دانشکده مهندسی برق

Digital Thermometers and Thermostats with SPI Interface

احمدرضا محمدی سیاوشی

استاد درس:

دکتر ستار میرزاکوچکی

تدریس‌یارها:

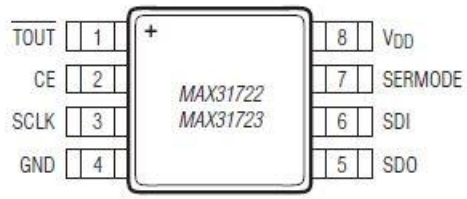
مهندس اصفهانی

مهندس پاکدامن

دی ماه 1401

قطعات MAX31722/MAX31723، دماسنج و ترموستات دیجیتالی هستند که SPI مربوط به آن ها را نوشته‌ایم. ولی قبل از نوشتن کد SPI مربوطه باید به یک سری نکات در دیتاشیت توجه کرد.

در شکل روبرو پین‌های این قطعه را مشاهده می‌کنید.



معرفی پین‌ها:

پین SERMODE، حالت serial-interface را مشخص می‌کند. به این صورت که اگر این پین به V_{DD} وصل باشد ارتباط SPI انتخاب می‌شود ولی اگر به زمین وصل شود ارتباط 3-wire انتخاب می‌شود.

CE: پین chip enable که برای شروع هر دو نوع ارتباط سریالی حتما باید active high شود.

SCLK: کلاکی که از بخش SPI به قطعه فرستاده می‌شود برای همگام‌سازی انتقال داده.

SDO و SDI: پین خروجی و ورودی داده‌های سریالی قطعه می‌باشد.

داده‌های دمایی به صورت مکمل دو در رجیستر دما ذخیره می‌شوند. برای ارتباط SPI، اول پرارزش‌ترین بیت فرستاده می‌شود و این بیت (پرارزش) در رجیستر دما مربوط به بیت علامت می‌باشد. قالب رجیستر دما به صورت شکل زیر می‌باشد.

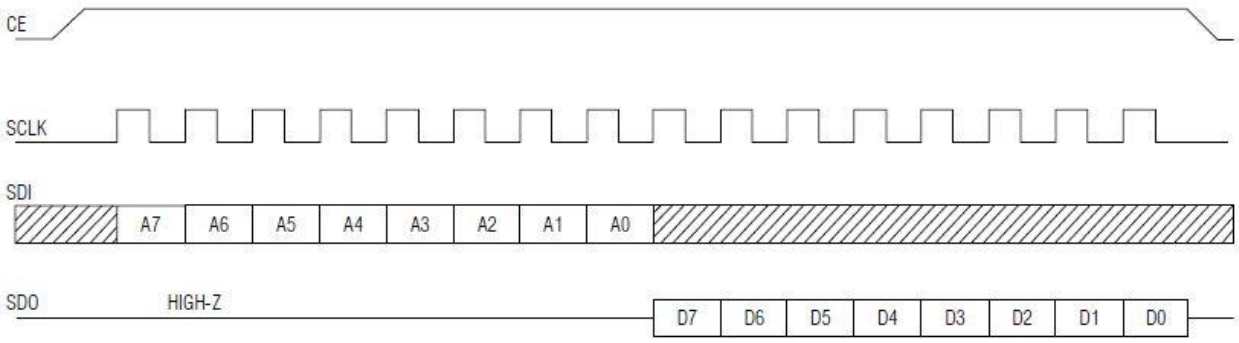
S	2^6	2^5	2^4	2^3	2^2	2^1	2^0	02h
MSB				(UNITS = °C)			LSB	
2^{-1}	2^{-2}	2^{-3}	2^{-4}	0	0	0	0	01h

دقت این دماسنج از 0.5 تا 0.0625 قابل انتخاب می‌باشد که باید از طریق رجیستر Configuration/Status تنظیم شود. نمونه‌ای از تبدیل دما به باینری که خود قطعه انجام می‌دهد را در شکل زیر مشاهده می‌کنید.

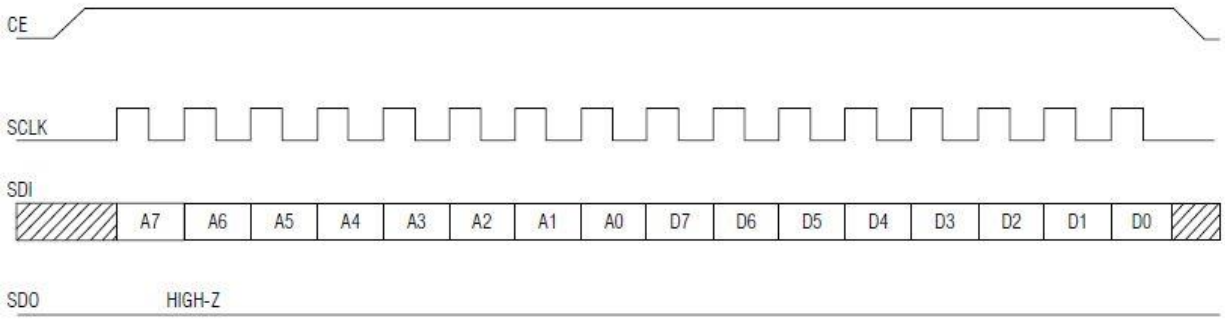
TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0111 1101 0000 0000	7D00
+25.0625	0001 1001 0001 0000	1910
+10.125	0000 1010 0010 0000	0A20
+0.5	0000 0000 1000 0000	0080
0	0000 0000 0000 0000	0000
-0.5	1111 1111 1000 0000	FF80
-10.125	1111 0101 1110 0000	F5E0
-25.0625	1110 0110 1111 0000	E6F0
-55	1100 1001 0000 0000	C900

بعد از اینکه CE فعال شود بایت آدرس، اولین داده‌ایست که از SPI به دیوایس داده می‌شود تا مشخص کند که از/ روی آن آدرس می‌خواهد بخواند یا بنویسد. اگر پرارزش‌ترین بیت آدرس A7 برابر صفر باشد از آدرس مورد نظرکه یک رجیستر را نشان می‌دهد شروع به خواندن می‌کند و اگر A7 برابر یک باشد روی آدرس مورد نظر شروع به نوشتن می‌کند.

عملیات Read:



عملیات Write:



همانطور که در دو شکل بالا می‌بینید خروجی دیوایس (SDO) که ورودی SPI می‌باشد وقتی که CE فعال نباشد یا عملیات نوشتن اتفاق میافتد High impedance می‌شود.

CPOL یا clock polarity را در شبیه‌سازی یک در نظر گرفتیم.

برای راه اندازی این دیوایس، SPI ای که در ادامه بیشتر توضیح خواهم داد را مدل کردم.

```
Port (
        -- Inputs
        CLK_SYS : in  STD_LOGIC;
        start : in  STD_LOGIC;
        SDO : in STD_LOGIC;
        reset : in STD_LOGIC;
        -- Outputs
        CE : out  STD_LOGIC;
        SCK : out  STD_LOGIC;
        SDI : out STD_LOGIC);
```

پورت‌ها به صورت روبرو تعریف شده‌اند. پورت SDI که خروجی تعریف شده است درواقع ورودی دیوایس می‌باشد و پورت SDO که ورودی تعریف شده است خروجی دیوایس می‌باشد.

همانطور که در شکل زیر می‌بینید دو سیگنال هشت بیتی تعریف کرده‌ایم؛ یکی برای بایت آدرس و یکی هم برای بایت دیتا. ماشین حالتی که تعریف کرده‌ام دارای 6 حالت می‌باشد و همچنین یه سیگنال شمارنده نیز تعریف شده‌است.

```
-- In/Outs
signal start_int : STD_LOGIC := '0';
signal reset_int : STD_LOGIC := '0';
signal SCK_int : STD_LOGIC := 'Z';
signal CE_int : STD_LOGIC := '0';
signal SDI_int : STD_LOGIC := '0';
signal SDO_int : STD_LOGIC := 'Z';
signal add_byte_int : STD_LOGIC_VECTOR (7 downto 0);
signal data_byte_int : STD_LOGIC_VECTOR (7 downto 0);
-- Control Signals
signal bit_cnt : unsigned (3 downto 0) := "0000";
-- states
type FSM is (idle,starter,starter_dly,read_data,write_data,rw_dly);
signal state : FSM := idle;

constant add_byte : STD_LOGIC_VECTOR (7 downto 0) := "10100101";
constant data_byte : STD_LOGIC_VECTOR (7 downto 0) := "01010101";
```

در شکل روبرو، قبل از شروع process که حساسیت آن را CLK_SYS تعریف کردیم سیگنال‌های داخلی را به پورت‌های خروجی می‌دهیم و پورت‌های ورودی را به سیگنال‌های داخلی مربوطه ارجاع می‌دهیم. همانطورکه مشخص است سیگنال reset به صورت سنکرون پیاده‌سازی شده است. اگر بعد از لبه بالارونده کلاک اصلی، سیگنال reset برابر یک شود مقادیر اولیه سیگنال‌ها را به آنها می‌دهد و اگر برابر صفر بود مقداری

```
37  begin
38
39      CE <= CE_int;
40      SDO_int <= SDO;
41      SDI <= SDI_int;
42      SCK <= SCK_int;
43      reset_int <= reset;
44
45      process (CLK_SYS)
46      begin
47
48          if (rising_edge (CLK_SYS)) then
49              if (reset_int = '1') then
50                  CE_int <= '0';
51                  state <= idle;
52                  SDI_int <= '0';
53                  SDO_int <= 'Z';
54              else
55                  add_byte_int <= add_byte;
56                  start_int <= start;
```

که در ثابت بایت آدرس قرار دارد را بر روی سیگنال آن قرار می‌دهد.

```
58              case state is
59                 when idle =>
60                    bit_cnt <= "0111";
61
62                    if (start_int = '1') then
63                       state <= starter;
64                       CE_int <= '1';
65                    else
66                       state <= idle;
67                       CE_int <= '0';
68                    end if;
```

در چند خط بالاتر وضعیت پیش‌فرض ماشین حالت را روی idle قرار داده بودیم پس SPI تا وقتی‌که سیگنال start آن برابر یک نشود در این حالت می‌ماند ولی اگر برابر یک شود به حالت starter می‌رود و سیگنال CE را یک و دیوایس را فعال می‌کند. همچنین قبل از چک کردن دستور if، مقدار "0111" را روی سیگنال شمارنده قرار دادیم.

```
when starter =>
   CE_int <= '1';
   SDI_int <= add_byte_int (to_integer(bit_cnt));


   if (bit_cnt /= 0) then
      state <= starter;
      bit_cnt <= bit_cnt - 1;
   else
      bit_cnt <= "1000";
      state <= starter_dly;
```

اولین کاری که بعد از فعال شدن CE اتفاق می‌افتد این است که مقدار سیگنال بایت آدرس را روی خروجی SDI قرار می‌دهیم و تا وقتی‌که شمارنده صفر نشده در این حالت باقی می‌ماند و بعد از صفر شدن به حالت بعدی می‌رود و مقدار "1000" را در شمارنده قرار می‌دهد.

```
when starter_dly =>
   CE_int <= '1';
   SDI_int <= 'Z';
   bit_cnt <= bit_cnt - 1;
      if (add_byte_int(7) = '0') then
      state <= read_data;

      else
      state <= write_data;
      data_byte_int <= data_byte;
      end if;
```

هدف از ایجاد این حالت این است که تاخیری بین بایت ادرس و ارسال یا دریافت بایت دیتا اتفاق بیافتد. وضعیت بعدی را با یک دستور شرطی بر روی پرارزش‌ترین بیت بایت آدرس مشخص می‌کنیم. اگر صفر باشد به وضعیت read_data می‌رود در غیر اینصورت به وضعیت write_data می‌رود و ثابت بایت دیتا را بر روی سیگنال آن قرار می‌دهد.

در حالت read_data سیگنال ورودی SDO را به سیگنال بایت دیتا ارجاع می‌دهیم و از پرارزش‌ترین بیت، شروع به پر کردن آن می‌کند و وقتی‌که به شمارنده صفر شود به حالت rw_dly می‌رود.

```
when read_data =>
   CE_int <= '1';
   data_byte_int (to_integer(bit_cnt)) <= SDO_int;

   if (bit_cnt /= 0) then
      state <= read_data;
      bit_cnt <= bit_cnt - 1;
   else
      state <= rw_dly;
   end if;
```

و در حالت write_data سیگنال بایت دیتا را از پرارزش‌ترین بیت آن بر روی سیگنال خروجی SDI قرار می‌دهد تا وقتی‌که شمارنده صفر شود و به وضعیت بعدی برود.

```
when write_data =>

   CE_int <= '1';
   SDI_int <= data_byte_int (to_integer(bit_cnt));

   if (bit_cnt /= 0) then
      state <= write_data;
      bit_cnt <= bit_cnt - 1;
   else
      state <= rw_dly;
   end if;
```

حالت rw_dly را برای تاخیری بین حالت خواندن یا نوشتن با حالت بعدی ایجاد کردم. در آن مقدار "0111" به شمارنده داده و CE برابر صفر می‌شود. حالت idle حالت بعدی ماشین می‌باشد یعنی به حالت اولیه می‌رود و منتظر سیگنال start می‌ماند.

```
when rw_dly =>
   state <= idle;
   CE_int <= '0';
   bit_cnt <= "0111";
```

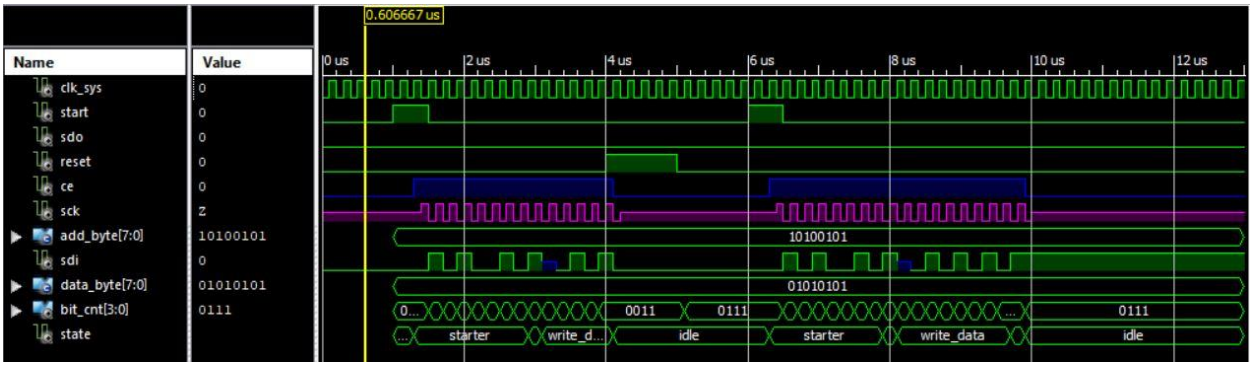
همانطور که مشخص است وقتی سیگنال CE برابر یک است و وضعیت در idle نباشد، خروجی سیگنال SCK که به کلاک دیواس داده می‌شود برابر با خلاف CLK_SYS می‌باشد در غیر اینصورت High-impedance می‌شود.

```
if (CE_int = '1' and state /= idle) then
   SCK_int <= not (CLK_SYS);
else
   SCK_int <= 'Z';
end if;
```

در شکل صفحه بعد که بخشی از کد testbench را نشان می‌دهد CLK_SYS_period برابر 200ns می‌باشد چون بیشترین فرکانسی که در دیتاشیت برای دیوایس مشخص شده بود برابر 5 MHz بود به همین خاطر این فرکانس را هم به SPI می‌دهیم هم به دیوایس. همچنین در پراسس start_pro ، سیگنالهای start و reset را مقداردهی کردیم.

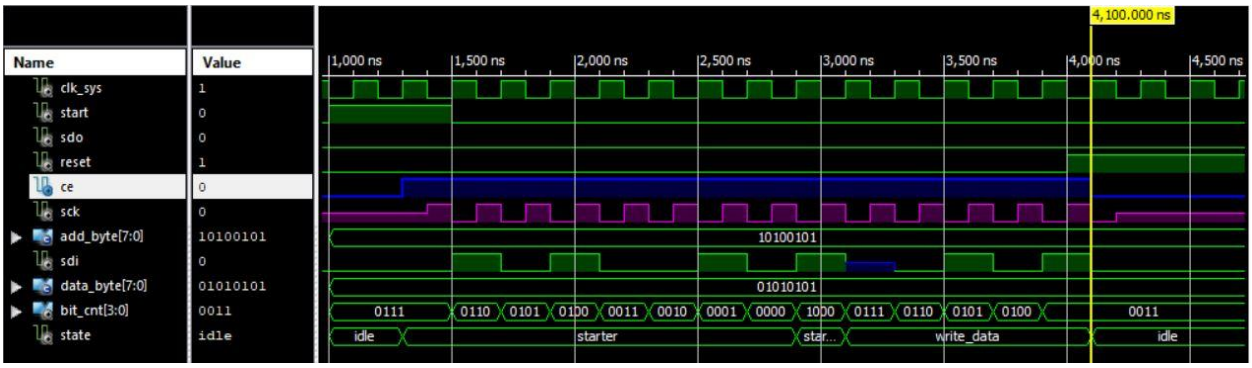
```
39    constant CLK_SYS_period : time := 200 ns;
40
41  BEGIN
42
43     -- Instantiate the Unit Under Test (UUT)
44     uut: entity work.SPI PORT MAP (
45            CLK_SYS => CLK_SYS,
46            start => start,
47            SDO => SDO,
48            reset => reset,
49            CE => CE,
50            SCK => SCK,
51            SDI => SDI
52          );
53
54     -- Clock process definitions
55     CLK_SYS_process :process
56     begin
57        CLK_SYS <= '0';
58        wait for CLK_SYS_period/2;
59        CLK_SYS <= '1';
60        wait for CLK_SYS_period/2;
61     end process;
62
63     start_pro: process
64     begin
65        start <= '0', '1' after 1us, '0' after 1.5us, '1' after 6us, '0' after 6.5us;
66        reset <= '0', '1' after 4us, '0' after 5us;
67        wait;
68     end process start_pro;
```

با توجه به کد اول و مقدار دهی در testbench تصویر زیر، خروجی شبیه‌سازی را نشان می‌دهد.

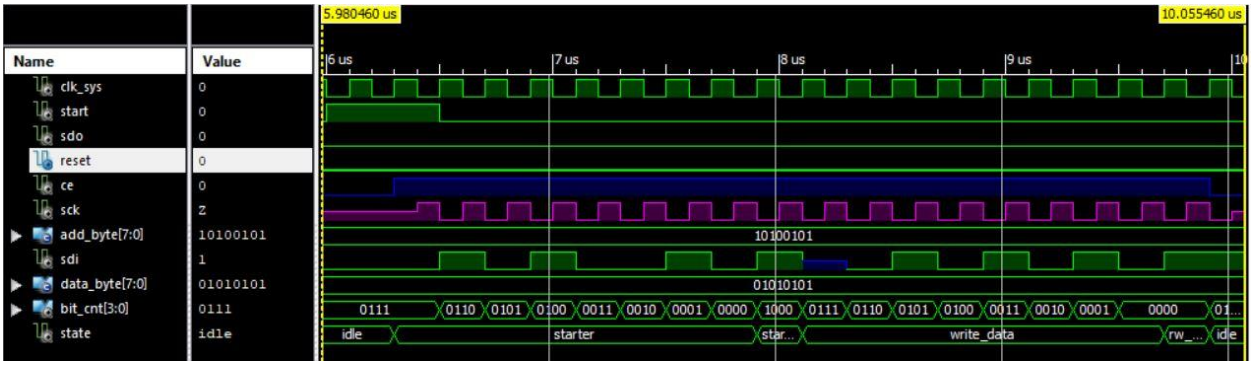


با یک شدن سیگنال start در لحظه 1us، CE در کلاک بعدی برابر یک می‌شود و SCK به دیوایس داده می‌شود. همچنین از حالت idle به حالت starter می‌رود و سیگنال بایت ادرس به SDI داده می‌شود. از انجاییکه مقدار بایت آدرس را به صورت فرضی "10100101" داده‌ایم و A7 = 1 میباشد بنابراین پس از حالت starter_dly به حالت write_data می‌رود و سیگنال بایت دیتایی که به صورت فرضی مقداردهی کرده بودیم در خروجی SDI قرار می‌گیرد و شروع به نوشتن بر روی دیوایس می‌کند. اما این حالت به انتها نمی‌رسد و به حالت idle می‌رود چون در لحظه 4us سیگنال reset یک می‌شود و مقادیری که مشخص کرده بودیم به سیگنال‌ها داده می‌شود البته در لبه بالارونده کلاک بعدی زمانیکه سیستم، سیگنال reset را تشخیص دهد. برای

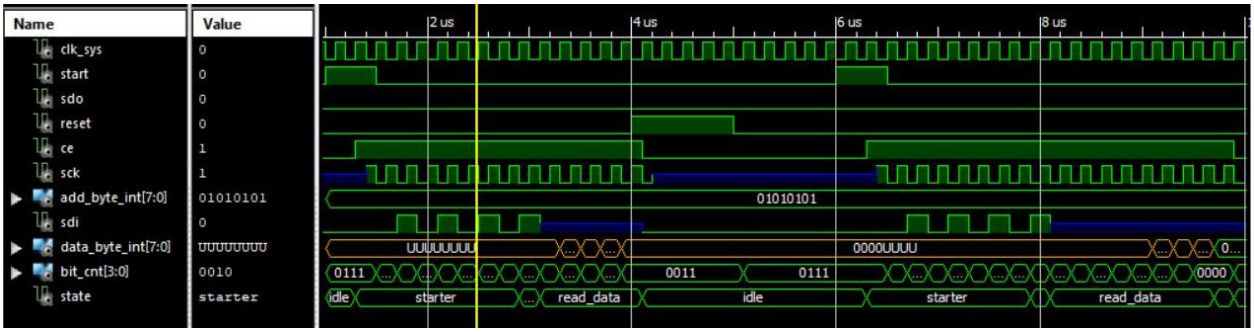
مثال سیگنال SCK برابر Z و CE و SDI برابر صفر می‌شود. در شکل صفحه بعدی به طور دقیقتر می‌توان اتفاقات این لحظه را دید.



دوباره در لحظه 6us، مقدار سیگنال start یک می‌شود و به حالت starter می‌رود و مراحل ذکر شده تکرار می‌شود و این بار حالت write_data به انتها می‌رسد و بعد از حالت rw_dly به حالت idle می‌رود و منتظر سیگنال start می‌ماند. در شکل زیر، این مراحل را به وضوح می‌توان دید.



حال اگر به بایت آدرس مقدار "01010101" بدهیم بعد از حالت starter_dly به حالت read_data می‌رود ولی از انجاییکه پورت SDO به طور واقعی به دیوایس وصل نیست مقدار آن در همه لحظات صفر می‌باشد.



بنابراین بعد از آنکه به حالت read_data می‌رود سیگنال SDO مقدار صفر را با هر شمارش درون سیگنال بایت دیتا می‌ریزد. در دو شکل زیر به طور واضح‌تر شکل بالا را می‌بینیم.