دانشکده مهندسی برق

گزارش پروژه درس VHDL

راه اندازی و شبیه سازی واحد SPI قطعه ADF4351

علی رضازاده – ۴۰۱۶۱۱۱۴۸

استاد:

دکتر ستار میرزاکوچکی

دی ماه 1401

فهرست مطالب

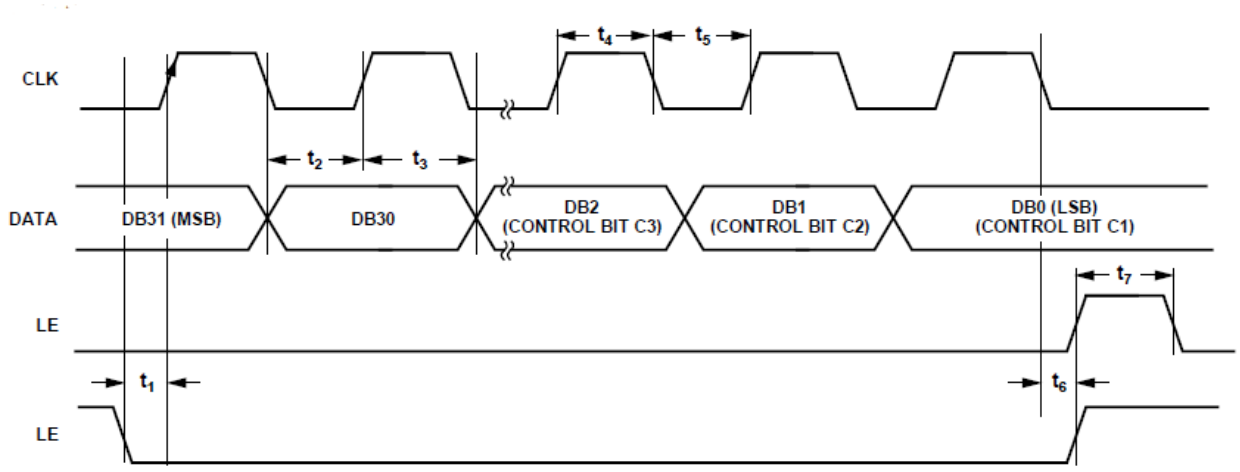
۱: مقدمه

در این بخش، به توضیحاتی ابتدایی در رابطه با واحد SPI قطعه ADF4351 می‌پردازیم.

ارتباط سریال این قطعه از ساختار ساده‌ای تشکیل شده است. پایه‌های مربوط به ارتباط سریال این قطعه عبارتند از CLK، LE و DATA. یکی دیگر از پایه‌های مهم مربوط به CE می‌باشد که در صورت فعال نبودن قطعه کاری انجام نخواهد داد.

قطعه ADF4351 دارای یک رجیستر ۳۲ بیتی بوده که اطلاعات دریافتی از طریق ارتباط سریال را در آن ذخیره می‌کند، سپس با توجه به بیت‌ها اطلاعات مورد نظر را به رجیستر دیگری فرستاده و یا کار خاصی انجام می‌دهد که مربوط به حوزه بررسی ما نمی‌باشند.

وظیفه پایه LE نیز به این شکل است که هنگامی که مقدار '0'رو آن قرار گرفت، ارسال ۳۲ بیت شروع شده و پس از آخرین کلاک دوباره مقدار '1' روی آن قرار می‌گیرد. شکل زیر نحوه عملکرد پایه‌ها را به خوبی نشان می‌دهد.



شکل ۱) دیاگرام زمانی مربوط به پایه‌های ارتباط سریال

طبق زمان بندی‌های مشخص شده در جدول دیتا شیت، دوره تناوب مناسب برای این واحد ۵۰ نانو ثانیه بوده که معادل فرکانس کاری 20 Mbps برای واحد SPI می‌باشد.

۲: تشریح کد

کد مربوطه در برنامه ISE از شرکت Xilinx نوشته و شبیه‌سازی شده است.

در اولین بخش کد کتابخانه‌های مورد نیاز برای انجام پروژه را اضافه می‌کنیم. در شکل زیر این کتابخانه‌ها قابل مشاهده می‌باشند.

```
1   -- Required Libraries
2   library IEEE;
3   USE IEEE.STD_LOGIC_1164.ALL;
4   USE IEEE.NUMERIC_STD.ALL;
```

شکل ۲) کتابخانه‌های افزوده شده

۲-۱: موجودیت

در این بخش ابتدا کتابخانه استاندارد IEEE افزوده شده، سپس با افزودن بخش‌های بعد قابلیت استفاده از تایپ‌های Std_Logic و عملیات‌های محاسباتی را فراهم ساخته‌ایم.

بخش بعدی کد مربوط به موجودیت پروژه بوده که در آن پورت‌های مورد نظر را تعریف می‌کنیم. پورت‌های Start و Rst تنها برای بررسی شبیه‌سازی بوده و در قطعه اصلی وجود خارجی ندارند. با '1'شدن Start شبیه سازی آغاز شده و Rst نیز عملیات ارسال را متوقف می‌کند.

```
6   entity Spi_module is
7      port( -- inputs --
8            Clk_sys  :in   std_logic;  -- Master Clock
9            Start    :in   std_logic;  -- Additional bit to start the process (not actual pin on device)
10           Rst      :in   std_logic;  -- a  synchronous Reset Pin (not actual pin on device)
11           -- outputs --
12           SCK      :out  std_logic;  -- Slave Clock
13           LE       :out  std_logic;  -- LE pin of the Slave (Read Report for more information)
14           CE       :out  std_logic;  -- Chip Enable of the Slave
15           MOSI     :out  std_logic); -- Data port of the Slave in Data Sheet
16  end Spi_module;
```

شکل ۳) موجودیت واحد SPI نوشته شده

*توضیحات مربوط به باقی پایه‌ها در بخش مقدمه داده شده است.

پس از پایان بخش موجودیت، بخش معماری کد نوشته شده را مورد بررسی قرار می‌دهیم.

۲-۲: معماری

بخش معماری به دو قسمت تقسیم می‌شود. بخش ابتدایی مربوط به تعریف سیگنال‌ها و مشخص کردن برخی از پارامترها بوده که با نام Declarative Part شناخته می‌شود.

```
18  architecture test of Spi_module is
19   -- In/Out signals --
20   signal  Start_s  :  std_logic := '0';
21   signal  Rst_s    :  std_logic := '0';
22   signal  LE_s     :  std_logic := '1';
23   signal  CE_s     :  std_logic := '0';
24   signal  MOSI_s   :  std_logic := '0';
25   signal  SCK_s    :  std_logic := 'Z';
26   signal  Data_s   :  std_logic_vector (31 downto 0) := (others => '0');
27   -- Data --
28   constant Data    :  std_logic_vector (31 downto 0) := "01101001000101111001011010010011";
29   -- Control signals --
30   signal  Cnt      :  unsigned (4 downto 0) := "11111";
31   -- States --
32   type    Mode  is (Idle, Send, delay_ins, delay_ce);
33   signal  State    :  Mode := Idle;
```

شکل ۴) بخش Declarative معماری

در قسمت ابتدایی، سیگنال اتصالی پورت‌ها تعریف شده است. این عملیات در شبیه‌سازی تأثیر خاصی نداشته اما در بهبود عملیات روتینگ توسط خود برنامه اثر قابل توجهی دارد. بیت‌هایی که قرار است ارسال شوند توسط دستور constant به شکل یک مقدار ثابت تعریف شده که بعدا آن را روی سیگنال Data_s قرار می‌دهیم. سیگنال Cnt نیز یک عدد ۵ بیتی با مقدار اولیه ۳۱ بوده که تعداد بیت‌هایی که قرار است ارسال شوند را مشخص می‌کنند. پس از ارسال هر بیت مقدار آن یکی کم می‌شود تا در نهایت برابر صفر بشود. صفر شدن این سیگنال تمام شدن عملیات ارسال را نشان داده و همانطور که مشاهده کردیم باید پس از آن سیگنال مربوط به LE دوباره ‘1’ بشود. این موارد را در بخش‌های بعد مشاهده خواهیم کرد. در بخش States ابتدا تایپی جدید به نام Mode را تعریف کرده که نشان دهنده هر مرحله از انجام کل عملیات می‌باشد. سیگنال State نیز از همین تایپ تعریف شده و مقدار اولیه Idle را به آن داده‌ایم. هنگامی که در حالت Idle هستیم هیچ کار خاصی را انجام نمی‌شود و تنها سیگنال‌های کنترلی برای شروع عملیات بررسی می‌شوند.

بخش بعدی نیز مربط به قسمت Instantiation بوده که بیشتر بخش کد را به خود اختصاص داده است. این قسمت را ریز به ریز مورد بررسی قرار می‌دهیم.

```
35  begin
36   -- signals to ports --
37     CE    <= CE_s;
38     LE    <= LE_s;
39     MOSI  <= MOSI_s;
40     SCK   <= SCK_s;
41     Rst_s <= Rst;
```

شکل ۵) شروع بخش Instantiation

دستور Begin نشان دهنده شروع این بخش از معماری می‌باشد، سپس سیگنال‌های تعریف شده در بخش پیشین را به پورت‌های مربوطه متصل کرده‌ایم.

```
42  -- Main Process --
43    Process (Clk_sys) Begin
44
45       if (Clk_sys ='1' and Clk_sys'event) then  -- Rising edge of Clock --
46
47          if (Rst_s ='1') then
48          state    <= Idle;
49          CE_s     <= '0';
50          LE_s     <= '1';
51          else
52             Data_s   <= Data;
53             Start_s  <= Start;
```

شکل ۶) شروع بخش Process

بدنه اصلی کد درون یک Process قرار دارد که حساس به تغییرات کلاک سیستم می‌باشد. شرط درون دستورif بالا رونده بودن لبه کلاک را بررسی می‌کند و در صورت بر قراری شرط دستورات درون آن اجرا می‌شود. در اولین مرحله بالاترین اولویت بررسی مربوط به Reset سنکرون می‌باشد. در صورتی که این سیگنال مقداری برابر ’1‘داشت، سیگنال‌های اصلی را ریست کرده و به حالت Idle که حالت اولیه می‌باشد برمی‌گردیم و تا زمانی که دوباره دستور Start داده نشود در این حالت باقی می‌مانیم.

در صورتی که شرط Reset برقرار نبود دستور Case مورد بررسی قرار می‌گیرید. دستور Case حالت‌های مختلف سیگنال State را چک کرده و با توجه به حالتی که در آن قرار دارد دستورات مربوطه را اجرا می‌کند. سیگنال State در حالت اولیه خود Idle می‌باشد. در این State سیگنال‌ها در حالت اولیه خود قرار داشته و شمارنده مقدار اولیه خود را دارد. در صورت ’1‘شدن مقدار سیگنال Start، فرایند ارسال دیتا شروع می‌شود. در این مرحله با ’1‘ کردن CE و ’0‘ کردن LE فرایند را آغاز کرده و سپس وارد حالت Delay_ins می‌شویم. در صورت ’0‘ بودن Start نیز در حالت اولیه Idle باقی می‌مانیم.

```
54              Case State is  -- Taking actions based on State value
55                 when Idle      =>
56                    MOSI_s   <= '0';
57                    Cnt      <= "11111";
58                    Ce_s     <= '0';
59                    SCK_s    <= 'Z';
60                    LE_s     <= '1';
61                    if (Start_s = '1') then -- checks if process should start or not
62                       State <= delay_ins;
63                       CE_s  <= '1';
64                       LE_s  <= '0';
65                    else                    -- if the previous condition is not true
66                       State <= Idle;       --- remain in the Idle Mode
67                       CE_s  <= '0';
68                       LE_s  <= '1';
69                    end if;
```

شکل ۷) بررسی حالت Idle

در مرحله Delay_ins اولین بیت آماده ارسال شده و روی MOSI قرار می‌گیرد، یکی از شمارنده کم شده و State وارد مرحله Send می‌شود. در صورت نبود این مرحله یک بیت کمتر برای Slave فرستاده خواهد شد.

```
71              when delay_ins =>
72                 State    <= Send;
73                 CE_s     <= '1';
74                 MOSI_s   <= Data_s (to_integer(Cnt)); -- puts data on MOSI
75                 Cnt      <= Cnt - 1;
76                 LE_s     <= '0';
```

شکل ۸) بررسی حالت Delay_ins

در مرحله Send یک شرط برای بررسی مقدار شمارنده وجود دارد و عملیات انتقال را تا زمانی که به صفر نرسیده انجام می‌دهد. دستور to integer مقدار binary این شمارنده را به integer تبدیل می‌کند. زمانی که مقدار Cnt برابر صفر شد آنرا به مقدار اولیه خود باز گردانده و وارد مرحله نهایی یعنی Delay_ce می‌شویم.

```
78              when Send      =>
79                 MOSI_s   <= Data_s (to_integer(Cnt));
80                 CE_s     <= '1';
81                 LE_s     <= '0';
82                 if (cnt /= 0) then -- checks if all bits have been sent
83                    State <= Send;
84                    Cnt   <= Cnt -1;
85                 else
86                    State <= delay_ce;
87                    Cnt   <= "11111";
88                 end if;
```

شکل ۹) بررسی حالت Send

مرحله Delay_ce نیز زمان کافی برای انتقال آخرین بیت را فراهم کرده و دقت شود که سیگنال‌های کنترلی مقدار خود را حفظ می‌کنند. پس از این مرحله دوباره وارد حالت Idle شده تا زمانی که دوباره سیگنال Start مقدار ’1‘ گرفته و دیتای بعدی طبق مراحل شرح داده شده ارسال شود.

```
90                  when delay_ce  =>    -- sending is complete and go back to
91                     State    <= Idle; --- the Idle State
92                     CE_s     <= '1';
93                     LE_s     <= '0';
94                     Cnt      <= "11111";
95                End Case;
96             End if;
```

شکل ۱۰) بررسی حالت Delay_ce

تنها بخش باقی مانده تولید کلاک Slave می‌باشد. از آنجایی که تولید کلاک باعث مصرف توان شده نمی‌توان این کلاک را همیشه فعال نگه داشت. به همین دلیل تا زمانی که در مرحله Idle هستیم کلاکی برای Slave ارسال نمی‌کنیم. با خارج شدن از حالت Idle و فعال شدن CE این کلاک را با اختلاف فاز ۱۸۰ درجه نسبت به کلاک سیستم به قطعه مورد نظر می‌دهیم. این کار باعث می‌شود تا دیتا در پایدار ترین زمان ممکن توسط Slave خوانده شود.

```
98          if (CE_s = '1' and state /= Idle)then -- When should give clock to slave
99             SCK_s <= not(CLK_sys); -- Best phase for slave clock
100         else
101            SCK_s <= 'Z';
102         end if;
103
104      End Process;
105   end test;
```

شکل ۱۱) تولید سیگنال SCK

۳: تست بنچ

تست بنچ بر خلاف کد اصلی تنها از بخش معماری تشکیل شده و موجودیت ندارد. در این بخش به سیگنال‌ها مقادیری برای بررسی شبیه سازی و اطمینان از صحت عملکرد کد داده می‌شود. در بخش Declarative سیگنال‌ها را تعریف و مقدار اولیه می‌دهیم. دوره تناوب کلاک اصلی را نیز به شکل یک مقدار ثابت تعریف کرده‌ایم.

```
5   ENTITY Spi_tb IS
6   END Spi_tb;
7
8   ARCHITECTURE behavior OF Spi_tb IS
9
10      -- Component Declaration for the Unit Under Test (UUT)
11
12     --Inputs
13     signal Clk_sys : std_logic := '0';
14     signal Start   : std_logic := '0';
15     signal SCK     : std_logic := '0';
16     signal Rst     : std_logic := '0';
17
18     --Outputs
19     signal LE : std_logic:='1';
20     signal CE : std_logic:='0';
21     signal MOSI : std_logic;
22     --inner
23
24     -- Clock period definitions
25     constant Clk_sys_period : time := 50 ns;
```

۱۲) بخش Declarative معماری تست بنچ

در بخش ابتدایی Instantiation معماری تست بنچ، قطعه طراحی شده در بخش پیشین را از کتابخانه Work فرخوانی کرده و سیگنال‌های تعریف شده را به پورت‌های معادل متصل می‌کنیم.

```
28      -- Instantiate the Unit Under Test (UUT)
29      uut:entity work.Spi_module PORT MAP (
30             Clk_sys => Clk_sys,
31             Start => Start,
32             SCK => SCK,
33             LE => LE,
34             CE => CE,
35             MOSI => MOSI,
36             Rst => Rst
37           );
```

۱۳) اتصال سیگنال‌ها به پورت‌های قطعه

در آخر نیز به دو سیگنال Start و Rst سیگنال هایی را داده که در شبیه سازی می‌توان نتیجه آنرا مشاهده کرد. سیگنال کلاک سیستم نیز بر اساس دوره تناوب تعریف شده ساخته می‌شود. به این شکل که نصف دوره تناوب مقدار ’1‘ و نصف دیگر مقدار ’0‘ به آن داده شده است.

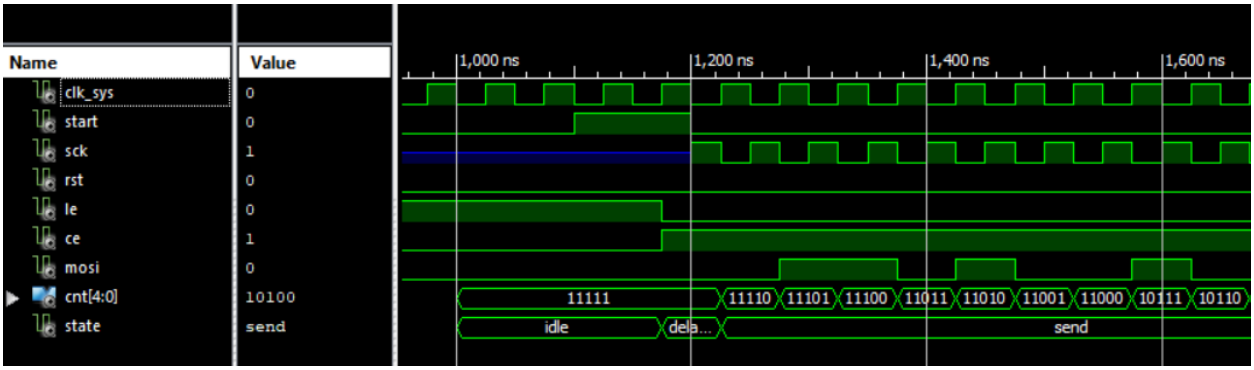
```
39      Start <='0', '1' after 100 ns, '0' after 200 ns,'1' after 1000 ns, '0' after 1100 ns;
40      Rst <='0', '1' after 800 ns, '0' after 900 ns;
41
42    -- Clock process definitions
43    Clk_sys_process :process
44    begin
45       Clk_sys <= '0';
46       wait for Clk_sys_period/2;
47       Clk_sys <= '1';
48       wait for Clk_sys_period/2;
49    end process;
50
```

۱۴) مقدار دهی به سیگنال‌ها و کلاک سیستم

۴: شبیه‌سازی

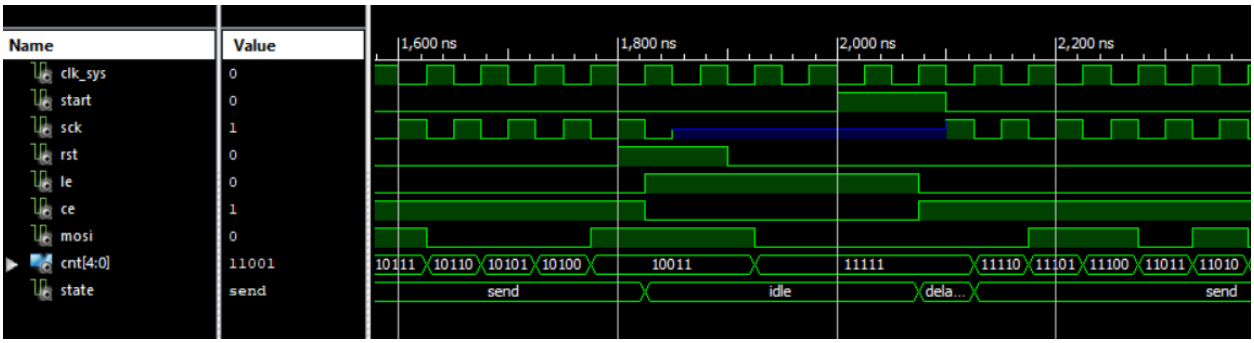
برای مشاهده نتیجه خروجی نیز تنها کافیست از بخش Simulation فایل تست بنچ نوشته شده را اجرا کرده، سیگنال‌های اضافی جهت بررسی را از بخش uut افزوده و با گذراندن زمان خروجی را مشاهده کنیم.

همانطور که در نتیجه شبیه‌سازی مشاهده می‌کنیم تا زمانی که سیگنال Start مقدار ’0‘ داشته در حالت Idle باقی مانده‌ایم. پس از ’1‘ شدن Start سیگنال‌های کنترلی و کلاک Slave فعال شده و به State بعدی رفته‌ایم. باقی مراحل نیز همانطور که در تشریح کد بیان شد مرحله به مرحله طی شده است.

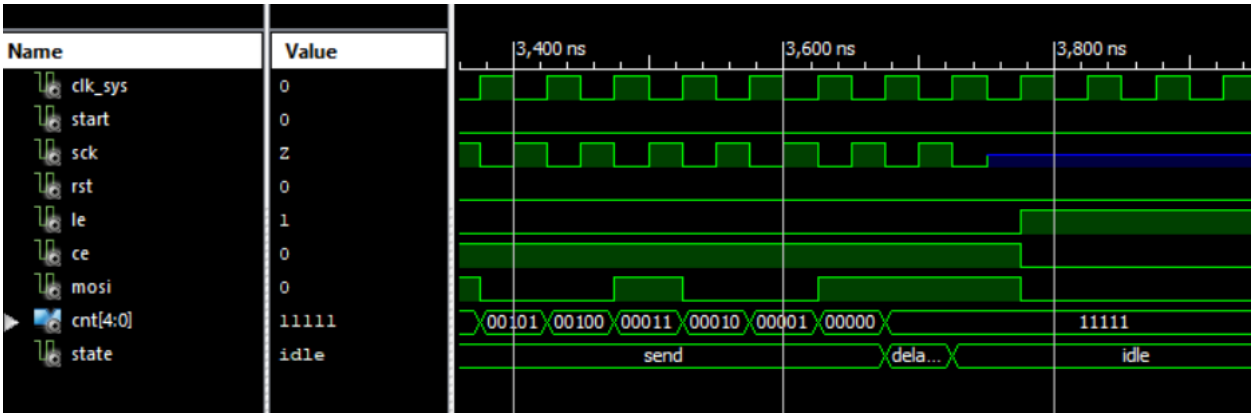


۱۵) بخش ابتدایی نتیجه شبیه‌سازی

در بخش زیر نیز نحوه عملکرد پایه ریست را مشاهده می‌کنیم. با فعال شدن ریست به حالت Idle رفته، سیگنال‌ها مقدار اولیه خود را دریافت کرده و سیگنال کلاک Slave را نیز قطع می‌کنیم. با دریافت دوباره سیگنال Start ارسال دیتا برای بار دیگر انجام شده و پس از نهایی شدن عملیات متوقف می‌شود.



۱۶) نحوه عملکرد Reset



۱۷) پایان ارسال دیتا در شبیه‌سازی