دانشکده مهندسی برق

۶ کانال، ۱۴ بیت، ۱.۵ Msps نمونه‌برداری هم‌زمان از ADC LTC2351-14 به کمک پروتکل SPI

دانشجو:

نیلوفر دباغی داریان

استاد:

دکتر ستار میرزاکوچکی

پاییز ۱۴۰۱


بسم الله الرحمن الرحيم

چکیده

پروتکل‌های ارتباطی بسیاری به‌منظور انتقال اطلاعات در سامانه‌های دیجیتالی وجود دارد. این پروتکل‌ها می‌توانند به‌صورت موازی یا سریال جهت برقراری ارتباط مورد استفاده قرار گیرند. یکی از پرکاربردترین ارتباط‌های سریال رابط جانبی سریال[1] (SPI) نام دارد که این پروژه سعی در پیاده‌سازی این پروتکل مطابق دیتاشیت مربوطه جهت برقراری ارتباط با ADC LTC2351-14 دارد. توضیحات مربوط به بلوک دیاگرام ADC مربوطه، نحوه عملکرد پایه‌های آن و تعامل آن با SPI به‌طور کامل بیان شده است. SPI ها با دو ساختار سه سیم و چهار سیم طراحی شده‌اند. SPI مربوط به معماری ذکر شده، دارای ارتباط سه سیم می‌باشد. آشنایی با این قطعه، پیکربندی و برنامه‌ریزی صحیح آن، از اهداف این پروژه می‌باشد. بدین منظور، از نرم‌افزار ISE جهت کدنویسی VHDL برای شبیه‌سازی و صحت‌سنجی نتایج خروجی مطابق با آنچه در دیتاشیت آمده است، استفاده کرده‌ایم.

واژه‌های کلیدی: دیتاشیت، ADC LTC2351-14، SPI، فرکانس، SDO، CONV، SELX.

[1] Serial Peripheral Interface

فهرست مطالب

فهرست تصاویر

فهرست جداول

فصل ۱:

معرفی پروتکل SPI

۱-۱- مقدمه

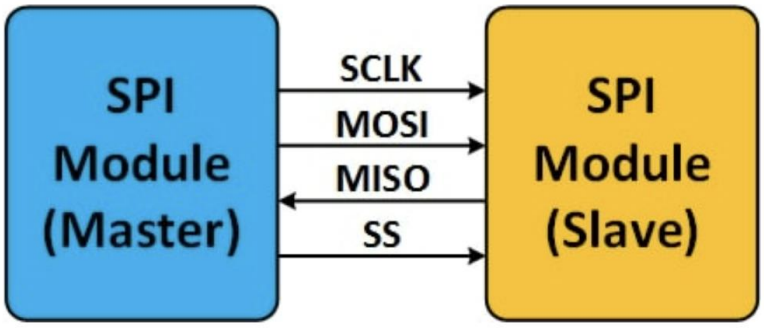
ارتباط SPI یک باس رابط است که معمولا برای ارسال داده بین میکروکنترلرها و لوازم جانبی کوچک مانند شیفت رجیسترها، حسگرها و SD کارت‌ها استفاده می‌شود. ارتباط سریال SPI از خطوط داده و کلاک‌های جداگانه و یک خط انتخاب برای گزینش دستگاهی که می‌خواهید با آن صحبت کنید، استفاده می‌کند.

۱-۲- عملکرد و معماری SPI

SPI یک باس داده همگام (سنکرون) است که از خطوط جداگانه برای داده‌ها استفاده می‌کند و نیز یک «ساعت» که هر دو طرف را در هماهنگی کامل نگه می‌دارد. ساعت یک سیگنال نوسانی است که به گیرنده زمان دقیق نمونه‌برداری از بیت‌های روی خط داده را می‌گوید. این فعال‌سازی می‌تواند در لبه بالارونده و یا لبه پایین رونده سیگنال ساعت باشد؛ دیتاشیت مشخص خواهند کرد که کدام یک از آن‌ها استفاده شده است. هنگامی که گیرنده آن لبه را تشخیص می‌دهد، بلافاصله خط داده را نمونه‌برداری می‌کند تا بیت بعدی را بخواند.

از این پروتکل در ارتباطات با فاصله کوتاه استفاده می‌شود و نوع ارتباط آن سنکرون است. در کامل‌ترین حالت، چهار سیم یا مسیر ارتباطی برای پیاده‌سازی این پروتکل به کار می‌رود.

به کمک این چهار سیم، امکان ایجاد یک ارتباط Full duplex به صورت Master/Slave فراهم می‌شود. اگر بیش از دو وسیله به کمک این پروتکل به یکدیگر متصل شوند، همواره یکی از آنها master و بقیه slave خواهند بود. شکل (۱-۱)، دیاگرامی از نحوه ارتباط بین دو وسیله را در پروتکل SPI نشان می‌دهد. لازم به ذکر است پروژه مربوطه دارای SPI سه سیم (SDO) است. با آشنایی با این پروتکل در حالت ۴ سیم، می‌توانیم آن را با معماری خواسته شده‌ی دیتاشیت پیاده‌سازی کنیم. لذا در ادامه به بررسی حالت کلی که ۴ سیم دارد، می‌پردازیم.



شکل (۱-۱): بلوک دیاگرام SPI.

از آنجایی که این یک ارتباط سریال سنکرون است، یکی از چهار خط ارتباطی پروتکل SPI سیگنال کلاک است. وسیله‌ای که سیگنال کلاک را ارسال می‌کند master نامیده می‌شود و وسیله‌ای که آن را دریافت می‌کند slave نام دارد. دومین خط ارتباطی، سیگنال انتخاب slave یا slave select (SS) است. این سیگنال در صورتیکه که فقط یک slave در سامانه وجود داشته باشد، می‌تواند همیشه فعال باشد و در این حالت نقش جدی در برقراری ارتباط ندارد.

دو سیگنال دیگر این ارتباط، برای انتقال داده‌ها از master به slave و بالعکس استفاده می‌شوند. سیگنال MOSI که مخفف Master Output Slave Input است برای انتقال داده‌ها از master به slave و سیگنال MISO که مخفف Master Input Slave Output است برای انتقال داده‌ها از slave به master استفاده می‌شود. این یک ارتباط سریال سنکرون است و بنابراین، انتقال هر بیت از داده‌ها بین master و slave هم‌زمان با لبه کلاک رخ می‌دهد. کلاک این ارتباط، در سمت master تولید شده و به slave ها ارسال می‌شود. چه در slave و چه در master، دریافت و ارسال بیت‌های داده‌ها باید هم‌زمان با این کلاک انجام شود.

در مورد سیگنال‌های ارتباطی پروتکل SPI به دو نکته توجه کنید. نکته اول اینکه در بعضی از سامانه‌ها، برای برقراری ارتباط SPI از سه سیم به جای چهار سیم استفاده می‌شود. در این حالت، به جای دو سیم برای انتقال داده‌ها بین master و slave، از یک سیم استفاده می‌شود. در نتیجه، پروتکل از حالت full duplex به half duplex تغییر ماهیت می‌دهد. نکته دوم در مورد سیگنال‌های پروتکل SPI مربوط به نام‌گذاری آنها است. در مراجع مختلف، ممکن است برای سیگنال SCLK از CLK یا SCK استفاده شود. به جای سیگنال MOSI ممکن است از SDI یا DI استفاده شود و به جای سیگنال MISO از SDO یا DO استفاده شود.

در ارتباطات آسنکرون، مثل پروتکل RS232، سرعت ارتباط باید از قبل بین دو وسیله که با هم در ارتباط هستند مشخص باشد. در غیر اینصورت، ارتباط با مشکل مواجه خواهد شد. اما در مورد ارتباط SPI، تعیین کننده سرعت ارتباط، همان سیگنال کلاک است که از master ارسال می‌شود. بنابراین نیازی به توافق قبلی بین master و slave نیست. البته هر مدار دیجیتالی، دارای یک حداکثر فرکانس کلاک قابل اعمال است و بنابراین، فرکانس کلاک SPI هم نمی‌تواند از آن مقدار بیشتر شود.

۱-۳- ارسال و دریافت داده‌ها در پروتکل SPI

از آنجاییکه SPI یک ارتباط سنکرون است، هر بیت داده در master و slave هم‌زمان با کلاک ارسال یا دریافت می‌شود. به عبارت دیگر، هم‌زمان با هر کلاک، یک بیت از داده‌ها از master به slave و یک بیت از slave به master می‌تواند منتقل شود. طول این داده‌ها بستگی به قراردادی دارد که در یک وسیله خاص وجود دارد و ممکن است ۸ بیت، ۱۰ بیت، ۱۶ بیت، ۳۲ بیت یا هر مقدار دیگری باشد.

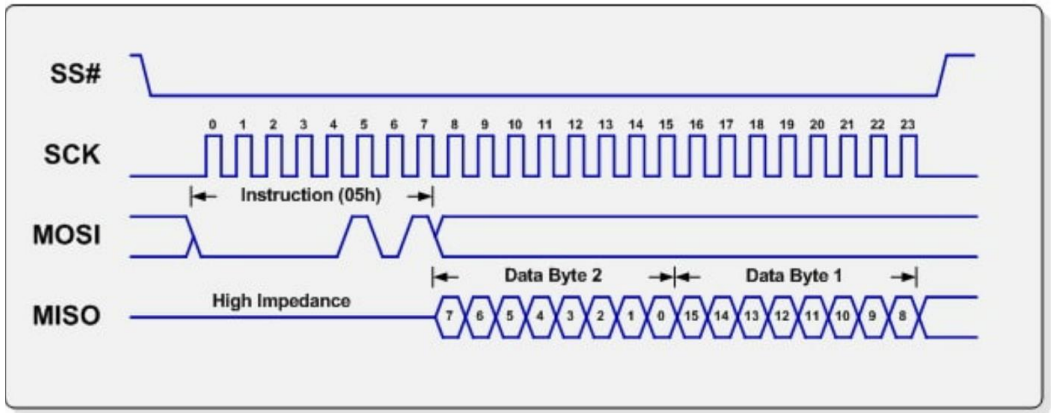
مثلا در یک ارتباط SPI برای انتقال داده‌های هشت بیتی، گیرنده SPI طوری طراحی شده است که بعد از فعال شدن ارتباط به وسیله سیگنال select slave، بعد از هر هشت کلاک، یک داده‌ی کامل دریافت شده است و در کلاک بعدی باید اولین بیت داده‌ی جدید را دریافت کند.

در پروتکل SPI، وسیله‌ای که در حال دریافت داده‌ها است، باید از قبل بداند که داده دریافتی چند بیتی است. مثلا اگر master داده‌ای را به slave ارسال کند و در پاسخ به آن داده، انتظار داشته باشد که slave هم داده‌ای را به master ارسال کند، master باید کلاک را به تعداد لازم ادامه دهد تا slave بتواند به کمک آن، داده خود را به master ارسال کند.

ارسال هر کدام از این داده‌ها می‌تواند از بیت سنگین یا از بیت سبک آغاز شود. این موضوع بستگی به قراردادی دارد که در سامانه مورد استفاده شما وجود دارد. در بسیاری از سامانه‌ها، اینکه ارسال داده‌ها از بیت سنگین شروع شود یا از بیت سبک، قابل تنظیم است.

مثلا ممکن است شما از قبل، ماجول یک ارتباط SPI را در FPGA به کمک زبان VHDL آماده کرده باشید و این ماجول را طوری طراحی کرده باشید که داده‌ها را از بیت سبک آنها ارسال و دریافت می‌کند. برای ارتباط ماجولی که طراحی کرده‌اید با یک ماجول خارجی، می‌توانید برای اینکه مجبور نباشید تغییری در کدی که نوشتید بدهید، ماجول خارجی را طوری تنظیم کنید که ارسال و دریافت داده‌ها را از بیت سبک انجام دهد تا با ماجول شما سازگار شود.

شکل(۱-۲)، یک نمونه از زمان‌بندی سیگنال‌ها را در ارتباط SPI نشان می‌دهد. در این شکل، سیگنال SS ابتدا صفر شده است که این باعث انتخاب و فعال شدن ماجول slave می‌شود. در ابتدا، یک داده‌ی هشت بیتی از سمت master به slave ارسال شده است. سپس دو بایت پشت سر هم از سمت slave به master ارسال شده است. توجه داشته باشید که حتی وقتی ارسال داده از slave به master است، کلاک ماجول‌ها توسط master تولید می‌شود. بنابراین، بلوک master باید از قبل، از طول داده‌ی دریافتی از slave مطلع باشد تا بتواند به تعداد لازم کلاک ارسال کند.



شکل (۱-۲): پروتکل SPI.

فصل ۲:

معرفی ADC LTC2351-14

۲-۱- ویژگی‌های

ADC مربوطه دارای ویژگی‌هایی به شرح زیر است:

- ✓ ۱.۵ Msps با شش نمونه به‌طور هم‌زمان
- ✓ ورودی‌های دیفرانسیل
- ✓ ۲۵۰kps گذردهی در هر کانال
- ✓ ۷۵ dB SINAD[1]
- ✓ اتلاف توان کم: ۱۶.۵ mW
- ✓ عملکرد یک منبع ۳ ولت
- ✓ مرجع فاصله باند داخلی ۲.۵ ولت
- ✓ رابط سریال سازگار با ۳ سیم SPI
- ✓ شروع تبدیل داخلی توسط پایه CONV
- ✓ حالت خاموش شدن خواب (۱۲ µW)
- ✓ حالت خاموش کردن استراحت (۴.۵ mW)
- ✓ محدوده ورودی ۰ ولت تا ۲.۵ ولت تک قطبی، یا ± ۱.۲۵ ولت دیفرانسیل دوقطبی
- ✓ بسته QFN[2] کوچک ۳۲ پین (۵mm * ۵mm)

۲-۲- بلوک دیاگرام

LTC®2351-14 یک ADC ۱۴بیتی با سرعت ۱.۵ Msps با شش ورودی دیفرانسیل نمونه‌برداری هم‌زمان است. LTC2351-14 شامل شش ورودی دیفرانسیل جداگانه است که به‌طور هم‌زمان در لبه افزایشی سیگنال CONV نمونه‌برداری می شود. سپس این شش ورودی نمونه‌برداری شده با نرخ ۲۵۰ kps در هر کانال تبدیل می‌شوند. بسته به وضعیت منطقی ورودی‌های SEL2، SEL1 و SEL0، توالی تبدیل را می‌توان برای تبدیل کمتر از شش کانال کاهش داد.

در این پروژه از ۱ کانال برای انتقال داده ۱۴ بیتی استفاده شده است. زیرا ۶ کانال ورودی ذکر شده، صرفا ورودی‌های آنالوگ قطعه هستند و در اینجا مورد تمرکز و هدف این پروژه قرار نگرفته‌اند. هدف این پروژه بررسی و شبیه‌سازی پروتکل SPI سه سیم با توجه به مشخصاتی است که به دلیل تعامل آن با قطعه ADC از روی دیتاشیت آن قابل دریافت است. با توجه به دیتاشیت، فرکانس ۱.۵ Msps می‌باشد که این مقدار، تعیین کننده‌ی فرکانس SPI است که در قسمت پیکربندی توضیحات بیشتر بیان شده است.

[1] Signal-to-Noise and Distortion Ratio
[2] Quad Flat No-Lead

۲-۲-۱- توصیف پایه‌ها

SDO (پایه ۱): خروجی داده سریال سه حالته. هر مجموعه از شش کلمه داده خروجی نشان دهنده شش کانال ورودی آنالوگ در شروع تبدیل قبلی است. داده‌های CH0 ابتدا و داده‌های CH5 در پایان منتشر می‌شوند. از هر کلمه داده ابتدا MSB بیرون می‌آید.

OGND (پایه ۲): بازگشت زمینی برای جریان‌های SDO. به صفحه زمین جامد متصل می‌شود.

OVDD (پایه ۳): منبع تغذیه برای پایه SDO. OVDD نباید بیش از ۳۰۰ میلی ولت بالاتر از VDD باشد و می توان آن را به ولتاژ پایین‌تری رساند تا با خانواده‌های منطقی ولتاژ پایین ارتباط برقرار کند. حالت HIGH بدون بار در SDO در پتانسیل OVDD است.

$CH0^+$ (پایه ۴): کانال غیر معکوس ۰. $CH0^+$ نسبت به $CH0^-$ با نوسان دیفرانسیلی کامل ۰ ولت تا ۲.۵ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

$CH0^-$ (پایه ۵): کانال معکوس ۰. $CH0^-$ نسبت به $CH0^+$ با نوسان دیفرانسیلی کامل ۲.۵- ولت تا ۰ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

GND (پایه ۶، ۹، ۱۲، ۱۳، ۱۶ و ۱۹): زمین‌های آنالوگ این پایه‌های زمین باید مستقیماً به صفحه زمین زیر قطعه گره بخورند. جریان سیگنال آنالوگ از طریق این اتصالات جریان دارد.

$CH1^+$ (پایه ۷): کانال غیر معکوس ۰. $CH1^+$ نسبت به $CH1^-$ با نوسان دیفرانسیلی کامل ۰ ولت تا ۲.۵ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

$CH1^-$ (پایه ۸): کانال معکوس ۰. $CH1^-$ نسبت به $CH1^+$ با نوسان دیفرانسیلی کامل ۲.۵- ولت تا ۰ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

$CH2^+$ (پایه ۱۰): کانال غیر معکوس ۰. $CH2^+$ نسبت به $CH2^-$ با نوسان دیفرانسیلی کامل ۰ ولت تا ۲.۵ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

$CH2^-$ (پایه ۱۱): کانال معکوس ۰. $CH2^-$ نسبت به $CH2^+$ با نوسان دیفرانسیلی کامل ۲.۵- ولت تا ۰ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

$CH3^+$ (پایه ۱۴): کانال غیر معکوس ۰. $CH3^+$ نسبت به $CH3^-$ با نوسان دیفرانسیلی کامل ۰ ولت تا ۲.۵ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

$CH3^-$ (پایه ۱۵): کانال معکوس ۰. $CH3^-$ نسبت به $CH3^+$ با نوسان دیفرانسیلی کامل ۲.۵- ولت تا ۰ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

$CH4^+$ (پایه ۱۷): کانال غیر معکوس ۰. $CH4^+$ نسبت به $CH4^-$ با نوسان دیفرانسیلی کامل ۰ ولت تا ۲.۵ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

$CH4^-$ (پایه ۱۸): کانال معکوس ۰. $CH4^-$ نسبت به $CH4^+$ با نوسان دیفرانسیلی کامل ۲.۵- ولت تا ۰ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

$CH5^+$ (پایه ۲۰): کانال غیر معکوس ۰. $CH5^+$ نسبت به $CH5^-$ با نوسان دیفرانسیلی کامل ۰ ولت تا ۲.۵ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

$CH2^-$ (پایه ۲۱): کانال معکوس ۰. $CH5^-$ نسبت به $CH5^+$ با نوسان دیفرانسیلی کامل ۲.۵- ولت تا ۰ ولت یا نوسان دیفرانسیلی ۱.۲۵± ولت و محدوده ورودی مطلق ۰ ولت تا VDD عمل می‌کند.

GND (پایه ۲۲): زمین آنالوگ برای مرجع. زمین آنالوگ باید مستقیماً به صفحه زمین زیر قطعه متصل شود. جریان سیگنال آنالوگ از طریق این اتصال جریان می‌یابد. خازن بای‌پس مرجع μF ۱۰ باید به این صفحه برگردانده شود.

V_{REF} (پایه ۲۳): مرجع داخلی ۲.۵ ولت بای‌پس به GND و یک صفحه زمین آنالوگ با خازن سرامیکی μF ۱۰ (یا تانتالیم μF ۱۰ به موازات با سرامیک μF ۰.۱).

VCC (پایه ۲۴): منبع تغذیه آنالوگ مثبت ۳ ولت. این پایه ۳ ولت را به بخش آنالوگ متصل می‌کند. بای پس به صفحه زمین آنالوگ با خازن سرامیکی µF ۱۰ یا تانتالیم μF ۱۰ به موازات با سرامیک μF ۰.۱. باید دقت کرد که خازن بای‌پس µF ۰.۱تا حد امکان نزدیک به پایه ۲۴ قرار گیرد. پایه ۲۴ باید به پایه ۲۵ گره بخورد.

VDD (پایه ۲۵): منبع تغذیه دیجیتال مثبت ۳ ولت. این پایه ۳ ولت را به بخش منطق متصل می‌کند. دور زدن پین DGND و صفحه زمین آنالوگ جامد با یک خازن سرامیکی µF ۱۰ (یا تانتالیوم µF۱ ۱۰ به موازات با سرامیک µF ۰.۱). به خاطر داشته باشید که جریان‌های سیگنال خروجی دیجیتال داخلی از طریق این پایه جریان می‌یابد. باید دقت کرد که خازن بای‌پس µF ۰.۱ تا حد امکان نزدیک به پایه ۲۵ قرار گیرد. پایه ۲۵ باید به پایه ۲۴ گره بخورد.

SEL2 (پایه ۲۶): با ارزش‌ترین (مهم‌ترین) بیت کنترل تعداد کانال‌های در حال تبدیل. در ترکیب با SEL1 و SEL0، ۰۰۰ فقط اولین کانال (CH0) را برای تبدیل انتخاب می‌کند. افزایش SELx کانال‌های اضافی (CH0-CH5) را برای تبدیل انتخاب می‌کند. ۱۰۱، ۱۱۰ یا ۱۱۱ هر شش کانال را برای تبدیل انتخاب کنید. در هنگام تبدیل و در طول تبدیل بعدی به خواندن داده‌ها باید در حالت ثابت نگه داشته شود.

SEL1 (پایه ۲۷): بیت قابل توجه متوسط کنترل تعداد کانال‌های در حال تبدیل. در ترکیب با SEL0 و SEL2، ۰۰۰ فقط اولین کانال (CH0) را برای تبدیل انتخاب می‌کند. افزایش SELx کانال‌های اضافی (CH0-CH5) را برای تبدیل انتخاب می‌کند. ۱۰۱، ۱۱۰ یا ۱۱۱ هر شش کانال را برای تبدیل انتخاب کنید. در هنگام تبدیل و در طول تبدیل بعدی به خواندن داده‌ها باید در حالت ثابت نگه داشته شود.

SEL0 (پایه ۲۸): حداقل بیت قابل توجه کنترل تعداد کانال‌های در حال تبدیل. در ترکیب با SEL1 و SEL2، ۰۰۰ فقط اولین کانال (CH0) را برای تبدیل انتخاب می‌کند. افزایش SELx کانال‌های اضافی (CH0-CH5) را برای تبدیل انتخاب می‌کند. ۱۰۱، ۱۱۰ یا ۱۱۱ هر شش کانال را برای تبدیل انتخاب کنید. در هنگام تبدیل و در طول تبدیل بعدی به خواندن داده‌ها باید در حالت ثابت نگه داشته شود.

BIP (پایه ۲۹): حالت دوقطبی / تک قطبی. محدوده اختلاف ورودی ۰ ولت تا ۲.۵ ولت وقتی BIP کم است، و ۱.۲۵ ± ولت وقتی BIP بالا است. در طول تبدیل و در طول تبدیل بعدی به داده‌ها باید در حالت ثابت نگه داشته شود. هنگام تغییر BIP بین تبدیل‌ها، باید قبل از شروع تبدیل بعدی، زمان کسب کامل مجاز باشد. داده‌های خروجی در شکل مکمل ۲ برای حالت دوقطبی و شکل باینری مستقیم برای حالت تک قطبی هستند.

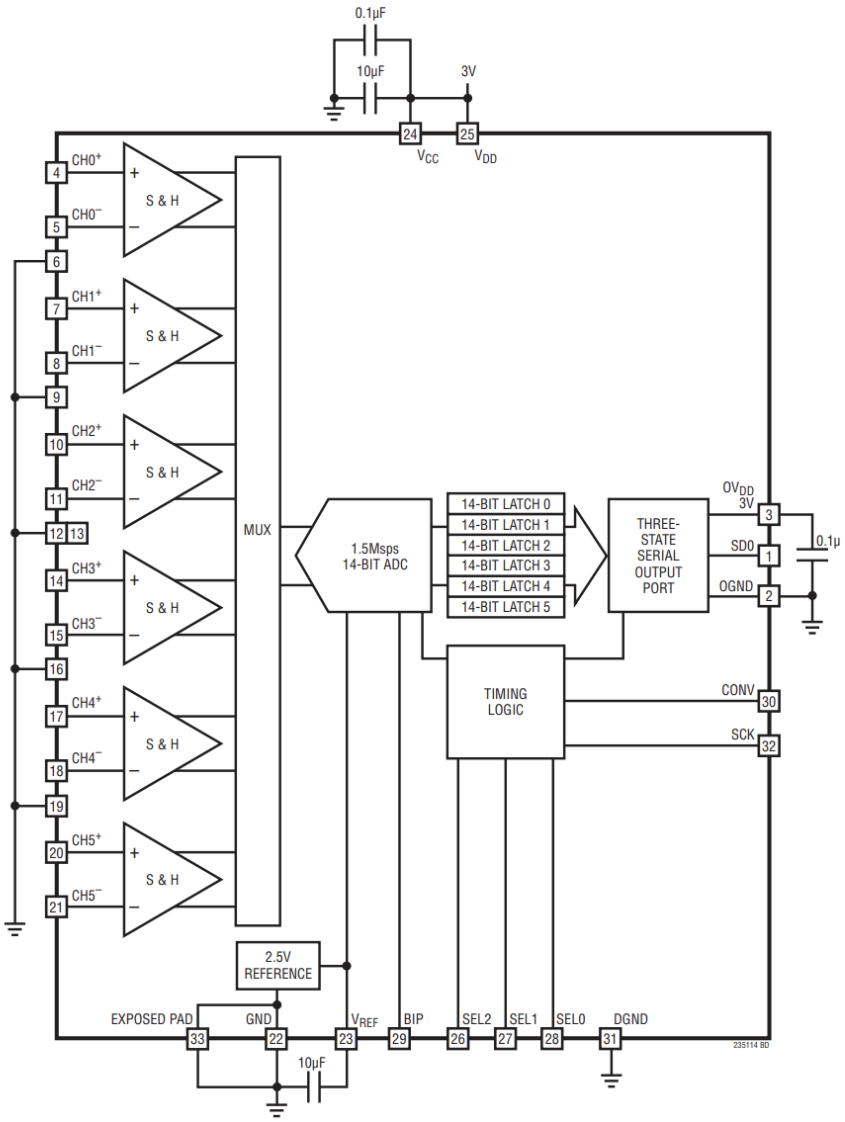
CONV (پایه ۳۰): شروع تبدیل. شش سیگنال ورودی آنالوگ را نگه می‌دارد و تبدیل را در لبه افزایشی شروع می‌کند. دو پالس CONV با SCK در حالت ثابت HIGH یا ثابت LOW حالت استراحت را شروع می‌کند. چهار یا چند پالس CONV با SCK در حالت ثابت HIGH یا ثابت LOW حالت خواب را شروع می‌کند.

DGND (پایه ۳۱): زمین دیجیتال. این پایه زمین باید مستقیماً به صفحه زمین متصل شود. جریان سیگنال ورودی دیجیتال از طریق این پایه جریان می‌یابد.

SCK (پایه ۳۲): ورودی ساعت خارجی. فرآیند نسخه تبدیل و دنباله خروجی را در SD0 (پایه ۱) در لبه افزایشی را پیش می‌برد. یک یا چند پالس SCK از حالت‌های ذخیره انرژی خواب یا استراحت بیدار می‌شوند. ۱۶ چرخه ساعت برای هر یک از کانال‌هایی که توسط SELx فعال می‌شوند (پایه‌های ۲۶، ۲۷، ۲۸) مورد نیاز است، تا در مجموع ۹۶ چرخه ساعت برای تبدیل و خواندن هر شش کانال مورد نیاز است.

Exposed Pad (پایه ۳۳): GND. باید مستقیماً به صفحه زمین متصل شود.

شکل (۲-۱) بلوک دیاگرام پایه‌ها را نمایش می‌دهد.



شکل (۲-۱): بلوک دیاگرام ADC LTC2351-14.

سه پایه (SELx) کنترل تعداد کانال‌های در حال تبدیل را انتخاب می‌کنند. ۰۰۰ فقط اولین کانال (CH0) را برای تبدیل انتخاب می‌کند. افزایش SELx کانال‌های اضافی را برای تبدیل انتخاب می‌کند، تا شش کانال. ۱۰۱، ۱۱۰ یا ۱۱۱ هر شش کانال را برای تبدیل انتخاب کنید. این پایه‌ها باید در هنگام تبدیل و در طول تبدیل بعدی برای خواندن داده‌ها در حالت ثابت نگه داشته شوند. هنگام تغییر حالت‌ها بین تبدیل‌ها، به خاطر داشته باشید که داده‌های خروجی یک کانال خاص تا زمانی که دوباره آن کانال تبدیل نشود، بدون تغییر باقی می‌ماند. به‌عنوان مثال: یک دنباله از چهار کانال (CH0، CH1، CH2، CH3) را با SELx = 011 تبدیل کنید، سپس، پس از تبدیل این کانال‌ها، SELx را به ۰۰۱ تغییر دهید تا فقط CH0 و CH1 تبدیل شود. جدول (۲-۱) را ببینید. در طول تبدیل اولین مجموعه از دو کانال، می‌توانید داده‌های همان دو کانال تبدیل شده را به عنوان بخشی از گروه چهار کانال قبلی را بخوانید. بعداً، می‌توانید چهار یا چند کانال را برای بازخوانی داده‌های خوانده‌نشده CH2 و CH3 که در اولین مجموعه چهار کانالی تبدیل شده بود، را تبدیل کنید. این پایه‌ها اغلب برای فعال کردن تعداد مناسب کانال برای یک برنامه خاص، سیم کشی شده‌اند. انتخاب تبدیل کانال کمتر به ازای هر تبدیل منجر به خروجی سریع‌تر آن کانال‌ها می‌شود. به‌عنوان مثال، شش کانال را می‌توان با سرعت ۲۵۰ kps/ch تبدیل کرد، در حالی که سه کانال را می‌توان با سرعت ۵۰۰ kps/ch تبدیل کرد.

جدول (۲-۱): کنترل توالی تبدیل.

SEL2	SEL1	SEL0	CHANNEL ACQUISITION AND CONVERSION SEQUENCE
0	0	0	acquire, CH0, acquire, CH0...
0	0	1	acquire, CH0, CH1, acquire, CH0, CH1...
0	1	0	acquire, CH0, CH1, CH2, acquire, CH0, CH1, CH2...
0	1	1	acquire, CH0, CH1, CH2, CH3, acquire, CH0, CH1, CH2, CH3...
1	0	0	acquire, CH0, CH1, CH2, CH3, CH4, acquire, CH0,CH1,CH2, CH3, CH4...
1	0	1	acquire, CH0, CH1, CH2, CH3, CH4, CH5, acquire, CH0, CH1, CH2, CH3, CH4, CH5...
1	1	0	acquire, CH0, CH1, CH2, CH3, CH4, CH5, acquire, CH0, CH1, CH2, CH3, CH4, CH5...
1	1	1	acquire, CH0, CH1, CH2, CH3, CH4, CH5, acquire, CH0, CH1, CH2, CH3, CH4, CH5...

LTC2351-14 دارای یک SPI 3 سیم (رابط جانبی سریال) است. ورودی‌های SCK و CONV و خروجی SDO این رابط را پیاده‌سازی می‌کنند. ورودی‌های SCK و CONV نوسان‌هایی را از منطق ۳ ولت می‌پذیرند و با TTL سازگار هستند، اگر نوسان منطقی از VDD بیشتر نباشد. شرح مفصلی از سه سیگنال درگاه سریال به شرح زیر است:

ورودی شروع تبدیل (CONV):

لبه افزایشی CONV یک تبدیل را شروع می‌کند، اما لبه‌های افزایشی بعدی در CONV توسط LTC2351-14 نادیده گرفته می‌شود تا زمانی که ۹۶ لبه بالارونده SCK زیر رخ دهد. چرخه وظیفه CONV را می‌توان به طور دلخواه انتخاب کرد تا به‌عنوان سیگنال همگام‌سازی فریم برای درگاه سریال پردازنده استفاده شود.

یک روش ساده برای تولید CONV ایجاد پالسی با عرض یک SCK برای درایو LTC2351-14 و سپس بافر کردن این سیگنال برای هدایت ورودی همگام‌سازی فریم درگاه سریال پردازنده است. تمرین خوبی است که ابتدا ورودی CONV LTC2351-14 را هدایت کنید تا از تداخل نویز دیجیتال در طول انتقال نمونه به نگه داشتن که توسط CONV در شروع تبدیل ایجاد می شود، جلوگیری شود. همچنین تمرین خوبی است که عرض قسمت پایین سیگنال CONV را بیشتر از ۱۵ ثانیه نگه دارید تا از ایجاد اشکال در انتهای جلویی ADC درست قبل از اینکه نمونه و نگه‌داشتن به حالت نگهداری در لبه افزایشی CONV برود، جلوگیری شود.

ورودی ساعت سریال (SCK):

لبه در حال افزایش SCK فرآیند تبدیل را پیش می‌برد و همچنین هر بیت را در جریان داده SDO به‌روز می‌کند. پس از افزایش CONV، سومین لبه بالارونده SCK حداکثر شش مجموعه از ۱۴ بیت داده را ارسال می کند و MSB ابتدا ارسال می‌شود. یک روش ساده این است که ابتدا SCK تولید کنید تا LTC2351-14 را هدایت کند و سپس این سیگنال را با تعداد مناسب معکوس بافر کنید تا ورودی ساعت سریال پورت سریال پردازنده را هدایت کند. از لبه پایین‌رونده ساعت استفاده کنید تا داده‌ها را از خروجی داده سریال (SDO) به درگاه سریال پردازنده خود متصل کنید. داده‌های سریال ۱۴ بیتی در شش کلمه ۱۶ بیتی با ۹۶ ساعت یا بیشتر در هر همگام‌سازی فریم دریافت می‌شود. اگر کمتر از شش کانال توسط SEL0-SEL2 برای تبدیل انتخاب شود، در این صورت برای تبدیل ورودی‌های آنالوگ و خواندن داده‌های حاصل پس از پالس تبدیل بعدی، به ۱۶ ساعت در هر کانال نیاز است. تمرین خوبی است که ابتدا ورودی SCK LTC2351-14 را هدایت کنید تا از تداخل نویز دیجیتال در هنگام تصمیم‌گیری مقایسه بیت داخلی توسط مقایسه‌کننده داخلی با سرعت بالا جلوگیری شود. برخلاف ورودی CONV، ورودی SCK به جیتر حساس نیست زیرا سیگنال ورودی قبلاً نمونه برداری شده و ثابت نگه داشته شده است.

خروجی داده سریال (SDO):

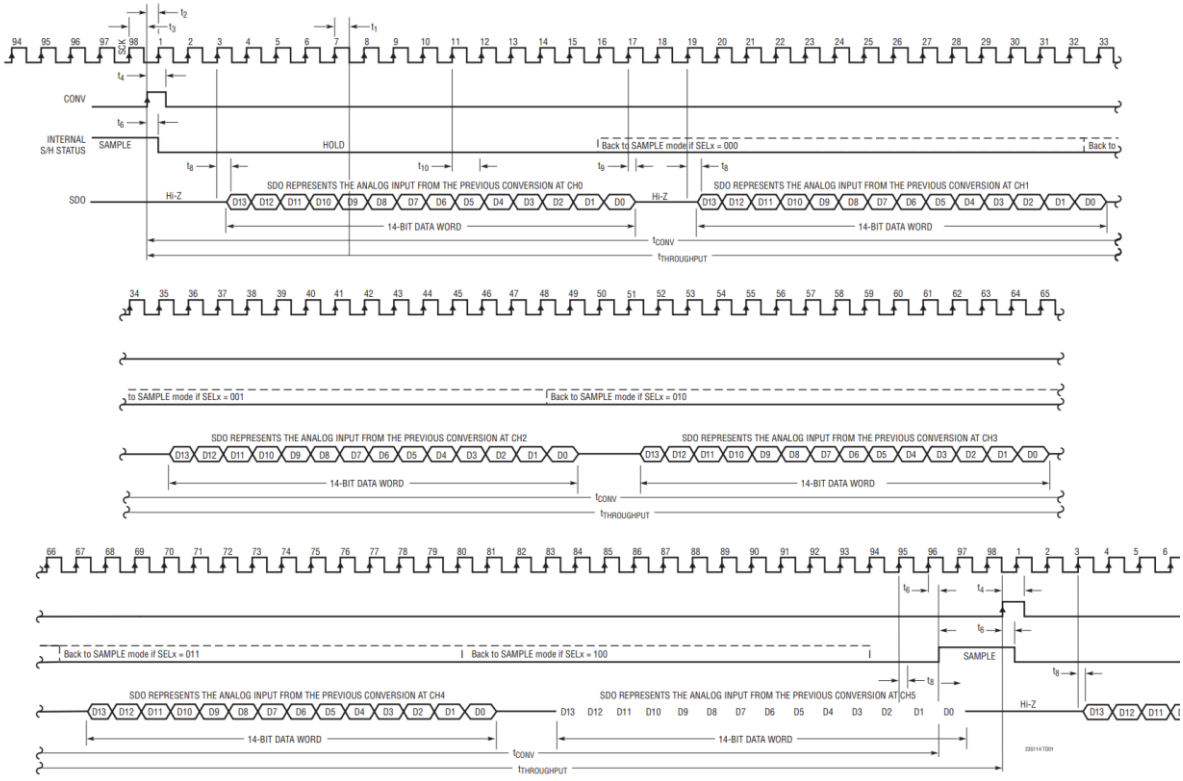
پس از روشن شدن، خروجی SDO به‌طور خودکار به حالت امپدانس بالا بازنشانی می‌شود. خروجی SDO در امپدانس بالا باقی می‌ماند تا زمانی که یک تبدیل جدید شروع شود. SDO تا شش مجموعه ۱۴ بیتی را در جریان داده خروجی پس از سومین لبه بالارونده SCK پس از شروع تبدیل با لبه افزایشی CONV ارسال می‌کند. شش یا کمتر کلمه ۱۴ بیتی با دو بیت مراقبتی و دو چرخه ساعت در حالت امپدانس بالا از هم جدا می‌شوند. لطفاً به مشخصات تأخیر از SCK به SDO معتبر توجه کنید. SDO همیشه با لبه بالارونده بعدی SCK معتبر است. جریان داده خروجی ۱۶ تا ۹۶ بیتی با درگاه سریال ۱۶ بیتی یا ۳۲ بیتی اکثر پردازنده‌ها سازگار است.

۲-۳- زمان‌بندی‌ها

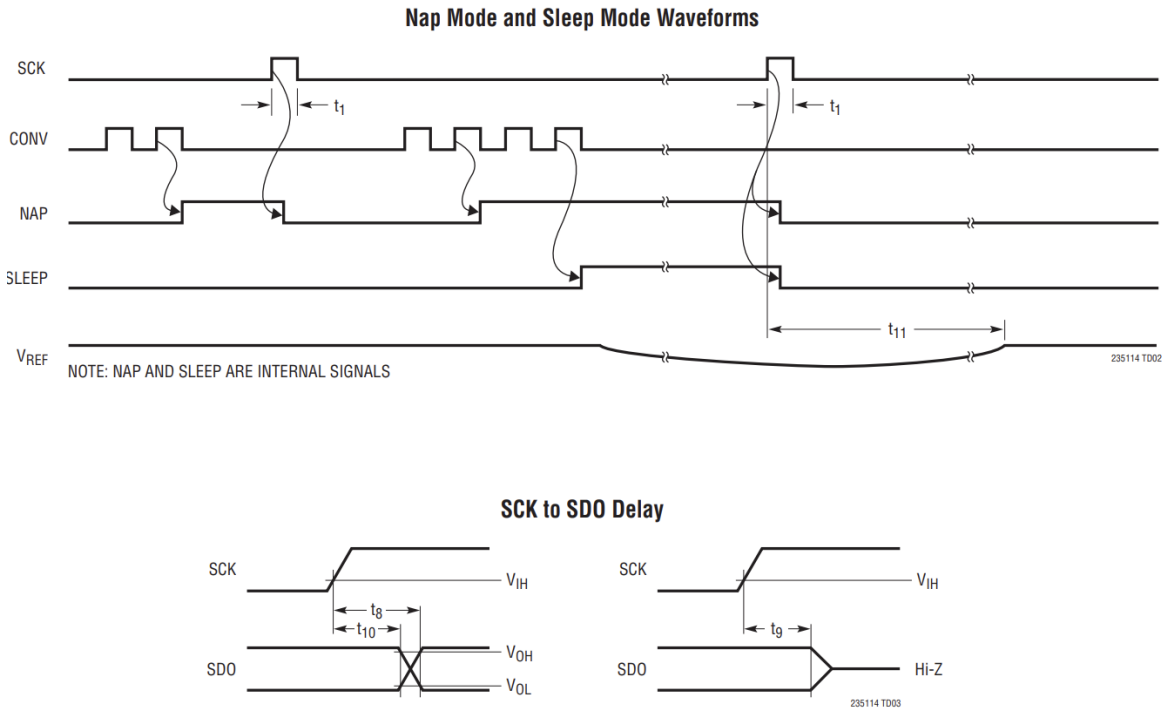
جدول (۲-۲) و شکل (۲-۲) زمان‌بندی قطعه را نمایش می‌دهند. نحوه استفاده و چگونگی شبیه‌سازی و زمان‌بندی در فصل بعد به تفصیل بیان شده است.

جدول (۲-۲): زمان‌بندی ADC LTC2351-14.

SYMBOL	PARAMETER
$f_{SAMPLE(MAX)}$	Maximum Sampling Rate per Channel (Conversion Rate)
$t_{THROUGHPUT}$	Minimum Sampling Period (Conversion + Acquisiton Period)
t_{SCK}	Clock Period
t_{CONV}	Conversion Time
t_1	Minimum High or Low SCLK Pulse Width
t_2	CONV to SCK Setup Time
t_3	SCK Before CONV
t_4	Minimum High or Low CONV Pulse Width
t_5	SCK↑ to Sample Mode
t_6	CONV↑ to Hold Mode
t_7	96th SCK↑ to CONV↑ Interval (Affects Acquisition Period)
t_8	Minimum Delay from SCK to Valid Bits 0 Through 11
t_9	SCK↑ to Hi-Z at SDO
t_{10}	Previous SDO Bit Remains Valid After SCK
t_{11}	V_{REF} Settling Time After Sleep-to-Wake Transition



(الف)

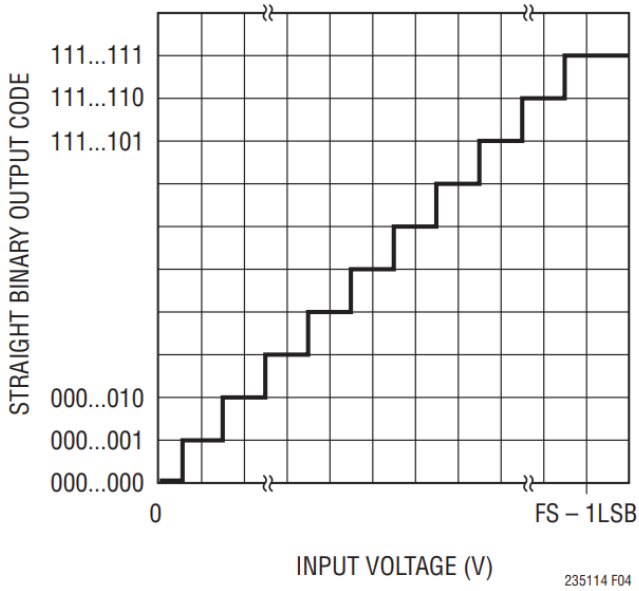


(ب)

شکل (۲-۲): زمان‌بندی ADC LTC2351-14.

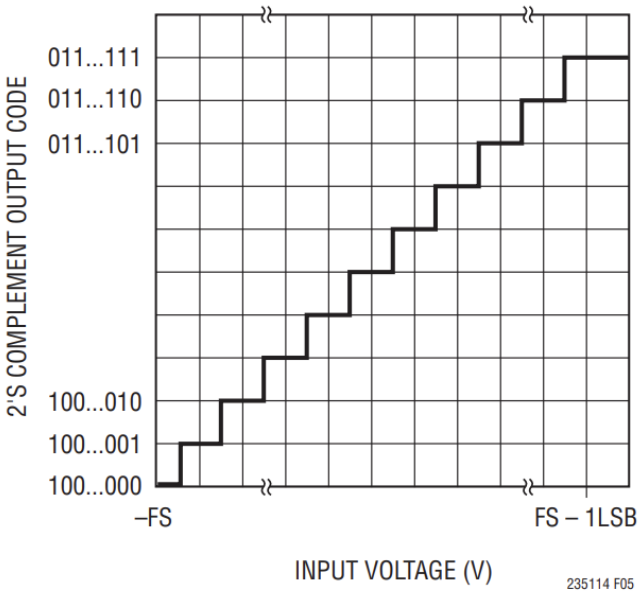
۲-۴- محاسبه سطح ولتاژ مبتنی بر داده‌ی دریافتی

محدوده ورودی دیفرانسیل دارای یک دامنه ولتاژ تک قطبی است که برابر با اختلاف ولتاژ در خروجی بافر مرجع VREF (پایه ۲۳) و ولتاژ در زمین است. محدوده ورودی دیفرانسیل ADC هنگام استفاده از مرجع داخلی ۰ ولت تا ۲.۵ ولت است. ADC داخلی به این دو گره ارجاع داده می‌شود. این رابطه با یک مرجع خارجی نیز صادق است. ADC همیشه تفاوت CH+ منهای CH- را مستقل از ولتاژ حالت مشترک در هر جفت ورودی تبدیل می‌کند. شکل (۲-۳) ویژگی‌های ورودی/خروجی ایده آل برای LTC2351-14 را در حالت تک قطبی (BIP = LOW) نشان می‌دهد. انتقال کد در میانه راه بین مقادیر LSB اعداد صحیح متوالی (یعنی ۰.۵ LSB، ۱.۵ LSB، ۲.۵ LSB، FS - 1.5 LSB) رخ می دهد. کد خروجی باینری مستقیم با 1LSB = 2.5V/16384 = 153µV برای LTC2351-14 است. LTC2351-14 دارای ۰.۷ LSB RMS نویز سفید گاوسی است.



شکل (۲-۳): مشخصه انتقال LTC2351-14 در حالت تک قطبی (BIP = LOW).

شکل (۲-۴) ویژگی‌های ورودی/خروجی ایده‌آل برای LTC2351-14 را در حالت دوقطبی نشان می‌دهد (BIP= HIGH). انتقال کد در میانه راه بین مقادیر LSB اعداد صحیح متوالی (یعنی ۰.۵ LSB، ۱.۵ LSB، ۲.۵ LSB، FS - 1.5 LSB) رخ می‌دهد. کد خروجی مکمل ۲ با 1LSB = 2.5V/16384 = 153µV برای LTC2351-14 است. LTC2351-14 دارای ۰.۷ LSB RMS نویز سفید گاوسی است.



شکل (۲-۴): مشخصه انتقال LTC2351-14 در حالت تک قطبی (BIP = HIGH).

فصل ۳:

نحوه پیکربندی SPI و تعامل آن با

ADC LTC2351-14

۳-۱- توصیف موجودیت

در اینجا ابتدا پس از معرفی سه کتابخانه اصلی، موجودیت کد به همراه پورت‌های مورد نیاز تعریف شده است. کد شبیه‌سازی شده مطابق دیتاشیت مربوطه، در حالت MOSI می‌باشد که معادل حالت SDO است. ابتدا پورت‌های ورودی و خروجی مورد نیاز، مطاق دیتاشیت را تعریف می‌نماییم. تمامی پورت‌های به صورت تک بیتی تعریف شده است. CLK_SYS توسط یک نوسان‌ساز تولید می‌شود. CONV_START که تغییر آن، محل شروع ارتباط SPI است. توجه شود نکته‌ای که اینجا حایز اهمیت است آن است که برخلاف ساختارهای معمول که کلاک، خروجی هست در اینجا کلاک (SCK) به عنوان ورودی در نظر گرفته شده است چرا که FPGA اینجا به عنوان تولیدکننده اصلی کلاک معرفی شده و در نهایت کلاک در قسمت تست بنچ تعریف می‌شود. Chip select و MOSI هم به عنوان پورت‌های خروجی تعریف شده است.

```
----------------------------------------
-- Engineer: Niloufar Dabaghi Daryan
-- Module Name:    SPI_MOSI/SDO Behavioral
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity SPI_MOSI is
    Port (

   -----INPUTS-----
           CLK_SYS     : in  STD_LOGIC;
           CONV_START  : in  STD_LOGIC;
           SCK         : in  STD_LOGIC;
   -----OUTPUTS-----
           CS_n        : out  STD_LOGIC;
           MOSI        : out  STD_LOGIC
      );

end SPI_MOSI;
```

شکل (۳-۱): توصیف موجودیت کد.

۳-۲- توصیف معماری

ابتدا سیگنال‌ها را قبل از BEGIN معرفی می‌کنیم. سپس تمام پورت‌های ورودی و خروجی به جز کلاک را به صورت رجیستر درآورده و یک سیگنال داخلی از هر کدام تولید می‌کنیم تا بتوانیم با کلاک به صورت سنکرون پیش برویم. این روش به بهینه شدن مدار و سخت‌افزارهای موجود کمک می‌کند. زیرا کوتاه‌ترین مسیر routing را برای ما ایجاد می‌کند. در اینجا مقدار CONV_START_INT به صورت پیش فرض "۰" تعریف می‌کنیم تا با "۱" شدن آن، ارسال داده انجام می‌شود. CS_n_INT به صورت active low تعریف می‌شود. به همین دلیل مقدار اولیه آن "۱" در نظر گرفته شده است. با توجه به دیتاشیت، داده ما ۱۴ بیتی (با مقدار دلخواه) است که به صورت وکتوری تعریف می‌شود و قرار است تک‌تک بیت‌ها به MOSI توسط counter Bit_CNT ارسال شود که تعیین می‌کند کدام بیت به MOSI انتقال پیدا کند تا همه ۱۴ بیت انتقال پیدا کنند (عملیات bit select). در اینجا مقدار اولیه Bit_CNT با مقدار باینری "۱۱۰۱" انتخاب شده است که معادل ۱۳ دسی‌مال است و می‌خواهیم counter را از مقدار ۱۳یکی یکی کم کنیم (کاهنده). چون انتقال دیتا به صورت MSB first و همچنین تک بیتی انجام می‌شود. در دیتاشیت تعریف شده است. داده دلخواه ۱۳ بیتی را توسط Data_In تعریف می‌کنیم. در اینجا همچنین لازم است برای تعریف حالت‌های مختلف ماشین حالت یک TYPE جدید تعریف کرده و تمام وضعیت‌های (مود) مورد نیاز مطابص دیتاشیت و همچنین تاخیرهای لازم جهت رعایت زمانبندی تعریف می‌کنیم. کد به صورت سنکرون نوشته شده است و با هر لبه بالارونده کلاک، تغییر ایجاد شده و تغییرات با لبه بالارونده بعدی کلاک دیده می‌شود. به همین دلیل، برای آن که یک بیت داده را از دست ندهیم، لازم است تاخیراتی را در نظر بگیریم. سپس سیگنال مورد نظر state را از نوع FSM تعریف می‌کنیم.

```
architecture Behavioral of SPI_MOSI is

  ---in/outs---
  signal CONV_START_INT    :   STD_LOGIC  :='0';
  signal CS_n_INT          :   STD_LOGIC  :='1';
  signal MOSI_INT          :   STD_LOGIC  :='0';
  signal Data_In_INT       :   STD_LOGIC_VECTOR (13 DOWNTO 0)  :=  (OTHERS => '0');
  --signal Data_In_INT     :   STD_LOGIC_VECTOR (23 DOWNTO 0)  :=  (OTHERS => '0');

  ---Ccontrol signals---
  signal   Bit_CNT         :  unsigned (3 DOWNTO 0)            := "1101";  ---MSB FIRST----
  --signal Bit_CNT         :  unsigned (4 DOWNTO 0)            := "10111";    ---MSB FIRST----

  ----STATES----
  type  FSM is   (idle, instruction, write_st, delay_instruction, delay_cs);
  signal state             :  FSM                              := idle;
  ----
  constant Data_In         : STD_LOGIC_VECTOR (13 DOWNTO 0) := "00100010000000";
  --constant  Data_In        : STD_LOGIC_VECTOR (23 DOWNTO 0) := "00100010000000000000000001";
```

شکل (۳-۲): توصیف معماری بخش اعلانات.

در اینجا وارد قسمت BEGIN می‌شویم. ابتدا ارجاعات مربوطه به اتصال به سیگنال‌های داخلی تعریف شده انجام می‌شود. یعنی در خارج PROCESS، دو عبارت ارجاع CS_n <= CS_n_INT و MOSI <= MOSI_INT داریم که با PROCESS موازی انجام می‌شود. درنتیجه، تاخیری برای خروجی‌ها نداریم. یک PROCESS داریم که مطابق تعریف دیتاشیت به لبه بالارونده کلاک حساس شده است. با اولین لبه بالارونده کلاک، ورودی‌ها دیده می‌شود که منجر به این می‌شود که مدار به صورت سنکرون عمل کند. ابتدا وروی‌های Data_In و CONV_START را به سیگنال‌های داخلی ارجاع می‌دهیم. سپس با استفاده از دستور CASE WHEN تمامی حالت‌های تعریف شده توسط ماشین حالت و همچنین تغییرات لازم در سیگنال‌های مربوطه و انتقال داده را انجام می‌دهیم. CASE نسبت به state تعریف شده حساس است و باید حالت‌های مختلف بررسی گردد.

```
begin

   ----
   CS_n  <= CS_n_INT;
   MOSI  <= MOSI_INT;
   ----

   Process(CLK_SYS)
   begin
   ---
      if (rising_edge (CLK_SYS)) then

         ----
         Data_IN_INT    <= Data_In;
         CONV_START_INT <= CONV_START;
         ----

         case  State is
         ---
```

شکل (۳-۳): توصیف معماری بخش PROCESS.

ابتدا در حالت بیکاری idle قرار داریم. در این حالت مقدار MOSI_INT = ‘0’ و Bit_CNT نیز مقدار اولیه ۱۳ را دارد و انتقال داده‌ای انجام نمی‌شود. با فعال شدن CONV_START_INT در واقع "۱" شدن آن، به مرحله بعدی delay_instruction می‌رویم و مقدار chip select active low فعال می‌شود. این موضوع را ما در لبه بالارونده بعدی متوجه می‌شویم. در غیر این صورت آنقدر در idle میمانم تا CONV_START_INT فعال شود.

```
when idle   =>
---
MOSI_INT <='0';
Bit_CNT  <="1101";
--Bit_CNT   <="10111";
---

   if (CONV_START_INT='1') then
      state <= delay_instruction;
      CS_n_INT <='0';
   else
      state <= idle;
      CS_n_INT <='1';
   end if;
---
```

شکل (۳-۴): توصیف معماری بخش idle.

حال در حالت delay_instruction هستیم و اینجا فقط صبر می‌کنیم تا اولین بیت داده را بتوانیم انتقال دهیم. چون اتفاقات در لبه بالارونده بعدی قابل مشاهده است و اینکار انجام می‌دهیم تا بیت داده از دست ندهیم. در واقع به دلیل رعایت زمان‌بندی دیتاشیت، اول chip select را صفر می‌کنیم. یک پریود صبر می‌کنیم و بعد کلاک را می‌زنیم تا بیتی را از دست ندهیم. سپس به حالت instruction می‌رویم و داده‌ها منتقل می‌شود. نکته‌ای که اینجا هست Bit_CNT از تایپ unsigned است و ما برای انجام اینکه بتوانیم شماره بیت رجیستر مشخص کنیم چون از نوع integer است پس لازم به تغییر تایپ داریم که با دستور زیر انجام شده است.

```
when delay_instruction  =>
---
   state    <= instruction;
   CS_n_INT <= '0';
   MOSI_INT <= Data_IN_INT (to_integer (Bit_CNT));
   Bit_CNT  <= Bit_CNT  -  1;
   ---
```

شکل (۳-۵): توصیف معماری بخش delay_instruction.

لازم به ذکر است چون در حالت MOSI هستیم پس read نداریم و حتما write داریم. در اینجا write_st از instruction جدا کردیم چون بایت اول، بایت دستورالعمل است. CS_n_INT صفر باید باشد چون داریم داده را منتقل می‌کنیم و برای MOSI هم مستقل از شرط‌ها، تک تک بیت‌های باید روش قرار گیرد. در ابتدا برای اینکه بایت دستورالعمل را دریافت کنیم به کمک دستور شرطی صبر می‌کنیم تا ۸ بیت خوانده شود و معادل

آن است که به بیتت شماره ۶ نرسیده باشیم. اگر بعد از دریافت ۸ بیت دستورالعمل به بیت ۶ رسیدیم به مرحله بعد یعنی write_st می‌رویم و Bit_CNT را روی عدد ۵ می‌گذاریم. چون بیت ۱۴ تا ۶ (بایت دستورالعمل) انتقال داده شده است.

```
when instruction  =>
   ---
   CS_n_INT <= '0';
   MOSI_INT <= Data_IN_INT (to_integer (Bit_CNT));
   ---
   if (Bit_CNT /=6)  then
   --if  (Bit_CNT /=16) then
      state    <= instruction;
      Bit_CNT  <= Bit_CNT  -  1;
   else
      state <= write_st;
      --Bit_CNT   <= "01111";
      Bit_CNT  <= "0101";
   end if;
   ---
```

شکل (۳-۶): توصیف معماری بخش instruction.

در اینجا نیز همچنان CS_n_INT صفر باید باشد و بیت‌ها را تا رسیدن به بیت ۰ انتقال می‌دهیم و از شمارنده یکی یکی کم می‌کنیم. هرگاه به بیت ۰ که اخرین بیت هست رسیدیم، یعنی انتقال داده تمام شد، Bit_CNT را به حالت اولیه ۱۳ بر می‌گردانیم و به مرحله delay_cs می‌رویم.

```
when  write_st =>
---
   CS_n_INT <= '0';
   MOSI_INT <= Data_IN_INT (to_integer (Bit_CNT));
   ---
      if (Bit_CNT /=0)  then
         state    <= write_st;
         Bit_CNT  <= Bit_CNT  -  1;
      else
         state <= delay_cs;
         --Bit_CNT   <= "10111";
         Bit_CNT  <= "1101";
      end if;
      ---
```

شکل (۳-۷): توصیف معماری بخش write_st.

دلیل اینکه اینجا همزمان CS_n_INT "۰" نشده و تاخیری ایجاد کردیم، این است که چون اتفاقات در لبه بالارونده بعدی است که رخ می‌دهد، اگر CS_n_INT فورا "۱" شود، بیت آخر را از دست می‌دهیم. چون آخرین بیت مرحله قبل در اولین بیت این مرحله دیده می‌شود. به همین دلیل CS_n_INT را "۰" نگه می‌داریم. سپس MOSI_INT "۰" شده و داده کامل انتقال پیدا کرده است و در آخر، Bit_CNT را به حالت اولیه ۱۳ بر می‌گردانیم.

```
                when delay_cs  =>
                ---
                   state <= idle;
                   CS_n_INT <= '0';
                   MOSI_INT <= '0';
                   Bit_CNT  <= "1101";
                   --Bit_CNT   <= "10111";
                ---

            end case;
            ---

        end if;
        ----

    end Process;

end Behavioral;
```

شکل (۳-۸): توصیف معماری بخش delay_cs.

سپس دوباره به مرحله idle می‌رویم و منتظر می‌شویم CONV_START "۱" شود و این الگو تکرار خواهد شد.

۳-۳- توصیف تست‌بنچ

سیگنال‌های لازم در اینجا تعریف شده است تا ما بتوانیم با فراهم کردن بستری برای ورودی‌ها و مقداردهی به آن‌ها، به خروجی‌های مورد نظر دست یابیم.

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

ENTITY SPI_MPSI_tb IS
END SPI_MPSI_tb;

ARCHITECTURE behavior OF SPI_MPSI_tb IS
```

شکل (۳-۹): توصیف معماری تست‌بنچ.

پس از ایجاد سیگنال‌ها، لازم است فرکانس کاری مدار را با توجه به دیتاشیت مشخص کنیم. فرکانس کاری ۱۵ گیگاهرتز معادل دوره تناوب ۶۶.۶ نانوثانیه می‌باشد. CLK_SYS_period و SCK_period هم فرکانس هستند و فقط باهم یک اختلاف فاز ۱۸۰ درجه دارند که به دلیل رعایت زمان setup و hold می‌باشد.

```
--Inputs
signal CLK_SYS          : std_logic := '0';
signal CONV_START       : std_logic := '0';
signal SCK              : std_logic := '0';
signal Data_In_INT      : STD_LOGIC_VECTOR (13 DOWNTO 0) :=  (OTHERS => '0');

--Outputs
signal CS_n             : std_logic;
signal MOSI             : std_logic;

--Internal signal
signal SCK_CONV_START   : std_logic := '0';

-- Clock period definitions
constant CLK_SYS_period : time := 66.6 ns;    ---15MHz
constant SCK_period     : time := 66.6 ns;    ---15MHz
```

شکل (۳-۱۰): توصیف معماری تست‌بنچ بخش پریود کلاک.

اینجا عملیات PORT MAP انجام می‌دهیم. سیگنال‌ها را ارجاع می‌دهیم. سپس کلاک‌ها را با دستور PROCESS ایجاد می‌کنیم. SCK_CONV_START_Pro به این دلیل است که بدانیم کلاک چه زمانی شروع می‌شود و بی دلیل کلاک به مدار ندهیم و توان اضافی مصرف نکنیم. با زمان‌بندی دلخواه آن را می‌سازیم.

```
BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: entity work.SPI_MOSI PORT MAP (
          CLK_SYS => CLK_SYS,
          CONV_START => CONV_START,
          SCK => SCK,
          CS_n => CS_n,
          MOSI => MOSI
        );

   -- Clock process definitions
   CLK_SYS_process :process
   begin
      CLK_SYS <= '0';
      wait for CLK_SYS_period/2;
      CLK_SYS <= '1';
      wait for CLK_SYS_period/2;
   end process;


   --- SCK start generator
   SCK_CONV_Start_Pro:  process
   begin
      SCK_CONV_START <= '0','1' after 632.7ns,'0' after 1520ns, '1' after 1740ns, '0' after 2700ns;
   wait;
   end process SCK_CONV_Start_Pro;
```

```
   ---SCK generator
   SCK_Pro: process
   begin
      if (SCK_CONV_START   =  '1')  then
         SCK   <= '0';
         wait for SCK_period/2;
         SCK   <= '1';
         wait for SCK_period/2;
      else
         SCK   <= '0';
         wait until  SCK_CONV_START =  '1';
      end if;

   end process SCK_Pro;


   ---CONV_START generator
   start_Pro:  process
   begin
      CONV_START  <= '0','1' after 490ns,'0' after 530ns, '1' after 1630ns, '0' after 1670 ns;
   wait;
   end process start_Pro;

END;
```

شکل (۳-۱۱): توصیف معماری تست‌بنچ بخش مقداردهی به ورودی‌ها.

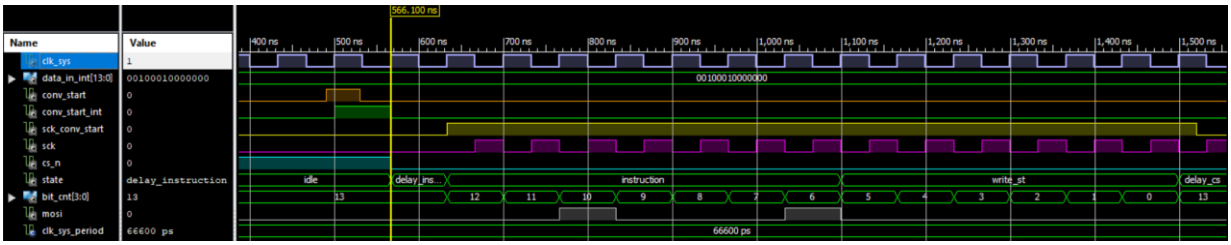
در فصل بعد، به چگونگی کارکرد پروتکل SPI و همچنین خروجی‌های آن می‌پردازیم.

فصل ۴:

شبیه‌سازی و بررسی نتایج

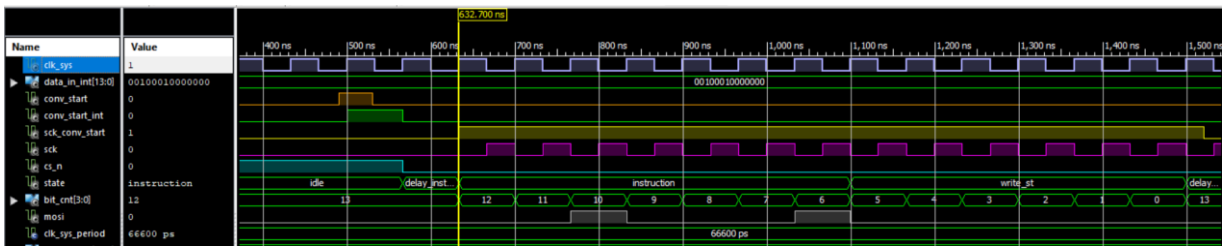
۴-۱- شبیه‌سازی و نتایج خروجی

در اینجا نتایج انتقال داده که به درستی در زمانبندی مربوطه انجام شده است را مشاهده می‌کنیم. پریود کلاک مطابق توضیحات بیان شده و همچین با توجه به دیتاشیت ۶۶.۶ نانوثانیه است. CONV_START در ۴۹۰ نانوثاینه اتفاق می‌افتد. اما کد در لبه بعدی بالارونده کلاک یعنی در ۴۹۹.۵۰۰ نانوثانیه این موضوع را متوجه می‌شود و عملیات رجیستر شدن (CONV_START_INT) در این مرحله انجام می‌شود. ما در حالت idle بیکار منتظر این هستیم که CONV_START_INT "۱" شود. حال که در ۴۹۹.۵۰۰ نانوثانیه CONV_START_INT "۱" شد، به دلیل تاخیرات موجود، ما در لبه کلاک بالارونده بعدی یعنی در ۵۶۶.۱۰۰ متوجه این موضوع می‌شویم و از حالت idle به حالت بعدی delay_instruction می‌رویم.



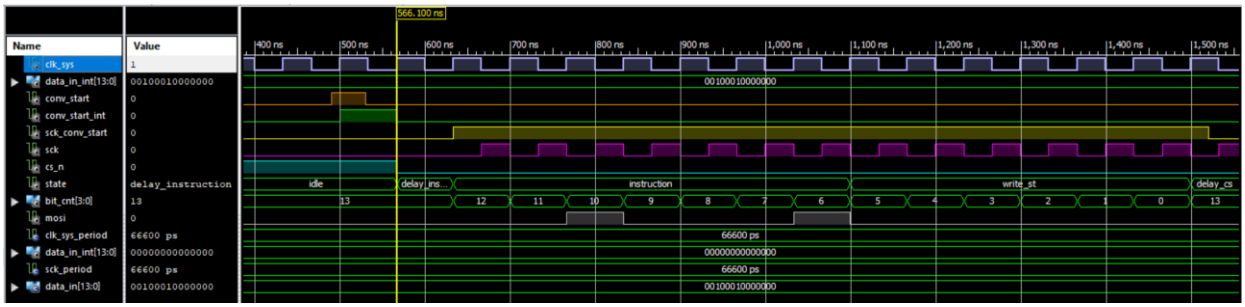
شکل (۴-۱): نمایش خروجی‌ها در حالت idle و delay_instruction.

همچنین cs_n "۰" در ۵۶۶.۱۰۰ می‌شود و یک پریود زمانی طول می‌کشد تا کلاک شروع به خوردن می‌کند تا مصرف توان نیز بهینه شود و بی دلیل کلاک نزنیم. هنگامی که در مرحله delay_instruction هستیم، در لبه بالارونده بعدی اولین بیت داده یعنی "۰" را در MOSI قرار می‌دهیم و یکی از Bit_cnt کم می‌کنیم. یک پریود زمانی صبر می‌کنیم و سپس کلاک در ۶۳۲.۷۰۰ جایی که SCK_CONV_START "۱" شده است شروع می‌شود. SCK_CONV_START یک سیگنال داخلی است که در تست‌بنچ ایجاد کردیم و گفتیم زمان‌هایی که "۱" می‌شود کلاک بخورد و اگز "۰" است صبر کنیم تا "۱" شود تا مصرف توان مدیریت شود. sck نیز همانطور که بیان شد، با clk_sys هم فرکانس هستند و فقط باهم یک اختلاف فاز ۱۸۰ درجه دارند. حال در مرحله instruction هستیم. مقدار داده در نظر گرفته شده "۰۰۱۰۰۰۱۰۰۰۰۰۰۰" است و انتقال MSB FIRST انجام می‌شود.



شکل (۴-۲): نمایش خروجی‌ها در حالت instruction.

باید به این نکته همواره توجه داشته باشیم که باید لبه به لبه صبر کنیم تا تغییرات اعمال شده قابل مشاهده باشد. در لبه بعدی بیت "۰" گرفته و در لبه بعدی بیت سوم که "۱" را می‌گیرد. سپس ۵ تا "۰" و یک "۱" و یک "۰" دریافت می‌کنیم که همگی به درستی در زمان درست دریافت شده‌اند. پس از اینکه بایت اول دریافت شد، به مرحله write_st می‌رسیم. اولین بیت write_st که ۵ است درواقع آخرین بیت مرحله قبلی یعنی instruction است. در write_st بیت‌های باقی مانده را انتقال می‌دهیم و کار به اتمام می‌رسد.



شکل (۴-۳): نمایش خروجی‌ها در حالت write_st و delay_cs.

روند فوق برای داده‌های بعدی و کانال‌های دیگر قابل تعمیم است.