Project FSM – Fire Hydrant System

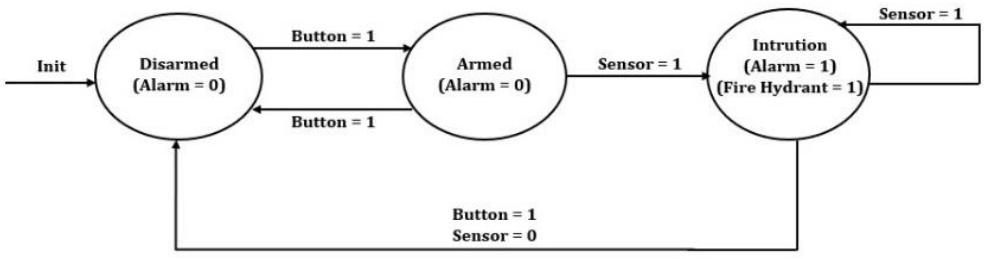
1. Tujuan

1. Menerapkan metode FSM dalam VHDL untuk sistem pemadam kebakaran.
2. Menampilkan simulasi sistem kendali pada sistem pemadam kebakaran untuk mengetahui aliran setiap perubahan state.

2. Teori

Finite State Machines (FSM) adalah sebuah metodologi perancangan sistem kontrol yang menggambarkan tingkah laku atau prinsip kerja sistem dengan menggunakan tiga hal berikut: State (Keadaan), Event (kejadian) dan action (aksi). Pada satu saat dalam periode waktu yang cukup signifikan, sistem akan berada pada salah satu state yang aktif. Sistem dapat beralih atau bertransisi menuju state lain jika mendapatkan masukan atau event tertentu, baik yang berasal dari perangkat luar atau komponen dalam sistemnya itu sendiri (misal interupsi timer). Transisi keadaan ini umumnya juga disertai oleh aksi yang dilakukan oleh sistem ketika menanggapi masukan yang terjadi. Aksi yang dilakukan tersebut dapat berupa aksi yang sederhana atau melibatkan rangkaian proses yang relative kompleks. Berdasarkan sifatnya, metode FSM ini sangat cocok digunakan sebagai basis perancangan perangkat lunak pengendalian yang bersifat reaktif dan real time. Salah satu keutungan nyata penggunaan FSM adalah kemampuannya dalam mendekomposisi aplikasi yang relative besar dengan hanya menggunakan sejumlah kecil item state. Selain untuk bidang kontrol, Penggunaan metode ini pada kenyataannya juga umum digunakan sebagai basis untuk perancangan protokol-protokol komunikasi, perancangan perangkat lunak game, aplikasi WEB dan sebagainya. Dalam bahasa pemrograman prosedural seperti bahasa C, FSM ini umumnya direalisasikan dengan menggunakan statemen kontrol switch case atau/dan if then. Dengan menggunakan statemen-statemen kontrol ini, aliran program secara praktis akan mudah dipahami dan dilacak jika terjadi kesalahan logika.

DIAGRAM KEADAAN



Gambar 1. Diagram State Fire Hydrant System

3. Alat dan Bahan

1. PC yang sudah terinstall ISE 13.1
2. Xilinx Sparatan 3
3. Downloader JTAG USB
4. Power Supply 5 volt

4. Kode Program

fire_alarm (Top Module)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.all;

entity fire_alarm is
   port( button: in std_logic;
               clr: in std_logic;
               clk: in std_logic;
               fire: in std_logic;
               sensor: out std_logic;
               alarm: out std_logic;
               hydrant: out std_logic;
               a_to_g : out STD_LOGIC_VECTOR (6 downto 0);
               an  : out STD_LOGIC_VECTOR (3 downto 0);
               dp  : out STD_LOGIC);
end fire_alarm;

architecture Behavioral of fire_alarm is
   component clkdiv is
         port( mclk : in std_logic;
                     clr : in std_logic;
                     clk48: out std_logic
                );
   end component;
   component x7segb is
         port ( x : in STD_LOGIC_VECTOR (15 downto 0);
               clk : in STD_LOGIC;
               clr : in STD_LOGIC;
               a_to_g : out STD_LOGIC_VECTOR (6 downto 0);
               an : out STD_LOGIC_VECTOR (3 downto 0);
               dp : out STD_LOGIC );
   end component;
   type state_type is (disarm,arm,intrution);
   signal state: state_type;
   signal clk48: std_logic;
   signal x: std_logic_vector(15 downto 0);

begin

   u1: clkdiv port map(
```

```
         mclk => clk,
         clr => clr,
         clk48 => clk48
         );

   process(button, fire, clk48)
   begin
         if clr='1' then
               state <= disarm;
         elsif clk48' event and clk48='1' then
               case state is
                     when disarm =>    if button = '1' then
state <= arm;
                                                      else state
<= disarm;
                                                      end if;
                     when arm => if button = '0' and fire = '1'
then state <= intrution;
                                                   elsif button = '1'
and fire = '0' then state <= disarm;
                                                   else state <= arm;
                                                   end if;
                     when intrution => if button = '0' and fire
= '1' then state <= intrution;
                                                            elsif
button = '1' and fire = '0' then state <= disarm;
                                                            else
state <= intrution;
                                                            end
if;
               end case;
               sensor <= fire;
         end if;
   end process;

   process(state, fire, clk48)
   begin
         case state is
               when disarm => alarm <= '0'; hydrant <= '0'; x
<= "0000000100110011";
               when arm => alarm <= '0'; hydrant <= '0'; x <=
"0000000000010010";
               when intrution => if fire = '1' then alarm <=
'1'; hydrant <= '1';
                                                      else alarm
<= '1'; hydrant <= '0';
                                                      end if;
         end case;
   end process;

   u2 : x7segb port map
```

```
               (x=>x, clk=>clk48, clr=>clr, a_to_g=>a_to_g,
an=>an, dp=>dp);

end Behavioral;
```

clkdiv

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity clkdiv is
Port (mclk : in  STD_LOGIC;
         clr : in STD_LOGIC;
         clk48 : out STD_LOGIC);
end clkdiv;

architecture Behavioral of clkdiv is
signal q:std_logic_vector(23 downto 0);
begin
   process(mclk, clr)
   begin
         if clr = '1' then
               q <= x"000000";
         elsif mclk' event and mclk = '1' then
               q <= q + 1;
         end if;
         clk48 <= q(22);
   end process;
end Behavioral ;
```

x7segb

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity x7segb is
   port( x : in STD_LOGIC_VECTOR (15 downto 0);
               clk : in STD_LOGIC;
               clr : in STD_LOGIC;
               a_to_g : out STD_LOGIC_VECTOR (6 downto 0);
               an : out STD_LOGIC_VECTOR (3 downto 0);
               dp :  out STD_LOGIC);
end x7segb;

architecture x7seg of x7segb is
         signal s : STD_LOGIC_VECTOR (1 downto 0);
         signal digit : STD_LOGIC_VECTOR (3 downto 0);
         signal aen : STD_LOGIC_VECTOR (3 downto 0);
         signal clkdiv : STD_LOGIC_VECTOR (20 downto 0);
```

```
begin
        s <= clkdiv(20 downto 19);
        dp <= '1';

-- set aen(3 downton 0) for leading blenks
        aen(3) <= x(15) or x(14) or x(13) or x(12);
        aen(2) <= x(15) or x(14) or x(13) or x(12) or x(11) or
x(10) or x(9) or x(8);
        aen(1) <= x(15) or x(14) or x(13) or x(12) or x(11) or
x(10) or x(9) or x(8) or x(7) or x(6) or x(5) or x(4);
        aen(0) <= '1'; -- digit 0 always on

-- quad 4 to 1 mux
        process(s,x)
        begin
                    case s is
                                when "00" => digit <= x(3
downto 0);
                                when "01" => digit <= x(7
downto 4);
                                when "10" => digit <= x(11
downto 8);
                                when others => digit <= x(15
downto 12);
                    end case;
        end process;

-- 7 segment decoder: hex7seg
        process(digit)
        begin
                    case digit is
                                when x"0" => a_to_g <=
"0000001";
                                when x"1" => a_to_g <=
"0000001";
                                when x"2" => a_to_g <=
"0001001";
                                when others  => a_to_g <=
"0111000";
                    end case;
        end process;

-- digit select: ancode
        process(s,aen)
        begin
                    an <= "1111";
                    if aen(conv_integer(s)) = '1' then
                                an(conv_integer(s)) <= '0' ;
                    end if;
        end process;

-- clock devider
```

```
        process(clk,clr)
        begin
                if clr = '1' then
                        clkdiv <=(others => '0');
                elsif clk' event and clk = '1' then
                        clkdiv <= clkdiv + 1 ;
                end if;
        end process;

end x7seg;
```

Pin

```
NET button LOC = M13;
NET clr LOC = L14;
NET fire LOC = K13;

NET hydrant LOC = K12;
NET alarm LOC = P14;
NET sensor LOC = P11;

NET a_to_g(0) LOC = N16;
NET a_to_g(1) LOC = F13;
NET a_to_g(2) LOC = R16;
NET a_to_g(3) LOC = P15;
NET a_to_g(4) LOC = N15;
NET a_to_g(5) LOC = G13;
NET a_to_g(6) LOC = E14;
NET dp LOC = P16;

NET an(0) LOC = D14;
NET an(1) LOC = G14;
NET an(2) LOC = F14;
NET an(3) LOC = E13;
```

5. Hasil Output

Gambar 2. State Disarmed (sensor = 1)

Gambar 3. State Armed

Gambar 4. State Intrution

Gambar 5. State Armed setelah Intrution

Gambar 6. State Disarmed (sensor = 1)

6. Analisa

Pada project *Fire Hydrant System* ini kita mengimplementasikan metode FSM (Finite State Machine) ke sistem pemandam api yang membutuhkan sensor untuk mendeteksi api, alarm sebagai indikator bahwa terjadi kebakaran, dan pemadam untuk memadamkan api tersebut. Sistem ini memiliki 3 kondisi, yaitu *disarmed*, *armed*, dan *intrution*. Disarmed adalah sebuah kondisi dimana sistem sedang tidak bekerja atau mati. Armed adalah sebuah kondisi sistem sudah siap untuk memadamkan api. Intrution adalah kondisi dimana api sudah terdeteksi dan sistem akan menyalakan alarm dan pemadam untuk memadamkan api.

Awal mula perangkat FPGA dihidupkan akan memasuki kondisi disarmed yang dimana sistem belum menyala sehingga ketika ada api yang terdeteksi perangkat tidak akan memadamkan api. Ketika tombol ditekan, state akan berganti ke kondisi armed. Ketika tombol ditekan lagi ketika dalam kondisi armed, maka state akan kembali ke disarmed. Saat ada api yang terdeteksi saat state armed, secara otomatis state berganti ke intrution lalu perangkat akan mulai memadamkan api dan menyalakan alarm. Ketika api sudah padam state kembali berganti ke armed.

7. Kesimpulan

Setelah membuat project ini, kita dapat menerapkan metode FSM dalam VHDL untuk sistem pemadam kebakaran serta mengetahui perubahan state nya.