VHDL: زبان توصیف سخت افزار

FPGA: براساس سلول های منطقی قابل برنامه ریزی طراحی شده اند که حتی ارتباط بین سلول ها نیز قابل برنامه ریزی می باشد

محاسن FPGA:

۱) مدارهای دیجیتال به آسانی قابل پیاده سازی می باشد.

۲) تست مدار سریع است.

۳) برای تولید کم ارزان تمام می شود

۴) می توان تغییرات لازم را متناسب با نیاز در ساختار جدید برنامه ریزی نمود.

۵) قابل برنامه ریزی توسط کاربر است

معایب:

۱) سطح سیلیکون FPGA بصورت بهینه استفاده نمی شود یعنی گیت های زیادی دارد که از آنها استفاده نمی کنیم

۲) تاخیر و توان مصرفی نسبت به IC های که توسط کارخانه ساخته می شود بیشتر است

* با توجه به محاسن گفته شده طراحی زبان دیجیتال با زبان VHDL و پیاده سازی آن روی FPGA روز به روز بیشتر می شود

۸ شرکت های مختلفی AT&T - Altera - XILINX - ACTEL

و غیره ، انواع مختلف FPGA را تولید و با ابزارهای مخصوص به خود نظیر ISE و Quartus و ... برنامه ریزی می کنند

طرز تهیه مدارهای دیجیتال: (ادامه 6)

مرحله 1: ابتدا ویژگی طرح توسط مهندس طراح، تهیه و به برنامه ی VHDL تبدیل می شود

مرحله 2: در این مرحله برنامه ی VHDL توسط ابزارهای برنامه ریزی FPGA مانند ISE و Quartus و غیره کامپایل و سنتز می شود و مطابق با برنامه قطعات مدار (گیت ها، فیلیپ فلاپ و ...) مشخص می شود

مرحله 3: در این مرحله می توان مدار مذکور را شبیه سازی کرده و از نحوه ی عملکرد آن مطلع شد و تغییرات و اصلاحات مورد نظر را در مدار اعمال نمود

مرحله 4: در این مرحله FPGA مورد نظر انتخاب می شود و ابزار برنامه ریزی مورد نظر اتصالات و جایگذاری ها را مشخص می کند

مرحله 5: در این مرحله شبیه سازی زمانی مدار صورت گرفته و مدار از نقطه نظر تأخیر و فرکانس بررسی می شود

مرحله 6: در مرحله ی آخر فایلی بصورت رشته هایی از ۰و۱ برای برنامه ریزی سوئیچ ها و تنظیمات های مختلف FPGA تهیه شده که می توان با آن FPGA را برنامه ریزی و تست نمود

3 Subject:
Date

انواع مدارهای منطقی برنامه پذیر:

۱) PLA: شامل یک طبقه از آرایه‌ای از گیت And و یک طبقه از آرایه‌ای از گیت OR است (اسلاید 7) به طوری که هریک از ورودی‌ها و مکمل آن‌ها به گیت AND متصل شده، در نتیجه خروجی گیت And حاصل ضرب ورودی‌ها یا مکمل آن‌هاست. به همین ترتیب خروجی هر گیت OR مجموع خروجی‌های And است. در نتیجه می‌توان با استفاده از PLA تعدادی توابع منطقی را پیاده‌سازی کرد چرا که هر تابع منطقی را می‌توان بصورت مجموع حاصل ضرب نوشت. یک روش بازدهی PLA سوزاندن (قطع یک فیوز) در مسیری است که نباید اتصال وجود داشته باشد برای این کار باید مقدار جریان بیش از حد را تحمل وارد شده تا فیوز مربوطه بسوزد. اشکال آن اینست که فقط یکبار می‌شود اینکار را کرد.

- روش دیگر بکار بردن ترانزیستور به عنوان سوییچ است که برای برقراری ارتباط استفاده می‌شود

اسلاید 8:

$$f_1 = x_1 x_2 + x_1 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3$$

$$f_1' = P_1 + P_2 + P_3$$

$$P_1 = x_1 x_2$$

$$P_2 = x_1 \bar{x}_3$$

$$P_3 = \bar{x}_1 \bar{x}_2 \bar{x}_3$$

$$P_4 = x_1 x_3$$

$$f_2 = P_1 + P_3 + P_4$$

$$f_2 = x_1 x_2 + \bar{x}_1 \bar{x}_2 x_3 + x_1 x_3$$

Subject:
Date

املایه ۹ و ۱۰

- در املایه شماره ۹ هر خط عمودی داخل آرایه‌ی And ، خروجی یک مدار And که در محل تقاطع هر سطر و ستون یک سوئیچ قابل برنامه‌ریزی قرار داده شده است .

- سوئیچ قابل برنامه‌ریزی معمولاً فیوز یا ترانزیستور باشد که با ابزارهای برنامه‌ریزی، برنامه‌ریزی می‌شود

قسمت پایین شکل شامل گیت‌های OR است که در محل خروجی And های بالا به معکوس کننده متصل شده‌اند. (برای تقویت جریان و ولتاژ)

- خروجی معکوس کننده ها به وسیله‌ی یک سوئیچ به خط‌های افقی متصل شده که همانند قسمت بالا خروجی هر خط افقی تولید یک And می‌کند بنابراین در روش فوق می‌توان ورودی And ها با ابزار برنامه‌ریزی ، برنامه‌ریزی نمود

PLA چون دو طبقه‌ی قابل برنامه‌ریزی دارد گران تمام می‌شود و تاخیر مدار نیز نسبتاً زیاد است و در نتیجه سرعت آن کم می‌شود برای حل این مسئله مدارهای قابل برنامه‌ریزی PAL ساخته شد.

5. Subject:
Date

۲) PAL : مدارهای PAL : در PLA هر دو آرایه And و OR قابل برنامه ریزی است

اما در PAL فقط آرایه And قابل برنامه ریزی است و خروجی های گیت های And بطور دائم

به ورودی های OR متصل می شود (اسلاید ۱۲)

در PAL چون فقط گیت های And قابل برنامه ریزی هستند، لذا ساخت آن

هزینه کمتری دارد و سرعت آن نیز بیشتر است

- برای جبران غیر قابل برنامه ریزی بودن OR سازندگان آن انواع مختلفی از PAL را به بازار

عرضه کردند که به لحاظ تعداد گیت های OR و نیز تعداد ورودی های آن با هم متفاوتند

- یکی از مشهورترین PAL های موجود در بازار، PAL "16R6" می باشد که در اسلاید

شماره 16 آنرا می بینید، دارای 8 ورودی با تاخیر، 8 گیت OR، 8 خروجی، که دوتا

از خروجی ها بصورت ترکیبی بوده و بقیه خروجی دیگر، خروجی های فیلیپ فلاپ D هستند

- معمولاً خروجی های PAL در گیت های OR به فیلیپ فلاپ D متصل است لذا می توان

مدارهای ترتیبی با پالس ساعت نیز در آن ها پیاده سازی نمود.

- در PAL های نسل های بعدی بجای فلیپ فلاپ از مجموعه‌ای از MUX ها و فلیپ فلاپ ها به نام Microcell استفاده شده که در این صورت با برنامه‌ریزی ورودی های MUX می توان خروجی PAL را مستقیماً برای تابع F یا معکوس آن استفاده نمود و یا خروجی تابع را از فلیپ فلاپ گرفت (اسلاید 16)

- مدارهای منطقی PAL و PLA به مدار های "sPLD" معروف هستند که ظرفیت آنها حدوداً 200 گیت است . sPLD ها برای محدوده‌ی وسیعی از کاربرد ها مناسب است اما برای طراحی سیستم های پیچیده‌ی دیجیتال باید از CPLD ها استفاده کنیم .

مدارهای منطقی برنامه پذیر CPLD : PAL ها و IC های قابل برنامه‌ریزی خوبی هستند ولی برای طراحی سیستم های دیجیتال پیچیده ممکن است نیاز به چندین PAL باشد ، برای این منظور CPLD ها طراحی شدند که از چندین بلوک منطقی همانند اسلاید 17 تشکیل شده اند و با سیستم‌های ارتباطی و سوییچ های قابل برنامه‌ریزی با هم ارتباط داده می‌شود .

- CPLD های تجاری با اندازه های مختلف ، از ۲ تا ۱۰۰ sPLD تشکیل شده‌اند

ظرفیتی حدود ۱۰۰ تا ۱۵۰۰۰ گیت منطقی دارد.

- CPLD ها تاخیر کمی در حدود چند نانو ثانیه دارند لذا بسیار سریع بوده و در حدود فرکانس ۱۰۰ مگاهرتز کار می کنند.

- فاکتورسل به مجموعه ای از فیلیپ فلاپ ها و mux ها و ... که در مدارهای برنامه پذیری می توان با برنامه ریزی ورودی ها خروجی ها را تعیین کرد Microcell می گویند (مانند اسلایس ۱۶)

در اسلایس ۱۸ سری max 7000 از شرکت Altera را می بینیم که در واقع نشان دهنده ای یک فاکتورسل در این تراشه است و همانطور که در شکل می بینیم ورودی در بقیه فاکتورسل را دریافت کرده و با گیت های And و Or و در آخر یک فیلیپ فلاپ خروجی مورد نظر را تولید می کند. شرکت Xilinx نیز سری XC 7000 را تولید نموده است که شبیه max 7000 از شرکت Altra است.

CPLD ها به علت سرعت زیاد و ظرفیت بالا (حدود چند هزار گیت در یک IC) برای نمونه سازی سیستم دیجیتال، نظیر کنترل کننده های گرافیکی، شبکه های کامپیوتری

قسمت‌هایی از processor ها (CPU) و غیره بکار گرفته می‌شود

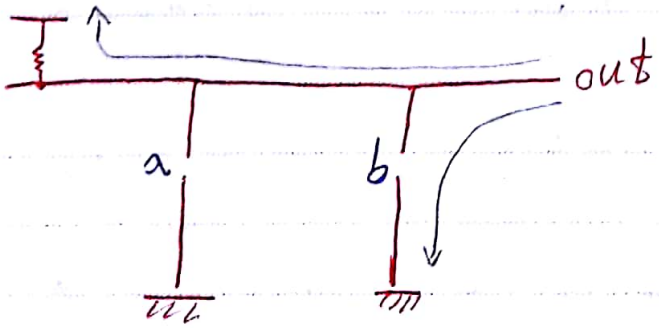
نکته: یکی از محاسن اصلی CPLD ها قابل برنامه‌ریزی مجدد می‌باشد یعنی می‌توان آن را بطور دلخواه توسط برنامه‌ریزی CAD (ابزارهای) برای طراحی سیستم‌های خاص برنامه‌ریزی کرد و در صورت لزوم آنرا مجدد برنامه‌ریزی نمود.

- سویچ‌های قابل برنامه‌ریزی FPGA, CPLD:

۱) ترانزیستور سویچ قابل برنامه‌ریزی با گیت شناور اولین نوع سویچ قابل برنامه‌ریزی فیوزی می‌باشد که با برنامه‌ریزی آن قطع می‌شود که در PLA ها استفاده می‌شود

برای مدارهای مجتمع با ظرفیت زیاد مانند CPLD ها، ترانزیستور با گیت شناور به عنوان سویچ قابل برنامه‌ریزی استفاده می‌شود. مانند حافظه‌های EPROM و EEPROM

ترانزیستور سویچ قابل برنامه‌ریزی با گیت شناور مانند یک فیوز عمل می‌کند که مطابق با اسلاید ۱۹ است



a	b	
0	0	1
0	1	0
1	0	0
1	1	0

ترانزیستورهای بایت شناور از نوع NMos به نام EPROM معروف است که به دفعات قابل پاک شدن و برنامه ریزی مجدد هستند این ترانزیستورها دارای دو گیت می باشند که یکی از آنها مانند ترانزیستورهای NMos معمولی به سیم اتصال خارج ترانزیستور متصل می شود گیت دوم به نام گیت شناور داخل عایق ترانزیستور قرار دارد.

برای برنامه ریزی این ترانزیستورها یک ولتاژ مثبت، در گیت بیرون قرار می گیرد و باعث می شود شارژهای منفی (الکترون) به گیت شناور وارد شوند. بعد از حذف ولتاژ مثبت مذکور این شارژهای منفی در گیت شناور باقی می ماند (حبس می شود) حال اگر ولتاژ یک منطقی یعنی 5 ولت در ورودی گیت بیرونی قرار گیرد، ترانزیستور نمی تواند به هدایت الکترون برود. به عبارت دیگر ترانزیستور برای همیشه قطع می ماند برای پاک کردن گیت شناور ترانزیستور مذکور باید این شارژهای منفی از بین بروند.

برای این کار اشعه ماورای بنفش به گیت ترانزیستور تابیده می شود تا شارژهای منفی از بین بروند، در اینصورت ترانزیستور مذکور همانند یک ترانزیستور معمولی NMos

با گیت خارجی کار می کند.

ترانزیستور EEPROM نیز شبیه ترانزیستور EPROM با این تفاوت که بصورت الکتریکی را یک پالس پاک می شوند.

۲) سوییچ های قابل برنامه‌ریزی با حافظه SRAM :

در برخی FPGA ها سوییچ های قابل برنامه ریزی ترانزیستوری به کار برده می شود که با حافظه SRAMs برنامه‌ریزی یا کنترل می شود. در اینصورت اگر خروجی حافظه SRAM برابر ۱ باشد، سوییچ ترانزیستور مذکور بسته می شود و دو سیم را به یکدیگر متصل می کند. همانطورکه در اسلاید ۲۰ می بینید. سوییچ های ترانزیستوری قابل برنامه‌ریزی یک بطرافتی که یک سلول منطقی هست را به یک بطر مجودی که ورودی سلول منطقی دیگری است متصل می نماید

در برخی FPGA ها حالتی بالاتر بعنوان سوییچ استفاده می شود. هرکیک از ورودی های مورد نظر MUX به خروجی آن متصل می شود، یعنی یک خط افقی به یک سیم (خط) مجودی که ورودی سلول منطقی دیگری است متصل می شود و MUX با حافظه SRAM کنترل می شود.

(اسلاید ۲۱) ساختار وپک سلول حافظه SRAM می باشد که از دو ترانزیستور (T_6 - T_4 - T_5 - T_3) تشکیل شده است که یک فلیپ فلاپ CMOS از حافظه SRAM را تشکیل می دهند.

اطلاعات از طریق ترانزیستور T_1 در حافظه SRAM ذخیره می شوند.

خروجی حافظه SRAM به ترانزیستور T_2 متصل است که اگر خروجی ۱ ضدکود برابر ۱ باشد، ترانزیستور T_2 بسته شده و اگر صفر باشد ترانزیستور باز می گردد (قطع می شود). بنابراین ترتیب سویچ T_2 توسط حافظه SRAM برنامه ریزی می گردد

- در اسلاید 22 سویچ قابل برنامه ریزی آنتی فیوز را می بینید. درواقع آنتی فیوز برعکس فیوز می باشد یعنی در ابتدا مدار باز می باشد و پس در صورت برنامه ریزی بسته می شود. چون ساختار آنتی فیوز با تکنولوژی CMOS ساده است، در برخی FPGA ها استفاده می شود.

نکته: سویچ آنتی فیوز فقط یکبار برنامه ریزی میشود. لذا FPGA برای همیشه برنامه ریزی شده و قابل برنامه ریزی مجدد نیست. اما سویچ SRAM با تغییر مجدد برنامه ریزی می شود.

پس FPGA های با سویچ SRAM را می توان دوباره برنامه ریزی نمود اما مشکل SRAM اینست که با قطع برق اطلاعات از بین می رود. لذا باید از باتری یا چیز های دیگر برای FPGA استفاده نمود

- ساختار FPGA : مدارهای قابل برنامه ریزی پرقدرت تر FPGA ها می باشند که از آرایه ای از بلوک ها یا سلول های منطقی مانند اسلایس 23 تشکیل شده اند. که توسط خطوط ارتباطی به یکدیگر متصل شده اند.

- ظرفیت FPGA ها معادل تعداد گیت های NAND دوورودی سنجیده می شوند. امروزه ظرفیت معمولی FPGA ها حدود ۲۰۰۰۰ گیت NAND به بالا می باشد.

- بلوک یا سلول منطقی می تواند از تعدادی mux یا جدول LUT (look up table) تشکیل شده باشد. FPGA ها با ظرفیت حدود چند هزار گیت، دارای امکانات بیشتری نسبت به CPLD ها هستند، لذا می توان با آنها سیستم های پیچیده تری را نیز طراحی نمود.

در FPGA ها سوییچ های قابل برنامه ریزی زیادی بکار برده می شوند که دارای تأخیر است. لذا FPGA ها دارای تأخیر بیشتری نسبت به CPLD ها و PAL ها می باشند

از طرفی مدارهای بسیار پیچیده تر که اصولاً باید به کارخانه ی IC سازی سفارش شوند را می توان با VHDL طراحی و با برنامه های برنامه ریزی FPGA سریعاً در FPGA پیاده سازی نمود.

روش های طراحی سیستم ها دیجیتال با FPGA : طراحی سیستم های دیجیتال با FPGA

با ابزارهای برنامه ریزی FPGA بصورت اتوماتیک انجام می شود این ابزارها توسط سازندگان FPGA تهیه و در اختیار ما قرار می گیرد.

برای طراحی سیستم های دیجیتال با ابزارهای مذکور و پیاده سازی آن بر FPGA، مراحل زیر باید طی شود

- وارد کردن طرح اولیه و کامپایل
- شبیه سازی
- سنتز و آماده کردن طرح برای پیاده سازی
- تهیه فایل پیاده سازی برای FPGA
- شبیه سازی زمانی
- برنامه ریزی FPGA

تکمیل اسلاید ۳۹

۱) وارد کردن طرح اولیه و کامپایل: طرح دیجیتال مورد نظر ابتدا ممکن است در محیط ابزار FPGA بصورت شماتیک کشیده شود و یا با editor متن (HDL) طرح مذکور با زبان توصیف سخت افزاری VHDL توصیف گردد

۲) شبیه سازی: برای شبیه سازی و بررسی طرح مذکور سیگنال های ورودی طرح داده می شود و سپس خروجی آن ها بررسی می گردد نمونه ای از این شبیه سازی را در اسلاید ۲۰ می بینید

ورودی های a و b و c و d ورودی mux بوده که یک بیتی بوده و sel ورودی کنترل دو بیتی و o خروجی mux یک بیتی است

۳) سنتز و آماده کردن طرح برای پیاده سازی: سنتز یعنی تبدیل برنامه ی VHDL به معادل عناصر منطقی آن مانند جمع کننده، mux، Dec، FF و غیره و اتصالات مربوطه که به آن Netlist می گویند

چون طرح اولیه منطقی معمولاً بهینه نیست لذا در برنامه های FPGA الگوریتمی وجود دارد که طرح اولیه را بهبود می دهد

۴) تهیه فایل پیاده سازی: در این مرحله نوع FPGA را مشخص می کنیم و با استفاده از فایل نتز که شامل بلوک های منطقی است فایلی با یک سری صفر و یک تهیه می شود که با استفاده از آن سوییچ ها و بلوک های منطقی FPGA می توان برنامه ریزی شود. علاوه بر این در این مرحله، محل بلوک های منطقی مورد نظر در FPGA مشخص شده و ارتباطات آن ها نیز مشخص می شود (مانند اسلاید ۴۱)

۵) شبیه سازی زمانی: بعد از مشخص شدن گیت ها و بلوک های منطقی و مسیر ارتباط آن ها مقدار تأخیر گیت ها نیز مشخص شده و در نتیجه زمان بندی و تأخیر واقعی مدار نیز مشخص می شود (این مرحله اختیاری است)

۶) برنامه ریزی FPGA: ابزار برنامه ریزی FPGA با کابلی به از کامپیوتر به برد متصل می شود و فایل نهایی برای برنامه ریزی FPGA در آن پیاده سازی می شود (اسلاید ۴۳ یک نمونه بردی را که برای پیاده سازی طرح دیجیتال در FPGA طراحی شده است را نشان می دهد)

« اتمام فصل ۱ »

نگاهی به زبان توصیف سخت افزاری VHDL : در طراحی مدارهای دیجیتال معمولاً مدار را به تعدادی بلوک دیاگرام تقسیم می‌کنیم سپس مدار داخلی هر بلوک را تقسیم می‌نماییم در VHDL نیز هر مدار به تعدادی بلوک تقسیم می‌شود سپس آن‌ها را توصیف می‌کنیم هر بلوک شامل مجموعه‌ای از Entity و Architect را یک طرح می نامیم سپس آن بلوک را توصیف می‌کنیم (اسلاید ۴۴)

در ساده‌ترین حالت یک طرح دیجیتال شامل یک بلوک است . همانطورکه در Data sheet یا برگه مشخصات یک IC، ورودی و خروجی های IC شرح داده شده است در VHDL نیز Entity ورودی وخروجی های مدار یا ارتباط طرح با خارج از مدار از طریق Port آن مشخص می شود

همچنین در Data sheet یک IC معماری داخل مدار نیز رسم می‌شود . (همانند اسلاید ۴۵) یعنی طرز اتصال قطعات مدار به ورودی و طرز کار مدار توصیف می‌شود . به عنوان مثال اگر طرح یک نیم جمع کننده را HA در نظر بگیریم در این صورت مدار آن مطابق "اسلاید ۴۵" می‌باشد که ورودی وخروجی آن بصورت

زیر توصیف می‌شود.

```
Entity ha is
    Port(a,b :in bit;
         s,co:out bit);
end ha;
```

۱) ورودی in : در توصیف شکل فوق ورودی های a و b بصورت زیر تعریف می‌شود

۲) نوع bit نوع ورودی نیز با bit مشخص می‌شود که معنی آن اینست که ورودی از نوع bit است (یعنی binary) و می‌تواند مقادیر ۰ یا ۱ را داشته باشد که نمایش صفر و یک منطقی است. با سیگنال های نوع bit می‌توان عملیات منطقی And ، or ، not ، NAnd ، Nor ، Xor و غیره را انجام داده و همچنین عملیات مقایسه =، /=، <، >، <= و >= و ... را نیز برای دو سیگنال از نوع bit انجام داد. که نتیجه آن boolean است. یعنی نتیجه آن بشکل True یا False باشد

۳) خروجی out : خروجی مدار فوق s و co نیز در زبان VHDL به شکل s,co : out bit ; نوشته می‌شود که کلمه out بجای ";" به معنی

خروجی است و نوع آن نیز بشکل bit مشخص شده است.

در پایان Entity کلمه end ha; قرار داده شده است که پایان Entity را مشخص می کند اما توصیف خود بلوک جمع کننده که در واقع طرز اتصالات گیت های آن است در Architecture مدار ما باشد مدار بلوک جمع کننده طبق کد زیر توصیف می شود

```
architecture dataflow of ha is        -- dataflow: نام دلخواه ، ha: نام تعریف شده
begin
    s <= a xor b; ①
    co <= a and b;
end dataflow;
```

کلمه ی Architecture معماری طرح را اعلام می کند و dataflow یک نام اختیاری است ! عبارت Architecture با کلمه Begin شروع شده و به این معنی است که ورودی های a و b با یکدیگر xor شده و نتیجه خروجی در s ریخته می شود.

شامل اختصاص یا تخصیص "<=" می باشد.

- با توجه به مطالب فوق حالت کلی Entity بصورت زیر می باشد.

Entity نام دلخواه is
port (نام ورودی ; in نوع بیت ها
; نام خروجی ; out نوع بیت ها
);
end نام دلخواه;

و برای Architecture

Archeitecture نام دلخواه of نام تعریف شده is
Begin
عبارت VHDL
end نام دلخواه;

۴) بافر Buffer : در برخی طرح ها اطلاعات خروجی باید خوانده شده و عملیاتی بر روی آنها ایجاد شود (همانند اسلاید ۴۶) بدین ترتیب بافر را هم می توان به عنوان نوع خروجی انتخاب کرد.

ق - ورودی - خروجی (in-out) : در برخی طرح ها که سیگنال های Port هم باید ورودی و هم خروجی باشند به شکل in-out تعریف می شود . در این موارد پایه های Port دو طرفه است یعنی هم می توان اطلاعات را از آن خواند و هم خروجی را در آن قرار داد

نکته : در Entity عبارت نام مدار بعد از end ، و در Architecture عبارت نام Architecture بعد از end اختیاریست .

مثال : Architecture در نظر بگیرید . جواب sum چقدر است ؟

```
Begin
   A<=7
   sum <=A+B;
    B<=3;                    =>sum=10
end;
```

نکته : ترتیب نوشتن عبارات در بدنه ی Architect هم نیست چون در Arch برنامه ی VHDL تمام عبارات همزمان اجرا می شود در واقع همانند مدار های گیت های مدار های ترکیبی است که همزمان کار می کنند

سیگنال: برای اتصال قسمت های مختلف مدار از سیگنال استفاده می شود. از نظر نرم افزار سیگنال مانند یک سیم که قسمت های مختلف طراحی به یکدیگر متصل می کند عمل می کند.

پورت نیز نوعی سیگنال است چرا که ورودی و خروجی مدار را به خارج از طرح متصل می کند

سیگنال از دو قسمت تشکیل شده است قسمت اول نام سیگنال و قسمت دوم که با دو نقطه (:) مجزا می شود نوع آن است. به عنوان مثال:

```
signal  a,b : bit;
```

اعلانات: سیگنال در قسمت اعلانات، قبل از begin آورده می شود.

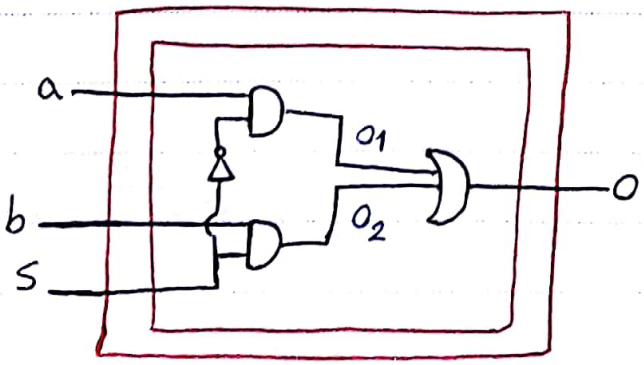
اگر سیگنال چند بیتی باشد، بصورت "bit_vector" معرفی می شود. به عنوان مثال:

```
signal  a,b ; bit_vector  (7 down to 0);
```

مثال: برنامه VHDL بنویسید که اگر s برابر صفر باشد، خروجی o برابر a و اگر s مساوی یک باشد، خروجی o برابر b می شود (اسلاید 47)

s=0 ⟶ o=a

s=1 ⟶ o=b



```
Entity Multi is
port (a, b, s : in bit;
      o       : out bit);
end;

Architecture Mux of Multi is
    signal o1, o2 : bit;
Begin
    o1 <= a and not(s);
    o2 <= b and s;
    o <= o1 or o2 ;
end;
```

نکات محوری در VHDL :

1- comment : در هر عبارت VHDL با قرار دادن "--" می توان comment ای برای آن نوشت.

-- Inja tozih midahim

2- نام گذاری در VHDL شامل حروف، ارقام و _ (underline) است. نام باید با حرف شروع شده و بهتر از کمتر از ۱۵ کاراکتر باشد.

3- حروف بزرگ و کوچک: VHDL به حروف کوچک و بزرگ حساس نیست، یعنی هردو را یکی یعنی می داند

4- نمایش 0 و 1 منطقی در VHDL : 1 و 0 منطقی در VHDL ، داخل '0' نمایش داده می شود، تعدادی 0 و 1 در "1001" نمایش داده می شود

5) نمایش اعداد در VHDL : در VHDL اعداد int ، بصورت پیش فرض اعداد ده دهی معمولی هستند مانند: 12 .. برای نمایش اعداد در پایه 2 یا باینری از علامت b جلوی عبارت استفاده می شود.

b "1001"
B "0101"

- برای نمایش اعداد در پایه ی هگزا دسیمال یا پایه 16 از x استفاده می کنیم .

- برای نمایش اعداد در پایه ی اوکتال (8) از علامت o استفاده می شود .

o"1314"

ب - مقدار اولیه دادن به سیگنال : در مواقعی که یک مدار شبیه سازی می شود ، تمام سیگنال ها یک مقدار اولیه ی پیش فرض می گیرند

signal o1 : bit := '1'; ← نحوه مقداردهی
o1 ← اسم

انواع دیگر نمایش اطلاعات سیگنال :

1 - نوع int ← integer : در این نوع سیگنال می توان عدد integer که اعداد مثبت یا منفی می باشند را ذخیره کرد . سیگنال نوع int برای انجام محاسبات + ، - ، * ، ÷ ، توابع abs ، مد (با قیمانده) و ... بکار برده می شود .

c = a mod b
با قیمانده تقسیم a بر b

مثال: برنامه VHDL بنویسید که دو عدد integer a و b را دریافت کرده و با یکدیگر جمع آنها را در c ذخیره کند.

ابتدا جمع a و b را در c ذخیره کنید بعد mod را بگیرید.

```
a,b integer range 0 to 7

Entity Add is
port (a,b: in integer range 0 to 7   (۳ بیت)
      c: out integer range 0 to 15);  (۴ بیت)
end;

Architecture Ali of Add is
begin
    c <= a+b;
end;
```

2 - نوع bit_vector : اکثر اوقات ورودی و خروجی های ما، چند بیتی هستند در VHDL، مجموعه ی چندین bit بصورت آرایه تعریف می شوند که نام آن ها bit_vector است.

نکته: برای اعداد باینری از down to استفاده می کنیم.

```
signal a,b : bit_vector (7 down to 0);
```

- اگر سیگنال c را c : bit_vector (0 to 3) تعریف کنیم، در اینصورت آرایه بصورت c(3) , c(2) , c(1) , کم ارزش ترین c(0) تعریف می شود.

در واقع bit lsb در سمت چپ آرایه و Msb که پرارزش ترین است در سمت راست آرایه قرار می گیرد.

```
signal x : bit_vector (1 down to 0);
signal y : bit_vector (0 to 1);

y <= x;
-> y(1) <= x(0);
   y(0) <= x(1);
```

۳- نوع Time (زمان) : این نوع برای نمایش و اندازه گیری زمان بکار می رود. به عنوان مثال:

```
q<=r nor nr after 5 ns

constant delay := 5ns

q<=r nor nr after delay
```

۴- نوع boolean : نوع boolean دارای دو مقدار false, true است

که اصولاً از نتیجه ی مقایسه ی دو مقدار، دو سیگنال حاصل می شود.

```
if r='1' then
    b<=a
    end if;
```

۵- نوع std_logic : همانطور که قبلاً توصیف کردیم در VHDL برای توصیف سیگنال ها از bit و boolean استفاده می شد. یعنی نوع bit مقادیر '0' یا '1' و نوع boolean مقادیر false و true را داشت.

```
type boolean is (False, true);

type bit is ('0', '1');
```

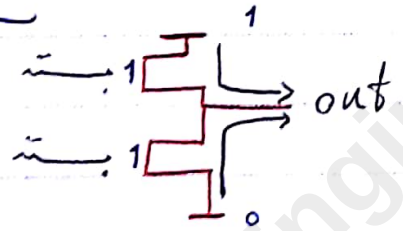
ولی همانطور که می دانیم:

تا دیر خروجی مدار همیشه ۰ و ۱ نیست . مثلاً می تواند امپدانس بالا (high impedance) یا همان z نیز داشته باشد . برای حل این موضوع ، شرکت های مختلف کلیپ های مختلفی را تعریف کردند . مثلاً نوع std_logic دارای ۹ مقدار مقدار U و X و ۰ و ۱ و Z و W و L و H می باشد .

std_logic ('U', 'W', 'X', 'L', '0', '-', '1', 'L', 'H')

U : به معنی این است که سیگنال مقداردهی ندارد . یعنی مقداری به آن تخصیص داده نشده .

X : اگر دو خروجی مختلف همزمان به مداری اعمال شود مثلاً ۰ و ۱ توسط CMOS خارج شوند



Z : زمانی که خروجی برابر امپدانس بالا باشد

W : تعریف فیزیکی ندارد .

L : صفر منطقی در حالت خواندن است

H : یک منطقی در حالت خواندن است .

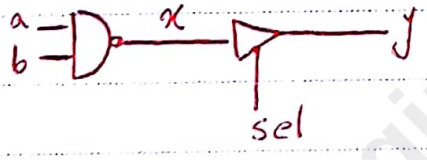
'-' Don't care است .

نکته: نوع داده std_logic این مزیت را دارد که دارای کتابخانه استاندارد در تمام FPGA ها است.

- عبارات فوق باید در تمام برنامه های VHDL که از std_logic استفاده می کنند، آورده شود تا بتوان از ویژگی های آن استفاده نمود.

```
library ieee;
use ieee.std_logic_1164.all;
```

مثال: مدار زیر را با استفاده از std_logic توصیف کنید.




```
library ieee;
use ieee.std_logic_1164.all
Entity AL is
    port (a,b,sel :in std_logic;
          y: out std_logic);
end AL;
Architecture Hassan of AL is
    signal x: std_logic;
begin
    x <= a Nand b;
    y <= x when s='0' else 'z';
end;
```

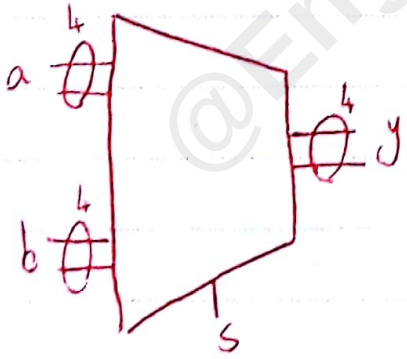
نوع std_logic_vector :

همانطورکه برای نوع bit آرایه داشتیم، برای نوع std_logic نیز آرایه‌ای از آنها در کتابخانه std_logic_1164 داریم که تمام عملگرهای bit، روی std_logic عمل می‌کند.

a,b : in std_logic_vector(7 down to 0)

نکته: در VHDL نوع signal های دو طرف عبارت باید یکسان باشد

مثال: برنامه‌ی mux2x1 با ورودی‌های a و b و خروجی y چهاربیتی با کنترل s یک بیتی بنویسید که اگر s برابر 1 باشد خروجی a برابر 1 باشد درغیراینصورت b باشد




```
library ieee;
use ieee.std_logic_1164.all;
Entity mux is
port (a,b:in std_logic_vector(3 downto 0);
      s:in std_logic;
      y:out std_logic_vector(3 downto 0));
end;
Architecture Hassan of mux is
begin
y <= a when s='0' else b;
end;
```

تخصیص سیگنال با عبارت کلی others: اگر در یک طرح تعداد بیت های سیگنال نسبتاً زیاد باشد و بخواهیم به تمام یا تعدادی از bit ها یک مقدار تخصیص دهیم در این صورت از عبارت others استفاده می کنیم

اگر بخواهیم در یک آرایه یکی از بیت ها را انتخاب کنیم از پرانتز و شماره آن بیت استفاده می کنیم

```
begin
  a <= (1 => '1' , 3 => '1' , others => '0');
```

```
b <= others => '0';
```

↳ `b <= "0000000";`

نکته: others باید آخرین انتخاب باشد.

عبارت others را در تخصیص مقدار به سیگنال به صورت شرطی نیز می توانیم استفاده کنیم

```
f <= n when s = '0' else
     m when others;
```

عملگرها در VHDL: در بخش های قبلی که تعداد کار از انواع سیگنال در VHDL بحث شد عملگرهای مختلفی را که با این سیگنال ها استفاده می شود دیدیم که به ترتیب اولویت عبارتند از:

الف) عملگرهای Not و قدر مطلق و توان

ب) عملگرهای ضرب، تقسیم، mod و Rem

ج) عملگرهای جمع، تفریق و چسباندن (&)

د) عملگرهای تساوی، عدم تساوی، کوچکتر ساوی، عدم تساوی

ه) عملگرهای منطقی AND و OR، Nand و —

و) عملگرهای علامت + و -

ا) عملگرهای منطقی:

نکته: در مورد سیگنال های bit_vector و std_logic_vector عملیات منطقی مذکور بر روی هریک از بیت های آن انجام شده و نتیجه به صورت بیت به بیت حاصل می شود

نکته: در عبارت c <= a mod b یعنی a بر b تقسیم می شود باقیمانده با علامت b به c داده می شود

- در عبارت d <= a rem b; یعنی a بر b تقسیم می شود باقیمانده با علامت a به d داده می شود

(reminder)

۲) عملگرهای ریاضی:

مثال: برنامه‌ای بنویسید که دو عدد چهاربیتی a و b را به صورت std_logic با هم جمع می‌کند و نتیجه را به sum می‌دهد

```
library ieee;
use ieee.std_logic_1164.all;
Entity Add is
port ( a,b : in std_logic_vector (3 down to 0);
       sum : out std_logic_vector (3 down to 0));
end;
Architecture Hassan of Add is
begin
   sum <= a+b;
end;
```

use .std.std_logic_1164.unsigned;

در برنامه ی فوق عبارت

استفاده می شود تا اجازه دهد عملیات ریاضی برای اعداد بدون علامت بر روی سیگنال های std_logic استفاده شود.

- عملگرهای مقایسه ای (نسبی): این عملگرها دو عملگر را با هم مقایسه کرده و نتیجه را اعلام می کند به عنوان مثال

```
if (A=B AND (B>=C or d/=e))
then
g <= 1;
end if;
```

- عملگر چسباندن (&): این عملگر در VHDL برای به هم چسباندن سیگنال یا گروهی از سیگنال ها استفاده می شود به عنوان مثال دو باس (Bus) چهار بیتی را می خواهیم به هم بچسبانیم و نتیجه ی خروجی را در یک Bus هشت بیتی ذخیره کنیم

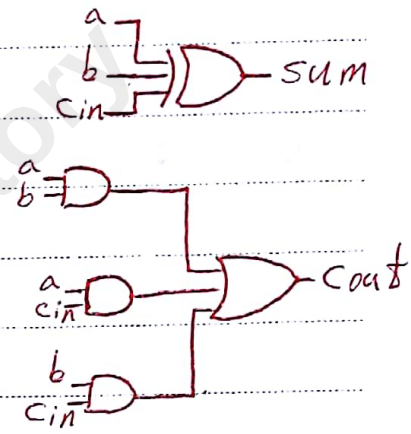
```
Architecture ____
signal a,b : bit_vector (3 down to 0);
signal c : bit_vector (7 down to 0);
begin
    c <= a&b;
end;
```

3 2 1 0
7 6 5 4 3 2 1 0

```
c(7)<=a(3);
c(6)<=a(2);
c(5)<=a(1);
c(4)<=a(0);
c(3)<=b(3);
c(2)<=b(2);
c(1)<=b(1);
```

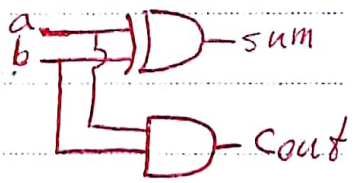
نکته: عملگر چسباندن حتماً در طرف راست عبارت قرار می‌گیرد و سیگنال‌ها از یک نوع را به یکدیگر می‌چسباند

تمرین: برنامه‌ی FA با ورودی‌های a,b,cin و خروجی Cout و sum/بنویسید




```
library ieee;
use ieee.std_logic_1164.all;
entity dataflow is
port (a,b,cin : in std_logic;
      sum,cout: out std_logic);
end;
Architecture df of dataflow is
begin
    sum <= a xor b xor cin;
   cout <= (a and b) or (b and cin) or (a and cin);
end;
```

سوال کوییز: برنامه‌ی Half Adder:




```
library ieee;
use ieee.std_logic_1164.all;
entity HA is
    port(a,b:in std_logic;
         sum,cout:out std_logic);
end;
Architecture H of HA is
sum <= a xor b;
cout <= a and b;
end;
```

روش های توصیف یا مدلسازی سخت افزار در VHDL:

الف) Behavioral level: اولین مرحله ی یک طرح، طراحی در سطح سیستم است که شامل عملیات منطقی و ریاضی روی ورودی ها می باشد. در این مرحله به پیاده سازی سخت افزاری نیازی نیست.

ب) توصیف Data Flow: در این مرحله مدار را به طریقی که داده ها داخل سیستم می شود و از قسمت های مختلف عبور می کند توصیف می کنند.

ج) توصیف structural: در این روش مدار بصورت مجموعه ای از قطعات و طرز اتصال آن ها توصیف می شود به عبارتی دیگر مشخص می شود مدار از چه گیت هایی تشکیل شده و طرز ارتباط آن ها با یکدیگر چگونه است با این ساختار می توان مدار را بصورت بلوک دیاگرام به شکل سلسله مراتبی (Hierarchy) توصیف نمود به عنوان مثال هر جمع کننده ی کامل (FA) از دو نیم جمع کننده (HA) تشکیل شده است

- اگر مدل Behavioral به ابزار سنتز بدهیم یک طرح کلی تولید می کند که ممکن است بهینه نباشد ولی اگر توصیف structural را به ابزار سنتز دهیم، طرح مشخص تر

با حداقل ابزار تولید خواهد کرد . البته هر یک از روش های فوق را می توان با هم

تلفیق نمود با توجه به مطالب فوق یک مدار دیجیتال را می توان همانند اسلاید ۵۷ بصورت های

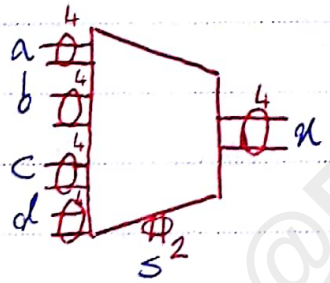
مختلف مدل نمود

همزمانی VHDL برای مدارهای ترکیبی با توصیف Data Flow

همانطور که در اسلاید ۵۸ مشاهده می کنید طرز یک عبارت VHDL بطور همزمان در Arch

توصیف Data Flow نشان می دهد که ترتیب نوشتن عبارات در Arch مهم نیست

مثال : برنامه ی VHDL مدار mux 4x1 4 بیتی را بنویسید شکل 63




```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
port(
  a,b,c,d :in std_logic_vector(3 down to 0);
    s:in std_logic_vector(1 down to 0);
    x:out std_logic_vector(3 down to 0);
  end;
Architecture M of Mux is
  begin
    if (s="00" then  x<=a;
else if (s="01" then  x<=b;
else if(s="10" then  x<=c;
else  x<=d;
end;
```

نوشت: قسمت Arch را می توان با when

```
x<=a when s="00" else
x<=b when s="01" else
x<=c when s="10" else
x<=d;
end;
```

تخصیص یا انتساب به سیگنال بصورت انتخاب : در این روش برحسب شرط یکی از مقادیر را به سیگنال سمت چپ اختصاص می‌دهیم.

فرق with s select با x<= a when (s="00") else این است که x<=a when "00"

که این روش تست شرط یک سیگنال است و اولویت تمام شرط‌ها نیز یکسان است

در روش تخصیص مقداری به سیگنال بصورت when_else شرط‌ها به ترتیب از بالا به پایین تست می‌شود در صورتی که در روش تخصیص مقدار بصورت with-select تمام شرط‌ها در سطرهای مختلف همزمان بررسی شده و اولویتی نسبت به هم ندارند

process در VHDL ترتیبی : process در Arch یک طرح قرار می‌گیرد و عبارت داخلی process مانند برنامه‌های معمولی برنامه نویسی مانند C به ترتیب اجرا شده و خطوط کدی پیش از دیگری اجرا می‌شود که به VHDL ترتیبی معروف است

برنامه‌ای VHDL برای یک FF ، Rs بنویسید که اگر S=0 خروجی Q=1 و اگر R=0 بود خروجی Q=0 گردد در غیر اینصورت تغییر نکند

```
library ieee;
use iee.std_logic_1164.all;
Entity FlipFlop is
port (s,r: in std_logic;
      q: in out std_logic);
end;
Architecture result of FlipFlop is
begin
 process (s,r,q)
  begin
   if s='0' then q<='1';
elseif r='0' then q<='0';
 else q<=q;
 end if;
 end process;
 end;
```

S=0 → Q=1
R=0 → Q=0

در Arch 1 طرح کلی به نام process بعنوان یک عبارت همزمان تلقی می شود

بنابراین اگر چندین process و چندین عبارت همزمان داشته باشیم

همه ی process ها و عبارت ها همزمان باهم اجرا می شود.

process با کلمه کلیدی process شروع شده و داخل پرانتز سیگنال هایی که توازات

با تغییر آنها عبارات داخل process اجرا شوند قرار می گیرد.

نکته: اگر if ساده یعنی بدون else باشد فقط یک عبارت را ارزیابی می کند و درصورت

صحیح بودن آنرا اجرا کرده و از if خارج می شود ولی اگر if else داشته باشد

بین عبارت ها انتخاب کرده و پس از اجرا از if خارج شده و برنامه را ادامه

می دهد.

مثال: برنامه‌ای VHDL‌ای به سورت behavioral برای FlipFlop بنویسید. d ورودی و q خروجی

بطوری که در لبه‌ی پایین رونده پالس ساعت، اطلاعات ورودی d به خروجی q منتقل شود

```
library ieee;
use ieee.std_logic_1164.all;
Entity FlipFlop is
    port (d,clk :in std_logic;
          q :out std_logic);
end;

Architecture FD of FlipFlop is
begin
 process (clk,d)
 begin
 if (clock'event and clk='0') then q<=d;

 end if;
 end process;
 end;
```

نکته : عبارت if از بالا به پایین به ترتیب اجرا می‌شود هر if با endif پایان یافته و اگر هیچ یک از شرط های if صحیح نباشد از if خارج شده و عبارت بعد از endif اجرا می‌شود

مثال: برنامه VHDL ای برای یک ثبات 8 بیتی با FF ، d بنویسید که با لبه‌ی بالارونده پالس ساعت اطلاعات 8 بیتی ورودی A را به خروجی منتقل کند

```
library ieee;
use ieee.std_logic_1164.all;
Entity FlipFlop is
 port( A: in std_logic_vector (7 down to 0);
       clk: in std_logic;
       D: out std_logic_vector (7 down to 0));
 end;
Architecture FD of FlipFlop is
 begin
 process (clk);
 begin
if (clk'event and clk='1') then D<= A;
endif;
end process;
end;
```

process در مدارهای ترکیبی: process برای مدارهای ترکیبی نیز می‌توان بکار برد

چون در مدارهای ترکیبی با هر تغییر ورودی، خروجی نیز تغییر می‌کند لذا تمام ورودی های

مدار باید در لیست سیگنال های process باشد تا با تغییر آنها process اجرا شده

و مقدارهای جدید محاسبه شود

مثال: مقایسه کننده ای دارای دو ورودی a و b 16 بیتی و خروجی equal یک بیتی

که اگر a=b باشد equal=1 شود در غیراینصورت equal=0 شود

```
library ieee;
use ieee.std_logic_1164.all
Entity FlipFlop is
 port(a,b :in  std_logic_vector(15 down to 0);
   equal :out  std_logic);
  end;
 Architecture result of FlipFlop is
  begin
  process(a,b)
  begin
  if a=b then equal <= '1' else
                 equal <= '0';
    end if;
    end process;
    end;
```

عبارت case در Process : عبارت case شبیه if بوده و وسیله‌ای برای تصمیم‌گیری و انشعابات شرطی است ولی نتیجه‌ی شرط if فقط در حالی که در مورد case این نتیجه می‌تواند integer ، std_logic یا هرچیز دیگری باشد در اینصورت نتیجه‌ی عبارت شرط برای هریک از مقدارهای شرط مقابل when نگاشته شده و اگر نتیجه‌ی شرط برابر بود آن عبارت اجرا می‌شود.

فرم کلی case بصورت زیری باشد.

case عبارت شرطی is

when مقدار شرط 1 => عبارت 1

when مقدار شرط 2 => عبارت 2

when others => عبارت n

end case;

نکته: عبارت null در VHDL یعنی کاری انجام نده و معمولاً در case استفاده می‌شود

45 Subject:
Date

مثال: برنامه‌ای VHDL ای و با استفاده از case , null برای یک طالتی پلکسر

۲ ورودی با ورودی‌های a و b ۴ بیتی و ورودی سلکتور sel و خروجی c بنویسید

```
library ieee;
use ieee.std_logic_1164.all
Entity mux is
 port (a,b :in std_logic_vector (3 down to 0);
       sel :in std_logic;
        c :out std_logic_vector(3 down to 0);
 end;

Architecture A of mux is
begin
 process (a,b,sel)
 begin
  case sel is

  when '0' => c<= a;
  when '1' => c <= b;
  when others null;
  end case;
 end process;
 end;
```

کاربرد متغیر (variable) در process: در توصیف dataflow برای نمایش اطلاعات معمولاً از سیگنال استفاده می شود. در process علاوه بر سیگنال برای نگهداری و ذخیره اطلاعات از variable استفاده می شود.

متغیر (variable) قبل از begin و در داخل process برای ذخیره اطلاعات در زمان اجرای process استفاده می شود.

در داخل process نمی توان port خروجی را خواند. برای این کار باید یک متغیر variable داخل process توصیف گردد. برای تخصیص مقدار به variable از =: استفاده می کنیم.

- تفاوت variable و signal در process: در process می توان signal یا variable استفاده نمود. اگر در process مقداری را با =: به variable اختصاص دهیم. در اینصورت بلافاصله variable مقدار را می گیرد. اما اگر با استفاده از <= مقداری به signal داده شود، بعد از آنکه تمام عبارات process اجرا شدند و process پایان یافت، مقدار مذکور به سیگنال داده می شود.

به عنوان مثال اگر درون process داشته باشیم:

~~signal x,y,z:bit~~ حذف می‌شود و تنها var باشد

```
process (y)
variable <x,z : bit>
  x<= y;
  z<= Not x;
end process;
```

حلقه‌ی loop: روشی برای تکرار اجرای عبارت است. سه فرم loop وجود دارد: 1) for-loop 2) while-loop 3) loop

for-loop: حلقه‌ی for-loop روشی برای تکرار قسمتی از کد در VHDL با تعداد معین است. فرم کلی آن به صورت زیر می‌باشد:

```
for i in 3 down to 0 loop;
```

در این صورت حلقه‌ی loop به مقدار i یعنی چهار بار و به ترتیب تا دیر ۰، ۱، ۲، ۳ برای i اجرا می‌گردد.

end loop; این برای پایان عبارت است.

مثال: برنامه ای بنویسید با استفاده از for_loop یک جمع کننده ۴ بیتی با جمع کننده FA تعریف نمایید

```
library ieee;
use ieee.std_logic_1164.all;
Entity FA is
port (a,b:in std_logic_vector (3 down to 0);
      cin:in std_logic;
      s : out std_logic_vector (3 down to 0);
     cout: out std_logic);
end;
Architecture result of FA is
begin
process (a,b)
variable c : std_logic_vector (4 down to 0)
begin
c(0) := cin;
for i in 0 to 3 loop;
sum (i) <= a (i) xoR b(i) xoR c(i);

c(i+1) := (a (i) and b(i)) oR (a (i) and c (i)) oR (b (i) and c(i)

end loop;
cout <= c(4);
end process;
end;
```

صفت ها : در VHDL تعدادی صفت تعریف شده است که به signal و متغیر نسبت داده می شود . صفت با علامت کوتیشن (') به متغیر یا signal داده می شود به عنوان مثال : اگر نوع type آرایه‌ای را به صورت زیر تعریف کنیم :

type my_array is array (-2 to 4) of integer;

در اینصورت صفت left به صورت زیر تعریف می شود .

signal a : my_array'Left;

عدد 2- یعنی عنصر طرف چپ آرایه‌ی My array به a تخصیص داده می شود

صفت Right : اگر سیگنال b را بصورت زیر تعریف کنیم

signal b: my_array'Right;

در اینصورت سیگنال b برابر عدد 4 یعنی سمت راست ترین سیگنال خواهد شد .

صفت high : اگر سیگنالی d به صورت زیر تعریف شده باشد ، در اینصورت عدد 2- یعنی کمترین مقدار به آن اختصاص می یابد

signal d: my_array'low;

صفت Length: اگر loop را به صورت زیر تعریف کنیم

```
for i in 0 to bin Length-1 loop
```

در اینصورت تعداد bit سیگنال bin مثلاً اگر 8 بیتی باشد، در اینصورت مقدار 7 برای loop استفاده شده و loop از 0 تا 7 عمل خواهد شد

عبارت exit در حلقه for-loop: در اجرای عبارات VHDL اگر بخواهیم به خارج از بروز قبل از تمام شدن loop می توانیم از عبارت exit استفاده کنیم

عبارت next در حلقه loop: این عبارت باعث می شود که از بعد از عبارت next صرف نظر شده و به ابتدای حلقه برگردد تا حلقه های بعدی اجرا شود.

حلقه while_loop: فرم کلی حلقه while--loop به صورت زیر می باشد

```
process
begin
   while(    شرط )loop
      ;
   end loop;
   end process;
```

حلقه‌ی loop ساده: حلقه‌ی loop ساده، معمولاً با عبارت exit استفاده می‌شود

و فرم کلی آن بصورت زیر است:

```
process
begin
loop
------ exit;
end loop;
end process;
```

تمرین: برنامه‌ی VHDL ای بنویسید که عناصر 4 آرایه‌ی 8 عنصری را با هم مقایسه کرده و در صورتی که 2 به 2 با هم برابر بودند، خروجی E (equal) برابر 1 کند.

```
library ieee;
use ieee.std_logic_1164.all;
Entity Array is
port(a,b,c,d:in std_logic_vector(7 down to 0);
          E :out std_logic);
end;
Architecture AR of Array is
signal equals: std_logic_vector(7 down to 0)
begin
   for i in 7 downto 0 loop
equals(i) <= a(i) xnor b(i) xnor c(i) xnor d(i);
end loop;
end process;
E <= equals(7) and equals(6) and equals(5) and equals(4) and equals(3)
        and equals(2) and equals(1) and equals(0);
end;
```

Subject :
Date

عبارت process بدون لیست sig nal و با عبارات wait until

wait on ، wait for ، wait ، ...

همانطورکه قبلاً دیدیم، process دارای یک لیست سیگنال حساسیت است که اگر هریک

از آنها تغییر نماید process اجرا می‌شود.

این حالت معمولی‌ترین روش توصیف فلیپ فلاپ دُبات و... است.

اما روش دیگر این است که process دارای لیست سیگنال نباشد؛ در اینصورت

عبارات process پشت سرهمدیگر اجرا می‌شوند تا به یک عبارت wait برسند.

عبارات wait until ، wait on : این عبارات به صورت زیر نوشته می‌شوند.

wait until (شرط)

① clock = '1' } حساس به سطح
② یا clock = '0'

③ یا (clock'event and clock = '1') حساس به لبه

process در برخورد به این عبارت آنقدر منتظر می‌ماند تا شرط مقابل این

true شود.

در عبارت ① process آنقدر منتظر می‌ماند تا clock 1 شود

- در عبارت (2) process آنقدر منتظر می ماند تا clock ۰ شود

- اما در عبارت سوم آنقدر منتظر می ماند تا لبه ی بالا رونده پالس ساعت بوجود آید

کاربرد constant: در VHDL زمانی که بخواهیم مقدار ثابتی را توصیف کنیم، ممکن است بخواهیم از عبارت constant به عنوان مقدار ثابت استفاده کنیم. این مقدار از ابتدا تا انتهای برنامه ثابت بوده و هیچ تغییری نمی کند

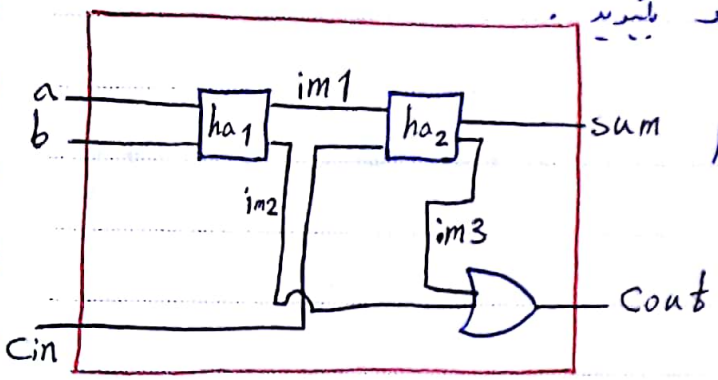
```
constant  P: time := 50 ns;
```

توصیف یا مدل سازی ساختاری (structual) یا سلسله مراتبی (hierarchy)

همانطور که یک مدار دیجیتال از چند IC (component) تشکیل شده است طراحی سخت افزار با روش ساختاری نیز از چند قطعه یا component تشکیل می شود. لذا مدارهای دیجیتال پیچیده را می توان به مجموعه ای از قطعات کوچکتر یا مدارهای سطح پایین تقسیم کرد و هر یک را به یک دونس اختصاص داد.

به عنوان مثال مدار را می توان با تعدادی component مانند گیت و فلیپ فلاپ توصیف نمود که اکثر آنرا net list می نامند

مثال: ساختار جمع کننده کامل زیر را در نظر بگیرید.



ha2, ha1 نیز جمع کننده component ۲ تا مطابق شکل درون آن است

```
library ieee;
use ieee.std_logic_1164.all;
Entity fa_1bit is
  port (a:in std_logic;
        b:in std_logic;
        Cin:in std_logic;
        sum:out std_logic;
        Cout:out std_logic;
end fa_1bit;
 Architecture fa_1bit_arch of fa_1bit is
① --component declaration,specifies component interface
 component ha_1bit port (x,y:in std_logic;
                         s,c:out std_logic);
 end component;
--signal declaration
 signal im1,im2,im3 :std_logic;
 begin
 ha1: ha_1bit port map (x=>a,y=>b, s=>im1, c=>im2);
 ha2: ha_2bit port map (x=>im1, y=>cin, s=>sum, c=>im3);
 cout <= im2 or im3;
 end fa_1bit_arch;
```

```
library ieee;
use ieee.std_logic_1164.all;
Entity ha_1bit is
port (x,y:in std_logic
      s,c:out std_logic);
end ha_1bit;

Architecture gate_level of ha_1bit is
begin
s<= x xor y;
c<= x and y;

end gate_level;
```

- در بخش ① اعلان component : در این قسمت قطعاتی که می خواهیم در مدار استفاده می کنیم. همراه interface (ورودی و خروجی) اعلام می کنیم.

نکته: نام ورودی و خروجی اعلان شده در component باید همان نام های استفاده شده در entity باشد.

نکته: در port map لیست عناصر با "," از یکدیگر جدا می شوند

برنامه تست برای مدارهای دیجیتال بصورت ساختاری (structral)

در تست مدارهای دیجیتال ۲ روش وجود دارد:

روش اول: که برای مدارهای کوچکتر استفاده می‌شوند. استفاده از کلیدهای دستی یا یک پالس ژنراتور در ورودی های مدارهای IC است و مشاهده خروجی با led یا اسیلوسکوپ

روش دوم: برای مدارهای پیچیده تر استفاده می‌کنیم، بدین صورت که تمامی ورودی مدار بصورت گرافیکی با یک ویرایشگر موج تولید می‌شود و همراه با اصل برنامه VHDL به شبیه ساز داده می‌شود. در این صورت شبیه ساز (simulator) ورودی ها را به مدار داده و خروجی ها را تهیه و رسم می‌کند.

نکته: برنامه تست در simulator باید به عنوان top level در نظر گرفته شود

(نمونه ای از برنامه test برای half_adder بعداً می‌آید)

نوع کاراکتر: نوع کاراکتر مانند زبان های برنامه نویسی در کد ASCII است لذا نمی توان با آنها عملیات محاسباتی انجام داد. کاراکتر در VHDL در یک تک کوتیشن (' - ') قرار می گیرد.

نوع رشته: آرایه ای از کاراکترهاست که مقدار آن در دبل کوتیشن (" ") قرار می گیرد

```
Constant P:string := "This is a test";
```

نکته: نوع کاراکتر و نوع string فقط در شبیه سازی و برنامه test استفاده می شود.

for...generate :

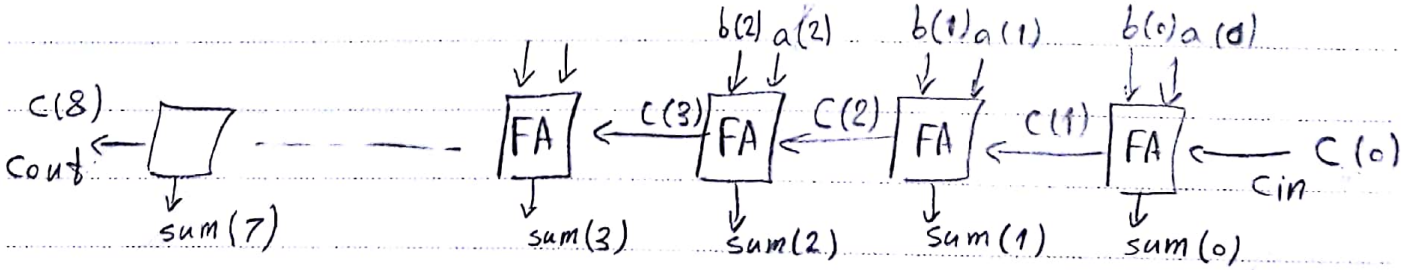
عبارت for...generate برای تکرار یا کپی کردن از سخت افزار است و بصورت همزمان اجرا می شود.

عبارات همزمان بین for...genelate و endgenerate نوشته می شود.

برای روشن شدن این مطلب یک مثال می زنیم:

تمرین: یک Full_adder 8 بیتی را با استفاده از for...generate طراحی کنید

```
library ieee;
use ieee.std_logic_1164.all;
Entity Add_8bit is
port (a,b: in std_logic_vector(7 down to 0);
      cin: in std_logic;
      Sum out std_logic_vector(7 down to 0);
      cout: out std_logic;
end;
Architecture Add_8bit_arch of Add_8bit is
 signal c: std_logic_vector(8 down to 0);
 begin
 c(0) <= Cin;
 gen: for i in 0 to 7 generate

 sum(i) <= a(i) xor b(i) xor c(i);
 c(i+1) <= (a(i) and b(i)) or (a(i) and c(i)) or (b(i) and c(i));
 end generate;
 cout <= c(8);
end;
```



مثال قبل بصورت 4 بیتی (component)

```
library ieee;
use ieee.std_logic_1164.all;
entity add_4bit is
port (a,b :in std_logic_vector (3 down to 0);
      Cin : in std_logic;
      sum: out std_logic_vector (3 down to 0);
      cout: out std_logic );
end;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity add_4 bit is
port (a, b :in std_logic_vector (3 down to 0);
      Cin : in std_logic;
      sum: out std_logic_vector (3 down to 0);
      cout: out std_logic);
end;
Architecture add_4bit-arch of add_4bit is
 signal c : std_logic_vector (3 down to 0);
component Full_adder
port (a,b, cin: in std_logic;
      sum, cout: out std_logic);
end component;
```

```
begin
u3 : full_adder portmap(a(3), b(3), c(2), s(3), c(3));
u2 : full_adder portmap (a(2), b(2), c(1), s(2), c(2));
u1 : full_adder portmap (a(1), b(1), c(0), s(1), c(1));
u0 : full_adder portmap ( a(0), b(0), cin , s(0), c(0));
 cout <= c(3);
end add_4bit_arch ;

library ieee
use ieee.std_logic_1164.all;

Entity Full_adder is
port (a,b, cin : in std_logic;
      sum, cout : out std_logic);
end;
 architecture behavior of full_adder is
 begin
 sum <= a xor b xor c;
cout <= (a and b) or (a and cin) or (b and cin);
end;
```

برنامه تست :

```
entity tb is
end tb;
Architecture behv of tb is
--Declarations
--inputs of ha declared signals
signal a: bit;
signal b: bit;
signal s: bit;
signal co: bit;
-- main circuit as component
component ha
port (a,b: in bit;
      s,co: out bit);
end component;
--end of Declarations
begin
--main circuits ha is instanstiated
U1: ha port map (a => a, b => b, s => s, co => co);
        a <= '0'
             '1' after 20 ns,
             '0' after 30 ns,
             '1' after 40 ns,
             '0' after 50 ns;
        b <= '0'
             '1' after 10 ns;
             '0' after 20 ns,
             '1' after 30 ns;
end;
```

عبارت generic در entity : عبارت generic برای تعریف constant با نوع integer در entity برای تعداد بیت سیگنال و یا نوع time برای تأخیر سیگنال استفاده می شود.

فرق for--loop با for--generate : for--generate مثل for--loop است ولی عبارت for--loop پشت سرهم (sequential) و داخل process اجرا می شوند، درصورتی که عبارات for--generate به صورت همزمان (concurrent) اجرا می گردند.

فصل سوم

انواع سیگنال در VHDL و کاربرد آنها در ماشین حالت:

نوع آرایه: آرایه مجموعه‌ای از عناصر از یک نوع می باشد. که هر عنصر با index آن که متقابل آن قرار می گیرد، مشخص می شود. به عنوان مثال:

```
signal c: std_logic_vector (6 down to 0);
```

index در آرایه به صورت رقم integer در محدوده‌ی آرایه می باشد.

نوع bit_vector : آرایه‌ای از bit

نوع stol_logic_vector : آرایه‌ای از stol_logic

نوع string : آرایه‌ای از character می‌باشد.

آرایه ممکن است محدود باشد، یعنی نوع عناصر آن مشخص و محدود باشد، همچنین ممکن است آرایه نامحدود باشد، یعنی اندازه و یا محدوده (range) آن تعیین نشده باشد.

به عنوان مثال آرایه‌های bit_vector زیر را در نظر بگیرید.

```
type bit is array (integer < >) of bit;
type bit is array (natural < >) of bit;
type bit is array (positive < >) of bit;
type bit is array (negative < >) of bit;
```

مثلاً عبارت 1 به صورت آرایه‌ای نامحدود با index از -2,147,483,648 تا +2,147,483,647 است

عبارت 2 آرایه‌ای نامحدود با index از 0 تا +2,147,483,647 است

عبارت 3 : آرایه ای نامحدود با index از 1 تا +2,147,483,647 است

عبارت 4 : آرایه ای نامحدود با index از منفی -2,147,483,64 تا -1 تعریف می شود

نکته : عناصر آرایه همیشه از چپ به راست درنظر گرفته می شود

آرایه با 2 index : آرایه می تواند دارای 2 index باشد به عبارت دیگر آرایه ای از یک آرایه باشد .

```
type Ali is array (Integer < > ) of
    std_logic_vector (7 down to 0);
```

یعنی آرایه ی Ali با range نامحدود است و هر عنصر آرایه ، آرایه ای از 8 بیت است

به عنوان مثالی دیگر آرایه ی Reza

```
type Reza is array (15 down to 0) of
    std_logic_vector (7down to 0);
```

یعنی یک آرایه ی 16 عنصره با نام Reza که هر عنصر 8 بیت است .

نوع تعریف شده توسط کاربر: ما میتوانیم نوع جدیدی را با استفاده از فرم زیر به عنوان مثال تعریف کنیم.

```
type semeghdar is ('1', '0', 'z');
```

در اینصورت نوع semeghdar شامل سه مقدار 0 ، 1 و z است.

زیر نوع (sub type): زیر نوع، نوع محدود شده یا تعداد عناصر کمتر آن نوع است.

به عنوان مثال زیر نوع one_byte

```
subtype one-byte. is integer range 0 to 255;
```

عبارت Alias: این عبارت برای نام دادن به قسمتی از سیگنال استفاده می شود.

در اینصورت می توان سیگنال های بزرگ را به قسمت های ساده تر تقسیم کرد و برای آنها دسترسی مستقیم پیدا نمود.

نوع کلی: `signal data is bit_vector (9 down to 0);`

```
Alias startbit: bit is data(1)

start bit <= '1';

data(1) <= '1';
```

در واقع

نوع Record type : مجموعه ای از عناصر است که هر عنصر می تواند دارای نوع تخصصی باشد. به عنوان مثال Record data_date زیر را در نظر بگیرید.

```
type data_date is record
year : Integer range 1900 to 2019;
day : Integer range 1 to 31;
month : Integer range 1 to 12;
data1 : std_logic_vector (3 down to 0);
end record;
signal d:data_date;
d.year <= 2018;
```

در قسمت اعلانات تعریف می شود.

مقدار دهی به سال

نوع Enumerated : ابزاری بسیار قوی برای استفاده در مدارهای دیجیتال است که می توان با استفاده از آن، مدل سازی مورد نظر را انجام داده این نوع شامل لیستی از نام هاست که به عنوان مثال در ماشین حالت، نوع و نام حالت را می توان مانند زیر تعریف نمود:

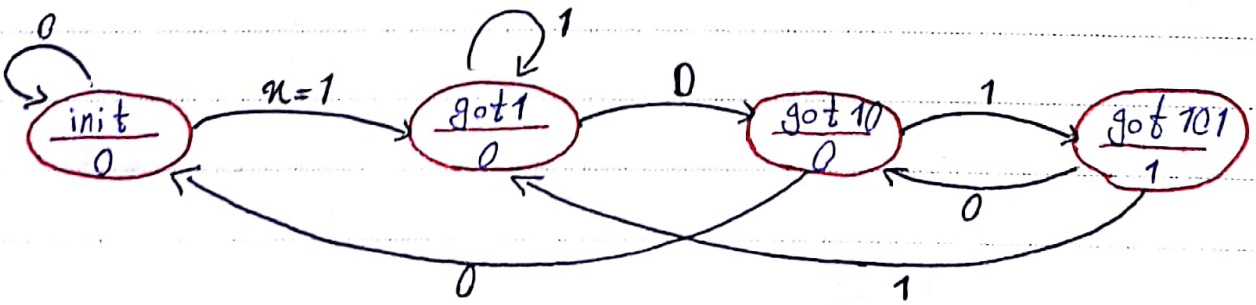
اگر مقدار ندهیم طبق default اولی را می‌برد

```
type states is (S0, S1, S2);

signal s: states;
```

نکته: در طراحی ماشین حالت باید نوع آن در قسمت اعلانات Architecture توصیف شود. علاوه بر این signal ای مثل state، next state، current state برای قرار دادن حالت مدار، در آن تعریف می‌شود.

برنامه VHDL ماشین حالت با استفاده از نوع Enumerated:

مثال الف) برنامه VHDL ای برای ماشین حالت moore بنویسید که اگر در ورودی x به ترتیب مقدار 101 آمد خروجی g برابر 1 شود.



```
library ieee;
use ieee.std_logic_1164.all;

Entity detector is
        port (x, clk : in std_logic;
                    g : out std_logic);
  end;

Architecture behavior of detector is
-- decleration
type states is (init, got1, got10, got101);
  signal current : states;
begin
  clock : process (clk)
label ↙
    begin
        if (clk'event and clk='1') then
                case current is

                    when init => if (x = '0') then
                                    current <= init;
                                    else
                                     current <= got1;
                                    end if;
```

Subject :
Date

```
when got1 => if (x = '1') then
                current <= got1;
             else
                current <= got10;
             end if;

when got10 => if(x = '1') then
                current <= got101;
              else
                current <= init;
              end if;

when got101 => if (x = '1') then
                 current <= got1;
               else
                 current <= got10;
               end if;
        end case;
        end if;
        end process;

lable <- output : process(current)
         begin
         Case current
```

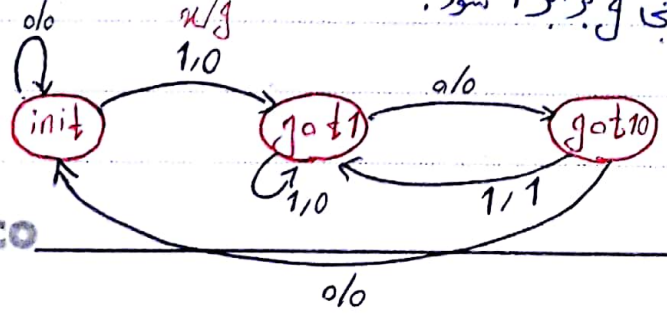
```
when init => g<='0';
when got1 => g<='0';
when got10 => g<='0';
when got101 => g<='1';

end case;
end process;
end Architecture;
```

توضیح ماشین mealy و moore:

ماشین حالت برای کنترل سیستم های دیجیتال و کامپیوتر بکار می رود. اصولاً دو نوع ماشین حالت داریم. ماشین حالت moore و mealy در ماشین حالت moore خروجی فقط تابع حالت فعلی مدارات. اما در ماشین mealy خروجی تابع حالت فعلی و ورودی های مدارات

نکته: process های clock و current به طور همزمان اجرا می شوند.

مثال: برنامه‌ی VHDL برای ماشین حالت mealy بنویسید که اگر در ورودی x به ترتیب مقدار 1، 0، 1 آمد، خروجی g برابر 1 شود.



```
library ieee;
use ieee.std_logic_1164.all;

entity mealy_detector is

port (x, clk : in bit;
      g : out bit;)

end mealy_detector;

Architecture data of mealy_detector is
        type state is (init, got 1, got 10);
signal current : state;
Begin
process (clk);
Begin
if (clk'event and clk = '1');
    case (current) is
when init => (if x = '0') then current <= init;
 else current <= got 1;
when got 10 => (if x = '0') current <= init;
else current <= got 1;

end case
end if;
end process;
```

```
output : process(current)
Begin
Case current
when init => if (x='0') g<=0;
             else       g<=0;
when got1 => if (x='0') g<=0;
             else       g<=0;
when got10 => if (x='1') g<=1;
              else       g<=0;
end case;
end process;
end Architecture;
```

نوع Real type : مقدار signal نوع Real دارای عدد صحیح و ۰ اعشاری می باشد. برای اعداد بزرگ در Real می توان از نوع صحیح نماد E نیز استفاده نمود

-1.6 E38
-1.6×10^{38}

```
signal A: real := 5.0
```

فصل ۴ توابع و پکیج

کتابخانه در VHDL محلی است که اطلاعاتی راجع به یک پروژه یا طرح را در آن ذخیره می کند.

پکیج فایلی است که محل اعلان نوع، قطعه component، constant، تابع و ... است و برنامه های مختلف VHDL می توانند از آنها استفاده کنند

- کلمه use برای استفاده از پکیج است و کلمه all یعنی از تمامی امکانات پکیج استفاده کند. در کتابخانه IEEE پکیج های دیگری نیز به شرح زیر قرار دارد.

الف) پکیج محاسباتی: در پکیج std_logic_arith عملیات ریاضی مانند جمع، تفریق، ضرب و... و همچنین امکان تبدیل نوع ها برای ما فراهم است.

ب) پکیج std_logic_unsigned ← این پکیج، مشابه بند (الف) ولی برای اعداد بدون علامت

ج) پکیج std_logic_signed ← این پکیج نیز مشابه بند (الف) است ولی برای اعداد با علامت

پکیج برای قراردادن عناصری می باشد، که برنامه های دیگر نیز بتوانند از آنها به طور مشترک استفاده کنند. به عبارت دیگر به جای اینکه عناصر را در هر برنامه‌ی VHDL قرار دهیم؛ یکبار آنها را تعریف کرده و در برنامه های دیگر در صورت لزوم از آنها استفاده می کنیم

مثال: پکیجی به نام utility 2 تعریف نمایید که شامل الف) یک نوع به نام tshift با مقادیر shiftpass، shL، shr، rotL و rotr می باشد.

```
library ieee;
use ieee.std_logic_1164.all

→ package utility2 is
        (اسم دلخواه)
type tshift is (shiftpass, shL, shr, rotL, rotr);

end utility2;
```

```
library ieee;
use ieee.std_logic_1164.all;
use work.utility2.all;
```

- برای استفاده از عناصر یک پکیج در برنامه های VHDL دیگر با عبارت use.work.نام پکیج.all استفاده می کنیم

- all به معنی استفاده از تمامی امکانات پکیج است.

- اگر قرار باشد عملیات در بدنه انجام شود از

```
Package body utility2 is
end;
```

استفاده می کنیم.

تابع Function: برای اجرای یک کار خاص می توانیم از تابع که دارای ورودی یا ورودی های مختلف است و همچنین دارای خروجی است استفاده کنیم.

مثال: تابع carry برای جمع کننده بصورت زیر است.

```
Function carry (bit1, bit2, bit3 : in std_logic)
return std_logic is
variable result : std_logic;
begin
result := (bit1 And bit2) OR
          (bit2 And bit3) OR
          (bit1 And bit3);
return result;
end carry;
```

نام اختیاری

متغیر های ورودی

carry سه ورودی و یک خروجی

نام متغیر خروجی

تعریف و عملیات

نکته: هر تابع باید با return , end پایان یابد

نکته: پارامترهای تابع ورودی آن است و هر تابع فقط یک خروجی دارد

نحوه نحوه گیری در تابع Architecture برنامه VHDL:

```
Entity  Ar  is
    port (a,b,c :in  std_logic;
          cout :out std_logic;
  end entity;
  Architecture
  function carry
    |
    |
    |
    end carry
begin
  cout <= carry (a,b,c);
  end Architecture;
```

تعریف تابع در پکیج و فراخوانی آن در برنامه دیگر:

تابع را می‌توان در یک پکیج توصیف نمود و در برنامه‌ی VHDL با استفاده از use و نام پکیج آنرا کاربرد. به عنوان مثال: تابع carry را می‌توانیم در پکیجی به نام My package قرار دهیم و در یک برنامه‌ی VHDL آنرا فراخوانی کنیم

```
library ieee;
use ieee.std_logic_1164.all;
package my_pack is
--declaring Function carry
function carry (bit1,bit2,bit3 :in std_logic )
  return std_logic;
end;
package body my_pack is
--defining function carry
function carry (bit1, bit2, bit3 : in std_logic)
   return std_logic is
variable result : std_logic;
begin
result :=(bit1 And bit2) oR (bit1 And bit3) oR (bit2 And bit3);
return result;
end carry;
end my_pack;
```

روال یا Procedure : یک برنامه فرعی، شبیه تابع function است که برای اجرای محاسبات و روتین های تکراری به کار می رود. Procedure مانند function در آرشیتکت Package اعلان می شود

و یا

پارامترهای procedure می توانند ورودی، خروجی یا دوسویه باشند. علاوه بر این procedure

از سیگنال in out استفاده کنید

می توانند هر تعداد باشد، در صورتی که در function فقط یک متغیر بصورت return در procedure خروجی می تواند

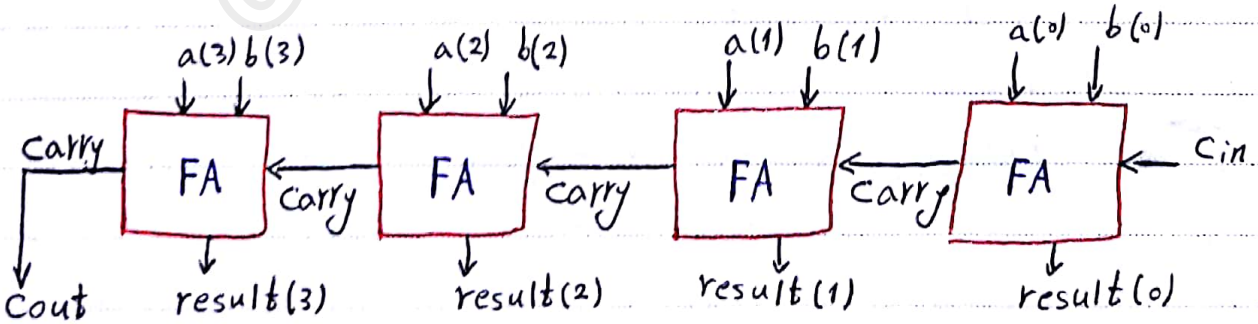
خارج می شود. به عنوان مثال یک procedure جمع کننده کامل سه ورودی را بنویسید.

```
procedure full_adder (a,b,c : in std_logic;
                      sum,cout:out std_logic) is
    begin
    sum := a xor b xor c;
    cout := (a and b) or (b and c) or (c And a);
end full_adder;
```

مثال: با استفاده از procedure جمع کننده ی کاملی بنویسید که دارای سه ورودی a و b و c و دو خروجی cout و sum باشد و آرشیتکت برنامه یک جمع کننده 4 بیتی را بنویسید.

```
library ieee;
use ieee.std_logic_1164.all;
entity procedure_fa is
 port (a,b : in std_logic_vector (3 down to 0);
       Cin : in std_logic;
       sum: out std_logic_vector(3 downto 0);
       cout: out std_logic);
 end entity;
Architecture proc of procedure_fa is
-- declaring procedure of Full Add 1-bit.
procedure full_adder(a,b,c:in std_logic;
                     sum,cout:out std_logic) is
```

```
begin
sum := a xoR b xoR c;
cout := (a and b) oR (b And c) oR (c And a);
end full_adder;
-- end declaring
begin
process (a,b,cin)
variable result : std_logic_vector (3 down to 0);
variable carry : std_logic;
begin
full_adder (a(0), b(0), cin, result(0), carry);
full_adder (a(1), b(1), carry, result(1), carry);
full_adder (a(2), b(2), carry, result(2), carry);
sum <= result; full_adder (a(3), b(3), carry, result(3), carry);
cout <= carry;
end process;
end architecture;
```



مثال: برنامه VHDL برای یک mux 2×1 بنویسید.

```
library ieee;
use ieee.std_logic_1164.all;
Entity mux is
port (a,b:in std_logic;
      o:out std_logic);
end entity;
Architecture Arch_mux of mux is
begin
 o <= a when s='0'
       else b;
end Architecture;

Entity testbench is
end testbench;

Architecture tb of testbench is
  signal a,b,s,o: bit;
component mux
        port(a,b,s:in std_logic;
             o:out std_logic);
end component;
begin
mux portmap (a=>a, b=>b, s=>s, o=>o)
```

```
a <= '0'
     '1' after 10 ns,
     '0' after 20 ns,
     '1' after 30 ns;

b <= '0'
     '1' after 10ns,
     '0' after 20ns,
     '1' after 30 ns;

end;
```