پیش گزارش راه اندازی کیبورد ماتریسی

نام: آناهیتا　　نام خانوادگی: اسدی　　شماره دانشجویی: 9724453

نام مدرس: سرکار خانم فرشته کلانتری　　روز و ساعت کلاس: یکشنبه ها، ساعت 13:30 الی 15:30

پاسخ مرحله 1

1. بله. چون کلیدهای کیبورد که فشرده میشوند، خاصیت فنری دارند و مدت زمانی طول میکشد تا بین قطع و وصل نوسان کنند و به سطح ثابتی از سیگنال خروجی برسند (bounce میکنند) و باید پس از مدت زمانی که خروجی stableتر شد خوانده شوند.

2.

- **ماژول keyboard:**

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

اضافه کردن کتابخانه های موردنیاز؛ کتابخانه اول برای گیت های منطقی (در این آزمایش not کردن یک سیگنال)، کتابخانه دوم برای عملیات ریاضی (مانند جمع کردن i و count با 1) و کتابخانه سوم برای اعداد غیرعلامت دار (برای مثال 1000 را 8 (و نه -7) درنظر میگیرد) است. کتابخانه آخر، برای خواندن فایل است که عملیات readline و... و تایپ های file و line در آن تعریف شده.

```
entity keyboard is port(   row: out std_logic_vector(3 downto 0);
                           column: in std_logic_vector(3 downto 0);
                           clk: in std_logic;
                           output: out std_logic_vector(3 downto 0);
                           hit: out std_logic);
end keyboard;
```

تعریف پورت های ورودی و خروجی؛ ورودی column و خروجی row که 4 بیتی هستند، ورودی clk، خروجی 4 بیتی برای نمایش عدد معادل خانه‌ی ماتریس، و خروجی 1 بیتی hit برای نمایش اینکه آیا کلیدی فشرده شده یا خیر. از آنجا که در هر مرحله تنها 1 کلید میتواند فشرده شود، این خروجی برابر 0 یا 1 خواهد بود.

```
architecture behavioral of keyboard is

type state is (check, row1, row2, row3, row4);
signal pr_state: state := check;
```

تعریف 5 حالت خواسته شده در ماشین حالت، و همچنین پوینتر ماشین حالت تحت عنوان pr_state با حالت اولیه check.

```
begin
process(clk)
   begin
   if rising_edge(clk) then
      case pr_state is  when check =>  row <= "0000";
                                       if column = "1111" then
                                          hit <= '0';
                                          pr_state <= check;
                                       else
                                          pr_state <= row1;
                                       end if;
```

حالت اول (check) تا وقتی که هیچکدام از ورودی های column فشرده نشده باشند، در این حالت میمانیم و خروجی row را 0000 میکنیم. زمانی که حتی یک کلید فشرده شود، به مرحله بعد (row1) میرویم.

```
when row1 =>    row <= "ZZZ0";
                if column = "1111" then
                   hit <= '0';
                   pr_state <= row2;
                else
                   hit <= '1';
                   if column(0) = '0' then
                      output <= "0000";
                   elsif column(1) = '0' then
                      output <= "0001";
                   elsif column(2) = '0' then
                      output <= "0010";
                   elsif column(3) = '0' then
                      output <= "0011";
                   end if;
                   pr_state <= check;
                end if;
```

این حالت برای وقتی است که قرار باشد اولین سطر row، 1 باشد. اگر دیگر کلیدی فشرده نشد، یعنی باید به مرحله بعد (انتخاب دومین سطر row) برویم. اما اگر کلیدی فشرده شد، خوانده میشوند که کدام یک از بیت های ورودی (column) فشرده شده اند (0 شده اند) و عدد متناسب با ماتریس، در output قرار داده میشود.

```
when row2 =>    row <= "ZZ0Z";
                if column = "1111" then
                   hit <= '0';
                   pr_state <= row3;
                else
                   hit <= '1';
                   if column(0) = '0' then
                      output <= "0100";
                   elsif column(1) = '0' then
                      output <= "0101";
                   elsif column(2) = '0' then
                      output <= "0110";
                   elsif column(3) = '0' then
                      output <= "0111";
                   end if;
                   pr_state <= check;
                end if;
```

این حالت برای وقتی است که قرار باشد دومین سطر row، 1 باشد. اگر دیگر کلیدی فشرده نشد، یعنی باید به مرحله بعد (انتخاب سومین سطر row) برویم. اما اگر کلیدی فشرده شد، خوانده میشوند که کدام یک از بیت های ورودی (column) فشرده شده اند (0 شده اند) و عدد متناسب با ماتریس، در output قرار داده میشود.

```
when row3 =>    row <= "Z0ZZ";
                if column = "1111" then
                   hit <= '0';
                   pr_state <= row4;
                else
                   hit <= '1';
                   if column(0) = '0' then
                      output <= "1000";
                   elsif column(1) = '0' then
                      output <= "1001";
                   elsif column(2) = '0' then
                      output <= "1010";
                   elsif column(3) = '0' then
                      output <= "1011";
                   end if;
                   pr_state <= check;
                end if;
```

این حالت برای وقتی است که قرار باشد سومین سطر row، 1 باشد. اگر دیگر کلیدی فشرده نشد، یعنی باید به مرحله بعد (انتخاب چهارمین سطر row) برویم. اما اگر کلیدی فشرده شد، خوانده میشوند که کدام یک از بیت های ورودی (column) فشرده شده اند (0 شده اند) و عدد متناسب با ماتریس، در output قرار داده میشود.

```
when row4 =>    row <= "0ZZZ";
                if column = "1111" then
                   hit <= '0';
                   pr_state <= check;
                else
                   hit <= '1';
                   if column(0) = '0' then
                      output <= "1100";
                   elsif column(1) = '0' then
                      output <= "1101";
                   elsif column(2) = '0' then
                      output <= "1110";
                   elsif column(3) = '0' then
                      output <= "1111";
                   end if;
                   pr_state <= check;
                end if;
```

این حالت برای وقتی است که قرار باشد چهارمین سطر row، 1 باشد. اگر دیگر کلیدی فشرده نشد، یعنی باید به مرحله بعد (check؛ چون شاید کلا نباید کلیدی فشرده میشد) برویم. اما اگر کلیدی فشرده شد، خوانده میشوند که کدام یک از بیت های ورودی (column) فشرده شده اند (0 شده اند) و عدد متناسب با ماتریس، در output قرار داده میشود.

```
                when others => pr_state <= check;
                               hit <= '0';
                               output <= "0000";
      end case;
   end if;
end process;
end behavioral;
```

برای حالات غیر مترقبه (مثلا انتخاب دو ورودی از column و...) تعریف شده که به حالت اولیه‌ی check برویم.

.3

- **ماژول sevensegment برای تبدیل ورودی binary به ورودی hexadecimal:**

```
library IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE STD.TEXTIO.ALL;
```

اضافه کردن کتابخانه ها. کتابخانه آخر، برای خواندن فایل است که عملیات readline و... و تایپ های file و line در آن تعریف شده.

```
entity sevensegment is port(  clk: in std_logic;
                              input: in std_logic_vector(3 downto 0);
                              output: out bit_vector(7 downto 0);
                              com: out std_logic_vector(3 downto 0);
                              hit: in std_logic);
end sevensegment;
```

تعریف پورت ورودی و خروجی. 4 بیت input خروجی کیبورد است، که میتواند 16 مقدار از 0 تا 15 بگیرد. و سپس به خروجی output برای نمایش روی سون سگمنت تبدیل میشود.

```
architecture behavioral of sevensegment is

   type RAM_type is array (0 to 9) of bit_vector(7 downto 0);

   function ram_initialize (filename: in string) return RAM_type is
      file filel: text is in filename;
      variable linel: line;
      variable RAM1: RAM_type;
      begin for i in RAM_type'range loop  readline(filel, linel);
                                          read(linel, RAM1(i));
                                    end loop;
      return RAM1;
   end function;
   signal RAM: RAM_type := ram_initialize("values.txt");
```

تعریف فانکشن (زیربرنامه) خواندن فایل. این فانکشن، نام فایل را میگیرد و در حافظه RAM (که پیشتر تعریف شده 16 بایت است) میریزد؛ به این صورت که فایل (که از نوع text است) را طبق نام داده شده باز میکند، و هر خط را بواسطه متغیر داخل RAM1 (خروجی) میریزد. و این کار را برای هر 16 بایت (که هر بایت در خط جدا ذخیره شده، پس برای 16 خط) تکرار میکند. سپس فانکشن را صدا میزنیم، نام فایل را بهش میدهیم، و خروجی فانکشن را در RAM ذخیره میکنیم.

```
   signal ones: std_logic_vector(3 downto 0) := "0000";
   signal tens: std_logic_vector(3 downto 0) := "0000";
begin
   tens <=  "0001" when input > "1001" else
            "0000";
   ones <=  input - tens;
```

یکان و دهگان ورودی، برای نمایش روی دو سون سگمنت مشخص میشوند.

```
   process(clk)
      variable flag: std_logic := '0';
   begin
   if rising_edge(clk) then
      if hit = '1' then
         if (flag = '0') then
            com <= "0001";
            output <= RAM(conv_integer(ones));
            flag := '1';
         else
            com <= "0010";
            output <= RAM(conv_integer(tens));
            flag := '0';
      end if;
      elsif hit = '0' then
         output <= RAM(10);
      end if;
   end if;
   end process;
end behavioral;
```

اگر hit برابر 0 بود و کلیدی فشرده نشده بود، خروجی سون سگمنت ها بصورت "- -" خواهد بود. درغیر این صورت، ورودی یکان، ابتدا به integer تبدیل میشود، که معادل 0 تا 15 خانه ی آرایه RAM شود. سپس مقدار خانه معادل، در خروجی ریخته میشود. این عمل برای دهگان نیز تکرار میشود. با هر کلاک، خروجی بین یکان و دهگان سوییچ میشود (با تغییر پایه common). چشم متوجه این سوییچ در فرکانس بالای کلاک (50MHz) نمیشود.

- **ماژول clock_division برای تبدیل فرکانس کلاک به 2kHz:**

لازم است محاسبه کنیم برای تولید هر فرکانس دلخواه، باید چه تعداد کلاک بگذرد:

$$DIV = \frac{f_{fpga\ clk}}{f_{sampling}} = \frac{50MHz}{2kHz} = 25000$$

2.

```
library IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;
```

اضافه کردن کتابخانه های موردنیاز؛ کتابخانه اول برای گیت های منطقی (در این آزمایش not کردن یک سیگنال)، کتابخانه دوم برای عملیات ریاضی (مانند جمع کردن i با 1) و کتابخانه سوم برای اعداد غیرعلامت دار (برای مثال 1000 را 8 (و نه -7) درنظر میگیرد) است.

```
entity clock_division is   generic( DIV  :  integer := 10);
                           port(    clk_in:  in std_logic;
                                    clk_out: out std_logic);
end clock_division;
```

تعریف پورت های ورودی و خروجی (generic در ماژول نهایی، 25000 قرار داده میشود)

```
   process(clk)
      variable flag: std_logic := '0';
   begin
   if rising_edge(clk) then
      if hit = '1' then
         if (flag = '0') then
            com <= "0001";
            output <= RAM(conv_integer(ones));
            flag := '1';
         else
            com <= "0010";
            output <= RAM(conv_integer(tens));
            flag := '0';
      end if;
      elsif hit = '0' then
         output <= "11111110";
      end if;
   end if;
   end process;
end behavioral;
```

تعریف DIV داخل برنامه و مطابق دستور داده شده، و همچنین تعریف clk چون پورت خروجی clk_out قابل خواندن و اجرای عملیات (not کردن clk_out) نیست، تعریف پروسه برای دستورات ترتیبی، و همچنین سیگنالی که پروسه توسط آن انجام میشود (در اینجا clk_in).

با هر لبه یالارونده کلاک، شمارنده (i) یک واحد زیاد میشود و اگر این عمل DIV بار انجام شود، سیگنال (clk) not میشود، و درنهایت clk را در خروجی قرارداده میشود.

- **ماژول debouncer بعلت توضیحات سوال 1:**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
```

تعریف کتابخانه ها

```
ENTITY debouncer IS
  GENERIC(
    counter_size  :  INTEGER := 10); --counter size (10 bits gives 40.9us with 50MHz clock)
  PORT(
    clk     : IN  STD_LOGIC;  --input clock 50MHz
    button  : IN  STD_LOGIC;  --input signal to be debounced
    result  : OUT STD_LOGIC); --debounced signal
END debounce;
```

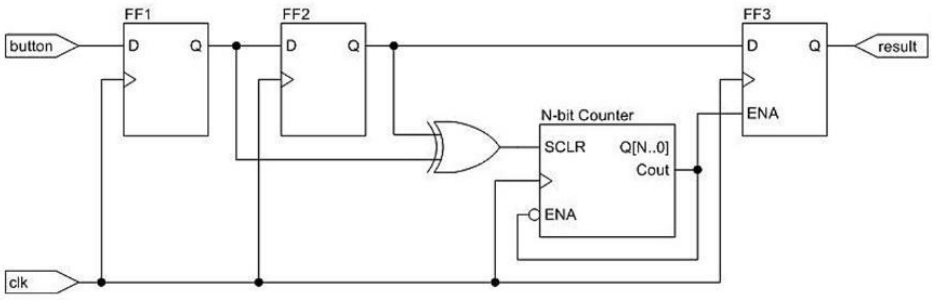
ورودی های این کد، کلاک 50MHz داخلی FPGA و داده یک بیتی هستند. خروجی قرار است همان داده یک بیتی باشد.

```
ARCHITECTURE logic OF debouncer IS
  SIGNAL flipflops   : STD_LOGIC_VECTOR(1 DOWNTO 0); --input flip flops
  SIGNAL counter_set : STD_LOGIC;                    --sync reset to zero
  SIGNAL counter_out : STD_LOGIC_VECTOR(counter_size DOWNTO 0) := (OTHERS => '0'); --counter output
BEGIN

  counter_set <= flipflops(0) xor flipflops(1);   --determine when to start/reset counter

  PROCESS(clk)
  BEGIN
    IF(clk'EVENT and clk = '1') THEN
      flipflops(0) <= button;
      flipflops(1) <= flipflops(0);
      If(counter_set = '1') THEN                  --reset counter because input is changing
        counter_out <= (OTHERS => '0');
      ELSIF(counter_out(counter_size) = '0') THEN --stable input time is not yet met
        counter_out <= counter_out + 1;
      ELSE                                        --stable input time is met
        result <= flipflops(1);
      END IF;
    END IF;
  END PROCESS;
END logic;
```

مطابق شکل زیر:



ورودی فلیپ فلاپ اول، همان ورودی ما (botton) است. باید از دو فلیپ فلاپ استفاده (برای مقایسه داده در دو زمان متفاوت –که به فرکانس کلاک بستگی دارد-) شود و خروجی هایشان مقایسه شود. این سیگنال چون در process(clk) قرار میگیرند خود به خود بصورت فلیپ فلاپ سنتز میشوند.

اگر ورودی هنوز bounce داشته باشد، خروجی های این دو فلیپ فلاپ متفاوت خواهند بود، پس خروجی گیت XOR (ورودی counter) برابر 1 میشود. طبق کد، اگر ورودی counter برابر 1 باشد، خروجی آن 0 است (reset شده) و چون خروجی counter به enable فلیپ فلاپ سوم وصل است، این فلیپ فلاپ خاموش میماند.

اگر bounce ورودی تمام شده باشد، خروجی فلیپ فلاپ ها یکسان خواهد بود. پس خروجی XOR برابر 0 میشود. طبق کد، در این حالت counter از 0 تا 01111111111 میشمارد (چون به محض 1 شدن بیت یازدهم، ورودی در result قرار میگیرد) پس یعنی 2^{11} تا میشمارد (یعنی تاخیر تقریبا برابر $40.6\mu s \cong \frac{2^{11}}{50\times10^{6}}$) و سپس خروجی فلیپ فلاپ دوم را بعنوان نتیجه نهایی میدهد.

یعنی ورودی پس از 40.6us نهایی میشود.

.4

- **ماژول keyboard_matrix که componentها در آن اضافه شده اند:**

```
library IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE STD.TEXTIO.ALL;
```

اضافه کردن کتابخانه هایی که در componentهای بالا ازشان استفاده شده

```
entity matrix_keyboard is port(  clk: in std_logic;
                                 row: out std_logic_vector(3 downto 0);
                                 column: in std_logic_vector(3 downto 0);
                                 output: out bit_vector(7 downto 0);
                                 common: out std_logic_vector(3 downto 0));
end matrix_keyboard;
```

تعریف پورت های ورودی و خروجی. output و common برای سون سگمنت هستند.

```
architecture Behavioral of matrix_keyboard is

   component debouncer  GENERIC( counter_size: INTEGER := 10);
                        PORT(    clk: IN STD_LOGIC;
                                 button: IN  STD_LOGIC;
                                 result: OUT STD_LOGIC);
   END component;

   component clock_division generic(DIV  :  integer := 10);
                            port(   clk_in:  in std_logic;
                                    clk_out: out std_logic);
   END component;

   component keyboard port(row: out std_logic_vector(3 downto 0);
                           column: in std_logic_vector(3 downto 0);
                           clk: in std_logic;
                           output: out std_logic_vector(3 downto 0);
                           hit: out std_logic);
   END component;

   component sevensegment port(  clk: in std_logic;
                                 input: in std_logic_vector(3 downto 0);
                                 output: out bit_vector(7 downto 0);
                                 com: out std_logic_vector(3 downto 0);
                                 hit: in std_logic);
   END component;
```

معرفی پورت ها و genericهای componentهایی که نیاز داریم.

```
   signal column_stable: std_logic_vector(3 downto 0) := "0000";
   signal clk_keyboard: std_logic := '0';
   signal output_keyboard: std_logic_vector(3 downto 0) := "0000";
   signal hit: std_logic := '0';
```

تعریف سیگنال های کمکی، که بین پورت های ورودی و خروجی قطعه نیستند، ولی برای ارتباط بین componentها به آنها نیاز خواهد شد.

```
begin

   B0: debouncer generic map (10) port map(clk, column(0), column_stable(0));
   B1: debouncer generic map (10) port map(clk, column(1), column_stable(1));
   B2: debouncer generic map (10) port map(clk, column(2), column_stable(2));
   B3: debouncer generic map (10) port map(clk, column(3), column_stable(3));

   U1: clock_division generic map (25000) port map(clk, clk_keyboard);
   U2: keyboard port map(row, column_stable, clk_keyboard, output_keyboard, hit);
   U3: sevensegment port map(clk, output_keyboard, output, common, hit);

end Behavioral;
```

Port map کردن componentها، و مشخص کردن generic آنها.

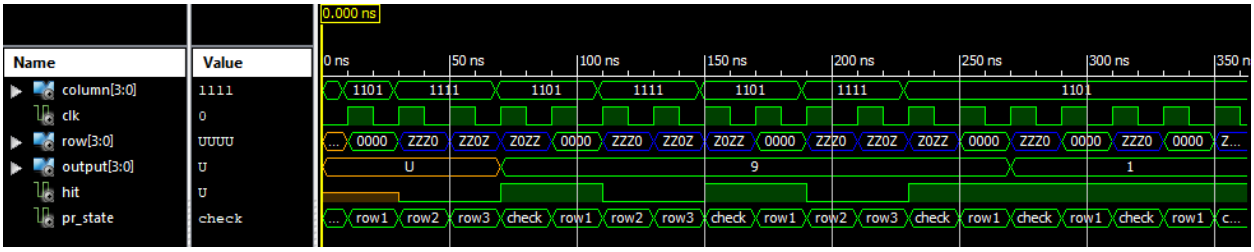
پاسخ مرحله 2

کد تست بنچ:

```
clk      <= not(clk) after 10 ns;
column   <= "1101" after 8 ns, "1111" after 28 ns, "1111" after 48 ns, "1101" after 68 ns,
            "1101" after 88 ns, "1111" after 108 ns, "1111" after 128 ns, "1101" after 148 ns,
            "1101" after 168 ns, "1111" after 188 ns, "1111" after 208 ns, "1101" after 228 ns;
```

باتوجه به کد نوشته شده، باید قبل اولین لبه بالارونده، عدد column دلخواه ولی مخالف "1111" باشد که از حالت check به row1 برویم. در حالات row1 و row2 باید "1111" باشد که وارد این حالات نشود. قبل اینکه وارد row3 شود (قبل چهارمین لبه بالارونده کلاک؛ یعنی بعد 68ns) باید column برابر "1101" باشد که معادل 9 است (سطر سوم، ستون دوم). و این سیکل تکرار شده تا output مدت طولانی تری برابر مقدار مطلوب باشد.

نتیجه:



مشاهده میشود در 70ns تا 270ns، output همان 9 شده است. ولی با لبه بالارونده در 230ns، وارد check میشود، و چون column ≠ 1111، وارد مرحله بعدی (row1) میشود، و چون column ≠ 1111، از بین حالات سطر اول، ستون دوم انتخاب میشود (بجای حالت مطلوب، یعنی سطر سوم، ستون دوم).

پاسخ مرحله 3

کد UCF:

```
net "clk" loc = p80;

net "row[0]" loc = p18;
net "row[1]" loc = p16;
net "row[2]" loc = p15;
net "row[3]" loc = p13;

net "column[0]" loc = p19;
net "column[1]" loc = p20;
net "column[2]" loc = p21;
net "column[3]" loc = p22;

net "output[0]" loc = p131;
net "output[1]" loc = p132;
net "output[2]" loc = p133;
net "output[3]" loc = p135;
net "output[4]" loc = p137;
net "output[5]" loc = p138;
net "output[6]" loc = p139;
net "output[7]" loc = p140;

net "common[0]" loc = p125;
net "common[1]" loc = p126;
net "common[2]" loc = p128;
net "common[3]" loc = p130;
```

برای clk از کلاک داخلی FPGA، برای row و column از keyboard، برای output و common از سون سگمنت استفاده شده است.