VHDL

فهرست مطالب

- مرور کلی بر طراحی مدارهای مجتمع الکترونیکی
- زبان VHDL (ساختارها و دستورالعمل ها)
- طراحی مدار در سطوح مختلف
  - طراحی در سطح الگوریتم
  - طراحی در سطح رجیستر
  - طراحی در سطح گیت
- مباحث زمان بندی و تخصیص منابع
- بهینه سازی
- نرم افزار شبیه ساز Active-HDL
- نرم افزار Xilinx ISEو پیاده سازی سخت افزاری توسط FPGA

مراجع درس

- 1. طراحی سیستم دیجیتالی با استفاده از VHDL، علیرضا فتاح.
- 2. VHDL: Analysis and Modeling of Digital Systems, Navabi
- 3. VHDL: Programming by Example. , Douglas Perry
- 4. RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability 1st Edition by Pong P. Chu
- 5. Circuit Design with VHDL, Volnei A. Pedroni

نحوه ارزیابی

- 1. میان ترم
- 2. پایان ترم
- 3. تکالیف

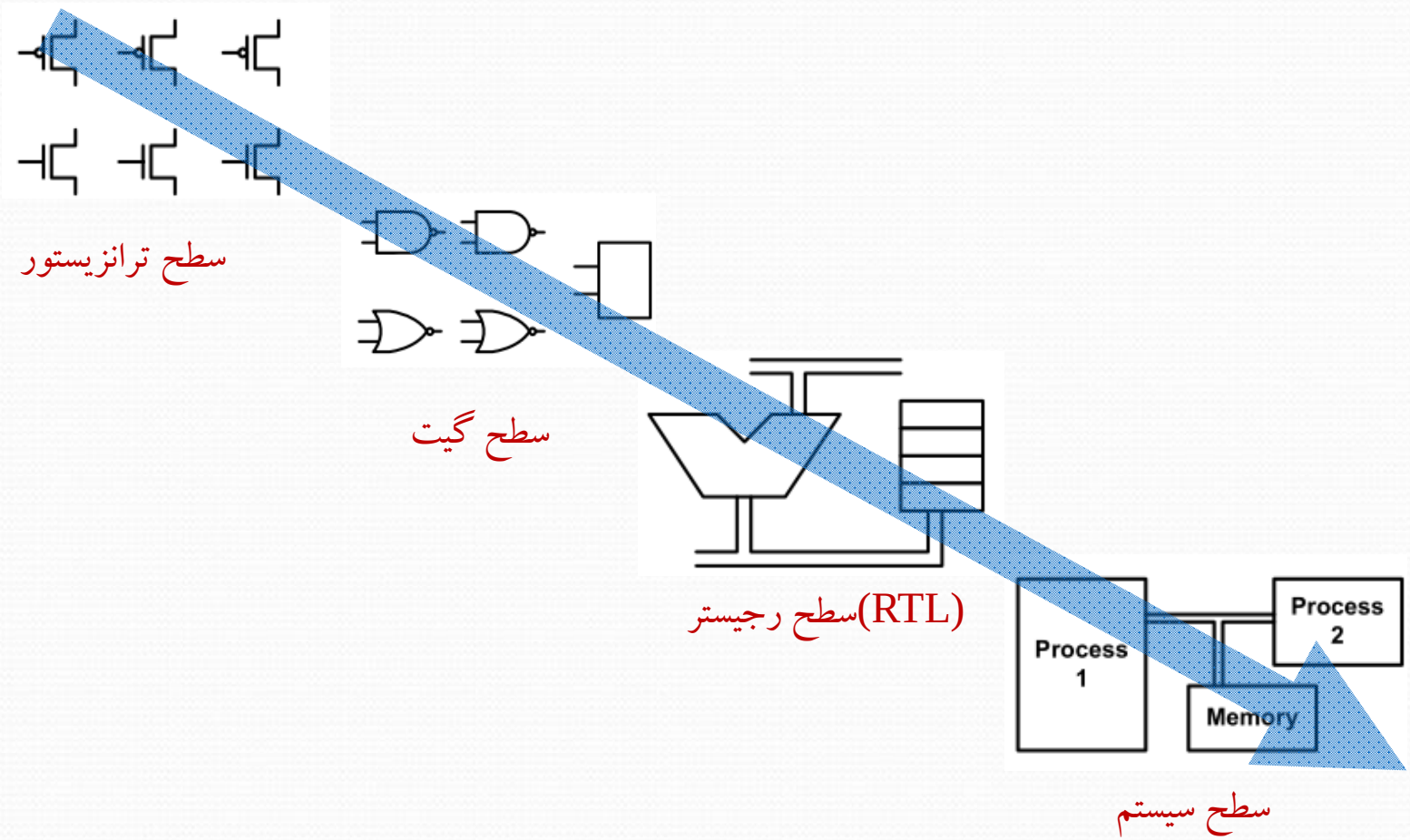
طراحی مدارهای الکترونیکی

- 1. دیجیتال
  - نویز کمتر
  - طراحی ساده تر
  - قابلیت برنامه ریزی مجدد
  - قابلیت پیکر بندی مجدد
- 2.آنالوگ
  - طراحی شامل در نظر گرفتن تعداد زیادی بده بستان است
  - طرح با تغییر شرایط مدار نیازمند به روز شدن است
- 3.دیجیتال - آنالوگ

چهار مرحله اساسی در ساخت مدارهای مجتمع

- 1.طراحی (design)
- 2. ساخت (fabrication)
- 3. تست (testing)
- 4. بسته بندی (packaging)

سطوح توصیف مدار



روند طراحی

- 1. با استفاده از روش های مرسوم شماتیک مداری ، جدول کارنو و ...

عدم کارامدی در شرایطی که مدار پیچیده است و ترانزیستورهای مدار بسیار زیاد است.

- 2. استفاده از ابزارهای خودکار کردن طراحی

(Electronic Design Automation) EDA Tools

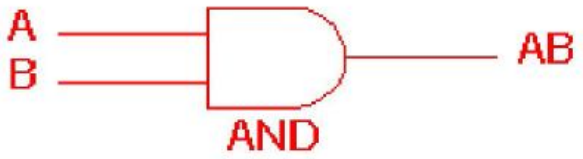
به وجود آمدن زبان های توصیف سخت افزاری (HDL)

یادآوری جدول درستی و جدول کارنو

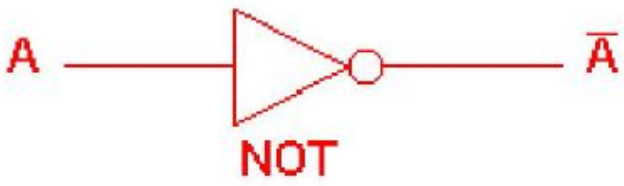
- جبر بولی یا جبر دو مقداره عملیات روی سیگنال های دو مقداره که فقط مقدار صفر یا یک دارند را گویند.
- برای عملگرهای مختلف در جبر بولی می توان جدول درستی رسم کرد که عبارت است از جدولی که تمام حالتهای ممکن متغیرها به همراه مقدار حاصل از اعمال عملگر مورد نظر روی آن را نشان دهد.
- سه عملگر اصلی در جبر دو مقداره عبارتند از
  - OR
  - AND
  - NOT

جدول درستی مربوط به سه عملگر اصلی

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

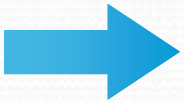


NOT gate	
A	$\bar{A}$
0	1
1	0

جدول کارنو

- جدول کارنو روشی گرافیکی برای ساده سازی یک عبارت جبر دو مقداره است.
- ساده سازی عبارت جبر دو مقداره به منظور طراحی ساده تر صورت می پذیرد.
- با توجه به شکل زیر برای یک جدول درستی دو متغیره جدول کارنویی معادل شکل زیر وجود دارد.
- برای 2 متغیر 4 حالت مختلف مینترم وجود دارد

x	y	minterm
0	0	x'y'
0	1	x'y
1	0	xy'
1	1	xy



X \ Y	0	1
0	x'y'	x'y
1	xy'	xy

ادامه:

- مینترم های چپ و راست به ترتیب شامل y’ و y هستند.
- مینترم های بالا و پایین به ترتیب شامل x’ و x هستند.

X	Y=0	Y=1
0	x’y’	x’y
1	xy’	xy

	Y’	Y
X’	x’y’	x’y
X	xy’	xy

- عبارت زیر برای یک جدول دو متغیره شامل جمع دو مینترم را در نظر بگیرید

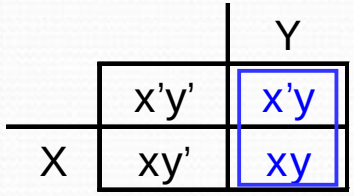
$$x’y’ + x’y$$

- هر دوی این عبارات در سطر بالای جدول کارنو ظاهر می شوند که فقط شامل x’ است. بنابراین با ساده سازی با جدول کارنو مقدار عبارت بالا برابر x’ است.
- اگر از جبر دو مقدار برای ساده سازی عبارت بالا استفاده شود داریم

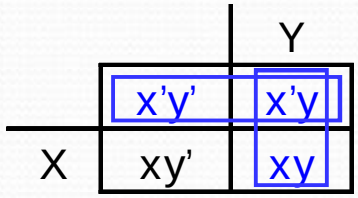
$$x’y’ + x’y = x’(y’ + y) \quad [\text{ خاصیت توزیع پذیری }]$$
$$= x’ \bullet 1 \quad [\, y + y’ = 1 \,]$$
$$= x’ \quad [\, x \bullet 1 = x \,]$$

مثالهای دیگر از ساده سازی با جدول دو متغیره

- مثال دیگر عبارت $x'y + xy$ است. هر دو مینترم در سمت راست جدول کارنو هستند که شامل y است بنابراین مقدار عبارت برابر با y است.



- مثال دیگر عبارت $x'y' + x'y + xy$ است.
  - مینترم های $x'y' + x'y$ در سطر بالای جدول کارنو هستند که شامل x' است
  - مینترم های $x'y + xy$ در سمت راست جدول کارنو هستند که شامل y است.
  - بنابراین حاصل ساده سازی عبارت بالا به صورت $x' + y$ خواهد بود



جدول کارنوی 3 متغیره

- نحوه قرار گیری مینترم ها در جدول 3 متغیره به صورت زیر است.

X \ YZ	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

X \ YZ	00	01	11	10
0	x'y'z'	x'y'z	x'yz	x'yz'
1	xy'z'	xy'z	xyz	xyz'

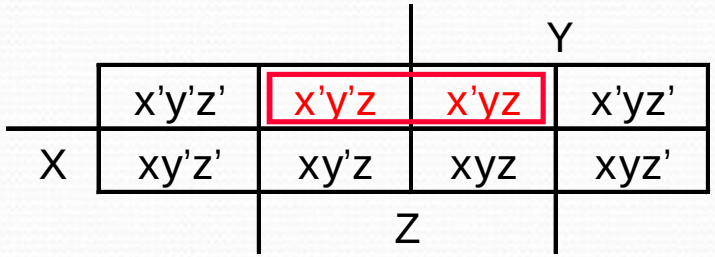
- راه دیگر برای برچسب گذاری جدول کارنو به صورت زیر است

X \ YZ	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

X \ YZ	00	01	11	10
0	x'y'z'	x'y'z	x'yz	x'yz'
1	xy'z'	xy'z	xyz	xyz'

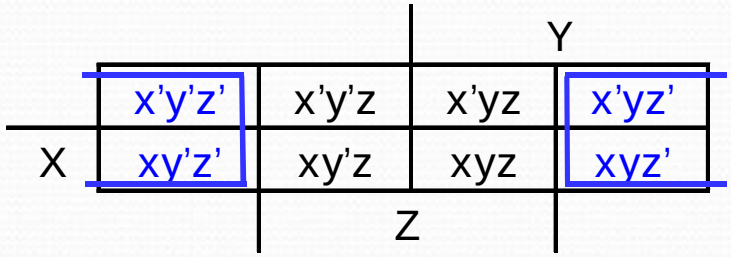
ادامه

- با این نحوه قرار گیری مینترم ها هر ترکیبی از مربع های 2 4 یا 8 تایی از مینترم ها قابل ساده سازی است.



$$x'y'z + x'yz$$
$$= x'z(y' + y)$$
$$= x'z \bullet 1$$
$$= x'z$$

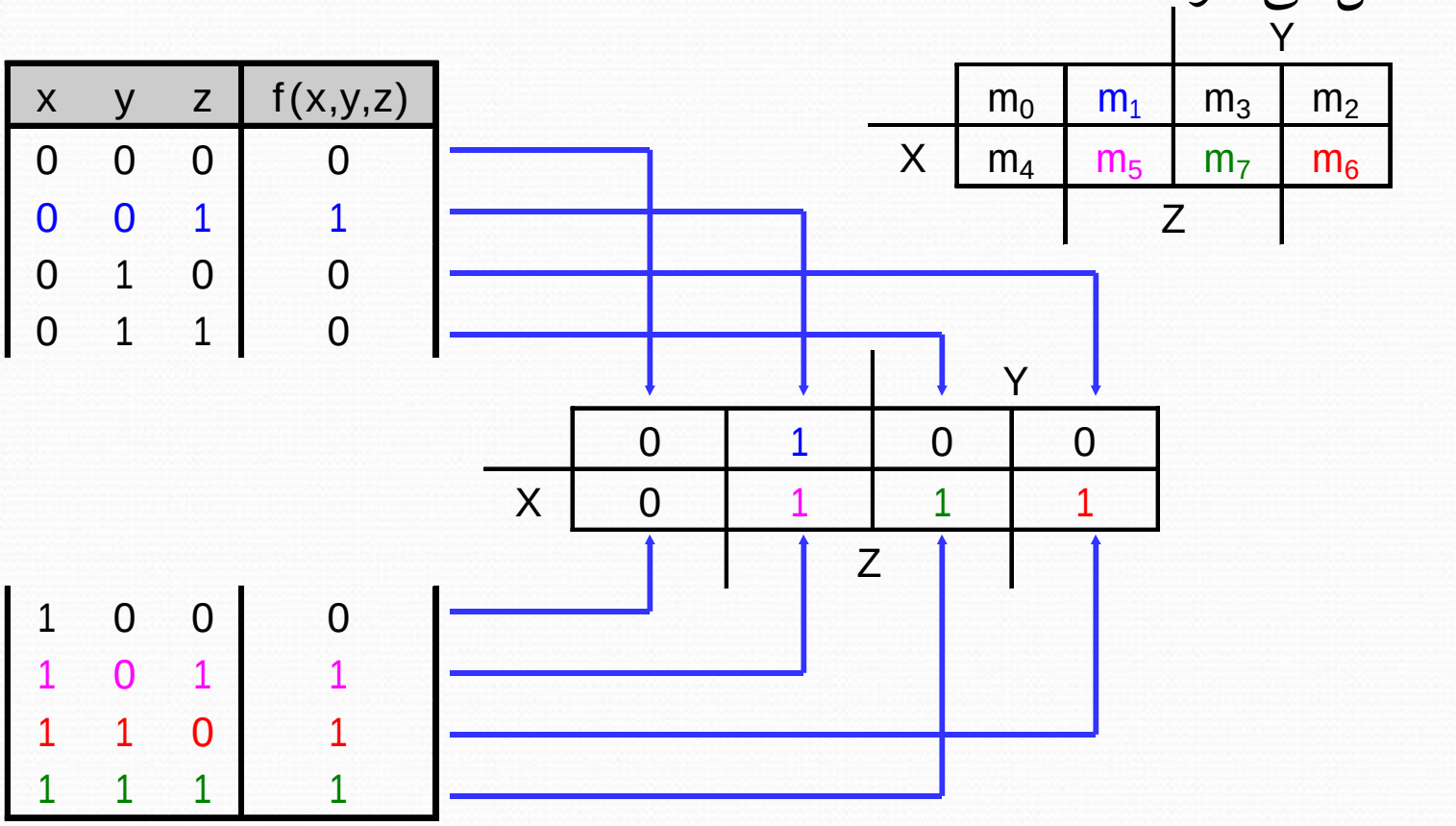
- در شکل زیر 4 مینترم از ستون سمت راست و چپ جدول قابل ساده سازی هستند.



$$x'y'z' + xy'z' + x'yz' + xyz'$$
$$= z'(x'y' + xy' + x'y + xy)$$
$$= z'(y'(x' + x) + y(x' + x))$$
$$= z'(y'+y)$$
$$= z'$$

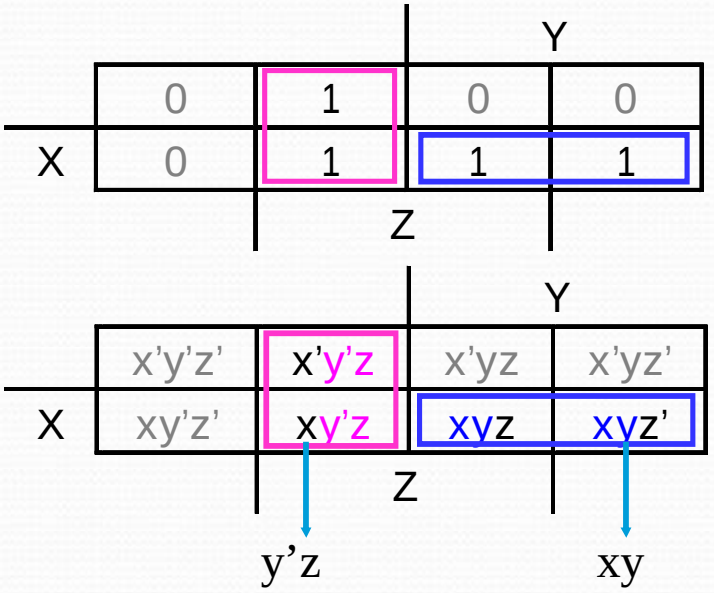
پر کردن جدول کارنو با استفاده از جدول درستی

- می توان به طور مستقیم جدول کارنو را با استفاده از جدول درستی پر کرد.
- در اینصورت خروجی هر سطر i از جدول درستی به مربع mi از جدول کارنو منتقل می شود.



نوشتن عبارات مینترم ها از روی جدول کارنو

- سخت ترین قسمت در ساده سازی با جدول کارنو گروه بندی 1 های جدول است.
- گروههای 1، 2، 4 و 8 تایی از 1 های جدول را دسته بندی کرده و دور آنها مستطیل کشیده و مینترم معادل آنها را بنویسید.



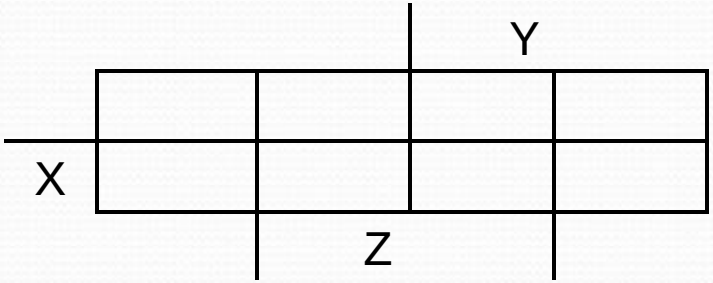
$F(x,y,z)= y'z + xy$

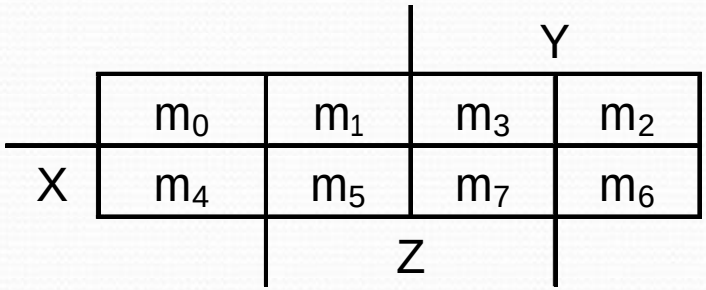
به منظور داشتن ساده ترین پاسخ

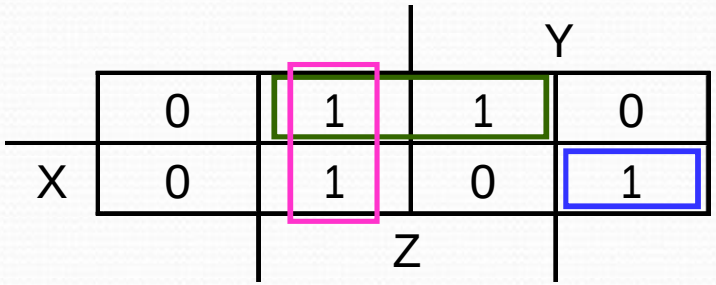
- باید مستطیل هایی که دور یک ها کشیده می شوند تعدادشان مینمم شود تا تعداد مینترم های نهایی حداقل شوند.
- مستطیل ها باید تا حد امکان ماکزیمم 1 های مجاور را در برگیرند مثلا 4 تا 1 مجاور بهتر است یک مستطیل 4 تایی تشکیل دهند تا دو تا دو تایی تا تعداد مینترم های حاصل مینمم شوند.
- مستطیل ها می توانند همپوشانی داشته باشند تا بزرگتر شوند.

تمرین: ساده سازی با جدول کارنو

- ساده سازی جمع مینترم های m1+m3+m5+m6



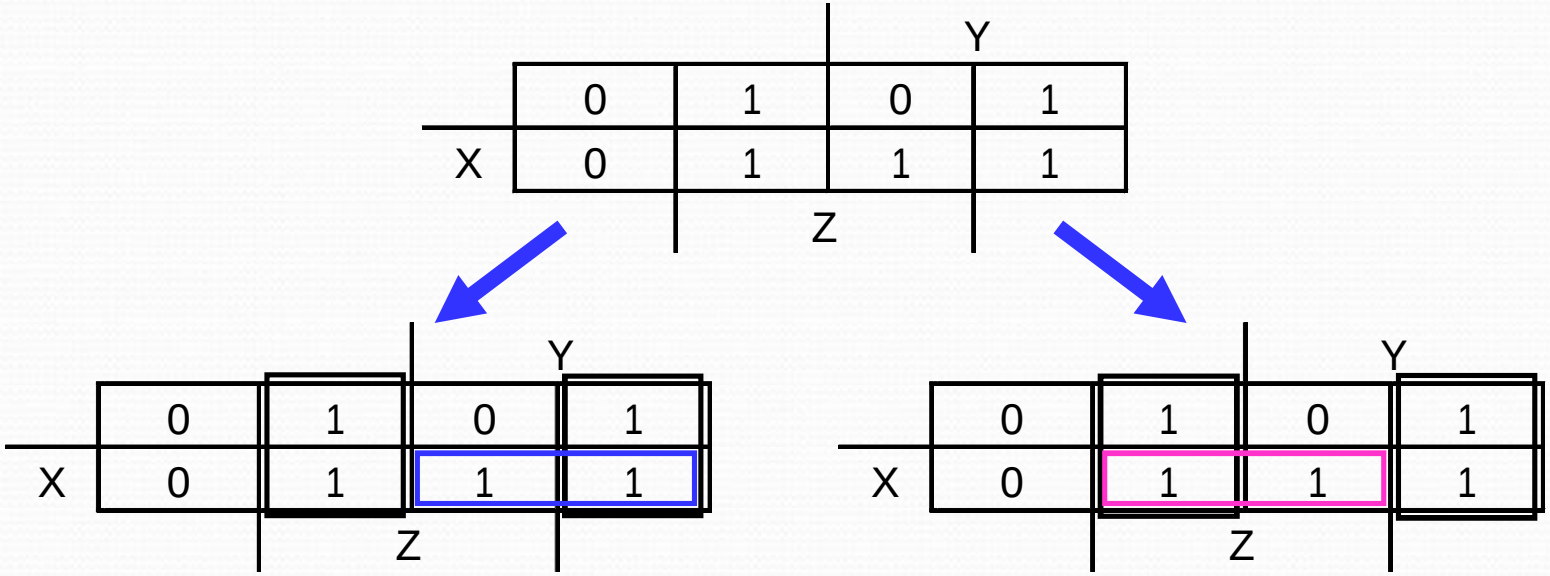




- عبارت ساده شده نهایی x’z + y’z + xyz’

جوابهای ممکن با جدول کارنو

- ساده سازی با جدول کارنو لزوما جواب یکتایی نمی دهد.

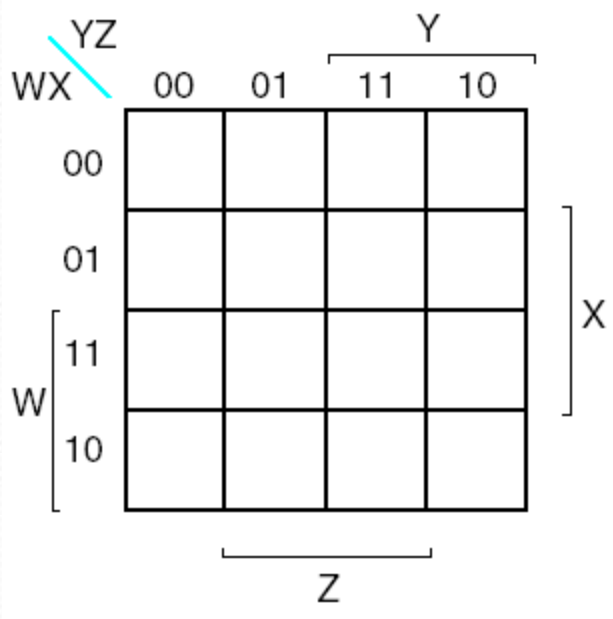


y’z + yz’ + xy

y’z + yz’ + xz

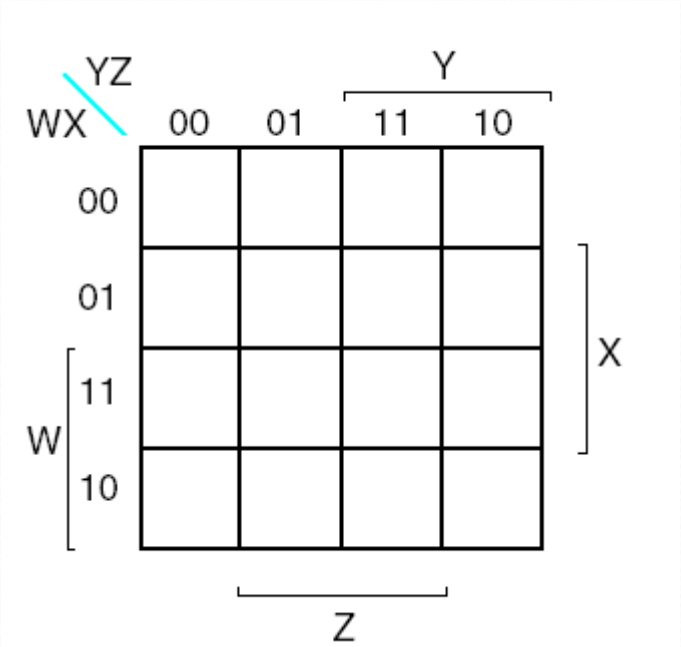
جدول کارنوی 4 متغیره

- در جدول کارنوی 4 متغیره مینترم ها در سطر سوم و چهارم و ستون سوم و چهارم جابه جا می شوند.
- با این کار ستون ها و سطر های کنار هم مختص یک متغیر یا معکوس آن متغیر خواهند بود.



- در اینجا نیز با دسته بندی مینترم ها به دسته های 1، 2، 4 ، 8 و 16تایی میتوان ساده سازی برای عبارت مورد نظر انجام داد.

نحوه قرار گیری مینترم ها در جدول کارنوی 4 متغیره



			Y	
w'x'y'z'	w'x'y'z	w'x'yz	w'x'yz'	
w'xy'z'	w'xy'z	w'xyz	w'xyz'	X
wxy'z'	wxy'z	wxyz	wxyz'	
wx'y'z'	wx'y'z	wx'yz	wx'yz'	
	Z			

W

			Y	
m_0	m_1	m_3	m_2	
m_4	m_5	m_7	m_6	X
m_{12}	m_{13}	m_{15}	m_{14}	
m_8	m_9	m_{11}	m_{10}	
	Z			

W

مثال: ساده سازی $m_0+m_2+m_5+m_8+m_{10}+m_{13}$

- جدول کارنوی مثال به صورت زیر است:

WX \ YZ	00	01	11	10
00	1	0	0	1
01	0	1	0	0
11	0	1	0	0
10	1	0	0	1

WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

- بنابراین می توان گروه بندی به صورت شکل زیر انجام داد و عبارت ساده شده نهایی به صورت $xzy'+x'z'$ خواهد بود.

WX \ YZ	00	01	11	10
00	1	0	0	1
01	0	1	0	0
11	0	1	0	0
10	1	0	0	1

WX \ YZ	00	01	11	10
00	w'x'y'z'	w'x'y'z	w'x'yz	w'x'yz'
01	w'xy'z'	w'xy'z	w'xyz	w'xyz'
11	wxy'z'	wxy'z	wxyz	wxyz'
10	wx'y'z'	wx'y'z	wx'yz	wx'yz'

مثال:پیاده سازی ضرب کننده 16*16

- دو ورودی 16 بیتی و یک خروجی 32 بیتی در مجموع 64 ورودی و خروجی
- نیاز به 16*16 گیت AND برای انجام ضرب بیت به بیت
- نیاز به تعداد زیادی مدار Full Adder برای جمع حاصلضرب های بیتی

- در چنین طرحی، طراحی بدون استفاده از ابزار CAD بسیار پیچیده و زمانبر خواهد بود و خطای زیادی خواهد داشت.

کد VHDL

- Entity MULT is

  port (A,B: in std _ logic(15 down to 0);

  Y:out std_logic (31 down to 0);

end MULT;

architecture Behav of MULT is

begin

Y <= A* B;

End Behav ;

مزایای استفاده از زبان HDL

- سرعت بالای طراحی
- خطای کم
- قابل دنبال کردن و خطایابی برنامه نوشته شده با شبیه سازی
- قابلیت استفاده از سخت افزار طراحی شده به صورت یک بسته در کنار سایر سخت افزارها.

- با استفاده از مثال ذکر شده توانایی HDL در طراحی سخت افزار تا حد زیادی قابل درک خواهد بود.

انواع HDL

- دو زبان استاندارد توصیف سخت افزاری پشتیبانی شده توسط IEEE وجود دارد:
- VHDL:

V : Very High Speed Integrated Circuit

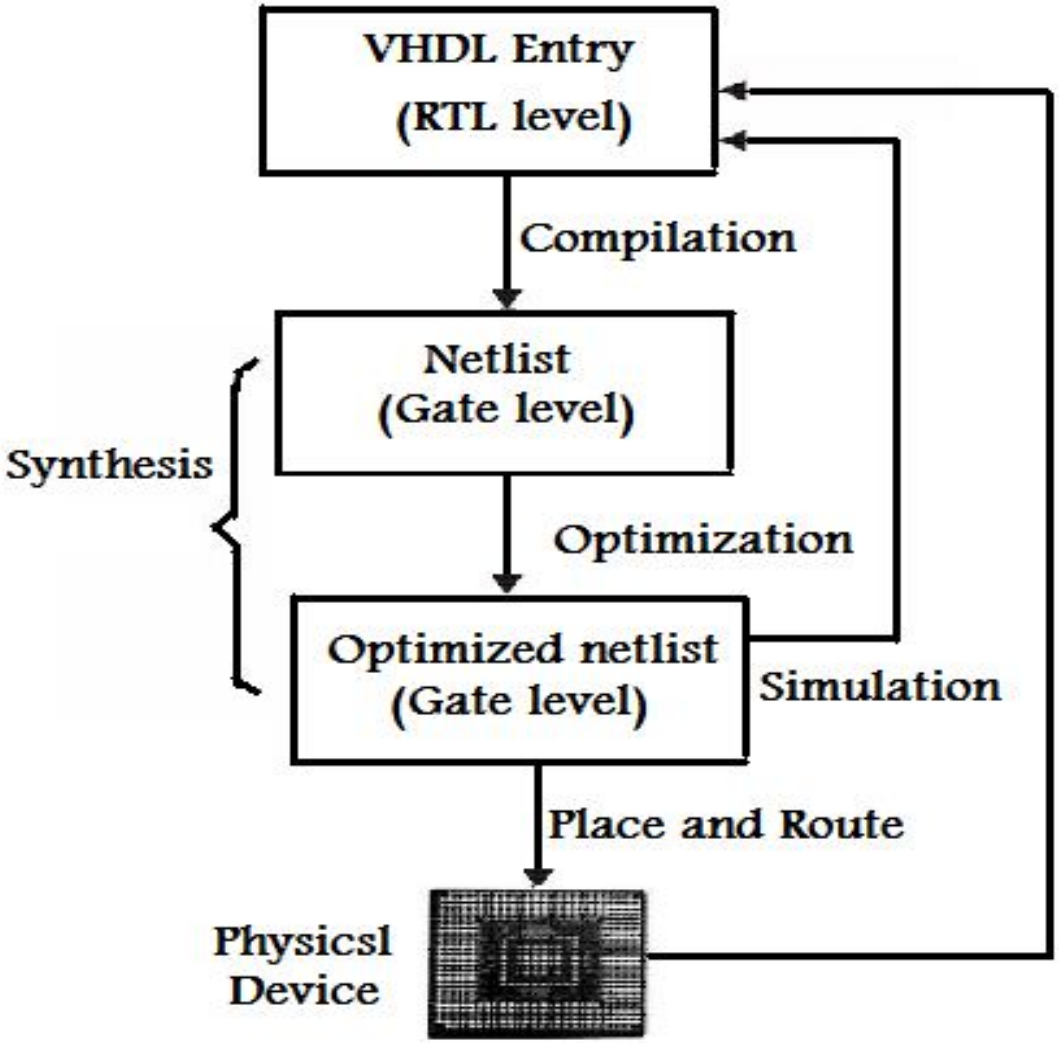
H : Hardware

D : Description

L: Language

- Verilog

مراحل طراحی با HDL



عبارات استفاده شده در طراحی با HDL

- Synthesis : تبدیل کد نوشته شده به مدار

- Simulation:

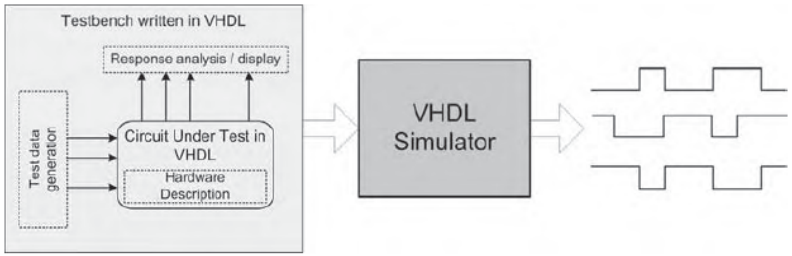
  اعمال یک سری ورودی به طرح و تست عملکرد آن

- Testbench:

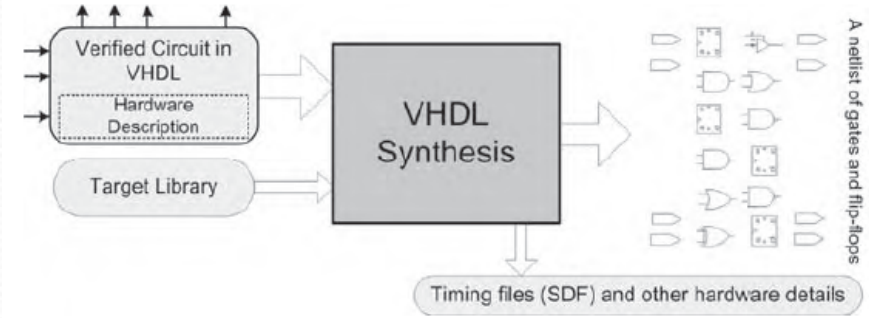
  فراهم کردن محیطی برای اعمال ورودی و تست عملکرد سیستم

مراحل مختلف طراحی با جزئیات بیشتر

- 1.شبیه سازی طرح قبل از سنتز



- 2. سنتز طرح



- 3. شبیه سازی بعد از سنتز

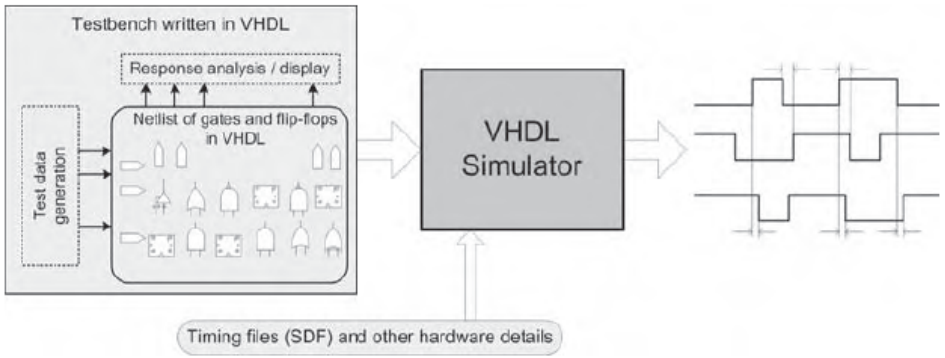


Figure 2.3 Post-synthesis Simulation in VHDL

ASICs یا FPGAs

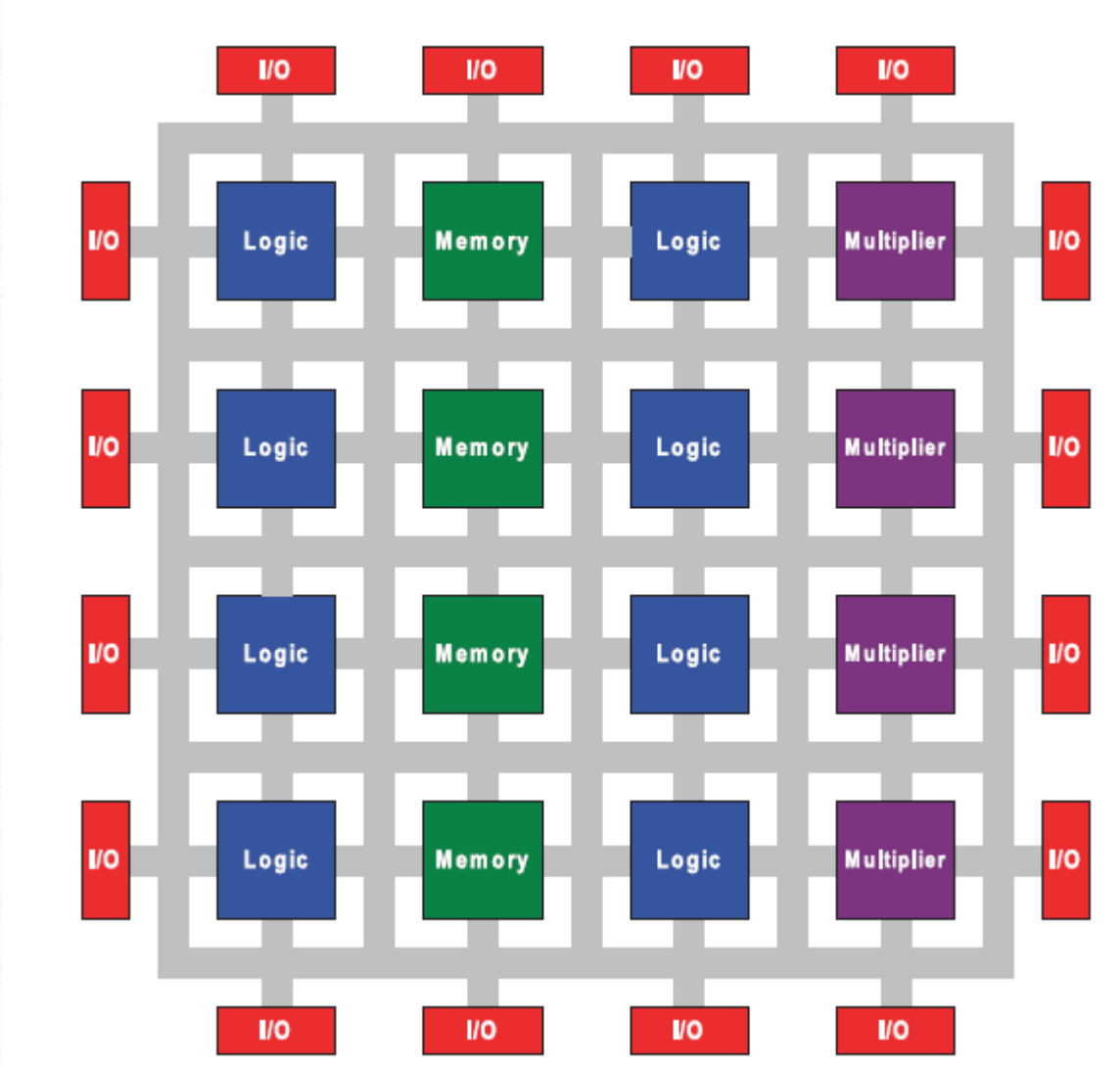
- ASIC (Application Specific Integrated Circuit)

  مدارهای مجتمع خاص منظوره

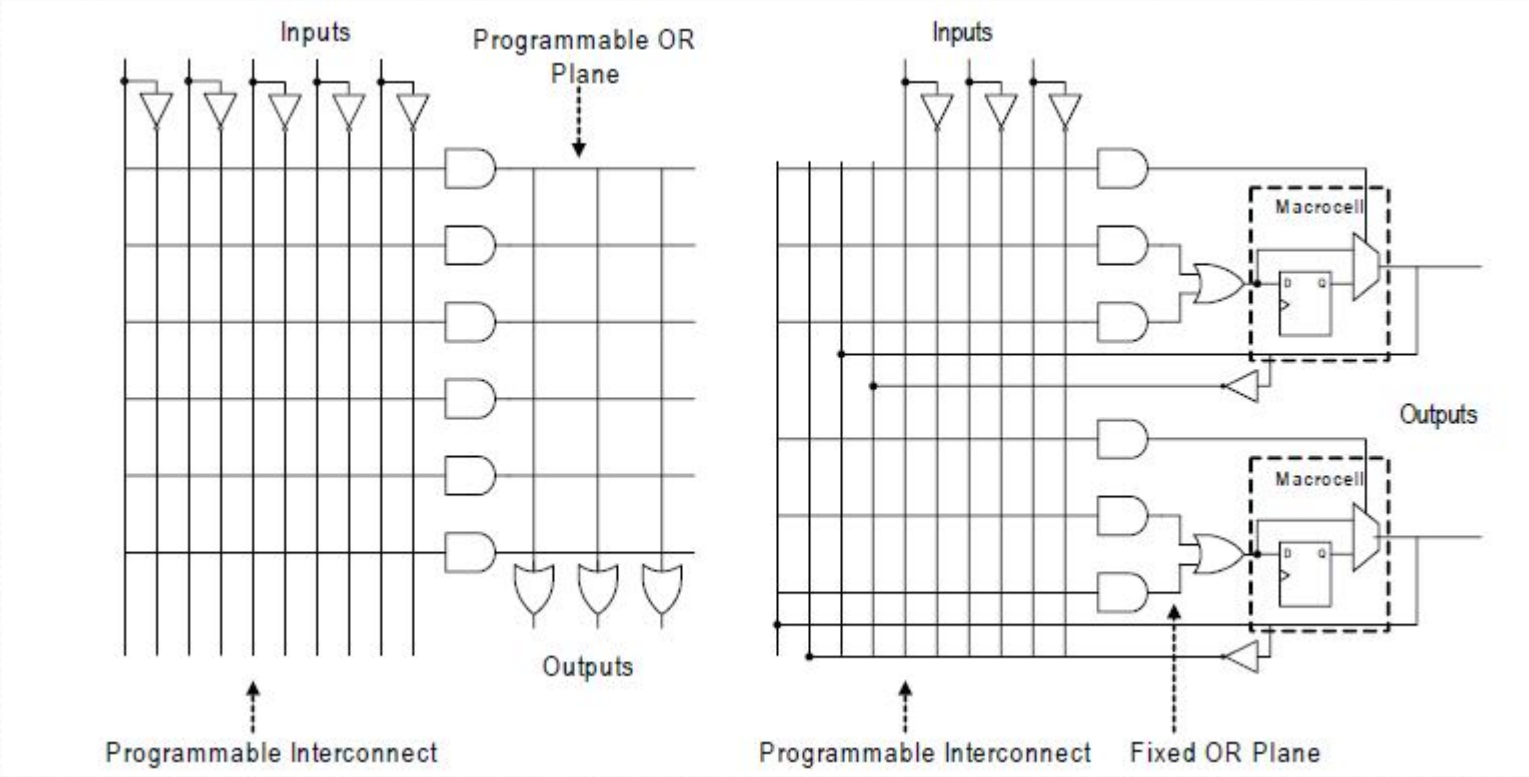
- FPGA (Field Programmable Gate Array)
- CPLD (Complex programmable logic devices)

- VHDL یک زبان استاندارد برای پیاده سازی سخت افزاری روی ساختارهای مختلف ASIC یا FPGA است.

ساختار FPGAها



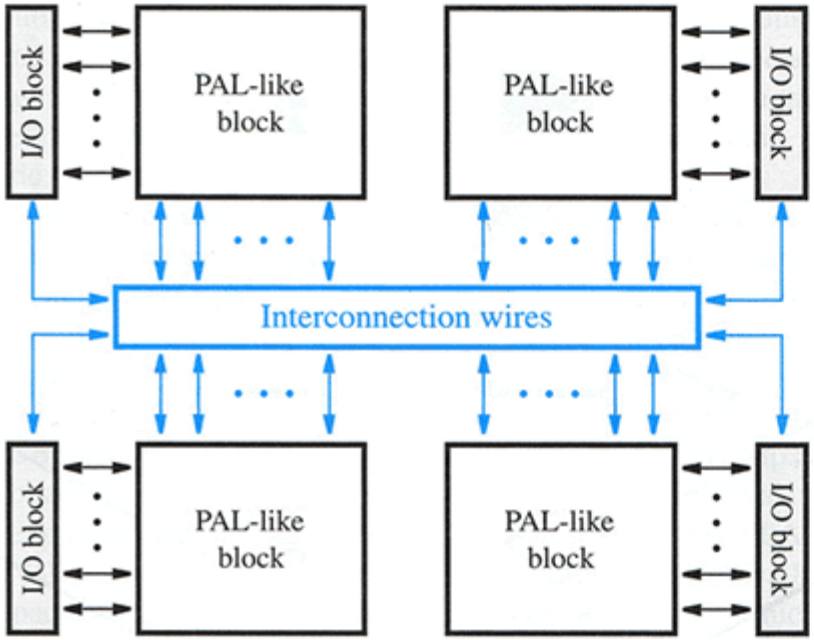
PLA و PAL (سخت افزارهای قابل برنامه ریزی مجدد)



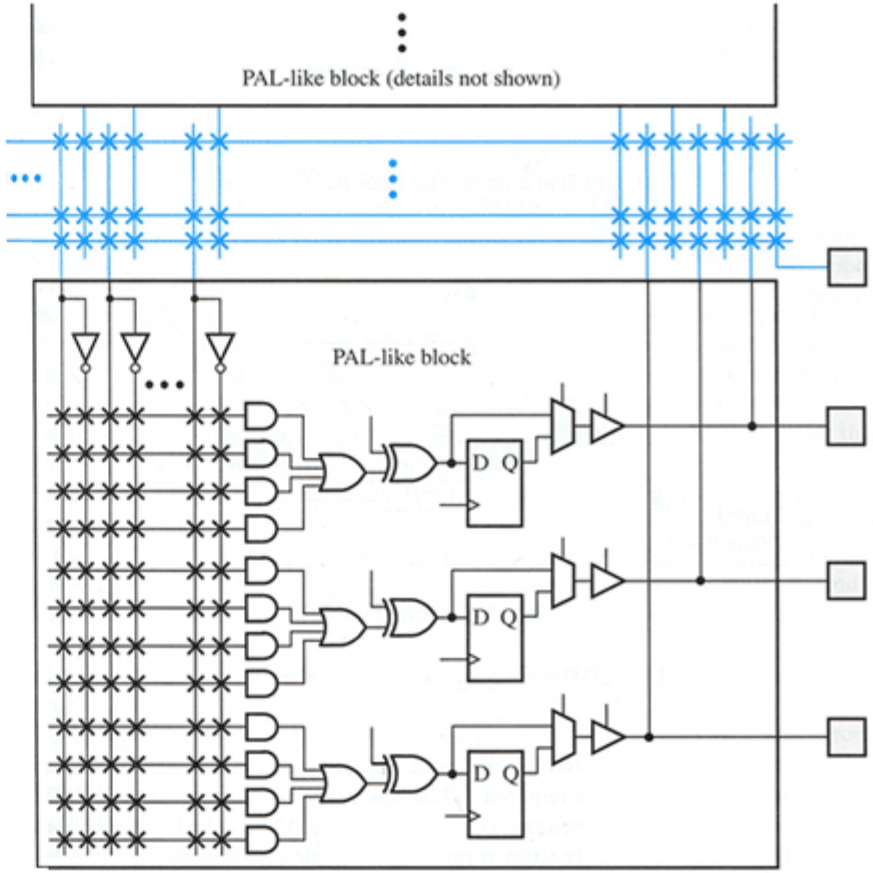
- PLA

PAL(programmable array logic)

ساختار CPLD ها



Structure of a complex programmable logic device (CPLD)



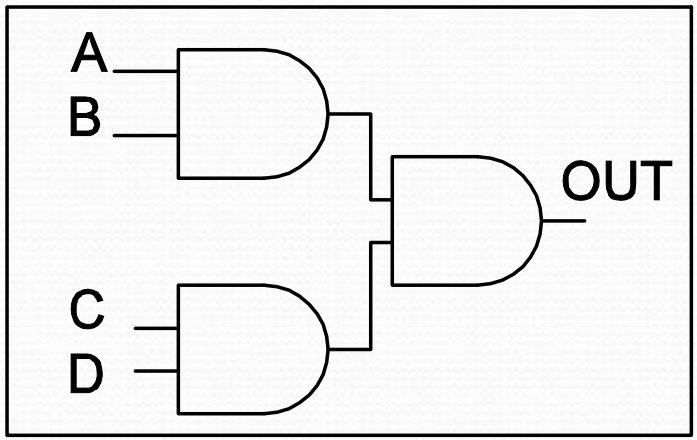
A section of CPLD

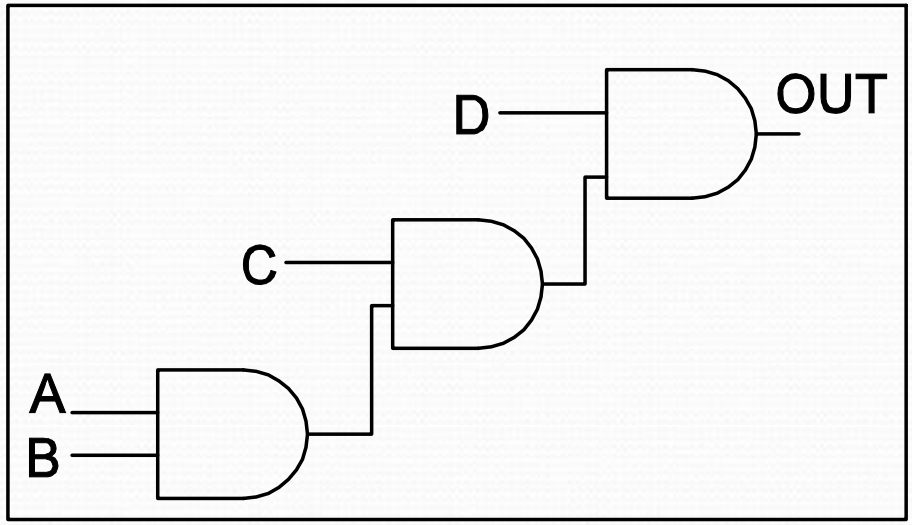
معیارهای ارزیابی طرح

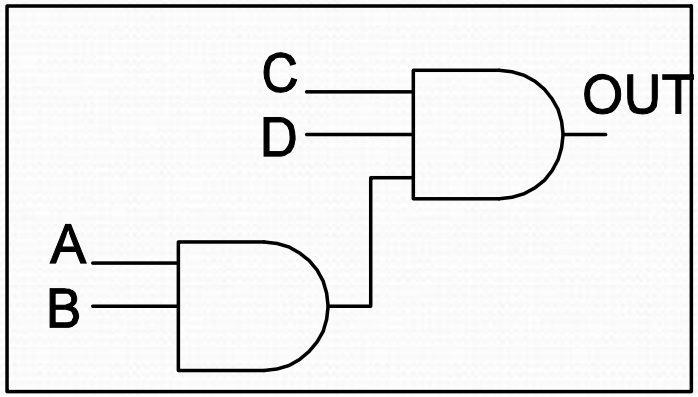
- 1. توان مصرفی
- 2. سطح اشغالی
- 3. تاخیر و سرعت
- 4. قابلیت تست
- 5. قیمت

مثال :پیاده سازی F=A.B.C.D

- طرح بهینه کدام است؟

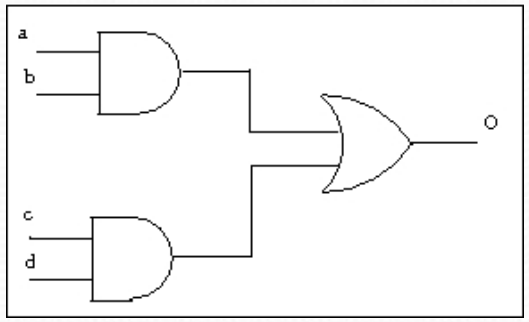


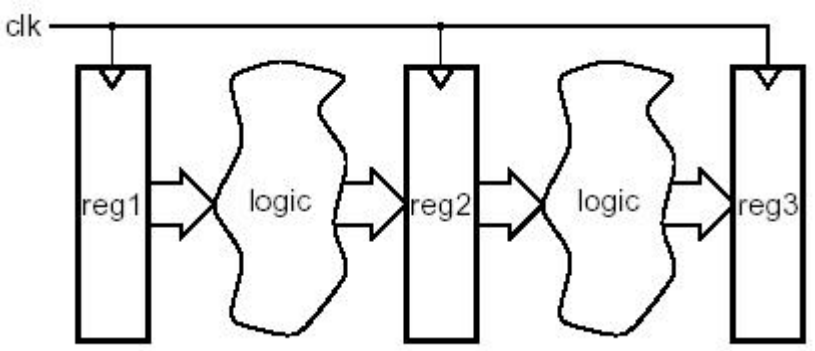




مدار ترتیبی و ترکیبی

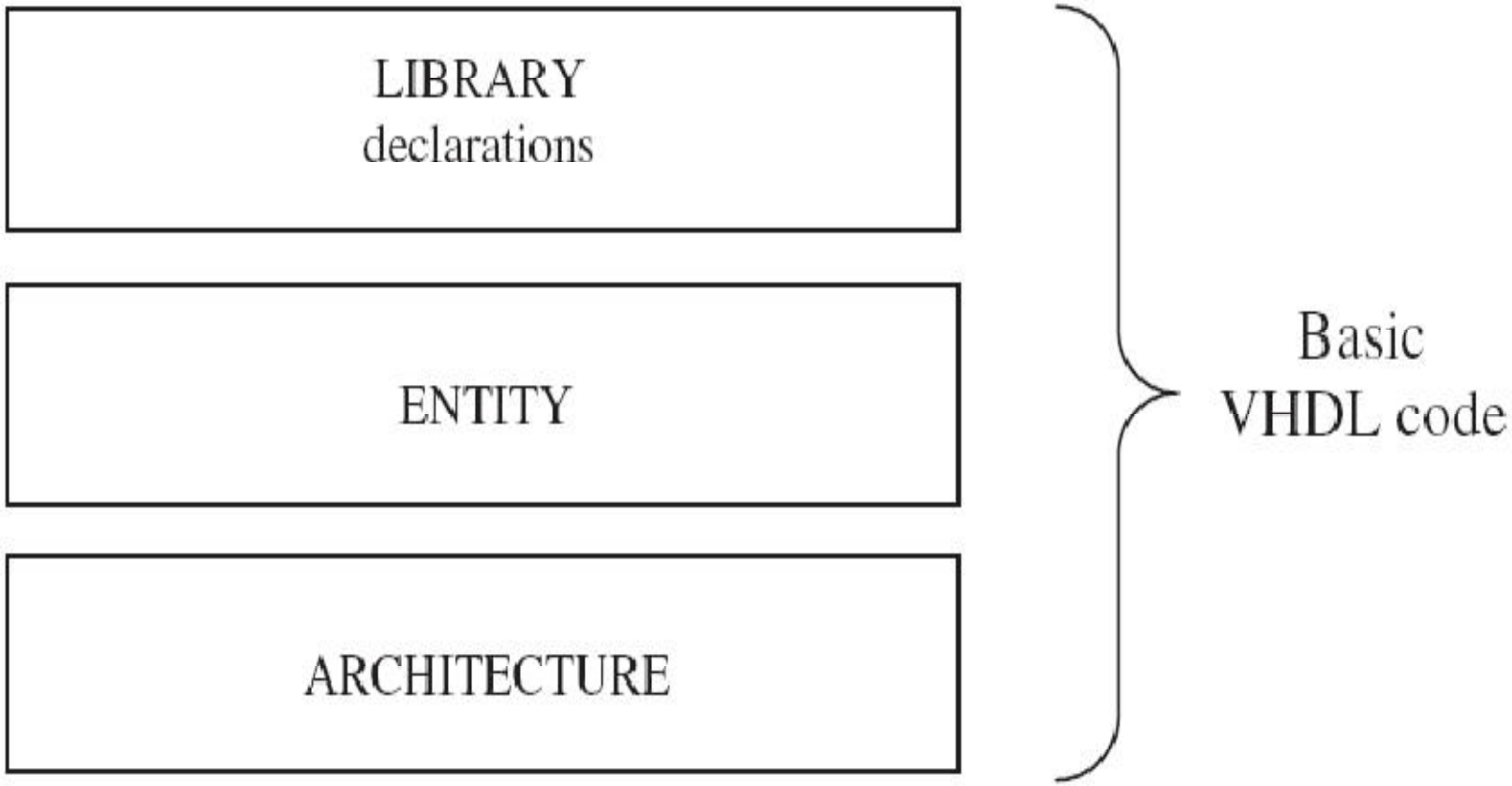
- مدار ترکیبی فاقد حافظه برای ذخیره سازی اطلاعات (مربوط به ورودی های) قبلی است.
- مدار ترتیبی دارای حافظه است.





برنامه نویسی به زبان VHDL

قسمت های اصلی کد VHDL



هر کد VHDL سه قسمت اصلی دارد:

- LIBRARY declarations:

  شامل کتابخانه هایی که در طرح استفاده می شوند.

- ENTITY:

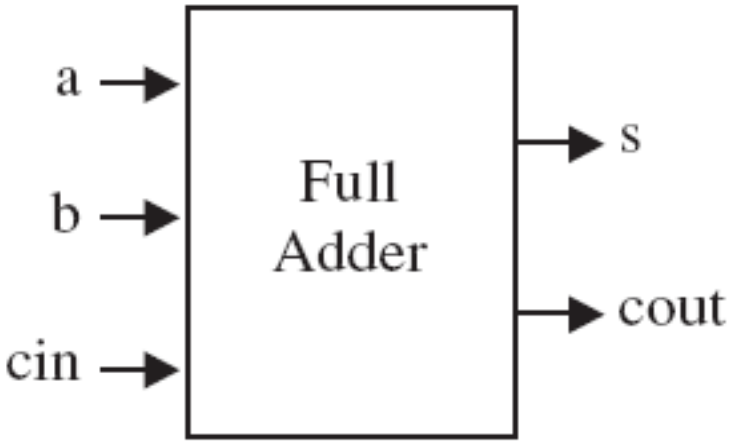
  بیان اتصالات خارجی طرح (مشخص کننده پین های ورودی-خروجی طرح).

- ARCHITECTURE:

  شامل یک کد VHDL که توصیفی از نحوه رفتار مدار می کند.

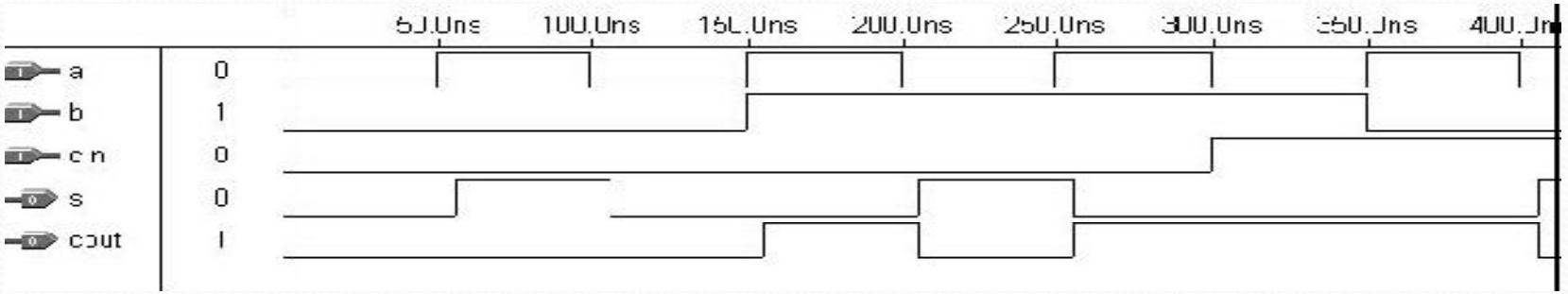
مثال از کد VHDL

شماتیک Full-adder



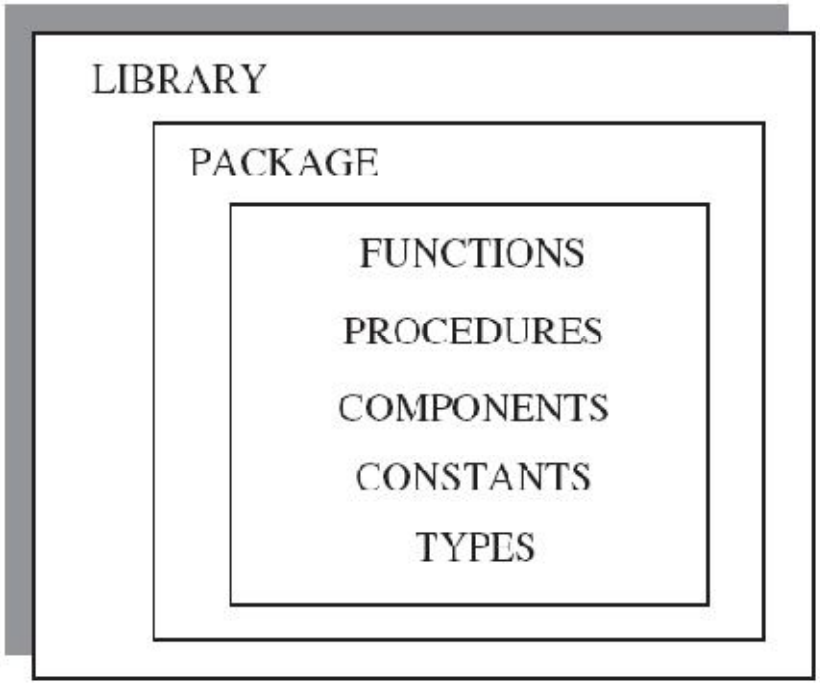
کد VHDL مربوط به مدار Full-adder

```
ENTITY full_adder IS
PORT (a, b, cin: IN BIT;
      s, cout: OUT BIT);
END full_adder;
---------------------------------------
ARCHITECTURE dataflow OF full_adder IS
BEGIN
   s <= a XOR b XOR cin;
   cout <= (a AND b) OR (a AND cin) OR
           (b AND cin);
END dataflow;
```



نتایج شبیه سازی کد VHDL مربوط به طرح

LIBRARY



- کتابخانه مجموعه ای از کدهایی است که معمولا در طرح استفاده می شوند.
- کدهای قرار داده شده در کتابخانه را می توان در طرح های مختلف استفاده کرد بدون آنکه نیاز به باز نویسی وجود داشته باشد.

- نحوه تعریف کتابخانه:

- LIBRARY library_name ;
- USE library_name.package_name.package_parts ;

ادامه

- معمولا حداقل سه بسته (package) از 3 کتابخانه متفاوت در یک طرح نیاز است.

- ieee.std_logic_1164(from the ieee library),
- standard (from the std library) , and
- work (work library)

ادامه

- بسته std_logic_1164 از کتابخانه ieee سیستم منطقی چند سطحی را تعریف می کند. این استاندارد یک نوع داده با 9 مقدار را مشخص می کند.

- کتابخانه workمکانی است که طرح مورد نظر و کل فایل هایی که توسط کامپایلر و سیمولاتور ساخته می شود در آنجا ذخیره می شود.

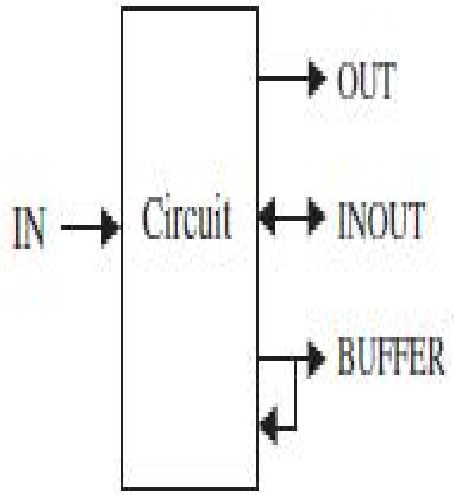
STD_LOGIC type

Value	Meaning
'U'	Uninitialized
'X'	Forcing (Strong driven) Unknown
'0'	Forcing (Strong driven) 0
'1'	Forcing (Strong driven) 1
'Z'	High Impedance
'W'	Weak (Weakly driven) Unknown
'L'	Weak (Weakly driven) 0. Models a pull down.
'H'	Weak (Weakly driven) 1. Models a pull up.
'-'	Don't Care

ENTITY

- ENTITY برای تعریف پین های (PORTS) ورودی- خروجی طرح مورد استفاده قرار می گیرد.
- نحوه تعریف ENTITY به صورت زیر است:

- ENTITY entity_name IS
  PORT (
  port_ name : signal_mode signal_type ;
  port_ name : signal_mode signal_type ;
- ...);
- END entity_name ;



ادامه

- مد سیگنال می تواند IN،OUT ، INOUT و یا BUFFER باشد.
- IN و OUT پین های یک جهته به ترتیب ورودی و خروجی هستند، در حالی که INOUT دو جهته ورودی-خروجی است.
- BUFFER وقتی استفاده می شود که سیگنال خروجی باید در داخل طرح استفاده شود(خوانده شود).
- Entity هر نامی به غیر از کلمات رزرو شده VHDL را می تواند داشته باشد.

ARCHITECTURE

- ARCHITECTURE توصیفی از عملکرد سیستم دارد و نحوه تعریف آن به صورت زیر است.

- ARCHITECTURE architecture_name OF entity_name IS
  [declarations]
  BEGIN
  (code)
  END architecture_name ;

هر ARCHITECTURE دو قسمت دارد،

1. قسمت declarations: توصیفی از سیگنال ها و ثابت های طرح می شود.
2. قسمت کد اصلی : از قسمت BEGIN تا END که کدهای اصلی برنامه در اینجا نوشته می شوند.

مدل سازی رفتاری یا ساختاری

- مدل رفتاری (Behavioral) بیان کننده رفتار سیستم است و رابطه بین ورودی و خروجی های سیستم را نشان می دهد، که می تواند یا به صورت الگوریتمی(یک تعدادی دستورالعمل) باشد یا یک عبارت جبر دو مقداره یا در سطح رجیستر (Register transfer level (RTL))

- مدل ساختاری (Structural) نوعا نمایش گرافیکی سیستم دیجیتال است بنابراین به نمایش فیزیکی سیستم واقعی نزدیکتر است. در این سطح از طراحی عملکرد مدار توسط ارتباط بین گیتهای مختلف و زیر بلوک های ساختار توصیف می شود.

ادامه

- طراحی در سطح الگوریتم:

در این مدل یک تعدادی از مجموعه دستورالعمل ها عملکرد سیستم را توصیف می کنند. این سطح از طراحی عموما برای مدارهای ترتیبی مورد استفاده قرار می گیرد.

- طراحی در سطح رجیستر RTL :

در این نوع مدلسازی عموما جریان داده (Data flow) در سیستم، (معمولا جریان داده بین رجیسترهای مختلف) نشان داده می شود. RTL معمولا برای طراحی سیستم های ترکیبی استفاده می شود

همزمان یا ترتیبی

- دستورات کد VHDL به صورت همزمان یا موازی اجرا می شوند. تنها جملاتی که در PROCESS، FUNCTION ویا PROCEDURE هستند به صورت ترتیبی اجرا می شوند.
- هرچند روند اجرای دستورات در این بلوک ها ترتیبی است ولی کل بلوک در کنار سایر جملات خارج از آن بلوک به صورت همزمان اجرا می شوند.
- کدهای همزمان را data flow نیز گویند.

کدهای همزمان

- در کدهای همزمان قسمتهای زیر قابل استفاده هستند:

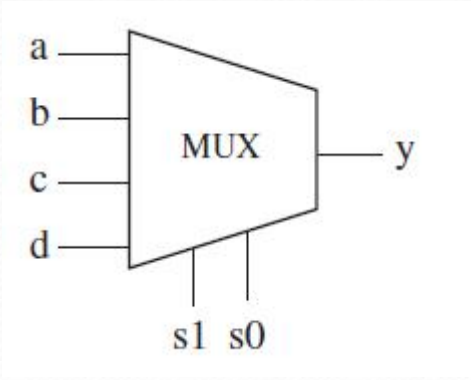
- Operators
- The WHEN statement (WHEN/ELSE or WITH/SELECT/WHEN)
- The GENERATE statement
- The BLOCK statement.

Operators

Operator type	Operators	Data types
Logical	NOT, AND, NAND, OR, NOR, XOR, XNOR	BIT, BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR, STD_ULOGIC, STD_ULOGIC_VECTOR
Arithmetic	+, –, *, /, ** (mod, rem, abs)	INTEGER, SIGNED, UNSIGNED
Comparison	=, /=, <, >, <=, >=	All above
Shift	sll, srl, sla, sra, rol, ror	BIT_VECTOR
Concatenation	&, (,,,)	Same as for logical operators, plus SIGNED and UNSIGNED

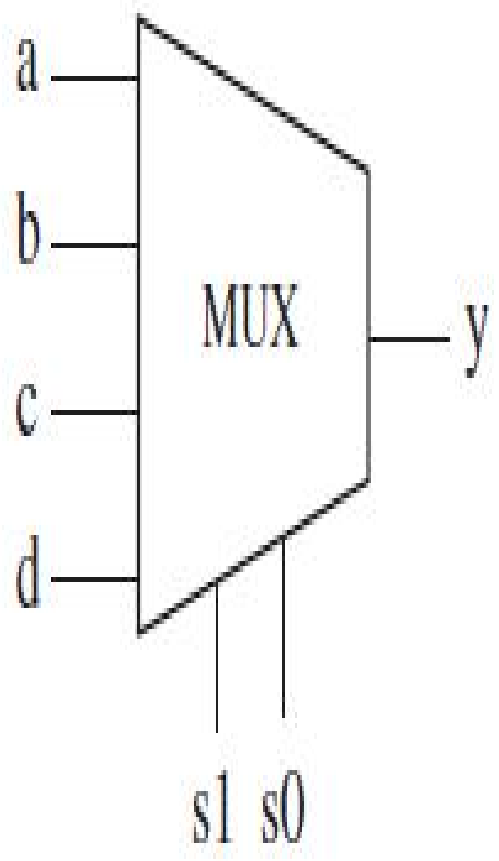
مثال : مالتی پلکسر

```
LIBRARY ieee;
 USE ieee.std_logic_1164.all;
 ENTITY mux IS
    PORT ( a, b, c, d, s0, s1: IN STD_LOGIC;
    y : OUT STD_LOGIC);
 END mux;
 ARCHITECTURE pure_logic OF mux IS
 BEGIN
    y <= (a AND NOT s1 AND NOT s0) OR
    (b AND NOT s1 AND s0) OR
    (c AND s1 AND NOT s0) OR
    (d AND s1 AND s0);
 END pure_logic;
```



با استفاده از دستور WHEN

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
 ENTITY mux IS
        PORT ( a, b, c, d: IN STD_LOGIC;
        sel: IN STD_LOGIC_VECTOR (1
        DOWNTO 0);
 y: OUT STD_LOGIC);
 END mux;
ARCHITECTURE mux1 OF mux IS
 BEGIN
        y <= a WHEN sel = "00" ELSE
        b  WHEN  sel= "01" ELSE
        c WHEN sel="10" ELSE   d ;
END mux1;
```



مثال بافر 3 حالته

```
LIBRARY ieee;
 USE ieee.std_logic_1164.all;
 ENTITY tri_state IS
    PORT ( ena: IN STD_LOGIC;
     input: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    output: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
 END tri_state;
 ARCHITECTURE tri_state  OF  tri_state IS
 BEGIN
    output <= input WHEN (ena='0')  ELSE
    (OTHERS => 'Z');
 END tri_state;
```

کدهای ترتیبی

- PROCESSES,FUNCTIONS و PROCEDURES کدها به صورت ترتیبی اجرا می شوند.

- با استفاده از کدهای ترتیبی می توان هم مدار ترتیبی هم ترکیبی ساخت.

جملات ترتیبی

- Variable Assignment Statement
- **PROCESS** Statement (*Concurrent and Sequential*)
- **IF** Statement
- **CASE** Statement
- **LOOP** Statement
- **NEXT** Statement
- **EXIT** Statement
- **WAIT** Statement
- *Subprograms*
  - **Function***s*
  - **Procedure***s*
- **ASSERT** Statement

جمع کننده ها

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
ENTITY adder8 IS PORT (
        a : in std_logic_vector (7 DOWNTO 0);
        b : in std_logic_vector (7 DOWNTO 0);
        ci : in std_logic;
        s : out std_logic_vector (7 DOWNTO 0);
        co : out std_logic );
END ENTITY adder8;
--
ARCHITECTURE equation OF adder8 IS
SIGNAL mid : std_logic_vector (8 DOWNTO 0);
BEGIN
        mid <= ('0'&a) + ('0'&b) + ci;
        co <= mid (8);
        s <= mid (7 DOWNTO 0);
END equation;
```

مثال ALU

```
ENTITY alu8 IS PORT (
    a, b : in std_logic_vector (7 DOWNTO 0);
    addsub : in std_logic;
    gt, zero, co : out std_logic;
    r : out std_logic_vector (7 DOWNTO 0));
END ENTITY alu8;
ARCHITECTURE assigns OF alu8 IS
SIGNAL mid : std_logic_vector (8 DOWNTO 0);
BEGIN
    mid <= ('0'& a) + ('0'& b) WHEN addsub = '1' ELSE
    ('0'& a) - ('0'& b);
    co <= mid (8);
    r <= mid (7 DOWNTO 0);
    gt <= '1' WHEN a > b ELSE '0';
    zero <='1' WHEN mid (7 DOWNTO 0) = "00000000" ELSE '0';
END assigns;
```

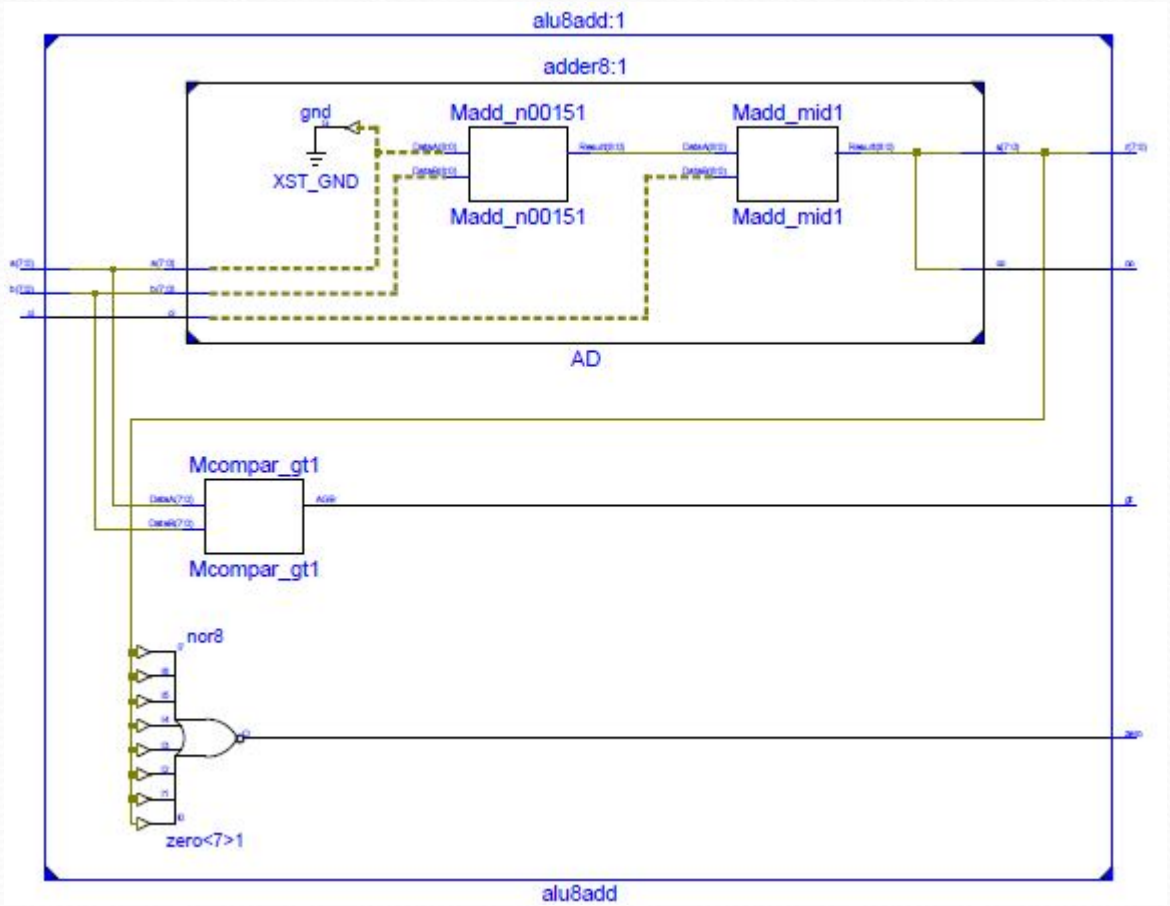
مثال ALU 2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
ENTITY adder8 IS PORT (
a : in std_logic_vector (7 DOWNTO 0);
b : in std_logic_vector (7 DOWNTO 0);
ci : in std_logic;
s : out std_logic_vector (7 DOWNTO 0);
co : out std_logic );
END ENTITY adder8;
--
ARCHITECTURE equation OF adder8 IS
SIGNAL mid : std_logic_vector (8 DOWNTO 0);
BEGIN
mid <= ('0'&a) + ('0'&b) + ci;
co <= mid (8);
s <= mid (7 DOWNTO 0);
END equation;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY alu8add IS PORT (
a, b : in std_logic_vector (7 DOWNTO 0);
ci: in std_logic;
gt, zero, co : out std_logic;
r : out std_logic_vector (7 DOWNTO 0));
END ENTITY alu8add;
ARCHITECTURE assigns OF alu8add IS
SIGNAL mid8 : std_logic_vector (7 DOWNTO 0);
SIGNAL mid1 : std_logic;
BEGIN
AD: ENTITY WORK.adder8 PORT MAP (a, b, ci, mid8, co);
-- AD: ENTITY WORK.adder8 PORT MAP
-- (a => a, b => b, ci => '0', s => mid8);
r <= mid8;
gt <= '1' WHEN a > b ELSE '0';
zero <= '1' WHEN mid8 = "00000000" ELSE '0';
END assigns;
```

نتیجه سنتز کد ALU



Signalsو variables

- *Variable ها برای تعریف متغیرهای موجود در قسمت های ترتیبی کد برنامه استفاده می شوند.*
- *در همان پروسه ای که تعریف شده اند فقط قابل دسترسی هستند.*
- *از علامت ":=" برای تعریف آن استفاده می شود*

- Signal برای تعریف اتصالات بین قسمت های مختلف سخت افزار مورد استفاده قرار می گیرد و خصوصیات اتصالات در یک اجزای سخت افزار واقعی را دارد.
- نمی توان مقادیر متفاوتی به یک سیگنال اختصاص دهیم مگر اینکه نوع داده آن resolved باشد
- از علامت "<=" برای اختصاص به یک سیگنال استفاده می شود.

مثال از تفاوت Signal و Variable

```
ARCHITECTURE mixed_processes_assignments OF aCircuit IS
  SIGNAL d: BIT;
  SIGNAL dv : BIT_VECTOR (7 DOWNTO 0);
BEGIN
  p1: PROCESS (a, b)
    VARIABLE e : BIT;
    VARIABLE ev : BIT_VECTOR (7 DOWNTO 0);
  BEGIN
    IF (a = b) THEN ev := av; ELSE ev := bv;
    IF (a = '1') THEN wv <= av; ELSE wv <= "1000111";
    d <= e;
  END PROCESS;
  dv <= av XOR bv;
  w <= d AND a;
END ARCHITECTURE mixed_processes_assignments;
```

استفاده از اندیس دهی برای سیگنال ها

```
ARCHITECTURE indexing_slicing OF aCircuit IS
SIGNAL d : BIT;
SIGNAL dv : BIT_VECTOR (7 DOWNTO 0);
BEGIN
    wv (3 DOWNTO 0) <= av (7 DOWNTO 4) AND cv (7
      DOWNTO 4);
    w <= cv (4);
    cv (7) <= av (0);
END ARCHITECTURE indexing_slicing;
```

Multiple Assignments to a Resolved Signal

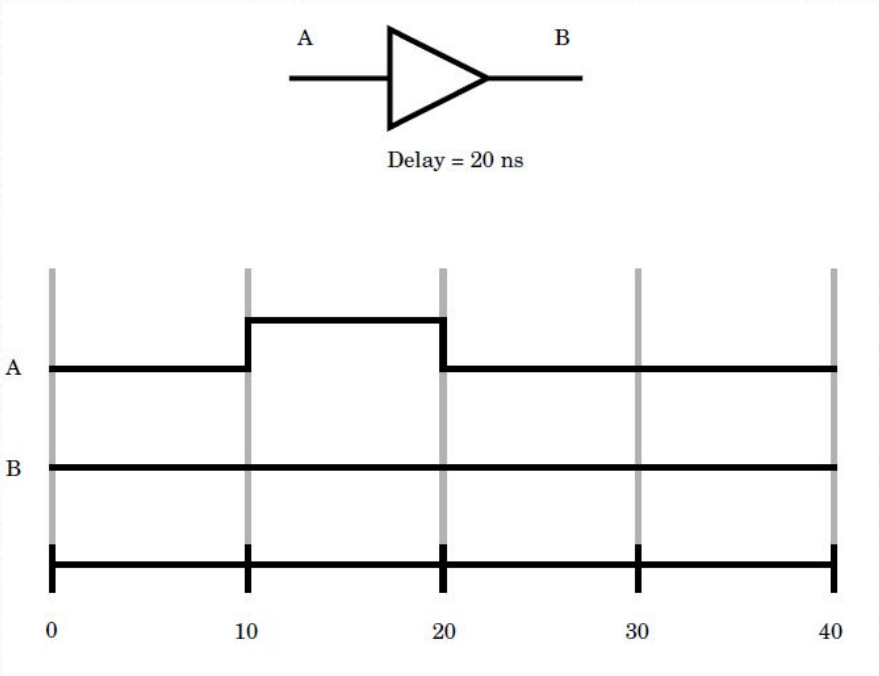
```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTIRY selector IS
    PORT (av, bv: IN std_logic_vector (7 DOWNTO 0),
    as, bs: IN std_logic;
    yv: OUT std_logic_vector (7 DOWNTO 0));
END ENTITY selector;
ARCHITECTURE multiple_drivers OF selector IS
BEGIN
    yv <= av WHEN as = '1' ELSE "ZZZZZZZZ";
    yv <= bv WHEN bs = '1' ELSE "ZZZZZZZZ";
END ARCHITECTURE multiple_drivers;
```

تاخیر خالص یا تاخیر انتقالی

- **b <= a AFTER 20 ns;** تاخیر خالص یا با اینرسی
  - تاخیر خالص برای مدلسازی تاخیر واقعی در قطعات الکترونیکی به کار می رود.
  - این تاخیر به عنوان پیش فرض اولیه در کلیه سیمولاتورها در نظر گرفته می شود.
  - در این تاخیر پالس های با پهنای کوچکتر از تاخیر گیت دیجیتال حذف می شوند.
- **b <= TRANSPORT a AFTER 20 ns;** تاخیر انتقالی
  - در این تاخیر هر پالسی با هر پهنایی بعد از تاخیر مربوط به گیت مورد نظر به خروجی منتقل می شود.

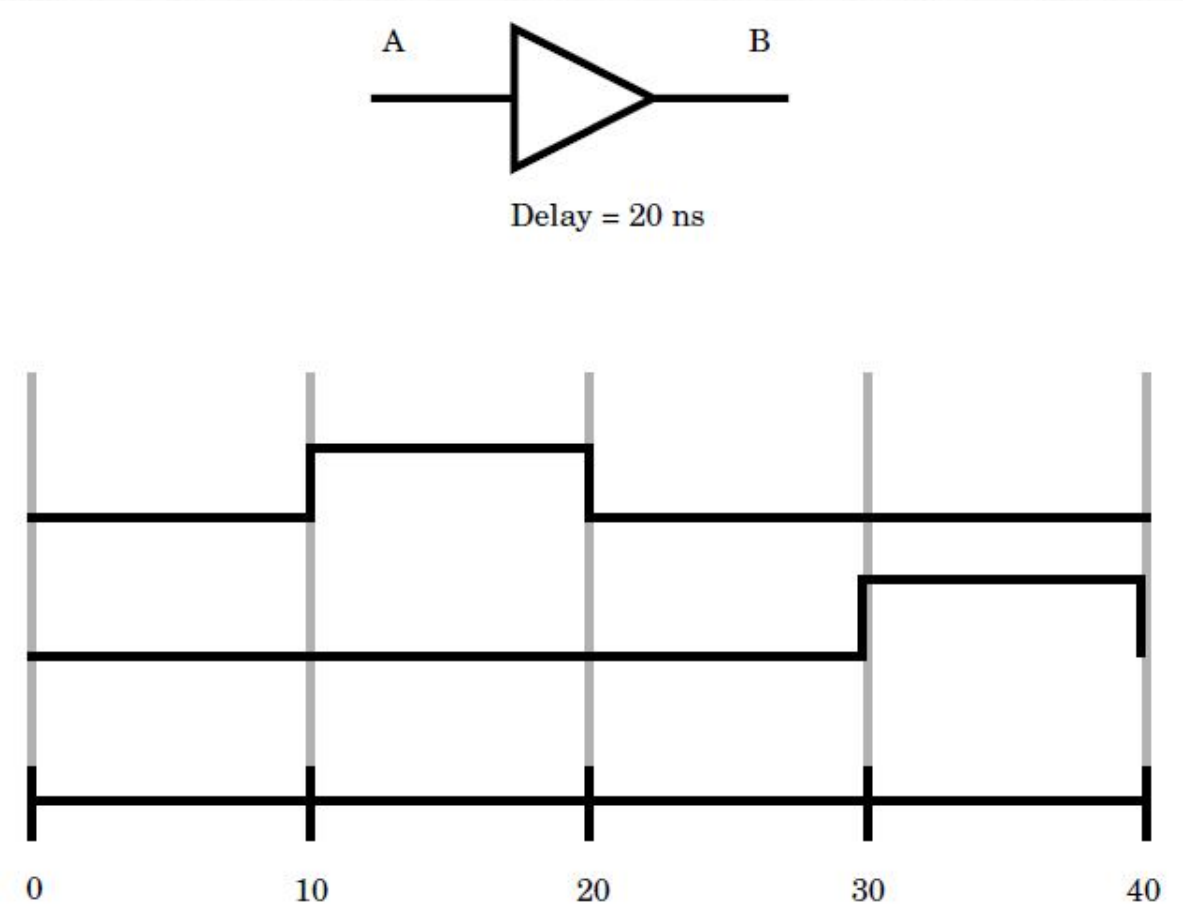
مثال از تاخیر خالص

- پالس با پهنای 10ns چون کمتر از تاخیر گیت دیجیتال است حذف شده است.



مثال از تاخیر انتقالی

- پالس مورد نظر بعد از گذشت تاخیر مربوط به گیت دیجیتال به خروجی منتقل شده است.



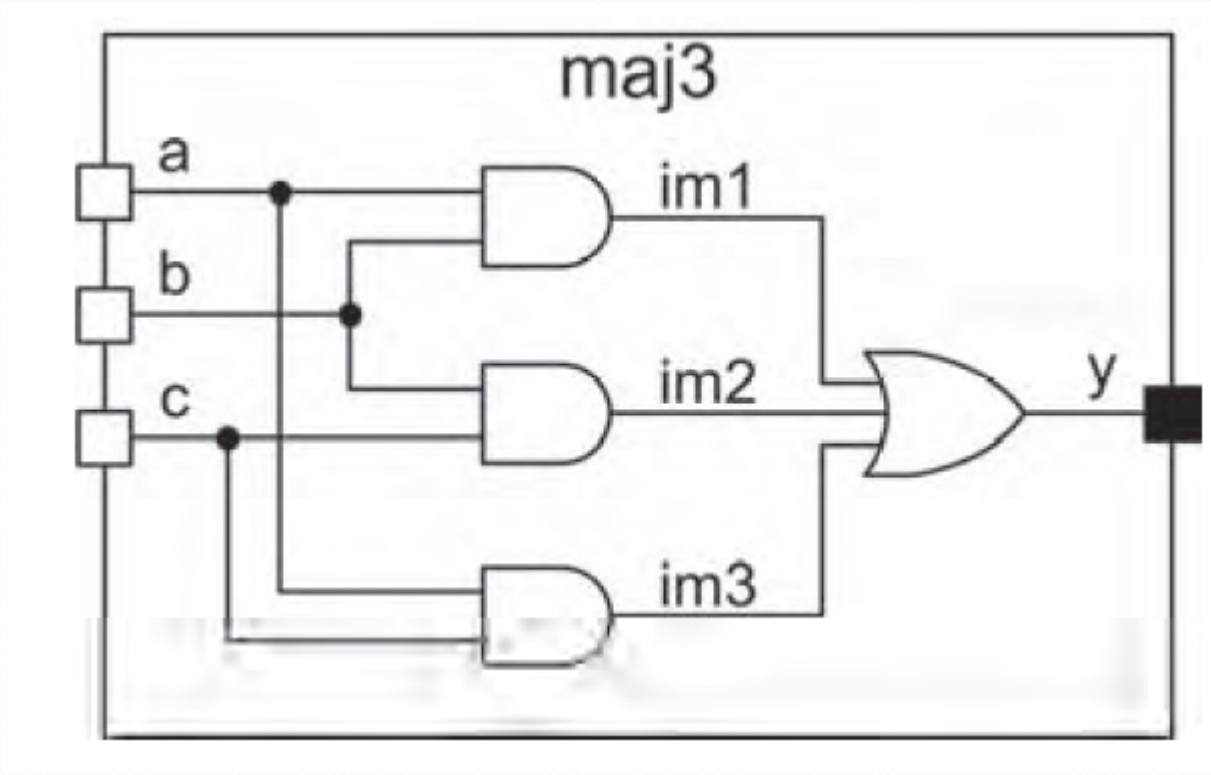
Generic

- ژنریک برای انتقال اطلاعات ثابت به entity استفاده می شود.
- این اطلاعات ثابت بیشتر به منظور مدلسازی تاخیر مربوط به گیت های دیجیتال مورد استفاده قرار می گیرند.
- ژنریک همچنین برای اطلاعات مربوط به مقدار خازن یا مقاومت بار به کار برده می شود.
- و نیز برای اطلاعات مربوط به تعداد بیتهای مسیر داده و سیگنال ها مورد استفاده قرار می گیرد.

مثال از تعریف ژنریک و مقدار دهی آن

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY test IS
    GENERIC(rise, fall : TIME; load : INTEGER);
    PORT ( ina, inb, inc, ind : IN std_logic;
    PORT ( out1, out2 : OUT std_logic);
END test;
ARCHITECTURE test_arch OF test IS
COMPONENT AND2
    GENERIC(rise, fall : TIME; load : INTEGER);
    PORT ( a, b : IN std_logic;
    PORT ( c : OUT std_logic);
END COMPONENT;
BEGIN
    U1: AND2 GENERIC MAP(10 ns, 12 ns, 3 ) PORT MAP (ina, inb, out1 );
    U2: AND2 GENERIC MAP(9 ns, 11 ns, 5 )  PORT MAP (inc, ind, out2 );
END test_arch;
```

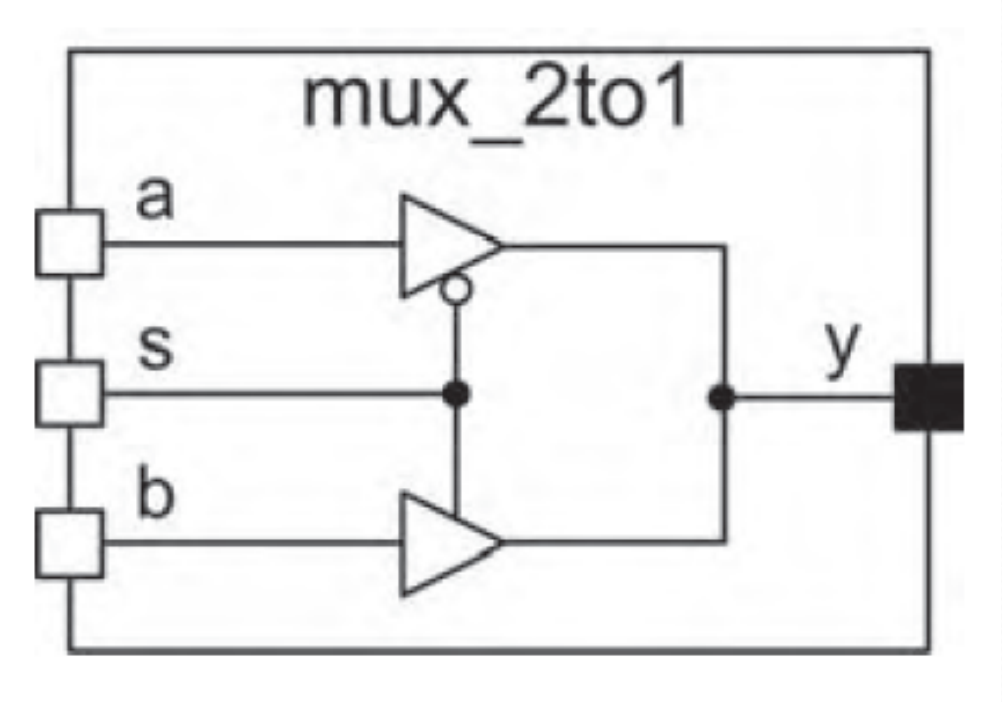
مثال 1 طراحی ساختاری



کد ساختاری مثال 1

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY maj3 IS
    PORT (a, b, c : IN std_logic;
    y : OUT std_logic);
END maj3;
ARCHITECTURE gate_level OF maj3 IS
SIGNAL im1, im2, im3 : std_logic;
BEGIN
    ANDa: ENTITY WORK.AND2 PORT MAP (a, b, im1);
    ANDb: ENTITY WORK.AND2 PORT MAP (b, c, im2);
    ANDc: ENTITY WORK.AND2 PORT MAP (a, c, im3);
    ORa : ENTITY WORK.OR3 PORT MAP (im1, im2, im3, y);
END ARCHITECTURE gate_level;
```

مثال 2 طراحی ساختاری مالتی پلکسر 2 به 1



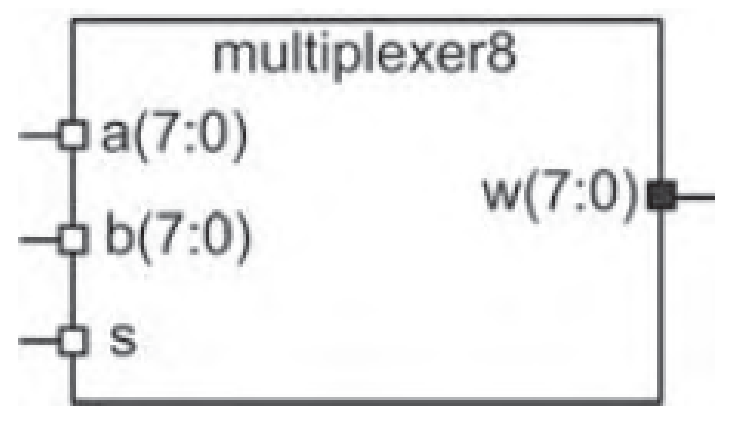
کد ساختاری

```
ENTITY mux_2to1 IS
    PORT (a, b, s: IN std_logic; y: OUT std_logic);
END ENTITY mux_2to1;
ARCHITECTURE gate_level OF mux_2to1 IS BEGIN
    BUFIF1a: ENTITY WORK.BUFIF1(example) PORT MAP
      (b, s, y);
    BUFIF1b: ENTITY WORK.BUFIF0(example) PORT MAP
      (a, s, y);
END ARCHITECTURE gate_level;
```

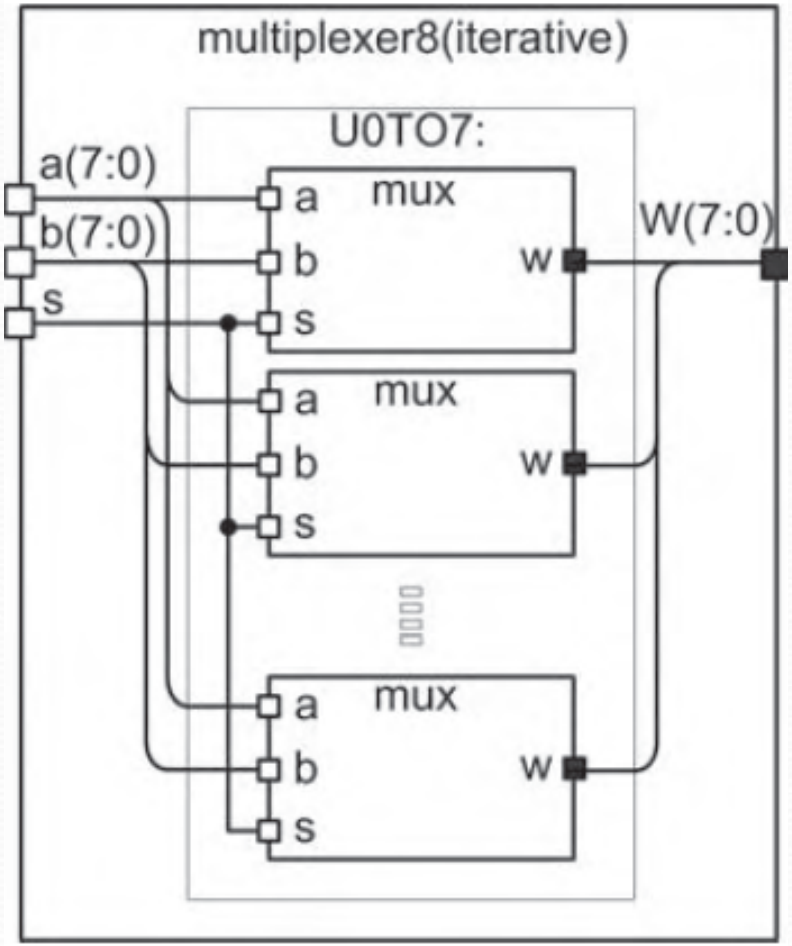
ساختارهای تکرار شونده:

- گاهی یک بلوکی با یک روال مشخص در یک ساختار تکرار می شود، برای کد نویسی برای این بلوکی که بارها با روال مشخصی در طرح تکرار شده است از دستور generate استفاده می شود.

- مثال : مالتی پلکسر 2 به 1 هشت بیتی



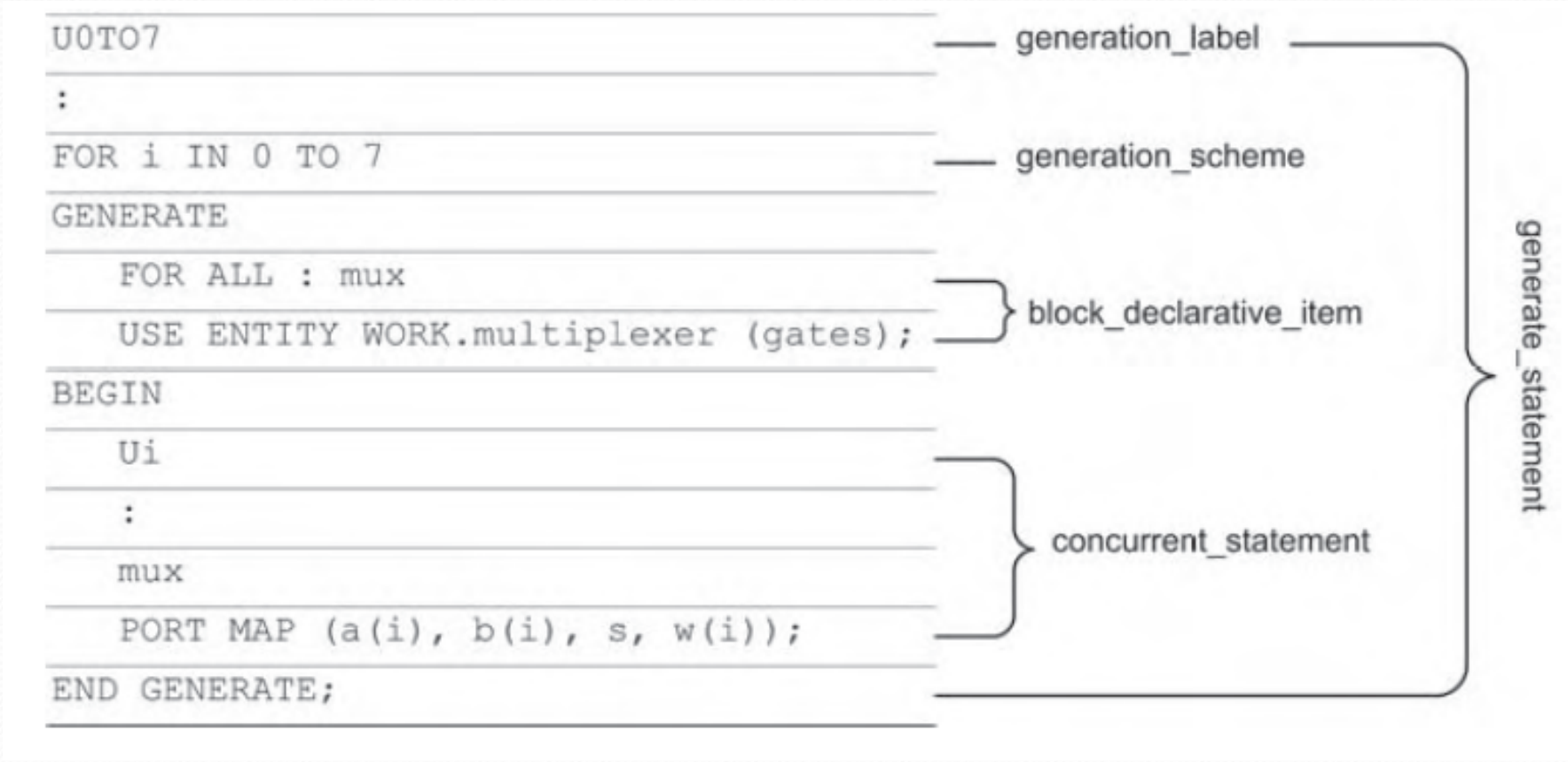
ساختار سلسله مراتبی مالتی پلکسر 2 به 1 هشت بیتی



کد مربوط به مالتی پلکسر 2 به 1 هشت بیتی

```
ENTITY multiplexer8 IS
    PORT (a, b : IN BIT_VECTOR (7 DOWNTO 0); s : IN BIT;
    w : OUT BIT_VECTOR (7 DOWNTO 0) );
END ENTITY;
---
ARCHITECTURE iterative OF multiplexer8 IS
COMPONENT mux
     PORT (a, b, s : IN BIT; w : OUT BIT);
END COMPONENT;
BEGIN
U0TO7: FOR i IN 0 TO 7 GENERATE
FOR ALL : mux USE ENTITY WORK.multiplexer (gates);
BEGIN
    Ui: mux PORT MAP (a(i), b(i), s, w(i));
END GENERATE;
END ARCHITECTURE iterative;
```

توضیحات مربوط به کد نوشته شده



طراحی مدارهای ترتیبی

PROCESS

- PROCESS قسمتی از کد ترتیبی VHDL است. در این کد از دستورات IF، WAIT یا LOOP و یک تعداد لیست حسایت تشکیل شده است. (به غیر از زمانی که از WAIT استفاده می شود)

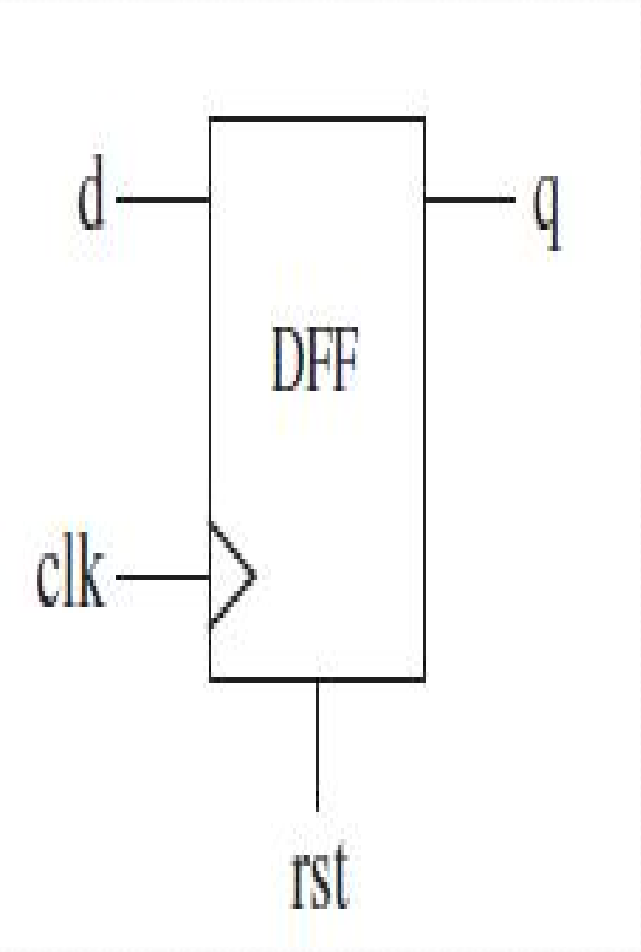
- PROCESS در کد اصلی استفاده می شود و در زمان ایجاد هرگونه تغییر در سیگنال موجود در لیست حساسیت اجرا می شود (یا زمانی که شرط مربوط به دستور WAIT ایجاد شد)

نحوه تعریف PROCESS

- PROCESS (sensitivity list)
- [VARIABLE name type [range] [:= initial_value;]]
- BEGIN
- (sequential code)
- END PROCESS [label];

فلیپ فلاپ D

```
LIBRARY ieee;
 USE ieee.std_logic_1164.all;
 ENTITY dff IS
     PORT (d, clk, rst: IN STD_LOGIC;
     q: OUT STD_LOGIC);
 END dff;
 ARCHITECTURE behavior OF dff  IS
 BEGIN
    PROCESS (clk, rst)
     BEGIN
              IF (rst='1') THEN
                        q <= '0';
              ELSIF (clk‘ EVENT AND clk='1') THEN
                        q <= d;
              END IF;
     END PROCESS;
 END behavior;
```



DFF- ALTERNATIVE

- LIBRARY ieee;
- USE ieee.std_logic_1164.all;
- ENTITY dff IS
- PORT (d, clk, rst: IN STD_LOGIC;
- q: OUT STD_LOGIC);
- END dff;
- ARCHITECTURE dff OF dff IS
-

```
BEGIN
 PROCESS
 BEGIN
 WAIT ON rst, clk;
 IF (rst='1') THEN
 q <= '0';
 ELSIF (clk'EVENT
AND clk='1') THEN
 q <= d;
 END IF;
 END PROCESS;
 END dff;
```

CASE

- CASE یکی دیگر از دستورهایی است که برای کدهای ترتیبی استفاده می شود.

```
CASE identifier IS
   WHEN value => assignments ;
   WHEN value => assignments ;
   ...
END CASE;
```

مثال با دستور CASE

```
CASE control IS
   WHEN "00" => x<=a; y<=b;
   WHEN "01" => x<=b; y<=c;
   WHEN  OTHERS => x<="0000";  y<="ZZZZ“ ;
END CASE;
```

کد D-FF با دستور CASE

- LIBRARY ieee;
- USE ieee.std_logic_1164.all;
- STD_LOGIC
- ENTITY dff IS
- PORT (d, clk, rst: IN BIT;
- q: OUT BIT);
- END dff;
- ARCHITECTURE dff3 OF dff IS
- BEGIN

```
PROCESS (clk, rst)
 BEGIN
 CASE rst IS
WHEN '1' => q<='0';
 WHEN '0' =>
 IF (clk'EVENT AND
clk='1') THEN
 q <= d;
 END IF;
 WHEN OTHERS =>
NULL;
 END CASE;
 END PROCESS;
 END dff3;
```

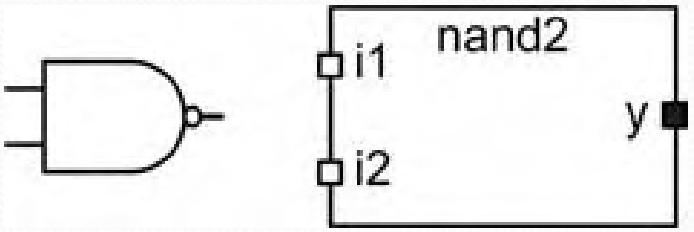
طراحی MUX ترتیبی

```
ENTITY multiplexer IS
    PORT (a, b, s : IN BIT; w : OUT BIT);
END ENTITY;
--
ARCHITECTURE procedural OF multiplexer IS BEGIN
PROCESS (a, b, s) BEGIN
    IF (s = '0') THEN w <= a;
    ELSE w <= b;
    END IF;
END PROCESS;
END ARCHITECTURE procedural;
```

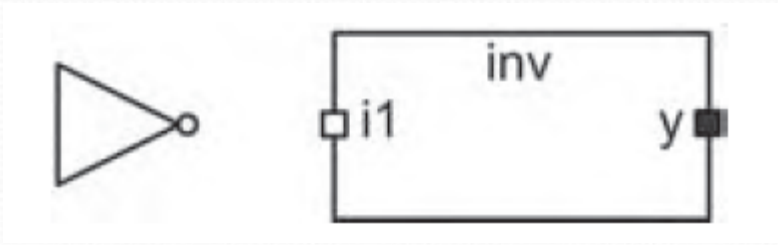
ALU ترتیبی

```
ENTITY alu8 IS
    PORT (left_i, right_i: IN std_logic_vector (7 DOWNTO 0);
    mode : IN std_logic_vector (1 DOWNTO 0);
    aluout : OUT std_logic_vector (7 DOWNTO 0));
END ENTITY;
--
ARCHITECTURE procedural OF alu8 IS BEGIN
PROCESS (left_i, right_i, mode) BEGIN
    CASE mode IS
        WHEN "00" => aluout <= left_i + right_i;
        WHEN "01" => aluout <= left_i - right_i;
        WHEN "10" => aluout <= left_i AND right_i;
        WHEN "11" => aluout <= left_i OR right_i;
        WHEN OTHERS => aluout <= "XXXXXXXX";
    END CASE;
END PROCESS;
END ARCHITECTURE procedural;
```

طراحی سلسله مراتبی

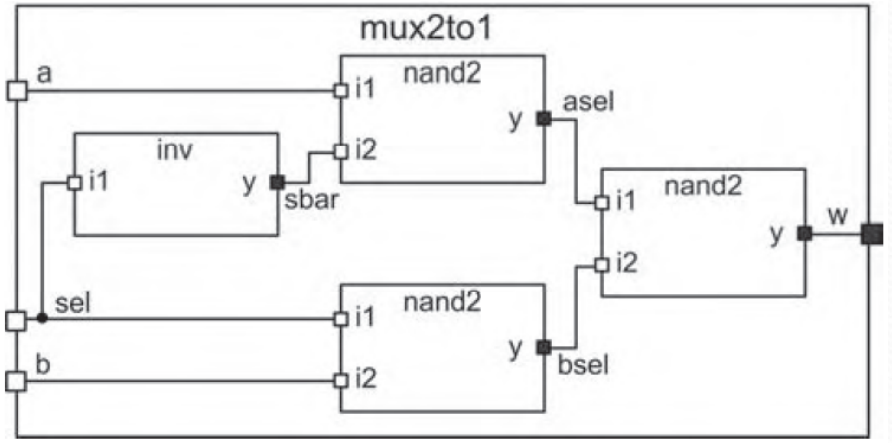



```
ENTITY nand2 IS
    PORT (i1, i2 : IN BIT; y : OUT
      BIT);
END ENTITY;
ARCHITECTURE delay1 OF
   nand2 IS
BEGIN
    y <= i1 NAND i2 AFTER 5 NS;
END ARCHITECTURE delay1;
```




```
ENTITY inv IS
    PORT (i1 : IN BIT; y : OUT
      BIT);
END ENTITY;
ARCHITECTURE delay1 OF inv
   IS
BEGIN
    y <= NOT i1 AFTER 3 NS;
END ARCHITECTURE delay1;
```

طراحی مالتی پلکسر 2 به 1 با استفاده از طرح های اولیه NOT و NAND با روش نمونه سازی مستقیم

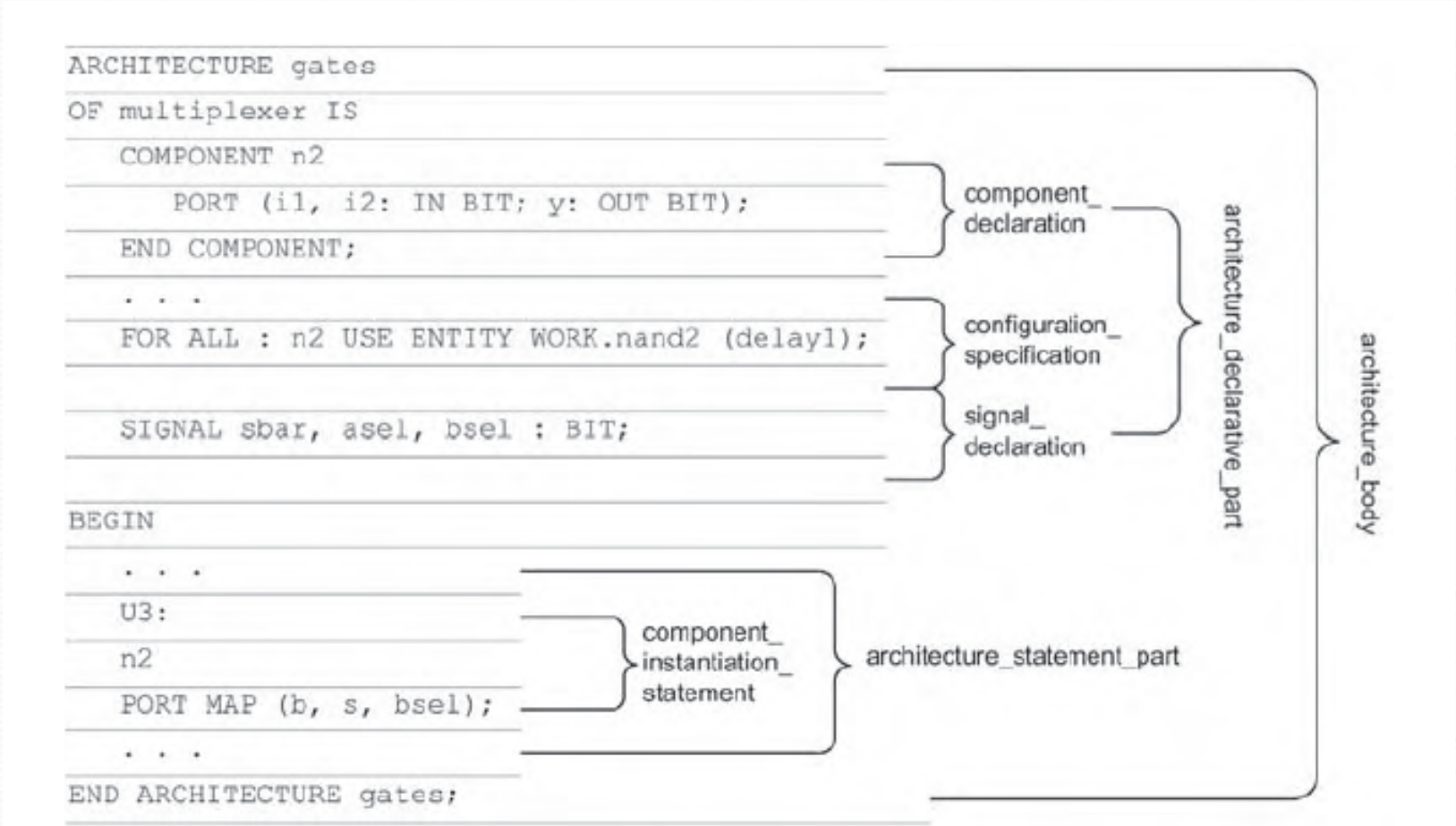


- ENTITY multiplexer IS
- PORT (a, b, s : IN BIT;
- w : OUT BIT);
- END ENTITY;
- --
- ARCHITECTURE direct OF multiplexer IS
- SIGNAL sbar, asel, bsel : BIT;
- BEGIN
- U1: ENTITY WORK.inv (delay1) PORT MAP (s, sbar);
- U2: ENTITY WORK.nand2 (delay1) PORT MAP (a, sbar, asel);
- U3: ENTITY WORK.nand2 (delay1) PORT MAP (b, s, bsel);
- U4: ENTITY WORK.nand2 (delay1) PORT MAP (asel, bsel, w);
- END ARCHITECTURE direct;

طراحی مالتی پلکسر 2 به 1 با استفاده از طرح های اولیه NOT و NAND با روش نمونه سازی با اجزای طرح

```
ARCHITECTURE gates OF multiplexer IS
COMPONENT n1
    PORT (i1: IN BIT; y: OUT BIT);
END COMPONENT;
COMPONENT n2
    PORT (i1, i2: IN BIT; y: OUT BIT);
END COMPONENT;
FOR ALL : n1 USE ENTITY WORK.inv (delay1);
FOR ALL : n2 USE ENTITY WORK.nand2 (delay1);
SIGNAL sbar, asel, bsel : BIT;
BEGIN
    U1: n1 PORT MAP (s, sbar);
    U2: n2 PORT MAP (a, sbar, asel);
    U3: n2 PORT MAP (b, s, bsel);
    U4: n2 PORT MAP (asel, bsel, w);
END ARCHITECTURE gates;
```

توضیحات کاملتر از کد نوشته شده با روش نمونه سازی با اجزای طرح



configuration تعريف

```
CONFIGURATION muxcon1 OF mux IS
FOR netlist
   FOR U1,U2 : inverter USE ENTITY WORK.myinv(version1);
   END FOR;
   FOR U3,U4,U5,U6 : andgate USE ENTITY
   WORK.myand(version1);
   END FOR;
   FOR U7 : orgate USE ENTITY WORK.myor(version1);
   END FOR;
END FOR;
END muxcon1;
```

تخصیص Guarded Signal

- تخصیص سیگنال guarded یا محافظت شده، به گونه ای اختصاص یک سیگنال به سیگنال دیگر را محافظت می کند.

- در اختصاص عادی یک سیگنال به سیگنال دیگر از نماد => استفاده می شود و سیگنال سمت چپ و راست تخصیص همیشه به هم متصل هستند.

- در اختصاص حفاظت شده اتصال یک سیگنال به سیگنال دیگر تحت شرایط خاصی صورت می پذیرد.

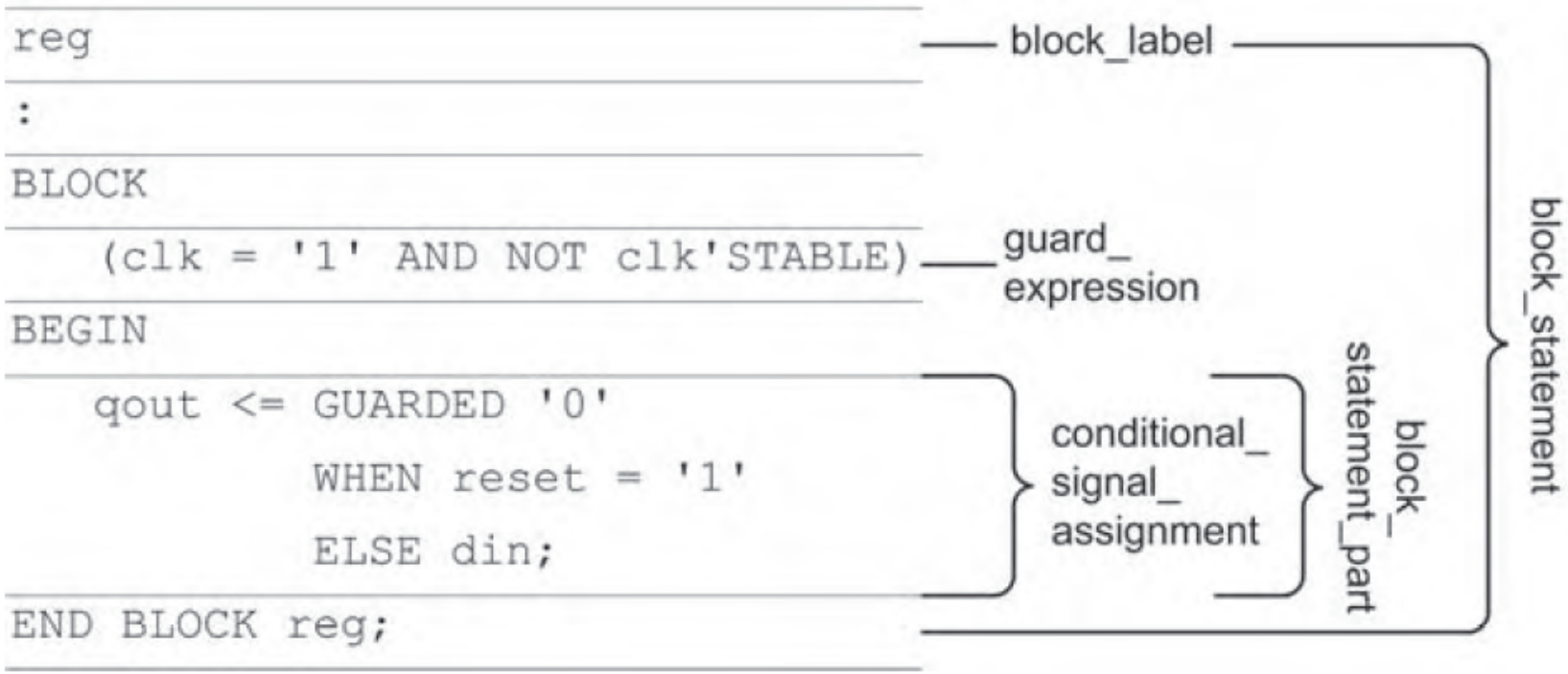
مثال از نحوه تخصیص سیگنال به صورت حفاظت شده

```
ARCHITECTURE explicit OF flipflop IS
SIGNAL GUARD : BOOLEAN;
BEGIN
    GUARD <= clk = '1' AND NOT clk'STABLE;
    qout <= GUARDED '0' WHEN reset = '1' ELSE din;
END ARCHITECTURE explicit;
```

- کلمه کلیدی GUARDED استفاده شده در سمت راست تخصیص سیگنال به این معنی است که در صورتی که سیگنال GUARD، TRUE باشد، تخصیص صورت پذیرد.
- این نحوه تخصیص را تخصیص به صورت حفاظت شده گویند و با سیگنال Guard کنترل می شود.
- توجه کنید که اختصاص سیگنال حفاظت شده فرقی با اختصاص سیگنال به صورت شرطی ندارد.

تعریف block

```
ARCHITECTURE blocking OF flipflop IS
BEGIN
    reg: BLOCK (clk = '1' AND NOT clk'STABLE) BEGIN
        qout <= GUARDED '0' WHEN reset = '1' ELSE din;
    END BLOCK reg;
END ARCHITECTURE blocking;
```



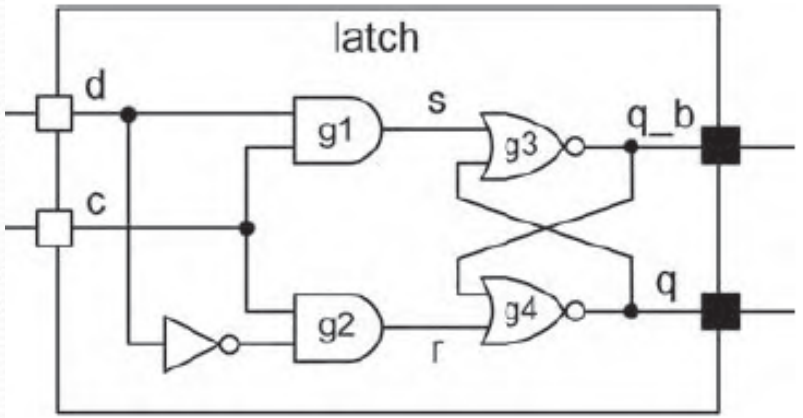
block تعریف

```
ARCHITECTURE blockport OF flipflop IS
BEGIN
reg: BLOCK (clk = '1' AND NOT clk'STABLE) IS
    PORT (r, d, c : IN BIT; q : OUT BIT);
    PORT MAP (reset, din, clk, qout);
    BEGIN
    q <= GUARDED '0' WHEN r = '1' ELSE d;
END BLOCK reg;
END ARCHITECTURE blockport;
```

تعریف block تودر تو

```
ENTITY latchflop IS
    PORT (din, clk : IN BIT; ql, qf : OUT BIT);
END ENTITY;
--
ARCHITECTURE nested OF latchflop IS
BEGIN
lat: BLOCK (clk = '1') BEGIN
    ql <= GUARDED din;
    reg: BLOCK (GUARD AND NOT clk'STABLE) BEGIN
        qf <= GUARDED din;
    END BLOCK reg;
END BLOCK lat;
END ARCHITECTURE nested;
```

D-FF با کلاک




```
ENTITY latch IS
    PORT (d, c: IN std_logic;
    q, q_b : BUFFER std_logic);
END latch;
ARCHITECTURE structural OF latch IS
SIGNAL s, r : std_logic;
BEGIN
    s <= c AND d AFTER 6 ns;
    r <= c AND (NOT d) AFTER 6 ns;
    q_b <= s NOR q AFTER 4 ns;
    q <= r NOR q_b AFTER 4 ns;
END structural;
```

فلیپ فلاپ Master-Slave

```
ENTITY master_slave IS
    PORT (d, c: IN std_logic;
    q : OUT std_logic);
END master_slave;
ARCHITECTURE dual OF master_slave IS
SIGNAL qm : std_logic;
BEGIN
    qm <= d WHEN c = '1';
    q <= qm WHEN c = '0';
END dual;
```

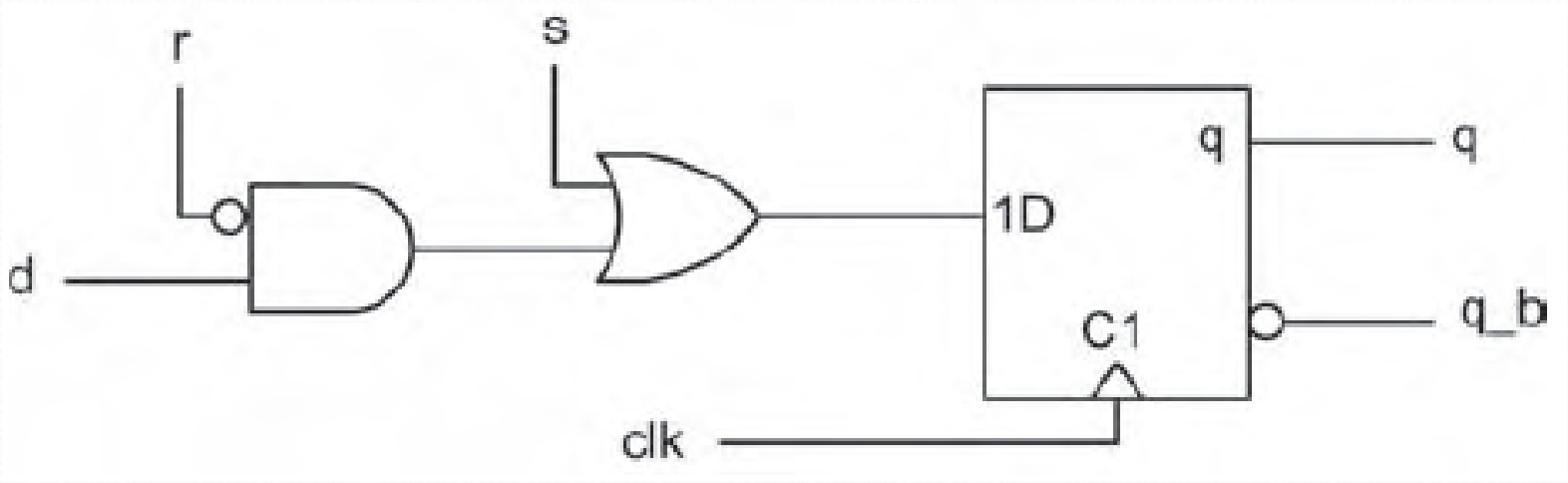
طراحی فلیپ فلاپ D با کنترل سنکرون با ورودیهای ست و ریست S و R

- تفاوت این فلیپ فلاپ با فلیپ فلاپ D معمولی در این است که خروجی در لبه بالا رونده سیگنال ساعت، سیگنال های S و R را بررسی می کند.

- در صورتی که S=1 خروجی 1 و اگر R=1 شود خروجی صفر خواهد بود.

- تنها زمانی که هر دو سیگنال S و R غیر فعال هستند خروجی برابر D خواهد بود.

ساختار مربوط به فلیپ فلاپ سنکرون شده



کد مربوط به فلیپ فلاپ سنکرون شده با S و R

```
ENTITY DFF1sr IS
    PORT (d, clk, s, r: IN std_logic; q : OUT std_logic);
END DFF1sr;
--
ARCHITECTURE behavioral OF DFF1sr IS
BEGIN
PROCESS (clk)
BEGIN
    IF clk = '1' AND clk'EVENT THEN
    IF s = '1' THEN
    q <= '1';
    ELSIF r = '1' THEN
    q <= '0';
    ELSE
    q <= d;
END IF;
END IF;
END PROCESS;
END ARCHITECTURE behavioral;
```

کنترل آسنکرون

- در کنترل آسنکرون سیگنالهای غیر از ساعت نیزکه در تغییرات سیگنال خروجی موثر هستند در لیست حساسیت پروسس وارد می شوند.

- در کنترل سنکرون فقط در صورت تغییر در سیگنال ساعت و بررسی مثلا لبه بالا روند در خروجی تغییر ایجاد میشد ولی در کنترل آسنکرون هر گونه تغییر در سیگنال هایی که در لیست حساسیت هستند یا سیگنال ساعت خروجی تغییر خواهد کرد.

- بنابراین جای IF یعنی شرطی که روی سیگنال ساعت برای کنترل سنکرون بود در کد مربوط به کنترل آسنکرون جابه جا می شود.

کد مربوط به کنترل آسنکرون

```
ARCHITECTURE asynchronous OF DFF1sr IS
BEGIN
PROCESS (clk, s, r) BEGIN
    IF s = '1' THEN
       q <= '1';
    ELSIF r = '1' THEN
       q <= '0';
    ELSIF clk = '1' AND clk'EVENT THEN
       q <= d;
    END IF;
END PROCESS;
END ARCHITECTURE asynchronous;
```