فهرست

فصل اول:انواع مدارهای منطقی قابل برنامه ریزی

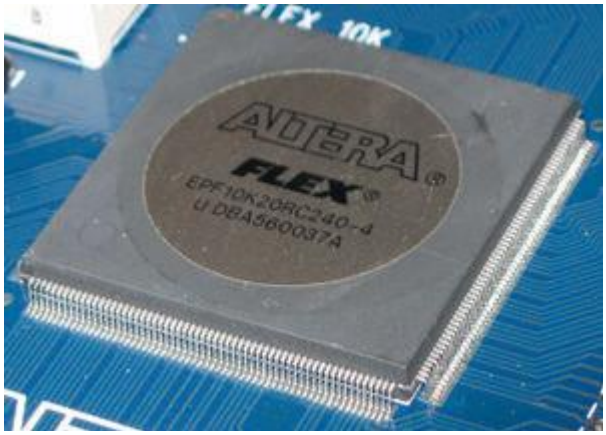
فصل دوم:معرفی FPGA

فصل سوم:ساختار FPGA

فصل چهارم:ایجاد یک پیکر بندی جدید درFPGA

فصل ششم: زبان توصیف سخت افزار VHDL

فصل هفتم:روند طراحی یک مدار قا بل برنامه ریزی بر اساس Xilinx foundation



مقدمه:

سالها پیش که طراحی دیجیتال پا به عرصه ی وجود نهاد وIC های استانداردی چون گیتها ، فلیپ فلاپ ها ، لچ ها شمارنده هاو... ساخته شدند و بعدها به تدریج پردازنده هایی با قدرت محدود که اولین کامپیوتر های شخصی بر اساس آنها طراحی شده بود دنیای دیجیتال را به وجود آوردند، تصور روزی که فاصله ی سخت افزار و نرم افزار به حد کنونی برسد به طوری که تمام مرزهای طراحی را در نوردیده و سخت افزار به نرمی و انعطاف پذیری درآید بسیار دشوار بود

اما بعد ها با طراحی حافظه های قابل برنامه ریزی دوباره و فن آوری حافظه های پایای با قابلیت برنامه ریزی و پاک شدن(EPROM) وPAL آرایه های منطقی قابل برنامه ریزی و سرانجام فن آوری آرایه های سوئیچ های فیوزهای قابل برنامه ریزی چند باره ، انقلابی نوین را درعرصه طراحی دیجیتال به وجود آورد.

فصل اول:

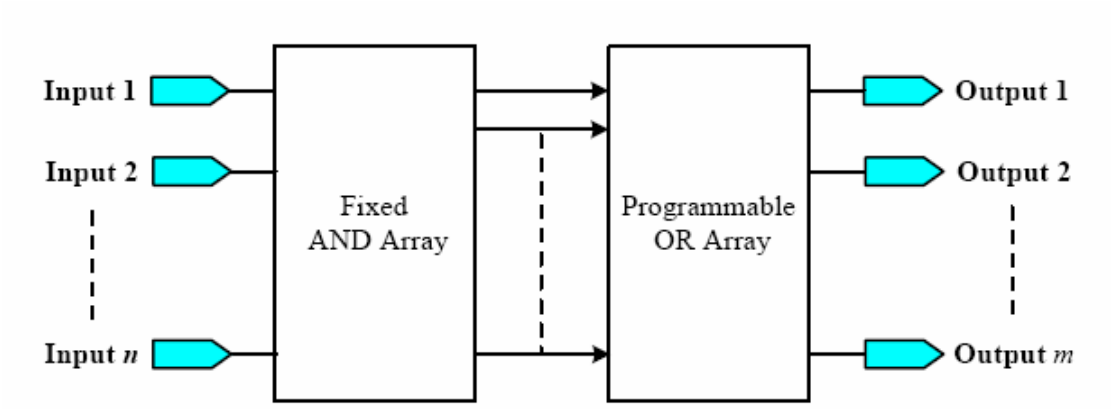
1-1انواع مدارهای منطقی قابل برنامه ریزی:

در گذشته طراحی سیستم های دیجیتا ل با مجمو عه ای از گیتها وIC های استاندارد انجام می شد ولی امروزه طراحی سیستم ها با استفاده از مدار های منطقی برنامه پذیر ،تغییرات زیادی نموده است. این مدار ها باظرفیتی حدود 100 تا چند هزار گیت در یک مدار مجتمع دیجیتال قرارVLSI می گیرند.

مدارهای منطقی بر نامه پذیر دارای انواع محتلف PLA، PAL، SPLD، CPLD، FPGA می باشند که هر کدام دارای ظرفیت،سرعت،وویژگی هایی می باشندو در اینجا به معرفی هر کدام می پردازیم.

:PROM

اولین تراشه های قابل برنامه ریزی که به بازارعرضه شدند،PROM ها بودند که خطوط آدرس به عنوان ورودی وخطوط داده به عنوان خروجی این تراشه ها بودند.PROMها شامل دسته ای از گیتهای AND غیر قابل برنامه ریزی است که به صورت رمز گشا بسته شده اند ونیز یک آرایه OR قابل برنامه ریزی است.ازآنجایی که PROM دارای قابلتهای لازم برای پیاده سازی دارهای منطقی نمی باشداز این تراشه ها بیشتر به عنوان حافظه های قابل برنامه ریزی استفاده می شود. نمودار قالبی PROM ها در شکل 1-1 نشان داده شده است.

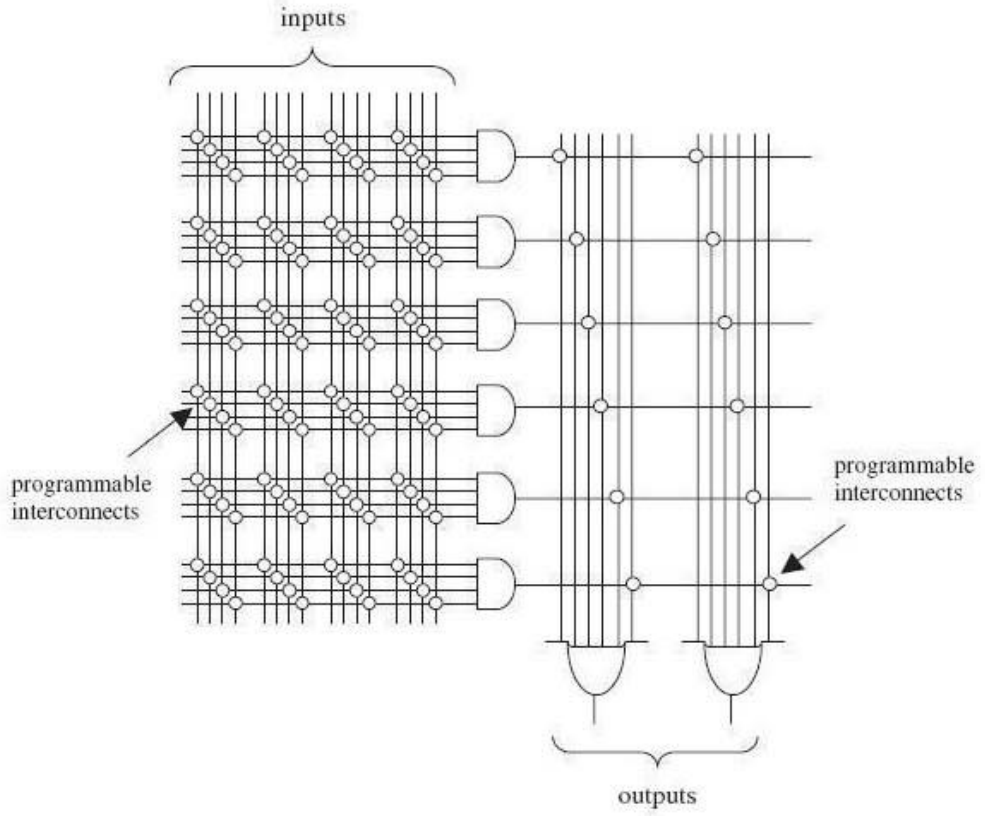


شکل ۱-۱: نمودار قالبی PROM (حافظه فقط خواندنی قابل برنامه‌ریزی)

:PLA

قطعات قابل برنامه ریزی(PLD) برای اولین بار در اواسط دهه هفتاد میلادی به وجود آمدند ایده اولیه این بود که قطعه های بسازند که بتواند مدارهای ترکیبی را حمایت کند بر خلاف میکرو پروسسورها که برنامه ها را با یک سخت افزار ثابت اجرا می کنند PLDها برنامه ریزی را در سخت افزار خود انجام میدهند به عبارتی دیگر pLDها قطعاتی هستند که می توانند برای پیاده سازی مدارهای مختلف مطلوب تغییر شکل دهند اولین نوع از این قطعات PLAها بودند که از دوآرایه قابل برنامه ریزی سازمان یافته بود یکی آرایه AND ودیگری

آرایهOR بود که به صورت عمودی به هم متصل شده بودند و ساختار آن را در شکل زیر می بینید این قطعات فقط قا بلیت پیاده سازی مدارهای ترکیبی را داشتند برای برطرف کردن این نقیصه یک فلیپ فلاپ به آنها اضافه شد که به این ترتیب می توانستند مدارهای ترتیبی را نیز پیاده سازی کنند.

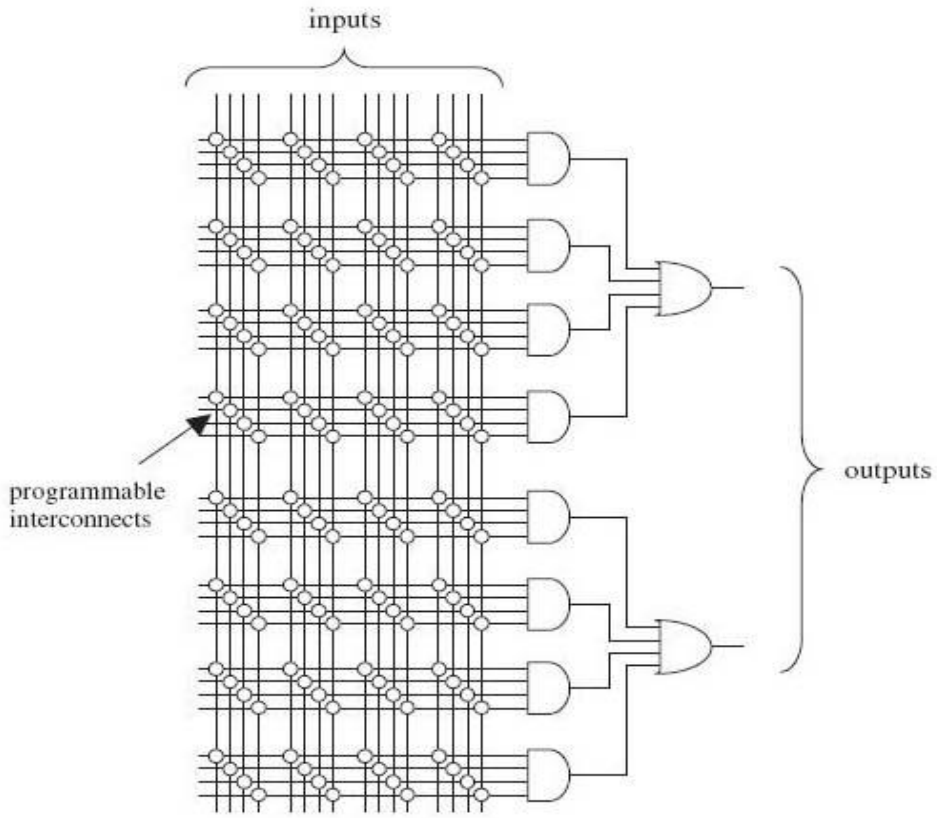


شکل1-2: ساختارداخلی PLAها

:PAL

PLAها دارای دو ایراد بودند یکی اینکه سرعت آنها کم بود و دیگری قیمت نسبتا بالای آنها بود به همین دلیل PALها روانه بازار شدند که دارای یک آرایه برنامه پذیر AND ویک آرایه ثابت OR بودند که همین امر باعث ایجاد سرعت بیشتر وهمچنین هزینه ساخت کمتر می شد برای رفع عیب ثابت بودن آرایه ORسازندگان انواع مختلفی از PALها را ارائه کردندکه از لحاظ تعدادORو همچنین تعداد ورودی های گیت ORبا هم متفاوت بودند

امروزدر اکثر PALها خروجی های گیت ORبه یک فلیپ فلاپ وصل می شوند و بدین ترتیب می توان مدارهای ترتیبی با پالس ساعت را نیز با آنها پیاده سازی نمود.



شکل1-3(ساختار داخلی PAL ها):

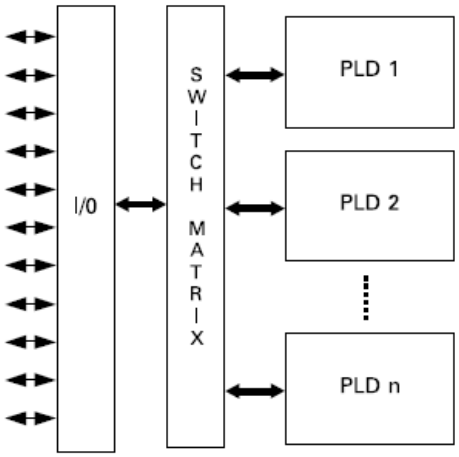
:CPLD

PAL ها IC های قابل برنامه ریزی خوبی هستند لی برای طراحی سسیستمهای پیچیده دیجیتال ممکن است چندین PAL نیاز باشد.برای این منظور مدارهای قابل برنامه ریزی CPLD طراحی شدند که از دو یا چندین بلوک منطقی و تعدادی ماکرو سل تشکیل شده اند که با سیستم های ارتباطی و سوئیچ های قابل برنامه ریزی با هم ارتباط داده می شوند.

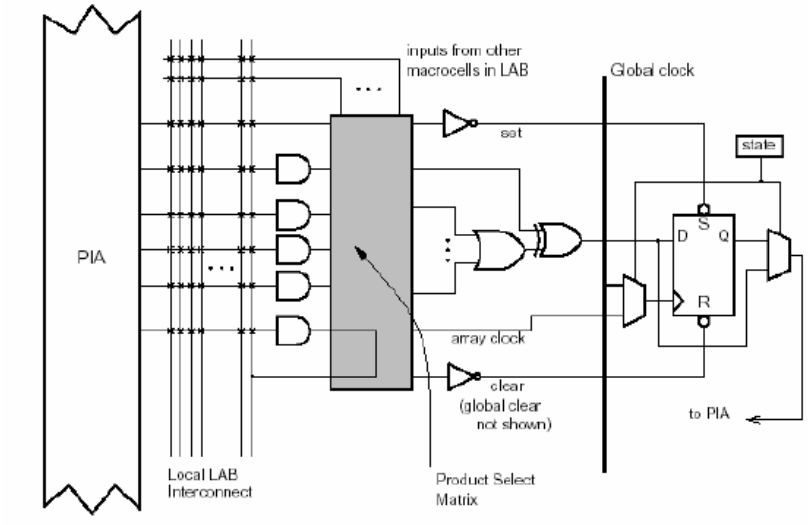
CPLD های تجاری با اندازه های مختلفی از 2 تا 100 بلوک منطقی یا SPLD و با ظرفیتی حدود 1000تا 15000 گیت ساخته شده اند.

وCPLD خانواده max9000 از شرکت ALTERA دارای ظرفیتی در حدود 12000 گیت می باشد.CPLD تاخیر کمی دز حدود نانو ثانیه دارند لذا بسیار سریع ودر حدود فرکانس 100مگا هرتز کار می کنند .

شرکت ALTERA سری های CPLD به شماره های MAX5000، MAX7000، MAX9000 را به بازار عرضه نموده استکه سری MAX7000 بیشتر به کار می رود.در MAX7000 CPLD هر بلوک منطقی ان دارای یک ماکرو سل است که ماکرو سل آن شامل آرایه های قابل برنامه ریزی AND می باشد که،خروجی آرایه AND به گیتOR وبالا خره به یک فیلیپ فلاپ منتهی می شود.



شکل4-1(ساختار کلی CPLDها):



شک5-1(ماکرو سلMAX7000 شرکت ALTERA):

شرکت XILINX نیزCPLD سریXC7000 راو نوع جدید آنXC9500 را به بازار عرضه نموده است که مشابه سریMAX7000 شرکتALTERA می باشد.

CPLD به علت سرعت زیاد وظرفیت بالا (حدود چند هزار گیت در یک IC)برای طراحی نمونه ساز های سیستم های دیجیتال،کنترل کننده های گرافیکی،شبکه های کامپیوتریLAN،قسمت های مختلف پروسسور و...به کار برده می شود.

1فصل دوم:

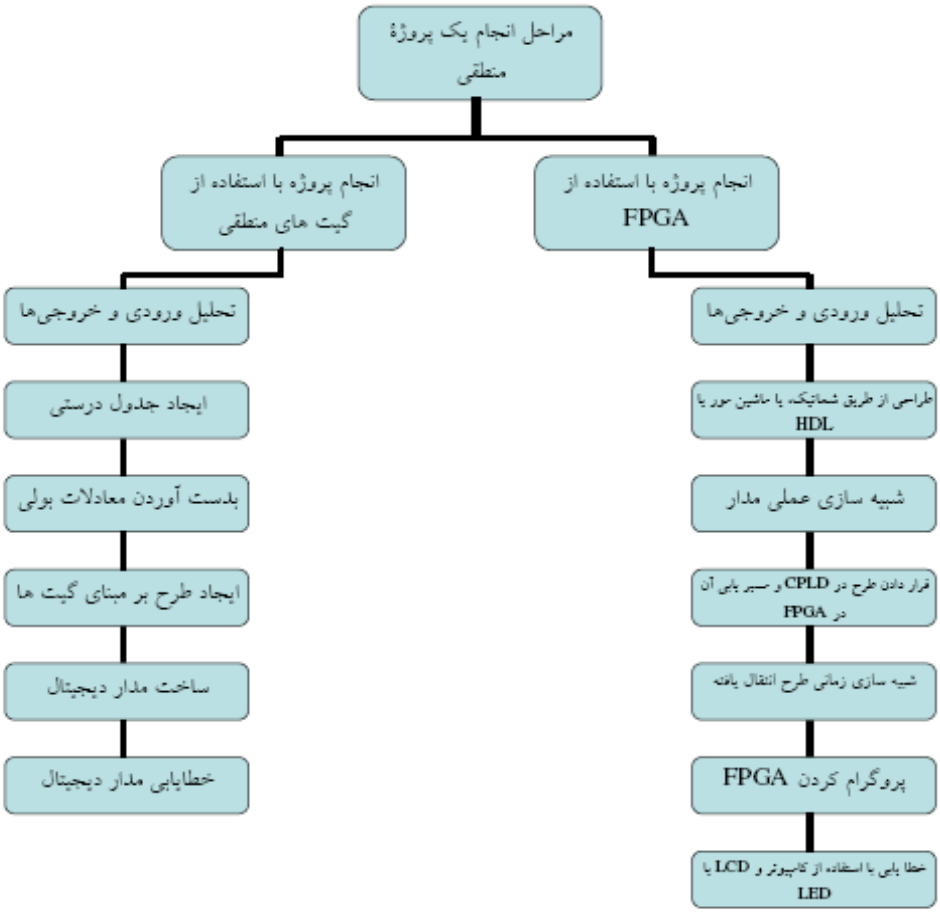
1-2 معرفی FPGA:

FPGAها Field Programmable Gate Area(آرایه های گیتی قابل برنامه ریزی میدانی)یک بوم نقاشی سفید را در اختیار طراح قرار می دهند که به او اجازه می دهد تا طراحی دیجیتال خود را آنچنان که می خواهد و با هر حجم و پیچیدگی لازم ،طراحی و سپس به جای اتخاب IC های استاندارد و جدا از هم و کنار هم قرار دادن آنها در روی یک مدار ووصل کردن آنها ازطریق یک بورد مدار چاپی(PCB) ،با استفاده از یکی از زبانهای توصیف سختافزاری نظیر VHDL ، هر یک از قطعات دیجیتالی مورد نیاز را نوشته و با وصل کردن نرم افزاری آنها ،سرانجام فایل کامپایل شده نهایی را زا طریق یک رابط سخت افزاری بر روی یک بسته سخت افزاری خام با تعداد پایه های مورد نیاز برنامه ریزی کرده و از این IC جدید "خود ساخته" استفاده کند.

شاید تا به حال مدارهای منطقی را به وسیله گیتهای or,and.xor, و... ساخته اید .برای ساخت چنین مدارهایی (از قبیل شمارنده ها وکنترل کننده ها ،و...)ابتدا باید تعریفی از مدار دردست باشد .سپس با توجه به منطق اعداد دودویی یک جدول صحت برای مدار تشکیل می شودوحالتهای مختلف مورد بررسی قرار می گیرد وحالتهای مختلف مدار مورد بررسی قرار می گیرد

سپس با توجه به جدول صحت مدار، توسط گیتهای منطقی مانند and,or,notو... طراحی می شود پس ازاین مرحله نوبت به پیاده سازی مدار بر روی برد توسط آی سی های منطقی می رسد

وهمانطور که میدانید یکی از سخت ترین ووقت گیرترین مراحل یک مدار همین قسمت است بعد ازاین مرحله نوبت به تست مدار جهت اطلاع از درستی مراحل کارکرد مداررسد اگر دریکی از مراحل قبل اشتباه کرده باشیم مطمئنا در مرحله تست مدار دچار مشکل میشویم . در صورت اشتباه در مراحل قبل باید تمام مراحل قبل را از آخر به اول یک به یک چک کنیم تا بتوانیم احتمالی موجود در نحوه بستن وسیم کشی مدارو وطراحی مدار از روی جدول صحت ودرستی جدول صحت مدار را برطرف کنیم .با توجه به مطالب گفته شده حتما به این نکته اذعان خواهید داشت که بیشترین اشتباهات در مرحله سیم کشی وبستن روی برد مدار پیش می آید.ممکن است سیمی در جایی وصل نشده باشد ویا ممکن است یک پایه به هیچج کجا متصل نباشد ویا اشتباهات مشابه اینها ... از طرف دیگر میدانییم هر چه مدار بزرگتروپیچیده تر باشد اشتباهات بیشتر وعیب یابی مشکل تر خواهد شد.اینجاست که نقش آی سی های FPGA نمایان تر می شود.آی سی ها یی که با داشتن انواع گیت هایی مختلف درون خود بسیاری از مشکلات ناشی از عیب یابی مدارهای منطقی را بر طرف کرده است.



2-1مزیتهای FPGA:

1- امکان تعریف هر یک از پایه های IC به صورت ورودی یاخروجی یا هر دو

2-امکان تعریف وضعیت عملکرد هر پایه در هنگام استفاده یا عدم استفاده.به عنوان مثال عملکرد HIGH امپدانس(Z)در هنگام عدم استفاده و یا قرار گرفتن در یک وضعیت منطقی صفر یا یک در هنگام استفاده

3- امکان تشخیص تغییرات سطوح یا لبه های پایین رونده یا بالا رونده منطقی اعمال شده به هر پایه.

4- امکان برنامه ریزی چند باره از طریق پایه های برنامه ریزی Jtag یکی از استاندارد

های برنامه ریزی(IEEE و تغییر معماری آن.

-5

امکان تغییر متناوب معمار ی داخلی با استفاده از سری های BOOTABLE که نقشه معماری آنها در یک حافظه خارجی نگهداری شده و با تغییر آدرس برنامه ریزی می توان IC را با معماری جدید BOOT کرد.

6- امکان برنامه ریزی در مدار (ISP) که این قابلیت را به وجود می آورد تا بدون اعمال تغییرات سخت افزاری و تنها از طریق پورت برنامه ریزی، معماری داخلی IC را تغییر داد,

7- محدوده گستره ای از پایه های قابل استفاده در این IC ها که از بسته های 44 پایه تا 514 پایه و حتی بالاتر با حجم گیتی داخلی متفا وت که بسته به نیاز بر اساس میزان پیچیدگی داخلی و تعداد پایه های IC را تغییر داد.

8-کاهش حیرت انگیز حجم مدار و مجتمع سازی در ابعادی تنها به مساحت چند سانتی متر مربع.

9- یکسان سازی عناصر طراحی و از میان بردن تمامی مشکلات ناشی از عدم تطابق استاندارد های مختلف.(LS,HC,s,AS,...)

10-از میان بردن تمامی نویز های ناشی از وجود قطعات مختلف و مجزا در مدار.

11- کاهش چشمگیر توان مصرفی و اتلاف توان.

12- افزایش سرعت پردازش و خطاهای انتشار به دلیل استفاده از فناوری پیشرفته و دستیابی به خطاهای انتشار تا 4 nsو فرکانس کلاک فرارتر از 178 مگاهرتز.

13- کار با دو سطح ولتاژ 5 vو 3.3 vجهت استفاده از آنها در دستگاه های قابل حمل مانند گوشی های موبایل

14- ضریب ایمنی صد در صد به دلیل عدم امکان دستیابی به محتوای داخلی و عدم توان

توصیف محتوای داخلی به د لیل انجام ساده سازی و فشرده سازی بسیار پیچیده.و بسیاری از قابلیتهای حیرت انگیز دیگر که امکان انجام یک طراحی مجتمع ،کم حجم ،بهینه و سریع را فراهم می آورد.

1-3 شرکتهای سازنده FPGA :

گر چه شرکتهای بسیاری بسته های FPGA را تولید می کنند اما از میان آنها در شرکت Alteraو Xilinx از جمله عمده ترین تولید کنندگان این محصول هستند شرکتهای Quicklogic,Lattice,Actel هم فقط تولیدات مخصوص خود را دارند.عمده تولیدات کمپانی Lattice،CPLDها می باشند همچنین FPGAهای Instant-on را هم تولید می کنند .کمپانی های QuICkLogic,Actel تولیدات ضد فیوز دارند یعنی فقط یک بار قابل برنامه ریزی هستند.

فصل سوم:

ساختار FPGA :

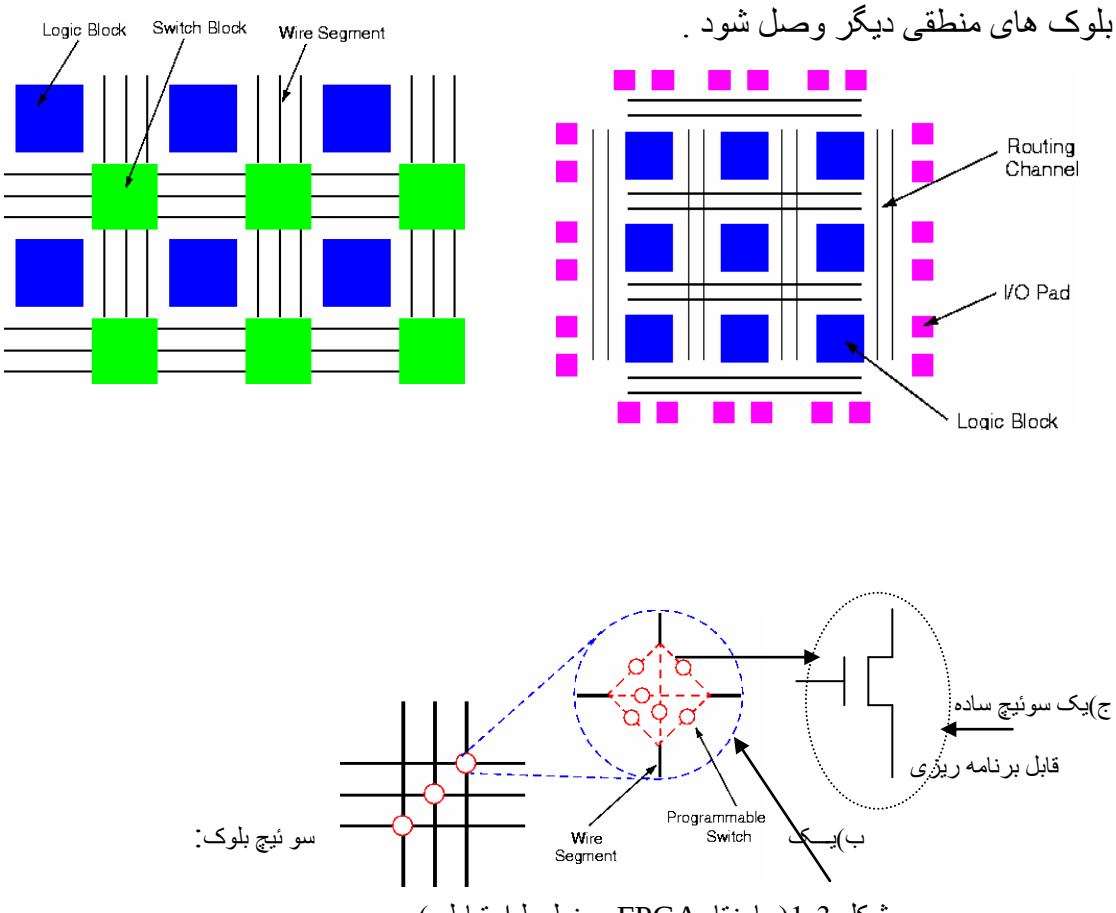
هرFPGA را به طور کلی می توان به صورت جزیره هایی مجزا در نظر گرفت که توسط شاه راه ها یی به هم متصل می شوند.به عبارتیFPGA شامل یک سری بلوکهای منطقی و نیز سیم های بین آنها می گردد.دو پد ورودی وخروجی نیز در انتهای هر یک از ردیف ها یا ستونها قرار داده شده است خطوط اتصال دهنده بین بلوکها از نظر تعداد واندازه یکسان می باشند.

به طور جز ئیتر می توان گفت هر FPGA ازعناصر مختلفی تشکیل شده است که روز به روز در حال پیشرفت می باشند مانند بلوکهای ورودی وخروجی ،بلوک های سوئیچ،بلوکهای منطقی ،بلوکهای RAM ،بلوک های ضرب کننده ،سیم های اتصال دهنده،وهمچنین در FPGAهای جدید تر امکانا تی چون میکرو پروسسورها ی متنوع ،ALU،FIFOو...

در این فصل هر کدام از این عناصر تا حدودی معرفی می شوند.

3-1ساختار کلیFPGA :

اندازه PAL ها محدود ودر حدود 200 گیت می باشد ودر ضمن پایه های خروجی آن مجموع حاصل ضرب می باشد .مدارهای قابل برنامه ریزی پر قدرت تر ، FPGA ها میباشند که از آرایه ای از بلوکها یا سلول های منطقی تشکیل شده اند که توسط خطوط ارتباطی واز طریق سوئیچهای قابل برنامه ریزی می توانند به هم متصل شوند .علاوه بر این، بافرهای ورودی وخروجی قابل برنامه ریزی برای ارباط با پایه های FPGA پیش بینی شده است شکل3-8 خطوط ارتباطی وهمچنین سوئیچهای قابل برنامه ریزیFPGAرابه طور دقیق تر نشان می دهد. همانطورکه ازشکل پیداست خروجی هر بلوک منطقی از طریق خطوط ارتباطی افقی وعمودی وهمچنین سوئیچ های قابل برنامه ریزی شکل ب می تواند به ورودی هر یک از بلوک های منطقی دیگر وصل شود .



شکل 3-1(سا ختارFPGA و خطوط ارتباطی):

بلوک سوئیچ شکل(3-1-ب) ا از تعدادی سوئیچ قابل برنامه ریزی شکل(3-1-ج) تشکیل شده است که با برنامه ریزی سوئیچهای مذکور توسط ابارهای برنامه ریزیFPGA خروجی هر سلول منطقی را می توان به ورودی سلول منطقی دیگر متصل نمود.سوئیچ های مذکور از تکنولوژی SRAMیا آنتی فیوز می باشند کهدر همین فصل مورد بحث قرارمی گیرند.

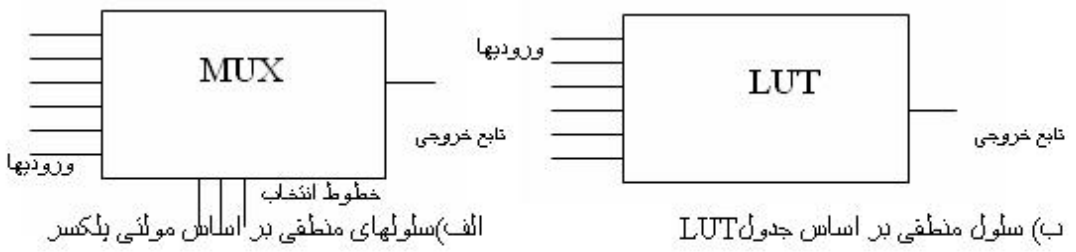
ظرفیتFPGA ها با معادل تعداد گیتهای NANDدو ورودی سنجیده می شوند. امروزه ظرفیت معمولFPGAها در حدود 20000 گیتNAND به بالا می باشدو با فرکانس حدود100 مگا هرتز کار می کنند.بلوک یا سلول منطقی می تواند :

1-از تعدادی مولتی پلکسر تشکیل شده باشد (شکل9- 1الف)

2-از جدول LUTتشکیل شود(9-1ب)

برخی FPGAعلاوه بر بلوک های منطقی دارای حافظه های RAM، ROM، بلوک های محاسباتیALU، FIFOو..نیز می باشند.

سلولهای منطقی به صورت ماتریسی در FPGAقرار گرفته اند ولی ممکن است سلولهای مذکور به شکل ردیفی نیز با شند که از طریق خطوط اتصال وسوئیچ ها به هم متصل می شوند.



شکل 3-2)اصول ساختار سلولهای منطقی:

2-3:سلولها یا بلوک منطقی قابل برنامه ریزی (LBیاLCیاCLB):

سلول های منطقی قابل برنامه ریزی برای پیاده سازی توابع منطقی با تعداد ورودی نسبتا زیاد

استفاده می شود پیچیدگی یک سلول منطقی تابع،تابع تعداد توابعی است که بر روی هر سلول قابل پیاده سازی است.همانطور که اشاره شد ساختار سلول منطقی قابل برنامه ریزی بر اساس مولتی پلکسر یا جدول LUT می باشد که در ذیل هر یک ازآنها مورد بحث قرار می گیرد.

الف) سلولهای منطقی بر اساس جدول LUT:

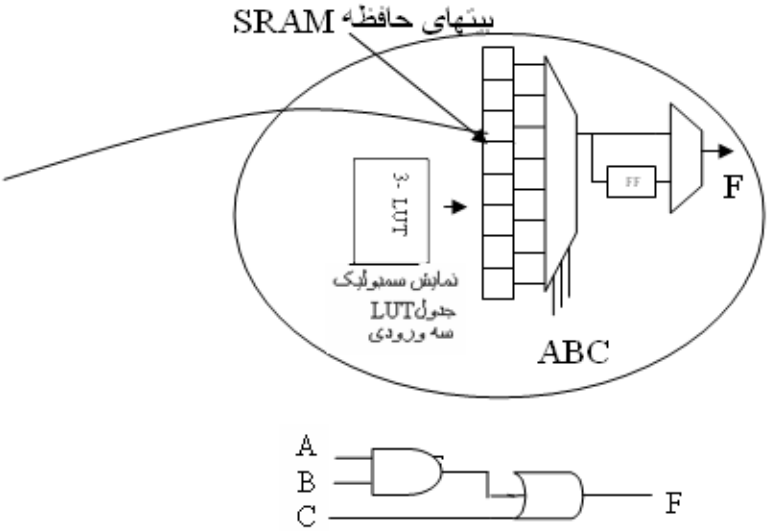
اکثر سلولهای منطقی FPGA بر اساس جدولLUT ساخته شدهاند.جدولLUT از تعدادی سلولهای حافظه SRAM ساخته می شود ،که در موقع برنامه ریزی به آن مقدار داده می شود.

در زمان کار تر کیب سیگنا لها ی ورودی ،یکی از سلولهای حافظه را انتخاب می کنند که محتوای آن ،به خروجی LUT منتقل می شود.به عنوان مثال تابع Fبر حسب سه متغیرA,B,C مطابق جدول درستی شکل زیر تعریف شده است.

مقادیر خروجی تابع F توسط ابزار برنامه ریزی FPGA ،در بیتهای خافظه SRAM شکل قرار داده می شود دراین صورت به ازاء هر یک از تر کیب های A,B,C که در ورودی

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

الف)جدول درستی تابع F
F=AB+C

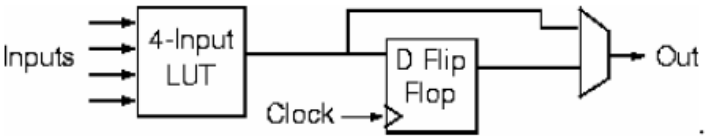


ب) پیاده سازی جدول درستی در جدول LUT سه ورودی 3-LUT با حا فظه SRAM

شکل3-3:سلول منطقی FPGA براساس جدولLUT

کنترل مولتی پلکسر قرار گیرد،یک آدرس حافظه SRAMیک بیتی انتخاب،ودر نتیجه محتوای حافظه که برابر با خروجی تابع F مطابق جدول شکل 1 می باشد به خروجی مولتی پلکسر جدولLUTمنتقل می شود.

در خروجی LUT یک فیلیپ فلاپ نیز می توان قرار داد که برای ذخیره اطلا عات می توان از آنها استفاده نمود.شکل(1-11)بدیهی است که با جدولLUT با ورودی بیشتر تاع های پیچیده تری نیز می توان پیاده نمود.

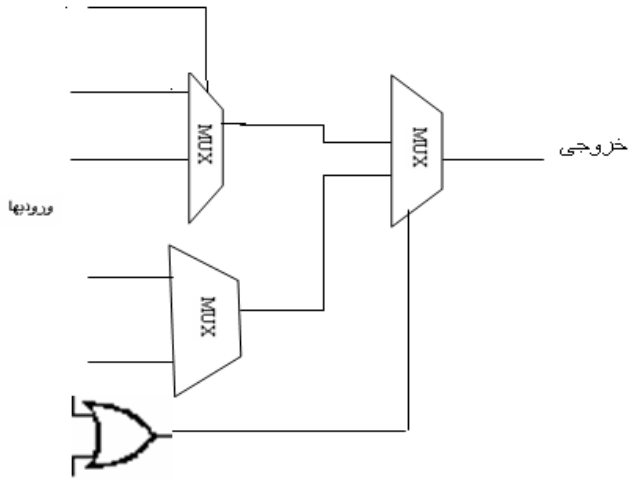


شکل(4-3) ساختار یک سلول منطقی(CLC) :

ب)سلول منطقی یر اساس مولتی پلکسر:

یکی از سولهای منطقی با مولتی پلکسر به نام سلو ل منطقی خانواده ACT-1 از شرکتACTELا ست که شامل 8ورودی ،سه مولتی پلکسر 2 به 1 ، یک گیت OR دو ورودی ویک خروجی می باشد.

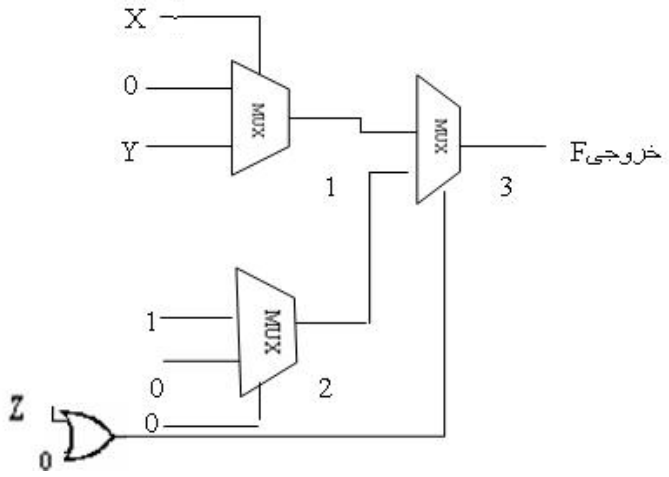
با سلول منطقی فوق می توان گیتهای AND،OR ، NAND، NOR با 2،3، یا 4 ورودی ،فیلیپ فلاپ،گیتهایXOR ویا هر تابع منطقی دیگر مانند AND-OR را پیاده سازی نمود.



شکل3-5: سلول منطقی FPGA مبتنی بر مولتی پلکسر(خانواده ACT-1 از شرکت ACTEL):

ظبه عنوان مثال اگر بخواهیم تابعF مطبق جدول درستی زیر ومعادله F=XY+Z را در سلول منطقی فوق پاده سازی کنیم ،کافیست ورودیهای مولتی پلکسر را توسط ابزار برنامه ریزی FPGA مطابق شکل قرار دهیم .

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



شکل3-6: پیاده سازی تا بع F=XY+Z با سلول منطقی خانواده ACT-1 شرکتACTEL

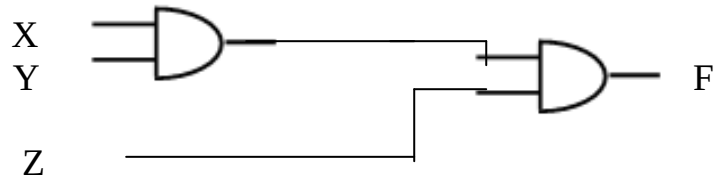
در این صورت در شکل3-6:

مولتی پکسر شماره 1: تابعAND خروجی آن یعنی تابع یعنیXYرا تولید می کند

مولت**ی پلکسر شماره 2:**خروجی آن برابر ورودی 0 مولتی پلکسر یعنی همیشه مسا وی 1 است.

مولتی پلکسر شماره 3: تابع OR را تولید می کند یعنی خروجی آن برابرXY+Z می باشد.

بـه ایـن ترتیـب بـا قـرار دادن مقـادیر 0و1وXوYوZ در ورودی سلول منطقـی مـذکور توسـط ابزار برنامه ریزی FPGA ،خروجی آن تابع یعنی F=XY+Z را مطابق شکل زیر تولید می نماید.



شکل: 3-7 پیاده سازی تابع F=XY+Z با سلول منطقی خانواده ACT-1 شرکتACTEL

لـذاردFPGA هـای شـرکتACTEL کـه تعـدادی ازایـن سـلولهای منطقـی ،بـر اسـاس مـولتی پلکسر است،می توان هر تابع منطقی ای راپیاده سازی کرد.

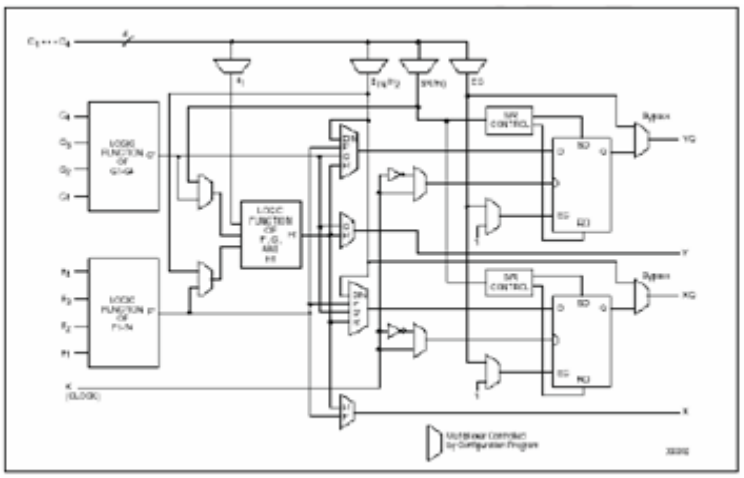
ج)سلولهای منطقی(FPGA CLB ها) شرکت Xilinx:

شرکت xilinx ،FPGAهای سری

XC2000،xc3000،xc4000,xc7000,xc5000,xc8100 وspartan .... را به

بازار عرضه نموده است که از نظر اصولی مشابه هستند

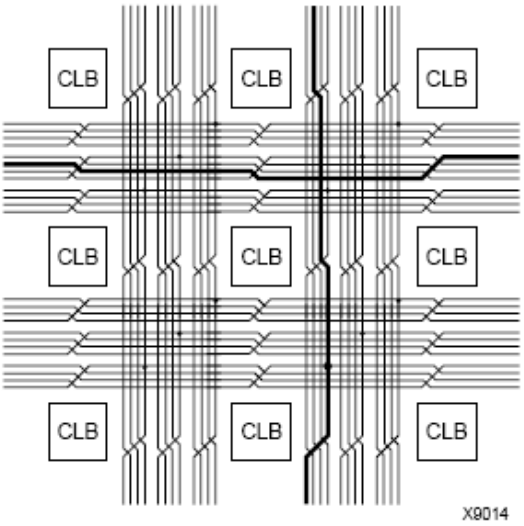
FPGAهای سری xc7000 دارای ظرفیت معادل 2000تا 15000 گیت هستند ومبتنی بر جدول LUTوتکنولوژی SRAMمیبا شند



شکل 3-8:سلول منطقی CLB ،FPGA سری شرکت Xilinx

سلول منطقی CLB سری xc4000 شامل دو جدولLUT چهار ورودی ویک جدول LUT دو ورودی وهمچنین دو فلیپ فلاپ خروجی می باشد در این CLB می توان هر تابع منطقی از جمله دستگاه محاسباتی ALU و... راپیاده سازی نمود. هر جدول LUT با یک بیت خروجی و k بیت ورودی دارای 1× 2k بیت حافظهSRAM است که کاربر با ابزار برنامه ریزی FPGA این حافظه های SRAM راطوری برنامه ریزی کند که تابع مورد نظر حاصل شود.

در FPGAهای سری XC4000 سلولهای منطقی (CLB) به صورت آرایه ای مطابق شکل 3-9قرار گرفته اند واز طریق کانالها و سیم های افقی وعمودی ارتباط آنها باهم برقرار می شود.



شکل9-3: آرایه بلوکهای منطقی(CLB) FPGA سری XC4000

سوئیچ های برنامه ریزی از نوع SRAM می باشد که هر سیم افقی را به سیم عمودی متصل می کند ،به عبارت دیگر خروجی هر CLB را به ورودی CLB دیگر متصل می کند.

نکته ای که قابل توجه است این است که سیگنال،از خروجی یک سلول CLBبا گذشتن ازسوئیچ های قابل برنامه ریزی ،به ورودی CLBهای دیگر وارد می شود،لذا تعداد سوئیچ

هایی که در یک طرح بخصوص استفاده می شود تابع نوع طرح می باشد.بنابراین تاخیر یک طرح در FPGA ،تابع تعداد سوئیچ هایی است که توسط ابزار برنامه ریزی FPGA برای پیاده سازی مدار برای آن استفاده شده است.

3-3 ساختار سوئیچهای قابل برنامه ریزی در FPGAوCPLD :

اولین نوع سوئیچ قابل برنامه ریزی فیوز بود که با برنامه ریزی ارتباط آنها قطع می شدودر PLAها به کار برده می شد.برای مدارهای مجتمع CMOS با ظرفیت زیاد مانند CPLD ،ترانزیستور با گیت شناوربه عنوان سوئیچ قابل برنامه ریزی ،مشابه حافظه های EPROMوEEPROMبه کار برده می شود.در FPGAها نیز از سوئیچهای با کنترل SRAM وآنتی فیوز برای برنامه ریزی آنها استفاده می شودکه در ذیل هر یک از آنها مورد بررسی قرار میگیرد.

3-3-1:ترانزیستور سوئیچ قابل برنامه ریزی با گیت شناور :

ترانزیستورهای سوئیچ با گیت شناور مانند یک فیوز عمل می کنند ودر یک آرایهCPLD قرار میگیرند.

ترانزیستورهای با گیت شناور از نوع NMOS به نام EPROM معروف بوده وبه دفعات قابل برنامه ریزی وپاک شدن هستند.این ترانزیستورها دارای دو گیت می باشند که یکی از آنها مانند گیت ترانزیستورهای NMOS معمولی به سیم اتصال خارج متصل می شود وگیت دوم به نام گیت شناور داخل عایق ترانزیستور قرار دارد. برای برنامه ریزی این ترانزیستور،یک ولتاژ مثبت در گیت بیرون قرار میگیرد که باعث می شود شارژهای منفی یا الکترون به گیت شناور وارد شوند. بعد از حذف ولتاژمثبت مذکور،این شارژهای منفی در گیت شناور باقی میمانند به عبارت دیگر در آن حبس می شوند.حال اگر ولتاژ 1 منطقی یعنی 5ولت درورودی گیت بیرونی قرار گیرد،ترانزیستور نمی تواند به هدایت برود،به عبارت

دیگر ترانزیستور برای همیشه قطع می ماند، مانند این است که فیوز قطع شده باشد یا برنامه ریزی شده باشد.

برای پاک کردن گیت شناور مذکور ،باید شارژ های منفی گیت شناور از بین برود، ،که در این صورت ترانزیستور مذکور مانند یک ترانزیستور معمولیNMOSبا گیت خارجی کار خواهد کرد.

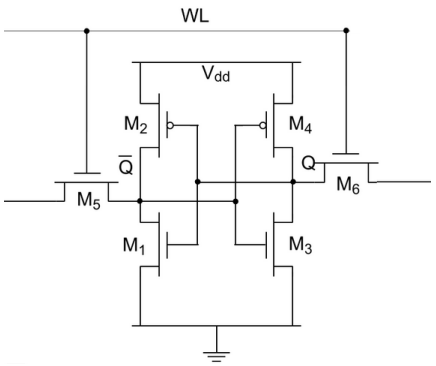
به این ترتیب ترانزیستور با گیت شناور NMOSقابل پاک شدن وبرنامه ریزی مجدد می باشد.

ترانزیستور EEPROMنیز مشابه ترانزیستور EPROM عمل میکند،با این تفاوت که به صورت الکتریکی با یک پا لس پاک می شود.

2-3-3:سوئیچ های قابل برنامه ریزی با حافظه SRAM:

در برخی FPGAها،سوئیچ های قابل برنامه ریزی ترانزیستوری به کار برده می شود که با حافظه SRAMبرنامه ریزی یا کنترل میشوند.در این صورت اگر خروجی حافظه SRAMبرابر 1 شود سوئیچ ترانزیستوری مذکور بسته می شود ودو سیم را به هم متصل می کند.در برخی FPGA ها مولتی پلکسر،به عنوان سوئیچ مورد استفاده قرار میگیرد که با قرار دادن مقادیر مختلف در خطوط select مولتی پلکسر، هر یک از ورودی های مورد نظر مولتی پکسر، به خروجی آن متصل می شود،یعنی یک سیم افقی که خروجی یک سلول منطقی می باشدبه یک سیم عمودی که ورودی سلول منطقی دیگر است، متصل می شود

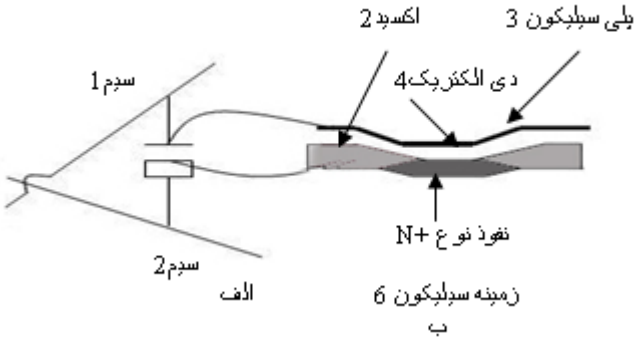
ساختار یک سلول حافظه SRAMمطابق شکل(3-10) می باشد.ترانزیستورهای M2-M1وM4-M3 دو معکوس کننده C-MOS می باشندکه یک فلیپ فلاپCMOS از حافظه SRAM را تشکیل می دهند.



.شکل(3- 10)

اطلاعات از طریق ترانزیستور T1 در حافظهSRAM ذخیره می شود،به عبارت دیگر حافظه برنامه ریزی می شود. خروجی حافظه SRAM یا فلیپ فلاپ به ترانزیستور سوئیچ T2 متصل است که اگر خروجی مذکور برابر 1 باشد ، ترانزیستور سوئیچ T2 هدایت میکند یعنی سوئیچ بسته می شود ودر صورتی که خروجی حافظه مساوی 0 باشد ترانزیستور سوئیچ T2 قطع یا باز می گردد. به این ترتیب ترانزیستور سوئیچT2،توسط فلیپ فلاپ یا یک سلول حافظه SRAM برنامه ریزی، یا کنترل می شود.

نوع دیگر سوئیچ قابل برنامه ریزی در FPGA ها سوئیچ آنتی فیوز می باشد.آنتی فیوز بر عکس فیوز در ابتدا مانند مدار باز عمل میکند وموقعی که برنامه ریزی شود مدار بسته یا اتصال وصل می گردد،لذا آنتی فیوز عکس فیوز عمل میکند.چون ساختن آنتی فیوز با تکنولوژی CMOS ساده است،لذا در برخی از FPGAها استفاده می شود.شکل ساختار آنتی فیوز شرکت ACTELرا نشان می دهد.

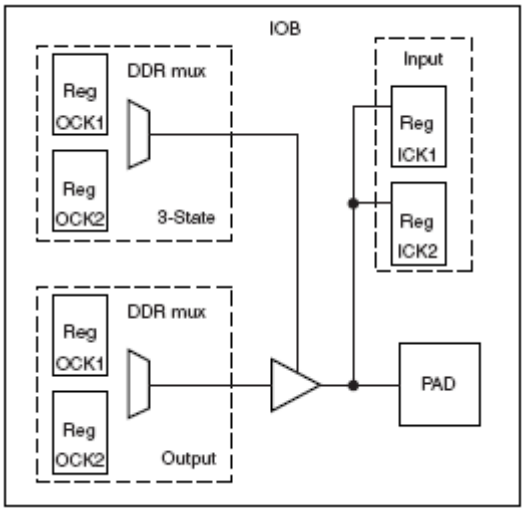


شکل 3-11 ساختار سوئیچ آنتی فیوز شرکت ACTEL :

همان طور که در شکل(3-11) مشاهده می شود،سوئیچ آنتی فیوز بین سیم1وسیم2که یکی خروجی یک سلول منطقی ودیگری ورودی سلول منطقی دیگر است قرار داده شده است که با برنامه ریزی سوئیچ مذکور،دو سیم به طور دائم به هم وصل میشوند.

سوئیچ آنتی فیوز از سه لایه تشکیل شده است که دو لایه بالا وپایین هادی ولایه وسط عایق یا دی الکتریک می باشد .زمانی که سوئیچ برنامه ریزی نشده باشد ارتباط بین دو لایه بالا وپایین قطع است یعنی سوئیچ باز می باشدولی هنگامی که سوئیچ برنامه ریزی شود،در این صورت لایه عایق تبد یل به لایه هادی می شودوارتباط دو لایه بالا وپایین برقرار می شود،به عبارت دیگر سوئیچ بسته می شود.در سوئیچ مذکور لایه بالا از نوع پلی سیلیکون(POLY-SI)ولایه پایین سیلیکون از نوع N+ که با اعمال ولتاژ بین پلی سیلیکون وسیلیکون نوع n+ ،عایق دی الکتریک ذوب می شودواتصال دو لایه بالاوپایین بر قرار می شود.

3-4:بلوک ورودی وخر وجی:



شکل 3-12:بلوک ورودی وخروجی

یک بلوک ورودی وخروجی (INPUT/OUTPUT BLOCK) در واقع آن بخش از FPGA است که رابط بین پایه های FPGA ومحیط خارج است .هر کدام از پایه های FPGA که مربوط به استفاده کننده باشد،(مثلا VCCویاGND نباشد بلکه در اختیاراستفاده کننده باشد که از ان به عنوان خروجی/ورودی داده استفاده کند.)به یک IOBربط دارد .سپس

IOB به مدارهای داخل FPGA وصل می شود. درحالت ساده یک IOB دارای سه فلیپ فلاپ است . یکی برای نگه داری داده هایی که از بخش داخلی FPGA می آید و قرار است بیرون برود استفاده می شود.یکی برای کنترل بافری که مقدار ولتاژ معادل صفر یا یک را درخروجی قرار می دهد ،استفاده می شود .به این ترتیب که اگر قرار باشد ازاین پایه FPGA برای ورود داده استفاده شود آن وقت Output Buffer فعال شده تا داده بتواند وارد FPGA شود.در نهایت فلیپ فلاپ سوم برای نگه داری داده ای که قرار است از بیرون وارد FPGA شود به کار میرود .هرسه FF دارای پالس ساعت یکسان وورودیSET/RESET یکسان هستند.باید دقت کرد که می توان از هیچ کدام از FFها استفاده نکرد ودر واقع یک حالت Combentional را برای ورودی وخروجی ها داشت.

5-3: بلوک های حافظه :

فرض کنید یک تابع منطقی با 20 متغیر را بخواهید روی FPGA پیاده کنید. در این صورت به تعداد زیادی CLB نیاز است . در حالیکه اگر RAMهای بزرگی روی مدار وجود داشته باشند می توان از آنها استفاده کرد.تمام FPGAهای جدید دارای بلوک های حافظه بسیارانعطاف پذیری میباشند.در اکثر موارد طراح نیاز دارد داه های مربوط به مداررا در جایی ذخیره کند ،تا در نهایت مثلا بتواند الگوریتمی را روی انها انجام دهد .از طرفی قرار دادن RAMهای خارجی که در بیرون FPGA قرار دارند وداده ها را ذخیره می کنند،همیشه به صرفه نیست. چرا که اولا سرعت عملکرد انها کند است،ثانیا طراحی مدار را مشکل میسازد،ثالثا تعدادی از پایه های FPGAرا از دست میدهیم.به عنوان مثال SPARTAN -II که یکی از FPGA های XILINXاست داارای بلوکهای حافظه هر کدام به ظرفیت 4096بیت است.طراح قادر است با این 4096 بیت هر کدام از ramهای ایستا 1×4096 یا 1×2048 یا

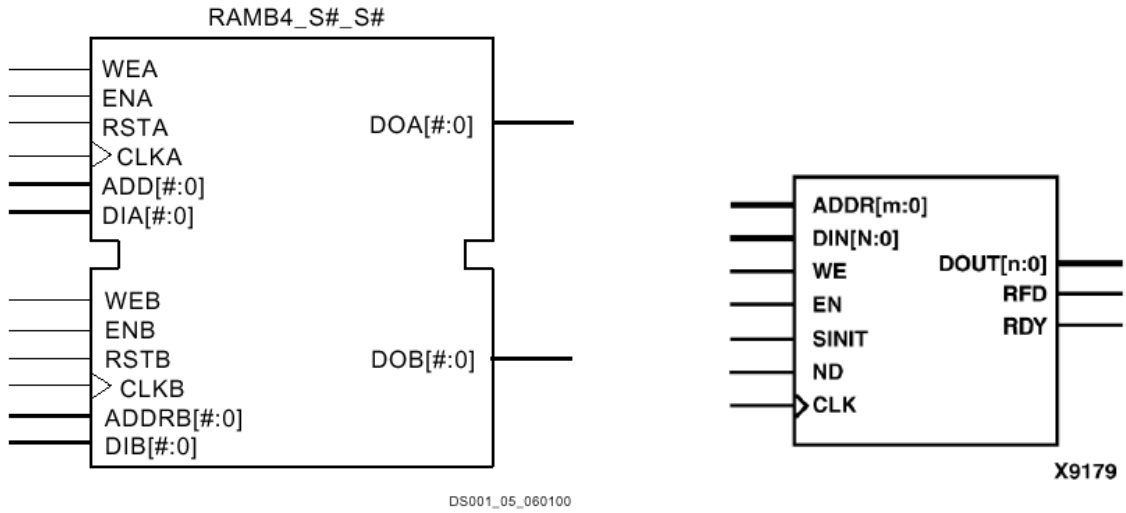
1×1024 یـا... را پیـاده کند.جدول 1 حالتهـای مختلـف را نشـان مـی دهد : (بـه صـورت تعداد بیتهای

ممکن برای عرض گذرگاه های آدرس وداده)

جـــدول1:حالتهـــای مختلف RAM

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>

از طرفی متوتن هر بلوک ram را بـه صـورت SIGNAL PORT و یـا DUAL PORT پیکربندی کرد.در شکل(3-13)بلوک دیاگرام هر نوع RAM نشان داده شده است:



شکل3-13

پورت های اصلی یکsignal port ram چنین است:

Add[m:0] : که ورودی ادرس حافظه است وتعیین میکند خواندن ویا نوشتن از وبه کجا باید انجام شود.

DIN[N:0] :که داده ورودی ایست که باید در حافظه ذخیره شود.

DOUT[n:0] : داده خروجی که حاصل عمل خواندن باید دران ریخته شود.

WE :هنگامی که بالا باشد نشان می دهد که داده ای باید درون حافظه ریخته شود.

CLK:پالس ورودی به حافظه است.کلیه اعمال با پالس ساغت سنکرون وانجام می شود.

هنگامی که یک BLOCK RAM به صورت DUAL PORT قرار است استفاده شود،دو پورت ورودی وخروجی داریم ودو پالس ورودی ساعت.هردو پورت به کل داده های موجود در RAMدسترسی دارند.مثلا در یک لبه بالا رونده پالیس ساعت هر دو می توانند از محل های مختلف حافظه بخوانندویا به محل های مختلف حافظه بنویسند.(البته هر دو به صورت همزمان نمی توانند در یک آدرس بنویسند.)چنین شاختاری برای پیاده شازی یک FIFOسنکرون بسیر ایده آل است.فرض کنید قرار است یک فرستنده بسیار سریع به یک گیرنده کند وصل شود.فرستنده داده ها را به صورت BURST انتقا ل می دهد.به این ترتیب که در یک بازه زمانی بسیار کوچک حجم بسیار زیادی از داده را با سرعت زیادی به گیرنده می دهد.(مثلا فرض کنید فرستنده یکSERVER بسیار سریع است که به نوبت به هر کدام از CILENT ها که تعدادشان زیاد است ولی کند هستند سرویس می دهد.)بنابر این به یک میانگیر بین فرستنده وگیرنده نیاز است،پالس ساعت گیرنده فرکانس اندکی دارد ولی دائمی است.در غوض فرستنده پالس ساعتی با فرکانس بالا دارد ولی به صورت لحظه ای اعمال می شود.اینجا یک FIFOآسنکرون بسیار ایده ال است.

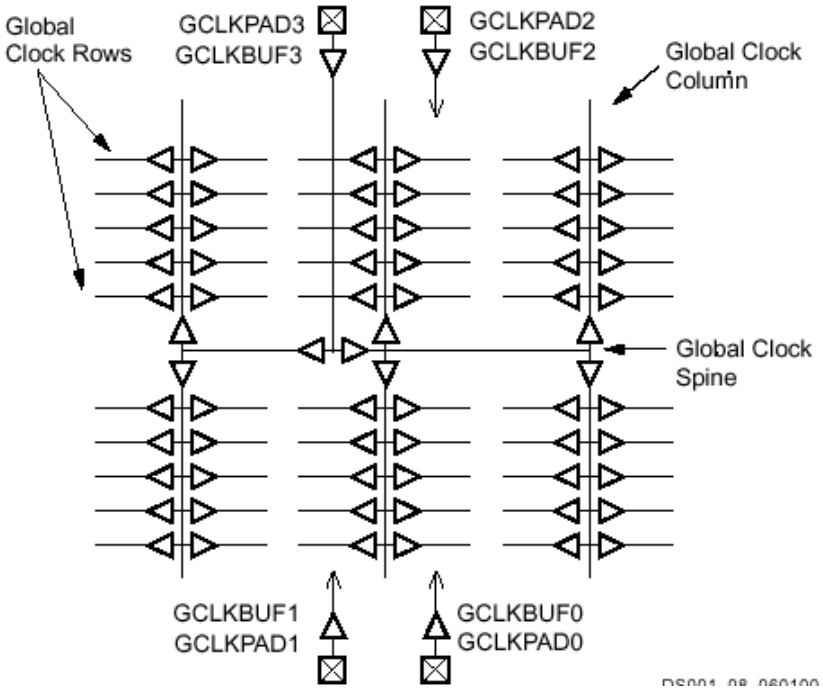
استفاده بسیار متداول دیگری که ازDUALPORT می شود به عنوان WIDTH CONVERTER می باشد.فرض کنیدعرض درگاه دادهA را16 بیت عرض درگاه B را 32 بیت تعریف میکنیم.

به این ترتیب می توان داده های 16 بیتی را دریافت وبه صورت 32بیت ارسال نمود ویابر عکس.

باBLOCK RAM می توان مدارهای متداول زیر را پیاده کرد. FIFO,STACK, LINKED LIST و... همچنین برای پیاده سازی ماشین حالت با سرعت های زیاد هم می توان ازآن استفاده کرد.بعضی ازبرنامه های سنتز مثل SYNPLIFY انقدر باهوش هستند که در برنامه VHDL ما قسمتی وجود دارد که برای پیاده سازی آن یکBlock Ram بسیار مناسب است خودشان به صورت خودکار از آن استفاده می کنند. ولی در حالت معمول ما باید به طور مشخص در برنامه ذکر کنیم،که می خواهیم برای پیاده سازی این قسمت ،(مثلا این حافظه)ازBlock Ram استفاده کنیم . نرم افزارXilinx Core Generator به ما این امکان را می دهد که از Block Ram ها درطرح هایمان استفاده کنیم. برای دیگر سازندگان FPGA هم نرم افزارهای مشابهی وجود دارد.

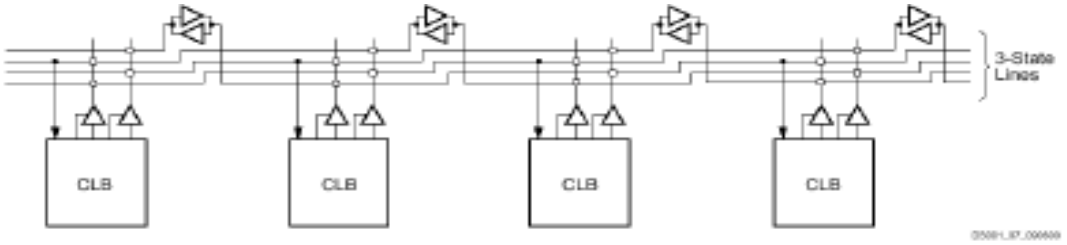
3-6: سیم های ارتباط داخلی:

داخل FPGA مقدار بسیار زیادی سیم وجود دارد. تعداد بسیار زیادی هم سوئیچ که می توانند هر کدام از پورتهایشان را به هر کدام دیگر وصل کنند. مجموعه این سیم ها وSitch Matrix ها به کار می روند تا قسمتهای مختلف مدار های Logic که قرار است روی FPGA پیاده شوند به هم متصل شوند.در هر FPGA برای clk وCarry سیم های خاصی وجود دارد که این امر در بسیاری از مواقع از جمله مواقعی که نیاز به انجام اعمال ریاضی است به سرعت عملکرد مدار کمک شایانی می نماید. شکل (3-14) شبکه ای که برای پخش کردن پالس داخلFPGA به کار می رود را نشا ن می دهد.



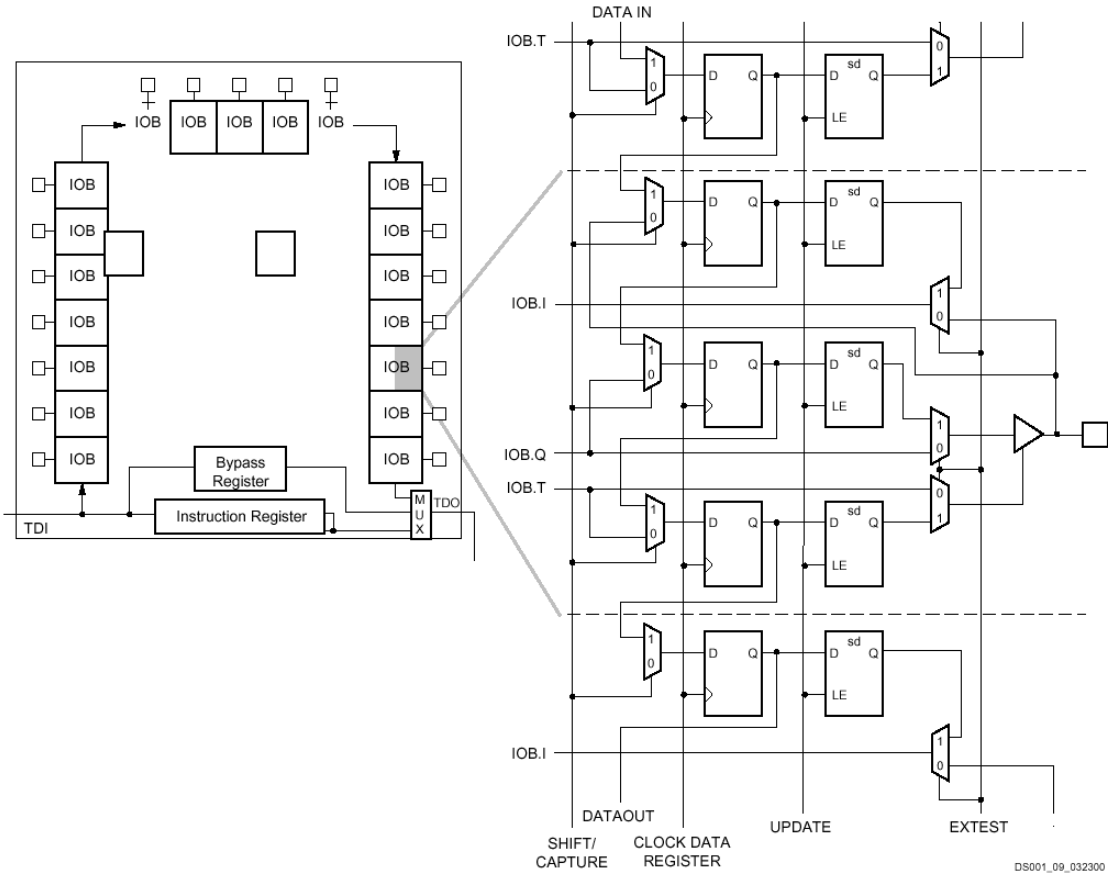
شکل 3-15 :شبکه پخش CLK در یک FPGA :

به عنوان مثال درSpartan-II تمام CLBهایی که در یک سطر قرار داند ،با خطوط ارتباطی مستقیم به هم وصل می شوند.شکل(3-15)یک نوع از منابع ارتباطی روی FPGA را که برایCLB ها هم به صورت(QUAD) 4تایی به کار می رود رانشان می دهد.



شکل 3-16 سیم های ارتباط مستقیم QUAD در درSpartan-II

7-3 مدار های موجود برای تست عملکرد:



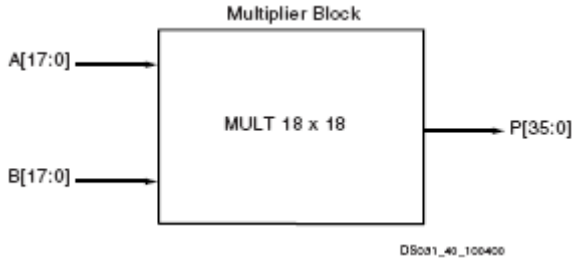
شکل3-17 :مدار Boundary scan

این مدار برای وقتی استفاده دارد که بخواهیم حین عملکرد FPGA در مدار وضعیت پایه های مختلف آن را به دست آوریم.این مدار در واقع یک شیفت رجیستر بزرگ است که دور تا دور FPGA را فرا گرفته است،چند پایه خاص روی هر FPGA وجود داردکه ورودی وخروجی این شیفت رجیستر وسیگنال های مربوط به کنترل آن را فراهم می آورد.به این ترتیب هم می توانیم وضعیت سیگنالهای داخل FPGA رادر یک زمان خاص پیدا کنیم وهم مقادیرمختلفی را به داخل FPGA بفرستیم،به مجموع این پاتیه ها درگاه jtag میگویند.

تمام IC های بزرگLogic 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 میکنیم ومقادیر موجود در روی شیفت رجیستر داخل IC را شیفت میدهیم بیرون وبه وضعیت پایه ها(بدون نیاز به اسکوپ)پی میبریم.وقتی که مدار بسیار بزرگ باشد پی به

ارزش این درگاه میبریم زیراکه می توان در یک لحظه وضعیت تمام خطوط ارتباطی را تشخیص داد.

3-8 :بلوکهای ضرب کننده:



شکل3-18:شکل نمایی یک ضرب کننده 18×18

در بعضی از محصولات جدید هر دو شرکت Xilinx وAltera بلوک های ضرب کننده وجود دارد.یعنی عمل ضرب را می توان به دو صورت انجام داد. یکی استفاده از CLB ها ودیگری استفاده از ضرب کننده ها. به طور مثال برای virtex-II این بلوک ها 18 بیتی میباشند.اینها می توانند به صورت pipeline ویا دریک سیکل کار کنند. در حالتpipe طبیعتا سرعت بالاتر است.(حدودا 200 مگاهرتز).برای استفاده از این ضرب کننده ها می توان از Core Generator کمک گرفت.

اگر ابزار سنتز پیشرفته باشد هنگامی که در برنامه از عمل ضرب استفاده شده باشد.به طور خودکار به سراغ بلوک های ضرب کننده ها می رود.این ضرب کننده ها معمولا برای پردازش سیگنال (که عملا چیزی جز کانولوشن نیستند).بسیار مطلوب می باشد.

3- 9 : میکرو پروسسور داخلی:

این قطعه در محصولات جدید ی همچون Virtex-II Pro از شرکت Xilinx به کار گرفته شده است یعنی علاوه بر تمام قسمتهای Virtex-II دارای چند میکرو پروسسور Power PC نیز می باشد که با آنها می توان مدارات مختلفی را پیاده سازی نمود.مدارLogic باVHDL یاVerilog طراحی می شود واعمالی که بایدPower PCانجام دهد به صورت برنامه(که

بعدا برا ی اجرا روی میکرو پروسسورCompile می شود.)نوشته می شود مجموعه اینها با هم عملکرد Virtex-II Pro را مشخص می کنند.این FPGA تخصصی بوده وبرای کارهای خاص (مثل سوئیچ های سریع) باید ار آن استفاده نمود.PowerPC در واقع میکرو پروسسوری است که توسط IBM تولید می شود. هنگامی که که چنین مداری در یکFPGA استفاده می شود اصطلاحا به آن Hard Core IP می گویند.

3-10یک مثال:

تمام خصوصیات خوب یک FPGA را می توان در Virtex-II دید کوچکترین FPGA این خانواده XC2V40 است که دارای 256 عدد Slice می باشد. همچنین 4 بلوک حافظه و 4 بلوک ضرب کننده دارد. نهایتا برای مدیریت پالس 4 عدد Digital Clock Manager در آن تعبیه شده است.XC2V40 میتوتند تا 88 پایه ICرا به استفاده کننده اختصاص دهد،استفاده کننده می تواند با این 88پایه هر کاری که می خواهد بکند. آنها را ورودی/خروجی تعریف کند وبه نحو مناسب آنها را به مدار داخلی FPGA ربط دهد .تمام این کارها بر اساس برنامه VERILOGو یاVHDL ای که استفاده کننده می نویسد ویا محدودیت هایی که در Implement تعیین می کند انجام می شود. بزرگترین ICدر این خانواده XC2V10000است که با 61440 ، Sliceکه می توان با ان مدارهای بسیار بزرگ را پیاده کرد. Sliceها در Virtex-II اصلاح شده اند تا مدارهای مختلفLogic بتوانند با تا خیرهای کمتر پیاده شوند . در حال حاضر Virtex-II در چند SPEED GRADE مختلف تولید می شود :5-،4-،6-،هر جه این عدد بالا تر رود به این مفهوم است که در ساختن این FPGA از مواد بهتر ،تکنولوژی پیشرفته تروا ستفاده شده استوتاخیرها در FPGA کوچکتر می باشد.پس یک FPGAبا speed grade برابر با 6- گران تر وسریعتر از همسان خود با سرعت 5- است.باVirtex-II می توان مدارهایی تا فرکانس 300 مگا هرتز را پیاده کرد

.تعداد Block Ram هایی که در این FPGA استفاده می شود نسبت به تمام سری های قبل بیشتر می باشد. حجم هر بلوک 18 کیلو بیت است.بلوک به صورت تک پورتی ودو پورتی قابل استفاده است وعرض پورتها می تواند متفاوت باشد . می توان تنظیم کرد که Block Ram وقتی هر دو پورت می خواهند بخوانند به یک محل حافظه دسترسی پیدا کنند (مثلا یکی میخواهد بخواند ودیگری میخواهد بنویسد .)به چه صورت عمل نماید .به طور مثال داده کنونی در آن محل به پورت مربوط به خواندن برود وداده جدید نوشته شود یا اینکه داده جدید نوشته شود و همان داده به پورت خواندن برود.

IOB در Virtex-II انواع مختلف تری از سیگنا لینگها را حمایت میکند . با استفا ده از LVDS (Low Voltage Differential Signaling) در Virtex-II می توان داده های سریال را با سرعت 840 Mbit/sec انتقال داد.

فصل چهارم

ایجاد یک پیکر بندی جدید درFPGA:

ایـن عمـل دارای مراحـل مختلفـی مـی باشـد کـه هـر مرحلـه نیـز دارای چنـدین ابـزار (برنامه)مخصوص به خود می باشد در همین راستا است که به معرفی برنامه هایی که در دنیای FPGA کاربرد دارند مانندHDS، Xilinx core generator، Fpga Compiler، Leonardo Spectrum، PKS، Simplify، Amplify، می پردازیم و درباره آنها بحث می کنیم.

1-4 روشهای پیکر بندی:

Configuration عملی است که طی آن وضعیت عملکرد تمام mux هاوLUT ها و switch matrix ها ی داخل FPGA تعیین می شود دو روش برای برای انتقال اطلاعات پیکر بندی به داخل FPGA وجود دارد.

همان طور که گفته شد با استفاده از JTAG می توان داده ها را به داخل FPGA فرستاد.مود خاصی از عملکرد FPGAوجود دارد که طی آن اطلاعات آمده از درگاه JTAG برای پیکر بندی استفاده شده،در محل مناسب در FPGA قرار میگیرد.برای این منظور لازم است بوردی راکه FPGA روی آن قرار دارد به کامپیوتر وصل کرده تا اطلاعات از طریق JTAG انتقال یابد.با این کار می توان در هر لحظه مدار را تست کرده وحالتهای مختلف را آزمایش کرده .کابل های JTAG مدارساده ای دارد وبه راحتی می توان ان را ساخت.کابل JTAG معمولا به صورت موازی به کامپیوتر وصل می شود.سپس به طور مثال برای FPGA ها ی شرکتXILINX برای آنکه داده های پیکر بندی را در FPGA ریخت باید از نرم افزار JTAG PROGRAMMER استفاده کرد.

روش دیگر (وقتی که قرار است FPGA دریک بورد مجزا کار کند)آن است که اطلاعات مربوط را در یک PROM بریزیم وآن را به FPGAوصل کنیم.(معمولا PROGRAM کردن این PROM ها با استفاده از پورت JTAG آن انجام می شود.)در هنگام پیکر بندی FPGA از اینPROMها استفاده میکند.انتقال بین داده ها به دو صورت موازی وسریال انجام می شود.همچنین دو حالت برای پالس ساعتی که با لبه بالا رونده آن داده انتقال می یابد ممکن است:حالتی که پالس ساعت ساعت از خارج FPGA تامین می شود که به آن حالت SLAVE میگویند.به طور مثال SLAVE SERIAL MODE که در آن با یک پالس خارجی داده به صورت سریال به FPGA انتقال می یابد. حالت دوم حالتMASTER است که در آن پالس انتقال داده را خود FPGA تولید میکند.در تمام FPGAها یک اسیلاتور نا

دقیق برای انجام این کار وجود دارد.این اسیلاتور یک پالس به طور مثال 2 تا8 مگا هرتزی (برای SPARTAN)تولید میکند.(که البته فرکانس آن دقیق نیست وتابع دما می باشد.)آنچه که در FPGA ریخته می شود یک فایل با پسوند BIT می باشدکه بعد از مرحله Implement ساخته می شود.آنچه که مهم است این است که طول فایل بیت صرف نظر از اینکه چه طراحی انجام شده است برای یک FPGA خاص همیشه مقداری ثابت می باشد . یعنی به هر حال وضعیت تمام قسمتهای FPGA مشخص می شود.PROMهای خاص برای انجام این کار توسط شرکت های سازنده FPGAساخته می شود.بعضی از آنهاOPT ROM هستند:One Time Programmable ROM که فقط یک بار روی آنها می توان برنامه ریزی کرد.مثل سریXC17Vxx .بعضی دیگر به تعداد بار دلخواه قابل برنامه ریزی اند مثلXC18Vxx

2-4 مراحل ایجاد پیکر بندی:

کلا برای هر طراحی که قرار است روی FPGA پیاده شود،باید یک سری کار هایی مشخص انجام شود. ابتدا باید مدار اصلی طراحی شود . در طراحی مدار اصلی معمولا به این صورت عمل می کنند که ابتدا تعداد مدولها ، عملکرد هر مدول وارتباط آنها را با هم مشخص می کنند وسپس 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 مدار را تعیین می کنید وسپس هر کدام از اجزا را به دقت توصیف می کنید تا آن نتیجه مطلوب حاصل شود. کلا کارهایی که تا مرحله آماده شدن کد Verilogبرای انجام شبیه سازی انجام می شود را design Entryمی گویند. پس بعد از مرحله Design Entry کدهای Verilog ما آماده هستند. حال به مرحله Function Simulation می رسیم. در این مرحله عملکرد مدار در حالت تاخیر صفر شبیه سازی می کنیم تا مطمئن شویم طراحی را درست انجام داده ایم . یعنی هر دفعه به ورودی

مقادیر مختلفی می دهیم وبا توجه به خروجی صحت عملکرد مدار را بررسی می کنیم .و اگر جواب هر یک از مراحل تست درست نباشد دوباره باید به مرحلهDesign Entry باز گشت وطرح را اصلاح کرد.معمولا همراه هر مدول Verilog که نوشته می شود یک برنامه دیگر (یک مدول دیگر) به اسم Verilog test fixture هم نوشته میشود.این برنامه سیگنال های ورودی مناسب برای هر مدول تحت تست را تولید می کند . به این ترتیب عملکرد کلی این طور است که با استفاده از یکی از نرم افزارهای شبیه سازی Verilog مجموعه مدول اصلی ومدول تست آن را ،که به هم وصل شده ا ند را شبیه سازی کرده و صحت عملکرد مدول اصلی را بررسی می کنیم.

پس از اطمینان از عملکرد صحیح مدار،نوبت به مرحله Synthesis می رسد. در این مرحله با استفاده از یکی از نرم افزار های Synthesizer مدار را تبدیل به مجموعهای از گیتهای منطقی می کنیم. می توان این مجموعه گیتها را دوباره تبدیل به برنامه Verilogکرد وحاصل راشبیه سازی کرد تا از صحت خروجی برنامهSynthesizer مطمئن شویم. تا این مرحله به عنوان ابزارهای شبیه سازی وسنتز از برنامه های هر شرکتی می توان استفاده نمود.مرحله آخر آن است که خروجی Synthesizer را به ابزار Implement بدهیم.ابزار Implement ابزاری است که خروجی Synthesizer را تبدیل به ترکیبی از المان های موجود در FPGA می نماید.سپس این عنصرها را در جای مناسب رویFPGA قرار می دهد . (Placement) و بین آنها را سیم کشی می کند(Routing) . در نهایت یک فایل با پسوند BIT تولید می کند که می توان از آن برایprogram کردن Rom ای که قرار است به FPGA وصل شود استفاده کرد.

ابزارImplementهر شرکت مخصوص خودش است،مثلا برایFPGA هایXilinx حتما باید ازابزار Implement خود Xilinx استفاده کرد.

4- 3 طراحی ابتدایی:

برای کسانی که می خواهند یک برنامه Verilog معمولی بنویسند ،به عنوان design entry tool فقط یک Editor کافیست ،تا بتواند برنامه Verilog خود را درآن بنویسد.ولی برای پروژه های بزرگ این اصلا کافی نیست.

1-3-4 نرم افزار(HDL Designer(HDS : یکی از قدر تمند ترین نرم افزارهایی که می توان برای Design Entry استفاده کرد HDL Designer تولید شرکت Mentor GraphICs می باشد . HDL Designer می تواند از SchematIC ای که برای مدار کشیده شده است ، کد Verilog بسازدبه عبارت ساده به کمک HDS می توان یک مدار شامل :

_Block Diagram هایی که ورودی /خروجیها ،مدول ها وارتباط های آنها با هم را نشان می دهد.

_State machine ماشین های حالت که بر اساس لبه ساعت ومقدار ورودی از یک حالت به یک حالت دیگر می روند و خروجی مناسب را هم تولید می کنند.

_Truth Table جدول های درستی را بر اساس ورودی،خروجی را مشخص می کنند.

_Flow Chart که نشان می دهد یک مجموعه منظم کارها رزا با توجه به شرایط در هر مرحله چگونه باید انجام شود.

طراحی کرد بدون اینکه نیاز باشد حتی یک سطر کدvrilog و یاVHDL نوشته شود. به طور مثال در پیاده کردن Machine State فقط حالتهای ماشین که به صورت دایره هستند در نقاط مناسب روی صفحه گذاشته میشوند و با یک سری خط جهت دار به هم وصل میشود. این خطوط در واقع بیانگر transistion از یک حالت به حالت دیگر هستند.شرطی را که طی آن این انتقال از یک حالت به یک حالت دیگر رخ می دهد،همچنین خروجی که باید در این هنگام تولید شود روی این خط نوشته می شود .در رسم بلوک دیاگرام بلوکها پهلوی هم قرار گرفته وبین انها سیم کشی می شود.در نهایت باید Truth Table ایجاد شود که تعیین میکند در ازای هر ورودی چه خروجی باید داده شود. درنهایت یک Flow Chart دقیقا عین همان است

که در برنامه ریزی از آن استفاده می شود. یک مجموعه اعمال مشخص از بالا به پایین به ترتیب انجام خواهند شد.شرط هایی که معمولا باعث پرش به نقاط مناسب می شود.

آنچه که HDS به وجود می آورد چیزی جز کد Verilog نیست .HDS می تواند طرح هایی را که اصلا Verilog نبوده خود به خود تبدیل به کد کند .کدی که تولید می کند synthesable است. یعنی قابل تبدیل به مجموعه گیت های منطقی می باشد وابزار سنتز می تواند آن را تحلیل کند. به هر حال این امکان HDS نیاز به دانستن Verilog را منتفی نمی کند،وبرای استفاده موثرازطرح های HDS نیاز به تسلط کامل بر VERILOG می باشد.آموزش طرز عملکردن با HDS ساعتها زمان را می طلبد.به عنوان یک مرجع خوب برای شروع کار می توان بهTutorialای که در بخش Help آن وجو دارد،رجوع کرد.

2-3-4 نرم افزارXilinx CoreGenerator : یک Core عبارت است از مقداری که قبلا آماده شده و می توان ازآن به صورت آماده در طرحها استفاده کرد. به طورمثال در بسیاری از مدارهای Logic شمارنده وجود دارد ، حال به جای آن که هر باربرای یک مدار یک شمارنده طراحی شود و بهینه سازی شود ،یک گروه طراح یک شمارنده قابل انعطاف وبهینه از لحاظ سرعت طراحی می کنند وآنها را به فروش می رسانند.به این ترتیب یک کار لازم نیست چندین بار تکرار شود.Core Generator نرم افزاری است که این نوع مدار های آماده را به صورت فایل edif تولید می کند.تنها باید در برنامه Verilog مدولی که توسط Core Generator تولید شده ،را وارد نمود ولی توصیف آن لازم نیست. بعدا هنگام انجام عمل Implemention فایل edf حاصل ازCore Gen در کنار فایل edf حاصل از سنتز مدار اصلی قرار می گیرد وبه این ترتیب مدار core به بقیه مدارها اضافه می شود.

Editor 3-3-4 های قدرتمند: یکی از مهم ترین ابزارهایی که درطراحی استفاده می شود Editorاست .به طور مثال Notepad یک Editor معمولی است که برای کارهای ساده مناسب است. یک Editor خوب حتما باید خاصیت syntax highlightingداشته باشد تا فهم

وغلط یابی کد Verilog را برای کاربر آسان کند. در حال حاضر دو editor فوق العاده قدرتمند برای نوشتن کد Verilog وجود دارند: اولی textpad که فقط تحت ویندوزموجود است و دومی nedit که تحت Linux کار می کند واز TextPAD بهتر می باشد زیرا هم OPEN SOURCE است وهم تمام آن کارها را انجام می دهد .نکته ای که نباید فراموش نمود رعایت نظم در نوشتن برنامه می باشد زیرا با رعایت نظم، هنگامی که برنامه ای بزرگ شود تست کردن وغلط یابی آن آسان تر می شود.

4-4 شبیه سازی:

برنامه هایی وجودد ارد که کد Verilog را می گیرند وعملکرد آن را شبیه سازی می کنند. به این ترتیب که شکل ورودی مدول تحت تست برای آنها تعیین می شود وآنها خروجی مدول را می دهند.

1-4-4 نرم افزار Modelism : Modelism محصول شرکتmodel technology که خود یکی ااز شعبات mentor graphics است یکی از قوی ترین ومشهور ترین شبیه ساز هایی است که که برای شبیه سازی مدارهای Logic که با Verilog یا VHDL یا هر دو توصیف شده اند،به کار می رود .کار در ساده ترین حالت خود حول یک صفحه به نام Waves می چرخد که درآن تمام شکل موج ها نمایش داده می شود. در ابتدای کار که Modelism بالا می آید تنها یک پنجره کوچک در اختیار استفاده کننده قرار می دهد تا دستورات خود را وارد نماید برای اینکه عملکرد مشخص شود با یک مثال ادامه می دهیم:

فرض کنید می خواهیم عملکرد مدار ساده مولتی پلکس 4 بیت را شبیه سازی کنیم. اولین کاری که لازم است،نوشتن یک مدول دیگر است که سیگنالهایی را که برای تست مدول اصلی لازم است،تولید کند،به این برنامه Verilog به اصطلاح Test Fixture می شود شاید به نظر دشوار بیاید که برای هر برنامه اصلی یک برنامه دیگرنوشته شود که نوشتن آن هم چندان آسان نیست وافراد مبتدی ممکن است که ترجیح دهند که شکل موجها رامستقیم وبدون

استفاده از یک برنامه دیگر به مدول اعمال کنند.هر چند با گذشت زمان فرد به اهمیت تست کردن مدول پی می برد.وبه نوشتنTest Fixture های مناسب روی خواهد آورد.

ممکن است که نوشتن Test Fixture ویا به عبارت دیگر Test Bench برای طراح 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 آیا ان عملکرد درست است یا خیر،که خود یک علم است.

شرکتهای فراوان ومحصولات متنوعی برای انجام عمل Verification وجود دارد.معمولا تست یک مدول به اِین صورت انجام می شودکه ابتدا با Verilog یا SystemC یا ...، برای آن یک مدول نوشته می شود.یعنی برنامه ای نوشته می شود که هر چند قابل سنتز نیست همان شکل موج ها را در مقابل بحریک ورودی تولید می کند. حال با استفاده از یکی از نرم افزارهای مشهور برای Verification به طور مثال Specman Elite از Verisity یاVera محصول شرکت Synopsys یا Test Builder محصول Cadence (که به صورت Open Source وfree منتشر می شود)به هرصورت با یکی از این نرم افزارها انواع واقسام شکل موجها وحالتهایی را که ممکن است پیش آید ایجاد می شود واین شکل موجها به هردو مدول (مدول اصلی که به آن DUT یعنی Design Under Test یا DUV یعنی Design Under VerifICation می گویند ومدلی که برای این مدول نوشته شده است .) فرستاده می شود حاصل خروجی مدول اصلی ومدل آن با هم مقایسه می شود واگر باهم انطباق داشتند از صحت عملکرد مداراطمینان حاصل می شود.(البته تا حدودی چون ممکن است که خود مدل هم اشتباه باشد.)Specman ویا Vera هر کدام برای تولید شکل موج های ورودی زبان برنامه نویسی مخصوص خودشان را دارند .معمولا این زبانها خیلی شبیه

به++C هستند.البته بحث راجع به Verification بسیار گسترده است ودر این جا فرصت بحث بیشتر درباره آن را نداریم.

برای مدول ساده مولتی پلکس 4 بیت TEST FIXTURE این چنین است:

```
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 میشود طیف متفاوتی از سیگنالهای ورودی به مدول اصلی اعمال می شود وخروجی آنها بررسی میگردد. پس هم اکنون دو فایل موجود است :اولی برنامه Verilog اصلی است با خود دارد و نام آن : four_bit_mux.v می باشد . فایل دیگر test bench است.و نام آن mux_test_fixture می باشد. نکته ای که باید در نظر داشت این است که اولا test fixture دارای پور تهای ورودی وخروجی نبوده وثانیا مدول اصلی در داخل test fixture به کار گرفته شده است تا به آن سیگنال داده شود .پس در اینجا مدول مرتبه بالاتر یا اصطلاحا top level ،مدول mux_test_fixture خواهد بود.

دو واژه test bench وtest fixture هر دو برای مدولی که به منظور تست برنامه دیگر نوشته می شود به کار می روند هر چند واژه درست تر test fixrure است . test bench مخصوص زبانVHDL بوده وtest fixture مخصوص Verilog.

حال اگرفرضاً دو فایل فوق در دایرکتوری D:\test قرار داشته باشند برای باز کردن اینها در modelism دستور زیر را باید وارد کرد :

Cd{D:/test}

حال لازم است که در این دایر کتوری یک library ساخته شودlibrary آن دایرکتوری است که برنامه هایVerilog ما به صورت کامپایل شده در ان قرار می گیرد وقتی قراراست عمل شبیه سازی انجام شود اطلا عات ازآن دایرکتوری خوانده می شود .پس این دستور وارد می شود:

Vlib sim_1

حال اگر به d:\test مراجعه شود یک دایرکتوری به نام simu_1 ساخته شده است. حال باید برنامه های Verilog کامپایل شده وداخل دایرکتوری sim_1 ذخیره شود:

vlog –work sim_1 four_bit_mux.v mux_test_fixture.v

با صادر کردن دستور vlog، Modelism آن دو فایل را می خواندوحاصل compile را داخل sim_1 ذخیره می کند.از طرفی گزارش می دهد که مدول top کدام است. حال می توان simulator را احضار کرد:

vsim sim_1.mux_test_fixture

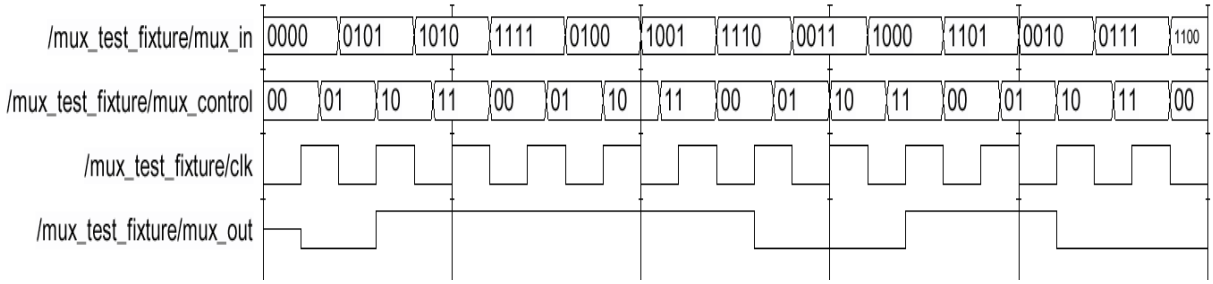
با این دستورها signalsوwaves آشکار می شود:

view signals; view wave

با رفتن به پنجره سیگنال و با کمک shift و ماوس تمام سیگنالها انتخاب می شوند .سپس می توان آنهارا با کشیدن به داخل پنجره wave منتقل نمود. با وارد کردن دستور زیر در پنجره MODELISM شبیه سازی برای یک میکرو ثانیه انجام می شود.

Run 1 us

می توان شکل موجهای حاصل در پنجره wave را دید.



حال بعد از صحت عملکرد مدار می توان به مرحله بعد یعنی Synthesis رفت کارهای فوق ساده ترین کارهایی بود که می توان در مدلیسم انجام داد .Modelism از Verilog PLI حمایت می کند. به این ترتیب به طور مثال می توان خروجی های شبیه سازی را به جای روی صفحه wave در پنجره برنامه ای که باvisual c نوشته شده است دید با Verilog pli می توان با هسته داخلی شبیه ساز ارتباط بر قرار کرد ،به آن داده وارد نمود یا داده از آن دریافت کرد.Modelismزبان Scripting، TCI را به طور کامل حمایت می کند.در واقع ورودی COMMAND در Modelism چیزی جز یک shell tci نیست.در دنیایی که

بزرگترین طرحها باید در کوتاه ترین زمان ممکن به مراحل نهایی خود برسند،scripting واتو ماتیک کردن کارها بسیار اهمیت دارد.

2-4-4نرم افزارLDV: LDV محصول شرکت Cadence در واقع مجموع چند شبیه ساز Verilog وVHDLاست که به صورت یک محیط مجتمع عرضه شده اند.اولین شبیه ساز Verilog-XL است که در واقع یک نسخه بهبود یافته ازVerilog بود Verilog-XL یک شبیه ساز تمام عیار Verilog است. به مفهوم این که هنگام کار با آنها بسیاری از مفاهیم برنامه نویسی باVerilog ،که به خاطر سخت بودن کار با آنها، در شبیه ساز های دیگر حذف شده اند و یا مورد توجه قرار نمی گیرند،درVerilog-XL آشکارا دیده می شوند. -Verilog XL قابلیت هایی دارد که بقیه شبیه سازها ندارند.Verilog-XL دارای یک سری SYSTEM TASK هایی است که بقیه شبیه ساز ها آن را ندارند.این به کاربر کمک می کند که بتواند برای شبیه سازی (به خصوص برای تست.)از قابلیت های بیشتری استفاده کند برنامه شبیه ساز دیگری که درLDV وجود دارد Affirma NC-Verilog است که یک شبیه ساز Verilog بسیار سریع است. عملکرد آن به این ترتیب است که برنامه Verilog ای که قرار است شبیه سازی شود می گیرد وآن را تبدیل به مجموعه ای ازدستورات اسمبلی مخصوصی CPU ای که قرار است شبیه سازی روی آن انجام شود می نماید.تبدیل به گونه ای انجام می شود که اجرای دستورات اسمبلی و مقادیری که برای هر کدام حاصل می شود ،معادل با وضعیت سیگنالها وسطح منطقی آنها در مدار تحت شبیه سازی باشدبه همین خاطر به آنها NC-Verilg یعنیNative Code Verilog می گویند.فایل دستورات اسمبلی حاصل مستقیما برای اجرا شدن بهCPU داده می شود.

مقادیر سیگنالها در یک database ذخیره می گردد تا استفاده کننده بتواند آنها را ببیند .به این ترتیب عمل شبیه سازی با سرعت بسیار بالایی انجام میشود.این نوع شبیه ساز ها برای طرح های بسیار بزرگ استفاده می شود. البته برای طرح های کوچک هم می توان از آنها استفاده

کرد .در نهایت می توان از نظرکارشناسان گفت که LDV از Modelism بسیار سریعتر،و پایدارتراست Modelism در حال حاضر هم مقدار زیادی bug دارد،ولی LDV از این لحاظ بسیار بهتر است.LDV هم مثل Modelism به طور کامل tci را حمایت می کند.باید دقت کرد از آنجا که توسعه دهنده اصلی Verilog خود شرکت Cadence است معمولا جدید ترین قابلیت ها به LDV اضافه می شوند. Verilog pliدر LDV هم وجود دارد.

5-4 سنتز:

مهم ترین مرحله برای ساختن هر IC منطقی که از زبانهای توصیف سخت افزاری برای آنها استفاده شده است ،مرحلهSynthesis است در این مرحله کد HDLتبدیل به مجموعهای از گیتهای مناسب می شود.

1-5-4 نرم افزارFPGA Express وFPGA Compiler II: FPGA Express محصول شرکتSynopsys یکSynthesizer آسان برای استفاده است.برنامه دارای یک Toolbar است که تمام کارها را از طریق آن می توان انجام داد.اولین دکمه روی Toolbar هنگا می که می خواهیم یک پروژه جدید راآغاز کنیم به کار می رود.مثلا فرض کنید بخواهیم four_bit_mux را سنتز کنیم ابتدا روی New Project کلیک کرده به دایرکتوری D:\test می رویم که فایل در آن است وسپس synth_1 را انتخاب می کنیم. حال باید نام فایلهایی که باید سنتز شوند را وارد کرد.four_bit _ mux.v را انتخاب می کنیم.FPGA Express فایل را برای اطمینان از لحاظ درستی syntax آنالیز می کند سپس در پنجره مقابل آن یک تیک می زند.حال روی علامت مثبتی که در کنار نام فایل وجود دارد کلیک میکنیم تا تا اسم مدولهای داخل فایل نشان داده شود. درداخل این فایل Verilog فقط یک مدول به نام four_bit_mux وجود دارد ان را انتخاب نموده و همراه با انتخاب آن به دکمه هایی که در Toolbar وجود دارد دقت می کنیم یکی از آنها پر رنگ می شودروی آن کلیک کرده یک پنجره ظاهر می شود که در آن باید مشخص نمود که سنتز برای کدام FPGAاز کدام شرکت

انجـام شـود.مثلاXILINX ویـا 2 Spartan ویـا 2S15 . بـا فشـار دادن OK عمـل سـنتز وOptimize شروع می شود.حال می توان حاصل را در پنجره سمت راست مشاهده نمود.در این بخش روی four_bit_mux-optimized کلیک کرده همزمان به Toolbarدقت می کنیم بـا انتخـاب دکمـه ای کـه فعـال شـده اسـت فایـل EDIF تولیـد مـی شـود.فایل EDIFاستانداردی پذیرفته شده در بین تمام شرکتهای تولید کننده مدارات Logicاست وحاوی اطلاعاتی مربوط به چگونگی وصل شدن گیتها بـه هم و نـوع گیت هـایی کـه استفاده شده، مـی باشد. در واقع یـک Netlist از مدار را به همراه خود دارد.این فایل ورودی مرحله بعد یعنی Implement است . پس روی دکمه ی مربوط درToolbar کلیک کرده وسپس okرا مـی زنیم.FPGA Express یک فایل با پسوند EDF در دایرکتوری Synth_1 کـه دایرکتـوری پـروژه بـود مـی سـازد.حال می توان به مرحلهimplemet رفت.

FPGA Compiler II نیزمحصـول شـرکت Synopsys مـی باشـد ودر واقـع بـا FPGA Express فرقی ندارد همیشه تکنولوژی های جدیدی که به برنامه سنتز کننده داده می شود بـه FPGA Compiler اضافه می شود.بعد ارز اینکه تست هـای کـافی روی آن بـه عمل آمد این امکانات بهFPGA Express هم اضافه می شود.

محیط آن کـاملا شبیه FPGA Express اسـت.در حـال حاضـر Synopsys اعـلام کـرده کـه دیگر نرم افزاری به نام FPGA Express تولید نخواهد کرد وفقط FPGA Compiler تولید خواهد شد.

2-5-4 نـرم افـزار Leonardo Spectrum : محصـول شـرکت Leonardo Spectrum examplar است کـه خـود جزئـی از شـرکت Mentor Graphics مـی باشـد . بـا Leonardo مـی تـوان همـان کارهـای FPGA Express را انجـام داد.Leonardo نسـبت بـه FPGA express ،دارای تنظیمـات بیشـتری است.سـه پنجـره اصـلی در آن وجـود دارد ،اولـی رونـد عمومی کار را ومراحلی که تا تولید شدن فایل edfباید طی شوند را نمـایش مـی دهد،در دومـی

مـی تـوان دسـتور یـا تنظـیم هـایی را کـه لازم اسـت وارد نمـو دو سـوم پیـا مهـایی اسـت کهLeonardo می فرستد.

مزیت Leonardo نسبت بهFPGA Express در این است که Command Shellآن (همـان پنجـره دوم)درمحـیط GUI قـرار دارد.در صـورتی کـه بـرای Express ایـن طـور نیسـت وshellبه صورت یک برنامه جدا ی ازGUI قرار گرفته است.در حال حاضر دیگر Mentor GraphICs نرم افزاری به نام Leonardo Spectrum تولید نمی کند و قصد داردبـه جـای آن نرم افزاری به نام precision Synthesizer ارائه دهد که به نظرمی رسد یک نـرم افـزار سنتز فیزیکی خواهد بود.

3-5-4 نرم افزارsimplify : simplify تولید شده توسط شرکت Synplicity است کـه بـا هوشـترین وبهتـرین نـرم افـزار سـنتزی اسـت کـه وجـود دارد(ابـزار سـنتز معمـولی نـه سـنتز فیزیکی)استفاده از این نرم افزار هر روز در بین طراحان متداول تر می شود.کار کردن بـا آن کمی سخت است ولی قابلیتهایی دارد که سایر ابزار سنتز ندارند.

4-5-4 سنتز فیزیکی:

وقتی ابزار سنتز می خواهد مداری را تبدیل به گیت هـای معـادل بنمایـد روشـهای متفـا وتـی را پیش رو دارد ولـی اینکـه کدام روش ممکن است رضـایت بیشتری بـه کـاربر بدهد یـک روش است که بستگی به خواسته او دارد مثلا اگر بـرای او سطح اشـغال شده توسط مدار بـر روی FPGA مهم باشد بایـد optimization بـر حسـب Area انجـام شـود ویـا اگـر بخواهـد مـدار سرعت بیشتری داشته باشد ومهم نباشد که چه سطحی وچه تعداد سلولی از FPGA اشغال مـی شود باید Optimization بر حسب speed انجـام شـود.یا شـاید بخواهـد حالـت متعـادل را پیـاده کند.

حال فرض می کنیم که Optimization باد بر حسب سرعت انجام شود معیار نرم افزار سنتز برای اینکه کدام مدار سریعتر است چیست؟نرم افزار سنتز دارای کتابخانـه هـایی مـی باشـد کـه

در آنها برای هرFPGA نوع گیتها و میزان تاخیر هر کدام مشخص شده است.پس برنامه می تواندبرای هر کدام از مدارها بیشترین تاخیر را با جمع زدن تاخیر گیتها روی طولانی ترین مسیرLogic ای که در هر مدار وجود دارد محاسبه کرده ومداری را انتخاب کند که کمترین تاخیر را دارا می باشد.

اما درا ینجا نکته ای وجود دارد وآن این است که آیا این تاخیرها در واقعیت هم همین قدر می مانند. حال با یک مثال موضوع را روشنتر می نماییم.عناصر روی FPGA باید با سیم با هم ارتباط داشته باشند که این سیم ها(منابع Routing)خودشان می توانند تاخیر قابل ملاحضه ای را در مدار ایجاد کنند ابزار سنتز هنگامیکه عمل سنتز را انجام میداد هیچ گونه اطلاعی از تاخیرها نداشت در حالیکه برای بهترین انتخاب باید اینها هم در نظر گرفته شوند.

یکPhisical Synthesizer در هنگام سنتز نه تنها به تاخیر گیتها بلکه به تمام تاخیرهای ممکن، با توجه به حرارتی که در هر ناحیه از IC تولید می شود ،با توجه به وضعیت سیمهای اتصالی ،نحوه قرار گرفتن Cell ها رویDevice و خلاصه با توجه به واقیت فیزیکی IC که قرار است این طرح روی آن پیاده شود توجه دارد .Phisical Synhesizer واقعی این توانایی رادارد که اعمال Placement وRouting را همزمان انجام دهد.یعنی همان طور که دارد کدها را سنتز وبهینه سازی می نماید هم زمانPlacement را هم انجام می دهد وتاخیرها را اندازه می گیرد واگر رضایت بخش بود کار را تمام می نماید واگر نه دوباره سنتز را به صورتی دیگر انجام می دهد پس خروجی نهایی یکPhisical Synthesizer فایلی است که که طرح Place&Route شده است.(یعنی نهایی ترین چیزی که باید روی IC پیاده شود.)اوایل عملSynthesis جدای از اعمال Place&Route انجام می شد.مثلا برای ASIC این طور بود که ابتدا مدار را با Design Compiler سنتز می کردند وسپس با یک ابزارPlace&Route مثلا Apollo از شرکت Avanti و یا Dracoulla ازCadence مدار نهایی IC با توجه به تکنولوژی مورد نیاز تولید می شد اما بعدها معلوم شد که اگر سنتز

Place&Route با هم انجام شود نتیجه بسیار بهتر است و در نتیجه هر کدام از شرکت های Synopsys وCadence شروع به تولید Phisical Synthesizer های مخصوص خود کردند.

Synopsys نرم افزار Phisical Compiler را تولید کرد وCadence نرم افزاری به نام Phisical Kowledgeable System یا (PKS) راتولید نمود که نرم افزار های بسیار قدرتمندی می باشند که در بسیاری از طرا حی های ASIC به کار می روند.

امادر حوزه FPGA مشکلاتی وجود دارد زیرا ابزارهایPlace &Routeفقط در دست سازندگان FPGA می باشد مثلا XILINX نرم افزار ISE را تولید می کند که هسته آن برنامه ساده ای به نام PAR می باشد که عملPlace&Route را انجام می دهد.پس شرکتهای سازنده ابزارهای سنتز فیزیکی به مشکل نداشتن برنامهPlce&Route برمی خورندsinplICity نرم افزاری به نامAmplify ساخته است که یکPhisical Synthesizer مخصوص FPGA استالبته این کار را با همکاری نزدیکXilinx انجام داده است.

با این ابزار می توان طرح را برای پیاده سازی در یک ناحیه خاص بهینه سازی نمود وتاخیرها را با نسبت خوبی تخمین می زند و آنچه را که در هنگام place&Route رخ می دهد را با تقریب نسبتا خوبی حدس می زند. Amplify می تواند طرحی را که یکبار Implementشده است را دریافت نماید وسنتز را دوباره طوری انجام دهد که تاخیرها کمتر شوند

5-5-4 مقایسه بین نرم افزارهای سنتز:

در فرکانسهای بالاتر از125 مگا هرتز Leonardo Spectrum از همه ضعیف تر عمل می نماید .در بین FPGA Express وSynplify هنوز هم جدال وجود دارد و هردو مدارها را به نحو مناسبی سنتز می کنند .هر چند گزارشهایی که Synplify می دهد بسیار دقیقتر از

Express است در نهایت شاید بتوان گفت که در حال حاضر بهترین گزینه استفاده از Amplify که یک ابزار سنتز فیزیکی است می باشد.

6-4 پیاده سازی:

آخرین مرحله انجام یک پروژه ساده با FPGA مرحله ای است که درآن فایل EDIFسنتز شده دریافت و،مدار مربوط به آن روی FPGA پیاده می شود.نرم افزارهای مر بوط به این بخش را فقط شرکت تولید کننده FPGA دارند. 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 ارئه می دهد.مانندXilinx Foundation و ISE . در واقع 3فایل اجرایی مهم وجود دارد که کل کار به عهده آنهاست وما در اینجا سه فایل مهم اجرایی لصلی و هر کدام را به صورت مختصر بررسی می کنیم:

1-6-4 :Translation

عملکرد برنامهNGDBUILDER.EXE به این صورت است که فایل EDFرا می گیرد واز روی آن یک فایل NGD می سازداین فایل NGD منبع تمام کارهای بعدی خواهد بود.تمامی اطلاعاتی که برای مراحل بعد لازم است از جمله محدودیت ها ،پایه ها و...در این فایل قرار می گیرند.از طرفی در برخی روشهای طراحی کل طرح را با هم سنتز نمی کنند بلکه آن را جدا جدا سنتز می کنند.بنا بر این طرحی که قرار است پیاده شود 2یا بیشتر فایل EDF پیدا می کند .NGDBUILDER اینها را به هم می چسباند ویک فایل واحد از روی آنها می سازد. به این کارDesign Expantion می گویند.(پس مثلا اگر Core یا Hardware Macro داشتیم در این مرحله از آنها استفاده می شود.)

2-6-4 Mapping: برنامه map.exe فایل NGD مرحله قبل را می گیرد و یک فایل NCD به ما می دهد. دقت کنید که هنوز برای هیچ کدام از عناصر NGD بر روی FPGA محلی در نظر گرفته نشده است بلکه فقط دسته بندی ثباتها وLUTها تعیین شده است.فایل

NCD حال تمام عناصری را که در مدار هستند را در خود ذخیره کرده است ولی به صورت Unplaced&Unrouted .

Placement 3-6-4:

در این مرحله برای تمام عناصر موجود درNCD محلی روی FPGA در نظر گرفته می شود.حاصل کار باز هم یک فایل NCD خواهد بود ولی این بار عناصر موجود در فایل NCD به صورت Placed شده هستند. هنوز هم سیم های بین عناصر Unrouted هستند واز منابع Routing استفاده نشده است.

Routing 4-6 -4:

آخرین وتقریبا مهم ترین مرحله کار Routing می باشد که طی آن عناصری که روی FPGA در مکان های مشخصی درBlock RAMهای،Sliceهاو... قرار دارند با سیم وبا استفاده از منابع Routing روی FPGA به هم متصل می شوند .نتیجه یک Routing خوب یک فرکانس عملکرد خوب است . هر چه الگوریتم های Routing بهتر عمل کند،کار بهتر انجام خواهد شد.فایل اجرایی ای که عمل Place&Route را انجام می دهد par.exe نام دارد.

7-4بر آورد سرعت نهایی مدار:

فرض کنید مداری راImplement کردیم برای دانستن فرکانس کاری مدار واینکه از هنگام ظاهر شدن داده لبه بالا رونده در کلاک تا زمانی که داده در خروجی ظاهر شود چه زمانی طول می کشد ،SET UP های مر بوط به فیلیپ فلایها چقدر است و.. نرم افزار Timing Analizer این کار را برای ما انجام می دهد.تمام تاخیرهای خروجی وتا خیرهای مدارهای داخلی محاسبه می شوند و به صورت گزارش به استفاده کننده داده می شوند.

فصل پنجم

زبان توصیف سخت افزار VHDL :

زبان توصیف VHDLبرای توصیف وشبیه سازی مدارهای دیجیتال از ساده ترین نوع گیتها،تا سیستم های پیچیده تر مداری مانند پروسسورها و....ویا در طراحی مدارهای کاربرد

خاص ASIC(APPILICATION SPECIFIC INTEGRATED CIRCUITS)به کار برده می شود.

VHDLیک زبان استاندارد بین المللی برای توصیف مدارهای دیجیتال به صورت BEHAVIORو STURCTURALاست ،که می توان با آن طراحی سیستم های دیجیتال را به صورت برنامه ،به صورت متن برای کامپیوترها مهیا نمود تا توسط مهندسین ، متخصصین وشرکتها در تمام دنیا قابل استفاده باشد.

اصولا VHDLبرای شبیه سازی مدارهای سخت افزاری طراحی شده بود .ودرآن موقع ابزارهای سنتز یا CADبرای طراحی یا پیاده سازی کامپیوتری وجود نداشت،ولی باپیشرفت ابزارهای برنامه ریزی FPGA، VHDLبرای طراحی مدارهای دیجیتال نیز مورد استفاده قرار گرفت.این ابزارها به طور اتوماتیک قطعات منطقی یا دیجیتال را لنتخاب،به هم وصل وطرح نهایی را بر روی مدارهای قابل برنامه ریزی FPGA، ,CPLDیا ASICپیاده می کنند.

در سالهای قبل از 1986 زبانهای توصیف سخت افزاری متفاوتی مانند PAL ASM ABLE و...توسط شرکتهای مختلف برای برنامه ریزی PAL,PLA,PLD وجود داشت که کاربران شکل سلیقه ای با آنها کار می کردند یعنی این زبانهای برنامه نویسی طرفداران مخصوص خود را داشتند و یک قالب جامع واستانداردی برای آهنها در نظر گرفته نشده بود اما در سال 1980 وزارت دفاع امریکا با همکاری IEEE با هدف طراحی یک ز بان جدید استاندارد وفراگیر برای توصیف مدارهای دیجیتال و توسعه در مدارهای مجتمع پر سرعت (CPLD,FPGA) و همچنین برای انتقال اطلاعات سیستم های ذیجیتالی از شرکتی به شذرکت دیگر ویا به کشور دیگر را،به سه شرگت قدرتمند INTERMETRICS,TEXAS INSTRUMENT,IBM سپرد تا شش سال بعد یعنی در سال 1986 اولین نسخه استاندارد وتایید شده آن به بازار عرضه شود(یعنی همان نسخه VHDL86)ونسخه بعدی آن یعنی VHDL 93 در سال 1994 به بازار آمد که از آن زمان تا به امروز نسخه مورد استفاده کار بران قرار گرفته استواین در حالی است که هر چند سالی یکبار اصلاحاتی جزئی در آن صورت می گیرد.

در اینجا یک نمای کلی از سلسله مراتب کدنویسی Verilog را مشاهده می کنید.

Y_HARTContext Clause

Library Clause

Use Clause

Library Units

Package Declaration(optional)

Package Body(optional)

Entity Declaration

Architecture Body

جدول تقسیمات دستوراتVHDL از لحاظ ترتیبی (Sequential)یا همزمانی (Concurrency) بودن آنها.

Concurrent VHDL	Sequential VHDL	Con. & Seq.
process	If then else	Signal assignment
When else	case	type constant declaration
With select	Variable declaration	Function procedure call
Signal decleration	Loop statement	Assert statement
Block statement	Variable assignment	Signal attribute
	return	
	null	
	wait	

Y_chart

یکی از معروفترین نمایشهایی است که نگرشهای متفاوت وسلسله مراتبی مراحل طی یک سیستم دیجیتالی را نشان می دهد فرض کنید در یکی از بخش های شرکت DSPمسئول طراحی نسل جدیدی از تراشه های پردازش سیگناالی دیجیتالی هستیم.با توجه به هزینه های ساخت و محدوده زما نی که تراشه بایستی وارد با زار شود تا قابل رقابت باشد،می خواهیم قبل از اقدام به ساخت بررسی کنیم که آیا این تراشه می تواند کار برهای مورد نیاز کاربر را تامین کند یا خیر. بر اساس روند طراحی Y_chart می دانیم که رفتار تراشه در سطوح مختلف طراحی از جمله سطح عملیاتی،RTL ،سطح منطقی و... قابل توصیف به زبانVHDL است.

در مرحله اول توصیف رفتاری(behavioral) که شبیه سازی بر پایه آن می تواند صحت عملکرد مدار را نشان دهد ضروری است.

عملکرد تراشه را در سایر سطوح نیز می توان بررسی کرد،مزیت چنین رویکردی در این

است که می توانیم ارزیابی را مستقل از روشهای پیاده سازی فیزیکی (physical) انجام دهیم.

پس از بررسی عملکرد می توانیم طرح را به یک توصیف ساختاری(structural) متشکل از واحدهای اصلی تراشه مانند و,alu وregister,memory تبدیل نماییم.بار دیگر به کمک شبیه ساز می توان مطمئن شد که طرح ساختاری اینDSP به وسیله واحدهای انتخاب شده طرح دلخواه را به درستی انجام می دهد یا خیر.

همانطور که در شکل می بینید توصیف سخت افزاری نیز می تواند با درجات متفاوتی از جزئیات ایجاد شود.

این توصیف را می توان آن قدر تکمیل کرد تا به یک توصیف فیزیکی (Phisical) دست پیدا کنیم که در نهایت مشخصات ساخت را از آن استخراج نماییم.

تمام این مراحل به کمک زبان توصیف سخت افزاری VHDL و ابزارهای برنامه ریزی FPGA به سادگی اما با پشتکار وحوصله ،امکان پذیر است.

(1)

BEHAVIORAL
توابع انتقالی
عبارات بولی
Shift registerها
الگوریتم

(2)

stuctural
ترانزیستورها
گیتها
Regfister ها
پردازنده ها

(3)

phisical
سلولها
moduleها
تراشه ها
بردها

نحوه فراخوانی وبه کارگیری یک کتابخانه:

LIBRARY library_name;

Library ieee;

Library altera;

```
Library work;
Library std;
Or
Library ieee,altera,work,std;
```

نحوه فرا خوانی وبکارگیری یک package از داخل کتابخانه اش:

```
USE library_name.package_name.ALL;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_signed.all;
Use altera.maxplus2.all;
Or
Use altera.maxplus2.max2lib,ieee.st_logic_1164.all;
```

فرم کلی: ENTITY:

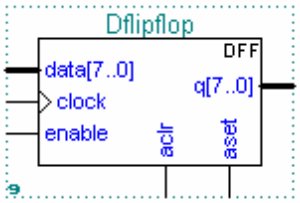
```
 ENTITY entity_name IS
        GENERIC( parameter_name : string := default_value;
                 parameter_name : integer := default_value);
        PORT(
              input_name , input_name     :IN STD_LOGIC;
              input_vector_name           :IN
STD_LOGIC_VECTOR(                                    *
high      downto low);
              bidir_name , bidir_name     : INOUT STD_LOGIC;
              output_name , output_name   : OUT STD_LOGIC);
END entity_name;
```

دستور ENTITY به منظور معرفی شکل ظاهری قطعه یا طرح از نوع پایه های ورودی وخروجی نه اتصالات داخلی استفاده می شود.

عبارت مربوط به GNERIC الزامی نیست.بلکه بنا به نیاز مدار در حال طراحی ویا تشخیص برنامه نویس ،می تواند نوشته شود .مثالهای زیر را ببینید:

مثال:نمونه ای از ENTITY یک فلیپ فلاپ نوع D .

```
ENTITY D_ FLIPFLOP IS
        PORT( Data      :in std_logic_vector(7 down to 0);
              Clock     :in std_logic;
              Enable    :in std_logic;
              Aclr      :in std_logic;
              Aset      :in std_logic;
              Q         :out std_logic_vector (7  downto 0));
End;
```



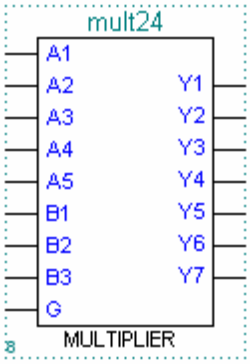
استفاده ز نام entity ویا خود entity در جلوی end آن اختیاری است.

مثال:

```
Entity generic_Example is
    Generic(delay:time:=10 ns);
    Port(a,b:in  std_logic;
         C:out std_logic);
End entity;
```

نمونه ای ازentity از یک Multiplier (ضرب کننده):

```
Entity multiplier is

Port(A:in std_logic_vectorr(4 downto 0);
     B:in std_logic_vector(2 downto 0);
    Y:in std_logic_vector(6 downto 0));
End multiplier;
```



سیگنال: (signal)

برای اتصال قسمتهای مختلف مدار دیجیتال،سیگنا ل استفاده می شود از نظر سخت افزار سیگنال مانند سیم اتصال کوتاه می باشد

طریقه اتصال سیگنالها به صورت همزمان:

signal <= expression; سیگنال Vhdl حکم سیم اتصال یا پایه قطعه را دارد.

در

مثال:

مقدار سیگنال bبه طور آنی به سیگنال aداده می شود.

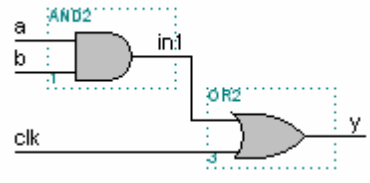
a <=b

فرم کلی : Architecture

```
ARCHITECTURE arch_name OF entity_name IS
        SIGNAL signal_name : STD_LOGIC;
        SIGNAL signal_name : STD_LOGIC;
BEGIN
   -- Process Statement
   -- Concurrent Procedure Call
   -- Concurrent Signal Assignment
   -- Conditional Signal Assignment-- Selected Signal Assignment
   -- Component Instantiation Statement
   -- Generate Statement
END arch_name;
```

مثال:

```
ARCHITECTURE arch_test of arch_example is
Signal  in1;std_logic;          محل تعریف اتصالات داخلی
Begin
in1<=a and b;                   شرح دادن عملکرد مدار
y <= 1n1 or clk;
end;
```



نوشتن نام architectureدر جلوی endان اختیاری است.

همین مدار را می توان به شکل زیر نوشت:

```
Architecture arch_test of arch_Example is
Signal in1: std_logic;        محل تعریف اتصالت داخلی
Begin
Y    <= in1 or clk;              شرح عملکرد مدار
 In1 <= a and b;
End;
```

اختلاف ظاهری این دو برنامه در جابجا نوشتن خطوط برنامه در بدنه Architectureاست اما نتیجه کار هیچ فرقی نمی کندو علت آن هم این است که بدنه Architecture،به شکل Concurrencyیا هم زمان عمل می کند یعنی بر خلاف دیگر برنامه ها مانند C، Pscal,Assemblyو...که خط به خط برنامه را پیش می لرند،vhdlدارای این ویژگی منحصر به فرد است که در داخل برنامه Architectureخود میی تواند تمامی دستورات وعبارات را به طور هم زمان و یکجا مقدار دهی وتحلیل کند)البته در واقعیت با یک تاخیر بسیار کوچک،زیرا گیتهای منطقی نیز در عمل دارای کمی تاخیر (delay)می باشند.

فرم کلی دستور CASE :

```
CASE expression IS
        WHEN constant_value =>
             statement;
             statement;
         WHEN constant_value =>
              statement;
              statement;
.
.
        WHEN OTHERS =>
             statement;
```

```
        statement;
END CASE;
```

تمامی این دستورات به صورت همزمان انجام می شوند.

دستور CASEچون یک دستور SEQUENTIALاست یعنی عبارات داخلی آن به ترتیب وخط به خط اجرا می شوند لذا حتما می بایست داخل بدنه PROCESSنوشته شود.

مثال:

```
CASE SELECT IS
        WHEN        '0'  => b <= 3;
        When        1| 2 => b <= 2
        when others      =>  b <=0;
End case;
```

عملگر (|) به معنی (یا) می باشد

مثال:

```
Case sel is
        When 0              =>q <=5;
        When 1 to 17        =>q <= 7;
        When 23 downto 18   => q <=2;
        When others         => q <=0;
End case;
```

محدوده حالتهای expressionنباید همپوشانی داشته باشند.

مثال:

```
Case int is
When "000000"    => a  <="000";
                    B <="111";
                   C  <= "001";
When  "001110"   => a   <= "011";
 When others     =>  qout<="001";
End case;
```

توجه داشته باشید تمام این caseها داخل بدنه processنوشته می شوند.

تخصیص (انتساب)سیگنالها به صورت شرطی:

```
label:
        signal <= expression WHEN boolean_expression ELSE
                  expression WHEN boolean_expression ELSE
                  expression;
```

مثال:

```
dbus <= dwhen en='1' else 'z';
  q  <=a when sel='0' wlse b;
```

هرگز این دستور رانبایید در داخل پروسس بنویسید.

```
   Z  <=A when sel='00'   else
         B when sel ='01'  else
         C when sel="10"  else
         D;
```

فرم کلی تعریف ومقدار دهی اولیه constant:

```
CONSTANT constant_name : type_name := constant_value;
```

مثال:

```
   Constant delay :time:=5 ns;
   Constant a : a_type:="1001";
```

فرم کلی سیگنال داخلی:

```
SIGNAL signal_name : type_name
```

مثال:

```
   Signal in1:std_logic;
```

فرم کلی متغیر : (variable)

```
VARIABLE variable_name : type_name;
```

مثال:

```
Variable count,temp : integer;
```

فرم کلی تعریف :process

معمولا با توجه به ساختار مدار می توان یکی از دو نمونه زیر را استفاده کرد.

Process-1با حساسیت(sensitivity list):

```
process_label:
PROCESS ( signal_name , signal_name , signal_name )
    VARIABLE variable_name : STD_LOGIC;
    VARIABLE variable_name : STD_LOGIC;
BEGIN
        -- Signal Assignment Statement
        -- Variable Assignment Statement
        -- Procedure Call Statement
        -- If Statement
        -- Case Statement
        -- Loop Statement__
END PROCESS process_label
```

خود دستور process، concurrentاست)یعنی داخل بدنه Architectureنوشته می شود(اما داخل بدنه اش بر عکس بدنه architectureبه ضورت sequentialعمل می کند.(البته با اپسیلن تا خیر.)

Process- 2بدون لیست حساسیت به کمک عبارات wait,wait until,wait for,wait on ():

```
process_label: _
_PROCESS ( signal_name , signal_name , signal_name )
        VARIABLE variable_name : STD_LOGIC;
        VARIABLE variable_name : STD_LOGIC;
  BEGIN
     WAIT UNTIL clk_signal = '1';
    -- Signal Assignment Statement
    -- Variable Assignment Statement
    -- Procedure Call Statement
    -- If Statement
```

غیرقابل سنتز

-- Case Statement

-- Loop Statement

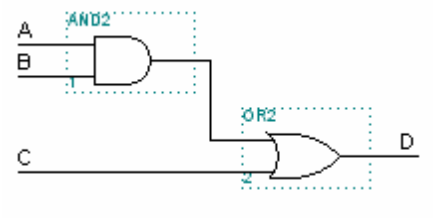
END PROCESS process_label;

خود دستور process، concurrentاست یعنی داخل بدنه Architectureنوشته می شوداما داخل بدنه اش بر عکس بدنه architectureبه ضورت sequentialعمل می کند(البته با اپسیلن تا خیر.)

می توان به جرات گفت که 90در صد کد نویسی vhdlبرای مدارات سخت افزاری دیجیتال ،توسط دستور processنوشته می شودو این یعنی اینکه processیکی از پر کار برد ترین دستورات VHDLمی باشد.

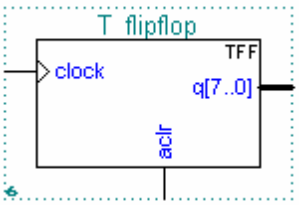
مثال:دستور PROCESSرا می توان برای مدارهای ترکیبی (COMBENTIONAL) اجرا نمود به این صورت که ورودی های ان راداخل لیست حساسیت قرار داد تا با کوچکترین تغییر وضعبت آنها ، PROCESSاین تغییر وضعیت را حس نموده و خروجی مورد نظر را تولید نماید.

```
Process(A,B,C,)
    Begin
        D<=(A and B) OR C;
End  process;
```



مثال :دستور processرا می توان برای مدارهای ترتیبی (sequintial)نوشت به این صورت که ورودی clockودر برخی موارد ،ورودی های enable,reset,presetرا در داخل لیست حساسیت processقرار داد تا به محض تغییر یکی از آنها ،یکبار دستور processاز بالا تا پایین عبارات ترتیبی داخلش را اجرا کند وخروجی مورد نظرش را تولید نماید.

```
Process(clock,aclr)

  Begin
      If  aclr='1' then
          q<=( others=>'0');
       elseif(clock'event  and clock='1')then
```



```
                q<=q=q+1;
        end if;
end process;
```

عبارت others =>'0' به این معنی است که تمامی بیتهای q را صفر کن وبرای سادگی کار از این فرم نوشتن استفاده شده و می توان به جای آن از فرم نسبتا طولانی تر q=>"00000000"; نیز استفاده نمود.

عبارت clock'event and clock='1' به معنی بالا رونده بودن لبه پالس clock می باشد . در واقع جمله clock'event' به معنی تغییر در لبه پالس clock می باشد و جمله clock='1' به مفهوم بالا رونده بودن لبه پالس ساعت می باشد.

فرم کلی دستور : if…then…else

```
IF expression THEN
    statement;
    statement;
 ELSIF expression THEN
      statement;
      statement;
  ELSE
       statement;
       statement;
END IF;
```

عبارات if به صورت ترتیبی خط به خط اجرا می شوند.

اگر if ساده بدون else باشد فقط یک عبارت را ارزیابی ودر صورت true بودن اجرا می کند و از if خارج می شود و برنامه ادامه می یابد ولی اگر if…else داشته باشیم بین دو عبارت یکی را انتخاب میکند.

مثال :

```
If sel ='1'then
    q<=a;
 else
```

```
    q<= b;
end if;
```

دستور if با دستور case با وجود شباهت عملکرد اما در VHDL سه فرق اساسی وجود دارد که بهتر است انها را مد نظر داشته باشیم:

-1عبارت CASE مشابه IF می باشد ووسیله ای برای تصمیم گیری وانشعاب شرطی است ولی نتیجه شکل شرط به شکل BOOLEAN می باشد)یعنی ,FALSE، TRUE 9 در حالیکه مورد CASE نتیجه شرط می تواند TYPE های مختلفی از جمله INTEGER ، ,ENUMERATED، BIT_VECTOR، Std_logic و ...باشد.

-2موقعی case به کار می رود ،تمامی انشعابها دارای اولویت مساوی می باشند در حالیکه در if این طور نیست.

-3در case شرط تست می گردد ومستقیما یکی از انشعابها انتخاب می شود ولی در دستور if چند انشعاب است که به ترتیب اولویت پشت سر هم اجرا می شوند.

فرم کلی انتساب در سیگنال ها:

```
Signal _name<=expression;
```

مثال :

```
qout<= a;
count<=cnt+1;
b<="00011010";
sum<=A Xor B Xor C;
int<=123;
```

فرم کلی انتساب در متغیر ها:

variable_name := expression ;

مثال:

```
Var:=var2;
Var2:=sig2;
A:=16;
Cnt:=cnt+1;
```

دستور wait وانواع آن:

همان طور که در قسمت processبیان شد نوع دیگر استفاده از این دستور،نوع لیست بدون حساسیت استکه بجای آن هم می بایست از یکی از انواع دستور waitبهره بگیریم.
البته این دستور غیر قابل سنتز است.

```
WAIT UNTIL clk_name = '1';
WAIT ON clk_name = ‘1’;
WAIT FOR time_value ns ;
WAIT;
```

مثال:این دستور در بدنه sequentialنوشته می شود..

Wait until clk ='1'; ~ wait until (clk event and clk='1'); ~wait on clk='1';

این سه دستور معادل هم هستند و به این معنی می باشند که آن قدر منتظر می مانند تا یک لبه بالا رونده برای clkبیابد.

```
Process
    Begin
        Wait until clk ='1';
         C> = a nor b;
End process;
```

دستور waitبه تنهایی معمولا در انتهای عبارات processاستفاده می شود و به معنی توقف کامل اجرای برنامه است در واقع waitبه تنهایی یعنی انتظار بی پایان.

فرم کلی دستور with…select:

```
label:
WITH expression SELECT
        signal <= expression WHEN _ constant_value,
                  expression _WHEN constant_value,
                  expression _WHEN constant_value,
                 expression _WHEN constant_value;
```

مثال:

```
With sel select
        z<=a when"00",
            b when "01",
```

```
        c when "10",
        d when "11";
```

تمامی حالتهای ممکن sellمی بایست پوشش داده شود حتی اگر لازم باشد می توان از othersنیز استفاده نمود.

مثال:

```
With inp select
        Target <=value1 when"000',
                Value2  when "001"| "110"|"011",
                Value3 when others;
```

فرم کلی دستور: for…loop:

```
loop_label:
FOR index_variable IN discrete_range LOOP
statement;
statement;
END LOOP
```

عبارات داخل for…loopبه صورت خط به خط اجرا می شوند
برعکس عبارات for…generate که به صورت همزمان اجرامی شوند.

مثال :

```
loop_label;
For I in 0 to 4 loop
        If a(i)='1' then
                q(i)<=b(i);
        end if;
end loop;
```

فرم کلی دستور : for…generate

```
generate_label:
FOR index_variable IN discrete_range GENERATE
```

statement;
statement;

عبارات for ... generate به صورت همزمان انجام می شوند بر عکس عبارات داخل for....loop که به صورت خط به خط اجرا می شوند

END GENERATE;

مثال:

```
Gen_test1:for I in 0 to 7 generate
        Sum(i)<=a(i)xorb(i)xorc(i);
        End generate;
```

فرم کلی دستور : if...generate

```
generate_label:
stateemnt;
    statement:
IF expression GENERATE
```

Statement;
Statement;

فرم کلی دستور : if... generate
(عبارت if... generate به صورت همزمان اجرا میشوند.)

```
 Gen_test2;if i<8 generate
        sum<=a(i)xorb(i)xorc(i);
      end generate;
```

فرم کلی تعریف یک آرایه یک بعدی یا چند بعدی:

```
TYPE type_name IS ARRAY <index_value>
 OF element_type;
STD_LOGIC_VECTOR( high DOWNTO low );
BIT_VECTOR( high DOWNTO low ) ;
INTEGER RANGE low TO high ;
```

I. انواع آرایه های یک بعدی که می توان سیگنالها و متغیرها و حتی ثابتها را از این نوع انتخاب نمود.

مثال:

```
Signal a:std_logic_vefctor(7 downto 0);
```

معادل است

با:

A(7),a(6),a(5),a(4),a(3),a(2),a(1)a(0)

مثال:

Type a_type is arry(3 downt 0)of std_logic;

Type my_int is integer range 0 to 15;

⟶ Signal d:a_type;

Variable q:my _int;

مثال:

Type var is array(0 to 7) of integer;

⟶ Constant addr:var:=(5,10,2,4,6,12,7,14,);

II.تعریف آرایه دو بُعدی

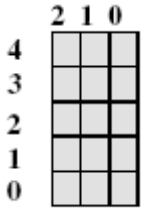
TYPE array_type_name IS ARRAY (_high DOWNTO low)
OF type_name;

TYPE array_type_name IS ARRAY (integer RANGE <>)
OF type_name;

مثال:

Type my_memory is array(4 down to 0) of std_logic_vector(2 downto 0);

⟶ Signal RAM: MY_MEMORY;



مثال :در بعضی از مواقع آسان تر است که در موقع تعریف نوع آرایه ،ابعاد آرایه را مشخص نکنیم.برای اسن کار از الگوی نوع دوم استفاده م کنیم.

TYPE MATRIX IS ARRAY (INTEGER RANGE<>) OF INTEGER:

⟶ Variable mat:matrix(2 downto -8):=(3,5,1,4,,7,9,12,14,20,18,);

فرم کلی نوع شمارشی: enumerated

TYPE enumerated_type_name IS (name , name , name);

نوع شمارشی شامل لیستی از نامها وکاراکترها است که با آنها می توان مدارهای دبجیتا ل را

متناسب با عملیاتی که انجام می دهند و انواع مقادیر خروجی ای که داریم بگیرند، با توجه به یاز برنامه نویس مدل سازی نمود.

این typeبیشتر در طراحی ماشین حال (state machin)استفاده می شود.

مثال:

type david_type is ('0','1','z'):

type state_type is (s0,s1,s2,s3):

type mano_pc is(add,sub,shiftr,shiftl,mult,div,inc,load,store);

type my_state is (start,idle,waiting,run);

⟶ signal msh : david;

signal state:state_type:=s1;

variable ALU_inputs:mano_pc;

فرم کلی تعریف subtype:

SUBTYPE subtype_name IS type_name RANGE low_value TO high_value;

SUBTYPE array_subtype_name IS array_type_name(high_index DOWNTO low_index);

subtypeدر واقع زیر مجموعه ای از typeاست یعنی نوع محدود شده ودارای عناصر کمتر type می باشد.

مثال:

subtype my_int is integer range 0 to 511;

subtype byte is bit_vector(7 downto 0);

⟶ signal a:my_int;

signal b:byte;

توجه داشته باشید که type وsubtype در قسمت اعلانات architecture یعنی قبل از begin تعریف می شوند.

فرم کلی تعریف وبه کار گیری component :

COMPONENT component_name

GENERIC(parameter_name : string := default_value; } بستگی

به

parameter_name : integer := default_value) انتخاب

برنامه

نویس دارد(اختیاری)

```
PORT(
        input_name , input_name : IN STD_LOGIC;
        bidir_name , bidir_name : INOUT STD_LOGIC;
        output_name , output_name : OUT STD_LOGIC);
END COMPONENT
```

نامهای ورودی وخروجیcomponentدقیقا باید مشابه نامهای ورودی وخروجی entity قعه مورد نظر باشد.

در ضمن تعریف قطعه یا همانcomponent باید در قسمت اعلانات architecture صورت بگیرد.

مثال:

```
component full_adder
                port(a.b,c,:in std_logic;
                        sum,carry:outstd_logic);
end component;
```

مثال:

```
component or2
                port(in1,in2:in std_logic;
                        out:out std_logic);
end component;
```

فرم کلی نمونه گیری از یک قطعه (component instance)

```
instance_name: component_name
                PORT MAP ( component_port => connect_port ,
                                component_port => connect_port );
instance_name: component_name
```

PORT MAP (connect_port , connect_port);

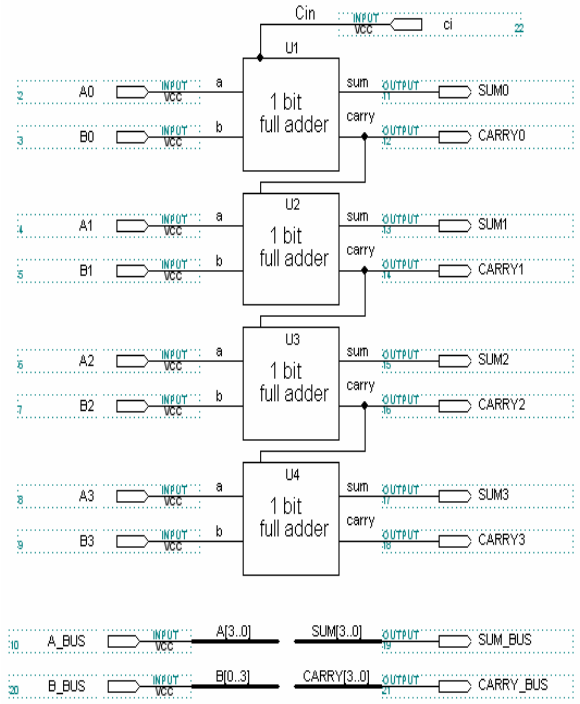
گاهی موقع لازم است از یک یا ندین قطعه متفاوت دیگر در طرح خود استفاده کنیم لذا در این مواقع دستور component به کمک ما خواهد آمد تابتوانیم این قطعات رابه طرح اصلی خود معرفی کنیم ودر نهایت به کمک دستور component instance از این قطعات استفاده کنیم .

نوع 1 تعریف کامل تری است زیرا به کمک آن می توانیم از برخی پایه های قطعه نمونه گرفته شده استفاده نکنیم در حالیکه در نوع 2

با وجود اینکه روش اتصال سیمها (سیگنالها) به پایه های قطعه نمونه گرفته شده آسانتر است اما در این روش باید از همه پایه ها استفاده شود همچنین امکان جابجا نوشتن پایه ها از لحاظ ترتیب تعریف آنها وجود ندارد.

در ضمن componenet instance در بدنهarchitcture نوشته می شود.

مثال:




```
u1: full_adder
     port
map(A(0)=>a,B(0)=>b,
                  cin=>ci,

sum(0)=>sum,

carry(0)=>cout);
u2:full_adder
     port
map(A(1)=>a,B(1)=>b,
                carry(0)=>ci,
                 sum(1)=>sum,
                 carry(1)=>cout);
u3:full_adder
    port map (A(2)=>a , B(2)=>b,
              CARRY(1)=>ci,
```

```
                SUM(2)=>sum,
                CARRY(2)=>cout);
u4: full_adder
   port map (A(3)=>a ,B(3)=>b,
                CARRY(2)=>ci,
                SUM(3)=>Sum,
                CARRY(3)=>cout);
```

اگر به عنوان مثال بخواهیم برای نمونه برداری از u3 از نوع2 دستور map استفاده کنیم،بدین شکل باید عملکرد:

```
U3:full_adder port map(A(2),B(2),CARY(1),SUM(2),CARRY(2) );
```

فرم کلی تعریف و به کار گیری PACKAGE :

```
PACKAGE pack_name IS
.
Pack_declarations
.
END pack_name ;
-------------------------------------------------
PACKAGE BODY pack_name IS
.
Pack_declarations
.
END PACKAGE BODY pack_name ;
```

مثال:

```
Pacage my_int is
    Type small_int is integer range 0 to 15;
```

End package;

حال می خواهیم از این type کهدر داخل pacage ای به اسمmy_unit است واین pacage به طور اتوما تیک در داخل کتا بخانه work (کتابخانه work یک کتابخانه پیش فرض برای قرار گیری اتوماتیک تمامی برنامه های کاربر می باشد)قرار گرفته در برنامه دیگری به نام small_adder استفاده می کنیم پس اینطور می نویسیم:

```
Use work.my int.all;
Entity smaller_adder is
Port (a,b :in small_int;
        S: out small_int);
End;
```

مثال: در اینجا قصد داریم دو تابع منطقی nandو xor دو ورودی را داخل یک package با نامlogical_pack قرار دهیم:

```
Library ieee;
Use ieee.std_logic_1164.all;
Package logical_pack is
  Component NAND2
        Port(in1,in2,:in std_logic;
                Out1:out std_logic);
End component;
End package;
Library ieee;
Use ieee.std_logic_1164.all;
Package body logical_pack is
      Entity nan2
        Port(in1,in2:in std_logic;
                Out1:out std_logic);
end NAND2;
Architecture nand2 arch of  NAND2 is
  begin
```

```
        out1<= in 1 nand in2;
end nand2arch;
Enttity xor2
  Port(in1,in2:in std _logic;
              out : out std_logic):
end xor2;
Architecture xor2arch of xor2 is
 begin
      out1<=in1 xor in2;
    end xor2arch;
end package body;
```

فرم کلی تعریف وفراخوانی (function) :

```
FUNCTION func_name ( func_inputs : in inputs_type ) RETURN
output_var_type IS
        VARIABLE var_name : var_type ;
```

⟶ .چون، تابع هیچگاه خروجی سیگنال نمی دهد بلکه

. باید حتما از یک متغیر تعریف شده داخل خود به

. عنوان خروجی برگشت استفاده کند.

.

```
BEGIN
.
.
```

. تابع در architecture, package ویا اعلان میشود

.

```
Output_var := statement ;
```

Return ; ⟶ هر تابع بر عکس procedure، فقط یک خروجی دارد

output

```
End func_name;
```

مثال:

```
Function analysis(value,maxmin:intger) return integer is
    Begin
        If  value>max then return max;
            Elseif value<min then return min;
                Elde ruturn value;
        End if;
End function;
```

فراخوانی تابع در جملات تخصیص (انتساب)متغیرو سیگنال نیز می تواند صورت بگیرد مانند:

```
Var1:=analysis(current_temperature+increment,10,100);
Count<=carry(a,b,c);
```

مثال:

```
Function carry (bit,bit2,bit3:in std_logic)return std_logic is
        Variable result :std_logic;
Begin
      Result:=(bit1 and bit2) or (bit1 and bit3)or (bit2 and bit3);
       Return result;
End carry;
```

فرم کلی تعریف وفراخوانی procedure :

```
PROCEDURE procedure_name ( procedure_inputs : in inputs_type ;
                                    procedure_outputs : out outputs_type )
IS
        VARIABLE var_name : var_type ;
        VARIABLE var_name : var_type ;
         .
         .
```

```
        .
    BEGIN
        .
        .
        .
     Output_var := statement ;
    Output_var := statement ;
END PROCEDURE procedure_name ;
```

Procedure همانندfunction یک برنامه فرعی برای انجام محاسبات وroutinهاایتکراری به کار می رود

فرقprocedureبا function در این است که procedure می تواند هر تعداد خروجی که بخواهیم داشته باشد در الیکه تابع فقط مجاز است به برگرداندن یک خروجی همچنین procedure داده هایی از نوع inout نیز داشته اشد یعنی اطلاعات واردprocedure شود سپس عملیاتی روی آن انجام گیردو در نهایت به خروجی فرستاده شود.

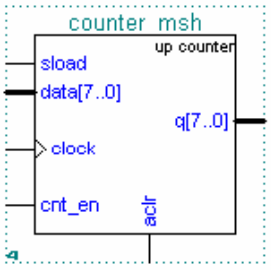
مثال:

```
Procedure full_adder4(a, :in std_logic_vector(3 doento 0);
                              Result: out std_logic_vector(3 downto 0);
                              Over flow;out Boolean) is
      Variable sum: std_logic_vector(3 doento 0 );
      Variable carry :bit='0';
Begin
   For I in 0 to3 loop
       Sum(i):=a(i)xorb(i)xor carry;
        Carry:=(a(i)andb(i))or(carry and (a(i) or b(i));
    End loop;
    Overflow:=carry='1';
End procedure;
```

دراینجافرم کلی یک شمارنده (counter) را ملاحضه می کنید:

```
LIBRARY ieee;
```

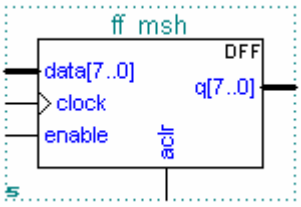
```
USE ieee.std_logic_1164.all;
ENTITY entity_name IS
        PORT
       (
                data_input_name : IN INTEGER
RANGE 0 TO count_value;
                clk_input_name : IN STD_LOGIC;
                clrn_input_name : IN STD_LOGIC;
                ena_input_name : IN STD_LOGIC;
                ld_input_name : IN STD_LOGIC;
                count_output_name : OUT INTEGER RANGE 0 TO
                count_value



       );
END entity_name;
ARCHITECTURE arch_name OF entity_name IS
      SIGNAL count_signal_name : INTEGER RANGE 0 TO
count_value;
   BEGIN
        PROCESS ( clk_input_name , clrn_input_name )
        BEGIN
           IF clrn_input_name = '0' THEN ⟶                معروف به

                                                ASYNCHRONOUSE  RESET
                                چون قبل از دستور بررسی لبه کردن لبه clk ،نوشته شده
                                است یعنی مستقل از آمدن لبه clk مدار را reset می کند.
                        count_signal_name <= 0
```



```
                ELSIF ( clk_input_name'EVENT AND clk_input_name =
'1'_) THEN
                        IF ld_input_name = '1' THEN
                                count_signal_name <=data_input_name;
                         ELSE
                                IF ena_input_name = '1' THEN
                                        count_signal_name <=
count_signal_name + 1;
                                 ELSE
                                        count_signal_name <=
count_signal_name;
                                END IF;
                        END IF;
                END IF;
        END PROCESS;
count_output_name <= count_signal_name;
END ARCH_NAME;
```

در اینجا فرم کلبیک کد نویسی یکFLIP_FLOP را ملاحضه می کنید:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY entity_name IS
        PORT
            d_input_name : IN STD_LOGIC;
            clk_input_name : IN STD_LOGIC;
             clrn_input_name : IN STD_LOGIC;
            ena_input_name : IN STD_LOGIC;
            q_output_name : OUT STD_LOGIC
         );
END entity_name;
```



```
ARCHITECTURE arch_name OF entity_name IS
        SIGNAL q_signal_name : STD_LOGIC;
   BEGIN
        PROCESS ( clk_input_name , clrn_input_name )
        BEGIN
                IF clrn_input_name = '0' THEN
                      q_signal_name <= '0';
                 ELSIF ( clk_input_name'EVENT AND clk_input_name =
'1' ) THEN
                          IF ena_input_name = '1' THEN
                                q_signal_name <= d_input_name;
                           ELSE
                                q_signal_name <= q_signal_name
                          END IF;
                END IF;
        END PROCESS;
        q_output_name <= q_signal_name;
END arch_name;
```

فرم کلی کد نویسی با ماشین حالت با asynchronous reset:

```
ENTITY machine_name IS
PORT(
        clk : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        input_name , input_name : IN STD_LOGIC;
        output_name , output_name : OUT STD_LOGIC);
END machine_name;
```

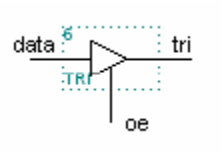
```
ARCHITECTURE arch_name OF machine_name IS
        TYPE STATE_TYPE IS ( state_name , state_name , state_name );
         SIGNAL state : STATE_TYPE;
BEGIN
      PROCESS (clk)
      BEGIN
         IF reset = '1' THEN ──────────►
```

معروف به asynchronouse reset چون قبل از دستور بررسی
به clk نو شته شده است یعنی مستقل ازآمدن clk مدار راrest می کند
اگر این دستور بعد ازدستور آمدن لبه clk بنویسد در این حالت گفته
می شود Synchronous reset زیرا باید'1' شدن Reset با آمدن لبه
clk همزمان صورت بگیرد.

```
 state <= state_name;
            ELSIF clk'EVENT AND clk = '1' THEN
                  CASE state IS
                        WHEN state_name =>
                              IF condition THEN
                                    state <=state_name;
                               END IF;

                        WHEN state_name =>
                              IF condition THEN
                                 state <=state_name;
                              END IF;
                        WHEN state_name =>
                              IF condition THEN
                                 state <= state_name;
                              END IF;
```

```
                END CASE;
            END IF;
    END PROCESS;
    WITH state SELECT
            output_name <= output_value WHEN state_name,
                           output_value WHEN state_name,
                           output_value WHEN state_name;
END arch_name;
```

در اینجا فرم کلی یک tri state buffer را ملاحضه می کنید:




```
USE ieee.std_logic_1164.all;
ENTITY entity_name IS
        PORT
        (
         oe_input_name : IN STD_LOGIC;
         data_input_name : IN STD_LOGIC;
          tri_output_name : OUT STD_LOGIC
        );
END entity_name;
ARCHITECTURE arch_name OF entity_name IS
BEGIN
        PROCESS ( oe_input_name , data_input_name )
        BEGIN
                IF oe_input_name = '0' THEN
                        tri_output_name <= 'Z';
                ELSE
                        tri_output_name <= data_input_name;
```

```
                END IF;
            END PROCESS;
END arch_name;
```

_____ _____ _____

_____

فصل ششم

روند طراحی یک مدار قا بل برنامه

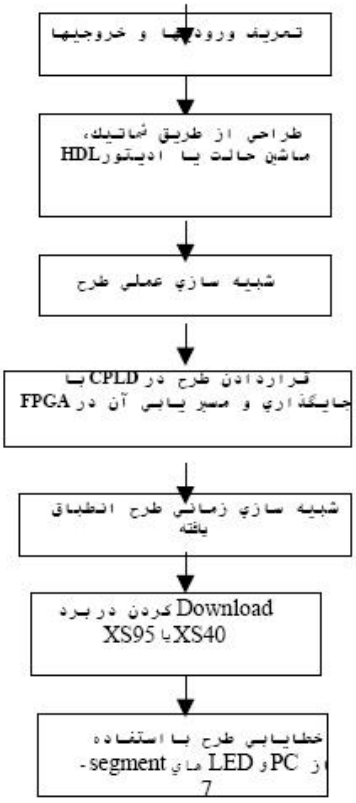
ریزی بر اساس Xilinx foundation :

نرم افزار Xilinx foundation محیطی جهت ایجاد برنامه هایی برای توصیف طرح منطقی مورد نظر می باشد.روند طراحی با استفاده از نرم افزار foundation به این ترتیب است:

7-1طرح مورد نظر با استفاده ازاد یتور شماتیک یا ادیتور ماشین حالت یا ادیتور متنی HDL (ABELیا VHDL) وارد می شوند.طرح مورد نظر می تواند توسط یکی از آنها یا تر کیبی از آنها ایجاد شود.

7-2یک سیمولاتور (شبیه ساز) عملی عملکرد یک طرح کامپایل شده را چک می کند وبه شما اجازه می دهد که تا نتایج را ببینیدو صحت یا عدم صحت نتایج را بررسی کنید .در صورت بروز هر گونه خطائی می توان به محیط ادیتوری شماتیک ،HDL یا ماشین حالت بر گشته خطا ها رااصلاح کنید

روند طراحی برای XC9500/XC4000



شکل 7-1تغییرات جریان طراحی دیجیتال هنگام استفاده از نرم افزار Xilinx Foundation برای xc9500 یاXC4000 FPGA :

7-3ابزار اجرای نرم افزار Foundation ابتدا لیست گیتها واتصالات ایجادشده را به یک فایل با فرمت باینری تبدیل می کند که جهت برنامه ریزی FPLD استفاده خواهد شد.در این مرحله است که یکDevice باید انتخاب شودمانند خانواده های XC95108وXC 400 و5XL برای Deviceهای XC9500 ،برنامه مورد نظر طراح در داخل یک CPLD قرار می گیرد ولی درXC4000 ،طرح در داخل یک FPGA قرار می گیرد.به این ترتیب که گیتها داخل CLB مشخص قرار می گیرند و مسیر یابی وسیم پیچی ها از طریق PSMها انجام می شود.

7- 4بعد از اینکه ابزار اجرائی نرم افزار FOUDATION تاخیرهای مربوط به گیتها وعمل مسیر یابی را مشخص کردند،شبیه سازی خصوصات زمانی طرح با یک MAPPING مشخص را دریک FPLD انجام می شوند.

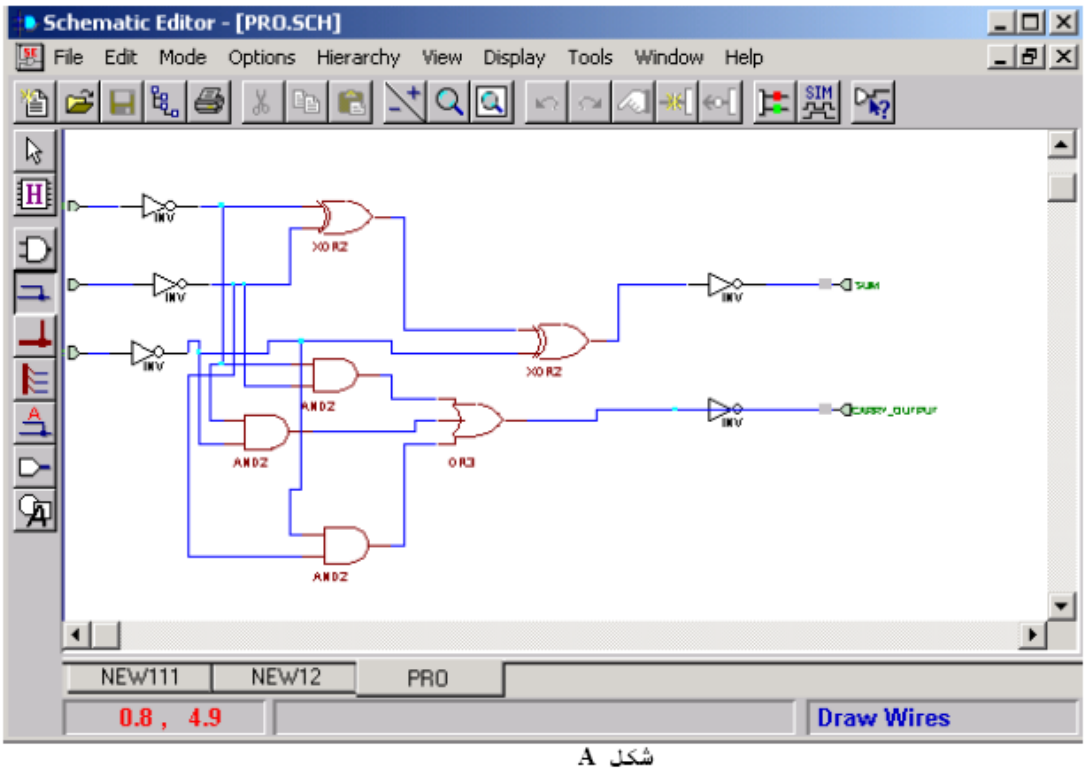
7-5 با وارد کردن طرح ورودی ها به یک بورد xs95 یا xs40 از طریق کابل پورت موازی کامپیوتر عمل خطا یابی (Debugging) انجام می شود

7-6مثالی در مورد نحوه برنامه ریزی یک بورد XS40 (جمع کننده تک بیتی):

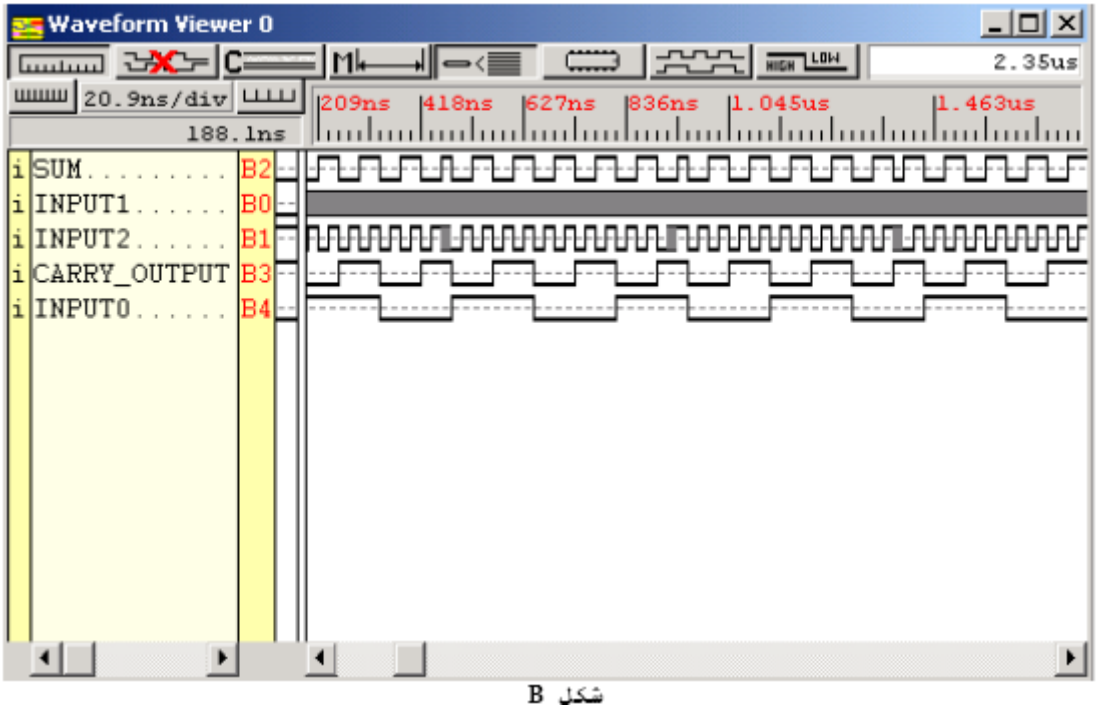
جدول حقیقت برای یک جمع کنندهٔ ۲ بیتی همراه باکری ورودی و خروجی :

Input 1	Input 0	Carry input	Sum output	Carry output
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

ابتدا طرح را به وسیله ادیتور شماتیک به نامADDL-40 تعریف می کنیم که طرح را برای XC4005XL ایجاد می کند.ادیتور شماتیک را فعال می کنیم وگیتها را مطابق شکل Aبه هم متصل می کنیم.شکل Aیک جمع کننده تک بیتی رابا اضافه کردن بافرهای ورودی وخروجی نشان می دهد.



شکل A



شکل B

وقتی شماتیک کامل شد ، آن را به عنوان ADD-SCH ذخیره می کنیم .سپس یک netlist را با انتخاب Netlist option-export ایجاد می کنیم آن را در فرمت EDIF 200 ارسال می کنیم.(همچنین با انخاب Option-integrity test می توان درستی همه چیز را قبل از

ادامه عملیات چک کرد) سپس از ادیتور روی کلید درflow tab از صفحه project manager کلیک می کنیم.

با انتخاب signal-Add signal از منو ،سیگنا های ورودی وخروجی را به صفحهWaveform viewer اضافه می کنیم. با انتخاب signal-Add Simulators از منو ورودی ها را به پا یین ترین سه بیت شمارنده (B0,B1,B2) متصل می کنیم .در نها یت روی دکمه SIMULATORE درToolbar logic کلیک می کنیم .شکل موجهای نشان داده شده در شکل Bظاهر می شوند.

می توان از دکمه وScrollbar در صفحه waveform جهت باز کردن ومتمرکز کردن شکل موج ها استفاده کرد)چنانچه با دقت بررسی کنیم می بینیم که نتیجه شبیه سازی با آنچه در جدول حقیقت نشان داده شده بود ،مطابقت دارد .بعد از چک کردن عملکرد جمع کننده تک بیتی می توانیم از سیمولاتور (شبیه ساز) خارج شده وفایل را برای XC05xl FPGA کامپایل کنیم.

در ابتدا لازم است که ترمینالهای ورودی وخروجی را به پینهای فیزیکی I/O درxx4005cl اختصاص دهیم .به این منظور از ادیتور HDL جهت ایجاد فایل ADDL-40.UCF وافزودن خطوط زیر به آن استفاده می کنیم.

```
NET INPUT 1              LOC=P64;
NET INPUT 0              LOC=P45;
NET CARRY-INPUT          LOC=P44;
NET SUM                  LOC=P25;
NET CARRY-OUTPUT         LOC=P26;
```

حال روی دکمه جهت شروع کردن کامپایل کردن طرح کلیک می کنیم.سپس در صفحه IMPLEMENT Design که ظاهر خواهد شد.روی دکمه option کلیک می کنیم. در صفحه option در قسمت user constraints ،نام ucfای را که ایجاد کرده ایم وارد می کنیم. مثلا c:\XCPROJ\ADDL-40\ADDL-40.UCF .(یا از دکمه BROWSE جهت انتخابUCF استفاده می کنیم.) سپس OK را برای باز گشت به صفحهimplement Design کلیک کرده وrunرا در این مرحله ،در این مرحله باید یک فایل addl-40.bit داشته باشیم که بتوانیم در بورد xs40 download کنیم.

1-چه فضائی از xc4005xl FPGA جهت ساخت یک جمع کننده یک بیتی اشغال می شود؟

2- آیا کامپایلر ورودی ها و خروجیها را به پینهایی که ما درخواست کرده بودیم اختصاص می دهد؟

ما می توانیم پاسخ این سوالها را از فایلهای گزارش که توسطFoundation implemention ایجاد می شود در یابیم.

در قسمت سمت راست صفحه ،project manager Report tab را انتخاب می کنیم سپس روی آیکون Implemention report file ،دو بار کلیک میکنیم،صفحهreport browser ظاهر خواهد شد.روی هر کدام از آیکونها دو بار کلیک می کنیم تا محتویات مربوط به آن گزارش را مشاهده کنیم..این گزارشها اطلاعات زیر را خلاصه می کنند:

7-6-1گزارش ترجمه:

هر مشکلی که در هنگام تبدیل netlist به فرمت داخلی که توسط ابزار foundation implemention استفاده می شود،مشاهده می شود ،در این قسمت لیست می شود.

در این قسمت چک کردن قوانین طراحی وفایل ucf جهت خطا یابی،نیز صورت می گیرد .معمولا بیشتر خطاهایی که در این قسمت می بینید مربوط به عبارات غیر قابل قبولی است که وارد کرده اید.

7-6-2گزارش انطباق:

در اینجا در مورد اینکه چه نوع بهینه سازی هایی روی netlist انجام شده است،اطلا عاتی به دست خواهید آورد.گیتهای منطقی جابجا واضافه می شوند به طوریکه بدون تغییر عملکرد در مدار،آن را بهینه سازی می کنند.یک مثا ل از جابجائی منطقی زمانی اتفاق می افتد که یک گیت را در طرح قرار می دهید ولی خروجی رابرای هیچ چیز استفاده نمی کنید.مثال دیگر گیتANDاست که با یک ورودی متصل شده به "0" منطقی (GND) است که نتیجه خروجی همیشه LOW خواهد بود.

هم چنین گزارش می دهد که گیتهای منطقی چگونه گروه بندی شده در CLBهایFPGAقرار می گیرند.

7-6-3 گزارش جابجایی ومسیر یابی:

این گزارش،انتخابهایی را که ما جهت اثر گذاری بر روند جایگذاری ومسیر یابی انجام داده ایم،یادداشت میکند .همچنین خطاها واخطار ها را لیست کرده وزمان صرف شده در جایگذاری ومسیر یابی های مختلف را ثبت می کند.این گزارش همجنین مشخصاتی آماری گوناگونی عملکرد زمانی مسیر یابی را گزارش می دهد.

ولی مهم تر از همه ،این گزارش پاسخ سوال اول ما را در مورد اینکه چه حجمی از XC4005XL جهت ساخت یک جمع کننده تک بیتی مورد استفاده قرار می گیرد ،خواهد داد.

خلاصه استفاده از فضای DEVICE :

IO	5/112	4%used
	5/61	8% bonded
LOGIC	1/961	0% used
IBO	5/112	4% used
CLB	1/196	0% used

جمع کننده تک بیتی دارای 3 ورودی و 2 خروجی می باشد ود مجموع 5 بلوک از 112 بلوک I/O (IOB) قابل دسترسی در XC4005XI FPGA را استفاده می کند.

البته فقط 61 بلوک از این IOBها واقعا به پینهای فیزیکی روی بسته 84 پینیPLCC متصل هستند.بنا براین تقریبا 8% از I/O های قابل دسترسی ،در جمع کننده یک بیتی استفاده می شوند.

هر دو مدار SUM وcarry-output در طر حها ،یک خروجی وسه ورودی دارند . هر مدار باید 4lut پینی قرار گیرد بنا بر این انتظار داریم که از دوlut برای مدار جمع کننده یک بیتی خود استفاده کنیم.ودر هرclb هم دو 4lut وجود دارد بنابر این فقط یک clbبرای طراحی یک جمع کننده بیتی مورد نیاز است.

4-6-7 گزارش PAD:

پرسش دوم در این گزارش پااسخ داده می شود این گزارش در مورد محل تر مینالهای I/O طرح ما با در نظر گرفتن پینهای بسته توضیح می دهد.در این جدول قسمتی از اطلاعات طرح ما قرار دارد

Comp name	Pin number
CARRY-INPUT	P44
CARRY-OUTPUT	P25
INPUTO	P45
INPUT	P46
SUM	P25

تخصیص پینها با آنچه در فایل UCF قرار دادیم باید یکی باشد.

در زیر گزارش لیستی از تخصیص پینها وجود دارد که می تواند در یک Phisical (PCF) file constraint استفاده شود.

```
COMP ` CARRY-INPU`T LOCATE = SITE `P44`;
COMP `CARRY-OUTPUT`LOCATE = SITE `P26`
COMP `INPUT1`LOCATE = SITE `P45`;
COMP `INPUT `LOCATE = SITE `P46 `;
COMP `SUM ` LOCATE = SITE ` P25 `;
```

pcfها از syntax های متفاوتی نسبت به ucfها استفاده می کنند.

Foundation Implemention را RUN و سپس از اسامی پینهای یافت شده در گزارش جهت ایجاد یک FCU استفاده کنید.به احتمال100% اختصاص پینها که توسط Compiler صورت می گیرد با آنچه شما می خواهید یکی نخواهد بود.ولی شما می توانید فا یل راآن طور که می خواهید ویرایش کنید.

7-6-5گزارش تا خیر آسنکرون:

تاخیر توزیع برای هر سیگنا ل مسیر یابی شده در این گزارش لیست شده اند.

7-6-6 گزارش تاخیر ناشی از Layout:

این گزارش هر مسیر جایگذاری و مسیر یابی را که محدودیت زمانی مورد نظر را از بین می برد گزارش می دهد.

مثلا چناچه مدار شما باید با فرکانس mhz50 کار کند (تمام عملیات منطقی در 20 ns کامل شوند) ومسیری یافت شود که دارای تاخیر بزرگتر از 20ns باشد در این گزارش ذکر می شود.

7-6-6گزارش تولید فایل بیت:

این گزارش تمام موارد وانتخاب ها ی موثر درهنگام تولید فایل بیت با ینری را یادداشت می کند .هر خطایی که در تولید با ینری اتفاق می افتد در این گزارش لیست می شود.

حال می توانیم مقادیر سه ورودی را به جمع کننده اعما ل کنیم.

انطباق ترمینا لهای ورودی مدار جمع کننده وپینهای XC4005XL در جدول زیر نما یش داده شده است.

ADDER TERMINAL	XC4005CL PIN	
CARRY-INPUT	44	
INPUTO	45	
INPUT1	46	
NOT Used	47	
NOT Used	48	
NOT Used	49	
NOT Used	32	
NOT Used	34	

همچنین فایل add1-40.ucf ،خروجی های Sum وcarry-out را به پینهای 25و26 از XC4005XL FPGAاختصاص می دهد.

پین 25 به S0 از7-SEGMENT متصل می شود در حالیکه پین 26 وs1 راDevice می کند.

چناچه همه چیز به درستی انجام شده باشد ،تست طرح Down load شده شما باید با نتایج شبیه سازی شده وجدول حقیقت منطبق باشد.

7-7 نکاتی در مورد برد آمو زشی XS40 :

مهم ترین عضو برد آزما یشی گفته شده یک XC95108 CPLD یاXC4005 FPGA می باشد FPLDبا فایلهای باینری که توسط نرم افزار Xilinx ایجاد می گردد بار می شود. این فایلهای باینری از طریق پورت کامپوتر در بوردdown load می شود.این پورت موازی همان طور که گفته شد برای اعما ل سیگنالهای تست نیز کاربرد دارد.قسمت7-segment وصل شده بهFPLD یک تصویر از چگونگی کارکرد مدار ،آماده می کند.،یک RAاستاتیک KB32نیز بهFPLDبرای آماده کردن دیتای ذخیره شده خارجی ،متصل می شود.

اجزای دیگری نیز برای آماده کردن کلاکها وقدرت تنظیم شده برای باقی قسمتها وجود دارند.

پیشنهادات:

با توجه به اینکه دنیا IC های دیجیتال روز به روز گسترده تر،پیشرفته تر وکار بردی تر می شود واین علم در دنیای امروزی جایگاهی بسیار مهم پیدا کرده است و در بعد صنعتی نیز یکی از مهم ترین صنایع جهان به شمار می رود به نظر می رسد که در کشور ما آن جایگاه لازم را پیدا نکرده ونمود آنرا می توان در دانشگاه هایمان نظاره گر باشیم .به هر حال به نظر من در حال حاضرمی توان با گنجاندن چند واحد درسی مانند آزمایشگاه میکروپروسسور و... وآشنا شدن دانشجویان با کار عملی در محیط میکرو چیپها این وضعیت را بهبود بخشید.

در رابطه با زمان پروژه نیز به نظر من می توان گفت دانشجویان ترم پایین در مضیقه هستند چرا که دانشجویانی که در ترم بهاراین واحدرا می گذرانند می توانند آنرادر پایان تابستان تحویل دهند و همچنین آنهایی که در ترم تابستان پروژه را تحویل می گیرند از دانشجویان ترم پایین بیشتر زمان دارند.

نتیجه گیری:

این پروژه فرصت خوبی بود تا بتوانم با نوعی از ICهای دیجیتالی آشنا شوم که در دنیای دیجیتال بسیار کاربرد داشته و یکی از گزینه های مناسب برای طرحهای دیجیتالی بسییار پیچیده می باشند.همچنین به اندکی از واقعیتهای موجود در ایران در رابطه با منابع علمی،تحقیقاتی،آشنا شدم واین امر برایم روشن شد که برای دستیابی به منابع علمی اطمینان چندانی به منابع فارسی نیست ویکی از امور مهم برای هر فعالیتی در این زمینه ها دانستن زبان انگلیسی می باشد.

منابع:

www.Xilinx.com

www.altera.com

www.eca.ir

WWW.FPGA4fun.com