به نام خدا

FPGA

www.Techno-Electro.com

مهندس حامد سقایی

۱. مقدمه

در این بخش به بررسی اهمیت FPGA برای پیاده سازی مدارهای منطقی می پردازیم. Field Programmable Gate Array یک IC کاملا قابل برنامه ریزی است. این IC را به هر مدار منطقی دلخواهی که بخواهیم می توان تبدیل کرد. اولین نقص FPGA این است که کاملا Digital است، اگرچه در دنیای امروز دیگر نمی توان این را یک نقص به شمار آورد. اکنون تمام بخش های صنعت به سمت Digital شدن پیش می رود. ایده اولیه بسیار ساده است. سیگنال آنالوگ از منبع تولید آن به یک مبدل Analog به Digital می رود، از آن مکان به بعد کلیه مدارها
به Analog تبدیل می شود و به محلی که قرار است از آن استفاده شود می رود.

با FPGA می توان یک میکروپروسسور ساخت. یا می توان یک Mux بزرگ طراحی کرد که چند درگاه انتقال داده را به هم وصل می کند. می توان یک فیلتر FIR را با آن پیاده کرد. می توان برای محاسبه FFT با سرعت زیاد از آن استفاده کرد. می توان با آن یک Up/Down Counter با عرض دلخواه n بیت ساخت. با یک FPGA ی ارزان که به راحتی در بازار پیدا می شود ، می توان تا ۸۰۰ میلیون ضرب را انجام داد. اگر قرار بود بخواهیم برای این کار از یک میکروپروسسور استفاده کنیم، هزینه چند برابر می شد. خلاصه ساختار داخلی FPGA به نحوی است که می توان آن را برنامه ریزی کرد تا تبدیل به هر مداری شود. عملیاتی که روی FPGA انجام می شود تا تبدیل به یک مدار خاص شود را Configuration می گویند.

۱.۱ محاسن استفاده از این آی سی ها

FPGA بسیار انعطاف پذیر است، و به راحتی می تواند جایگزین بسیاری از مدارها شود. مثلا یک بورد بزرگ که یک دستگاه خاص را کنترل می کند ، و شامل یک مدار منطقی پیچیده است را می توان با یک بورد کوچک که فقط یک FPGA روی آن قرار دارد جایگزین کرد. این دفعه تمام آن مدار کنترلی به داخل FPGA منتقل شده است. این اتفاقی است که هر روز دارد در دنیای ما رخ می دهد. بوردها هر روز کوچکتر می شوند، در واقع تمام اجزایی که زمانی یک بورد بزرگ را می ساختند، اکنون با پیشرفت صنعت نیمه هادی، در یک IC کوچولو خلاصه می شوند. مثلا یک نگاهی به کارت گرافیکی خود بندازید، فقط یک IC توپول خواهید دید، تمام مدارهای دیجیتال و آنالوگ لازم داخل همین قرار دارد.

در یک تقسیم بندی ساده می توان گفت FPGA در دو حوزه استفاده می شود:

۱- محل هایی که قرار است یک تعدادی اعمال کنترلی با سرعت نه چندان زیاد انجام شود.

۲- سیستم های انتقال داده و مخابراتی.

خلاصه: اول اون جاهایی که سرعت مهم نیست. دوم اون جاهایی که سرعت مهمه. مثلا فرض کنید می خواهیم کنترلر آسانسور یک ساختمان را با FPGA بسازیم. این مثالی از حالت اول است. یک FPGAی ساده چون Spartan برای این کار کافیست.

یا فرض کنید می خواهیم یک کنترلر درگاه AGP (Accelerated Graphics Port) بسازیم. این درگاهی است که برای تمام کارت های گرافیکی سه بعدی استفاده می شود و سرعت انتقال داده در اون به حدود ۱۰۰۰ مگابایت بر ثانیه می رسد. این مثالی از حالت دوم است. برای ساختن یک همچین چیزی نیاز به یک FPGAی پیشرفته مثل Virtex-II و یا APEX-II داریم.

در مورد اول معمولا مدارهای منطقی برای کنترل دستگاهها به کار می روند. هر دستگاه برای خودش یک بخش کنترلی دارد که مثلا سرعت چرخش موتورها و زمان روشن و خاموش بودن هرکدام را تعیین می کند. در عین حال یک کنترل عمومی بر تمام این بخش ها وجود دارد. سنسورها اطلاعات مربوط به هر بخش (مثلا دمای مایعی که مراحل مختلف عملیات شیمیایی دارد به ترتیب روی آن انجام می شود.) را به یک مرکز کنترلی منتقل می کنند. سیاست های کلی سیستم بر اساس پردازش انجام شده روی داده های دریافتی از سنسورها، اتخاذ می شود. سیگنال های کنترلی مربوط به هر دستگاه بر اساس این سیاست ها به دستگاه ارسال می شود. پس می بینیم که در هر دو سطح پردازش نقش عمده ای را ایفا می کند و از طرفی می دانیم که بهترین وسیله برای انجام پردازش روی داده ها مدارهای منطقی هستند. حالا مثلا ممکنه روی هر دستگاه یک FPGA این کار رو انجام بده. یا حتی پردازنده اصلی هم می تونه FPGA باشه.

پس در مدارهای کنترلی به طور بسیار وسیعی می توان از FPGA استفاده کرد. معمولا هنگامی که از FPGA برای این گونه مدارها استفاده می شود نیاز به عملکرد بسیار سریع و تعداد گیت زیاد وجود ندارد. FPGA های کوچک و معمولی می توانند نیاز این بخش را فراهم کنند.

در مورد دوم یعنی مدارهای سرعت بالا، FPGA کاربرد بسیار زیادی دارد. مبنای بسیاری از سیستم های مخابراتی آن است که روی تمام داده ها الگوریتم یکسانی انجام شود. معمولا نیاز است که اجرای این الگوریتم با سرعت زیادی صورت پذیرد. برای فهم بهتر این مثال را در نظر بگیرید:

یک شخص می خواهد با گوشی سیار خود یک تماس تلفنی با یک محل برقرار کند. در ابتدا نیاز است از میان سرویس دهنده های مبایلی که گوشی می تواند با آنها ارتباط برقرار کند ، یکی انتخاب شود. این کار با انجام محاسباتی و فرستادن پیامهای مختلف بین سرویس دهنده ها انجام می شود. سیگنالهای کنترلی به گوشی می گویند که با کدام سرویس دهنده ارتباط برقرار کند. تمام محاسبات طی این مراحل به صورت رقمی و توسط مدارهای منطقی و پردازش گرها انجام می شود. از آنجا که تعداد متقضیان زیاد است محاسبات مربوط به هرکس باید سریع انجام شود تا اولا استفاده کننده ناراضی نوشد و ثانیا نوبت دیگران هم برسد. ارتباط که برقرار شد لازم است کارهای زیر انجام شود: حرفهای شخص دیجیتایز شود، سپس کدگذاری شود، Header و Footer مناسب به آن اضافه شود تا معلوم باشد این بسته داده مربوط به کیست ، سپس ارسال شود. در سرویس دهنده مبایل برای به

دست آوردن داده اصلی لازم است عکس اعمال بالا انجام شود. حال بسته داده (که در واقع صحبت های شخص است.) لازم است به محل مناسب ارسال شود. این کار به کمک سوئیچ های مختلف انجام می گیرد. یک سوئیچ داده ها را دریافت کرده هر داده را به محل مناسب می فرستد. سوئیچ باید بسیار سریع کار کند تا مثلا بتواند نیاز ده هزار نفر را بطور همزمان بر آورده کند.

برای طراحی این نوع مدارها، FPGA ایده آل است. چراکه الگوریتهایی نه چندان پیچیده لازم است روی مقدار بسیار زیادی داده با سرعت بسیار زیاد انجام شود.

پس اگه خواستیم ببینیم ، برای پروژه ای که می خواهیم انجام بدیم، FPGA خوب هست یا نه، نگاه می کنیم به نوع محاسباتی که باید انجام بدیم. اگه لازمه یک سری الگوریتم های ساده (مثلا کانولوشن) روی یک حجم زیادی داده بطور یکسان انجام بشه (الگوریتم سادس ، ولی حجم ردازش زیاد) FPGA را حل خوبیه. ولی لگه الگوریتمی که قراره پیاده بشه ، پیچیده باشه، یا اینکه بخواهد برای داده های مختلف متفاوت باشه (برای همه داده ها قرار نباشه یک کار یکسان انجام بدیم.) استفاده از یک میکروکنترلر یا میکروپروسسور که سرعت لازم رو داره، مناسب تره.

FPGA هایی که برای مدارهای مخابراتی استفاده می شود، معمولا باید پرسرعت بوده، تعداد گیت بسیار زیادی داشته باشند تا بتوان مدار بزرگی را روی آنها پیاده کرد.

۱.۲. در باره آزمایشگاه تحقیقات شبکه

آزمایشگاه سوئیچینگ (که متاسفانه در زمان اصلاح دوم این متن، دیگر در دانشکده برق دانشگاه صنعتی وجود ندارد و به شهرک علمی تحقیقاتی منتقل شده) (و حالا در زمان اصلاح سوم این متن، اسمش عوض شده، به آزمایشگاه تحقیقات شبکه و سوئیچ یا یه همچین چیزی!) محلیست که یک پروژه بزرگ در آن در حال اجراست (ساخت یک NP یعنی یک Network processor با سرعت 2.5 گیگابیت بر ثانیه). برای انجام این پروژه مخصوصا از FPGA استفاده می شود. تا زمان اصلاح سوم این متن هنوز هیچ کار عملی انجام نشده. خوب در واقع هیچ نیازی هم نیست که ما خودمون مدارها رو ببندیم و تست کنیم. با استفاده از شبیه سازها ما به دقت می تونیم صحت عملکرد مدار رو چک کنیم. اینجا ذکر یک نکته ضروریه: وقتی یک مدار خیلی بزرگ رو میخواهیم طراحی و پیاده سازی کنیم دو حالت وجود داره: یا امکان ساخت IC در اختیارمون هست یا نیست. تو حالت اول، اصلا نیاز نیست از FPGA استفاده کنیم. می تونیم یک ضرب طراحی رو برای ASIC انجام بدیم. (منظور از ASIC اون IC ای هست که یک شرکتی که خط تولید سیلیکون داره و تکنولوژی IC ساختن داره، مدار ما رو توش پیاده می کنه. این IC دیگه فقط یک کار می کنه و قابل Configure نیست. فقط همون طراحی که ما انجام دادیم اینجا توی IC پیاده شده.) اما وقتی شما هیچ کارخانه سازنده IC در اختیارت نباشه، یکی از بهترین روشها اینه که اول مدار رو روی FPGA پیاده کنیم و اینقدر بهینه سازی انجام بدیم تا مطمئن بشیم مدار کاملا درست و با یک فرکانس خیلی خوب می تونه کار کنه. وقتی به این مرحله رسیدیم طی مدت زمان کوچیکی می تونیم طرحمون رو برای یک

تکنولوژی خاص پیاده کنیم. مثلا فرض کنید الان کارمون خلاص شده. نگاه می کنیم می بینیم توی تایوان شرکت TSMC تکنولوژی ساخت IC با دقت 0.13 میکرون رو داره، می گیم چه خوب! می ریم اونجا. می گیم آقا چقدر می گیری این مدار ما رو که مثلا 500 هزار گیت داره برامون IC کنی؟ مثلا میگن 700 هزار دلار. بر می گردیم و مطمئن می شیم مدار ما همه چیزش کاملا درست کار می کنه. چون شرایط FPGA و ASIC تقریبا یکیه ، اگه روی این خوب کار کنه، احتمال درست کار کردنش روی اون هم خیلی زیاده. اینجا باید حواسمون رو خیلی جمع کنیم، اگه مداری که دادیم ASIC کنن، غلط داشته باشه، یه 700 هزار تای دیگه باید بدیم تا مدار بدون عیب رو دوباره برامون بزنن. خوب فرض کنید روی FPGA مدار ما 125 مگاهرتز کار می کرده. حالا که می ره روی ASIC مثلا میشه 400 مگاهرتز. حالا ببنیم فرق اصلی تو کجاست: ASIC که ساخته شد و به تولید انبوه رسید ، مثلا هرکدوم از IC های ما با قیمت 200 دلار فروش میره. در حالی که اگه می خواستیم روی FPGA طرحمون رو بفروشیم، فقط خود FPGA ی خالیش اقلا 4000 تا 5000 دلار قیمت داره. می بینیم که فروختن طرح روی FPGA تقریبا غیر ممکنه، و FPGA اینجا فقط به عنوان یک وسیله که درستی مدار روش تست میشه کاربرد داره.

حرف بالا وقتی که با مدارهای کوچیک سر و کار داریم اصلا درست نیست: فرض کنیم ما تو یک شرکت کوچیک هستیم که کنترلر پورت USB تولید می کنه. کل مدار کنترلر کمتر از 20 هزارتا گیته و به راحتی داخل یک FPGA ی XC2S50 (که یک Spartan-II با 50 هزار گیته) جا میشه. حالا آیا به نفع ماست که باز هم کنترلرمون رو ببریم روی ASIC؟ این دفعه دیگه نه. قیمت هر XC2S50 چیزی کمتر از 10 دلاره، پس به راحتی می تونیم تعداد زیادی از این FPGA ها رو بخریم و روی بوردهامون از همین FPGA ها استفاده کنیم. این کار ، علاوه بر اینکه اون پول چند صد هزار دلاری برای ساخت ASIC رو لازم نیست بدیم، حسن های دیگه ای هم داره. اولا که مدارمون اگه توش عیب پیدا کردیم، خیلی سریع، بدون اینکه کوچکترین مشکلی بریا حتی بوردهایی که تاحالا ساختیم پیش بیاد ، عیب رو رفع می کنیم. دوم اینکه می تونیم مدار هامون رو Upgrade کنیم بدون اینکه نیاز به عوض کردن بورد باشه. مثلا فرش کنید بورد و نرم افزار مربوط به اون رو طوری طراحی می کنیم که وقتی کامپیوتر طرف وصل شده به اینترنت ، بورد بتونه به سایت ما وصل بشه و مدار جدید و Update شده رو از سایت بگیره. حالا فقط کافیه این مدار جدید ریخته بشه توی اون PROM ای که اطلاعات Configuration مربوط به FPGA توش ذخیره شده. دفعه بعد که سیستم رو روشن می کنیم، در واقع داریم از یک مدار جدید برای کنترلر USB استفاده می کنیم. به هر حال FPGA اطلاعات مربوط به Configuration رو هر دفعه که برق می یاد توش از PROM ای که به اون وصله می خونه.

البته شما راحت می تونین به من گیر بدین : که خوب اینکار که برای خیلی مدارهای دیگه هم عملیه. بله درسته، مثلا فرض کنید ما یک مودم بسازیم که توش از یک پردازنده DSP برای پردازش استفاده کرده باشیم. خوب این پردازنده هم، برنامه ای رو که اجرا می کنه از PROM ای که بهش وصله می خونه. پس هردفعه که

ارتباط برقرار شد، میشه محتوای این PROM رو Update کرد. کلا این حسن مدارهای Configurable است. که هر زمان خواستیم ، می تونیم ساختار اونها رو به راحتی و با سرعت عوض کنیم.

ما همواره باید سعی کنیم مدارهامون رو طوری طراحی کنیم که حداکثر انعطاف پذیری ممکن رو داشته باشند. و تحت سخت ترین شرایط با بهترین performance کار کنند.

۲. زبانهای توصیف سخت افزار

وقتی می خواهیم یک مدار Logic طراحی کنیم، چه کار می کنیم؟ خوب یک روش مثلا اینه که شکل مدار رو به صورت مجموعه ای از گیت ها و ارتباط آونها باهم بکشیم. شما بارها و بارها توی OrCAD مدارهایی رو برای انجام شبیه سازی کشیدین. بعد از اینکه شما مدار رو می کشید، OrCAD مدار شما رو تبدیل می کنه به یک برنامه، به زبان PSPICE حالا این برنامه که هیچی جز همون مدار شما نیست فرستاده می شه به شبیه ساز و شکل موجها رسم میشه. شاید شما این برنامه رو بعضی وقت ها که OrCAD از مدارتون غلط می گیره دیده باشین. پس وقتی می خواهیم یک مدار Logic طراحی کنیم (یا کلا یک مدار طراحی کنیم، چه آنالوگ و چه دیجیتال) یک راه بهتر اینه که جملاتی بنویسیم که بیانگر شکل مدار ما باشند. اینطوری نقل و انتقال طرح راحت تر می شه و همه می تونن از روی اون جملات روباره مدار ما رو بسازن. خوب فرض کنید مدار رو که مجموعه چندین تا گیت هست به صورت جمله نوشتیم و می خواهیم بدیم این رو برامون IC کنن. ولی یک مشکل بزرگ وجود داره، هرکدوم از کارخانه های ساخت IC برای خودشون یک تکنولوژی خاص دارند. مثلا کارخانه A توی کتابخانه گیت هاش ، فقط گیت NAND داره و XOR و بنابراین اگه ما می خوایم این کارخانه برامون IC تولید کنه باید مدارمون فقط از این دو مدل گیت تشکیل بشه. ولی کارخانه B فقط AND و NOT داره و ما باید مدار رو دوباره برای این کارخانه اصلاح کنیم. حالا فرض کنید دو سال گذشت و اصلا تکنولوژی ساخت IC و کتابخانه گیت ها عوض شده. مدار ما کاملا به درد نخور میشه و باید دوباره از نو طراحی بشه. پس خوب نیست جملات توصیفی ما (که مدارمون رو مشخص می کنند.) خیلی low level باشند. به جای اینکه بیایم و در سطح گیت های منطقی جملاتمون رو بنویسیم ، بهتره یک کم سطح بالاتر کار کنیم. مثلا موقع طراحی یک شمارنده، به جای اینکه جملات توصیفی ما گیت ها و ارتباط اونها باهم رو مشخص کنند بهتره از گیته ها صرف نظر کنیم و بنویسیم: "در هر لبه بالارونده خروجی مساوی با خروجی قبلی بعلاوه 1".

حالا به جای مدار شمارنده می تونیم این عبارت توصیفی را ذخیره کنیم. نرم افزارهایی وجود دارد که کارشان این است که عبارت توصیفی ما را (که توصیف کننده یک مدار منطقی است و به صورت یک برنامه نوشته شده.) می گیرند و به ما آن آرایشی از گیت ها را که معادل عبارت های ما است می دهد. این برنامه ها بسیار هوشمند هستند

و بهترین و پرسرعت ترین مدار منطقی ممکن را به ما می دهند. به این نرم افزارها Synthesizer می گویند. این Synthesizer ها با گذشت زمان به روز می شوند. برنامه ای که ما نوشته ایم ، چون هیچ حرفی در مورد گیت ها داخلش زده نشده همواره قابل استفاده خواهد بود. هر دفعه که خواستیم از آن استفاده کنیم به ابزار سنتز می گوییم تا مدار معادل آن را (که گیتهای لارم و ارتباط آنها باهم است.) به ما بدهد. همچنین به ابزار سنتز می گوییم که این مدار را برای کدام سازنده IC می خواهیم. ابزار سنتز یک کتابخانه کامل و به روز از آن کارخانه دارد و با توجه به آن کتابخانه عمل تبدیل برنامه سطح بالا به گیت ها و عناصر اولیه را انجام می دهد. دقت کنید ، کتابخانه هر کارخانه سازنده IC الزاما مجموعه ای از گیت ها نیست. مثلا برای NEC کتابخانه دارای انواع و اقسام فلیپ فلاپ ، جمع کننده ، Mux ، دی کدر و .. می باشد. برای مثلا IBM کتابخانه دارای گیت Xor ، فلیپ فلاپ ، ضرب کننده و ... می باشد. برای Xilinx کتابخانه شامل عناصر پایه تشکیل دهنده FPGA های تولیدی Xilinx می باشد. برای Altera به نحو مشابه و خلاصه هرکسی برای خودش یک کتابخانه دارد و برنامه شما به راحتی توسط ابزار سنتز تبدیل می شود به آن آرایشی از عناصر پایه که برای پیاده سازی روی IC های آن کارخانه به کار می روند. می بینیم که ابزار سنتز نقش فوق العاده مهمی دارد.

از طرفی برنامه سطح بالایی که ما نوشته ایم عمل شبیه سازی و خطا یابی را برای ما بسیار آسان می کند.به جای آنکه برنامه ما وضعیت ساختاری مدار را بخواهد مشخص کند، صرفا بیان می کند "چه کاری می بایست انجام شود." خوب برنامه خیلی کوچولو تر، ساده تر و قابل فهم تر می شود.

تعداد زبانهای توصیف سخت افزار زیاد است ، در حال حاضر دو تا از همه مشهورتر و پراستفاده تر هستند: VHDL و Verilog .اخیرا یک زبان توصیف سخت افزار جدید دارد راه تکامل را طی می کند تا به این دو تا بپیوندد : SystemC که در آن به C سخت افزار مورد نظرمان را طراحی می کنیم.

۲.۱ . زبان توصیف سخت افزار وی اچ دی ال

VHDL نتیجه همکاری دو شرکت بزرگ IBM و Texas Instruments همراه با یک شرکت کوچک دیگر است که ابتدا برای منظورهای نظامی نوشته شد، تا آنها بتوانند طرح IC های پر سرعت و بزرگ خود را به یک روش مطمئن و انعطاف پذیر بایگانی کنند. VHDL مخفف:

Very high speed integrated circuit Hardware Description Language

است. بعدا استفاده از این زبان برای کارهای تجاری طراحی IC های logic متداول شد. و برنامه های شبیه ساز و سنتز کننده مربوط به آن به بازار عرضه شدند. هم اکنون این نرم افزارها در بازار موجودند.

۲.۲ . زبان توصیف سخت افزار وریلوگ

Verilog زبان توصیف سخت افزاری است که به صورت موازی با VHDL به وجود آمد. شرکت خاصی تولید کننده آن نیست. به هرحال شرکت Cadence در توسعه Verilog نقش بسیار عمده داشته است. یک سازمان به

اسم: Open Verilog International وجود داشته که وظیفه توسعه اولیه ، و تعریف استانداردها را بر عهده داشته. هم اکنون Verilog یکی از استانداردهای IEEE است. (IEEE 1364 Verilog Hardware Description Language Standard) آموزش زبان Verilog و روشهای برنامه نویسی با آن خود نیاز به یک کتاب دارد. Verilog زبان برنامه نویسی بسیار آسانی است. ساختار آن بسیار شبیه به C است با این تفاوت که اینجا کلمات begin و end به جای آن آکولادها قرار گرفته اند. فراگیری این زبان با اسفاده از کتاب خوبی چون Verilog Quick Start برای یک شخص معمولی باید طی زمانی کمتر از یک ماه ممکن باشد. به هرحال توجه کنید که این یک زبان توصیف سخت افزار است و ساختار آن به همین دلیل با زبانهای برنامه نویسی معمولی مثل C اندکی فرق می کند. اینجا ما طی چند مثال، که روند آسان به سخت دارد، این زبان را بررسی می کنیم:

```
module full_adder (S, C, A, B, Cin);
output    S, C;
input     A, B, Cin;

assign S = A ^ B ^ Cin;
assign C = (A & B ) | (A & Cin) | (B & Cin);

endmodule
```

نکته مهم اول اینکه زبان Verilog به بزرگ و کوچک بودن حرفها حساس است. برنامه کوچک بالا یک Full Adder است. در Verilog همه چیز به صورت مدول تعریف می شود. یعنی اینکه تمام مدارهای منطقی به صورت یک جعبه در نظر گرفته می شوند که چند ورودی و چند خروجی دارد. این جعبه یک اسم دارد، در برنامه بالا اسم این جعبه یا module را full_adder گذاشته ایم و سپس تعریف کرده ایم که این مدول چه ورودیها و چه خروجی هایی (چه درگاهها یا پورت هایی) دارد. در مثال بالا S و C خروجیهایی مدول و A، B و Cin ورودیهای مدول می باشند. تمام این پورت ها تک بیتی هستند، یعنی عرض آنها یک بیت است. یک FPGA تعداد زیادی پایه به اسم User I/O دارد، وقتی این مدار را ببریم روی FPGA، هرکدام از پورتهای تک بیتی در مدول فوق ، یکی از این پایه ها را اشغال خواهند کرد. بقیه پایه ها آزاد خواهند ماند.

پس از تعیین نام ورودی و خروجی ها و جهت هرکدام از آنها (که با کلمات input و output در سطرهای دوم و سوم انجام شده.) می رسیم به خود مدار Logic ای که ارتباط بین ورودی و خروجی را به وجود می آورد: در دستور assign اول گفته شده S = A^B^Cin یعنی خروجیِ S برابر است با xor شده A، B و Cin. در دستور assign دوم مقدار C که در واقع رقم نقلی خروجی است تعیین شده. علامت & بیانگر عمل and و علامت | بیانگر عمل or می باشد. نهایتا با دستور endmodule اعلام می کنیم که توصیف مدار داخل مدول به طور کامل انجام شده و دیگر چیزی برای نوشتن نداریم. اگر مدار منطقی یکFull adder را در نظر بیاوریم می بینیم که عبارت های بالا دقیقا معادل همان مدار هستند. به جای کشیدن شکل گیت ها می توان عبارات بالا را نوشت. برتری زبان

توصیف سخت افزار بر روش کشیدن schematic هنگامی معلوم می شود که بدانیم برنامه فوق را به صورت زیر هم می توان نوشت:

```
module full_adder (S, C, A, B, Cin);
output     S, C;
input      A, B, Cin;

assign {C, S} = A + B + Cin;

endmodule
```

در این برنامه مستقیما گفته شده که ترکیب {C,S} به عنوان یک عدد دو بیتی برابر است حاصل جمع A، B و Cin. این برنامه وقتی به نرمافزار Synthesizer داده شود، تبدیل به گیت های مناسب که عمل جمع را انجام می دهند را خود ابزار سنتز به صورت بهینه انجام می دهد. در برنامه بالا تعریف پورت ها کاملا مثل قبل است. ترکیب {C, S} در واقع یک عدد دو بیتی است. این عدد مساوی قرار گرفته با حاصلجمع تمام ورودی ها. در عدد دو بیتی {C, S} متغیر C بیت پرارزش و S کم ارزش است. این دقیقا همان اتفاقی است که در یک full adder می افتد.

پس یک مدار جمع کننده را به دو روش می توان مدل کرد: Gate Level Modeling که در آن خود گیت هایی که مدار را می سازند مستقیما بیان می کنیم و دوم Behavioral Modeling که در آن با عبارت هایی عملکرد مدار را توصیف می کنیم ولی در مورد اینکه چه گیت هایی لازم است تا این عملکرد ایجاد شود حرفی نمی زنیم و آن را به عهده Synthesizer می گذاریم. طبیعتا Verilog به شما اجازه می دهد مدار ساختمان مدار خود را (که مثلا از چندین مدول تشکایل شده و تمام این مدول ها زیر یک مدول اصلی هستند.) به راحتی بیان کنید.

مثال زیر یک نمونه از این خاصیت Structural Modeling موجود در Verilog را نمایش میدهد:

```
module two_bit_adder (S, A, B);
output [2:0]  S;
input  [1:0]  A, B;

wire C_b;

full_adder I0 ( S[0], C_b, A[0], B[0], 1'b0 );
full_adder I1 ( S[1], S[2], A[1], B[1], C_b );

endmodule
```

در مدار فوق در داخل مدول two_bit_adder دو بار یک مدول دیگر (full_adder که در بالا برنامه اش نوشته شده.) استفاده شده است.

Full adder اول بیت های اول از پورت های دو بیتی A و B را دریافت می کند. (A[0] و B[0]) حاصل جمع را توی S[0] که بیت اول پورت خروجی S است می ریزد و بیت انتقالی موجود را به وسیله سیمی به نام C_b به Full adder دوم انتقال می دهد. می بینیم که ورودی پورت C برای Full adder دوم سیم C_b است. از طرفی دو ورودی دیگر A[1] و B[1] هستند که بیت های بالایی پورتهای ورودی A و B می باشند. حاصل جمع در Full adder دوم به S[1] و بیت نقلی نهایی در S[2] ریخته می شود. در اینجا مدار را به صورت Structural مدل کردیم.

تا بحال با مدارهای Sequential کاری نداشتیم. عنصر ذخیره کننده در مدارهایمان وجود نداشت و فقط داده ها از گیت ها عبور می کردند و اعمال محاسباتی روی آنها انجام می شد. حالا میریم ببینیم مدارهایی که فلیپ فلاپ و در نتیجه پالس ساعت دارند، چه جوری پیاده می شوند:

```
module d_flip_flop (Q, D, clk);
output Q;
input  D;
input  clk;

reg Q;

always @(posedge clk)
     Q <= D;

endmodule
```

این برنامه یک فلیپ فلاپ D ساده را نمایش می دهد. دقت می کنیم که خروجی Q علاوه بر فرم output ، به صورت reg هم تعریف شده است. دو خط اصلی برنامه با یک دستور always شروع می شود. این دو خط می گوید: همواره ، در هر لبه بالا رونده مقدار D را بریز توی Q. ابزار سنتز وقتی این را می بیند می فهمد که باید یک فلیپ فلاپ را پیاده کند و می فهمد که سیگنالی که اسم آن clk است باید به عنوان پالس ساعت در این فلیپ فلاپ استفاده شود. حالا فرض کنید می خواستیم اولا، این کار در لبه های پایین رونده انجام شود، ثانیا مقدار not شده D در Q ریخته شود، کد Verilog به این صورت می شد:

```
module d_flip_flop (Q, D, clk);
output Q;
input  D;
input  clk;

reg Q;

always @(negedge clk)
     Q <= ~D;
```

```
endmodule
```

به تغییراتی که نسبت به برنامه بالایی به وجود آمد دقت کنید: اولا به جای posedge از negedge استفاده می کنیم. ثانیا یک علامت not یعنی "~" آمده جلوی D. به این مفهوم که Q مقدار not شده D را دریافت خواهد کرد. حالا فرض کنید می خواستیم یک ورودی reset هم برای فلیپ فلاپ بگذاریم که در هر لبه بالا رونده ای که مقدار آن 1 شد، خروجی فلیپ فلاپ برابر با صفر شود:

```
module d_flip_flop (Q, D, clk);
output Q;
input  D;
input  clk;

reg Q;

always @(posedge clk)
     if ( reset )
          Q <= 0;
     else
          Q <= D;
endmodule
```

حالا فرض کنید بخواهیم این reset، Asynchronous باشد، یعنی اینکه نیازی به لبه بالا رونده ساعت برای صفر کردن خروجی نداشته باشد و هر زمان خودش یک شد، خروجی را صفر کند، بلوک اصلی برنامه اینطوری می شود: (بقیه برنامه همانطور که بوده می ماند.)

```
always @(posedge clk  or posedge reset)
     if ( reset )
          Q <= 0;
     else
          Q <= D;
```

همواره در هر لبه بالا رونده ساعت و یا هر لبه بالا رونده reset هر کدام که روی داد، کد داخل بلوک always اجرا می شود: اگر reset بالا باشد در خروجی صفر و در غیر این صورت مقدار D روی Q می رود.

حالا یک مثال مفصل تر را بررسی می کنیم:

```
module four_bit_mux (mux_out, mux_in, mux_control, clk);
output      mux_out;
input [3:0] mux_in;
input [1:0] mux_control;
input       clk;
reg         mux_out;
```

```
wire mux_out_w;
assign mux_out_w=(mux_control == 2'b00) ? mux_in[0]
                :(mux_control == 2'b01) ? mux_in[1]
                :(mux_control == 2'b10) ? mux_in[2]
                :mux_in[3];

always @(posedge clk)
     mux_out <= mux_out_w;

endmodule
```

در برنامه بالا یک Multiplexer که 4 ورودی و یک خروجی و یک ورودی کنترل دارد را نشان می دهد. خروجی تک بیتی است ولی ورودی ها به صورت بردار (Bus) انتقال داده تعریف شده اند و هر کدام بیشتر از یک بیت عرض دارند. یک سیم به اسم mux_out_w تعریف شده که خروجی مدار mux است. خود مدار mux با استفاده از یک دستور assign که مقدار دهی را به طور شرطی انجام می دهد، پیاده شده است. این دستور 4 سطر برنامه را تشکیل می دهد. شرطها، در پرانتزهایی هستند که قبل از علامت ? قرار دارند. علامت : به معنی else و علامت ? معادل با if است. این روش شرطی کردن، دقیقا مطابق با آن چیزی است که در زبان برنامه نویسی C وجود دارد. در نهایت با استفاده از یک بلوک always خروجی مدار mux با لبه های بالارونده ساعت روی پورت mux_out قرار می گیرد. در واقع آن بخشی از مدار که با استفاده از دستور assign پیاده می شود، معادل با یک مدار combinational است و آن بخشی که در آن به register ها مقدار داده می شود، معادل با یک مدار sequential است. باید دقت نمود که برنامه نویسی Verilog با بقیه زبانهای برنامه نویسی تفاوت های محسوس دارد. در Verilog امکان دارد تمام خط های یک برنامه باهم و به صورت موازی در حال اجرا شدن (پیاده شدن.) باشند. برنامه بالا را با دستور case هم می توان پیاده کرد:

```
module four_bit_mux (mux_out, mux_in, mux_control, clk);
output      mux_out;
input [3:0] mux_in;
input [1:0] mux_control;
input       clk;
reg         mux_out;
always @(posedge clk)
     case ( mux_control )
     2'b00 : mux_out <= mux_in[0];
     2'b01 : mux_out <= mux_in[1];
     2'b10 : mux_out <= mux_in[2];
     2'b11 : mux_out <= mux_in[3];
     default : mux_out <= mux_out;
```

```
     endcase
endmodule
```

برنامه بالا با استفاده از دستور case ، در هر لبه بالا رونده وضعیت ورودی mux_control را چک می کند و برای هر مقدار از mux_control ورودی مناسب را روی خروجی می گذارد. اگر هیچ کدام از چهار مقدار ذکر شده در دستور case در ورودی mux_control نباشد (مثلا mux_control برابر با 2'bzz امپدانس بالا باشد.) آنوقت کد نوشته شده در بخش default اجرا خواهد شد. یعنی اینکه mux_out مقدار قبلی خود را حفظ خواهدکرد و مقدار آن تغییر نخواهد کرد.

یک مثال دیگر:

```
module ram_interface (address_out, data, rnw, reset,
clk);
output [15:0] address_out;
inout  [15:0] data;
output        rnw;
input         reset, clk;

reg    [15:0] address_out;
reg           rnw;
reg    [15:0] access_address;
reg           read_write_turn;
reg    [15:0] data_in_register;

assign data = (! rnw ) ? 0 : 16'bz;

always @(posedge clk)
     if ( reset ) begin
          address_out <= 0;
          rnw <= 1;
     end
     else begin
          if ( rnw ) begin
               data_in_register <= data;
               access_address <= access_address + 1;
          end
          if ( data_in_register == 16'hff0f )
               rnw <= ~ rnw;
          if (! rnw )
               rnw <= ~ rnw;
```

```
    end
endmodule
```

کار این برنامه این است که محتوای خانه های یک ram را از آدرس صفر به ترتیب می خواند، و چک می کند که آیا مقدار آن خانه برابر با عدد 0xff0f هست یا نه. اگر هست مقدار آن خانه از ram را صفر می کند و ادامه می دهد، اگر نیست ادامه می دهد. با هر reset کار دوباره از آدرس صفر شروع می شود، وقتی که رسیدیم به آخر حافظه دوباره بر می گردیم بالا. اینجا فرض شده عرض گذرگاه داده برای حافظه 16 بیت است و حافظه دارای 64K خانه است. (برای همین گذرگاه داده هم 16 بیتی انتخاب شده.) این برنامه را می شود روی یک FPGA یاده کرد، و FPGA را به ram وصل کرد تا عمل بالا را برای ما روی ram انجام دهد.

نکته جدیدی که در برنامه فوق وجود دارد اضافه شدن یک نوع پورت جدید است: inout پورتی است که از طریق آن داده هم می تواند داخل شود و هم به خارج برود. دقت کنید که چگونه مدیریت داده مربوط به این پورت انجام شده است. هر زمان rnw (read not write) برابر با 1 باشد، یعنی قرار باشد از ram بخوانیم، داده روی پورت data که ورودی/خروجی است ، ریخته می شود داخل یک ثبات به اسم: data_in_register. از طرفی در بالای برنامه، توسط یک دستور assign به پورت data هنگامیکه قرار است از این پورت داده به بیرون برود، مقدار اختصاص داده شده. (این مقدار برابر صفر است.) دقت کنید که هنگامیکه data در حالت خواندن است دستور assign مقدار 16'bz را که در واقع امپدانس بالا و مدار قطع است روی data قرار می دهد. به عبارت دیگر در این حالت پورت data را drive نمی کند و هر کس دیگری می تواند روی آن داده قرار دهد. در این هنگام ram داده خود را روی پورت data قرار می دهد، و FPGA می تواند آن داده ها را بخواند. نکته دیگری که باید به آن توجه کرد، وجود دستورات begin و end است که بلوک های کد، که در هنگام درست بودن یک شرط باید اجرا شوند را مشخص می کند. برای اینکه درست تر درک کنید چه اتفاقی افتاد لطفا در فصل مربوط به ساختار FPGA شکل مدار بلوک IOB را به دقت وارسی کنید. ارتباط های این بلوک را با مدار داخل FPGA پیدا کنید. حالا باید بتوانید آن سیمی که برای انتقال داده از PAD (پایه FPGA) به بیرون به کار میرود (سیم ورود داده، یعنی همان که داده های ram وقتی بافرهای drive کننده خروجی را خاموش می کنیم، می آیند و روی آن سیم قرار می گیرند.) را از آن سیمی که برای انتقال داده از FPGA به PAD (یعنی محیط بیرون) به کار می رود، تشخیص دهید. برای این کار می توانید از نرم افزار FPGA Editor که در ISE وجود دارد استفاده کنید. سیم های مربوط به خروج داده از سمت چپ صفحه وارد می شوند و نهایتا به PAD ختم می شوند. سیم های مربوط به ورود داده از PAD شروع شده از سمت راست صفحح خارج می شوند. (در واقع از اینجا می رند داخل FPGA.)

هنوز هیچ نظر قطعی مبنی بر اینکه کدامیک از زبانهای توصیف سخت افزار Verilog یا VHDL کاملتر و بهتر هستند وجود ندارد. هر دو بسیار انعطاف پذیرند و امکانات زیادی را در اختیار طراح قرار می دهند. برای هردو نرم افزارهای فراوانی وجود دارد. Library های مختلفی برای هر دو توسع یافته است. VHDL شبیه به پاسکال و Verilog بسیار شبیه به C است. اخیرا (در استاندارد جدید 2002 مربوط به Verilog) تغییرات جدیدی اعمال شده که آنرا بسیار قابل انعطاف تر می کند. در حال حاضر تمام نرم افزارهای شبیه ساز و تقریبا تمام ابزارهای سنتز

از استاندارد های جدید حمایت می کنند. نویسنده اکیدا توصیه می کند ، که اگر شما در ابتدای راه هستید و می خواهید بین VHDL و Verilog یکی را انتخاب کنید، ابتدا چند برنامه مثالی که به هردو زبان نوشته شده را با دقت مطالعه کنید تا بهتر بودن یکی از آنها و زیباتر بودنش بر شما ثابت شود.

۳. ساختارهای بکاررفته

اولین سوالی که پیش می آید این است که ساختار داخلی FPGA چه جوریه که می تونه به این سهولت به هر مدار منطقی دلخواهی تبدیل بشه. ایده اصلی اینه که:

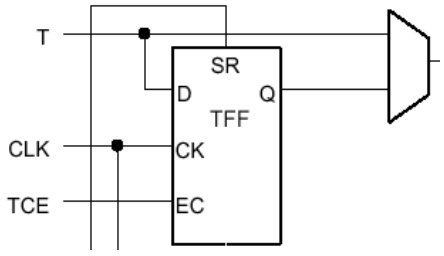
می توان هر مدار منطقی ترکیبی دلخواه را که مثلا 4 ورودی و 1 خروجی دارد، صرف نظر از اینکه چه گیت هایی در آن به کار رفته ، با یک ROM که 4 بیت ورودی آدرس و یک بیت خروجی داده دارد، جایگزین کرد. پس ایده اصلی این است که ببینیم چه مدار منطقی را می خواهیم پیاده کنیم. سپس تعیین می کنیم که به ازای هر مجموعه ورودی خاص، خروجی چیست. حال با استفاده از مجموعه ای از بلوک های Ram کوچک همان مدار را می سازیم. به این ترتیب که مقدار هرکدام از خانه های Ram ها را طوری تعیین می کنیم که وقتی قرار شد مقدار آن خانه خروجی مدار باشد، دقیقا خروجی همان چیزی باشد که مدار واقعی می داد. پس چند Ram داریم که به طور خاصی به هم وصل شده اند. ورودی مدار به آدرسهای این Ram ها وصل شده است ، بر حسب اینکه ورودی مدار چه باشد، یکی از خانه های یکی از Ram ها در خروجی ظاهر می شود. از قبل مقدار تابع منطقی ای (را که می خواستیم با این مدار آن را بسازیم.) به ازای این ورودی خاص در این Ram قرار داده ایم. دو نکته مهم اینجا به نظر می رسد:

۱. در FPGA ما از مجموعه ای از Ram های ایستا برای پیاده سازی مدار منطقی استفاده می کنیم. داده هر Ram همانطور که می دانیم فرار است، یعنی با رفتن برق از مدار از بین می رود، لذا هر دفعه که قرار است FPGA وارد مدار شود ، (هر زمان که برق از نو به مدار وصل می شود.) لازم است مقدار این Ram ها دوباره تعیین شود یا به عبارت دیگر FPGA از نو Configure شود. اطلاعات مربوط به Configure شدن FPGA را معمولا روی یک PROM کپی کرده ، PROM را به FPGA وصل مینمایند. هر دفعه که برق از نو وارد مدار می شود، FPGA به طور خودکار سراغ PROM میرود و اطلاعات مربوط به مداری را که باید پیاده شود از آن می خواند.

۲. هر زمان که بخواهیم می توانیم FPGA را Reconfigure کنیم. به این مفهوم که می توان مداری را که FPGA پیاده می کند عوض کرد. مثلا تا الان FPGA داشته به صورت یک ضرب کننده عمل می کرده و می خواهیم از این زمان به بعد به صورت جمع کننده عمل کند، کافیست FPGA را reset کنیم و اطلاعات مربوط به Configuration جدید را به آن بدهیم.

سوالی که پیش می آید این است که: آیا FPGA می تواند مدارهای Sequential را نیز حمایت کند، یا اینکه فقط مخصوص مدارهای Combinational است؟ جواب آن است که هر نوع مدار Sequential را نیز می توان با FPGA پیاده کرد. تمام FPGA ها امکانات ویژه برای پیاده سازی مدارهای ترتیبی دارند. همانطور که می دانیم بخش اعظم مدارهایی که در مکانهای مختلف استفاده می شود، عملا ترتیبی هستند و Flip Flop عنصر بسیار مهمی است که در تمام آنها وجود دارد. اکثر FPGA ها ورودی خاص برای پالس ساعت دارند. ولی ساختار داخلی چگونه باید باشد تا هم بتوان مداری ترکیبی داشت و هم مداری ترتیبی؟ در شکل (1) این مطلب نشان داده شده است.

در این شکل ورودی اصلی با T نام گذاری شده است. خروجی مدار در واقع خروجی Mux است. Mux دو ورودی دارد: T مستقیما یکی از ورودی های آن است. ورودی دیگر همان T است که از یک Flip Flop عبور میکند. حال بسته به اینکه مقدار ورودی کنترل Mux چه باشد...



شکل (1)

مدار، ترتیبی (وقتی خروجی Mux همان Q باشد.) و یا ترکیبی (وقتی T مستقیما به خروجی بیاید.) خواهد بود. مقدار ورودی کنترل Mux هم یکی از مقادیری است که موقع Configuration تعیین می شود. به این ترتیب هر نوع مدار منطقی سنکرونی را می توان با FPGA به راحتی پیاده کرد. مجموعه زیادی از آن Ram های کوچک (که آنها را Look up Table یا LUT می نامیم.) به همراه ساختارهایی مشابه شکل فوق یک FPGA را می سازند. حال به طور دقیق به بررسی هرکدام از قسمت های FPGA می پردازیم.

۳.۱. بلوک ورودی خروجی

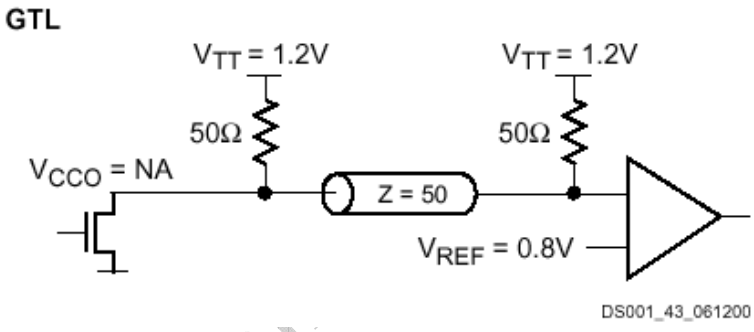
عکس یک بلوک ورودی/خروجی (Input/Output block) در صفحه بعد نشان داده شده است. (تصویر دارای کیفیت خوبی است. لطفا برای دیدن تمام اجزای آن از Zoom استفاده کنید.) IOB در واقع آن بخش از FPGA است که رابط بین پایه های FPGA و محیط خارج است. هر کدام از پایه های FPGA که مربوط به استفاده کننده باشد، (مثلا VCC و یا GND نباشد، بلکه در اختیار استفاده کننده باشد که از آن به عنوان خروجی/ورودی داده استفاده کند.) به یک IOB ربط دارد. سپس IOB به مدارهای داخل FPGA وصل می شود. در حالت ساده یک IOB دارای سه فلیپ فلاپ است. یکی برای نگهداری داده ای که از بخش داخلی FPGA می آید و قرار است بیرون برود استفاده می شود. (FF سمت چپ پایین) یکی برای کنترل بافری که مقدار ولتاژ معادل صفر یا یک را در خروجی قرار می دهد، استفاده می شود (سمت چپ بالا). به این ترتیب که اگر قرار باشد از این پایه FPGA برای ورود داده استفاده شود آنوقت Output Buffer غیر فعال می شود تا داده بتواند به FPGA وارد شود. در نهایت FF سوم (سمت راست پایین) برای نگهداری داده ای که از بیرون می آید و قرار است به مدارات داخلی FPGA وارد شود به کار می رود. هر سه FF دارای پالس ساعت یکسان و ورودی Set/Reset یکسان هستند. دقت می کنیم که می توان از هیچ کدام FF استفاده نکرد و در واقع یک حالت Combinational را برای ورودی و خروجی ها داشت.

شکل (2) مربوط است به Spartan-II که یکی از محصولات شرکت Xilinx است. از آنجا که FPGA قرار است بسیار انعطاف پذیر باشد، باید بتواند با انواع مختلفی از Signaling Standards ارتباط برقرار کند. مثلا در استاندارد LVTTL (Low Voltage TTL) یک منطقی برابر با 3.3 ولت و صفر منطقی برابر با 0 ولت است. در حالی که برای LVCMOS2 این مقادیر برابر با 2 ولت و 0 ولت است. پس ساختار داخلی باید به گونه ای باشد که بتوان با هر دو نوع استاندارد ارتباط برقرار نمود.

در شکل (2) اگر به عنصر Programmable output buffer دقت کنیم می بینیم که سه ورودی دارد. یکی از آنها با نام OE کنترل می کند که کی این عنصر فعال باشد و کی نباشد. ورودی VCC مقدار ولتاژی است که برای یک منطقی باید در خروجی قرار داده شود. همانطور که می بینیم این ورودی به یکی از پایه های FPGA وصل شده. استفاده کننده بر حسب اینکه از کدام استاندارد سیگنالینگ می خواهد استفاده کند باید ولتاژ مناسب را روی این پایه FPGA یعنی VCCO قرار دهد. مثلا برای LVTTL باید 3.3 ولت و برای

LVCMOS2 باید 2 ولت قرار گیرد. حالت مشابه برای عنصر Programmable Input Buffer برقرار است. بر حسب اینکه از چه استانداردی برای

سیگنالینگ استفاده می شود لازم است ولتاژ VREF به درستی تعمین گردد. مثلا VREF برای استاندارد GTL+ برابر با 1 ولت در حالی که برای AGP-2X برابر با 1.3 ولت است. نهایتا می دانیم که تمام خطوط انتقال که داده را با سرعت زیاد منتقل می کنند، خود در واقع دارای مقاومت، خازن و سلف هستند. وقتی سرعت نوسان بالا رود یعنی فرکانس زیاد شود، این عناصر اثر خود را آشکار می کنند. می دانیم که برای از بین بردن این اثر لازم است خط انتقال را به نحو مناسبی Terminate کنیم. به عنوان یک مثال بسیار متداول می دانیم که وقتی امپدانس موجود در انتهای خط برابر با امپدانس توزیع شده خط باشد، موجی که به انتهای خط می رسد بازتاب نخواهد داشت و حداکثر میزان انرژی انتقال پیدا خواهد کرد. در طراحی بوردها لازم است این ملاحظات در نظر گرفته شوند. شکل زیر یک بخشی از مداری که باید پیاده شود را نشان میدهد:



شکل (3)

شکل (3) مقاومت ها و ولتاژهایی را که لازم است برای خط انتقال به کار برده شود، (وقتی از استاندارد GTL برای سیگنالینگ استفاده می کنیم.) نشان می دهد. جدول (1) استانداردهای متداول همراه با مقادیر ولتاژهای مربوط به هر استاندارد را نشان می دهد.

نکته نهایی در مورد IOB آنکه وقتی خواستیم برای ارتباط با محیط بیرون از یک استاندارد ولتاژ خاص استفاده کنیم، لازم است اعمال خاصی را در روند برنامه نویسی، و ... انجام دهیم. اگر این کارها به درستی انجام شود، بخشی از پایه های FPGA به آن مود خاص سیگنالینگ می روند. دقت می کنیم که نمی توان هر پایه ای را به هر مود دلخواه برد، بلکه پایه ها دسته دسته به مود های خاص سیگنالینگ میروند. به هرکدام از مجموعه پایه ها که یک مود دارند یک IO Bank گفته می شود. بعضی استانداردها با هم تطابق دارند، طوری که از یک IO Bank برای چند استاندارد می توان استفاده نمود. در Datasheet تمام FPGA ها ، جدول هایی وجود دارد که برای هر نوع سیگنالینگ خاص مقدار VCCO و VRef را تعیین می کند. همچنین بانک های IO مربوط به آن FPGA و محدوده هر کدام را تعیین می کند. VCCO و VREF هر بانک ورودی خروجی از بقیه مجزاست. برای دیدن این جداول لطفا به Datasheet مربوط به Virtex-II و یا Spartan-II مراجعه کنید.

۳.۲. بلوک منطقی قابل برنامه ریزی

Configurable Logic Block در واقع اصلی ترین بخش از هر FPGA است که توابع منطقی را پیاده می کند. هر CLB شامل دو Slice است و هر Slice شامل دو Logic Cell (LC) . پس کوچک ترین عنصر مجزا LC است. هر LC دارای یک فلیپ فلاپ، ویک Function Generator یا به عبارت دیگر Look up table است. (این همان عنصری است که در بخش معرفی FPGA از آن صحبت شد.) علاوه بر این در هر LC مدارهای کوچک دیگری هم برای انجام کارهای خاص وجود دارد. معمولا LC را به گونه ای می سازند که FPGA بتواند اعمال ریاضی بسیار مهم مانند ضرب دو عدد در هم، یا جمع دو عدد باهم را به سرعت انجام دهد. پس در LC علاوه بر LUT که به کمک آن هر تابع منطقی با یک خروجی و 4 ورودی قابل پیاده سازی است، عناصر دیگری چون گیت های And و Xor مجزا نیر وجود دارد. مجموعه دو LC وقتی باهم به کار روند می توانند هر تابع منطقی با 5 ورودی و یک خروجی را پیاده کنند. مجموعه دو Slice یعنی چهار LUT وقتی باهم به کار روند می توانند هر تابع منطقی 6 متغیره را پیاده سازی کنند. برای تمام این اعمال در CLB مدارهای خاص وجو دارد. شکل (4) ساختار ساده شده یک Slice را نمایش می دهد.

در هر Slice مدارهای خاص برای انتقال سیگنال Carry وجود دارد. می دانیم که در انجام اعمال ریاضی چون مقایسه ، جمع، و ... بیت نقلی بسیار مهم است و یکی از عواملی است که به راحتی می تواند سرعت عملکرد مدار را کمکند. تاخیر مدارهای مربوط به انتقال Carry بسیار اندک و در حد نانو ثانیه است. به این ترتیب FPGA می تواند اعمال ریاضی چون جمع و ضرب را با سرعت مناسب انجام دهد. LUT عنصری است بسیار تنعطاف پذیر. از آن می توان به عنوان یک Ram که 16 بیت ظرفیت دارد استفاده کرد. یا می تواند یک Rom باشد، یا یک Shift Register و یا یک Function Generator. گاهی اوقات برای آنکه از LUT بتوان به عنوان Ram و یا Shift Register استفاده کرد لازم است از یک روش خاص طراحی استفاده شود. در این روش طراح خودش مدار Shift Register و یا Ram را طراحی نمی کند بلکه از طرح های پیش ساخته ای که مثلا Xilinx در اختیار او قرار داده (و بطور بهینه برای هر FPGA طراحی شده.) استفاده می کند. طراح با استفاده از نرم افزار Core Generator می تواند این طرحها را به دست آورد. این طرحهای از پیش آماده شده، مثلا یک شیفت رجیستری به ما می دهند که با LUT ها و به صورت بهینه ساخته شده است. پس ما دیگر برای این بخش مدار برنامه نمی نویسیم و از برنامه های از پیش نوشته شده استفاده می کنیم. بعضی از ابزارهای سنتز مثل Synplify آنقدر خوب و قوی هستند که حتی اگر خودمان هم کد مربوط به مثلا شیفت رجیستر بزرگی را به Verilog بنویسیم، خودشان آن را به مدار بهینه که قابل پیاده سازی رویFPGA باشد تبدیل می کنند دقت کنید در حالت طبیعی که چنین ابزار سنتز قوی در دسترس شما نیست، نباید خودتان برنامه مثلا رم را بنویسید، بلکه باید از core ها از

پیش نوشته شده و آماده استفاده کنید. شکل صفحه بعد یک Slice از یک CLB را نمایش می دهد. لطفا از Zoom برای دیدن جزئیات استفاده کنید. در شکل گیت های dedicated برای اعمال جمع و ضرب (گیت های xor و and) دیده می شوند.

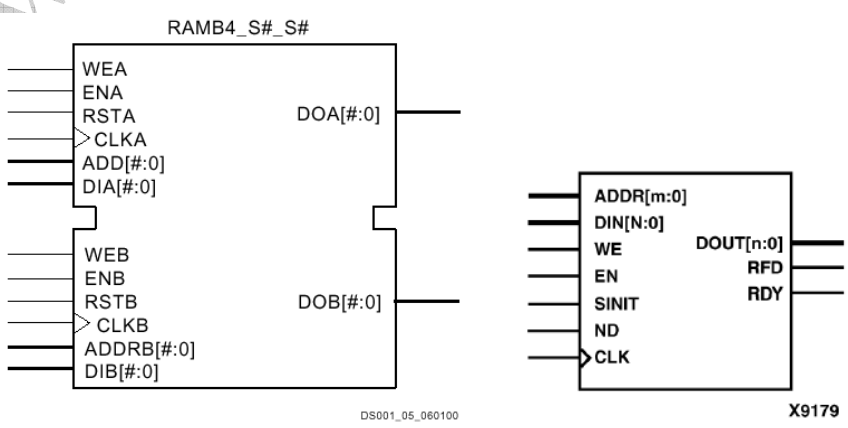
۳.۳ بلوک های حافظه

فرض کنید یک تابع منطقی با 20 متغیر را بخواهید روی FPGA پیاده کنید. در این صورت لازم است تعداد زیادی CLB مصرف شود. در حالیکه اگر Ram های بزرگی روی FPGA موجود باشد می توان از آنها استفاده کرد. تمام FPGA های جدید دارای بلوک های حافظه بسیار انعطاف پذیر روی خود هستند. در اکثر موارد طراح نیاز دارد داده های مربوط به مدار را در جایی ذخیره کند، تا مثلا در نهایت بتواند الگوریتمی را روی آنها انجام دهد. از طرفی قرار دادن Ram های خارجی که در بیرون FPGA قرار دارند و داده های را ذخیره می کنند، همیشه به صرفه نیست. چراکه اولا سرعت عملکرد آنها کند است. ثانیا طراحی بورد را مشکل می کنند، ثالثا بیخودی تعدادی از پایه های FPGA را مجبوریم صرف آنها کنیم. به عنوان مثال Spartan-II که یکی از FPGA های Xilinx است دارای بلوک های حافظه، هرکدام با ظرفیت 4096 بیت است. طراح قادر است با این 4096 بیت هر کدام از Ram های 4096×1 یا 2048×2 یا 1024×4 یا ... را بسازد. جدول (3) حالت های ممکن را نشان می دهد: (به صورت تعداد بیت های ممکن برای عرض گذرگاهای داده و آدرس)

Width	Depth	ADDR Bus	Data Bus
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

جدول (۲)

از طرفی می توان هر بلوک Ram را به صورت Single port و یا Dual port پیکربندی کرد. در شکل (5) بلوک دیاگرام هر نوع Ram نشان داده شده است:



شکل (۵)

پورت های اصلی یک Single port ram چنین است:

ADDR[m:0] : که ورودی آدرس حافظه است و تعیین می کند خواندن و یا نوشتن از و به کجا باید انجام شود.

DIN[N:0] : که داده ورودی به حافظه است و قرار است در آن ذخیره شود.

DOUT[n:0] : داده خروجی که حاصل عمل خواندن از حافظه روی آن ریخته می شود.

WE : هنگامیکه بالا باشد نشان می دهد که باید داده ای داخل حافظه نوشته شود.

CLK : پالس ساعت ورودی به حافظه است. کلیه اعمال با پالس ساعت سنکرون و انجام می شود.

هنگامیکه یک BlockRAM به صورت Dual port قرار است استفاده شود ، دو تا پورت ورود و خروج داده داریم، و دو تا پالس ورودی ساعت. هر دو پورت به کل داده های موجود در این BlockRAM دسترسی دارند. مثلا در یک لبه بالا رونده ساعت هردو می توانند از محل های مختلف حافظه بخوانند و یا به محل های مختلف بنویسند. (البته هر دو به یک محل به صورت همزمان نمی توانند بنویسند.) چنین ساختاری برای پیاده سازی یک FIFO آسنکرون بسیار ایده آل است. فرض کنید قرار است یک فرستنده بسیار سریع به یک گیرنده کند وصل شود. فرستنده به صورت Burst داده ها را انتقال می دهد. به این ترتیب که در یک بازه زمانی بسیار کوچک حجم زیادی از داده را با سرعت زیادی به گیرنده می دهد. (مثلا فرض کنید فرستنده یک Server بسیار سریع است که به نوبت به هرکدام از Client ها که تعدادشان زیاد است ولی کند کار می کنند، سرویس می دهد.) بنابراین یک میانگیر بین فرستنده و گیرنده نیاز است، پالس ساعت گیرنده فرکانس اندکی دارد ولی دائمی است. در عوض فرستنده پالس ساعتی با فرکانس بالا دارد ولی به صورت لحظه ای اعمال می شود. اینجا یک FIFO آسنکرون بسیار مناسب است.

استفاده بسیار متداول دیگری که از dual port ram می شود به عنوان Width Converter است. فرض کنید عرض درگاه داده A را 16 بیت و عرض درگاه داده B را 32 بیت تعریف می کنیم. به این ترتیب می توان مجموعه داده هایی که به صورت بسته های 16 بیتی وارد می شوند را در بسته های 32 بیتی برداشت و یا برعکس.

مثلا ممکن است نیاز داشته باشیم داده هایی که به صورت یک باس ۳۲ بیتی و با یک پالس ساعت 125 مگاهرتزی وارد FPGA می شوند را بفرستیم به مداری که اولا با 133 مگاهرتز کار می کند و ثانیا عرض باس داده آن ۱۲۸ بیت است. در این حالت به یک Width Converter Asynchronous FIFO نیاز خواهیم داشت. طراحی یک همچین مداری کار آسانی نیست. به خصوص که پالس ساعت مربوط به نوشتن و پالس ساعت مربوط به خواندن داده با هم هیچ ارتباطی ندارند. به هرحال این موضوع که چنین مداری چگونه ایجاد می شود و اینکه بهینه ترین مدار ممکن برای پیاده سازی به عنوان کنترل کننده FIFO چیست، باعث نوشته شدن مقاله های زیادی شده است. هم در FPGA و هم در ASIC روشهای خاص و مشخصی برای طراحی چنین مدارهایی وجود دارد. کلا اینجاست که بحث Multi Clock Logic Circuits و روشهای طراحی آنان پیش کشیده می شود.

با BlockRAM می توان مدارهای متداول زیر را پیاده کرد: FIFO، Stack ، Linked list، ... همچنین برای پیاده سازی ماشین های حالت با سرعت های زیاد هم می توان از آن استفاده کرد. بعضی از ابزارهای سنتز مثل Synplify آنقدر باهوش هستند که وقتی ببینند در برنامه Verilog ما تکه ای وجود دارد که برای پیاده سازی آن استفاده از BlockRAM خیلی مناسب است، خودشان به صورت خودکار از این بلوک ها استفاده می کنند. ولی در حالت معمولی ما باید بطور مشخص در برنامه ذکر کنیم ، که می خواهیم برای پیاده سازی این قسمت، (مثلا این حافظه) از BlockRAM استفاده کنیم. نرم افزار Xilinx Core Generator به ما این امکان را می دهد که از Ram Block در طرحهایمان استفاده کنیم. برای دیگر سازندگان FPGA هم نرم افزارهای مشابهی وجود دارد.

۳.۴. مدیریت پالس ساعت

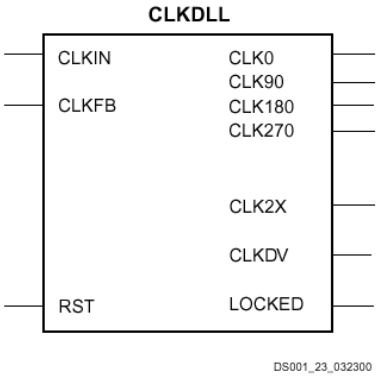
فرض کنید چند فلیپ فلاپ پشت سر هم قرار گرفته اند. و قرار است که یک شیفت رجیستر تشکیل دهند. خطای تاخیر هر FF را 1 نانو ثانیه در نظر بگیرید. به این مفهوم که فاصله زمانی لبه بالا رونده ساعت تا آماده شدن داده جدید روی خروجی فلیپ فلاپ 1 نانو ثانیه است. یک پالس ساعت یکسان به تمام فلیپ فلاپ ها می رود. حداکثر تاخیر مجاز برای سیمی که پالس را به فلیپ فلاپ ها می رساند چقدر است؟ طبیعی است که اگر مثلا لبه بالا رونده به FF آخر 1.2 نانو ثانیه دیرتر از FF قبلی برسد، شیفت رجیستر دیگر درست عمل نخواهد کرد. چرا که زمانی که FF آخری می خواهد داده روی ورودی خود را بردارد، عملا دیگر داده ای در ورودی وجود ندارد. (داده قبلی از بین رفته و داده جدید دارد جای آن را می گیرد.) یعنی یک بیت داده از دست می رود.

حال فرض کنید یک منبع تولید پالس داریم که با فرکانس 100 MHz کار می کند. این پالس ساعت به دو IC که در مجاورت یکدیگر قرار دارند می رود. خروجی IC اول به ورودی IC دوم وصل شده است. پالس ساعت از بافر ورودی که برای آن روی هر IC تعبیه شده عبور می کند و وارد IC می شود. حال فرض کنید داخل IC اول تعداد اندکی فلیپ فلاپ وجو دارد بنابراین از بافر مربوط به پالس ساعت جریان اندکی کشیده می شود. میزان خازنی که بافر در خروجی خود می بیند کوچک است لذا شکل پالس ساعت داخل IC حفظ می شود، زمانهای بالا رفتن و پایین آمدن برای پالس تغییری نمی کنند. فرض کنید در IC دوم بر خلاف IC اول تعداد زیادی فلیپ فلاپ وجود دارد، اگر طراحی بافر مربوط به پالس ساعت خوب انجام نگرفته باشد زمان های صعود و نزول پالس در اثر ظرفیت های خازنی داخل IC بزرگ خواهد شد. بنابراین وقتی منبع اصلی تولید پالس یک لبه بالارونده ایجاد می کند ، FF های داخل IC اول بلافاصله آن را می بینند و مطابق با آن عمل می کنند در حالیکه FF ها IC دوم لبه بالارونده را با تاخیر خواهند دید. به سادگی این منجر به از دست رفتن داده هایی می شود که قرار بود بین دو IC انتقال یابد. فرض کنید IC اول یک SDRAM و IC دوم یک FPGA است، از دست رفتن بعضی از بلوک های داده می تواند به سادگی منجر به مختل شدن کل عملکرد مدار شود.

Delay Locked Loop عنصری است که داخل FPGA وجود دارد و کارش این است که بر حسب میزان باری که روی پالس ساعت است، عناصر تاخیری در مدار پالس ساعت داخلی را کم و زیاد کند تا اینکه لبه های

بالارونده داخل FPGA و خارج آن هماهنگ شوند. از طرفی سیمهایی که برای انتقال پالس ساعت داخل FPGA وجود دارند به روش خاصی ساخته می شوند تا تاخیری بسیار اندک (در حد دهم نانوثانیه) داشته باشند.

حالا فرض کنید یک منبع پالس 50 MHz داریم که به یک FPGA و یک DDR SDRAM متصل است. DDR SDRAM در هردو لبه بالارونده و پایین رونده کار می کند. مثلا در هر دو لبه برای FPGA داده می فرستد. FPGA برای آنکه بتواند تمام داده ها را دریافت کند لازم است با سرعت 100 MHz کار کند. بنابراین نیاز به یک دو برابر کننده پالس ساعت داریم. این را هم DLL می تواند انجام دهد. روش دیگر آن است که FF ها را به دو دسته تقسیم کنیم: آنها که به لبه بالا رونده CLK0 کار می کنند، و دسته دوم آنهایی که با لبه بالا رونده CLK180 کار می کنند. CLK180 به اندازه 180 درجه با CLK0 اختلاف فاز دارد. DLL می تواند از CLK برای ما یک CLK180 تولید کند. در شکل (6) نمایش بلوکی DLL آمده است.



شکل (6)

درگاههای اصلی به این ترتیب است:

CLKIN : پالس ساعت اصلی که وارد FPGA می شود.

CLK0 : پالس ساعت اصلاح (De Skew) شده که به مدارات logic داخل FPGA می رود. (CLK90 نود درجه اختلاف فاز دارد و ...)

CLK2X : پالس ساعت با فرکانس دو برابر CLKIN

CLKDV : پالس ساعت با فرکانسی به نسبت دلخواه نسبت به CLKIN

LOCKED : نشان می دهد که آیا خروجی های DLL به حالت متعادل خود رسیده اند یا نه.

CLKFB : یک نمونه از پالس ساعتی که داخل FPGA وجود دارد برای مقایسه با پالس ساعت اصلی به DLL باز می گردد.

RST : ورودی Reset

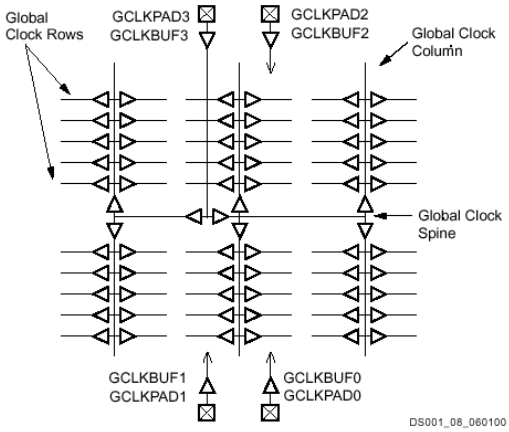
برای استفاده از DLL در مدارهایمان باید برنامه ها و طراحی ها را به شکل خاصی انجام دهیم.

در Virtex-II نام دیگری برای ClockDLL انتخاب شده است: Digital Clock Manager یا DCM، توانایی های این مدول نسبت به نگارش های قبلی خود بسیار افزایش پیدا کرده است. یک مثال عملی، فرض کنید

شما در مدارهای خود داخل FPGA نیاز پیدا کردید به یک حافظه خیلی بزرگ برای ذخیره کردن داده ها. مثلا فرض کنید به شما گفتند یک مدار فشرده ساز تصویر رو روی Virtex-II پیاده کنید. این مدار مثلا قراره به یک دستگاه MRI از یک طرف و از طرف دیگه به یک رابط Ethernet فوق العاده سریع وصل بشه تا به این ترتیب بشه عکس های اورژانسی یک مریض رو به سرعت گذاشت روی شبکه تا بقیه پزشک ها در جاهای دیگه دنیا ببینند و اظهار نظر کنند. کاری که فشرده ساز تصویر شما این وسط می کنه اینه که عکس های حاصل از MRI رو که مثلا هر کدوم ۱۰ مگابایت حجم دارند، فشرده کنه تا روی شبکه سریع تر انتقال پیدا کنند. مدار شما اینطوری کار می کنه که عکس رو از بیرون می گیره، می ریزه توی بافر خودش، روش پردازش انجام می ده و فشردش می کنه و سپس می ده به رابط Ethernet. فرض کنید که شما یک الگوریتمی برای فشرده کردن تصویر استفاده می کنید که برای یک عکس ۱۰ مگابایتی کلا به ۲۵۶ مگابایت حافظه احتیاج داره. ولی کل رم موجود داخل یک Virtex-II حداکثر ۴ مگابیت بیشتر نیست که اصلا به درد کار ما نمی خوره. پس ما باید رم خارجی به FPGA وصل کنیم. این رم باید خیلی سریع باشه، همچنین باید حجم زیادی هم داشته باشه. یکی از بهترین انتخاب های ممکن DDR SDRAM هایی است که روی کارت های گرافیکی وجود داره. این DDR SDRAM ها علاوه بر اینکه با فرکانس های خیلی بالایی می تونن کار کنن، اولا خیلی گرون نیستند و ثانیا قدرت نقل و انتقال داده خیلی خوبی دارند. (DDR SDRAM گرافیکی با DDR SDRAM معمولی فرق داره. Delay ها توی اولی خیلی کمتره. فقط یک نوع DRAM دیگه هست که از اینها سریع تره : و اون DDR FCRAM است که خیلی گرون قیمته.) حالا فرض کنید فرکانس ارتباط با DDR SDRAM قراره ۲۰۰ مگاهرتز باشه. و مدار داخلی باید ۱۲۵ مگاهرتز کار کنه. Virtex-II دارای IOB هایی است که مخصوص ارتباط DDR طراحی شدن. یعنی بر خلاف Spartan-II که ۳ تا فلیپ فلاپ داخل IOB داره، Virtex-II دارای ۶ تا فلیپ فلاپه. ۳ تا با لبه بالا رونده و ۳ تا با لبه پایین رونده پالس ساعت کار می کنند. خوب دقت کنید. فرکانس کار خیلی خیلی بالاست. طراحی باید به دقت انجام بشه. حالا فرض کنید روی برد یک منبع پالس ۱۲۵ مگاهرتزی داریم، اولین کاری که لازمه بکنیم ، اینه که از روی اون ، یک پالس ۲۰۰ مگاهرتزی تولید کنیم. DCM های Virtex-II به راحتی این کار رو انجام می دن. یک DCM می تونه یک پالس ساعت از ورودی دریافت کنه، و در خروجی پالسی تولید کنه که فرکانسش $\frac{M}{D}$ فرکانس پالس ورودی باشه. از طرفی وقتی DDR SDRAM داده هاش رو گذاشت روی باس تا FPGA بخونه ، لبه پالس مربوط به خوندن داده باید یک وقتی به فلیپ فلاپ ها بره ، که تمام خروجی های RAM در حالت Valid باشند. مثلا باید پالس مربوط به خوندن داده ، با پالس ساعت ۲۰۰ مگاهرتزی که به RAM می ره، ۱۴۰ درجه اخلاف فاز داشته باشه. DCM به راحتی می تونه چنین پالسی رو تولید کنه. می بینیم که مدیریت پالس ساعت چقدر مهمه.

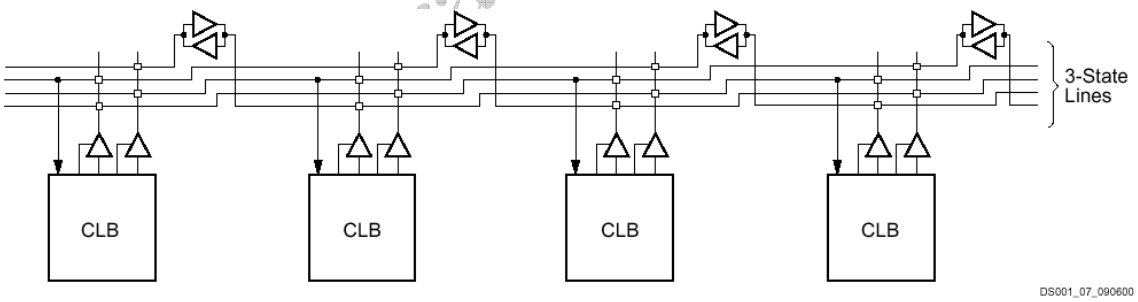
۳.۵. سیمهای ارتباطی داخلی

داخل FPGA مقدار بسیار زیادی سیم وجود دارد. تعداد بسیار زیادی هم سوئیچ هایی که می توانند هر کدام از پورتهایشان را به هر کدام دیگر وصل کنند. مجموعه این سیم ها و Switch Matrix ها به کار می روند تا قسمت های مختلف مدار logic که قرار است روی FPGA پیاده شود به هم متصل شوند. همانطور که قبلا توضیح داده شد برای CLK و Carry روی FPGA سیم های انتقالی خاص وجود دارد. شکل (7) شبکه ای که برای پخش کردن پالس ساعت داخل FPGA به کار می رود را نشان می دهد.



شکل (7)

به عنوان مثال در Spartan-II تمام CLB هایی که در یک سطر قرار دارند، با خطوط ارتباطی مستقیما به هم وصل می شوند. شکل (8) یک نوع از منابع ارتباطی روی FPGA را که برای CLB ها به هم به صورت 4 تا 4 تا، به کار میرود، نشان می دهد.



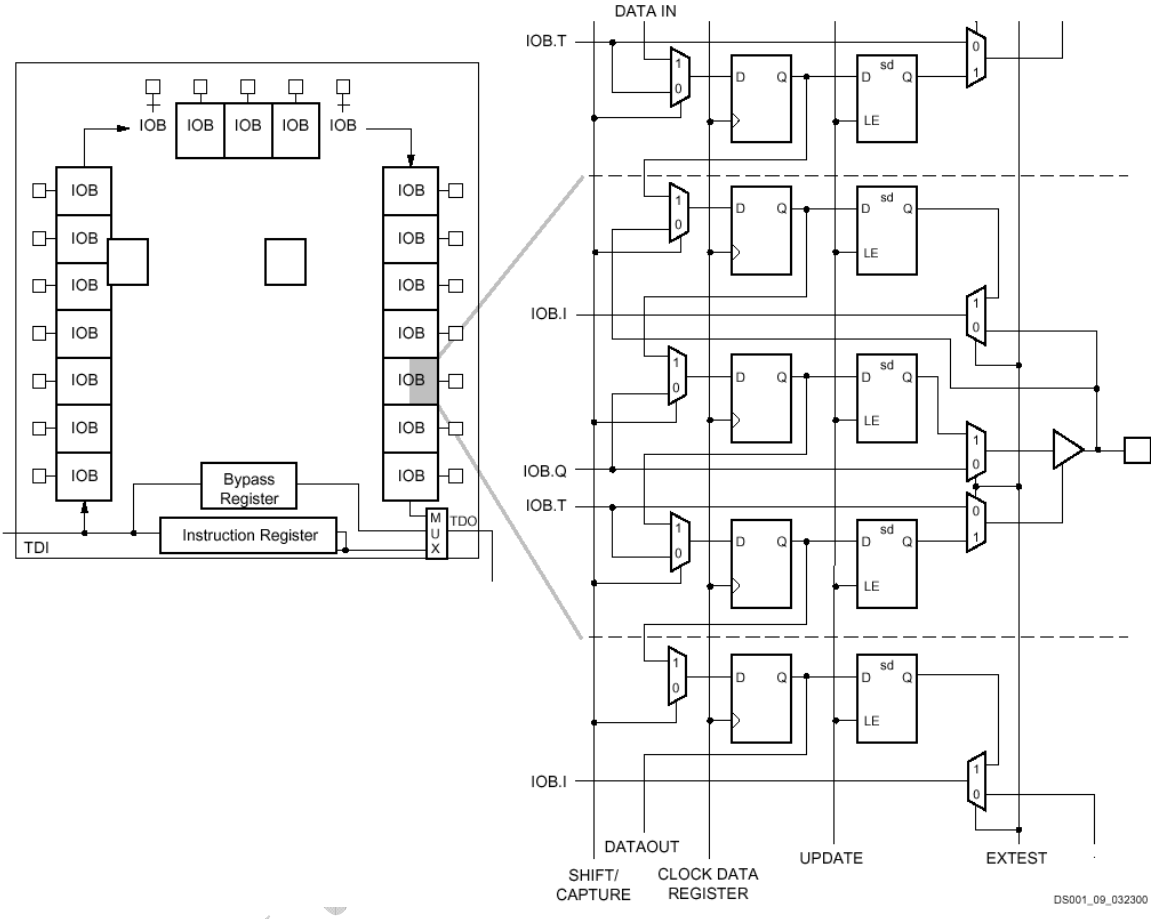
شکل (8)

معمولا هر Switch Matrix از چهار طرف به مشابه های خود به وسیله تعداد زیادی سیم وصل است. هر CLB هم به یک ماتریس وصل است. از طرفی هر ماتریس با یک مجموعه سیم به میاتریس های 6 دسته آنطرف تر هم وصل می شود.(در Spartan-II.) مقدار سیمها و منابع routing داخل FPGA خیلی خیلی زیاد است و شما اصلا لازم نیست نگران تمام شدنشون باشید. چیزی که به هرحال شما موقع طراحی باید مواظبش باشید، اینه که مدار با فرکانس خوبی کار کنه. معمولا یکی از مهمترین جاهایی که باعث می شه تاخیر مدارها زیاد بشه و فرکانس کاریشون پایین بیاد سیم ها هستند. تاخیر یک سیم داخل FPGA به راحتی می تونه چندی برابر تاخیر مدارهای

logic باشه. به هر حال تعداد سیمهایی که می تونن سیگنال رو برای طولهای زیاد، داخل FPGA با تاخیر کم انتقال بدن محدوده.

۳.۶. مدارهای موجود برای تست عملکرد

شکل (9) مدار Boundary Scan را نمایش می دهد.



شکل (9)

این مدار برای وقتی استفاده دارد که بخواهیم حین عملکرد FPGA در مدار وضعیت پایه های مختلف آنرا به دست آوریم. این مدار در واقع یک شیفت رجیستر بزرگ است که دور تا دور FPGA را فرا گرفته، چند پایه خاص روی هر FPGA وجود دارد که ورودی و خروجی این شیفت رجیستر و سیگنالهای مربوط به کنترل آن را فراهم می کند. به این ترتیب هم می توانیم وضعیت سیگنالهای داخل FPGA را در یک زمان خاص پیدا کنیم و هم مقادیر مختلفی را بفرستیم داخل FPGA. به مجموعه این پایه ها درگاه JTAG می گویند.

تمام IC های Logic بزرگ حتما درگاه JTAG دارند. به این وسیله می توان تست کردن عملکرد بخش های مختلف یک بورد را بسیار آسان کرد. به این ترتیب که در یک لحظه خاص IC را متوقف می کنیم و مقدارهای موجود روی Shift Register داخل IC را شیفت می دهیم بیرون و به وضعیت پایه ها (بدون نیاز به اسکوپ) پی

می بریم. وقتی که PCB خیلی بزرگ می شه و سه جهار تا IC توپول می یاد روش، این خیلی به درد می خوره. چون باش می شه فهمید در یک لحظه وضعیت تمام سیمهای روی PCB که IC ها رو به هم وصل می کنند چه جوریه.

۳.۷. ضرب کننده

در بعضی از محصولات جدید هردو شرکت Altera و Xilinx بلوک های ضرب کننده وجود دارد. یعنی عمل ضرب را در این FPGA ها می توان دو جور پیاده کرد: یکی با استفاده از CLB ها و دوم با کمک همین ضرب کننده ها. مثلا برای Virtex-II این بلوک ها دو عدد 18 بیتی را در ورودی می گیرند و حاصلضرب دو عدد (تا 18 بیت) را می دهند. اینها می توانند در یک سیکل و یا به صورت Pipe line کار کنند. در حالت Pipe طبیعتا سرعت بالا تر است. (حدودا ۲۰۰ مگاهرتز) برای استفاده از اینها می توان از Core Generator کمک گرفت. دوباره مثل قبل اگه ابزار سنتز باهوش باشه وقتی شما توی برنامه ای که نوشتید از عمل ضرب استفاده کرده باشین، خودش میره از این dedicated multiplier های موجود داخل FPGA استفاده می کنه. این ضرب کننده ها معمولا برای کارهای پردازش سیگنال (که عملا چیزی جز کانولوشن نیستند.) خیلی به درد می خوره.

۳.۸. میکروپروسسور داخلی

جدید ترین محصول Xilinx (Altera هم اخیرا چنین محصولاتی تولید کرده.) یعنی Virtex-II Pro دارای چند میکروپروسسور PowerPC داخل خودش است. یعنی علاوه بر تمام قسمت های یک Virtex-II یک یا چند میکروپروسسور هم داخل FPGA وجود دارد که می توان از آنها هم برای پیاده سازی مدار استفاده کرد. مدار logic با VHDL یا Verilog طراحی می شود، و اعمالی که باید PowerPC انجام دهد به صورت برنامه (که بعدا برای اجرا روی این میکروچروسسور compile می شود.) نوشته می شود. مجموعه اینها باهم عملکرد Virtex-II Pro را مشخص می کنند. این FPGA واقعا تخصصی است و برای کارهای خاص (مثل سوئیچ های سریع) باید از آن استفاده کرد. PowerPC در واقع میکروپروسسوری است که توسط IBM تولید می شود. هنگامیکه یک همچین مداری داخل FPGA استفاده می شود اصلاحا آن را Hard Core IP می نامند.

۳.۹. روشهای پیکربندی

Configuration همان عملی است که طی آن وضعیت عملکرد تمام Mux ها و LUT ها و Switch Matrix های داخل FPGA تعیین می شود و به این ترتیب معلوم می شود FPGA قرار است چه کاری انجام دهد. باید به یک روشی اطلاعات مربوط به Configuration را به FPGA داد. حال ما دو روش را بررسی می کنیم:

همانطور که گفتیم با استفاده از JTAG می توان داده ها را فرستاد داخل FPGA. مود خاصی از عملکرد FPGA وجود دارد که طی آن اطلاعات آمده از درگاه JTAG برای پیکربندی استفاده شده، در محل مناسب در

FPGA قرار می گیرد. برای این منظور لازم است بوردی را که FPGA روی آن قرار دارد به کامپیوتر وصل کرد تا اطلاعات از طریق JTAG انتقال یابد. این برای مصارف تجاری مناسب نیست. ولی برای اول کار، وقتی که شما دارید تازه اولین کپی از محصول خود رو می سازید، خیلی خیلی عالیه. چون به شما اجازه می ده مدار رو پی در پی تست کنید و حالت های مختلف رو آزمایش کنید. کابل JTAG مدار خیلی ساده ای داره، و به راحتی خودتون می تونید یکیش رو بسازید. همه جا هم می تونید ازش استفاده کنید. معمولا این کابل به پورت موازی کامپیوتر متصل میشه. بعد مثلا برای FPGA های Xilinx برای اینکه داده های Configuration رو بریزید روی FPGA از نرم افزار JTAG Programmer استفاده می کنید. چیزی که همیشه باید حواستون بهش باشه، اینه که خود کابل موقع Config کردن خراب نباشه.

روش دیگر (که وقتی استفاده می شود که قرار است FPGA روی یک بورد به صورت مجزا کار کند.) آن است که اطلاعات مربوط را در یک PROM بریزیم و آن را به FPGA وصل کنیم. (معمولا program کردن این PROM با استفاده از پورتJTAG آن انجام می شود.) هنگامیکه FPGA بخواهد خود را Configure کند سراغ این PROM می آید. انتقال داده بین FPGA و PROM می تواند به صورت سریال و یا موازی انجام شود. همچنین دو حالت برای پالس ساعتی که با لبه بالارونده آن داده انتقال می یابد ممکن است: حالتی که پالس خارجی است ، یعنی یک عنصر خارجی به PROM و FPGA می گوید که کی داده را روی باس قرار دهند و کی بردارند. به این مود انتقال داده Slave می گویند. مثلا Slave Serial Mode که در آن با یک پالس خارجی داده به صورت سریالی به FPGA انتقال می یابد. حال دوم حالت Master است که در آن پالس انتقال داده را خود FPGA تولید می کند. در تمام FPGA ها یک اسیلاتور نادقیق برای انجام این کار وجود دارد. این اسیلاتور یک پالس مثلا ۲ تا ۸ مگاهرتزی (برای Spartan) تولید می کند. (که البته فرکانس آن دقیق نیست و تابع دما است.) آنچه که باید روی PROM ریخته شود یک فایل با پسوند BIT است که بعد از انجام مرحله Implement ساخته می شود. آنچه که مهم است این است که طول فایل بیت صرف نظر از اینکه چه مداری قرار است روی FPGA پیاده شود ، برای یک FPGA خاص ثابت است. یعنی به هر حال وضعیت تمام قسمت های FPGA تعیین خواهد شد. PROM های خاص برای این کار توسط خود شرکت های تولید کننده FPGA ساخته می شود. بعضی از آنها OPT ROM هستند : One Time Programmable ROM که فقط یک بار می توان آنها را برنامه ریزی کرد. مثل سری XC17Vxx. بعضی دیگر به تعداد بار دلخواه قابل برنامه ریزی اند مثل XC18Vxx. قیمت این PROM ها راستش رو بخواین ، یک کم عجیب غریبه! چون قیمتش نزیک به قیمت خود FPGA هاست. مثلا اگه FPGA هست ۱۹ هزار تومان ، یکی از این PROM ها هست ۱۶ هزار تومان. حالا شاید اینجا، اینطور باشد. به هرحال مثلا اگه شما یک XC2S100 بخرید (که یک Spartan-II با ۱۰۰ هزار گیت است.) به قیمت ۲۰ هزار تومان آنوقت باید ۴۸ هزار تومان هم بدهید ، IC ها PROM لازم برای آن را خریداری کنید.

۳.۱۰. یک مثال

تمام خصوصیات یک FPGA ی کامل را می توان در Virtex-IIدید. (لطفا برای دیدن جداول مربوط به تعداد Slice ها و مقدار بلوک های حافظه به Datasheet مربوط به Virtex-II مراجعه کنید.) کوچکترین FPGAی این خانواده XC2V40 است که دارای 256 عدد Slice می باشد. همچنین 4 بلوک ضرب کننده و 4 بلوک حافظه در آن وجود دارد. نهایتا برای مدیریت پالس ساعت 4 عدد Digital Clock Manager در آن تعبیه شده است. XC2V40 می تواند تا 88 پایه IC را به استفاده کننده اختصاص دهد، استفاده کننده می تواند با این 88 پایه هر کاری که می خواهد بکند. آنها را ورودی ، خروجی و یا ورودی/خروجی تعریف کند و به نحو مناسب آنها را به مدار داخلی FPGA ربط دهد. تمام این کارها بر اساس برنامه Verilog و یا VHDL ای که استفاده کننده می نویسد و محدودیت هایی که در مراحل Implement تعیین می کند، انجام می شود. بزرگترین IC در این خانواده XC2V10000 است با 61440 ، Slice که می توان مدارهای بسیار بزرگ را با آن پیاده کرد. Slice ها در Virtex-II اصلاح شده اند تا مدارهای Logic بتوانند با تاخیرهای کمتر پیاده شوند. در حال حاضر Virtex-II در چندین Speed Grade مختلف تولید می شود: 5- , 4- و 6- ، هرچه این عدد بالا تر رود به این مفهوم است که در ساختن FPGA از مواد بهتر ، تکنیک ها و تکنولوژی پیشرفته تر استفاده شده است و تاخیر ها در FPGA کوچکتر هستند. پس یک FPGA با Speed Grade برابر با 6- گران تر و سریع تر از همسان خود با سطح سرعت 5- است. با Virtex-II می توان مدارهایی تا فرکانس 300 مگاهرتز را پیاده کرد. تعداد Block RAM هایی که در این FPGA استفاده می شود نسبت به تمام سری های قبل بیشتر است. حجم هر بلوک ۱۸ کیلوبیت است. بلوک به صورت تک و دو پورتی قابل استفاده است و عرض پورت ها می تواند متفاوت باشد. می توان تنظیم کرد که Block RAM وقتی هر دو پورت می خواهند به یک محل حافظه دسترسی پیدا کنند (مثلا یکی می خواهد بخواند و دیگری می خواهد بنویسد.) چه جوری عمل کند. مثلا داده کنونی در آن محل به پورت مربوط به خواندن برود و داده جدید نوشته شود، یا اینکه داده جدید نوشته شود و همین داده به پورت خواندن برود.

IOB در Virtex-II انواع متنوعتری از استانداردهای سیگنالینگ را حمایت می کند. با استفاده از LVDS (Low Voltage Differential Signaling) در Virtex-II می توان داده های سریال را با سرعت 840 MBit/Sec انتقال داد.

۴. ابزارهای مورد استفاده

کلا برای هر طرحی که قرار است روی FPGA پیاده شود، باید یک سری کارهای مشخص انجام شود. ابتدا باید مدار اصلی طراحی شود. در طراحی مدار اصلی معمولا اینطور عمل می کنند که ابتدا تعداد مدول ها، عملکرد هر کدام و ارتباط آنها باهم را مشخص می کنند و سپس به طراحی تک تک مدول ها می پردازند. به این روش طراحی، روش top top down design گفته می شود. یعنی شما ابتدا عملکرد کلی را تعیین می کنید و بعد هر کدام از اجزا را به دقت توصیف می کنید تا آن نتیجه مطلوب حاصل شود. کلا کارهایی که تا مرحله آماده شدن کد Verilog برای انجام شبیه سازی انجام می شود را Design Entry می گویند. پس بعد از مرحله Design Entry کد های Verilog ما آماده هستند. حال به مرحله Functional Simulation می رسیم. در این مرحله عملکرد مدار را در حالت ایده آل (تاخیر صفر) شبیه سازی می کنیم تا مطمئن شویم طراحی را درست انجام داده ایم. یعنی به ورودی هایی مدار هر دفعه سیگنالهای متفاوتی می دهیم و سپس به خروجی نگاه می کنیم تا ببینیم خروجی درست داده می شود یا نه. طبیعی است که اگر جواب برای هر یک از مراحل تست درست نباشد دوباره باید به مرحله Design Entry بازگشت و طرح را اصلاح کرد. معمولا همراه هر مدول Verilog که می نویسیم یک برنامه دیگر (یک مدول دیگر) به اسم Verilog Test Fixture هم می نویسیم. این برنامه کارش این است که سیگنالهای مناسب را برای ورودی مدول تحت تست تولید می کند. به این ترتیب عملکرد کلی این است که با استفاده از یکی از نرم افزارهای شبیه سازی Verilog مجموعه مدول اصلی و مدول تست آن، را که به هم وصل شده اند شبیه سازی می کنیم و می بینیم که آیا مدول اصلی درست کار می کند یا نه.

پس از اطمینان از عملکرد صحیح مدار به مرحله Synthesis می رویم. در این مرحله با استفاده از یکی از نرم افزارهای Synthesizer مدار را تبدیل به مجموعه ای از گیت های منطقی می کنیم. باز این مجموعه گیت ها را می شود تبدیل به یک برنامه Verilog کرد و حاصل را شبیه سازی کرد. به این ترتیب مطمئن می شویم خروجی ابزار سنتز همان چیزی است که ما می خواهیم. تا این زمان به عنوان ابزار های شبیه سازی و سنتز از محصولات هر شرکتی که بخواهیم می توانیم استفاده کنیم. مرحله آخر آن است که خروجی Synthesizer را به ابزار Implement بدهیم. ابزار Implement فایلی که خروجی Synthesizer است را می گیرد و آن را تبدیل می کند به ترکیبی از المانهای موجود روی FPGA . سپس این عنصر ها را سر جای مناسب روی FPGA قرار می دهد. (Placement) و بین آنها را سیم کشی می کند. (Routing) در نهایت یک فایل با پسوند BIT تولید می شود که ما می توانیم از آن برای Program کردن Rom ای که قرار است به FPGA وصل شود استفاده کنیم.

ابزار Implement هر شرکت مخصوص خودش است، مثلا برای FPGA های Xilinx حتما باید از ابزار Implement خود Xilinx استفاده کرد.

۴.۱. طراحی ابتدایی

برای کسانیکه می خواهند یک برنامه Verilog معمولی بنویسند، به عنوان Design Entry Tool فقط یک Editor کافیست، تا بتوانند برنامه Verilog خود را در آن تایپ کنند. ولی برای پروژه های بزرگ این اصلا کافی نیست.

نرم افزار HDL Designer (HDS) : یکی ازقدرتمندترین نرم افزارهایی که میتوان برای Design Entry استفاده کرد HDL Designer تولید شرکت Mentor Graphics می باشد. HDL Designer می تواند از schematic ای که شما برای مدار کشیده اید، کد Verilog بسازد. به عبارت ساده شما به کمک HDS می توانید یک مدار شامل:

- Block Diagram هایی که ورودی/خروجی ها، مدول ها و ارتباط های آنها با هم را نشان می دهد.
- State machine ماشین های حالتی که بر اساس لبه ساعت و مقدار ورودی از یک حالت به حالت دیگری می روند و خروجی مناسب را هم تولید می کنند.
- Truth table جدول های درستی که بر اساس ورودی، خروجی را مشخص می کنند.
- Flow chart که نشان می دهد یک مجموعه منظم کارها با توجه به شرایط درهر مرحله گونه باید انجام شود.

طراحی کنید بدون اینکه حتی یک سطر کد Verilog یا VHDL بنویسید. مثلا در پیاده کردن State Machine فقط حالتهای ماشین حالت را که به صورت دایره هستند در نقاط مناسب روی صفحه می گذاریم و آنها را با یک سری خط جهت دار به هم وصل می کنیم. این خطوط در واقع بیانگر transistion از یک حالت به حالت دیگر هستند. شرطی که طی آن این انتقال از یک حالت به حالت دیگر رخ می دهد، همچنین خروجی ای که باید در هنگام این انتقال تولید شود را روی خود خط می نویسیم. در رسم بلوک دیاگرام بلوکها را پهلوی هم قرار داده بین آنها سیم کشی می کنیم. در Truth table یک جدول را پر می کنیم که تعیین می کند برای هر ورودی چه خروجی باید داده شود. در نهایت Flow chart دقیقا عین همان است که در برنامه نویسی از آن استفاده می شود. یک مجموعه اعمال مشخص از بالا تا پایین به ترتیب انجام خواهند شد. شرطهایی وجود دارد که معمولا باعث پرش به نقاط مناسب می شود.

آنچه که HDS به ما می دهد، چیزی جز کد Verilog نیست. HDS می تواند تمام طرحهای ما را که اصلا به Verilog نبوده خود به خود تبدیل به کد کند و به ما دهد. کدی که تولید می کند Synthesizable است. یعنی قابل تبدیل به مجموعه گیت های منطقی می باشد و ابزار سنتز آن را می فهمد. به هر حال این امکان HDS نیاز به دانستن Verilog را منتفی نمی کند، کسی که بخواهد به صورت موثر در طرحهایش از HDS استفاده کند، لازم است حتما کاملا بر Verilog مسلط باشد. آموزش طرز عمل کردن با HDS ساعتها زمان طلب می کند. به

عنوان یک مرجع خوب برای شروع کار با آن می توان به Tutorial ای که در بخش Help آن وجود دارد، رجوع کرد.

نرم افزار Xilinx Core Generator : یک Core عبارت است از یک مدار که قبلا آماده شده و ما می توانیم از آن بصورت آماده در طرحهایمان استفاده کنیم. مثلا در بسیاری از مدارهای Logic شمارند وجود دارد، حال به جای اینکه هر کس برای خودش یک شمارنده طراحی کند و آن را بهینه کند، یک گروه طراح یک شمارنده قابل انعطاف و بهینه از لحاظ سرعت طراحی می کنند و آن را به هر کسی که بخواهد می فروشند. به این ترتیب یک کار لازم نیست چند بار انجام شود. Core Generator نرم افزاری است که این نوع مدارهای آماده را به صورت فایل EDIF تولید می کند. شما در برنامه Verilog خود فقط مدولی را که توسط Core gen تولید شده، صدا می زنید. ولی آن را توصیف نمی کنید. بعدا هنگام انجام Implementation فایل edf حاصل از Core gen پهلوی فایل edf حاصل از سنتز مدار اصلی قرار می گیرد و به این ترتیب مدار core به مدارهای ما اضافه می شود.

Editor های خوب : یکی از مهمترین ابزارهایی که برای طراحی استفاده می کنیم Editor است. مثلا notepad یک Editor کاملا معمولیست که برای کارهای ساده مناسب است. یک Editor خوب حتما باید خاصیت Syntax highlighting داشته باشد تا فهم و غلط یابی کد Verilog را برای ما آسان کنید. در حال حاضر دو Editor فوق العاده قدرتمند برای نوشتن برنامه های Verilog وجود دارند: اولی TextPAD که فقط تحت ویندوز موجود است و دومی Nedit که تحت Linux کار می کند و نگارنده ادعا می کند از TextPAD بهتر است. چون هم Open Source است و هم تمام آن کارها را انجام می دهد. هیچ وقت و تحت هیچ شرایطی نظم در نوشتن برنامه ها را فراموش نکنید. وقتی برنامه بزرگ می شود، تست کردن آن و غلط یابی برنامه فقط وقتی ممکن است که نظم در نوشتن آن رعایت شده باشد.

۴.۲. ابزارهای شبیه سازی

برنامه هایی وجود دارد که کد Verilog ما را می گیرند و عملکرد آن را برای ما شبیه سازی می کنند. به این ترتیب که ما شکل ورودی مدول تحت تست را برای آنها تعیین می کنیم و آنها خروجی مدول را به ما می دهند.

نرم افزار ModelSim : ModelSim محصول شرکت Model Technology که خود یکی از شعبات Mentor Graphics است، یکی از قوی ترین و مشهورین شبیه سازهایی است که برای شبیه سازی مدارهای Logic که با Verilog یا VHDL و یا هردو توصیف شده اند، به کار می رود. کار در ساده ترین حالت خود حول یک صفحه با نام waves می چرخد که در آن تمام شکل موج ها نمایش داده می شود. در ابتدای کار که ModelSim بالا می آید تنها یک پنجره کوچک در اختیار استفاده کننده قرار می دهد تا دستورات خود را وارد کند. برای اینکه عملکرد مشخص شود با یک مثال ادامه می دهیم:

فرض کنید می خواهیم عملکرد مدار مثال (3) در بخش قبل را شبیه سازی کنیم. اولین کاری که لازم است، نوشتن یک مدول دیگر است که سیگنالهایی را که برای تست مدول اصلی مان نیاز داریم تولید کند، به این برنامه Verilog به اصطلاح Test Fixture گفته می شود. شاید به نظر دشوار بیاید که بخواهیم برای تست کردن برنامه اصلی یک برنامه دیگر بنویسیم که نوشتن آن چندان هم آسان نیست. معمولا افرادی که تازه کار هستند بیشتر ترجیح می دهند موجها را مستقیما و بدون استفاده از یک برنامه دیگر به مدول اعمال کنند. هرچند با گذشت زمان فرد به اهمیت تست کردن مدول پی می برد. و کم کم به نوشتن Test Fixture های مناسب رو خواهد آورد. ممکن است نوشتن Test Fixture و یا به عبارت دیگر Test Bench برای طرح از خود مدت زمان لازم برای طراحی بیشتر طول بکشد. در همین زمان است که بسیاری از نکات مخفی طرح برای طراح مشخص می شود و به بسیاری از اشتباهاتش پی می برد. Verify کردن یک مدول، یعنی بررسی اینکه آیاعملکرد آن درست هست یا نه، خود یک علم است.

شرکت های فراوان و محصولات متنوعی برای انجام عمل Verification وجود دارد. معمولا تست یک مدول به این صورت انجام می شود که ابتدا با Verilog یا SystemC یا یک چیز دیگر، برای آن مدول یک مدل نوشته می شود. یعنی برنامه ای نوشته می شود که هرچند قابل سنتز نیست، همان شکل موجها را در مقابل تحریک ورودی تولید می کند. حال با استفاده از یکی از نرم افزاهای مشهور برای Verification مثلا Specman Elite از Verisity یا Vera محصول Synopsys یا Test Builder محصول Cadence (که البته شایان ذکر است ، Cadence اینجا مرام گذاشته و Test Builder را به صورت Open Source و free منتشر می کند. عوضش پولش رو در LDV و Transaction Navigator می گیره.) خلاصه با یکی از این نرم افزارهای انواع و اقسام شکل موجها و حالت هایی رو که فکر می کنیم ممکنه پیش بیاد درست می کنیم و این شکل موجها به هردو مدول (مدول اصلی که به آن DUT یعنی Design Under Test یا DUV یعنی Design Under Verification می گویند. و مدلی که برای این مدول نوشته شده.) می فرستیم. حاصل خروجی مدول اصلی و مدل آن را می گیریم و باهم مقایسه می کنیم. اگر با هم هیچ تفاوتی نداشتند تا حدودی (نه کاملا، چون ممکن است مدل هم خودش غلط باشد.) مطمئن می شویم عملکرد مدار ما درست است. Specman و یا Vera هرکدام برای تولید شکل موجهای ورودی زبان برنامه نویسی مخصوص خودشان را دارند. معمولا این زبانها هم خیلی شبیه به C و C++ هستند. بحث ابزارهای Verification را در همین جا خلاص می کنیم، اگه نه ، مجبوریم تا فردا صبح در موردشون حرف بزنیم.

برای مدول ساده ما Test Fixture زیر کافیست:

```
module mux_test_fixture;
reg   [3:0]   mux_in;
reg   [1:0]   mux_control;
reg           clk;
wire          mux_out;
```

```
four_bit_mux I0(.mux_out (mux_out),
                .mux_in (mux_in),
                .mux_control (mux_control),
                .clk (clk));

initial begin
mux_control = 0;
clk = 0;
mux_in = 0;
end

always #4 clk = ~clk;
always #6 mux_control = mux_control + 1;
always #8 mux_in = mux_in + 5;

endmodule
```

در نوشتن Test Fixture سعی می شود طیف متفاوتی از سیگنالهای ورودی به مدول اصلی اعمال شود و خروجی آنها بررسی گردد. پس هم اکنون دو تا فایل داریم: یکی برنامه Verilog اصلی را با خود دارد و نام آن: four_bit_mux.v می باشد. فایل دیگر test bench است. نام mux_test_fixture.v را برای آن انتخاب می کنیم. نکته ای که باید به آن دقت کرد، این است که اولا test fixture دارای پورت های ورودی و خروجی نیست و ثانیا در داخل test fixture مدول اصلی را احضار کرده ایم تا به آن سیگنال بدهیم. پس اینجا مدول مرتبه بالاتر یا اصطلاحا top level، مدول mux_test_fixture خواهد بود.

خواننده ممکن است تعجب کند که چرا ما test bench و test fixture هر دو را برای یک منظور به کار می بریم. در واقع هر دو واژه فوق را برای مدولی که به منظور تست کردن یک مدول دیگر نوشته می شود به کار می برند. هرچند واژه درست تر test fixture است. Test bench مخصوص زبان VHDL بوده و test fixture مخصوص Verilog.

پس فرض کنید دو فایل فوق در دایرکتوری D:\test قرار دارند. ModelSim را اجرا می کنیم و به این دایرکتوری می رویم، دستور زیر را در ModelSim وارد می کنیم:

cd {D:/test}

حال لازم است در این دایرکتوری یک library بسازیم. Library آن دایرکتوری است که برنامه های Verilog ما به صورت کامپایل شده در آن قرار می گیرند. وقتی که قرار است عمل شبیه سازی انجام شود اطلاعات از آن دایرکتوری خوانده می شود. پس این دستور را وارد می کنیم:

vlib sim_1

حال اگر به D:\test بروید می بینید که یک دایرکتوری به نام simul_1 برای شما ساخته شده است. حال لازم است برنامه های Verilog خود را کامپایل کنید و داخل دایرکتوری sim_1 بریزید:

vlog –work sim_1 four_bit_mux.v mux_test_fixture.v

توجه : اگر دارد از همین فایلهایی که در متن گزارش هست به صورت Copy و Paste شده استفاده می کنید ، ممکن است در فایل four_bit_mux.v ، vlog از شما غلط بگیرد. این غلط مربوط است به علامت "'" که برای اعداد به کار می رود. مثلا 2'b01 یک عددباینری دو بیتی است. حال باید در یک editor مناسب (مثلا notepad) خودتان این علامت را (که در سمت راست صفحه کلید قرار دارد.) از نو تایپ کنید.

با صادر کردن دستور vlog ، ModelSim آن دو فایل را می خواند و حاصل compile را داخل sim_1 می ریزد. از طرفی گزارش می دهد که مدول top کدام است. حال می توانیم simulator را احضار کنیم:

vsim sim_1.mux_test_fixture

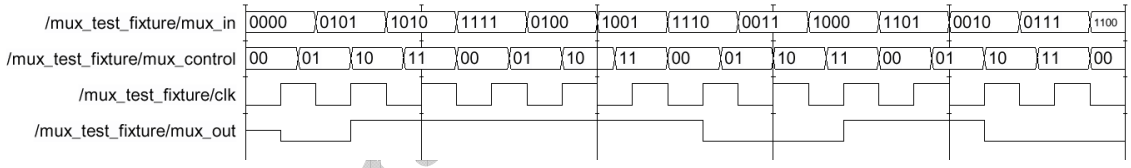
پنجره های Signals و wave را آشکار می کنیم:

view signals; view wave

به پنجره سیگنال می رویم. با دکمه Shift و با کمک ماوس تمام سیگنال ها را انتخاب می کنیم. سپس آنها را می کشیم و می اندازیم داخل پنجره wave. همه چیز آماده است. در پنجره ModelSim دستور زیر را وارد می کنیم:

run 1 us

شبیه سازی برای 1 میکرو ثانیه انجام می شود. می توان شکل موجهای حاصل در پنجره wave را دید.



شکل (10)

حال که از عملکرد مدار مطمئن شده ایم می توانیم به مرحله بعد یعنی Synthesis برویم. کارهایی که در فوق انجام دادیم ساده ترین اعمالی بود که ModelSim می توانست انجام دهد. ModelSim از Verilog PLI حمایت می کند. به این ترتیب مثلا شما می توانید خروجی های شبیه سازی را به جای روی صفحه wave روی پنجره برنامه ای که با Visual C نوشته اید ببینید. با Verilog PLI می توان با هسته داخلی شبیه ساز ارتباط برقرار کرد، به آن داده داد و یا داده از آن دریافت کرد. ModelSim زبان scripting ، Tcl را بطور کامل حمایت می کند. در واقع ورودی command در ModelSim چیزی جز یک Tcl Shell نیست. در دنیایی که بزرگترین طرحها باید در حداقل زمان ممکن به مراحل نهایی خود برسند، scripting و اتوماتیک کردن کارها بسیار اهمیت دارد.

قبل از آنکه سراغ ابزارهای سنتز برویم، چند شبیه ساز دیگر را هم که متداول هستند بررسی می کنیم.

نرم افزار LDV : LDV محصول شرکت Cadence در واقع مجموعه چند شبیه ساز Verilog و VHDL است که با هم به صورت یک محیط مجتمع عرضه شده اند. اولین شبیه ساز Verilog-XL است که در واقع زمانی

یک نسخه بهبود یافته از Verilog بود. Verilog-XL یک شبیه ساز تمام عیار Verilog است. به این مفهوم که هنگام کار با آن بسیاری از مفاهیم برنامه نویسی با Verilog ، که به خاطر سخت بودن کار با آنها، در شبیه سازهای دیگر حذف شده اند و یا مورد توجه قرار نمی گیرند، در Verilog-XL آشکارا دیده می شوند. چند سال پیش Verilog-XL قابلیت هایی داشت که شبیه سازهای معمولی Verilog فاقد آنها بودند. البته در زمان اصلاح دوم این متن (June – 2002) هنوز هم Verilog-XL قابلیت هایی دارد که بقیه شبیه سازها ندارند. Verilog-XL دارای یک سری system task هایی است که بقیه شبیه سازها آنها را ندارند. این به ما کمک می کند که بتوانیم برای شبیه سازی (به خصوص برای تست.) از قابلیت های بیشتری استفاده کنیم. برنامه شبیه ساز دیگری که در LDV وجود دارد Affirma NC-Verilog است که یک شبیه ساز Verilog بسیار سریع است. عملکرد آن به این ترتیب است که برنامه Verilog ای را که قرار است شبیه سازی شود می گیرد و آن را تبدیل به مجموعه دستورات اسمبلی مخصوص CPU ای که قرار است شبیه سازی روی آن انجام شود می کند. تبدیل به گونه ای انجام می شود که اجرای دستورات اسمبلی و مقادیری که برای هرکدام حاصل می شود، معادل باشد با وضعیت سیگنال ها و سطح منطقی آنها در مدار تحت شبیه سازی. به همین خاطر به آن NC-Verilog یعنی Native Code Verilog می گویند. فایل دستورات اسمبلی حاصل مستقیما برای اجرا شدن به CPU داده می شود. مقادیر سیگنالها در یک database ذخیره می گردد تا استفاده کننده بتواند بعدا آنها را ببیند. به این ترتیب عمل شبیه سازی با سرعت فوق العاده زیادی انجام می گیرد. این نوع شبیه سازها برای طرحهای بسیار بزرگ استفاده دارد، البته برای طرحهای کوچک هم می توان از آنها استفاده کرد. در نهایت نظر نگارنده آن است که LDV از ModelSim بسیار سریع تر، و پایدار است. ModelSim هنوز هم که هنوزه یک عالمه bug دارد، ولی LDV نه. LDV هم مثل ModelSim بطور کامل Tcl را حمایت می کند. دقت کنید که از آنجا که توسعه دهنده اصلی Verilog خود شرکت Cadence است معمولا جدید ترین قابلیت ها ابتدا در LDV اضافه می شوند. Verilog PLI اینجا هم وجود دارد.

۴.۳. ابزارهای سنتز

مهم ترین مرحله برای ساختن هر IC منطقی که از زبانهای توصیف سخت افزار برای طراحی آن استفاده شده است، مرحله Synthesis است. در این مرحله کد HDL تبدیل به مجموعه گیت های مناسب می شود.

نرم افزار FPGA Express و FPGA Compiler II : FPGA Express محصول شرکت Synopsys یک Synthesizer آسان برای استفاده است. برنامه دارای یک Toolbar است که تمام کارها را می توان از طریق آن انجام داد. هیچ نیازی به منو ها نیست. اولین دکمه روی Toolbar هنگامیکه می خواهیم یک پروژه جدید را آغاز کنیم به کار می رود. مثلا فرض کنید بخواهیم همان four_bit_mux را سنتز کنیم. ابتدا New Project را می زنیم. از ما اسم پروژه و محل آن را می خواهد. به دایرکتوری D:\test می رویم. و سپس تایپ می کنیم synth_1. یعنی اسم پروژه را synth_1 انتخاب می کنیم. سپس از ما لیست فایلهایی را که باید

سنتز شوند می خواهد، دقت کنید فایل مربوط به test fixture را نباید اضافه کرد، آن فایل فقط برای تست کردن بود. پس فقط four_bit_mux.v را انتخاب می کنیم. توجه کنید که می شد چند فایل را انتخاب کرد ولی فعلا طرح ما فقط همین یک فایل است. FPGA Express فایل را آنالیز می کند تا مطمئن شود که فایل از لحاظ syntax درست است. سپس در پنجره پروژه روی آن یک تیک می زند. حال روی علامت مثبتی که پهلوی اسم فایل هست کلیک کنید تا اسم مدولهای داخل فایل نشان داده شود. در داخل این فایل Verilog فقط یک مدول با نام four_bit_mux وجود دارد. آن را انتخاب کنید و هم زمان با انتخاب کردن آن، به دکمه هایی که روی Toolbar وجود دارد دقت کنید. یکی از آنها پررنگ می شود، روی همان کلیک کنید. یک پنجره ظاهر می شود در این پنجره شما می توانید انتخاب کنید که سنتز برای کدام FPGA از کدام شرکت انجام شود. مثلا Xilinx و Spartan2 و 2S15 را انتخاب کنید. با فشار دادن OK عمل سنتز و optimize شروع می شود. حال می توانید حاصل را در سمت راست پنجره پروژه مشاهده کنید. در بخش سمت راست پنجره روی four_bit_mux- Optimized کلیک کنید. همزمان به Toolbar دقت کنید. یک دکمه دیگر فعال می شود. این دکمه است که فایل EDIF را برای شما می سازد. فایل EDIF استانداردی پذیرفته شده بین تمام شرکت های تولید کننده مدارات logic است و حاوی اطلاعات مربوط به چگونگی وصل شدن گیت ها به هم و نوع گیت هایی که استفاده شده می باشد. در واقع یک Netlist از مدار را به همراه خود دارد. این فایل ورودی مرحله بعد یعنی implement است. پس روی دکمه مربوط در Toolbar کلیک کنید و سپس OK را بزنید. FPGA Express برای شما یک فایل با پسوند EDF در دایرکتوری synth_1 که دایرکتوری پروژه بود می سازد. حال می توانیم به مرحله Implement برویم.

FPGA Compiler II نیز محصول Synopsys است و در واقع با FPGA Express فرقی ندارد. همیشه تکنولوژی و پیشرفت های جدیدی که در برنامه سنتز کننده داده شده به FPGA Compiler اضافه می شود. بعد از اینکه تست های کافی روی آنها به عمل آمد، این امکانات به FPGA Express هم اضافه خواهد شد. محیط آن کاملا شبیه FPGA Express است. در زمان اصلاح دوم این متن Synopsys اعلام کرده که دیگر محصولی به اسم FPGA Express تولید نخواهد کرد و فقط FPGA Compiler تولید خواهد شد.

نرم افزار Leonardo Spectrum : Leonardo Spectrum محصول شرکت Exemplar است که خود جزئی از شرکت Mentor Graphics می باشد. با Leonardo هم می شود همان کار FPGA Express را انجام داد. Leonardo نسبت به FPGA express ، دارای تنظیمات بیشتری است. سه پنجره اصلی در آن وجود دارد، اولی روند عمومی کار و مراحلی که تا تولید شدن فایل EDF باید طی شوند را نمایش می دهد، در دومی می توانیم دستور وارد کنیم و اگر خواستیم تنظیم خاصی انجام دهیم آن را وارد کنیم. و سومی پیام هایی است که Leonardo برای ما می فرستد. حسن Leonardo نسبت به FPGA Express در آن است که command shell آن (پنجره دوم) در همان محیط GUI اصلی قرار دارد. برای Express اینطور نیست و shell به صورت یک برنامه جدای از GUI قرار گرفته است. در آزمایشهایی که ما برای Virtex-II بین FPGA express نگارش

FE2000-11 3.5 و Leonardo نگارش 2001.1a انجام دادیم ، حاصل سنتز شده طرح توسط Express با فرکانس بالاتری نسبت به حاصل سنتز Leonardo ، کار می کرد. در زمان اصلاح سوم این متن Mentor دیگر نرم افزاری به اسم Leonardo Sprectrum تولید نمی کند. تولید آن متوقف شده و به جای آن نرم افزار precision synthesis ارائه می شود. که ظاهرا قرار است یک نوع physical synthesizer باشد. در مورد physical synthesizer ها در ادامه صحبت خواهد شد.

نرم افزار Sinplify : Sinplify تولید شده توسط شرکت Synplicity بهترین و باهوشترین ابزار سنتزی است (ابزار سنتز معمولی ، نه physical synthesizer) که وجود دارد. استفاده از این نرم افزار هر روز بین طراحان متداول تر می شود. کار کردن با آن آسان است و قابلیت هایی دارد که بقیه ابزارهای سنتز ندارند.

۴.۴. سنتز و بهینه سازی فیزیکی

وقتی ابزار سنتز می خواهد برنامه Verilog ما را تبدیل به گیت کند، معمولا چندین راه مختلف و چندین مدار مختلف را پیش رو دارد. وقتی می خواهیم یک شمارنده 5 بیتی بسازیم به حالات مختلفی می توان این کار را انجام داد. ولی کدام حالت از بقیه بهتر است؟ این بستگی به خواست استفاده کننده دارد، ممکن است او بخواهد کمترین تعداد سلول روی IC مصرف شود، یعنی مدارش کمترین مساحت ممکن روی IC را اشغال کند. یا ممکن است بخواهد مدارش با بیشترین سرعت ممکن کار کند و اصلا برایش اهمیتی نداشته باشد که چه سطحی و چه تعداد سلول روی IC اشغال خواهد شد. و یا ممکن است بخواهد حالت متعادل را رعایت کند. بنابراین هنگامیکه عمل Optimization قرار است انجام شود، استفاده کننده باید تعیین کند که این کار برای Area انجام شود و یا برای Speed. در مدارهای مخابراتی که FPGA کاربرد بسیار زیادی در آنها دارد، اهمیت چندانی ندارد که Area چقدر باشد، بلکه آنچه مهم است سرعت است.

حالا فرض کنید قرار است Optimization بر اساس سرعت انجام شود، معیار نرم افزار سنتز برای اینکه کدام مدار سریع تر است، چیست؟ نرم افزار سنتز دارای کتابخانه هایی می باشد که در آنها برای هر FPGA نوشته شده این IC چه نوع گیت هایی دارد و تاخیر هر کدام چقدر است. پس برنامه سنتز می تواند برای هر کدام از مدارها بیشترین تاخیر را با جمع زدن تاخیر گیت ها روی طولانی ترین مسیر logic که در هر مدار وجود دارد پیدا کند. حالا آن مداری انتخاب می شود که کمترین تاخیر را داشته باشد.

ولی آیا واقعا کار، درست و با توجه به همه جوانب انجام شده؟ آیا مداری که انتخاب شده واقعا بهترین تاخیر را دارد؟ ابزار سنتز این مدار را انتخاب کرد چون حاصلجمع تاخیر گیت های آن مینیمم بود، در حالیکه الزامی نداریم، بعد که این مدار روی FPGA پیاده شد، باز هم همینطور باشد. برای درک بهتر یک مثال می زنیم: عناصر روی FPGA باید با سیم به هم ارتباط یابند، این سیمها (منابع Routing) خودشان می توانند تاخیر های قابل ملاحظه داشته باشند. ابزار سنتز هنگامیکه عمل سنتز را انجام می داد هیچ اطلاعی در مورد تاخیر این سیم ها نداشت. در حالیکه برای انتخاب بهترین حالت ممکن لازم است اینها هم در نظر گرفته شوند.

یک Physical Synthesizer کارش این است که عمل سنتز را نه فقط با توجه به تاخیر گیت ها، بلکه با توجه به تمام تاخیر های ممکن ، با توجه به حرارتی که در هر ناحیه از IC تولید می شود با توجه به وضعیت سیم های اتصالی، نحوه قرار گرفتن Cell ها روی device و خلاصه با توجه به واقعیت فیزیکی IC ای که قرار است این طرح روی آن پیاده شود، انجام می دهد. یک Physical synthesizer واقعی این توانایی را دارد که اعمال سنتز و Placement و Routing را همزمان انجام دهد. یعنی همینطور که دارد کدها را سنتز و بهینه سازی می کند، همزمان Placement را هم انجام می دهد و تاخیر ها را اندازه می گیرد، اگر خوب بود خلاص می کند و اگر نه ، کار را دوباره به صورت دیگری انجام می دهد. پس خروجی نهایی یک Physical Synthesizer واقعی فایلی است که طرح Place & route شده (یعنی نهایی ترین چیزی که باید برود روی IC) را در خود دارد. قدیمها عمل Syntheis جدای از Place & route انجام می شد. مثلا برای ASIC روند به این صورت بود که ابتدا مدار را با Design Compiler سنتز می کردند و سپس با یک ابزار Place & route مثلا Apollo از شرکت Avanti و یا Dracoulla از Cadence مدار نهایی IC با توجه به تکنولوژی مورد نیاز تولید می شد.) ولی یک روز ، یک سیبه خورد تو کله یک آقاهه، گفت من چرا این کار رو انجام بدم؟ می یام سنتز و placement و routing رو باهم انجام می دم. به این ترتیب راحت می تونم بفهمم چه جوری مدار رو ستنز کنم که موقع place & route بهترین نتیجه به دست بیاد. برای طراحی ASIC بعد از این اتفاق دو شرکت Synopsys و Cadence شروع کردند به تولید Physical Synthesizer های خودشان :

Synopsys نرم افزار Physical Compiler را تولید کرد. و Cadence یک نرم افزار به اسم : Phsysically knowledgeable system یا خلاصه PKS. هرکدوم از اینها برای خودشون یک پا نرم افزارند. وقتی می رید داخلش احساس می کنید به یک سیستم عامل جدید وارد شدید. به هر حال کاری که اینها انجام می دهند بسیار پیچیدس. کامپیوتری که این نرم افزارها روش نصب می شند، به راحتی می تونه چندید گیگ رم داشته باشه و چند تا پردازنده توش موازی کار کنند. قیمت استفاده یک ساله از Physical Compiler 350 هزار دلار است. و Physically knowledgeable system یا PKS از Cadence با هزینه یک ساله 500 هزار دلار. خوب با این قیمت ها کی می تونه اینها رو بخره! این نرم افزارها مطلقا تحت سیستم عامل ویندوز منتشر نمی شوند. نگارنده اطمینان دارد که Physical Compiler تحت سیستم عامل Linux عرضه شده (علتش آن است که نگارنده افتخار پیدا کرده Physical Compiler را روی سیستم خودش داشته باشد.) ولی در مورد PKS چیزی نمی دانیم. از لحاظ قانونی تا زمانی که از این نرم افزارها فقط برای امور آموزشی استفاده شود، کار کردن با این نرم افزارها منعی ندارد. دقت کنید که این دو نرم افزار مخصوص طراحی ASIC می باشند. مثلا پردازنده گرافیکی GForce2 از شرکت Nvidia که شامل 29 میلیون ترانزیستور است، با استفاده از Physical Compiler طراحی شده است. Texas Instruments برای ساختن پردازنده هایی که در گوشی های مبایل خود از آنها استفاده می کند و با فرکانس اقلا 140 مگاهرتز باید کار کنند از همین نرم افزار استفاده می کند.

اما در حوزه FPGA کار به سادگی ASIC نخواهد بود. اینجا ابزار Place & route تنها در دست شرکتی است که FPGA را تولید می کند. مثلا Xilinx نرم افزار ISE را تولید می کند که هسته آن برنامه ساده ای است به اسم par (البته نه همچین ساده.) که عمل Place and route را انجام میدهد. پس شرکت هایی که می خواهند برای FPGA ابزار سنتز فیزیکی درست کنند دچار مشکل خواهند شد، چون Place & route دست آنها نیست. Synplicity نرم افزاری به اسم Amplify ساخته که یک Physical Synthesizer مخصوص FPGA است. Synplicity این کار را با همکاری نزدیک Xilinx که تولید کننده خود FPGA ها است، انجام داده. این نرم افزار قابلیت های جالبی دارد، مثلا شما می توانید یک ناحیه روی FPGA را مشخص کنید و به آن بگوئید: برنامه Verilog من را برای قرار گرفتن در این ناحیه بهینه کن. Amplify می تواند با دقت خوبی تاخیر ها را تخمین بزند. یعنی در هنگام سنتز تقریبا می تواند آنچه را که موقع Place & route رخ خواهد داد، حدس بزند. اینجا وضع به خوبی ASIC نیست ولی به هر حال کاچی بعض هیچی! شما به Amplify می گویید که مدار شما را برای قرار گرفتن در کدام ناحیه FPGA بهینه کند. طبیعتا روشی که Amplify برای سنتز و بهینه سازی مدار اتخاذ می کند با توجه به محل مدار در FPGA فرق خواهد کرد. Amplify می تواند طرحی را که یک بار Implement شده (فایل نهایی طرح پیاده شده را) بگیرد، و سنتز را با توجه به آن دوباره طوری انجام دهد که تاخیرها در مدار کمتر شده و بهبود یابند.

طبق آزمایشات ساده ای که انجام شد (اصلاح دوم متن june 2002) نشان می دهد که Leonardo spectrum از لحاظ توانایی برای سنتز مدارهایی با فرکانس بالای ۱۲۵ مگاهرتز از همه ضعیف تر عمل می کند. (شاید هم این نتیجه گیری غلط و به خاطر ضعف دانش ما در مورد Leonardo باشد.) بین FPGA Express و Synplify هنوز جدال وجود دارد. هردو مدارها را به نحو مناسبی سنتز می کنند. هرچند گزارش هایی که Synplify می دهد بسیار دقیق تر از Express هستند. در نهایت شاید بتوان گفت در حال حاضر استفاده از Amplify بهترین حالت ممکن باشد.

۴.۵. ابزارهای پیاده سازی

می رسیم به آخرین مرحله انجام یک پروژه ساده با FPGA. مرحله ای که در آن فایل EDIF سنتز شده دریافت می شود، و مدار مربوط به آن روی FPGA پیاده می گردد. نرم افزارهای مربوط به این بخش را فقط شرکتی که FPGA را تولید می کند، می سازد. Xilinx هم برنامه هایی را که برای پیاده سازی مدار روی FPGA و بعد شبیه سازی نتیجه کار به کار می روند به صورت یک مجموعه برنامه در یک محیط مجتمع ارائه می دهد. در واقع 3 فایل اجرایی مهم وجود دارد که کل کار به عهده آنهاست و ما در این جا آن سه فایل اجرایی اصلی و عمل هرکدام را به صورت مختصر بررسی می کنیم:

Translation : برنامه NGDBuild.exe کارش این است که فایل EDF را می گیرد و از روی آن یک فایل NGD می سازد. این فایل NGD منبع تمام کارهای بعدی خواهد بود. تمام اطلاعاتی که برای مراحل بعد لازم

است، از جمله محدودیت ها، محل پایه ها و ... در این فایل قرار می گیرد. از طرفی در بعضی از روشهای طراحی کل طرح را باهم و به صورت همزمان سنتز نمی کنند بلکه آن را جدا جدا سنتز می کنند. بنابراین طرحی که قرار است پیاده شود 2 یا بیشتر فایل EDF پیدا می کند، NGDBuild اینها را به هم می چسباند و یک فایل واحد از روی آن می سازد. به این کار design expansion می گویند. (پس اگر مثلا Core یا Hardware macro وجود داشته در این مرحله از آنها استفاده می شود.)

Mapping : آنچه که ابزار سنتز به صورت فایل EDF میدهد در واقع جواب این سوال است که مدار منطقی که کار مورد نظر را برای ما انجام می دهد، باید چه ترکیبی از کوچکترین عناصر سازنده FPGA یعنی LUT ها، بافرها، بلوک های رم ، ... باشد؟ حال لازم است جواب این سوال هم تعیین شود: کدامیک از LUT ها در یک Slice قرار بگیرند؟ LUT ها به چه نحو بین Slice ها تقسیم شوند؟ کدامیک از رورودی خروجی های مدول از ثباتهای موجود در IOB استفاده کنند و کدامیک نکنند؟ این بخش از طرح باید روی Ram block پیاده شود یا نه؟

جواب تمام این سوالها در مرحله map داده می شود. برنامه map.exe فایل NGD مرحله قبل را می گیرد و یک فایل NCD به ما می دهد. دقت کنید هنوز برای هیچ کدام از عناصر فایل NGD روی FPGA محل مشخص در نظر گرفته نشده بلکه دسته بندی LUT ها و ثباتها تعیین شده است. فایل NCD حاصل تمام عناصری که عضو مدار هستند را با خود دارد ولی به صورت Unplaced و Unrouted.

Placement : در این مرحله محلی برای هرکدام از عناصر موجود در فایل NCD روی FPGA در نظر گرفته می شود. حاصل کار باز هم یک فایل NCD خواهد بود منتهی این دفعه عناصر موجود در فایل به صورت Placed شده هستند. هنوز سیم های بین عناصر Unrouted می باشند و از منابع routing استفاده نشده است. برای Placement و اینکه بهترین فرکانس کاری ممکن برای مدار نهایی به دست آید، الگوریتم هایی وجود دارد که چندان دقیق و بهینه نیستند. ممکن است از چند بار Placement متوالی پشت سر هم یکی خیلی بهتر از بقیه در آید. این تقریبا یک امر شانسی است. یکی از روشهای افزایش فرکانس همین است که چندین بار پی در پی مراحل Placement و Routing انجام شوند و آنکه فرکانس بالاتری می دهد به عنوان حاصل انتخاب شود. به این کار Multipath Place and route می گویند.

Routing : آخرین و تقریبا می توان گفت مهمترین مرحله کار routing است که طی آن عناصری که روی FPGA در مکان های مشخصی در ram block ها ، Slice ها و ... قرار دارند با سیم و با استفاده از منابع routing موجود روی FPGA به هم وصل می شوند. نتیجه یک routing خوب یک فرکانس عملکرد خوب است، اگر routing نتواند با موفقیت به پایان برسد، کل کاری که تابحال انجام شده بیهوده و حتی ممکن است مجبور شویم قسمت هایی از برنامه Verilog اصلی را تغییر دهیم. هر چه الگوریتم های routing بهتر عمل کنند، کار بهتر انجام خواهد شد. فایل اجرائی که اعمال Place and route را انجام می دهد par.exe نام دارد.

تمام مراحلی را که در بالا گفتیم می توان در محیط های مجتمعی که Xilinx در اختیار استفاده کنندگان قرار می دهد، انجام داد. هر چند همواره Integration ما را از جزئیات کار دور خواهد کرد، کار کردن با این محیط ها آسان تر از مستقیما دستور وارد کردن است. دو محیط در حال حاضر وجود دارد: یکی design manager که عمل implement را انجام می دهد و دیگری یک محیط مجتمع به اسم Project Navigator مربوط به برنامه Foundation ISE نگارش 3.3 و یا بیشتر، که در آن کل کار از ابتدایی ترین مرحله نوشتن کد Verilog تا آخر را می توان انجام داد. این کار را سریع و راحت می کند، هرچند برای طرحهای بزرگ و طرحهایی که سرعت در آنها مهم است، بهتر است از این محیط استفاده نشود تا تسلط بیشتری بر این که چه دارد می گذرد داشته باشیم. در زمان اصلاح سوم این متن ISE 5 در اختیار ماست.

۴.۶. برآورد سرعت مدار

حال فرض کنید مدار را Implement کردیم. می خواهیم بدانیم حداکثر فرکانس کاری مدار چقدر است، از لبه بالا رونده تا ظاهر شدن داده در خروجی چقدر طول می کشد، setup time های مربوط به ورودی فلیپ فلاپ ها چقدر هستند و ... نرم افزار timing analyzer این کار را برای ما انجام می دهد. تمام تاخیرهای ورودی و خروجی و تاخیرهای مدارهای داخلی محاسبه می شوند و به صورت گزارش به استفاده کننده داده می شوند.

۴.۷. تخصیص دهی ناحیه به هریک از مدول ها

فرض کنید شما بخواهید یک بخشی از مدولهایتان در یک ناحیه خاصی از FPGA قرار بگیرد. به عبارت دیگر نمی خواهید به ابزار Place&route اجازه دهید هرجایی که می خواهد عناصر را قرار دهد، بلکه می خواهید برای آن یک ناحیه مشخص کنید. این کار را می توان با Floorplanner کرد. حاصل عمل Floorplanning یک فایل fnf و mpf خواهد بود که بعدا وقتی دوباره implement می کنیم اینها به عنوان محدودیت هایی برای قرار دادن عناصر در خواهند آمد که باید از آنها تبعت شود. Floorplanning برای طرحهای بزرگ خیلی کاربرد دارد. اگر در حالت UCF Flow از Floorplanner استفاده کنیم آنوقت حاصل کار یک فایل با پسوند UCF خواهد بود.

۴.۸. اعمال محدودیت روی مدار

فرض کنید می خواهید از یک بلوک ram خاص روی FPGA استفاده کنید. یا می خواهید یکی از پورت های مدول شما روی مجموعه مشخصی از پایه های FPGA برود. یا حتی می خواهید فرکانس کاری مدار شما از مقدار مشخصی بالاتر باشد. این محدودیت ها را با نرم افزار Constraints Editor می توان تعیین کرد. فایل اصلی که Constriant Editor دریافت می کند، فایل NGD است. حاصل کار در یک فایل UCF مخفف User Constraints File ذخیره می شود. فایل UCF یکی از فایلهایی است که NGDBuild به عنوان ورودی می خواند.

۴.۹. مشاهده مدار نهایی

این برنامه فایل نهایی NCD را دریافت می کند و آن را به شما نشان می دهد. در واقع به شما نشان می دهد که بعد از Placement و routing داخل FPGA چه خبر است. و عناصر کجا قرار گرفته اند و چه جوری به هم وصل شده اند. با این نرم افزار کارهای بسیار زیاد و متنوعی می توان انجام داد. در واقع این را می توان به جای تمام 3 برنامه ای که در بالا ذکر شد استفاده کرد. البته انعطاف پذیری و قابلیت فوق العاده معمولا همراه با سختی کار است. کار کردن با FPGA Editor کمی دشوار است. یعنی خلاصه تجربه می خواهد و حوصله. البته این هیچ وقت جای اونها رو نمی تونه بگیره.

۴.۱۰. شبیه سازی مدار نهایی

فایل NCD نهایی را می توان تبدیل به یک فایل Verilog نمود. این دفعه برنامه Verilog واقع سطح پایین است و فقط عناصر و ارتباط آنها با هم در آن مشخص شده. همراه با فایل Verilog یک فایل SDF مخفف Standard Delay Format هم ساخته می شود که تاخیر ها را در خود دارد. این دو فایل را به ModelSim می بریم. و آنجا دوباره مدار را شبیه سازی می کنیم. منتهی این دفعه تاخیر ها صفر نیست و همه چیز بسیار نزدیک به آن چیزی است که در واقعیت اتفاق خواهد افتاد.

۴.۱۱. قرار دادن طرح نهایی روی پی روم

حال که همه چیز آماده است می توان فایل NCD را تبدیل به یک فایل bit کرد و بعد bit فایل را ریخت روی یک PROM و PROM را به FPGA وصل کرد. هر زمان که برق وارد مدار می شود FPGA با توجه به PROM خود را Configure خواهد کرد و به صورت مدار دلخواه عمل خواهد کرد.

۴.۱۲. بررسی عملکرد واقعی مدار با کامپیوتر

برنامه و سخت افزار خاصی وجود دارد که با آن می توان سیگنالهای واقعی را روی صفحه کامپیوتر دید. یک بورد خاص که FPGA ی مورد نظر ما روی آن قرار دارد و مثلا به پورت USB کامپیوتر وصل می شود، همراه با یک نرم افزار که هر زمان با بورد ارتباط برقرار می کند و وضعیت سیگنالها را روی صفحه نمایش می دهند، عناصر اصلی یک hardware debugger می باشند.