دانشکده مهندسی برق

پیاده سازی و شبیه سازی پروتکل SPI مربوط به ADAR7251

گزارش پروژه درس VHDL

نام دانشجو

علی احدزاده آقبلاغ

استاد:

دکتر میرزاکوچکی

بهمن ماه ۱۴۰۱


بسم الله الرحمن الرحيم

فهرست مطالب

فهرست اشکال

فهرست جداول

فصل ۱:

مقدمه

۱-۱- مقدمه

در سال ۱۹۸۴ برای اولین بار ایده طراحی یک تراشه خام که بتوان طراحی داخلی آن را عوض کرد توسط راس فریمن مطرح شد. این ایده منجر به تولید تراشه های FPGA و تاسیس شرکت Xilinx شد. در حال حا ضر بیش از ۵۳ در صد از بازار FPGA در د ست شرکت Xilinx ا ست. همچنین تکنولوژی ساخت تراشه های ۱۶ نانومتری تنها در اختیار شرکت Xilinx است. تکنولوژی ساخت با نانومتر پایین تر به معنای تراشه های سریعتر با مصرف کمتر ا ست. در این فصل به بررسی خانواده های مختلف تراشه های FPGA، پروتکل های ارتباطی با FPGA و به خصوص پروتکل SPI پرداخته می شود. در انتهای فصل به معرفی کلی مبدل آنالوگ به دیجیتال ADAR7251 پرداخته می شود.

۱-۲- تراشه های FPGA

تولیدات چیپ های کلاسیک برنامه پذیر Xilinx به دو شاخه CPLD و FPGA تقسیم می شوند. دسته بندی اصلی FPGA ها بر اساس تکنولوژی ساخت انجام می گیرد. به صورت کلی تکنولوژی ساخت FPGA ها به د سته های 45nm، 28nm، 20nm و 16nm تق سیم می شوند. جدول ۱ ن شان دهنده ی د سته بندی FPGA های خانواده Xilinx است.

جدول ۱- دسته بندی FPGA های مختلف شرکت Xilinx

45 nm	28 nm	20 nm	16 nm
SPARTAN 6	VIRTEX 7 KINTEX 7 ARTIX 7 SPARTAN 7	VIRTEX UltraScale KINTEX UltraScale	VIRTEX UltraScale+ KINTEX UltraScale+

خانواده ا سپارتان ها جزو ارزانترین FPGA های شرکت Xilinx ا ست و ارزان ترین I/O ن سبت به قیمت را دارا هستند. خانواده Artix جزء سری های ارزان و اقتصادی Xilinx ا ست که بهترین نسبت توان پردازشی به توان مصرفی و بالاترین نسبت پهنای باند به قیمت را در بین همه خانواده های Xilinx دارد. خانواده KINTEX توان پردازشی بالا، توان مصرفی کم و قیمت مناسبی دارند. در میان تمام خانواده ها، نهایت منابع، سرعت و پهنای باند در سری VIRTEX قرار دارد.

تمامی ابزارهای پیاده سازی مانند میکرو کنترولر، پردازنده GPU، DSP و FPGA نقاط قوت، ضعف و محدودیت هایی دارند. در حوزه FPGA ، انجام همه کارها با FPGA شدنی نیست یا صرفه اقتصادی هزینه زمان ندارد. از این رو برای غلبه به این محدودیت ها بایستی این ابزار ها را بایکدیگر ترکیب نمود. با پیشرفت تکنولوژی این ترکیب به داخل چیپ ها کشانده شده است و دریک چیپ همه ابزار ها را ترکیب شده اند. واژه SoC به معنی System on Chip هست و اصطلاحا به ترکیب و تجمیع زیر سیستم هایی در یک چیپ گفته می‌شود که برای اجرا سیستم عامل لازم است. باید به این نکته دقت شود که FPGA معمولا برای

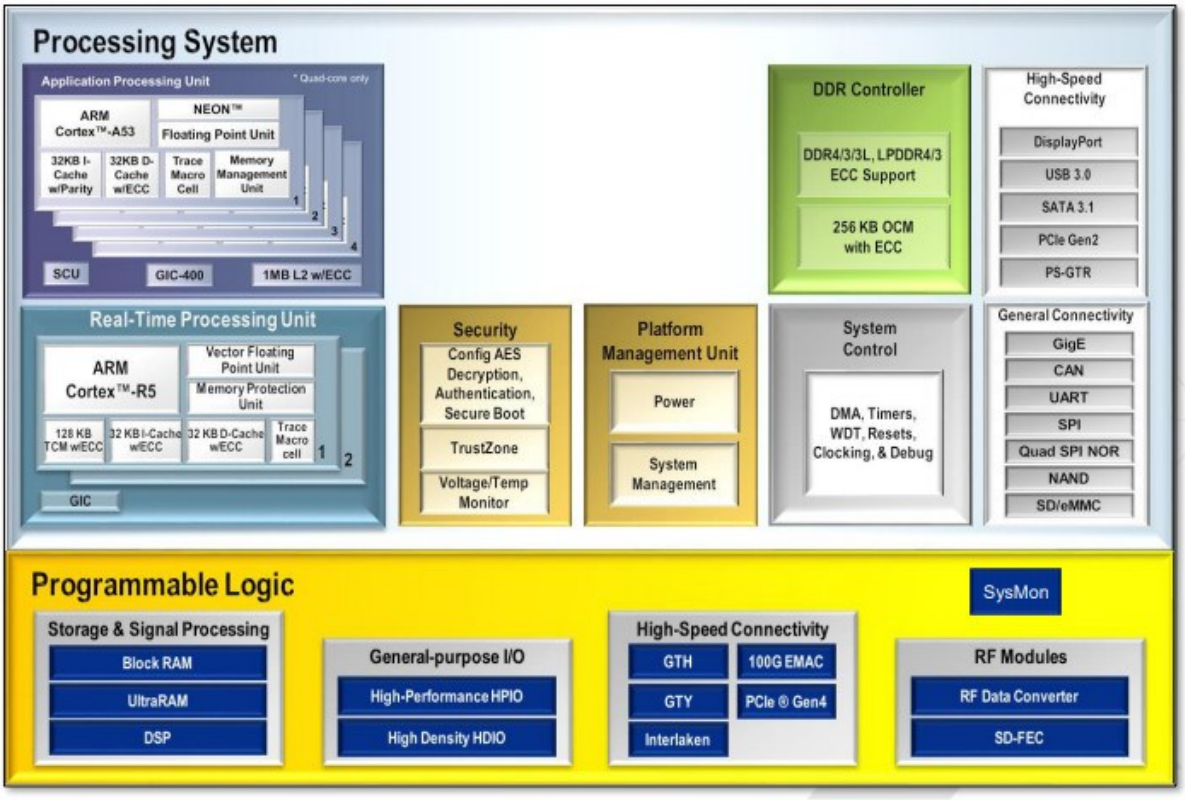
کار کنترل و یا پردازش در سیستم ها بکار گرفته می‌شود و در اینجا SoC هایی که دارای زیر سیستم FPGA هستند به هدف ساخت یک سیستم عامل پردازشی کنترلی ساخته می‌شوند. در این بین XILINX هم محصولات جالب و کاربردی ارائه کرده است. که استفاده از آنها پیاده سازی ها را ساده تر و ارزانتر می نماید. همچینین پیاده سازی الگوریتم های غیر ثابت (Adaptive) بسیار ساده تر و مقرون بصرفه شده است. خانواده SoC شرکت XILINX با نام ZYNQ ساخته شده و در چند کلاس ارائه می‌شود:

- ZYNQ 7000 (سری ۷) با FPGA از خانوادهARTIX
- ZYNQ 7000 (سری ۷) با FPGA از خانوادهKINTEX
- ZYNQهای Ultrascale با FPGA های خانوادهUltrascale
- ZYNQهای +Ultrascale با FPGA هی خانواده+Ultrascale

شـــرکت XILINX بعد از تولید MPSoC گام را فراتر نهاده و مدارات آنالوگ، تقویت کننده ها، میکسر، ADC و DAC پر سرعت را با MPSoC در یک چیپ ترکیب کرده است. چیپ های RFSoC علاوه بر پردازنده های مختلف و FPGA های بسیار قدرتمند+Ultrascale ، مدارات انالوگ ADC و DAC چندین کاناله را نیز دارا هستند. این چیپ ها ساخت رادیو های نرم افزاری، رادارها پرقدرت و .... را در سایز های بسیار کوچک فراهم نموده است و نیاز به PCB بزرگ و پیچیده را بسیار کاهش داده است. شکل ۱ نشان دهنده ی ساختار RFSoC است.

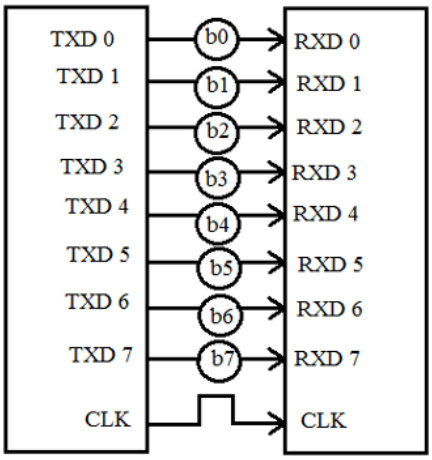
۳-۱- معرفی پروتکل های ارتباطی

ســیســـتم های نهفته و مدارات الکترونیک دیجیتال نیازمند ارتباط با یکدیگر هســـتند. برای مبادله اطلاعات، این سیستم ها نیازمند استفاده از پروتکل ارتباطی مشترک هستند. پروتکل های ارتباطی زیادی برای دستیابی به تبادل اطلاعات تعریف شده است که به طور کلی هر کدام از آن ها به دو دسته ی موازی یا سریال تقسیم می شوند. در ارتباط موازی چندین بیت همزمان انتقال می یابند. پروتکل های ارتباطی موازی معمولا از هشت یا شانزده خط انتقال داده استفاده می کنند. شکل ۲ نشان دهنده ی ارتباط موازی برای انتقال داده اســت. ارتباط موازی قطعاً مزیت‌های خودش را دارد ســریع، ســاده و قابل‌اجرا اســت. اما خطوط ورودی/خروجی (I/O) بســیار بیشــتری نیاز دارد. در ارتباط ســریال، در هر لحظه یک بیت از داده ها انتقال می یابند. این رابط ها می توانند به کوچکی یک سیم عمل کنند و معمولا بیشتر از چهار سیم نمی شوند. شکل ۳ نشان دهنده ی پروتکل ارتباطی سریال برای انتقال یک بیت در هر پالس است. USB و اترنت، دو نمونه از پروتکل‌های ارتباطی سریال محاسباتی معروف هستند. پروتکل‌های ارتباطی سریال دیگری مانند

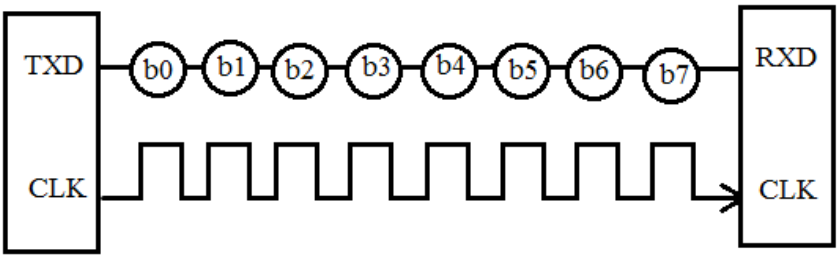


شکل ۱- ساختار داخلی RFSoC

I2C و SPI برای انتقال داده به صورت سریال ا ستفاده می شوند. هر یک از ارتباطات سریال را می‌توان به یکی از این دو گروه آ سنکرون یا سنکرون اخت صاص داد. ارتباط سریال سنکرون همی شه خطوط داده‌های خود را به سینگال ساعت وصل می‌کند. بنابراین تمام دستگاه‌های موجود در باس سریال سنکرون یک پالس ساعت مشخص را به اشتراک می‌گذارند که این باعث انتقال سریال ساده‌تر و در اغلب موارد سریع‌تر می‌شود، اما این ارتباط حداقل به یک سیم اضافی بین دستگاه‌های ارتباطی نیاز دارد. پروتکل ارتباطی SPI و I2C در این د سته قرار می گیرند. در ارتباط آ سنکرون داده‌ها بدون پ شتیبانی یک سیگنال ساعت خارجی منتقل می‌شوند. این روش انتقال برای کوچک کردن سیم‌ها و پین‌های I/O موردنیاز مناسب است، اما در این روش انتقال و دریافت مطمئن‌تر داده‌ها پیچیدگی بیشـــتری دارد. پروتکل ارتباطی UART در این دســـته جا می گیرد. در ادامه به معرفی مخت صر پروتکل I2C به عنوان یک پروتکل ارتباطی سریال سنکرون پرداخته می شود.

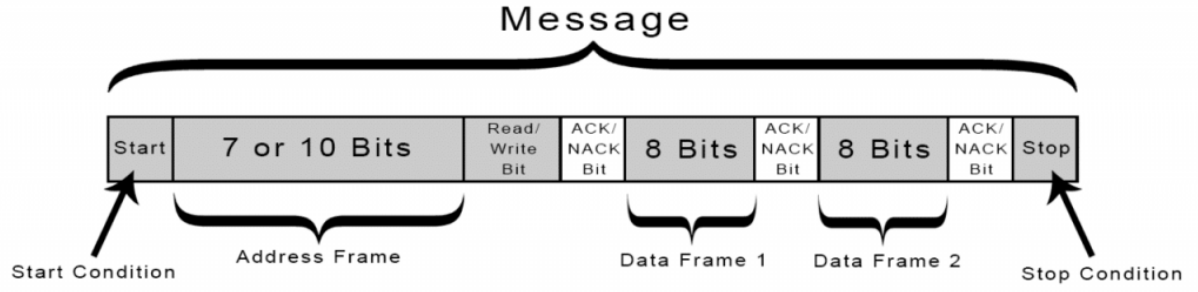


شکل ۲- پروتکل ارتباطی موازی



شکل ۳- پروتکل ارتباطی سری

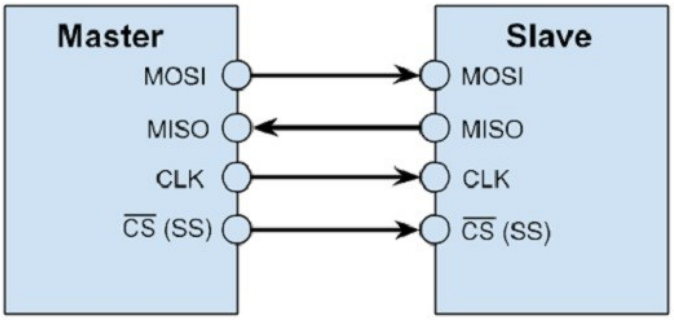
ارتباط I2C توسط Phillips Semiconductor معرفی شد و سالها بعد شرکت اینتل پروتکل SMBusرا به عنوان I2C تعریف کرد. پروتکل I2C بهترین ویژگی های SPI و UART را با هم ترکیب می کند. با استفاده از I2C می توان چندین Slave به یک Master واحد متصل کرد. همچنین با استفاده از این پروتکل می توان چندین Master را کنترل کرد که یک یا چند Slave را کنترل کنند. مانند ارتباطات UART، ارتباط I2C نیز فقط از دو سیم برای انتقال داده بین دستگاه ها استفاده می کند. مشابه SPI ، ارتباطI2C نیز سنکرون است، بنابراین خروجی بیت ها با نمونه برداری از بیت ها توسط یک سیگنال ساعت مشترک بین Master و Slave هماهنگ می شوند. سیگنال ساعت همیشه توسط Master کنترل می شود. با استفاده از I2C ، داده ها در پیام ها منتقل می شوند. پیام ها به فریم های داده تقسیم می شوند. هر پیام دارای یک فریم آدرس است که شامل آدرس باینری Slave و یک یا چند فریم داده است که حاوی داده های منتقل شده است. این پیام همچنین شامل شرایط شروع و توقف ، بیت های خواندن / نوشتن و بیت های ACK / NACKبین هر فریم داده است. شکل ۴ نشان دهنده ی فریم داده پروتکل I2C است.



شکل ٤- فریم داده در پروتکل ارتباطی I2C

۴-۱- معرفی پروتکل ارتباطی SPI

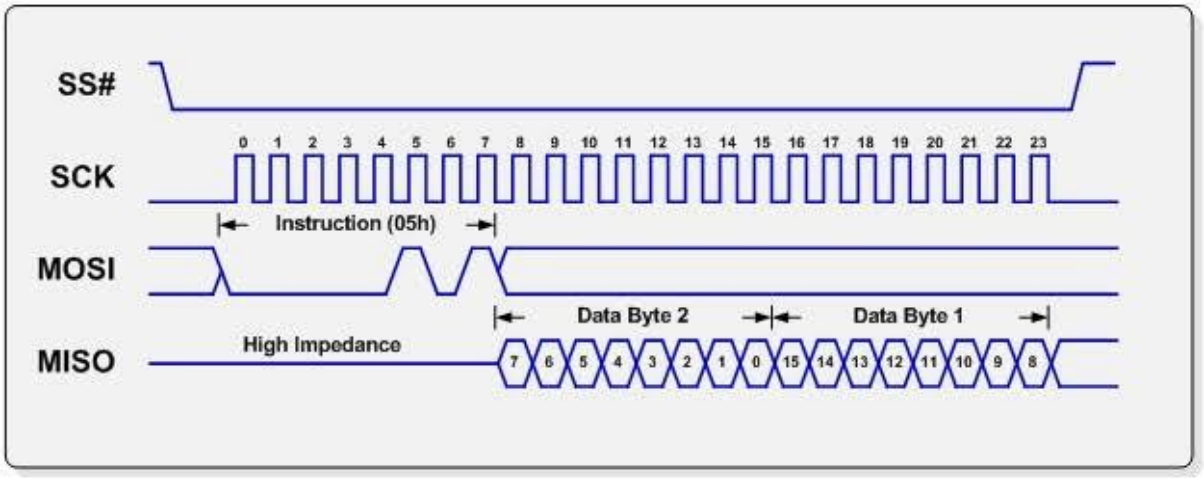
پروتکل SPI(Serial Peripheral Interface) یک رابط برای برقراری ارتباطات سنکرون سریال است و بیشتر برای ارتباطات از راه دور کاربرد دارد. این رابط در اواسط دهه ۱۹۸۰ توسط موتورولا توسعه یافته و به یک استاندارد تبدیل شده است. دستگاه های SPI از معماری Master-Slave استفاده می کنند و ارتباط در آن ها، به شکل دوطرفه برقرار می شود. Master چهارچوب را برای خواندن و نوشتن ایجاد می کند. این رابط به کمک خطوط Slave Select که Slave را مشخص می کند، چندین slave را می تواند پشتیبانی کند. شکل ۵ نشان دهنده ارتباط میان Master و Slave در پروتکل SPI را نشان می دهد.



شکل ۵- ارتباط میان Master-Slave با پروتکل ارتباطی SPI

برای شروع ارتباط، پس از تنظیم کلاکMaster ، میکروکنترلر Slave با خط انتخاب در سطح منطقی ۰ را بر می گزیند. در صورت نیاز به یک دوره انتظار،Master قبل از صدور چرخه های کلاک باید حداقل آن مدت زمان را منتظر بماند. در طی هر چرخه کلاک SPI، انتقال داده به صورت دوطرفه رخ می دهد. Master یک بیت روی خط MOSI می فر ستد و slave آن را می خواند .همچنین Slave یک بیت را روی خط MISO می فرستد و Master آن را می خواند. حتی در حالاتی که انتقال یک طرفه داشته باشیم، این روند همچنان حفظ می شود. پین انتقالی معمولاً شامل دو رجیستر به اندازه ۱۶ بیت است، یکی در دستگاه Master و دیگری در Slave است. داده ها معمولاً با بیت پر ارزش خارج می شوند .عمل انتقال ممکن است برای چند دوره کلاک ادامه پیدا کند. پس از اتمام، Master مسئول متوقف کردن سیگنال کلاک ا ست و به طور معمول Slave را از حالت انتخاب خارج می کند. طول داده های انتقالی اکثرا به اندازه

۸ بیت اســت. با این حال، انتقال با اندازه های دیگر نیز اتفاق می افتد؛ به عنوان مثال کلمات شــانزده بیتی برای کنترل کننده های صــفحه لمســی یا کدهای صــوتی، کلمات دوازده بیتی برای بســیاری از مبدل های دیجیتال به آنالوگ یا آنالوگ به دیجیتال و ... استفاده می شود. امکان جداسازی آسان، ارتباط دو طرفه کامل، توان عملیاتی بالا، عدم نیاز به اسیلاتور خارجی برای Slave ها، عدم نیاز به آدرس منحصر به فرد برای Slave ها، انعطاف پذیری در تعداد بیت های انتقالی، استفاده از چهار پایه در IC ها و پیاده سازی نرم افزاری ساده از جمله مزایای ا ستفاده از پروتکل SPI ا ست. عدم توانایی کنترل جریان سخت افزاری تو سط Slave، عدم تشخیص خطا و عملکرد صحیح در مسافت های کوتاه(البته با ا ستفاده از فر ستنده و گیرنده می توان فا صله را افزایش داد) از جمله معایب ا ستفاده از پروتکل SPI ا ست. شکل ۶ ن شان دهنده ی سیگنالینگ پروتکل SPI است.



شکل ٦- سیگنالینگ پروتکل ارتباطی SPI

۵-۱- ساختار گزارش

در ادامه این گزارش به بررسی مبدل آنالوگ به دیجیتال ADAR7251 پرداخته شده است. همچنین طریقه ی پیاده سازی پروتکل سریال سنکرون SPI برای این مبدل شرح داده شده ا ست. شبیه سازی این پروتکل نیز در ب ستر ت ست[۱] در نرم افزار Vivado ارائه شده ا ست. کدهای VHDL مربوط به پیاده سازی پروتکل SPI در پیوست این گزارش آورده شده است.

[۱] Test Bench –

فصل ۲:

بررسی ADAR7251

۱-۲- مقدمه

برای پیاده سازی پروتکل SPI از مبدل آنالوگ به دیجیتال ADAR7251 محصول شرکت Analog Device استفاده شده است. این مبدل ٤ کاناله با رزولوشن ١٦ بیت در سیستم های جمع آوری داده و سیستم های راداری استفاده می شود. این مبدل از پروتکل های ارتباطی موازی و سریال برای دستیابی به نرخ نمونه برداری ۳۰۰ کیلو نمونه بر ثانیه تا ۱٫۸ کیلو نمونه بر ثانیه پشتیبانی می کند. این مبدل دارای یک حلقه قفل فاز[1] روی تراشه است که طیف وسیعی از فرکانس های کلاک را پشتیبانی می کند. سیگنال های Conv_Ready و Data_Ready خروجی مبدل را با یک رمپ خارجی برای کاربردهایی نظیر رادارهای FMCW سنکرون می کند. مبدل آنالوگ به دیجیتال ADAR7251 با استفاده از پروتکل SPI کنترل می شود و برای ارتباطات سرعت بالا از اینترفیس سریال استفاده می کند. همچنین این مبدل دو ورودی/خروجی برای کاربردهای عمومی داراست. از دیگر ویژگی های این مبدل این است که به فیلتر ضد تشنج[2] نیاز ندارد و دارای LNA و PGA با 45dB بهره است. ورودی اسیلاتور/حلقه قفل فاز داخلی در محدوده 16MHz تا 54MHz است. محدوده دمایی این مبدل از ٤٠- تا ١٢٥ درجه سانتی گراد است. در فصل بعد توضیحات بیشتری از مبدل آنالوگ به دیجیتال ADAR7251 ارائه شده است.

۲-۲- قابلیت ها و ویژگی ها

قابلیت های مبدل آنالوگ به دیجیتال در ادامه تشریح شده است. همچنین در شکل ۷ بلوک دیاگرام عملکردی مربوط به ADAR7251 نشان داده شده است.

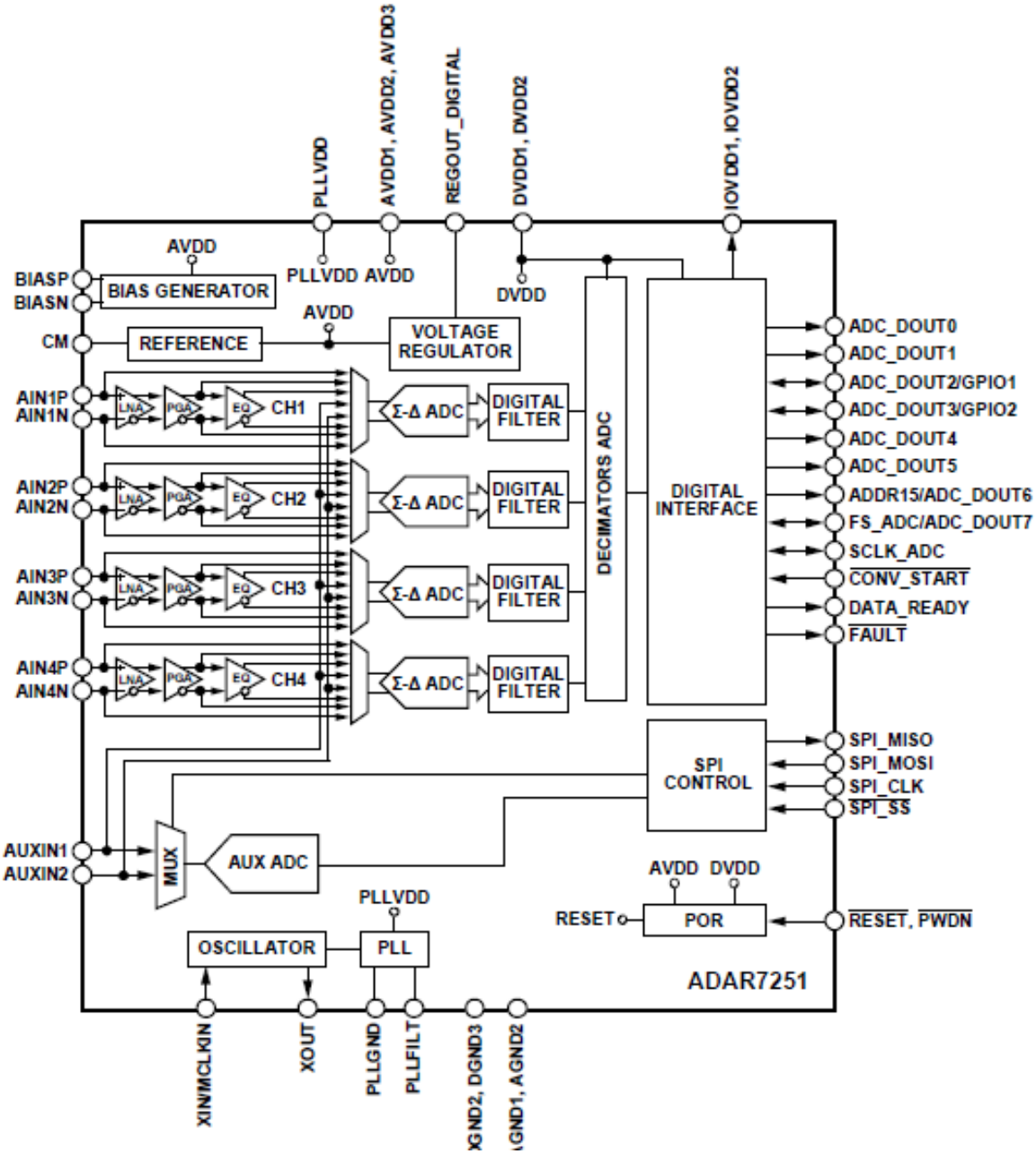
- استفاده از ۴ مبدل آنالوگ به دیجیتال در ساختار ADAR7251
- بهره گیری از LNA و PGA با حداکثر بهره 45 dB
- پشتیبانی از چهار کانال به صورت زمان پیوسته
- پهنای باند وسیع سیگنال ورودی: 500KHz در نرخ نمونه 1.2MSps
- پشتیبانی از نرخ داده های متنوع: 300kSps، 450kSps، 600kSps، 900kSps و 1.8MSps
- رزولوشن نمونه برداری ۱۶ بیتی
- پشتیبانی از اینترفیس داده سریال با سرعت بالا

---

[1] – PLL (Phased Lock Loop)

[2] – Anti-Aliasing

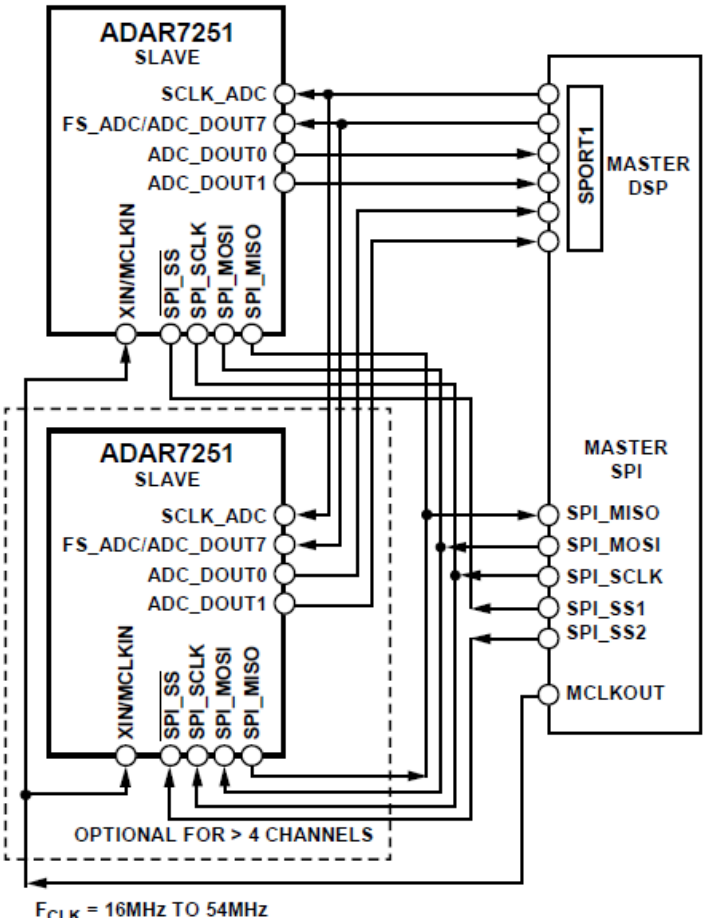
- پشتیبانی از پروتکل SPI برای کنترل
- ورودی اسیلاتور/ حلقه قفل فاز داخلی در محدوده 16MHz تا 54MHz
- پشتیبانی از مدولاسیون FSK برای سیستم های راداری FMCW
- محدوده دمایی ۴۰- تا ۱۲۵ درجه سانتی گراد



شکل ۷- بلوک دیاگرام عملکردی ADAR7251

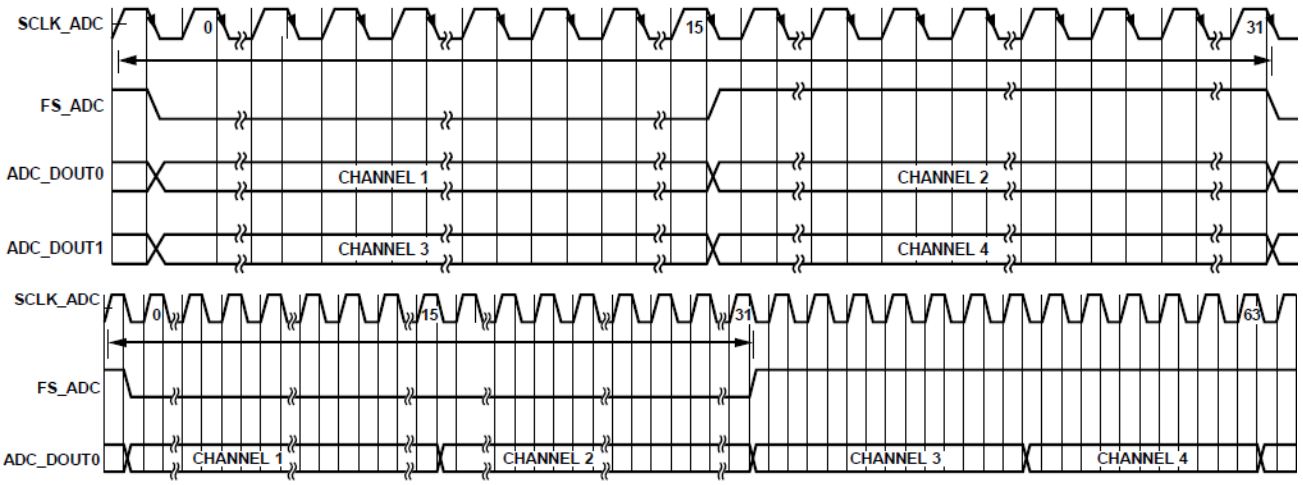
۳-۲- پروتکل ارتباطی ADAR7251

در حالت Master، ADC سیگنال های کلاک (SCLK_ADC) و سنکرون سازی فریم (FS_ADC) را تولید می کند. نرخ نمونه در حالت سریال به حداکثر ۱,۲ مگاهرتز می رسد. دو پورت ADC_DOUT0 و ADC_DOUT1 برای داده های سریال خروجی در نظر گرفته شده است. علاوه بر این، هر چهار کانال را می توان از یک پورت داده، ADC_DOUT0، خروجی گرفت. نرخ کلاک به نرخ نمونه و تعداد کانال ها بستگی دارد. شکل ۸ نشان دهنده ی اتصالات مود ADC Slave است.



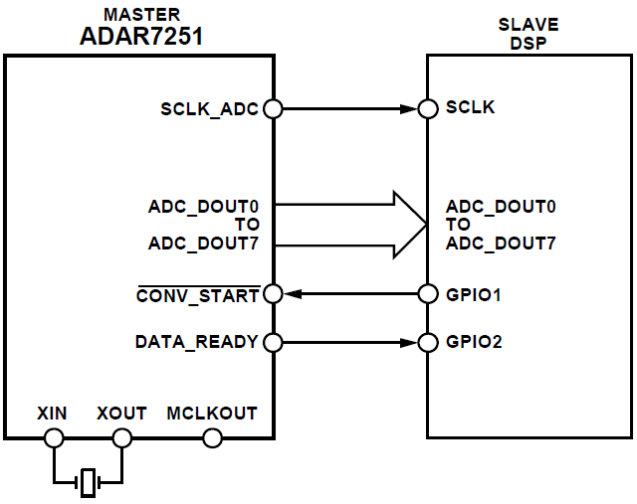
شکل ۸- اتصال سریال ADAR7251 در مد Slave

ارسال سریال داده در در یک کانال خروجی یا دو کانال خروجی در شکل ۹ نشان داده شده است. با استفاده از High و Low کردن Frame Sync می توان خروجی های مورد نظر را به صورت سریال در پورت های خروجی ارسال کرد.



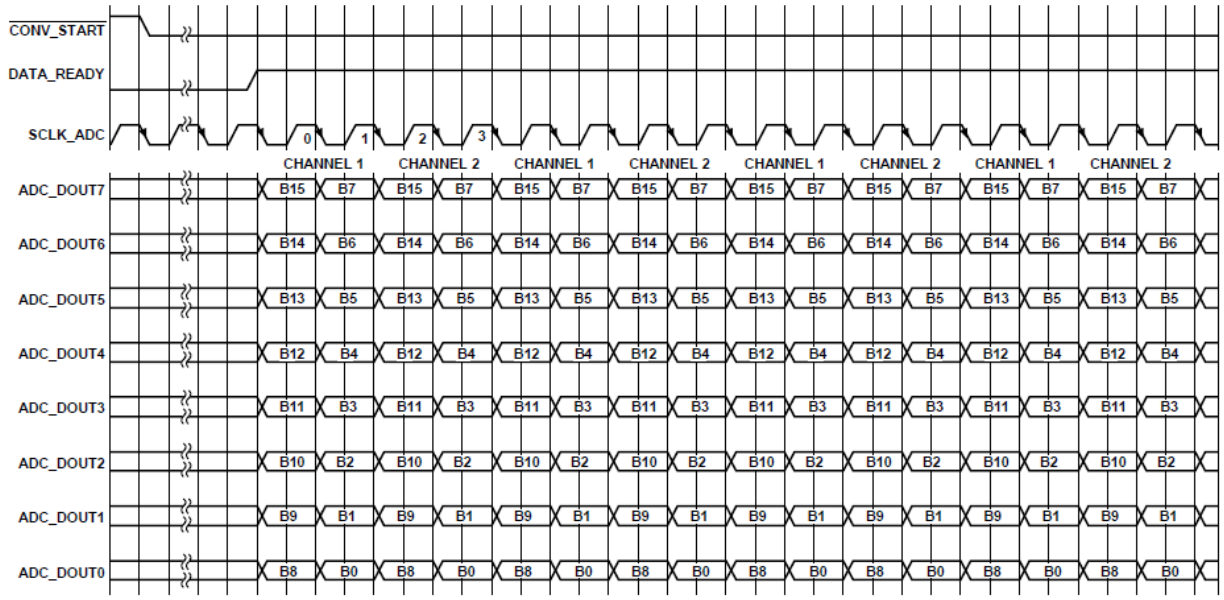
شکل ۹- مد سریال، ارسال دو کانال در هر یک از دو پورت خروجی و ارسال چهار کانال در یک پورت خروجی

ماژول ADAR7251 از پروتکل ارتباطی موازی PPI نیز پشـــتیبانی می کند. حالت ADC PPI حالت موازی بایت گسترده است و در این حالت، ADAR7251 همیشه در حالت Master است. در این حالت حداکثر نرخ نمونه 3.6 MHz پشـتیبانی می شـود. در این حالت در یک لحظه در ۸ خروجی Dout0 تا Dout7 یک بایت ظاهر می شود. شکل ۱۰ نشان دهنده ی ارتباطات در حالت PPI است.



شکل ۱۰- ارتباطات در مد PPI

سیگنالینگ دو کاناله در مد PPI در شکل ۱۱ نشان داده شده است.

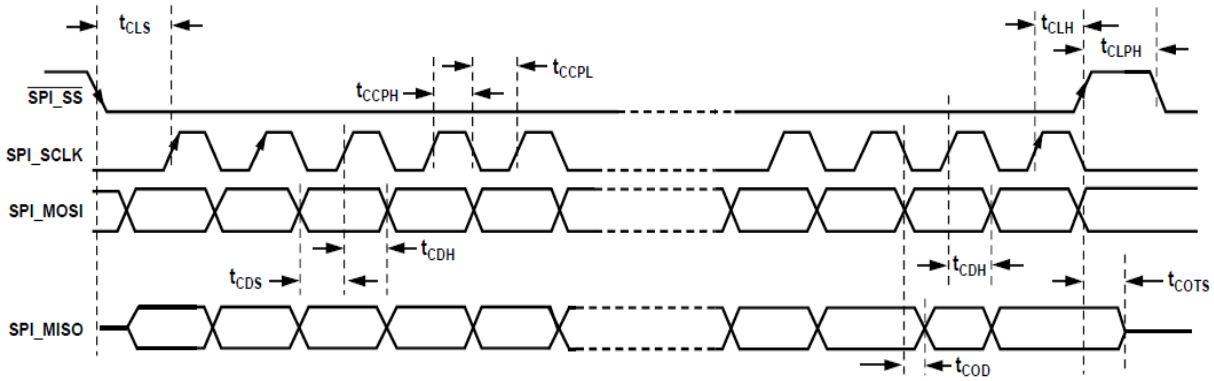


شکل ۱۱- سیگنالینگ در مد PPI – دو کاناله

با صفر شدن پین Conv_Start فرآیند تبدیل داده شروع می شود. وقتی ADC با داده های تبدیل آماده شد، پین DATA_READY را ۱ می شود تا وضعیت آماده بودن داده را به DSP نشان دهد. سپس ADC کلاک SCLK_ADC را ارائه می دهد. داده ها در لبه افزایشی کلاک در دسترس هستند.

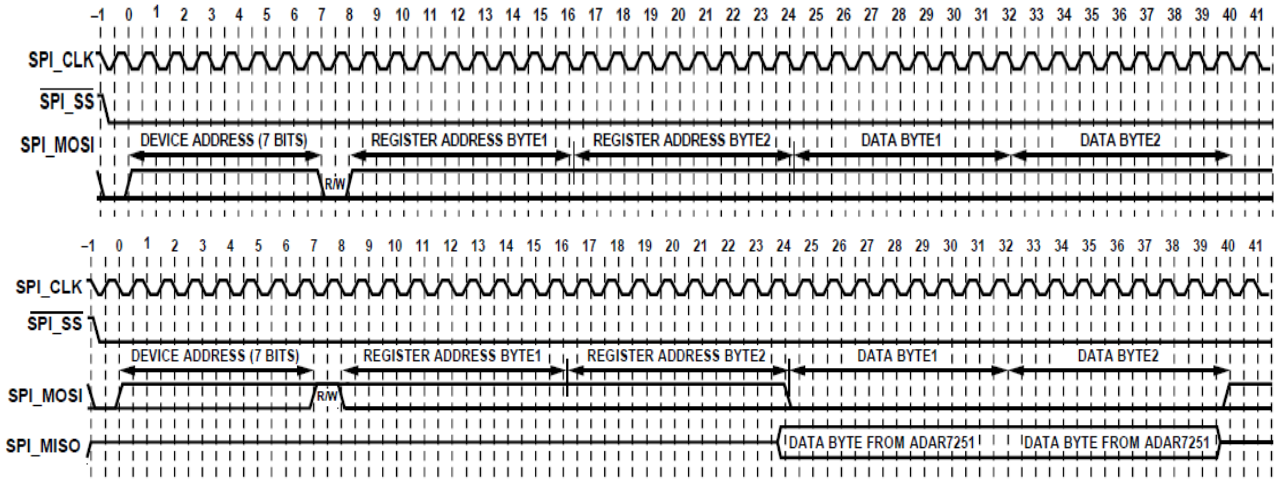
۲-4- پروتکل ارتباطی SPI

پورت کنترلی ADAR7251 از پروتکل ارتباطی SPI با ۳ سیم استفاده می کند. پورت SPI رجیسترهای داخلی دستگاه را تنظیم می کند. SPI امکان خواندن و نوشتن رجیستر ها را فراهم می کند. همه رجیسترها ۱۶ بیت عرض دارند. پورت کنترل SPI فقط از حالت Slave پشتیبانی می کند و بنابراین، برای کار کردن به Master در سیستم نیاز دارد. رجیسترها بدون کلاک اصلی دستگاه قابل دسترسی نیستند. رابط کنترل سریال همچنین به کاربر اجازه می دهد تا عملکردهای کمکی دستگاه مانند GPIO و ADC کمکی را کنترل کند. شکل ۸ نشان دهنده ی زمان بندی پروتکل ارتباطی SPI در ADAR7251 است.



شکل ۱۲- زمان بندی پورت SPI در ADAR7251

SPI_SS در ابتدای تراکنش پایین و در پایان تراکنش بالا می رود. سیگنال SPI_CLK از SPI_MOSI در انتقال SPI_CLK کم به بالا نمونه برداری می کند. بنابراین، داده هایی که باید روی دستگاه نوشته شوند باید در طول این لبه پایدار باشند. داده ها از SPI_MISO در لبه سقوط SPI_CLK به خارج منتقل می شوند و باید در یک دستگاه گیرنده، مانند یک میکروکنترلر، در لبه افزایشی SPI_CLK کلاک وارد شوند. سیگنال SPI_MOSI داده های ورودی سریال را به ADAR7251 و سیگنال SPI_MISO داده های خروجی سریال را از دستگاه حمل می کند. سیگنال SPI_MISO تا زمانی که عملیات خواندن درخواست نشود، به صورت تری استات باقی می ماند. زمان بندی جزیی تمامی جابجایی ها در شکل ۱۳ نشان داده شده است.



شکل ۱۳- زمان بندی و فریم داده های ارسالی در مد SPI

جدول ۲ پورت های ورودی و خروجی در حالت SPI را نشان می دهد.

جدول ۲- عملکرد پورت های کنترلی در حالت SPI

Pin No.	Mnemonic	Pin Function	Pin Type
32	ADDR15	Sets the device address for the SPI	Input
38	SPI_MISO	SPI port outputs data from the ADAR7251	Output
39	SPI_MOSI	SPI port inputs data to the ADAR7251	Input
40	SPI_CLK	SPI clock to the ADAR7251	Input
41	$\overline{\text{SPI\_SS}}$	SPI slave select to the ADAR7251	Input

فصل ۳:
پیاده سازی و شبیه سازی پروتکل SPI

۳-۱- مقدمه

در این فصل به بررسی پیاده سازی و شبیه سازی پروتکل SPI در ماژول ADAR7251 پرداخته می شود. نتایج در نرم افزار Vivado با نو شتن بستر تست[۱] شبیه سازی شده ا ست. با توجه به ندا شتن بستر سخت افزاری مطلوب اطلاعات وارد شده از پایه ی MISO تنها با تمام صفر و چک کردن بیت R/W بررسی شد. در صورت موجود بودن سخت افزار مناسب مقدار Lock bit، فعال بودن یا نبودن LNA ها و ... می توانست بررسی شود. خروجی MOSI با قراردادن اطلاعات در رجیسترها و فریم های داده تست و بررسی شد. نتایج بررسی ها در بخش ۳-۳ ثبت شده است.

۳-۲- روش پیاده سازی

برای پیاده سازی پروتکل ارتباطی SPI نیاز است تا ابتدا Entity را تعریف کرد. در Entity پورت های ورودی و خروجی تعریف می شوند. ۷ ورودی و خروجی در قسمت Entity تعریف می شود. جدول ۳ بیانگر تعریف پورت های تعریف شده است.

جدول ۳-پورت های ورودی و خروجی های تعریف شده برای پروتکل SPI

اسم پورت	In/out	توضیحات	اسم پورت	In/out	توضیحات
O_SPI_SDO	Out	داده خروجی	i_SPI_SDI	In	داده ورودی
O_SPI_SCLK	Out	کلاک SPI	i_clk	In	کلاک ورودی
O_SPI_CSB	Out	انتخاب Chip	i_reset	In	ریست ورودی
O_SPI_DIR	Out	نشان دهنده مسیر ورودی و خروجی	i_enable	In	نشانگر فعال بودن
O_reg_data	Out	رجیستر داده	i_reg_rd_wrb	In	خواندن یا نوشتن
O_busy	Out	نشانگر مشغول بودن ADC	i_reg_addr	In	رجیستر آدرس
O_finish	Out	نشانگر پایان عملیات	i_reg_data	In	رجیستر داده

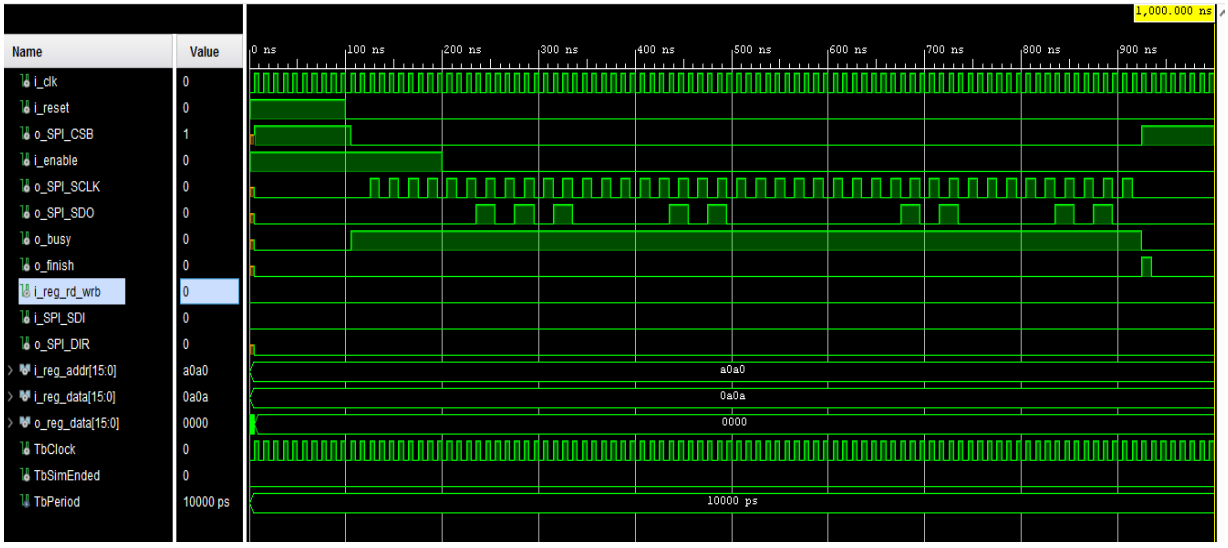
برای پیاده سازی این پروتکل از توصیف رفتاری استفاده شده است. کد راه اندازی این پروتکل ارتباطی به روش State Machine نوشته شده است و پنج حالت NOP، FirstBit، RxTxData، Read_state و LastBit تعریف شده است. در حالت NOP مقادیر اولیه تنظیم می شوند و آماده برای انتقال به حالت بعدی FirstBit می شود. اولین بیت در اندیس ۳۹ ام ا ست. توجه شود که در هر فریم مطابق شکل۱۳، ۵

[۱] Test Bench –

بایت انتقال اطلاعات صورت می گیرد. در حالت RxTxData، تصمیم گیری می شود که آیا ماژول فقط در حالت ارسال اطلاعات است یا قرار است اطلاعات از پایه ی MISO نیز اطلاعاتی خوانده شود. در صورتی که نیاز باشد از پایه ی MISO اطلاعاتی قرائت شود، برنامه به حالت Read_State منتقل می شود. در انتها زمانی که اندیس ۰ شود، برنامه به حالت Last_Bit منتقل می شود و نشانگر o_finish به نشانه ی اتمام فریم برابر با ۱ و نشانگر o_busy به نشانه ی اتمام فعالیت برابر با ۰ می شود.

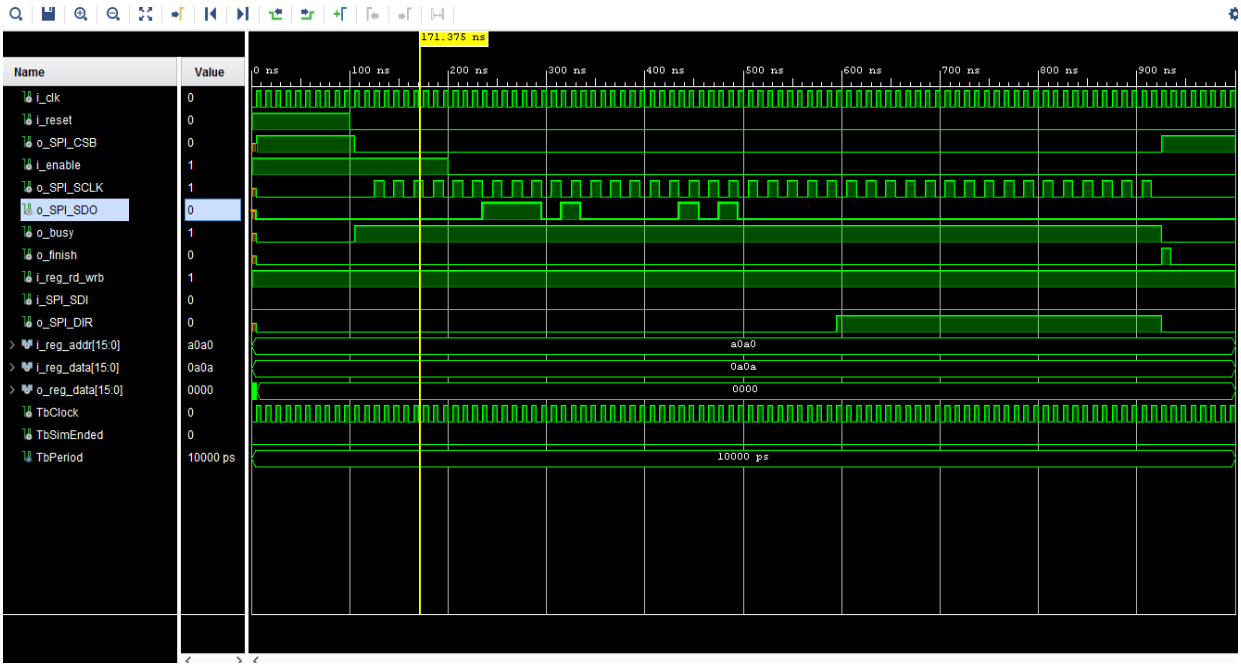
۳-۳- نتایج شبیه سازی

برای شبیه سازی نتایج از بستر تست استفاده شده است. این بستر هم به زبان VHDL و هم به زبان Verilog نوشته شده است. برای پیاده سازی شبیه سازی لازم است تا ابتدا Component مربوط به SPI را نمونه سازی کرد. سپس با تغییر مقادیر ورودی و کلاک می توان نتایج خروجی را در محیط Simulator نرم افزار Vivado بررسی کرد. شکل ۱۴ نشان دهنده ی شبیه سازی مربوط به پورت MISO است. از آنجا که امکانات سخت افزاری فراهم نبود تست پورت MISO امکان پذیر نیست ولی می توان با نوشتن آدرس دلخواه رجیستر و داده دلخواه رجیستر خروجی SDO را بررسی و مشاهده نمود. برای شبیه سازی مقدار آدرس رجیستر برابر با A0A0 قرار داده می شود و مقدار دیتا رجیستر برابر با 0A0A قرار داده می شود. همانگونه که مشاهده می شود با فعال شدن CSB و غیر فعال شدن Reset و Enable با کلاک های i_clk داده به خروجی SDO منتقل می شود. در اتمام شبیه سازی نیز پایه ی Busy برابر با صفر و پایه ی finish یک پالس می خورد. شکل ۱۴ نشان دهنده ی شبیه سازی این موضوع در نرم افزاری Vivado است.



شکل ۱۴- شبیه سازی پروتکل ارتباطی SPI در حالت write

برای شبیه سازی حالت خواندن در پروتکل SPI کافی است تا رجیستر مربوطه را برابر با ۱ قرار داد. یک نکته قابل توجه این است که در کد ADDR15 که در دیتاشیت بیان شده بود که از پایه های ولتاژی چیپ دیگر گرفته شود برابر با مقدار ثابت ۰ تعریف شده است.



شکل ۱۵- شبیه سازی پروتکل ارتباطی SPI در حالت Read

مشخص است که در حالت خواندن، بیت های ثابت ابتدایی در خروجی ظاهر شده است. در این حالت زمانی که به حالت Read_state منتقل می شویم، DIR مقدارش عوض می شود.

مراجع

مراجع

[1] SPI Interface Specification, Technical Note 15, VTI Technologies, 19 sep 2005.

[2] Abhimanyu Pandit, Serial Communication Protocols, circuitdigest.com, 2019.

[3] https://www.xilinx.com/products/silicon-devices/fpga.html

[4] ADAR7251 Datasheet, Analog Devices.

[5] ارتباط سریال- پروتکل UART، سایت میکرو دیزاینر الکترونیک، www.melec.ir

[6] معرفی شرکت Xilinx و خانواده FPGA های Xilinx، www.taksuntech.ir

پیوست:

کد VHDL پیاده سازی SPI مربوط به

ADAR7251

کد پیاده سازی SPI

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
--#############################################################################
entity SPI_ADAR7251_40bit is
	port(
		-- Signal from\to Bord
		o_SPI_SDO    : out     std_logic;  --connect to SDIO
		i_SPI_SDI    : in      std_logic;  --connect to SDIO
		o_SPI_SCLK   : out     std_logic;  --connect to SCLK
		o_SPI_CSB    : out     std_logic;  --connect to CSB
		o_SPI_DIR    : out     std_logic;  --connect to SDO
		-- Signal from\to other blocks
		i_clk        : in      std_logic;
		i_reset      : in      std_logic;
		i_enable     : in      std_logic;
		i_reg_rd_wrb : in      std_logic;
		i_reg_addr   : in      std_logic_vector(15 downto 0);
		i_reg_data   : in      std_logic_vector(15 downto 0);
		o_reg_data   : out     std_logic_vector(15 downto 0);
		o_busy       : out     std_logic;
		o_finish     : out     std_logic
		);
end SPI_ADAR7251_40bit;
--
#################################################################################
architecture Behavioral of SPI_ADAR7251_40bit is
--#############################################################################
-- Constant Declerations
--#############################################################################
	Constant C_ADDR15 : STD_LOGIC := '1';
--#############################################################################
-- Type Decleration
--#############################################################################
	type   SPIStateType is (NOP, FirstBit, RxTxData, Read_state ,LastBit);
--#############################################################################
-- Signal Decleration
--#############################################################################
```

```
        signal SPIState    : SPIStateType;
        signal SClkSig     : std_logic;
        signal sdo_tmp     : std_logic;
        signal DataOut     : std_logic_vector(15 downto 0);
        signal data_in_reg : std_logic_vector(39 downto 0);
--
################################################################################
begin
        --
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   o_SPI_SCLK     <= SClkSig;
   o_reg_data     <= DataOut(15 downto 0);
        o_SPI_SDO      <= sdo_tmp ;
        --
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   spi_trx : process(i_clk)
      variable Index : integer range 0 to 39;
   begin
                if rising_edge(i_clk) then
                        ----------------------
                        if (i_reset = '1') then
                                SPIState    <= NOP;
                                sdo_tmp            <= '0';
                                o_SPI_CSB    <= '1';
                                SClkSig     <= '0';
                                DataOut     <= (others => '0');
                                Index       := 39;
                                o_busy      <= '0';
                                o_finish    <= '0';
                                o_SPI_DIR   <= '0';
                                data_in_reg <= (others => '0');
                        ----------------------
                        else
                                case SPIState is
                                        --**************************
                                        when NOP =>
                                                o_SPI_DIR <= '0';
                                                o_finish  <= '0';
                                                if (i_enable = '1') then
                                                        SPIState    <= FirstBit;
                                                        o_SPI_CSB   <= '0';
                                                        SClkSig     <= '0';
```

```
                                        o_busy      <= '1';
                                        if i_reg_rd_wrb = '1' then
                                                data_in_reg   <=   "000000"   &
C_ADDR15 & "1" & i_reg_addr & i_reg_data;
                                        else
                                                data_in_reg   <=   "000000"   &
C_ADDR15 & "0" & i_reg_addr & i_reg_data;
                                        end if;
                                elsif
                                        o_SPI_CSB    <= '1';
                                end if;
                        --**************************
                        when FirstBit =>
                                SPIState <= RxTxData;
                                Index    := 39;
                                sdo_tmp  <= data_in_reg(Index);
                        --**************************
                        when RxTxData =>
                                SClkSig <= not SClkSig;
                                if (SClkSig = '1') then
                                        if data_in_reg(32) = '0' then
                                                if (Index = 0) then
                                                        SPIState <= LastBit;
                                                else
                                                        Index   := Index - 1;
                                                        sdo_tmp                 <=
data_in_reg(Index);
                                                end if;
                                        else
                                                if (Index = 16) then
                                                        SPIState  <= Read_state;
                                                        Index     := Index - 1;
                                                        o_SPI_DIR <= '1';
                                                else
                                                        Index   := Index - 1;
                                                        sdo_tmp                 <=
data_in_reg(Index);
                                                end if;

                                        end if;
                        end if;
                        --**************************
```

```
                                    when Read_state =>
                                            SClkSig <= not SClkSig;
                                            o_SPI_DIR     <= '1';
                                            if SClkSig = '1' then
                                                    if index = 0 then
                                                            SPIState <= LastBit;
                                                    end if;
                                                    DataOut(Index) <= i_SPI_SDI;
                                                    Index := Index - 1;
                                            end if;
                                    --**************************
                                    when LastBit =>

                                            o_SPI_DIR <= '0';
                                            SPIState <= NOP;
                                            o_SPI_CSB    <= '1';
                                            o_finish <= '1';
                                            o_busy   <= '0';


                            end case;
                    end if;
            end if;
    end process;
--
################################################################################
end Behavioral;
```

کد شبیه سازی (test bench) مربوط به SPI ADAR7251 (زبان VHDL)

```
library ieee;
use ieee.std_logic_1164.all;


entity tb_SPI_ADAR7251_40bit is
end tb_SPI_ADAR7251_40bit;


architecture tb of tb_SPI_ADAR7251_40bit is


    component SPI_ADAR7251_40bit
        port (o_SPI_SDO    : out std_logic;
```

```
            i_SPI_SDI    : in std_logic;
            o_SPI_SCLK   : out std_logic;
            o_SPI_CSB    : out std_logic;
            o_SPI_DIR    : out std_logic;
            i_clk        : in std_logic;
            i_reset      : in std_logic;
            i_enable     : in std_logic;
            i_reg_rd_wrb : in std_logic;
            i_reg_addr   : in std_logic_vector (15 downto 0);
            i_reg_data   : in std_logic_vector (15 downto 0);
            o_reg_data   : out std_logic_vector (15 downto 0);
            o_busy       : out std_logic;
            o_finish     : out std_logic);
    end component;


    signal o_SPI_SDO    : std_logic;
    signal i_SPI_SDI    : std_logic;
    signal o_SPI_SCLK   : std_logic;
    signal o_SPI_CSB    : std_logic;
    signal o_SPI_DIR    : std_logic;
    signal i_clk        : std_logic;
    signal i_reset      : std_logic;
    signal i_enable     : std_logic;
    signal i_reg_rd_wrb : std_logic;
    signal i_reg_addr   : std_logic_vector (15 downto 0);
    signal i_reg_data   : std_logic_vector (15 downto 0);
    signal o_reg_data   : std_logic_vector (15 downto 0);
    signal o_busy       : std_logic;
    signal o_finish     : std_logic;


    constant TbPeriod : time := 10 ns;
    signal TbClock : std_logic := '0';
    signal TbSimEnded : std_logic := '0';


begin


    dut : SPI_ADAR7251_40bit
    port map (o_SPI_SDO    => o_SPI_SDO,
            i_SPI_SDI    => i_SPI_SDI,
            o_SPI_SCLK   => o_SPI_SCLK,
```

```
            o_SPI_CSB    => o_SPI_CSB,
            o_SPI_DIR    => o_SPI_DIR,
            i_clk        => i_clk,
            i_reset      => i_reset,
            i_enable     => i_enable,
            i_reg_rd_wrb => i_reg_rd_wrb,
            i_reg_addr   => i_reg_addr,
            i_reg_data   => i_reg_data,
            o_reg_data   => o_reg_data,
            o_busy       => o_busy,
            o_finish     => o_finish);


--    Clock generation
   TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';
   i_clk <= TbClock;


   stimuli : process
   begin
      i_SPI_SDI <= '0';
      i_enable <= '1';
      i_reg_rd_wrb <= '1';
      i_reg_addr <= "1010000010100000";
      i_reg_data <= "0000101000001010";


--       Reset generation
      i_reset <= '1';
      wait for 100 ns;
      i_reset <= '0';
      wait for 100 ns;
      i_enable <= '0';
      wait for 100 * TbPeriod;


--       Stop the clock and hence terminate the simulation
      TbSimEnded <= '1';
      wait;
   end process;


end tb;
```

کد شبیه سازی (test bench) مربوط به SPI ADAR7251 (زبان Verilog)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
module TB_SPI_40;


	//Inputs
	reg i_SPI_SDI;
	reg i_clk;
	reg i_reset;
	reg i_enable;
	reg i_reg_rd_wrb;
	reg [15:0] i_reg_addr;
	reg [15:0] i_reg_data;


	//Outputs
	wire o_SPI_SDO;
	wire o_SPI_SCLK;
	wire o_SPI_CSB;
	wire o_SPI_DIR;
	wire [15:0] o_reg_data;
	wire o_busy;
	wire o_finish;


	//Instantiate the Unit Under Test (UUT)
	SPI_40bit uut (
		.o_SPI_SDO(o_SPI_SDO),
		.i_SPI_SDI(i_SPI_SDI),
		.o_SPI_SCLK(o_SPI_SCLK),
		.o_SPI_CSB(o_SPI_CSB),
		.o_SPI_DIR(o_SPI_DIR),
		.i_clk(i_clk),
		.i_reset(i_reset),
```

```
                .i_enable(i_enable),
                .i_reg_rd_wrb(i_reg_rd_wrb),
                .i_reg_addr(i_reg_addr),
                .i_reg_data(i_reg_data),
                .o_reg_data(o_reg_data),
                .o_busy(o_busy),
                .o_finish(o_finish)
        ;(


        initial begin
                //Initialize Inputs
                i_SPI_SDI = 0;
                i_clk = 1;
                i_reset = 1;
                i_enable = 0;
                i_reg_rd_wrb = 1;
                i_reg_addr = 16'h0505;;
                i_reg_data = 16'haaaa;


                //Wait 100 ns for global reset to finish
                ;#١٠١
      i_reset = 0;
                ;#٥٠
                i_enable =1;
                ;#١٠
                i_enable =0;
                //Add stimulus here


        end
 always #5 i_clk = ~i_clk;
endmodule
```