

دانشكده مهندسی برق

**پیاده سازی و شبیه سازی پروتکل SPI مربوط به ADAR7251**

گزارش پروژه درس VHDL

نام دانشجو

علی احدزاده آقبلاغ

استاد:

دكتر میرزاکوچکی

بهمن ماه 1401



فهرست مطالب

[فصل 1: مقدمه 1](#_Toc125588519)

[1-1- مقدمه 2](#_Toc125588520)

[1-2- تراشه های FPGA 2](#_Toc125588521)

[1-3- معرفی پروتکل های ارتباطی 3](#_Toc125588522)

[1-4- معرفی پروتکل ارتباطی SPI 6](#_Toc125588523)

[1-5- ساختار گزارش 7](#_Toc125588524)

[فصل 2: بررسی ADAR7251 8](#_Toc125588525)

[1-2- مقدمه 9](#_Toc125588526)

[2-2- قابلیت ها و ویژگی ها 9](#_Toc125588527)

[3-2- پروتکل ارتباطی ADAR7251 11](#_Toc125588528)

[4-2- پروتکل ارتباطی SPI 13](#_Toc125588529)

[فصل 3: پیاده سازی و شبیه سازی پروتکل SPI 16](#_Toc125588530)

[1-3- مقدمه 17](#_Toc125588531)

[2-3- روش پیاده سازی 17](#_Toc125588532)

[3-3- نتایج شبیه سازی 18](#_Toc125588533)

[مراجع 20](#_Toc125588534)

[پيوست‌: 22](#_Toc125588535)

فهرست اشکال

[شکل 1- ساختار داخلی RFSoC 4](#_Toc125585165)

[شکل 2- پروتکل ارتباطی موازی 5](#_Toc125585166)

[شکل 3- پروتکل ارتباطی سری 5](#_Toc125585167)

[شکل 4- فریم داده در پروتکل ارتباطی I2C 6](#_Toc125585168)

[شکل 5- ارتباط میان Master-Slave با پروتکل ارتباطی SPI 6](#_Toc125585169)

[شکل 6- سیگنالینگ پروتکل ارتباطی SPI 7](#_Toc125585170)

[شکل 7- بلوک دیاگرام عملکردی ADAR7251 10](#_Toc125585171)

[شکل 8- اتصال سریال ADAR7251 در مد Slave 11](#_Toc125585172)

[شکل 9- مد سریال، ارسال دو کانال در هر یک از دو پورت خروجی و ارسال... 12](#_Toc125585173)

[شکل 10- ارتباطات در مد PPI 12](#_Toc125585174)

[شکل 11- سیگنالینگ در مد PPI – دو کاناله 13](#_Toc125585175)

[شکل 12- زمان بندی پورت SPI در ADAR7251 14](#_Toc125585176)

[شکل 13- زمان بندی و فریم داده های ارسالی در مد SPI 14](#_Toc125585177)

[شکل 14- شبیه سازی پروتکل ارتباطی SPI در حالت write 18](#_Toc125585178)

[شکل 15- شبیه سازی پروتکل ارتباطی SPI در حالت Read 19](#_Toc125585179)

فهرست جداول

جدول 1- دسته بندی FPGA های مختلف شرکت Xilinx ....................................................................................2

جدول 2- عملکرد پورت های کنترلی در حالت SPI .............................................................................................15

جدول 3- پورت های ورودی خروجی تعریف شده برای پروتکل SPI ..................................................................17

1. مقدمه

# 1-1- مقدمه

در سال 1984 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 حاضر بیش از 53 درصد از بازار FPGA در دست شرکت Xilinx است. همچنین تکنولوژی ساخت تراشه های 16 نانومتری تنها در اختیار شرکت Xilinx است. تکنولوژی ساخت با نانومتر پایین تر به معنای تراشه های سریعتر با مصرف کمتر است. 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.

# 1-2- 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 دسته های 45nm، 28nm، 20nm و 16nm تقسیم می شوند. جدول 1 نشان دهنده ی دسته بندی FPGA های خانواده Xilinx است.

جدول 1- دسته بندی FPGA های مختلف شرکت Xilinx

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

خانواده اسپارتان ها جزو ارزانترین FPGA های شرکت Xilinx است و ارزان ترین I/O نسبت به قیمت را دارا هستند. خانواده Artix جزء سری های ارزان و اقتصادی Xilinx است 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.

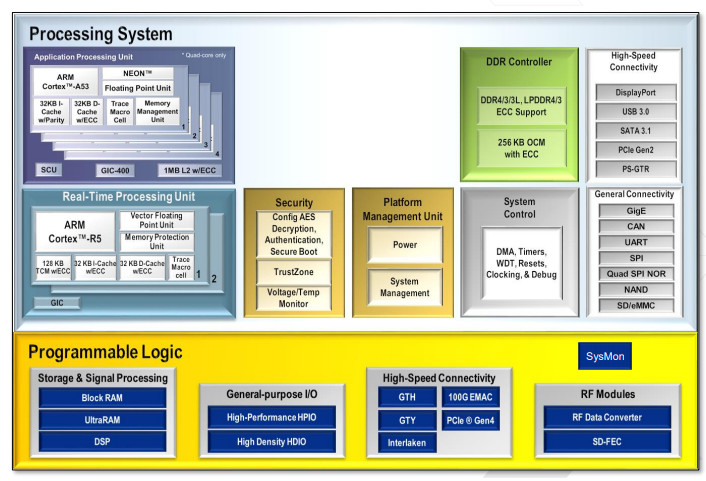
تمامی ابزارهای پیاده سازی مانند میکرو کنترولر، پردازندهGPU ، DSP و FPGA نقاط قوت، ضعف و محدودیت هایی دارند. در حوزه FPGA ، انجام همه کارها با FPGA شدنی نیست یا صرفه افتصادی هزینه زمان ندارد. از این رو برای غلبه به این محدودیت ها بایستی این ابزار ها را بایکدیگر ترکیب نمود. با پیشرفت تکنولوژی این ترکیب 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 خانواده ARTIX
* ZYNQ 7000 (سری ۷) با FPGA از خانواده KINTEX
* ZYNQ های Ultrascale با FPGA های خانواده Ultrascale
* ZYNQ های Ultrascale+ با FPGA هی خانواده Ultrascale+

شرکت XILINX بعد از تولید MPSoC گام را فراتر نهاده و مدارات آنالوگ، تقویت کننده ها، میکسر، ADC و DAC پرسرعت 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 1 نشان دهنده ی ساختار RFSoC است.

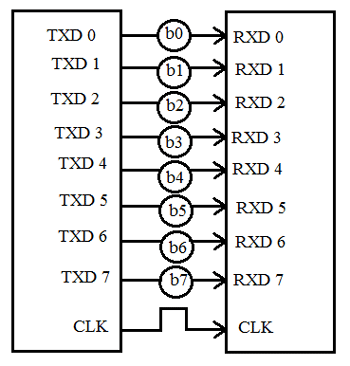
# 1-3- معرفی پروتکل های ارتباطی

سیستم های نهفته و مدارات الکترونیک دیجیتال نیازمند ارتباط با یکدیگر هستند. 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 2 نشان دهنده ی ارتباط موازی برای انتقال داده است. ارتباط موازی قطعاً مزیت‌های خودش را دارد سریع، ساده و قابل‌اجرا است. اما خطوط ورودی/خروجی (I/O) بسیار بیشتری نیاز دارد. در ارتباط سریال، 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 3 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

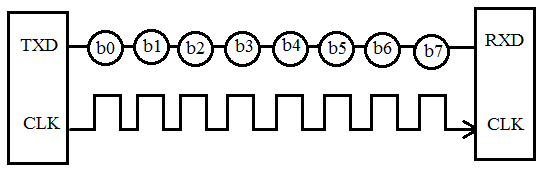


شکل 1- ساختار داخلی RFSoC

I2C و SPI برای انتقال داده به صورت سریال استفاده می شوند. هر یک از ارتباطات سریال را می‌توان به یکی از این دو گروه آسنکرون یا سنکرون اختصاص داد. ارتباط سریال سنکرون همیشه 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 دسته قرار می گیرند. در ارتباط آسنکرون داده‌ها بدون پشتیبانی 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 بیشتری دارد. پروتکل ارتباطی UART در این دسته جا می گیرد. در ادامه به معرفی مختصر پروتکل I2C به عنوان یک پروتکل ارتباطی سریال سنکرون پرداخته می شود.

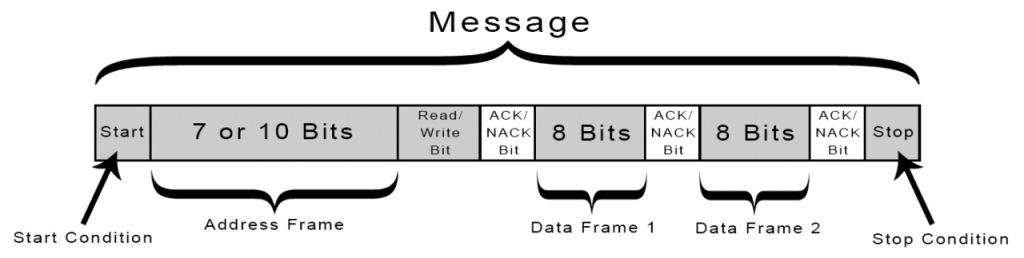


شکل 2- پروتکل ارتباطی موازی



شکل 3- پروتکل ارتباطی سری

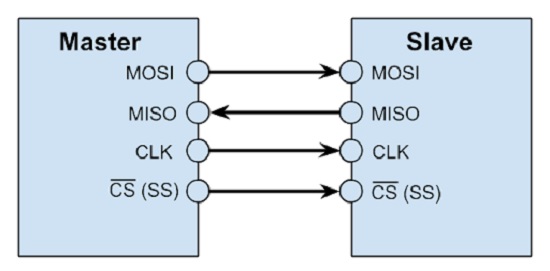
ارتباط I2C توسط Phillips Semiconductor معرفی شد و سالها بعد شرکت اینتل پروتکل SMBus را به عنوان I2C تعریف کرد. پروتکل I2C بهترین ویژگی های SPI و UART را با هم ترکیب می کند. با استفاده از I2C می توان چندین Slave به یک Master واحد متصل کرد. همچنین با استفاده از این پروتکل می توان چندین Master را کنترل 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 ، ارتباط I2C 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 / NACK بین هر فریم داده است. شکل 4 نشان دهنده ی فریم داده پروتکل I2C است.



شکل 4- فریم داده در پروتکل ارتباطی I2C

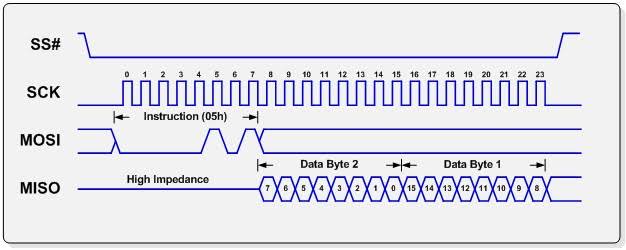
# 1-4- معرفی پروتکل ارتباطی SPI

**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 1980 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 5 نشان دهنده ارتباط میان Master و Slave در پروتکل SPI را نشان می دهد.



شکل 5- ارتباط میان Master-Slave با پروتکل ارتباطی SPI

برای شروع ارتباط، پس از تنظیم کلاک Master، میکروکنترلر Slave با خط انتخاب در سطح منطقی 0 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 فرستد 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 16 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، Masterمسئول متوقف کردن سیگنال کلاک است و به طور معمول Slave را از حالت انتخاب خارج می کند. طول داده های انتقالی اکثرا به اندازه 8 بیت است. با این حال، انتقال با اندازه های دیگر نیز اتفاق می افتد؛ به عنوان مثال کلمات شانزده بیتی برای کنترل کننده های صفحه لمسی یا کدهای صوتی، کلمات دوازده بیتی برای بسیاری 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 استفاده از پروتکل SPI است. عدم توانایی کنترل جریان سخت افزاری توسط Slave، عدم تشخیص خطا و عملکرد صحیح در مسافت های کوتاه( البته با استفاده از فرستنده و گیرنده می توان فاصله را افزایش داد) از جمله معایب استفاده از پروتکل SPI است. شکل 6 نشان دهنده ی سیگنالینگ پروتکل SPI است.



شکل 6- سیگنالینگ پروتکل ارتباطی SPI

# 1-5- 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 است. شبیه سازی این پروتکل نیز در بستر تست[[1]](#footnote-1) در نرم افزار Vivado ارائه شده است. کدهای VHDL مربوط به پیاده سازی پروتکل SPI در پیوست این گزارش آورده شده است.

1. بررسی ADAR7251

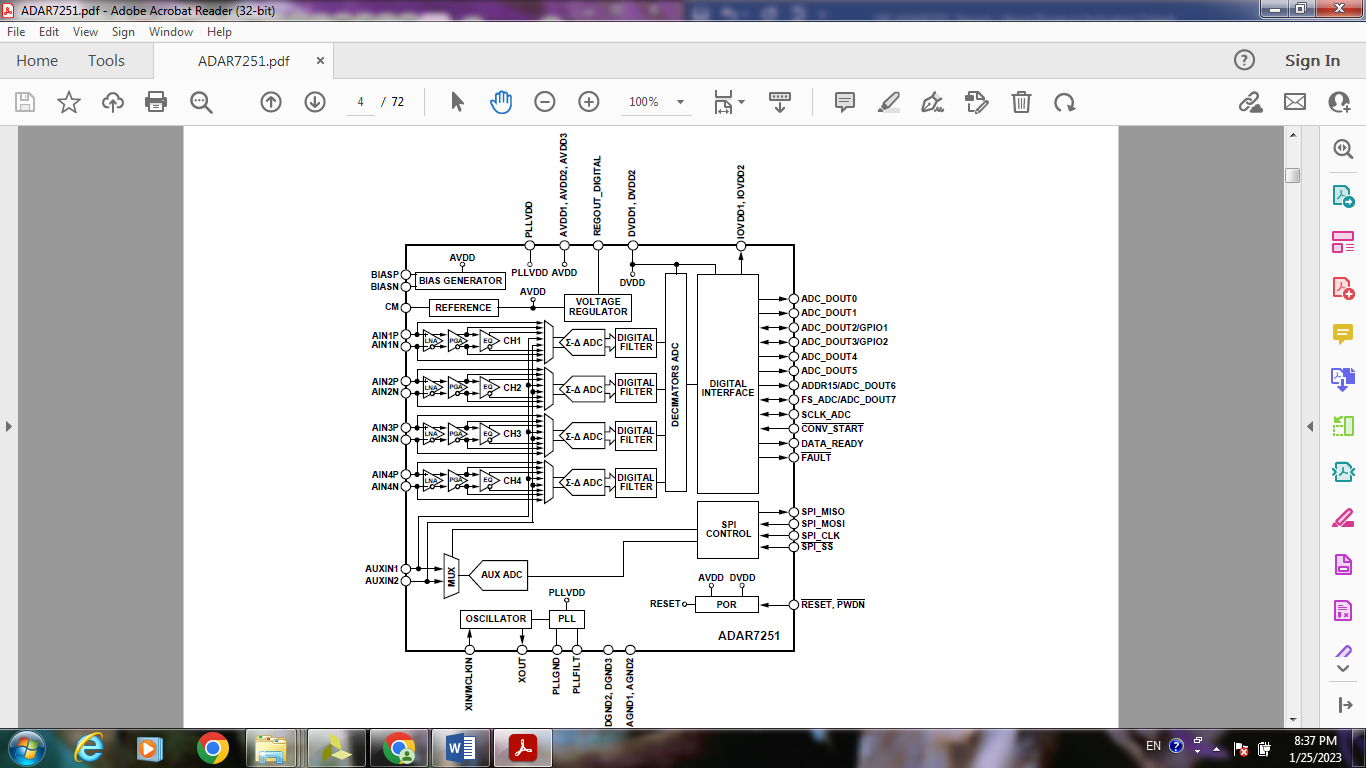
# 1-2- مقدمه

برای پیاده سازی پروتکل SPI از مبدل آنالوگ به دیجیتال ADAR7251 محصول شرکت Analog Device استفاده شده است. این مبدل 4 کاناله با رزولوشن 16 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 300 کیلو نمونه بر ثانیه تا 1.8 کیلو نمونه بر ثانیه پشتیبانی می کند. این مبدل دارای یک حلقه قفل فاز[[2]](#footnote-2) 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[[3]](#footnote-3) 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 40- تا 125 درجه سانتی گراد است. در فصل بعد توضیحات بیشتری از مبدل آنالوگ به دیجیتال ADAR7251 ارائه شده است.

# 2-2- قابلیت ها و ویژگی ها

قابلیت های مبدل آنالوگ به دیجیتال در ادامه تشریح شده است. همچنین در شکل 7 بلوک دیاگرام عملکردی مربوط به ADAR7251 نشان داده شده است.

* استفاده از 4 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 16 بیتی
* پشتیبانی از اینترفیس داده سریال با سرعت بالا
* 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 40- تا 125 درجه سانتی گراد

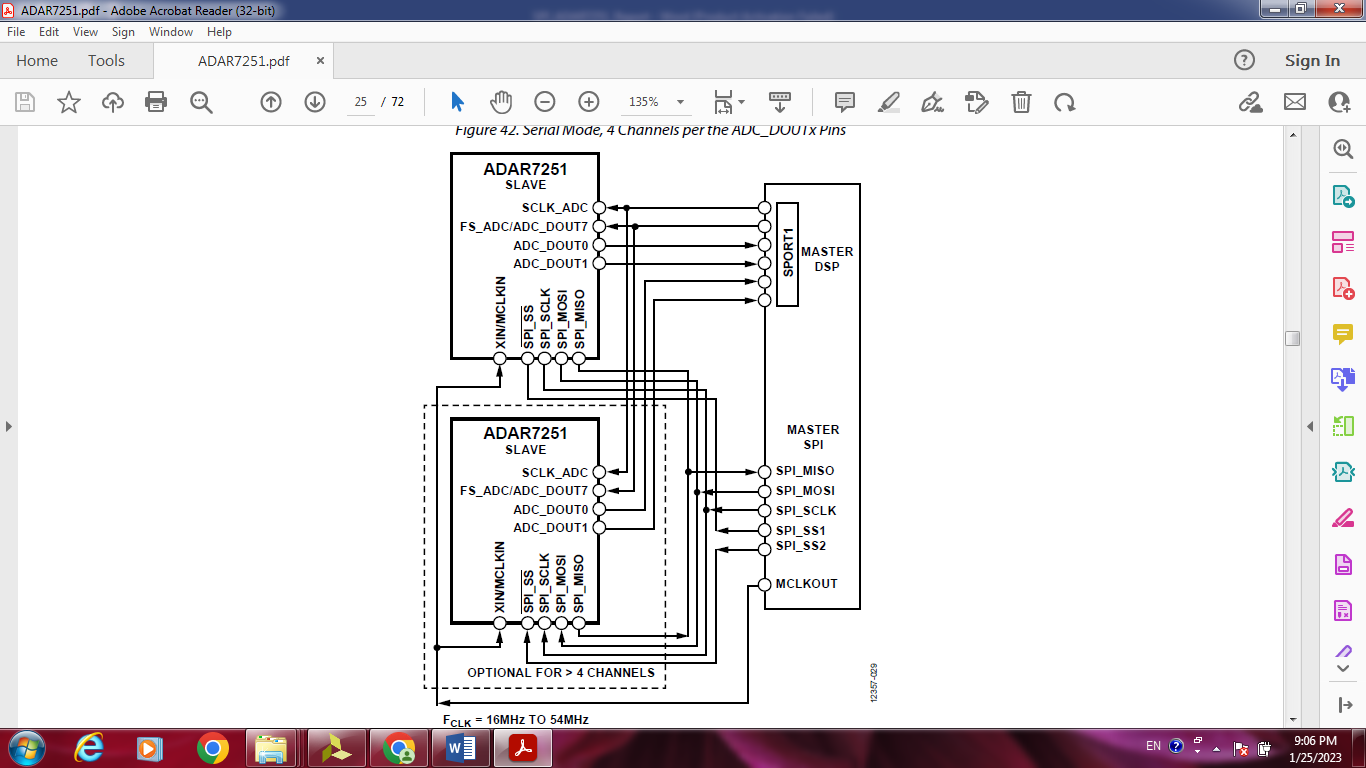


شکل 7- بلوک دیاگرام عملکردی ADAR7251

# 3-2- پروتکل ارتباطی ADAR7251

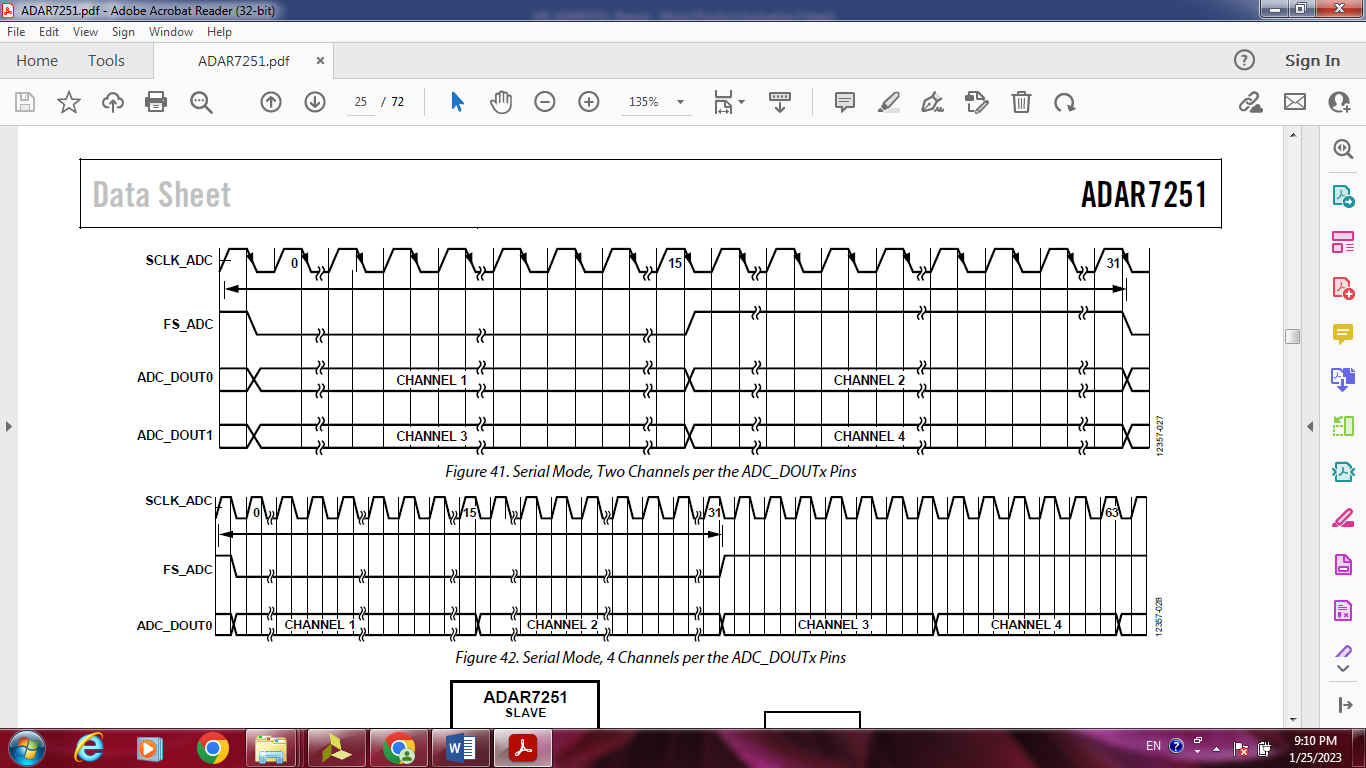
در حالت Master، ADC سیگنال های کلاک (SCLK\_ADC) و سنکرون سازی فریم (FS\_ADC) را تولید می کند. نرخ نمونه در حالت سریال به حداکثر 1.2 مگاهرتز می رسد. دو پورت

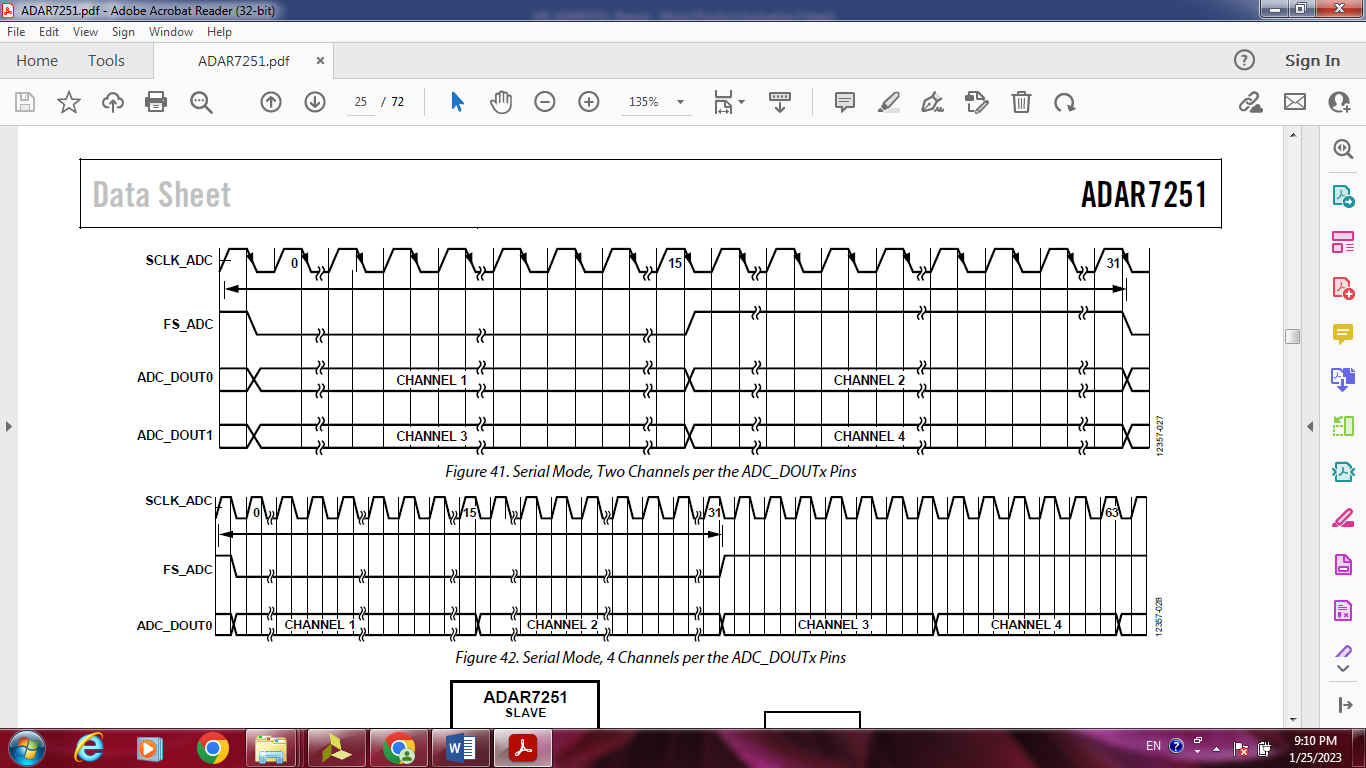
ADC\_DOUT0 و ADC\_DOUT1 برای داده های سریال خروجی در نظر گرفته شده است. علاوه بر این، هر چهار کانال را می توان از یک پورت داده، ADC\_DOUT0، خروجی گرفت. نرخ کلاک به نرخ نمونه و تعداد کانال ها بستگی دارد. شکل 8 نشان دهنده ی اتصالات مود Slave ADC است.



شکل 8- اتصال سریال ADAR7251 در مد Slave

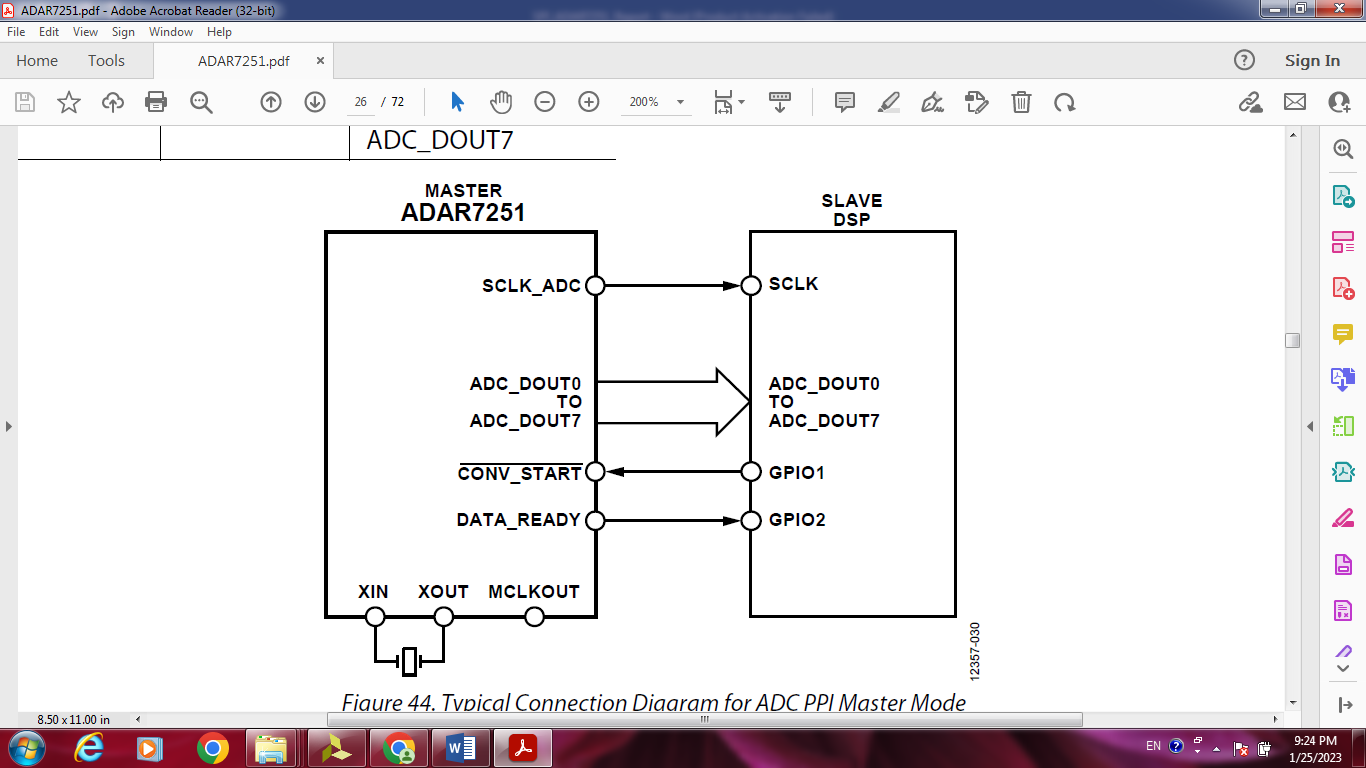
ارسال سریال داده در در یک کانال خروجی یا دو کانال خروجی در شکل 9 نشان داده شده است. با استفاده از High و Low کردن Frame Sync می توان خروجی های مورد نظر را به صورت سریال در پورت های خروجی ارسال کرد.





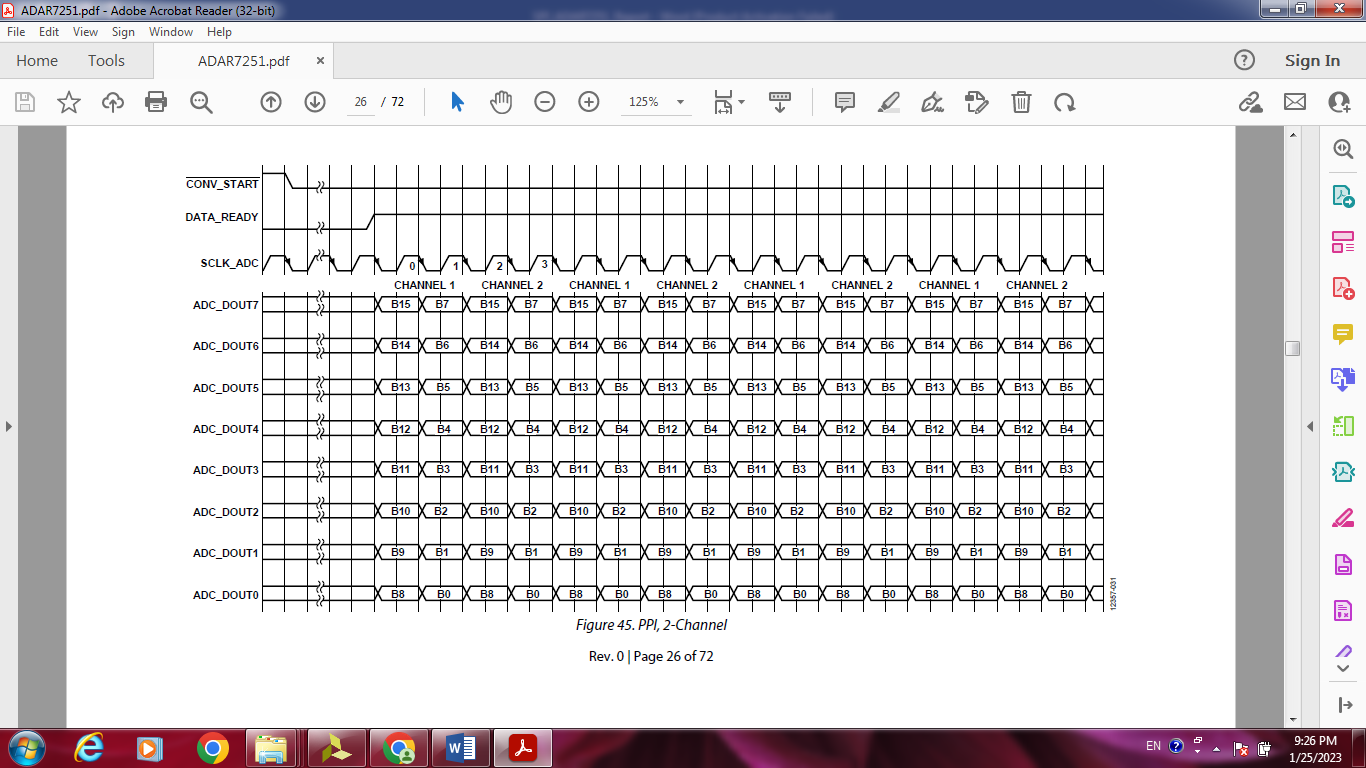
شکل 9- مد سریال، ارسال دو کانال در هر یک از دو پورت خروجی و ارسال چهار کانال در یک پورت خروجی

ماژول ADAR7251 از پروتکل ارتباطی موازی PPI نیز پشتیبانی 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 پشتیبانی می شود. در این حالت در یک لحظه در 8 خروجی Dout0 تا Dout7 یک بایت ظاهر می شود. شکل 10 نشان دهنده ی ارتباطات در حالت PPI است.



شکل 10- ارتباطات در مد PPI

سیگنالینگ دو کاناله در مد PPI در شکل 11 نشان داده شده است.

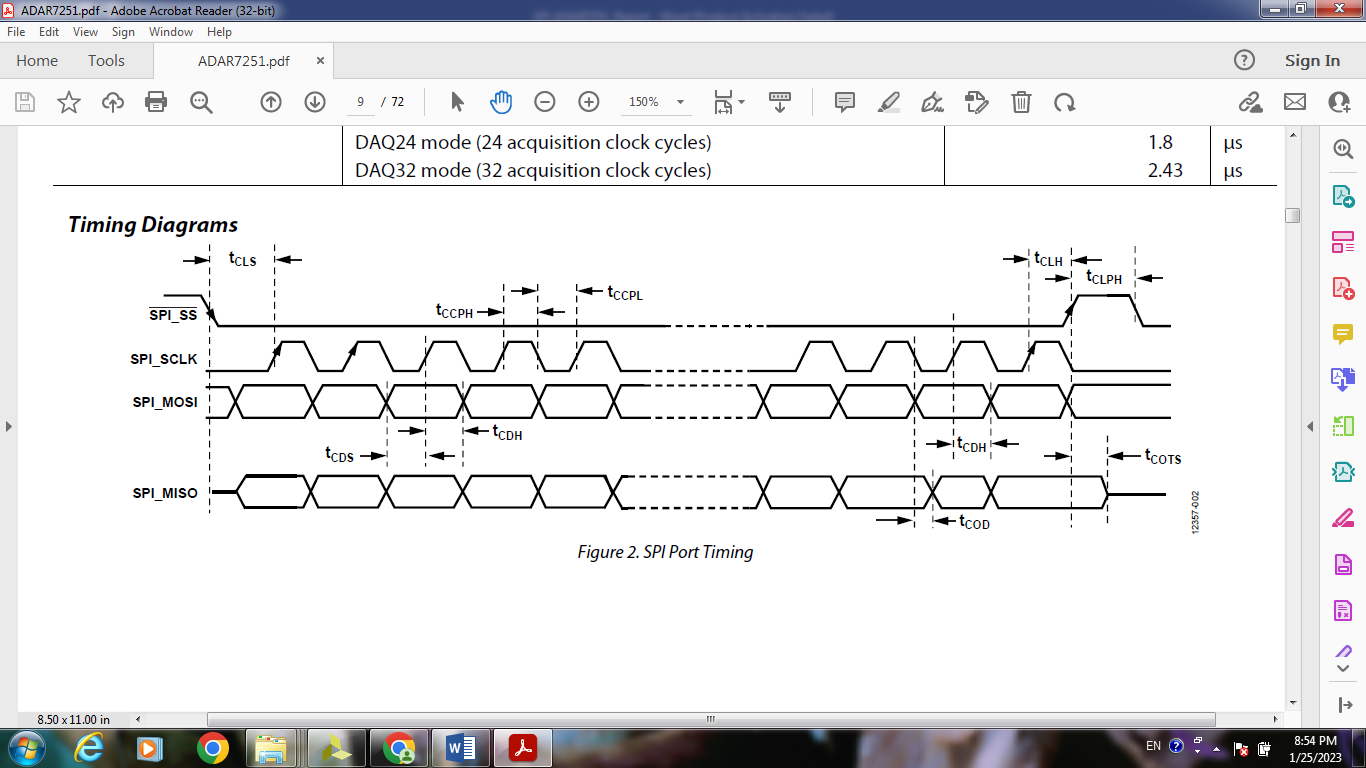


شکل 11- سیگنالینگ در مد PPI – دو کاناله

با صفر شدن پین Conv\_Start فرآیند تبدیل داده شروع می شود. وقتی ADC با داده های تبدیل آماده شد، پین DATA\_READY را 1 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.

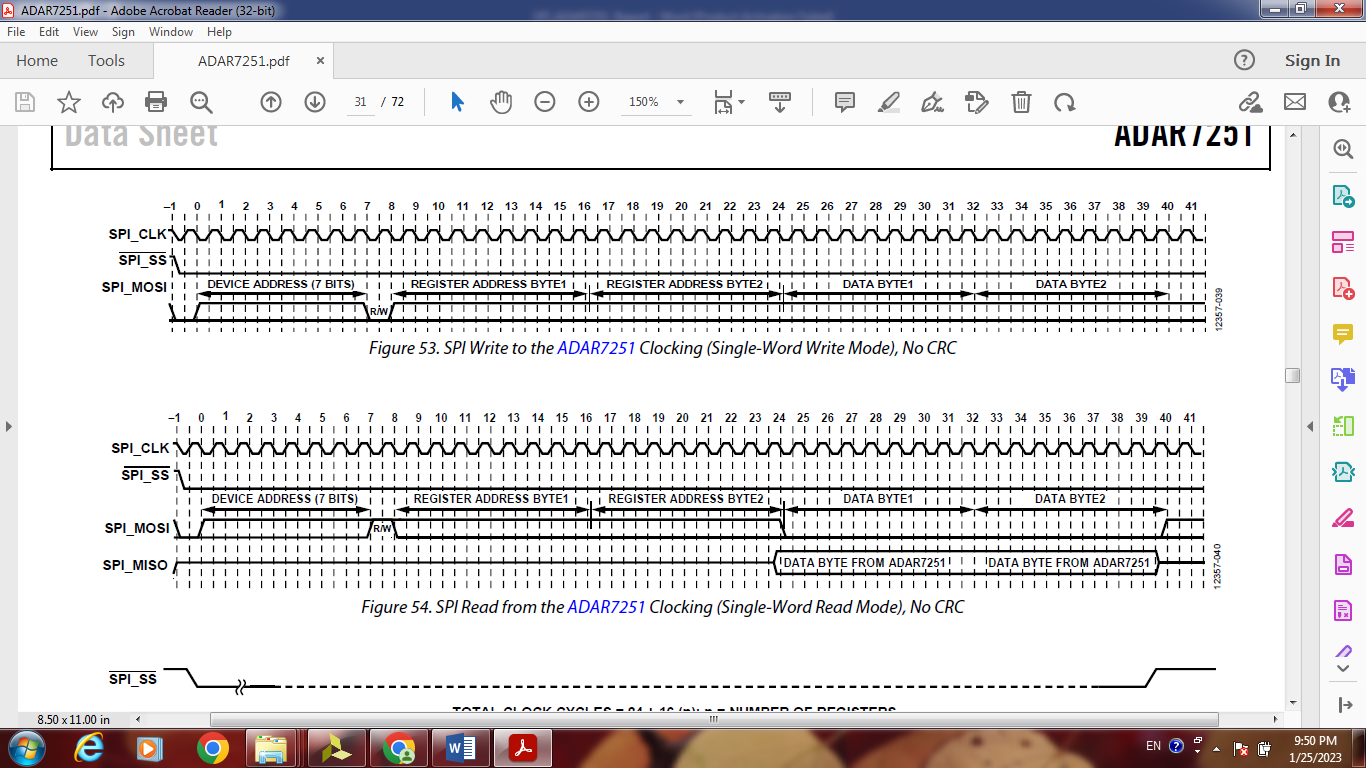
# 4-2- پروتکل ارتباطی SPI

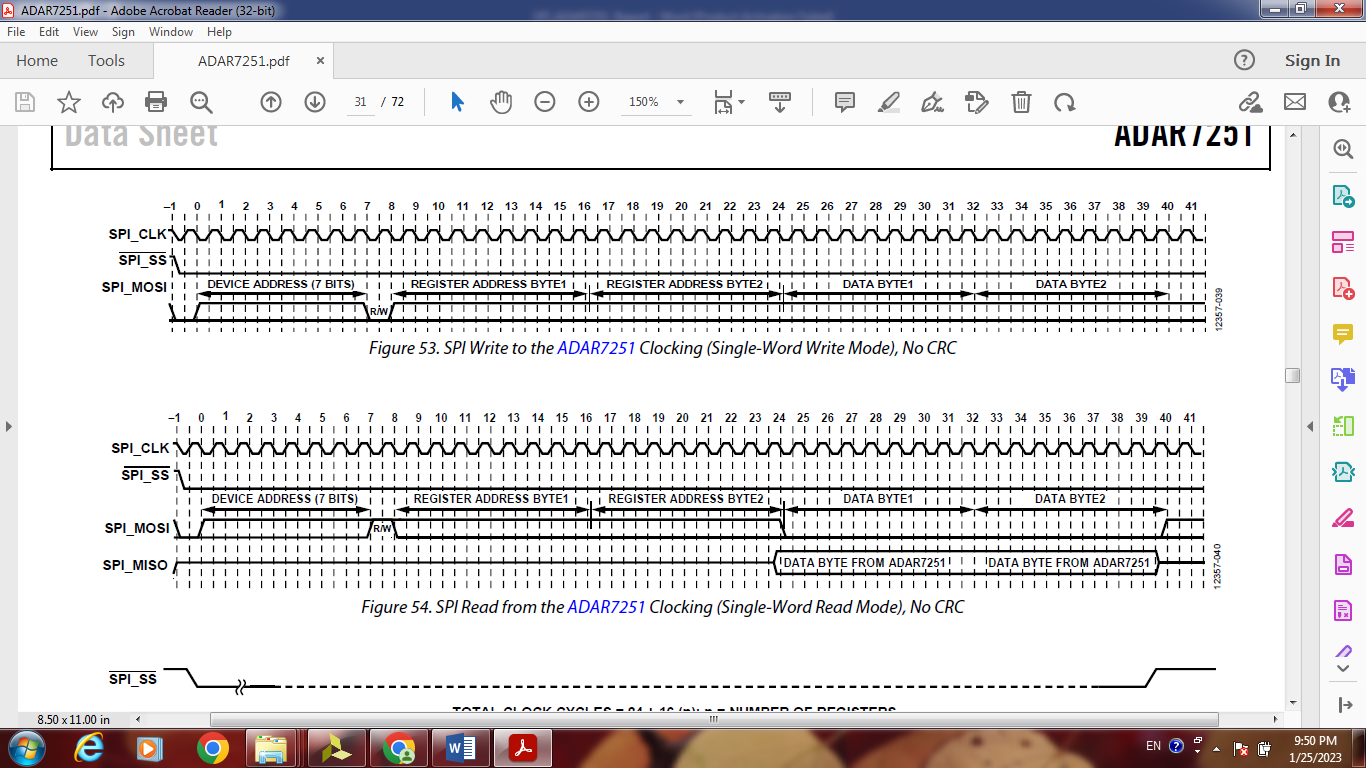
پورت کنترلی ADAR7251 از پروتکل ارتباطی SPI با 3 سیم استفاده می کند. پورت SPI رجیسترهای داخلی دستگاه را تنظیم می کند. SPI امکان خواندن و نوشتن رجیستر ها را فراهم می کند. همه رجیسترها 16 بیت عرض دارند. پورت کنترل SPI فقط از حالت Slave پشتیبانی می کند و بنابراین، برای کار کردن به Master در سیستم نیاز دارد. 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 8 نشان دهنده ی زمان بندی پروتکل ارتباطی SPI در ADAR7251 است.



شکل 12- 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 13 نشان داده شده است.

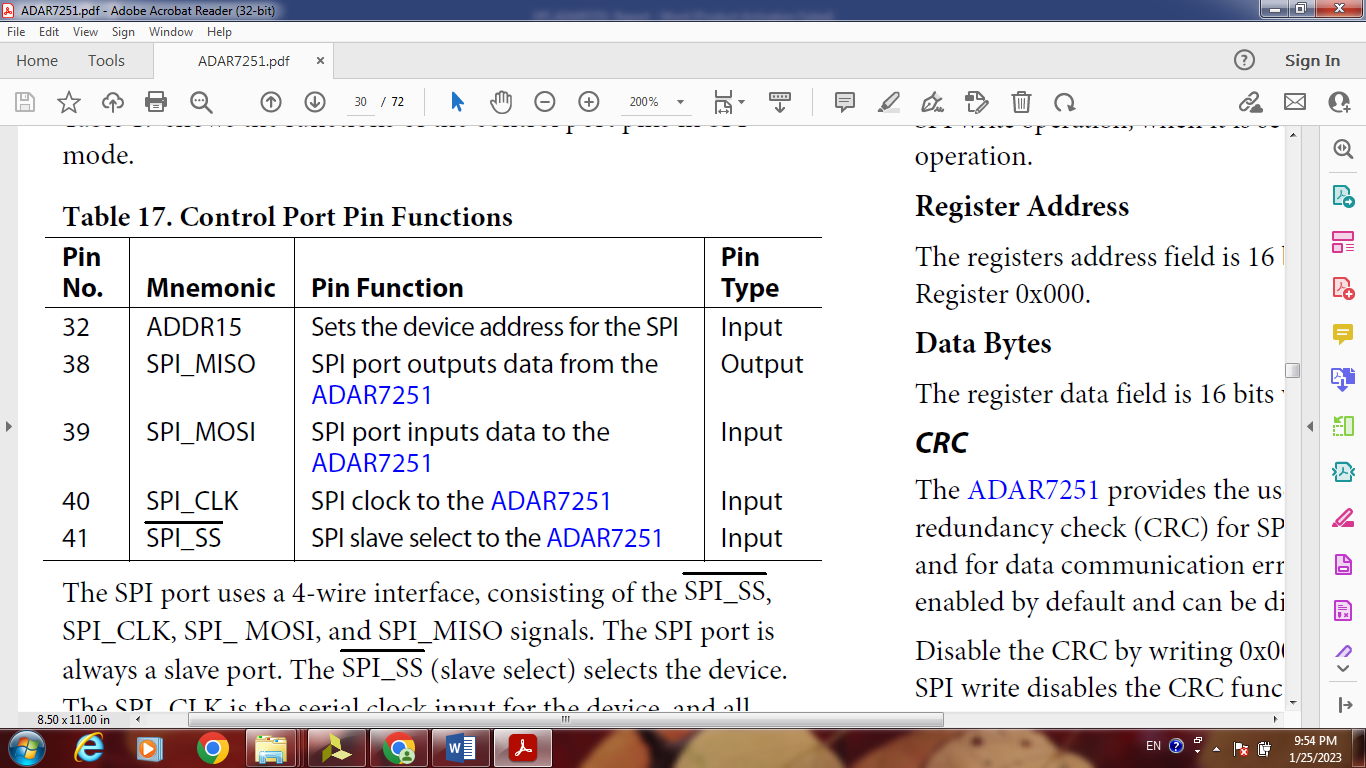




شکل 13- زمان بندی و فریم داده های ارسالی در مد SPI

جدول 2 پورت های ورودی و خروجی در حالت SPI را نشان می دهد.

جدول 2- عملکرد پورت های کنترلی در حالت SPI



1. پیاده سازی و شبیه سازی پروتکل SPI

# 1-3- 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 نوشتن بستر تست[[4]](#footnote-4) شبیه سازی شده است. با توجه به نداشتن 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 3-3 ثبت شده است.

# 2-3- 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. 7 ورودی و خروجی در قسمت Entity تعریف می شود. جدول 3 بیانگر تعریف پورت های تعریف شده است.

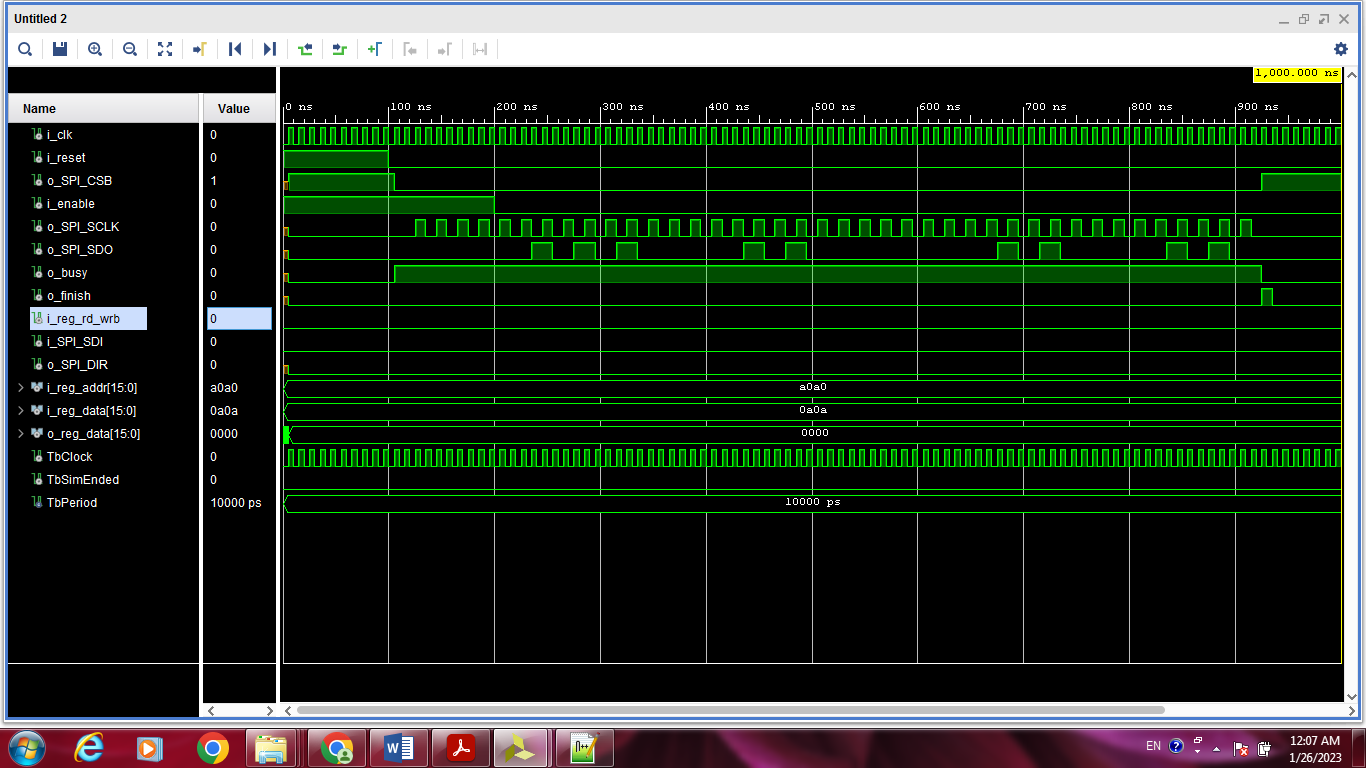
جدول 3-پورت های ورودی و خروجی های تعریف شده برای پروتکل SPI

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| توضیحات | In/out | اسم پورت | توضیحات | In/out | اسم پورت |
| داده ورودی | In | i\_SPI\_SDI | داده خروجی | Out | O\_SPI\_SDO |
| کلاک ورودی | In | i\_clk | کلاک SPI | Out | O\_SPI\_SCLK |
| ریست ورودی | In | i\_reset | انتخاب Chip | Out | O\_SPI\_CSB |
| نشانگر فعال بودن | In | i\_enable | نشان دهنده مسیر ورودی و خروجی | Out | O\_SPI\_DIR |
| خواندن یا نوشتن | In | i\_reg\_rd\_wrb | رجیستر داده | Out | O\_reg\_data |
| رجیستر آدرس | In | i\_reg\_addr | نشانگر مشغول بودن ADC | Out | O\_busy |
| رجیستر داده | In | i\_reg\_data | نشانگر پایان عملیات | Out | O\_finish |

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 39 ام است. توجه شود که در هر فریم مطابق شکل13، 5 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 0 شود، برنامه به حالت Last\_Bit منتقل می شود و نشانگر o\_finish به نشانه ی اتمام فریم برابر با 1 و نشانگر o\_busy به نشانه ی اتمام فعالیت برابر با 0 می شود.

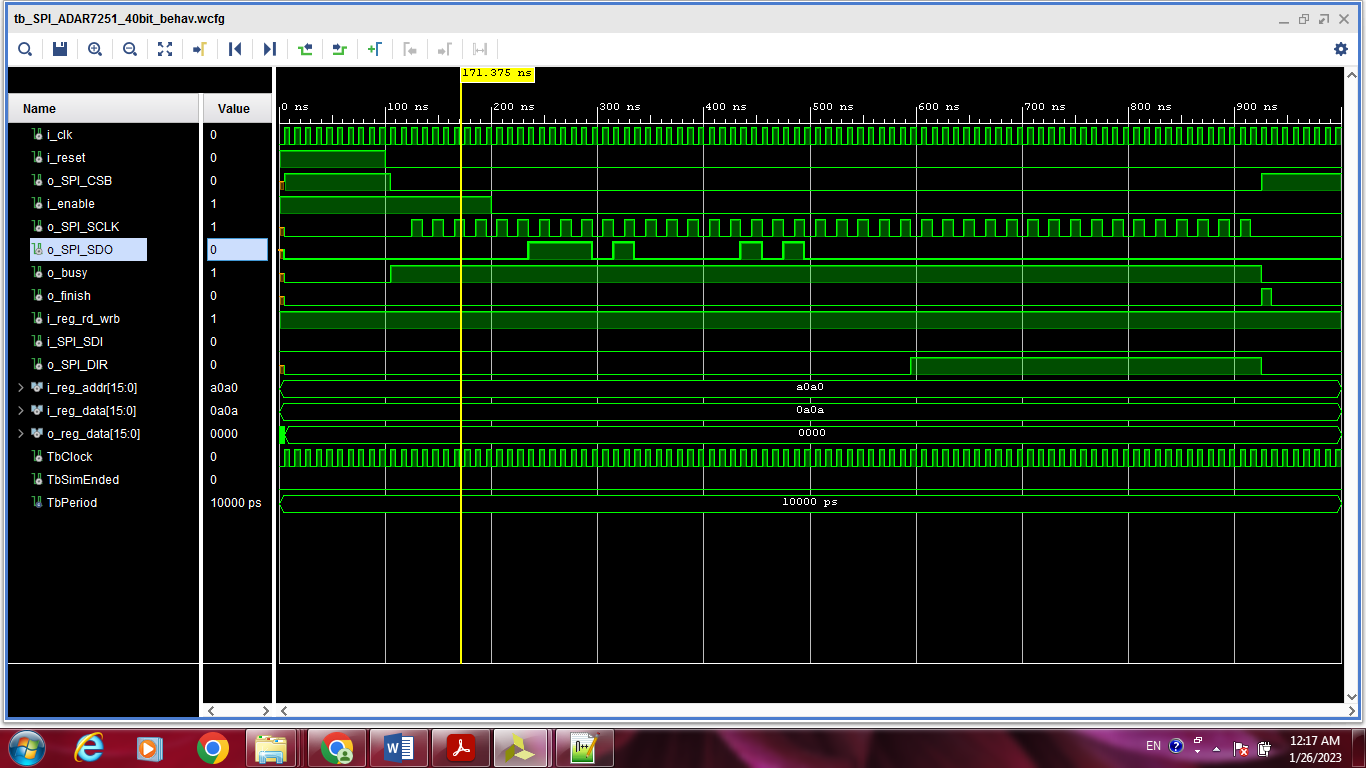
# 3-3- نتایج شبیه سازی

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 14 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 14 نشان دهنده ی شبیه سازی این موضوع در نرم افزاری Vivado است.



شکل 14- شبیه سازی پروتکل ارتباطی SPI در حالت write

برای شبیه سازی حالت خواندن در پروتکل SPI کافی است تا رجیستر مربوطه را برابر با 1 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 0 تعریف شده است.



شکل 15- 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](http://www.melec.ir)
6. معرفی شرکت Xilinx و خانواده FPGA های Xilinx، [www.taksuntech.ir](http://www.taksuntech.ir)

پيوست‌:

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;  else  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  #101;  i\_reset = 0;  #50;  i\_enable =1;  #10;  i\_enable =0;  // Add stimulus here  end  always #5 i\_clk = ~i\_clk;  endmodule |

1. - Test Bench [↑](#footnote-ref-1)
2. -PLL (Phased Lock Loop) [↑](#footnote-ref-2)
3. - Anti-Aliasing [↑](#footnote-ref-3)
4. - Test Bench [↑](#footnote-ref-4)