

الحمد لله
الرحمن الرحيم



گزارش کار آزمایشگاه معماری

استاد ارجمند:

سرکار خانم مهندس اسدی

تهیه کنندگان:

آرزو گردانی و ساراسکوت

و محبوبه صالحی

اردیبهشت ۱۳۹۶

مدارهای منطقی با دیجیتال:

از ساده ترین گیت های منطقی مثل And , or,xor ,.... ساخته میشوند تا به مدارهای منطقی مجتمع یا همان IC ها برسد که متشکل از چندین گیت های ساده میباشد. با پیچیده تر شدن مدارهای منطقی مجتمع یا همان IC ها برسد که متشکل از چندین بیت های ساده می باشند. با پیچیده تر شدن مدارهای منطقی دیجیتال قابل برنامه ریزی به نام PLA ارائه شده اند و بعد از آن مدارهای قابل برنامه ریزی PLA ,PLD نیز به بازار عرضه شدند و بالاخره مدارهای پیچیده تر قابل برنامه ریزی FPGA,CPLD ساخته شده اند.

FPGA, CPLD براساس سلول های منطقی قابل برنامه ریزی طراحی شده اند که ارتباط بین این سلول ها نیز قابل برنامه ریزی میباشد. چون FPGA,CPLD از نظر برنامه ریزی و کاربرد مشابه به هم هستند. در برخی نوشته ها CPLD را نوعی FPGA می نامند.

اولین FPGA به وسیله ی شرکت Xilinx ارائه شده از آن زمان به بعد FPGA های متفاوتی توسط شرکت های مختلف ارائه میشوند.

نکته مهم:

این است که FPGA ها را نباید به Cpu ها و میکروکنترل هایی مثل ARM , AVR اشتباه گرفت زیرا آنها خودشان ساختار سخت افزار از پیش آماده دارند و فقط کافی است با زبان مورد نظرشان به آنها دستوراتی داد تا اجرا کنند ولی FPGA ها در ابتدا داخلشان هیچ چیز نیست و این شماست که با زبان VHDL سخت افزار مورد نظرتان را پیاده سازی کنید. مثلا میتوانید همان میکروکنترلر را داخل FPGA پیاده سازی کنید. در واقع ساختار سخت افزار آنها کاملا متغیر است.

نرم افزار ISE:

قسمت بزرگی از کار کردن با FPGA مربوط به زبان VHDL میباشد. با توجه به این که این زبان وابستگی به هیچ

نرم افزاری ندارد میتوان برنامه VHDL نوشته شده برای یک FPGA شرکت Altera را بر روی یک FPGA شرکت Xilinx ریخت با این تفاوت که باید مجددا توسط کامپایلر ساخته شود.

برنامه نویسی و طراحی FPGA به طور کلی به دو صورت انجام می پذیرد:

➤ الف) با استفاده از زبان های توصیف سخت افزار می باشد.

نظیر: VHDL, Verilog, HDL

➤ ب) با استفاده از طراحی مدار

VHDL: یکی از زبانهای توصیف سخت افزار می باشد. از گیت های ساده AND, OR گرفته تا CPU یعنی هر مدل سخت افزار دیجیتال VHDL توصیف کرد و رفتار آن را روی کاغذ آورد بدون نیاز به این که بخواهیم سخت افزار مورد نظر را به صورت کیفی توصیف کنیم. این زبان برای برنامه ریزی در Program, Ic، های نظیر cpld, pld, pla ها و البته FPGA استفاده میشود.

نرم افزار ISE سه پنجره اصلی دارد:

۱. Source:

می شود به snaption سورس ها و فایل هایی که ساخته می شود دسترسی پیدا کرد.

۲. Processes:

وقتی فایلی از قسمت سورس انتخاب می کنیم داخل این بخش عملیاتی که باید به روی فایل انجام شود نشان داده می شود.

۳. Transcript:

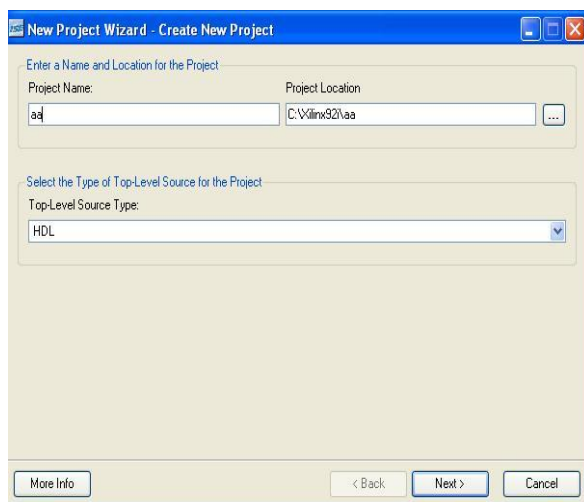
وقتی عملیات را در Processes روی فایل انجام می دهد در این قسمت گزارش روند عملیات داده می شود.

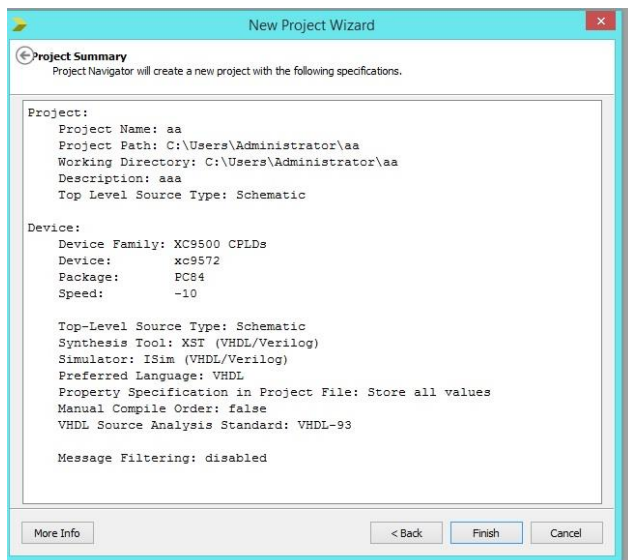
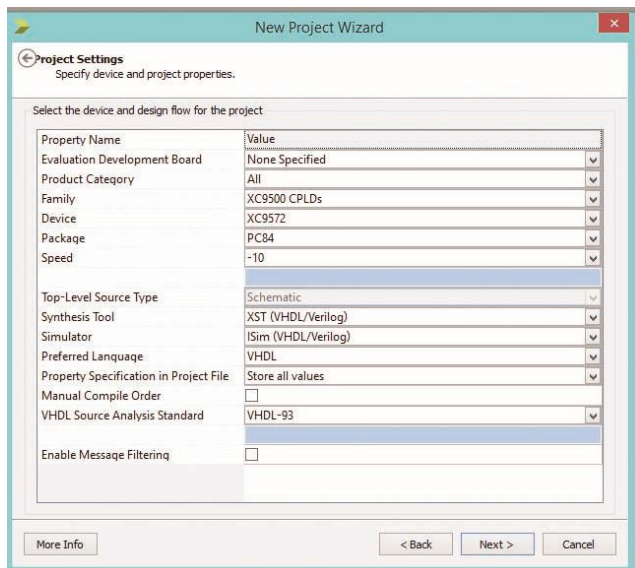
ابتدای زمانی که نرم افزار را باز کردیم یک پروژه جدید ایجاد می کنیم. مراحل به صورت زیر است:

از منوی file گزینه ی new project و سپس نام برای پروژه انتخاب کرده و سپس قسمت Top Level Source که حالت توصیف سخت افزار برای برنامه نویسی به زبان VHDL انتخاب می کنیم و سپس گزینه ی Next را انتخاب میکنیم. که این قسمت گزینه ی Schematic را انتخاب می کنیم و سپس گزینه ی Next را انتخاب میکنیم.

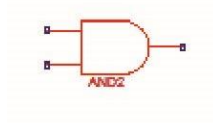
در صفحه بعد باید تنظیمات به صورت زیر باشد:

Product Category	→	All
Family	→	Mc۹۵۰۰ CpldS
Device	→	Xc۹۵۷۲
Package	→	Pc۸۴
Speed	→	-۱۰
Top-Level Source Type	→	schematic
Synthesis Tool	→	XST (VHDL/Verilog)
Simulator	→	LSE Simulator (VHDL/Verilog)
Preferred Language	→	VHDL

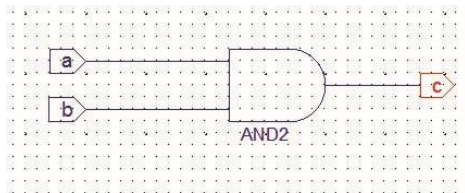




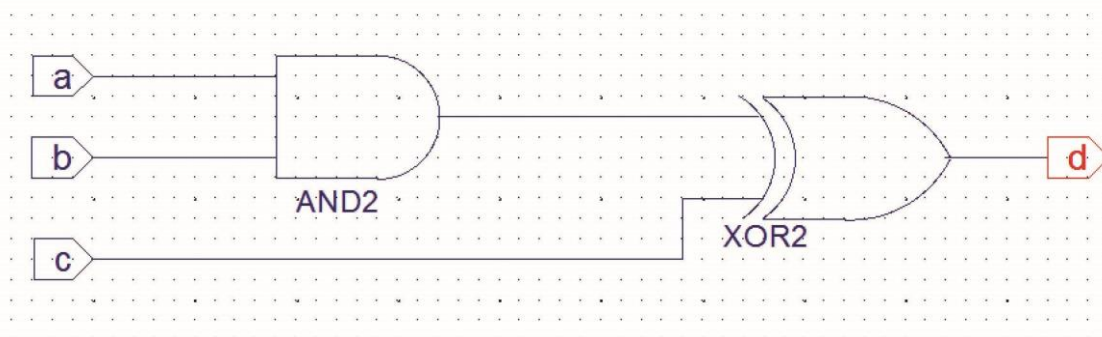
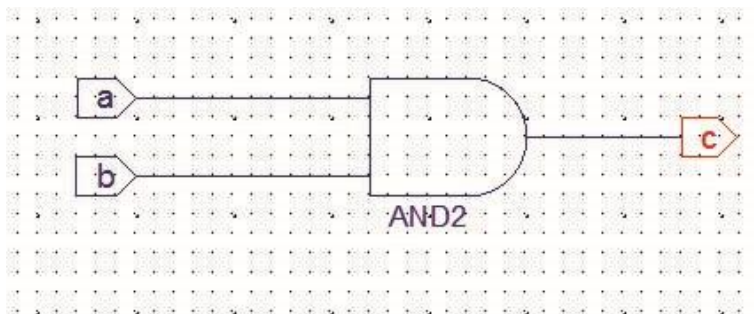
سپس Next و در آخرین مرحله finish را انتخاب می کنیم. سپس در صفحه ای که باز می شود در پنجره source قسمت categories گزینه All symbols را انتخاب می کنیم و سپس از قسمت symbols گیت ۲ And را انتخاب می کنیم که به شکل زیر می باشد.



ابزار Add wire را به ورودی و خروجی ۲ And وصل می کنیم ابزار Add Io / marker wire را باید به ورودی و خروجی وصل کنیم به آن یک اسم مثل A می دهیم.



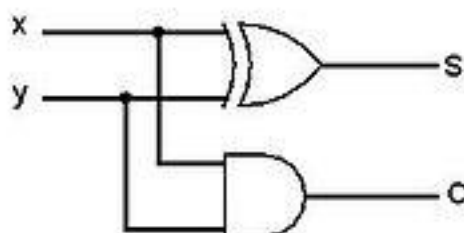
زمانی که مدار را طراحی کردیم باید آن را کامپایل کنیم که برای کامپایل کردن ابتدا فایل را save می کنیم و سپس در پنجره source مورد نظر را انتخاب و در قسمت process را انتخاب و روی گزینه Implement Design دابل کلیک می کنیم تا شروع به کامپایل شدن کند. زمانی پیغام مناسب نمایش داده شده پیغام مبنی بر کامل شدن کامپایل در پنجره source گزینه new source ایجاد کرده و در پنجره ای که باز می شود گزینه Test bench waveform را انتخاب و سپس گزینه ی finish را انتخاب کرد. پنجره ای باز می شود روی قسمت آبی رنگ ها کلیک کرده و سپس در پنجره ی source قسمت source to Behavioral Simulation را انتخاب کرده و سپس فایل source مورد نظر را انتخاب کرده و از قسمت Process گزینه ی Xilinx Ise simulator و سپس گزینه ی Behavioral mode را انتخاب می کنیم که بینیم مدار درست کار میکند یا خیر؟



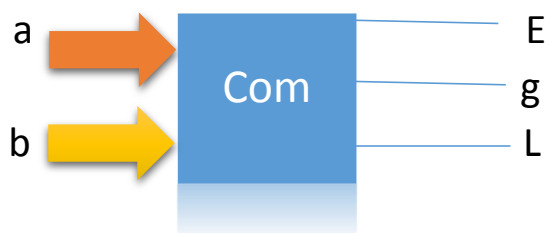
مدارنیم جمع کننده:

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1






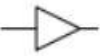


جدول درستی



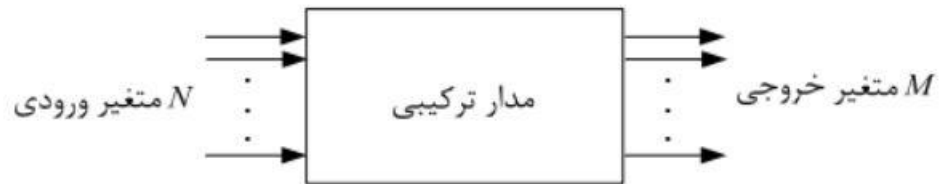
دیاگرام مدار منطقی نیم جمع کننده



انواع گیت های منطقی :

نام گیت	نماد	تابع جبری	جدول درستی															
AND		$f = xy$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	f	0	0	0	0	1	0	1	0	0	1	1	1
x	y	f																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$f = x + y$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	f	0	0	0	0	1	1	1	0	1	1	1	1
x	y	f																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NAND		$f = \overline{xy}$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	f	0	0	1	0	1	1	1	0	1	1	1	0
x	y	f																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$f = \overline{x + y}$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	f	0	0	1	0	1	0	1	0	0	1	1	0
x	y	f																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
NOT		$f = \overline{x}$	<table><tr><th>x</th><th>f</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	f	0	1	1	0									
x	f																	
0	1																	
1	0																	
Buffer		$f = x$	<table><tr><th>x</th><th>f</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	f	0	0	1	1									
x	f																	
0	0																	
1	1																	
XOR		$f = x \oplus y$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	f	0	0	0	0	1	1	1	0	1	1	1	0
x	y	f																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
XNOR		$f = \overline{x \oplus y}$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	f	0	0	1	0	1	0	1	0	0	1	1	1
x	y	f																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

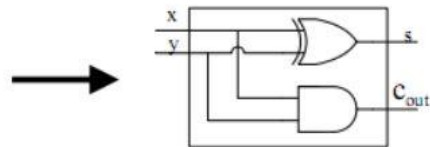
مدارهای منطقی ترکیبی:



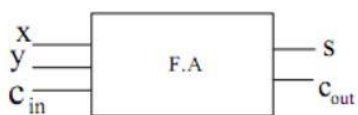
۱. نیم جمع کننده:

	x	y	s	c _{out}
	0	0	0	0
	0	1	0	1
	1	0	0	1
	1	1	1	0

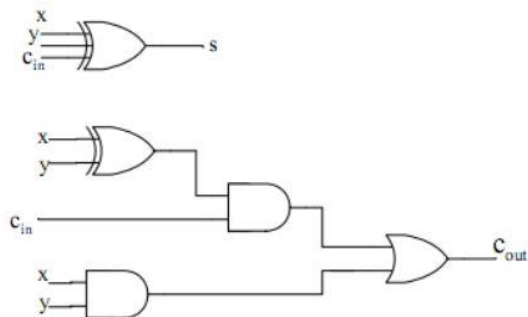
$$\begin{cases} s = x \oplus y \\ c_{out} = xy \end{cases}$$



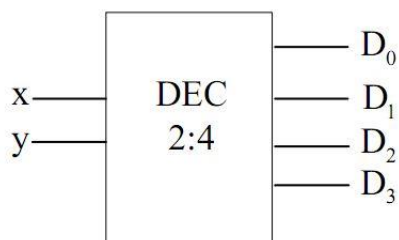
۲. تمام جمع کننده:



$$\begin{cases} s = x \oplus y \oplus c_{in} \\ c_{out} = xy + c_{in}(x \oplus y) \end{cases}$$

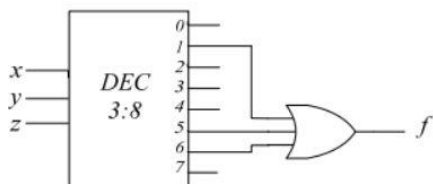


۳. دیکدر:

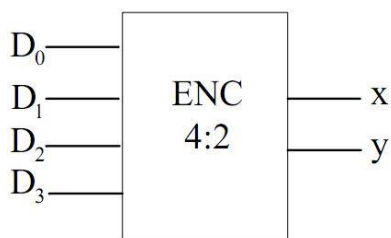


y	x	D ₃	D ₂	D ₁	D ₀
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

پیاده سازی تابع $f(z, y, x) = \sum m(1, 5, 6)$ با استفاده از دیکدر:

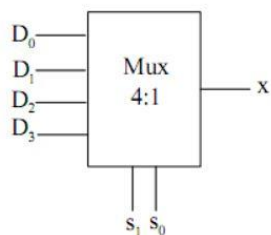


۴. انکدر:



D ₃	D ₂	D ₁	D ₀	y	x
0	0	0	1	0	0
0	0	1	0	0	1
1	1	0	0	1	0
1	0	0	0	1	1

۵. مالتی پلکسر:



s ₁	s ₀	x
0	0	D ₀
0	1	D ₁
1	0	D ₂
1	1	D ₃

پیاده سازی تابع $f(z, y, x) = \sum m(1, 2, 6, 7)$ با استفاده از مالتی پلکسر ۴ به ۱ (برای پیاده سازی یک تابع n متغیره به یک ماکس 2^{n-1} بیتی نیاز است)

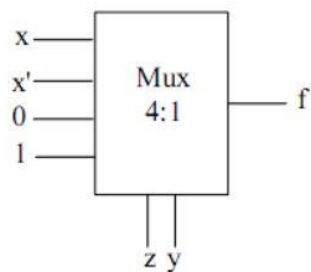
z	y	x	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$f=x$

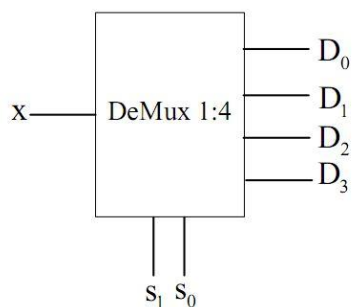
$f=x'$

$f=0$

$f=1$

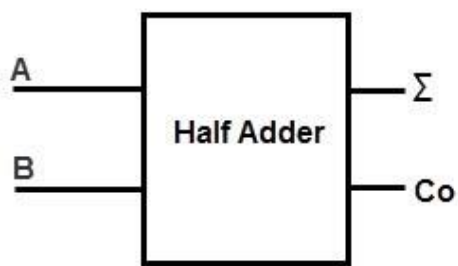


۶. دی مالتی پلکسر:



s ₁	s ₀	D ₃	D ₂	D ₁	D ₀
0	0	0	0	0	x
0	1	0	0	x	0
1	0	0	x	0	0
1	1	x	0	0	0

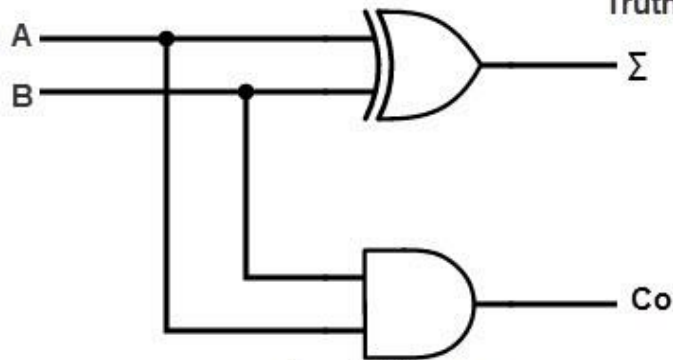
:HAFF adder



Block Diagram

A	B	Sum (Σ)	Carry Out (Co)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

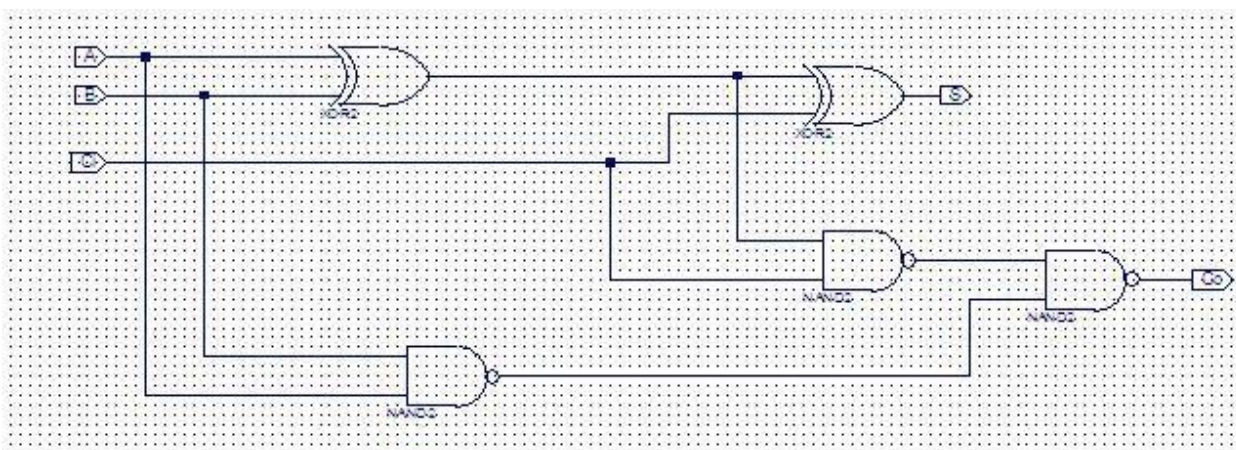
Truth Table



Logic Diagram

:Full adder

Input			Output	
A	B	Cin	Sum	Carry
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



$S \leq (a \text{ xor } b) \text{ xor } ci;$

$Co \leq (a \text{ nand } b) \text{ nand } (ci \text{ nand } (a \text{ xor } b));$

معرفی پورت های ورودی و خروجی به وسیله جمله `port()` انجام می شود. داخل پرانتز ابتدا نام ورودی ها و خروجی ها نوشته می شود. اگر برنامه چندین `port` ورودی و خروجی داشته باشد می توان همه ورودی ها و یا همه خروجی ها را کنار هم با علامت ، بینشان وارد کنیم . بعد از نام پورت علامت : گذاشته می شود و بعد نوع `port` که ورودی یا خروجی است مشخص شود.

سپس با وجود یک `space` (فاصله) جنس پورت مشخص می شود که تا این جا به صورت پیش فرض `std-logic` انتخاب می شود.

در قسمت پورت برای معرفی ورودی ها و خروجی ها باید بینشان علامت ; قرار داده شود.

قسمت اصلی برنامه قسمت سوم آن می باشد که بدنه اصلی برنامه است.

قسمت سوم برنامه `architecture` می باشد. که معماری اصلی برنامه در این قسمت انجام می شود.

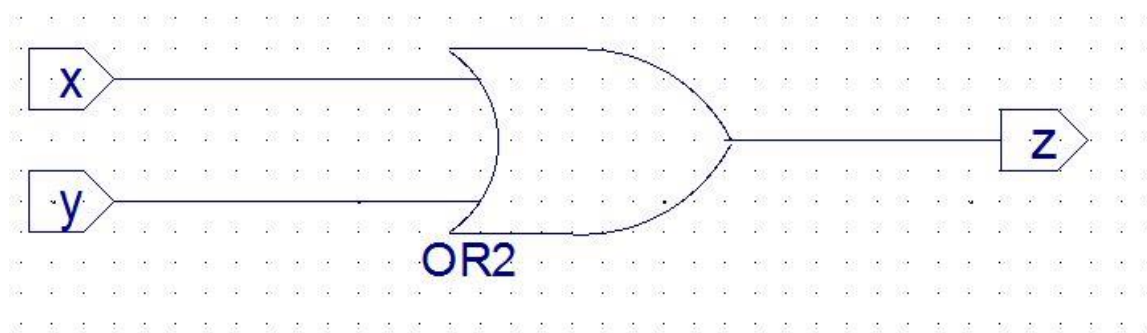
قسمت سوم با جمله `architecture Behavioral of aa is` شروع می شود و با جمله `end Behavioral` تمام می شود.

برای نوشتن دستورات برنامه ابتدا جمله begin در قسمت architecture وارد میشود وبعد دستور برنامه نوشته می شود.

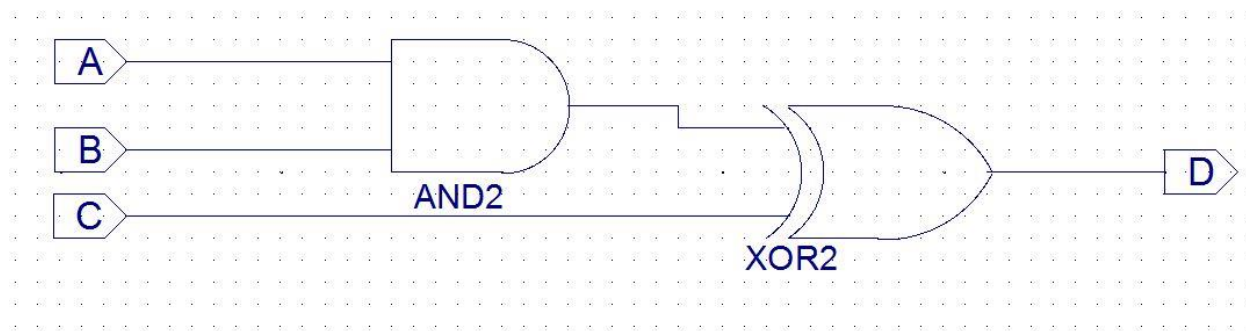
نکته: تمامی گیت های منطقی برای زبان VHDL تعریف شده است و زمانی که تایپ شوند با رنگ آبی مشخص می شوند.

نکته: برای مقداردهی به خروجی از علامت کوچکتر مساوی استفاده می شود.

نکته: برای نوشتن اعداد منطقی تک بیتی داخل برنامه آن ها را داخل ' ' قرار می دهیم و برای نوشتن اعداد منطقی چندبیتی داخل برنامه " " قرار می دهیم.



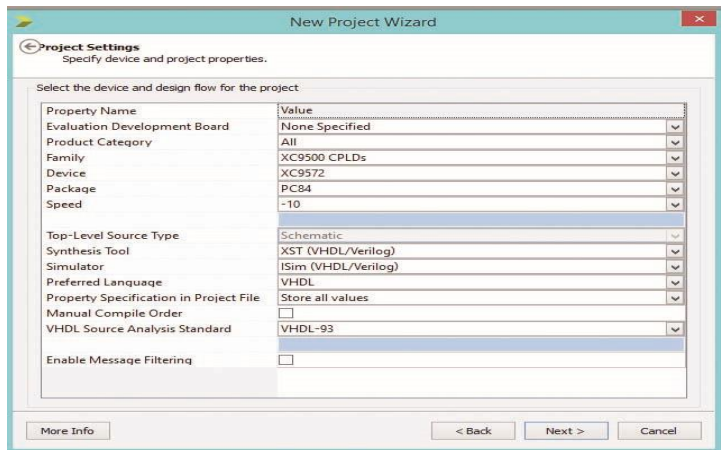
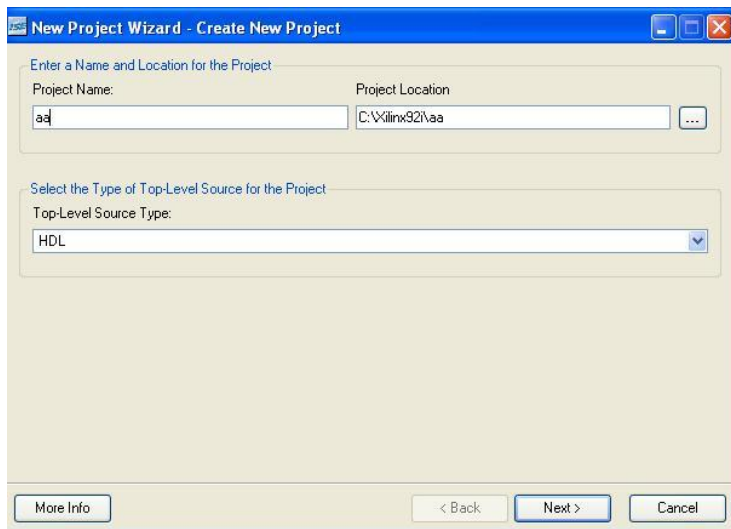
$Z \leq x \text{ or } y;$



$d \leq (a \text{ and } b) \text{ xor } c$

برنامه نویسی به زبان VHDL :

از فایل سمت چپ Projeh را انتخاب می کنیم بعد new project اسم جدید می دهیم و گزینه Top level source و گزینه HDL انتخاب می کنیم و سپس در پنجره باز شده (device properties) تنظیمات اول برنامه را انتخاب می کنیم.



بعد تنظیمات گزینه next و finish را می زنیم.

روی source کلیک راست کرده و گزینه new source و یک اسم به فایل می دهیم VHDL MODULE را انتخاب میکنیم.

نکته : توجه داشته باشید که اسم ماژولها نباید گیت یا اسم های خاص باشد.

سپس گزینه ی NEXT را می زنیم. سپس یک پنجره جدید باز می شود که در این جا باید ورودی ها و خروجی ها را X,Y و خروجی را Z انتخاب می کنیم.

در قسمت **Direction** اسم پورت های ورودی و خروجی وارد می شود. سه پورت Z,X,Y را وارد می کنیم.

X in (ورودی)

Y in (ورودی)

Z out (خروجی)

در قسمت port name اسم پورت های ورودی و خروجی را وارد کرده و در قسمت

Direction باید اسم پورت های ورودی و خروجی وارد شود.

اگر ورودی و خروجی ها تک بیتی نبود Bus را انتخاب می کنیم و سپس گزینه next را انتخاب می کنیم.
Summary قسمت زیادی از برنامه را خودش می نویسد.

Library کتابخانه VHDL است. بخش دوم **Entity a** اطلاعات مدار وارد می شود با فایل های اطلاعات پورت وارد می شود و در این بخش شیوه وارد کردن اطلاعات پورتها به این صورت است . پورت ();

داخل پرانتز ابتدا اسم پورت نوشته می شود و بعد علامت دو نقطه (:). گذاشته می شود بعد از آن اسم پورت که ورودی و خروجی مشخص می شود و سپس space که نوع پورت است را انتخاب می کنیم. نوع پورت SW کاملترین نوع پورت است که انتخاب می کنیم.

داخل VHDL باید ; گذاشته شود. در زبان VHDL یک سری کلمات کلیدی وجود دارد که هر کدام معنای خاصی دارد برای مثال دو خط تیره یا - برای ارائه توضیحات در برنامه استفاده می شود و برای برنامه ندارد پس هر کجای بخواهیم بنویسیم دو خط تیره می گذاریم و تا پایان خط می توانیم گزارش را بنویسیم اگر بیشتر از یک خط بود علامت خط تیره را قرار می دهیم.

archtve behavioral offaais: بعد از اجرای begin شروع می شود و با دستور end behavioral شروع می شود بعد begin وارد میشود بعد دستور نمی شود و behavioral وارد می شود.

در زبان VHDL برای مقداردهی به پارامترها کوچکتر بزرگتر نوشته می شود. انتهای دستور ; هست . Or آبی می شود و دستور را در archtve وارد می کنیم.

Z<x or y

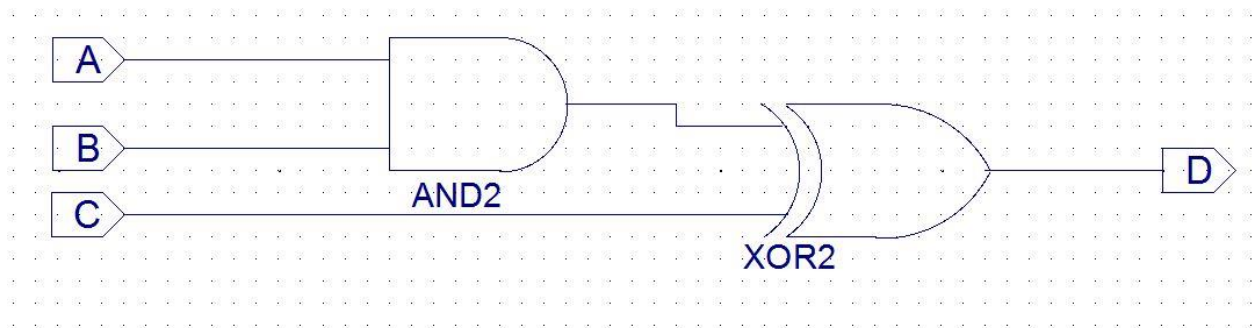
وقتی کامل شد برنامه را save را انتخاب می کنیم . هر برنامه ای که شما بنویسید ابتدا باید کامپایل بشود بعد باید به خروجی برویم. در قسمت source پسوند VHDL را انتخاب کرده سپس روی آن کلیک کرده بعد قسمت Increment design ، procees را دابل کلیک می کنیم بعد شروع به کامپایل می کند . پنجره جدید باز می کند که پیغام Succesfully می دهد.

دستور ورود مدار منطقی:

در برنامه نویسی به زبان VHDL برای مقادیر تک بیتی از علامت سینگل کوتشین استفاده می شود و برای دوبیتی " " استفاده می شود.

در قسمت archtve

d<=a(a and b) or c



لیست حساسیت process :

یک ساختار طراحی VHDL است که برای شکل دهی به الگوریتم می باشد. اگر process در برنامه اضافه شود تنها زمانی که لیست حساسیت تغییر کند خروجی ها محاسبه می گردند و در توان مصرفی CPU و زمان مصرفی محاسبات صرفه جویی می شود.

لیست Process میتواند شامل یک نام اختیاری باشد. برای مثال این نام : مجزا می شود.

Ha: نام لیست process اختیاری است که مخفف (Haff adder) است.

HA: process(x, y) is

لیست حساسیت که در جلوی کلمه process آورده می شود نشان دهنده این است که process براساس این سیگنال ها اجرا می گردد یا اصطلاحا حساس می باشد که در اینجا سیگنالهای x,y لیست حساسیت process هستند.

Process بعد از begin archtive آورده می شود و بعد از آن دومرته کلمه begin آورده شود و بعد از آن دومرته کلمه begin آورده می شود. archtive میتواند شامل چندین process باشد که موازی باهم اجرا می شود.

Process(x,y)is

Process با این جمله آغاز می شود و در انتهای آن لزومی به علامت ; نمی باشد.

لیست process با جمله end process; به پایان می رسد.

نکته: در process میتوان انواع دستوراتی که در VHDL وجود دارد استفاده کرد. به کمک این دستورات می توانیم نحوه ی ارتباط بین ورودی ها و خروجی ها را برقرار کنیم.

Process(x,y) is

Begin

C<=x and y;

S<=x oxr y;

End process;

متغیر واسط:

نکته: از متغیرها در process وزیر برنامه ها می توان استفاده کرد و در محدوده process یا زیر برنامه باید معرفی شود.

نکته: از یک متغیر در دو process نمی توان استفاده شود.

نکته: متغیرها برخلاف سیگنال ها دارای اجزای حافظه ای نیستند بنابراین تاخیر ندارند و معمولاً برای اهداف محاسباتی استفاده می شود، متغیرها فقط مقداری را در یک زمان نگه می دارند و نمی توان شکل موج آنها را مشاهده کرد.

نکته: در طول برنامه برای مقداردهی به متغیرها از نماد =: استفاده می شود.

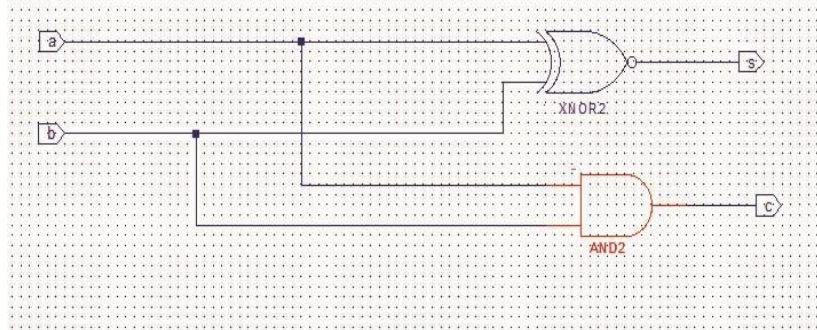
ممکن است در یک process از چندین متغیر واسط استفاده شود که همه آنها را در کنار هم معرفی می کنیم و با علامت ; آنها را مجزا می کنیم برای مثال:

Variable a,b : STD_LOGIC;

a : اسم متغیر واسط

STD_LOGIC: جنس متغیر واسط که اکثراً موارد STD_LOGIC انتخاب می شود.

مدار HAFF ADDER:



برنامه Haff adder:

Process(x,y) is

Begin

C<=x and y;

S<=x oxr y;

End process;

.....

$S \leq (a \text{ oxr } b) \text{ xor } c;$

$Co \leq (a \text{ nand } b) \text{ nand } (ci \text{ nand } (a \text{ xor } b));$

Msb=۳

Lsb=0

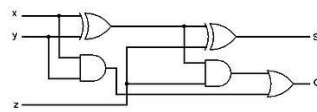
دیکدر (Decoder):

$C \leq a \text{ or } b ;$

مدار full adder :

z	x	y	S	C
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

جدول درستی



دیگرام مدار منطقی تمام جمع کننده

Process (a,b, ci) is

Variable w: std_logic;

Variable d: std_logic;

Variable x: std_logic;

Begin

W:= a xor b;

D:= (a nand b);

x:=ci nand w;

s<= ci xor w;

End process;

Schematic مالتی پلکسر:

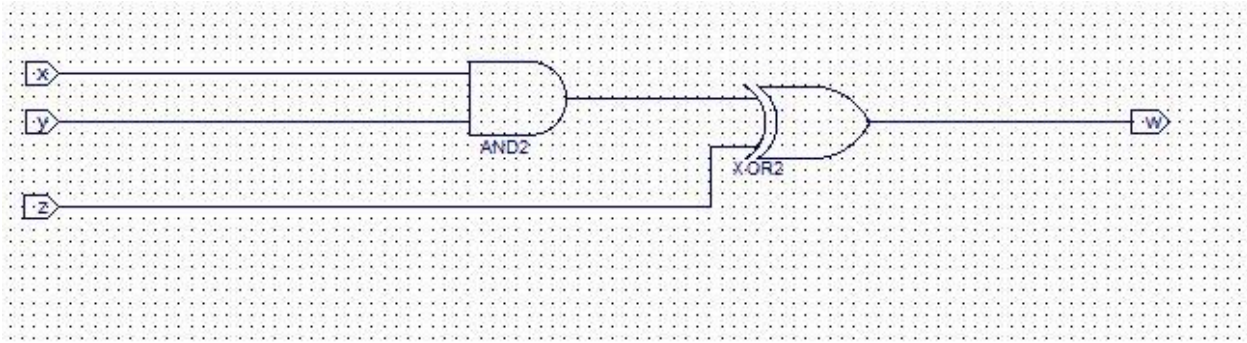
برنامه نویسی برای mux مالتی پلکسر:

```
Process (i0, i1, s)
Variable a: std_logic;
Variable b: std_logic;
Begin
a:=i0 and (not s);
b:= s and b;
0<= a and b;
End process;
```

: Mux ۴*۱

```
IF- the-else
if(condition)then
do something;
else
do something different
end if;
```

```
Process (i0, i1, s)
Begin
If(s='0') then
0<=io;
Else if(s='1') then
o<=i1;
End if;
End process;
```



Architecture Behavioral of A is

Process(x,y,z) is

Variable a: STD_LOGIC

Begin

Process(x,y,z) is

Variable a: STD_LOGIC

A: =x and y;

w <= a or z;

End process;

End behavioral;

process (x,y,z)is

variable a:std_logic;

variable b:std_logic;

begin

a:=x and y;

b: a xor z;

w<= a nand b;

end if;

end process;

شکل مدار بالا

ساختار مدار ترتیبی و ترکیبی:

مدارهای منطقی ترکیبی	مدارهای منطقی ترتیبی
ساختارها و دستورات ترتیبی در زبان VHDL	ساختارها و دستورات ترکیبی در زبان VHDL
IF Case Loop Variable Process Function Procedure	When/Else With/Select For/Generate Block

دستورات ترکیبی

۱- دستور When- else

مثال	قالب دستور
<pre> outp <= "000" WHEN (inp='0' OR reset='1') ELSE "001" WHEN ctl='1' ELSE "010"; </pre>	else شرط ۱ مقدار ۱ <= نام سیگنال else شرط ۲ مقدار ۲ مقدار آخر;

۲- دستور Which - select

مثال	قالب دستور
<pre> With a select b <= d0 when "000" , d1 when "001" , d2 when "002" , Unaffected when others; </pre>	Select سیگنال شرط With مقدار سیگنال شرط when مقدار ۱ <= نام سیگنال , مقدار سیگنال شرط when مقدار ۲ , when others; مقدار آخر;

۳- دستور for/generate

مثال	قالب دستور
<pre> OK: for i in 0 to 7 generate output(i) <= '1' when (a(i) and b(i))='1' else '0'; end generate; </pre>	تکریب (اجباری) generate انتهای حلقه to ابتدای حلقه in نام سیگنال begin عبارت; عبارت; end generate;
<pre> Check: If (Reset = '1') generate begin Output <= '0'; End generate; </pre>	تکریب (اجباری) generate شرط if begin عبارت; عبارت; end generate;

۴- ساختار Block:

<p>Block: برچسب (اجباری) معرفی سیگنال های مورد نظر Begin عبارت; عبارت; End Block برچسب</p>	<pre>b1: BLOCK SIGNAL a: STD_LOGIC; BEGIN a <= input_sig WHEN ena='1' ELSE 'Z'; END BLOCK b1;</pre>
<p>Block (عبارت شرط): برچسب (اجباری) معرفی سیگنال های مورد نظر Begin عبارت (مشروط یا غیر مشروط) عبارت (مشروط یا غیر مشروط) End Block برچسب</p>	<pre>b1: BLOCK (clk='1') BEGIN q <= GUARDED d; END BLOCK b1;</pre>

دستورات ترتیبی عبارتند از:

- ۱) If
- ۲) Wait
- ۳) Case
- ۴) Loop
- ۵) Variable

این دستورات تنها در بلوک های زیر می توانند نوشته شوند:

Process

Function

Procedure

بلوک های ترتیبی

(۱) Process :

مثال	قالب بلوک Process
<pre> Process(clk,rst) Begin If (rst = '1') then q <= '0'; elsif (clk'event and clk = '1') then q <= d; end if; End process; </pre>	<p>(برچسب اختیاری) :</p> <p>لیست سیگنال هایی که پروسس با تغییر آن ها اجرا می شود</p> <p>Process (برچسب اختیاری)</p> <p>Begin</p> <p>دستورات ترتیبی</p> <p>END Process (برچسب اختیاری) ;</p>

برای محدود کردن اجرای برنامه به یکی از لبه های کلاک باید از دستورات زیر در برنامه استفاده کنیم:

- If (clk'event and clk = '1') then مدار به لبه بالا رونده کلاک حساس است
- If (rising_edge(clk)) then مدار به لبه بالا رونده کلاک حساس است
- If (clk'event and clk = '0') then مدار به لبه پایین رونده کلاک حساس است
- If (falling_edge(clk)) then مدار به لبه پایین رونده کلاک حساس است

دستورات ترتیبی :

(۱) Variable , single :

در VHDL برای تعریف سیگنال های میانی از دروش زیر استفاده می شود.

:Single

:Variable

مثال	قالب دستور
<pre> signal control: bit := '0' signal count: integer range 0 to 100; signal y: std_logic_vector(7 downto 0); </pre>	<p>؛ مقدار اولیه =: نوع سیگنال : نام سیگنال</p> <p>signal</p>
<pre> variable control: bit := '0' variable count: integer range 0 to 100; variable y: std_logic_vector(7 downto 0); </pre>	<p>؛ مقدار اولیه =: نوع متغیر : نام متغیر</p> <p>variable</p>

Signal	Variable	
Package و Architecture	بلوک های ترتیبی	محل استفاده
مقدار Signal سراسری است و در کل برنامه می توان مقدار آن را خواند یا آن را مقدار دهی کرد.	مقدار variable محلی است و تنها داخل بلوک ترتیبی می تواند مقدار دهی شود و یا مقدار آن خوانده شود.	محلی/سراسری
وقتی داخل بلوک های ترتیبی مقداردهی شود مقدار آن پس از پایان اجرای بلوک آپدیت می شود.	مقدار آن بلافاصله آپدیت می شود.	زمان مقداردهی
<=	:=	علامت تخصیص مقدار

if (۲)

مثال	قالب دستور
<pre> if (rst='1') then output <= "00000000"; elsif (clk'event and clk='1') then output <= input; end if; </pre>	<p>If شرط then دستورات Elsif شرط then دستورات End if;</p>

Wait (۳)

مثال	قالب دستور
<pre> Process --no sensitivity list Begin wait until (clk'event and clk='1'); if (rst='1') then output <= "00000000"; elsif (clk'event and clk='1') then output <= input; end if; End Process; </pre>	Wait Until شرط;
<pre> Process --no sensitivity list Begin wait on rst, clk; if (rst='1') then q <= '0'; elsif (clk'event and clk='1') then q <= d; end if; End Process; </pre>	سیگنال ۱ و سیگنال ۲ و ... Wait ON
<pre> wait for 10 ns; output <= '1'; </pre>	زمان Wait For

:Case (۴)

مثال	قالب دستور
<pre> case sel is when "00" => x<=a; when "01" => x<=b; when "10" => x<=c; when others => x<=d; end case; </pre>	<p>Case is نام سیگنال</p> <p>When دستورات => مقدار</p> <p>.....</p> <p>When دستورات => مقدار</p> <p>END Case;</p>

:Loop (۵)

مثال	قالب دستور
<pre> for i in 1 to inp'high loop outp(i) <= inp(i-1); end loop; </pre>	<p>رنج IN نام سیگنال حلقه FOR: (برچسب اختیاری)</p> <p>LOOP</p> <p>کدهای ترتیبی</p> <p>END LOOP (برچسب اختیاری);</p>
<pre> while (i<10) loop wait until clk'event and clk='1'; q <= a; end loop; </pre>	<p>LOOP شرط حلقه While: (برچسب اختیاری)</p> <p>کدهای ترتیبی</p> <p>END LOOP (برچسب اختیاری);</p>

در حلقه for از دستورات زیر می توان استفاده کرد:

Exit: برای خارج شدن از حلقه:

Exit(when شرط);

(مثال)

For l in to ۱۵ loop

Next when i=skip; -- jumps to next iteration

(...)

End loop;

:Next

برای پرش به اندیس بعدی حلقه

Next (when شرط);

```
FOR i IN 0 TO 15 LOOP
    NEXT WHEN i=skip;    -- jumps to next iteration
    (...)
END LOOP;
```

پیاده سازی مدارهای ترکیبی با استفاده از دستورات ترتیبی

نحوه پیاده سازی مالتی پلکسر ۴*۱

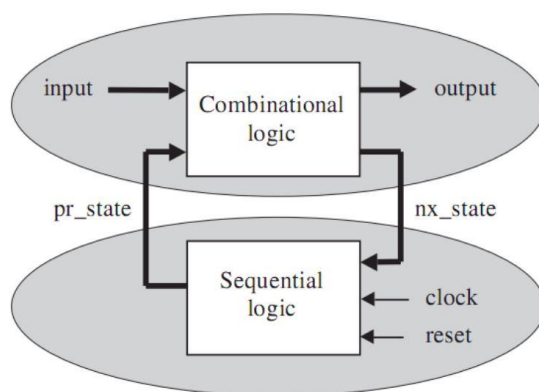
```
architecture archi of mux is
begin
    process (a, b, c, d, s)
    begin
        if      (s = "00") then o <= a;
        elsif  (s = "01") then o <= b;
        elsif  (s = "10") then o <= c;
        else o <= d;
        end if;
    end process;
end archi;
```

ماشین حالت

۱- میلی: در ماشین حالت میلی، وضعیت خروجی به حالت فعلی و ورودی های فعلی بستگی دارد.

۲- مور: در ماشین حالت مور، خروجی تنها به وضعیت فعلی بستگی دارد.

شکل زیر بلوک دیاگرام کلی یک ماشین حالت را نشان می دهد:



برای تعریف یک ماشین حالت ابتدا باید متغیری از نوع شمارشی تعریف کرد.

Type **is** نام **is** (مقادیر مورد نظر)

برای مثال برای تعریف ماشین حالتی که چهار حالت دارد باید متغیری مانند زیر تعریف کرد. نام اختصاص داده شده به حالت ها اختیاری است.

Type **my_state is** (s0,s1, s2, s3);

حالت فعلی و حالت بعدی سیگنال های داخلی هستند که نوع آن ها باید مطابق با نوع شمارشی تعریف شده باشد. مثال:

Signal pr_state, nx_state: my_state;

۱- میلی: در ماشین حالت میلی، وضعیت خروجی به حالت فعلی و ورودی های فعلی بستگی دارد.

۲- مور: در ماشین حالت مور، خروجی تنها به وضعیت فعلی بستگی دارد.

شکل زیر بلوک دیاگرام کلی یک ماشین حالت را نشان می دهد:

قالب کلی کد به قسمت ترتیبی صورت زیر است:

در این قسمت با لبه کلاک سیستم به حالت بعدی می رود. در صورتی که ریست مدار فعال شود سیستم به حالت اولیه برمی گردد.

```
PROCESS (reset, clock)
BEGIN
    IF (reset='1') THEN
        pr_state <= state0;
    ELSIF (clock'EVENT AND clock='1') THEN
        pr_state <= nx_state;
    END IF;
END PROCESS;
```

قالب کلی قسمت ترتیبی :

در این قسمت خروجی ها مقداردهی می شود و حالت بعدی نیز مشخص می شود.

```
PROCESS (input, pr_state)
BEGIN
    CASE pr_state IS
        WHEN state0 =>
            IF (input = ...) THEN
                output <= <value>;
                nx_state <= statel;
            ELSE ...
            END IF;
        WHEN statel =>
            IF (input = ...) THEN
                output <= <value>;
                nx_state <= state2;
            ELSE ...
            END IF;
        WHEN state2 =>
            IF (input = ...) THEN
                output <= <value>;
                nx_state <= state2;
            ELSE ...
            END IF;
        ...
    END CASE;
END PROCESS;
```


قالب کلی طراحی ماشین:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-----
ENTITY <entity_name> IS
    PORT ( input: IN <data_type>;
          reset, clock: IN STD_LOGIC;
          output: OUT <data_type>);
END <entity_name>;
-----
ARCHITECTURE <arch_name> OF <entity_name> IS
    TYPE state IS (state0, statel, state2, state3, ...);
    SIGNAL pr_state, nx_state: state;
BEGIN
    ----- Lower section: -----
    PROCESS (reset, clock)
    BEGIN
        IF (reset='1') THEN
            pr_state <= state0;
        ELSIF (clock'EVENT AND clock='1') THEN
            pr_state <= nx_state;
        END IF;
    END PROCESS;
    ----- Upper section: -----
    PROCESS (input, pr_state)
    BEGIN
        CASE pr_state IS
            WHEN state0 =>
                IF (input = ...) THEN
                    output <= <value>;
                    nx_state <= statel;
                ELSE ...
                END IF;
            WHEN statel =>
                IF (input = ...) THEN
                    output <= <value>;
                    nx_state <= state2;
                ELSE ...
                END IF;
            WHEN state2 =>
                IF (input = ...) THEN
                    output <= <value>;
                    nx_state <= state3;
                ELSE ...
                END IF;
            ...
        END CASE;
    END PROCESS;
END <arch name>;
```



مقدمه :

FPGA چیست؟ FPGA از تعداد بسیار زیادی گیت منطقی در داخل خودش ساخته می شود. این آی سی نسبت به عدم نویز پذیری بسیار مقاوم تر از انواع میکروکنترلر ها است. ویژگی مثبت FPGA ها نسبت به تراشه های میکروپروسسورها و میکروکنترلر ها و DSP ها این است که می تواند به صورت موازی بستورات را اجرا کند. همچنین توسط این آی سی می توان هر نوع میکروکنترلر و میکروپروسسوری را پیاده سازی نمود. به دلیل پردازش موازی این آی سی در صنایع مخابرات جهت سوییچ های مخابراتی و لایه های شبکه استفاده می شود. همچنین در صنایع نظامی جهت کنترل موشک استفاده می شود. این آی سی می تواند به عنوان پردازش سیگنال و تصویر نیز استفاده شود که این ویژگی توسط نرم افزار MATLAB پشتیبانی می شود. با استفاده از این آی سی می توان تمامی مدارات منطقی را پیاده نمود. همچنین این آی سی در PLC هایی مانند SV-400 استفاده می شود. ویژگی منفی این آی سی ها این است که قیمت آنها نسبت به میکروکنترلر ها بالاتر است.

زبان برنامه نویسی FPGA چیست؟ زبان های برنامه نویسی مانند C ، Basic و اسمبلی به صورت خط به خط دستورات را اجرا می کنند. این زبان ها قدرت کافی جهت پردازش موازی و حفظ حالات قبلی مدار را ندارند. در نتیجه زبان های توصیف سخت افزار HDL ساخته شد. مشهور ترین این زبان ها عبارتند از:

- (۱) System C : این زبان شباهت بسیار زیادی به زبان C دارد.
- (۲) System Verilog : این زبان نیز شباهت هایی به زبان C دارد. این زبان توسط شرکت Cadence گسترش پیدا کرد و کمی بعد از زبان VHDL استاندارد IEEE شد.
- (۳) VHDL : این زبان یکی از مشهور ترین زبان های توصیف سخت افزار می باشد که در وزارت دفاع آمریکا با همراهی شرکت های Texas Instrument ، IBM و Intermetrics ساخته شد و اولین زبان توصیف سخت افزار بسیار قوی می باشد.

امروزه مشهور ترین زبان های توصیف سخت افزار زبان VHDL و Verilog می باشد.

چگونه از نرم افزار ISE استفاده کنیم؟ راه اندازی سون سگمنت کاتد مشترک با زبان VHDL و FPGA های (Xilinx) قسمت بزرگی از کار کردن با FPGA مربوط به برنامه نویسی به زبان VHDL می باشد. با توجه به اینکه این زبان وابستگی به هیچ نرم افزاری ندارد، یعنی می توان برنامه ی VHDL نوشته شده برای یک FPGA Altera ر بر روی یک FPGA Xilinx ریخت با این فرق که باید مجددا توسط کامپایلر Xilinx سنتز شود. البته ممکن است بعضی از FPGA ها بلوک هایی داخل خودشان داشته باشند که دیگر FPGA ها ندارند) مانند پردازنده DSP که این یک حالت استثنا می باشد.

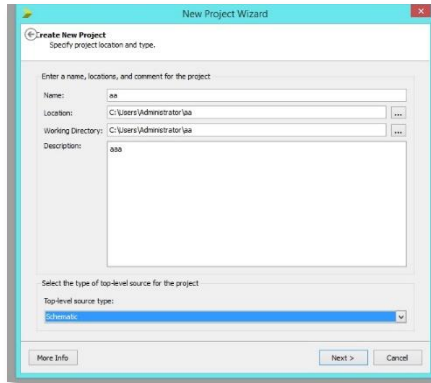
در این قسمت از نرم افزار ISE ورژن ۱۴ استفاده شده است که با Windows ۷ نیز کار می کند Crack. این نرم افزار را می توانید از قسمت داتلود دریافت نمایید.

نخست نرم افزار ISE را باز نمایید.

مراحل استفاده از برنامه

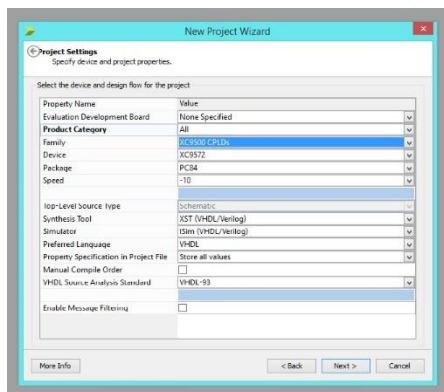
برای شروع برنامه از قسمت start/ ALL program/Xilinx Ise/ISE Design tools/ project

اجرا کنید. از منوی file گزینه New Project را انتخاب کنید. پنجره new project را انتخاب کنید.

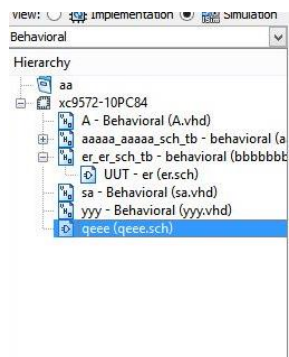
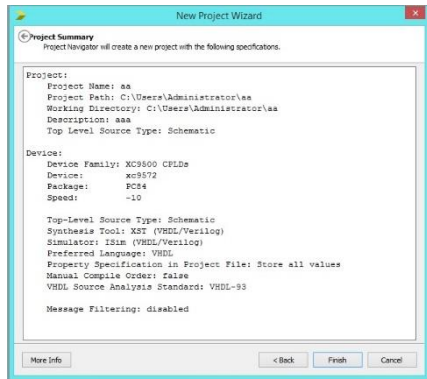


در قسمت new نام پروژه را تایپ می کنیم. مسیر مورد نظر را برای ذخیره پروژه جدید انتخاب کنید و در قسمت

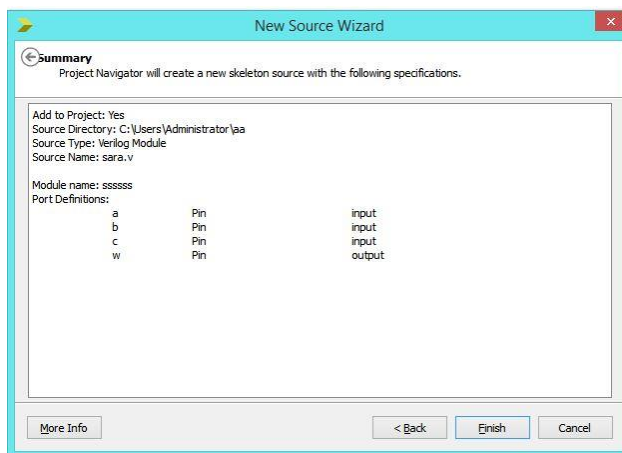
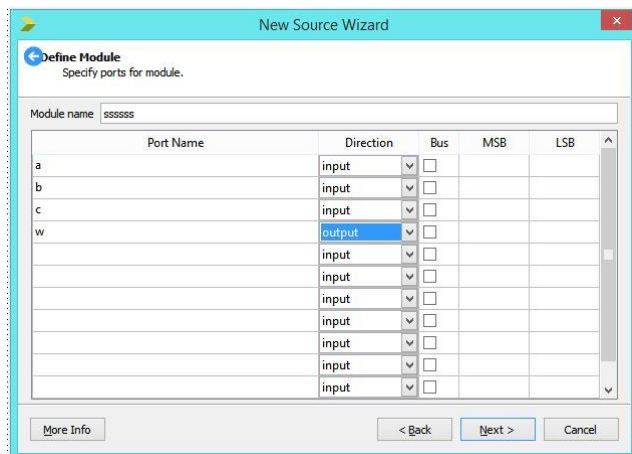
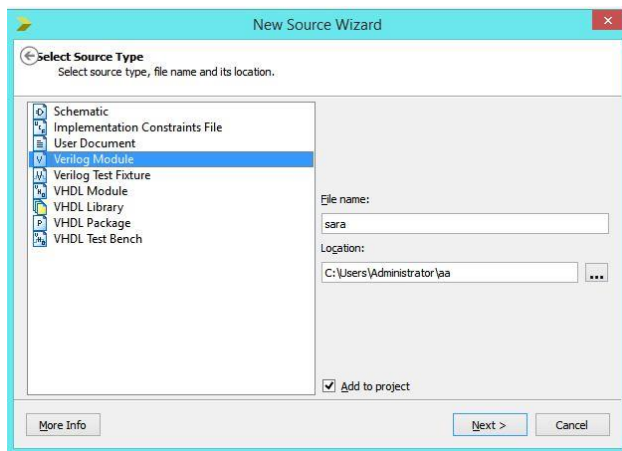
Top level source type گزینه HDL را انتخاب می کنیم. و سپس next را انتخاب می کنیم تا وارد پنجره project settings شوید.



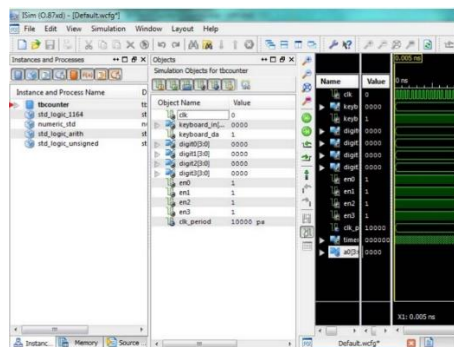
با کلیک کردن روی گزینه next خلاصه مشخصات پروژه نمایش داده می شود. با کلیک روی گزینه finish پروژه ایجاد می شود و در پنجره روبرو نمایش داده خواهد شد.



ابتدا در source کلیک راست کرده و گزینه Verilog Module را انتخاب کرده و next را انتخاب کرده و سپس پنجره بعد برای وارد کردن نام ورودی و خروجی های مدار است و گزینه finish را کلیک کرده.



برای نوشتن فایل test bench کلیه دستوراتی را که برای نوشتن برنامه اصلی استفاده می کنیم .



می توان به کاربرد علاوه براین سه دستور خاص نیز وجود دارند که فقط در محیط شبیه سازی و برای نوشتن فایل test bench

(۱) Wait:

این دستور به سه شکل زیر نوشته می شود

Wait for پر کاربردترین شکل این دستورات است.

(۲) Report:

در قسمت پایین پنجره پنبلی به نام console وجود دارد که گزارشی از روند شبیه سازی در آن جا نمایش داده می شود با استفاده دستور Report می توان پیغام خودرانیز مشخص کنید.

Report string [severity type] → مثال: Report "Test Completed" severity note

(۳) Assert:

با استفاده از این دستور می توان وضعیت یک سیگنال را با وضعیت مطلوب مقایسه کرد و در صورتی که مقدار سیگنال خلاف مقدار مورد نظر باشد در پنجره console نمایش داد.

مثال → Assert condition [report string] [severity type]

Assert s='1' report "Error, Expected 1 for S" severity error;

Test bench برای یک جمع کننده چهاربیتی:

```
-- *** Test Bench - User Defined Section ***
tb : PROCESS
BEGIN
    -- Initialize input signals
    A <= "0000";
    B <= "0000";
    Ci <= '0';

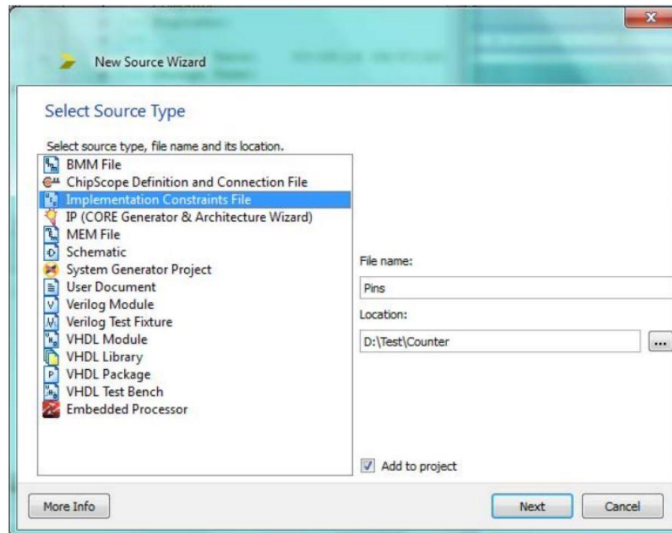
    -- Loop over all values of A
    for I in 0 to 15 loop
        -- Loop over all values of B
        for J in 0 to 15 loop
            -- Wait for output to update
            wait for 10ns;
            -- Check value of Sum
            assert (S = A + B) report "Expected sum of " &
                integer'image(to_integer(unsigned((A + B)))) & " for A = " &
                integer'image(to_integer(unsigned((A)))) & " and B = " &
                integer'image(to_integer(unsigned((B)))) & ", but was " &
                integer'image(to_integer(unsigned((S)))) severity ERROR;
            -- Increment to the next value of B
            B <= B + "0001";
        end loop;
        -- Increment to next value of A
        A <= A + "0001";
    end loop;

    -- Echo to user that test is finished
    report "Test completed";
    wait;
END PROCESS;
-- *** End Test Bench - User Defined Section ***
```

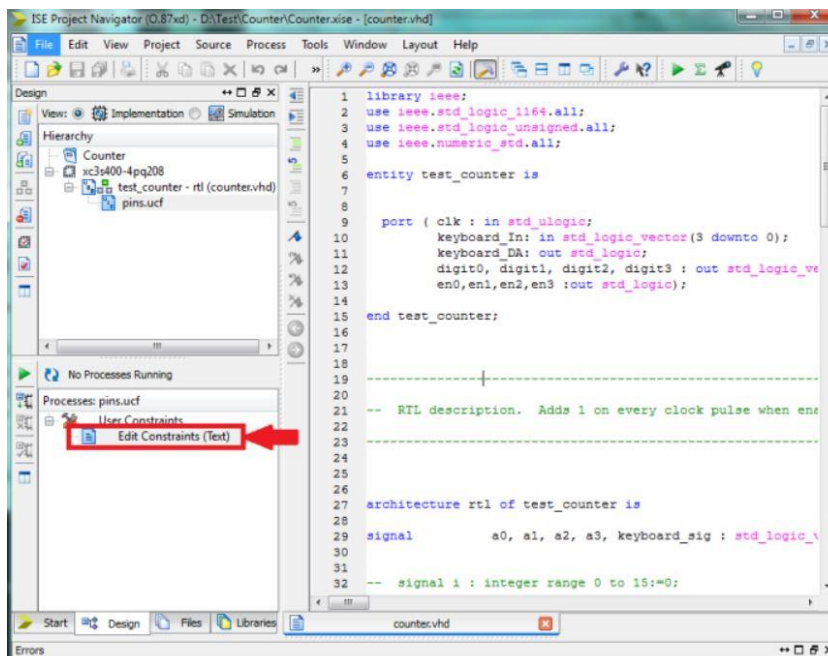
اختصاص شماره پایه های مورد نظر تراشه به ورودی ها و خروجی ها

بعد از رفع خطاهای احتمالی و شبیه سازی برنامه، باید فایل با پسوند ucf. جهت اختصاص پایه های مورد نظر از FPGA به ورودی خروجی های برنامه را ایجاد کرد. ابتدا بر روی پروژه کلیک راست کرده و گزینه New Source را انتخاب می نماییم. سپس طبق شکل ۴-۲۴ گزینه Implementation Constrains را انتخاب و در قسمت File name اسم فایل دلخواه را وارد می کنیم و بر روی Next کلیک و سپس Finish را می زنیم. در این حالت فایل مورد نظر با پسوند ucf. ایجاد می گردد (مثلا Pins.ucf).

ایجاد فایل usf برای اختصاص پایه های Fpg



حال مطابق شکل زیر به قسمت User Constraints رفته و زیر شاخه Edit Constraints را کلیک می کنیم تا باز شود (شکل ۴-۲۵).



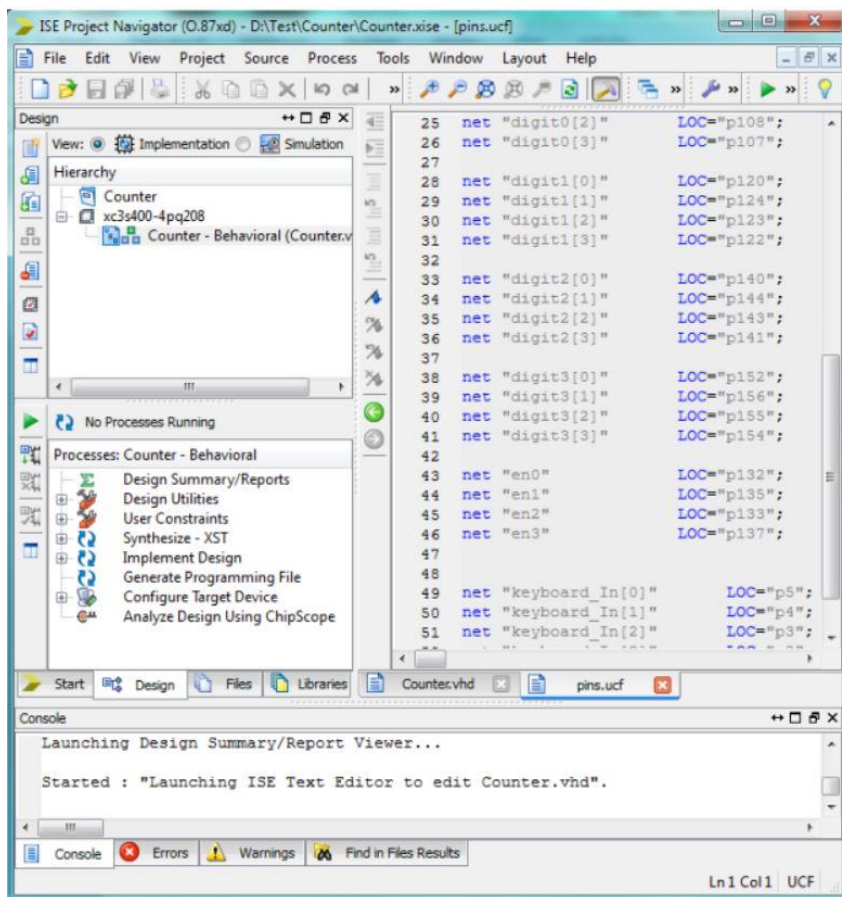
برای اختصاص هر پایه به یکی از پورت های ورودی یا خروجی به شکل زیر عمل می شود.

"شماره پایه مورد نظر از تراشه LOC="P نام پین ورودی یا خروجی " NET

برای مثال اگر بخواهیم پایه ۱۰۶ از تراشه را به پورت a اختصاص دهیم باید بنویسیم:

NET "a" LOC="P106 ";

شکل ۴-۲۶ نمونه ای از یک فایل ucf. نوشته شده را نشان می دهد.



نکته:

ساختار When-else دستوری است که برای مقداردهی سینگل ها به صورت شرطی به کار می رود به این معنا که سیگنال به یک مقدار تعلق می گیرد در صورتی که شرط آن برقرار باشد.

برای مثال برنامه مالتی پلکسر ۱*۲ با دستور when, else به این شکل می باشد.

0 <= t 0 when s='0' else

1 when s='1';

برای تک بیتی " می گذاریم و برای بیش از تک بیتی " " می گذاریم.

در جمله اول ; احتیاج ندارد.

جمله دوم با شرط دوم شروع می شود.

حتما انتهای دستور یعنی در خط دوم باید ; قرار داده شود.

1*۴ Mux:

ابتدا ورودی ها و خروجی ها را مشخص می کنیم.

ورودی ها: 0, 1, 2, 3, i

خروجی ها: 0

برای s باید bus فعال شود و 0=lsb, 1=msb باید باشد.

0 <= i 0 when s="00" else

1 when s="01" else

2 when s="10" else

3 when s="11" else;

ساختار if- then –else برای مقداردهی و یا انتخاب جملات می باشد و اجرای آن براساس ارزیابی صحیح یا خطا از یک شرط می باشد.

اگر شرط داخل پرانتز if برقرار باشد بعد از اجرای جمله ی do something انتخاب می گردد و اگر شرط برقرار نباشد بعد از else جمله something deferent انتخاب می گردد . درانتهای ساختار با end if; بسته می شود.

نکته: این شرط ها و جملات به همین شکل می توانند با دستور else if ادامه داشته باشد.

:Mux ۴*۱

If- the – else

If (condition) then

Do something;

Else

Do something;

Else

Do something different

End if;

.....

:Encoder

Y را به عنوان خروجی

آن را به صورت زیر Msb,Hsb هردو رافعال کرده و قسمت bus را به عنوان ورودی در نظر می گیریم و سپس w تنظیم می کنیم:

۱) Msb=۱ Lsb=۰

۲) Msb=۳ Lsb=۰

$Y \leq "۰"$ when $w(۰) = '۱'$ else

$"۰۱"$ when $w(۱) = '۱'$ else

$"۱۰"$ when $w(۲) = '۱'$ else

$"۱۱"$ when $w(۳) = '۱'$ else

مدار مقایسه کننده:

دو عدد تک بیت A<B را باهم مقایسه می کند A>B A=B A<B

Process (A,B) is

Begin

If(a>b) then

G<='۱'; L<='۰'; E<='۰'

Else if(a<b) then

G<='۰'; L<='۱'; E<='۰'

Else if(a=b) then

G<='۰'; L<='۰'; E<='۱'

End process;

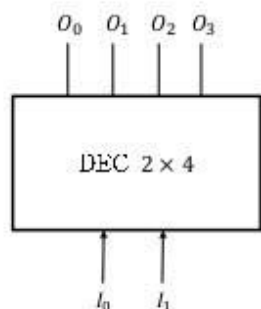
مقایسه کننده ۴ بیتی :

اگر A,B هر کدام چهاربیت باشند. مانند برنامه قبل عمل میکنیم.

:Decoder

دیگر:

مداری ترکیبی دارای n ورودی و 2^n خروجی



I_1	I_0	O_3	O_2	O_1	O_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

برنامه decoder:

$W \leftarrow "100"$

$W \leftarrow "001" \text{ when } y = "11" \text{ else}$

$W \leftarrow "0010" \text{ when } y = "10" \text{ else}$

$W \leftarrow "0100" \text{ when } y = "01" \text{ else}$

$W \leftarrow "1000" \text{ when } y = "00" \text{ else}$

در اینجا باید فقط y را به عنوان ورودی و w را به عنوان خروجی تعریف کنیم و bus آن هم فعال شود نیازی به تعریف w_3 تا w_7 نمی باشد.

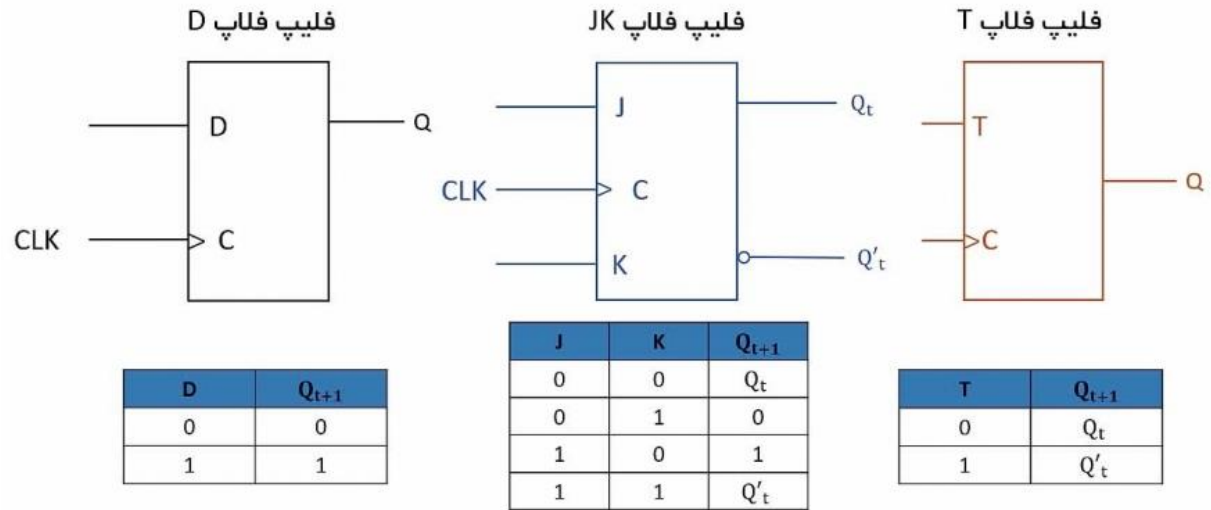
انواع مدارات ترتیبی:

➤ فلیپ فلاپ d

➤ فلیپ فلاپ t

➤ فلیپ فلاپ jk

انواع فلیپ فلاپ ها:



مدارات منطقی سنگرون:

If (clk'event and clk='1') then	لبه بالارونده
If (clk'event and clk='0')	لبه پائین رونده

A: out std- Logic: = '0'

Process (clk, t) is

Begin

If (clk'event 'and clk='1') then

If (t='10') then

q<=not (q);

End if;

End if;

End process;



بعد خط (clk 'event') if نوشته میشود و یک end if اضافه میشود.

اگر reset برابر صفر باشد فلیپ فلاپ نوع D کار می کند به هر مقداری در ورودی D باشد به q منتقل می شود ولی زمان rest برابر ۱ باشد خروجی بدون در نظر گرفتن مقدار ورودی D صفر می شود.

T	q
۰	Q'(T)
۱	Q (T)

Q: input std-logic: ='۰'

Process (clk, T) is

Begin

If (clk'event

and clk='۱') then

If (t='۰') then

q<=not (q);

Else

q<=q;

End if;

End if;

End process;

نکته :

Entetie: مقداردهی باید شود.

'=' : نوشته میشود.

فلیپ فلاپ z-k:

q: out std- logic = '0'

Process (clk, j, k) is

Begin

If (clk'event 'and clk='0') then

If (j='0' and clk='0') then q<=q;

Elsif (j='0' and k='0') then q<='0';

Elsif (j='1' and k='0') then q<='1';

Elsif (j='1' and k='1') then q<=not (q);

End if;

End if;

End process;

jk با خط rest

If (rst='0') then q<='0';

Elsif (rst='1') then q<='0';

عامل لبه باشد

If (clk='0') then

Q<=d;

End if;

End process;

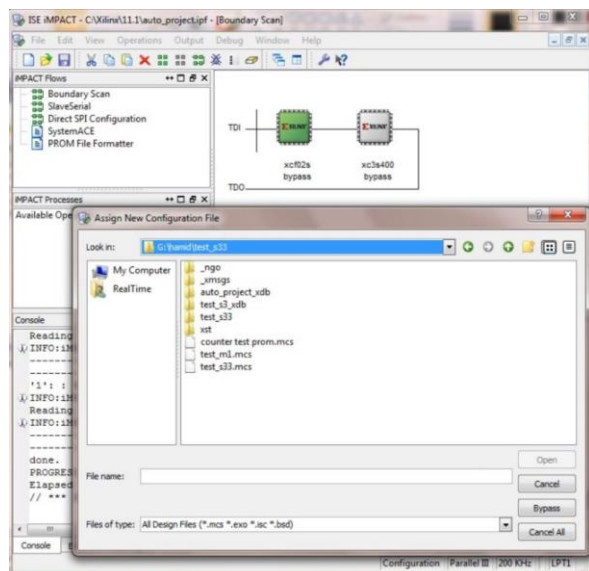
Process (clk, d) is

هر برنامه که می‌خواهیم فایل usf برایش می‌سازیم برای ساختن usf روی گزینه source بعد گزینه new source را گزینه innmeplaton انتخاب می‌کنیم یک اسم برایش انتخاب می‌کنیم بعد next و finsh را انتخاب می‌کنیم.

فایل usf کلیک و بعد قسمت process گزینه user..... می‌کنیم گزینه assign packaging پنجره پایه ic رامشخص می‌کنیم.

فایل save و ok را می‌زنیم. Source فایل هایلات می‌شود. Process گزینه Implanted design گزینه grantee programing file و بعد گزینه configure device را دابل کلیک کرده می‌کنیم. در پنجره finsh را انتخاب می‌کنیم (شمای ic) پنجره ایی که باز شده نشان برنامه ایی کابلها را شناسایی شده و اتوماتیک کابلها وصل شده. فایل با پسوند Jed انتخاب می‌کنیم و open را می‌زنیم. کنار ic یک خط آمده روش یک بار کلیک راست کرده که ic سبز شود و هنگامی که سبز شد اولین آیکون program بعد پنجره ok را می‌زنیم. اگر موفقیت آمیز باشد آبی رنگ succfull را می‌دهد ic به برنامه داده شده.

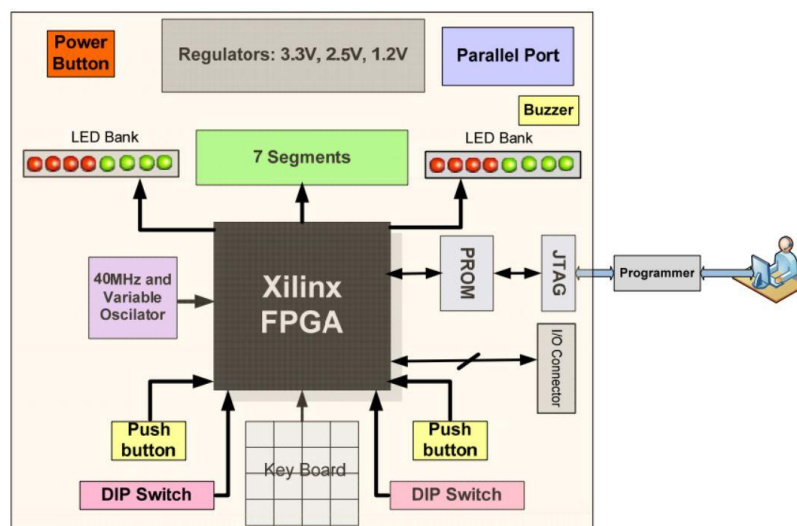
برنامه ریزی FPGA یا حافظه :



پس از انتساب فایل مورد نظر به FPGA یا حافظه، روی تراشه مربوطه کلیک راست کرده و گزینه Program را انتخاب می کنیم. در پایان پیغامی مبنی بر program succeeded ظاهر می شود. تفاوتی در اولویت انتخاب تراشه FPGA یا حافظه جهت برنامه ریزی وجود ندارد. اگر فقط تراشه FPGA را برنامه ریزی کرده باشیم، با قطع تغذیه برد، برنامه ی پروگرام شده بر روی تراشه پاک می شود. در صورتی که حافظه را برنامه ریزی کرده باشیم با قطع تغذیه برنامه پروگرام شده بر روی آن از بین نمی رود.

معرفی سیستم آزمایشگاه:

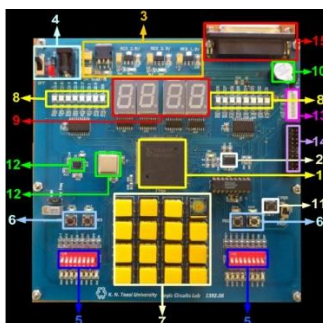
بورد FPGA موجود در آزمایشگاه بر اساس تراشه Spartan3-XC3s400 شرکت Xilinx طراحی شده است. شکل زیر بلوک دیاگرام کلی بورد طراحی شده را نشان می دهد. نحوه ارتباط بخش های مختلف با FPGA در شکل زیر دیده می شود.



امکانات بورد عبارتند از:

- تراشه FPGA از خانواده SPARTAN3 و نوع XC3S400-PQG208
- حافظه XCF02S به ظرفیت 2MBit
- دو عدد دیپ سویچ هشت تایی جهت اعمال داده های ورودی به تراشه
- چهار عدد کلید فشاری جهت اعمال ورودی های پالسی به مدارهای طراحی شده
- کیبورد ۱۶ تایی برای اعمال اعداد دهدهی ۰ تا ۱۵ به ورودی های مدار
- شانزده عدد دیود نوری مجهز به بافر جهت نمایش داده های خروجی

- ۴ عدد سون سگمنت جهت نمایش داده های خروجی
- یک زنگ برای اعمال خروجی های صوتی
- کلید ریست برای ریست کردن FPGA
- دو نوع مدار کلاک با فرکانس های مختلف: یک کلاک ۴۰ مگاهرتز و یک کلاک متغیر برای فرکانس های پایین
- امکان برنامه ریزی تراشه FPGA با استفاده از پروگرامر USB و یا پارالل



۱- تراشه FPGA: تراشه FPGA هسته اصلی برد را تشکیل می دهد. این تراشه از سلول های منطقی قابل برنامه ریزی ساخته شده است که ارتباط بین این سلول ها نیز قابل برنامه ریزی است. تراشه های FPGA بسیار سریع هستند و برای پیاده سازی مدارهای دیجیتال استفاده می شوند. پس از برنامه ریزی FPGA می توان مدار طراحی شده را تغییر داد و FPGA را مجدداً برنامه ریزی نمود. سرعت بالای این تراشه ها آن ها را مساعد کارهای پردازشی سنگین مثل پردازش تصویر، صدا و ... می کند و سرعت این پردازش نسبت به سیستم های دیگر خیلی بالاتر است. سرعت بالای این تراشه ها از قابلیت پردازش موازی و انجام چندین عملیات به صورت هم زمان ناشی می شود. میکروکنترلرها، تراشه های DSP و میکروپروسسورها دستورات را به ترتیب و پشت سر هم اجرا می کنند و قابلیت پردازش موازی در آن ها وجود ندارد.

در این برد از تراشه XC3S400-4PQ208I که از خانواده Spartan3 و از تولیدات شرکت Xilinx است استفاده شده است. این تراشه دارای ۴۰۰۰۰۰ هزار گیت منطقی و حداکثر ۲۸۴ پایه I/O است. کلیه کارهای پردازش و برقراری ارتباط با قطعات جانبی و کاربر توسط این تراشه انجام می شود.

۲- حافظه: شرکت Xilinx حافظه هایی با ظرفیت ۱ تا ۳۲ مگابایت را که از نوع Platform Flash PROM هستند عرضه کرده است. این حافظه ها که در دو سری XCFxxS (تغذیه ۳,۳ ولت) و XCFxxP (تغذیه ۱,۸ ولت) معرفی شده اند برای برنامه ریزی FPGA در مد Master/Slave Serial به کار می روند. در این برد از حافظه XCF02S با ظرفیت 2MBit استفاده شده است.

۳- رگولاتورها: تراشه XC3S400 و سایر آی سی های روی برد به چهار ولتاژ مختلف برای کار نیاز دارند: ۱,۲ ولت، ۲,۵ ولت، ۳,۳ ولت و ۵ ولت. ولتاژ ۵ ولت توسط آداپتوری که به تغذیه برد متصل می شود تامین می گردد. سایر ولتاژ ها با استفاده از سه رگولاتور یا تنظیم کننده ولتاژ تامین می شوند. کنار تغذیه ورودی و هر یک از رگولاتورها یک دیود نوری تعبیه شده است که وجود یا عدم وجود ولتاژ مربوطه را نشان می دهد. یک دیود نیز برای حفاظت از تغذیه معکوس نیز روی برد قرار داده شده است.

۴- تغذیه: تغذیه برد ولتاژ ۵ ولت است که با اتصال یک آداپتور ۵ ولت به کانکتور تغذیه، وارد برد می شود. با اتصال کلید کنار کانکتور، برد روشن می شود.

۵- دیپ سویچ: این برد شامل دو عدد دیپ سویچ هشت تایی است که برای گرفتن ورودی از کاربر استفاده می شوند. یک دیپ سویچ مجموعه ای از چند کلید جداگانه است که در بسته ای مشابه با یک IC قرار دارند. برای اطمینان از اتصال کلیدهای دیپ سویچ، هر کلید به یک دیود نوری مجهز شده است که به هنگام اتصال کلید روشن می شود.

۶- کلیدهای فشاری: وجود چهار عدد کلید فشاری امکان اعمال ورودی های پالسی به مدار را فراهم ساخته است. برای هر کلید فشاری یک یک دیود نوری قرار داده شده است که به هنگام فشار دادن کلید روشن می شود.

۷- صفحه کلید: این صفحه کلید از ۱۶ عدد کلید فشاری تشکیل شده است. با فشردن هر کلید یک سطر و یک ستون از صفحه کلید فعال می شود. خروجی سطرها و ستون های صفحه کلید به آی سی 74922 داده می شوند و آی سی کد باینری چهار بیتی مربوطه را استخراج می کند و به عنوان ورودی به FPGA می دهد.

۸- دیودهای نوری: دیودهای نوری در چهار رنگ مختلف و برای نمایش خروجی مورد نظر کاربر روی برد قرار داده شده اند. این برد مجهز به ۱۶ دیود نوری است و بنابراین قابلیت نمایش اعداد تا ۱۶ بیت را داراست.

۹- سون سگمنت: چهار عدد سون سگمنت برای نمایش اعداد به صورت دهدهی روی برد قرار داده شده است. هر سون سگمنت مجهز به یک آی سی مبدل باینری به سون سگمنت است. بنابراین کاربر می تواند هر خروجی ۴ بیتی را مستقیماً روی یک سون سگمنت نشان دهد.

۱۰- زنگ: اگر در مدارهای طراحی شده نیاز به یک خروجی صوتی باشد از این زنگ استفاده می شود.

۱۱- کلید ریست: از این دکمه فشاری برای ریست کردن FPGA استفاده می شود.

۱۲- اسیلاتور: یک اسیلاتور با فرکانس 40MHz روی برد قرار گرفته که به یکی از پایه های FPGA متصل می باشد و می تواند به عنوان کلاک اصلی برد مورد استفاده قرار گیرد. اسیلاتور دیگری روی برد تعبیه شده است که فرکانس آن قابل تنظیم است و می تواند از ۰/۰۰۱ هرتز تا ۳۰۰ کیلوهرتز باشد. این اسیلاتور در مواردی استفاده می شود که به فرکانس های پایین نیاز است و برای راحتی کاربر قرار داده شده است. فرکانس این کلاک با یک مقاومت متغیر تنظیم می شود. با جابجا کردن یک Jumper بین دو حالت ممکن خود، می توان انتخاب کرد کدام یک از کلاک ها به مدار اعمال شوند.

۱۳- پورت JTAG: در صورتی که بخواهیم تراشه FPGA را با استفاده از پروگرامر USB برنامه ریزی کنیم از باید آن را پورت JTAG روی برد متصل کنیم. FPGA از طریق پورت JTAG با کامپیوتر ارتباط برقرار کرده و تبادل داده می کند. JTAG در واقع استاندارد است که برای برقراری ارتباط بین یک دستگاه و اجزای جانبی استفاده می شود. وظیفه اصلی این پورت برنامه ریزی FPGA و یا خواندن برنامه ذخیره شده

۱۴- کانکتور I/O: برای ارتباط با خارج برد یک کانکتور ورودی-خروجی ۲۰ پایه در نظر گرفته شده که به پایه های کاربری باقیمانده FPGA متصل شده است و می توان از آن برای ارتباط با اجزای جانبی مانند LCD و یا برد های دیگر استفاده کرد.

روی آن است. به هنگام برنامه ریزی FPGA، داده از طریق کامپیوتر به یک پروگرامر داده می شود و سپس از طریق پورت JTAG به حافظه یا FPGA منتقل می شود.

۱۴- کانکتور I/O: برای ارتباط با خارج برد یک کانکتور ورودی-خروجی ۲۰ پایه در نظر گرفته شده که به پایه های کاربری باقیمانده FPGA متصل شده است و می توان از آن برای ارتباط با اجزای جانبی مانند LCD و یا برد های دیگر استفاده کرد.

۱۵- پورت موازی: برای برنامه ریزی تراشه FPGA از طریق پورت موازی کامپیوتر، باید یک سر کابل پارالل را به کامپیوتر و سر دیگر آن را به کانکتور نشان داده شده روی برد متصل کرد.

معرفی سیستم آزمایشگاه:

FPGA از طریق پایه های I/O با امکانات جانبی مورد ارتباط برقرار می کند. ورودی ها و خروجی های روی برد مطابق جدول زیر به پایه های I/O تراشه FPGA متصل شده اند.

INPUT PINS			
DIP Switch 2		DIP Switch 1	
D0 8	D4 4	D0 43	D4 37
D1 9	D5 5	D1 42	D5 36
D2 10	D6 12	D2 40	D6 35
D3 11	D7 15	D3 39	D7 34
Key board			
Key – D3	Key – D2	Key – D1	Key – D0
52	51	50	48
Clocks			
183			

OUTPUT PINS							
LEDs							
LED 1	LED 2	LED 3	LED 4	LED 5	LED 6	LED 7	LED 8
93	94	95	96	97	100	101	102
LED 9	LED 10	LED 11	LED 12	LED 13	LED 14	LED 15	LED 16
161	162	165	166	167	168	169	171
7SEG 4		7SEG 3		7SEG 2		7SEG1	
D0 152		D0 140		D0 120		D0 106	
D1 156		D1 144		D1 124		D1 109	
D2 155		D2 143		D2 123		D2 108	
D3 154		D3 141		D3 122		D3 107	
7SEG 4 _En		7SEG 3 _En		7SEG 2 _En		7SEG 1 _En	
137		133		132		132	
BUZZER							
111							