```
                        explosion1 : OUT std_logic;
                        explosion2 : OUT std_logic;
                        explosion5 : OUT std_logic;
                        explosion6 : OUT std_logic);
        END COMPONENT;
        SIGNAL buff_out1, buff_out2 : std_logic;
BEGIN
        -- Buffer input player 1
        BUFF1 : buffer_bomb PORT MAP(clk, reset, bombp1, buff_out1);
        -- Buffer input player 1
        BUFF2 : buffer_bomb PORT MAP(clk, reset, bombp2, buff_out2);
        -- Bomb handling entity
        MAIN  : bomb_handling PORT MAP(clk, reset, buff_out1, buff_out2, explosion1, explosio
        explosion <= explosion1 OR explosion2 OR explosion5 OR explosion6; -- 4 input OR gate
END behaviour;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY ff2 IS
        PORT (
                read : IN std_logic;
                FF2_reset : IN std_logic;
                clk : IN std_logic;
                FF2_read : OUT std_logic
        );
END ff2;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
ARCHITECTURE behaviour OF ff2 IS
        SIGNAL FF2, new_FF2 : unsigned(2 DOWNTO 0);
BEGIN
        lbl1 : PROCESS (clk)
        BEGIN
                -- If the reset becomes 1 the FF2 counter needs to be reset. If it is not, th
                IF (clk'EVENT AND clk = '1') THEN
                        IF FF2_reset = '1' THEN
                                FF2 <= (OTHERS => '0');
                        ELSE
                                FF2 <= new_FF2;
                        END IF;
                END IF;
        END PROCESS;
        lbl2 : PROCESS (read, FF2)
                -- Every time read becomes 1, a tile has been checked and thus need to be cou
        BEGIN
                IF (read = '1') THEN
                        new_FF2 <= FF2 + 1;
                ELSE
                        new_FF2 <= FF2;
                END IF;
                -- The last part checks for the right amount of times that different tiles ne
```

```
                IF (FF2 = "100") THEN
                        FF2_read <= '1';
                ELSE
                        FF2_read <= '0';
                END IF;
        END PROCESS;
END behaviour;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY hitscan IS
        PORT (
                X_b : IN std_logic_vector(3 DOWNTO 0);
                Y_b : IN std_logic_vector(3 DOWNTO 0);
                res : IN std_logic;
                clk : IN std_logic;
                explode : IN std_logic;
                FF2_read : IN std_logic;
                X_p1 : IN std_logic_vector(3 DOWNTO 0);
                Y_p1 : IN std_logic_vector(3 DOWNTO 0);
                X_p2 : IN std_logic_vector(3 DOWNTO 0);
                Y_p2 : IN std_logic_vector(3 DOWNTO 0);
                victoryv : OUT std_logic_vector(1 DOWNTO 0);
                read : OUT std_logic;
                FF2_reset : OUT std_logic;
                lethaltile_x : OUT std_logic_vector(3 DOWNTO 0);
                lethaltile_y : OUT std_logic_vector(3 DOWNTO 0)
        );
END hitscan;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
ARCHITECTURE behaviour OF hitscan IS
        TYPE hit_state IS (rest, undicided, horizontal, plusform, vertical, vert_wait, vert_o
        SIGNAL state, new_state : hit_state;
        SIGNAL FF1 : std_logic;
        SIGNAL coor_signed, coor_signed_x, coor_signed_y : signed(4 DOWNTO 0);
        SIGNAL coor_signed_p1 : signed(4 DOWNTO 0);
        SIGNAL coor_signed_p2 : signed(4 DOWNTO 0);
BEGIN
        lbl1 : PROCESS (clk)
        BEGIN
                IF (clk'EVENT AND clk = '1') THEN
                        IF res = '1' THEN
                                state <= rest;
                        ELSE
                                state <= new_state;
                        END IF;
                END IF;
        END PROCESS;
        lbl2 : PROCESS (Y_b, X_b, X_p1, Y_p1, X_p2, Y_p2, explode, state)
        BEGIN
```

```
24                  CASE state IS
25                          -- Everything needs to be reset in the rest state and checked if a bo
26                          WHEN rest =>
27                                  FF1 <= '0';
28                                  FF2_reset <= '1';
29                                  read <= '0';
30                                  victoryv <= "00";
31                                  lethaltile_x <= "0000";
32                                  lethaltile_y <= "0000";
33                                  coor_signed_x <= "0" & signed(X_b);
34                                  coor_signed_y <= "0" & signed(Y_b);
35                                  coor_signed_p1 <= "0" & signed(Y_p1);
36                                  coor_signed_p2 <= "0" & signed(Y_P2);
37                                  IF (explode = '1' AND X_b(0) = '1') THEN
38                                          new_state <= horizontal;
39                                  ELSIF (explode = '1') THEN
40                                          new_state <= undicided;
41                                  ELSE
42                                          new_state <= rest;
43                                  END IF;
44                                  -- If a bomb is not right above and below a wall, the system
45                          WHEN undicided =>
46                                  IF Y_b(0) = '1' THEN
47                                          new_state <= vertical;
48                                  ELSE
49                                          new_state <= plusform;
50                                  END IF;
51                                  -- It has been decided that a plus form needs to be generated
52                          WHEN plusform =>
53                                  FF1 <= '1';
54                                  new_state <= vertical;
55                                  -- All the requirements to make a vertical line are being ini
56                          WHEN vertical =>
57                                  FF2_reset <= '1';
58                                  lethaltile_x <= std_logic_vector(coor_signed_x(3 DOWNTO 0));
59                                  coor_signed <= coor_signed_y - "00010";
60                                  new_state <= vert_wait;
61                          WHEN vert_wait =>
62                                  FF2_reset <= '0';
63                                  read <= '0';
64                                  new_state <= vert_out;
65                                  -- If a player is hit, the victory state will be taken. If a
66                          WHEN vert_out =>
67                                  coor_signed <= coor_signed + "00001";
68                                  IF coor_signed(4) = '0' THEN
69                                          lethaltile_y <= std_logic_vector(coor_signed(3 DOWNTO
70                                          read <= '1';
71                                  END IF;
72                                  read <= '1';
73                                  IF (lethaltile_x = X_p1 AND coor_signed_p1 = coor_signed) THE
74                                          new_state <= victory_2;
75                                  ELSIF (lethaltile_x = X_p2 AND coor_signed_p2 = coor_signed)
76                                          new_state <= victory_1;
77                                  ELSIF coor_signed < 0 THEN
78                                          new_state <= vert_wait;
79                                  ELSIF (coor_signed > 9 AND FF1 = '1') THEN
80                                          new_state <= horizontal;
```

```
81                                      ELSIF (FF2_read = '1' AND FF1 = '1') THEN
82                                              new_state <= horizontal;
83                                      ELSIF FF2_read = '1' THEN
84                                              new_state <= rest;
85                                      ELSE
86                                              new_state <= vert_wait;
87                                      END IF;
88                                      -- If a player is hit, the victory state will be taken. If it
89                              WHEN horizontal =>
90                                      read <= '0';
91                                      FF2_reset <= '1';
92                                      lethaltile_y <= std_logic_vector(coor_signed_y(3 DOWNTO 0));
93                                      new_state <= hori_wait;
94                                      coor_signed <= coor_signed_x - "00010";
95                              WHEN hori_wait =>
96                                      FF2_reset <= '0';
97                                      read <= '0';
98                                      new_state <= hori_out;
99                              WHEN hori_out =>
100                                     coor_signed <= coor_signed + "00001";
101                                     IF coor_signed(4) = '0' THEN
102                                             lethaltile_x <= std_logic_vector(coor_signed(3 DOWNTO
103                                             read <= '1';
104                                     END IF;
105                                     IF (lethaltile_y = Y_p1 AND coor_signed_p1 = coor_signed) THE
106                                             new_state <= victory_2;
107                                     ELSIF (lethaltile_y = Y_p2 AND coor_signed_p2 = coor_signed)
108                                             new_state <= victory_1;
109                                     ELSIF coor_signed < 0 THEN
110                                             new_state <= hori_wait;
111                                     ELSIF FF2_read = '1' THEN
112                                             new_state <= rest;
113                                     ELSE
114                                             new_state <= hori_wait;
115                                     END IF;
116                                     -- The victory states will generate a victory signal with the
117                             WHEN victory_1 =>
118                                     victoryv <= "10";
119                                     new_state <= victory_1;
120                             WHEN victory_2 =>
121                                     victoryv <= "11";
122                                     new_state <= victory_2;
123                     END CASE;
124             END PROCESS;
125     END behaviour;
```

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.ALL;
3   ENTITY hitscan_top IS
4           PORT (
5                   X_b : IN std_logic_vector(3 DOWNTO 0);
6                   Y_b : IN std_logic_vector(3 DOWNTO 0);
7                   res : IN std_logic;
8                   clk : IN std_logic;
9                   explode : IN std_logic;
```

```
10                  X_p1 : IN std_logic_vector(3 DOWNTO 0);
11                  Y_p1 : IN std_logic_vector(3 DOWNTO 0);
12                  X_p2 : IN std_logic_vector(3 DOWNTO 0);
13                  Y_p2 : IN std_logic_vector(3 DOWNTO 0);
14                  victoryv : OUT std_logic_vector(1 DOWNTO 0);
15                  lethaltile_x : OUT std_logic_vector(3 DOWNTO 0);
16                  lethaltile_y : OUT std_logic_vector(3 DOWNTO 0)
17          );
18  END hitscan_top;
```

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.ALL;
3   -- This connects EVERYTHING
4   ARCHITECTURE structural OF hitscan_top IS
5           COMPONENT ff2
6                   PORT (
7                           read : IN std_logic;
8                           FF2_reset : IN std_logic;
9                           clk : IN std_logic;
10                          FF2_read : OUT std_logic
11                  );
12          END COMPONENT;
13          COMPONENT hitscan
14                  PORT (
15                          X_b : IN std_logic_vector(3 DOWNTO 0);
16                          Y_b : IN std_logic_vector(3 DOWNTO 0);
17                          res : IN std_logic;
18                          clk : IN std_logic;
19                          explode : IN std_logic;
20                          FF2_read : IN std_logic;
21                          X_p1 : IN std_logic_vector(3 DOWNTO 0);
22                          Y_p1 : IN std_logic_vector(3 DOWNTO 0);
23                          X_p2 : IN std_logic_vector(3 DOWNTO 0);
24                          Y_p2 : IN std_logic_vector(3 DOWNTO 0);
25                          victoryv : OUT std_logic_vector(1 DOWNTO 0);
26                          read : OUT std_logic;
27                          FF2_reset : OUT std_logic;
28                          lethaltile_x : OUT std_logic_vector(3 DOWNTO 0);
29                          lethaltile_y : OUT std_logic_vector(3 DOWNTO 0)
30                  );
31          END COMPONENT;
32          SIGNAL read, FF2_read, FF2_reset : std_logic;
33  BEGIN
34          pm_ff2 : FF2
35          PORT MAP(read, FF2_reset, clk, FF2_read);
36          pm_hs : hitscan
37          PORT MAP(X_b, Y_b, res, clk, explode, FF2_read, X_p1, Y_p1, X_p2, Y_p2, victoryv, rea
38  END structural;
```

6.2 Figures

Please put in main text in section Design

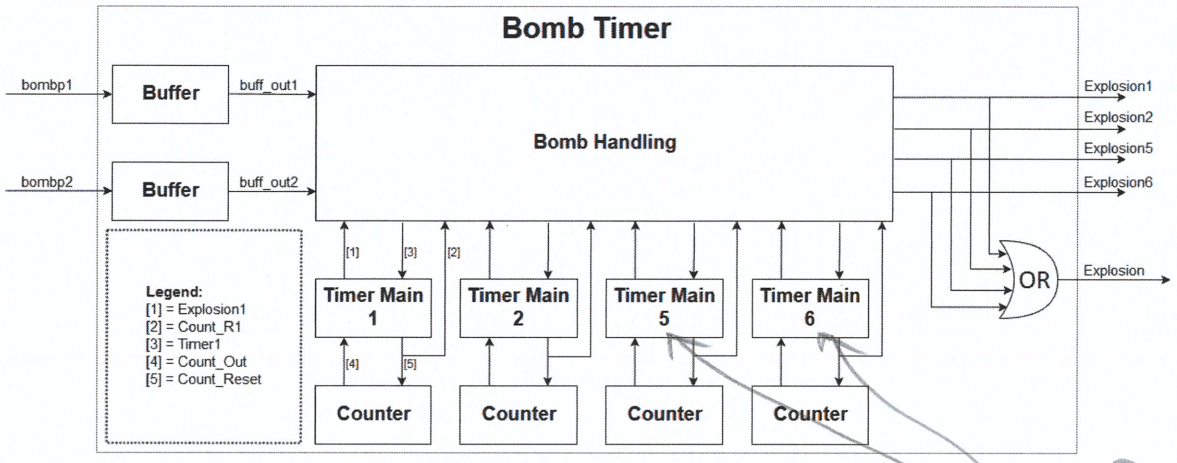

Figure 3: The overview of the "Bomb timer" part.

3, 4 ?

how connected?

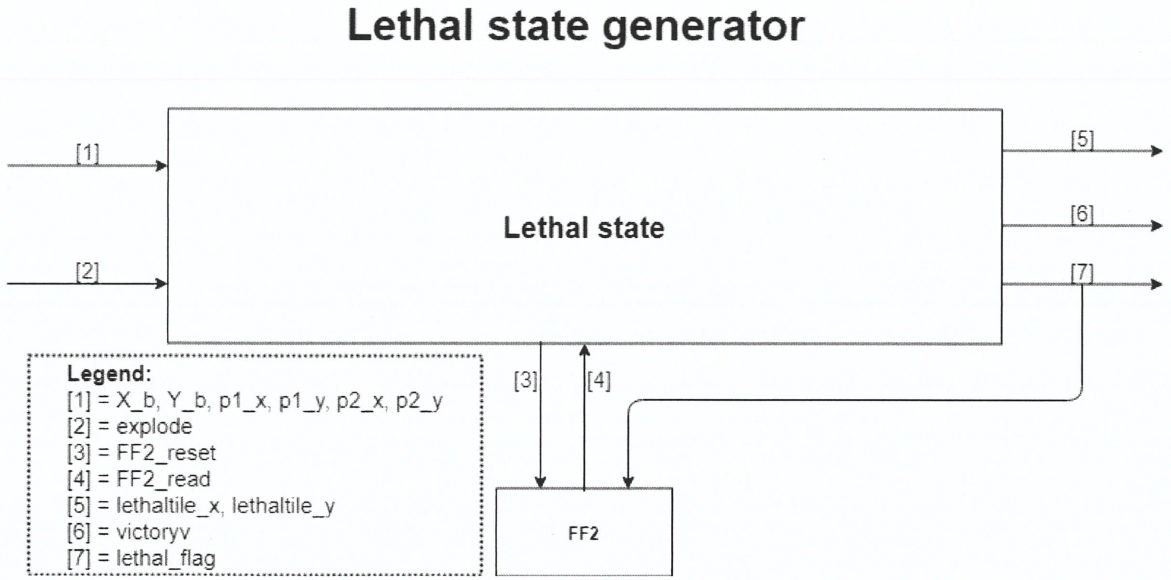

Figure 4: The overview of the "Lethal state generator" part.

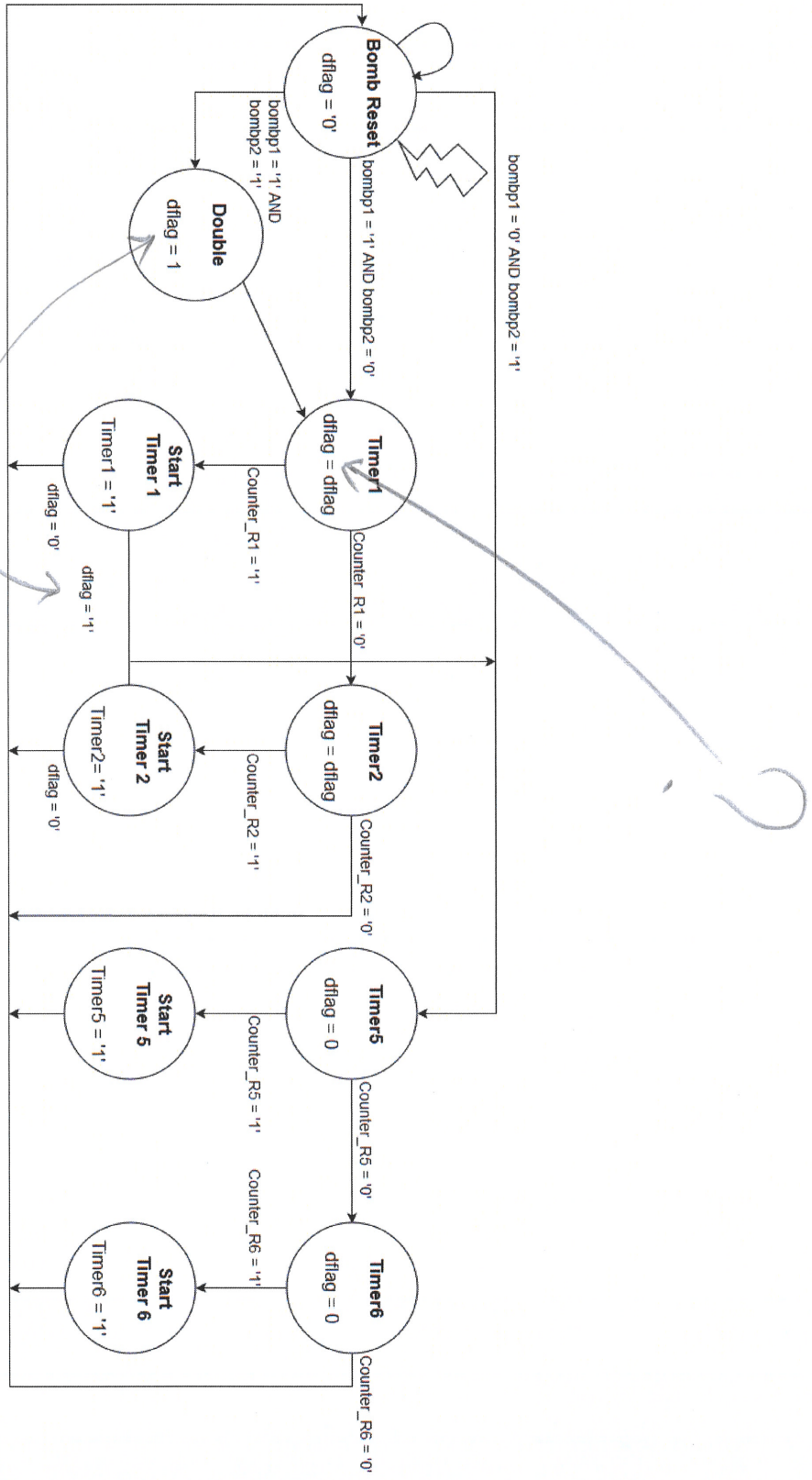

Figure 5: The FSM of the Handling Main entity.

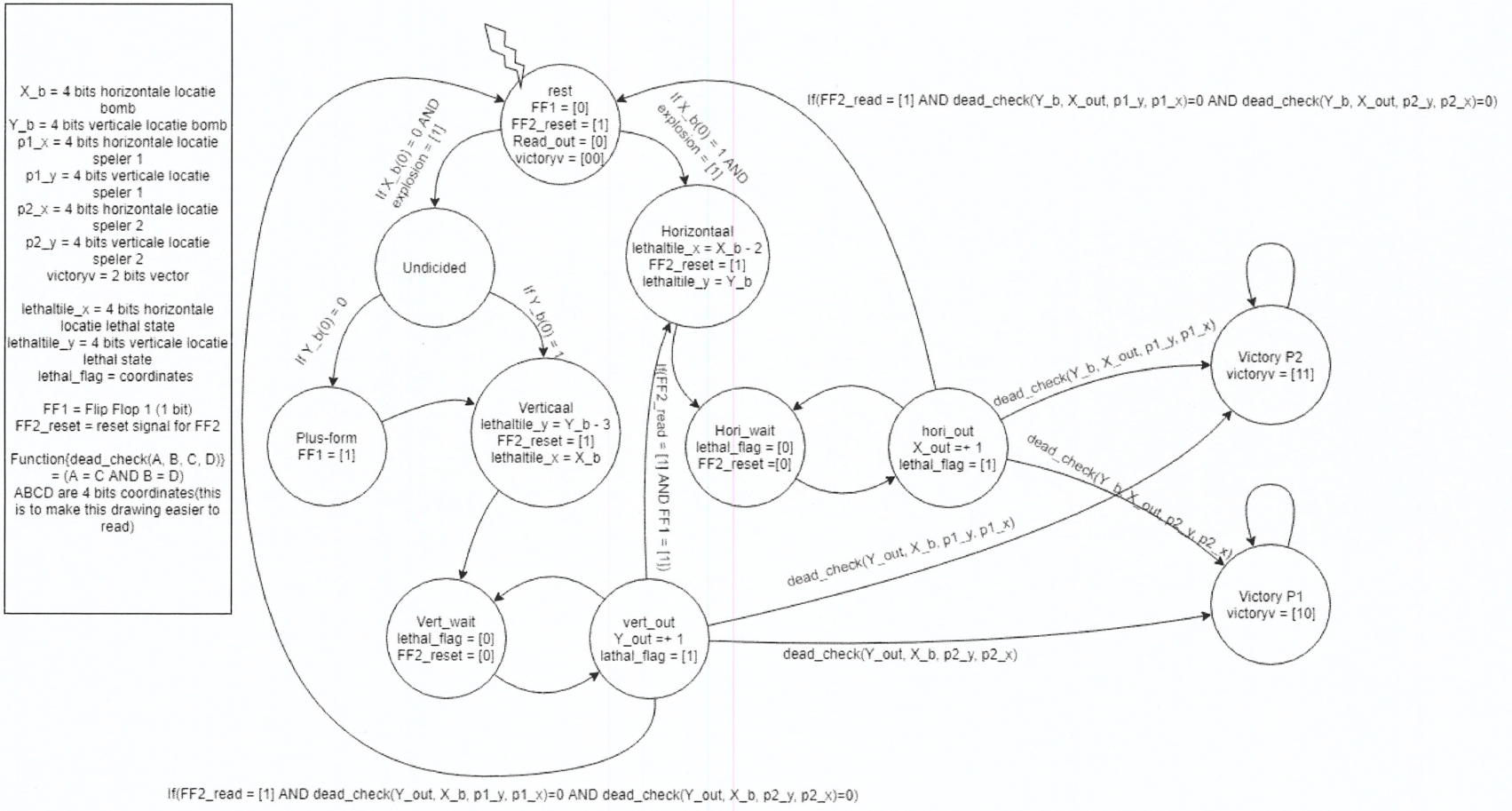

Figure 6: The “Lethal state” FSM

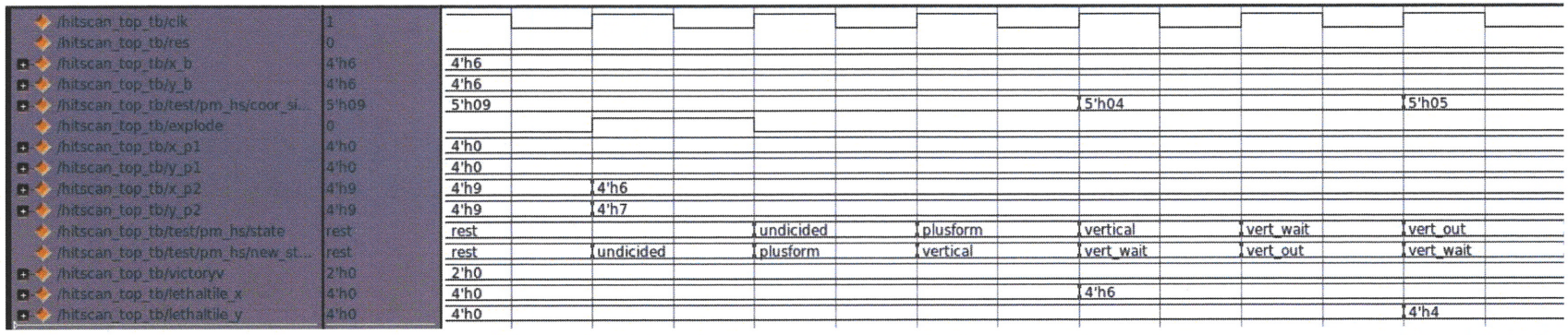

Figure 7: Simulation with a hit(seen by the fact that the FSM stays in victory1 state) part 1

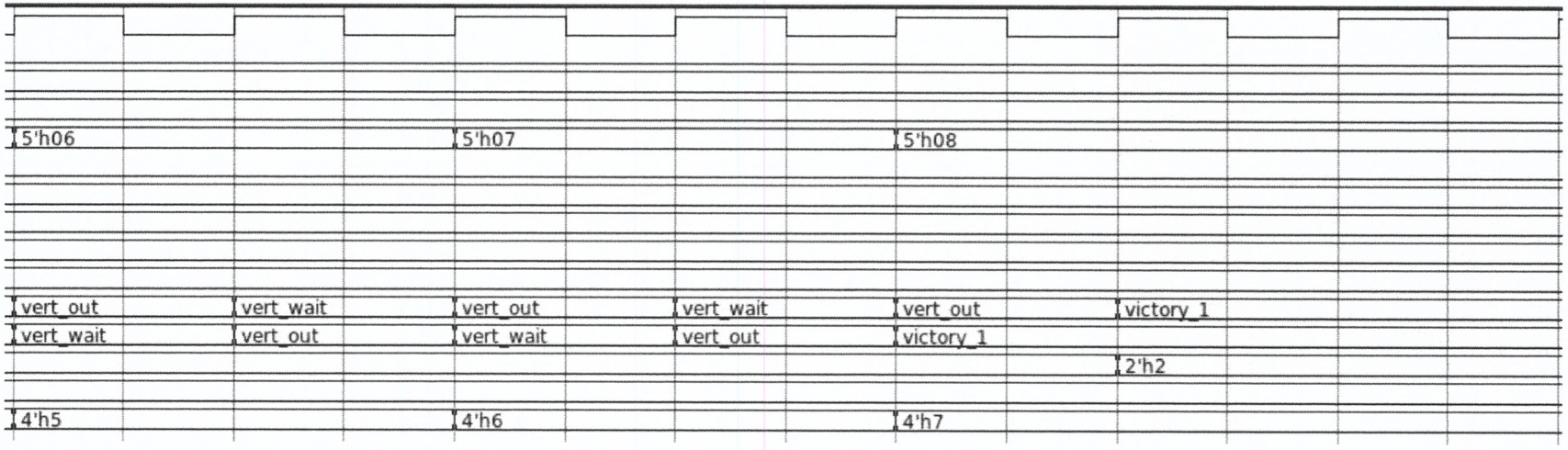

Figure 8: Simulation with a hit(seen by the fact that the FSM stays in victory1 state) part 2

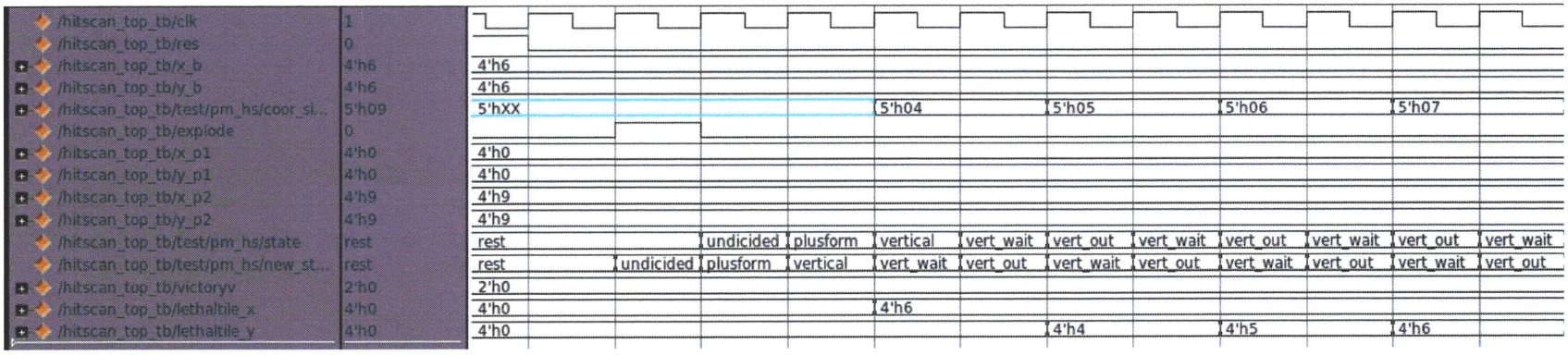

Figure 9: Simulation without a hit(seen by the fact that the FSM goes back to rest state) part 1

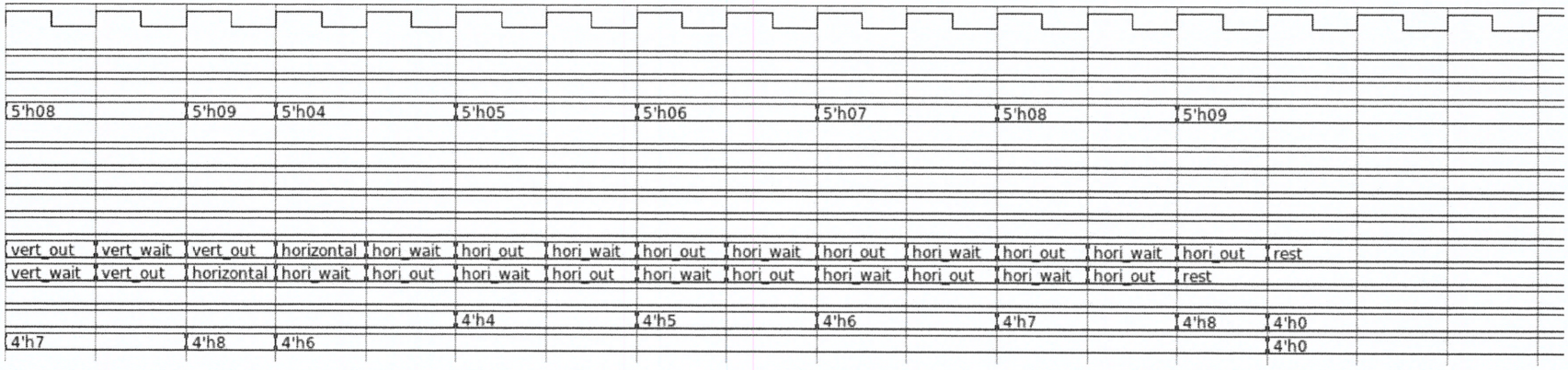

Figure 10: Testbench without a hit part 2

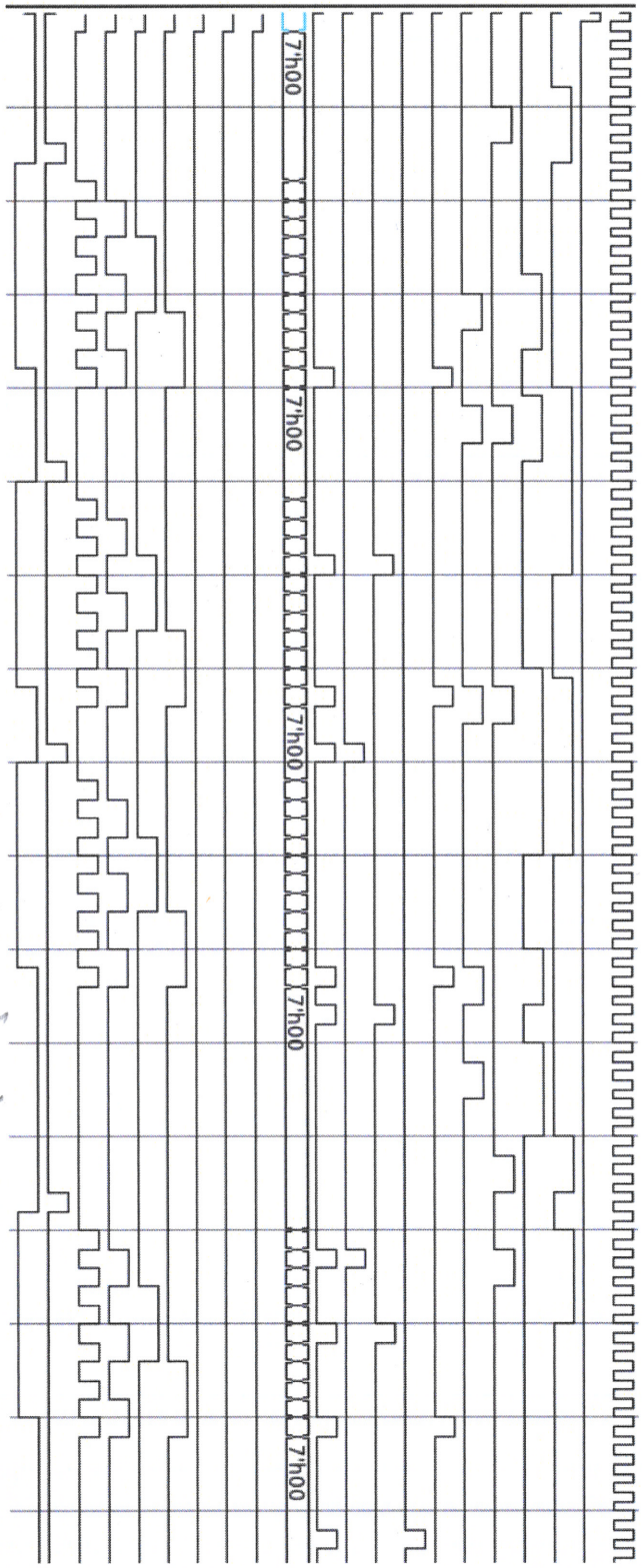

Figure 11: Caption

- Intro, specification and design are mixed up
- Reasonably clear text and figures
- Mealy problem

7 1/2