REPORT

전자공학도의 윤리 강령 (IEEE Code of Ethics)

(출처: http://www.ieee.org)

나는 전자공학도로서, 전자공학이 전 세계 인류의 삶에 끼치는 심대한 영향을 인식하여 우리의 직업, 동료와 사회에 대한 나의 의무를 짐에 있어 최고의 윤리적, 전문적 행위를 수행할 것을 다짐하면서, 다음에 동의한다.

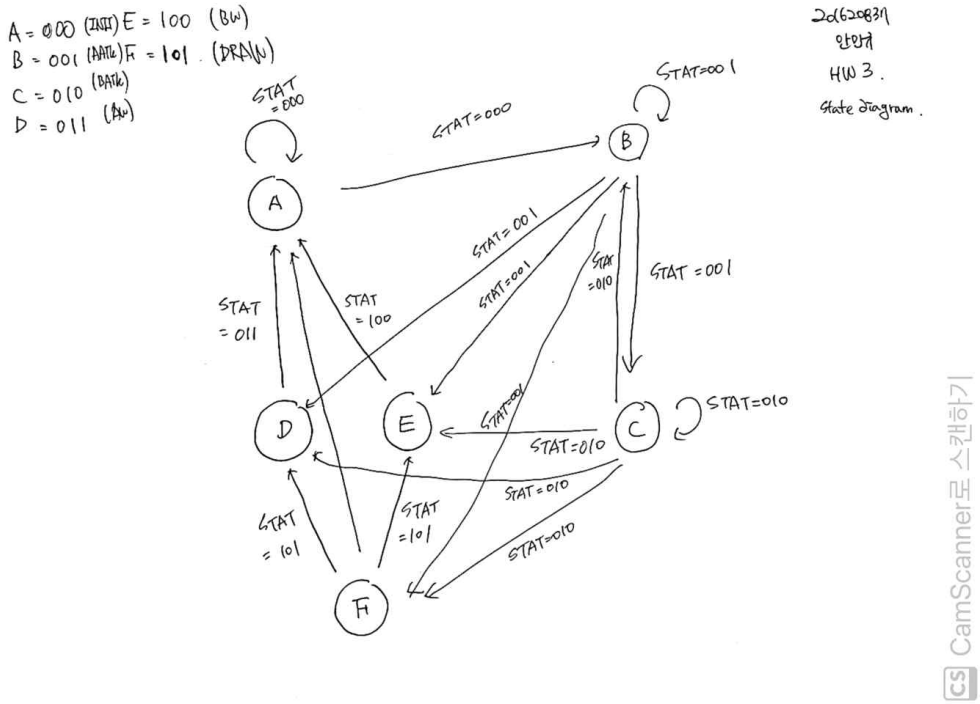
1. **공중의 안전, 건강 복리에 대한 책임**: 공중의 안전, 건강, 복리에 부합하는 결정을 할 책임을 질 것이며, 공중 또는 환경을 위협할 수 있는 요인을 신속히 공개한다.
2. **지위 남용 배제**: 실존하거나 예기되는 이해 상충을 가능한 한 피하며, 실제로 이해가 상충할 때에는 이를 이해 관련 당사자에게 알린다. (이해 상충: conflicts of interest, 공적인 지위를 사적 이익에 남용할 가능성)
3. **정직성**: 청구 또는 견적을 함에 있어 입수 가능한 자료에 근거하여 정직하고 현실적으로 한다.
4. **뇌물 수수 금지**: 어떠한 형태의 뇌물도 거절한다.
5. **기술의 영향력 이해**: 기술과 기술의 적절한 응용 및 잠재적 영향에 대한 이해를 높인다.
6. **자기계발 및 책무성**: 기술적 능력을 유지, 증진하며, 훈련 또는 경험을 통하여 자격이 있는 경우이거나 관련 한계를 전부 밝힌 뒤에만 타인을 위한 기술 업무를 수행한다.
7. **엔지니어로서의 자세**: 기술상의 업무에 대한 솔직한 비평을 구하고, 수용하고, 제공하며, 오류를 인정하고 수정하며, 타인의 기여를 적절히 인정한다.
8. **차별 안하기**: 인종, 종교, 성별, 장애, 연령, 출신국 등의 요인에 관계없이 모든 사람을 공평하게 대한다.
9. **도덕성**: 허위 또는 악의적인 행위로 타인, 타인의 재산, 명예, 또는 취업에 해를 끼치지 않는다.
10. **동료애**: 동료와 협력자가 전문분야에서 발전하도록 도우며, 이 윤리 헌장을 준수하도록 지원한다.

위 IEEE 윤리헌장 정신에 입각하여 report를 작성하였음을 서약합니다.

학 부: 전자공학부
마감일: 2021. 06. 11.
과목명: 디지털시스템설계
교수명: 양회석 교수님
학 번: 201620837
성 명: 안민규

Homework 3. Finite State Machines

1. Overview of design



<state diagram>

Main FSM인 RSP는 synchronized이고 FSM이므로 State memory 부분과 Next state부분, Output logic의 3부분으로 나누어 구현하였다. 기본적으로 RST가 들어오면 S_INIT으로 초기화되게 하였고, 이 때 먼저 들어온 입력값을 저장하는 a_prev와 b_prev를 2'bxx로 초기화시킨다. 이후 다음에 들어오는 Next 상태가 게임이 종료된건지 아닌지에 따라 a_prev, b_prev에 넣는 값을 달리하여 State memory를 end시켰다.

Next state logic을 구성할 때는 S_reg와 들어오는 입력이 변할 때 실행되게 설정하였다. 이후 S_reg의 값에 따라 case문을 이용해 next state를 설정하였다.

현재 상태가 INIT일 때는 아직 게임이 시작하지 못한 상태이므로 AS, BS와 관련된 제어문은 없고, 오로지 들어오는 입력에 따라서만 next state가 결정되게 구현하였다. 정해진 rule 2, 3에 맞추어 INVALID가 들어올 때를 나누었고, A와 B 모두 올바른 사인이 들어왔을 때를 기준으로 남은 부분을 제어하였다. 우선 게임 시작이므로 둘 중에 한 명이 이기기 전이므로 같은 값이 나왔다면 INIT을 next state로 결정하였고, 서로 다를 때 서로의 사인에 따라 승자를 결정하는 방식을 선정하였다.

INIT 상태에서 A가 이겨 현재 상태가 AATK이 됐다면 AS와 BS를 확인해 이를 통해 게임의 결과를 결정하였다. 이후 AS, BS가 모두 0일 때 INVALID 사인이 들어왔는지를 판별한 후 A, B 사인에 따라 next state를 결정하였다. BATK의 경우 또한 똑같이 결정하였다.

A가 이기거나 B가 이겼을 때의 상태라면 다음에 게임을 다시 새로 시작해야 하므로 next state에는 INIT을 넣었다.

A와 B가 비겼다면 공격권이 정해진 상태라면 같은 값이면 비기는 것이 아니라 공격권을 가진 쪽이 승리하고, 이긴 상태에서는 DRAW가 나오지 않기 때문에 이는 둘 다 INVALID 사인을 보였을 때만 나온다. 따라서 DRAW 상태에서는 INIT 상태 혹은 INVALID가 또 들어왔을 때에 맞추어 승자를 결정한다.

마지막 Output logic의 경우 현재 상태에 따라 STAT에 3'b000부터 순차적으로 1씩 증가시키며 할당하였다. rule 2와 rule 3가 가장 높은 priority를 가지므로 INVALID 사인이 들어왔는지를 먼저 판별한 후 들어오지 않았을 때 작동하도록 설계하였다.

rule 4, 5를 위해 Series를 이용해 AS, BS를 정해준다. Series의 경우 CLK과 RST이 asynchronous이므로 CLK이랑 RST이 다른 순간에 입력이 들어올 수 있게 된다. 따라서 RST가 들어올 때 모든 값들이 초기화되도록 설정을 하였고, RST 신호가 들어오지 않았을 때를 구현하였다. AS, BS의 경우 같은 값이 3번 나왔을 때 1이 ASSERTED 된다. 따라서 cnt를 이용했고 같은 값이 나올 때마다 cnt를 1씩 증가시키는 알고리즘을 구현하였다. 우선적으로 INVALID 사인이 들어오면 cnt나 S에 상관없이 게임이 종료되므로 AS, BS를 초기화시킨다. INVALID 사인이 들어오지 않았다면 현재 cnt 값에 따라 나누었고, 들어온 입력과 이전 입력 prev_v를 비교하여 cnt를 증감 혹은 유지 혹은 초기화를 진행하였고, s를 0 또는 1의 값으로 변경하였다. 이 Series 내부에서 같은 값이 3번이 나오는 것을 다 감지하였고, 이를 통해 RSP에 AS 혹은 BS 값이 ASSERTED 되었을 때 전 값과 같다면 바로 게임이 종료되게끔 설정하였다. 이로 인해 강의노트에 제시된 예상 결과사진과 약간 다르게 나타났으나 모든 결과를 확인할 수 있었다.

Score를 계산하고 SC_LDISP, SC_RDISP에 점수를 출력하기 위해 점수 계산 모듈을 생성하였다. 이는 RST 신호가 들어오면 점수를 초기화하고, 들어오지 않았다면 RSP로부터 나오는 STAT에 맞추어 점수를 올리거나 유지하였다. STAT은 현재상태 S_reg에 따라 결정되므로 next state보다 조금 늦게 점수가 올라가는 결과가 나타난다. 또한 주어진 조건대로 점수가 9점보다 높아지면 0으로 초기화되도록 구현하였다.

Display decoder 모듈은 조합회로로써 CLK나 RST이 들어오지 않는다. 따라서 들어오는 인풋에 따라 always 구문이 실행되도록 구현하였고, 앞서 제시된 function들을 이용하였다. 게임이 시작한 순간이나 게임이 진행중에는 A, B가 낸 사인이 DISP에 나타나야하므로 앞서 제시된 sign_dec 함수를 이용하여 입력을 넣고 이를 LDISP와 RDISP에 연결하였다. 그 후 게임이 종료된 순간 (이기거나 비기거나)에는 주어진 조건에 맞추어 ->,-<,--를 출력하게 따로 설정하였다. Score의 경우 항상 출력이 되어야하므로 number_dec 함수를 이용하여 ASC, BSC를 넣어 SC_LDISP, SC_RDISP에 연결하였다.

HW3 모듈에서는 앞서 생성한 모듈들을 차례차례 불러왔고 각각 모듈에서 나오는 아웃과 인풋을 이어주기 위해 wire를 임의로 선언하여 사용하였다.

TestBench에서는 모든 rule을 확인할 수 있게끔 여러 인풋을 상황에 맞추어 선언하였다.

2. Verilog code

2-1) RSP (Implementation 1)

```
module Vr_HW3_RSP (CLK, RST, AIN, BIN, AS, BS, STAT);
  /* I/O interface */
  input CLK, RST;
  input [0:1] AIN, BIN;
  input AS, BS;
  output [2:0] STAT;
```

```
  reg [2:0] S_reg, S_next; /* state memory and next state signals */
  reg [0:1] a_prev, b_prev; /* previous a, b values */
  reg [2:0] S_STAT;

  /* state declarations */
  parameter [2:0] S_INIT = 3'b000,
                  S_AATK = 3'b001,
                  S_BATK = 3'b010,
                  S_AW = 3'b011,
                  S_BW = 3'b100,
                  S_DRAW = 3'b101;

  /* RSP */
  parameter [0:1] ROCK=2'b00, SCISSORS=2'b01, PAPER=2'b10, INVALID=2'b11;


  /* Implementation 1: main FSM */
  always @ (posedge CLK) begin //State memory
   if (RST) begin S_reg <= S_INIT; a_prev <= 2'bxx; b_prev <= 2'bxx; end //RST 들어오면 초기화
   else begin //RST 들어오지 않으면
        if (S_next == S_AW || S_next == S_BW) begin //다음 상태값이 AW 혹은 BW이라면
         S_reg <= S_next; a_prev <= 2'bxx; b_prev <= 2'bxx; //prev값을 초기화
        end
        else begin //둘 중에 한명도 이기지 않은 상황이라면
         S_reg <= S_next; a_prev <= AIN; b_prev <= BIN; //prev값에 인풋과 S_reg에 S_next 저장
        end
   end
  end

  always @ (S_reg, AIN, BIN, AS, BS) begin //Next-state logic
    case (S_reg) //S_reg에 따라서
        S_INIT : if (AIN == INVALID && BIN != INVALID) S_next = S_BW; //A가 INVALID라면 BW
                 else if (AIN != INVALID && BIN == INVALID) S_next = S_AW; //B가 INVALID라면
AW
                 else if (AIN == INVALID && BIN == INVALID) S_next = S_DRAW; //둘 다 INVALID
라면 DRAW
                 else if (AIN != INVALID && BIN != INVALID) begin //둘 다 올바른 사인이라면
                 if (AIN == BIN) S_next = S_INIT; //둘이 같다면 INIT -> 한 명이 공격할 때까지 반복
                 if (AIN == ROCK && BIN == SCISSORS) S_next = S_AATK;
                 else if (AIN == SCISSORS && BIN == PAPER) S_next = S_AATK;
                 else if (AIN == PAPER && BIN == ROCK) S_next = S_AATK; //A가 이기므로 AATK
                 else if (AIN == ROCK && BIN == PAPER) S_next = S_BATK;
                 else if (AIN == SCISSORS && BIN == ROCK) S_next = S_BATK;
                 else if (AIN == PAPER && BIN == SCISSORS) S_next = S_BATK; //B가 이기므로 BA
```

```
TK
                end

        S_AATK : if (AS) begin //AS가 1이면
                  if (BS) begin //BS도 1이면
                   if (AIN == a_prev && BIN == b_prev) //둘 다 같은 값이 들어왔으므로
                    S_next = S_DRAW; //DRAW

                   else if (AIN == a_prev && BIN != b_prev) //A만 같은 값을 내므로
                    S_next = S_BW; //BW

                   else if (AIN != a_prev && BIN == b_prev) //B만 같은 값을 내므로
                    S_next = S_AW; //AW
                  end

                 else begin //BS는 0이면
                  if (AIN == a_prev) S_next = S_BW; //A는 같은 값을 내므로 BW
                 end
                end

                else if (BS) begin //AS는 0이고 BS가 1이면
                 if (BIN == b_prev) S_next = S_AW; //B는 같은 값을 내므로 AW
                end

                else begin //AS, BS 모두 0일 때
                 if (AIN == INVALID && BIN != INVALID) S_next = S_BW; //A는 INVALID->BW
                 else if (AIN != INVALID && BIN == INVALID) S_next = S_AW; //B는 INVALID ->
AW
                 else if (AIN == INVALID && BIN == INVALID) S_next = S_DRAW; //둘 다 INVALI
D이므로 DRAW
                 else if (AIN != INVALID && BIN != INVALID) begin //둘 다 올바른 사인이라면
                   if (AIN == BIN) S_next = S_AW; // 둘이 같다면 AW

                   else if (AIN == ROCK && BIN == SCISSORS) S_next = S_AATK;

                   else if (AIN == SCISSORS && BIN == PAPER) S_next = S_AATK;

                   else if (AIN == PAPER && BIN == ROCK) S_next = S_AATK; //또 A가 이기므로
AATK

                   else if (AIN == ROCK && BIN == PAPER) S_next = S_BATK;

                   else if (AIN == SCISSORS && BIN == ROCK) S_next = S_BATK;
```

```
                    else if (AIN == PAPER && BIN == SCISSORS) S_next = S_BATK; //B로 공격권
넘어감
                   end
                  end

        S_BATK : if (AS) begin //AS가 1이면
                   if (BS) begin //BS도 1이면
                    if (AIN == a_prev && BIN == b_prev) //둘 다 같은 값이 들어왔으므로
                     S_next = S_DRAW; //DRAW
                    else if (AIN == a_prev && BIN != b_prev) //A만 같은 값을 내므로
                     S_next = S_BW; //BW
                    else if (AIN != a_prev && BIN == b_prev) //B만 같은 값을 내므로
                     S_next = S_AW; //AW
                   end

                   else begin //BS는 0이면
                    if (AIN == a_prev) //A는 같은 값을 내므로 BW
                     S_next = S_BW;
                   end
                  end

                  else if (BS) begin  //AS는 0이고 BS가 1이면
                  if (BIN == b_prev) //B는 같은 값을 내므로 AW
                   S_next = S_AW;
                  end

                  else begin //AS, BS 모두 0일 때
                   if (AIN == INVALID && BIN != INVALID) S_next = S_BW; //A는 INVALID->BW
                   else if (AIN != INVALID && BIN == INVALID) S_next = S_AW; //B는 INVALID ->
AW
                   else if (AIN == INVALID && BIN == INVALID) S_next = S_DRAW; //둘 다 INVALI
D이므로 DRAW
                   else if (AIN != INVALID && BIN != INVALID) begin //둘 다 올바른 사인이라면
                    if (AIN == BIN) S_next = S_BW; // 둘이 같다면 BW

                    else if (AIN == ROCK && BIN == SCISSORS) S_next = S_AATK;

                    else if (AIN == SCISSORS && BIN == PAPER) S_next = S_AATK;

                    else if (AIN == PAPER && BIN == ROCK) S_next = S_AATK; //A가 이기므로 A
로 공격권 넘어감

                    else if (AIN == ROCK && BIN == PAPER) S_next = S_BATK;
```

```
                    else if (AIN == SCISSORS && BIN == ROCK) S_next = S_BATK;

                    else if (AIN == PAPER && BIN == SCISSORS) S_next = S_BATK; //B가 또 이겼
으므로 BATK
                  end
                 end

        S_AW : S_next = S_INIT; //이겼으면 게임 종료하고 INIT

        S_BW : S_next = S_INIT; //이겼으면 게임 종료하고 INIT


        S_DRAW : if (AIN == INVALID && BIN != INVALID) S_next = S_BW; //A가 INVALID이므로
BW
                 else if (AIN != INVALID && BIN == INVALID) S_next = S_AW; //B가 INVALID이므
로 AW
                 else if (AIN == INVALID && BIN == INVALID) S_next = S_INIT; //둘 다 INVALID이
므로 DRAW
                  else if (AIN != INVALID && BIN != INVALID) S_next = S_INIT; //둘 다 올바른 사인
이므로 INIT
    endcase
  end

  always @ (S_reg) begin //Output logic
    case (S_reg)
        S_INIT : S_STAT = 3'b000; //S_reg가 INIT이라면 S_STAT에 3'b000
        S_AATK : S_STAT = 3'b001; //S_reg가 AATK이라면 S_STAT에 3'b001
        S_BATK : S_STAT = 3'b010; //S_reg가 BATK이라면 S_STAT에 3'b010
        S_AW : S_STAT = 3'b011; //S_reg가 AW이라면 S_STAT에 3'b011
        S_BW : S_STAT = 3'b100; //S_reg가 BW이라면 S_STAT에 3'b100
        S_DRAW : S_STAT = 3'b101; //S_reg가 DRAW이라면 S_STAT에 3'b101
        default : S_STAT = 3'b000; //default라면 다시 INIT -> S_STAT에 3'b000
    endcase
  end

  assign STAT = S_STAT; //output STAT에 S_STAT 연결
endmodule
```

2-2) Series Detector (Implementation 2)

```
module Vr_HW3_Series (CLK, RST, V, S);
  /* I/O interface */
  input CLK, RST;
  input [0:1] V;
  output S;
```

```
  /* parameterizable length threshold */
  parameter integer len = 2;

  integer cnt = 32'b0; //같은 값이 하나씩 나올 때마다 1씩 증가하는 cnt값
  reg [0:1] prev_v;
  reg s = 0; //output에 할당하기 위해 선언한 reg


  /* Implementation 2: Series detector */

  always @ (posedge CLK or posedge RST) begin //asynchronous
    if (RST) begin //RST이 들어오면 초기화
        prev_v <= 2'bxx; //이미 저장된 값 2'bxx로 초기화
        cnt <= 32'b0;    //cnt 값을 초기화
        s <= 0;          //s 0으로 초기화
    end

    else begin //RST 신호가 안들어오면
         if (V == 2'b11) begin s = 0; cnt <= 32'b0; prev_v <= 2'bxx; end //INVALID가 들어오면,
경기가 종료되므로 RST처럼 작동함

         else begin //INVALID가 들어오지 않았다면
          if (cnt == 32'b010) begin s = 1; cnt <= 32'b0; prev_v <= 2'bxx; end //cnt가 2가 되면
(같은 인풋 3번 나옴) 지정한 threshold에 도달하므로 s를 1, cnt는 초기화, pre_v는 초기화

          else begin //cnt가 아직 2가 아니라면
           if(cnt == 32'b01) begin //같은 인풋이 나온게 두 번째라면
            if (V == prev_v) begin cnt <= cnt + 32'b01; prev_v <= V; s = 1; end //이번에도 전
인풋과 같다면 cnt를 1 늘리고 prev_V에 이번 값 저장, s는 1로 바꿈
             else begin prev_v <= V; cnt <= cnt - 32'b01; s = 0; end //전 인풋과 같지 않다면 s=
0, cnt는 1 줄이고 prev_v에 현재 입력 V 저장
           end

           else begin //같은 인풋이 한 번 나왔다면
            if (V == prev_v) begin cnt <= cnt + 32'b01; prev_v <= V; s = 0; end //같은 인풋이
나왔다면 cnt를 1 증가시키고 s=0, prev_v는 V를 저장한다.
             else begin prev_v <= V; cnt <= cnt; s = 0; end //같지 않다면 cnt와 s는 본래 값을 유
지하고 prev_v에 V 저장
           end
         end
         end
      end
    end
```

```
  assign S = s; //output S에 s 연결
endmodule
```

2-3) Score (Implementation 3)

```
module Vr_HW3_Score (CLK, RST, STAT, ASC, BSC);
  /* I/O interface */
  input CLK, RST;
  input [2:0] STAT;
  output [3:0] ASC, BSC;

  /* state declarations */
  parameter [2:0] S_INIT = 3'b000,
                  S_AATK = 3'b001,
                  S_BATK = 3'b010,
                  S_AW = 3'b011,
                  S_BW = 3'b100,
                  S_DRAW = 3'b101;

  reg [3:0] a_score, b_score;
  /* Implementation 3: Score Calculator */
  always @ (posedge CLK) begin //Synchronized
     if (RST) begin
        a_score <= 4'b0000;
        b_score <= 4'b0000;
     end //RST 신호 들어오면 score를 0으로 초기화

     else begin
      case (STAT) //STAT에 따라 결정
           3'b000, 3'b001, 3'b010, 3'b101 : begin a_score = a_score; b_score = b_score; end //INIT,
AATK, BATK, DRAW일 때는 점수 변동 X
           3'b011 : a_score = a_score + 4'b0001; //AW
           3'b100 : b_score = b_score + 4'b0001; //BW
      endcase

      if(a_score > 4'b0101) a_score = 4'b0000; //현재 점수가 9점보다 높아지면 0으로 초기화

      if(b_score > 4'b0101) b_score = 4'b0000; //현재 점수가 9점보다 높아지면 0으로 초기화
     end
  end

  assign ASC = a_score; //a_score를 ASC에 연결
  assign BSC = b_score; //b_score를 BSC에 연결
endmodule
```

2-4) Display Decoder (Implementation 4)

```
module Vr_HW3_Disp_Decoder (AIN, BIN, STAT, ASC, BSC, LDISP, RDISP, SC_LDISP, SC_RDISP);
  /* I/O interface */
  input [0:1] AIN, BIN;
  input [2:0] STAT;
  input [3:0] ASC, BSC;
  output reg [0:12] LDISP, RDISP;
  output reg [0:12] SC_LDISP, SC_RDISP;

  /* state declarations */
  parameter [2:0] S_INIT = 3'b000,
                  S_AATK = 3'b001,
                  S_BATK = 3'b010,
                  S_AW = 3'b011,
                  S_BW = 3'b100,
                  S_DRAW = 3'b101;
  parameter [0:1] ROCK=2'b00, SCISSORS=2'b01, PAPER=2'b10, INVALID=2'b11;

  function [0:12] number_dec;
    input [3:0] in_val;
    if(in_val==4'b0000) number_dec = 13'b1111110000000; // 0
    else if(in_val==4'b0001) number_dec = 13'b0110000000000; // 1
    else if(in_val==4'b0010) number_dec = 13'b1101101000000; // 2
    else if(in_val==4'b0011) number_dec = 13'b1111001000000; // 3
    else if(in_val==4'b0100) number_dec = 13'b0110011000000; // 4
    else if(in_val==4'b0101) number_dec = 13'b1011011000000; // 5
    else if(in_val==4'b0110) number_dec = 13'b1011111000000; // 6
    else if(in_val==4'b0111) number_dec = 13'b1110010000000; // 7
    else if(in_val==4'b1000) number_dec = 13'b1111111000000; // 8
    else if(in_val==4'b1001) number_dec = 13'b1111011000000; // 9
    else number_dec = 13'b0000000000000;
  endfunction

  function [0:12] sign_dec;
    input [0:1] in_val;
    if(in_val==2'b00) sign_dec = 13'b0011101000111; // rock
    else if(in_val==2'b01) sign_dec = 13'b0011101101010; // scissors
    else if(in_val==2'b10) sign_dec = 13'b0111111010000; // paper
    else if(in_val==2'b11) sign_dec = 13'b0000000101101; // invalid
    else sign_dec = 13'b0000000000000;
  endfunction

  function isRock;
```

```
    input [0:1] in_val;
    if (in_val == ROCK) isRock = 1'b1;
    else isRock = 1'b0;
  endfunction

  function isScissors;
    input [0:1] in_val;
    if (in_val == SCISSORS) isScissors = 1'b1;
    else isScissors = 1'b0;
  endfunction

  function isPaper;
    input [0:1] in_val;
    if (in_val == PAPER) isPaper = 1'b1;
    else isPaper = 1'b0;
  endfunction

  function isInvalid;
    input [0:1] in_val;
    if (in_val == INVALID) isInvalid = 1'b1;
    else isInvalid = 1'b0;
  endfunction

  function RCPAWin;
    input [0:1] Aval; input [0:1] Bval;
    if (isRock(Aval) && isScissors(Bval)) RCPAWin = 1'b1;
    else if (isScissors(Aval) && isPaper(Bval)) RCPAWin = 1'b1;
    else if (isPaper(Aval) && isRock(Bval)) RCPAWin = 1'b1;
    else RCPAWin = 1'b0;
  endfunction

  function RCPBWin;
    input [0:1] Aval; input [0:1] Bval;
    if (isRock(Bval) && isScissors(Aval)) RCPBWin = 1'b1;
    else if (isScissors(Bval) && isPaper(Aval)) RCPBWin = 1'b1;
    else if (isPaper(Bval) && isRock(Aval)) RCPBWin = 1'b1;
    else RCPBWin = 1'b0;
  endfunction

  /* Implementation 4: display decoder */
  always @ (AIN or BIN or STAT or ASC or BSC) begin
   if (STAT != 3'b011 && STAT != 3'b100 && STAT != 3'b101) begin
    LDISP = sign_dec(AIN);
    RDISP = sign_dec(BIN);
```

```
    end

    else if (STAT == 3'b011) begin//AW
     LDISP = 13'b0000001000000; // LDISP 에는 -
     RDISP = 13'b0000000100100; // RDISP 에는 >
    end

    else if (STAT == 3'b100) begin//BW
     LDISP = 13'b0000000001001; //LDISP 에는 <
     RDISP = 13'b0000001000000; //RDISP 에는 -
    end

    else if (STAT == 3'b101) begin//DRAW
     LDISP = 13'b0000001000000;
     RDISP = 13'b0000001000000; //LDISP, RDISP 모두 -
    end

   SC_LDISP = number_dec(ASC); //ASC를 앞서 선언된 function에 넣어 값 도출
   SC_RDISP = number_dec(BSC); //BSC를 앞서 선언된 function에 넣어 값 도출

   end
endmodule
```

2-5) HW3 (Implementation 5)

```
module Vr_HW3 (CLK, RST, AIN, BIN, LDISP, RDISP, SC_LDISP, SC_RDISP);
   /* I/O interface */
   input CLK, RST;         //CLK, RST
   input [0:1] AIN, BIN; //A와 B의 인풋
   output [0:12] LDISP, RDISP; //A와 B의 사인 및 이겼을 때 결과 나타내는 OUT
   output [0:12] SC_LDISP, SC_RDISP; //A와 B의 현재 점수를 나타내는 OUT

   /* Implementation 5: top-level module */
   /* see page 10 for its structure */
   /* instantiate one main FSM, two series detectors, one score calculator,
       and one display decoder; and properly connect them with each other */
   wire AS;
   wire BS; //series 값 나타내기 위한 wire
   wire [2:0] STAT; //STAT
   wire [3:0] ASC; //A의 SCORE
   wire [3:0] BSC; //B의 SCORE

   Vr_HW3_Series A_Series (CLK, RST, AIN, AS); //1 series detectors - A
   Vr_HW3_Series B_Series (CLK, RST, BIN, BS); //2 series detectors - B
```

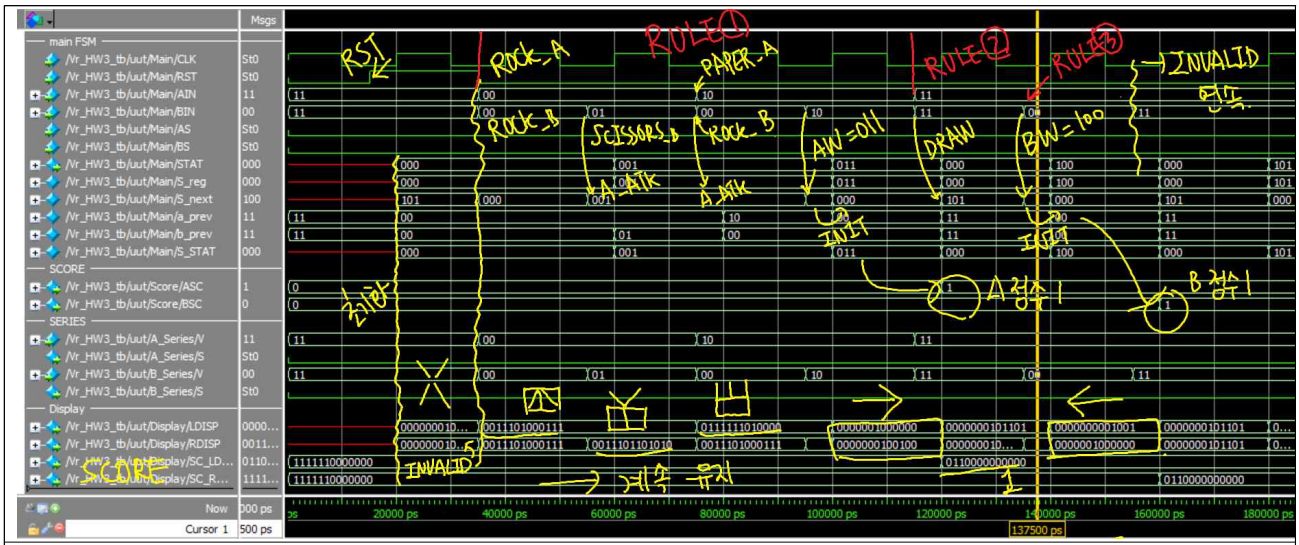
```
    Vr_HW3_RSP Main (CLK, RST, AIN, BIN, AS, BS, STAT); //Main FSM

    Vr_HW3_Score Score (CLK, RST, STAT, ASC, BSC); //Score calculator

    Vr_HW3_Disp_Decoder Display (AIN, BIN, STAT, ASC, BSC, LDISP, RDISP, SC_LDISP, SC_RDISP); //Di
splay decoder
endmodule
```

2-6) HW3_tb (Implementation 6)

```
`timescale 1 ns / 100 ps
module Vr_HW3_tb ();

    reg sig_clk, sig_rst;
    reg [0:1] sig_ain, sig_bin;
    wire [0:12] sig_ldisp, sig_rdisp, sig_sc_ldisp, sig_sc_rdisp;

    parameter [0:1] ROCK=2'b00, SCISSORS=2'b01, PAPER=2'b10, INVALID=2'b11;



    /* main top-level file */
    Vr_HW3 uut (.CLK(sig_clk), .RST(sig_rst), .AIN(sig_ain), .BIN(sig_bin), .LDISP(sig_ldisp), .RDISP(sig_rdis
p), .SC_LDISP(sig_sc_ldisp), .SC_RDISP(sig_sc_rdisp));

    /* display module */
    Vr_HW3_Disp u_disp (.CLK(sig_clk), .RST(sig_rst), .LDISP(sig_ldisp), .RDISP(sig_rdisp), .SC_LDISP(sig_sc
_ldisp), .SC_RDISP(sig_sc_rdisp) );

    /* clock/reset generation */

    always begin
        sig_clk = 1; #10;
        sig_clk = 0; #10;
    end

    always begin
        sig_rst = 0; #15;
        sig_rst = 1; #20;
        sig_rst = 0; #10000;
    end

    /* game playing scenarios */
    always begin
        sig_ain = INVALID; sig_bin = INVALID; #35;
        sig_ain = ROCK; sig_bin = ROCK; #20;
```

```
    sig_ain = ROCK; sig_bin = SCISSORS; #20;
    sig_ain = PAPER; sig_bin = ROCK; #20;
    sig_ain = PAPER; sig_bin = PAPER; #20; // A WIN (RULE 1)
    sig_ain = INVALID; sig_bin = INVALID; #20; //  A is expected to win the game here


    sig_ain = INVALID; sig_bin = ROCK; #20; // B is expected to win the game (RULE 3)
    sig_ain = INVALID; sig_bin = INVALID; #20;

    sig_ain = INVALID; sig_bin = INVALID; #20; // They are expected to draw (RULE 2)
    sig_ain = INVALID; sig_bin = INVALID; #20;


    sig_ain = PAPER; sig_bin = SCISSORS; #20;
    sig_ain = ROCK; sig_bin = SCISSORS; #20;
    sig_ain = ROCK; sig_bin = PAPER; #20;
    sig_ain = ROCK; sig_bin = SCISSORS; #20;
    sig_ain = INVALID; sig_bin = INVALID; #20; //DRAW

    /* Implementation 6: complete the test bench to check more scenarios */
    sig_ain = PAPER; sig_bin = PAPER; #20;
    sig_ain = PAPER; sig_bin = PAPER; #20;
    sig_ain = PAPER; sig_bin = PAPER; #20; //ATK X - SAME 3 -> INIT

    sig_ain = PAPER; sig_bin = SCISSORS; #20;
    sig_ain = PAPER; sig_bin = SCISSORS; #20;
    sig_ain = PAPER; sig_bin = ROCK; #20; //BATK 2, AATK 1 - A SAME 3 -> B WIN (RULE 4)

    sig_ain = PAPER; sig_bin = ROCK; #20;
    sig_ain = PAPER; sig_bin = ROCK; #20;
    sig_ain = SCISSORS; sig_bin = ROCK; #20; //AATK 2, BATK 1 - B SAME 3 -> A WIN (RULE 4)

    sig_ain = PAPER; sig_bin = ROCK; #20;
    sig_ain = SCISSORS; sig_bin = PAPER; #20;
    sig_ain = SCISSORS; sig_bin = PAPER; #20;
    sig_ain = SCISSORS; sig_bin = PAPER; #20; //AATK 1 - A, B SAME 3 -> DRAW (RULE 5)

  end

endmodule
```

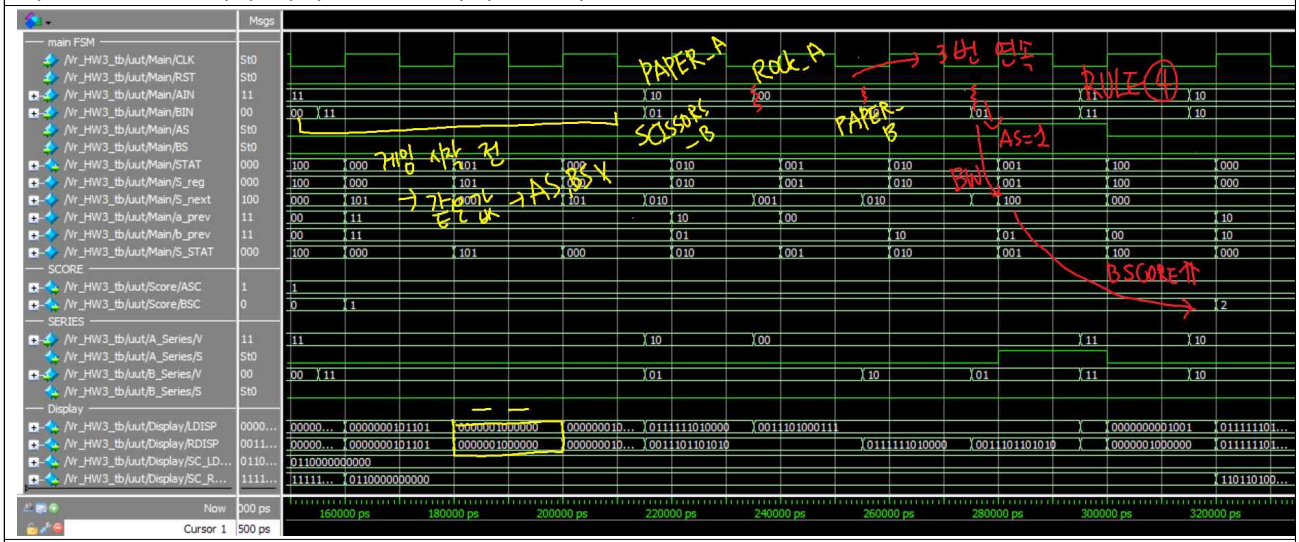
3. Screenshots of the simulation results



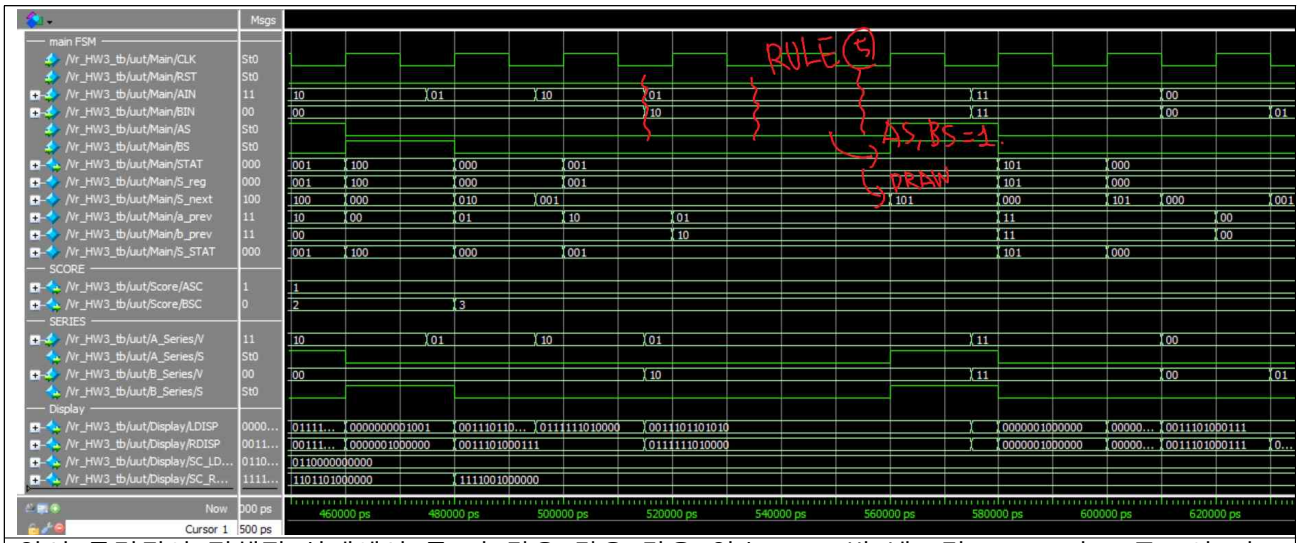
RST가 들어왔을 때는 모든 것들이 초기화가 된다. SCORE는 항상 출력이 되고 CLK, RST가 들어오지 않는 조합회로이므로 시작부터 0(1111110000000)을 출력하고 있다. 이후에도 ASC, BSC가 들어오지 않으면 계속해서 값을 유지한다.

첫 CLK 틱에 두 인풋이 모두 INVALID이므로 LDISP, RDISP는 INVALID X를 표시한다. 이후 A, B 둘다 주먹을 냈을 때는 둘 다 주먹을 표시한다. 이때까지도 아직 게임이 시작이 안 되었으므로 계속 INIT 상태이다. 이후 B가 가위를 냈을 때 A에게 공격권이 생기고 상태가 001이 되고 STAT이 001로 CLK 틱에 변한다. 이때 LDISP는 계속 주먹을 RDISP는 가위를 표시한다. 이후 게임이 진행되면서 A 공격권에서 둘 다 보자기를 냈을 때 게임이 AW가 되고 A에 점수를 1점 올려준다. 이후 SC_LDISP가 변한다. 또한 게임이 끝났으므로 INIT으로 바뀐다. 이를 통해 기본 묵찌빠 게임인 rule 1을 확인할 수 있다.

다음으로 둘 다 INVALID 값을 냈을 때는 DRAW로 상태가 바뀌고 이를 통해 RULE2를 확인할 수 있다. 다음에 A는 INVALID 값을 B는 INVALID가 아닌 값을 내어 B가 이기게된다. STAT은 BW이 되고 BSC는 1 증가한다. 이를 통해 RULE 3을 확인할 수 있다. 또한 A의 INVALID값이 계속해서 나오나 아무도 공격권을 얻은 상태가 아니므로 cnt 하지 않는다.



A는 보자기를, B는 가위를 내 B가 공격권을 가진 상태로 게임이 시작되는데 A가 바위를 3번 연속으로 낸다. 이 결과 AS에 1이 들어오고 다음 상태에서 BW으로 바뀌고 BSC가 1 증가한다. 이를 통해 RULE 4를 확인할 수 있다.



앞서 공격권이 정해진 상태에서 둘 다 같은 같은 값을 연속으로 3번 냈으면 AS, BS가 모두 1이 되고 누구의 승으로 결정되지 않고 DRAW 상태가 되어 게임이 종료되고 INIT 상태로 돌아간다. 이를 통해 RULE 5를 확인할 수 있다.