بنام خدا

گزارش پروژه VHDL

میلاد کلوندی

استاد درس: دکتر میرزا کوچکی

معرفی پروتکل SPI:

--پروتکل سریال بسیار پرکاربرد که با ۴ سیم دیتا را به صورت سنکرون را منتقل میکند

معنی سنکرون بودن یعنی به همراه دیتا کلاک هم برای گیرنده ارسال میشود و گیرنده دیتا را به نحو مناسبی نمونه برداری می کند معمولاً در لبه بالا رونده کلاک محل نمونه برداری بیت های دیتا است

--برای انتقال دیتا در فواصل نزدیک و سرعت نسبتاً زیاد

--انتقال دیتا به صورت فول دوبلکس انجام میشود

پکتSPI :

--برعکس RS232 پکت های SPI فقط شامل بیت های دیتا هستند

--همچنین طول پکت ها می توانند متفاوت باشند ۸ ۱۶ ۲۴ و ۳۲

--سیگنال cs مشخص کننده ابتدا و گاهی انتهای پکت است در پکت های SPI ما فقط دیتا را ارسال میکنیم چیزی در این پکت ها وجود ندارد که مشخص کند ابتدا و انتهای پکت کجاست

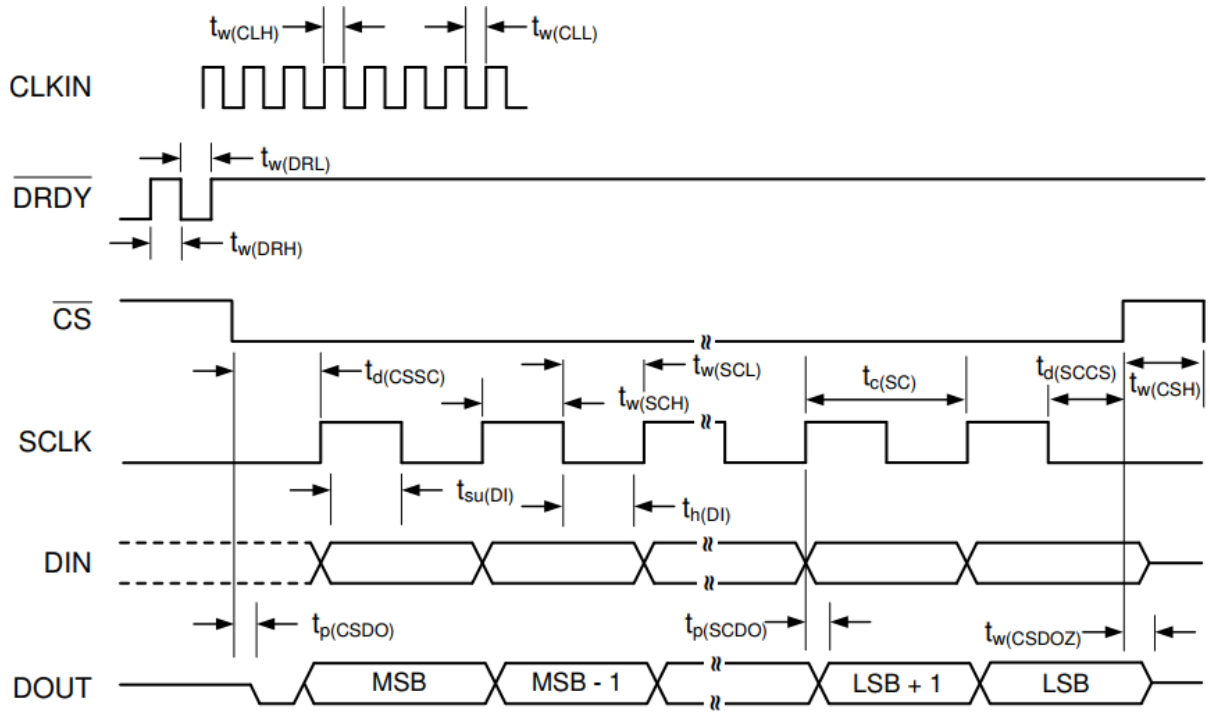
تایمینگ پکتSPI :

--سرعت بیشتر نسبت بهRS232

--برخلاف RS232 که دارای یکسری سرعت های استاندارد است و گیرنده و فرستنده باید روی این سرعت به توافق برسند و بر مبنای آن سرعت ارسال انجام می‌شود ولی هر پریفرالی که از پروتکل SPI استفاده می کند یک ماکسیمم سرعت کلاک برای SPI مشخص میکند

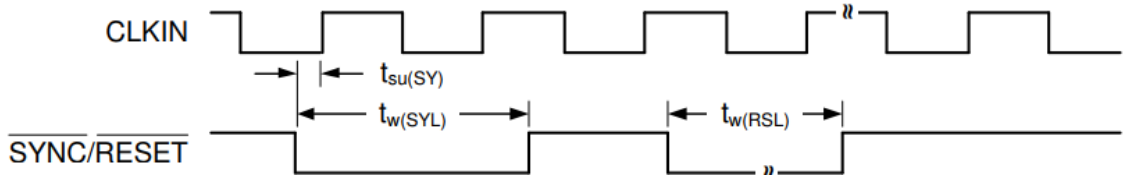
--طول پاکت معمولاً ضریبی از ۸ ارسال ممکن است به صورت MSB_FIRST یا LSB_FIRST فرصت انجام شود

Tc(sc)	Min 64
Td(cssc)	Min 16
Td(sccs)	Min 10
Tw(csh)	Min 15
Tsu(di)	Min 5
Th(di)	Min 10
Tw(drh)	Min 4
Tw(drl)	Min 4
Tp(scdo)	Min 5



'OL = 0 and CPHA = 1. $\overline{CS}$ transitions must take place when SCLK is low.

Figure 6-1. SPI Timing Diagram



ما در اینجا فرکانس سیستم را ۱۰۰ مگاهرتز در نظر گرفتیم فرکانس خروجی SCK را برابر مقدار ۱۰ مگاهرتز در نظر گرفتیم یک پریود ده نانو ثانیه به ما میدهد بنابراین شرط پریود SCK برقرار میشود

در اینجا ما دو بلوک به نامهای SPI و ADS داریم بلوک SPI برای ارسال داده های سریال استفاده می شودو بلوکه ADS برای ارسال کامند رجیستر های این بلوک استفاده می شود

ابتدا بلوک مربوط به SPI را توضیح می دهیم:

```
-----clock generator
if (SCK_Clock_Divider < to_unsigned(5,4)) then
    SCK_Int                          <=  '0';
end if;

if (SCK_Clock_Divider = to_unsigned(9,4)) then
    SCK_Clock_Divider                <=  (others=>'0');
end if;
```

در اینجا ما دو تا شرط داریم که کلاک spi نوشتیم علیرغم تایمینگی که در دیتاشیت داریم ماکسیمم فرکانس کلاک میتونست 15.625 مگاهرتز باشد اما من در پیاده سازی ۱۰ مگاهرتز در نظر گرفتم که پریود کلاک 100 نانو ثانیه ای به ما میدهد با انتخاب کلاک مدار ۱۰۰ مگاهرتز و پریود ۱۰ نانو ثانیه است اگر یک کانتر درست کنیم هر واحدی که به این کانتر اضافه میشه در واقع ده نانو ثانیه است پس برای اینکه ما صد نانو ثانیه رو ایجاد کنیم که معادل ده مگاهرتز است نیاز داریم که کانتر ما ده تا شمارش انجام بده کانتری که در اینجا تعریف کردیم برای تولید کلاک اسمش SCK_Clock_Divider است در بدنه process می‌بینیم که این شمارنده بصورت free runingهمیشه در حال شمارش است در داخلی if شرطی گذاشتیم که اگر کانتر کوچکتر از ۵ بود یعنی ۰ ۱ ۲ ۳ و ۴ مقدار SCK_Int برابر صفر باشد در غیر اینصورت هیچ شرطی قرار ندادیم و شرط تمام شده اما در خط ۸۲ مقدار SCK_Int برابر '1' قرار دادیم که در صورت اجرا نشدن شدن شرط بالا مقدار SCK_Int ‘1’خواهد شد در شرط دوم نوشتیم که اگر کانتر ما برابر ۹ شد حالا کانتر ریست شود

```
---- START SENDING
if (Send_Int = '1' and Send_Prev = '0' and Busy_Int = '0') then

    Data_In_Buff                <=  Data_In_Int;
    Force_CS_Buff               <=  Force_CS_Int;
    SPI_Data_Bit_Width_Buff     <=  (others=>'0');
    SPI_Data_Out_Bit_Width      <=  to_unsigned(0,3);
    CS_Disable_Counter          <=  (others=>'0');
    SCK_Clock_Divider           <=  to_unsigned(6,4);
    SPI_Write_State             <=  '1';
    SPI_OPcode_Byte             <=  '0';
    Busy_Int                    <=  '1';
    Set_SCK_Disable             <=  '1';
    SPI_Transmission_End        <=  '0';
    SCK_Disable                 <=  '0';

end if;
```

در اینجا شرطی داریم که شروع ارسال است که با لبه بالا رونده Send کار میکند در داخل شرط اگر لبه بالارونده Send رخ دهد و سیگنال Busy_Int هم برابر '0' باشد فرایند ارسال شروع میشود در این فرآیند ابتدا یک سری سیگنال ها مقدار دهی اولیه می شوند مقدار Data و Force_Cs بافر می شوند که اگر در ادامه تغییر هم کرد مقدار آن لحظه را داشته باشیم چون ممکنه فرمان Send بیاید و ما شروع به کار با دیتا کنیم ولی در حین کار مقدار دیتا کسی که با ماژول استفاده می کند در بیرون عوض کنه این نباید باعث خراب شدن عملکرد ماژول ما بشه برای اینکه این اتفاق نیفته به محض دیدن لبه بالارونده send مقدار دیتای ورودی و مقداری که برای Force_Cs بوده رو در سیگنال میانی قرار میدیم و با این دو سیگنال میانی در پروژه کار میکنیم

ورودی دیگری بود به اسم Command_Type_Int که نشان میده طول پکت ما چند بیتی است (00 برای 8 - 01 برای 16 - 10 برای 23 - 11 برای 32 بیت)

```
type        SPI_Data_Bit_Width_Array is array (0 to 3) of unsigned(4 downto 0);
signal  SPI_Data_Bit_Width          :   SPI_Data_Bit_Width_Array        :=
            (   to_unsigned(7,5),
                to_unsigned(15,5),
                to_unsigned(23,5),
                to_unsigned(31,5)
            );
```

در اینجا آرایه تعریف کردیم به اسم SPI_Data_Bit_Width و یک تاییپی تعریف کردیم به صورت آرایه ای که چهار خونه داره که هر خونه یک unsined پنج بیتی است یک خط پایین تر یک سیگنال SPI_Data_Bit_Width کردیم که اولی ۷ دومی ۱۵ سوم ۲۴ و آخرین ۳۱ که همان طول پکت‌های ۸ بیتی ۱۶ بیتی و ... است

حالا با توجه به اینکه Command_Type_Int چه بوده و تبدیل به اینتیجر من از اون آرایه مقدار متناظر اون رو میریزم در یک سیگنال دیگه به اسم SPI_Data_Bit_Width_Buff (مثلاً اگر Command_Type_Int روکاربر برابر صفر قرار داده باشه پس خونه شماره صفر آرایه خونده میشه که مقدار ۷ در آن بوده پس طول دیتای ما ۸ بیتی خواهد بود یعنی سیگنال دیتای ما که ۳۲ بیتی بود خونه های7 تا 0 ما نیاز داریم و استفاده میکنیم)

سیگنال دیگری به اسم SPI_Data_Out_Bit_Width که کانتر است برای شمردن بیت های خوندن از اسلیو چون ما بیت هارو ۸ بیت ۸ بیت از اسلیو می خونیم و در داخل شرط send مقدار ثابت صفر به این کانتر به عنوان مقدار اولیه دادیم زیرا سیگنال ما LSB_First است

دو خط پایین تر یک کانتر دیگر که مقدار آن را صفر گذاشتیم و خود کانتری است که کلاک را می شمارد SCK_Clock_Divider و مقدار آن را شش قرار دادیم و سیگنال دیگری به نام SPI_Write_Stateرو برابر یک کردیم چون قراره به استیت نوشتن ببریم

سیگنال دیگری به نام SPI_OPCode_Byte را برابر صفر کردیم Busy_Int را برابر یک کردیم (دلیل آن مشخص است) سیگنال دیگه ای به نام Set_SCK_Disable برابر یک شده و سیگنال SPI_Transmission_End و SCK_Disable را برابر صفر گذاشتیم

```
if (CS_Disable_Counter < to_unsigned(2,3)) then

    CS_Disable_Counter              <=  CS_Disable_Counter + 1;
    CS_Int                          <=  '1';

end if;
```

در ابتدای کد یک سیگنال داریم که CS_Disable_Counter نام دارد (در دیتاشیت گفتیم که CS فعال پایین است و همیشه برابر صفر است ولی وقتی فرمان Send ارسال می شود یک مدت کوتاهی CS یک می شود که slave ریست بشود و ابتدای پکت معلوم شد دوباره CSبرابر صفر میشه و ارسال انجام میشه مقدار اولیه این کانتر CS_Disable_Counter) (هفت است چون مقدار هفت داره شرط برقرار نیست وقتی فرمان send ارسال میشه ما کانتر رو ریست میکنیم بعد شرط ما برقرار میشه به مدت ۲ کلاک و تا زمانی که شرط برقراره CS_Intبرابر یک است CS_Int رو ما همیشه برابر صفر قرار دادیم در زمان هایی که شرط برقراره اون رو برابر یک قرار میدهیم

به این ترتیب به محض این که فرمان Send اعمال شد ما اون CS رو مدتی Disable میکنیم و دوباره enaable میکنیم حالا چرا دو کلاک چون در دیتاشیت نوشته بود حداقل باید ۱۶ nsبشه دوکلاک میشه ns20

یک سیگنال دیگه ای به نام SCK_Disable برابر صفر قرار دادیم یعنی غیر فعال کردن کلاک SPIدر قسمت concurent ماکلاک SPI رو داریم که SCK_Int رو بهش ارجاع دادیم ولی اون رو با SCK_Disable اند کردیم وقتی SCK_Disable برابر صفر باشه فرقی نداره که SCK_Intچی باشه SCKبرابر صفر میشه در زمانی که ما داریم سیگنال CSرو غیر فعال می کنیم نیاز به کلاک زدن نیست بنابراین در این لحظات ما کلاک رو غیرفعال کردیم

بخش نوشتن:

```
    ---write
if (SCK_Clock_Divider = to_unsigned(0,4) and SPI_Write_State = '1') then

    MOSI_Int                        <= Data_In_Buff(to_integer(SPI_Data_Bit_Width_Buff));
    SPI_Data_Bit_Width_Buff         <=  SPI_Data_Bit_Width_Buff + 1;
    SCK_Disable                     <=  Set_SCK_Disable;

    if (SPI_Data_Bit_Width_Buff = to_unsigned(to_integer(SPI_Data_Bit_Width(to_integer(Command_Type_Int))),5)) then
        SPI_Transmission_End        <=  '1';
    end if;

    if (SPI_Transmission_End = '1') then

        SPI_Transmission_End        <=  '0';
        SCK_Disable                 <=  '0';
        Busy_Int                    <=  '0';
        Set_SCK_Disable             <=  '0';
        SPI_Write_State             <=  '0';

    end if;

end if;
```

اما بعد از اون میرسیم به شرطی که عکس آن در بالا قرار داده شده گفتیم که اگر (SCK_Clock_Dividerهمان کانتری که کلاکSPIرو تولید می کرد (برابر صفر باشه(زمانی برابر صرف میشه که مقدار اون از صفر تا ۴ باشه پس وقتی صفر هست یعنی شروع پریود این کلاک هستیم که صفره پس ما در ابتدای پریود هستیم) و SPI_Write_Stateبرابر یک باشد)که همین طور هم هست در زمان ارسال که شرط sendبرقراره مقدار اون رو یک کردیم(کانتر ما شمرد از ۶ تا ۹ و که شرط برقرار شد مقدار اون رو صفر قراردادیم شرط بالا برقرار می شود و درست در اولین پریود اولین کلاک هستیم میخواهیم اولین بیت دیتا رو خروجی قرار بدیم و اون رو روی MOSIقرار بدیم قبلا دیتا رو روی Data_In_Buffریخته بودیم که ۳۲ بیتی هست سیگنال شمارنده SPI_Data_Bit_Width_Buffکه شماره بیت را مشخص می‌کرد

با فرض اینکه مقدار این سیگنال صفر باشه پس از اول بیت صفرم رو روی خروجی می ریزیم چون قرار سیستم LSB_First رو پیاده سازی کنیم ضمن این کار یکی به اون اضافه میکنیم که وقتی شرط برقرار شد بیت بعدی روی خروجی قرار بگیره در عین حال وقتی وارد این شرط میشیم سیگنال SCK_Disable رو برابر میکنیم با Set_SCK_Disable که سیگنال ای است که در قسمت پایین کد برابر یک قرار دادیم پس اینجا SCK_Disable برابر یک میشه حالا که این سیگنال برابر یک شد تازه کلاک شروع به فعالیت می کند) چون در بالا این سیگنال رو با SCK_Intاند کردی به SCKدادیم (پس پس کل شرط بالا تمام کارهای مربوط به نوشتن است

در داخل شرط دونه دونه بیتا نوشته میشه خط بعد یکی به کانتر اضافه میشه خط بعدی اون هم کلا کرو فعال میکنه در داخل این شرط یک شرط دیگه داریم گفتی اگر SPI_Data_Bit_Width_Buff برابر طول دیتا شد (وقتی برابر طول دیتا میشه که سنگین ترین بیت رو ارسال کردید) باید یک پریود دیگه صبر کنیم که یک سیگنال دیگه به نام SPI_Transmision_Endرو که در ابتدای کار برابر صفر کرده بودیم اینجا اونو یک می کنی تا یک پریود دیگه بگذره و این شرط برقرار بشه وقتی این شرط برقرار شد شرط بعدی اون هم برقرار میشه چون SPI_Transmision_Endرو دفعه قبل برابر یک قرار دادیم وقتی این شرط برقرار بشه اولا اینکه همین سیگنال رو برابر صفر می کنیم چون ارسال تموم شده و میخواهیم SCK_Disableرو هم دوباره صفر می کنیم چون سال تموم شده و می خواهیم SCKمتوقف بشه و Busyرو برابر صفر می کنیم چون ارسال تموم شده و می توانیم ارسال جدیدی داشته باشیم Set_SCK_Disable رو هم برابر صفر می کنیم چون دیگه قراره از کلاک بعدی این شرط برقرار نباشه

بخش مربوط به خواندن:

```
    -----read
    if (SCK_Clock_Divider = to_unsigned(9,4) and SCK_Disable = '1') then

        Data_Out_Int(to_integer(SPI_Data_Out_Bit_Width))    <=  MISO_Int;
        SPI_Data_Out_Bit_Width      <=  SPI_Data_Out_Bit_Width + 1;

        if (SPI_Data_Out_Bit_Width = to_unsigned(to_integer(SPI_Data_Bit_Width(to_integer(Command_Type_Int))),3)) then

            Data_Out_Valid_Int      <=  SPI_OPcode_Byte or (not Force_CS_Buff);
            SPI_OPcode_Byte         <=  '1';

        end if;

    end if;
```

اگر فاز مربوط به خواندن داشته باشیم این بخش به کار ما می آید یک شرطی گذاشتیم که اگر SCK_CLOCK_Dividerکانتر مربوط به تولید کلاک (SPI برابر نه شد شرط برقرار بشه به نظر خودم بهترین جا برای خوندن دیتا انتهای پریود کلاک هست درست قبل از لبه پایین رونده کلاک اگر کانتر برابر نه باشد یعنی آخرین لحظه هستی که هنوز مقدار کلاک یک است و در انتهای کلاک است و در کلاک بعد کلاک SPIمیره صفر و لبه پایین رونده اول ایجاد میشه و تازه دیتا شروع میکنه به خوندن و شروع میکنه به برگشتن به مستر بنابراین قبل از اینکه لبه پایین رونده رخ بده دیتا رو بخونم یعنی دیتایی که لبه پایین رونده قبلی خارج شده الان به صورت پایدار در

دست من است است و در اینجا میتونم اون رو بخونم(دلیل اینکه اینجا مقدار و نه قرار دادیم) البته شرط دیگری قرار دادیم که SCK_Disable برابر یک است ما SCK_CLOCK_Dividerرو مقدار اولیه ۶ به اون دادیمبعد سه کلاک مقدار اون به نه میرسه و شرط برقرار میشه اما قرار نیست بعدسه کلاک این شرط انجام بشه بنابراین and دیگری گذاشتیم و اگر SCK_Disableبرابر یک شد شرط خوندن انجام بشه چون ابتدا برابر صفر است شرط انجام نمیشه تا اینکه برسه به ۹ و بعد بشه صفر بعد بیاد داخل شرط و مقدار اون تازه برابر یک بشه و سیکل بعد شرط خوندن برقرار بشه.

رسیدیم به انتهای پریود کلاک SPI می خواهیم دیتا رو بخونیم و MISO_Int رو در یک سیگنال میانی Data_Out_Int میریزیم که ۸ بیتی است.

یک سیگنال کانتر تعریف کردیم به اسم SPI_Data_Out_Bit_Width که قبلاً مقدار اون برابر صفر بود اونو در سبک ترین قرار میدهیم LSB_First هست یکی به کانتر اضافه میکنیم و شرطی هم گذاشتیم هر وقت مقدار این کانتر به هفت رسید کار خوندن تمامه اگر شرط برقرار بشه یک سیگنال Validرو به مدت یک کلاک برابر با یک قرار میدهیم که به صورت دائمی برابر صفر است فقط در این لحظه در یک کلاک برابر یک قرار می دهیم این لحظه وقتی است که آخر این بیت رو دریافت کردیم و به خروجی ارجاع داد همزمان ولید رو هم برابر یک قرار می دهیم ولی به جای اینکه Data_Out_Valid_Intرو مستقیماً برابر یک قرار بدیم آن را برابر SPI_OPCode_Byte OR (NOT Force_CS_Buff)قرار دادیم که سیگنال SPI_OPCode_Byteوقتی سیگنال sendاومد برابر صفر کردیم و اولین باری که SCK_CLOCK_Dividerبرابر نه میشه و همزمان شرط خط ۱۱۸ برقراره مقدار SPI_OPCode_Byteبرابر صفره زیرا در فاز اول خوندن هستیم برای مثال: فرض کنیم ارسال ما ۱۶ بیتی است بایت اول رایت و بایت دوم رید هست یعنی میخواهیم از اسلیو بخونیم اولین باری که این شرط اجرا می شود تازه فاز رایت هستیم یعنی اگر بخونیم دیتا ها بی اهمیت هستند چون در فاز رایت بودیم چون این دیتاها معنی ندارد موقعی که شرط خط ۱۲۳ میرسه نباید validبرابر یک بشه باید هنوز صفر بمونه تا دیتای اشتباهی نخونیم پس از یک سیگنال ای به نام SPI_OPCode_Byteاستفاده کردیم که در واقع مشخص میکند ارسالیم که به بایت اول ارسال میگیم OPCode۱۱ کد دستورالعمل داخل اون هست پس در اینجا صفر و

به Data_Out_Valid_Intارجاع می دهیم پس هیچ اتفاقی نیفتاده و کار ادامه پیدا میکنه و در همین شرط مقدار SPI_OPCode_Byteبرابر یک می کنیم چون در بایت های بعدی میتونیم عمل خوندن انجام بدیم مثلاً در بایت دوم دیتا ای که میخونم valid برابر یک میشه.

همین با یک سیگنال دیگه ای NOT Force_CS_Buff اور کردیم که وقتی سیگنال sendاومد Force_CS_Intرو داخل این سیگنال قرار دادیم تا نگه داریم.

کاربرد:Force_CS_Int

اگر بخواهیم ارسالی داشته باشیم و درون ارسالcs ، Didableنشه Force_CS_Buff باید برابر صفر بشه حالا فرض کنیم قرار بیشتر از هزار دیتا از اسلیو بخونیم قاعدتاً CS باید Disableبشود و بعد اون OPCodeرو ارسال کنم و بعد هزار بایت دیتا من از اسلیو بیاد تا بخونم وقتی اون هزار بایت رو بخونم با توجه به اینکه پرورت دیتا ورودی من ۳۲ بیتی است میتونم اونو در دیتاهای بلوک ۳۲ بیتی قرار بدم و در این ۳۲ بیت ها نباید CSما Disableبشه اما کد اینگونه است که هر بار یک دیتا ۳۲ بیتی قرار بدم و فرمان sendرو ارسال بکنم یکبار CSرو Disableبکنم برای رفع این مشکل ما از سیگنال Force_CSاستفاده کردیم یکبار یک فرمانی رو ارسال میکنیم که قاعدتاً باعث میشه CSهمDisableبشه و فرض کنیم بعد این فرمان قرار هزار بار بخونیم بدون اینکه CSما Disable بشود باید Force_CSروی یک بکنیم اگر به ابتدای کد ببریم CS_Intاند شده با Force_CS_Buffکه اند هر چیزی باصفر میشه صفر که باعث میشه هرچندCS_Intرو یک می کنم ولی اثری نمی گذارد چون اون رو با ۰ and کردیم

بلوک مربوط به کامند رجیستر های ADS :

```
GENERIC(ADS_NULL        : unsigned (31 downto 0):="00000000000000000000000000000000";
        ADS_RESET       : unsigned (31 downto 0):="00000000000000000000000000010001";
        ADS_STANDBY     : unsigned (31 downto 0):="00000000000000000000000000100010";
        ADS_WAKEUP      : unsigned (31 downto 0):="00000000000000000000000000110011";
        ADS_LOCK        : unsigned (31 downto 0):="00000000000000000000010101010101";
        ADS_UNLOCK      : unsigned (31 downto 0):="00000000000000000000011001010101");
```

در کد بالا مقادیر هر کامند به انها اختصاص داده شده که در واقع 16 بیت ان مورد نیاز است این حذف 16 بیت اضافی را SPI_Command_Type انجام میدهد

```
if rising_edge (clock) then
    SPI_Busy_Int                <= SPI_Busy;
    if (rst_ads  = '1' AND SPI_Busy_Int = '0') then
        SPI_Data_In_Int <= ADS_RESET;
        SPI_Send_Int <= '1';
        SPI_Command_Type_Int <= "11";
        SPI_Force_CS_Int <= '1';
        if SPI_Send_Int = '1' then SPI_Send_Int <= '0'; end if;
    end if;
```

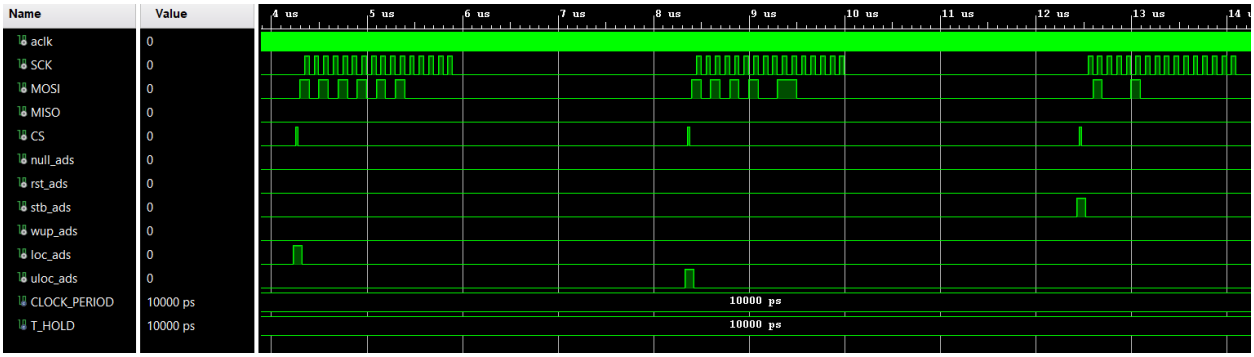
در بالا یک نمونه کد ارسال دیتا داریم اگر rst_ads برابر یک شد و SPI هم بیزی نبود انگاه مقدار دیتا ریست ارسال شده و send هم برابر 1 میشودSPI_Command_Type هم برابر 11 شده یعنی کل 32 بیت برای ما اهمیت دارد اگر میخواستیم 16 بیت ارسال کنیم میبایست مقدار ان را برابر 01 قرار میدادیم

در شرط بعدی ما مقدارsend را برابر صفر کردیم تا برای ارسال دیتا بعدی اماده باشد

تست بنچ و سیمولیشن :

```
63          -- Run for long enough to produce of outputs
64          wait for 100 ns;
65          rst_ads <= '1';
66
67          wait for 100 ns;
68          rst_ads <= '0';
69
70          wait for 4000 ns;
71          loc_ads <= '1';
72
73          wait for 100 ns;
74          loc_ads <= '0';
75
76          wait for 4000 ns;
77          uloc_ads <= '1';
78
79          wait for 100 ns;
80          uloc_ads <= '0';
81
82          wait for 4000 ns;
83          stb_ads <= '1';
84
85          wait for 100 ns;
86          stb_ads <= '0';
87          wait;
88
89      end process stimuli;
90
```

در کد تست بنچ بالا چند کامند به صورت تصادفی و در زمان های مختلف داده شده و نتیجه این کد رو در سیمولینک میتوان دید:



در کد شبیه سازی شده زیر یک سیگنال در زمان ارسال و برای ایجاد اختلال به SPI دادیم اما کد این را ریجکت کرد و در ارسال خود هیچ خللی به وجود نیامد.