

دانشكده مهندسي برق

**پروژه درس VHDL**

**شبیه سازی پروتکل انتقال داده ADF4360-5**

استاد :

دکتر ستار میرزا کوچکی

دانشجو:

بهداد صادقیان پور

دی 1401



فهرست مطالب

[فصل 1: 1](#_Toc125484661)

[مقدمه 1](#_Toc125484662)

[1‏-‏1‏-‏ مقدّمه 2](#_Toc125484663)

[فصل 2: 3](#_Toc125484664)

[پروتکل ارتباطی SPI و انواع آن 3](#_Toc125484665)

[2‏-‏1‏-‏ ارتباط سریال SPI و انواع آن 4](#_Toc125484666)

[2‏-‏1‏-‏1‏-‏ ارتباط سریال SPI از نوع 4-wire 4](#_Toc125484667)

[2‏-‏1‏-‏2‏-‏ ارتباط سریال SPI از نوع 3-wire 6](#_Toc125484668)

[فصل 3: 8](#_Toc125484669)

[پروتکل ارتباطی ADF4360-5 8](#_Toc125484670)

[3‏-‏1‏-‏ مقدمه 9](#_Toc125484671)

[3‏-‏1‏-‏1‏-‏ ویژگی‌های ADF4360-5 9](#_Toc125484672)

[3‏-‏1‏-‏ نمودار مشخصات زمانی 11](#_Toc125484673)

[3‏-‏1‏-‏2‏-‏ نتیجه‌گیری 14](#_Toc125484674)

[3‏-‏2‏-‏ توضیحات پیرامون رجیسترهای ADF4360-5 14](#_Toc125484675)

[فصل 4: 16](#_Toc125484676)

[نرم‌افزار ISE 16](#_Toc125484677)

[4‏-‏1‏-‏ مقدمه 17](#_Toc125484678)

[4‏-‏2‏-‏ آشنایی کوتاه با محیط ISE 18](#_Toc125484679)

[4‏-‏3‏-‏ توصیف و طراحی کلی ماژول SPI به زبان VHDL 21](#_Toc125484680)

[4‏-‏3‏-‏2‏-‏ پیاده‌سازی اصولی کلاک با استفاده از DCM 23](#_Toc125484681)

[4‏-‏3‏-‏3‏-‏ بررسی ماژول SPI در نرم‌افزار ISP 28](#_Toc125484682)

[4‏-‏3‏-‏4‏-‏ کدهای ماژول SPI 41](#_Toc125484683)

[فصل 5: 46](#_Toc125484684)

[شبیه‌سازی و نتایج 46](#_Toc125484685)

[نمودارها و کدهای پروژه 47](#_Toc125484686)

فهرست اشکال

[شکل (1‏-‏1) توصیف نحوة ارتباط قطعه جانبی با قطعه دیجیتالی اصلی 2](#_Toc125483913)

[شکل (2‏-‏1) نمایش کلی ارتباط بین دستگاه‌های الکترونیکی در SPI نوع 4-wire 5](#_Toc125483914)

[شکل (2‏-‏2) نمایش کلی ارتباط بین دستگاه‌های الکترونیکی در SPI نوع 3-wire 7](#_Toc125483915)

[شکل (3‏-‏1) بلوک دیاگرام عملکردی در synthesizer 10](#_Toc125483916)

[شکل (3‏-‏2) نمودار مشخصات زمانی 12](#_Toc125483917)

[شکل (3‏-‏3) نمودار زمانی برنامه‌ریزی لچ ها 15](#_Toc125483918)

[شکل (4‏-‏1) فضای کلی نرم‌افزار ISE 18](#_Toc125483919)

[شکل (4‏-‏2) پنجرة new project wizard 18](#_Toc125483920)

[شکل (4‏-‏3) پنجرة new project wizard 19](#_Toc125483921)

[شکل (4‏-‏4) پنجره اصلی نرم‌افزار ISE 19](#_Toc125483922)

[شکل (4‏-‏5) پنجرة new source wizard 20](#_Toc125483923)

[شکل (4‏-‏6) محیط کدنویسی به زبان VHDL 20](#_Toc125483924)

[شکل (4‏-‏7) طرح کلی طراحی ماژول SPI 3-wire 21](#_Toc125483925)

[شکل (4‏-‏8) پنجرة مربوط به IP Core 24](#_Toc125483926)

[شکل (4‏-‏9) پنجره Clocking Wizard 24](#_Toc125483927)

[شکل (4‏-‏10) مشخصات کلاک ورودی 25](#_Toc125483928)

[شکل (4‏-‏11) پنجره مشخصات کلاک خروجی 26](#_Toc125483929)

[شکل (4‏-‏12) پنجره سیگنال‌های کنترلی بلوک DCM 26](#_Toc125483930)

[شکل (4‏-‏13) پنجره مربوط به نام پورت‌های بلوک DCM 27](#_Toc125483931)

[شکل (5‏-‏1) خروجی شماره 1 48](#_Toc125483932)

[شکل (5‏-‏2) خروجی شماره 2 49](#_Toc125483933)

[شکل (5‏-‏3) خروجی شماره 3 50](#_Toc125483934)

[شکل (5‏-‏4) خروجی شماره 4 50](#_Toc125483935)

[شکل (5‏-‏5) خروجی شماره 5 51](#_Toc125483936)

[شکل (5‏-‏6) خروجی شماره 6 51](#_Toc125483937)

[شکل (5‏-‏7) خروجی شماره 7 52](#_Toc125483938)

[شکل (5‏-‏8) خروجی شماره 8 52](#_Toc125483939)

[شکل (5‏-‏9) خروجی شماره 9 53](#_Toc125483940)

[شکل (5‏-‏10) خروجی شماره 10 53](#_Toc125483941)

[شکل (5‏-‏11) تایمینگ برنامه ریزی Latch ها 54](#_Toc125483942)

[شکل (5‏-‏12) خروجی شماره 11 54](#_Toc125483943)

[شکل (5‏-‏13) خروجی شماره 12 55](#_Toc125483944)

[شکل (5‏-‏14) خروجی شماره 13 55](#_Toc125483945)

[شکل (5‏-‏15) خروجی شماره 14 56](#_Toc125483946)

[شکل (5‏-‏16) خروجی شماره 15 56](#_Toc125483947)

[شکل (5‏-‏17) خروجی شماره 16 57](#_Toc125483948)

فهرست جداول

[جدول (3‏-‏1) توضیحات عملکرد پایه‌های ADF4360-5 11](#_Toc125483949)

[جدول (3‏-‏2) مقادیر زمانی پالس‌ها 12](#_Toc125483950)

[جدول (3‏-‏3) مقادیر خازن و گپ زمانی 15](#_Toc125483951)

[جدول (4‏-‏1) توضیح عملکرد پورت‌های ماژول SPI 22](#_Toc125483952)

[جدول (4‏-‏2) مشخصات پورت‌های ماژول SPI 30](#_Toc125483953)



مقدمه

* 1. مقدّمه

در ابتدا برای آشنایی کلی با روند پروژه و کارهای صورت‌گرفته شده، توضیح مختصری در مورد پیاده‌سازی اینترفیس و مدارات اینترفیسی داده می‌شود. در حالت کلی به مجموعه تمهیدات سخت‌افزاری و نرم‌افزاری پیاده‌سازی شده، برای دو قطعه دیجیتال، اینترفیس می‌گویند.

Interface

Synthesizer

FPGA

توصیف نحوة ارتباط قطعه جانبی با قطعه دیجیتالی اصلی

در شکل بالا synthesizer (در ادامه توضیح داده خواهد شد)، به‌عنوان یک peripheral یا قطعه جانبی در نظر گرفته شده است. هدف ما از این پروژه توصیف ارتباط بین FPGA و synthesizer است. برای ایجاد این ارتباط باید تمهیداتی در نظر گرفته شود. برای مثال پین‌هایی از هر دو قطعه دیجیتالی به یکدیگر متصل شود و سیگنال‌هایی بین این دو ردوبدل شود. منبع اصلی این طراحی دیتاشیت قطعه جانبی است. در این دیتاشیت اطلاعات کاملی از پروتکل ارتباطی، نحوة اتصال پایه‌ها، وظیفة هر پایه و ... نوشته شده است. برای آشنایی بیشتر در فصل بعدی توضیح مختصری در مورد قطعه جانبی این پروژه یعنی ADF4360-5 که یک نوع synthesizer است داده می‌شود تا درک بهتر و بیشتری از آنچه در ادامه بیان می‌کنیم داشته باشیم.



پروتکل ارتباطی SPI و انواع آن

* 1. ارتباط سریال SPI و انواع آن

در حالت کلی دو نوع ارتباط SPI[[1]](#footnote-2) داریم، 3-wire و 4-wire که در اینجا به توضیح هرکدام می‌پردازیم. در ابتدا به توضیح SPI از نوع 4-wire می‌پردازیم که باعث می‌شود درک بهتری نسبت به 3-wire داشته باشیم.

* + 1. ارتباط سریال SPI از نوع 4-wire

پروتکل SPI یک ارتباط داده سریال هماهنگ است که در حالت کاملاً دوطرفه عمل می‌کند. دراین‌ارتباط یک قطعه دیجیتالی به‌عنوان master عمل می‌کند که وظیفة کنترل و صدور فرمان برای برقراری ارتباط را دارد. قطعه یا قطعه‌های دیجیتالی دیگر به‌عنوان slave عمل می‌کنند که وظیفة دریافت فرمان صادر شده از master و دریافت اطلاعات ارسال شده را دارند.

در حالت کلی SPI چهار سیگنال دارد که در زیر به توضیح هر یک از آن‌ها می‌پردازیم.

1. SCLK : سیگنال کلاک سریال (معمولاً توسط master صادر می‌شود)
2. MOSI : Master Out – Slave In
3. MISO : Master In – Slave Out
4. SS : انتخاب قطعه الکترونیکی جانبی برای برقراری ارتباط

سیگنال SS در صورت‌های دیگری هم از جمله LE[[2]](#footnote-3) و CS[[3]](#footnote-4) می‌تواند ظاهر شود. نکته‌ای که باید به آن توجه داشت این است که سیگنال SS معمولاً به‌صورت فعال-پایین (Active Low) از آن استفاده می‌شود.

* + - 1. عملکرد

در ارتباط SPI می‌توان یک master با یک یا چند slave ارتباط برقرار کرد. انتخاب اینکه چه slave ای باید انتخاب شود توسط پایه کنترلی SS صورت می‌گیرد. در زیر نمای کلی این ارتباط را مشاهده می‌کنید.

MASTER

Slave 1

MSIO

MOSI

SCLK

MSIO

MOSI

SCLK

SS

SS

SCLK

MOSI

Slave 2

SS

MSIO

MOSI

SCLK

Slave 3

SS

MSIO

نمایش کلی ارتباط بین دستگاه‌های الکترونیکی در SPI نوع 4-wire

برای شروع تبادل داده‌ها، دستگاه master نخست پالس ساعت را با فرکانسی کمتر یا برابر با حداکثر مقداری که دستگاه slave پشتیبانی می‌کند تنظیم می‌کند که معمولاً در حد چند مگاهرتز است. سپس یک سیگنال صفر منطقی از خط انتخاب قطعه به salve می‌فرستد. برای این به دستگاه صفر می‌فرستیم که سطح فعال آن صفر است (Active Low) یعنی سطح خاموش آن، یک منطقی است. اگر نیازی به وقفه بود (مانند در قطعه‌های تبدیل آنالوگ به دیجیتال) دستگاه master باید حداقل به آن میزان صبر کند و سپس پالس ساعت را به Slave بفرستد.

در هر دورة پالس ساعت SPI، یک تبادل داده کاملاً دوطرفه رخ می‌دهد master . داده را از خط MOSI می‌فرستد و Slave نیز آن را از همان خط دریافت می‌کند.slave داده را از خط MISO می‌فرستد و master نیز آن را از همان خط دریافت می‌کند.

در بالا توضیح مختصری دررابطه‌با نحوه عملکرد پروتکل SPI از نوع 4-wire داده شد. نکته‌ای که باید به آن توجه داشت این است که پروتکل ارتباطی موردنیاز ما برای ADF4360-5 ، SPI و از نوع 3-wire است که به دلیل شباهت این دو نوع با یکدیگر ابتدا 4-wire توضیح داده شد. در بخش بعدی به توضیح 3-wire

می‌پردازیم.

* + 1. ارتباط سریال SPI از نوع 3-wire

همان‌طور که در بخش قبلی گفته شد برای ارتباط 4-wire از 4 سیگنال یا پایه استفاده می‌کنیم. همان‌طور که از اسم 3-wire مشخص است برای استفاده از این نوع SPI نیاز به 3 سیگنال یا پایه داریم.

در این پروتکل ارتباطی، ارسال داده و دریافت آن فقط بر روی یک‌پایه صورت می‌گیرد که در قطعات جانبی مختلف این پایه نام متفاوتی دارد؛ ولی معمولاً با نام SDIO[[4]](#footnote-5) آن را می‌شناسیم.

معمولاً به ارتباط 3-wire ارتباط Bidirectional یا دوطرفه نیز گفته می‌شود. در زیر نمای کلی از این پروتکل را مشاهده می‌کنید.

MASTER

Slave 1

SDIO

SCLK

SDIO

SCLK

SS

SS

SCLK

SDIO

Slave 2

SS

SDIO

SCLK

Slave 3

SS

نمایش کلی ارتباط بین دستگاه‌های الکترونیکی در SPI نوع 3-wire

نکته‌ای که باید به آن توجه داشت وضعیت پایة SDIO در حالت ارسال و یا دریافت اطلاعات است که باید باتوجه‌به دیتاشیت قطعه الکترونیکی در نظر گرفته شود. نکتة دیگر این است که بر خلاف 4-wire که یک ارتباط full-duplex بود، 3-wire یک پروتکل half-duplex است.



پروتکل ارتباطی ADF4360-5

* 1. مقدمه

در این فصل به بررسی ADF4360-5 به‌عنوان یک synthesizer می‌پردازیم. سینتزایزر یک قطعه الکترونیک است که محدوده‌ای از فرکانس‌ها را توسط یک فرکانس مرجع تولید می‌کند. از این قطعه در گیرنده‌های رادیویی، تلویزیون‌ها، موبایل‌ها و ... استفاده می‌شود.

سینتزایزر ها به سه دستة کلی تقسیم‌بندی می‌شوند که به ترتیب عبارت‌اند از:

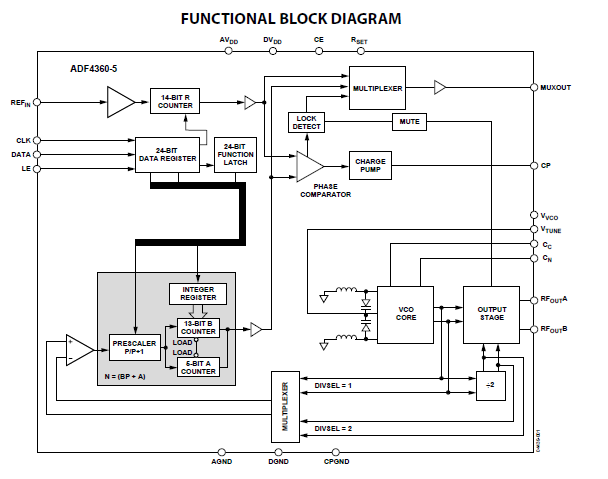
1. direct analog synthesizer
2. direct digital synthesizer
3. indirect digital synthesizer

نوع سوم بیان شده به دو قسمت integer-N و fractional-N تقسیم‌بندی می‌شود که ADF4360-5 از نوع integer-N است.

* + 1. ویژگی‌های ADF4360-5

1. محدوده فرکانسی خروجی: 1200 MHz to 1400 MHz
2. ولتاژ تغذیه: 3.0 تا 3.6 ولت
3. استفاده از پروتکل SPI از نوع 3-wire

در ادامه تصویر کلی از عملکرد واحدهای مختلف synthesizer آورده شده است که بیانگر وظیفة هر واحد بکار گرفته شده در synthesizer است.



بلوک دیاگرام عملکردی در synthesizer

همان‌طور که در بالا گفته شد، پروتکل ارتباطی ADF4360-5، SPI و از نوع 3-wire است. که موضوع اصلی پروژه هم دررابطه‌با پیاده‌سازی این پروتکل است و در این قسمت از توضیح مداری سینتزایزر خودداری شده است؛ بنابراین تمرکز اصلی این فصل مربوط به نمودارهای زمانی و مسائل مربوط به پیاده‌سازی این پروتکل است.

* 1. نمودار مشخصات زمانی

یکی از مهم‌ترین بخش‌ها برای طراحی و پیاده‌سازی پروتکل ارتباطی نمودار مشخصات زمانی پالس‌ها است که اطلاعاتی مهمی از جمله فرکانس پالس ساعت و عرض پالس‌های مختلفی که برای برقراری ارتباط نیاز داریم را در اختیار ما قرار می‌دهد. قبل از اینکه به سراغ توصیف پارامترهای زمانی بپردازیم، به معرفی پایه‌های مربوطه برای برقراری ارتباط 3-wire می‌پردازیم.

همان‌طور که در بخش‌های قبلی گفته شد برای برقراری ارتباط 3-wire نیازمند سه سیگنال اصلی هستیم. در اینجا این سه سیگنال با نام‌های Clock ، DATA و LE مشخص شده است که در جدول زیر به تعریف هرکدام از این پایه‌ها پرداخته‌ایم.

نکتة مهمی که باید به آن توجه داشت این است که پایة DATA در ADF4360-5 فقط وظیفه ی دریافت اطلاعات از master را دارد و دیتا و یا بیتی ارسال نمی کند.

توضیحات عملکرد پایه‌های ADF4360-5

|  |  |  |
| --- | --- | --- |
| توضیحات | Mnemonic | شمارة پایه |
| ورودی کلاک. از این پایه برای دریافت سیگنال پالس ساعت تولید شده توسط master استفاده می‌شود. دیتاهای دریافتی با هر پالس ساعت در رجیسترهای مخصوصی ذخیره می‌شوند. | CLK | 17 |
| از این پایه برای دریافت اطلاعات استفاده می‌شود. در ابتدا بیت‌هایی که دریافت می‌شوند به‌صورت بیت‌های باارزش مکانی بالاتر است. (MSB) | DATA | 18 |
| زمانی که این پایه صفر فعال است عملیات انتقال دیتا صورت می‌گیرد.  در آخر برای ذخیره‌سازی بیت‌های دریافتی در رجیستر مربوطه که با بیت‌های کنترلی انتخاب می‌شود، این پایه باید به یک منطقی برسد. | LE | 19 |

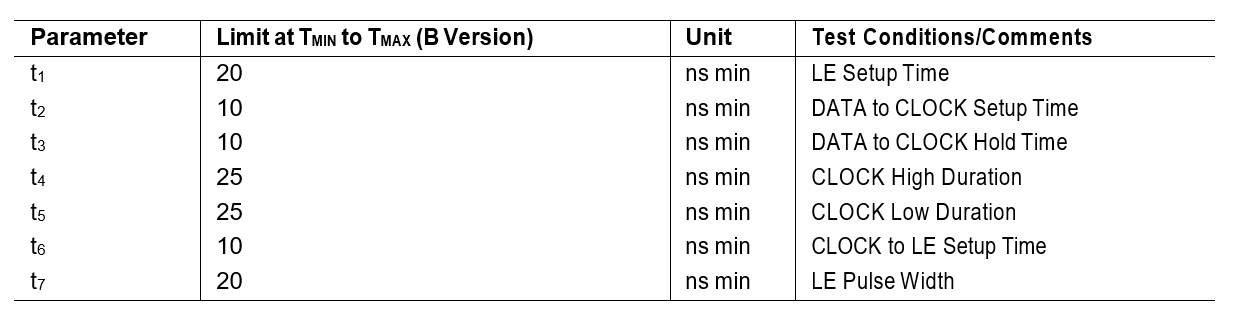
بنابراین، با استفاده از جدول بالا به تمام ویژگی‌های پایه‌هایی که برای برقراری ارتباط 3-wire استفاده می‌شوند پی می‌بریم. شکل (2-3) به توضیح زمانی شکل موج‌های این پایه‌ها برای برقراری ارتباط 3-wire می‌پردازد.



نمودار مشخصات زمانی

اطلاعات زمانی شکل موج‌های شکل (3-2) در جدول آمده است.

مقادیر زمانی پالس‌ها



در ابتدا به بررسی سیگنال کلاک می‌پردازیم. همان‌طور که در شکل 3-2 مشخص است. دوره تناوب سیگنال پالس ساعت برابر است با:

بنابراین، داریم:

باتوجه‌به رابطة دوره تناوب و فرکانس داریم:

بنابراین، باتوجه‌به روابط بالا فرکانس کلاک ADF4360-5 برابر با 20 MHz است.

پارامتر زمانی دیگری که مورد بررسی قرار می‌دهیم، LE Setup Time یا است. در واقعیت پالس های مربعی در لحظه به صفر و یک منطقی تبدیل نمی شوند، این تغییر حالت پالس ها با تاخیر همراه است. برای اینکه این تاخیر در پیاده سازی های مدارات دیجیتالی مشکل زا نشود تاخیری، برای اطمینان از حالت اصلی پالس، در نظر می گیرند. برای مثال در اینجا t1 نشان دهنده ی این است که بعد از مدت زمان 20 نانو ثانیه بعد صفر شدن پایه ی LE، سیگنال پالس ساعت می بایست شروع به نوسان کند.

همچنین همین حالت هم برای بیت‌های ارسالی بر روی پایه Data باید در نظر گرفته شود که در جدول بالا با پارامتر t2 نشان‌داده‌شده است.

تعریف دیگری که در اینجا داریم hold time است. hold time در مدارات دیجیتالی عبارت است از حداقل مدت زمانی که می‌بایست دادة ورودی در حالت پایدار خود باقی بماند تا این داده به‌عنوان دادة معتبر از دید کلاک در نظر گرفته شود را hold time می‌گوییم. در اینجا hold time در نظر گرفته شده برای داده برابر است با 10 ns این به این معنا است که داده حداقل به مدت 10 ns باید در حالت پایدار و معتبر خود باقی بماند و بعد سیگنال کلاک لبة مربوطه را فعال کند؛ بنابراین پارامترهای زمانی جدول 3-2 تعریف شدند.

نکتة بعدی که باید در نمودارهای زمانی به آن توجه کرد سیگنال داده است، همان‌طور که در نمودار مشخص است، اولین بیت ارسالی از طرف master به‌عنوان بیت باارزش مکانی بیشترین (MSB) در نظر گرفته می‌شود و بعد با هر لبة بالا رنده کلاک بیت بعدی باارزش مکانی کمتر ارسال می‌شود. پارامترهای t2 و t3 مربوط به زمان‌های hold time و setup time است که در بالا توضیح داده شد.

پارامتر زمانی t7 مدت زمانی است که سیگنال LE باید در یک منطقی باقی بماند و بعد از آن دوباره می‌توانیم عملیات ارسال (برنامه‌ریزی رجیستر بعدی) را شروع کنیم.

* + 1. نتیجه‌گیری

برای برقراری ارتباط با ADF4360-5 ابتدا پایة LE را در صفر منطقی قرار می‌دهیم، سپس به‌اندازه t1 در این حالت باقی بماند و بعد از آن سیگنال کلاک فعال شود و به تعداد 24 لبة بالارونده کلاک بیت‌های داده با رعایت hold time و setup time ارسال شوند و در آخر هم برای ذخیره‌سازی داده‌ها در لچ مربوطه و شروع دوبارة عملیات ارسال پایه LE به مدت t7 در وضعیت یک منطقی باقی بماند و دوباره به حالت صفر منطقی بازگردد.

* 1. توضیحات پیرامون رجیسترهای ADF4360-5

پیاده‌سازی این پروتکل ارتباطی به‌منظور برنامه‌ریزی رجیسترهای داخلی ADF4360-5 صورت می‌گیرد، باتوجه‌به اینکه این پروژه در حد شبیه‌سازی است، و بر روی برد عملی سنتز و پیاده‌سازی نمی‌شود، بنابراین ترتیب بیت های داده ارسالی و تعریف هر کدام از این بیت ها برای ما چندان اهمیتی ندارد تمرکز اصلی این پروژه بر روی پیاده سازی پروتکل ارتباطی 3-wire است، اما در ادامه به توضیح مختصر رجیستر

های ADF4360-5 می پردازیم.

ADF4360-5 به طور کلی شامل 3 رجیستر به نام های Control Latch ، N Counter Latch و

R Counter Latch است. هرکدام از این لچ ها 24 بیتی است، بنابراین دادة ارسالی ما برای برنامه‌ریزی این لچ ها هم همان‌طور که در نمودار زمانی مشخص شد 24 بیتی است. ترتیب پیشنهادی ارائه شده در دیتاشیت به‌صورت زیر است.

1. R Counter Latch
2. Control Latch
3. N Counter Latch

نکته‌ای دیگر که باید به آن توجه کرد نمودار برنامه‌ریزی این لچ ها است که در زیر به آن اشاره شده است.

POWER-UP

CLOCK



DATA



**CONTROL LATCH DATA**

**N COUNTER LATCH DATA**

**R COUNTER LATCH DATA**

LE

REQUIRED INTERVAL CONTROL LATCH WRITE TO N COUNTER LATCH WRITE

04439-020

نمودار زمانی برنامه‌ریزی لچ ها

همان‌طور که مشاهده می‌شود بعد از برنامه‌ریزی Conntrol Latch باید یک گپ زمانی در نظر گرفت و سپس N Counter Latch را برنامه‌ریزی کنیم. مدت‌زمان این فاصله زمانی توسط مقدار خازن Cn مشخص می‌شود. جدول زیر بیانگر مقادیر استاندارد برای این مسئله است.

مقادیر خازن و گپ زمانی

|  |  |  |
| --- | --- | --- |
| **CN Value** | **Recommended Interval Between Control Latch and N Counter Latch** | **Open-Loop Phase Noise at 10 kHz Offset** |
| 10 μF  440 nF | ≥ 5 ms  ≥ 600 μs | −88 dBc  −87 dBc |

برای مثال با مقدار خازن 10 میکروفاراد مقدار این فاصلة زمانی بیشتر از 5ms است.

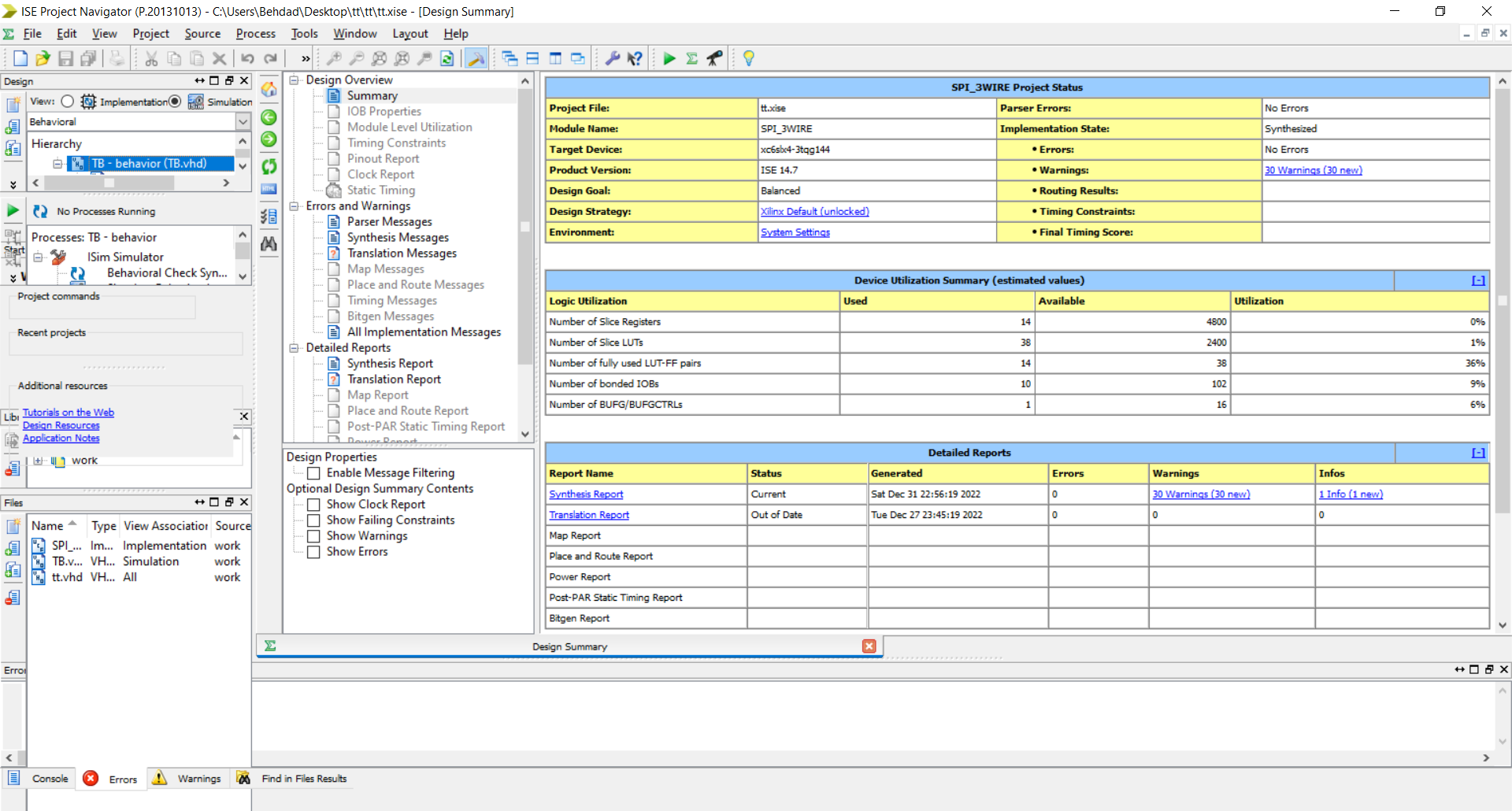


نرم‌افزار ISE

* 1. مقدمه

نرم افزار ISE Design Suite محصولی از کمپانی Xilinx است که برای بهینه سازی نیرو و هزینه، از طریق بهره وری طراحی بیشتر، تولید شده است. به کمک نرم‌افزار ISE Design Suite می‌توانید تمام مراحل طراحی و پیاده‌سازی شامل ورود طرح، شبیه‌سازی، سنتز، جانمایی و مسیریابی رو انجام بدهیم. بعد از آن فایل پیکره‌بندی را ایجاد کنیم و FPGA را پروگرام کنیم .به کمک نرم‌افزار ISE می‌توانیم انواع تحلیل‌های زمانی و توان مصرفی را برای طرحی که پیاده‌سازی کردیم انجام بدیم. این نرم‌افزار IP Coreها یا کدهای از پیش نوشته شده زیادی را در اختیارتان قرار می‌دهد که می‌توانیم به کمک آنها مدارات بزرگ را سریع‌تر طراحی و تست کنیم.

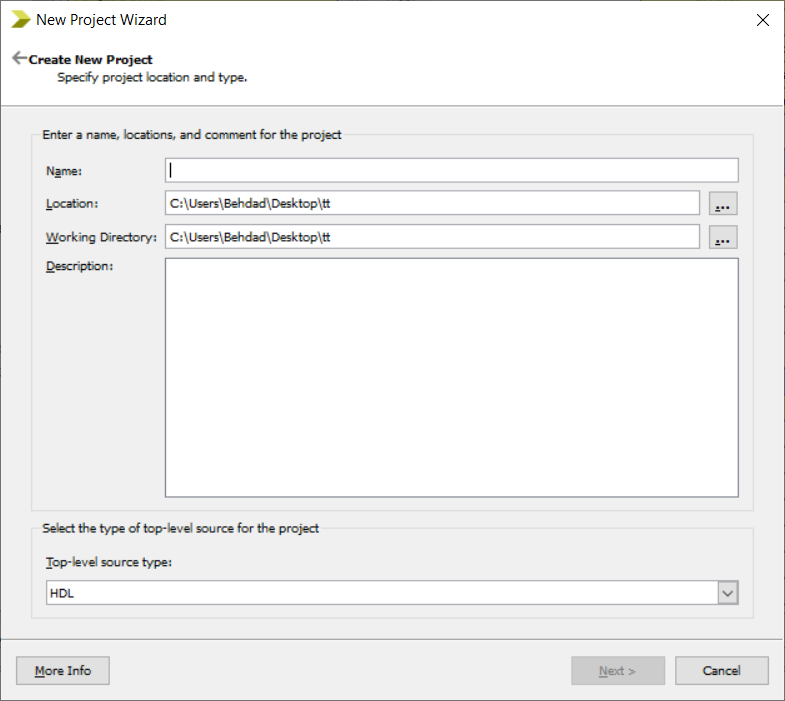
* 1. آشنایی کوتاه با محیط ISE



فضای کلی نرم‌افزار ISE

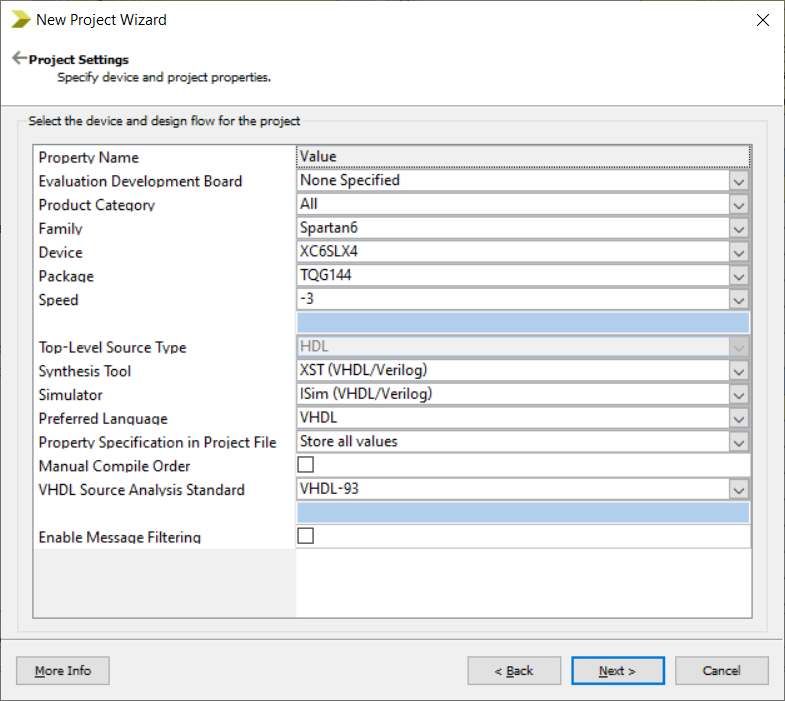
پنجرة Designe summary توضیحات مختصری دربارة پروژة ساخته شده ارائه می‌دهد.

برای ساخت پروژة جدید از مسیر file/new project وارد پنجرة new project wizard می‌شویم.



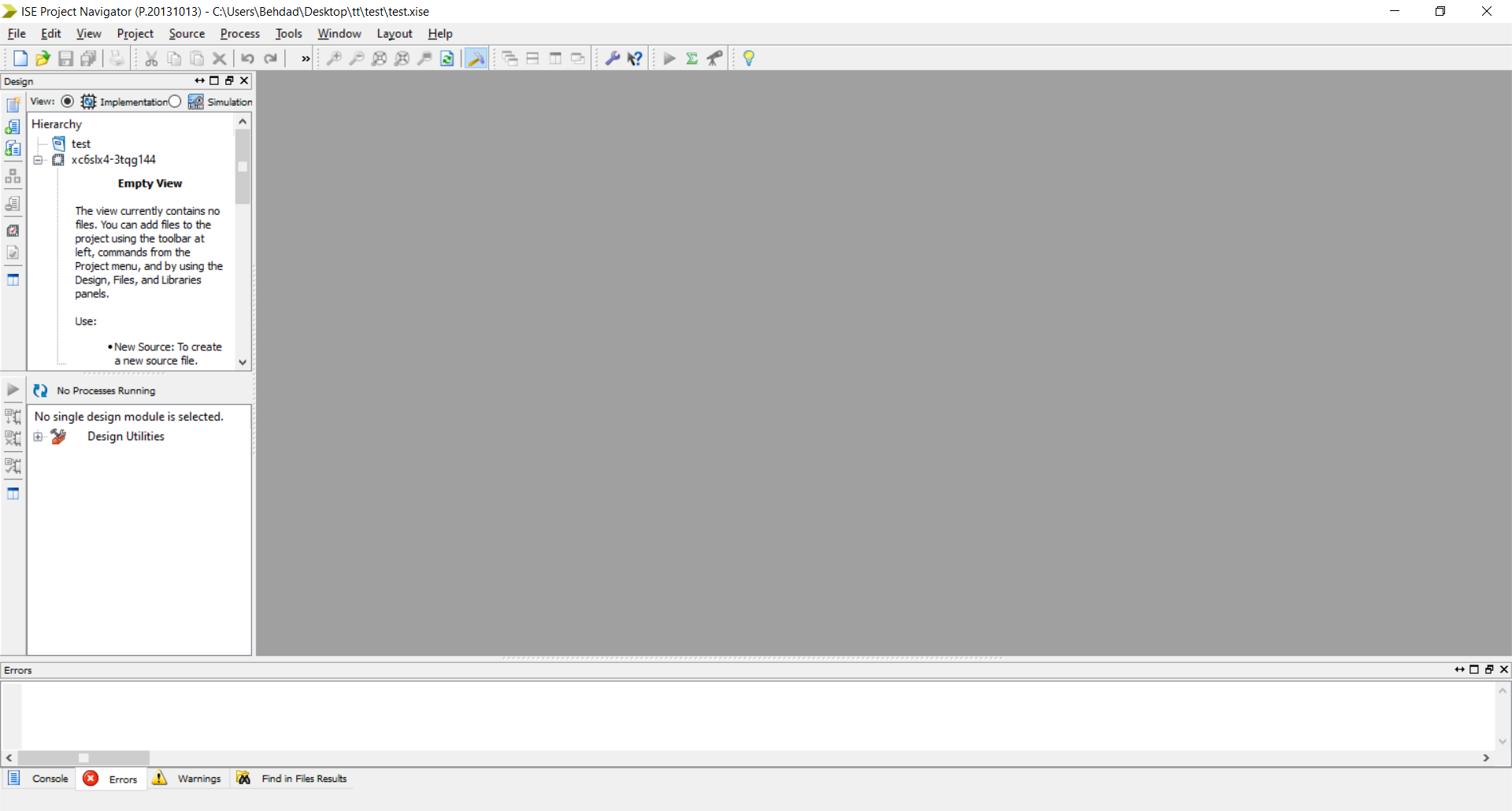
پنجرة new project wizard

در این پنجره نام پروژه، آدرس ذخیره‌سازی و توضیحات پروژه را مشخص می‌کنیم.



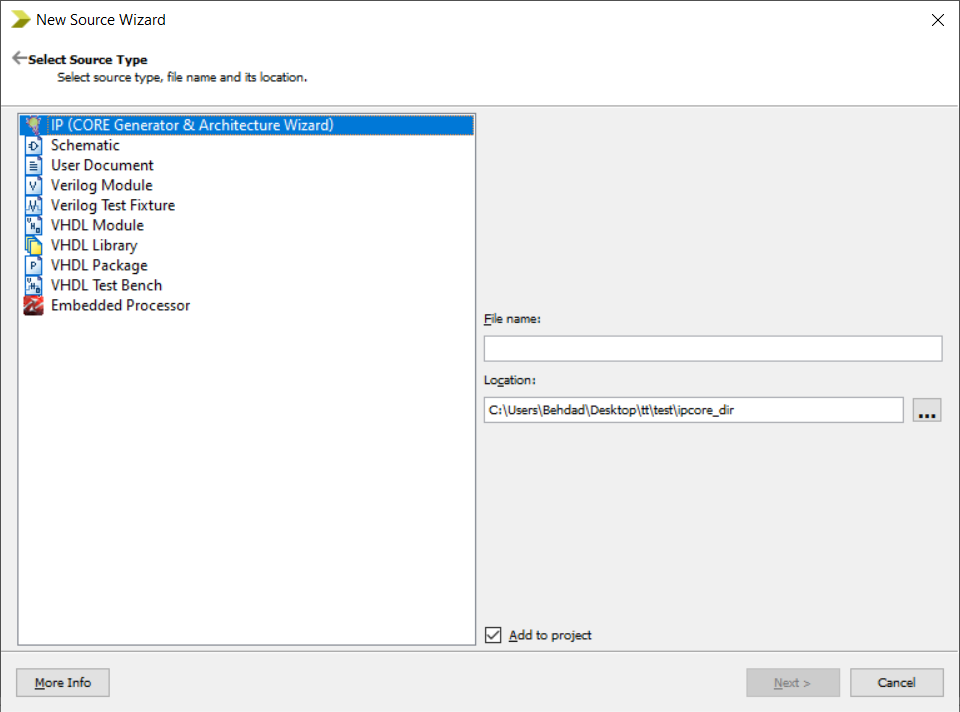
پنجرة new project wizard

در این پنجره زبان برنامه‌نویسی، مدل و خانوادة FPGA موردنظر و ... را مشخص می‌کنیم.



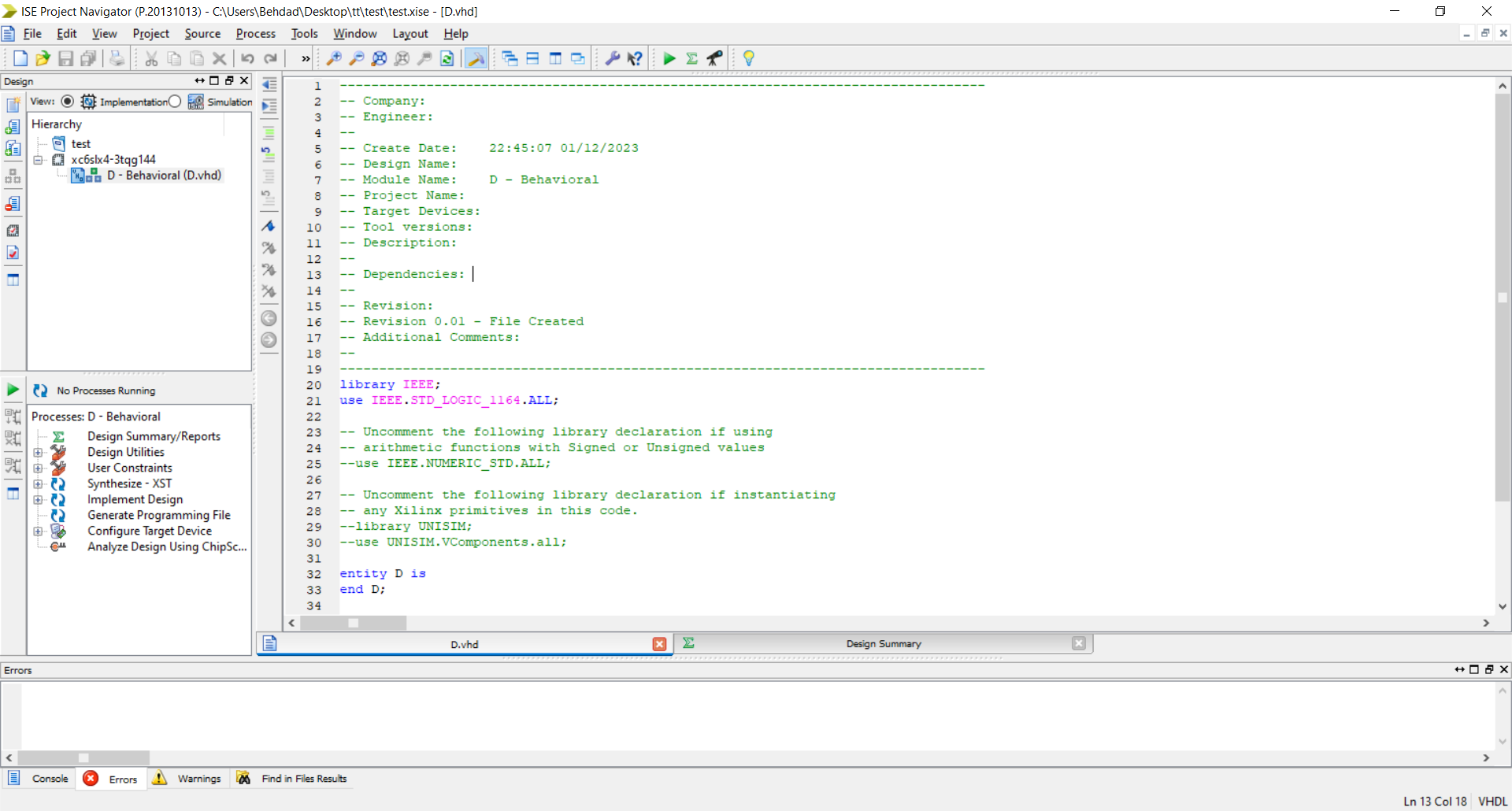
پنجره اصلی نرم‌افزار ISE

با کلیک راست کردن بر روی قسمت hierarchy و انتخاب گزینة new source وارد پنجره زیر می‌شویم.



پنجرة new source wizard

با انتخاب گزینة VHDL Module فایلی با پسوند .vhd نرم‌افزار ایجاد می‌کند که محیط کدنویسی را برای ما فراهم می‌کند.



محیط کدنویسی به زبان VHDL

* 1. توصیف و طراحی کلی ماژول SPI به زبان VHDL

در ابتدا به توصیف ماژول SPI طراحی شده می‌پردازیم و هریک از ویژگی‌های آن توضیح می‌دهیم.

SPI 3-wire Module

Busy

DATA

SCLK\_20Mhz

LoadEnable

R\_W

SyncReset

ENABLE

SCLK\_Pha

SCLK\_Pol

Clock

DATA

LE

Clock

Synthesizer

ADF4360-5

OSC

طرح کلی طراحی ماژول SPI 3-wire

همان‌طور که در شکل بالا مشاهده می‌کنید، ماژول SPI طراحی شده، دارای 6 ورودی و 4 خروجی است که در جدول زیر عملکرد و وظیفه هرکدام از پورت‌ها توضیح داده شده است.

توضیح عملکرد پورت‌های ماژول SPI

|  |  |  |
| --- | --- | --- |
| **توضیحات** | **نوع پورت** | **نام پورت** |
| پایه ورودی کلاک ماژول است که پالس ساعت را از مولد کلاک دریافت می‌کند. | ورودی | Clock |
| مقدار اولیه کلاک SPI توسط این پایه مشخص می‌شود. | ورودی | SCLK\_Pol |
| فاز کلاک خروجی تولید شده برای پروتکل SPI توسط این پایه مشخص می‌شود. | ورودی | SCLK\_Pha |
| برای شروع کار ماژول SPI استفاده می‌شود. اگر این پایه یک منطقی باشد باتوجه‌به شرایط پروتکل، ماژول SPI شروع به کار می‌کند. | ورودی | ENABLE |
| پایه سنکرون با کلاک ریست برای ریست‌کردن ماژول SPI استفاده می‌شود.  این پایه active high است. | ورودی | SyncReset |
| برای انتخاب مُد نوشتن یا خواندن استفاده شده است.  اگر یک منطقی باشد از این ماژول برای نوشتن (ارسال داده) و اگر صفر منطقی باشد برای خواندن اطلاعات است. در اینجا فقط برای ارسال داده استفاده می‌شود؛ بنابراین مقدار آن همیشه برابر با یک منطقی در نظر گرفته می‌شود. | ورودی | R\_W |
| پورت خروجی داده پروتکل SPI که خروجی در نظر گرفته می‌شود. | خروجی | DATA |
| پورت LE برای انتخاب قطعه جانبی مدنظر که در اینجا ADF4360-5 است.  این پایه active low است. | خروجی | LoadEnable |
| پورت خروجی برای نشان‌دادن وضعیت ماژول SPI اگر در حال ارسال داده باشد مقدار این پورت برابر با یک منطقی می‌شود در غیر این صورت صفر منطقی است. | خروجی | Busy |
| پورت خروجی مربوط به کلاک پروتکل SPI است که در اینجا باتوجه‌به  دیتاشیت ADF4360-5 فرکانس این کلاک 20 مگاهرتز در نظر گرفته شده است. | خروجی | SCLK\_20MHz\_IBUFG |

* + 1. پیاده‌سازی اصولی کلاک با استفاده از DCM

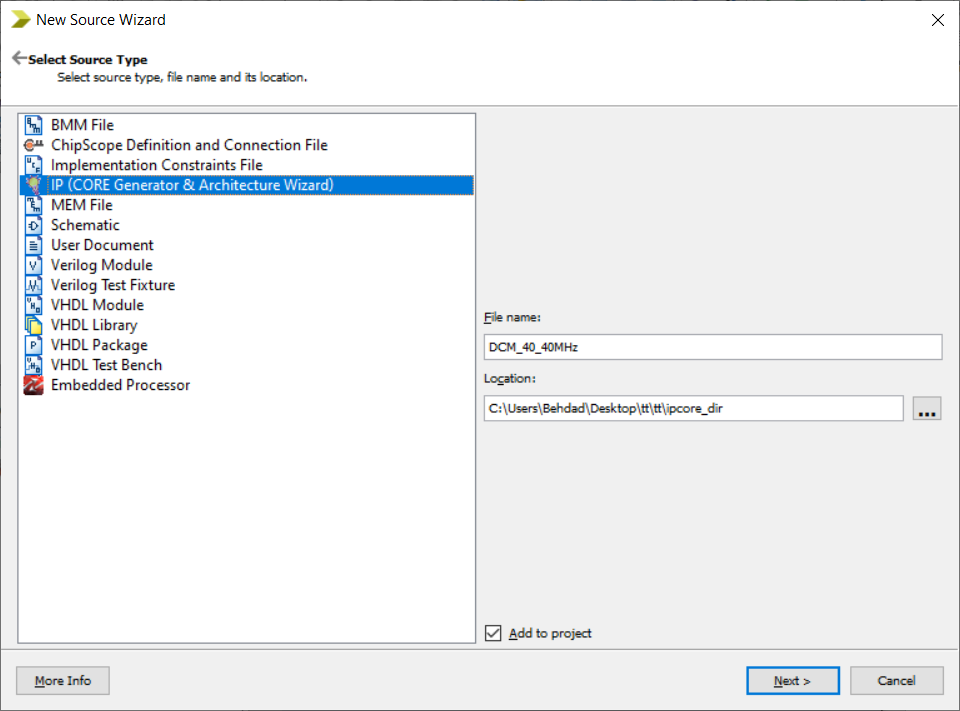
DCM بلوکی است سخت‌افزاری که در FPGAهای شرکت Xilinx تعبیه شده است. با استفاده از این کلاک می‌توانیم فرکانس‌های کلاک جدیدی را از کلاک اصلی مدار، تولید کنیم. فرض کنیم کلاک اسیلاتور روی بورد برابر با 100 مگاهرتز است، اما فرکانس کلاک موردنیاز برای ما، 20 مگاهرتز است؛ ازاین‌رو بهتر است برای تولید کلاک 20 مگاهرتز از بلوک سخت‌افزاری DCM استفاده کنیم. هر FPGA دارای تعدادی از بلوک‌های DCM است، هر بلوک DCM ورودی‌ای به‌عنوان کلاک اصلی دارد و خروجی آن کلاک موردنظر ما برای استفاده از درون FPGA است.

نکته‌ای که باید به آن توجه داشت مزیت بلوک DCM تنها تغییر فرکانس کلاک نیست؛ بلکه از این بلوک می‌توان برای کاهش جیتر[[5]](#footnote-6) کلاک استفاده کرد. به همین علت توصیه می‌شود برای تولید کلاک از بلوک DCM استفاده کرد، هرچند که نیازی به تغییر فرکانس در کلاک نباشد. در این پروژه فرض بر این است که کلاک اصلی مدار (کلاک اسیلاتور) برابر است با 40 مگاهرتز و ورودی کلاک ماژول SPI مدنظر ما هم برابر با 40 مگاهرتز است که برای تولید 40 مگاهرتز از بلوک DCM استفاده شده است. بلوک DCM تمام بافرهای مخصوص کلاک که لازم است در مدار داشته باشند، به طور خودکار در مدار قرار می‌دهد، بنابراین نیازی به تعریف بافر نیست.

برای استفاده از DCM می‌بایست از IP Core ها استفاده کرد.

برای استفاده از IPCoreها ابتدا بر روی محیط hierarchy کلیک راست کرده، سپس گزینة

New Source را انتخاب می‌کنیم بعد از آن از پنجرة New Source Wizard گزینة IP را انتخاب می‌کنیم و سپس یک نام برای این IP انتخاب می‌کنیم. شکل زیر بیانگر مراحل بالا است.

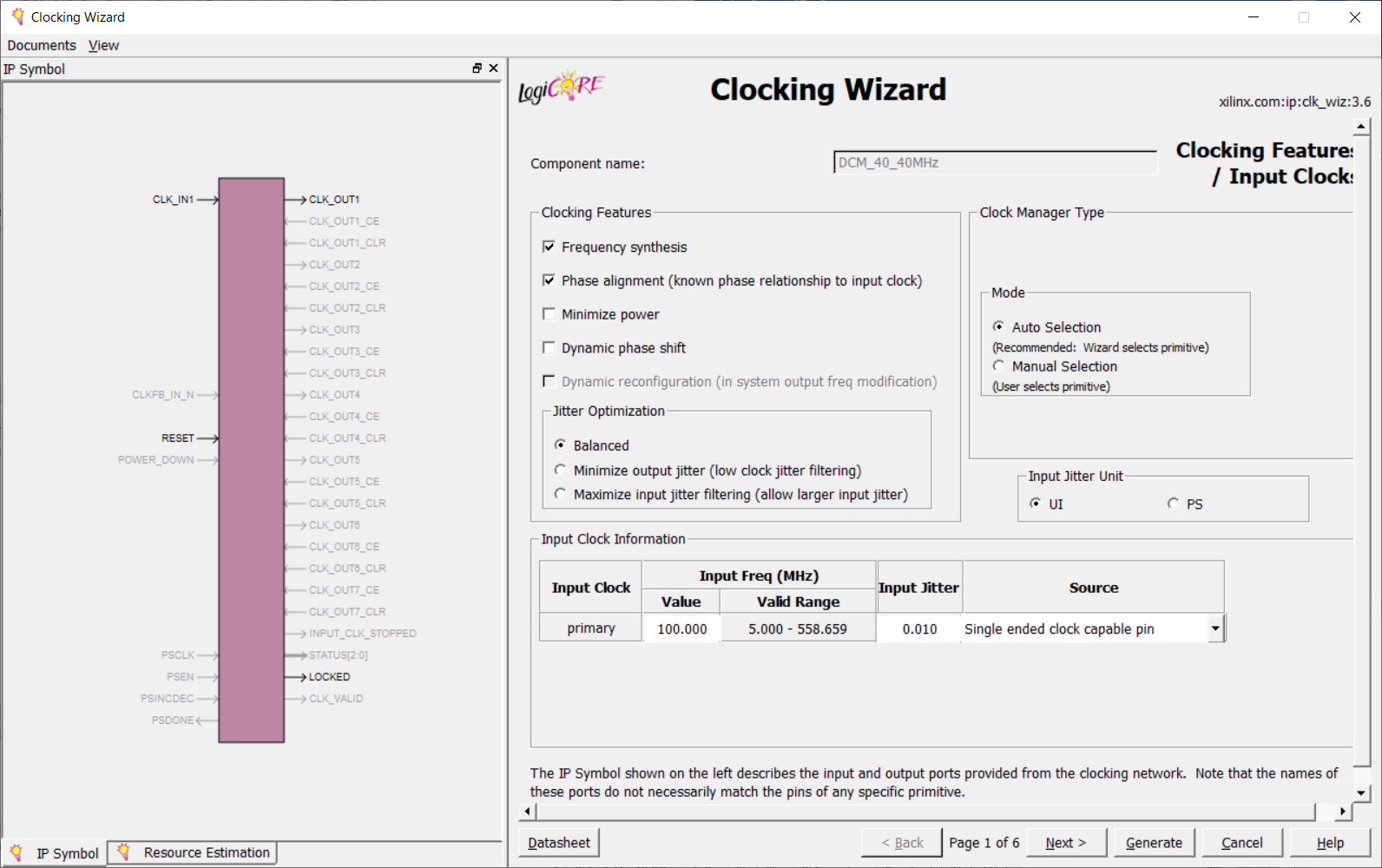


پنجرة مربوط به IP Core

سپس گزینة Next را می‌زنیم، در پنجرة باز شده مراحل زیر را طی می‌کنیم.

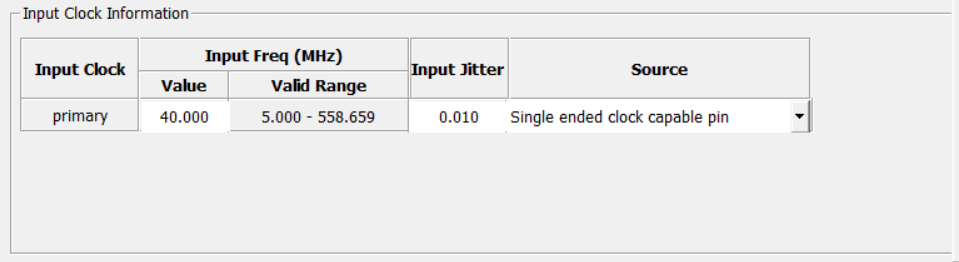
FPGA Features and Design > Clocking > Clocking Wizard ، و سپس گزینة Next را می‌زنیم.

پس از ساخت IPCore مدنظر، پنجرة مربوط به تنظیمات Clocking Wizard باز می‌شود.



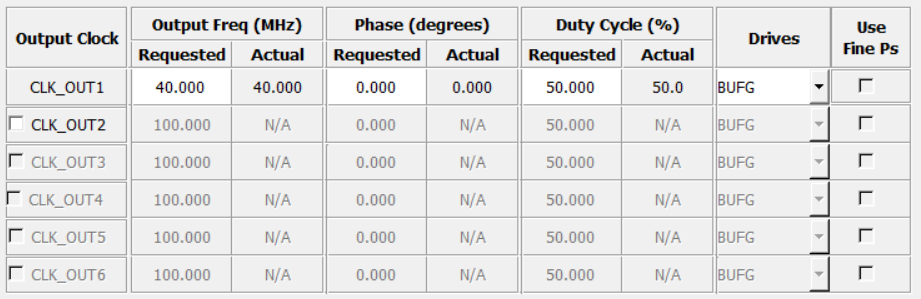
پنجره Clocking Wizard

در این پنجره تنظیمات مختلفی وجود دارد که مربوط به تولید است. برای مثال frequency synthesis برای تولید فرکانس جدید ، Minimize Power برای کاهش توان مصرفی ، phase alignment برای تعین فاز کلاک استفاده می‌شود. در قسمت Input Clock Information فرکانس کلاک ورودی را انتخاب می‌کنیم در اینجا فرکانس ورودی 40 مگاهرتز در نظر گرفته شده است. برای تعیین جیتر کلاک می‌توان به دو صورت عمل کرد حالت اول UI و حالت دوم PS یا مخفف pico Second است که می‌توان زمان جیتر را وارد کرد. در اینجا حالت پیش‌فرض نرم‌افزار در نظر گرفته شده است. در قسمت Source، منبع کلاک DCM را انتخاب می‌کنیم که 4 حالت مختلف دارد. اگر منبع ورودی به‌صورت تک‌ورودی در نظر گرفته شده باشد از گزینة single ended clock capable pin ، اگر به‌صورت تفاضلی وارد شده باشد از گزینة Differential clock capable pin ، اگر ورودی از خارج FPGA وارد نمی‌شود و از داخل FPGA تأمین می‌شود و قبل از اینکه به ورودی DCM وارد شود، از یک بافر عبور داده شده است از گزینه BufG استفاده می‌کنیم و اگر در این حالت از بافر استفاده نشده باشد از گزینه No Buffer استفاده می‌کنیم. در اینجا از گزینة Single ended clock capable pin استفاده می‌کنیم.



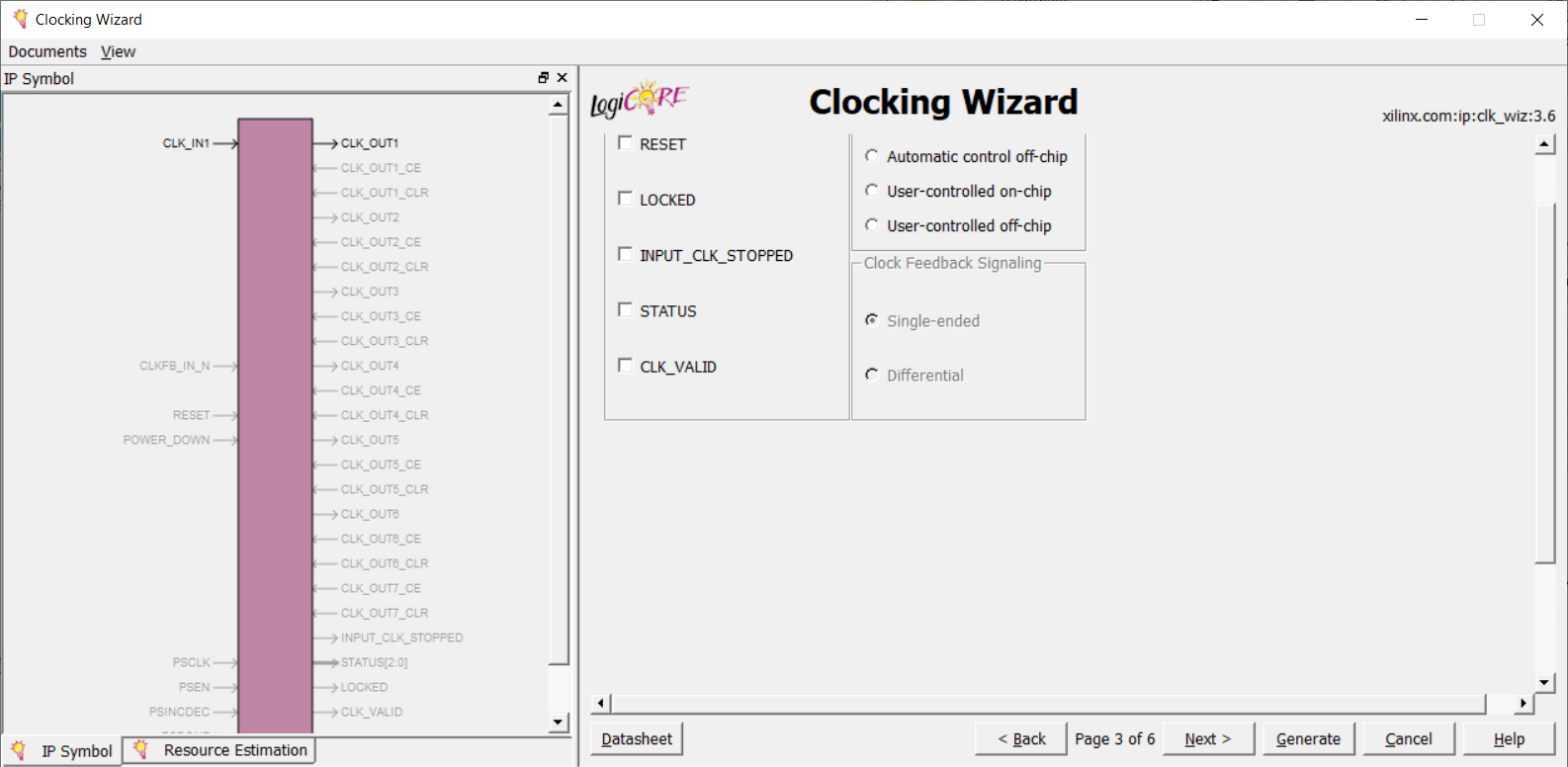
مشخصات کلاک ورودی

در پنجرة بعدی مشخصات کلاک خروجی را در نظر می‌گیریم. ابتدا مقدار فرکانس خروجی را وارد می‌کنیم. در اینجا 40 مگاهرتز وارد شده است، فاز کلاک خروجی را صفر در نظر می‌گیریم. Duty Cycle پالس را هم برابر با 50 درصد در نظر می‌گیریم.



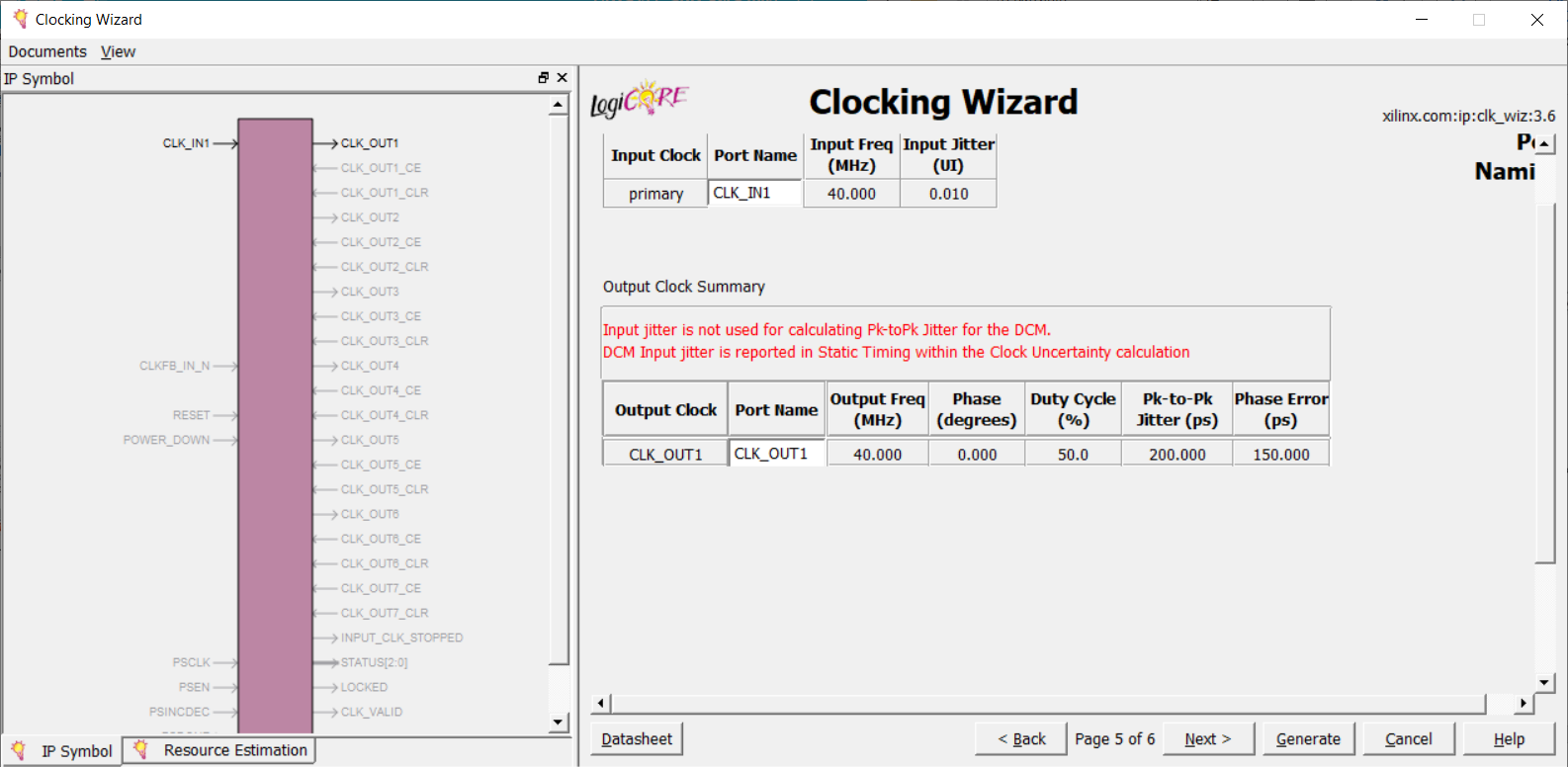
پنجره مشخصات کلاک خروجی

پنجرة بعدی مربوط است به سیگنال‌های کنترلی بلوک DCM که در این جا هیچ‌کدام از این سیگنال‌ها را فعال نمی‌کنیم.



پنجره سیگنال‌های کنترلی بلوک DCM

در پنجرة بعدی نام پورت‌های ورودی و خروجی بلوک DCM را می‌توانیم تغییر دهیم. در اینجا مقادیر پیش‌فرض در نظر گرفته شده است.



پنجره مربوط به نام پورت‌های بلوک DCM

بعدازاین پنجره بر روی گزینة FINISH کلیک می‌کنیم و تنظیمات بلوک DCM به پایان می‌رسد.

بعد از ساخته‌شدن بلوک DCM از گزینة view HDL instantiation مراحل مربوط به نمونه‌سازی آن در کد اصلی را انجام می‌دهیم.

کدهای مربوط به تعریف بلوک DCM:

**component** **DCM\_40\_40MHz**

**port**

(-- Clock in ports

CLK\_IN1 : **in** **std\_logic**;

-- Clock out ports

CLK\_OUT1 : **out** **std\_logic**

);

**end** **component**;

کدهای بالا را در قسمت decleration معماری تعریف می‌کنیم ، سپس عملیات Port Map را انجام می‌دهیم.

ورودی بلوک DCM باید پورت کلاک باشد و خروجی آن سیگنال میانی ای که کلاک 40 مگاهرتز تولیدی DCM است؛ بنابراین کلاکی برای ماژول SPI از آن استفاده می‌کنیم همین سیگنال میانی است.

کدهای مربوط به عملیات Port Map:

DCM\_BLOCK : DCM\_40\_40MHz

**port** **map**

(-- Clock in ports

CLK\_IN1 => Clock,

-- Clock out ports

CLK\_OUT1 => Clock\_40MHz);

بنابراین، کلاکی که برای ماژول SPI از آن استفاده می‌کنیم CLOCK\_40MHz است. نکته‌ای که باید به

* + 1. بررسی ماژول SPI در نرم‌افزار ISP
       1. تعریف کتابخانه

در این ماژول از کتابخانه‌های استاندار مربوط به IEEE استفاده شده است. کتابخانة UNSIM برای استفاده از اعداد signed و unsigned و همچنین استفاده از IBUFG است.

کدهای مربوط به تعریف کتابخانه:

--DEFINE LIB--

**library** **IEEE**;

**USE** **IEEE.STD\_LOGIC\_1164.ALL**;

**USE** **IEEE.NUMERIC\_STD.ALL**;

--For IBUFG--

**LIBRARY** **UNISIM**;

**use** **UNISIM.vcomponents.all**;

--

* + - 1. تعریف موجودیت

کدهای مربوط به موجودیت ماژول SPI:



در کدهای بالا، ابتدا موجودیتی به نام SPI\_3WIRE تعریف شده است و سپس در بدنة آن به تعریف پورت‌های موردنظر پرداخته‌ایم.

در بخش generic ، متغیر DataWidth\_CMD برای مشخص‌کردن تعداد داده‌های ارسالی تعریف شده است. متغیر SlavesNum برای مشخص‌کردن تعداد synthesizerهای متصل شده به FPGA تعریف شده است، در اینجا تعداد برابر با یک در نظر گرفته شده است.

در بخش Port به تعریف پورت‌ها پرداخته شده است. در جدول زیر توضیحات مربوط به هر پایه نوشته شده است.

مشخصات پورت‌های ماژول SPI

|  |  |  |  |
| --- | --- | --- | --- |
| **توضیحات** | **نوع** | **جهت** | **نام پورت** |
| کلاک اصلی ماژول SPI | STD\_LOGIC | ورودی | Clock |
| تعیین پلاریتة کلاک | STD\_LOGIC | ورودی | SCLK\_Pol |
| تعیین فاز کلاک | STD\_LOGIC | ورودی | SCLK\_pha |
| تعیین وضعیت فعال و غیرفعال بودن ماژول SPI .  فعال: 1 غیرفعال: 0 | STD\_LOGIC | ورودی | Enable |
| ریست سنکرون با کلاک  فعال: 1 غیرفعال: 0 | STD\_LOGIC | ورودی | SyncReset |
| تعیین حالت خواندن یا نوشتن. در اینجا تنها از حالت نوشتن استفاده می‌شود.  نوشتن: 1 خواندن: 0 | STD\_LOGIC | ورودی | R\_W |
| پایه ارسال داده ماژول SPI | STD\_LOGIC | خروجی | Data |
| پایه Load Enable برای انتخاب Synthesizer موردنظر | STD\_LOGIC\_VECTOR | خروجی | LoadEnable |
| پایه‌ای برای نمایش وضعیت ماژول SPI | STD\_LOGIC | خروجی | Busy |
| کلاک خروجی ماژول SPI | STD\_LOGIC | خروجی | SCLK\_20MHz\_IBUFG |

* + - 1. تعریف معماری

ابتدا معماری‌ای با نام Behaviroal مربوط به موجودیت SPI\_3WIRE تعریف می‌کنیم.

سپس به تعریف سیگنال‌های میانی می‌پردازیم. در اینجا به‌ازای هر پورت تعریف شده در موجودیت، یک سیگنال میانی تعریف شده است. علت این کار را در ادامه به آن می‌پردازیم.

تعریف سیگنال SCLK برای تولید کلاک SPI :



تعریف سیگنال‌های میانی:



تعریف تایپ‌های موردنیاز:



در این قسمت دو نوع تایپ تعریف شده است. تایپ FSM که به ماشین حالت مربوط می‌شود تایپ دوم مربوط می‌شود به رجیسترهای داخلی ADF4360-5 که در اینجا R , C و N به ترتیب مخفف رجیستر ControlLatch ، R Latch و N Latch است. در ادامه از هر کدام سیگنالی با این تایپ ها تعریف کرده ایم.

تعریف رجیسترهای ADF4360-5 :



برای برنامه‌ریزی رجیسترهای ADF4360-5، هرکدام از این رجیسترها را به‌صورت عدد ثابت constant تعریف کرده‌ایم. هرکدام از این رجیسترها از نوع STD\_LOGIC\_VECTOR ؛ 24 بیتی و دارای مقدار اولیه‌ای دلخواه هستند. به دلیل اینکه این پروژه به‌صورت عملی بر روی بورد پیاده‌سازی نمی‌شود؛ بنابراین مقدار اولیة این رجیسترها برای ما اهمیتی ندارد و تنها برای نشان‌دادن درستی ارسال این داده‌ها به آن مقدار داده شده است.

تعریف بلوک DCM:



همان‌طور که گفته شد برای کلاک ورودی مدار از بلوک DCM استفاده شده است. این بلوک DCM توسط IPCoreها ساخته شده است که در بخش‌های قبلی به توضیح آن پرداخته‌ایم.

بلوک DCM آخرین قسمتی بود که در decleration معماری behavioral تعریف کردیم. بعد از آن وارد بدنة اصلی معماری می‌شویم. بعد از کلمة begin معماری به تعریف واحدهای دیگر می‌پردازیم.

نمونه‌سازی از IBUFG:



در این ماژول برای ساخت کلاک SPI از سیگنالی با نام SCLK استفاده کردیم. برای اینکه ویژگی‌هایی بر روی این سیگنال صورت بگیرد تا بتوان از آن به‌عنوان کلاک استفاده کرد بهتر است که از ویژگی IBUFG استفاده کرد. ورودی این نمونه را SCLK قرار می‌دهیم و خروجی آن را SCLK\_MHz\_IBUFG قرار می‌دهیم. بنابراین کلاک SPI، پایه خروجی SCLK\_MHz\_IBUFG است.

نمونه‌سازی بلوک DCM:



ورودی این بلوک را Clock قرار می‌دهیم و خروجی آن که به‌عنوان کلاک از آن استفاده می‌کنیم سیگنالی با نام Clock\_40MHz استفاده می‌کنیم.

بنابراین، سیگنال کلاک اصلی ما در اینجا Clock\_40Mhz است.

* توجه: برای اینکه بتوانیم در شبیه‌سازی نتایج بهتری داشته باشیم، بهتر است که از سیگنال کلاک استفاده کنیم. در اینجا تنها به این نکته اشاره شده است که در واقعیت بهتر است که برای تولید کلاک از بلوک DCM استفاده کنیم.

عملیات رجیستر:



هرکدام از پورت‌های ورودی و خروجی را به جزء پورت کلاک به سیگنال میانی مربوطه متصل می‌کنیم. به این کار رجیستر کردن پورت‌ها می‌گویند. این عمل باعث می‌شود که FPGA نزدیک‌ترین و سریع‌ترین مسیر را برای هرکدام از این پورت‌ها انتخاب کند. تمام دستورات بالا در محیط concurrent معماری صورت می‌گیرد. بعد از آن به سراغ process می‌رویم.

توضیحات مربوطه به process:

در لیست حساسیت process تنها، سیگنال کلاک را می‌نویسیم، تا این process تنها به سیگنال کلاک حساس باشد.

**process**(Clock)

سپس به تعریف متغیرهایی که در این process نیاز داریم، می‌پردازیم.



در ادامه به کاربرد هریک از این متغیرها می‌پردازیم.

بعد تعریف متغیرها و بعد از کلمة begin شرط بالاروندة سیگنال کلاک را با استفاده از دستور زیر چک می‌کنیم.



سپس با به بررسی سیگنال ریست که یک سیگنال سنکرون با لبة بالارونده کلاک است می‌پردازیم.



در بلوک if-else بالا، ابتدا شرط فعال‌بودن ریست چک شده است که در صورت فعال‌بودن تمام پایه‌های ارسال و کنترلی به حالت اولیه بر می‌گردند.

در ادامه به تغییراتی که بعد از فعال‌شدن ریست رخ می‌دهد می‌پردازیم.

:Busy\_BUF برای نشان‌دادن وضعیت ماژول SPI است که در اینجا فرض شده است اگر حالت ریست فعال باشد این پایه برابر با یک منطقی شود.

:Data\_Buf به پین ارسال داده مربوط می‌شود که فرض کردیم اگر داده‌ای ارسال نشود این پین برابر با ‘z’ یا High-impedance باشد.

LoadEnable\_BUF(ADR): اگر برابر با یک باشد یعنی synthesizer ای انتخاب نمی‌شود. ADR آدرس slave مدنظر است که در اینجا به صفر در نظر گرفته شده است.

State: با استفاده از این سیگنال حالت FSM را کنترل می‌کنیم که در اینجا بعد از ریست‌شدن می‌بایست State مقدار حالت اولیه یعنی Idle را به خود بگیرد.

SCLK: مقدار اولیه کلاک را که با استفاده از سیگنال SCLK\_Pol\_BUF تعیین می‌کنیم به SCLK تخصیص می‌دهیم.

Counter: این متغیر برای کنترل تعداد بیت ارسالی صورت می‌گیرد. در اینجا چون 24 بیت ارسال می‌شود؛ بنابراین مقدار آن را برابر با 23 (از 0 تا 23) که در مجموع 24 بیت می‌شود قرار می‌دهیم. در این قسمت می‌توانستیم از متغیر DataWidth\_CMD نیز استفاده کنیم تا کدنویسی حالت عمومی‌تر پیدا کند.

بعد از حالت ریست که اولویت بالاتری داشت به بیان حالت‌های مختلف ارسال می‌پردازیم. در اینجا برای ارسال از FSM یا ماشین حالت استفاده شده است که 6 حالت مختلف دارد که به توضیح هرکدام می‌پردازیم.

حالت اولیه این ماشین برابر Idle است مقدار متغیر حالت یعنی State هم در ابتدا این حالت را دارد. در این حالت چون داده‌ای ارسال نمی‌شود؛ بنابراین فرض کردیم که پایه Data به‌صورت High-impedance است. Busy\_BUF برابر با صفر است؛ یعنی ماژول SPI آزاد است و داده‌ای ارسال نمی‌کند. مقدار Counter برابر با 23 قرار می‌گیرد به این دلیل که در ابتدا هیچ داده‌ای ارسال نشده است و تازه ماژول در حالت شروع به کار است. سپس با بلوک if-else به چک‌کردن پایة Enable می‌پردازیم. اگر Enable برابر با 1 بود آنگاه حالت State به InitDelay تغییر می‌کند، Busy\_Buf 1 می‌شود به معنای مشغول بودن، LoadEnable برابر با صفر می‌شود به معنای انتخاب slave موردنظر و در آخر فاز کلاک در بافر آن قرار می‌گیرد.

اگر پایة Enable فعال نبود حالت State همان idle باقی می‌ماند و LoadEnable برابر با 1 می‌شود تا salve مدنظر انتخاب نشود.

توضیحات بالا مربوط به بلوک زیر است.



حال به بیان حالت بعدی یعنی InitDelay می‌پردازیم.

این حالت در اصل برای رعایت Setup time و Hold time است. یعنی بعد از صفرشدن LoadEnable طبق دیتاشیت باید زمانی بعد دیتا شروع به ارسال کند؛ بنابراین این زمان را با استفاده از این حالت بررسی کرده‌ایم. کد زیر بیانگر این حالت است.



همان‌طور که مشاهده می‌شود حالت بعدی WriteStatus است که به متغیر State تخصیص‌داده‌شده است.

نکته‌ای که باید به آن توجه داشت ترتیب ارسال داده و برنامه‌ریزی رجیسترها است. باتوجه‌به دیتاشیت و مطالب بیان شده ترتیب پیشنهادی برای برنامه‌ریزی به‌صورت زیر است.

R Latch

Control Latch

N latch

یعنی ابتدا R Latch ، سپس Control Latch و در اخر هم N latch برنامه‌ریزی شود.

این ترتیب به‌صورت متغیرهایی از جنس Boolean کنترل شده است. que متغیر کنترلی از جنس Boolean است که سه مقدار کلی می‌تواند داشته باشد. چون در حالت قبل یعنی Idle متغیر que برابر با R بود؛ بنابراین در یک کلاک بعد از آن یعنی حالت InitDelay در بلوک case که مربوط به ارسال داده می‌شود ابتدا بیت‌های مربوط به R Latch ارسال می‌شود.

نکته‌ای که باید به آن توجه کنیم این است که تمامی مقادیر سیگنال‌هایی که در Process مقدار جدید برای آن‌های تخصیص داده می‌شود در یک کلاک بعد مقدار سیگنال تغییر می‌کند. (پایان Process)

بنابراین، اولین بیت هر رجیستر در حالت InitDelay ارسال می‌شود؛ چون در یک کلاک بعد یعنی زمانی که State در حالت WriteStatus است بیت MSB بر روی Data قرار بگیرد.

حال به بیان مهم‌ترین حالت یعنی WriteStatus می‌پردازیم.



کلاک اصلی ماژول برابر با 40 مگاهرتز است، یعنی ورودی ماژول 40 مگاهرتز اما نکته‌ای که باید به آن توجه داشت این است که کلاک خروجی ماژول یا همان کلاک SPI برابر با 20 مگاهرتز است. کلاک 20 مگاهرتز توسط Toggle کردن سیگنال SCLK ساخته شده است. با هر لبة بالارونده کلاک اصلی، مقدار SCLK معکوس می‌شود. یعنی در هر تناوب کلاک اصلی سیگنال SCLK یک مقدار ثابت دارد؛ بنابراین فرکانس SCLK نصف فرکانس اصلی می‌شود؛ بنابراین سیگنال SCLK برابر با 20 مگاهرتز است.

بعد از چک‌کردن مقدار Enable، فاز SCLK و فلگ شمارش یک بیت از داده مدنظر را ارسال می‌کنیم و از شمارنده یک واحد کم می‌کنیم. Counter\_flag برای کنترل ارسال بیت آخر استفاده شده است که در ادامه توضیح خواهد داده شد.

بلاک SEND ترتیب ارسال داده‌های رجیسترهای مختلف را چک می‌کند که در اینجا مقدار آن برابر است با R یعنی اولین رجیستر در حال برنامه‌ریزی است. نکته‌ای که باید به آن توجه داشت این است گه اگر Counter برابر با صفر شود مقدار دو بیت آخر به دلیل تغییر در حالت state ارسال نمی‌شوند؛ بنابراین باید تأخیری در این حالت ایجاد کرد که علاوه بر عدم متوقف کردن سیگنال اصلی کلاک حالت State هم تغییر نکند.

در اینجا این تغییر با استفاده از متغیر counter\_flag و counter\_delay صورت گرفته است. زمانی که counter صفر می شود به دلیل تاخیر در ارسال ( تاخیر process) و تغییر در حالت State دو بیت آخر (LSB) ارسال نمی شوند بنابراین حدودا می بایست 5 کلاک اصلی تاخیر داشته باشیم تا این دو بیت ارسال شوند. بنابراین با صفر شدن counter مقدار counte\_flag برابر با true می شود تا دیگر از مقدار counter کاسته نشود و مقدار معتبر بر روی DATA قرار گیرد. زمانی که مقدار Counter\_delay برابر با 1 شده است می بایست بیت آخر ارسال شود که این شرط توسط بلوک if کنترل می شود.همچنین یک واحد به counter\_delay اضافه می شود زمانی که counter\_delay به عدد 5 رسید، یعنی تاخیر مد نظر اعمال شد و داده های مورد نظر ارسال شدند اکنون می بایست حالت State تغییر کند؛ نکته ای که باید به آن توجه کرد این است که با توجه به دیتاشیت بعد از برنامه ریزی ControlRegiste ، مدت زمانی به عنوان تاخیر اعمان کنیم و سپس به برنامه ریزی N LATCH بپردازیم. این مدت زمان تاخیر با مقدار خازن CN که در قسمت های قبلی توضیح داده شده است، متفاوت است. در اینجا مقدار تاخیر حدودا 500 میکرو ثانیه در نظر گرفته شده است. بنابراین اگر در حال برنامه ریزی Control latch بودیم یعنی مقدار que برابر با C بود باید حالت بعدی برابر با CtrlDelay باشد. در این بلاک توسط متغیری تاخیر مد نظر ایجاد شده است. اگر que برابر با C نبود به حالت FinalDelay تغییر حالت می دهیم. شکل (3-3) بیان گر تاخیر در برنامه ریزی رجیستر ها است. بلوک CtrlDelay:



در این حالت نباید بعد از برنامه‌ریزی Control Latch پایه LoadEnable برابر با یک شود؛ بنابراین توسط بلوک case بالا کنترل شده است.

دو حالت بعدی با نام‌های FinalDelay و FinalState حالت‌های پایانی ارسال هستند. برای درنظرگرفتن تأخیرهای مدنظر همچنین مقداردهی به que برای انتخاب رجیستر بعدی از این stateهای استفاده می‌کنیم.



به مدت یک کلاک در حالت FinalDelay تأخیر ایجاد می‌کنیم؛ اگر que برابر با C نبود آنگاه loadEnable باید 1 شود همان‌طور که در بالا اشاره شد.

در حالت Final state هم توسط بلوک case ترتیب برنامه‌ریزی را رعایت می‌کنیم و همچنین State را به حالت اولیه مقداردهی می‌کنیم.

* + 1. کدهای ماژول SPI













شبیه‌سازی و نتایج

برای شبیه‌سازی از نرم‌افزار ISIM استفاده شده است. ابتدا باید فایل TestBench را به پروژه اضافه کنیم. برای این کار از قسمت hierarchy کلیک راست کرده و سپس از قسمت NewSource فایل تست بنچ را اضافه می‌کنیم. قسمت‌های مهم تست بنچ تعیین فرکانس کلاک و process مقداردهی ورودی‌ها است.

نمودارها و کدهای پروژه

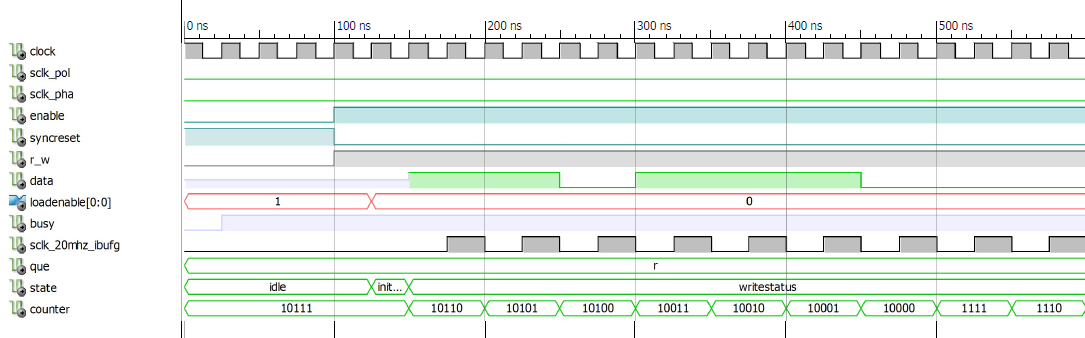


ثابت CLOCK\_period تعیین شده است. باتوجه‌به فرکانس کلاک اصلی که برابر با 40 مگاهرتز است دورة تناوب کلاک را برابر با 25 ns در نظر می‌گیریم.

**constant** CLOCK\_period : **time** := **25** ns;

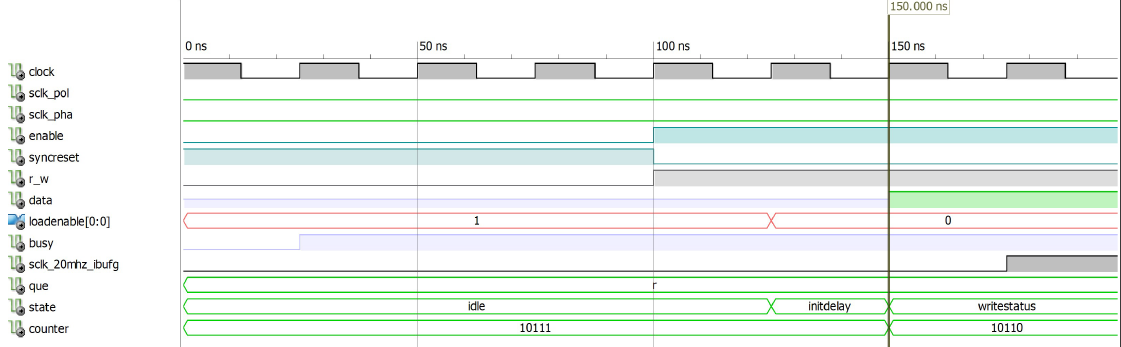
حال در Process شبیه‌سازی، مقادیر ورودی را مقداردهی می‌کنیم. در شبیه‌سازی ابتدایی فرض می‌کنیم پایة ریست به مدت 100 نانوثانیه فعال می‌شود؛ بنابراین در این حالت می‌بایست داده‌ای ارسال نشود و همچنین Busy\_BUF باید 1 شود. بعد از 100 نانوثانیه مقدار پورت‌های R\_W و Enable 1 می‌شوند همچنین ریست را غیرفعال می‌کنیم. مقدار و فاز اولیه کلاک راه هم برابر با 0 در نظر می‌گیریم.



باتوجه‌به کدهای بالا، بعد از 100 نانوثانیه ماژول به ترتیب شروع به ارسال بیت‌های رجیسترهای R latch ، C latch و N Latch کند.

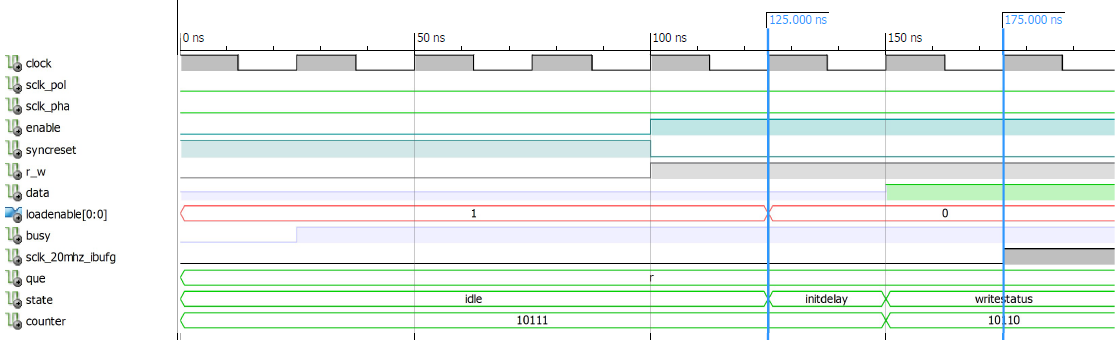
خروجی شماره 1

همان‌طور که در شکل (5-1) مشخص است تا 100 نانوثانیه ابتدایی، مدار در حالت ریست قرار دارد. در این حالت فرض کردیم پایه Busy یک شود. بعد از ریست‌شدن ماژول در زمان 100 نانوثانیه ریست غیرفعال شده و پایه enable و R\_W به‌منظور شروع عملیات ارسال فعال می‌شوند.

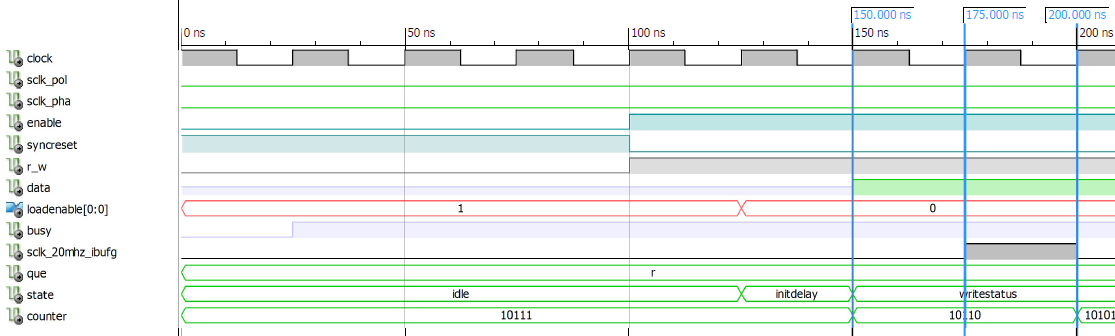
بعد از اینکه لبة بالارونده در وضعیت enable = 1 و R\_W = 1 برای اولین‌بار فعال شد، متغیر State به حالت InitDelay تغییر حالت می‌دهد و اولین بیت در این حالت ارسال می‌شود. نکته‌ای که باید به آن توجه داشت این است که اولین بیت ارسالی یا همان MSB در کلاک بعدی ارسال می‌شود. شکل (5-2) زمان ارسال اولین بیت بر روی پایه Data را نشان می‌دهد.

خروجی شماره 2

قبل از زمان 150 نانوثانیه پایه Data فرض شده است در حالت high-impedance قرار دارد. باتوجه‌به دیتاشیت setup-time ، LoadEnable برابر با 20 نانوثانیه است؛ یعنی بعد از گذشتن حداقل 20 نانوثانیه باید کلاک SPI اولین لبة بالارونده خود را فعال کند که در شکل (5-3) رعایت این تایمینگ را مشاهده می‌کنید.

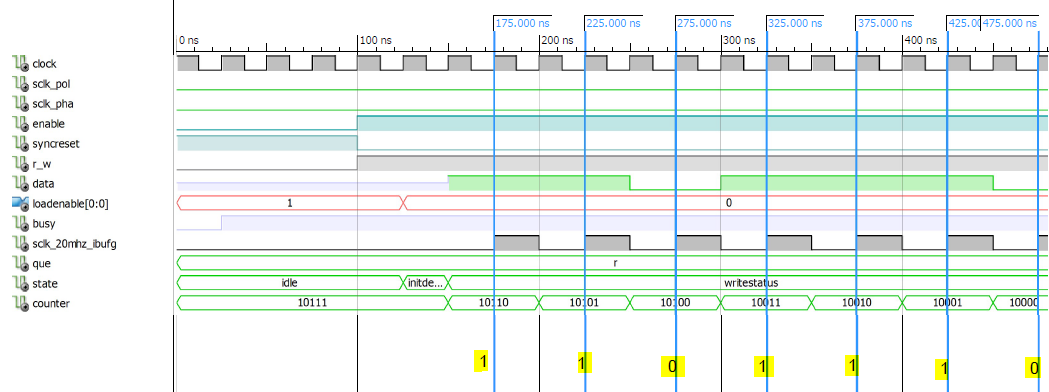


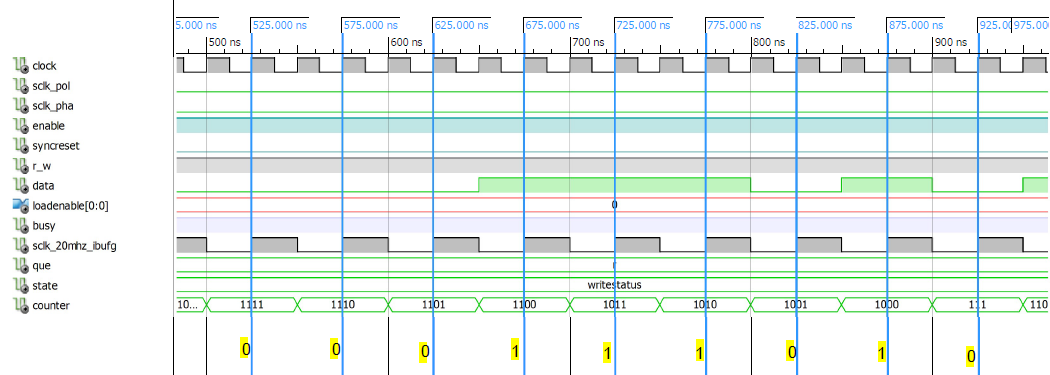
خروجی شماره 3

نکته دیگری که باید توجه داشت، setup time و hold time دیتا است. که باتوجه‌به دیتاشیت حداقل این مقادیر برابر با 10 نانوثانیه است. شکل (5-4) رعایت این تامینگ را نشان می‌دهد.

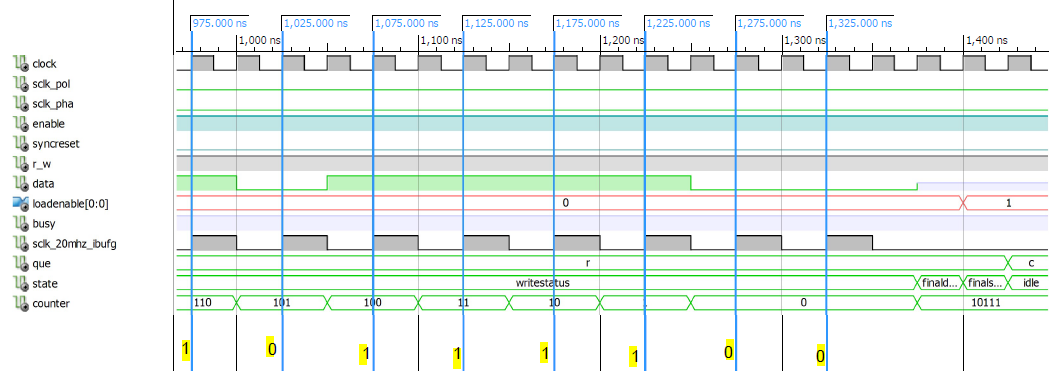
خروجی شماره 4

بعد از بررسی مقادیر زمانی حال به ارسال داده می‌پردازیم.

همانطور که از مقدار que مشخص است اولین رجیستری که باید برنامه ریزی شود R Latch است که مقدار آن را برابر با "110111000011101010111100" در نظر گرفتیم. حال به بررسی این مقادیر بر روی پایه DATA می پردازیم. با هر لبه ی بالا رونده SCLK\_20mhz\_ibufg می بایست یک بیت از سمت چپ (MSB) بر روی پایه Data قرار گیرد. در شکل‌های زیر نتایج ارسال این رجیستر را مشاهده می‌کنید.

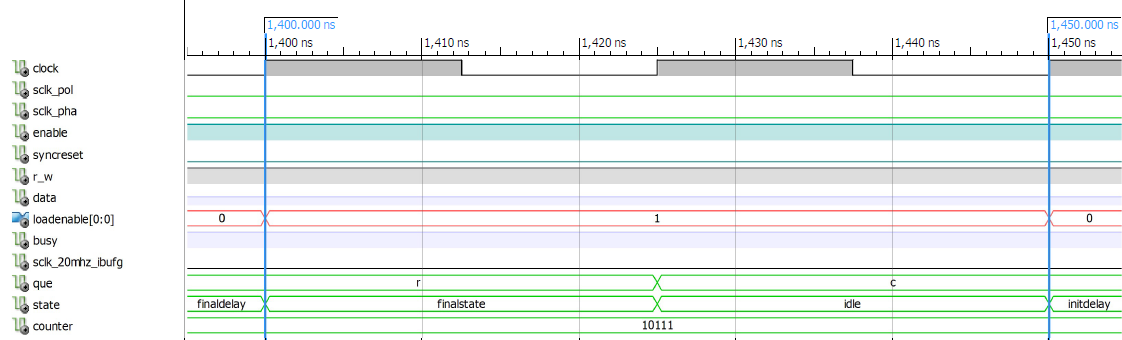
خروجی شماره 5

خروجی شماره 6

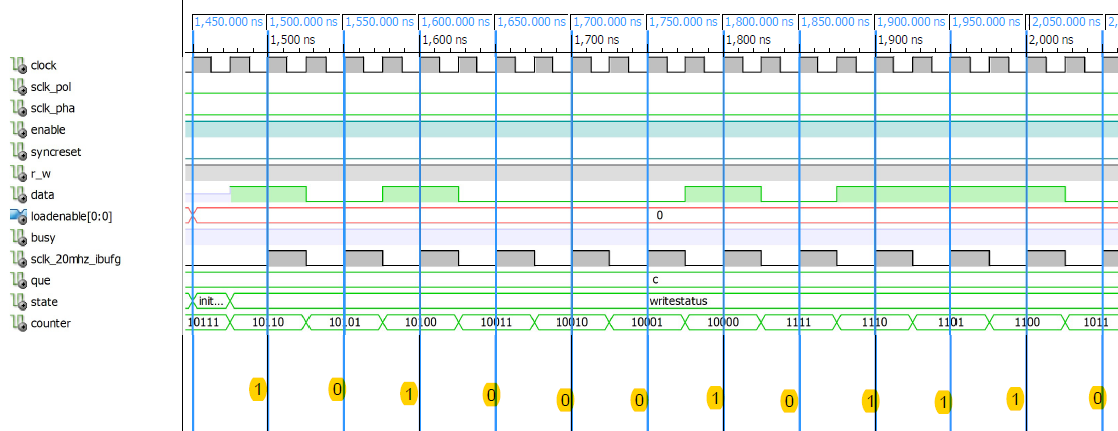


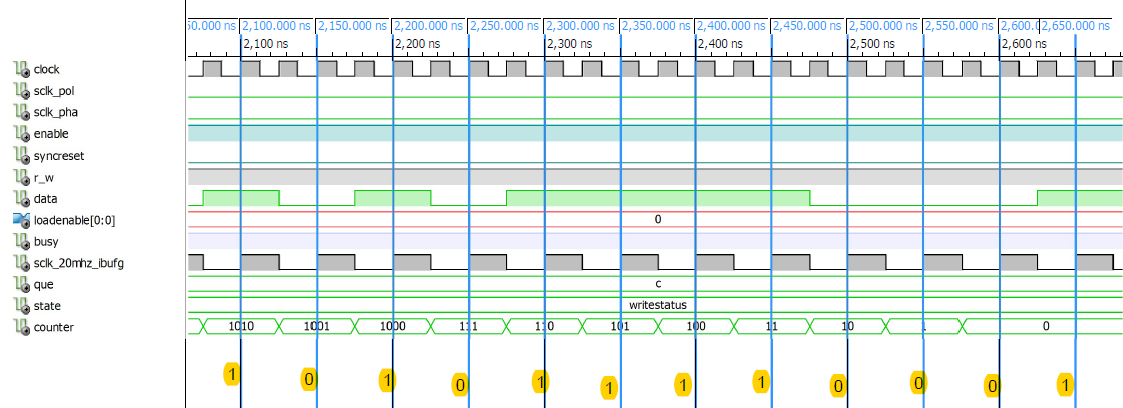
خروجی شماره 7

همان‌طور که در شکل‌های 5، 6 و 7 مشاهده می‌کنید دیتای R Latch به‌درستی ارسال شده است.

بعد از ارسال می‌بایست پایه LoadEnable به مدت 20 نانوثانیه 1 شود و سپس 0 شود و بیت‌های مربوط به رجیستر بعدی ارسال شود. در شکل (5-8) رعایت این تامینگ را مشاهده می‌کنید.

خروجی شماره 8

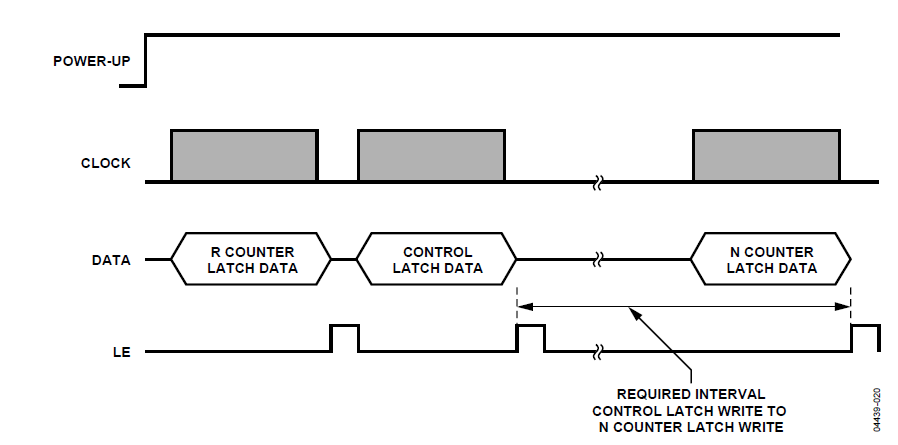
بعد از ارسال رجیستر R Latch باید رجیستر Control Latch برنامه‌ریزی شود. همان‌طور که مشاهده می‌شود مقدار que برابر با C شده است. در شکل‌های زیر به بررسی ارسال دیتاهای Control Latch می‌پردازیم. مقدار Control Latch برابر با "101000101110101011110001" است.

خروجی شماره 9

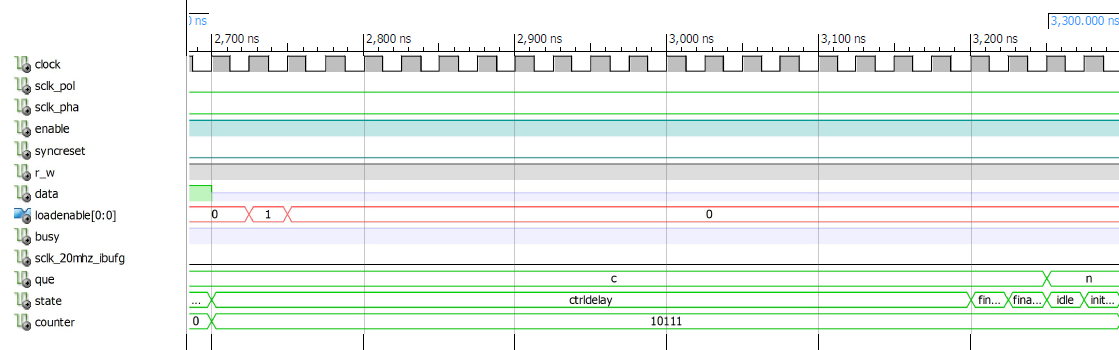
خروجی شماره 10

همان‌طور که در شکل‌های بالا مشاهده می‌کنید دیتای Control Latch به‌درستی ارسال شده است.

نکته‌ای که باید به آن توجه کرد گپ زمانی بین برنامه‌ریزی ControlLatch و N Latch باید ایجاد شود که شکل (5-11) این موضوع را نشان می‌دهد.



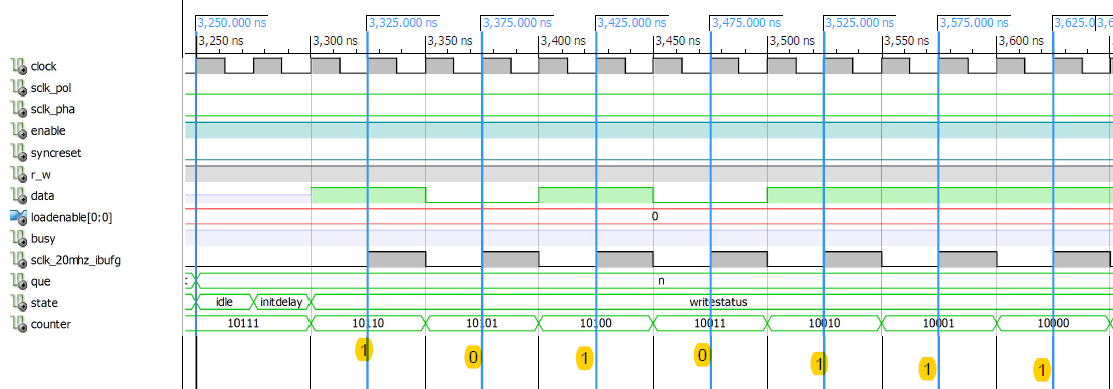
تایمینگ برنامه ریزی Latch ها

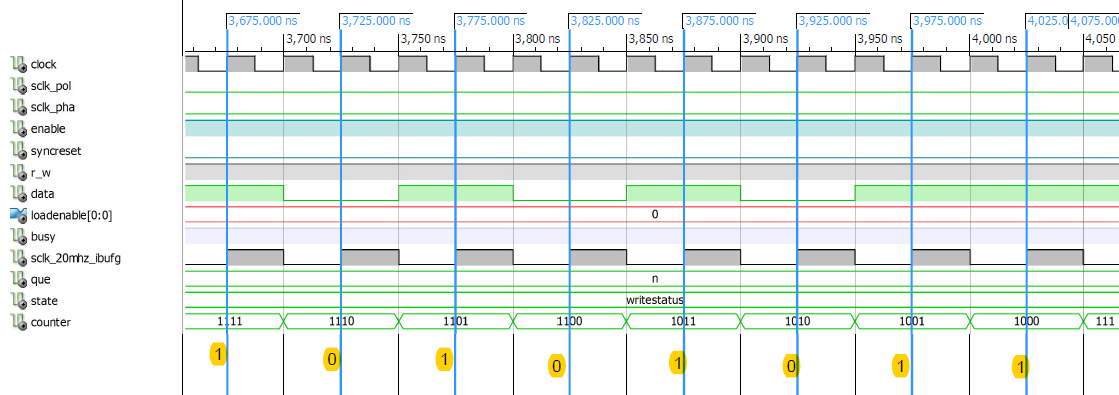
شکل (5-12) تایمینگ شبیه سازی شده را نشان می دهد.

خروجی شماره 11

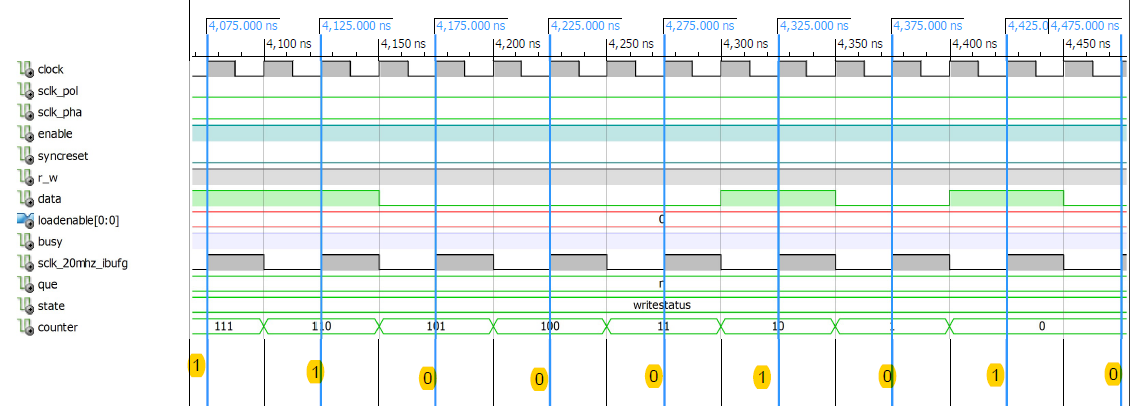
همانطور که گفته شده تاخیر بین Control Latch و N Latch را توسط حالت CtrlDelay ایجاد کردیم که در شکل (5-12) این موضوع مشخص شده است.

حال به بررسی ارسال بیت‌هایی رجیستر N latch می‌پردازیم. مقدار آن برابر با "101011110101011110001010" است.



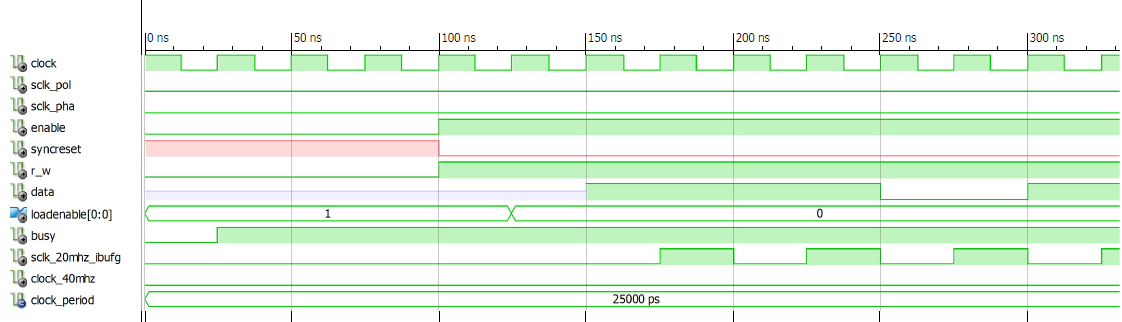
خروجی شماره 12

خروجی شماره 13

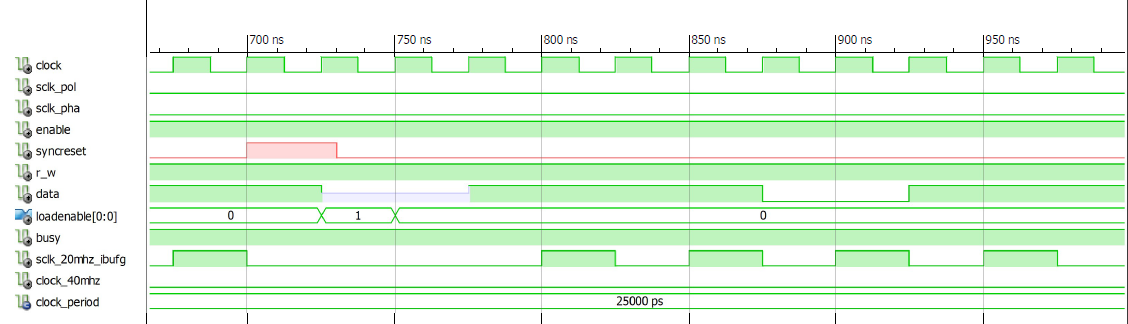


خروجی شماره 14

همان‌طور که در شکل‌های بالا مشاهده شد بیت‌های ارسالی معتبر هستند. بعد از پایان ارسال می‌توانیم عملیات ارسال را متوقف کنیم؛ اما در اینجا این عملیات مجدد صورت می‌گیرد.

در ادامه به بررسی حالت بحرانی ای می‌پردازیم. فرض کنید 500 نانوثانیه از زمان ارسال گذشته است و در یک‌لحظه ریست فعال می‌شود حال انتظار داریم بعد از غیرفعال‌شدن ریست ارسال دوباره صورت گیرد. در ادامه به بررسی این موضوع می‌پردازیم.

خروجی شماره 15



خروجی شماره 16

همان‌طور که در شکل‌های بالا مشاهده می‌کنید بعد از فعال‌شدن دوباره ریست ارسال داده‌های R Latch دوباره از اول شروع می‌شود.

* کدهای تست بنچ در فایلی با پسوند TB قرار گرفته‌اند.

نکته ای که باید به آن توجه داشت فایل ucf پروژه است.( برای پیاده سازی عملی بر روی بورد.) کد های مربوط به UCF:



در کد بالا مشخص شده است که سیگنال کلاک به کدام پایه متصل شود همچنین dutycycle این سیگنال هم مشخص شده است.

1. Serial Peripheral Interface [↑](#footnote-ref-2)
2. Load Enabel [↑](#footnote-ref-3)
3. Chip Select [↑](#footnote-ref-4)
4. Serial Data I/O [↑](#footnote-ref-5)
5. jitter [↑](#footnote-ref-6)