جزوه اول:

سیستم pure hardware مشکل این هست که ساخته بشود دیگر امکان تغییر نداریم. یک روش دیگر این هست هم بخش سخت افزار داشته باشد هم بخش نرم افزارش دست خودمان باشد تا بعدا هم بتوانیم الگوریتم آن را تغییر بدهیم. پیاده سازی 3 تا روش دارد که روش اول توی ریز پردازنده ها هست روش آخر هم که گفتیم بدرد ما نمیخورد توی این درس روی همون روش دوم تمرکز میکنیم همون سخت افزار خالص هستیم. توی همین روش خالص دیگر کاری به جزئیات نداریم فقط ورودی میدهیم و خروجی میگیریم خاص منظوره هست و performance بهتری دارد. اونایی که ASIC نباشند مثل فلش ها و حافظه ها و FPGA ها که میتوانند قابلیت برنامه نویسی مجدد داشته باشند ASIC ها خاص منظوره و داخل کارخانه ها ساخته میشوند. FULL CUSTOM ASIC یعنی همه ریز با جزئیات را خودمان میسازیم و بنابراین بهینه خواهد بود و کارکرد خوبی دارد چون فکر کن خونه را کلا خودت بسازی خوب آجر هم قوی هست ولی وقتی خود را درگیر جزئیات میکنی ساختن طرح های پیچیده خیلی سخت تر میشوند چون تا حد آجر وارد جزئیات شده هستی و دیگر به معماری آن نمیتوانی فکر کنی پس این دسته درسته خیلی بهینه هستند ولی چون جزئیات پیاده سازی زیادی هستند برای سیستم های بزرگ پیچیده هستند و برای سیستم های کوچک مورد نیاز هستند مثلا یک جمع کننده یا گیت اند اور یا مالتی پلکسر نه یک پردازنده یا یک خط لوله این روش در سطح ترانزیستور کار میکند. CMOS یعنی از دو قسمت مکمل به نام PMOS NMOS ساخته شده است. PMOS 1 را خوب عبور میدهد و افت ولتاژ ندارد و NMOS 0 منطقی را خوب عبور میدهد واسه همین NMOS. از اینها برای ساختن اند اور اینها استفاده میشود.

Gate array ASIC: میخواهیم یک مجموعه ای از گیت ها داشته باشیم مثلا یک ضرب کننده خط لوله ای و دیگه کاری نداریم گیت ها کجا قرار بگیرند فقط سیم بندی ها را مشخص میکنیم ورودی خروجی کنترل میکنیم فقط داخل اتصالات داخلی نیاز داریم مثلا خروجی یک گیت اند به ورودی یک گیت دیگه هست.

ASIC یا باید از ترانزیستور استفاده میکردیم یا کتاب خونه که توش ادر و گیت اندر اور اینا داشته باشه. 3 تا ASIC 2 تا in the field هست.

FPGA ها دو قسمت دارند logic cell و سوییچ ها که کار سوییچ ها وصل کردن لاجیک سل ها هست. سوییچ از یک سری مالتی پلکسر متصل شده است قابلیت یک برنامه ریزی دارد و دیگر مقدار آنها تغییر نمیکند و حافظه دارد. از دی فلیپ فلاپ برای نیاز به حافظه یا مدار های ترتیبی استفاده میشود. look up table هم برای پیاده سازی مدارات ترکیبی هست. شاید بهینه نباشند چون از همه امکانات استفاده نمیکنیم. یکی دیگر از راه ها PROM ها هستند. دیگر simple fpga ها هستند و راه آخر PCB یا بورد دیزاین هست که فقط ارتباطات را پیاده سازی میکنیم و اجزا بقیه ثابت هستند مثلا چطوری خروجی کدوم به کدوم وصل شود.

بر اساس یک سری متریک میگیم کدوم یکی از این 3 دسته مناسب هستند اول اندازه یا AREA که خود این به معماری و تکنولوژی ربط دارد. ASIC خیلی بهینه هست چون فقط قسمت هایی که نیاز داریم و میزاریم و اندازش کوچک هست GATE ARRAY ها چون بعضی قسمت هاش رو ممکن هست استفاده نکنیم یک مقدار بزرگتر هست و دسته آخر FGPA ها هستند که بسیار بزرگ هستند اندازه ثابت هست ولی بزرگ هستند. دوم کارایی یا سرعت هست، که اینجا منظور ما latency هست یعنی چه قدر طول میکشد یک معماری را پیاده سازی کنیم. هر چی سایز بزرگ کامپوننت ها دور تر و کارایی پایین تر هست با فرض معماری یکسان بهترین آن ASIC هست و بدترین FGPA. سوم از لحاظ پاور هست FPGA ها مصرف بالاتری دارد چون کلی سوییچ و سل داریم و نیاز به پاور دارند حتی اگر یک جا را فقط برنامه نویسی کنی باز باید کلش را برنامه بدهی تا به خروجی برسد و کمترین ASIC هست. چهارم هزینه هست. هزینه FGPA ثابت ولی هزینه ASIC زیاد هست زمان ساخت هم زیاد تر ولی FPGA ها کمتر هست. برای نمونه های اولیه یا شروع و تست کردن از FPGA و بعد از ساخته شدن IC ASIC جایگزین میکنیم باید هر دو باشند. توان انرژی در یک لحظه هست ما پول برق میدهیم. FPGA ها در پردازنده XEON همان شتاب دهنده ها هستند.

توصیف رفتاری این هست که بگوییم این سیستم چطوری کار میکند مثلا سی دی بگذاریم میتوانیم گوش بدهیم بابت این ورودی این خروجی را خواهیم داشت. توصیف ساختاری هست که میگیم کامپیوتر از چه قسمت هایی تشکیل شده است یک کامپوننت رو به ریز مولفه تبدیل میکردیم. یک توصیف دیگر توصیف فیزیکی هست که عین همون توصیف ساختاری ولی با جزئیات بیشتر مثلا این کامپوننت اندازش چه قدر هست کجا قرار گرفته طول این سیم چه قدر هست و یا کلاک طول آن چه قدر هست و ... .

از نرم افزار هر چه قدر به سخت افزار بروی جزئیات بیشتر میشود مثلا پایین ترین سطح ترانزیستور هست. توی اینجا هم بحث abstraction داریم برای productivity بالاتر و حذف جزئیات و هر چه قدر پایین تر یعنی تو سطح ترانزیستور خواستیم بریم جزئیات بیشتر مطرح میشود تا مدل ما ساده باشد. 4 لول انتزاع داریم پایین ترین ترانزیستور هست در این قسمت همه چی به صورت ولتاژ و جریان هست به صورت آنالوگ، گیت لول دیگه کاری بین صفر و پنج ولت نداریم فقط صفر و یک داریم و کلی جزئیات را حذف کردیم. سطح سوم سطح RTL هست. کلا رجیستر میشناسی با combinational logic اینجا توصیف ماشین حالت داریم یا state machine و اینجا سیگنال کلاک داریم که رهبر هست و همه المان ها داخل سیستم دیجیتال خود را با این کلاک هماهنگ میکنند. در سطح آخر و بعدی اینجا همه سطح پردازنده هست و با برنامه نویسی c مینویسیم همین لول هست. اگر نتوانیم کار بکنیم باید هعی جزئیات اضافه کنیم و بریم لول های پایین تر. توی اون 3 تا دسته رفتاری ساختاری فیزیکی پر جزئیات ترین فیزیکی هست.

هدف از سنتز این هست که هر توصیفی داشتیم ما یک سری گیت منطقی داشته باشیم که بهم وصل باشند یعنی توصیف ساختاری در سطح گیت باشد. هدف نهایی همین هست.

اگر سنتز در مرحله الگوریتم با زبان های برنامه نویسی رایج باشد اول باید بیاوریم تو سطح RTL بعد از این مرحله از توصیف رفتاری باید به توصیف ساختاری برویم با بهینه سازی بعد با توصیف ساختاری RTL باید برسیم به توصیف ساختاری گیت لول و این مرحله آخر هست. در آخر یک بهینه سازی هم میزند و بر اساس کارنو مپ هست و بدون در نظر گرفتن تکنولوژی هست. به این مراحل netlist گفته میشود.

در توصیف فیزیکی از ساختاری میایم به فیزیکی و باید بعد از سنتز و داشتن netlist یک خروجی بگیریم که بهش lay out گفته میشود اول تو سطح پردازنده بعد تو سطح RTL، بعد تو سطح گیت بعد میگیم چه سیم های داریم و چه شکلی هستند مرحله آخر این هست که تاخیر ها را به میدهند مثل گیت ها و سیم هم اینقدر تاخیر دارد بعد تاخیر این سیستم را میتوانیم بدست بیاوریم مثلا نواحی بحرانی و ... و بعد از آن بررسی میکنیم که چطوری باید کلاک ها پخش بکنیم داخل مدار. مرحله دوم طراحی فیزیکی بود مرحله سوم verification هست و چک میکنیم آیا سیستم مدل شده درست هست یا نیست. تست کردن تایم باید بعد از place and rout باشد یعنی گیت ها طول گیت ها جای گیت ها و سیم ها مشخص باشد. Simulation رفتار را شبیه سازی میکند و یک نرم افزار هست و شبیه سازی میکند سیستم واقعی را. Emulation رفتار را شبیه سازی نمیکند از خود ماژول استفاده میکند و از اون بهره میبرد.

EDA یک بحث transformation دارد یعنی نگاه میکند چه کدی نوشتیم آن را به صورت سخت افزار مناسب جایگزین میکند و بعد بخاطر اون توصیف یک بهینه سازی میکند مثلا اند سه ورودی بشود دو تا اند دو ورودی. حسن زبان های توصیف سخت افزاری این هست که در درجات بالا هست و احتمال خطا را کم میکند. دیزاین ما مستقل از تکنولوژی خواهد بود هم روی FPGA میشود هم روی ASIC با طول سیم مختلف. خروجی کد توصیف سخت افزار netlist هست. هر مدار ممکن هست simulation داشته باشد ولی سنتز نه لزوما چون باید با یک سری گیت های منطقی پیاده سازی کند و توسط یک سری گیت خروجی تولید میشود و اگر منظور ما سخت افزار خاصی باشد ولی قابلیت پیاده سازی شدن توسط گیت را نداشته باشد قابلیت سنتز شدن ندارد ولی هر چیزی شبیه سازی دارد. در زبان های سطح بالا مفهوم زمان مطرح نیست ولی در توصیف سخت افزار مفهوم زمان دارد و هر گیت تاخیر خاص خودش را دارد.

Delta delay: هنگام شبیه سازی دونه به دونه خط ها را میخواند و انتظار ما این است به صورت موازی اجرا شود ولی اینکه همزمان هست پس با یک تاخیر به نام دلتا اجرا میشود خط اول که خط آخر هم رسیده باشد و اجرا شود یعنی خط خوانده میشود ولی اجرا نمیشود.

شبیه سازی روی توصیف هیچ تاخیری ندارد. اگر netlist داشتیم تاخیر داریم.

VHDL:

برای پیاده سازی کارا به توصیف دقیق سخت افزار نیاز داریم. Entity declaration, architecture body باید این 2 تا باشند تا قابل سنتز شدن باشد سیستم. با entity ورودی خروجی تعریف میکنیم. Mode جهت را مشخص میکند یعنی ورودی یا خروجی هست. تو architecture بادی، میایم میگیم برای این ورودی خروجی ها چه رفتاری باید انجام شود توصیف پیاده سازی سیستم هست در واقع. بعد از begin هر چی بنویسی به صورت موازی هست یا concurrent چون ماهیت سیستم دیجیتالی به این شکل هست. از configuration استفاده میکنیم تا بگیم کدوم یکی از architecture ها رو باید برای اون entity استفاده کنی.

Signal تو بادی arch تعریف میشود بین arch و begin. مقدار دادن <= هست.

مقدار دهی اولیه: signal x: bit := ‘0’ ; هست. سیگنال همان سیم هست یا حافظه دارد که به یک دی فلیپ فلاپ وصل هست. هر چی راجب سیگنال بگیم مثل همان پورت هست. variable: همون متغیر های زبان سطح بالا هستند. موقع سنتز سیگنال یا سیم دو نقش دارد یا سیم خالی مثل باس یا سیم حافظه دار مثل دی فلیپ فلاپ اگر کلاک بود دومی هست اگر مدار ترکیبی بود سیم خالی. از متغیر برای محاسبه کردن مقادیر میانی هست وقتی میخواهیم موقت باشد اسکوپ آن داخل process ها هستش و داخل arch نیست یعنی قبل از begin مگر داخل process. متغیر بعد زمانی ندارد. داخل arch میتوانی process باشد و بین begin, end هست.

Constant داخل declaration arch هست همون ثابت خودمان هست. integer ها 32 بیتی هستند به صورت دیفالت.

اگر unsigned گذاشتیم برای عملیات محاسباتی میگذاریم و 8 بیتی هست جمع کننده 8 بیتی میگذارد جا 32 بیت الکی.

مقدار unsigned با signed جمع نمیشود.

Entity پورت ورودی و خروجی را بگذار.

Std\_logic: همون 0 و 1 و z هست.

Full adder: برای sum دو تا xor میزاریم که دو تاش a b هست سومی c in.

فصل سوم:

ترکیبی ها خروجی وابسته هست به ورودی و حافظه ندارند. لچ و فلیپ فلاپ ندارند. ترتیبی انگار از یک حالت به حالت دیگر میرویم. خروجی به حالت فعلی وابسته دارد.

After 10ns یعنی حاصل محاسبه میشود و بعد از 10 ثانیه در خروجی ریخته میشود. این برای سنتز نیست و برای تست ها هست چون زمان از دید زمان و مقاومت هست. test bench همون شبیه سازی هست.

Closed feedback loop: که نباید مدار ترکیبی داشته باشد یعنی سیستم خروجی وابسته به ورودی هست یا ورودی های گذاشته و باید بررسی کنیم که نباشند و ورودی گذشته نسبت به ورودی های گذشته تر بدست آمده است و شده است مدار ترتیبی ولی مدار ما ترکیبی باید باشد. چطوری بفهمیم ؟ دو طرف مقدار دهی سیگنال ها اگر یک نوع سیگنال بود خودش یا نات آن این مشکل را داریم.

تو مقدار دهی شرط اگر همه شرط ها درست بود اولین اولویت دارد برای مقدار دهی. Declaration قبل از begin توصیف ها بعد از begin.

اگر اون آخر when بگذاری برای شرط آخر ممکن است مشکل highampedance رو نادیده بگیری و 77 حالت دیگر را نادیده گرفتی و باعث شدی که مدار حافظه داشته باشد چون قرار نیست خروجی مدار تغییر بکند چون z1 یا مثلا 0z مدار خروجی قبلی خودش را حفظ میکند و انگار حافظه دارد در صورتی که در ترکیبی نباید حافظه داشته باشیم مالتی پلکسر ما، از نوع لچ در صورتی که نباید داشته باشد. دقت کن 0z بستگی دارد حالت قبلی چی بوده و انگار حافظه دارد چون مقدار قبلی را باید نگه داریم و خروجی عوض نمیشود.

جمع تفریق داشتی numeric\_std.all رو بیار چون با std\_logic این کار هارو نمیتوانی بکنی. بر اساس out باید یا بگی std\_logic یا هر چی.

Ctrl (1 downto 0 ) = “00” مساوی هست با بگی که ctrl = “100” فرقی نمیکند. چرا از بیت استفاده نمیکنیم ؟ چون ناقص هست فقط 0 و1 دارد ولی z رو کاور نمیکند واسه همین از std\_logic استفاده میکنیم.

تمام کد های شرطی فقط مالتی پلکسر میسازند بیس همه چیز مالتی پلکسر هست بخاطر همین داخل استاندارد سل ها همیشه مالتی پلکسر هست.

توی with select دیگر اولویت نداریم و انتخاب ها نباید یکسان باشد choice منظورمان هست mutual exclusive. باید همه حالات را تعریف کنیم all inclusive و گرنه همان مشکل قبلی که لچ هست پدید میاد حافظه دار میاد برای اینکه این مشکل پیش نیاد میگیم others. به جای choice آخر میزارن others رو.

جلوی with اونی که قرار هست بهش وابسته شود را میگذاریم. از سلکت سیگنال واسه وقتی که اولویت نداریم یا جدول درستی بکشیم استفاده میشود و از شرطی برای جایی که اولویت نیاز داریم. با این شرطی ها میتوانیم بیش از یک شرط را چک کنیم پرانتز بگذاریم و and یا or بعد از when. ولی سلکت ها فقط روی یک شرط هستند.

فصل چهارم:

برای نوشتن دستورات به صورت ترتیبی نه به صورت موازی از process استفاده میکنیم، ترتیبی کلا بهتر هست هر چی داخل این بدنه باشد به صورت ترتیبی اجرا میشود و این process بین begin, end arch هست و اگر چند تا بود به صورت موازی اجرا میشوند و هر کدام یک statement هستند. برای توصیف رفتاری به کار میرود هم مدار ترتیبی و ترکیبی را میتوان توصیف کرد. دیگه توصیف ساختاری نداریم ولی میتوانیم بنویسیم. دو صورت دارد یکی با sensitivity list که یک سری سیگنال توش هست به ازای تغییر اونها بدنه process یکبار اجرا میشود، اگر یکی از سیگنال ها را ننویسیم process اجرا نمیشود و مقدار قبلی خود را حفظ میکند و انگار حافظه دارد و دیگر ترکیبی نیست ترتیبی هست پس برای ترکیبی باید تمام ورودی ها را در لیست حساسیت بنویسیم و گرنه تشکیل حافظه میدهد، کلا ترکیبی ها باید به ازای تغییر ورودی خروجی تغییر کند. یک پیاده سازی پراسس با wait هم داریم 3 نوع دارد wait for برای زبان مثلا 10 ثانیه، wait until برای عبارت ترو فالس و wait on برای یک سری سیگنال ها صورت میگیرد. داخل پراسس فقط یک wait on میتوانی داشته باشی بیشتر باشد دیگه سیستم قابلیت سنتز شدن ندارد و بیشتر برای تست بنچ هست. مقدار دهی سیگنال هم مثل قبل هست فقط یک نکته داخل پراسس ها مقدار دهی سیگنال ها یک نکته دارد، 2 مرحله دارد یک evaluate و update که اولی مال سمت راست عبارت مقدار دهی سیگنال هست و دومی مال سمت چپ و اپدیت کردن مقدار آن در end process یعنی انتهای آن اپدیت میشود.

Variable استفاده نمیکنی خروجی که استفاده میشود چی هست و به صورت فوری هست و دیگه 2 مرحله قبلی سیگنال را ندارد یعنی evaluate update دیگر تا آخر پراسس نیست همون لحظه اپدیت میشود. evaluate آخر هر خط ولی اپدیت انتهای پراسس رخ میدهد.

از سیگنال استفاده کن.

در ساختار when else فقط یک مقدار دهی سیگنال داشتی ولی برای if then میتوانی چند تا سیگنال مقدار دهی کنی، دقت کن if then فقط داخل process ها هستند و when else ها داخل arch body هستند. ورودی ها را داخل پراسس بگذار. اگر وابسته بود باید بیاد داخل پراسس ولی نیومده همون داخل arch بگذاری مهم نیست. اگر داخل if then فقط یک مقدار دهی سیگنال داشته باشی همون when else هست بدون هیچ تفاوتی زمان تفاوت هست که به ازای هر if then بیش از یک مقدار دهی داشته باشیم به ازای تعداد مقدار دهی سیگنال ها مالتی پلکسر خواهیم داشت فقط سلکتور های آنها یکسان هستند یعنی به ازای همه اون سیگنال هایی که سمت چپ هستند مقدار دهی میکنیم در خروجی مالتی پلکسر ها قرار میگیرند فقط سلکتور های اینها مشترک هستند. همه if else مثل when else مالتی پلکسر دارند. If else جز sequential assignment ها هستند ترتیبی هستند و داخل پراسس ها فقط هستند ولی when else و with select جز موازی ها یا concurrent ها هستند برای مقدار دهی سیگنال ها چه با اولویت چه بی اولویت و داخل arch قرار میگیرند.

برای incomplete branch میایم با else این مشکل را حل میکنیم.

مورد دیگر incomplete signal assignment هست که این هم یک مشکل دیگر هست که باید بالای پراسس یعنی درست بعد از begin دوباره مقدار دهی دیفالت که همان صفر هست بدهیم چون اگر ندهیم انگار حافظه دارد و همه خروجی ها یک میشود به ازای هر مقادیری مثلا چه a بزرگتر باشد چه b بزرگتر باشد چه مساوی باشند خروجی 1 میدهد و عملا بی فایده هست دیگر مقایسه گر نیست.

سیگنال آخر پراسس اپدیت میشود. هر جا دنبال اکتیو هستی بنویس با غیر اکتیو ها کاری نداشته باشد. اگر چند تا if به صورت جدا از هم تعریف کردیم if آخری بیشترین اولویت را دارد.

Case when == with select سمت راستی رو قبلا داشتیم اینجا هم دقیقا همان شکلی هست منتها داخل پراسس.

When else = if else.

دقت کن با case when هم مشکل incomplete signal assignment خواهیم داشت و به جای اینکه داخل هر شرط مقدار دهی کنیم یکبار همون بالا همه را صفر میکنیم.

این هم مانند if else بیش از یک سیگنال میتوانیم مقدار دهی کنیم که با when else نمیتوانستیم و دقت کن کنترلر اینجا هم یکسان هست.

Others یعنی سیم هایی که مورد استفاده نیست به هم وصل شوند و بروند به یک حالت.

جایی از loop استفاده میکنیم که قرار هست یک کار تکراری میخوایم انجام بدهیم. و دقت کن لوپ داخل پراسس هست و خارج از پراسس نمیتوان استفاده کرد دقت کن loop range باید از قبل مشخص باشد و ایستا باشد نمیتوانی مثلا از کاربر بگیری.

+= ببین تو vhdl چه شکلی هست.

برای یک مدار ترکیبی دو حالت داریم concurrent و sequential.

Concurrent : برای توصیف سخت افزاری هست و میدانیم مالتی پلکسر داریم.

Sequential: بیشتر برای توصیف رفتاری هست و ساده تر هست مثلا پراسس داشتیم.

If ها برای وقتی هست که اولویت داریم دقیقا مثل when else ها.

Case when ها برای وقتی هست که اولویت برای ما مهم نیست دقیقا مثل with select.

دقت کن سخت افزاری برای لوپ نداریم و یک راه میان بر هست و همون رو باز کنی به جای 4 تا خط یک خط نوشتی و اصلا معادل فیزیکی ندارد.

فصل پنجم:

مدار ترکیبی حافظه ندارد یعنی ورودی گیت اندر کاری ندارد قبلا چی بوده بر اساس ورودی الان خروجی تغییر میکند ورودی تابعی از خروجی هست واسه همین پراسس داشتیم و لیست ورودی ها را مینوشتیم.

مدار ترتیبی یعنی ترتیب برای ما مهم هست پس در واقع حافظه داریم چون بر اساس ورودی های هست که قبلا به سیستم ما داده است، به 2 دسته سنکرون و آسنکرون تبدیل میشوند و معمولا سنکرون هستند چون پیاده سازی آنها ساده هست.

D latch: یک کلاک دارد که همان پایه enable هست اگر فعال بود ورودی را بزار روی خروجی ولی اگر صفر بود خروجی را hold کن.

D flip flop: این دیگه با enable کار نمیکند در یک لحظه فعال میشود سر لبه کلاک بالا رونده یا پایین رونده عمل میکند کلاک چه صفر یا چه 1 باشد مقدار قبلی را نگه میدارد ولی وقتی پایه کلاک بالا رفت ورودی را بر روی خروجی میگذارد کنترل آن دست خود ما هست. آسنکرون یعنی به کلاک کاری ندارد مثل ریست 1 شد خروجی صفر هست دیگه کاری ندارد کلاک چی هست.

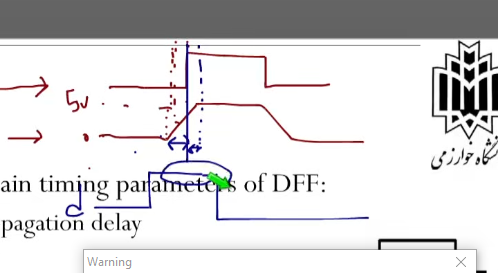
لچ دو تا مشکل داشت یکی اون شرایط رقابتی پیش میومد جفت صفر میشد و دو شرایط نویز پیش میومد و روی خروجی تاثیر میزاشت چون تو بازه فعال شدن کلاک خروجی کلا به ورودی نگاه میکند و نویز میگیرد با بالا و پایین شدن آن. ولی این 2 تا مشکل را فلیپ فلاپ ندارد. به مشکل دوم glitch گفته میشود. بخاطر همین داخل پایپ لاین از دی فلیپ فلاپ استفاده میکردیم تا دیگه ثبات داشته باشیم چون اگر لچ بود هعی باید چک میکردیم و تا میخواستیم بنویسیم مشکل داشتیم و واسه همین از فلیپ فلاپ استفاده میکنیم تا زمان بخریم تا بتوانیم بنویسیم و ثبت کنیم. این 3 تا مزایا دی فلیپ فلاپ هست. فلیپ فلاپ 2 برابر لچ حجم میگیرد ولی مزایای آن بیشتر هست واسه همین چشم پوشی میکنیم.

دی فلیپ فلاپ باید یک مدتی قبل از بالا رفتن کلاک مقدار ورودیش ثابت باشد تا دی فلیپ فلاپ بتواند به درستی بخواند به این میگوییم ستاپ تایم.

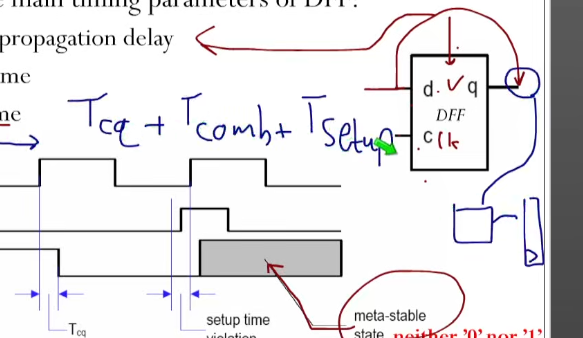
بعد از لبه کلاک هم یک مدتی زمانی این مقدار ورودی ثابت باشد به این hold time گفته میشود. ورودی همان d هست تا دی فلیپ فلاپ درست دریافت کند.

به مدت زمان تاخیری که ورودی روی خروجی قرار بگیرد میگوییم cq یا کلاک به خروجی.

دقت کن ستاپ تایپ یعنی وقتی کلاک صفر هست منظور هست و کلاک وقتی 1 هست میشود hold تایم.



مدت زمان حداقل یک کلاک در نظر بگیریم:



یکی تاخیر کلاک به خروجی یکی تاخیر کامپوننت ترکیبی و یکی ستاپ تایم دی فلیپ فلاپ چون قبل اون باید ورودی ثابت شود. یک تقسیم بر اینها میشود ماکسیموم فرکانسی که سیستم باهاش کار میکند.

Meta stable: در یک کلاک حذف میشود ولی در چند کلاک اصلا امکان حذف شدن ندارد.

ترتیبی ها دو دسته میشوند 1. با یک کلاک کار بکنند بدون مشکل 2. هر کامپوننت با کلاک خودش بکند و سنکرون باشد ولی به صورت کلی آسنکرون و با چند کلاک مختلف بهش GALS گفته میشود. چون بحث فاصله بین کامپوننت ها را داریم نمیتوانیم از یک کلاک استفاده بکنیم.

Clock skew: یعنی به خاطر فاصله و تاخیر دی فلیپ فلاپ های مختلف لبه کلاک را همزمان احساس نمیکنند و با تاخیر مختلف دریافت میکنند.

سیستم آسنکرون: یعنی کلاک یک دی فلیپ فلاپ به خروجی دی فلیپ فلاپ قبلی بستگی دارد یعنی کلاک کلی ندارند و یا یک دسته دیگر این هست که بدون کلاک هستند یا به سطح حساس هستند یا closed feedback loops هستند. با اینها مشکل داریم.

پس دنبال مدار ترتیبی سنکرون هستیم که تمام دی فلیپ فلاپ هاش از یک کلاک و رهبر استفاده میکنند. و اگر یک گروه فرمان بگیرند بقیه هم میتوانند فرمان بگیرند.

هر مدار ترتیبی سنکرون 3 بخش دارد

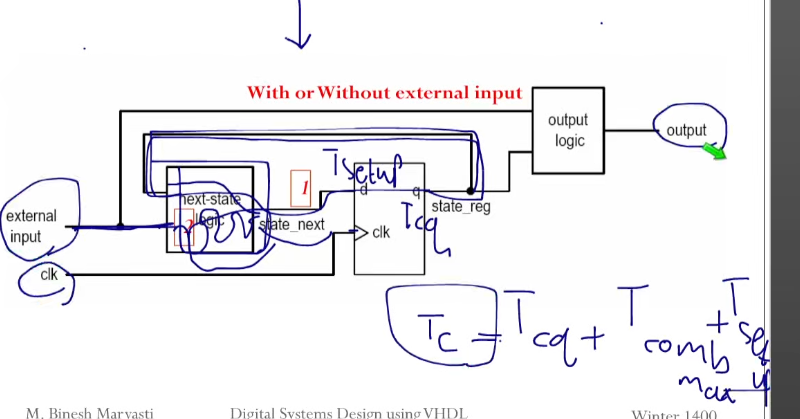
State register میگوید مدار ما در چه حالتی هست حافظه هست مثلا 10 یا 15

نکست استیت لاجیک یعنی کلاک بعدی تو چه حالتی باید بریم مثلا الان اگر ده باشیم این از جنس ترکیبی هست.

اوت پوت لاجیک که خروجی مدار را میسازد مور فقط به استیتی که داخلش هستیم وابسته هست میلی علاوه بر اون به ورودی هم وابسته هست. این هم ترکیبی هست.

استیت رجیستر همان دی فلیپ فلاپ هست خروجی آن state\_reg هست. دقت کن state\_next که ورودی آن هست به اندازه ستاپ تایم و هولد تایم تغییر نکند.

کلاک سیستم:



جلسه 14 فروردین اسلاید 16 دوباره ببین.

توی این سیستم ها نویز تاثیری ندارد چرا ؟ چون فقط تو یک لبه کلاک نگاه میکنیم و تو طول زمان اهمیتی ندارد.

گلوبالی سنکرون بر اساس next state خودش 3 دسته میشود. 1 بر اساس استیت قبلی استیت بعدی را حدس میزنیم مثل شمارنده که بهش regular میگیم. حالت دوم این شکلی هست ترتیبی و الگو خاصی نمیبینیم و وابسته به ورودی هست و random هست و FSM ها اینجا هستند. دسته سوم استیت های آنها به صورت رندوم هست و دیتا پث هم داخل خودشان دارند یعنی فقط خروجی نمیسازد دیتا پث هم دارد. حالت اول اینطوری هست که state\_next <= state\_reg + 1 یعنی مقدار فعلی بعلاوه یه عدد که نقش شمارنده را دارد.

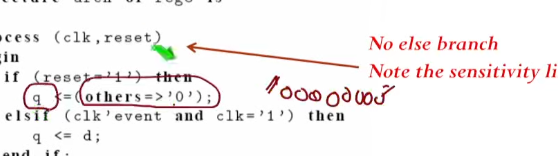
طراحی state register: دقت کن تو این سیستم سنکرون طراحی دو قسمت دیگر ترکیبی بود بلدیم این قسمت حافظه چند نوع داریم لچ و closed feedback و دی فلیپ فلاپ.

لچ: از قالب استفاده میکنیم. استفاده نکنیم مالتی پلکسر میگذارد مشکل closed feedback loop این هست که بر اساس وقتی که دی لچ کلاک صفر شود خروجی قرار هست قبلی را نگه دارد ولی یک لحظه خروجی عوض میشود چون تاخیر پایینی از بالایی بیشتر هست بالایی که صفر هست زودتر قرار میگیرد به ازای یک لحظه. از حافظه ها را خودت ننویس.

دقت کن داخل لیست حساسیت باید d را هم برای لچ بگذاری چون اگر کلاک فعال باشد دی تغییر کند خروجی هم تغییر کند. ولی واسه دی فلیپ فلاپ فقط کلاک را میگذاریم چرا چون یک لحظه فقط به کلاک میخواهیم نگاه کنیم فقط یک لحظه و همیشه نیست. clk’ event یعنی مقدار کلاک تغییر کرد مثلا از صفر شد به 1. یا بنویسیم rising\_edge(clk). لبه پایین رونده میشه falling\_edge(clk). آسنکرون ها رو هم بیار تو لیست حساسیت چون دیگه منتظر کلاک نباید بمونی اونا هم عوض شدن باید بیای داخل پراسس. ساختار if else میشود همان مالتی پلکسر ولی ساختار elsif مثل قبلی نیست.

رجیستر همان تعداد فلیپ فلاپ هست

نوع بعدی حافه RAM هست.



نوع بعدی حافظه RAM هست برای ترتیبی ها و از لچ ها ساخته میشوند و همان قالب لچ با کنترل یونیت ولی استاندارد نیست بهترین کار این هست که از IP CORE استفاده بکنی و این حافظه استاندارد هست و همان RAM هست.

هر مدار ترتیبی سنکرون next-state logic دارد.

آقا کد state register رو حفظ کن از خودت چیزی اختراع نکن کد رجیستر مال دی فلیپ فلاپ ها رو حفظ کن از خودت چیزی اختراع نکن.

راجب next state هم با when else هم با with select میتوان و هم میتوان پراسس نوشت و با if else نوشت.

Free running یعنی کنترلی روش نیست همینطوری شیفت میدهد 4 بیتی هم یعنی بعد 4 کلاک سایکل ورودی در خروجی هست بعد از شیفت.

دقت کن هر مدار ترتیبی اون 3 تا قسمت را دارد یعنی state register و نکست استیت و output logic. دقت کن خروجی برنامه بیت کم ارزش هست واسه همین شیفت و دقت کن واسه هر مدار ترتیبی باید این 3 تا قسمت رو رعایت کنی. سعی کن درست مدل کنید. مالتی پلکسر با اون 4 تا شرطی ایجاد میشود.

Right shift: d از چپ میاد وارد میشود و 3 تا 1 را باید پر کنی

Left shift: d از راست میاد و از صفر تا 2 را باید پر کنی.

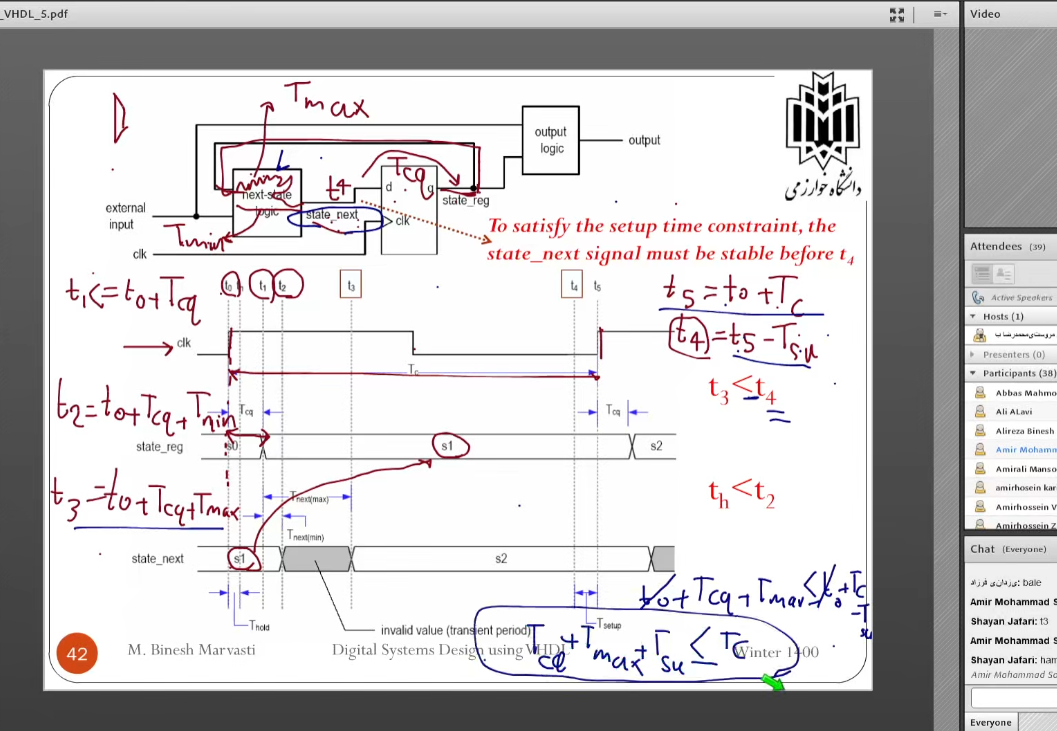
مدار ترکیبی بر اساس طول سیم و تاخیر گیت ها هست همین فقط تاخیر انتشار. ولی در ترتیبی فرق میکند چون المان حافظه ای مثل دی فلیپ فلاپ داریم و رفتار متفاوت دارند و ستاپ و هولد و کلاک به خروجی باید لحاظ شود. مهم ترین سیگنال که باید این مسائل توش رعایت شود state\_next هست که به اندازه ستاپ تایم باید قبلش اماده شده باشد. برای همین هم دقت کن یک مدار ترکیبی داریم و یکی هم external input که گفته میشود که خود این 2 حالت دارد یا همین بخش از یک سیستمی دیگری با کلاک یکسان تولید میشود که تحت کنترل ماست و یا نه با کلاکی تولید میشود که با کلاک سیستم فعلی متفاوت هست. در مدار ترکیبی هم دو تا تاخیر ماکسیمومم و مینیمومم داریم چون بیش از یک مسیر برای تولید این سیگنال تولید شده هست. مدار ترکیبی یکی state\_reg هست که همیشه هست و همیشه تحت کنترل هست بخاطر کلاک چون مدیریت میکنیم بحث همون external input هست که قبلا توضیح دادیم.

کلاک سایکل تایم: فاصله بین 2 لبه بالا رونده کلاک.

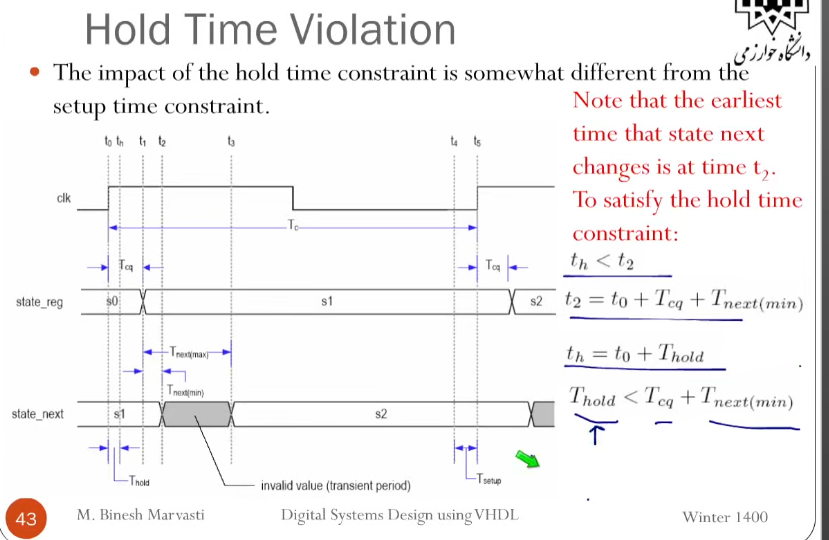
Duration با t بزرگ و TC همان کلاک سایکل تایم هست. دقت کن دیتا باید یک مدت زمانی قبل از بالا رفتن کلاک آماده باشد که اسمش را T4 گذاشتیم همان ستاپ تایم و از منهای T5 منهای زمان ستاپ تایم هست.

چه زمانی طول میکشید state\_next بیاد تو state\_reg؟ تایم کلاک به خروجی یا TCQ. کلاک به اوت پوت. و T1= T0 + TCQ. دقت کن باید زمانی بین ماکسیموم و مینیموم بین این 2 انتخاب کنیم. T3 دیرترین زمانی هست که ورودی میتواند انتخاب کند پس باید همواره کوچکتر مساوی T4 باشد اگر مساوی باشد میشود ماکسیموم کلاک فرکانس سیستم.

ماکسیموم کلاک فرکانس:



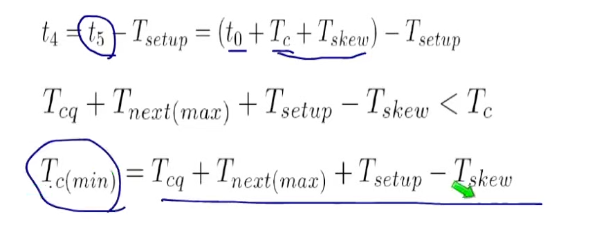
یک تقسیم بر همین میشود مثلا 2 مگاهرتز کلاک سیستم. T HOLD چرا به کار نمیاد؟ چون همیشه از TCQ کمتر هست. HOLD تایم میگوید به اندازه این تایم بعد از لبه کلاک باید STATE\_NEXT ای ورودی ثابت باشد سریعترین زمان تغییر آن t2 هست. چون باید روی خروجی بنشیند و کلا سریعترین میشود یک زمان TCQ و مینیموم مدار ترکیبی. واسه همین هولد تایم باید از T2 کمتر باشد.



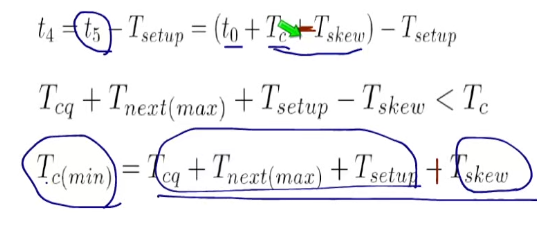
اگر فقط سیم باشد دیگر ترکیبی نباشد T NEXT صفر میشود و میتوان تضمین کرد به این شکل باشد TH < TCQ چرا چون اون ممکن هست فقط سیم باشد و صفر باشد.

HOLD TIME VIOLATION داریم حتی با حل این مشکل که بهش clock skew میگوییم. تو سیستم سنکرون قرار هست همه کلاک را به موقع و همزمان دریافت کنند ولی وقتی یکی زودتر و یکی دیرتر دریافت کند کلاک را بهش clock skew گفته میشود. این مشکلات دارد همین به 2 قسمت تقسیم میشود یکی positive یکی negative، اولی یعنی اینکه کلاک از بالا به سمت پایین میاد و اگر مسیر دیتا هم به همین شکل باشد به این میگوییم مثبت یعنی هم کلاک از 1 به 2 و هم داده از 1 به 2 میرود این مثبت هست منفی چی ؟ همین منتها مسیر کلاک برعکس باشد از 2 به 1. مشکل این مثبت چی هست؟ وقتی مثبت رخ بدهد یک مدت زمانی اضافه تر میشود و به نفع من هست و مشکل ستاپ تایم حل میشود و انگار زمان بیشتری برای حل ستاپ تایم داریم. ولی مسئله این هست که روی hold time تاثیر منفی میگذارد چون یک مدت زمانی برای تثبیت ورودی قرار میدادیم و زمان هولد تایم کمتر میشود چون دیرتر لبه بالا تر رفته هست. Tc همان تایم کلاک سایکل تایم هست یعنی فاصله بین 2 لبه. مثبت باشد فرکانس ما بالا تر میرود چون اون زمان بود یه مقدار هم کمتر میشود حالا و کلاک سایکل تایم بیشتر میشود در واقع clock skew باعث میشود مشکل کمتر شود چون از اون سایکل تایم یک مقداری هم داریم کمتر میکنیم فرکانس بیشتر میشود. برعکس همین وجود دارد واسه منفی مشکل ستاپ تایم حالا رخ میدهد.

کلاک سایکل تایم مثبت :



کلاک سایکل تایم با تاثیر skew منفی:

 که باعث مشکل ستاپ تایپ میشویم چون skew هم میاد زیاد میشود و نیاز به زمان بیشتر داریم و روی ستاپ تایپ رخ میدهد. hold time در حالت عادی رخ نمیدهد در موقعی که skew رخ بدهد این هم رخ میدهد که فرمول آن هم بود t hold < TCQ – T skew. انگار هولد تایم تو تنگنا قرار گرفته است.

حل مشکل setup time violation: برای clock skew negative طبیعتا چون تو مثبت که رخ نمیدهد راه حل این است که باید کلاک سایکل تایم را افزایش بدهیم چون باید جمع کنیم دیگه و فرکانس میاد پایین.

Hold time violation: برای skew مثبت راه حلی ندارد. مشکل ستاپ تایم با کاهش فرکانس سیستم حل میشود ولی هولد تایم را نمیتوان کاری کرد.

پیاده سازی:

2 راه حل داریم: one segment coding: یک پراسس مینوشتیم و هر 3 قسمت مدارات ترتیبی در آن بود اون state register و next state و ... ، .

هر نوع مقدار دهی سیگنال زیر کلاک تشکیل دی فلیپ فلاپ میدهد. و انگار دی فلیپ فلاپ اضافه گذاشتی و مشکل این هست که کلاک سایکل دیر تر خروجی تولید میشود انگار max pulse با یک کلاک دیر تر تولید میشود چون سر همون کلاک صفر نمیشود. فقط دقت کن مقدار دهی زیر کلاک اگر سمت چپ یکی بود از یک جنس هستند و به ازای همان یک دی فلیپ فلاپ اضافی داریم. در واقع هر سیگنال یا پورت یک دی فلیپ فلاپ هستند که ورودی آنها مقدار سمت راست مساوی هست. سمت چپ روی خروجی قرار میگیرد. دقت کن فقط سیگنال ها دی فلیپ فلاپ میشوند.

اول درست مدل کن. سنکرون هست یک state register که همان دی فلیپ فلاپ هست.

عملیات محاسباتی مثل جمع و تفریق داشتی باید از numeric\_std استفاده کنی به جای std\_logic و بعد هم تازه باید از نوع unsigned تعریف کنی. فقط دقت کن چون خروجی رو از جنس std\_logic تعریف کردی تهش باید unsigned رو تبدیل کنی به std\_logic.

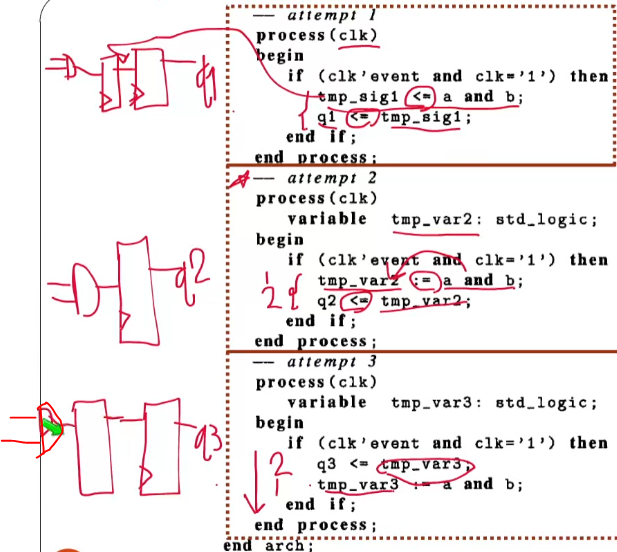
صفحه 55 نکته others.

نکته 55 یکی این هست که همون دی فلیپ فلاپ و یکی اینکه اون مقدار 15 که قرار هست بعدش صفر شود در انتهای پراسس اپدیت میشود و r\_reg اپدیت نمیشود و هعی max pulse صفر میماند در صورتی که شمارنده تا 15 شمرده است و انگار max pulse زمانی 1 میشود که اون شمارنده صفر شده است و فایده ندارد. one segment coding ننویس. اگر خواستی بنویسی خارج پراسس بگذار.

جلسه بعدی 21 فروردین:

Mod m counter یعنی شمارنده ما تا اون m میشمرد بعد صفر میشود. روی اون سیگنال یا ورودی که میخوای عملیات محاسباتی اعمال کنی unsigned استفاده کن و numeric\_std. سیگنال و پورت <= و متغیر := هست مقدار دهی آنها. متغیر ها زمان ندارند و هر لحظه که evaluate میشود اپدیت هم میشود یعنی سمت راست هر چی گیت بود خروجی آن میشود سمت چپی.

دقت کن داخل پراسس ترتیبی جلو میرویم و داخل صفحه 65 اون مورد سوم tmp\_var3 خط دومی یک چیز جدید هست و با خط اول فرق دارد و انگار یک حافظه هست نه همان مدار سمت راست چون انگار حافظه دارد و تو کلاک قبلی اون and وارد میشود.

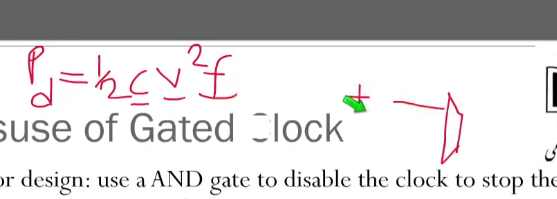
در حالت 3 همونطور که توضیح دادم اون tmp\_var3 در کلاک قبلی بدست آمده است و خط بعدی یک چیز جدید هست پس برای خط اول نیاز به حافظه داریم تا مقدار آن در کلاک قبلی را نگه دارد تا بتوانیم این خط اجرا شود و بعد برای خط جدید اون گیت مورد نظر را بکشیم و خروجی گیت را دوباره باید ذخیره کنیم در حافظه برای خط اول.

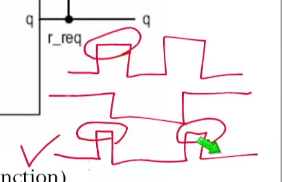
در بعضی جا ها استفاده میشود مثل همون قبلی که در آخر پراسس اپدیت میشد و یکی عقب میفتاد شمارنده ما ولی اینجا با متغیر راحت حل میشود.

فصل ششم:

اسلاید 7: مشکل این هست که اون شمارنده یک لحظه 10 میشود ولی نمیخواهیم هیچ موقع 10 بشود و بخاطر تاخیر گیت or هست در صورتی که اون مدت زمان باید صفر میبود و خروجی ما مدت زمانی کمتر دارد مثل 0. سیگنال های آسنکرون نباید دستکاری بشوند. به جای این روش از همان روش مرسوم که داریم مالتی پلکسر هست استفاده میکنیم. از آسنکرون همینطوری نباید استفاده بکنیم. روی سیگنال های کلاک و ریست هیچ گیت نباید بگذاریم چون تنظیم بهم میخورد.

Gated clock: یکی از روش های رایج کاهش توان مصرفی هست. فرمول توان دینامیک یا pd میشود یک دوم مدارات cmos یا یک دوم c v 2 f. و بحث این هست ما عملیات های مختلفی داریم ولی ما در هر لحظه به یکی نیاز داریم چرا همش باید اجرا شود و اگر فرکانس را صفر بکنیم دیگه مصرف دینامیک نخواهد داشت.

 یعنی میگه وقتی به ضرب کننده نیاز نداری یک گیت اند بگذار روی مسیر کلاک و enable تا اگر کلاک نزد صفر بشود و توان مصرفی بشکند اما راه حل درست نیست تو کد هم جای کلاک gated clock داریم. کلاک باید به صورت کلی همه جا اعلام نفوذ کند اگر روی مسیرش گیت اند بگذاری مشکل رخ میدهد با یک نویز. و اگر enable قطع شود کلاک هم قطع میشود خروجی گیت اند هم خیلی کم میشود و سیستم درست کار نمیکند چون اندازه کلاک کوچک تر شده.

 به جای اینکار از یک سیگنال استفاده میکنیم همون حالت مرسومی که یاد گرفتیم که اگر مقدار en 1 شد شما بیا این عملیات را انجام بده و دیگر تاثیری رو کلاک نمیگذاریم و توان مصرفی دینامیک نداریم چون واسه وقتی هست اونکه خروجی از 1 به صفر یا برعکس بشود اگر تغییر نداشته باشیم در خروجی توان مصرفی دینامیک نداریم. یک نکته دیگه شرکت سازنده گفته en فقط تو لبه پایین رونده میتواند تغییر بکند برای اینکه تاثیری در کلاک نگذارد.

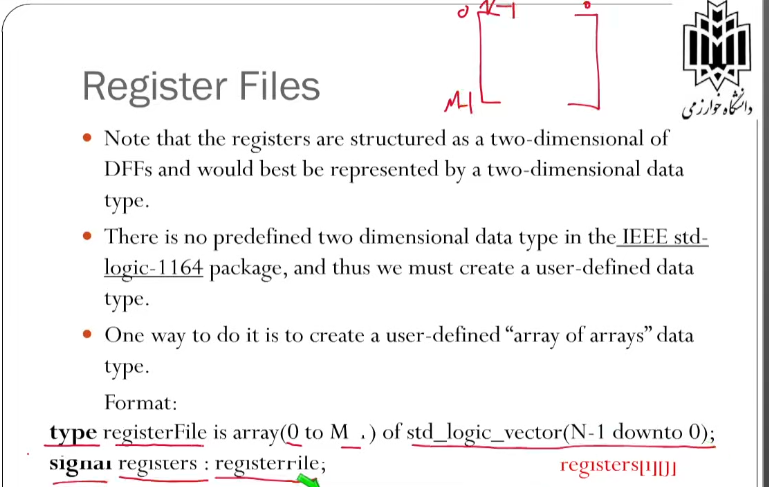
نویز روی کلاک هم تاثیری ندارد مگر توی ستاپ و هولد تایم باشد. استفاده از clock divider هم مناسب نیست چون دیگر سنکرون نیست.

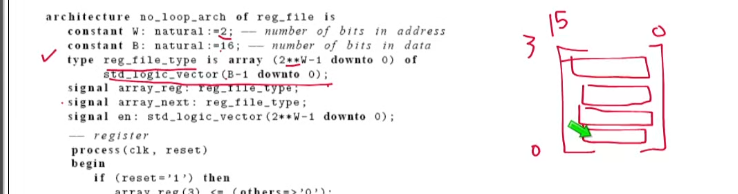
16 17 18 ویدیو رو ببین دوباره.

برای حل مشکل میتوانیم از سیگنال enable استفاده بکنیم با یک کلاک بگیم یکی در میان کار بکن به ازای هر کلاک. دقت کن اگر 3 تا مدار ترتیبی داشتیم هر 3 تا اون بخش را دارند و خوبیش این هست همه با یک کلاک کار میکنند و از این نظر state register مشکلی نداریم و یکی هستند.

کتاب پومپیچو

PMW: میخواهیم عرض پالس را کم و زیاد کنیم و تغییر بدهیم. اگر روشن خاموش شدن موتور، موتور احساس میکند همیشه روشن هست حالا اون میزان روشن بودن و ولتاژی که حس میکند میانگین زمانی هست که حس میکند روشن بودن هست. و این میاد یک پالس صفر و یک تولید میکند و میانگین این برای ما اهمیت دارد و ولتاژ تولید شده برای ما هم اهمیت دارد. یعنی مدت زمانی که 12 هست را بیشتر کردیم و مدت زمانی که صفر بوده است موتور را کمتر کردیم. و ولتاژ ما بیشتر شده است یا خروجی ما. یک مدت زمان یک بودن و یک مدت زمان صفر بودن داریم duty cycle یعنی اینکه چه مدت زمانی در یک کلاک سایکل تایم سیگنال من 1 بوده یک زمان 50 درصد یک بار 70 درصد با افزایش duty cycle میانگین ولتاژی که مدار حس میکند بیشتر خواهد شد. این قسمت duty cycle را بالا پایین میکنیم و تغییر میدهیم در واقع همین که رو پدال فشار میدهیم همین هست. باید کلاک سایکل تایم اینقدر بالا باشد که مقدار روشن خاموش شدن را احساس نکند و فکر کند همیشه روشن هست واسه همین فرکانس آن در رنج هرتز نمیتواند باشد باید در رنج 10 کیلو هرتز به بالا باشد. پیاده سازی PWM هم به این شکل هست که یک مدت زمان معین داشتیم باشیم که همان کلاک سایکل تایم هست و باید یک قسمتی 1 و یک قسمت صفر باشد این مدت زمان 1 بودن را duty cycle مشخص میکند این مشخص کردن صفر و یک هم با شمارنده پیاده سازی میکنیم شمارنده هم که یک مدار ترتیبی هست. از duty cycle هم با یک مالتی پلکسر پیاده سازی میکنیم و خروجی هم یک دی فلیپ فلاپ میگذاریم تا خروجی برای یک کلاک سایکل باقی بماند چه 1 چه 0. کلاک سایکل تایم ثابت هست. ترتیبی ها چه یک دی فلیپ فلاپ چه 2 همه در یک جا باید باشند در همان پراسس. تو یک ک لاک مینویسیم تو کل کلاک استفاده میکنیم.



فقط دقت کن که این قسمت باید اول اعداد را به unsigned بعد به integer تبدیل کنی برای پیدا کردن خانه های آرایه ها. چون std\_logic\_vector هست. مقدار دهی یک خط سیگنال constant variable را بپرس. 

خروجی این رجیستر میشود arr\_reg و ورودی آن میشود \_next. رفتاری سعی کن توصیف کنی. اولویت با توصیف رفتاری هست.

اگر آخرین عملیات صف نوشتن بود یعنی full هست با یک فلگ و اگر آخرین عملیات read باشد یعنی صف خالی هست. برای read pointer و write pointer از شمارنده استفاده میکنیم.

فصل هفتم:

در این FSM دیگر next state لاجیک مشخص نیست که گام بعدی چی هست و رندوم هست در moore خروجی فقط به state\_reg وابسته هست ولی در میلی علاوه بر آن به ورودی هم وابسته هستند جفت میلی و مور ترکیبی هستند نه ترتیبی. میلی به ازای تغییر ورودی تغییر میکند ولی مور نه فقط به state\_reg وابسته هست تاثیر نویز روی میلی بیشتر هست چون تا ورودی تغییر کند بخاطر نویز آن هم تغییر میکند ولی روی مور کمتر هست. سرعت میلی بالاتر هست تا ورودی تغییر کند تغییر میکند اما در مور باید state\_reg محاسبه شود. در مور خروجی داخل حالت چون فقط به حالت وابسته هست اما در میلی خروجی روی خط انتقال بین دو تا نود هست یا 2 تا استیت.

Conditional box ها برای خروجی های میلی هستند بیضی هست. decision box ها که جواب ترو فالس دارند برای مور ها هست لوزی هست که برای خروجی های میلی یا next state box هست. به ازای هر state یک ASM block داریم و فقط خروجی هایی که فعال میشوند را نام میبریم و اگر جایی نام نبرد یعنی صفر میشود. به ازای هر دایره انگار یک مستطیل هم داریم. خروجی مور داخل مستطیل دقت کن اگر خروجی state یکی باشد دیگر نیازی به لوزی نداریم یا decision box. خروجی میلی نداشتی نیازی به بیضی نیست. دقت کن بیضی داخل اون بلاکی قرار میگیره که از اون state شروع شده یعنی اگر از s0 رفتیم به s1 و روی مسیر انتقال y0 = 1 بود این بیضی باید داخل s0 باشد. اگر از 2 تا مسیر خارج شدیم به یک لوزی نیاز داریم. از 3 تا مسیر به 2 لوزی نیاز داریم به ازای هر شرط مثلا یکی برای a یکی برای b. یعنی اگر a فالس بود برگردد به خودت اگر ترو بود حالا b ترو هست برو به s1 فالس هست برو به s2. به ازای یک شرط نمیتوانیم بیش از یک خروجی داشته باشیم. ورودی فقط به استیت هست ولی از هر جایی میشود خارج شد. انتقال بین 2 بلاک در مدت زمان یک لبه بالا رونده هست و مدت زمان داخل یک بلاک به اندازه یک کلاک سایکل تایم هست.

مور: فقط به استیت وابسته هست و غیر مستقیم با کلاک سنکرون هست و فقط به اندازه تاخیر اوت پوت مدار مور هست بعد از کلاک و بعد از اون خروجی آماده هست و تقریبا سنکرون هست.

میلی: با کلاک سنکرون نیست. هر مداری که میلی تولید بکند مور هم تولید میکند. توانایی یکسان هست.

فقط خصوصیات توانی و area متفاوت هست. کنترلر آنها هم فرق میکند. دقت کن بعد از اینکه لبه بالا رونده اومد مستقیم وارد استیت بعدی که edge هست نمیشیم یک مدت زمانی به نام کلاک تو اوت پوت یا TCQ طول میکشد. اگر کاربر دستش را بکشد دوباره به حالت اول برمیگردیم ولی اگر برندارد وارد استیت وان میشویم که فقط یکی حساب میکند. در زمان T2 چون تو حالت edge هستیم اونجا خروجی p1 یک هست چون فقط به استیت بستگی دارد و p1 میتواند edge detector بکند و counter در زمان t2 یکی افزایش پیدا میکند.

در میلی چون به ورودی هم وابسته هست تاثیر مستقیم میپذیرد یعنی خروجی میلی یک میشود در لبه بالا رونده بعدی ما وارد استیت one میشویم چون خروجی نداریم صفر میشود دقت کن این واسه وقتی هست که هنوز کاربر دستش روی کلید هست.

مدار میلی سایز کمتری نسبت به مور دارد. در مدار میلی نویز و گلیچ تاثیر میگذاند ولی مور نه. میلی سریعتر هست از مور اون edge detection مثلا. عرض پالس مور دقیقا یک کلاک سایکل هست ولی میلی حداکثر یک کلاک سایکل یعنی کمتر هم میتواند باشد. برای نوشتن در SRAM چون نیاز داریم عرض پالس دقیقا یک کلاک سایکل باشد مور بهتر هست برای فعال کردن سیگنال شمارنده هر دو خوب هستند ولی میلی بهتر هست چون سریعتر هست.

FSM به ASM و ASM به ترتیبی. State\_reg و state\_next بر اساس دیتا تایپ هستند. ورودی های next state بر اساس ورودی های خودش مثل memory rw هست و burst که میگه یهو چند تا خونه بخون و بعد بر اساس لوزی ها یا decision box های ASM هست. هم میتوانی با process پیاده سازی کنی که 4 تا ورودی رو بگذاری داخل پراسس یا ترکیبی پیاده کنی.

مور همونجا مشخص هست به state\_reg بستگی دارد یا همان استیت یعنی وقتی از استیت اصلی اومدیم بیرون بر اساس اون مور را میسازیم مثلا یکی برای memory هست یکی برای output. برای میلی یکی به state وابسته هست یکی ورودی چه ورودی هایی تاثیر میگذارد؟ اون لوزی هایی که روی بیضی تاثیر میگذارند اون هارو بردار ببر بچسبون به عنوان ورودی یعنی اون شرطی که داخل لوزی هست اون روی بیضی تاثیر میگذارد مثل mem و rw.

برای مور فقط باید state\_reg رو بریزی چیز دیگه باشد دیگه مور نیست سیگنال ها هم باید صفر باشد خیر مواقعی که میخواهد مقدار بگیرد خروجی مور بیرون باکس های مستطیل هستند دقت کن و داخل idle اصلا خروجی نداشتیم. برای نوشتن write و برای خواندن read ها را 1 میکنیم.

راه حل برای گلیچ:

Clever state assignment: یک state را جوری مقدار دهی کن که هوشمند باشد مثلا بر اساس استیت ها مقدار دهی کن خروجی را. با این کار مور خروجی را کاملا حذف میکنیم و کلا یک سیم میکنیم به جای گیت و مثلا میگیم بیت پوزیشن سوم output باشد و بیت پوزیشن دوم write باشد. یک حسن دیگر این هست که مور به صورت غیر مستقیم با کلاک سنکرون هست حالا میگیم مستقیم سنکرون هست چون دیگر گیت نداریم و تاخیر نداریم و فقط یک سیم داریم. و t output از فرمول T co = TCQ + T output حذف میشود چون دیگر گیت نداریم و تاخیر نداریم. برای استیت های کم.

روش بعدی look ahead output buffer هم روی مور هم روی میلی جواب میدهد ولی معمولا روی مور هست روی میلی نیست و در همه جا استفاده میشود و میگوید روی خروجی مور و میلی دی فلیپ فلاپ بگذار و هر نویز و گلیچ باشد از بین میرود مگر اینکه نویز تو بازه ستاپ تایم یا هولد تایم باشد که اون هم زمانش خیلی کم هست و مشکلی نیست مسئله این هست که تاخیر به اندازه یک کلاک سایکل دیر تر آماده میشود توی PWM هم بخاطر همین بود استفاده میکردیم یکی بخاطر ثبات پالس بود یکی بخاطر حذف گلیچ. State reg همان state\_next کلاک قبلی هست خوب به جایی اینکه ما از state reg بخوانیم بیایم و از state\_next بخوانیم و این دور را بزنیم یک کلاک سایکل تایم کم میشود و خنثی میشود با اون یک کلاک سایکل دی فلیپ فلاپ که تاخیر را 1 کلاک برده بود بالا و حالا هم زمان تغییر نکرد هم نویز حذف شده است. تنها فرق در کد این هست که اوت پوت لاجیک قبلا تنها یک ورودی به نام state\_reg داشت الان state\_next را دارد و هر جا reg بود next میگذاریم و اون reg سمت چپ را حذف میکنیم از معادلات. در خروجی هم یک دی فلیپ فلاپ باید قرار بدهیم پس یک buf\_next, buf\_reg داریم. این قسمت کار بکن. Process دیگر به state\_reg وابسته نیست به نکست وابسته هست و به state\_next حساس هست و هر جا reg دیدی next بگذار.

فصل هشتم:

طراحی سلسله مراتبی: مدیریت بکنیم سیستم ها را که بزرگ هستند. راه حل اول divide and conquer هست یعنی یک سیستم را به ساب سیستم های مختلف بشکنیم.

Component همان entity هست که هویت دارد منتها میخواهد embedded بشود داخل یک سیستم دیگر. در واقع نقش یک ماژول در یک سیستم بزرگتر خواهد داشت. و بعد ورودی و خروجی های پورت آن را باید مشخص کنیم که به چه چیزی در داخل اون entity وصل بشود. کجا کامپوننت قرار میگیرد؟ همان جا کنار سیگنال ها یعنی قبل از begin در architecture. سمپل گیری از کامپوننت چه شکلی هست؟ مثلا پورت های formal رو چطوری به پورت های واقعی استفاده بکنیم؟ یک لیبل میگذاریم بعد : بعد نام کامپوننت بعد تابع port map را مینویسیم. یکی به وسیله name association هست یعنی این پورت فورمال به کدام پورت واقعی وصل شده است در اینجا ترتیب برای من مهم نیست. در positional association جا برای ما مهم هست و سیگنال را جای پورت فورمال میکنیم. یعنی سیگنال های واقعی جای پورت فورمال قرار میگیرند در همان port map مشکل این هست که جا به جا بشود یا دیتا تایپ مخالف باشد به مشکل میخوریم.

تبدیل entity به component: اول is را حذف میکنیم جای entity ها component میگذاریم آخرش هم به جای نام entity خود end component قرار میدهیم. در پورت مپ سمت چپ ها پورت های کامپوننت و سمت راست سیگنال ها هستند. اگر pulse => open بگذاری یعنی از این پورت این کامپوننت نمیخواهی استفاده بکنی. شرط هم مالتی پلکسر هست این 100 بار. اگر ورودی را open بگذاری یعنی اجرای عملیات را زیر سوال بردی و هیچ موقع نمیتوانی برای پورت ورودی بگذاری.

Generic: یکی از روش های دیگر مثل کامپوننت هایی که میخواستیم داخل یک کد دیگه اضافه کنیم. این برای این استفاده میشود که میخواهیم طراحی پارامتری باشد یعنی فعلا یه چیز بنویسیم موقع سنتز اون مقدار width را بدهیم حتی برای کامپوننت هم باید بدهیم. برای این از جنریک استفاده میکنیم که برای انتقال اطلاعات به entity و کامپوننت هست مقدار های پارامتری. در واقع پورت های آنها را پارامتری تعریف میکنیم و generic باید قبل از پورت ها باشد. همه چیز مثل قبل هست فقط داخل entity قبل پورت میایم و generic ها را تعریف میکنیم و بعد پورت را بر اساس پارامتر که متغیر پذیر هست طراحی میکنیم. اگر از 0 تا 9 بخواهد بشمرد از mod 10 خواهد بود. Width همان تعداد خروجی هست مثلا خروجی 4 بیتی هست. در کامپوننت ها هم فقط کافی هستش که نمونه بگیریم بعد قبل port map کردن بیایم و generic ها را تعریف کنیم از ; هم نباید استفاده کنیم.

قسمت سوم طراحی سلسله مراتبی:

Generate statement: دقت کن نه تنها کد generic میشود حتی تعداد کامپوننت ها هم generic میشود. مثلا 10 تا از این کامپوننت ولی نمیخواهیم 10 تا بنویسیم یا شرطی بکنیم instanciation ها را. این برای عملیات و hardware های تکراری استفاده میشود. for loop داخل process بود ترتیبی بود و این در architecture استفاده میشود و concurrent هست. برای عملیات فقط تکراری هست. دقت کن چه تو for loop چه تو loop-range باید static باشد ولی اینجا میتواند generic باشد یعنی متغیر باشد. دقت کن مقدار دهی سیگنال ها چه یکی چه ده تا همزمان با هم انجام میشوند چون دیگر داخل پراسس نیستیم. For loop داخل پراسس ترتیبی. For generate داخل architecture و موازی.

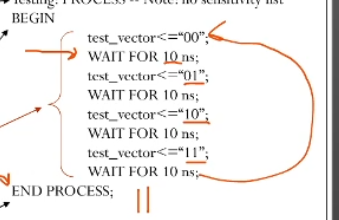
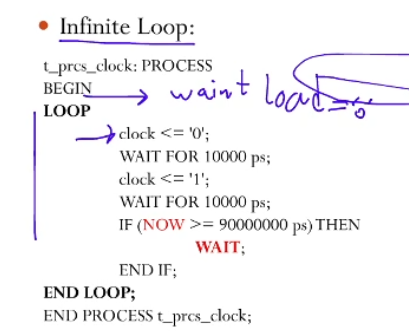
If generate statements: ساختن hardware شرطی میشود. داخل بدنه for هست.

Natural بپرس.

فصل نهم:

در شبیه سازی قرار نیست hardware ساخته شود و entity داریم بدون پورت ورودی و خروجی و یک architecture فقط داریم. برای تست اول باید اون کامپوننت مورد نظر را نمونه گیری بکنیم.

Driver model اونی هست که مثلا جواب را دارد ورودی میدهد بر اساس خروجی میسنجد. دقت کن after ها به زمان اولیه ربط دارند نه زمان قبلی مثلا میگیم اول 13 ثانیه بعد زمان اول بعد 50 ثانیه بعد زمان اول.

 wait داخل پراسس هست. positional بر حسب ترتیب ورودی و خروجی هست که در پورت تعریف کردیم. Wait خالی یعنی تا ابد و هیچکاری نکن. صفحه 22 بخون. 

به هر سیگنال فقط داخل یک پراسس میتوانی مقدار بدهی. عملیات ضرب همان عملیات and کردن است. در یک کلاک سایکل در دی فلیپ فلاپ هم میتوانی بنویسی که در لبه هست هم میتوانی تو کل کلاک بخوانی. در این دی فلیپ فلاپ ها 2 نوع چیز ذخیره میکنیم 1. دیتایی که توسط استیج قبلی بدست آمده است. 2. یک سری دیتایی هستند که ممکن هست در استیج یک یا دو استفاده نشوند و در استیج سوم باید ذخیره شوند. دقت کن یک تاخیر ستاپ تایم داریم در استیج 1 به اندازه ستاپ تایم باید قبل اون آماده بشود و تاخیر دوم TCQ هست که ورودی روی خروجی قرار بگیرد. پس میشود کل زمان ها هر استیج بعلاوه 4 ضربدر این 2 تا تاخیر. حاصل محاسبات از نوع NEXT و اونی را که میخوانیم و استفاده میکنیم از نوع REG. و PP1 ها جز NEXT هستند.

PP1 از نوع نکست و PP1<= PP0 + BP1 یعنی PP0 از نوع REG هست. دقت کن مقدار دهی سیگنال ها از نوع NEXT هستند و سمت راستی ها که داریم میخوانیم و استفاده میکنیم از نوع REG هستند. داخل NEXT مینویسیم و از نوع REG میخوانیم. دقت کن فقط a, b قرار هست از پایپ لاین عبور کنند پس نیاز به REG, NEXT دارند نه BV0 و بقیه که. دقت هم کن PP1 هم نیاز به NEXT, REG دارند چون قرار هست بعدا استفاده بکنیم یا در آن بنویسیم. دقت کن اگر ریست بزنیم همه REG ها صفر میشوند در غیر این صورت سر لبه بالا رونده کلاک همه NEXT ها داخل REG ریخته میشوند. سمت چپی ها NEXT ها هستند و سمت راستی ها REG ها.

فصل یازدهم:

Synchronizer: میاد ورودی را جوری تنظیم میکند که تو بازه ستاپ تایم و هولد تایم ورودی تغییر نکند و انگار یک نوعی زمان برای ما میخرد تا حالت metastability رد شود. با یک نویز ساده این ناپایداری رخ میدهد و همه سیستم ها این مشکل را دارند. هر چه resolution time بیشتر باشد احتمال ناپایداری کمتر میشود این همان زمانی هست که ما به حالت پایدار برسیم. این ها چیزی جز دی فلیپ فلاپ نیستند چون ورودی آسنکرون هست و هر لحظه تغییر میکند.

نرخ synchronization failure به 2 چیز وابسته هست 1. اینکه ورودی در اون بازه ستاپ تایم و هولد تایم تغییر بکند w اندازه ستاپ تایم هولد تایم هست و میخواهیم بدانیم چه کسری هست از کلاک و بعد مثلا ممکن هست تو هر 10 تا کلاک این اتفاق بیفتد پس یک نرخ کلاک یا f clk هم نیاز داریم. و 2 اینکه بعد از اتفاق افتادن ناپایداری چه قدر احتمال وجود دارد resolution time نتواند آن را حل کند و ما هنوز مشکل داشته باشیم. ورودی syncornizer باید گلیچ فری باشد چون هر تغییری در فرستنده ایجاد شود ما فکر میکنیم داده درست هست چون نمیدانیم چه خبر هست. پس ورودی آنها باید outpot buffer بگذاریم تا نویز را حذف کند. سینک کردن باید فقط یکجا باشد. سینک خروجی را گارانتی نمیکند فقط ناپایداری را کم میکند. Multiple related signal: یعنی همه با هم مفهوم دارند و تکی مفهوم دارند کجا استفاده میشود؟ یعنی ورودی و خروجی بر اساس این سیگنال های تک بیتی تشکیل میشود و کل اینها با هم مفهوم میدهد نه یکی اینها در کجا استفاده میشود؟ همین address bus data bus مثلا میگی دیتا باس 16 بیتی هست یعنی 16 تا تک بیتی و تک به تک اینها مفهومی با همدیگر ندارند. پس جایی که بحث باس ها مطرح بود و آسنکرون بود روی اینها عمل سینک کردن نکن چون خروجی های تولید شده ممکن است مفهومی نداشته باشد. ناپایداری یک پدیده آنالوگ هست و هر موقع میتواند رخ بدهد بنابراین با random variable ها سرو کار داریم. تنها چیزی که میتوانیم تغییر بدهیم و تنظیم کنیم Tr یا resolution time هست.

خروجی گارانتی نمیشود در synchronizer ها و فقط سعی میکنیم ناپایداری به حداقل برسد. یک مسئله دیگر داریم به نام edge detector که خروجی را گارانتی کند که قبل از سینک کننده بیاد همان ورودی که قبل سینک کننده بوده را به کامپوننت بعدی بدهد. چرا ناپایداری؟ چون از ورودی خبر نداریم هر لحظه ممکن هست تغییر کند و دچار ناپایداری میشویم.

Edge detection: اگر فعلی 1 هست و کلاک قبلی صفر هست یعنی لبه بالا رونده داشتیم. اگر همچین چیزی داشتیم یک پالس تولید بکن با کلاک تولید میکنند همخوانی داشته باشد با دی فلیپ فلاپ مقداری قبلی را نگه میداریم. پس 2 تا مقدار داریم فعلی و قدیم از گیت اند باید استفاده کنیم یک نات هم میخواهیم روی past بگذاریم لبه بالا رونده و روی now بگذاریم لبه پایین روند را پیدا میکند. خروجی این یک پالس هست اندازه این پالس به اندازه کلاک سیستم برابر هست. سیگنال های آسنکرون رو نباید همینطوری استفاده کنی روشون گیت بزاری چون ممکن هست نویز بگیرند. این یک مشکل و مشکل بعدی این هست کلاک نباید به چیز دیگری وابسته باشد رهبر سیستم هست و مشکل ایجاد میشود و دیگر سنکرون نیست.

روش handshaking: فرستنده میخواهد دیتا بفرستد باید سیگنال بفرستد request و گیرنده که ack میکند این درخواست را آماده دریافت داده میشود. گیرنده سر لبه کلاک خودش میفهمد که درخواست اومده و داده را برمیدارد و ack میفرستد و سر کلاک بعدی فرستنده میفهمد که گیرنده به درستی داده را دریافت کرده اگر ack درست باشد و میاد request را برمیدارد بعد از این گیرنده هم میبیند دیگر درخواستی نیست اون هم ack را برمیدارد. نیاز هست در این 4 مرحله حالات قبلی خودش را حذف میکند پس به یک سری state machine نیاز داریم. مشکل این هست که ممکن هست کلاک خیلی کند باشد و کلاک دریافت کننده خیلی سریع باشد و ما دریافت نکنیم پس نیاز داریم که از دریافت کننده feedback بگیریم که روش handshake را معرفی کردیم. سیگنال ready اگر فعال شود فرستنده میتواند داده بفرستد. ارسال کننده در این مرحله سیگنال request out که همان نوشتن هست فعال میکند و دیتا هم روی باس میگذارد. روی ورودی سینک کننده به هیچ عنوان نباید نویز داشته باشد و از out put buffer یا look ahead میگذاریم همان دی فلیپ فلاپ هست و روی مسیر req\_out میایم و دی فلیپ فلاپ میگذاریم. Ack\_out هم باید گلیچ فری باشد نویزی روی آن نباشد پس دی فلیپ فلاپ قرار میدهیم. ورودی ها را در decision box ها مینویسیم.