**12주차 결과 보고서**

20150555 남민혁

**1.** **2-bit counter 의 결과 및 Simulation 과정에 대해서 설명하시오. (verliog source, 출력 예시, 과정 상세히 적을것!)**

2-bit counter는 clock에 의하여 신호가 들어올 때 정해진 순서대로 00, 01, 10, 11을 순환하며 출력하는 논리회로이다. 구체적인 입력과 그에 따른 출력의 결과는 다음과 같다.

**- State Table**

|  |  |  |
| --- | --- | --- |
| Present State | Next Step / Output | |
| Input CLK = 0 | Input CLK = 1 |
| 00 (0) | 00 (0) | 01 (1) |
| 01 (1) | 01 (1) | 10 (2) |
| 10 (2) | 10 (2) | 11 (3) |
| 11 (3) | 11 (3) | 00 (0) |

CLK이 꺼져있는 상태에서는 다음 상태가 현재 상태와 출력이 동일하며, CLK이 켜진 경우에는 다음 상태가 다음 숫자로 바뀌며, 마지막 state인 11 (3)에 도달한 경우 다음 state은 00 (0)으로 초기화된다.

**-Verilog 코딩**

Two-bit counter구현을 위하여 module two\_bit\_counter(clk, reset, q);로서 모듈을 선언하였으며, 각각의 입력에 따라 state table에서 정해진 결과를 출력하도록 코딩하였다. 구현 조건으로서 11주차에 구현한 JK Flip-flop을 활용하기 위하여 module t\_jk\_ff(j, k, clk, q, qc); 모듈도 선언되었다. 특히 two\_bit\_counter에서 tmpq와 qc는 각각 JK Flip-flop에 의하여 반전된 결과의 출력을 넣은 곳이며, tmpwire는 JK Flip-flop의 입력 값으로 사용되는 JK를 결정짓는 입력 값이다. 첫번째 비트는 매 CLK마다 반전이 발생하며, 두번째 비트는 이전 상태가 01이거나 11인 경우이만 해당되므로, 해당 조건인 경우에만 반전되도록 구현하였다.

각 행은 논리회로 구성을 위한 Verilog 코드, 이를 바탕으로 모든 가능한 입력으로 부터 기대되는 출력이 나타나는지를 확인하기 위한 Test bench 코드이다.

|  |  |
| --- | --- |
| Verilog | `timescale 1ns / 1ps  module two\_bit\_counter(  input clk, reset,  output wire [1:0] q  );    wire[1:0] qc;  wire[1:0] tmpq;  wire[2:0] tmpwire;  t\_jk\_ff t\_b1(1'b1,1'b1,clk,tmpq[0], qc[0]);  and(tmpwire[0], tmpq[0], qc[1]);  and(tmpwire[1], tmpq[0], tmpq[1]);  or(tmpwire[2], tmpwire[1], tmpwire[0]);  t\_jk\_ff t\_b2(tmpwire[2], tmpwire[2], clk, tmpq[1], qc[1]);    and(q[0], ~reset, tmpq[0]);  and(q[1], ~reset, tmpq[1]);  endmodule    module t\_jk\_ff(  input j, k, clk,  output reg q, qc  );  initial begin  q=0;  qc=1;  end  always@(negedge clk)begin  if(j==0&&k==0) begin  q<=q;  qc<=qc;  end  if(j==0&&k==1) begin  q=0;  qc=1;  end  if(j==1&&k==0) begin  q=1;  qc=0;  end  if(j==1&&k==1)begin  q<=qc;  qc<=q;  end  end  endmodule |
| Test Bench | `timescale 1ns / 1ps  module two\_bit\_counter\_tb();  reg clk; reg reset;  wire [1:0] O;  two\_bit\_counter connect(clk, reset, O);  initial begin  clk = 0; reset = 0;  end;  always clk = #10 ~clk;  always @(posedge clk) begin  end  endmodule |

**-Simulation 출력 결과 비교**

**A picture containing timeline

Description automatically generated**

모든 가능한 입력 값에 대한 출력값을 확인하기 위하여 Simulation을 시행한 결과 의도한 진리표와 모든 결과가 동일함을 확인하였다.

**-출력 예시**

비대면 수업으로 인하여 실제 FPGA로 부터 입력을 받지 못하였으나, 기존 실험때와 마찬가지로CLK 비트를 스위치로, 첫번째와 두번째 비트를 각각 LED에 불이 들어오는 것을 1로 가정하면, 스위치가 한번 왔다 갔다 할 때마다 00, 01, 10, 11로 바뀌고, 마지막에는 다시 00으로 돌아오는 것을 LED의 상태 변화로서 확인할 수 있을 것이다. 이는 위의 Simulation 결과를 통하여 확인할 수 있다.

**-구현된 Schematic**

Diagram

Description automatically generated

**2. 4-bit decade counter의 결과 및 Simulation 과정에 대해서 설명하시오. (verliog source, 출력 예시, 과정 상세히 적을것!)**

4-bit decade counter는 clock에 의하여 신호가 들어올 때 정해진 순서대로 0000 (0) 부터 1001 (9)까지 10개의 상태를 순환하며 출력하는 논리회로이다. 구체적인 입력과 그에 따른 출력의 결과는 다음과 같다.

**-Truth Table**

|  |  |  |
| --- | --- | --- |
| Present State | Next Step / Output | |
| Input x = 0 | Input x = 1 |
| 0000 (0) | 0000 (0) | 0001 (1) |
| 0001 (1) | 0001 (1) | 0010 (2) |
| 0010 (2) | 0010 (2) | 0011 (3) |
| 0011 (3) | 0011 (3) | 0100 (4) |
| 0100 (4) | 0100 (4) | 0101 (5) |
| 0101 (5) | 0101 (5) | 0110 (6) |
| 0110 (6) | 0110 (6) | 0111 (7) |
| 0111 (7) | 0111 (7) | 1000 (8) |
| 1000 (8) | 1000 (8) | 1001 (9) |
| 1001 (9) | 1001 (9) | 0000 (0) |

CLK이 꺼져있는 상태에서는 다음 상태가 현재 상태와 출력이 동일하며, CLK이 켜진 경우에는 다음 상태가 다음 숫자로 바뀌며, 마지막 state인 1001 (9)에 도달한 경우 다음 state은 0000 (0)으로 초기화된다.

**-Verilog 코딩**

Four-bit decade counter구현을 위하여 module four\_bit\_decade\_counter(clk, reset, q);로서 모듈을 선언하였으며, 각각의 입력에 따라 state table에서 정해진 결과를 출력하도록 코딩하였다. 구현 조건으로서 11주차에 구현한 JK Flip-flop을 활용하기 위하여 module t\_jk\_ff(j, k, clk, q, qc); 모듈도 선언되었다. 특히 four\_bit\_decade\_counter에서 tmpq와 qc는 각각 JK Flip-flop에 의하여 반전된 결과의 출력을 넣은 곳이며, tmpwire는 JK Flip-flop의 입력 값으로 사용되는 JK를 결정짓는 입력 값이다. 첫번째 비트는 매 CLK마다 반전이 발생하며, 두번째 비트는 이전 상태가 0xx1인 경우에만 해당되므로, 해당 조건인 경우에만 반전되도록 구현하였다. 세번째 비트 역시 비슷하게 xx11인 경우에만 반전이 필요하며, 마지막 네번째 비트는 x111인 경우나, 1xx1인 경우에 해당되므로, 해당 조건이 만족하는 경우에 JK 입력이 모두 1이 되도록 구현하였다.

각 행은 논리회로 구성을 위한 Verilog 코드, 이를 바탕으로 모든 가능한 입력으로 부터 기대되는 출력이 나타나는지를 확인하기 위한 Test bench 코드이다.

|  |  |
| --- | --- |
| Verilog | `timescale 1ns / 1ps  module four\_bit\_decade\_counter(  input clk, reset,  output wire [3:0] q  );    wire[3:0] qc;  wire[3:0] tmpq;  wire[6:0] tmpwire;  jk\_ff b1(1'b1,1'b1,clk,tmpq[0], qc[0]);  and(tmpwire[0], tmpq[0], qc[3]);  jk\_ff b2(tmpwire[0], tmpwire[0], clk, tmpq[1], qc[1]);  and(tmpwire[1], tmpq[0], tmpq[1]);  jk\_ff b3(tmpwire[1], tmpwire[1], clk, tmpq[2], qc[2]);  and(tmpwire[2], tmpwire[1], tmpq[2]);  and(tmpwire[3], tmpq[0], tmpq[3]);  or(tmpwire[4], tmpwire[2], tmpwire[3]);  jk\_ff b4(tmpwire[4], tmpwire[4], clk, tmpq[3], qc[3]);    and(q[0], ~reset, tmpq[0]);  and(q[1], ~reset, tmpq[1]);  and(q[2], ~reset, tmpq[2]);  and(q[3], ~reset, tmpq[3]);  endmodule    module jk\_ff(  input j, k, clk,  output reg q, qc  );  initial begin  q=0;  qc=1;  end  always@(negedge clk)begin  if(j==0&&k==0) begin  q<=q;  qc<=qc;  end  if(j==0&&k==1) begin  q=0;  qc=1;  end  if(j==1&&k==0) begin  q=1;  qc=0;  end  if(j==1&&k==1)begin  q<=qc;  qc<=q;  end  end  endmodule |
| Test Bench | `timescale 1ns / 1ps  module four\_bit\_decade\_counter\_tb();  reg clk; reg reset;  wire [3:0] O;  four\_bit\_decade\_counter connect(clk, reset, O);  initial begin  clk = 0; reset = 0;  end;  always clk = #10 ~clk;  always @(posedge clk) begin  end  endmodule |

**-Simulation 출력 결과 비교**

**A picture containing chart

Description automatically generated**

모든 가능한 입력 값에 대한 출력값을 확인하기 위하여 Simulation을 시행한 결과 의도한 진리표와 모든 결과가 동일함을 확인하였다.

**-출력 예시**

비대면 수업으로 인하여 실제 FPGA로 부터 입력을 받지 못하였으나, 기존 실험때와 마찬가지로CLK 비트를 스위치로, 각 비트를 순서대로 LED에 불이 들어오는 것으로서 1로 가정하면, 스위치가 한번 왔다 갔다 할 때마다 0000부터 1001까지 바뀌며 다시 0000으로 돌아오며 순환하는 것을 LED로서 확인할 수 있을 것이다. 이는 위의 Simulation 결과를 통하여 확인할 수 있다.

**-구현된 Schematic**

Diagram

Description automatically generated

**3. 4-bit 2421 decade counter 의 결과 및 Simulation 과정에 대해서 설명하시오. (verliog source, 출력 예시, 과정 상세히 적을것!)**

4-bit 2421 decade counter는 clock에 의하여 신호가 들어올 때 정해진 순서대로 0000 (0) 부터 1111 (9)까지 10개의 상태를 순환하며 출력하는 논리회로이다. 이때 각 출력은 2421코드로 구현되었다. 구체적인 입력과 그에 따른 출력의 결과는 다음과 같다.

**-Truth Table**

|  |  |  |
| --- | --- | --- |
| Present State | Next Step / Output | |
| Input x = 0 | Input x = 1 |
| 0000 (0) | 0000 (0) | 0001 (1) |
| 0001 (1) | 0001 (1) | 0010 (2) |
| 0010 (2) | 0010 (2) | 0011 (3) |
| 0011 (3) | 0011 (3) | 0100 (4) |
| 0100 (4) | 0100 (4) | 1011 (5) |
| 1011 (5) | 1011 (5) | 1100 (6) |
| 1100 (6) | 1100 (6) | 1101 (7) |
| 1101 (7) | 1101 (7) | 1110 (8) |
| 1110 (8) | 1110 (8) | 1111 (9) |
| 1111 (9) | 1111 (9) | 0000 (0) |

CLK이 꺼져있는 상태에서는 다음 상태가 현재 상태와 출력이 동일하며, CLK이 켜진 경우에는 다음 상태가 다음 숫자로 바뀌며, 마지막 state인 1111 (9)에 도달한 경우 다음 state은 0000 (0)으로 초기화된다.

**-Verilog 코딩**

Four-bit 2421 decade counter구현을 위하여 module tfto\_decade\_counter(clk, reset, q);로서 모듈을 선언하였으며, 각각의 입력에 따라 state table에서 정해진 결과를 출력하도록 코딩하였다. 구현 조건으로서 11주차에 구현한 JK Flip-flop을 활용하기 위하여 module t\_jk\_ff(j, k, clk, q, qc); 모듈도 선언되었다. 더불어 2421 코드로서 출력 값을 표현하기 위하여 4비트 Full Adder도 함께 포함되었으며, 이를 위하여 1비트 Full Adder도 선언되었다. 또한 tfto\_decade\_counter은 사실상 기존 four bit decade counter와 동일하고, 출력의 형태만 달라지므로, 앞서 구현한 four\_bit\_decade\_counter(clk, reset, q) 도 함께 포함되어 있다.

출력값이 0부터 4까지는 동일하고, 5이상 부터는 BCD 코드 기준으로 0110 (6)이 더해진 값과 동일하므로, four\_bit\_decade\_counter(clk, reset, q)의 결과가 5를 넘는 경우 4비트 Full Adder로 0110을 더하는 방식으로 구현하였다.

각 행은 논리회로 구성을 위한 Verilog 코드, 이를 바탕으로 모든 가능한 입력으로 부터 기대되는 출력이 나타나는지를 확인하기 위한 Test bench 코드이다.

|  |  |
| --- | --- |
| Verilog | `timescale 1ns / 1ps  module tfto\_decade\_counter(  input clk, reset,  output wire [3:0] q  );  wire [3:0] tmpq;  wire check;  wire tmpwire;  reg [3:0] B;  tfto\_four\_bit\_decade\_counter tfto\_fourbit(clk, reset, tmpq);  assign check = (tmpq[3]&~tmpq[2])|(tmpq[2]&tmpq[0])|(tmpq[2]&tmpq[1]&~tmpq[0]);  always @(check)  if(check == 1'b1) begin  B = 4'b0110;  end  else begin  B = 4'b0000;  end  adder4bit add0(tmpq, B, 0, q, tmpwire);  endmodule  module tfto\_four\_bit\_decade\_counter(  input clk, reset,  output wire [3:0] q  );    wire[3:0] qc;  wire[3:0] tmpq;  wire[6:0] tmpwire;  tfto\_jk\_ff tfto\_b1(1'b1,1'b1,clk,tmpq[0], qc[0]);  and(tmpwire[0], tmpq[0], qc[3]);  tfto\_jk\_ff tfto\_b2(tmpwire[0], tmpwire[0], clk, tmpq[1], qc[1]);  and(tmpwire[1], tmpq[0], tmpq[1]);  tfto\_jk\_ff tfto\_b3(tmpwire[1], tmpwire[1], clk, tmpq[2], qc[2]);  and(tmpwire[2], tmpwire[1], tmpq[2]);  and(tmpwire[3], tmpq[0], tmpq[3]);  or(tmpwire[4], tmpwire[2], tmpwire[3]);  tfto\_jk\_ff tfto\_b4(tmpwire[4], tmpwire[4], clk, tmpq[3], qc[3]);    and(q[0], ~reset, tmpq[0]);  and(q[1], ~reset, tmpq[1]);  and(q[2], ~reset, tmpq[2]);  and(q[3], ~reset, tmpq[3]);  endmodule    module tfto\_jk\_ff(  input j, k, clk,  output reg q, qc  );  initial begin  q=0;  qc=1;  end  always@(negedge clk)begin  if(j==0&&k==0) begin  q<=q;  qc<=qc;  end  if(j==0&&k==1) begin  q=0;  qc=1;  end  if(j==1&&k==0) begin  q=1;  qc=0;  end  if(j==1&&k==1)begin  q<=qc;  qc<=q;  end  end  endmodule  module adder1bit(A, B, Ci, S, Co);  input A, B, Ci;  output S, Co;  assign S=A^B^Ci;  assign Co=(A&B)|((A^B)&Ci);  endmodule  module adder4bit(A, B, Ci, S, Co);  input [3:0] A, B; input Ci;  output [3:0] S; output Co;  wire [3:0] A, B, S; wire Ci, Co;  wire [2:0] C;  adder1bit add1(A[0], B[0], Ci, S[0], C[0]);  adder1bit add2(A[1], B[1], C[0], S[1], C[1]);  adder1bit add3(A[2], B[2], C[1], S[2], C[2]);  adder1bit add4(A[3], B[3], C[2], S[3], Co);  endmodule |
| Test Bench | `timescale 1ns / 1ps  module tfto\_decade\_counter\_tb();  reg clk; reg reset;  wire [3:0] O;  tfto\_decade\_counter connect(clk, reset, O);  initial begin  clk = 0; reset = 0;  end;  always clk = #10 ~clk;  always @(posedge clk) begin  end  endmodule |

**-Simulation 출력 결과 비교**

**A picture containing diagram

Description automatically generated**

모든 가능한 입력 값에 대한 출력값을 확인하기 위하여 Simulation을 시행한 결과 의도한 진리표와 모든 결과가 동일함을 확인하였다.

**-출력 예시**

비대면 수업으로 인하여 실제 FPGA로 부터 입력을 받지 못하였으나, 기존 실험때와 마찬가지로CLK 비트를 스위치로, 각 비트를 순서대로 LED에 불이 들어오는 것으로서 1로 가정하면, 스위치가 한번 왔다 갔다 할 때마다 0000부터 1111까지 2421 코드로 0부터 9까지 바뀌며 다시 0000으로 돌아오며 순환하는 것을 LED로서 확인할 수 있을 것이다. 이는 위의 Simulation 결과를 통하여 확인할 수 있다.

**-구현된 Schematic**

Diagram, timeline

Description automatically generated

**4. 결과 검토 및 논의 사항.**

이번주 실험에서는 2비트 카운터, 4비트 Decade 카운터와 이를 2421 코드로 변환하여 출력하는 카운터를 구현하였다. 이때 11주차에 구현한 JK Flip-Flop과 10주차에서 구현한 4비트 Full Adder가 함께 활용되었다. State Table을 작성한 후 조건문을 사용하여 모든 가능한 상황에 대한 회로의 행동을 제어하는 방식으로 Verilog 코드를 작성하였다. 이후 모든 가능한 입력에 대해 simulate 하여 기대되는 결과가 나타나는지 확인하였다. 그 결과 모든 디자인이 의도한 State Table와 다르지 않고 동일하게 결과가 출력됨을 확인하였다.

**5. 추가 이론 조사 및 작성.**

이번 주에 구현한 카운터는 실제 디지털 시계를 구현하는데 가장 바탕이 되는 개념이다. 이를 구현하는 방식에 대해 간단하게 살펴보자.

우선 디지털 시계를 구현하기 위해서는 아래와 같은 과정을 거쳐야 한다.

발진회로 -> 분주회로 -> 카운터회로 -> 디코더회로 -> 표시회로

각 요소의 특성과 역할에 대해 살피면 다음과 같다.

* 발진회로: 디지털 시계에서 안정적인 클록을 제공할 목적을 설계되는 회로로서, 크게 세가지 방법을 이용한다. (1) 가정용 220v 전원의 60Hz 주파수, (2) CR 발진 회로, (3) 수정 발진자를 사용한다.
* 분주회로: 발진회로로 부터 얻어진 구형파를 이용하여 디지털 시계의 기본 단위인 1초를 나타내기 위한 1Hz 주파수를 얻는 회로이다.
* 카운터회로: 분주회로부터 계산된 1Hz를 바탕으로 60초, 60분, 12시간(또는 24시간) 주기로 상태를 변환해주는 회로이다. 즉, 1Hz의 발생 횟수를 각 단위별로 센다.
* 디코더회로: 카운터회로로부터 계산된 이진수를 표시회로에 맞는 코드로 변환하는 회로이다.
* 표시회로: 코드를 바탕으로 7-segment display에 구현하여, 실제 사람이 읽을 수 있는 숫자로 보여주는 회로이다.

**6. 참고 문헌**

강석태, “Verilog HDL Summary”, <http://vlsi.hongik.ac.kr/lecture/%EC%8B%A4%ED%97%98/Verilog_Summary.pdf>.

박재동, “[Verilog 예제] CLA (Carry Look Ahead) 모듈”, <http://egloos.zum.com/rabe/v/1285020>.

한국기술교육대학교 전기전자통신공학부, “디지털시스템 설계 및 실습” Verilog HDL 문법 강의자료, <https://cms3.koreatech.ac.kr/sites/yjjang/down/dsys11/M01_VerilogHDL01.pdf>.