***Câu 2: Cấu trúc chung của CPLD và FPGA? Khái niệm về Logic cell, CLB, I/O block, PSM?***

CPLD và FPGA là các vi mạch dùng cấu trúc mảng phần tử logic mà người dùng có thể lập trình được. Cấu trúc của vi mạch CPLD và FPGA là tương tự nhau, bao gồm:

+ Các [khối logic](http://vi.wikipedia.org/w/index.php?title=Kh%E1%BB%91i_logic&action=edit&redlink=1) có khả năng lập trình được (*configurable logic block “CLB”* )

+ Hệ thống mạch liên kết lập trình được

+ Khối vào/ra (*IO Pads*)

+ Phần tử thiết kế sẵn khác như [DSP slice](http://vi.wikipedia.org/w/index.php?title=DSP_slice&action=edit&redlink=1), [RAM](http://vi.wikipedia.org/wiki/RAM), [ROM](http://vi.wikipedia.org/wiki/ROM), nhân vi xử lý...

FPGA và CPLD đều là linh kiện logic lập trình được nhưng chúng rất khác nhau. Những điểm khác biệt đó là:

FPGA và CPLD đều cấu tạo từ các cell-logic là sự kết hợp của một khối logic và Flip-Flop. Nhưng, FPGA là linh kiện “fine-grain”, nghĩa là mật độ các cell logic lớn (lên đến cả trăm ngàn) vì kích thước các khối lôgic nhỏ. CPLD là linh kiện "coarse-grain", nghĩa là mật độ các cell logic rất ít (xét trên cùng diện tích với FPGA) vì kích thước các khối lôgic lớn.

FPGA giống như RAM, phải nạp lại mỗi khi bật nguồn. CPLD giống như EEPROM chỉ cần nạp một lần và không bị mất chức năng sau khi ngắt nguồn.

FPGA có nhiều tài nguyên định tuyến đặc biệt để thực hiện các chức năng toán học như các bộ đếm, bộ cộng, bộ so sánh, v.v… còn CPLD thì không.

\* Logic Cells (Các tế bào logic): được tích hợp trong vi mạch nằm trong FPGA (hoặc CPLD) dùng để thực hiện các mạch logic và được kết nối với nhau bởi ma trận kết nối và chuyển mạch lập trình được. Nói cách khác, chúng là một chip silicon chứa trong đó rất nhiều “khối logic”, có thể tái cấu hình CLB (Configurable Logic Blocks) được liên kết với nhau thành các liên kết khả trình (Programable interconner). Các khối vào ra(I/O block) được phân bố xung quang chip tạo thành các liên kết bên ngoài. Bên trong khối logic CLB có bảng LUT(Look-Up table) và các phần tử nhớ (Flip Flop hoặc bộ chốt).

***Câu 3 : Nguyên tắc thiết kế một mạch điện tử là gì? Việc phân chia thiết kế (Partitioning a design) dựa trên những cân nhắc gì?***

\* Nhìn chung, nguyên tắc và các bước thiết kế của FPGA và CPLD là giống nhau, người thiết kế có thể nhập vào một thiết kế dưới dạng HDL, VHDL,…

Một thiết kế có thể bao gồm cả dưới dạng sơ đồ và HDL, phần mềm ISE được kết hợp với bộ mô phỏng MXE đưa ra khả năng mô phỏng và kiểm tra chức năng của VHDL. Bộ mô phỏng MXE đưa ra một biểu đồ kiểm tra, nó cho phép người dùng đưa ra các mẫu thử test các chức năng trong quá trình tổng hợp .

* Với FPGA quá trình thiết kế gồm 4 bước cơ bản;

+ Translate- dịch thiết kế và cho chạy kiểm tra theo quy luật thiết kế

+ Map- Tính toán và cấp phát tài nguyên trong thiết bị đích

+ Place and Route - Sắp đặt các khối logic, định cấu hình phù hợp với các vị trí logic và sử dụng các tài nguyên định tuyến.

+ Generate Programming file - tạo ra dòng các bit chương trình

* Với CPLD: quá trình thực hiện bao gồm 3 bước cơ bản;

+ Translate- dịch thiết kế và cho chạy kiểm tra theo quy luật thiết kế

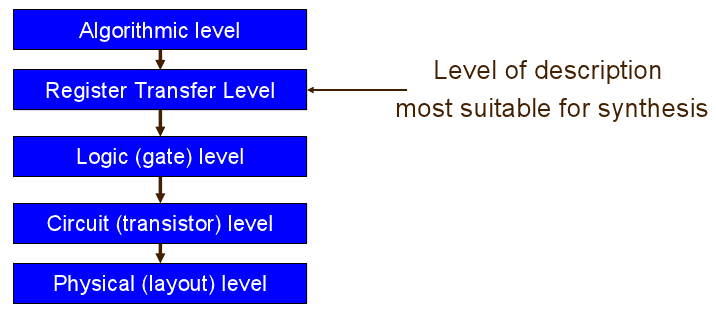
+ Fit – Cấp phát tài nguyên và kết nối

+ Gennerate programming file – Tạo fle JED cho chương trình.

\*)Việc phân chia thiết kế dựa trên những nguyên tắc: ?

***Câu 4 : Các mức mô tả một đơn vị thiết kế là những mức gì? Mức nào phù hợp với việc tổng hợp (synthesis) mạch logic số?***

Các mức mô tả một đơn vị thiết kế là:



+ Mức thuật toán

+ Mức thanh ghi truyền/chuyển (**mức này thích hợp cho tổng hợp**)

+ Mức Logic (cổng)

+ Mức mạch (transistor)

+ Mức vật lý

***Câu 5: Qui trình thiết kế mạch điện tử số trên CPLD và FPGA? Mô tả các bước thiết kế? Kết quả của từng giai đoạn trong qui trình thiết kế?***

\* Quy trình thiết kế mạch điện tử số trên CPLD và FPGA:



Phần mềm thiết kế ISE của Xilinx: ISE (Integrated Soft Enviroment – Môi trường phần mềm tích hợp) là công cụ thiết kế tổng thể, bao hàm các công cụ phần mềm thiết kế chuyên dụng khác nhau, là công cụ sử dụng nhiều nhất trong thiết kế các PLD (Programmable Logic Device) của Xilinx. Các bước tiến hành thiết kế cho FPGA và CPLD là giống nhau, có thể nhập vào dưới dạng một sơ đồ hoặc dạng HDL (VHDL, Verilog hoặc ABEL). Phần mềm ISE được kết hợp với bộ mô phỏng MXE, đưa ra một biểu đồ kiểm tra và cho phép đưa ra các mẫu thử để test các chức năng trong quá trình tổng hợp (TestBencher).

\* Với FPGA: Gồm 4 bước

Bước 1: Translate - Dịch thiết kế và cho chạy kiểm tra theo quy luật thiết kế.

Bước 2: Map - Tính toán và cấp phát tài nguyên trong thiết bị đích.

Bước 3: Place anh Route - Sắp đặt các khối logic, định cấu hình phù hợp với các giá trị logic và sử dụng tài nguyên định tuyến.

Bước 4: Generate Programmable File - tạo ra dòng các bit chương trình (Tạo file.Bit)

\* Với CPLD: Gồm 3 bước

Bước 1: Translate - Dịch thiết kế và cho chạy kiểm tra theo quy luật thiết kế.

Bước 2: Fit - Cấp phát tài nguyên và kết nối.

Bước 3: Generate Programmable File - Tạo file JED cho chương trình.

***Câu 6: Mô hình hóa các mạch (đơn vị) phần cứng sử dụng ngôn ngữ gì? Kể tên các ngôn ngữ mà đồng chí biết? Ngôn ngữ nào được chấp nhận là chuẩn IEEE (Institute of Electrical and Electronics Engineers)?***

+ Để mô hình hoá các mạch (đơn vị ) phần cứng có thể sử dụng các ngữ VHDL, Verilog, ABEL. Trong đó tôi được học và sử dụng ngôn ngữ VHDL. Ngôn ngữ VHDL là chuẩn hoá, độc lập với các hãng sản xuất, có tính linh hoạt cao và khả năng tái sử dụng.

Hai ứng dụng chính của VHDL là trên các thiết bị PLD (CPLD và FPGA) và trên các thiết bị ASIC.

Với chương trình VHDL, chúng có thể nạp trên các chip của các hãng (Xilinx, Altera…) cũng như trong việc sản xuất chip ASIC. Hiện nay, nhiều chip thương mại phức tạp (Vi điều khiển,….) được thiết kế theo dạng này.

+ Trong các ngôn ngữ VHDL, Verilog, ABEL thì ngôn ngữ VHDL được chấp nhận là chuẩn IEEE (Institute of Electrical and Electronics Engineers**)**

***Câu 7: Có phải tất cá các cấu trúc mô hình hóa bằng VHDL đều có thể tổng hợp được không? Có thể cho ví dụ?***

Ngôn ngữ VHDL dùng để mô phỏng và tổng hợp mạch số. Tuy nhiên, ngôn ngữ VHDL có thể hoàn toàn mô phỏng được các cấu trúc, xong không phải tất cả các cấu trúc mô hình hoá bằng VHDL đều có thể tổng hợp được (có những cấu trúc, những vấn đề mô phỏng được bằng ngôn ngữ VHDL nhưng lại không thể dùng ngôn ngữ VHDL để tổng hợp chúng ).

***Câu 8 : Thế nào là câu lệnh đồng thời và câu lệnh tuần tự? Mạch tổ hợp có thể mô hình hóa bằng những câu lệnh nào? Có thể mô hình hóa mạch tuần tự chỉ dùng câu lệnh đồng thời không?***

**Câu lệnh đồng thời:**

Các phát biểu đồng thời được thực hiện song song cùng thời điểm mô phỏng, chúng không thực hiện theo thứ tự mà chúng được viết ra trong một kiến trúc. Các phát biểu đồng thời chuyển thông tin thông qua các đường tín hiệu.

**Câu lệnh tuần tự:**

Phát biểu tuần tự chỉ ra sự thực hiện từng bước của một quá trình. Chúng thực hiện từ câu lệnh đầu tiên, câu lệnh thứa hai, … câu lệnh cuối cùng. Các phát biểu nằm trong một phát biểu quá trình (phát biểu **process**) được gọi là phát biểu tuần tự.

Mạch tổ hợp có thể được xây dựng bằng cả câu lệnh tuần tự lẫn câu lệnh đồng thời.

Có thể mô hình hóa mạch tuần tự bằng câu lệnh đồng thời.

***Câu 9 : Các cấu trúc cơ bản của ngôn ngữ VHDL: Entity, Architecture, Package, Library?***

CẤU TRÚC CỦA MỘT THIẾT KẾ DÙNG NGÔN NGỮ VHDL.

---------------------------------

-- Ghi chú

---------------------------------

PACKAGE (Tùy chọn)

library (Thư viện)

use

ENTITY (Bắt buộc)

ARCHITECTURE (Bắt buộc)

CONFIGURATIONS (Tùy chọn)

**1. PACKAGE (KHỐI).**

Package là một vùng lưu trữ các dữ liệu để dùng chung cho các entity.

Mô tả dữ liệu bên trong một package cho phép được tham khảo bởi một entity khác, vì vậy dữ liệu có thể được dùng chung.

Một package bao gồm 2 phần: phần mô tả định nghĩa giao diện cho package, phần thân ấn định các hoạt động cụ thể của package.

Cú pháp khai báo khối được xác định là:

PACKAGE example\_arithmetic IS

-- các khai báo khối có thể chứa các khai báo sau:

- Subprogram declaration.

- Type, subtype declaration.

- Constant, deferred constant declaration.

- Signal declaration creates a global signal.

- File declaration.

- Alias declaration.

- Attribute declaration, a user-define attribute.

- Attribute specification.

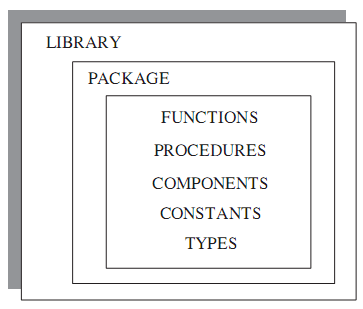
- Use clause.

END example\_arithmetic;

Tất cả các đối tượng khai báo trong package có thể được truy xuất bởi bất kỳ một thiết kế nào đó bằng cách sử dụng mệnh đề use và khai báo library.

* 1. **THƯ VIỆN**

A LIBRARY là một tập các đoạn mã thường được sử dụng. Đặt các đoạn mã thường sử dụng vào thư viện cho phép chúng có thể được tái sử dụng hoặc chia sẻ giữa các thiết kế khác nhau.



library my\_lib;

use my\_lib.example\_arithmetic.all;

**Một số thư viện chuẩn**

* Thư viện IEEE:
* std\_logic (8 mức logic),std\_ulogic (9 mức logic)
* Kiểu std\_logic có thể có các giá trị
* U Uninitialized
* X Unknown
* 0 Zero
* 1 One
* Z Tristate (Must be upper case!)
* W Weak unknown
* L Weak Zero
* H Weak One
* - Don't care
* std\_logic\_arith : Thực hiện các phép toán số học và so sánh.
* std\_logic\_signed : Thực hiện các phép toán với kiểu DL std\_logic\_vector, dữ liệu coi là có dấu
* std\_logic\_unsigned Thực hiện các phép toán với kiểu DL std\_logic\_vector, dữ liệu coi là không dấu.
* std\_logic\_1164: là thư viện logic chuẩn của IEEE (đoạn 1164), mục đích cung cấp các chuẩn cơ bản để có thể mô tả các kiểu dữ liệu kết nối trong VHDL.

Ngoài ra có thể tự tạo riêng các thư viện trong thiết kế.

**2. ENTITY**

Khai báo entity chỉ dùng để mô tả ngõ vào và ngõ ra của một thiết kế. Mức cao nhất của bất kỳ một thiết kế VHDL là một khai báo entity đơn, khi thiết kế trong VHDL, tên của file lưu trữ phải trùng với tên theo sau của từ khóa entity.Trong khai báo ENTITY, có 4 kiểu tín hiệu khác nhau:

• IN: mô tả các ngõ vào entity. Kiểu IN thì được sử dụng cho các ngõ vào clock, các ngõ vào điều khiển,…

• OUT: mô tả dòng dữ liệu đi ra khỏi entity, entity sẽ không thể đọc các tín hiệu này, kiểu OUT chỉ được sử dụng khi tín hiệu không được sử dụng bởi bất kỳ kiểu nào trong entity.

• BUFFER: Kiểu tín hiệu này mô tả dòng dữ liệu đi ra khỏi entity, nhưng entity có thể đọc những tín hiệu này (mục đích đọc lại tín hiệu ngõ ra ở bên trong của cấu trúc). Tuy nhiên , tín hiệu sẽ không thể điều khiển từ ngõ ra của entity, vì vậy nó không thể được sử dụng cho các dữ liệu ngõ vào.

• INOUT: Kiểu tín hiệu này cho phép tín hiệu có thể có cả hai kiểu: vào và ra, khi khai báo tín hiệu theo kiểu INOUT thì tín hiệu có thể được điều khiển từ ngõ ra của entity. Kiểu tín hiệu này chỉ nên sử dụng khi thật cần thiết (ví dụ như bus dữ liệu 3 trạng thái), và khi sử dụng kiểu tín hiệu này thì mã của chương trình sẽ trở nên khó hiểu hơn đối với người thiết kế.

Loại tín hiệu cũng phải được mô tả trong khai báo PORT, loại tín hiệu sẽ mô tả các giá trị mà tín hiệu đó có thể được ấn định, ngoài ra có thể mô tả một vector những tín hiệu có cùng loại.

**3. ARCHITECTURE.**

Chức năng của architecture là mô tả mối liên hệ giữa các tín hiệu ngõ vào và tín hiệu ngõ ra (bao gồm cả những tín hiệu BUFFER), có thể viết nhiều kiến trúc khác nhau trong một entity, nhưng chỉ một trong số đó có khả năng xuất hiện trong mã VHDL. architechture có 3 dạng: mô tả cấu trúc (structure); mô tả dòng dữ liệu (data flow); mô tả hành vi (behavioral)

Mô tả architechture dưới dạng cấu trúc (structure)

Ví dụ, chương trình bộ cộng trên sẽ có dạng:

ENTITY HALF\_ADDER IS

PORT( A,B : IN BIT;

SUM, CARRY : OUT BIT);

END HALF\_ADDER;

ARCHITECTURE HA\_STRUCTURE OF HALF\_ADDER IS

Component XOR2

Port(X,Y: in BIT; Z: out BIT);

End component

Component AND2

Port(L,M: in BIT; N: out BIT);

End component

BEGIN

X1: XOR2 port map (A,B,SUM);

A1: AND2 port map (A,B,CARRY);

END HA\_STRUCTURE;

Mô tả architechture dưới dạng dòng dữ liệu (data flow)

Ví dụ, chương trình bộ cộng trên sẽ có dạng:

ENTITY HALF\_ADDER IS

PORT( A,B : IN BIT;

SUM,CARRY : OUT BIT);

END HALF\_ADDER;

ARCHITECTURE HA\_STRUCTURE OF HALF\_ADDER IS

BEGIN

SUM<=A xor B;

CARRY<=A and B;

END HA\_STRUCTURE;

Mô hình dòng dữ liệu sử dụng phát biểu gán tín hiệu đồng thời, ký hiệu <= chỉ giá trị được gán cho tín hiệu. Phép gán được thực hiện khi có 1 sự kiện tín hiệu của biểu thức bên phải.

Mô tả architechture dưới dạng hành vi (behavioral)

Kiểu mô tả hành vi bao gồm tập hợp thứ tự các phép gán tín hiệu tuần tự được khai báo

bên trong phát biểu process.

Biến được khai báo trong process là biến cục bộ. Tín hiệu không được khai báo trong process

Ví dụ, chương trình bộ cộng trên sẽ có dạng:

ENTITY HALF\_ADDER IS

PORT(A,B : IN BIT;

SUM,CARRY : OUT BIT);

END HALF\_ADDER;

ARCHITECTURE HA\_STRUCTURE OF HALF\_ADDER IS

BEGIN

Process(A,B);

Begin

SUM<=A xor B;

CARRY<=A and B;

End process;

END HA\_STRUCTURE;

Mô tả architechture dưới dạng hỗn hợp (mixed style)

Có thể trộn lẫn 3 kiểu trong một architechture

Ví dụ, mô tả mạch cộng toàn phần (FULL\_ADDER) sau

ENTITY FULL\_ADDER IS

PORT( A,B,CIN : IN BIT;

SUM, COUT : OUT BIT);

END DFF;

ARCHITECTURE FA\_MIXED OF FULL\_ADDER IS

Component XOR2

Port(X,Y: in BIT; Z: out BIT);

End component

Signal S1: BIT --Khai báo tín hiệu cục bộ trong architechture

BEGIN

X1: XOR2 port map (A,B,S1); --structure

Process(A,B,CIN); --behavior

Variable T1,T2,T3: BIT;

Begin

T1:=A and B;

T2:=A and CIN;

T3:=B and CIN;

COUT<=T1 or T2 or T3;

End process;

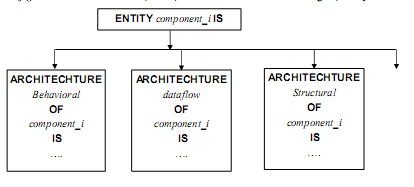
SUM<=S1 xor CIN; --dataflow

END FULL\_ADDER;

**4. CONFIGURATION**

Configuration dùng để liên kết các hợp phần tới entity

configuration ban đầu có thể được ấn định tới rất nhiều architecture trong một entity.



Định cấu hình cho phép mô phỏng các thiết kế ở thành phần con, dễ dàng kiểm tra được thiết kế hơn một chương trình lớn.

Một cấu hình được dùng để nối từng cặp như sau:

• Một thân architecture tới khai báo entity của nó.

• Một component với một entity.

Ví dụ, xây dựng một entity FULL\_ADDER ở trên có thể tách ra làm ba architecture

body: FA\_BEHAVIOR, FA\_STRUCTURE, và FA\_MIXED. Một architecture bất kỳ có thể được chọn bằng cách đặc tả một configuration thích hợp.

Library HS\_LIB, CMOS\_LIB;

ENTITY FULL\_ADDER IS

Port(A,B,Cin : IN BIT; SUM, Cout: OUT BIT);

END FULL\_ADDER;

ARCHITECTURE FA\_STR OF FULL\_ADDER IS

Component XOR2

Port(D1,D2 : IN BIT; DZ : OUT BIT);

END componen;

Component AND2

Port ( Z: OUT BIT; B0,B1: IN BIT);

END Component;

-- Đặc tả cấu hình:

CONFIGURATION FA\_BINDING OF FULL\_ADDER IS

For FA\_STR

For X1,X2 : XOR2

use entity WORK.XOR2(XOR2BEH);

End for;

-- liên kết thực thể với nhiều đối tượng của một thành phần.

For A3: AND2

Use entity HS\_LIB.AND2HS(AND2STR);

PORT MAP (HS\_B => A1, HS\_Z=>Z, HS\_A => A0 );

End for;

--liên kết thực thể với các đối tượng đơn của một thành phần.

For all : OR2

Use entity CMOS\_LIB.OR2CMOS(OR2STR);

End for;

-- liên kết thực thể với tất cả các đối tượng của thành phần OR2.

For others: AND2

Use entity WORK.A\_GATE(A\_GATE\_BODY);

PORT MAP(A0,A1,Z);

End for;

-- liên kết thực thể với tất cả các đối tượng không được liên kết của

thành phần AND2.

End for;

End for;

Signal S1, S2, S3, S4, S5: BIT;

BEGIN

X1: XOR2 port map(A, B, S1);

X1: XOR2 port map(S1, Cin, SUM);

A1: AND2 port map(S2, A, B);

A2: AND2 port map(S3, B, Cin);

A3: AND2 port map(S4, A, Cin);

O1: OR2 port map(S2, S3, S5);

O2: OR2 port map(S3, B, Cin);

NAND\_GATE port map(S4,S5,Cout);

END FA\_STR;

Có 4 đặc tả cấu hình trong phần khai báo của thân kiến trúc (architecture body):

• Đặc tả thứ nhất chỉ rằng đối tượng có nhãn X1 và X2 của component XOR2 liên kết với entity bởi cặp entity - architecture XOR2 và XOR2BEH có sẵn trong thư viện WORK.

• Đặc tả thứ hai liên kết đối tượng component AND2 có nhãn A3 đến entity bởi cặp entity - architecture AND2HS và AND2STR đã có trong thư viện thiết kế HS\_LIB. Anh xạ của cổng thành phần (AND2 ) và các cổng thực thể(AND2HS) được liên kết theo tên (name association).

• Đặc tả thứ ba chỉ rằng tất cả các đối tượng của component OR2 được liên kết với entity bởi cặp entity - architecture có sẵn trong thư viện thiết kế CMOS\_LIB.

• Đặc tả cuối cùng chỉ rằng tất cả các đối tượng không liên kết (unbound) của component AND2, đối tượng A1 và A2 được liên kết tới entity khác là: A\_GATE có architecture A\_GATE\_BODY, kiến trúc này có sẵn trong thư viện WORK.

***Câu 10 : Một chương trình VHDL cơ bản gồm mấy phần? Ý nghĩa của từng phần?***

Tất cả các thiết kế VHDL cung cấp một giao diện bên ngoài và sự thực thi bên trong, gồm hai thành phần chính là các ENTITY và ARCHITECTURE

. Các ENTITY:

- Entity là một đặc tả giao diện bên ngoài của thiết kế

- Entity khai báo các thông số như sau:

+Tên của Entity

+ Một bộ các khai báo generic xác định các thông số đặc trưng

+ Một bộ các khai báo cổng định nghĩa các đầu vào và ra của thiết kế phần cứng

- Khai báo generic và port là tùy chọn

+khai báo Entity

ENTITY entity\_name IS

PORT (cổng 1: chiều kiểu cổng;

Cổng 2: chiều kiểu cổng;

….

);

END entity\_name

Các Architecture

- Một Architecture là một đặc tả thực thi bên trong của thiết kế

- Nhiều Architecture có thể được tạo ra trong một entity riêng biệt

Cách khai báo trong Architecture được xác định như sau:

ARCHITECTURE tên kến trúc OF entity\_name IS

BEGIN

--- Đoạn chương trình đặt ở đây

END tên kiến trúc;

***Câu 11 : Giả sử bạn đã thiết kế thực thể AND\_GATE có kiến trúc đặt tên là DATAFLOW. Chúng ta có thể thiết kế một thực thể OR\_GATE khác với kiến trúc cũng có tên là DATAFLOW không?***

Có thể. Kiến trúc behavior có thể dùng cho nhiều entity trong 1 project. Kiến trúc dataflow cũng vậy.

***Câu 12 : Giả sử bạn đã định nghĩa một hàm “PARITY” trong chương trình nguồn và nó trùng tên với một hàm trong một thư viện mà bạn khai báo dùng. Vậy phiên bản nào của hàm sẽ được chọn?***

Giống như nhiều ngôn ngữ khác, VHDL hỗ trọ overloading nên khi khai báo hàm trùng tên với một hàm trong thư viện thì hàm trong mã nguồn sẽ được gọi.

***Câu 13 : Trong phần khai báo entity, có mấy mode khai báo cổng? Mối liên hệ của mode với vị trí của tên cổng trong toán tử gán tín hiệu “<=” ? Sự khác nhau giữa toán tử gán “<=” và “:= ?” ?***

Trong phần entity khi khai báo cổng thì có các mode sau:

* in: cổng vào,
* out: cổng ra,
* inout vào ra hai chiều.
* buffer: cổng đệm có thể sử dụng như tín hiệu bên trong và output.
* linkage: Có thể là bất kỳ các cổng nào kể trên

Khi sử dụng toán tử “<=” sẽ theo cấu trúc như sau:

Output <= input

Toán tử <= thì lệnh gán sử được thực hiện khi kết thúc 1 cycle (kết thúc toàn bộ process hoặc 1 câu lệnh song song)

Toàn tử := thì lệnh gán được thực hiện ngay lập tức

***Câu 14 : Có mấy kiểu kiến trúc? Một Entity có thể gồm nhiều kiến trúc không (đa kiến trúc (multiply architecture) ?***

Một kiến trúc đưa ra kết cấu bên trong của một Entity. Một Entity có thể có nhiều hơn một kiến trúc, nó chỉ ra mối quan hệ giữa đầu vào và đầu ra của một Entity mà quan hệ này được biểu diễn bởi các thuật ngữ sau:

+ Kiểu hành vi hoạt động (Behavioral): Hoạt động của nó chỉ ra hoạt động mà một hệ thống riêng biệt nào đó phải thực hiện trong chương trình, nó như việc diễn tả các quá trình hoạt động, không cung cấp chi tiết mà thiết kế được thực thi như thế nào.

+ Kiểu hoạt động của các kiểu luồng dữ liệu (Data flow): Kiểu này mô tả đồng thời các luồng điều khiển và dịch chuyển dữ liệu.

+ Kiểu cấu trúc (Structure): Kiểu kiến trúc chỉ ra sự thực thi cấu trúc theo dạng sử dụng các khai báo phần tử và các thể hiện phần tử đó.

Một kiến trúc xác định hành vi của một Entity, nó bao gồm khai báo (khai báo các tín hiệu, các hằng, khai báo các kiểu, các phần tử, các phát biểu đồng thời)

Khai báo kiến trúc sử dụng cấu trúc sau:

Architecture architecture\_Name of Entity\_name Is

{architecture\_declarative\_part}

Begin

{Concurrent\_Statement}

End [architecture\_name]

Trong VHDL có 3 kiểu kiến trúc data flow, behavior, struct trong đó Data flow là kiểu luồng dữ liệu

Một Entity có thể gồm nhiều kiến trúc

***Câu 15 : Mô hình hóa tín hiệu bằng VHDL như thế nào? Sử dụng cú pháp nào? Ví dụ mô tả tín hiệu tín hiệu kiểu BIT và kiểu STD\_LOGIC? Sự khác nhau giữa 2 kiểu?***

- Mô hình hóa tín hiệu bằng VHDL như thế nào?

signals: dùng để kết nối những khối riêng biệt của kiến trúc. Mỗi tín hiệu tương ứng với một kiểu dữ liệu xác định dạng của dữ liệu được xử lý.

- Các kiểu tín hiệu ( Signal ):

Tín hiệu được dùng để kết nối các Entity của thiết kế lại với nhau và trao đổi các giá trị biến đổi ở trong phát biểu process. Chúng có thể được xem như các dây dẫn hay các bus nối ở trong mạch thực tế. Tín hiệu có thể được khai báo trong các đóng gói (Package), trong các khai báo Entity, trong khai báo kiến trúc (Architecture), trong các khối ( Block ). Với các tín hiệu được khai báo trong các package thì tín hiệu này được gọi l\_ tín hiệu to\_n cục (Các thiết kế có thể sử dụng chúng ), các tín hiệu được khai báo trong Entity là tín hiệu toàn cục trong một Entity, tương tự với tín hiệu được khai báo trong một kiến trúc, nó l\_ tín hiệu dùng chung trong một kiến trúc đó.

Cú pháp của chúng có dạng như sau:

Signal Signal\_name {,signal\_name}: type [:=value]; Ví dụ :

signal BEEP : BIT:= '0';

signal TEMP: STD\_LOGIC\_VECTOR (8 downto 0);

signal COUNT: INTEGER range 0 to 100 :=5;

Ví dụ:

signal IN1, IN2, OU : STD\_LOGIC;

process (IN1, IN2)

begin

if IN1 = '0' or IN2 = '0' then

OU <= '0' ;

elsif IN1 = 'X' or IN2 = 'X' then

OU <= '1';

else

OU <= '1' ;

end if;

end process;

Mô hình hóa tín hiệu VHDL

Tín hiệu kiểu bit là kiểu mô tả 2 giá trị rời rạc bằng việc sử dụng các ký tự ‘0’ và ‘1’. Đôi khi các kiểu này còn được tạo ra kiểu chữ bit tường minh dùng để phân biệt với các kiểu ký tự.

VD: ‘1’, ‘0’, bit(‘0’)

Kiểu STD\_LOGIC dùng để tạo mẫu tín hiệu nhiều hơn hai giá trị (‘0’,’1’) VHDL định nghĩa 9 khoảng trong gói chuẩn,

‘U’: Không khởi tạo giá trị

‘X’: Không xác định,

‘0’: Kiểu mưc thấp

‘1’: Kiểu mức cao

‘Z’: Trở kháng cao

‘W’: Không xác định ở mức yếu

‘L’: Mức thấp yếu

‘H’: Mức cao yếu

‘\_’: Không quan tâm đến giá trị

***Câu 16 : Mô tả bus tín hiệu như thế nào? Ví dụ mô tả bus dữ liệu D[7:0]?***

Một hệ thống Bus có thể được xây dựng với các cổng ba trạng thái thay vì các cổng multiplexers.

Người thiết kế phải đảm bảo không có nhiều hơn một bộ đệm ở trạng thái kích hoạt tại bất kỳ thời điểm nào. Các bộ đệm kết nối cần phải được điều khiển vì vậy chỉ có bộ đệm ba trạng thái truy cập đường Bus trong khi các bộ đệm khác duy trì ở trạng thái trở kháng cao.

Thông thường các phép gán tín hiệu tức thì, chẳng hạn như các đường Bus trong ví dụ dưới đây đã không được phép ở mức một kiến trúc. Tuy nhiên các kiểu dữ liệu STD\_LOGIC và STD\_LOGIC\_VECTOR có thể có nhiều đường điều khiển.

***library IEEE****;*

***use*** *IEEE.STD\_LOGIC\_1164.all;*

***Entity*** *BUS* ***is***

***port*** *(S : in STD\_LOGIC\_VECTOR ( 1 downto 0 );*

*OE : buffer STD\_LOGIC\_VECTOR ( 3 downto 0 );*

*R0, R1, R2, R3 : in STD\_LOGIC\_VECTOR ( 7 downto 0 );*

*BusLine : out STD\_LOGIC\_VECTOR ( 7 downto 0 ) );*

***end*** *BUS ;*

***architecture***  *IMP* ***of*** *BUS* ***is***

***Begin***

***Process*** *(S)*

***Begin***

***Case*** *(S)* ***is***

***when*** *" 00 " => OE <= "0001";*

***when*** *" 01 " => OE <= "0010";*

***when*** *" 10 " => OE <= "0100";*

***when*** *" 11 " => OE <= "1000";*

***when others*** *=> null;*

***end Case****;*

***end Process*** *;*

*BusLine <= R0* ***when*** *OE (0) = ' 1' else "ZZZZZZZZ";*

*BusLine <= R0* ***when*** *OE (1) = ' 1' else "ZZZZZZZZ";*

*BusLine <= R0* ***when*** *OE (2) = ' 1' else "ZZZZZZZZ";*

*BusLine <= R0* ***when*** *OE (3) = ' 1' else "ZZZZZZZZ";*

***end*** *IMP;*

****

**Cấu trúc đường Bus tám bit**

***Câu 17 : Phân biệt câu lệnh gán biến, gán tín hiệu? Phân biệt vị trí khai báo biến và tín hiệu trong chương trình VHDL?***

Phép gán biến: Gán biến là thiết lập giá trị mới cho biến. Cú pháp:

*biến := biểu\_thức;*

Biến phải được khai báo từ trước, biến và biểu thức phải cùng kiểu. Một biến được khai báo trong một quá trình hoặc chương trình con sẽ tồn tại cục bộ trong quá trình hoặc chương trình con đó và không thể truy cập tới từ các cấu trúc song song khác.

Phép gán tín hiệu: Gán tín hiệu dùng để thay đổi giá trị của tín hiệu. Các tín hiệu luôn được biểu diễn kết hợp với biến thời gian. Phép gán tín hiệu thay đổi giá trị của tín hiệu tương ứng theo thời gian và phụ thuộc vào các mô hình quá trình trễ trong các phần tử mạch.

Cú pháp:

*tín\_hiệu\_đích <=* [transport] *biểu\_thức* [after *thời\_gian*]

*biểu\_thức* xác định giá trị gán. Kiểu của *biểu\_thức* phải trùng với kiểu của *tín\_hiệu\_đích*

*Thời\_gian* là biểu thức có kiểu TIME

***Câu 18 : Phát biểu if... elsif, case ... is, with ... select, loop***

Câu lệnh IF: Câu lệnh if tạo nên phân nhánh trong khi thực hiện chương trình. Tùy theo kết quả của biểu thức điều kiện mà có thể hoặc một số lệnh hoặc không có lệnh nào được thực hiện. Cú pháp:

if <điều\_kiện> then

{<câu\_lệnh\_tuần\_tự>}

{elsif <điều\_kiện> then}

{<câu\_lệnh\_tuần\_tự>}

[else {<câu\_lệnh\_tuần\_tự>}]

end if;

Trong đó, <điều\_kiện> là biểu thức kiểu Boolean

Câu lệnh Case: Câu lệnh case sử dụng trong trường hợp có một biểu thức kiểm soát nhiều rẽ nhánh trong chương trình VHDL. Cú pháp

case *<biểu\_thức>* is

when *<lựa\_chọn> =>*

*{<câu\_lệnh\_tuần\_tự>}*

*{*when *<lựa\_chọn> =>*

*{<câu\_lệnh\_tuần\_tự>} }*

*end case;*

*Lựa chọn cuối cùng có thể là* others

***Lệnh with…select***

Cú pháp:

**with** <bien1> **select**

<bien2> <= <bien2\_value1> **when** <bien1\_value1>,

<bien2\_value2> **when** <bien2\_value2>,

<bien2\_valuen> **when others;**

Lệnh này giống với lệnh case, chỉ có điểm khác là ko nhất thiết phải sử dụng trong process, function, procedure.

. Các lệnh vòng lặp: Lệnh Loop chứa thân vòng lặp bao gồm dãy các câu lệnh sẽ được thực hiện không hoặc nhiều lần. Cú pháp

[<nhãn>:] [<sơ\_đồ\_lặp>] loop

{<lệnh\_tuần\_tự>}|

{next [<nhãn>] [when <điều\_kiện>];}|

{exit [<nhãn>] [when <điều\_kiện>];}

end loop [nhãn];

Sơ\_đồ\_lặp: vòng lặp với sơ đồ for, while, và vòng lặp không chứa sơ đồ lặp

***Câu 19 : Phát biểu Generic. Chức năng. Cú pháp.***

Generic là một khai báo một phần chung trong ngôn ngữ lập trình VHDL.

Chức năng của Generic chỉ là thay thế một giá trị bằng một nhãn nào đó và có hiệu lực trong toàn chương trình, được khai báo trên thực thể entity.

Cú pháp:

Entity bai is

Generic(<tên nhãn>:type:= giá trị được gán).

Port ();

End entity;

Ví dụ:

entity chiatan is

Port ( fvao : in STD\_LOGIC;

fra : out STD\_LOGIC;

reset : in STD\_LOGIC);

end chiatan;

architecture Behavioral of chiatan is

-- Tan so vao: 10MHz, tan so ra: 1MHz => chia cho 10 lan

-- Tuc la tan so vao thuc hien duoc 10 dao dong thi tan so ra thuc hien duoc 1 dao dong

-- Mot dao dong co Time\_high = Time\_low

-- Can mot counter dem den 10/2 = 5 => counter 3 bit (2^3 ma hoa duoc 8 truong hop)

signal dem: std\_logic\_vector (2 downto 0);

signal temp: std\_logic;

begin

process (fvao,reset)

begin

if reset='1' then

dem <= "000";

temp <= '0';

elsif fvao='1' and fvao'event then

dem <= dem + 1;

if dem = "100" then -- 0-4 la 5 gia tri dem

dem <= "000";

temp <= not temp;

end if;

end if;

end process;

fra <= temp;

end Behavioral;

***Câu 20 : Câu lệnh đồng thời và tuần tự. Câu lệnh Process.***

**Câu lệnh đồng thời:**

Các phát biểu đồng thời được thực hiện song song cùng thời điểm mô phỏng, chúng không thực hiện theo thứ tự mà chúng được viết ra trong một kiến trúc. Các phát biểu đồng thời chuyển thông tin thông qua các đường tín hiệu.

**Câu lệnh tuần tự:**

Phát biểu tuần tự chỉ ra sự thực hiện từng bước của một quá trình. Chúng thực hiện từ câu lệnh đầu tiên, câu lệnh thứa hai, … câu lệnh cuối cùng. Các phát biểu nằm trong một phát biểu quá trình (phát biểu **process**) được gọi là phát biểu tuần tự.

**Câu lệnh Process:**

Phát biểu Process là phát biểu bao gồm một tập các phát biểu tuần tự và phát biểu Process lại chính là phát biểu đồng thời. Có nghĩa là tất cả các phát biểu Process trong một thiết kế được thực hiện một cách đồng thời. Tuy nhiên tại một thời điểm nhất định được đưa ra chỉ có một phát biểu tuần tự được thực hiện trong mỗi Process. Một Process được kết nối với phần còn lại của thiết kế bởi việc đọc hoặc viết ra các giá trị từ các tín hiệu và các cổng mà chúng đã được khai báo phía ngoài Process. Cú pháp:

[label:] process [(sensitivity\_list)]

{process declaration part}

Begin

{sequential statements}

End process [label];

Phần khai báo của một process chỉ ra các đối tượng mà vùng hoạt động của nó chỉ thuộc vùng của một process và chúng có thể là các đối tượng sau đây:

- Khai báo biến.

- Khai báo hằng.

- Khai báo các kiểu.

- Khai báo các kiểu con.

- Khai báo các bí danh Alias.

- Các mệnh đề USE.

Một sensitivity list (Tập các sự kiện thay đổi trạng thái cần xử lý trong một quá trình ) có cùng ý nghĩa với một Process có chứa phát biểu wait, mà phát biểu wait này là phát biểu cuối cùng trong một process và chúng có dạng sau: **Wait on** sensitivity list ;

Một process có chức năng giống như một vòng lặp vô hạn mà trong nó có chứa toàn bộ các phát biểu tuần tự được chỉ ra trong vòng lặp đó. Vì vậy một phát biểu process cần phải có hoặc một sensitivity list hoặc một phát biểu wait on hoặc cả hai.

Ví dụ 1:

***architecture*** *A2* ***of*** *example* ***is***

***signal*** *i1, i2, i3, i4, and\_out, or\_out : bit;*

***begin***

*pr1 :* ***process*** *(i1, i2, i3, i4)*

***begin***

*and\_out <= i1* ***and*** *i2* ***and*** *i3* ***and*** *i4;*

***end process***  *pr1;*

*pr2 :* ***process*** *(i1, i2, i3, i4)*

***begin***

*or\_out <= i1* ***or*** *i2* ***or*** *i3* ***or*** *i4 ;*

***end process*** *pr2;*

***end*** *A2*

***Câu 21: Mô hình hóa kiểu cấu trúc (structural), khai báo thành phần, tín hiệu.***

Khai báo một kiến trúc sử dụng cú pháp sau:

architecture **architecture\_name** of ***entity\_name*** is

***{*****architecture\_declarative\_part *}*** (Các khai báo thành phần)

***Begin***

{*concurrent\_statement*} (Các phát biểu đồng thời)

***end*** [ *architecture\_name* ];

Một kiến trúc kiểu cấu trúc (Structure) chỉ ra sự thực thi cấu trúc theo dạng sử dụng các khai báo phần tử và các thể hiện của phần tử đó. Ví dụ dưới đây chỉ ra sự diễn tả cấu trúc của một bộ cộng FULL\_ADDER như trên đã giới thiệu. Hai kiểu phần tử được sử dụng trong ví dụ này là HALF\_ADDER và OR\_GATE.

architecture **STRUCTURE** of **FULL\_ADDER** is

component **HALF\_ADDER**

port **(L1, L2 : in BIT;**

**CARRY, SUM : out BIT);**

end component**;**

component **OR\_GATE**

port **(L1, L2 : in BIT;**

**O: out BIT);**

end component**;**

begin

**HA1: HALF\_ADDER** port map **(A,B,N1,N2);**

**HA2: HALF\_ADDER** port map **(N2,CIN,N3,SUM);**

**OR1 : OR\_GATE** port map **(N1, N3,COUT);**

end **STRUCTURE;**

***Câu 22: Thuyết minh (instatiation) của thành phần trong một kiến trúc. Ví dụ khai báo và thuyết minh thành phần của một kiến trúc là bộ cộng Full Adder 4 bit .***

Thuyết minh của thành phần trong một kiến trúc là việc sử dụng các thành phần con để cấu thành nên cấu trúc. Trước khi sử dụng các thành phần cần khai báo như sau:

Component tên\_thành\_phần is

Port (

Port\_name : signal\_mode signal\_type

);

End component

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**STD\_LOGIC\_ARITH**.ALL;**

**use** IEEE**.**STD\_LOGIC\_UNSIGNED**.ALL;**

**entity** my\_FA1 **is**

**Port** **(** a**,**b **:** **in** STD\_LOGIC\_VECTOR **(**3 **downto** 0**);**

cin **:** **in** STD\_LOGIC**;**

s **:** **out** STD\_LOGIC\_VECTOR **(**3 **downto** 0**);**

cout **:** **out** STD\_LOGIC**);**

**end** my\_FA1**;**

**architecture** Structural **of** my\_FA1 **is**

**signal** C\_temp **:** std\_logic\_vector **(**2 **downto** 0**);**

--declaration full adder

**component** my\_full\_adder

**Port** **(** a **:** **in** STD\_LOGIC**;**

b **:** **in** STD\_LOGIC**;**

cin **:** **in** STD\_LOGIC**;**

s **:** **out** STD\_LOGIC**;**

cout **:** **out** STD\_LOGIC**);**

**end** **component;**

**begin**

--phần thuyết minh của 1 cấu trúc

u0**:** **component** my\_full\_adder

**port** **map** **(**a **=>** A**(**0**),** b **=>** B**(**0**),** cin **=>** cin**,**

s **=>**s**(**0**),** cout **=>** C\_temp**(**0**));**

u1**:** **component** my\_full\_adder

**port** **map** **(**a **=>** A**(**1**),** b **=>** B**(**1**),** cin **=>** C\_temp**(**0**),**

S **=>**s**(**1**),** Cout **=>** C\_temp**(**1**));**

u2**:** **component** my\_full\_adder

**port** **map** **(**a **=>** A**(**2**),** b **=>** B**(**2**),** cin **=>** C\_temp**(**1**),**

S **=>**s**(**2**),** Cout **=>** C\_temp**(**2**));**

u3**:** **component** my\_full\_adder

**port** **map** **(**a **=>** A**(**3**),** b **=>** B**(**3**),** cin **=>** C\_temp**(**2**),**

S **=>**s**(**3**),** Cout **=>** cout**);**

**end** Structural**;**

***Câu 23. Các thiết bị logic cơ bản (Cổng logic AND, OR, NAND, NOR, XOR, Mạch lật, Bộ đếm, Thanh ghi, Bộ dồn kênh MUX, Bộ phân kênh, Bộ mã hóa, Bộ giải mã, Bộ cộng).***

**- Cổng AND2:**

Entity cong\_and2 is

Port(A:in STD-LOGIC;

B:in STD-LOGIC;

Z:out STD-LOGIC;)

And cong\_and2;

Architecture Behavioral of cong\_and2 is

Begin

**Z<=(A and B)AFTER 144 ps;**

End Behavioral

**- Cổng OR2:**

Entity cong\_or2 is

Port(A:in STD-LOGIC;

B:in STD-LOGIC;

Z: STD-LOGIC;)

And cong\_or2;

Architecture Behavioral of cong\_or2 is

Begin

**Z<=(A or B)AFTER 154 ps;**

End Behavioral

**- Cổng NAND2:**

Entity cong\_nand2 is

Port(A:in STD-LOGIC;

B:in STD-LOGIC;

Z: STD-LOGIC;)

And cong\_nand2;

Architecture Behavioral of cong\_nand2 is

Begin

**Z<=(not (A and B)AFTER 114 ps;**

End Behavioral

**- Cổng NOR2 đầu vào:**

Entity cong\_nor2 is

Port(A:in STD-LOGIC;

B:in STD-LOGIC;

Z: STD-LOGIC;)

And cong\_nor2;

Architecture Behavioral of cong\_nor2 is

Begin

**Z<=(not (A or B)AFTER 124 ps;**

End Behavioral

**- Cổng XOR3:**

Entity cong\_xor3 is

Port(A:in STD-LOGIC;

B:in STD-LOGIC;

C: in STD-LOGIC;

Z: out STD-LOGIC);

And cong\_xor3;

Architecture Behavioral of cong\_xor3 is

Begin

**Z<= (A xor B)AFTER 174 ps;**

End Behavioral

***Câu 24. Thiết kế VXL: cấu trúc VXL, các tín hiệu liên kết, quy trình thiết kế, công cụ thiết kế VXL.***

***Project 5: BTL***

***Câu 25. Thiết kế hệ thống nhúng: cấu trúc HTN, công cụ thiết kế EDK, quy trình thiết kế HTN.***

***Project6 : BTL***