



دانشکده مهندسی برق

پیاده سازی پروتکل ارتباطی SPI_3 Wire در قطعه‌ی AD9177

جواد سرمیلی ۴۰۱۶۱۱۱۷۵

استاد راهنما:

دکتر میرزا کوچکی

بهمن ماه ۱۴۰۱

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

فهرست مطالب

فصل اول: معرفی پروتکل SPI	۵
۱-۱ پروتکل SPI	۶
۱-۲ انواع سیگنال های منطقی SPI	۷
۱-۳ پروتکل SPI چگونه کار می کند؟	۷
۱-۴ انتخاب اسلیو	۸
۱-۵ اسلیوهای متعدد	۸
۱-۶ پین MOSI و MISO	۹
۱-۷ مراحل انتقال داده در پروتکل SPI	۹
۱-۸ پین ها	۱۰
۱-۹ پیکربندی مرجع	۱۱
فصل دوم: پیکربندی ثبات ها	۱۳
۲-۱ مقدمه	۱۴
۲-۲ دستورالعمل	۱۴
۲-۳ خواندن/نوشتن	۱۴
۲-۴ آدرس	۱۴
۲-۵ داده ها	۱۵
۲-۶ زمان بندی	۱۵
۲-۷ چرخه نوشتن	۱۶
۲-۸ چرخه خواندن	۱۶
۲-۹ پیکربندی ثبات ها	۱۷
فصل سوم: کد نویسی پروتکل SPI_3Wire	۲۶
۳-۱ مقدمه	۲۷
۳-۲ بدنه کد	۲۷
فصل چهارم: شبیه سازی پروتکل SPI_3Wire	۳۵

۳۶.....	۴_۱_ چک کردن Snytax
۳۶.....	۴_۲_ نوشتن Test_Bench
۳۹.....	۴_۳_ شبیه سازی بر اساس مقادیر دیتاشیت
۴۲.....	مراجع

فصل اول

معرفی پروتکل SPI

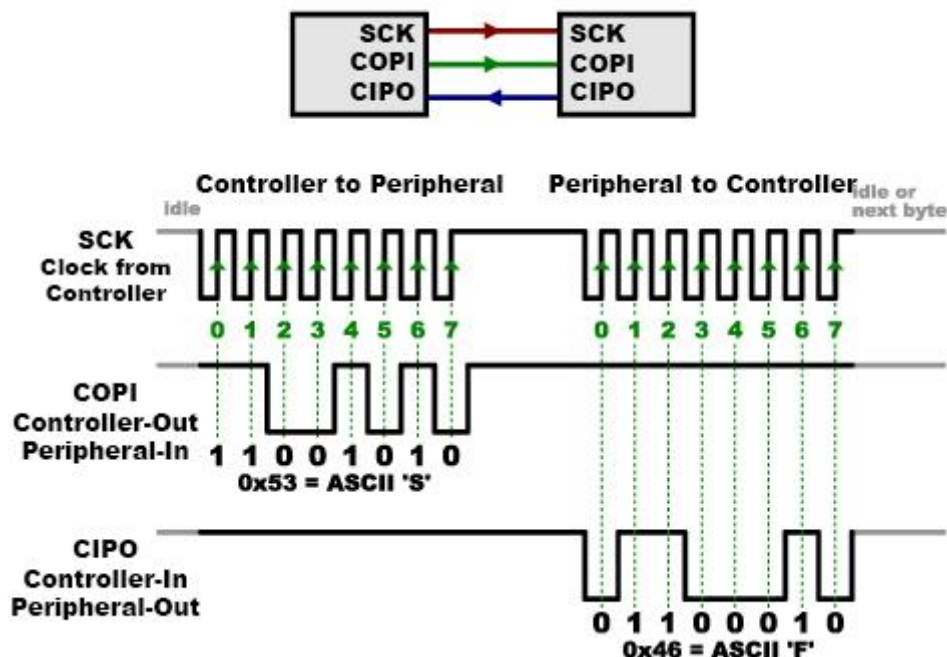
۱-۱_ پروتکل SPI

در ارتباط بین دستگاه های الکترونیکی، لازم است که دو طرف زبان یا منظور یکدیگر را متوجه شوند. در الکترونیک این زبان مشترک بین دستگاه ها پروتکل نامیده می شود. در واقع پروتکل ها، قرار دادهایی هستند که بین دو دستگاه در مورد نوع انتقال دیتا وجود دارند.

یکی از انواع این پروتکل ها، پروتکل SPI است که برای ارتباط بین دو یا چند دستگاه استفاده می شود. در این نوع ارتباط یک دستگاه به عنوان Master و دستگاه های دیگر به عنوان Slave شناخته می شوند.

Master که عمدتاً یک لاجیک منطق پذیر است وظیفه کنترل ارتباط و دیتا یا اطلاعات منتقل شده را دارد در حالی که Slave معمولاً انواع سنسورها، نمایشگر ها و آیسی های حافظه هستند که از مرکزی واحد دستور می گیرند.

در پروتکل SPI چهار خط یا سه خط انتقال بین مرکز واحد و هر یک از مصرف کننده ها وجود دارند که هر یک وظیفه خاصی را بر عهده دارند (شکل ۱-۱).



شکل ۱-۱: یک ارتباط SPI.

۲-۱_ انواع سیگنال های منطقی SPI

سیگنال SCLK : پالس ساعت

سیگنال MOSI : خروجی واحد مرکزی

سیگنال MISO : ورودی مصرف کننده

سیگنال SS : انتخاب مصرف کننده

این پورت ها به نام های زیر نیز شناخته می شوند:

MOSI: SIMO, SDO, DO, DOUT, SI, MTSR

MISO: SOMI, SDI, DI, DIN, SO, MRST

SS: nCS, CS, CSB, CSN, nSS, STE, SYNC

SCLK: SCK, CLK

سیگنال Chip Select کمتر به صورت Active High دیده می شود و در این صورت نشانه گذاری پایه ها آن را مشخص می کند.

۳-۱_ پروتکل SPI چگونه کار می کند؟

سیگنال کلاک، بیت های خروجی از طرف مستر را با بیت های نمونه از طرف اسلیو سنکرون می کند. هر بیت از داده بر اساس یک سیکل زمانی خاص ارسال می شود. بنابراین سرعت انتقال داده توسط فرکانس سیگنال کلاک تعیین می شود. ارتباط SPI همواره توسط مستر آغاز می شود چرا که مستر سیگنال کلاک را می سازد و تعریف می کند.

هر پروتکل ارتباطی که در آن دستگاه ها بر اساس یک سیگنال کلاک کار می کنند، پروتکل سنکرون نامیده می شود. البته متدهای غیر سنکرونی وجود دارند که از سیگنال کلاک بهره نمی گیرند به عنوان مثال در UART، اسلیو و مستر نرخ ارسال و دریافت داده ای از پیش تعیین شده ای دارند.

سیگنال کلاک در پروتکل SPI می تواند از طریق تنظیم مشخصات فاز کلاک تعیین شود. در این جا ۲ المان وجود دارد که تعیین می کنند بیت ها چه زمانی ارسال یا نمونه برداری شوند. پلاریته کلاک توسط مستر مشخص می شود و پس از نمونه برداری از بیت های خروجی که توسط اسلیو انجام می شود ، دو دستگاه با

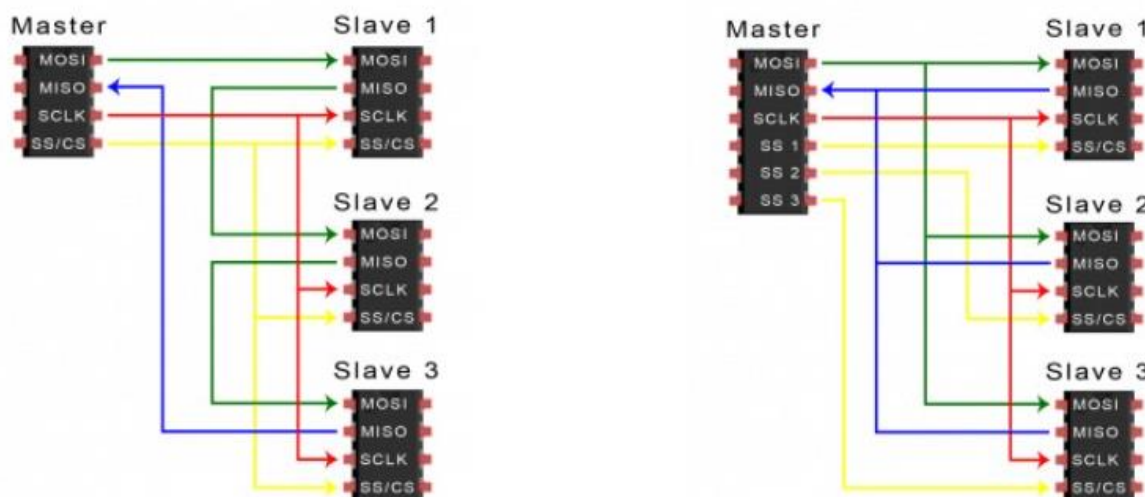
یکدیگر سنکرون خواهند شد. در این شرایط اسلیو بیت‌هایی را در اولین یا دومین لبه پالس کلاک (فارغ از اینکه پالس در حال صعود یا نزول باشد) از طرف مستر دریافت می‌کند.

۱-۴ انتخاب اسلیو

مستر می‌تواند تصمیم بگیرد که با کدام اسلیو ارتباط برقرار کند انتخاب اسلیو توسط مستر از طریق خط CS/SS در سطح ولتاژ پایین صورت می‌گیرد. در حالتی که هیچ داده‌ای ارسال نمی‌شود خط انتخابی اسلیو در سطح ولتاژ بالا نگه داشته می‌شود. چندین پین CS/SS ممکن است روی تراشه مستر به چشم بخورند که امکان اتصال اسلیو ها به صورت موازی را فراهم می‌کنند. اگر تنها یک پین CS/SS موجود باشد، اسلیوها می‌توانند از طریق زنجیره دایسی^۱ به مستر متصل شوند.

۱-۵ اسلیوهای متعدد

SPI می‌تواند به گونه‌ای تنظیم شود تا یک مستر کنترل چند اسلیو را به دست بگیرد. ۲ راه برای اتصال چند اسلیو به مستر وجود دارد. اگر مستر پین‌های متعددی برای اتصال اسلیوها داشته باشد. اسلیوها می‌توانند به صورت موازی به مستر متصل شوند (شکل ۱-۲ راست). اما اگر تنها یک پین برای اتصال اسلیوها موجود باشد اسلیوها به صورت زنجیره دایسی مطابق شکل زیر (شکل ۱-۲ چپ) به مستر متصل خواهند شد.



شکل ۱-۲: نحوه اتصال اسلیوهای متعدد.

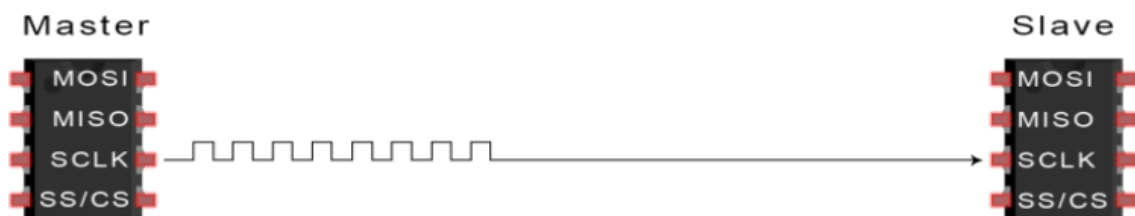
^۱ Daisy- Chaining

۶-۱. پین MOSI و MISO

مستر اطلاعات را بیت به بیت و به صورت سریال از طریق خط MOSI ارسال می‌کند، اسلیو اطلاعاتی که از طرف مستر ارسال می‌شود را از طریق پین MOSI دریافت می‌کند. در داده‌هایی که از مستر به اسلیو ارسال می‌شوند، معمولاً اولین بیت پر ارزش‌ترین بیت است.

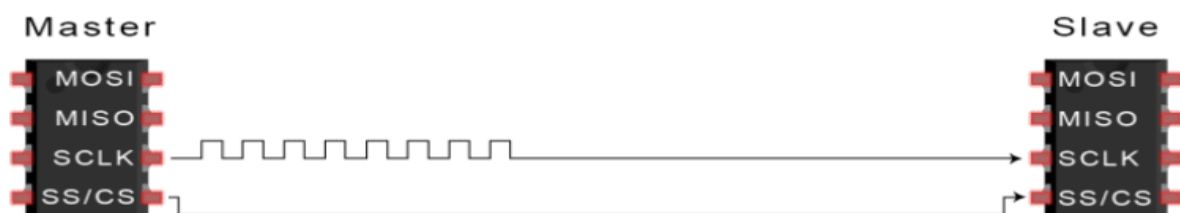
۷-۱. مراحل انتقال داده در پروتکل SPI

۱- مستر سیگنال کلاک را به اسلیو ارسال می‌کند (شکل ۳-۱).



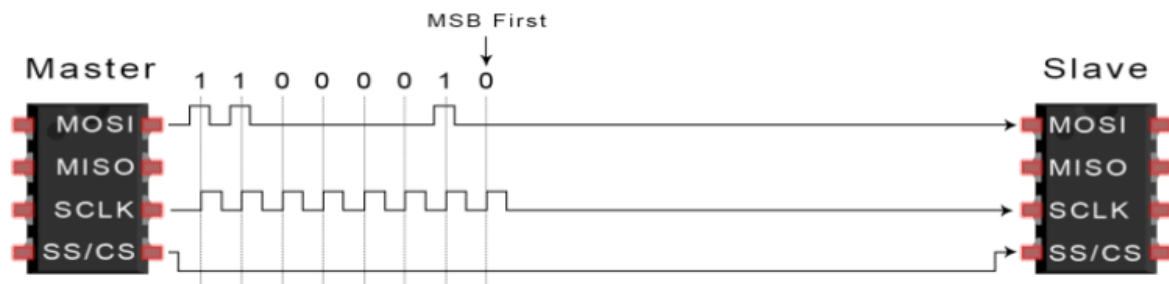
شکل ۳-۱: مرحله اول در پروتکل SPI.

۲- مستر سطح ولتاژ پین SS/CS را پایین می‌آورد و به این وسیله اسلیو را به حالت فعال می‌برد (شکل ۴-۱).



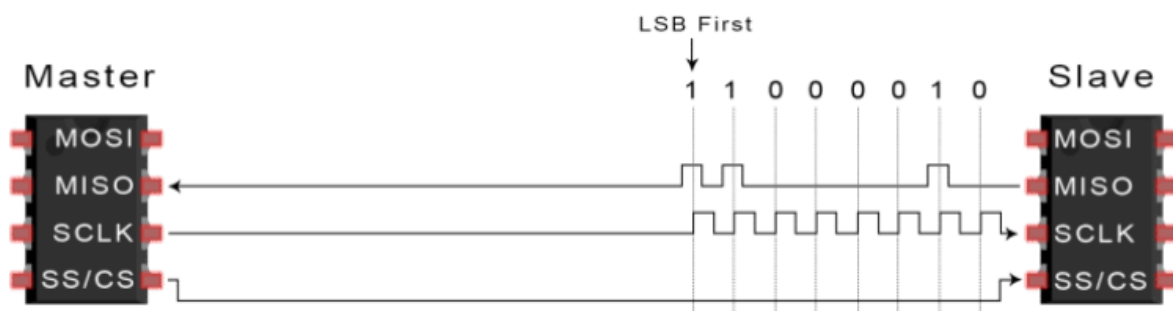
شکل ۴-۱: مرحله دوم در پروتکل SPI.

۳- مستر از طریق خط MOSI شروع به انتقال داده به اسلیو می‌کند (شکل ۵-۱).



شکل ۱-۵: مرحله سوم در پروتکل SPI.

۴- اگر قرار باشد اسلیو به ماستر پاسخ بدهد، این پاسخ از طریق خط MISO ارسال می شود (شکل ۱-۶).



شکل ۱-۶: مرحله چهارم در پروتکل SPI.

۱-۸- پین ها

رابط سریال باید از پایه های SCLK، SDIO، CSB و SDO تشکیل شده باشد.

CSB

CSB انتخاب چیپ است، یک سیگنال Active Low که دستگاه Slave را که Master قصد دارد با آن ارتباط برقرار کند، انتخاب می کند. به طور معمول، یک CSB اختصاصی بین Master و هر Slave وجود دارد. CSB همیشه توسط Master هدایت می شود.

CSB همچنین برای همگام سازی و قالب بندی ارتباطات به دستگاه Slave عمل می کند. هنگامی که CSB غیرفعال می شود، Slave به حالت آماده در انتظار دستور بعدی برمی گردد.

SCLK

SCLK سیگنال ساعت سریال است که دستگاه(های) Slave را با Master همگام می‌کند. به طور معمول، SCLK برای همه دستگاه‌های Slave در گذرگاه سریال به اشتراک گذاشته می‌شود. SCLK همیشه توسط Master هدایت می‌شود.

SDIO

SDIO سیگنال داده دو جهته است. به طور معمول، SDIO برای همه دستگاه‌های Slave در گذرگاه سریال به اشتراک گذاشته می‌شود. SDIO یک پین دو طرفه با امپدانس بالا، هم در Master و هم بر روی Slave است.

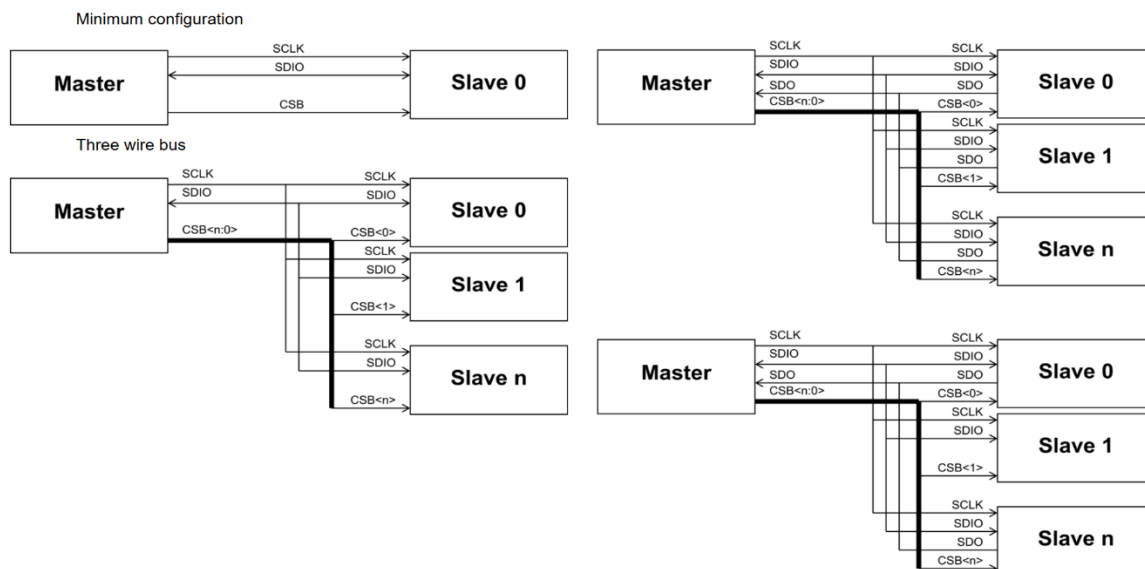
برای سازگاری، همه دستگاه‌ها باید دارای پشتیبانی دو جهته در حالت ۳ سیم باشند. اگر پشتیبانی از حالت ۴ سیم در دسترس باشد، ممکن است پین SDO به عنوان بخشی از پیکربندی رابط توسط کاربر وارد و فعال شود، اما قابلیت دو جهته SDIO همچنان مورد نیاز است، حتی اگر در طول پیکربندی غیرفعال شود. اگر ۴ سیم با پیکربندی SDO فعال شود، SDIO فقط به عنوان ورودی (SDI) عمل می‌کند تا زمانی که SDO غیرفعال شود.

SDO

SDO پین خروجی داده‌های Slave است. به طور معمول، SDO برای همه دستگاه‌های Slave در گذرگاه سریال به اشتراک گذاشته می‌شود. SDO در هر زمان تنها توسط یک دستگاه Slave هدایت می‌شود، در غیر این صورت در امپدانس بالا است.

۹-۱_ پیکربندی مرجع

نمودارهای زیر پیکربندی ارتباط سریال در دستگاه‌های سازگار با این ارتباط را نشان می‌دهند. پیکربندی‌های دیگری نیز امکان‌پذیر است، اما نباید در عملکرد دستگاه‌های پشتیبانی‌شده اختلال ایجاد کند (شکل ۷-۱).



شکل ۱_۷: پیکربندی ارتباط سریال.

فصل دوم

پیکربندی ثبات‌ها

۱-۲_ مقدمه

ارتباط سریال به دو مرحله مجزا از هم تقسیم می‌شود. فاز اول مرحله دستورالعمل است و فاز دوم فاز داده است که یا به دستگاه Slave منتقل می‌شود تا کار کند یا در پاسخ به مرحله دستورالعمل از دستگاه Slave دریافت می‌شود. مرحله دستورالعمل به عنوان یک کلمه ۱۶ بیتی در نظر گرفته می‌شود و بسته به پیکربندی دستگاه ابتدا MSB یا LSB جابجا می‌شود. در حالی که MSB First پیش فرض است.

CSB ممکن است صفر نگه داشته شود و چندین بایت داده ممکن است در طول فاز داده جابجا شوند. آدرس‌های ترتیبی ممکن است به ترتیب صعودی یا نزولی بر اساس نحوه تنظیم ثبات‌های پیکربندی ارسال شوند. پیش فرض آدرس دهی آدرس‌های متوالی نزولی است. با این تکنیک ممکن است یک یا چند بایت بدون نیاز به ارائه آدرس برای هر یک نوشته یا خوانده شود.

۲-۲_ دستورالعمل

بلافاصله به دنبال لبه پایین رونده CSB که یک چرخه SPI را آغاز می‌کند، مرحله دستورالعمل است. اگر دستور نوشتن داده‌ها در یک ثبات هدف باشد، بایت‌های داده به آدرس هدف هدایت می‌شوند که با آدرس مشخص شده در مرحله دستورالعمل شروع می‌شود. اگر دستور خواندن داده‌ها از دستگاه هدف باشد، آدرس مشخص شده اولین آدرسی است که دستگاه Slave در تکمیل مرحله دستورالعمل به آن پاسخ می‌دهد.

دستورالعمل همیشه شامل ۱۶ بیت است و می‌تواند به طور مستقیم آدرس‌ها را تا 0x7FFF را نشان دهد. در صورت تمایل می‌توان از یک ثبات صفحه‌بندی برای تغییر آدرس اشاره شده در دستورالعمل استفاده کرد یا به عنوان شاخصی برای پخش همزمان به چندین دستگاه به دلخواه یا مورد نیاز برای یک برنامه خاص استفاده کرد. پیاده سازی هر کدام اختیاری است و ممکن است با مشخصات محصول تعیین شود.

۳-۲_ خواندن/نوشتن

مهم ترین بیت مرحله دستورالعمل، بیت نشانگر خواندن و نوشتن است. اگر این بیت تنظیم شود، یک دستورالعمل خواندن را نشان می‌دهد. اگر این بیت صفر باشد، یک دستورالعمل نوشتن را نشان می‌دهد.

۴-۲_ آدرس

۱۶ بیت به جز MSB مرحله دستورالعمل، دسترسی مستقیم به رجیسترهای 0x7FFF را فراهم می‌کند که هر کدام ۸ بیت داده را نشان می‌دهند.

۲_۵_ داده‌ها

داده‌ها همیشه در ۸ بیت سازماندهی می‌شوند و از مرحله دستورالعمل پیروی می‌کنند. اگر یک رجیستر به بیش از ۸ بیت نیاز داشته باشد، بایت‌های متوالی در حافظه باید به گونه‌ای استفاده شود که آدرس پایین‌تر نشان دهنده بایت با اهمیت کمتر باشد. به عنوان مثال اگر ۱۶ بیت باید ذخیره شود، کم ارزش ترین بایت باید در 0x0010 مهم ترین بایت در 0x0011 ذخیره شود.

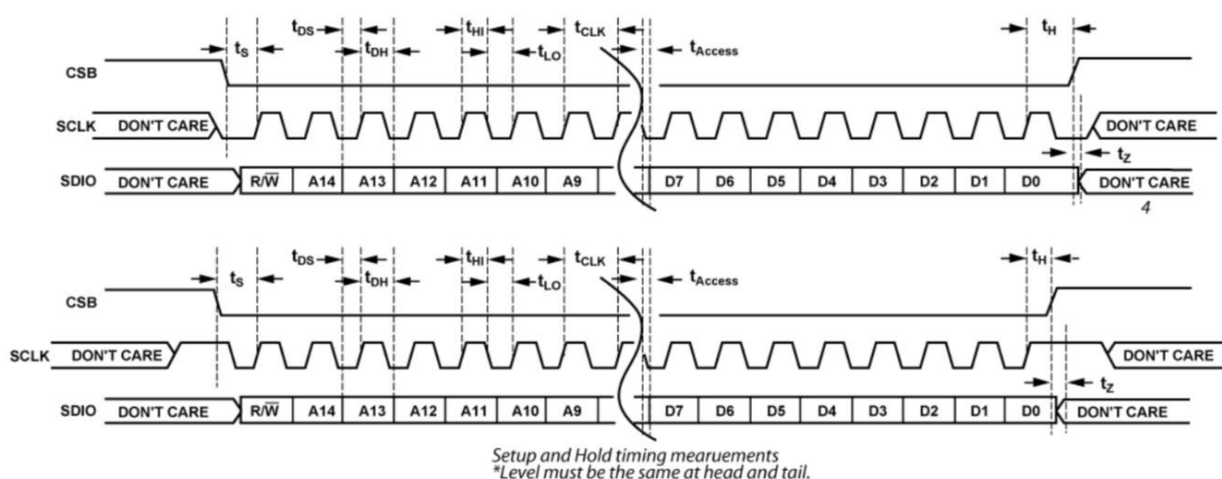
هنگامی که یک ثبات به بیش از ۸ بیت نیاز دارد و ۲ بایت یا بیشتر را در بر می‌گیرد و در عین حال آخرین بایت را به طور کامل اشغال نمی‌کند، توصیه می‌شود که داده‌ها LSB ارسال شوند (شکل ۲_۱).

	MSB							LSB
Position ->	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Address n	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Address n+1	n/a	n/a	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8

شکل ۲_۱: تعداد داده‌ها به طور کامل یک یا چند بایت را پوشش نمی‌دهند.

۲_۶_ زمان بندی

شکل ۲_۲ نمودار زمان بندی قابل قبولی را نشان می‌دهند. تنها تفاوت بین این دو، حالت اولیه ساعت در رابطه با خط CSB است.



شکل ۲_۲: زمان بندی قابل قبول در یک ارتباط سریال.

زمان تنظیم و نگهداری داده‌ها در سیگنال بالا تعریف شده است. سایر پارامترهای مهم زمان بندی نیز در شکل ۲_۳ نشان داده شده است.

Spec Name	Meaning
t_{DS}	Setup time between data and rising edge of SCLK
t_{DH}	Hold time between data and rising edge of SCLK
t_{CLK}	Period of the clock
t_s	Setup time between CSB and SCLK
t_H	Hold time between CSB and SCLK
t_{HI}	Minimum period that SCLK should be in a logic high state
t_{LO}	Minimum period that SCLK should be in a logic low state
t_z	Maximum time delay between CSB deactivation and SDIO or SDO bus return to high impedance.
t_{Access}	Maximum time delay between falling edge of SCLK and output data valid for a read operation.

شکل ۳-۲: سایر پارامترهای مهم زمان بندی.

۲-۷_ چرخه نوشتن

بیت‌های دستورالعمل و به دنبال آن داده‌ها به صورت سریال از طریق پین SDIO در لبه‌های افزایش یافته SCLK در دستگاه Slave نوشته می‌شوند. بسته به تنظیمات، داده‌ها را می‌توان ابتدا MSB یا LSB منتقل کرد.

به طور پیش فرض، دستگاه در حالت اول MSB و آدرس دهی نزولی پیکربندی شده است. هر دو ترتیب بیت و ترتیب آدرس دهی متوالی را می‌توان از طریق پیکربندی ثبات مناسب تغییر داد.

۲-۸_ چرخه خواندن

عملیات خواندن می‌تواند برای استفاده از ۳ سیم یا ۴ سیم (در صورت پشتیبانی) همانطور که در بالا توضیح داده شد پیکربندی شود و می‌تواند ابتدا با MSB یا ابتدا LSB فرمت شود. هدر دستورالعمل ابتدا روی دستگاه MSB یا LSB نوشته می‌شود (بسته به حالت). عملیات چهار سیم اختیاری ممکن است با تنظیم ثبات SDO Enable پیکربندی شود. در حالت ۳ سیمی، پین SDIO پس از دریافت هدر دستورالعمل با درخواست بازخوانی، به یک پایه خروجی تبدیل می‌شود. در حالت ۴ سیمی، درایورهای خروجی SDO بعد از آخرین لبه بالارونده SCLK چرخه دستورالعمل فعال می‌شوند. و تا پایان چرخه خواندن فعال می‌مانند. هنگامی که CSB قطع می‌شود، SDO تا عملیات خواندن بعدی به امپدانس بالا باز می‌گردد.

۹-۲_ پیکربندی ثبات‌ها

پیکربندی ثبات 0x0000

ثبات 0x0000 باید رجیستر پیکربندی رابط سریال باشد و به صورت یک ثبات ۸ بیتی به صورت آینه ای اجرا می شود. این مورد تضمین می کند که صرف نظر از اینکه داده‌ها در کدام سمت جابجا می‌شوند، در صورت از بین رفتن همگام‌سازی دستگاه، بتوان دستگاه را برنامه‌ریزی کرد. بنابراین هنگام نوشتن به این آدرس، همیشه لازم است که از هر دو طرف نوشته شود تا هرگونه ابهامی در پیکربندی این ثبات برطرف شود.

بیت ۷ و ۰

بیت ۷ و بیت ۰ تنظیم مجدد تراشه را اجرا می‌کنند. تنظیم این بیت یک تابع بازنشانی را اجرا می‌کند که در غیر این صورت معادل یک تنظیم مجدد سخت است، با این استثنا که بیت های 0x0000 (این ثبات) و ماشین حالت SPI تحت تأثیر قرار نمی‌گیرند. علاوه بر این، سایر رجیسترهایی که از قبل روی هارد ریست بارگذاری نشده اند، تحت تأثیر قرار نمی‌گیرند.

بیت ۶ و ۱

هنگامی که این بیت تنظیم می شود، LSB ابتدا برای همه عملیات جابجا می‌شود. هنگامی که این بیت صفر است، MSB ابتدا برای همه عملیات جابجا می‌شود. پیش فرض این بیت صفر است و در نتیجه اولین عملیات MSB انجام می‌شود. با توجه به مرحله دستورالعمل، کل ۱۶ بیت به صورت کامل معکوس می‌شوند. اگر این بیت تنظیم شود، ابتدا LSB مرحله دستورالعمل ارسال می‌شود. اگر این بیت صفر باشد، بیت کنترلی R/W ابتدا ارسال می شود زیرا در موقعیت MSB دستورالعمل قرار دارد.

بیت ۵ و ۲

بیت ۵ و ۲ بیت کنترل افزایش آدرس است. وقتی این بیت صفر می‌شود، آدرس‌های متوالی نزولی صعود می‌کنند. پیش فرض این بیت صفر است و در نتیجه آدرس‌ها در حال نزول هستند.

بیت ۴ و ۳

بیت ۴ و ۳ بیت SDO Active است. دستگاه‌هایی که در حالت ۴ سیم کار می‌کنند وقتی این بیت تنظیم شود SDO را فعال می‌کنند و SDIO به پین فقط ورودی تبدیل می‌شود. اگر این بیت صفر باشد، SDO غیر فعال است و تمام عملیات ورودی و خروجی از طریق SDIO انجام می‌شود. اگر دستگاه فقط یک دستگاه ۳ سیم است، تنظیم این بیت تاثیری در عملکرد ندارد. پیش فرض برای این بیت صفر است که منجر به عملکرد ۳ سیم می‌شود.

در شکل ۲_۴ عملکرد بیت های ثبات به طور خلاصه بیان شده است.

Register 0x0000 Details

	Bit Name	Effect	Default
Bit 7 & 0	Soft Reset	Setting this bit initiates a reset equivalent to a hard reset with the exception that the bits of 0x0000 (this register) and the SPI state machine are unaffected. This bit is auto-clearing after the soft reset is complete.	Clear
Bit 6 & 1	LSB First	When set causes input and output data to be oriented as LSB first. If this bit is clear, data is oriented as MSB first.	Clear – MSB first
Bit 5 & 2	Address Ascension	When set causes Address Ascension address mode to be enabled. When clear, addresses descend.	Clear – Addresses Descending
Bit 4 & 3	SDO Active	When set causes SDO to become active. When clear, the SDO pin remains in high impedance and all read data is routed to the SDIO pin.	Clear – SDIO is used for both input and output.

شکل ۲_۴: ثبات 0x0000.

پیکربندی ثبات 0x0001

ثبات 0x0001 تنظیمات اضافی، اما غیر بحرانی را برای اینترفیس فراهم کند.

بیت ۷

بیت ۷ باید بیت کنترلی تک دستورالعمل باشد. هنگامی که این بیت یک می شود، جریان داده غیرفعال می شود و تنها یک عملیات خواندن یا نوشتن بدون توجه به وضعیت خط CSB انجام می شود. وقتی این بیت صفر باشد، پخش جریانی فعال می شود. اگر این بیت یک شود و CSB ثابت بماند، ماشین حالت پس از بایت داده بازنشانی می شود، گویی CSB قطع شده است و منتظر دستور بعدی است. این امر باعث می شود که هر بایت داده با یک دستورالعمل جدید قبل از آن قرار گیرد.

بیت ۶

رزرو شده است

بیت ۵

در دستگاه‌هایی که از بافر Master/Slave استفاده می کنند، تنظیم بیت ۵ امکان بازخوانی خروجی های فلیپ فلاپ اصلی را به جای خروجی های Slave فراهم می کند. صفر کردن این بیت دسترسی به خروجی های Slave را فراهم می کند. پیش فرض این بیت صفر است، به طور پیش فرض به خروجی های Slave می رسد. دستگاه هایی که از بافر M/S استفاده نمی کنند، این بیت تاثیری نخواهد داشت. این بیت در دستگاه هایی که از بافر M/S استفاده می کنند اختیاری نیست.

بیت ۴

هنگامی که از یک میکروکنترلر کند یا میزبان دیگری برای اتصال به دستگاه‌های Slave استفاده می‌شود، ممکن است لازم باشد زمان بیشتری برای دستگاه اصلی برای غیرفعال کردن درایورهای خروجی آن در نظر گرفته شود. این بیت برای کمک به این فرآیند ارائه شده است. علاوه بر تنظیم این بیت، یک مقاومت پول‌آپ ضعیف خارجی نیز برای کمک به این فرآیند مورد نیاز است.

این بیت اختیاری است و برای دستگاه‌هایی ارائه می‌شود که قرار است با میکروکنترلرهایی ارتباط برقرار کنند که باید به صورت دستی بین فرآیندهای نوشتن و خواندن تغییر کنند. در هنگام اتصال به FPGA یا ASIC در نظر گرفته نمی‌شود.

بیت ۳

رزرو شده است

بیت ۲ و ۱

بیت ۲ و ۱ به عنوان بازنشانی نرم عمل می‌کند که ممکن است توسط تعریف محصول به عنوان بخشی از یک تابع تنظیم مجدد سطحی تعریف شود. این تابع اختیاری است. اگر استفاده نشود، این بیت‌ها باید توسط ماشین حالت نادیده گرفته شود یا باید همان اثر بیت‌های ۰ و ۷ از ثبات 0x0000 را داشته باشد.

در شکل ۲-۵ عملکرد بیت‌های ثبات 0x0001 به طور خلاصه بیان شده است.

Register 0x0001 Details

	Bit Name	Effect	Default
Bit 7	Single Instruction	When set disables streaming regardless of the state of CSB. When clear, streaming is enabled.	Clear – Streaming enabled.
Bit 6	Reserved		
Bit 5	Master/Slave Readback Control	When set allows readback from master/buffer flip-flops on devices/registers using MS buffering. When clear allows readback from slave/active flip-flops. For devices/registers not using MS buffering this bit has no effect.	Clear – Slave readback enabled
Bit 4	Slow Interface Control	When set, the slave device allows for more time changing between input and output.	Clear – Normal operation of SDIO
Bit 3	Reserved		Clear
Bit 2	Soft Reset 1	Setting this bit initiates a chip defined reset. This bit is auto-clearing after the soft reset is complete.	Clear
Bit 1	Soft Reset 0	Setting this bit initiates a chip defined reset. This bit is auto-clearing after the soft reset is complete.	Clear
Bit 0	Reserved		Clear

شکل ۵_۲: ثبات 0x0001.

پیکربندی ثبات 0x0002

هدف این رجیستر ارائه یک رابط استاندارد برای قرار دادن همه دستگاه‌ها در حالت‌های عملیاتی شناخته شده است. الزامی نیست که دستگاه‌ها از همه حالت‌های عملیاتی پشتیبانی کنند، اما مواردی که پشتیبانی می‌شوند باید با مواردی که در اینجا تعریف شده است مطابقت داشته باشند.

بیت‌های این ثبات به ۳ گروه تقسیم می‌شوند.

گروه اول:

بیت‌های ۰ و ۱ چهار حالت عملکرد عادی را تعریف می‌کنند.

- حالت ۰ (۰۰) عملکرد معمولی تراشه است و با عملکرد کامل سازگار است.
- حالت ۱ (۰۱) عملکرد عادی با کاهش توان و عملکرد مربوطه است.
- حالت ۲ (۱۰) حالت آماده به کار است. در این حالت تراشه در حالت کم مصرف و غیرعملیاتی است اما در کمترین زمان به حالت کامل باز می‌گردد. دستگاه‌هایی که حالت ۲ را اجرا نمی‌کنند، اگر دستور کار در حالت ۲ داده شود، باید به حالت ۳ برگردند.
- حالت ۳ (۱۱) حالت خواب است که با عدم فعالیت تراشه به جز درگاه SPI مشخص می‌شود.

گروه دوم:

بیت های ۲ تا ۴ حالت عملکرد بالقوه وابسته به دستگاه را تعریف می‌کنند. این حالت ها ممکن است برای تقویت حالت عملیاتی ۰ برای بهینه سازی عملکرد برای یک پیکربندی معین استفاده شوند. این بیت ها اختیاری هستند.

گروه سوم:

بیت های ۴ تا ۷ بیت‌های وضعیت وابسته به دستگاه هستند. این بیت‌ها اختیاری هستند و ممکن است توسط محصول مشخص شوند تا وضعیت عملکرد دستگاه خاص را نه محدود به بلکه به طور بالقوه شامل شرایط خطا، وضعیت قفل PLL، وضعیت بازنشانی یا هر شرایط عملیاتی دیگری که محصول مورد نیاز است، نشان دهد. حالت بالای فعال برای این بیت ها نشان دهنده عملکرد مناسب است. به این ترتیب، بیت‌های استفاده نشده باید یک تنظیم شوند. حالت پایین نشان دهنده عملکرد نامناسب است. علاوه بر این، این بیت‌های وضعیت، می‌توانند همراه با وقفه دستگاه استفاده شوند و با استفاده از این ثبات، Master می‌تواند به سرعت تعیین کند که شرایط وقفه یا خطا چیست.

پیکربندی ثبات 0x0003

این یک رجیستر فقط خواندنی است و برای همه محصولات چه از Chip ID استفاده شده باشد چه نشده باشد، لازم است. این ثبات به طور منحصر به فرد نوع محصول را تعریف می‌کند و به شناسه های تراشه گره خورده است. لیست انواع دستگاه های تعریف شده در حال حاضر در شکل زیر نشان داده شده است.

Type	Code
Not Assigned	00 & FF
RF Products	01
IF Products	02
High Speed ADCs	03
High Speed DACs	04
Clocks	05
PLLs	06

Type	Code
Precision ADCs	07
Precision DACs	08
Transceiver Products	09
	0A
	0B
	0C
	0D

شکل ۲-۶: ثبات 0x0003 برای تشخیص نوع محصول.

پیکربندی ثبات‌های 0x0004 و 0x0005

شناسه محصول یک شناسه منحصر به فرد است که به هر محصول و نوع محصول اختصاص داده می‌شود. شناسه محصول ۲ بایت است. شناسه محصول 0x0000 و 0xFFFF نامعتبر است. شناسه‌های محصول ممکن

است در انواع تراشه‌های مختلف کپی شده باشند. این ثبات اختیاری است و ممکن است در همه دستگاه‌ها وجود نداشته باشد.

پیکربندی ثبات 0x0006

درجه تراشه از دو نیبل تشکیل شده است. نیبل بالایی برای نشان دادن تغییرات محصول مانند درجه سرعت یا درجه خطی استفاده شده. نیبل پایینی باید برای نشان دادن تغییرات ماسک یا اصلاح استفاده می‌شود. یک یا هر دو نیبل اختیاری است و ممکن است در همه محصولات استفاده نشود.

پیکربندی ثبات 0x0007

رزرو شده است

پیکربندی ثبات‌های 0x0008 و 0x0009

این ثبات به عنوان یک اشاره گر افست استفاده می‌شود که برای تغییر قسمت آدرس دستورالعمل SPI استفاده می‌شود و ممکن است به روش‌های مختلفی پیاده سازی شود. ممکن است به عنوان یک افست مستقیم اضافه شده به بخش آدرس دستورالعمل SPI استفاده شود. همچنین ممکن است به عنوان یک صفحه افست برای انتخاب نقشه‌های حافظه جایگزین یا افزایش فضای حافظه فراتر از 0x7FFF با دسترسی مستقیم استفاده شود. روش سوم استفاده از این ۸ بیت به عنوان انتخاب دستگاه است. در این حالت، یک یا چند دستگاه نوشتن را به طور همزمان می‌پذیرند. این حالت زمانی مفید است که دستگاه دارای نسخه‌های یکسانی از سخت‌افزار (ADC، DAC، ساعت و غیره) باشد که باید همان پیکربندی را دریافت کنند.

پیکربندی ثبات 0x000A

این رجیستر عمدتاً برای اشکال زدایی نرم افزار و اعتبار سنجی ارتباط دو طرفه در محصول نهایی استفاده می‌شود. این رجیستر یک مکان مناسب را فراهم می‌کند که می‌تواند برای آزمایش هر دو فرآیند نوشتن و خواندن بدون تأثیر بر پیکربندی یا عملکرد دستگاه استفاده شود.

پیکربندی ثبات 0x000B

این ثبات برای نشان دادن نسخه SPI پیاده سازی شده استفاده می‌شود. نسخه اولیه روی 0x00 تنظیم خواهد شد.

پیکربندی ثبات‌های 0x000C و 0x000D

تعریف شده توسط linux-usb.org.

پیکربندی ثبات 0x000E

رزرو شده است.

پیکربندی ثبات 0x000F

هنگامی که تک بیت شماره ۰ این رجیستر تنظیم می‌شود باعث انتقال داده از رجیستر اصلی به Slave می‌شود. این بیت پس از تکمیل انتقال به صورت خودکار پاک می‌شود.

خلاصه عملکرد رجیسترهای مورد بحث در شکل‌های ۲_۷ و ۲_۸ آمده است.

Addr (Hex)	Parameter Name	Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)	Def. Value (Hex)	Default Notes and comments
Chip Configuration Registers											
0x0000	interface_config_A	Soft Reset	LSB First	Address Ascensi on	SDO active	SDO active	Address Ascensi on	LSB First	Soft Rest	00	Chip interface configuration A
0x0001	Interface_config_B	Single Instruction		Master/ Slave Readback Control	Slow Interface Control		Soft Reset 1	Soft Reset 0		00	Chip interface configuration B
0x0002	device_config	Status bit 3	Status bit 2	Status bit 1	Status bit 0	Optional customer operating modes		0 – normal operation mode 1 – low power normal operation 2 – medium power standby 3 – low power sleep mode		x0	Device configuration registers
0x0003	chip_type					0 & FF – not assigned 1 – RF 2 – IF 3 – High Speed ADC 4 – High Speed DAC 5 – Clock 6 – PLL 7 – Precision ADC 8 – Precision DAC 9 – Transceivers					Defines the device type. Read only.
0x0004	product_ID (low byte)										Unique Product ID managed by each product team and is clearly identified in customer documentation to identify each product. Chip IDs 0x0000 & 0xFFFF are not allowed. Read only.
0x0005	product_ID (high byte)										
0x0006	chip_grade	Product grade				Device Revision					Defines product variations such as speed and performance and device revisions. Read only.
0x0007	reserved										

شکل ۲-۷: نقشه حافظه رجیسترهای موثر در راه اندازی ابتدایی.

Addr (Hex)	Parameter Name	Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)	Def. Value (Hex)	Default Notes and comments
0x0008	page_pointer device index									00	Defines options address offset or device index
0x0009	device index									00	Secondary device index register
0x000A	scratch_pad										Used by SW to test read & write
0x000B	spi_revision									00	Initial release
0x000C	Vendor ID low byte	0x56								0x56	Defined by linux-usb.org
0x000D	Vendor ID high byte	0x04								0x04	
0x000E	reserved										
0x000F	Transfer Register								Master- Slave Transfer bit		

شکل ۸_۲: ادامه نقشه حافظه رجیسترهای موثر در راه اندازی ابتدایی.

این رجیسترها برای راه اندازی اولیه پروتکل SPI می‌باشند.

فصل سوم

کد نویسی پروتکل SPI_3Wire

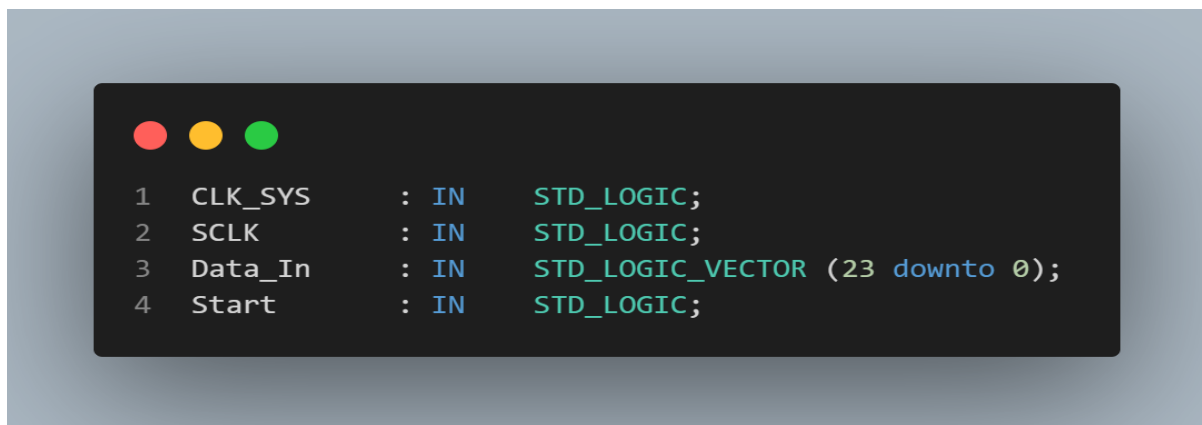
۳_۱_ مقدمه

SDIO پایه ای که به صورت in/out تعریف شده و در مواقعی که عملیاتی اجرا نمی شود به صورت امپدانس بالا می باشد. چالش اصلی این پروژه، همین پایه و همزمانی عملیات خواندن و نوشتن از روی یک پایه است.

۳_۲_ بدنه کد

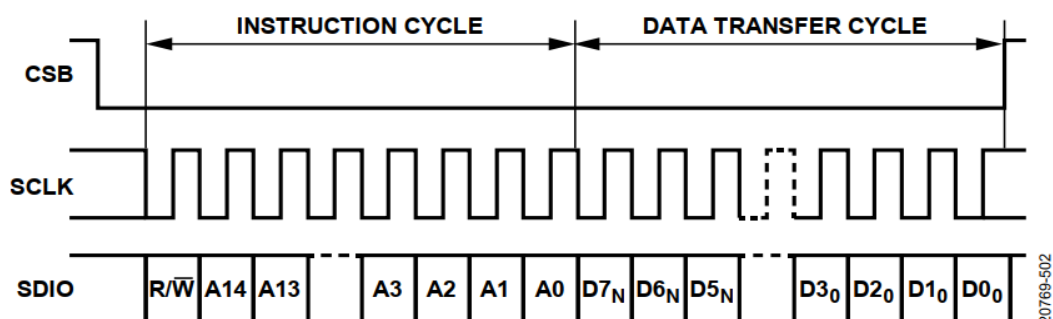
در ابتدای کد کتابخانه های استاندارد زبان VHDL اضافه شده است.

در قسمت entity کد ۴ پورت ورودی داریم CLK_SYS و SCLK به ترتیب کلاک سیستم و کلاک سنکرون ساز برای ماژول که به جهت شبیه سازی به صورت ورودی تعریف شده اند. Start نشان دهنده ی شروع عملیات است (شکل ۳_۱).



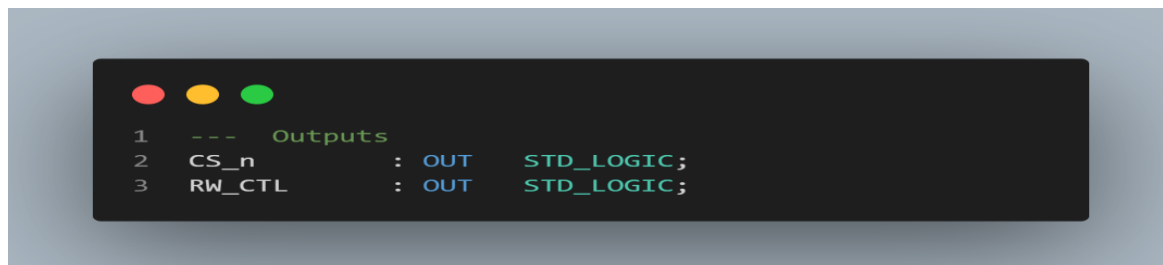
شکل ۳_۱: پورت های ورودی.

Data_In که داده های مد نظر برای عملیات خواندن و نوشتن است که با توجه به دیتاشیت ۲۴ بیت است که شامل ۱۶ بیت دستور العمل و ادرس دهی و ۸ بیت داده است (شکل ۳_۲).



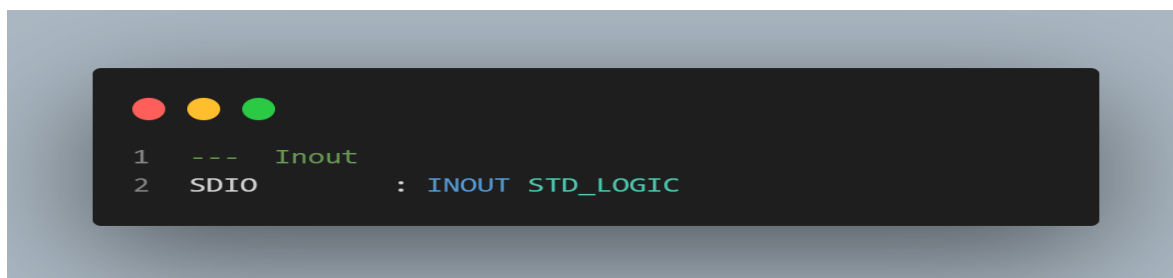
شکل ۳_۲: شکل موج موجود در دیتاشیت.

CS_n پایه که برای انتخاب ایسی و شروع عملیات است که باید بصورت منطق صفر برای شروع عملیات عمل کند. RW_CTL پایه ای که تعیین می کند چه زمانی روی پایه SDIO می خواهیم بخوانیم یا بنویسیم. و به صورت خروجی است که در تست بنچ مقدار دهی می شود (شکل ۳_۳).



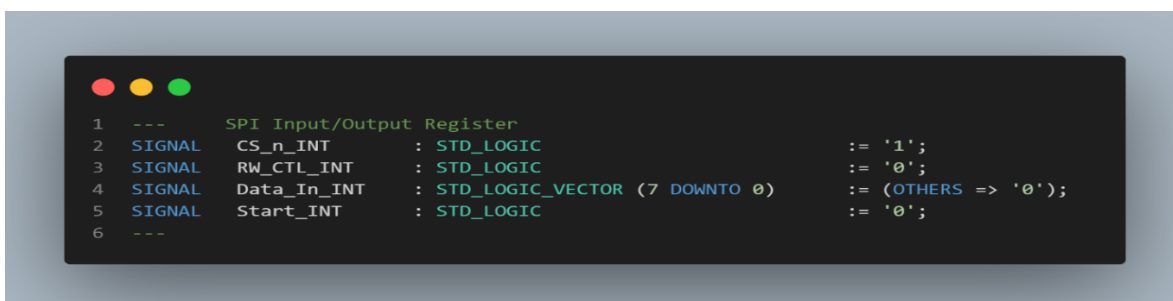
شکل ۳_۳: پورت های خروجی.

SDIO پایه ای که داده ها برای خواندن یا نوشتن روی آن قرار می گیرد و به صورت ورودی خروجی تعریف شده است (شکل ۳_۴).



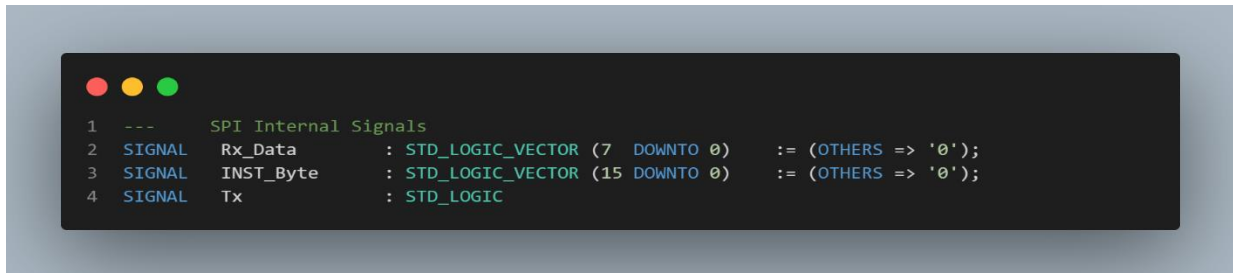
شکل ۳_۴: پورت دو طرفه.

در قسمت Behavioral ابتدا تمام ورودی ها و خروجی ها به جز کلاک ها رجیستر شده اند. علت این کار رعایت استاندارد کد نویسی و جلوگیری از ایجاد مشکلاتی در مرحله های بعدی پیاده سازی می باشد و همچنین به نرم افزار پیاده ساز کمک می کند کوتاه ترین مسیر ممکن را پیدا و پیاده سازی کند که باعث افزایش سرعت مدار می شود (شکل ۳_۵).



شکل ۳_۵: رجیستر کردن ورودی و خروجی.

در قسمت SPI Internal Signals سیگنال‌های داخلی تعریف شده‌اند. سیگنال Rx_Data داده‌های دریافتی روی SDIO را ذخیره می‌کند. سیگنال INST_Bytes ۱۶ بیت دستور العمل را شامل می‌شود. علت تعریف این دو سیگنال برای عدم تغییر ورودی Data_In است. سیگنال Tx هم برای ارسال داده است علت وجود این سیگنال این است که شبیه ساز اجازه استفاده مستقیم از پین SDIO را هم برای خواندن و هم برای نوشتن نمی‌دهد (شکل ۳_۶).



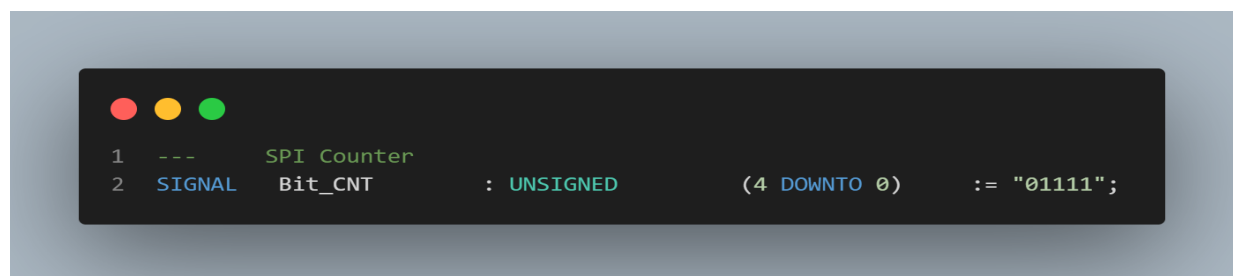
```

1  ---      SPI Internal Signals
2  SIGNAL    Rx_Data      : STD_LOGIC_VECTOR (7 DOWNTO 0)    := (OTHERS => '0');
3  SIGNAL    INST_Byte    : STD_LOGIC_VECTOR (15 DOWNTO 0)    := (OTHERS => '0');
4  SIGNAL    Tx           : STD_LOGIC

```

شکل ۳_۶: سیگنال‌های داخلی.

در قسمت SPI Counter یک شمارنده جهت عملیات شمارش تعداد بیت‌ها تعریف شده است (شکل ۳_۷).



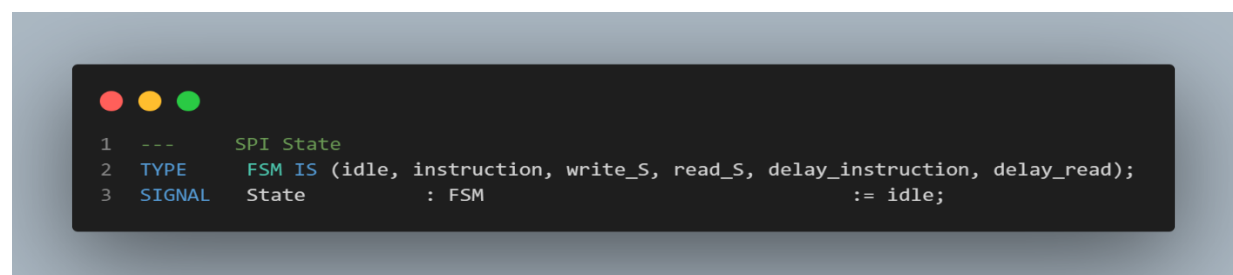
```

1  ---      SPI Counter
2  SIGNAL    Bit_CNT      : UNSIGNED          (4 DOWNTO 0)    := "01111";

```

شکل ۳_۷: شمارنده.

در قسمت SPI State یک تایپ جهت تعریف وضعیت‌های مختلف تعریف شده است که این تایپ در این کد ۶ وضعیتی است (شکل ۳_۸).




```

1  ---      SPI State
2  TYPE      FSM IS (idle, instruction, write_S, read_S, delay_instruction, delay_read);
3  SIGNAL    State       : FSM                := idle;

```

شکل ۳_۸: تعریف تایپ و وضعیت‌های مختلف آن.


یکی از نکات مهم، رجیستر کردن ورودی‌ها و خروجی‌ها بود که انجام شد. برای جلوگیری از یک دوره تاخیر در خروجی، خروجی‌ها را قبل از پراسس مقدار دهی می‌کنیم تا به صورت موازی با پراسس مقدار دهی صورت بگیرد و ورودی‌ها در بدنه پراسس مقدار دهی می‌شود (شکل ۳_۹).



```
1 CS_n    <= CS_n_INT;
2 RW_CTL  <= RW_CTL_INT;
```

شکل ۳-۹: جلوگیری از تاخیر.


چالش اصلی در ارتباط سه سیمه همین یک خط کد است به صورتی که با استفاده از RW_CTL یک مدار ترکیبی تشکیل داد که در صورتی $RW_CTL_INT = 0$ باشد SDIO به صورت خروجی و در غیر این صورت امپدانس بالا باشد. نکته مهم در مورد این خط این است که باید دقیقاً برعکس این خط کد را در تست بنچ داشته باشیم (شکل ۳-۱۰).



```
1 SDIO    <= Tx when RW_CTL_INT = '0' else 'Z';
```

شکل ۳-۱۰: مدار ترکیبی برای SDIO.

در این قسمت صرفاً جدا سازی بیت‌ها به دو دسته داده و دستور العمل صورت گرفته است (شکل ۳-۱۱).



```
1 ---
2 IF (rising_edge(CLK_SYS)) THEN
3   ---
4   Data_In_INT <= Data_In (7  downto 0);
5   INST_Byte  <= Data_In (23 downto 8);
6   Start_INT  <= Start;
7   ---
```

شکل ۳-۱۱: جداسازی بیت‌ها.

پس از آن به ساختار Case در کد می‌رسیم که ما را بین state های مختلف جا به جا می‌کند. علت استفاده از case به جای if های تو در تو پیاده سازی موازی case برخلاف پیاده سازی سریال if است البته باید توجه داشت انعطافی که در شرط‌های if وجود دارد در Case موجود نیست و باید تا حد امکان از شرط‌های طولانی

و ترکیبی خودداری کرد چون باعث درگیر شدن بیهوده سخت افزار برای پیاده سازی آن می شود (شکل ۳-۱۲).

نکته مهم دیگر در Case، اگر از if در آن استفاده می کنیم حتما else داشته باشد و تمام سیگنال های تعریف شده حتما مقدار دهی شوند.

```

1 CASE State IS
2     ---
3     WHEN idle =>

```

شکل ۳-۱۲: ساختار Case.

ابتدا وارد حالت idle می شویم. مقدار دهی های اولیه انجام می شود و در صورت یک بودن سیگنال شروع، وارد حالت delay_instruction می شویم و پایه CS_n فعال می شود در غیر این صورت در حالت idle می مانیم (شکل ۳-۱۳).

```

1 WHEN idle =>
2     ---
3     RW_CTL_INT <= '0';
4     Tx <= 'Z';
5     Bit_CNT <= "01111";
6     Rx_Data <= (others => '0');
7     ---
8     IF (Start_INT = '1') THEN
9         ---
10        State <= delay_instruction;
11        CS_n_INT <= '0';
12        ---
13    ELSE
14        ---
15        State <= idle;
16        CS_n_INT <= '1';
17        ---
18    END IF;

```

شکل ۳-۱۳: حالت idle.

در یک پروسس دستورات در یک لبه بالارونده اجرا می شود و در لبه بالا رونده بعدی اعمال می شود. علت وجود حالت های delay هم همین مورد است در غیر این صورت داده ها از دست می رود. در حالت delay_instruction با توجه به مدار ترکیبی ساخته شده و مقدار شمارنده که با توجه به تعداد بیت

دستورالعمل روی ۱۵ تنظیم شده این مقدار به SDIO منتقل می شود و از مقدار کانتر یک واحد کاسته می شود (شکل ۳-۱۴).

```

1  WHEN delay_instruction =>
2      ---
3      State      <= instruction;
4      RW_CTL_INT <= '0';
5      CS_n_INT   <= '0';
6      Tx         <= Inst_Byte (to_integer(Bit_CNT));
7      Bit_CNT    <= Bit_CNT - 1;
8      Rx_Data    <= (others => '0');
9      ---

```

شکل ۳-۱۴: حالت delay_instruction.

در حالت instruction مشابه قبل عملیات انتقال داده به روی پین SDIO ادامه دارد تا زمانی که مقدار شمارنده صفر شود و این به معنای پایان این حالت است. مجدد مقدار شمارنده ۷ می شود این مقدار به معنای ۸ بیت جهت عملیات خواندن یا نوشتن است. عملیات خواندن یا نوشتن با سنگین ترین بیت دستور العمل یعنی بیت ۱۵ مشخص می شود (شکل ۳-۱۵).

در حالت write_S مشابه حالت های قبلی عملیات اجرا می شود با این تفاوت که با اتمام این حالت دوباره به حالت idle باز می گردیم و مقدار شمارنده به عدد ۱۵ جهت شروع دستور العمل بعدی تنظیم می شود (شکل ۳-۱۶).

در حالت delay_read با توجه به توضیحات قبلی $RW_CTL_INT = 1$ می شود و در این حالت مقادیر روی SDIO از طریق TEST_BENCH شبیه سازی و درون سیگنال Rx_Data ریخته می شود. و پس از انتقال ۸ بیت داده به حالت idle باز می گردیم (شکل ۳-۱۷).

پس از اتمام حالت ها بدنه دستور CASE بسته شده و قسمت معماری کد به پایان می رسد.


```

1  WHEN instruction =>
2      ---
3      RW_CTL_INT  <= '0';
4      CS_n_INT    <= '0';
5      Tx          <= Inst_Byte (to_integer(Bit_CNT));
6      Rx_Data     <= (others => '0');
7      ---
8      IF (Bit_CNT /= Zero) THEN
9          ---
10         State  <= instruction;
11         Bit_CNT <= Bit_CNT - 1;
12         ---
13     ELSE
14         ---
15         Bit_CNT <= "00111";
16         ---
17         if(Inst_byte(15) = '0') then
18             ---
19             State <= write_S;
20             ---
21         else
22             ---
23             State <= delay_read;
24             ---
25         end if;
26         ---
27     end if;

```

شکل ۳-۱۵: حالت instruction.

```

1  WHEN write_S =>
2      ---
3      RW_CTL_INT  <= '0';
4      CS_n_INT    <= '0';
5      Tx          <= Data_In_INT (to_integer(Bit_CNT));
6      Rx_Data     <= (others => '0');
7      ---
8      IF (Bit_CNT /= Zero) THEN
9          ---
10         State  <= write_S;
11         Bit_CNT <= Bit_CNT - 1;
12         ---
13     ELSE
14         ---
15         State  <= idle;
16         Bit_CNT <= "01111";
17         ---
18     END IF;
19     ---

```

شکل ۳-۱۶: حالت write_S.

```

1  WHEN delay_read =>
2      ---
3      State      <= read_S;
4      RW_CTL_INT <= '1';
5      CS_n_INT   <= '0';
6      Tx         <= 'Z';
7      Bit_CNT    <= Bit_CNT - 1;
8      Rx_Data    (to_integer(Bit_CNT)) <= SDIO;
9
10     ---
11     WHEN others =>
12         ---
13         RW_CTL_INT <= '1';
14         CS_n_INT   <= '0';
15         Tx         <= 'Z';
16         Rx_Data    (to_integer(Bit_CNT)) <= SDIO;
17         ---
18         if (Bit_CNT /= Zero) then
19             ---
20             State      <= read_S;
21             Bit_CNT    <= Bit_CNT - 1;
22             ---
23         else
24             ---
25             State      <= idle;
26             Bit_CNT    <= "01111";
27             ---
28         end if;
29     end case;

```

شکل ۳-۱۷: حالت delay_read.

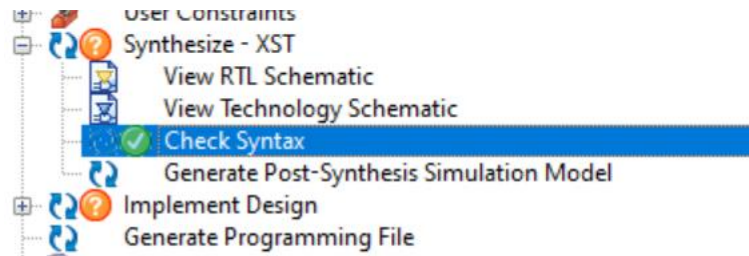
مرحله‌ی بعدی سنتز و شبیه سازی کد است.

فصل چهارم

شبیه سازی پروتکل SPI_3Wire

۴_۱_ چک کردن Syntax

پس از پایان کد نویسی، باید کد نوشته شده ابتدا از لحاظ Syntax چک شود تا دارای خطا نباشد (شکل ۴_۱).



شکل ۴_۱: چک کردن Syntax کد.

پس از بدون خطا بودن کد از لحاظ Syntax، کد را سنتز می‌کنیم تا دارای Error هم نباشد. کد نوشته شده دارای خطایی نیست و فقط دارای سه Warning بخاطر مقادیری که در Test_Bench باید مقدار دهی شود می‌باشد (شکل ۴_۲).

WARNING: HDLCompiler:1127 - "D:\VHDL_PROJECT\test\test\SPI_SDIO\SPI_SDIO.vhd"
Line 50: Assignment to rx_data ignored, since the identifier is never used

WARNING: Xst:647 - Input <SCLK> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.

WARNING: Xst:2041 - Unit SPI_SDIO: 1 internal tristate is replaced by logic (pull-up yes): Tx.

شکل ۴_۲: Warning کد نوشته شده.

۴_۲_ نوشتن Test_Bench

پس از اتمام مراحل فوق، نوشتن تست بنچ را جهت تست عملکرد درست کد آغاز می‌کنیم.

در نرم افزار ISE قسمت‌های تعریف کردن سیگنال‌ها و تولید کلاک اصلی توسط خود نرم افزار ایجاد می‌شود. تنها نکته ی مهم، دقت به مقادیر تخصیص داده شده به سیگنال‌هاست تا این مقادیر با مقادیر اولیه در کد یکسان باشد در غیر این صورت شبیه سازی دچار اشکال در اجرا می‌شود. مقدار کلاک هم به صورت فرضی ۲۵ مگاهرتز قرار داده شده است (شکل ۴_۳). تمامی سیگنال‌ها همان سیگنال‌های موجود در کد هستند به جز سیگنال SCLK_Start که شروع کلاک زدن SCLK را مشخص می‌کند و با استفاده از سیگنال SCLK_Start، سیگنال SCLK تولید شده است (شکل ۴_۴). سیگنال Start هم سیگنالی است که دستور انتقال داده را صادر می‌کند (شکل ۴_۵). در مورد سیگنال Data_In به دو صورت امکان تعریف این

سیگنال وجود دارد یکی به صورت یک ورودی ثابت در کد و دیگری به صورت یک پراسس در بدنه Test_Bench که در این کد، ورودی در بدنه تست بنچ تعریف شده است. (شکل ۴_۶).

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  ---
5  ENTITY SPI_SDIO_tb IS
6  END SPI_SDIO_tb;
7  ---
8  ARCHITECTURE behavior OF SPI_SDIO_tb IS
9
10     ---Inputs
11     signal CLK_SYS    : std_logic := '0';
12     signal SCLK       : std_logic := '0';
13     signal Start      : std_logic := '0';
14     signal Data_In    : std_logic_vector(23 downto 0) := (others => '0');
15
16     ---BiDirs
17     signal SDIO       : std_logic;
18
19     ---Outputs
20     signal CS_n       : std_logic;
21     signal RW_CTL     : std_logic;
22
23     --- Internal Signal
24     signal SCLK_Start : std_logic := '0';
25
26     ---Clock period definitions
27     constant CLK_SYS_period : time := 40 ns;
28     constant SCLK_period   : time := 40 ns;
29
30 BEGIN

```

شکل ۴_۳: تعریف سیگنال‌ها در Test_Bench.

```

1  --- SCLK_Start generator
2  SCLK_Start_Pro : process
3  begin
4      SCLK_Start <= '0','1' after 580ns;
5  wait;
6  end process SCLK_Start_Pro;
7  ---
8
9  --- SCLK Generate
10 SCLK_Pro : process
11 begin
12     ---
13     if(SCLK_Start = '1') then
14         ---
15         SCLK <= '0';
16         wait for SCLK_period/2;
17         SCLK <= '1';
18         wait for SCLK_period/2;
19         ---
20     else
21         ---
22         SCLK <= '0';
23         wait until SCLK_Start = '1';
24         ---
25     end if;
26     ---
27 end process SCLK_Pro;

```

شکل ۴_۴: تولید سیگنال SCLK.

```

1  --- Start generator
2  Start_Pro :process
3  begin
4      ---
5      Start <= '0', '1' after 490ns, '0' after 530ns, '1' after 1490ns, '0' after 1530ns,
6              '0' after 2650ns, '1' after 3890ns, '0' after 3930ns, '1' after 5450ns,
7              '1' after 7090ns, '0' after 7130ns, '1' after 8770ns, '0' after 8810ns,
8              '0' after 10890ns;
9  wait;
10
11 end process start_Pro;

```

شکل ۴_۵: تولید سیگنال Start.

```

1  --- data_In_generator
2  Data_In_Pro:process
3  begin
4      ---
5      Data_In <= "11010101010101010101",
6                "0010101010101010101010" after 1500ns;
7      wait;
8      ---
9  end process Data_In_Pro;

```

شکل ۴_۶: تولید سیگنال Data_In.

مهمترین قسمت در Test_Bench قسمت مرتبط با SDIO است. SDIO هم مانند کد اصلی دارای یک مدار ترکیبی است که به RW_CTL وابسته است. به صورتی که اگر $RW_CTL = 0$ باشد مقادیر را از بدنه کد می گیرد و در صورتی که یک باشد مقادیر را از Test_Bench می گیرد (شکل ۴_۷).

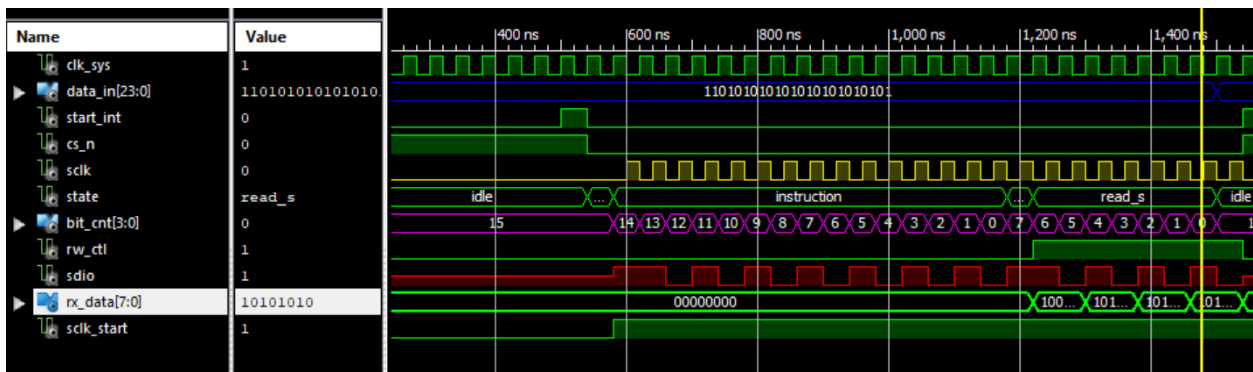
```

1  --- SDIO generator
2  SDIO <= 'Z' when RW_CTL = '0' else '1' , '0' after SCLK_period ,
3          '1' after SCLK_period*2 , '0' after SCLK_period*3,
4          '1' after SCLK_period*4 , '0' after SCLK_period*5,
5          '1' after SCLK_period*6 , '0' after SCLK_period*7;
6  END;
7

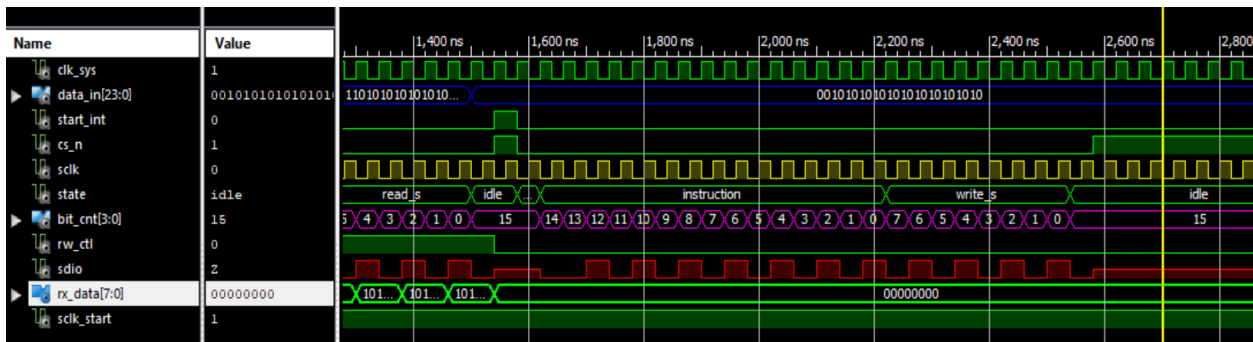
```

شکل ۴_۷: تولید سیگنال SDIO جهت عملیات Read در Test_Bench.

پس از شبیه سازی دو مقدار ورودی ۱۱۰۱۰۱۰۱۰۱۰۱۰۱۰۱۰۱۰۱ (شکل ۴_۸) و ۰۰۱۰۱۰۱۰۱۰۱۰۱۰۱۰۱۰۱۰۱ (شکل ۴_۹) نتایج به صورت زیر خواهد بود.



شکل ۴_۸: شبیه سازی مقدار ۱۱۰۱۰۱۰۱۰۱۰۱۰۱۰۱۰۱۰۱



شکل ۴_۹: شبیه سازی مقدار ۰۰۱۰۱۰۱۰۱۰۱۰۱۰۱۰۱۰۱

۴_۳- شبیه سازی بر اساس مقادیر دیتاشیت

پس از اطمینان از صحت نتایج شبیه سازی، شبیه سازی را بر اساس دیتاشیت آغاز می کنیم. حداکثر فرکانس SCLK بر اساس دیتاشیت برای MSB First Format ۱۵ مگاهرتز است که برای شبیه سازی عدد ۱۰ مگاهرتز معادل ۱۰۰ نانو ثانیه در نظر گرفته شده است (شکل ۴_۱۰).

SERIAL PORT INTERFACE (SPI) WRITE OPERATION				
Maximum SCLK Clock Rate	$f_{SCLK}, 1/t_{SCLK}$		33	MHz
SCLK Clock High	t_{PWH}	SCLK = 33 MHz	8	ns
SCLK Clock Low	t_{PWL}	SCLK = 33 MHz	8	ns
SPI READ OPERATION				
LSB First Data Format				
Maximum SCLK Clock Rate	$f_{SCLK}, 1/t_{SCLK}$		33	MHz
MSB First Data Format				
Maximum SCLK Clock Rate	$f_{SCLK}, 1/t_{SCLK}$		15	MHz

شکل ۴_۱۰: مقادیر حداکثر فرکانس کاری برای R/W.

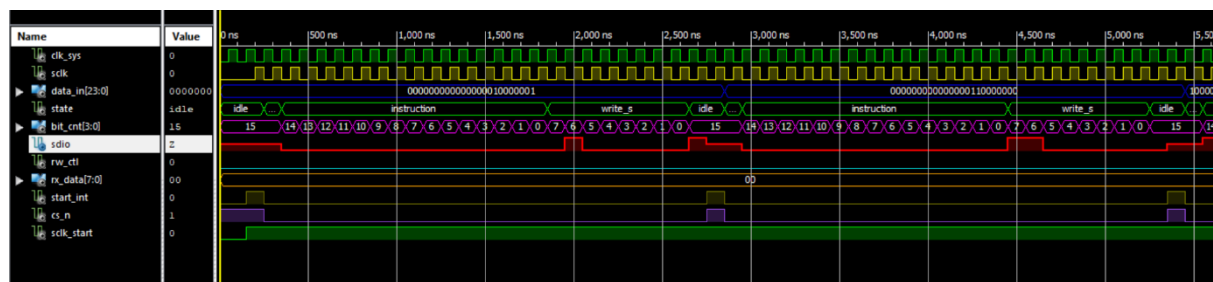
برای شبیه سازی از مقادیر واقعی بر اساس دیتاشیت استفاده شده است. بر اساس دیتاشیت، ۲ رجیستر interface_config_A و Interface_config_B برای پیکربندی ابتدایی ارتباط SPI استفاده می‌شوند.

ابتدا رجیستر A با ادرس 0x0000 برای پیکربندی با مقادیر موجود در شکل ۴-۱۱ استفاده می‌شود. علت این کار پاک شدن مقادیر پیکربندی در سایر ثبات‌ها در ابتدای شروع عملیات است. سپس رجیستر B با ادرس 0x0001 برای پیکربندی با مقادیر موجود در شکل ۴-۱۱ استفاده می‌شود. علت این کار تک سیکله بودن هر سری از دستورات است.

0x0000	interface_config_A	Soft Reset 1	LSB First 0	Address Ascendi 0	SDO active 0	SDO active 0	Address Ascendi 0	LSB First 0	Soft Rest 1	00	Chip interface configuration A
0x0001	Interface_config_B	Single Instruction 1	0	Master/ Slave Readback 0	Slow Interface Control 0	0	Soft Reset 1 0	Soft Reset 0 0	0	00	Chip interface configuration B

شکل ۴-۱۱: مقادیر پیکربندی برای دو ثبات با ادرس‌های 0x0000 و 0x0001.

شکل حاصل از شبیه سازی این دو ثبات به صورت زیر است (شکل ۴-۱۲).



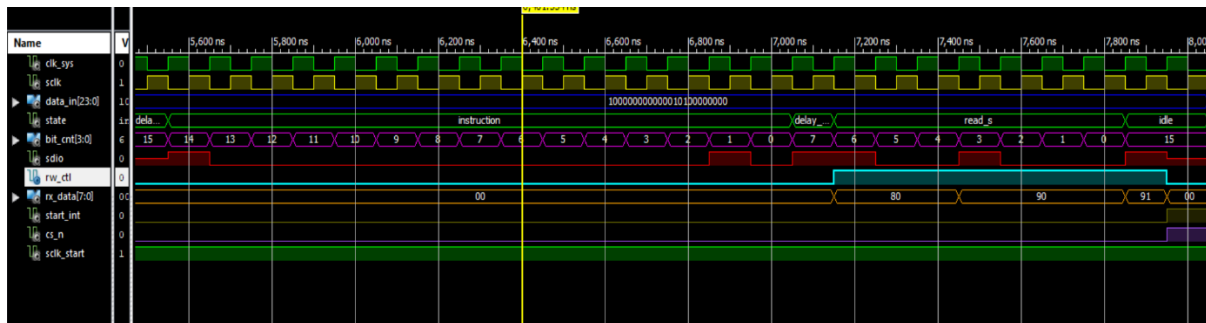
شکل ۴-۱۲: مقادیر شبیه سازی برای دو ثبات با ادرس‌های 0x0000 و 0x0001.

سپس برای تست عملیات خواندن از رجیستر PROD_ID_MSB استفاده شده است. این رجیستر یک عدد ۸ بیتی انحصاری برای هر سری از محصولات را نمایش می‌دهد (شکل ۴-۱۳). با توجه به دیتاشیت عدد منحصر به فرد برای AD9177 معادل 0x91 است.

0x0005	PROD_ID_MSB	[7:0]	PROD_ID_MSB	Product ID MSB. 0x90 AD9081 / AD9082. 0x99 AD9988 / AD9986. 0x92 AD9207 / AD9209. 0x91 AD9177.	0xXX	R
--------	-------------	-------	-------------	--	------	---

شکل ۴-۱۳: مقادیر منحصر به فرد برای ثبات PROD_ID_MSB.

شکل حاصل از شبیه سازی این ثبات به صورت زیر است (شکل ۴-۱۴).



شکل ۴-۱۴: مقادیر شبیه سازی برای ثابت با ادرس 0x0005.

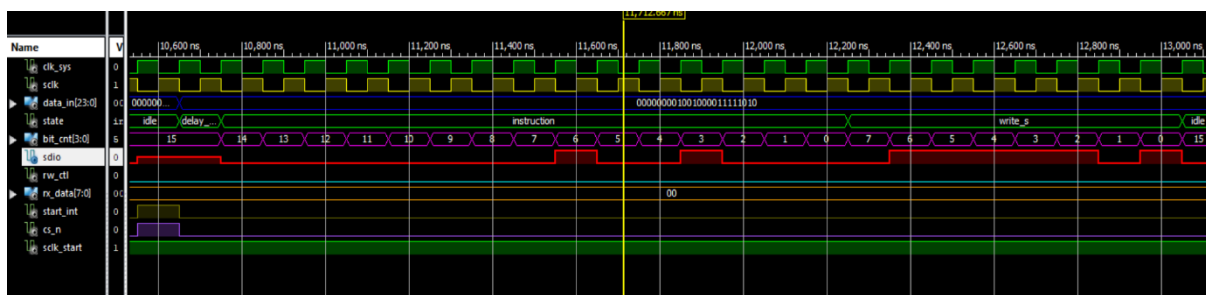
یکی دیگر از رجیسترهای موثر در عملیات ادرس دهی، رجیستر مرتبط با فعال سازی صفحه بندی است. چون نیازی به صفحه بندی نیست جهت شبیه سازی تمام مقادیر صفر در نظر گرفته شده است. ادرس این رجیستر 0x0008 است.

رجیستر DAC_POWERDOWN برای قطع توان برای DAC های موجود در سیستم است. ادرس این رجیستر 0x0090 است (شکل ۴-۱۵). اگر نیاز به قطع توان در دسترس DAC است باید رجیستر روی صفر تنظیم شود.

0x0090	DAC_POWERDOWN	[7:4]	RESERVED	Reserved.	0xF	R
		3	DAC_PD3	Powers down DAC 3.	0x1	R/W
		2	DAC_PD2	Powers down DAC 2.	0x1	R/W
		1	DAC_PD1	Powers down DAC 1.	0x1	R/W
		0	DAC_PD0	Powers down DAC 0.	0x1	R/W

شکل ۴-۱۵: ثابت DAC_POWERDOWN.

شکل حاصل از شبیه سازی این ثابت به صورت زیر است (شکل ۴-۱۶). در این حالت DAC 2 و DAC 0 در حالت خاموش و ۲ مورد دیگر در حالت عادی هستند.



شکل ۴-۱۶: مقادیر شبیه سازی برای ثابت با ادرس 0x0090.

پروتکل SPI_3Wire بر اساس مقادیر موجود در دیتاشیت شبیه سازی شد.

ترتیب و شکل درست سیگنال ها در Test_Bench، در فایل tb.wcfg موجود است.

مراجع

- [1] Tuan, Min-Chun, et al. "3-wire SPI Protocol Chip Design with Application-Specific Integrated Circuit (ASIC) and FPGA Verification." Proceedings of the 3rd World Congress on Electrical Engineering and Computer Systems and Science, Rome, Italy. 2017.
- [2] <https://www.analog.com/en/products/ad9177.html#product-overview>