12주차 결과보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20191629 이름: 이주헌

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

2비트 계수기는 0부터 3까지의 수를 셀 수 있는 계수기이다. 계수기는 크게 동기 계수기와 비동기 계수기, 두 가지 종류로 나눌 수 있는데, 여기에서는 동기 계수기를 ㄹ구현하였다. 동기 계수기의 구현체도 여러 가지 종류로 나뉘는데, 가장 간단한 T 플립플롭을 사용한 동기 계수기를 만들어 보았다. T 플립플롭도 여러 가지 방법으로 구현할 수 있지만, 이미 11주차 실험에서 JK 플립플롭을 구현한 바가 있고, JK 플립플롭의 J 입력과 K 입력이 모두 1이면 T 플립플롭과 같은 역할을 한다는 점을 고려하여 JK 플립플롭을 이용하였다. 재설정 기능을 추가한 JK 플립플롭의 Verilog 소스코드는 다음과 같다.

```
`timescale 1ns / 1ps

module jk_flip_flop(j, k, clk, rst, q, nq);
    input j, k, clk, rst;
    output q, nq;
    reg s, r;

    initial begin
        s = 0;
        r = 1;
    end

    always @(posedge clk) begin
        if (rst) begin
            s <= 0;
            r <= 1;
        end
        else begin
            s <= j & nq;
            r <= k & q;
        end
    end

    nor_sr_latch device(.s(s), .r(r), .q(q), .nq(nq));
```

```
endmodule
```

여기서 nor_sr_latch 모듈은 11주차에서 구현한 RS 래치 모듈이다. 재설정 입력을 1으로 하고 클럭 신호를 주면 Q 출력이 0, $\neg Q$ 출력이 1으로 재설정된다는 점을 제외하면, 11주차에서 구현한 모듈과 동작이 완전히 똑같으므로 이 모듈의 시뮬레이션은 생략한다. 이 JK 플립플롭을 두 개 연결하면 다음과 같이 2비트 계수기를 구현할 수 있다.

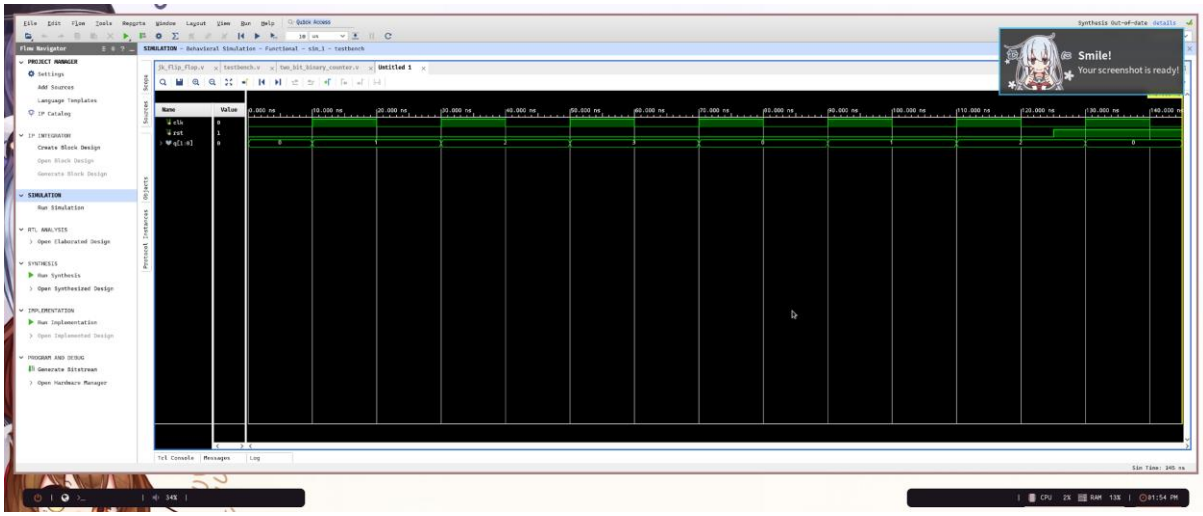
```
`timescale 1ns / 1ps

module two_bit_binary_counter(clk, rst, q);
    input clk, rst;
    output [1:0] q;

    jk_flip_flop
jk0(.j(1), .k(1), .clk(clk), .rst(rst), .q(q[0]), .nq());
    jk_flip_flop
jk1(.j(q[0]), .k(q[0]), .clk(clk), .rst(rst), .q(q[1]), .nq());

endmodule
```

여기서 2비트 크기의 출력 q가 지금까지 센 수이다. 개별적으로 T 플립플롭이 연결되어 있는 이 계수기의 특성상, 현재 숫자가 11_2일 때에도 따로 재설정을 해줄 필요가 없다. 여기서 7번 클럭 신호를 주고 마지막 클럭 신호에 1회 재설정하면 다음과 같은 결과가 도출되어야 한다.

클럭 신호 횟수	[1:0] q의 결과
0회	00
1회	01
2회	10
3회	11
4회	00
5회	01
6회	10

7회 (재설정)	00

이 결과는 시뮬레이션을 통해 다음과 같이 검증할 수 있다.



2. 4-bit decade counter의 결과 및 simulation 과정에 대해서 설명하시오.

4비트 십진 계수기는 BCD 숫자를 세는 계수기이다. 즉, 0부터 15까지의 수를 셀 수 있는 일반적인 4비트 계수기와는 달리, 0부터 9까지의 숫자만 셀 수 있다. 모든 경우의 수에 대하여 진리표를 그리고, 카르노 맵을 통하여 올바른 식을 도출하는 것도 방법이지만, 이미 기본적인 계수기를 JK 플립플롭을 이용하여 구현할 수 있다는 점에서 해당 방법은 불필요하게 복잡한 방법이라고 생각한다. 4비트 십진 계수기의 알고리즘을 생각해보면 다음과 같이 쓸 수 있다.

1. 4비트 이진 계수기를 작성한다.
2. 만약 계수기가 나타내는 숫자가 1001_2 미만이면 계속 다음 숫자를 센다.
3. 만약 계수기가 나타내는 숫자가 1001_2 이상이면 다음 클럭 신호에 계수기를 초기화하여 0000_2으로 돌린다.

알고리즘의 (3)번 단계의 “초기화” 작업을 JK 플립플롭의 재설정 입력으로 구현하면 훨씬 더 간단하게 4비트 십진 계수기를 구현할 수 있다. 계수기의 Verilog 소스코드는 다음과 같다.

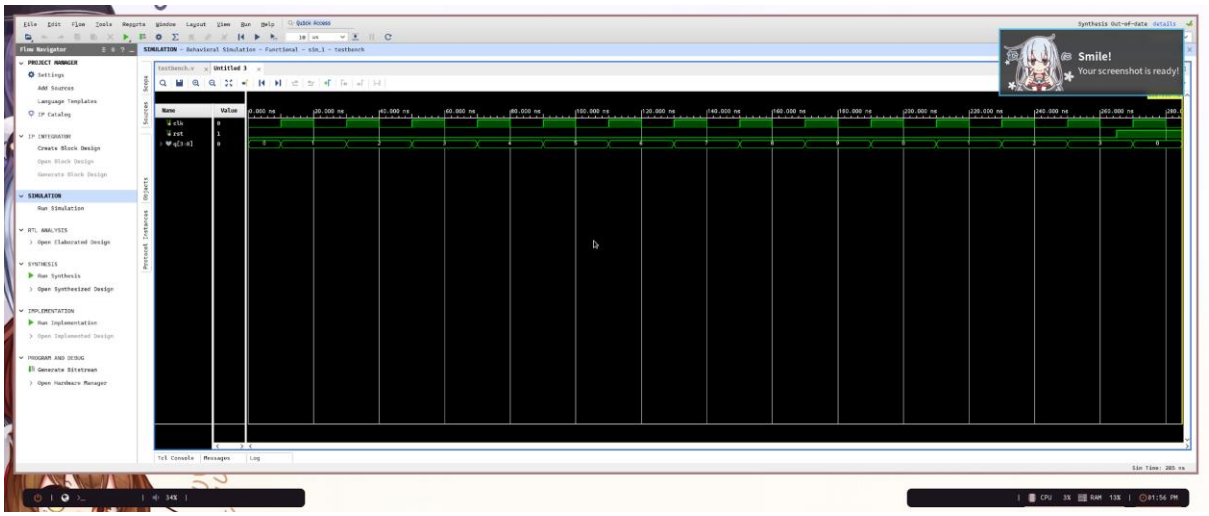
```
`timescale 1ns / 1ps
```

```
module four_bit_8421_decade_counter(clk, rst, q);
    input clk, rst;
    output [3:0] q;

    wire rst_combined;

    // 9(10) = 1001(2)              1       0       0       1
    assign rst_combined = rst | (q[3] & ~q[2] & ~q[1] & q[0]);

    jk_flip_flop
jk0(.j(1), .k(1), .clk(clk), .rst(rst_combined), .q(q[0]), .nq());
    jk_flip_flop
jk1(.j(q[0]), .k(q[0]), .clk(clk), .rst(rst_combined), .q(q[1]), .nq());
    jk_flip_flop jk2(.j(q[1] & q[0]), .k(q[1] &
q[0]), .clk(clk), .rst(rst_combined), .q(q[2]), .nq());
    jk_flip_flop jk3(.j(q[2] & q[1] & q[0]), .k(q[2] & q[1] &
q[0]), .clk(clk), .rst(rst_combined), .q(q[3]), .nq());

endmodule
```

재설정 플래그와 초기화 입력을 OR 게이트를 사용하여 연결하였기 때문에 초기화 입력도 문제없이 사용할 수 있다. 2비트 계수기와 마찬가지로 14번 클럭 신호를 주고, 마지막 클럭 신호에 1회 재설정 입력을 주면 다음과 같은 결과가 나와야 한다.

클럭 신호 횟수	[3:0] q의 결과
0회	0000
1회	0001
2회	0010
3회	0011
4회	0100
5회	0101
6회	0110
7회	0111
8회	1000

9회	1001
10회	0000
11회	0001
12회	0010
13회	0011
14회 (재설정)	0000



위 코드를 시뮬레이션한 결과, 예상했던 것과 같은 결과가 나온다는 사실을 확인할 수 있었다.

3. 4-bit 2421 decade counter의 결과 및 simulation 과정에 대해서 설명하시오.

2421 BCD 코드는 4비트 수의 최상위 비트를 8이 아닌 2로 해석하는 십진수 표현법으로, 5를 경계로 대칭 모양을 한다는 점이 특징이다. 즉, 0부터 4까지는 평소와 같이 수를 세다가 5로 넘어갈 때 4를 반전하여 저장하는 것이다. 4비트 8421 계수기의 때와 같이 2421 계수기의 알고리즘을 생각해보면 다음과 같이 쓸 수 있다.

1. 4비트 이진 계수기를 작성한다.
2. 현재 숫자가 4가 아닐 경우, 평소와 같이 숫자를 센다.
3. 현재 숫자가 4일 경우, 다음 클럭 신호 때 현재 숫자의 4비트 NOT 연산을 취한다.

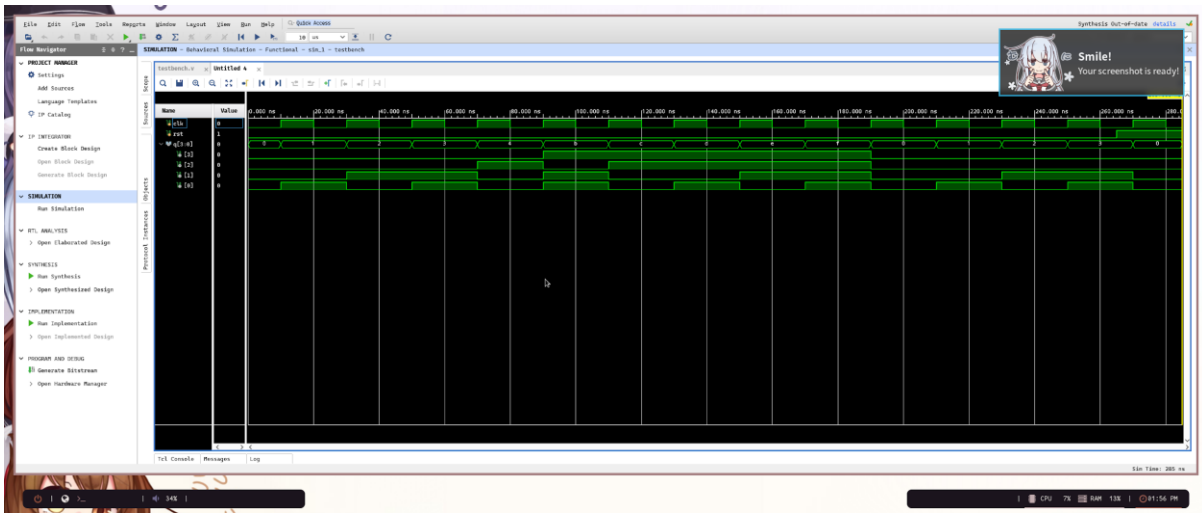
알고리즘의 (3)번 단계의 경우, NOT 연산을 취하는 것으로 작성하였지만, 실제로는 NOT 연산을 취하는 경우가 현재 숫자가 4인 경우 뿐이므로 만약 현재 숫자가 0100_2인 경우, 다음 클럭 신호 때 현재 숫자를 1011_2로 변경한다는 것으로 해석할 수 있다. 여기서 우리가 T 플립플롭을 사용하는 것이 아니라, JK 플립플롭을 사용하고 있기 때문에 각 플립플롭의 J와 K 입력을 잘 조절하면 출력 숫자를 임의로 조절할 수 있다. 해당 알고리즘을 구현하는 Verilog 코드는 다음과 같다.

```
`timescale 1ns / 1ps

module four_bit_2421_decade_counter(clk, rst, q);
    input clk, rst;
    output [3:0] q;

    wire jump_flag;

    // Jump when q = 4 =>  0      1       0       0
    assign jump_flag = ~q[3] & q[2] & ~q[1] & ~q[0];

    jk_flip_flop
jk0(.j(1), .k(~jump_flag), .clk(clk), .rst(rst), .q(q[0]), .nq());
    jk_flip_flop jk1(.j(jump_flag | q[0]), .k(~jump_flag &
q[0]), .clk(clk), .rst(rst), .q(q[1]), .nq());
    jk_flip_flop jk2(.j(~jump_flag & q[1] & q[0]), .k(jump_flag | q[1] &
q[0]), .clk(clk), .rst(rst), .q(q[2]), .nq());
    jk_flip_flop jk3(.j(jump_flag | (q[2] & q[1] & q[0])), .k(~jump_flag
& (q[2] & q[1] & q[0])), .clk(clk), .rst(rst), .q(q[3]), .nq());
endmodule
```

4비트 8421 계수기와 마찬가지로, 14번 클럭 신호를 주고, 마지막 클럭 신호에 1회 재설정 입력을 주면 다음과 같은 결과가 나와야 한다.

클럭 신호 횟수	[3:0] q의 결과
0회	0000
1회	0001
2회	0010
3회	0011

4회	0100
5회	1011
6회	1100
7회	1101
8회	1110
9회	1111
10회	0000
11회	0001
12회	0010
13회	0011
14회 (재설정)	0000



위 코드를 시뮬레이션한 결과, 우리가 예상한 것과 같은 결과가 나오는 것을 확인할 수 있었다.

4. 결과 검토 및 논의 사항

이번 실습에서는 3가지 종류의 계수기를 만들었다. 가장 먼저 만든 것은 2비트 이진 계수기로, 여러 개의 T 플립플롭을 연결하여 제작하였다. 이 때 미리 만들어 둔 모듈을

수정하고, 다시 재활용해 볼 수 있는 기회가 되었다. 또, 실제 FPGA를 플래싱하여 실험하였을 때, 스위치가 아닌 버튼을 사용하여 실험하였는데, 어떤 경우에는 두 번 이상 클럭 신호가 들어가는 것을 확인할 수 있었다. 이는 금속 판이 또 다른 금속 판과 접촉하여 전기 신호를 흘려보내는 버튼의 물리적 한계로서, 버튼이 눌릴 때 금속 판이 떨려서 접촉이 여러 번 발생하는 것이 원인으로 판명되었다.

다음으로 4비트 십진 계수기를 구현하였다. 8421 십진 계수기의 경우 0부터 시작하여 10이 되기 전에 0으로 숫자를 되돌리는 방식으로 동작하기 때문에, T 플립플롭의 재설정 입력을 사용하여 손쉽게 구현할 수 있었다. 십진 계수기의 기본이 되는 이진 계수기 역시 2비트 계수기의 패턴을 그대로 적용하여 만들었다.

마지막으로 4비트 2421 계수기를 구현하였다. 이 계수기의 경우, 0으로 되돌아가는 것이 아닌, 0101_2부터 1010_2까지의 범위를 건너뛰어야 하기 때문에 일반적인 T 플립플롭으로는 구현하기 까다로웠다. 그러나 이 실험에서 모든 T 플립플롭은 이전 실험에서 구현한 JK 플립플롭의 J와 K 입력을 연결하여 구현되었기 때문에, J와 K 입력은 각각 RS 래치의 S와 R 입력에 대응된다는 점을 이용하여 비교적 쉽게 “건너뛰는” 동작을 구현할 수 있었다.

5. 추가 이론 조사 및 작성

영어 위키백과의 Binary-coded decimal 페이지를 살펴보면, 우리가 구현한 8421과 2421 코드 말고도 여러 종류의 BCD 표현법이 있다는 사실을 알 수 있다. 여기에서 4비트 XS-3 십진 계수기를 구현해 보자.

XS-3 십진 코드는 9의 보수를 쉽게 취할 수 있다는 것이 장점인 십진수 표현법으로, 0부터 9를 각각 0011_2부터 1100_2에 대응시킨다는 것이 특징이다. 따라서, 1100_2가 넘어가면 0000_2로 돌아가는 것이 아니라 0011_2으로 돌아가야 한다. 또한, 재설정 시에도 0000_2이 아닌 0011_2으로 재설정되어야 한다. 이를 알고리즘으로 나타내면 다음과 같다.

1. 4비트 이진 계수기를 구현한다.
2. 만약 현재 숫자가 0011_2 이상, 1100_2 미만이면 평소와 같이 숫자를 센다.
3. 만약 현재 숫자가 1100_2이면 다음 클럭 신호에 0011_2으로 되돌아간다.

이 알고리즘을 Verilog로 구현하면 다음과 같다.

```
`timescale 1ns / 1ps
```

```
module four_bit_xs3_decade_counter(clk, rst, q);
    input clk, rst;
    output [3:0] q;

    wire rst_combined;

    // 9(10) = 1100(2)              1      1       0       0
    assign rst_combined = rst | (q[3] & q[2] & ~q[1] & ~q[0]);

    jk_flip_flop
jk0(.j(~rst_combined), .k(1), .clk(clk), .rst(0), .q(q[0]), .nq());
    jk_flip_flop jk1(.j(~rst_combined & q[0]), .k(rst_combined |
q[0]), .clk(clk), .rst(0), .q(q[1]), .nq());
    jk_flip_flop jk2(.j(rst_combined | q[1] & q[0]), .k(~rst_combined &
q[1] & q[0]), .clk(clk), .rst(0), .q(q[2]), .nq());
    jk_flip_flop jk3(.j(rst_combined | q[2] & q[1] &
q[0]), .k(~rst_combined & q[2] & q[1] &
q[0]), .clk(clk), .rst(0), .q(q[3]), .nq());

endmodule
```