 code 형식으로 변환한 것이 counter의 state가 된다고 할 수 있다.



위에서 설명했듯이, 2421 decade counter를 만들기 위해선 2.번에서 작성한 decade counter module에 추가적으로 2421 code converter를 생성해야 하므로, 해당 code converter module을 Verilog code로 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  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 |

위 모듈을 사용해 2421 decade counter를 구현한 것이 다음 코드이다.

|  |
| --- |
| `timescale 1ns / 1ps  module code2421\_counter(  clk, rst, out  );  input clk, rst;  output[3:0] out;    wire[3:0] connecter;    decade\_counter dc(  .clk(clk),  .rst(rst),  .out(connecter)  );    bcd\_to\_2421 bt2(  .in(connecter),  .out(out)  );  endmodule |

스크린샷이(가) 표시된 사진

자동 생성된 설명

위 코드의 회로도 다이어그램을 나타내면 위와 같은 결과를 얻을 수 있는데, 코드에서 볼 수 있듯이 이미 구현한 decade counter의 출력을 bcd to 2421 code converter의 입력으로 연결시키고, 해당 모듈의 출력을 내보내는 것을 확인할 수 있다. 구현한 bcd to 2421 code converter의 구현한 회로 모습은 다음과 같이 + 아이콘을 눌러 확인할 수 있었다.

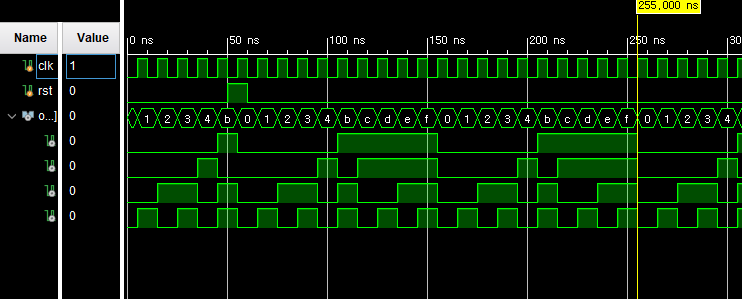
지도이(가) 표시된 사진

자동 생성된 설명

2421 decade counter의 구현이 정상적으로 이루어졌는지 확인해보기 위해, 다음의 simulation code를 작성하고 테스트해보고자 했다.

|  |
| --- |
| `timescale 1ns / 1ps  module code2421\_sim;  reg clk, rst;  wire[3:0] out;    code2421\_counter cc(  .clk(clk),  .rst(rst),  .out(out)  );    initial clk = 0;  initial rst = 0;  always clk = #5 clk + 1;    initial begin  #50  assign rst = ~rst;  #10  assign rst = ~rst;  #1600  $finish;  end  endmodule |

이 때, 결과에서 확인해보고 싶었던 두 가지는, 하나는 clk값의 leading edge에서 정상적으로 out 값이 2421 code에 맞춰 증가하는지였고, 다른 하나는 rst 값이 high 상태일 때 레지스터가 2421 code에서 0에 해당하는 값, 즉 0000이 되는가의 여부였다. 시뮬레이션 결과는 다음과 같이 나왔다.



위 결과에서부터, 10ns의 clock cycle이 지날 때 마다 register out의 값이 2421 code 시스템에 맞춰 1씩 증가한다는 것과, rst 값에 high가 들어왔을 때 out의 값이 0000으로 초기화된다는 사실을 확인할 수 있었고, 결과적으로 구현한 모듈이 의도한 작업을 잘 수행한다고 결론지었다.

1. 결과 검토 및 논의 사항

3가지 종류의 counter를 작성할 때, 중요했던 점은 3가지로 나눌 수 있었다. 첫 번째는 clock signal의 leading edge에서 counter의 state값이 변화해야 했고, 두 번째로 state 값이 설정된 최대 값(2bit의 경우 11, 4bit decade은 1001, 2421 decade은 1111)인 상태에서 clock signal을 받을 때 초기 값인 0으로 state가 돌아와야 할 것, 그리고 마지막 세 번째로는 reset line에 신호가 들어오면 state를 0으로 다시 초기화해서 다시 카운팅을 시작해야 한다는 것이 중요했다.

세 방식의 counter 구현에 있어서 위에서 언급한 이 세 가지의 기능을 모두 다 충족했음을 확인할 수 있었기 때문에, 구현한 counter 모듈이 정상적으로 잘 구현되어 작동한다는 것을 확실할 수 있다.

1. 추가 이론 조사 및 작성

* 2bit counter의 simulation 결과로부터, out[1]의 값은 clock signal의 4배 주기로, out[0]의 값은 clock signal의 2배 주기로 비트가 flip된다는 특성을 추가적으로 확인할 수 있었다. 이는 T Flip-Flop이 clock signal을 나누는 것과 같은 방식으로 사용되어, 다른 주기를 갖는 clock signal을 생성하는데 사용될 수 있다.