
بسم الله الرحمن الرحيم

دانشکده مهندسی برق

پیاده سازی و شبیه سازی ارتباط SPI برای ماژول AD9255

پروژه درس VHDL

مهندسی برق سیستم های دیجیتال

استاد :

دکتر ستار میرزا کوچکی

نویسنده : علی زارع

بهمن 1401

تشکر و قدردانی :

تشکر از جناب دکتر ستار میرزاکوچکی و مهندس اصفهانی بابت راهنمایی هایشان در انجام این تحقیق .

چکیده

در این پروژه قصد داریم ارتباط سریال میان ماژول AD9255 که یک مبدل آنالوگ به دیجیتال است را شبیه سازی کنیم. ارتباط سریال انتخابی به صورت SPI می باشد . SPI یک رابط همزمان و دوطرفه کامل مبتنی بر زیرگره اصلی است. داده های اصلی یا فرعی در لبه ساعت در حال افزایش یا کاهش همگام سازی می شوند. هر دو گره اصلی و فرعی می توانند داده ها را به طور همزمان انتقال دهند. رابط SPI می تواند 3 سیم یا 4 سیم باشد. ماژول AD9255 ارتباط سریال سه سیمه را پشتیبانی می کند که شامل سه پین مشترک SCLK، Cs و SDIO می باشد.

کلمات کلیدی : ارتباط SPI ، SCLK، Cs ، SDIO و مبدل آنالوگ به دیجیتال

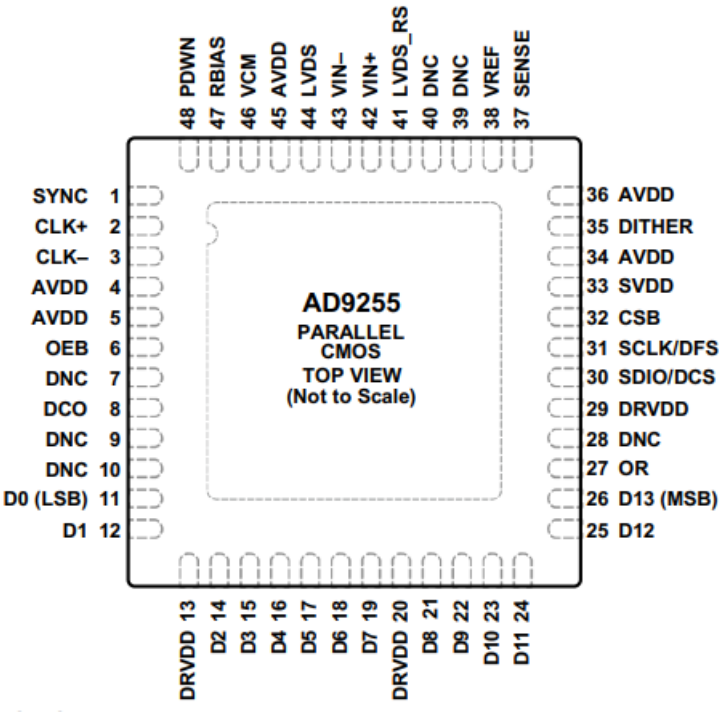
فهرست مطالب

فصل اول

مقدمه

1-1 مبدل آنالوگ به دیجیتال AD9255

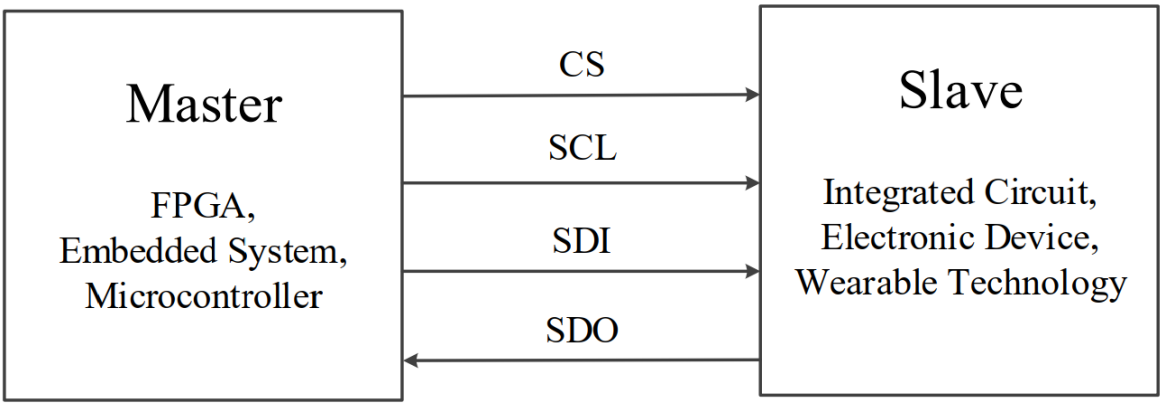
AD9255 یک مبدل آنالوگ به دیجیتال 14 بیتی با نرخ 125MSPS است. AD9255 برای پشتیبانی از برنامه های ارتباطی طراحی شده است که در آن عملکرد بالا همراه با هزینه کم، اندازه کوچک و تطبیق پذیری مورد نظر است. برنامه نویسی برای راه اندازی و کنترل با استفاده از یک رابط سریال 3 سیمه سازگار با SPI انجام می شود. گزینه های انعطاف پذیر خاموش کردن برق باعث می شود در صورت تمایل صرفه جویی قابل توجهی در مصرف برق داشته باشید. برای بهبود عملکرد SFDR با سیگنال‌های ورودی آنالوگ کم توان، یک عملکرد انشعاب اختیاری در دسترس است. ترتیب پین های این ماژول به صورت شکل 1-1 می باشد که پین های 30 ، 31 و 32 مربوط به ارتباط SPI است.



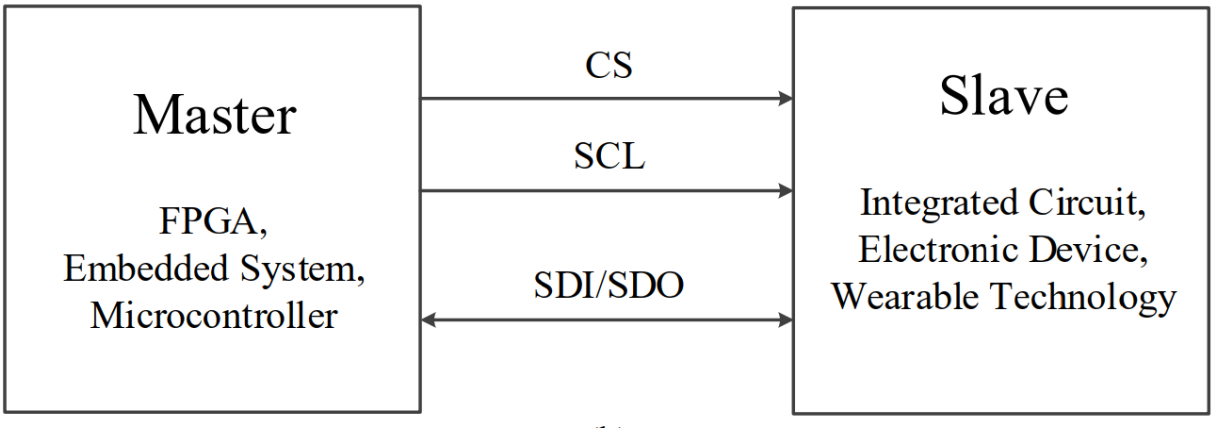
شکل 1-1 ترتیب پین ها در AD9255

2-1 ارتباط SPI

رابط جانبی سریال (SPI) یکی از پروتکل های ارتباطی بین دستگاه اصلی و slave است. دستگاه اصلی را می توان به عنوان یک سیستم تعبیه شده، آرایه دروازه قابل برنامه ریزی میدانی (FPGA) یا میکروکنترلر و دستگاه slave را می توان به عنوان مدار مجتمع، دستگاه الکترونیکی یا فناوری پوشیدنی در نظر گرفت. شکل 2-1 بلوک دیاگرام پروتکل 4 سیم SPI را نشان می دهد. این شامل یک دستگاه اصلی و یک دستگاه برده است. رابط SPI از دو سیگنال کنترل و دو سیگنال داده تشکیل شده است که به ترتیب عبارتند از انتخاب چیپ (CS)، ساعت سریال (SCL)، ورودی داده سریال (SDI) و خروجی داده سریال (SDO). به طور کلی جهت دو سیگنال کنترلی و ورودی داده سریال (SDI) توسط دستگاه اصلی ارسال و توسط دستگاه slave دریافت می شود که ما آن را "Master out Slave in (MOSI)" می نامیم. برعکس، جهت خروجی داده های سریال (SDO) توسط دستگاه Slave ارسال می شود و توسط دستگاه اصلی دریافت می شود، ما آن را "master in slave out (MISO)" می نامیم. اگرچه پروتکل ارتباطی 4 سیمه خطوط انتقال متعدد را چه در فرآیند نوشتن داده یا خواندن داده ها ساده می کند، هزینه سیلیکون و مصرف انرژی بار اصلی در فناوری VLSI است، مانند اندازه بسته IC و مقدار پد بنابراین، برای به حداقل رساندن سطح سیلیکون و افزایش عملکرد سیستم SPI برای برنامه های کاربردی VLSI، شکل 3-1 بلوک دیاگرام پروتکل 3 سیم SPI را نشان می دهد. اصل پروتکل 3 سیم SPI مشابه با نوع 4 سیم است. در مقایسه با پروتکل های SPI 4 سیم سنتی، سیگنال داده به صورت پورت اشتراکی طراحی شده است. مزیت ورودی داده سریال ادغام شده 3 سیم (SDI) و خروجی داده سریال (SDO) در یک پورت که دو جهته است. این قابلیت را دارد که عملکردهای نوشتن داده و خواندن داده ها را در یک پورت انجام دهد. در مقایسه با شکل 2-1، پروتکل 3 سیم SPI به کارایی مقرون به صرفه و بالاتر برای برنامه های VLSI دست می یابد.



شکل 2-1 ارتباط سریال 4 سیم



شکل 3-1 ارتباط سریال 3 سیم

1-3 پین های پورت SPI

1-3-1 کلاک سریال (SCLK)

پین SCLK ساعت شیفت سریال در پین است. این پین با یک ماشه اشمیت برای به حداقل رساندن حساسیت به نویز در خط ساعت اجرا می شود و توسط یک مقاومت اسمی 50 کیلو اهم به زمین کشیده می شود. این پین ممکن است در بالا یا پایین متوقف شود . SCLK برای همگام سازی خواندن و نوشتن رابط سریال استفاده می شود. داده های ورودی در لبه افزایشی این ساعت و انتقال داده های خروجی در لبه سقوط ثبت می شوند. زمان نگهداری معمولی 2 ns (tDH) است و حداقل زمان راه اندازی 2 ns (tDS) بین SCLK و SDIO لازم است. برای بهینه‌سازی

زمان‌بندی داخلی و خارجی، اتوبوس می‌تواند وضعیت خط SDIO را در نیم چرخه SCLK بچرخاند. این بدان معناست که پس از ارسال اطلاعات آدرس به مبدل درخواست کننده خواندن، خط SDIO از یک ورودی به یک خروجی در نیمی از یک چرخه ساعت منتقل می شود. این تضمین می کند که تا زمانی که لبه سقوط چرخه ساعت بعدی رخ می دهد، داده ها را می توان به طور ایمن در این خط سریال قرار داد تا کنترل کننده آن را بخواند.

1-3-2 پین داده سریال ورودی و خروجی (SDIO)

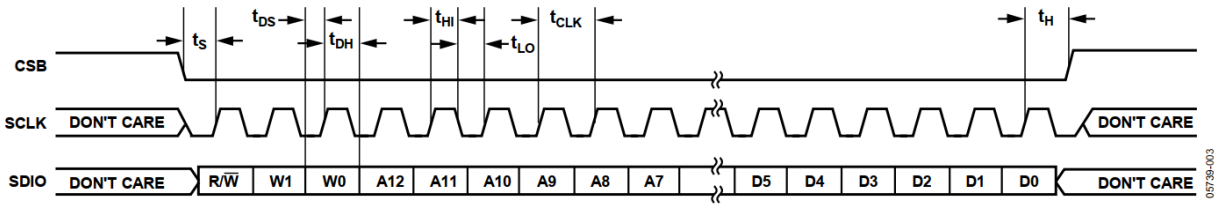
پین SDIO یک پایه دو منظوره است. نقش معمولی برای این پین به عنوان ورودی یا خروجی است، بسته به دستورالعمل ارسال شده (خواندن یا نوشتن) و موقعیت نسبی (دستورالعمل یا فاز داده) در چارچوب زمانی. در مرحله اول نوشتن یا خواندن، این پین به عنوان ورودی ای عمل می کند که اطلاعات را به ماشین حالت داخلی منتقل می کند. اگر دستور به عنوان یک دستور خواندن تعیین شود، ماشین حالت این پین (SDIO) را به یک خروجی تغییر می‌دهد، که سپس داده‌ها را به کنترل‌کننده ارسال می‌کند.

1-3-3 انتخاب چیپ (CS)

CSB یک کنترل active low است که چرخه های خواندن و نوشتن را فعال می کند. چندین حالت وجود دارد که در آن CSB می تواند کار کند. برای شرایطی که کنترلر دارای خروجی انتخاب چیپ یا ابزار دیگری برای انتخاب چندین دستگاه است، این پین را می توان به خط CSB متصل کرد. هنگامی که این خط low است، دستگاه انتخاب می شود و اطلاعات مربوط به خطوط SCLK و SDIO پردازش می شود. اگر این پین high باشد، دستگاه هر گونه اطلاعات روی خطوط SCLK و SDIO را نادیده می گیرد.

1-4 زمان بندی ارتباط SPI

برای برقراری ارتباط SPI باید حداقل زمان بندی برای بخش های مختلف رعایت شود که این زمان ها در شکل 1-4 نشان داده شده و در شکل 1-5 تعریف شده اند.



شکل 1-4 زمان های خاص در SPI

SPI TIMING REQUIREMENTS[1]			
t_{DS}	Setup time between the data and the rising edge of SCLK	2	ns
t_{DH}	Hold time between the data and the rising edge of SCLK	2	ns
t_{CLK}	Period of the SCLK	40	ns
t_S	Setup time between CSB and SCLK	2	ns
t_H	Hold time between CSB and SCLK	2	ns
t_{HIGH}	SCLK pulse width high	10	ns
t_{LOW}	SCLK pulse width low	10	ns
t_{EN_SDIO}	Time required for the SDIO pin to switch from an input to an output relative to the SCLK falling edge	10	ns
t_{DIS_SDIO}	Time required for the SDIO pin to switch from an output to an input relative to the SCLK rising edge	10	ns

شکل 1-5 مقدار زمان های ارتباط SPI ماژول AD9255

1-5 بخش های مختلف SPI

1-5-1 فرمت

لبه پایین رونده CSB، در ارتباط با لبه بالارونده SCLK، شروع کادربندی را تعیین می کند.

هنگامی که شروع قاب مشخص شد، زمان بندی ساده است. مرحله اول انتقال، مرحله دستورالعمل است که شامل 16 بیت است و سپس داده هایی که می توانند با طول های متغیر در مضرب های 8 بیتی باشند. اگر دستگاه با CSB پایین پیکربندی شده باشد، کادربندی با اولین لبه بالارونده SCLK آغاز می شود.

1-5-2 فاز دستور (instruction phase)

مرحله دستورالعمل 16 بیت اول ارسال شده است. همانطور که در شکل 6-1 نشان داده شده است، مرحله دستورالعمل به چند قسمت تقسیم می شود.

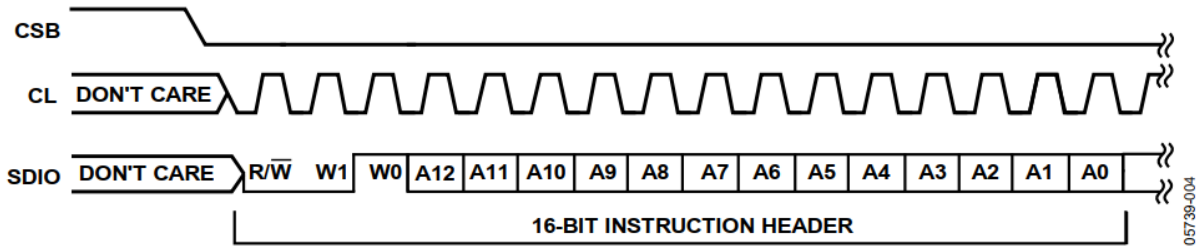
بیت خواندن و نوشتن

اولین بیت در جریان، بیت نشانگر خواندن/نوشتن است . هنگامی که این بیت بالا است، خواندن درخواست می شود. در پایان مرحله دستورالعمل (16 بیت اول)، ماشین حالت داخلی از اطلاعات ارائه شده برای رمزگشایی آدرس داخلی برای خواندن استفاده می کند. جهت خط SDIO از ورودی به خروجی تغییر می کند و تعداد مناسب کلمات تعریف شده با طول کلمه از دستگاه خارج می شود. هنگامی که اولین بیت در جریان داده کم است، یک فاز نوشتن وارد می شود. در پایان مرحله دستورالعمل، ماشین حالت داخلی از اطلاعات ارائه شده برای رمزگشایی آدرس داخلی برای نوشتن استفاده می کند. تمام داده ها پس از دستورالعمل در پین SDIO جابجا می شوند و به آدرس های مورد نظر ارسال می شوند. در حالت خواندن یا نوشتن، این روند تا رسیدن به طول کلمه یا تا زمانی که خط CSB یک شود ادامه می یابد.

طول کلمه

W1 و W0 تعداد بایت های داده را برای انتقال یا خواندن یا نوشتن نشان می دهند. مقدار نشان داده شده توسط W1:W0 + 1 تعداد بایت هایی است که باید منتقل شوند. اگر تعداد بایت‌های انتقال سه یا کمتر باشد (00، 01 یا 10)، CSB می‌تواند بعد از ارسال 8 بیت یک شده و دوباره برای ارسال بایت بعدی صفر شود. توقف در یک مرز غیر بایتی چرخه ارتباطات را خاتمه می دهد. اگر این بیت ها 11 باشند، داده ها را می توان تا انتقال CSB انتقال داد. CSB مجاز نیست در طول فرآیند پخش جریانی یک شود. هنگامی که پخش جریانی شروع شد (تعریف شده به عنوان فراتر از سومین بایت داده)، CSB اجازه ندارد تا زمانی که عملیات کامل شود، بالا برود. هنگامی که CSB بالا می رود، جریان پایان می یابد، و دفعه بعد که CSB پایین می آید، یک چرخه دستورالعمل جدید آغاز می شود. اگر CSB در یک مرز غیر 8 بیتی بالا برود، چرخه ارتباطات

پایان می یابد و هر بایت ناقصی از بین می رود. با این حال، بایت های داده تکمیل شده به درستی مدیریت می شوند.



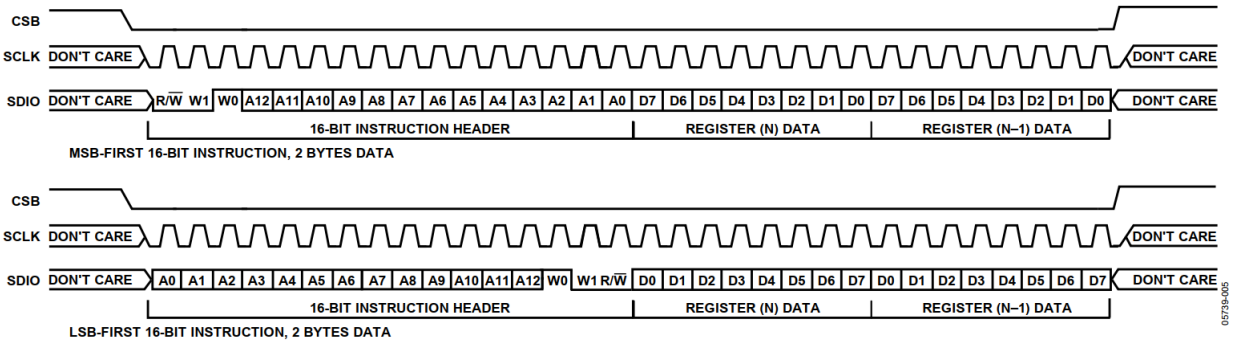
شکل 6-1 فرمت دستورالعمل در SPI

[W1:W0] Setting	Action	CSB Stalling
00	1 byte of data can be transferred.	Optional
01	2 bytes of data can be transferred.	Optional
10	3 bytes of data can be transferred.	Optional
11	4 or more bytes of data can be transferred. CSB must be held low for entire sequence; otherwise, the cycle is terminated, and an instruction cycle is anticipated when CSB returns low.	No

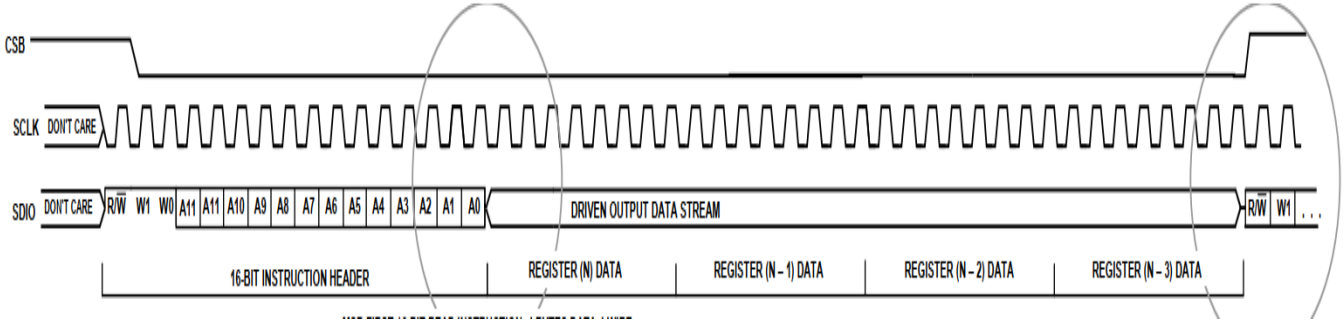
شکل 7-1 تنظیمات طول کلمه

ترتیب بیت

داده ها را می توان در حالت اول MSB یا حالت اول LSB ارسال کرد .هنگام روشن شدن، حالت MSB-first پیش‌فرض است. این را می توان با برنامه ریزی ثبات پیکربندی تغییر داد. در حالت MSB-first، مبادله سریال با بیت بالاترین مرتبه شروع می شود و با بیت کم اهمیت (LSB) به پایان می رسد. در حالت LSB-first، ترتیب برعکس می شود. این دستورالعمل 16 بیتی است که از 2 بایت تشکیل شده است که قبلاً توضیح داده شد. در حالت MSB-first، ترتیب بیت از بیت‌های بالاترین مرتبه به بیت‌های پایین‌ترین مرتبه است. همانطور که در شکل 8-1 نشان داده شده است، در حالت LSB-first، کل 16 بیت معکوس می شوند .



شکل 8-1 نحوه ارسال به صورت MSB-first و LSB-first



شکل 9-1 فرمت کلی از خواندن دیتا 4 بایتی

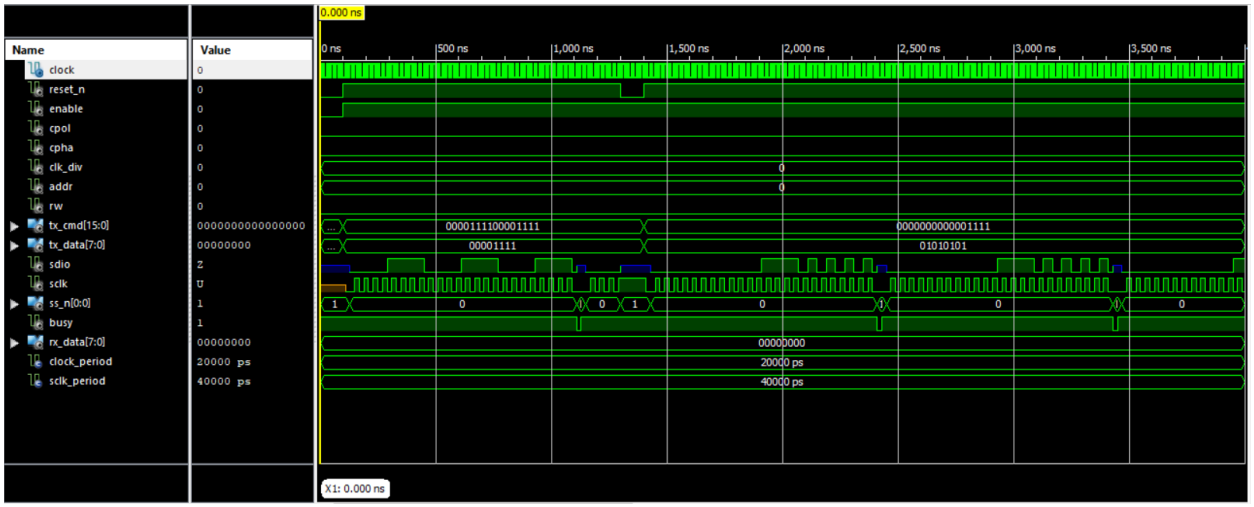
فصل دوم

پیاده سازی و شبیه سازی

2-1 پیاده سازی کد SPI و شبیه سازی آن در ISE

کد مربوط به ارتباط SPI در بخش الف پیوست آمده است. این کد همه 4 مد SPI را شامل می شود.در قسمت generic می توان طول داده ارسالی و طول دستورالعمل را مشخص کرد. تست بنچ حاصل برای کد مورد نظر در قسمت ب پیوست نوشته شده است. حالا به بررسی نتایج حاصل از کد می پردازیم.

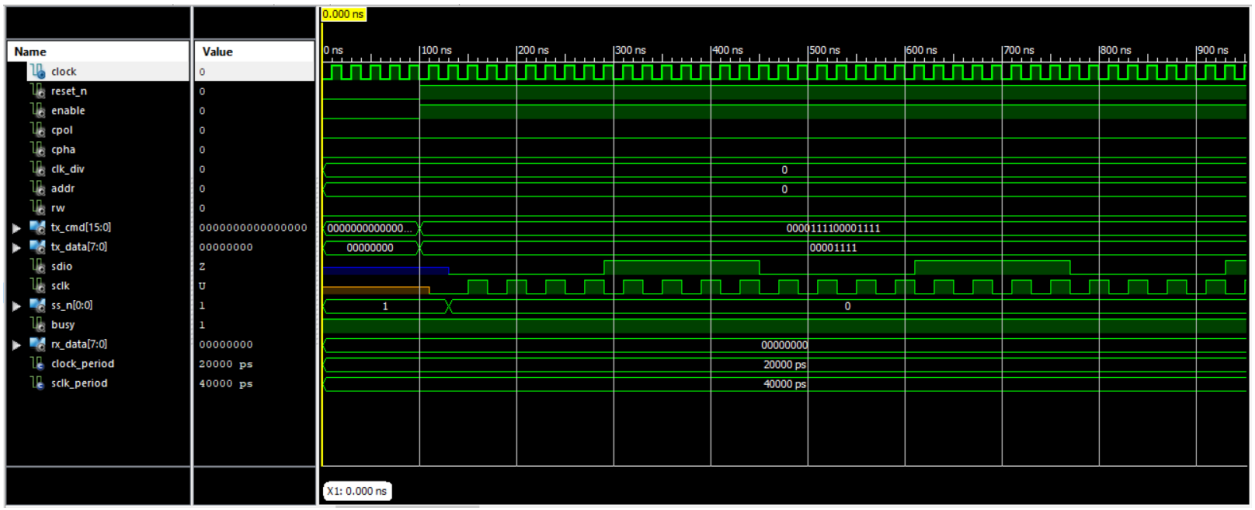
در ابتدا یک دستور به صورت "0000111100001111" می فرستیم و بعد دیتا یک بایتی "00001111" را می فرستیم. سپس دکمه ریست را فعال می کنیم . (ریست مدار active low) می باشد. بعد از گذشت مدت زمانی مشخص دستور "0000000000001111" و دیتا "01010101" را می فرستیم. نتایج حاصل از کد به صورت زیر می باشد.



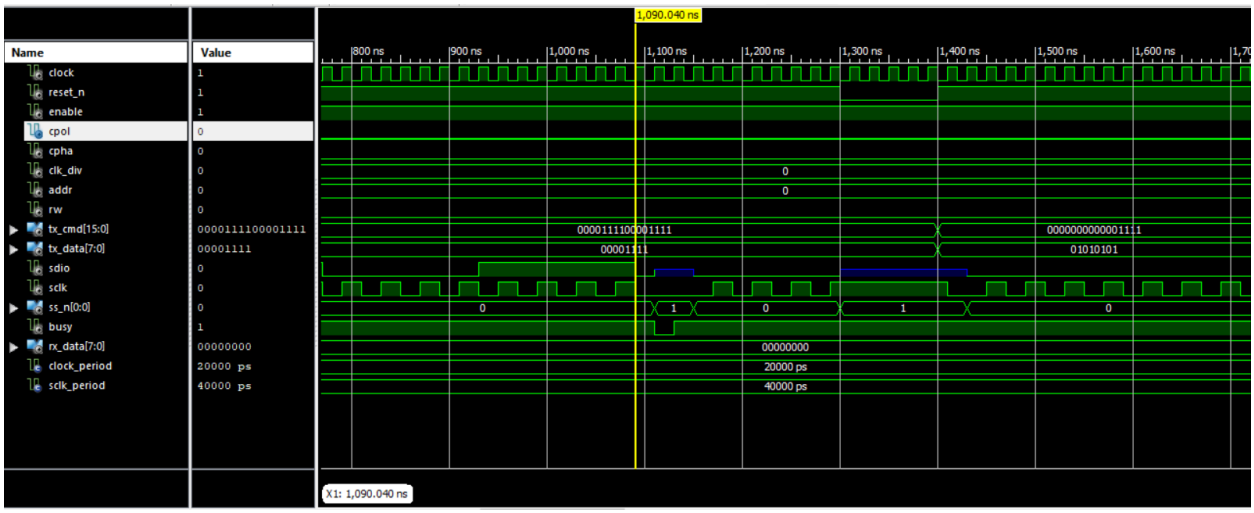
شکل 2-1 نتیجه حاصل از شبیه سازی

برای تحلیل بیشتر بخش های مختلف کد را به تفکیک نشان می دهیم .

در شکل 2-2 با یک کردن ریست و enable سیگنال SCLK شروع می به نوسان می کند و ss (همان CB در SPI) صفر می شود. همان طور که از sdio مشخص است دستور به درستی ارسال می شود.



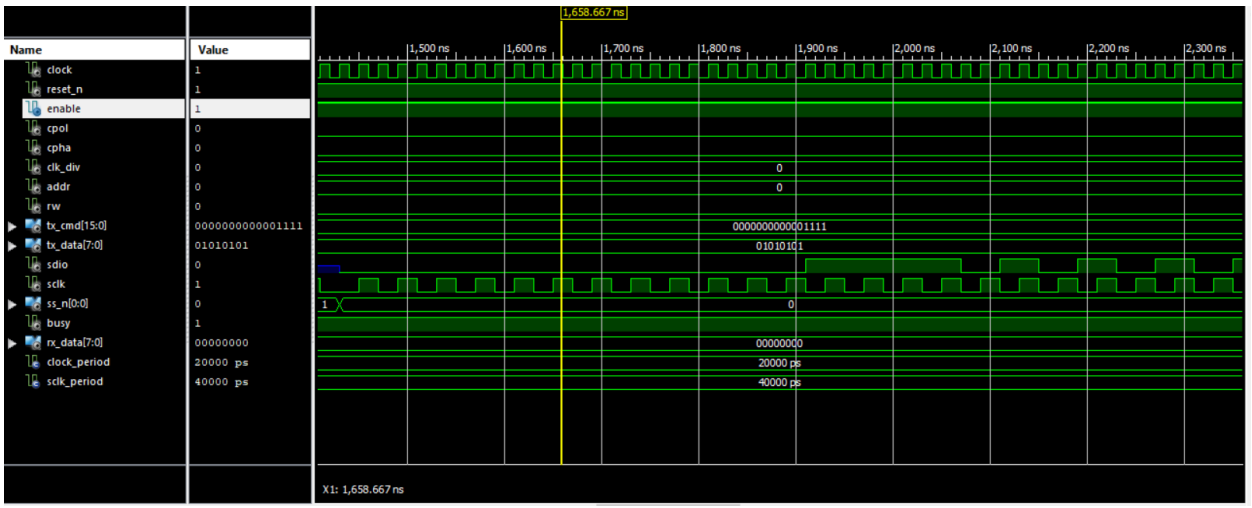
شکل 2-2 تحلیل قسمت ارسال دستور "0000111100001111"



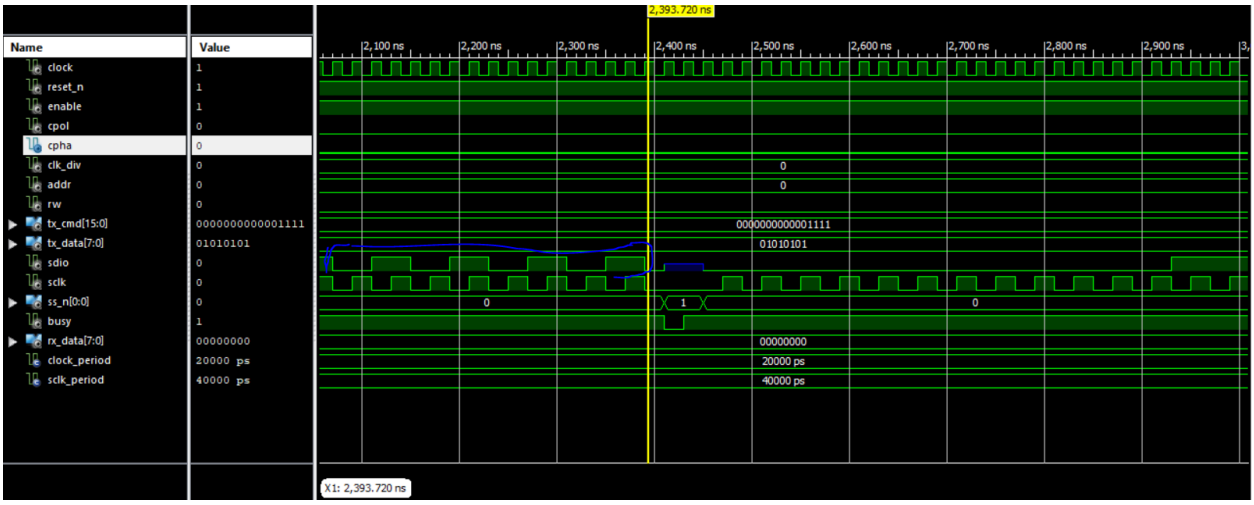
شکل 2-3 ارسال دیتا "00001111"

در شکل 2-3 بعد از ارسال دستورالعمل داده یک بایتی "00001111" به درستی ارسال می شود که از روی sdio کاملا مشخص است سپس در ثانیه 1300ns دکمه ریست صفر (فعال) شده و sdio به حالت 'Z' در می آید.

در شکل 2-4 بعد از یک کردن ریست این بار دستور "0000000000001111"را ارسال کرده و نتیجه را در sdio مشاهده می کنیم سپس در ادامه این دستور دیتا "01010101" را در شکل 2-5 می بینیم که به درستی بر روی sdio جهت ارسال قرار می گیرد.



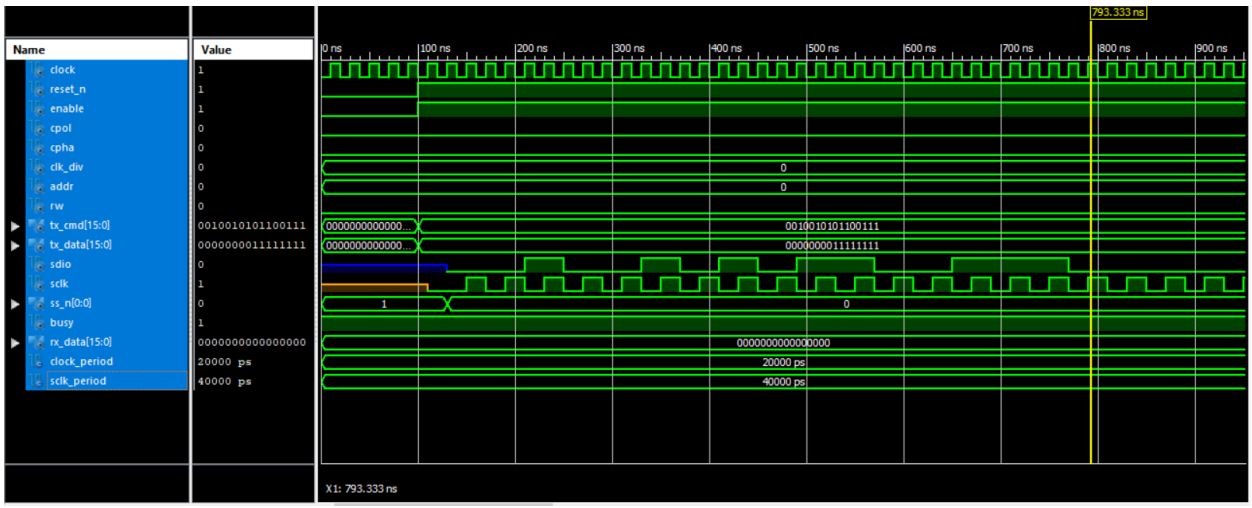
شکل 2-4 ارسال دستور "0000000000001111"



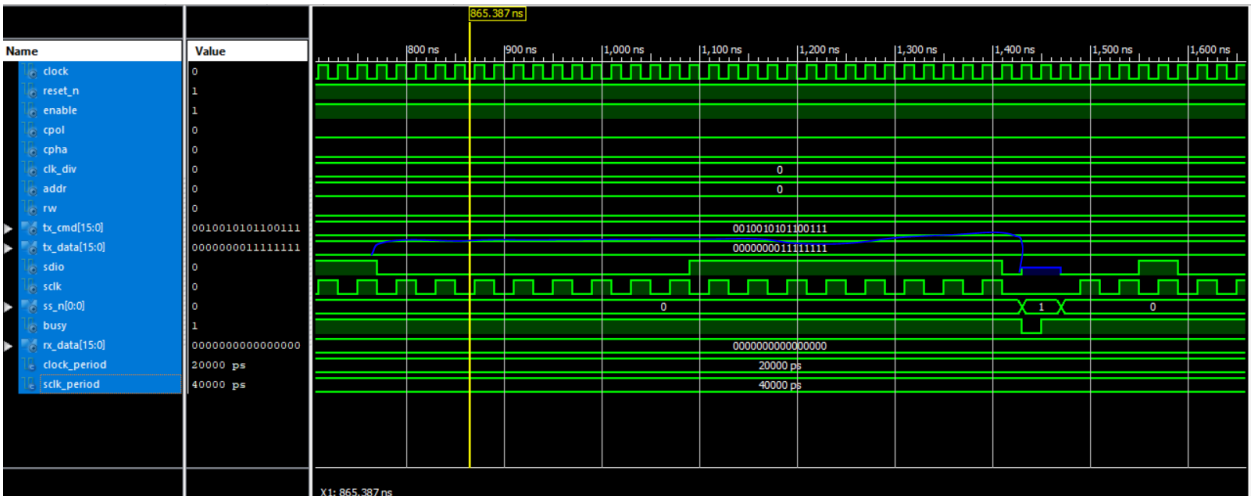
شکل 2-5 ارسال دیتا "01010101"

از اشکال 2-1 تا 2-5 مشخص است که ارتباط SPI به درستی کار می کند. حال با تغییر طول دیتا در generic کد را برای دیتا با طول های متفاوت تست می کنیم .

برای دیتا 16 بیتی نتایج به صورت دو شکل 2-6 و 2-7 خواهند بود.

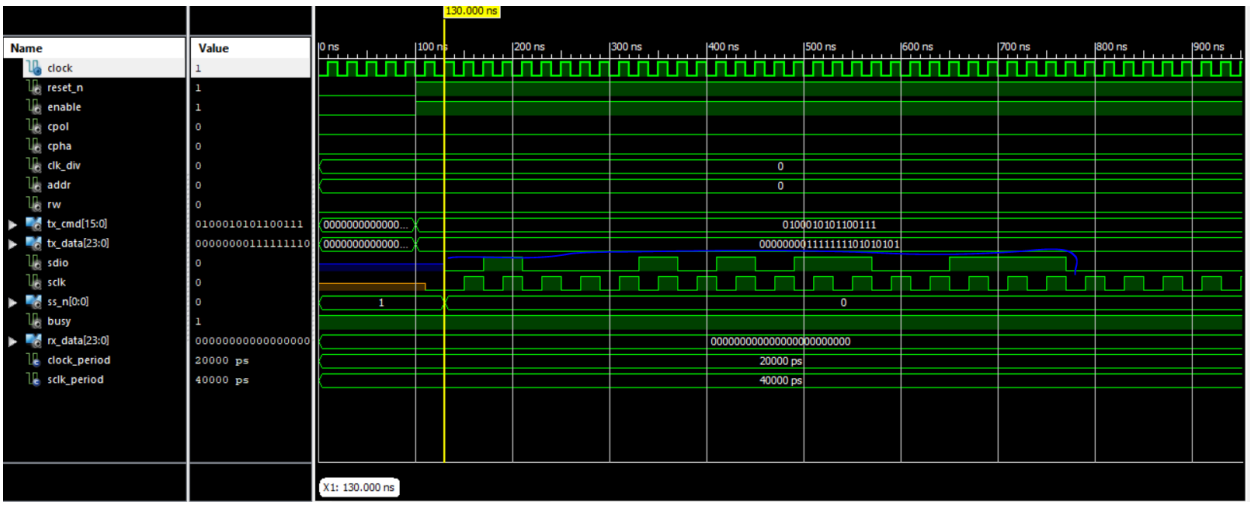


شکل 2-6 ارسال دستور "0010010101100111"

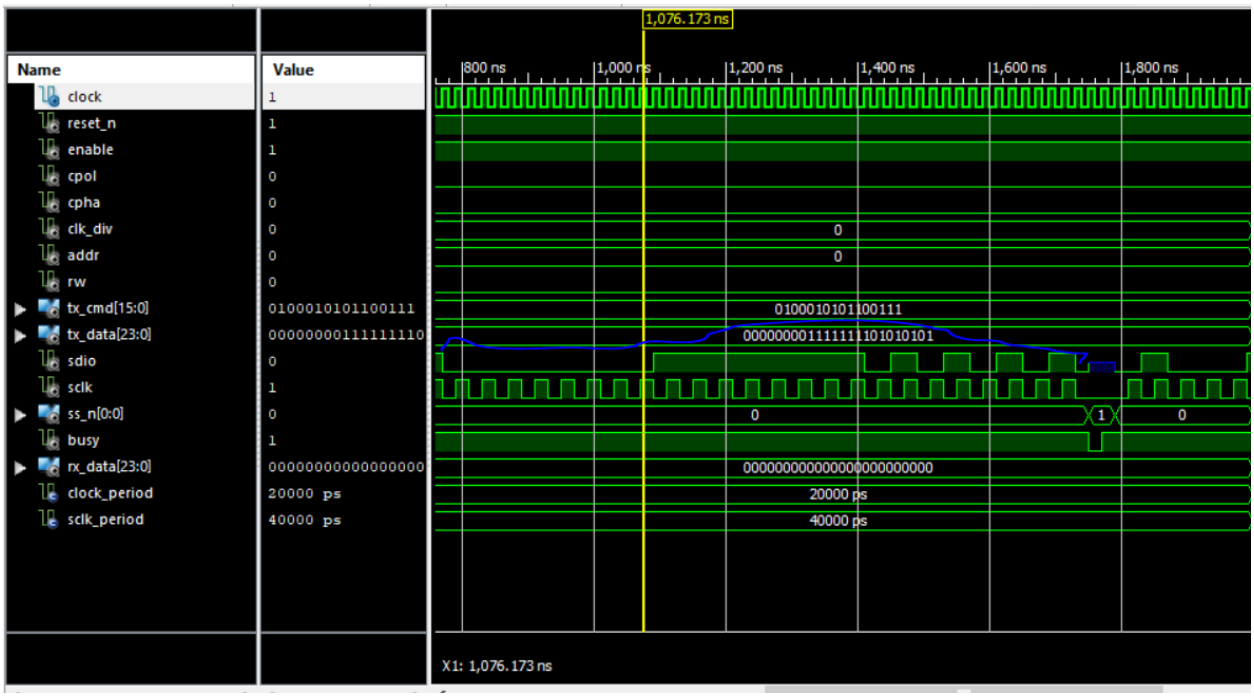


شکل 2-7 ارسال دیتا "0000000011111111"

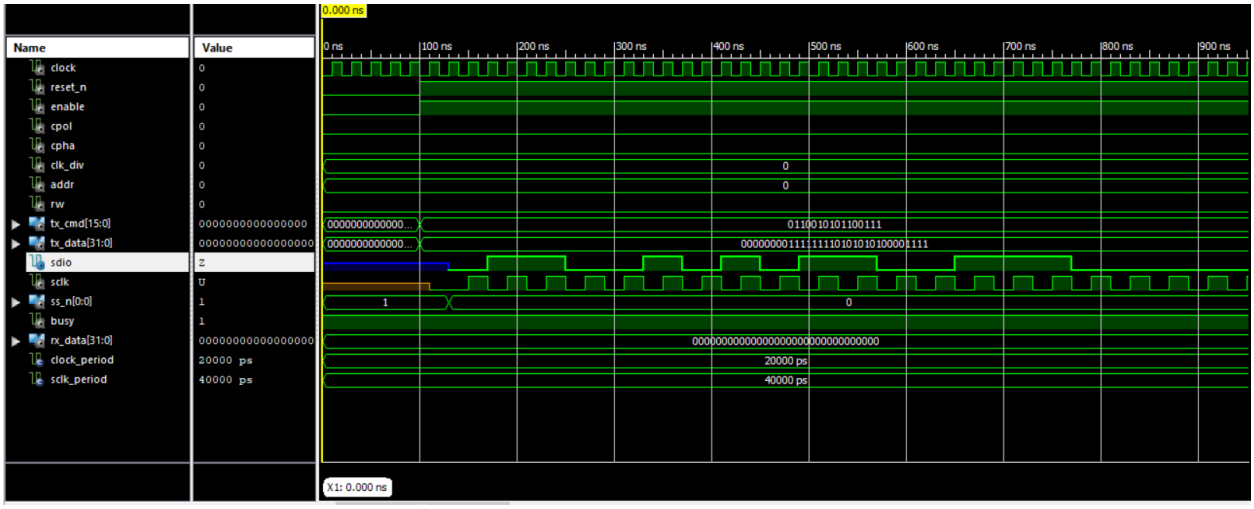
در شکل های 2-8 تا 2-11 نیز دیتا های به طول 24 و 32 ارسال می شوند.



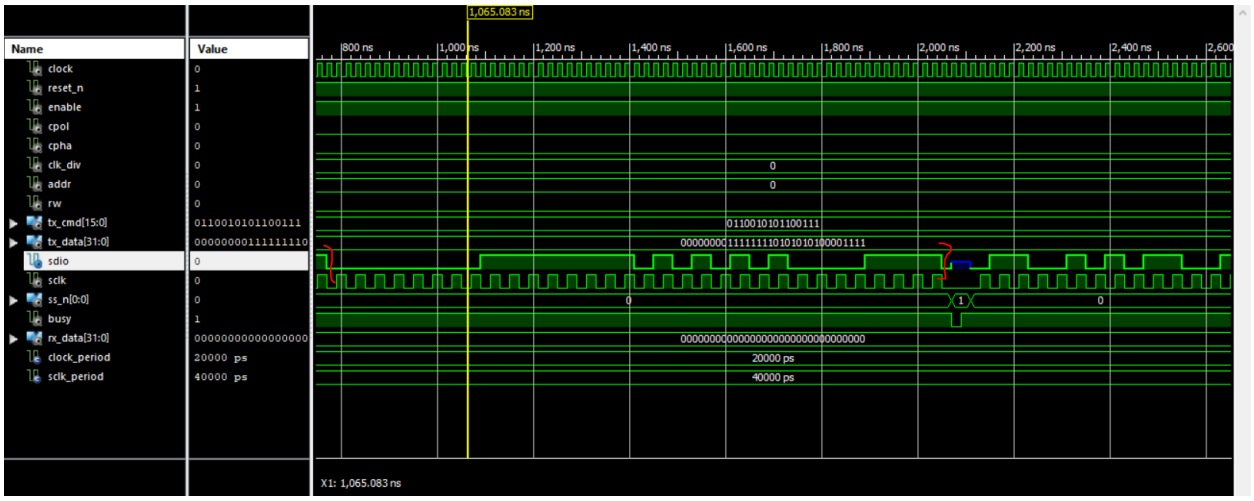
شکل 2-8 ارسال دستور "0100010101100111"



شکل 2-9 ارسال دیتا "00000000111111110101010"



شکل 2-10 ارسال دستور "0110010101100111"



شکل 2-11 ارسال دیتا "00000000111111110101010100001111"

فصل سوم

پیوست ها

قسمت الف

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

ENTITY spi_3_wire_master IS
 GENERIC(
  slaves   : INTEGER := 1;  --number of spi slaves
  cmd_width : INTEGER := 16;  --command bus width
  d_width   : INTEGER := 8); --data bus width
 PORT(
  clock   : IN    STD_LOGIC;                     --system clock
  reset_n : IN    STD_LOGIC;                     --asynchronous reset
  enable  : IN    STD_LOGIC;                     --initiate transaction
  cpol    : IN    STD_LOGIC;                     --spi clock polarity
  cpha    : IN    STD_LOGIC;                     --spi clock phase
  clk_div : IN    INTEGER;                       --system clock cycles per 1/2 period of sclk
  addr    : IN    INTEGER;                       --address of slave
  rw      : IN    STD_LOGIC;                     --'0' for read, '1' for write
  tx_cmd  : IN    STD_LOGIC_VECTOR(cmd_width-1 DOWNTO 0); --command to transmit
  tx_data : IN    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);   --data to transmit
  sclk    : BUFFER STD_LOGIC;                    --spi clock
  ss_n    : BUFFER STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);    --slave select
  sdio    : INOUT  STD_LOGIC;                    --serial data input output
```

```
busy    : OUT    STD_LOGIC;                      --busy / data ready signal
   rx_data : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0));  --data received
END spi_3_wire_master;

ARCHITECTURE logic OF spi_3_wire_master IS
  TYPE   machine IS(ready, execute);                    --state machine data type
  SIGNAL state      : machine;                          --current state
  SIGNAL slave      : INTEGER;                          --slave selected for current transaction
  SIGNAL clk_ratio   : INTEGER;                          --current clk_div
  SIGNAL count      : INTEGER;                          --counter to trigger sclk from system clock
  SIGNAL clk_toggles : INTEGER RANGE 0 TO (cmd_width+d_width)*2 + 2; --count spi clock toggles
  SIGNAL assert_data : STD_LOGIC;                       --'1' is tx sclk toggle, '0' is rx sclk toggle
  SIGNAL rw_buffer   : STD_LOGIC;                       --read/write buffer
  SIGNAL cmd_buffer  : STD_LOGIC_VECTOR(cmd_width-1 DOWNTO 0);      --command buffer
  SIGNAL d_buffer    : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);        --data buffer
  SIGNAL last_bit_rx : INTEGER RANGE 0 TO (cmd_width+d_width)*2;     --last rx data bit location
BEGIN
  PROCESS(clock, reset_n)
  BEGIN

    IF(reset_n = '0') THEN       --reset system
      busy <= '1';               --set busy signal
      ss_n <= (OTHERS => '1');    --deassert all slave select lines
      sdio <= 'Z';               --set master data io to high impedance
      rx_data <= (OTHERS => '0'); --clear receive data port
      state <= ready;            --go to ready state when reset is exited

    ELSIF(clock'EVENT AND clock = '1') THEN
      CASE state IS              --state machine
```

```
WHEN ready =>
        busy <= '0';             --clock out not busy signal
        ss_n <= (OTHERS => '1');  --set all slave select outputs high
        sdio <= 'Z';             --set master data io to high impedance

        --user input to initiate transaction
        IF(enable = '1') THEN
         busy <= '1';            --set busy signal
         IF(addr < slaves) THEN   --check for valid slave address
          slave <= addr;         --clock in current slave selection if valid
         ELSE
          slave <= 0;            --set to first slave if not valid
         END IF;
         IF(clk_div = 0) THEN     --check for valid spi speed
          clk_ratio <= 1;        --set to maximum speed if zero
          count <= 1;            --initiate system-to-spi clock counter
         ELSE
          clk_ratio <= clk_div;  --set to input selection if valid
          count <= clk_div;      --initiate system-to-spi clock counter
         END IF;
         sclk <= cpol;           --set spi clock polarity
         assert_data <= NOT cpha; --set spi clock phase
         rw_buffer <= tx_cmd(cmd_width-1);        --clock in read/write instruction
         cmd_buffer <= tx_cmd;    --clock in command for transmit into buffer
         d_buffer <= tx_data;     --clock in data for transmit into buffer
         clk_toggles <= 0;        --initiate clock toggle counter
         last_bit_rx <= (cmd_width+d_width)*2 + conv_integer(cpha) - 1; --set last rx data bit
         state <= execute;        --proceed to execute state
        ELSE
```

```
state <= ready;         --remain in ready state
        END IF;

      WHEN execute =>
        busy <= '1';       --set busy signal
        ss_n(slave) <= '0'; --set proper slave select output

        --system clock to sclk ratio is met
        IF(count = clk_ratio) THEN
          count <= 1;                  --reset system-to-spi clock counter
          assert_data <= NOT assert_data;  --switch transmit/receive indicator
          clk_toggles <= clk_toggles + 1;  --increment spi clock toggles counter

          --spi clock toggle needed
          IF(clk_toggles < (cmd_width+d_width)*2 + 1 AND ss_n(slave) = '0') THEN
            sclk <= NOT sclk;  --toggle spi clock
          END IF;

          --transmit spi clock toggle
          IF(assert_data = '1' AND clk_toggles < last_bit_rx - d_width*2) THEN  --command part of
transaction
            sdio <= cmd_buffer(cmd_width-1);                    --clock out command bit
            cmd_buffer <= cmd_buffer(cmd_width-2 DOWNTO 0) & '0';          --shift command transmit
buffer
          END IF;
          IF(assert_data = '1' AND rw_buffer = '0' AND clk_toggles > last_bit_rx - d_width*2) THEN  --write
command and data part of transaction
            sdio <= d_buffer(d_width-1);                              --clock out data bit
            d_buffer <= d_buffer(d_width-2 DOWNTO 0) & '0';                    --shift data transmit
buffer
```

```
END IF;

        IF(assert_data = '1' AND rw_buffer = '1' AND clk_toggles > last_bit_rx - d_width*2) THEN  --read command and data part of transaction

          sdio <= 'Z';                                                --set serial data line to high impedance

        END IF;


        --receive spi clock toggle

        IF(assert_data = '0' AND clk_toggles < last_bit_rx + 1) THEN

          IF(rw_buffer = '1' AND clk_toggles > last_bit_rx - d_width*2) THEN --read transaction and data part of transaction

            d_buffer <= d_buffer(d_width-2 DOWNTO 0) & sdio;              --shift in received bit

          END IF;

        END IF;


        --end of transaction

        IF(clk_toggles = (cmd_width+d_width)*2 + 1) THEN

          busy <= '0';            --clock out not busy signal

          ss_n <= (OTHERS => '1');  --set all slave selects high

          sdio <= 'Z';            --set master data io to high impedance

          IF(rw_buffer = '1') THEN  --if transaction was a read

            rx_data <= d_buffer;    --clock out received data to output port

          END IF;

            state <= ready;        --return to ready state

          ELSE                    --not end of transaction

            state <= execute;       --remain in execute state

        END IF;


      ELSE                --system clock to sclk ratio not met

        count <= count + 1;  --increment counter
```

```
state <= execute;    --remain in execute state
        END IF;

     END CASE;
   END IF;
 END PROCESS;
END logic;
```

کد بالا با نام AD9255SPI.vhd در فایل تحویلی قرار داده شده است.

قسمت ب

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

USE ieee.numeric_std.ALL;

ENTITY AD9255TB IS
END AD9255TB;

ARCHITECTURE behavior OF AD9255TB IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT spi_3_wire_master
   PORT(
      clock : IN  std_logic;
      reset_n : IN  std_logic;
      enable : IN  std_logic;
      cpol : IN  std_logic;
      cpha : IN  std_logic;
      clk_div : IN  integer;
      addr : IN  integer;
      rw : IN  std_logic;
      tx_cmd : IN  std_logic_vector(15 downto 0);
      tx_data : IN  std_logic_vector(7 downto 0);
      sclk : buffer  std_logic;
      ss_n : buffer  std_logic_vector(0 downto 0);
```

```
sdio : INOUT  std_logic;
        busy : OUT  std_logic;
        rx_data : OUT  std_logic_vector(7 downto 0)
       );
    END COMPONENT;


   --Inputs
   signal clock : std_logic := '0';
   signal reset_n : std_logic := '0';
   signal enable : std_logic := '0';
   signal cpol : std_logic := '0';
   signal cpha : std_logic := '0';
   signal clk_div : integer := 0;
   signal addr : integer := 0;
   signal rw : std_logic := '0';
   signal tx_cmd : std_logic_vector(15 downto 0) := (others => '0');
   signal tx_data : std_logic_vector(7 downto 0) := (others => '0');

            --BiDirs
   signal sdio : std_logic;

            --Outputs
   signal sclk : std_logic;
   signal ss_n : std_logic_vector(0 downto 0);
   signal busy : std_logic;
   signal rx_data : std_logic_vector(7 downto 0);
```

```
-- Clock period definitions
  constant clock_period : time := 20 ns;

  constant sclk_period : time := 40 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
  uut: spi_3_wire_master PORT MAP (
       clock => clock,
       reset_n => reset_n,
       enable => enable,
       cpol => cpol,
       cpha => cpha,
       clk_div => clk_div,
       addr => addr,
       rw => rw,
       tx_cmd => tx_cmd,
       tx_data => tx_data,
       sclk => sclk,
       ss_n => ss_n,
       sdio => sdio,
       busy => busy,
       rx_data => rx_data
     );

  -- Clock process definitions
  clock_process :process
  begin
```

```
                clock <= '0';
                wait for clock_period/2;
                clock <= '1';
                wait for clock_period/2;
  end process;

  -- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100 ns.
    wait for 100 ns;

    reset_n <='1' ;
                enable <='1' ;
                cpol<='0' ;
                cpha<='0' ;
                tx_cmd <="0000111100001111" ;
                tx_data <= "00001111" ;
                wait for 1200 ns ;
                reset_n<= '0' ;
                wait for 100 ns ;
reset_n <='1' ;
                enable <='1' ;
                cpol<='0' ;
                cpha<='0' ;
                tx_cmd <="0000000000001111" ;
                tx_data <= "01010101" ;

    -- insert stimulus here
```

```
wait;
  end process;

END;
```

کد بالا با نام AD9255TB.vhd در فایل ارسالی قرار دارد.