컴퓨터공학실험2 12주차 결과 보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20201635 이름: 전찬

**0. 목차**

1. 2-bit counter의 결과 및 simulation 과정 설명

2. 4-bit decade counter의 결과 및 simulation 과정 설명

3. 4-bit 2421 counter의 결과 및 simulation 과정 설명

4. 결과 검토 및 논의 사항

5. 추가 이론 조사 및 작성

**1. 2-bit counter의 결과 및 simulation 과정**

2-bit counter은 실제로 1~3의 신호의 발생을 count할 수 있는 회로이다. 1~3은 각각 01, 10, 11로 표현되며, 신호의 발생을 count하기 위해서 1bit를 저장할 수 있는 D/J-K flip-flop을 사용할 수 있다. 이번 실습에서는 D flip-flop의 형태로 코드를 작성했으며, 그 코드는 아래와 같다.

`timescale 1ns / 1ps

module counter\_2bit(

input pulse, reset,

output reg Q1, reg Q2

);

always @(posedge pulse)

begin

if(reset==1) begin

Q1 = 0;

Q2 = 0;

end

else begin

Q1 = ~Q1;

if(~Q1)

Q2 = ~Q2;

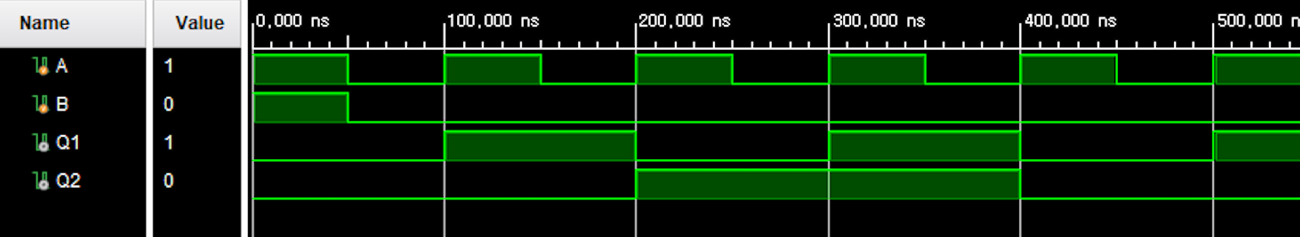
end

end

endmodule

<2-bit counter의 verilog code>

코드를 보면, positive pulse가 존재할 때, reset / count를 할 수 있는 형태이다. 또한 always @(posedge ~~) 형태를 사용해서, positive edge(rising edge)가 발생했을 때, always 안의 내용을 수행할 수 있도록 설계했다. 처음에는 내부적으로 D flip-flop module을 작성해서 사용하려고 했는데, always 문 안에 module을 넣는 방법에서 Verilog 오류의 표시(빨간 밑줄)이 나와서 그 방법을 사용하지는 못했다. 위 code에 대한 Verilog simulation 결과는 아래와 같다.(input A = pulse, input B = reset 이며 2bit = Q2 Q1 이다.)



<2-bit counter의 simulation 결과>

**2. 4-bit decade counter의 결과 및 simulation 과정**

위와 같이, 연속적으로 D flip-flop을 사용하면 counter을 구현할 수 있다. 4-bit counter을 구현하기 위해서는, 4개의 D flip-flop을 연속적으로 연결하면 된다. 여기에서 BCD code를 생각해보자. BCD code는 실제로 4bit에 15까지 저장할 수 있지만, 여러 가지 편의를 위해서 0~9 까지를 저장하는 형태이다. 이처럼 counter 또한 4bit(4개의 D flip-flop) 내에 1111까지 저장하는 것이 아닌, 0000->0001->0010-> …. ->1001->0000 형태를 반복하는 형태를 구현할 수 있다. 이 형태가 4-bit decade counter이며, 코드는 아래와 같다.

`timescale 1ns / 1ps

module decade\_counter(

input pulse, reset,

output reg Q1, reg Q2, reg Q3, reg Q4

);

always @(posedge pulse)

begin

if(reset==1) begin

Q1 = 0;

Q2 = 0;

Q3 = 0;

Q4 = 0;

end

else begin

Q1 = ~Q1;

if(~Q1)

Q2 = ~Q2;

if(~Q1&~Q2)

Q3 = ~Q3;

if(~Q1&~Q2&~Q3)

Q4 = ~Q4;

if(Q2&Q4) begin

Q1 = 0;

Q2 = 0;

Q3 = 0;

Q4 = 0;

end

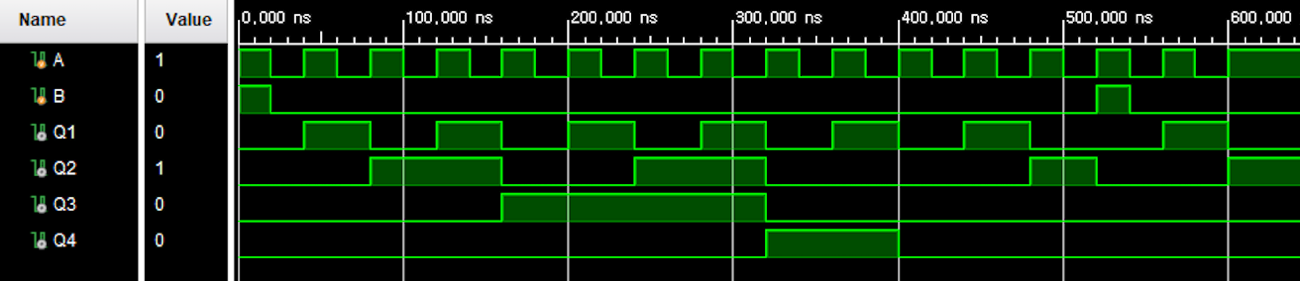
end

end

endmodule

<4-bit decade counter의 verilog code>

위 code를 보면, 4개의 D flip-flop에서 Q4, Q3, Q2, Q1에 1bit씩을 저장하며, 만약 Q4 = 1, Q2 = 1, 형태가 된다면 Q4, .., Q1을 다시 reset하는 형태를 볼 수 있다. 이는 1001을 저장한 이후 그 다음 count인 1010이 저장된 직후에, 0000으로 바꾸어주는 작업이라고 할 수 있다. 또한 이에 대한 simulation 결과는 아래와 같다.(A = pulse, B = reset 이다.)



<4-bit decade counter의 simulation 결과>

위 결과를 보면, 1001이 저장된 이후 그 다음 A = 1이 들어왔을 때, output = 0000이 되는 형태를 볼 수 있다.

**3. 4-bit 2421 counter의 결과 및 simulation 과정**

BCD를 통해서 4bit에 0~9를 저장하는 것처럼 2421 code를 통해서도 4bit에서 0~9를 저장할 수 있으며, 이전 실습에서 이를 구현한 적이 있다. 따라서 4-bit 2421 counter은 위에서 구현한 decade counter에 BCD to 2421 converter을 추가한 형태로 구현할 수 있다. 이에 대한 코드는 아래와 같다.

`timescale 1ns / 1ps

module BCD\_converter(

input A, B, C, D,

output W, X, Y, Z

);

assign W = ~((~A) & (~(B&D)) & (~(B&C)));

assign X = ~((~A) & (~(B&(~D))) & (~(B&C)));

assign Y = ~((~A) & (~(B&(~C)&D)) & (~((~B)&C)));

assign Z = D;

endmodule

module decade\_counter(

input pulse, reset,

output reg Q1, reg Q2, reg Q3, reg Q4

);

always @(posedge pulse)

begin

if(reset==1) begin

Q1 = 0;

Q2 = 0;

Q3 = 0;

Q4 = 0;

end

else begin

Q1 = ~Q1;

if(~Q1)

Q2 = ~Q2;

if(~Q1&~Q2)

Q3 = ~Q3;

if(~Q1&~Q2&~Q3)

Q4 = ~Q4;

if(Q2&Q4) begin

Q1 = 0;

Q2 = 0;

Q3 = 0;

Q4 = 0;

end

end

end

endmodule

module counter\_2421(

input pulse, reset,

output X1, X2, X3, X4

);

wire BCD\_1, BCD\_2, BCD\_3, BCD\_4;

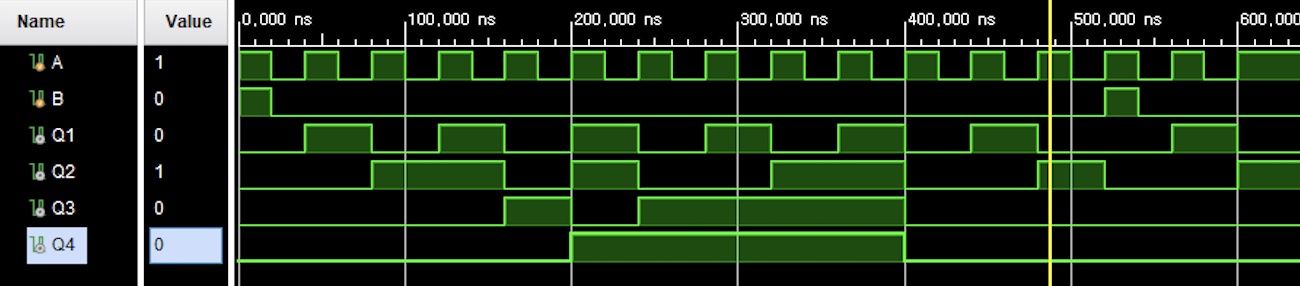
decade\_counter count(.pulse(pulse), .reset(reset), .Q1(BCD\_1), .Q2(BCD\_2), .Q3(BCD\_3), .Q4(BCD\_4));

BCD\_converter convert(.A(BCD\_4), .B(BCD\_3), .C(BCD\_2), .D(BCD\_1), .W(X4), .X(X3), .Y(X2), .Z(X1));

endmodule

<4-bit 2421 counter의 verilog code>

또한 simulation 결과는 아래와 같다.(A = pulse, B = reset 이다.)



<4-bit 2421 counter의 simulation 결과>

**4. 결과 검토 및 논의 사항**

이번 실습에서는 D flip-flop의 toggle을 활용해 counter을 구현하며, 또한 decade / 2421 counter 등 다양한 형태의 counter을 구현했다. 또한 input으로 pulse 뿐만이 아닌, reset 또한 input으로 활용하는 형태를 구현하기도 했다.

또한 처음에 구현하려고 했던 형태인 always loop 안에 module이 적용된 형태로는 왜 구현할 수 없는지에 대해서 알아봤는데, 공식적인 문서는 찾을 수 없었지만, always loop가 parallel한 형태로 인식되기 때문에 실제로 몇 개의 module이 필요한지 파악할 수 없어 사용할 수 없다는 내용들이 stackoverflow, wikibooks와 같은 site에 존재하기는 했다. 이와 동일한 개념으로, wire 형식은 loop 내부에서 사용할 수 없기 때문에, 실제로 위의 코드들에서 볼 수 있듯이, output을 reg로 정의하고 코드를 작성하기도 했다.

**5. 추가 이론 조사 및 작성**

이번 실습에서 D flip-flop을 활용해서 counter을 구현했는데, D flip-flop에는 특별한 성질이 존재한다. 바로 Frequency divider이라는 것인데, clock pulse가 존재할 때만 input D = Q\_com; 형태로 대응하게 되면, 원래 존재하는 clock pulse에 대해 frequency가 1/2로 나눠지는 새로운 신호를 만들어낼 수 있다는 것이다. 바꿔서 이야기하자면 T(주기)가 2배인 pulse를 만들어낼 수 있다. 또한 이는 통신 회로에서 여러 방법으로 적용할 수 있다. 그 예시로, frequency division multiplexing이라는 기술을 통해서 다양한 신호가 여러 frequency로 들어올 때, multiplexer을 통해서 그 중에서 하나의 input 신호만을 선택하는 기술이 존재한다. 이때 신호를 하나는 frequency F, 나머지는 F/2, F/4, … 형태로 구현한다면 원하는 frequency에 맞게 필요한 정보만 받아서 사용하는 기술이 frequency division multiplexing 이다.