**جزوه اول:**

ارزش یک رقم در هر مبنا میشود r به توان n-1 که در آن r مبنای عدد هست و n شماره اون عدد هست مثلا عدد 2 شماره آن میشود 3 یا صد گان، توی منفی ها دیگه n-1 نداری فقط n داری مثلا ارزش a-1 میشود یک دهم یا -1، اگر ارقام رو ضربدر ارزش ها بکنی مقدار اون عدد در مبنای 10 بدست میاد، در تبدیل مبنا ها وقتی به تکرار رسیدی دیگه ننویس یعنی عدد تکراری بود ننویس،

برای چک کردن اعداد مختوم در قسمت اعشار به 5 بیشتر دقت کن چون اونها هستند که صفر میسازند، بعضی مبنا ها را میتوان به هم تبدیل کرد اگر یک مبنا توانی از مبنای دیگر باشد و برعکس. و به ازای اون توان مثلا دو به توان چهار بیا برای مبنای 16 به ازای هر 4 تا رقم بگذار و ... ، و بر عکس از 16 به 2 هر 4 تا رقم رو یکی بکن، توی تبدیلا رقم کم اوردی توی قسمت صحیح سمت چپ عدد و توی اعشار قسمت راست عدد صفر بگذار.

برای تبدیل توانی اگر از بالا رفتی به پایین باید بیت اضافه کنی چه قدر؟ به توان مثلا از 16 به 2 باید 4 تا 4 تا رقم اضافه کنی به ازای هر رقم. اگر از پایین به بالا رفتی همین برعکس.

مختوم بودن به این معنا هست که اولین رقم سمت راست اعشاری صفر شود. وقتی هم که مثلا عدد دوباره اعشاری شد 0.6 یعنی اینکه دیگه نمیتونی ادامه بدی. اگر هم در تبدیل اعشاری در یک مبنای دیگر قسمت صحیح صفر شد دیگه نمیشود ادامه داد.

در کسری ها اول ببین به در صورت چند تا مخرج هست و بعد اون قسمت را به عنوان صحیح و بقیه را به عنوان اعشاری رفتار کن.

توضیح روش 2: اگر یک عدد رو مبنای خودش به توان 2 برسونی دو تا شیفت میخوره اونور بعد بیا به توان دو خودش برسان دو طرف را بعد منهای بالا کن. بعد حاصلی که مربوط به مبنای غیر 10 هست را ببر به مبنای ده. یا مبنای مورد نظر. بعد ایکس را بدست بیار. روش 1 همون دنباله تصاعد هندسی برو بعد در بزرگترین توان ضرب کن بتونی کسر ها را ساده کنی.

برای بدست آوردن بزرگترین عدد n رقمی در مبنای r: مبنا به توان تعداد رقم منهای یک.

برای بدست آوردن بزرگترین عدد اعشاری n رقمی در مبنای r: یک منهای مبنا به توان منفی رقم.

**جلسه دوم:**

عدد ایکس در مبنای ار چند رقم دارد؟ سقف لگاریتم ایکس بعلاوه 1 در مبنای ار یا کف لگاریتم ایکس در مبنای ار بعلاوه 1. بزرگترین عدد N رقمی در مبنای ار؟ ار به توان N منهای یک. بزرگترین عدد اعشاری هم میشود 1 منهای ار به توان منفی N.

اعداد n رقمی در مبنای r به مبنای t به سقف n در لگاریتم r در مبنای t رقم نیاز داریم. همون بزرگترین عدد بعد هم فرمول تعداد ارقام.

مکمل r-1 خواستی منهای بزرگترین عدد همون رقمی در همون مبنا بکن. R خواستی به صفر ها دست نزن اولین رقم را از r و بقیه را از r-1 کم کن. حتی اعشاری ها هم همین رفتار را داشته باش.

مکمل R خواستی مکمل R-1 رو بدست بیار به راست ترین مقدار یکی اضافه کن بدست میاد.

تو 3 تا روش برای نشان دادن اعداد منفی در قسمت مثبت ها یکی هستند و در قسمت منفی ها این اشتراک را دارند که چپ ترین بیت آن 1 هست برای اعداد منفی. در روش علامت مقدار چپ ترین عدد فقط نشان دهنده علامت عدد هست. در روش مکمل یک میگیم منفی میخوای نشون بدی برعکس کن. تو مکمل 2 هم که روش میدونی از راست برو به صفرا دست نزن به اولین یک رسیدی برعکس کن دقت هم کن این روش برای همه مبنا ها هست که صفرا بزار کنار به اولین عدد برس.

تو سیستم علامت مقدار بیت سمت چپ اصلا مقداری ندارد و برای تبدیل مبنا حساب نمیشود، تو مکمل 1 عین مکمل 2 هست یعنی منفی کن بعد از اون مقدار یکی کم کن مثلا اونجا 32 بود تو بی علامت اینجا -31 هست. تو مکمل 2 اون بیت سمت چپ مثل بی علامت مقدار دارد منتها منفی هست مثلا اونجا 32 بود اینجا -32 هست.

تو سیستم های مکمل 1 و علامت مقدار صفر مثبت و منفی داریم. با n بیت در سیستم بی علامت و مکمل 2 ما 2 به توان n عدد میتوانیم نمایش دهیم ولی در مکمل 1 و علامت مقدار دو به توان n منهای یک هست بخاطر همونکه صفر را به دو صورت مثبت و منفی نشان میدهیم.

جلسه سوم:

عدد مکمل 2 را یکبار دیگه به مکمل 2 ببری قرینه میشود. اگر بیت سمت چپ عدد در مکمل 2 را تکثیر کنی مثلا اگر 1 بود 1 را تکرار کنی و اگر صفر بود صفر را تکرار کنی عدد تغییر نمیکند.

در سیستم مکمل 2 میتوانی به راحتی اعداد را زیر هم بنویسی و جمع و کم کنی و ساده هست ولی بقیه سیستم ها به این شکل ساده نیست. برای overflow چک کن که عدد تو مکمل 2 اون چیزی که انتظار داریم باشه و از اون محدود نزنه بیرون. مثلا جمع دو عدد مثبت منفی شود. مثلا تو مثال اول جمع دو تا عدد شده -13 از اونجایی که -13 با 4 بیت قابل نمایش نیست ما میگویم سر ریز رخ داده هست و خطا شده هست.

تشخیص سر ریز در مکمل 2: روش 1: اگر جمع دو عدد مثبت منفی شود یا جمع دو عدد منفی مثبت شود سر ریز است. جمع یک عدد مثبت و یک عدد منفی هیچوقت خطا رخ نمیدهد. روش 2: اگر رقم نقلی وارده به بیت سمت چپ با رقم نقلی خارج شده از آن یکی نباشد سر ریز است.

قرینه= مکمل 2، پس فرمول تفریق در سیستم مکمل 2 به این شکل هست که دومی رو قرینه کن و بعد با هم جمع کن و بعلاوه 1 بکن. تا تو هزینه کمتر بشود.

اون بعلاوه یک رو به عنوان کری اول بگذار، تفریق دو عدد هم علامت همیشه نتیجه آن داخل range هست و سر ریز ندارد، و علامت اولی با علامت حاصل فرق داشت سر ریز رخ داده است.

دقت کن وقتی دومین عدد از سمت راست صفر بود برای تبدیل به مکمل 2 کری به وجود میاد که ما نمیبینیم، وقتی نات میکنیم مکمل 1 میکنیم در واقع و بعد بعلاوه 1 میکنیم.

سیستم مکمل 1 در هیچ سیستمی استفاده نمیشود. Sign extend که بیت سمت چپ رو بخوای تکرار کنی عدد عوض نمیشود در مکمل 1 هم صحیح هست. تو جمع مکمل 1 حاصل جمع را یکبار دیگر با کری جمع کن.

تو تفریق دومی رو نات کن جمع کن فقط تو جمع با کری یکبار دیگه جمع کن. در جمع بی علامت اگر فضای ما n بیتی باشد کری نشانه سر ریز هست. منها در بی علامت هم مثل علامت 2 هست. منتها اونجا دیگه نات نکن مکمل 2 بگیر. این واسه تفریق با روش جمع بود. روش قرض اونجا کری به معنای قرض گرفتن هست. و فقط تو مرحله آخر رو نگاه میکنیم و اونجا اگر کری یا قرض صفر شود یعنی طبیعتا عدد اول از دوم بزرگتر بوده هست و برعکس. برای تبدیل اعداد مثبت به منفی مکمل 1، مکمل 1 بگیر یا نات کن و مکمل 2 را مکمل 2 بگیر.

جلسه سوم:

جلسه چهارم:

اگر خواستی ببینی یک کدی وزن دارد به رقم هاش یک مقداری نسبت بده بعد ببین به تناقض میخوری یا نه مثلا بگو اولین رقم a باشه بعد مثلا 5 مقدارش بعد برو بقیه ببین این 5 گذاشتی به تناقض میخوری یا درست هست، مکمل، مکمل یک عدد میشود خودش. خود مکمل یعنی اگر یک نمایش از عدد رو بگیری نات کنی یا مکمل 1 کنی بشود مکمل 9 که در EXCESS3 این قضیه رخ میدهد و یک مزیت هست و سخت افزار کمتری دارد چون در تفریق همینطوری میتونی کم کنی.

چطوری میتواند خود مکمل باشد؟ باید جمع وزن ها 9 شود. باید عبارات به 9-a برسند تا خود مکمل شوند.

کد گری: آینه کن به بالای آینه صفر بده به پایین 1.

XOR: تابع فرد هست یعنی وقتی زوج باشند خروجی صفر هست اینطوری بگو اگه Aو B برابر بودند میده صفر، جور دیگه اینه که جمع بزن معمولی بعد باقی مانده را به 2 بگیر.

تبدیل باینری به گری از نوع انعکاسی: اولین بیت از سمت چپ رو معمولی بیار پایین بقیه رو با سمت راستیش XOR کن.

گری به باینری: روش اول که هیچی روش دوم به این شکل که از هر جایی که کد گری رو میری سمت چپ را XOR کن و حاصل را تو همون راستا پایین بنویس.

جلسه پنجم:

نحوه تشخیص یک بیت خطا: distance: اختلاف بین 2 بیت هست، مثلا اگر اختلاف دو تا عدد 4 هست یعنی 4 بیت باید تغییر کند تا عدد تبدیل شود به اون یکی و توی مبنا هایی که تا الان خوندیم حداقل فاصله بین هر عدد 1 هست، یعنی 1 بیت تغییر کند میشود اون یکی، برای تشخیص خطا واسه یک بیت باید اعداد حداقل 2 فاصله داشته باشند برای سیستم های تک بیتی پریتی اضافه میکنیم بر حسب یک اگر تعداد یک ها زوج بود صفر میشود بیت پریتی و اگر فرد بود 1 میشود بیت پریتی دقت هم بکن و با این سیستم تعداد فرد خطا میشود تشخیص داد ولی زوج نمیشود اونجا فقط یک خطا تشخیص داده میشود، ولی نمیشود تشخیص داد کدوم بیت بوده است که غلط هست فقط امکان تشخیص داری تصحیح نداری. فرستنده دستگاه XOR داره که میاد چک میکنه اگه 1 شد یعنی تعداد 1 ها فرد هست و 1 میگذاره یک قطعه کوچک سخت افزاری هست در گیرنده هم میاد همه ورودی ها را با هم XOR میکند.

کدی که فاصله اش d است تا d-1 خطا را میتواند تشخیص دهد و تا کف d-1 /2 را میتواند تصحیح کند.

کد همینگ : برای تصحیح خطا هست. برای اضافه کردن بیت ها که فرمول مشخص هست فقط دقت کن برای عدد ها بالاتر اون آزمایش خطاست یعنی باید عدد بگذاری تا معادله درست در بیاد مثلا اون عدد 27 حداقل به 6 تا برای نمایش دادن نیاز دارد.

پایان فصل اول و جزوه اول.

**فصل دوم:**

منها بسته نیست چون در مجموعه اعداد طبیعی 1 منهای 2 منفی میشود پس نسبت به منها بسته نیست. برای نشان دادن جبر بول باید 6 تا ویژگی را اثبات کنی براش. ضرب اولویتش از جمع بالاتر هست.

برای شرکت پذیری به این شکل فکر کن که علامت ها باید یکسان باشند. برای توزیع پذیری به این صورت فکر کن که یک علامت متفاوت با یک علامت دیگر باید شرکت کند. عضو خنثی یعنی اینکه اگر یک عدد را در جبر بول با خودش جمع کنی یا ضرب کنی خودش بشود. البته دقت کن اون عاملی که جمع میشود یا ضرب میشود باید عامل خنثی عمل اول یا جمع و عامل خنثی عمل دوم یا ضرب باشد.

مکمل یعنی اینکه اگر یک عدد را با نات خودش ضربدر کردی عامل خنثی جمع و اگر یک عامل را با نات خودش جمع کردی عامل خنثی ضرب بدست بیاد. عامل خنثی را میتوانی صفر یا 1 حساب کنی مثلا در جمع صفر حساب کن و در ضرب 1 حساب کن بهتر هست البته از نظر خودت.

XOR ، OR انحصاری هست یعنی وقتی میگیم یا این یا اون یا هر دو 1 بود بشود 1، اینجا فقط میگیم یکی فقط 1 باشد.

XNOR نات XOR هست. برای کوچک بزرگ برو جایی ببین که یکی 1 شده اون یکی صفر شده توی همون سطری که 1 شده هست در F ها ضابطه هم که مشخص هست. به همین توابع، توابع ممنوعه هم گفته میشود چون نات لیترال اون یکی لیترال را از بین میبرد.

XOR و XNOR رو به صورت POS فکر کن یا جمع حاصل ضرب ها.

اون جدول و این پایین توابع مکمل رو یکبار سریع ببین.

جلسه 2.2:

دقت کن تو دوگان متغیر ها را دست نمیزنی یعنی نات بود نمیکنی معمولی یا معمولی بود نمیکنی نات. XOR دوال آن میشود XNOR و برعکس. XOR مکمل آن میشود XNOR و برعکس. دوال NAND، NOR هست و برعکس به ضرب یا جمع تبدیل کن.

تابع سلف دوال با دوگان خودش مساوی هست یا هم ارز یا متحد دو لیترال وقتی برابر هست که از نظر جدول درستی هم برابر باشد.

SOP همون اند اور هاست خواستی بزاری فقط دنبال 1 بگرد. یعنی چی یعنی مثلا داخل هر کدوم از آن مینترم ها اگر هر دو 1 بودند اونجا ها 1 بگذار یعنی بگرد هر جا اونا 1 بودن 1 بزار.

POS ها صفر های آن ها بدست میان چرا چون هر جا هر کدوم صفر شود کل تابع صفر میشود. چون اند هست دیگه پس بگرد ببین کجا ها جفتش صفر هست. چون اور هست دیگه اور وقتی صفر هست که هر دو صفر باشند.

اصل دوگانگی: اگر یک تابع با تابع دیگری برابر بود نتیجه بگیر دوال آن ها هم با هم برابر هستند. خود توانی یعنی هر کی با خودش اند و اور شود خودش میشود. طبق اصل دوگانگی ما از 6 تا خاصیت جبر بولی دو تا داریم خاصیت جذب این هست که جمله کوچک به شرطی اینکه در جمله بزرگ حضور داشته باشد آن را نابود میکند. اگر دمورگان یک جا بود اون رو حذف کن و تابع اصلی که دمورگان آن را حذف کردی تو جواب بگذار.

XOR هر چیز با صفر خود آن هست و با 1 نات آن هست.

توی قضیه شانون باید در تابع جایگذاری کنی. شانون را میتوانی برای دو تا هم استفاده میکنی و میتوانی در مالتی پلکسر استفاده کنی.

جمله ضرب یعنی توش عمل جمع وجود ندارد و میتواند متغیر تنها باشد. یک لیترال تنها هم sop هست هم pos، یک جمع و ضرب تنها هم sop هست هم pos. دقت کن عبارت خودش شاید به هیچ فرمی نباشد ولی اگر بازی کنیم به چیزی تبدیل میشود پس اگر خودش بود بگو هیچی. دوال هیچ عبارتی به خودش ندارد.

برای تبدیل sop به اون یکی با جدول کارنو راحت میشود.

مینترم اگر یک بودند خودش اگر صفر بود نات آن. ماکسترم نات مینترم هست. یعنی اگر صفر بود خودش و اگر 1 بود نات آن.

ضرب دو تا مینترم که شماره های آن یکی نیست صفر میشود. مینترم سه فقط در حالت 3 یک هست و در بقیه حالت صفر هست. هر مینترم در یک حالت فقط 1 هست اون هم شماره خودش هست و در بقیه حالت صفر میشود. ماکسترم برعکس هست یعنی همه جا 1 هست جز حالتی که با شماره خودش برابر هست صفر میشود پس جمع و ضرب بی معنی هست. اگر یک مینترم با ماکسترم ضرب بشود میشود مینترم و اگر جمع بشود میشود ماکسترم. ضرب یک مینترم با ماکسترم خودش صفر میشود چون مکمل هستند و جمع آنها یک میشود.

یکی از فرم ها را بدست بیاری بقیه رو میتوانی از رو شماره ها واسه اون یکی فرم ها بدست بیاوری و هر دو این فرم ها یک تابع را نشان میدهند. مینترم شامل همه حروف هست.

با روش 2 sop را خیلی راحت به مینترم و pos را به ماکسترم میتوانی تبدیل کنید. تو روش کدینگ رقم واقعی رو بنویس یعنی درسته نات شده ولی ببین رقم واقعی چی بوده که نسبت به اون فرم ما نات گذاشتیم. یعنی اگر sop بود باید صفر بگذاری چون نات کرده 1. و برعکس.

جمع همه مینترم ها میشود تاتولوژی و جمع همه ماکسترم ها میشود تناقض همیشه صفر هست. وقتی XOR دادن فکر کن کجا ها تعداد 1 ها فرد هست چون جواب 1 میشود همون ها میشود مینترم اونها میشود.

مینترم: جمع، sop

ماکسترم: ضرب، pos شک دارم.

جمع یکی 1 باشد کافی هست ضرب حتما باید هر دو باشند چون وقتی مینترم داده یعنی در آن خانه ها یک هست دیگه.

جلسه چهارم:

واسه تاخیر بیا بزرگترین مسیر را بگو. مدارات دو سطحی تاخیر آنها دوبرابر یک گیت هست. چرا میگن دو سطحی؟ چون مثلا یک سطح اند یک سطح اور. دقت کن در ورودی گیت ها یک لیترال با نات آن هست. SOP و POS به صورت دو سطحی هستند.

واسه پیچیدگی سخت افزار تعداد ورودی ها را بشمر. با یک فاکتور گیری میشود سخت افزار را کاهش داد ولی تعداد سطوح معمولا افزایش پیدا میکند. پس باید یک مصالحه برقرار کنیم. هدف از ساده سازی همان قوانین جذب هست. یک راه این هست. باید به دو فرم sop یا pos ساده کنیم.

برای sop ها یا مینترم ها فقط میتوانی از تکرار یا مجاور استفاده کنی. به راحت میتونی با جدول کارنو حل کنی. در جدول کارنو هر خونه با خونه اطرافش مجاور منطقی هست یعنی یکی رو میتوانی خط بزنی و ساده کنی. در کارنو سلول های مجاور در یک بیت اختلاف دارند و باید طبق دسته هایی به طول زوج اون متغیر مشترک را بنویسیم. کارنو را هر شکلی بکشی مشکلی ندارد. هر سلول با 3 سلول دیگه مجاور هست.

دسته ها باید توانی از 2 باشند. در 4 متغیر با 4 تا سلول مجاور هست در 3 متغیر با 3 سلول و با 2 متغیر در 2 سلول مجاور هست. هر تک سلول یک مینترم هست در یکی 4 متغیر در یکی 3 متغیر. یک سلولی 4 متغیر بعد 2 سلولی میشود 3 متغیر. 4 سلولی 2 متغیر هست. در 3 متغیر 2 سلولی بیا تعداد 3 تا از 4 تا را حساب کن چون هر کدوم صفر یا 1 هستند ضربدر دو به توان متغیر ها کن خودش و نات آن.

دسته ها باید بزرگترین دسته ممکن باشند. میتوانی طبق اصل تکرار یک خانه تکراری در دو دسته باشد. مجاور ها توانی از 2 هست تعداد آنها. دقت کن بزرگترین دسته ممکن را باید انتخاب کنیم. نات یک عبارت sop یعنی صفر های جدول 1 و 1 های جدول را صفر بگذار بعد بین 1 ها sop بگیر.

اول به سلول های ایزوله تر نگاه کن. و دسته باید بزرگترین حالت ممکن باشد. ایجاب کننده را بر اساس مینترم ها گفتیم بر اساس ماکسترم ها صفر ها هم میشود گفت.

مینترم هایی که باعث میشوند pi بشوند EPI ضروری ساز هستند یعنی مینترم هایی که فقط یک دسته آنها را پوشش داده هست و اشتراکی با بقیه ندارد.

دقت کن که یک مینترم که در یک بسته یا PI باشد و در PI دیگری نباشد ضروری ساز میشود و کل اون دسته ضروری ساز میشود. تو ساده سازی ها باید اول ضروری ساز ها را انتخاب کنی اگر کامل پوشش داد که هیچ اگر نداد نگاه بهتری داشته باشی به طوری که همه را بتوانی پیدا کنی تا ساده سازی کنی چون طبیعی هست دیگه اگر یک PI انتخاب کنی میتونی اون مشترک ها را حذف کنی. توی همین بحث اگر دیدی جملات به صورت جمع بودند یعنی صفر ها را گرفته و اگر 1 ها را گرفته بود یعنی به شکل ضرب بودند.

دقت کن ما دنبال این هستیم که input ها را در حالت دو سطحی کاهش بدهیم نه همینطوری.

حالات بی اهمیت: مینترم ها یا ماکسترم هایی هستند که خروجی چه صفر باشد چه 1 برای ما اهمیتی ندارد یعنی از اون مقدار عدد همه آن را نیاز نداریم. پس یا مهم نیست یا هم ممنوعه هست و باعث خرابی میشود و در ساده سازی جوری میگیریم به نفع ساده سازی باشد پس صفر یا 1 گرفتنش به ما بستگی دارد. مثلا از ده تا پانزده در سیستم NBCD برای ما بی اهمیت هست. بی اهمیت ها را دسته نمیگیریم باید به نفع ساده سازی ما باشند. Pi را میتوانی نسبت به 1 و بی اهمیت هم بگی دسته ای که فقط بی اهمیت هم داشت pi هست. این تعاریف برای صفر هم میشود مثلا جا مینترم ماکسترم جا 1 ضروری ساز بشود صفر. دسته هر چی بزرگتر بشود یک متغیر ازش میپرد یعنی تک سلولی 5 متغیر 2 سلولی 4 متغیر و 3 سلولی 3 متغیر و 4 سلولی 2 متغیر و ... . دقت کن تو کارنو 5 تایی دو تا جدول نظیر به نظیر مجاور هستند نه قرینه یا تقارن. دقت کن a رو تو جفت جدول ها میگیری فقط بحث معمولی یا نات بودن آن هست.

در جدول mq اون هایی که تیک میخورند pi نیستند چون مشخص هست تو دو مجموعه هستند. تو مقایسه دوم خط تیره و اختلاف یک بیت مد نظر باشه توی جدول یک به دو فقط باید تو یک بیت اختلاف داشته باشند که همون رو خط تیره بگذاریم.

هدف ساده سازی: از بین pi ها کدوم را انتخاب کنیم که همه مینترم ها را پوشش بدهیم با کمترین تعداد pi ها و بزرگترین pi ها. در این جدول هر کدوم که تو یک ستون 1 ضربدر خوردند epi هستند. واسه جدول اول باید بی اهمیت ها رو هم بنویسی.

توی روش ساده تر دقت کن که دسته های بالاتر باید مینترم بالا تری باشند یعنی اگر 15 در دسته بالاتر بود و با 19 که 4 تا اختلاف دارن و توانی از 2 هست درست هست مشکلی ندارن ولی 19 با 15 نمیشود.

دقت کن اختلاف باید توانی از 2 باشند.

تو جدول 2 به 3 برو سراغ اونایی که اختلاف دو تای آنها یک توانی از 2 هست مثلا دوتاشون 16 تا اختلاف داشته باشند. اونایی که نوشتی رو تیک بزن که بفهمی pi نیستند. اونایی که تکراری هستند را ننویس ولی تیک بزن.

دقت کن ببین ما اول باید epi ها رو انتخاب کنیم بعد ببینیم همه مینترم ها پوشش داده میشوند یا خیر اگر پوشش داده نشده بودند باید برویم و از غیر epi ها هم انتخاب کنیم.

ستون های یک ضربدری pi آنها میشود epi.

برای ساده سازی جدول دوم qm برو ببین که در صورت مسئله مینترم هایی که اومده در کدام pi هست همان ها را به صورت or بنویس و بعد همه رو باهم and کن. مثلا ببین مینترم 2 در کدام pi ها ظاهر شده است و اون هارا بنویس. اون هایی هم که فقط یک در یک pi بودند به صورت تک لیترال بنویس. بعد با خاصیت جذب و تکراری بودن لیترال ها یک سری از جملات را میتوانی حذف کنی. بعد سعی کن باقی مانده عبارت با کمترین تعداد 1 همه رو یک کنی. حالات بی اهمیت هم تو جدول اول حساب کن. برای ساده کردن و آوردن به جدول دوم باید دو تا عدد با هم اختلاف توانی از 2 داشته باشند. دقت کن مینترم دسته پایین تر باید از مینترم دسته بالا تر بزرگتر باشد. بعد از جدول دوم به سوم اونهایی انتخاب کن عدد داخل پرانتز آنها یکی باشد و تفاضل مینترم ها یکسان و توانی از 2 باشد اختلاف هر کدوم از اونها باید باشد. دقت کن دسته اول با دسته دوم. دسته دوم با دسته سوم و دسته سوم با دسته چهارم. تکراری ها را هم فقط تیک بزن. بعد که خواستی pi بنویسی فقط به مینترم ها دقت کن نه بی اهمیت ها.

**نکات فصل دوم:**

تعداد کل دسته ها 3^n در جدول n متغیر هست.

تابع xor داشتی اونجا ها که یک بودن مینترم های اون تابع هستند.

روش کدینگ رو برای نوشتن یک تابع به صورت مینترم برو به این صورت که برو ببین داخل هر جمله کدوم لیترال نیست جای خالی اون را یک خط تیره بگذار و ببین با صفر و یک گذاشتن چه اعدادی بدست میاد خروجی دهدهی آنها همان مینترم ها هستند. دقت هم کن عدد ها باید طبق اصل خودشون نوشته شده باشند یعنی اگر دنبال مینترم یا یک سوپ بودی طبیعی هستی وقتی متغیری نات دارد یعنی اصل آن صفر بود که بعد نات شده است.

ضرب مینترم و ماکسترم هم شماره میشود صفر. جمع آنها میشود یک. ضرب یک مینترم با ماکسترم غیر هم ردیف میشود مینترم. جمع آنها میشود ماکسترم. تو جمع و ضرب چند تایی مینترم ها و ماکسترم ها دنبال اون هم شماره ها برو. جمع دو ماکسترم غیر هم ردیف 1 و ضرب دو مینترم غیر هم ردیف صفر میشود. ماکسترم از نات مینترم بدست میاد.

**فصل سوم:**

XOR و XNOR شرکت پذیر هستند مثل AND و OR. در توابعی که به زوج و فرد حساس هستند اگر زوج بار عمل NOT را انجام بدهی نتیجه عوض نمیشود. XNOR همان علامت NOT کردن هست و جز آن حساب میشود. دو ورودی مساوی میشود یکی.

دقت کن توی مثال فقط حق داری متغیر و عدد بدی حق نداری تو مرحله اول نات بدی چون میخوای اون رو بسازی بعد اون میتوانی از نات تو ورودی استفاده کنی.

با 2 تا NAND میتوان AND دو ورودی ساخت. دقت کن که اون وسط یک نات میگذاری بعد با یک NAND خنثی میکنی اون گردی رو. OR که ورودی آن نات هست برابر هست با AND که خروجی آن نات هست. دمورگان. AND که ورودی آن NOT هست NOR میشود. OR که ورودی آن NOT هست میشود NAND.

برای NAND-AND: F نات SOP بگیر و ساده کن بعد اند و اور بکش. بعد F نات رو به NAND NAND تبدیل کن بعد گردی آخر رو بردار.

**نکات فصل سوم:**

NAND دو متغیر یکسان نات خودش هست. NOR نات اند را میدهد. و NAND نات اور را میدهد. اند یا اور 3 تا دایره داشت اون یکی رو میده. 2 تا دایره داشت نات اون یکی رو میده. NAND سوپ میخواد و NOR پوس میخواد. با 4 تا Nand میتوان یک xor ساخت. یکی رو با خودش Nand کنی نات خودش را میدهد. یکی رو هم با خودش nor کنی not خود را میدهد. هر چیزی با 1 xor شود میشود نات خودش با 0 شود میشود خودش. با Nand خواستی and بسازی 2 تا بزار اول نات بدست میاد بعد همون رو به صورت دو ورودی بده تا نات خودش بدست بیاد که میشود همون and. Nand باید به and-or یا sop ساخته شود و nor-nor باید با pos ساخته شود و تبدیل به or-and. خواستی از طریق مینترم ها گیت سنتز کنی دقت کن که اگر or یا nor بود pos و اگر به صورت and یا Nand بود sop کن دو سطحی کن بعد گردی بزار تهش.

تابع and مینیمم گیر و or ماکسیمم گیر هست.

اینطوری فکر کن که دو تا ترکیب نباید یک گیت یکی باشند.

تو 2 سطحی اگر OR اول بود OR-AND کن و اگر AND اول برعکس. اگر NAND دوم بود باید دنبال نات پوس باشی و اگر نور دوم بود باید دنبال نات سوپ باشی.

نور اور: نات پوس بگیر نور nor کن به شکل دو سطحی اور اند. بعد گردی آخر رو خط بزن.

NAND رو توی مدارات منطقی شبیه به AND تصور کن. گیت ها باید وقتی میخوان درست باشن مخالف هم باشن.

دقت کن pos میشود or-and و sop میشود and-or. کلا or با pos کار میکند و and با sop. همچنین nor هم همین شکلی فکر کن.

منطق سه حالته: GND قسمت منفی هست اتصال به زمین صفر و اتصال به سر منبع تغذیه میشود 1. حالا تکلیفی سیمی که نه به زمین وصل هست نه به منبع تعذیه چی؟ سیم قطع میگوییم. یک متغیر دیگری هم داریم به نام بی اهمیت مثلا مورد تایید ما نیست یا ممنوع هست یا برای ما مهم نیست. اگه دو سیمی که یکیش مقدار صفر و دیگری مقدار یک میشود همین بی اهمیت که جرقه میزند و میترکد منبع تغذیه.

اگر هیچکدام از دو پایه کنترلی فعال نبودند بی اهمیت چرا چون مدار نمیداند قطع چیست. اگر بودند ولی مقادیری که میرفت مخالف هم بود بی اهمیت در جدول کارنو به مدار. برای این کار بهتر هست کارنو بکشی.

گیت های 3 حالته: همون گیت های معمولی هستند فقط یک پایه کنترلی یا اضافه دارند و اگر 1 باشد گیت کار عادی را انجام میدهد اما اگر اون پایه صفر باشد خروجی کلا قطع میشود. یا Z میشود. Active low یعنی صفر و یک جا به جا شده هست. برعکس کار میکند. گیت هایی با خروجی z را میتوانیم به هم وصل کنیم جز این باشد میسوزد. ممنون همان بی اهمیت هست در مثال دقت کن. هر دو کنترلی که ضابطه آنها ضد هم باشد میتوان تابع نوشت. دقت کن وقتی تابع مینویسی باید نات کنی. ببین گیت نمیداند مقدار قطع چیست پس ممنوع هست واسه همین توی اون تست دوم دو تا خونه ac دونت کر شدند. کارنو را با مقادیر خونه هاش درون متغیر ها بگذار. با یک مثلث سه حالته هم میگذارند.

کاربرد مقاومت در مدار منطقی: فقط دقت کن اگر اون نقطه که بین مقاومت زمین و سیم و مقاومت منبع تغذیه هست اگر از بقیه جا ها قطع باشد میتوانی مقدار زمین یا منبع تغذیه رو بندازی تو سیم یعنی صفر یا یک.

گیت open collector: خروجی آن یا صفر هست یا قطع. لوزی درون آن دارند. اگر 1 بود قطع هست کلا خروجی.

سوییچ ها 3 پایه دارند. طرز کار مثل کلید هست اگر c یک باشد اون دو تا به هم متصل میشوند. ورودی و خروجی فرقی ندارند. P که مثبت هست شکل ها برعکس هست یعنی گردی دارد و به ازای صفر متصل میشوند و n که منفی هست شکل برعکس هست و گردی ندارد و به ازای 1 متصل میشوند. میره تو p بره بیرون n. کلا خواستی ببینی مدار چه کار میکنه کارنو بکش به ازای متغیر جایگذاری کن ساده sop را بنویس.

با سوییچ میتوان همه چیز تولید کرد منطق کامل هست.

COMS: مخالف هم هستند دوگان هستند بالا PMOS پایین NMOS. و F یه به بالا وصل هست یا به پایین. برای تابع هم خروجی را SOP ساده کن. ترانزیستور نوع N به زمین معمولا وصل میشود چون صفر را به خوبی رد میکنند همین برعکس واسه P.

Transmission gate : هم صفر را رد میکند هم 1 را خوب رد میکند.

مخاطره: ما فقط با مخاطره potential کار میکنیم. در جدول کارنو دو تا یک مجاور که در یک دسته نیستند مخاطره داریم. جا به جایی بین اونها منظورم هست گذر از یکی به دیگری امکان بروز مخاطره هست. و این مخاطره دو طرف نیست یعنی یا از این به اون مخاطره میشه یا از اون به این یکی هر دو با هم نمیشود. اگر خواستی مخاطره پیش نیاد یه دسته جدید بگیر و یک گیت جدید اضافه کن. اگر یک تابعی شامل همه pi ها باشد امکان مخاطره ندارد. خروجی sop کارنو همان ساده شده تابع هست. همه بحث ها تا الان استاتیک سطح 1 هستند. دقت کن ساده شده تابع به صورت بدون مخاطره میشود اونهایی که ساده شده بودند بعلاوه اون جملاتی که اضافه کرده تا مخاطره حذف شود . حذف مخاطره هم که یادت هست یه طوری بکن تو دسته که 1 مجاور غیر هم دسته نباشد. مخاطره در sop در استاتیک سطح 1 اتفاق میفتد. و اگر تابعی شامل همه pi باشد مخاطره ندارد. عکس این جمله درست نیست یعنی تابعی میتواند همه pi ها را بر ندارد و مخاطره هم نداشته باشد. بی اهمیت ها را وقتی جز دسته میگیری 1 میشود. اگر تو مثال دادند همین شکلی دو نوع ساده سازی و گفتند کدوم بهتر هست اونکه gate input کمتری دارد مد نظر هست. مخاطره استاتیک سطح صفر دقیقا برعکس سطح 1 هست و به شکل or-and یا pos هست. و مخاطره این هست که مثلا خروجی باید صفر باشد ولی 2 نانو ثانیه 1 میشود.

مخاطره دینامیک: حالتی است که یک ورودی را عوض کرده ایم و خروجی چندین بار ناخواسته موقتی عوض شود. تاخیر نا مساوی به معنی وجود مخاطره نیست.

مخاطره functional تو تغییر چند تا متغیر هست ولی potential تو تغییر یک ورودی. تو دومی دلمان میخواد 1 باشد ولی گاهی اوقات صفر میشود و برعکس. دینامیک هم بیشتر از یکبار تغییر میکند.

**فصل چهارم:**

دو نوع مدار داریم ترکیبی و ترتیبی در مدار ترکیبی فاقد حافظه هست در ترتیبی ورودی های که قبلا دادیم در خروجی تاثیر دارد ولی در ترکیبی فقط به مقدار فعلی کار دارند. با n تا متغیر 2^2^n تابع یا فانکشن میتوانیم داشته باشیم.

طراحی یک مدار رای گیری اکثریت برای 3 ورودی یعنی چی یعنی (یعنی اینکه جدول درستی رو بگیر ببین مثلا از 3 تا کجا ها 2 تاش 1 هستن اونجا ها را بزار داخل جدول کارنو ساده شده را بدست بیار بعد مدار را طراحی کن) خروجی وقتی 1 هست که مثلا از 3 تا 2تای آن یک باشد. برای طراحی اول جدول درستی بکش نسبت به شرط مسئله بعد کارنو بکش و تابع را ساده کن. دقت کن مثلا برای 4 تا نیاز نیست با 4 متغیر بنویسی چون با قانون جذب از بین میرود همینکه 3 تا بگوید اکثریت هست. و ترکیب 3 از 4 هست برای 5 متغیر هم همین هست باید 3 تا 1 باشند ترکیب 3 از 5 هست. اگر یک کارنو 1 ها شطرنجی بودند تابع یا زوج هست یا فرد اگر خانه هایی فقط 1 بود که تعداد 1 ها زوج هستند پس تابع زوج است و XNOR منظورش هست. برای اینجور جداول کارنو بکش. روال کار کلا همینه جدول بکش. واسه دو بیتی برای حالت مساوی یا باید نظیر به نظیر مساوی باشند یا باید نظیر به نظیر یکی مساوی و دیگری بزرگتر باشد. خارج از بحث: دقت کن and رو دو تا نات بزاری پشت اون میشود nor همین برای or و Nand. برای نشان دادن مساوی میتوان از XNOR استفاده کرد چون زوج هست دیگه. واسه 4 بیتی هم همین هست یعنی مثلا 4 بیت یا همشون بزرگتر هست یا 3 تاشون و یکی مساوی یا 2 تاشون و 2 تا مساوی یا 3 تا مساوی و یکی بزرگتر هست. برای هشت بیتی هم دو تا 4 بیتی میگذارند اون 3 تا ورودی پایین مدار هم برای توسعه دادن هست. برای هشت بیت تو اولی 4 بیت با ارزش هر کدوم رو میدی و برای دومی 4 بیت کم ارزش خروجی نهایی باید از پر ارزش ها گرفته شود. برای بزرگتر یا کوچکتر بودن ولی برای حالت مساوی باید برویم سراغ کم ارزش ها و به اون 3 تا ورودی پایین می چسباند. و اگر بیشتر بود خروجی نهایی بیشتر و ... فقط به ورودی آبشاری 3 تا پایینی باید g l را صفر بدهیم ولی e را یک بدهیم که اگر تو 4 بیت کم ارزش هم مساوی بودند خروجی نهایی واقعا E=1 شود. مقایسه از بیت های پر ارزش شروع میشود. در مقایسه کی برابر هستند؟ بیت ها نظیر به نظیر مساوی باشد. کی الف از ب بزرگتر هست؟ اگر پر ارزش ترین رقم الف از پر ارزش ترین ب بزرگتر باشد این 2 مساوی باشند و رقمی بعدی بزرگتر باشد. با NOR کردن دو تا از GEL اون یکی که نداریم بدست میاد چون NOR 2 تا صفر را 1 میدهد. پایه های آبشاری برای این هست که مدار را گسترش بدهیم. برای مقایسه 8 بیت به 8 بیت 4 بیت پر ارزش اولی و 4 بیت پر ارزش دومی را به مقایسه گر اولی و اون بقیه به مقایسه گر دومی بده همچنین خروجی را از پر ارزش ترین بگیر. دقت کن وقتی مقایسه گر اولی برابر باشند به GEL نگاه میکند. E دومی هم بخاطر این 1 دادیم که وقتی اون 4 تا پایینی هم مساوی بودند بیا 1 بده اون پایه را که تا اولی همینطوری 1 برود.

روش برش بیت: به این شکل هست که اگه میخوای برای n بیت یک مداری بسازی اول بیا برای 1 بیت بساز بعد اون رو n بار تکرار کن. منتها باید جوری باشد که قابلیت آبشاری یا بسط را داشته باشد. دقت کن وقتی در مثال i را صفر میگیری یعنی کل جواب A را داری با کل B مقایسه میکنی پس جواب هم همین شکل هست. از XNOR برای تساوی دو متغیر استفاده بکن. واسه اولی باید E رو 1 بدی چرا چون قبل اون همش صفر هست پس منطقی هست نه بزرگتر هست نه کوچکتر و فقط E 1 میشود و بقیه صفر. برای مداری که ساختیم E را 1 بده که تایید کنیم از قبل این اعداد که 0 هستند با هم مساوی هستند و دو تای دیگر را 0 که خوب منطقی هست. دقت کن به and باید ورودی بی اثر بدهیم تا نابود نشود.

نیم جمع کننده: دو بیت میگیرد و این 2 تا را با هم جمع میزند و دو تا خروجی به نام حاصل و کری میدهد. دقت کن که هر ستون که سمت چپ میرویم ارزش بیت ها دو برابر میشود اگر مبنای ده بود ده برابر میشد. دقت کن به sum نگاه میکنیم. ارزش کری دو برابر sum هست. نیم جمع کننده کریش اند هست و خروجیش xor.

کری ارزش از حاصل دو برابر هست. آیا نیم جمع کننده کامل هست؟ یعنی میشود اند اور نات رو ساخت؟ بله چرا چون داخل آن اند رو دارد و XOR هم دارد. حاصل XOR ورودی و کری AND ورودی ها هست.

تمام جمع کننده: کار آن جمع 3 بیت هست. یادت باشد با 2 تا نیم جمع کننده و یک OR میشود یک تمام جمع کننده ساخت. جای OR میتوان XOR گذاشت. اگر دو نفر AND صفر داشتند OR=XOR هست. اگر مکمل کنیم حاصل همان قدیم هست فقط نات میشود. کری هم میشود همان کری قدیم فقط نات کن. این یک مفهوم دیگر سلف دوال هست. اونجایی که نات شده نات نیست و نات نشده نات هست میتونی نات ها رو اینطوری جا به جا کنی. تمام جمع کننده کریش majority و sum آن odd function هست. اگر دو نفر اند یکسانی داشتند xor با or آنها برابر هست. دقت کن اونجا که داریم تمام جمع کننده میسازیم ورودی اون or هیچ موقع جفت 11 نمیشود پس میشود xor استفاده کرد. اینکه اگر ورودی ها را نات بدهی سام و کری نات قبلی هستند مفهوم دوگان هست. و انگار مفهوم زوج و فرد بودن جا به جا میشود. در واقع تو در FA میتوانی هر جا خواستی نات ها را بزاری.

جمع دو عدد 4 بیتی: جمع نوع اول یک نیم جمع کننده میخواد و سه تمام جمع کننده در واقع تمام جمع کننده برای بسط دادن هست. به روش بیت اسلایس هست. پس ساخته شدن FA بخاطر همین برش بیتی هست. یک چیز را 0 xor کنی میشود خودش و با 1 کنی میشود نات خودش. قانون نات: از خروجی ها مثلا 2 تا برداشتی یا کلا برداشتی نات های ورودی هم کلا بردار اونجایی هم که نات نداشت نات بزار در FA جواب میدهد.

مقایسه کننده 4 بیتی: واسه مساوی خیلی ساده هست طبیعتا هر 4 تا خروجی باید 0 شوند تا بفهمیم مساوی بوده اند پس یک تابع NOR اضافه میکنیم و اگر 1 شد میفهمیم مساوی بودند. کری اوت رو نات میکنیم تا اگر 1 شد یعنی کوچکتر بوده که کری شده 0 و اون دو تا رو اگر NOR کنی سومی که G هست بدست میاد.

اگر 2 از gel را داشتی اون یکی را میتوانی با nor بسازی. دقت کن کری تمام جمع کننده majority 1 هست. برای تفریق بدون تبدیل به جمع دقت کن وقتی 1 بیت قرض میگیری یک پرچم به کناری میفرستی میگی که از بغلی کم کن چون یکی قرض گرفتم در واقع a-b-borrow میشود. اگر آخری قرض خواست و 1 بود یعنی دومی از اولی کوچک تر هست. nor فقط صفر را میفهمد. یادت باشد که اگر دو کری آخر با هم یکی نباشند یعنی سر ریز رخ داده است. در سیستم مکمل 2 همون xor همون قبلی که خوندیم راجب فلگ ها در سیستم بدون علامت هم که خوب اگر کری تغییر کرد میفهمیم کی بیشتر هست.

نیم تفریق کننده: یک بیت از اون یکی بیت کم میشود و وقتی بارو 1 شود یعنی بیت دومی بزرگتر هست و نیاز به بیت کمکی داریم.

تمام تفریق کننده: میاد اولی رو منهای دو تای دیگه میکند. بارو دقیقا عین کری هست منتها فقط x آن نات شده است. پس بارو میشود نات خط اول. دقت کن تمام تفریق کننده دومی اولی رو منهای دومی و منهای بارو میکند چرا چون دقیقا از اون قرضی که داده خودش را از آن کم میکند. تمام تفریق کننده در کامپیوتر استفاده نمیشود چون سخت افزار اضافی باید اد کنیم.

جمع کننده BCD: میخواهیم جوری بسازیم که خروجی همون جمع کننده یکان ما را بسازد یعنی مثلا 5 بشود 0101 و خروجی کری هم دهگان هست اگر 1 بود میشود 15. چرا واسه دهگان یک پایه کافی هست چون یا 0 هست یا 1. حاصل دودویی را با 6 جمع کنی حاصل دهدهی را میدهد.

در یک جمع کننده 4 بیتی برای اینکه بفهمیم خروجی بزرگتر از ده هست یا کوچک تر از ده، چون اگر بالای ده بود باید 6 تا اضافه کنیم. اگر خروجی چهارم و خروجی دوم 1 بشوند یعنی حاصل دست کم ده هست. 10 و 11 اینجوری هستند. ولی 12 خروجی چهارم و سوم یک هستند. 13 اینجوری هست 14 و 15 شرط اول هم دارند ولی از 16 تا 19 کری آنها 1 هست چون دقت کنی یکی کری اضافه آورده دیگه. اگه هر کدوم از این شروط بودند یعنی حاصل 10 تا 19 هست پس باید 6 تا بهش اضافه شود. حاصل اون or، 1 شد یعنی یکی از اون 3 شروط بر آورده شد. همین or هست که دهگان را نشان میدهد. دقت کن هر پایه یک توانی از 2 هست که حالات دارد پس 4 بیت باینری جمع کننده که 9 ورودی دارد 2^9 حالت دارد که البته خیلی هاش معتبر نیست. 200 حالت معتبر کلی هست چون 10 تا اولی و 10 تا دومی و کری ورودی هم 2 حالت.

به ازای 9 ورودی ما 2 به توان 9 تا حالات داریم ولی همشون معتبر نیستند کلا 200 تا حالات معتبر هست 10 ضربدر 10 ضربدر 2 به خاطر کری ورودی که یا صفر یا 1. بعد منهای کل حالات میکنی نا معتبر ها بدست میاد.

سام یک مقداری تاخیر بیشتری دارد. تاخیر موازی بود یکی بگیر. تولید دهگان با 6 تا تاخیر بدست میاد. دقت کن چون دنبال این هستیم که دهگان جدا از یکان باشد نیاز به BCD adder داریم. در این مدار تاخیر یکان از دهگان بیشتر هست و سومین یکان بیشترین تاخیر را دارد. تاخیر گیت ها یادت باشد.

BCD مبنای 10 هست که به شکل دودویی نشان میدهیم.

تو مکمل r-1 به اندازه r-1 از اون عدد کم کن مکمل بدست میاد. تاخیر خواستی بدست بیاری بدترین خروجی را برو اگر چند رقمی هم بود همه ارقام را از 9 کم کن مثلا.

دقت کن در جمع BCD یکان و دهگان جدا از هم تولید میشوند. خود مکمل بودن یک مزیت هست چون به سخت افزار کمتری نیازی هست و صرفا با 4 تا نات گذاشتن در ورودی ها این مشکل بر طرف میشود ولی برای مکمل 9 NBCD که خود مکمل هم نیست به سخت افزار بیشتری نیاز داریم. مد رو باید به Cin وصل کنیم و با خروجی اون بعلاوه 6 مکمل 9 XOR کنیم تا اگر 1 بود نات B بیاد و اگر صفر بود خود B بیاد چون همزمان میخواهیم جمع و تفریق انجام دهیم. تاخیر ریپل 8 برابر قبلی هست. اما میتوانیم بعضی جا ها را کاهش بدهیم.

CLA: میخواهیم اون کری هایی که منتظر بودیم و باعث تاخیر میشد را پیشبینی کنیم تا تاخیر کمتر شود، دقت کن FA را میتوانستیم با 2 تا HA و یک OR بسازیم یا به نوعی 2 تا XOR و 2 تا AND و 1 OR. بعد یک سری فرمول بدست میاوریم که کری ها را بدست بیاوریم و کری C2 دیگر لازم نباشد منتظر C1 باشد و همزمان بتوانیم محاسبه کنیم همینطوری برای C3 به C2. با رابطه شماره 2 کری ها را میسازیم و با روابط شماره 3 SUM ها را میسازیم که میشود XOR روابط 1 و 2. 4 لول گیت نیاز داریم بر خلاف قبلی که به 8 لول گیت نیاز داشتیم. رابطه 1: 1 لول گیت XOR رابطه 2: 2 لول گیت هستند چون SOP هستند یعنی AND-OR و گیت رابطه سوم که XOR هست. خود XOR بین 1 تا 2 گیت تاخیر دارد. و تاخیر در کل میشود 6T که به نسبت 8T خیلی بهتر هست. به طور پیشفرض XOR را 2T بگیر و در کل تاخیر CLA را 6 بگیر. برای 8 بیت هم لول های ما زیاد نمیشود فقط یک سری مدار اضافه میشود زیر بقیه و تاخیر زیاد نمیشود برخلاف ریپل چون 4 تا FA دیگه به صورت آبشاری باید اضافه کنیم و زیاد میشود. دقت کن ممکن است همین ایده جدید که گفتیم واقعا زیاد هست و محدودیت داریم و در واقعیت پیاده سازی نمیشود. به صورت آبشاری بستن با ریپل هم حتی اینقدر تاخیر ما را زیاد نمیکند نسبت به ریپل. دقت هم کن در شکل اولی C1 وارد قسمت اول نمیشود و مستقیم وارد قسمت دوم میشود و C4 وارد قسمت سوم نمیشود و مستقیم از دومی خارج میشود این موارد برای محاسبه مسیر بحرانی و تاخیر ضروری هست. تاخیر خواستی حساب کنی برو به خروجی ها ببین کدوم بدترین هست بعد برو ببین کدوم ورودی به اون خروجی میرسد. اون 4 تا فرمول ساخت CLA بود. روشی که پترسون گفته با 3T حساب میکند چون سر بسته اول جا 2 میشود 1. منتها این قسمت sum مشکل میخورد چون دیگر از فرمول xor که ساختیم نمیتوانیم استفاده کنیم و باید با AND-OR بسازیم و میشود تاخیر 5T.

Decoder: دقت کن در مثال اول وقتی میگوید تاخیر FA برابر 10 ثانیه هست یعنی تاخیر هم تولید SUM هم CARRY یعنی وقتی وارد کردیم 10 نانو ثانیه بعد خروجی میاد. دقت کن در لحظه 10 فقط همه به ورودی نگاه میکنند و کری همه صفر هست چون حتی اگر کری تولید شده باشد در اون لحظه تاثیری ندارد و از چپ شروع میکنیم به خروجی نوشتن در لحظه 20 همه همون خروجی را میدهند فقط همون FA دوم که کری آن عوض شده است خروجی برعکس میدهد. دقت کن قبل لحظه 40 خروجی درست نیست.

مدارات ماژولار ترکیبی: decoder: n تا ورودی و 2^n خروجی دارد. یعنی به ازای هر حالت ورودی یک خروجی فعال میشود یعنی در هر لحظه فقط یکی 1 هست یا فعال هست. با ارزش یعنی اینکه اول اون خط را میخواند. در رمز گشا انگار هر کدام مینترم هستند یا تولید کننده مینترم هست یعنی در هر بار فقط یکی 1 هست. یک ورودی فعال ساز دارد که وقتی 1 هست کار میکند وقتی 0 بدهیم همه خروجی ها صفر میشود و ورودی اهمیت ندارد. پس دقت کن یک e با کل توابع and میشود. رمز گشا 2 به 4 چهار از 4 تا گیت سه ورودی and و 2 تا گیت نات تشکیل شده است میتواند نات ها نباشند. رمز گشا 3 به 8 از 8 تا and 4 ورودی تشکیل شده است. 4 به 16 هم از 16 تا گیت 5 ورودی و 4 تا هم گیت نات. در active high یعنی وقتی 1 باشد فعال میشود و active low یعنی وقتی 0 باشد فعال هست در active low رمز گشا ماکسترم ها را میدهد فقط از گیت NAND اضافه میکنیم جا AND و E هم یک نات دارد. دقت کن در جدولی که این 2 به صورت XOR شده اند برای SOP به صورت نظیر به نظیر XOR گرفتیم بین 2 کارنو تا کارنو حاصل بدست آمده است. واسه مثال بعدی مینترم ها را بدست آورده است با عدد دهی و محاسبه اون یکی متغیر که نیست. andیی که ورودی آن نات شده nor هست. orیی که ورودی آن نات شده Nand هست. این ترکیب هیچ فرقی با مدارات 2 سطحی ندارد و ترکیب های درست هستند. هر چیزی خواستی با decoder بسازی نیازی به مینترم ماکسترم داری. از اونجایی که sum، FA تابع فرد هست پس فقط اعدادی را مینویسیم که در آن تعداد 1 ها فرد هست اون خونه ها میشود مینترم. کری majority بود یعنی جاهایی یک میشود که 2 یا 3 تا 1 داشتیم یعنی بیشترین 1 پس مینترم اون خانه ها و چون اونا ماکسترم هستند در مثال fa پس اگر Nand شوند میشوند جمع مینترم ها یا sum. پس با یک decoder و یک گیت یک تابع با 2 تا 2 تابع و ... . در مثال بعدی اگر ترتیب ABC درست بود مشکلی نداشتیم ولی چون ترکیب بهم ریخته ما اون خطوطی که هست را معادل آن را با ورودی فعلی مینویسیم بعد تبدیل میکنیم به ABC بعد مینترم مشخص شده توسط اون عدد را مینویسیم. در مثال بعدی باید با کمک 4 تا decoder 2 به 4 بیایم یک decoder 4 به 16 را درست کنیم پس 4 تا کنار هم قرار میدهیم فقط بحث ورودی ها هست. ورودی ها را به این شکل بده که ببین کدوم ها ثابت هستند بهشون دست نزن اونایی که تغییر میکنند را به عنوان ورودی بده و یک رمز گشا هم اول بگذاریم که اون 2 تا که تغییر نمیکنند هر موقع رخ دادند اون رمز گشا متناظر فعال شود. اصولا دو بیت سمت راست یک عدد میشوند باقیمانده به 4 مثلا اگر جای اون 2 تا ثابت با اون 2 تا متغیر عوض شد رمز گشا اول فقط مضرب 4 ها مثل 0 و 4 و 8 و 12 را میدهند همین برای بقیه مثلا رمز گشا بعدی 1 5 9 13 را میدهد. روش ماتریسی برای پیاده سازی رمز گشا: کمی سخت افزار را کاهش میدهد ولی تاخیر بیشتر میشود و سطوح بیشتر.

Encoder: یا رمز گذار بر عکس رمز گشا هست پس 2^n تا ورودی و n خروجی دارد دقیقا برعکس و از بین ورودی ها اگر 1 شد خروجی متناظر یک میشود و همان شماره تولید میشود. دقیقا کن در رمز گذار فقط یک ورودی میتواند 1 باشد و اگر همزمان چند تا با هم 1 شوند غیر مجاز یا ممنوعه هست. یعنی از 16 تا 4 تا مجاز و بقیه غیر مجاز یا don’t care هست. در واقع سایر حالات را بدهی ممکن هست خروجی بدهی که خروجی متناظر آن اصلا فعال نشده باشد. پس باید رمز گذاری بدهیم که همزمان چند تا ورودی 1 شوند و اگر چند تا خروجی تولید شد اونی که دوست داشتی را قرار بده یعنی به ورودی ها اولویت میدهیم همون 1 شد کل خروجی را 1 میکنیم. در جدول این مدار ما 15 حالت را پوشش میدهیم ( 8 تا اولی 4 تا دومی و 2 تا سومی) ولی وقتی همه ورودی 0 باشد چی؟ اون موقع هر چی بدهیم انگار یک ورودی دیگر مدار فعال هست پس از بیت valid استفاده میکنیم و اگر 1 بود به مدار نگاه میکنیم و اگر صفر بود اهمیتی ندارد برای ما. از خروجی zero هم میتوان استفاده کرد که نات valid هست. دقت کن enable هم داریم. دقت کن در فرمول نویسی not ها با شبه جذب حذف شده اند مثل x3 نات در فرمول A1. اولویت با معمولی همیشه بیت بالای خروجی آنها مثل هم هست از لحاظ فرمول.

مالتی پلکسر: 2^N ورودی ولی فقط یک خروجی دارد. N تا سلکت داریم که اون انتخاب میکند کدوم ورودی برود به خروجی چرا چون 2^n حالت دارد. دقت کن برای اینکه بفهمی به چند بیت نیاز داری برای نمایش یا چند بیت برای سلکت از ورودی ها لگاریتم بگیر. برای نوشتن ضابطه اگر جدولی بود که فقط 0 و 1 نبود کافی هست مینترم ورودی ها را بنویس ضرب کن در مقدار تابع که همان i هست در اینجا. دقت کن که مالتی پلکسر را میتوان هم با رمز گشا هم با بافر 3 حالته ساخت فقط دقت کن خروجی بافر به شکل or هست. دقت کن در مثال بعدی واسه جدول کارنو گفتیم که چون اولی که 0 هست or هست بیا مطابق اون ستون که توش cd 00 هستند مانند or عمل کن همین برای بقیه ستون ها هست نسبت به تابعی که در ورودی هست. دقت کن حالات اجباری sop که قرمز رنگ هستند هیچی برای اون 3 بیت سمت چپ پایین که باقی مانده اند میتوانی 3 حالت بگیری که در تست ها بر اساس هر کدام که گرفت اقدام کن. در تست بعدی کار نداشته باش abcd کجا هستند فقط اون جدولی که خودت بلدی را بکش و جایگذاری کن خوب حالا در کارنو برای ورودی اول فرمول به شکل b+ not(d) هست برو ببین در چه خانه هایی 00 هستند و ما نیاز به این ورودی داریم مقدار را قرار بده مشخص هست که 4 تا سمت چپ بالا هستند چون هم ac 00 هستند. راه حل تستی: اگر مینترم یا ماکسترم دادن تو بیا اون رو بده به ورودی ببین خروجی 1 میدهد یا نه اگر داد یعنی هست. مینترم وظیفه داشت خروجی را 1 کند. اونی را بردار که مثلا در 2 گزینه هستند در 2 گزینه نیستند. در مثال اول به ردیف اول همه s0 را بده چون فرق آنها در یک 0 و 1 هست. 127 تا ماکس 2 به 1 میخواهیم تا یک ماکس 128 به یک بسازیم 63 تا هم برای 64 تا میخواهیم یادت باشد. دقت کن در مثال بعدی مورد اول 8 تا 4 به 1 داریم که 8 تا خروجی میده پس بین اون 8 تا که نمیتوانیم خروجی بدهیم باید یکی خروجی بدهیم پس یک ماکس هشت به 1 هم برمیداریم تا یک خروجی بده. در مثال بعدی به اولی چون مینترم هایی هستند که تولید میکند اون تابع 1 دادیم. دقت کن در حالت ب و ج شاید بخواهیم یک گیت نات اضافه کنیم اگر نداشت. کلمه محض یعنی بی علامت. چرا در مثال بعدی بیت های پر ارزش را دادیم به سلکت؟ چون مالتی بر اساس سلکت تصمیم میگیرد و بزرگتر باشد خیلی چیز ها معلوم هست. در شرایط 01 چون یعنی ب از الف بزرگتر هست و خط 1 انتخاب میشود پس خط 1 را بده 0.

دی مالتی پلکسر: بر خلاف قبلی که بهش data selector گفته میشد اینجا توزیع کننده گفته میشود. دقیقا برعکس مالتی پلکسر هست از لحاظ ورودی و خروجی سلکت هم مثل مالتی پلکسر هست. و بر اساس سلکت ها ورودی در یکی از این خط ها میرود. این خیلی شبیه به decoder هست به شرطی که i رو یک بزاری حتی اگر 0 هم بزاری i را انگار رمز گشا فعال نیست در واقع سخت افزار این 2 یکی هست فقط به ورودی میگوید enable و به سلکت میگوید ورودی. تنها فرق این هست که اگر خواستی رمز گشا بسازی و e داشت باید با i اول and کنی تا فعال شود.

قطعات منطقی برنامه پذیر: PLD یا programmable logic device در واقع IC هست که به ما اجازه میدهد برنامه نویسی کنیم. ASIC ها از قبل مشخص هستند مثل این رمز گشا هست یا رمز گذار یا مالتی پلکسر. ولی PLD ها توابع داخل آنها معلوم نیست و میتوانیم تغییر بدهیم. چند نوع هم دارند. SPLD باهاش سروکار داریم که کلا 500 تا گیت دارد همه اینها به صورت مدار دو سطحی هستند که پیش فرض AND-OR دارند. ROM طبقه اول که AND هست ثابت ولی OR برنامه پذیر هست یعنی ورودی اور را میتوانی تغییر بدهی. یادت هم هست که قطع شدن واسه گیت معنایی ندارد. بر حسب نیاز بعضی خروجی را قطع میکنیم بعد به زمین وصل میکنیم که 0 برود داخل. با اون or ورودی ها را میخواهیم با نقاط برمیداریم بقیه را قطع میکنیم و 0 میشوند. ضرب m بیت در n بیت کلا میشود m+n بیت بیشتر هم نمیشود. اگر تستی اومد 512 رو بزن جا 1024 اگر گفتند یک Cin اضافی هم داریم. یعنی حتی بخواهیم اون مدار جمع کننده را گسترش بدهیم 9 تا ورودی کافی هست. دقت کن اگر 3 تا 5 تا ورودی داشت جدول درستی بکش ببین کدوم ها 0 و کدوم ها 1 هستند. برای اینکه بدانی مربع کننده چند بیت خروجی دارد بزرگترین عدد اون بیتی رو فکر کن توان 2 کن مشخص میشود. اونایی که همیشه صفر بودند را به زمین وصل میکنیم و نیاز به گیت ندارد اون یکی هم چون همیشه مثل c هست گیت نمیخواد فقط وصل میکنیم به سی. و 2 تا خروجی کم کردیم. تابع خواستی کارنو. چرا شد گزینه 2 ؟ f5 با یک 2 تایی راه میفتد ولی بقیه 3 تا میخوان.

PLA: هر وقت دیدی طبقه اول یا and برنامه ریزی شد باید sop ساخته کنی در ROM اند ها داخل رمز گشا بودند و نمیشد بهشون دست زد ولی الان میتوان توابع را ساده کرد. PLA از لحاظ سخت افزاری بهتر هست چون گیت های کمتری دارد. ROM فقط مینترم بدست میاوردی نقطه میزاشتی. در مثالش چون 3 تا ورودی داریم و 2 تا خروجی پس 8 در 2 ROM میخواهیم. در حالت عادی باید 8 تا اند و 2 تا اور باید مصرف کنیم ولی 4 تا داریم پس هزینه نصف ROM میشود. وقتی کارنو کشیدیم 6 تا اند میخواد ولی ما 4 تا داریم پس اند ها را بریم ساده کنیم تا یک ترکیبی باشد که بتوانیم 2 تا یکی کنیم. دقت کن ساده به صورت SOP. دقت کن بعد ساخت اونی رو که نات کردی دوباره نات کن تا اصل آن بدست بیاد. در خروجی ها اون فیوز که گذاشتیم میخواهیم مشخص کنیم سالم بیاد بیرون یا نات بیاد بیرون. خود توابع ساده نات ساده یک طوری هم گیت نسبت بده به چیزایی که داری. بخاطر اتصالات زیاد یک مدار دیگه ساختن.

PAL: مزیت اول اینه توابع را ساده کن مورد دوم این هست که چون اور ثابت هست از ورودی تا خروجی کلا یک فیوز وجود دارد و سرعت بیشتری دارد. دقت کن در مثالش فقط 3 تا جمله ضرب میتوانی داشته باشی پس باید کارنو بکشی و ساده کنی. خوبی که دارد این هست که بعضی خروجی ها میتوانند به عنوان ورودی عمل کنند یعنی اون جمله ضرب 4 تایی که در حالت عادی نمیتوانیم بسازیم را دو تا اند بزاریم اونا رو یکی کنیم بعد از خروجی بیاریم ورودی به عنوان W با 2 تا دیگه میشود 3 تا جمله ضربی. اون اند ی را که نمیخواهیم یک ضربدر میگذاریم تا 0 بدهند ولی فیوز سالم هست. به این خروجی به ورودی توسعه دهنده گفته میشود. مزیت این بود که یک ضربدر میخورد یعنی یک فیوز داریم. وقتی تعداد ضرب بیشتر بود توسعه دهنده .

**فصل پنجم:**

جلسه اول:

مدارات ترتیبی: مدارات ترکیبی خروجیش به ورودی هایی که همون لحظه میدهند وابسته هست ولی به مقادیر گذشته وابسته نبود در واقع حافظه نداشت. ولی در ترتیبی ها ما حافظه داریم و بر اساس اون خروجی میدهد. یعنی خروجی ما به حالت حافظه هم بستگی دارد و ورودی هایی که قبلا دادیم در این حافظه ذخیره شده اند و به نوعی دارند تاثیر میگذارند. اون حافظه یا لچ هست یا فلیپ فلاپ. هر کدوم از این 2 تا باشد میتوانند 1 بیت را ذخیره کنند. پس یا صفر هستند یا یک.

لچ SR: یا لچ پایه هم گفته میشود و بقیه لچ و فلیپ فلاپ از این ساخته میشود. شکل ساده ای دارد دو تا شکل NOR که خروجی هر کدومشون وصل هست به ورودی یکی دیگر. چرا حافظه هست؟ چون از خروجی به ورودی مسیر دارد . Q بیت حافظه هست و مقدار در آن ذخیره میشود. مقدار Q با فید بک عوض نمیشود. دقت کن مقدار S R وقتی صفر باشد هر دو مقدار Q تغییر پیدا نمیکند و هر چی اول دادیم بعد از چند لحظه همان میماند یعنی 2 صفر بود حفظ حالت. ولی در حالات دیگر حتی مثلا 1 هم باشد در هر صورت صفر میشود اگر R یک باشد یعنی ریست پس خروجی در هر حالتی صفر میشود همان Q یعنی و اگر S باشد یعنی SET و خروجی در هر حالتی 1 میشود دو تا یک هم غیر مجاز هست. دقت کن درست هست تاخیر دارد اما بعد از گذشت چند ثانیه هم بلاخره خروجی NOR همان چیزی که درست هست میشود. دقت کن چندین فید بک رخ میدهد تا مقدار stable شود. مقدار RESET: S صفر باشد و R 1 باشد چون مقدار Q میشود در هر صورت. و SET برعکس همین حالت. وقتی که SR هر دو 1 باشند ما وضعیت را دوست نداریم چون مقدار هر دو باهم برابر میشود در Q به این وضعیت غیر مجاز میگوییم. چون کیو با کیو بار برابر میشود. Reset یعنی صفر شدن یعنی R وقتی صفر هست اول حروف همون هم هست پس خروجی صفر میشود و S اول حرف SET هست و وقتی 1 باشد مدار 1 میشود. به لچ SR نباید 11 بدهیم دقت کن. شرط اند SR بخاطر این است که جفتشون 1 نباشند چون غیر مجاز هست. چطوری میبریم یک مقدار را در لچ؟ اول RESET OR SET کن بعد hold کن. احتمال اینکه SR وقتی جفت یک هستند بخواهند بشوند صفر همزمان رخ نمیدهد یعنی یکی زودتر هست یکی دیرتر چون زمان پیوسته هست پس احتمال صفر هست چون تو زمان پیوسته دقیقا برابر صفر نمیشوند پس احتمال صفر هست و رقابت شکل میگیرد بین این 2 که مثلا اول S صفر بشود بعد R صفر و یا برعکس و 50 درصد بین این 2 حالت هست. (خارج از بحث دقت کن وقتی R 1 شود یعنی فعال یعنی RESET کن. ). در نتیجه وقتی از 11 بخواهد بشود 00 معلوم نیست در نهایت Q چه مقداری میگیرد و قطعی نیست واسه همین هست که میگوییم 11 مجاز نیست. لچ SR را با 2 NAND هم میتوان ساخت ولی نتیجه برعکس 2 تا NOR هست و جای کیو با کیو بار هم عوض میشود. در واقع در NAND، active low میشود یعنی 00 غیر مجاز هست.

C را آوردیم تا عملکرد این لچ را کنترل کنیم اگر 1 بود به همین شکل لچ عادی ولی اگر صفر بود به SR توجه نکنیم و Q را صرفا حفظ کنیم. انگار دو تا AND گذاشتی وسط خوب وقتی صفر بدهیم C را صفر میدهد انگار 00 میشود SR که حفظ حالت میکند به این لچ SR گیت شده گویند. همین را هم میتوان هر 4 تا را به NAND تبدیل کرد بدون اینکه active low شود ها دقت کن. جمله طلایی: ORیی که ورودی هاش نات هست NAND هست. فقط جای کیو با کیو بار عوض میشود. کلاک سیگنالی هست که دائما 0 و 1 میشود و برای زمان بندی و کنترل زمان استفاده میشود و کلاک باعث میشود که اجزا همگام شوند. و اون کلاک را به C لچ وصل میکنیم و به این میگویند فلیپ فلاپ SR. یعنی وقت هایی که 1 هست کار خودش را میکند. موقعی که صفر باشد HOLD میکند. 4 نوع فلیپ فلاپ داریم. فلیپ فلاپ D از روی SR هست. فلیپ فلاپ دی یک خط جدید هست به نام دی که خودش به S وصل میشود ناتش به R. و خروجی کیو با D برابر هست. چون میگویند دیتا فلیپ فلاپ چون دیتا را میاورد داخل. لبه مثبت: وقتی از صفر میخواهیم یک شویم. لبه منفی: از یک در کلاک میخواهیم صفر شویم. فلیپ فلاپ هایی معرفی شده فعلا حساس به سطح مثبت هستند ولی در مدارات واقعی از فلیپ فلاپ های حساس به لبه استفاده میشود. یعنی فقط همون لحظه به مقدار نگاه میکند و بقیه hold میکند چون ما دوست داریم فلیپ فلاپ فقط یک لحظه به ورودی خودش نگاه کند و سایر مواقع حفظ حالت کند چون سطح مثبت زمان طولانی برای فعال بودن هست برای نگاه کردن به آن و همگام سازی. یک پالس ساعت: از یک لبه مثبت تا لبه مثبت بعدی یک پالس ساعت گفته میشود. حساس به لبه مثبت یک مثلث در ورودی c قرار میدهند. دقت کن وقتی به سطح مثبت میخواهیم نگاه کنیم تا وقتی که دوباره سطح مثبت نشده است مقدار فعلی را hold میکنیم. انگار که توی اون لحظات دیگر کلاک بالا نیست که بخواهیم دوباره نگاه کنیم و دقت هم بکن که اگر کلاک سطح بالا بود ولی SR صفر بودند همان وضعیت hold هست و مقدار را باید نگهداری. ولی اگر حساس به لبه باشد فقط جاهایی که لبه مثبت هست به ورودی نگاه میکنیم یا همون کلاک و در سایر مواقع حفظ حالت میکنیم. بعضی کتابها کلا به اونا که حساس به لبه هستند میگویند فلیپ فلاپ و حساس به سطح را میگویند لچ.

فلیپ فلاپ JK: اصلا حساس به سطحش وجود ندارد و بی معنی هست. کیو را اند کن با k چسبان به r و کیو بار را اند کن با j و بچسبان به s. SR وقتی یکی شوند غیر مجاز هست. و در این فلیپ فلاپ اون ورودی 11 که تو SR غیر مجاز بود اینجا مکمل میشود نات میشود. حالا چرا حساس به سطح خوب نیست چون فقط یکبار به همون لبه نگاه کنیم و یکبار عوض کنیم. این فلیپ فلاپ با SR ساخته میشود فقط دو تا اند بزار اولش یکی برا J یکی برا K.K همان R هست پس اگر 1 شود ریست. RK و JS. حالت 11 مکمل Q هست.

فلیپ فلاپ T: یا تریگر یا عوض شدن همون JK هست فقط یک ورودی T میدهیم که به JK وصل میکنیم و دو حالت برای کیو هست T صفر باشد hold میکنیم 1 باشد نات میکنیم. دقت کن فرق D با T این هست که D اگر صفر بیاد خروجی صفر و اگر 1 بیاد خروجی 1 هست و به کیو کاری ندارد ولی T صفر بیاد همون Q میاد یا حفظ حالت و 1 بیاد کیو بار میاد یا ریست یا نات آن مشخص هم هست چون T XOR Q هست.

جلسه دوم:

در فلیپ فلاپ D دقت کن که با Q رابطه مستقیم دارد صفر بدهیم صفر یک بدهیم یک سایر مواقع حفظ حالت یعنی وقتیکه سیگنال کنترلی فعال نیست. ببین نگاه کن وقتی سیگنال کنترلی فعال هست تو نگاه کن به سطح نگاهت به سطح باشد بالا شد بالا کن. دقت کن تاخیر ها اول همه ناچیز هست. نگاه کن لبه یک ثانیه فقط نگاه میکند سطح تمامش را نگاه میکند به ورودی. T صفر بدهی حفظ حالت 1 بدهی Q عوض میشود یعنی نات کن. همه فلیپ فلاپ ها به جز D حفظ حالت دارند. همه فلیپ فلاپ ها وقتی فعال نیستند حفظ حالت میکنند. در XOR میشود دو طرف راستی را چپ و راست کنی. دقت کن وقتی میخواهی یک فلیپ فلاپ بسازی پس ورودی آن را قرار بده. یادت باشه همونطوری که با SR اومدی D ساختی یعنی یک ورودی خودش یک ورودی نات همونطوری هم برای JK میتوانی بسازی. دقت کن سطح بازه طولانی هست و دقت کن جز SR بقیه FF ها را میشه از روی هم ساخت. چون SR حالت غیر مجاز دارد. FF ها خوب است حساس به لبه باشند علت این هست که وقتی حساس به لبه هستند با هم همه یک کاری انجام میدهند و سنکرون هستند. و چون سطح بازه طولانی هست اصلا فلیپ فلاپ T و JK بی معنی هست. دقت کن فقط وقتی به ورودی نگاه میکند که تو لبه باشد و گرنه جای دیگری کاری ندارد. دقت کن در هر مسیری گردی بگذاری میشود active low ولی اگر نزاری میشود active high.

نکاتی راجب ساخت FF ها از روی هم: اولا همه FF وقتی فعال نیستند حفظ حالت میکنند و وقتی فعال هستند جز D حفظ حالت دارند. برای ساخت FF ها از روی همدیگر فقط کافی است معادله آن FF را بنویسی مثلا تی را میتوانیم با یک XOR دی بسازی یا همین شکلی دی بسازی که T را با Q XOR کنی. ساخت JK-FF با D کافی هست که همان فرمول JK را اجرا کنی روی ورودی D برعکس هم عین SR برخورد کن یعنی دی بزار به J خودش و به K نات را بده. با JK میشود تی ساخت به این صورت که ورودی های این 2 را همون T معمولی بده برعکس هم با دو اند و یک اور میشود که نیاز هست فرمول کمی تغییر کند. با SR میشود تی ساخت به دو روش یا اینکه XOR بزاری بعد روی خروجی D بگذاری و با دی بسازی یا اینکه با JK که دو اند به ورودی ها زده اند بسازی و T را به ورودی این 2 وصل کنی.

از CLR و PRE برای مقدار دهی اول استفاده میشود حساس به چیزی نیستند و هر موقع میتوانند اجرا شود اولی هر وقت اجرا شد Q صفر و دومی Q را 1 میکند. مثلا وقتی میخواهی همه فلیپ فلاپ ها را موقع شروع صفر کنی دقت کن این 2 وقتی فعال باشند اصلا به ورودی نگاه نمیکنیم و بعد استفاده باید خاموش کنیم دقت کن همزمان هر دو نمیتوانند فعال باشند مگر اولویت داشته باشند. Active low یعنی هر غلطی میکنند نات میشود پس حالت نات رو فرض کن. دقت کن فرض این هست که این ورودی PRE,CLR با کلاک سنکرون نیستند ممکن است سوال بیاد که سنکرون هستند. دقت کن به ازای تمام وقتی که این دو تا فعال هستند خروجی باید همین شکلی باشد و وقتی خاموش شدند تا لبه کلاک بعدی حفظ حالت میکنیم.

**فصل ششم:**

دقت کن اول تابع ورودی ها را بنویس در قسمت تحلیل و بعد بیا تابع حالت بعدی را بنویس حالت بعدی میشود همین معادله اون FF منتها شخصی سازی شده با نام خودمان. ببین در ساده سازی هم علامت رو ضربدر 1 فکر کن و غیر هم علامت را نابود کننده مثلا الف با الف خالی میشود همان الف ولی الف با نات الف کلا صفر میشود. اینها مراحل تحلیل هستند دقت کن. تابع حالت بعدی بر حسب حالت فعلی میشود مدار یا تحلیل. دقت کن ما با N فلیپ فلاپ 2^N حالت داریم. بعد مقادیر جدول حالت را در اون تابعی که بر حسب حالت بعدی بدست آوردی بگذار. یک روش دیگر هم این هست که مثلا مقادیر جدول حالات را بگذاری در FF ها و بعد که مقادیر ورودی مشخص شد سیگنال کنترلی را فعال کنید. طرز کار اون FF هم باید بلد باشی. حالات مختلف را تحلیل کن. اگر مثلا گفت بعد از 50 کلاک چه طوری میشود در اون ماشین حالت بچرخ 50 تا الگو بدست بیاری کافیه. فرکانس برعکس پریود هست. از یک لبه مثبت تا لبه مثبت بعدی را یک پالس ساعت گویند. مدت زمان طول کشیدن پالس را پریود کلاک گویند. فرکانس یا نرخ کلاک برعکس پریود کلاک هست و برحسب هرتز هست. فرکانس میشود تعداد پالس در ثانیه. یعنی 300 هرتز در ثانیه یعنی 300 پالس در یک ثانیه.

واسه فرکانس سایکل اصلی اون ماشین را بکش به جدا ها کاری ندارد. پس برای بدست آوردن فرکانس اون FF ها از ماشین حالت استفاده میکنیم فرمولی هم که گفته شد مشخص هست. دقت کن شمارش صفر به 1 در اون فرمول یکی هست یکبار بشمر. شمارنده ای که 3 تا عدد را هعی بشمرد میگویند شمارنده MOD 3 ببین فقط به چرخه یا سایکل نگاه کن نه کل حالات اون ماشین. نکات اضافی: دقت کن حالات خروجی های جدول هستند پس فقط به A میشود به بیت اول کار داشته باش برای اون فرمول مورد نظر تعداد حالات سیکل هم فقط بشمر دیاگرام اول 4 حالت دارد ولی 3 تاش داخل سیکل هست دقت کن وقتی گفتند فرکانس A چه قدر هست یعنی نسبت به فرکانس خود اون فرکانس کلاک چه قدر هست. Self start: از چه حالتی شروع کند مهم نیست بازم همون سایکل را فقط میچرخد حتی اگر PRE زده باشیم و جفت 11 باشند. دقت کن حتما چرخه وجود دارد در هر صورتی چگونه میشود خود آغاز نباشد؟ وقتی که بیشتر از 2 سایکل داشته باشد. دقت کن ماشین را میتوانیم بدون جدول هم بسازیم فقط داخل اون فرمول حالت بعدی بگذار. دقت کن وقتی میشماریم میایم نقطه شروع مثلا تو مثال دوم که مد 6 هست بعد از 48 تا میایم خونه 000 به این توجه کن. دقت کن وقتی 3 تا فلیپ فلاپ داریم یعنی 8 حالت داریم پس ماشین آن باید 8 حالتی باشد اگر 6 حالت کشیدی فراموش نکنی بقیه رو هم بکشی. چرخه معیوب: سایکلی که مد نظر ما نیست. برای همون 50 تا سایکل رو ببین، ببین mod چند هست. به ترتیب MSB دقت کن به ترتیب اون هست. دقت کن در مدارات خروجی هم داریم که باید در نظر گرفته بشود. چیز هایی که از فلیپ فلاپ بیرون بیاد میگوییم بیت حالت. خارج از بحث دقت کن در مدار ورودی فلیپ فلاپ را بگیر خروجی را تحویل بده. تحلیل مدار هم خیلی ساده هست یا فرمول اون فلیپ فلاپ ها را بنویس یا از روی گیت ها تحلیل کن. دقت کن اگر خروجی و ورودی یا حالت بعدی به ورودی وابسته بود باید بیت اون رو هم لحاظ کنی. دقت کن تو حالت دیاگرام این نوع ماشین باید لحاظ کنی ورودی را و وابستگی خروجی به ورودی را. دقت کن نسبت به ورودی های مختلف باید self start را بررسی کنی. مدارات ترتیبی سنکرون دو نوع هستند میلی و مور. میلی خروجی نسبت به ورودی وابسته هست یعنی در فرمول خروجی، ورودی هم ظاهر شده هست. مور یا اصلا ورودی خروجی ندارد یا به خروجی وابستگی ندارد یعنی ما اصلا X در فرمول خروجی نداریم و دو تا مقدار 0,1 در جدول ظاهر نمیشود برای راحتی هم در فرمول ها یکبار X را 0 فرض کردیم و به ازای X=0 ها در فرمول های مخصوص به آن و به ازای X=1 ها در فرمول های مخصوص به آن قرار میدهیم. دقت کن ورودی تاثیر میگذارد ولی مستقیم نیست پس مور هست. دقت کن اول وضعیت ورودی های FF را برای هر ورودی D یا JK یا SR یا T مشخص کن بعد فرمول حالت بعدی را بنویس جایگذاری کن این روال کلی برای تحلیل هست. دقت کن میتواند خروجی به ورودی وابسته نباشد ولی حالات بعدی اگر ورودی ظاهر شد باید لحاظ کنیم به ازای صفر و یک. خروجی در مور ساده هست چون کلا معادله کلی خروجی را باید با عدد جایگذاری کنی. اگه از رو شکل رفتی باید مقادیر ورودی را بدست بیاوری اول. مور را خروجی را به جای اینکه زیر ورودی بنویسیم زیر حالت ها مینویسیم چون به حالت ها وابسته هست. در مدار مور خروجی با کلاک سنکرون هست چون فقط به حالت نگاه میکند ولی در میلی خروجی با کلاک سنکرون نیست زیرا به ورودی وابسته هست و ورودی در هر لحظه ممکن هست تغییر کند. در میلی در حرکت ماشین و کلاک آن تغییر حالت با کلاک هست ولی کلا با ورودی تغییر میکند یعنی کاملا نگاه به ورودی همگام به ورودی ولی تغییر حالت موقع کلاک هست. میلی از مور تعداد حالات کمتری دارد خوبی مور این هست با کلاک سنکرون هست و هر میلی را به مور میتوان تبدیل کرد اگر خروجی یکسان بود که حله یک حالت میشود ولی اگر دو خروجی داشت باید به ازای یک حالت میلی دو حالت مور بزاری.

جلسه دوم:

در مدار مور خروجی فقط به حالت وابسته هست و حالت فقط با لبه کلاک ممکن است عوض میشود پس خروجی با کلاک سنکرون هست چون خروجی هم با لبه ها تغییر میکند. ولی در مدار میلی خروجی به ورودی وابسته هست و ورودی هر زمان ممکن است عوض شود پس خروجی هر زمان ممکن است عوض شود یعنی خروجی با کلاک سنکرون نیست این مشکل میلی هست منتها حالات میلی کمتر است و فلیپ فلاپ کمتری میخواهد. دقت کن برای شکل خروجی موجی هم باید ماشین حالت بکشی. برای کشیدن موج وقتی لبه فعال شد در مور به ورودی نگاه کن که به کدوم حالت میرویم به هر حالتی رفتیم برو ببین خروجی در آن حالت چند هست همون رو بکش. منتها فرقی که در حالت میلی داریم این هست که دقیقا همین شکلی هست منتها به فلش هایی که از حالت میزند بیرون نگاه کن و ببین به ازای چه ورودی خروجی چند میشود در واقع ورودی با خروجی برابر هست در اون حالت و فرق بعدی این هست که مور فقط تو لبه نگاه میکرد به ورودی ولی این توی اون حالتی که هستیم کلا نگاهمون به ورودی هست عوض شود عوض میکنیم.

مور باشه فقط با لبه کلاک عوض میشود. هر مدار میلی را میتوان به مور تبدیل کرد. خروجی یکی بود ببر زیر همون حالت بزار اگر خروجی های حالت یکی نبود اون حالت رو دو تا کن زیرش خروجی همون رو بزار پس تبدیل میلی به مور نهایت تبدیل به حالت ورودی میشود.

مدارات معروف سنکرون: رجیستر و شمارنده. سنکرون یعنی FF ها کلاک مشترک دارند. ببین دقت کن در تحلیل مدار اگر گفتند سیکل چند هست برو همون سایکل ببین چه اعدادی پوشش میدهد مثلا 6 تا 6 تا. طراحی برعکس تحلیل هست طراحی یعنی از توصیف برسیم به یک طراحی. هر بیت یک FF نیاز دارد. چرا به فلیپ فلاپ اول 1 دادیم ؟ چون وقتی T یک باشد هعی Q به ازای هر کلاک عوض میشود اولی به ازای هر کلاک تغییر میکند. واسه دومی وقتی میخواهیم عوض شود که اولی یک باشد و تغییر کرده باشد. دقت کن اگر اولی صفر باشد دومی حفظ حالت هست. صعودی یعنی همه بیت های قبل تر یک باشند. برای سومی هم وقتی که هر دو تا یک باشند همین هم برای چهارمین FF. واسه 20 بیتی هم همین شکلی هست یا 5 بیتی. نزولی دقیقا همین شکلی هست منتها از Q بار بگیر به جای کیو. معنی سوالاتی که میگویند صعودی/نزولی یعنی یک سیگنال کنترلی برای دو حالتی کردن بکنیم مثلا صفر صعودی باشد و وقتی 1 شد نزولی باشد. نکته یک چیز با صفر XOR شود خودش میشود با 1 شود ناتش میشود. دقت کن در 2 حالتی ها فقط کافی هست که به ازای هر ورودی روی تمام خطوط XOR داشته باشیم. امکان حفظ حالت یعنی کلاک زدیم تغییری نکند. یعنی تو قبلی ها وقتی کلاک زده میشد شمارش انجام میشد. اند گذاشتیم که اگر صفر بود و خواستیم حفظ حالت کنیم ورودی صفر شود تا حفظ حالت شود. یک ورودی کنترلی دو حالت بیشتر نمیدهد پس برای 3 کار دو حالت نیاز داریم. واسه اولویت دادن اونکه میخوای بالاتر باشد رو با اولویت پایینی اند کن منتها نات رو بده به عنوان ورودی که اگر 1 بود اون پایینی بسوزد.

نکات جلسه دوم: دقت کن که اگر میگویند شمارنده 4 بیتی یعنی به 4 FF نیاز داریم زیرا هر کدام فقط 1 بیت را ذخیره میکنند همچنین دقت کن در قبلی ها ما مدار داشتیم و نیاز داشتیم فرمول حالت بعدی و جداول و ماشین حالت را بکشیم ولی اینجا به ما فرمول ها را میدهند و میگویند مدار را بکش پس میایم فرمول های ورودی ها را بدست میاوریم دقت کن این فرق دارد با اینکه ما بیایم و برای خروجی ها معادله حالت بکشیم و حالت بعدی را حساب کنیم زیر آن موقع به معادله یک FF نیاز داشتیم.

لود کردن یعنی وارد کردن یک چیزی در FF چه زمانی؟ وقتی کلاک میاد. یادت باشد اولویت clear از همه بالاتر هست و به کلاک اصلا نگاه نمیکند. Count 1 باشد شمارش صعودی رخ میدهد در اون مدار بزرگ چرا چون اولیش که 1 هست بعد با بقیه اند میکند. دقت کن اولویت اول CLR PRE هستند که خروجی بلافاصله بدون توجه به کلاک دارند بعد لود بعد up down هستند.

در اون سوال بعدی به این علت اند گذاشتیم ورودی لود که وقتی که به 9 رسیدیم لود کنیم چون اون 2 پایه 1 میشوند در نتیجه خروجی 1 میشود و مدار فعال میشود. برگردیم دوباره سراغ 5 و توی اون نقطه اولویت لود از up که شماره صعودی هست بالاتر میرود. 5 برای لود از کجا اومد؟ تو همون پایه های I گذاشتیم دیگه. بعد از 5 میشمارد تا 9 به صورت مرتب. دور اول دقت کن از صفر شروع میکند تا 9 پس دور اول مهم نیست حالت پایدار مهم هست. BCD گفتند نوع را نگفتند NBCD هست. شمارنده BCD که از صفر تا 9 هست عین بالایی هست منتها اگر میخواهی از CLR استفاده کنی که از اول بشمرد باید دقت کنی که 10 را بدهی به پایه کلیر چونکه تا 9 بشود همه صفر میشوند و انگار 9 را نمایش نمیدهد. چرا چون اولویت CLR از همه بالا تر هست و بدون نیاز به کلاک و بلافاصله عوض میشود.

جلسه سوم:

شمارنده حلقوی: شمارنده های قبلی همه با T و JK ساخته میشدند. اما این یکی با D هست. حلقوی به این شکل هست که خروجی هر FF میدهد به راستی و آخری را میدهی به اولی. اگر همه صفر باشند هیچ چیزی را نمیشمارد در هر کلاک انگار همه CLR هستند و هر ضربه کلاک صفر را میدهند به بقلی و اتفاقی نمیفتد نکته این هست که نباید هر 4 تا صفر باشند و یک سیگنال داریم که به پایه ها وصل میکنیم و به PRE اولی وصل میکنیم و به ورودی CLR بقیه FF ها وصل میکنیم و شروع ما از 1000 هست نه از 0000. خروجی هر کدام میدهد به بقلی. خروجی ها در هر کلاک هست دقت کن. هزینه خیلی زیاد دارد چون با 4 تا FF کلا 4 حالت میشمارد و 4 عدد میشمارد در صورتی که میشود 16 تا شمرد. خوبی آن هست که در هر کلاک فقط یکی از آنها یک هست. که به اینها سیگنال زمانی گفته میشود برای زمان بندی عملیات ترتیبی. با این سیگنال ها به پردازنده میفهمونیم که الان نوبت کدوم عمل هست. داخل شمارنده 4 بیتی UP 4 تا فلیپ فلاپ T هست دقت کن. این حلقوی عین decoder هست که در هر کلاک فقط یکی یک هست در هر دوره شمارش.

شمارنده جانسون: مثل حلقوی هست از D-FF درست شده هست و سنکرون هست. فقط فرق این هست که آخری نات کیو را میدهد به اولی. شروع کار همه صفر هستند و حالت شروع 0000 هست و حالت اولیه آن با CLR است که همه رو صفر میکند. و کلاک بزنیم آخری نات میشود میاد تو اولی. با این هم میتوان سیگنال زمانی درست کرد. هر کدوم از این حالت ها این هست که هر کدوم از این شماره ها در 2 بیت با بقیه فرق دارد یعنی همه آنها. یعنی مثلا اولی بیت اول و آخر آن صفر هست و مثل هیچکس نیست. با این ها میتوان سیگنال درست کرد که با یک تابع بولی بگوییم در این حالت 1 شود و بقیه صفر شود و اون تابع که سیگنال هست فقط واسه اون حالت هست بخاطر همین خاصیت. جانسون در واقع همین 4 تا D هست بعلاوه 8 تا گیت اند 2 ورودی، سیگنال زمانی میتواند بدهد با گیت کم، دقت کن هر سطر یک فرمول مخصوص دارد که با بقیه فرق دارد. 8 تا سیگنال زمانی درست میکنیم دقت کن ورودی ما هم خود متغیر را دارد هم نات آنرا. همین کار را حلقوی با 8 تا فلیپ فلاپ 8 تا سیگنال زمانی درست میکند کلا همه اینها وظیفه تولید سیگنال را دارند. این روش از حلقوی بهتر هست. آیا جانسون self start است؟ یعنی حالت ها تشکیل یک سایکل را میدهند یا نه. چون 4 تا فلیپ فلاپ داریم 16 تا حالت دارد 8 تا بررسی کردیم 8 ماند اگر با اون 8 تا به داخل این 8 تا رفتیم یعنی سایکل رخ داده است. اون 8 تا که باقی مانده را بررسی کن ببین کدوم ها تو 8 تا اول نیستن. در جانسون دقت کن همه میروند سمت راست آخری نات آن میاد اول. جانسون self start نیست و 2 سیکل دارد چون اون 8 تا دوم خودشون یک سیکل دارند و بیرون نمی آیند. با اون تغییر اون بالا که دادیم در فرمول جانسون، self start میشود. دقت کن باید اون 8 تا اولی که مورد قبول ما هست را بشمرد تحت هر شرایطی. رجیستر: ذخیره کننده هست یا ثبات که ثبت کننده هست. سریعترین حافظه است که داخل پردازنده است معمولا با فلیپ فلاپ دی ساخته شده است واسه شمارنده از t بود اندازش معمولا از 16 اینا هست. رجیستر باید شیفت راست به چپ داشته باشد، حفظ حالت و لود موازی، اگر رجیستر شیفت به راست داشته باشد میشود شیفت رجیستر. دقت کن شیفت رجیستر به این شکل هست که همه رو هل میدهیم به راست آخری را میندازیم بیرون از اونور هم اون ورودی که وارد کردیم وارد میشود. Rotate هم داریم که انگار اونکه هل میدهیم بیرون از اونور وارد میشود. شیفت سمت راست دو نوع هست یکی شیفت arithmetic هست که همیشه بیت علامت سمت چپ را به عنوان ورودی میدهد یکی هم شیفت راست لاجیکال هست که در هر صورت صفر را به عنوان ورودی میدهد دقت کن واسه چپ فرقی ندارد rotate چپ هم مشخص هست که میاد اون بیت که از سمت چپ پرت میشود بیرون از راست وارد میشود. شیفت راست: 4 تا فلیپ فلاپ دی با هر کلاک خروجی این یکی را بدهیم به بقلی سمت راستی و شیفت میکنیم در هر کلاک. شیفت چپ هم دقیقا برعکس همین روش هست. امکان حفظ حالت: با 4 تا فلیپ فلاپ دی هر کی خروجی را میدهد به خودش. امکان load: ورودی خارجی را وارد فلیپ فلاپ کنیم و به این صورت هست که از ورودی میدهیم به هر فلیپ فلاپ با یک سیم از اونور.

چون 4 تا کار میخواهیم انجام بدهیم پس به 4 تا حالت کنترل نیاز داریم. جمع کننده سریال: قبلی ها ورودی را به عنوان موازی میدادیم. ولی در این جمع کننده ورودی ها را بیت به بیت میدهیم و در هر کلاک یک بیت را جمع میزنیم یعنی کلا یک عدد FA داریم ولی زمان بیشتر میشود از 2 تا رجیستر استفاده میکنیم یکی واسه عدد اول یکی واسه عدد دوم بعد خروجی هر کدام از اینها که یک بیت هستند را به FA میدهیم و جمع میکند SUM را در رجیستر دیگری ذخیره میکنیم و کری را از طریق فلیپ فلاپ دی به بقیه میدهیم تا با 2 بیت بعدی جمع شوند. فلیپ فلاپ کری واسه بیت اول صفر هست. دقت کن با هر کلاک یکی به راست حرکت میکنند و کری هم تازه میاد اونور میرود واسه بیت بعدی. یک روش دیگر داریم که خروجی FA را دیگر در رجیستر نریزیم به عنوان SI بدهیم به رجیستر های ورودی. واسه رجیستر دوم هم خودش را به عنوان ورودی میدهیم تا خودش را حفظ کند. شمارنده آسنکرون یا شمارنده موج گونه یا ریپل، T-FF وقتی T مساوی 1 باشد فرکانس کلاک را نصف میکند. D-FF که ورودی آن از کیو بار هست هم همین خاصیت دارد یعنی فرکانس کلاک را نصف میکند. ولی شکل ریپل یا موج گونه به گونه ای هست که دیگه مدار سنکرون نیست و خروجی اولی کلاک میدهد به دومی. و مشخص هست که FF اول فرکانس آن نصف کلاک هست بعد FF دوم چون فرکانس آن هم نصف کلاک ورودی خودش هست پس میشود ¼ کلاک اصلی و FF سوم هم 1/8 کلاک اصلی هست. اونکه هعی عوض میشود LSB هست. اولی با هر ضربه مثبت کلاک عوض میشود دومی وقتی عوض میشود که اولی از صفر به یک بیاد یعنی در لبه مثبت. لبه مثبت یعنی از صفر به یک و لبه منفی یعنی از یک به صفر. ریپل سخت افزاری کمتری دارد و گیت نمیخواد بر خلاف سنکرون که نیاز داشتیم و بدی این هست که اینها با هم تغییر نمیکنند و حالت گذرا هم داریم این وسط یعنی یک لحظه یک سری اعدادی میبینیم که درست نیست که این بدی هست که خروجی با هم عوض نمیشود. لبه مثبت شمارنده را نزولی کرد همین شکل در لبه منفی بگیری شمارنده میشود صعودی به شرطی که از کیو کلاک بعدی را تغذیه کنی یعنی دقیقا همین شکل. این کلاک دارد ولی مشترک نیست.

وقتی در ورودی یک کلاک نات میگذاریم یعنی لبه منفی هست و وقتی فعال میشود که اونجایی که بهش وصل هست از 1 به صفر بیاد. دقت کن FFیی که ورودی هاش از قبل مشخص هست و به چیزی وصل نیست لازم به نوشتن فرمول نیست چون مشخص هست. دقت کن فرکانس رو از تبدیل صفر به 1 اون FF بدست بیار.

**فصل هفتم:**

**جلسه اول**:

مد چند بودن یعنی چند حالت را میشمارد. Ring: 1 را میبرد سمت راست و مدام rotate میدهد. ff تی هم وقتی لبه مثبت بود و یک بود شروع به عوض کردن خروجی میکند. دقت کن وقتی سی 1 میشود تو خروجی بعدی تغییر میکند ff تی ما، دقت کن جایی که برگشت به حالت اول ببین چند تاست مد همون هست. وقتی به جانسون میدهیم از 6 به 12 میرسد در واقع منطق آن به این شکل هست که انگار اون ring خودش 3 تا میشمرد بعد میدهد به فلیپ فلاپ 2 برابر میکند میشود 6 تا. یا اگر جانسون باشد خودش 6 تا شمرده هست میدهد به یک ff میشود 12 تا. کلا وقتی دیدی ورودی یک سیگنال نات هست پس active low هست. برای فرکانس فقط نگاه کن از صفر به یک چند تاست.

طراحی یا سنتز مدارات ترتیبی سنکرون: در تحلیل قبلی یک مدار به ما میدادند و ما بر اساس حالت ها پیش بینی میکردیم حالت بعدی چی هست نتیجه تحلیل یک دیاگرام حالت بود حالا بحث طراحی هست یعنی مثلا به ما میگویند یک شمارنده بساز در طراحی حالت فعلی و حالت بعدی را داریم جنس ff را هم میدانیم ولی ورودی را نمیدانیم و میخواهیم بدانیم چی باشد تا مناسب ما باشد. قبلا یک جدول مشخصه یا جدول حالت داشتیم ولی الان جدول تحریک داریم، دقت کن تی XOR اون کیو و کیو بار هست و دی همان کیو بار هست. دقت کن شمارنده گری عین کارنو هست. دقت کن ما دنبال ورودی ها هستیم که ببینیم ورودی فلیپ فلاپ ها چی بوده است، فقط اگر جدول حالت دادند به مقداری که میخواهی تو کارنو بریزی دقت کن. در محاسبه ادامه جدول حالت دقت کن واسه T باید فقط XOR کنی تمام. دقت کن که اول به ما جدول حالات را میدهند بعد ما باید به ازای هر متغیری که داریم که نشان دهنده یک ff هست بریزیم در کارنو و به صورت SOP ساده کنیم بعد از روی فرمول های بدست آمده ورودی های FF را مشخص بکنیم.

روش سریع طراحی JK: باید A\* رو طبق فرمول حالت بعدی بدست بیاوریم در واقع فقط کافی است از همون جدول حالتی که سوال به ما داده است برویم و کارنو بکشیم. یعنی واسه A\* یک کارنو بعد همون کارنو را بیا به دو قسمت تقسیم کن نات جایی که A صفر هست و جایی که 1 هست خود متغیر بعد بیا هر کدام از دو قسمت را جداگانه ساده کن. بعد اون A رو در حالت بکش بیرون. اگر A\* را بدست آوردی ضریب A بار میشود J و ضریب A میشود K بار و الان همه فرمولی که میخواستیم را بدست آوردیم. در هر کارنو مربوط به FF از اون متغیر مربوطه فاکتور بگیر. واسه SR روش دوم رو نداریم و فقط روش اول هست. دقت کن در مدار های میلی میتواند چندین ورودی و خروجی باشد. در جدول حالت دیاگرام اونایی که نیستند کلا don’t care بگیر باید به تعداد ff هایی که داریم و یکی هم به ازای خروجی کارنو بکشیم. کارنو مثلا در این مثال 4 متغیر هست، واسه نوشتن در اون جدول کارنو برو ببین اون 4 تا در اون جدول به ازای اون شماره ای که دارند چه مقداری دارند بعد همون مقدار را در کارنو قرار بده. دقت کن اینکه با کدوم FF بسازیم دلخواه ما هست. دقت کن واسه هر کدوم از اون 4 تا باید 4 تا کارنو بکشیم بعد برید ببینید تو اون خونه ها اونی که کارنو مربوط بهش هست چه مقداری دارد. همیشه هم دقت کن کلا T، XOR هست از A و A\*. واسه T-FF باید T اون 3 تا متغیر و خروجی که همان Y هست کارنو بکشیم A,A\* واسه TA و ... ، باز هم دقت کن مقادیر کارنو از جدول حالت میاد. با JK هم دقیقا همین شکل هست از اون روش دو برو که 2 تا کارنو میکشی تقسیم میکنی هر کارنو را. فرمول هایی که در نهایت بر اساس اون باید مدارات را بسازیم از ساده سازی کارنو بدست میاد.

**جلسه دوم**:

تشخیص الگو: خروجی را همیشه 0 تولید میکنیم مگر اینکه دنباله را مشاهده کنیم که یک میشود، برای طراحی این مدارات باید دیاگرام حالت بکشیم، به ازای هر بیت حالت را عوض میکنیم، بعد اگر اون الگو کشف شد دوباره میرویم سراغ حالت اول دقت کن تخصیص حالت را به ما میدهند کلا روال این هست که اول اون چیزی که دادن را به ماشین تبدیل کنیم بعد حروف را به 0و1 تبدیل کنیم بعد جدول حالت بکشیم دقت شود که جدول حالت را طبق حرکت های ماشین حالت پر میکنیم x همان صرف رشته ما هست و y هم عدد بعد پرانتز که خروجی ما هست. البته باید بعد دیاگرام حالت مدار را بکشی یا سنتز کنی، 4 تا حالت داریم پس به هر کدوم باید بیت اختصاص دهیم چون 4 تاست طبیعی هست که باید 2 بیت اختصاص دهیم یا فلیپ فلاپ چون هر فلیپ فلاپ یک بیت هست. دقت کن از ما میپرسند که چند کارنو باید بکشیم خوب طبق دیاگرام حالت میبینیم به چند ff برای نمایش دادن نیاز داریم بعد یکی بیشتر نیاز داریم بخاطر خروجی.

سوالات تخصیص بیت را به ما میدهند. روال طراحی این هست که از روی توصیف مسئله دیاگرام حالت بکشیم بعد حالات را به بیت تبدیل کنیم یعنی شماره بهش اختصاص بدهیم بعد جدول حالت بکشیم. بعد جدول حالت هم کارنو بکشیم برای ساده سازی. دقت کن نوع FF که میگوید طراحی کن را دلخواه هست. دقت کن با ساده سازی کارنو میتوانی ورودی FF ها را بدهی همچنین اگر گفتند چند کارنو نیاز داریم ببین چند حالت داریم بعد به چند بیت یا FF نیاز داریم برای اتلاق بیت به آنها بعد بکش دقت کن یک کارنو هم همیشه واسه خروجی باید بکشی. دقت کن همه اینها واسه مداری هست که تشخیص الگو داشته باشد. حالا با دیاگرام مور: نکته اول این هست که عین میلی هست یعنی تا وقتی الگو را ندیدی خروجی صفر هست. نکته بعدی این هست که حالات مور از میلی بیشتر هست. نکته بعدی این هست که اندازه کارنو ها بر خلاف میلی یکسان نیست بلکه ff ها به اندازه 4 متغیر و خروجی 3 متغیر هست چرا که به خروجی وابسته نیست و x مستقیم در خروجی ظاهر نمیشود. دقت کن به ازای خروجی یک کارنو باید حتما بکشی بعد اونجا چرا 3 تا FF داشتیم ؟ چون 5 تا حالت داشتیم دقت کن تعداد متغیر کارنو ها یکسان نیست مثل قبلی. دقت کن دیاگرام حالت باید از روی الگو طراحی شود. Overlap اگر داشته باشیم یعنی اگر الگو را دیدیم و در ادامه آن با یک عضوی از دنباله اول دنباله دوم هم که الگو هست دیدیم موردی ندارد ولی اگر همپوشانی نداشتیم میگوید دنباله را دیدی بزار کنار از بیت بعدی برو دنبال الگو بگرد. پیش فرض این است که همپوشانی مجاز هست. همپوشانی روی دیاگرام حالت تاثیر میگذارد. اگر میگفتند همپوشانی مجاز نیست برو شروع اگر مجاز بود میای حالت های وسط. دقت کن خروجی وقتی 1 میشود که الگو دیده شود.

کاهش حالت: کاهش حالت یا ff ها را کم میکند یا باعث میشود که مقادیر دونت کر زیاد شود که ساده سازی بهتر شود در هر صورت خوب هست. در دیاگرام حالت باید کاهش حالت بدهیم بهتر هست حالات تکراری که یکسان یا معادل هستند را حذف کنیم تا FF ها کمتر شود. البته همیشه اینطوری نیست. خوبی اینکه مثلا از 7 بشود 5، don’t care زیاد میشود و بهتر ساده میشود و ترکیب ها کمتر میشود. اول باید ماشین را به جدول حالت تبدیل کنی شرط اول این هست که 2 حالت ابتدا خروجی یکسان داشته باشند به ازای ورودی یکسان، دوما حالت بعدی هم نباید سازگار باشد یعنی اینکه مثلا خروجی a b یکسان بود باید ببینی حالاتی که میروند با هم سازگار هستند یا نه اگر سازگار بودند میتوانی ادغام کنی. جدول رو بکش عین نظریه زبان اون خونه هایی که ضربدر نخوردند را با هم معادل بگیر. و از اون خروجی معادل نتیجه میگیریم که میشود یکی از حالت های طرفین تساوی را حذف کنیم. به این روش جدول ایجاب گفته میشود. در جدول ایجاد هر حالت دقیقا یکبار با حالت دیگر مقایسه میشود برای پر کردن این جدول اول اونایی که خروجی یکسان در یکی از دو الفبا را ندارند ضربدر بزن، بعد اونایی که خروجی یکسان دارند ببین حالاتی که میروند با هم سازگار هستند یا نه پس داخل اون خانه ها حالتی که میروند را بنویس و ببین اونا ضربدر خوردند یا نه. بعد اونایی که یکسان بودن را یکی را حذف کن اونی هم که حذف شد هر جا اومده بود حالت معادل آن را بگذار. روش افراز: بیاید حالات را دسته بندی کنید. اونایی که خروجی یکسان دارند را بگذار در یک دسته و این خاصیت هر دسته هست. بعد از این بررسی کنید حالات بعدی هم هر کدوم در یک دسته هستند یا نه دقت کن به ازای هر ورودی بررسی کن که به حالات بعدی رفتند که با هم در یک دسته باشند؟. بعد به دسته های کوچکتر افراز کن اگر در یک دسته نبودند حالات بعدی آنها اینقدر افراز بساز که دیگر کوچکتر آن نشود افراز هم به این شکل هست که اونایی که به حالت مشترک رفتند یک دسته اونی که به حالت جدا رفته بود یک دسته دیگر. دقت کن یک مرحله بعدش برو اگر افرازی رخ داد. بعد تعداد دسته های باقی مونده میشود همون تعداد حالت کوچکتر شده. پس مرحله اول خروجی مرحله دوم به حالات در یک دسته باید بروند. دقت کن دسته دوم باید به یک چیزی که یا در همان دسته هست یا مشترک با هم دیگر به یک حالتی میروند در یک دسته دیگر اونکه از این قاعده پیروی نکرد بیرون کن.

**جلسه سوم**: کاهش حالات با دونت کر ها: قاعده افراز برای دیاگرام حالاتی که don’t care دارند، با روش افراز نمیشود باید یک سری کار های دیگه کنیم. دقت کن که یعنی اگر خروجی دونت کر باشد یعنی اهمیتی ندارد خروجی چی باشد. در دونت کر دار ها تعدی بر قرار نیست. در 3 مرحله: 1. ابتدا جدول ایجاد میکشیم و زوج های سازگار را بیابیم. بعد در مرحله دوم سازگاری های ماکسیمال را بیابیم. و در مرحله آخر از بین سازگار های ماکسیمال حداقل تعداد را طوری بیابیم که پوششی و بسته باشند. تو روش جدول اول ببین که اونایی که خروجی های یکسان ندارند را ضربدر بزن. تو خونه هایی که باقی موندن برو نسبت به حروفی که اون تو دارند بررسی کن و مشترک ها را در نظر نگیر. اونجا هایی که ضربدر زدی ببین خونه هست به اینها وابسته باشد یا نه اگر بود ضربدر بزن. در جداول don’t care ندارند تعدی دارند یعنی الف با ب و ب با ث معادل باشند الف با ث معادل هستند ولی این واسه don’t care دار ها درست نیست. اون دایره با خطوط میشود نمودار ادغام یعنی یک دایره بکش نقاط روی دایره میشود همان حالات ما و همچنین پاره خط ها میشوند مجموعه حاصل از جدول ایجاب . هر شکلی حاصل شد تفکیک کن اونها میشود سازگار های ماکسیمال. دقت کن از بالا به پایین حرکتی میکنی یعنی اول چند ضلعی بعد مثلث بعد پاره خط بعد نقطه تنها. نقطه مثلث چند ضلعی و پاره خط بهش سازگار های ماکسیمال گفته میشود. دقت کن اگر بالایی نبود برود پایین. پس با این نمودار ادغام میتوانیم تعداد حالات را کاهش بدهیم. البته فقط اونها که از نمودار بدست بیاد نیست چون دقت کن باید پوششی باشد، یعنی باید به طور حداقل طوری انتخاب کنیم که همه ضلع ها را پوشش دهد یا همه حالات را و این حالات حاصل یعنی اینکه ماشین 5 حالته ما به 2 حالت کاهش پیدا میکنند، البته باید بسته هم باشد یعنی هر حالتی که انتخاب کردیم که در دسته هست باید ایجاب کننده آن هم در دسته هایی که انتخاب کرده ایم باشد یعنی دسته ایجاب باید یکسان باشد.. تو جدول ببین اون مثلا 3 تا که در یک دسته هستند به کجا میروند بعد هر جا که رود اون دسته که نام مستعار دادیم را بنویس یکی اون دسته ای که میروند را باید بنویسی یعنی ABD هر کدام با X=0 به چه حالتی میروند و این حالات در کدام دسته هستند و اینکه خروجی آنها چی هست اگر حداقل یک، 1 داشتیم بنویس خروجی را 1 همین برای صفر اگر همه خروجی ها دونت کر بودند بزن دونت کر دقت کن همزمان نمیتوانند صفر و یک بیاد چون بیاد یعنی اینکه معادل نبودند. دقت کن اگر همه حالات چند ضلعی دایره را انتخاب کنی دیگر لازم نیست بسته بودن را چک کنی. خروجی هم بنویس اگر همه don’t care بودند بزن همین اگر حداقل یکی 1 بود بزن 1 و همین برای صفر. نمودار ادغام از حالت های جدول ایجاب که تیک خورده اند شکل میگیرد. به اونایی که از جدول بدست میاد سازگار های ماکسیمال گفته میشود ولی بدرد ما نمیخورد چون باید پوششی و بسته باشند. اگر بعد از همه اینکار ها بسته بودن به وجود نیومد اول بیا اون مشترک رو به نوعی خط بزن که پوششی خراب نشه و بسته هم بشود اگر شد که هیچ اگر نشد باید اون یکی حالت ماکسیمال را با نیز حساب کنی. چند نکته: باید حالات در یک دسته را 2 به 2 چک کنی که ایجاب کننده آنها در یک دسته باشد، یا نیست برای بررسی کردن بسته بودن. دقت کن که لازم نیست حتما ایجاب کننده و حالات ادغام شده حتما در یک دسته باشند بلکه ایجاب کننده هر دو حالت باید در یک دسته واحد باشند و اگر نباشند یعنی انتخاب ما بسته نیست. برای بسته بودن میتوانی یک کار دیگر هم بکنی حتما لازم نیست حالت اضافه کنی میتوانی حالات را حفظ کنی اما برخی از حالت های تکراری را حذف کنی مثلا در مثال سوم شما نمیتوانی D را حذف کنی چون پوششی بودن اسیب میبیند پس حذف کردن باید بدون دست زدن به پوششی بودن باشد ولی میتوانی A را حذف کنی و مشکل بسته بودن حل شود به جای اضافه کردن حالات جدید. اگر بسته بودن به وجود اومد که هیچ اگر نیومد همون حالت اضافه کن. **تاخیر انتشار**: کلاک اومد چه قدر طول میکشد در FF، Q یا همان خروجی عوض شود. فقط وقتی سوال گفت راجب تاخیر در نظر بگیر آنرا. زمان راه اندازی یا setup time: حداقل مدت زمانی است که لازم است ورودی ff قبل از آمدن لبه کلاک مقدار خود را مشخص کرده باشد و عوض نشود یعنی اینکه ورودی اگر صفر هست و یک شود نباید حین لبه 1 شود بلکه باید از قبل 1 شده باشد به این زمان راه اندازی تثبیت گفته میشود که تا قبل لبه کلاک باید خروجی تثبیت شده باشد. زمان نگهداری: حداقل مدت زمانی است که لازم است ورودی ff پس از آمدن کلاک مقدار خود را حفظ کند و عوض نشود. یعنی اون مدتی باید خود را نگهدارد تا عوض نشود. پس ورودی ما قبل از کلاک به اندازه زمان راه اندازی و بعد از کلاک به اندازه زمان نگهداری نباید عوض شود. زمان نگهداری اگر کمتر از زمان تاخیر ff بود چون داخل همون تاخیر این هم حساب میشود در نظر نمیگیریم ولی به یک شرط به این شرط که زمان نگهداری ff دوم کوچکتر مساوی زمان تاخیر ff اول باشد چون اگر بیشتر باشد اون هعی میاد نگهدارد ولی ورودی جدید میاد. حداکثر فرکانس: کمینه پریود. دقت کن وقتی میخواهیم کلاک بدهیم باید ورودی فلیپ فلاپ ها هم اماده شده باشد هم تثبیت. پس باید دنبال کند ترین ff باشی. دقت کن تاخیر واسه همه ff ها هست. دقت کن تاخیر ورودی گیت ها را بیشترین آنها را بگیر. طریقه ساخت ff حساس به لبه: دو روش، روش اول ارباب و برده، در این ff دو تا حساس به سطح هستند مثلا یکی حساس به سطح مثبت و یکی حساس به سطح منفی و به صورت سریال به هم وصل هستند. ارباب حساس به سطح مثبت باشد برده حساس به سطح منفی میشود حساس به لبه منفی همین برعکس کنیم میشود حساس به لبه مثبت. روش 2: ff بساز که ذاتا حساس به لبه باشد.

دقت کن در مثال حداقل اندازه پریود کلاک برای چی زمان نگهداری را در نظر نگرفتیم؟ چون اون 8 ثانیه در داخل 70 ثانیه رعایت میشد ولی اگر زمان نگهداری 75 ثانیه باشد اصلا مدار به درستی کار نمیکند چون FF دوم میگوید من باید 75 ثانیه صبر کنم تا ورودی عوض شود ولی هر 70 ثانیه یکبار عوض میشود و مشکل ساز هست.

پریود کلاک: فاصله بین لبه مثبت تا لبه مثبت بعدی. فرکانس: عکس پریود. پس اگر حداکثر فرکانس خواستی مثل مثال قبل باید حداقل پریود را بدست بیاری.

ارباب برده به این شکل هست که یکی حساس به سطح مثبت هست یکی حساس به سطح منفی و این 2 تا به شکل سریال بهم وصل هستند. حالا اگر این 2 را داخل یک جعبه بگذاریم میشود یک ff حساس به لبه و یکی حساس به سطح مثبت و دیگری حساس به سطح منفی هست. دقت کن تا وقتی که سطح مثبت باشد نگاه میکنیم سطح تغییر کرد حفظ حالت هست!. و z انگار حساس به سطح منفی شده است چون وقتی سطح منفی شده عوض شده است. اگر ارباب حساس به سطح مثبت بود و برده حساس به سطح منفی میشود حساس به لبه منفی و برعکس همین برای حساس به لبه مثبت هست. روش اول کلک بود روش دوم میخواهیم ذاتا حساس به لبه بسازیم که دیگر از حساس به سطح استفاده نکنیم، از 3 لچ جفت NAND استفاده کرده است ( برای حفظ کردن فقط)، FF از نوع D هست و حساس به لبه مثبت هست. لچ جفت NAND با ورودی 11 میدانیم حفظ حالت هست.

نکات (**جلسه اول فصل هفتم**): در طراحی ما حالت فعلی و بعدی را داریم ولی ورودی فلیپ فلاپ ها را نداریم پس به یک سری جداول تحریک نیاز داریم تا ورودی ها را پیدا کنیم. این جدول حالت فعلی q و بعدی q\* مشخص هست. دقت کن جدول را خودت باید بسازی بر اساس جدول رفتار هر فلیپ فلاپ تعیین میکنی که اون ورودی چی بوده است مثلا وقتی حالت فعلی و بعدی 00 است به طور مشخص در SR ورودی S صفر بوده است که خروجی صفر شده هست و R هم مهم نیست چون اگر 00 بوده هست که همین جواب میشود حتی اگر 1 هم باشد چون ریست هست پس بازم صفر میشود پس don’t care میشود. برای 01 یعنی اینکه حالت بعدی 1 شده است پس قطعا s 1 بوده هست که ست شده است و r صفر بوده است. یعنی مسیر حرکت از q به q\* را ببین که از 0 به 1 شده است یا 1 به 0 شده است. دقت کن اگر kill فعال شد از 1 به 0 دیگر j اهمیتی ندارد چون حتی اگر 1 هم باشد مکمل هم باشد همین نتیجه هست. 11 هم یعنی kill نشده به هر حال حالا هر چیزی شده باشد اهمیتی ندارد j. در اینجا برعکس فرمول t هست و q XOR q\* شده است d هم برابر با q\* هست یعنی هر چی باشد همان هم d میشود. 2 بیتی یعنی 2 فلیپ فلاپ میخواد. واسه طراحی یک مدار اولین قدم جدول حالت و جدول تحریک هست. دقت کن برای راحتی کار 11 را زودتر بکش عین کارنو که خودش انگار یک جدول کارنو درون خودش دارد. معمولا تو صورت سوال جدول را میدهند. بعد انگار همون میشود جدول اگر عین کارنو بگذاری عدد گذاری ها را و بعد از ساده گذاری های کارنو همون 2 تا فلیپ فلاپ را بکش بعد با همان فرمول ها ورودی ها را بکش واسه A از کارنو A واسه B همینطور مهم ترین نکته این هست که توابع را بدست بیاوری. دقت کن جداول تحریک لزوما نام Q,Q\* ندارند بلکه میتوانند به نام خود فلیپ فلاپ ها باشند مثل A,A\* یا ... ، برای T FF دقت کن که باید A را ستون بعدی A\* بیای XOR کنی به ازای X متناظر همین هم برای T B. دقت کن همونجا تو جدول ساده کنی. در JK به Aو A\* باید نگاه کنی بعد اون حالات بیاد تو ذهنت مثلا اگر A صفر بوده و A\* هم صفر بوده یعنی jump نشده kill هم شده باشه اهمیتی ندارد چون به هر حال صفر مونده اگر 0 بوده بعد 1 شده یعنی jump شده و kill اهمیتی ندارد چرا چون حتی اگر 11 هم باشد مکمل میشود چه برسد به اینکه 10 باشد همین برای بقیه حالات که البته این روش طولانی هست. دقت کن بعد از بدست آوردن فرمول در جدول کارنو باید ورودی ها را بکشی. اما در روش دوم که سریعتر تو بیا واسه A\* کارنو بکش یعنی همون کاری که واسه D کردی رو اینجا هم انجام بده منتها دیگه مثل قبل ساده نکن بیا جدول رو به 2 نیم تقسیم کن اونجایی که A صفر هست را بکن A¯ و اونا رو جدا با هم ساده کن SOP و اونجایی هم که A یک هست را جدا ساده کن یعنی به 2 نیم تقسیم کن و ساده کن بعد فرمول ها بدست میاد چطوری؟ گفتیم که A ها را چه نات چه خودش فاکتور بگیر از جملات بعد ضریب A بار میشود J و ضریب A میشود K بار که باید نات بگیری تا خود K بدست میاد و اینجوری با یک کارنو 2 تا فرمول را مربوط به A بدست آوردی همین برای B فقط حواست به منفی و مثبت بودن ها باشد. واسه SR خودت بکش ولی دقت کن که روش دوم ندارد و فقط روش اول هست. در سوال بعدی دقت کن که درست هست که همه state ها را نداریم ولی تو میخوای بکشی باید همه را در جدول حالت بنویسی بعد هم حواست باشد چون میلی هست و میلی به خروجی وابسته هست باید جدا واسه اونم در نظر بگیری. بعد یک چیزی واسه خروجی دقت کن کاری به بقیه موارد نداری فقط ببین خروجی هات به ازای 0 ها چند هست و به ازای 1 ها چند هست همون رو بزار توی اون جدول به باقی چیز ها کاری نداشته باش. اونایی که تو اون حالت نبود بی اهمیت هست یعنی مهم نیست. مثلا یکی از سوالات میتواند این باشد که ما چند تا کارنو باید بکشیم خوب مشخص هست یکی واسه A\* و B\* و C\* و یکی هم Y و کارنو ها 4 متغیر هستند چرا چون ABC و X. واسه کارنو y هم که مشخص هست همینطوری 4 متغیر فقط مقادیر y را به ازای x=0 و x=1 قرار بده. واسه فلیپ فلاپ D که چون با Q\* برابر هست که هیچی ولی واسه T مثلا عین قبلی ها باید بری XOR A با A\* را بدست بیاوری که قبلا انجام دادی واسه بقیه متغیر ها هم همین صورت هست بعد هم به ازای TA باید کارنو ها را پر کنی. واسه JK هم روش دوم را برو.

**فصل 8:**

مدار آسنکرون مداری هست که کلاک ندارد فقط فید بک دارد پس یک بخش ترکیبی دارد با حافظه که فید بک را تولید میکند. حتی لچ میتواند باشد. اگر کلاک بخورد میشود سنکرون اگر کلاک نخورد میشود آسنکرون، اسنکرون سریع تر از سنکرون هست سریعتر هست چون تا ورودی عوض میشود تغییر میکند بدرد بلادرنگ میخورد همچنین ارزان تر هم هستند چون مدار تولید کلاک نیاز ندارند ولی طراحی آنها دردسر دارد. یکی از شرایط این هست که اگر چند تا ورودی داشتیم در هر زمان فقط یک ورودی را میتوانیم عوض کنیم. 2 فید بک یا 2 بیت حافظه داریم هر فید بک یک ff و اسم آنها متغیر تحریک هست با حروف کوچک هنگام ورودی و با حروف بزرگ هنگام خروجی نمایش میدهیم. بعد باید توابع را براشون بنویسی، بعد باید بریزی داخل یک جدولی مثل کارنو، داخل جدول مقادیر تحریک را مینویسیم. بهش جدول گذر میگوییم چون میفهمیم از چه حالتی به چه حالتی گذر میکنیم. یک جاهایی هست که خروجی با ورودی برابر هست بهش میگوییم حالت پایدار یعنی با صفر و ورودی صفر همونجا میمانیم. برای اینکه از روی جدول مدار طراحی کنیم اول باید جدول را به 2 جدول خروجی و تحریک برسیم. خروجی را به خروجی جدول نگاه کن و رابطه را براش بنویس و جدول تحریک را نسبت به حروفی که اختصاص دادیم نگاه کن بعد با ساده کردن رابطه را مینویسیم بعد طراحی میکنیم. ما در این مدار ها اجازه نداریم در هر لحظه یک ورودی بیشتر را تغییر دهیم یعنی 00 یهو 11 نمیتواند بشود.

توی جدول فقط تو حالات پایدار باقی میمانیم و در غیر پایدار باشیم میافتیم پایین و کلا خروجی عوض میشود اگر بخواهیم همزمان دو ورودی را عوض کنیم. ریس: دو بیت تحریک بخواهند همزمان عوض بشوند و اینجا ریس رخ میدهد یعنی مثلا یکی زودتر عوض شود ولی توی اون جدول اولی که مثال زد ریس داریم ولی بحرانی نیست چون باز به حالت پایدار میرسیم هر کی زودتر عوض شود. تشخیص ریس آسان است اگر در یک جدولی دیدیم حالت پایدار میخواهد تعویض شود به حالت ناپایدار و بیشتر از یک بیت میخواهد عوض شود ریس داریم بعد باید بررسی بحرانی بودن را بکنیم.

تخصیص حالت بدون ریس: یک جدول میدهد میگوید به حروف جوری بیت نسبت بده که ریس پیش نیاد، بهتر است یک نمودار انتقال بکشیم و ببینیم هر حروف به کدام حروف منتقل میشوند. اگر در این نمودار مثلث شکل گرفت یعنی بدون ریس نمیتوان بیت تخصیص داد در جدول 3 سطری. پس باید یک حالت اضافه کنیم بین b,c، و اونجایی که b به c میرود مستقیم نرود اول به d که جدید اضافه کردیم برود. حالت ها حداکثر دو برابر میشوند و بیشتر از اون نیست یعنی مثلا نهایتا 4 حالت باید اضافه شود و از 2 بیت به 3 بیت لازم هست برای تخصیص، اول باید یک جدول شبیه به کارنو بکشیم. مثلا به a یک مقداری بده بعد ببین به کدوم حالات گذار داره بیا مجاور آن بگذار اگر در مراحل بعدی بیت تخصیص دادی ولی مجاور هم نبود بیا یک سری حالات واسط بگذار. به این روش سطر مشترک گفته میشود. یک روش چند سطری هم داریم: الگوریتمی هست این روش و هر حالت را به 2 حالت تبدیل میکنیم لزوما بهینه هم نیست، پس به هر حالت باید 3 بیت نسبت بدهیم و یکی نات اون یکی میشود. یعنی به اولی یک چیزی نسبت بده به دومی نات آن را دقیقا. دقت کن حالات پایدار حالت جدید نات دقیقا زیر همون جاهای هستند که اولی پایدار بود. بعد اینجا نسبت تخصیص مهم هست چونکه وقتی مثلا از الف به ب میرویم باید به اونی برویم که 1 بیت اختلاف داریم از اون حالت جدید که برای ب در نظر گرفتیم. بعد جدولی که بدست اومد میزاریم تو کارنو و مدارات را میکشیم.

**فصل نهم:**

گیت اند با کلمه کلیدی and مشخص میشود یک اسم اختیاری میدهیم و در اینجا قانون این هست که اول خروجی باید نوشته شود بعد ورودی ها. ترتیب خطوط اهمیتی ندارد، شناسه ها مثل همون اسم ماژول یا اسم پورت و یا اسم گیت میتوانند شامل حروف الفبا انگلیسی و ارقام باشند و نباید با ارقام شروع شود از underline هم میشود استفاده کرد به این توصیف در سطح گیت گفته میشود. در این زبان 12 گیت پایه داریم در Verilog و 8 تا گیت معمولی و 4 تا سه حالت هستند. وقتی میخواهیم از اون 8 تا استفاده کنیم باید کل کنیم یا instantiate میکنیم.

مثلا اگر AND را فراخوانی کردی و چند تا گیت اند داشتی لازم نیست به ازای همه این نام را استفاده کنی یکبار اول بگو بعد بقیه رو فقط اسم و ورودی و خروجی ها را بنویس. عدد اول بعد از شارپ تاخیر rise هست یعنی خروجی اگر از 0 بخواهد 1 شود 4 واحد تاخیر دارد، دومی تاخیر fall هست یعنی اینکه از 1 بخواهد برود صفر 3 واحد تاخیر دارد. تاخیر ها توسط سنتز نادیده گرفته میشود و فقط بدرد شبیه سازی میخورد در اجرا نادیده گرفته میشوند.

گیت های not,buf فقط یک ورودی دارند ولی میتوانند یک تا چند خروجی داشته باشند بنابراین وقتی تابع را مینویسیم آخری از سمت راست همیشه ورودی و بقیه هر چند تا بود خروجی هستند. BUF همان 3 حالته ها هستند. از سمت چپ به راست: اولی خط کنترل یک باشد ورودی را مستقیم به خروجی میدهد اگر صفر بدهیم خروجی میشود Z یا قطع، دومی برعکس اولی هست از لحاظ ورودی، سومی اگر کنترل را یک بدهیم خروجی میشود نات ورودی و اگر صفر بدهیم میشود Z، چهارمی برعکس سومی هست. برای تابع این مدارات واسه تاخیر سه متغیر داریم که 2 تای اولی که مثل همان قبلی ها هست ولی سومی تاخیری که خروجی به Z تبدیل میشود هست. به جای استفاده از کلمات AND,NOT,OR,WIRE میایم و از کلمه کلیدی assign استفاده میکنیم خروجی را میگذاریم و فرمول تابعی آنرا مینویسیم. منتها لازم است که عملگرهای وریلاگ را بلد باشیم مثلا and:&& یا or=|| هست. parameter متغیر تعریف میکند با مقدار اولیه. Reg میگوید این متغیر ها 4 بیتی هستند، انگار 4 متغیر یهو تعریف کردیم. عملگر های مقایسه ای خروجی فالس یا ترو دارند. عملگر های بیتی روی بیت عملوند ها عمل میکنند. عملگر منطقی خروجی فالس یا ترو دارند. صفر یعنی فالس و یک یعنی ترو. صفر کلا فالس و 1 یعنی ترو. عملگرهای کاهشی: تک عملوندی هستند عملوند میگیرند روش عمل میکنند و یک خروجی میدهند چرا میگویند کاهشی چون چند بیت میگیرد و یک خروجی میدهد. شیفت چپ یعنی از سمت راست 2 واحد صفر اضافه کن. داخل [] بگذارند یعنی میگویند اینها مثلا برداری چند بیتی هستند. در وریلاگ به صورت پیشفرض دسیمال هستند ولی بقیه مبنا ها را هم میتوان نوشت. البته در کامپیوتر به صفر و یک تبدیل میشود. اون تعداد بیتی که اول میگذاریم یعنی تو چند بیت قرار هست ذخیره شود بعد مبنا هست بعد خود عدد. {} مقادیر را میاد با هم concatenate میکند. شبیه سازی: ورودی میدهیم تا تست کنیم کد ما خروجی درست میدهد یا نه و بعد میدهیم که سنتز شود. ما میتوانیم خودمون گیت تعریف کنیم و فراخوانی کنیم، با کلمه کلیدی primitive. دقت کن در جدول درستی خود وریلاگ میفهمد کدوم ها مال کدوم پورت ها هستند. دقت کن فقط سیم های داخلی را با wire مشخص میکنی. منطق یک ماژول مدل های مختلفی دارد، مثلا با assign بیایم توابع بولی بنویسیم مدل دیتا فلو یا مثلا از نظر سطح گیت بنویسیم یعنی گیت ها را بکشیم، مدل رفتاری هم داریم که سطح بالاتری دارد و با کلمه کلیدی always استفاده میکنیم و برای مدارات ترتیبی هست. اگر ; گذاشتی دوباره باید اون تابع primitive را فراخوانی کنی ولی , بگذاری نیازی به این نیست. در زبان وریلاگ برای مدل گیت دو نگاه پایین به بالا و بالا به پایین داریم از پایین به بالا تک به تک گیت ها را مشخص میکنیم از ریز ترین اجزا و بعد به طور کلی مینویسیم. برای ساخت FA 2 تا HA و یک OR را صدا بزنی. مالتی پلکس را میتوانیم با بافر 3 حالته بسازیم. کلمه کلیدی tri: اگر یک خروجی چندین درایور داشته باشد یعنی چندین ورودی داشته باشد که بخواهد یک خروجی را بسازد، از نوع tri معرفی کنیم. در دیتا فلو به جای گیت ها توابع را فقط مینویسیم. جلسه سوم: مدل رفتاری زیاد خودش را درگیر جزئیات نمیکند. دستورات داخل تابع به صورت خط به خط اجرا میشدند مثل assign نبود که تا وقتی تغییر نمیکند تا متغیر های سمت راست یک تغییری بکنند فقط وقتی تغییر میکند که متغیر های حساس تغییر میکنند یعنی دستورات بدنه به ترتیب فقط وقتی اجرا شوند که اون لیست حساس تغییر کند دقت کن میتواند بدون ترتیب نیز باشد. متغیر هایی که چیزی بهشون نسبت میدهیم باید reg باشد. 2 نوع انتساب داریم بلاک و نا بلاکی، به جای always از initial هم استفاده میکنند ولی اون موقتی هست و سنتز نمیشود. انتساب بلاکی ترتیبی انجام میشود ولی نا بلاکی به صورت موازی انجام میشود، در نا بلاکی اول سمت راستی ها محاسبه میشوند بعد انتساب داده میشود به چپی دقت کن مقادیر قدیم خود را میگیرند سمت راستی ها، بلاکی واسه ترکیبی و موازی واسه ترتیبی.

تا دقیقه 11 دیده شد.

داخل بدنه always اگر فقط یک دستور بود نمیخواد begin end بگذاری اگر بیشتر بود باید بگذاری. اگه میخوای با always کار کنی حتما نوع خروجی را باید از نوع رجیستر مشخص کنی. اون تابع فقط رفتار یک مالتی پلکسر را نشان میدهد اگر دنبال نوشتن تابعی بولی بودیم از assign استفاده میکردیم. اون حالات کیس که ننوشتیم اگر رخ داد اتفاقی رخ نمیدهد default هم دارد که در صورتی هست که هیچ کدوم از کیس ها رخ ندهد. شرط هم از بالا به پایین بررسی میشود. در وریلاگ دو کلمه کلیدی داریم واسه ff: posedge وقتی رفتیم به لبه مثبت و negede وقتی رفتیم به لبه منفی. Always یا همه متغیر هاش حساس به لبه هستند یا حساس به سطح. واسه هر پایه که آسنکرون هست باید یک if بگذاریم و آخرین else مربوط به کلاک هست.

دقت کن که اگر به صورت دهدهی بخواهی مقدار دهی کنی در وریلاگ 32 بیت یا 16 بیت در نظر میگیرد که خیلی زیاد هست، always ها از طریق مقادیر مشترک با هم فعالیت میکنند. در تابع میلی که کشیدیم اول کار always اول کار میکند که reset هست و تا وقتی صفر هست که هیچی وقتی 1 میشود میرویم به حالت بعدی وقتی حالت عوض میشود 2 تا always دیگر شروع میکنند به فعالیت، always دوم next state را برای لبه کلاک بعدی انتخاب میکند و لبه کلاک بعدی always اول تغییر حالت میدهد. always دوم به x in هم عکس العمل نشان میدهند و سومی خروجی و دومی حالت را مشخص میکند.

بخش چهارم:

انواع توصیف ها در وریلاگ: ساختاری: میای اجزا مدار را مشخص میکنی مثلا یک decoder داریم یا یک شمارنده داریم یعنی گیت ها را مینویسیم اگر هم اون گیت را نداشت با module خودمون را مینویسیم. از توصیف جریان داده یعنی از توصیف رفتاری بریم به توصیف ساختاری از ماشین حالات سخت افزار را بکشیم بهش میگویند سنتز کردن دقت هم کن که در توصیف جریان داده و یا ساختاری ترتیب مهم نیست و همروند هست یعنی میتونی جا به جا بنویسی مشکلی نمیخورد. در توصیف رفتاری ترتیب مهم هست. reg همون رجیستر نیست متغیر هست در واقع داریم یک چیزی تعریف میکنیم که مقداری بهش نسبت دهیم و باید مشخص کنیم از نوع reg هست اگر اشاره نکنیم از نوع wire دیفالت در نظر گرفته میشود. در always با هر ورودی که مدار با آن واکنش نشان میدهد باید بنویسیم. اگر اون کلمات کلیدی posedge را ننویسیم به این معنی هست که CLK با هر تغییر بیاد تو چه از صفر به یک چه از 1 به 0 . کلیر آسنکرون باشد با کلاک کار نمیکند از اینکه وقتی 1 میشود خروجی صفر میشود میفهمیم active high است. CLK وقتی از 0 به 1 بیاد یعنی مدار کلاک خورده هست پس به خروجی یکی اضافه میکنیم که این انگار شمارنده هست. وقتی 3 بیتی تعریف میکنیم و نگفته باشیم بیت 4 ام را جایی ذخیره کند از 111 میرود به 000 بعد از کلاک خوردن. خروجی از جنس wire تعریف میشود.

توصیف رفتاری جمع کننده 4 بیتی وقتی always را گفتیم دیگر کاری نداریم چه لبه ای هست به ازای هر تغییر A و B را بیار تو و جمع کن. انتساب نا بلاکی به این معنی هست که مثلا یک سری متغیر میخواهند ریخته شوند درون متغیر دیگر این صورت نمیگیرد تا سری بعدی که دوباره به این always رسیدیم. یعنی بعد دوباره که حساس شدیم انتساب ها انجام میگیرد. نا بلاکی یعنی مقادیر را عوض کن و بلاکی یعنی مقادیر یکی میشود. Active low گفت یعنی negedge. یعنی وقتی 0 شد کاری انجام بدهد. If گذاشتی یعنی آسنکرون هست و به کلاک کاری نداشته باش. SIN یعنی serial input. این اصلا مشخص هست که شیفت رایت هست که از سمت چپ SIN وارد میشود.

Ff فقط وقتی کلاک عوض شد حساس میشود ولی بافر هم به کلاک کار دارد هم به ورودی، دقت کن که نات z هم معنی ندارد پس اگر یک تابعی اومد کل عبارت شرطی را نات کرد و z هم نات شد معنی ندارد. دقت کن کلاک هم به صورت posedge هست و دقت کن اگر صحبت راجب active-high کرد یعنی حتما باید posedge باشد. آسنکرون هم یعنی به کلاک نگاه نکن مثلا 0 شد خروجی را فلان کن. و اگر یک چیز آسنکرون بود نباید در ورودی FF ها باشد چون به کلاک وابسته میشود. دقت کن اگر یک پایه به FF هایی وصل شود در قسمت آسنکرون حتما در قسمت always باید باشد چون به کلاک وابسته نیست دلیل رد گزینه 4 و اگر قرار هست چیزی به کلاک وابسته باشد باید در ورودی ff ها تاثیر بگذارد. و اگر اون پایه ای که داریم در always نیاد یعنی با 0 و 1 شدن کاری انجام نمیدهد پس نباید هم تاثیری داشته باشد و به دنبال مداری هستیم که تاثیر نداشته باشد.

کلمات کلیدی با حروف کوچک نوشته میشوند. Primitive ها فقط یک خروجی دارند. برای براکت موقع تعریف پشت متغیر و موقع استفاده جلوی متغیر میگذاریم. @ این علامت صبر میکند تا جلوی آن تغییر کند. ورودی های آسنکرون حتما باید در لیست حساسیت بیان و اونایی که سنکرون هستند را نباید بیاری فقط باید کلاک را بیاری.

در assign به wire مقدار میدهیم در ساختاری هم همینطور ولی در always فقط به reg ها مقدار میدهیم. بدنه assign و always به طور موازی یا همروند با هم اجرا میشوند.