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.