```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4
5
6
7
8
9
10
11
12
13  entity watering is
14      Port ( clk : in  STD_LOGIC;
15             i_reset : in  STD_LOGIC;
16             i_start : in  STD_LOGIC;
17             i_M : in  STD_LOGIC_VECTOR (2 downto 0);
18             i_T : in  STD_LOGIC;
19             i_L : in  STD_LOGIC;
20             o_state : out  STD_LOGIC_VECTOR (1 downto 0);
21             o_SEG : out  STD_LOGIC_VECTOR (7 downto 0);
22             o_watering : out  STD_LOGIC;
23             o_M : out  STD_LOGIC_VECTOR (2 downto 0);
24             o_T : out  STD_LOGIC;
25             o_L : out  STD_LOGIC);
26  end watering;
27
28
29
30
31
32
33
34
35
36
37
38
39
40  architecture Behavioral of watering is
41
42     SUBTYPE Tstate IS std_logic_vector (1 downto 0);
43
44      constant St0 : Tstate := "00";
45      constant St1 : Tstate := "01";
46      constant St2 : Tstate := "11";
47
48     SIGNAL state: Tstate;
49     SIGNAL next_state: Tstate;
50
51  begin
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68  o_M <= i_M;
69  o_T <= i_T;
```

در اینجا کتابخانه ها و پکیج های مورد نیاز به کد اضافه شده است.

نکته عمومی در کد:
در زبان کدنویسی، اگر بخواهیم به یک متغیر یک بیتی مقدار دهی کنیم، از
single quote = '1' , '0' , ...
استفاده میکنیم

و برای متغیرهای بیشتر از یک بیت، از
"11" , "000" , ...
استفاده میکنیم

در اینجا ورودی ها و خروجی ها در قسمت
entity
اضافه شده اند

ورودی ها و خروجی ها دقیقا مشابه فایل پی دی اف ارسال شده هستند

ورودی ها و خروجی ها اگر بیشتر از یک بیت باشند از نوع
std_logic_vector
تعریف شده اند

مثل
i_M , o_SEG , ...

ورودی ها و خروجی ها اگر تک بیتی باشند، از نوع
std_logic
تعریف میشوند

مثل
clk , o_M , ...

بین
architecture
و
begin
میتوانیم موارد زیر را تعریف کنیم
type - subtype - signal - constant , ...

در خط 42 ابتدا یک
subtype
به اسم
Tstate
تعریف شده است
چون متغیرهای حالت ما یعنی همان
St0 - St1 -St2
دو بیتی هستند
.این هم دو بیتی تعریف شده است
البته تعریف آن ضروری نبود ولی
به ساده تر شدن کد کمک میکند

در خطوط 44 - 45 - 46 متغیرهای حالت
با مقادیر اختصاص داده شده به خود
به صورت ثابت یا
constant
تعریف شده اند
این موارد همواره در کد ثابت هستند
و تغییر نمیکنند
به عنوان مثال همواره در کد داریم
St0="00"

در خطوط 48-49
دو سیگنال مربوط به حالت فعلی و حالت بعدی
به ترتیب به اسم های
state
next_state
تعریف شده اند
در طول کد ما حالت فعلی و حالت بعدی را به
آنها ارجاع میدهیم

```
70  o_L <= i_L;
71  o_state <= state;
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89      PROCESS(clk)
90      BEGIN
91           IF rising_edge(clk) THEN
92               IF i_reset ='1' then
93               state <= St2;
94               ELSE
95               state <= next_state;
96               END IF;
97         END IF;
98      END PROCESS;
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117 PROCESS (state, i_M, i_T , i_L , i_start)
118
119     BEGIN
120
121      next_state <= St2;
122
123        CASE state IS
124
125           WHEN St0 =>
126                  o_watering <= '0';
127                  o_SEG <= "01000000";
128
129                  if (i_T='0' and i_L='0' and i_M>"011") then
130                      next_state <= St0;
131                  elsif ((i_T='1' or i_L='1') and i_M>"001") then
132                      next_state <= St0;
133                  elsif (i_T='0' and i_L='0' and i_M<="011") then
134                      next_state <= St1;
135                  elsif ((i_T='1' or i_L='1') and i_M<="001") then
136                      next_state <= St1;
137                  end if;
138
```

همانطور که گفته شده است، خروجی
o_state
حالت فعلی را در خروجی نشان میدهیم
پس باید مقدار حالت فعلی
یا
state
را به آن نسبت داد
که در خط 71 انجام شده است

چون خروجی های
o_M , o_T , o_L
مستقیما از ورودی های
i_M , i_T , i_L
گرفته می شوند، در اینجا
یعنی در ابتدای کد در خطوط 68-69-70
تکلیف آن ها را مشخص کرده ایم

در اینجا یک پراسس ترتیبی نوشته شده است.
چون میخواهیم همه کارها سنکرون با کلاک باشد
یعنی همه کارها در لبه بالا رونده کلاک صورت پذیرد
در لیست حساسیت آن فقط کلاک نوشته شده است

در خط 91 چک می شود که آیا لبه مثبت کلاک آمده است یا خیر. اگر آمده بود، چک می شود که آیا ریست یک شده است یا خیر. اگر ریست یک شده بود، باید به حالت
St2
برویم پس مقدار حالت فعلی یا
state
را مساوی آن قرار میدهیم

این کار در خط 93 انجام شده است. اما اگر ریست یک نشده بود، لازم است که حالت بعدی را در حالت فعلی بریزیم
این کار در خط 95 انجام شده است
یعنی
next_state
را در
state
ریخته ایم

در اینجا یک پراسس ترکیبی نوشته شده است. در لیست حساسیت آن باید همه سیگنال هایی که در شرط ها وجود دارد نوشته شده باشد. همچنین همه سیگنال هایی که در سمت راست تساوی ها وجود دارد باید نوشته شود

در ادامه یک کیس نوشته شده است
و ماشین حالت پیاده سازی شده است. چون ماشین حالت
از نوع مور است، خروجی در هر حالت فقط به حالت مدار
وابسته است پس خروجی ها
یعنی
o_watering , o_seg
در همانجا مقدار دهی شده اند

ما در طول کد و در داخل این پراسس، به متغیر حالت بعدی یا
next_state
مقدار دهی کرده ایم و این مقدار دهی ها در شرط های مختلف صورت گرفته است.
ابتدای کد یک مقدار با متغیر حالت بعدی نسبت میدهیم
تا اگر هیچکدام از شرایط صادق نبود،
این مقدار به آن نسبت داده شود.
این کار در خط 121 انجام شده است

در ادامه شرط های ماشین حالت از روی فایل پی در اف، وارد شده است.

```
WHEN St1 =>

        o_watering <= '1';
        o_SEG <= "01110110";

        if (i_T='0' and i_L='0' and i_M<"111") then
            next_state <= St1;
        elsif ((i_T='1' or i_L='1') and i_M<"011") then
            next_state <= St1;
        elsif (i_T='0' and i_L='0' and i_M>="011") then
            next_state <= St0;
        elsif ((i_T='1' or i_L='1') and i_M>="111") then
            next_state <= St0;
        end if;
```

توضیحات این قسمت مانند قسمت قبلی است. در قسمت قبلی حالت
St0
پیاده سازی شده بود.
در این قسمت حالت
St1
پیاده سازی شده است

```
WHEN St2 =>
        o_watering <= '0';
        o_SEG <= "01000000";

        if (i_start='1') then
            next_state <= St0;
        else
            next_state <= St2;
        end if;
```

در این قسمت حالت
St2
پیاده سازی شده است

```
208
209
210
211
212
213
214
215             WHEN OTHERS =>
216                    next_state <= St2;
217                    o_watering <= '0';
218                    o_SEG <= "01000000";
219                END CASE;
220
221
222
223                         در این قسمت، حالتی که که حالت فعلی روی هیچ کدام از حالت های قبلی نباشد،
224                         پیاده سازی شده است
225
226
227
228
229
230
231
232
233
234       END PROCESS;
235
236
237   end Behavioral;
238
239
```