BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

ISO 9001:2008

ĐỒ ÁN TỐT NGHIỆP

NGÀNH: ĐIỆN TỬ VIỄN THÔNG

Sinh viên : Hoàng Văn Thơi
Giảng viên hướng dẫn : ThS. Đoàn Hữu Chức

HẢI PHÒNG - 2013

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

NGHIÊN CỨU CÔNG NGHỆ FPGA VÀ PHÁT TRIỂN CÁC ỨNG DỤNG TRÊN KIT SPARTAN 3E.

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY

NGÀNH: ĐIỆN TỬ VIỄN THÔNG

Sinh viên : Hoàng Văn Thơi

Giảng viên hướng dẫn : ThS. Đoàn Hữu Chức

HẢI PHÒNG – 2013

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

NHIỆM VỤ ĐỀ TÀI TỐT NGHIỆP

Sinh viên : Hoàng Văn Thơi

Giảng viên hướng dẫn : ThS. Đoàn Hữu Chức

Tên đề tài nghiên cứu công nghệ FPGA và phát triển các ứng dụng trên kit Spartan 3E.

:

NHIỆM VỤ ĐỀ TÀI

1. Nội dung và các yêu cầu cần giải quyết trong nhiệm vụ đề tài tốt nghiệp(về lý luận, thực tiễn, các số liệu cần tính toán và các bản vẽ).

……..
……..
……..
……..
……..
……..
……..
……..

2. Các số liệu cần thiết để thiết kế, tính toán.

……..
……..
……..
……..
……..
……..
……..
……..
……..

3. Địa điểm thực tập tốt nghiệp.

……..
……..
……..

CÁN BỘ HƯỚNG DẪN ĐỀ TÀI TỐT NGHIỆP

Người hướng dẫn thứ nhất:

Họ và tên:.....

Học hàm, học vị:.....

Cơ quan công tác:.....

Nội dung hướng dẫn:.....

Người hướng dẫn thứ hai:

Họ và tên:.....

Học hàm, học vị:.....

Cơ quan công tác:.....

Nội dung hướng dẫn:.....

Đề tài tốt nghiệp được giao ngày 25 tháng 03 năm 2013

Yêu cầu phải hoàn thành xong trước ngày 29 tháng 06 năm 2013

Đã nhận nhiệm vụ ĐTTN	Đã giao nhiệm vụ ĐTTN
Sinh viên	*Người hướng dẫn*

Hải Phòng, ngày ..... tháng.....năm 2013

Hiệu trưởng

GS.TS.NGƯT *Trần Hữu Nghị*

PHẦN NHẬN XÉT CỦA CÁN BỘ HƯỚNG DẪN

1. Tinh thần thái độ của sinh viên trong quá trình làm đề tài tốt nghiệp:

……..
……..
……..
……..
……..
……..
……..

2. Đánh giá chất lượng của khóa luận (so với nội dung yêu cầu đã đề ra trong nhiệm vụ Đ.T. T.N trên các mặt lý luận, thực tiễn, tính toán số liệu…):

……..
……..
……..
……..
……..
……..
……..
……..
……..

3. Cho điểm của cán bộ hướng dẫn (ghi bằng cả số và chữ):

……..
……..
……..

Hải Phòng, ngày ... tháng ... năm 2013

Cán bộ hướng dẫn

(Ký và ghi rõ họ tên)

PHẦN NHẬN XÉT TÓM TẮT CỦA NGƯỜI CHẤM PHẢN BIỆN

1. Đánh giá chất lượng đề tài tốt nghiệp về các mặt thu thập và phân tích số liệu ban đầu, cơ sở lý luận chọn phương án tối ưu, cách tính toán chất lượng thuyết minh và bản vẽ, giá trị lý luận và thực tiễn đề tài.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

2. Cho điểm của cán bộ phản biện (Điểm ghi cả số và chữ).

.....

.....

.....

.....

.....

.....

.....

.....

Hải Phòng, ngày……tháng……năm 2013

Người chấm phản biện

MỤC LỤC

MỤC LỤC

DANH SÁCH HÌNH VẼ

LỜI NÓI ĐẦU

Ngày nay với sự phát triển hết sức mạnh mẽ công nghệ, thuật toán ngày càng được đổi mới và tối ưu hoá nhằm nâng cao tính hiệu quả của nó. Tuy nhiên, công nghệ phát triển càng cao thì đòi hỏi phần cứng phải đủ nhanh để xử lý. Các mạch lọc tương tự trước đây không còn đủ khả năng để đáp ứng yêu cầu đó nữa. Vì vậy, FPGA đã ra đời như một giải pháp cung cấp môi trường làm việc hiệu quả cho các ứng dụng thực tế. Tính linh động cao trong quá trình thiết kế cho phép FPGA giải quyết những bài toán phức tạp mà trước kia chỉ thực hiện nhờ phần mềm máy tính. Ngoài ra, nhờ mật độ cổng logic cao, FPGA được ứng dụng cho những bài toán đòi hỏi khối lượng tính toán lớn và dùng trong các hệ thống làm việc theo thời gian thực. Những ứng dụng trong thực tế của FPGA rất rộng rãi, bao gồm: các hệ thống hàng không, vũ trụ, quốc phòng,…Đặc biệt, với khả năng tái lập trình, người sử dụng có thể thay đổi lại thiết kế của mình chỉ trong vài giờ.

Nhờ những đặc điểm mạnh mẽ và ứng dụng thực tiễn của FPGA em đã chọn đề tài "Nghiên cứu công nghệ FPGA và phát triển các ứng dụng trên KIT Spartan 3E".

Để thực hiện được đồ án này em xin gửi lời cảm ơn chân thành nhất đến tất cả các thầy cô trong khoa Điện – Điện tử nói riêng đã đạy dỗ, và giúp đỡ em suốt thời gian em học tại trường.

Em xin chân thành cảm ơn thầy giáo hướng dẫn, Thạc sỹ - Giảng viên Đoàn Hữu Chức, khoa Điện – Điện tử, trường Đại học Dân Lập Hải Phòng đã nhiệt tình hướng dẫn, chỉ bảo và cung cấp cho em nhiều kiến thức cũng như tài liệu trong suốt quá trình làm đồ án. Nhờ sự giúp đỡ của thầy em mới có thể hoàn thành được đồ án này.

Hải phòng, ngày 29 tháng 6 năm 2013

Tác giả

Hoàng Văn Thơi

NHỮNG TỪ VIẾT TẮT

Ký Hiệu	Diễn Giải
ASIC	ApplicationSpecific Integrated Circuit
ALU	Arithmetic Logic Unit
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CLB	Configurable Logic Blocks
DSP	Digital Signal Processing
FPGA	FieldProgrammable Gate Array
HDL	Hardware Description Language
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
PAL	Programmable Array Logic
PLA	Programmable Logic Array
RAM	Random Access Memory
ROM	Read Only Memory
SoC	System on chip
SPLD	Simple Programable Logic Device
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Itergrated Circuit

CHƯƠNG 1 TỔNG QUAN VỀ FPGA VÀ NGÔN NGỮ VHDL

1.1 TỔNG QUAN VỀ FPGA

1.1.1 FPGA là gì?

FPGA (Field-Programmable Gate Arry) là vi mạch dùng cấu trúc mảng phần tử logic mà người dùng có thể lập trình được. Vi mạch FPGA được cấu thành từ các bộ phận :

- Các khối logic cơ bản lập trình được(logic block)
- Hệ thống mạch liên kết lập trình được
- Khối vào/ra (IO Pads)
- Phần tử thiết kế săn khác như DSP sline, Ram, ROM, nhân vi xử lý…

So sánh FPGA với ASIC và các vi mạch bán dẫn khác:

ASIC (Application-Specific Integrated Circuit) là một vi mạch IC được thiết kế dành cho một ứng dụng cụ thể .FPGA cũng được xem như một loại vi mạch bán dẫn chuyên dụng ASIC, nhưng nếu so sánh FPGA với những ASIC đặc chế hoàn toàn hay ASIC thiết kế trên thư viện logic thì FPGA không đạt được mức độ tối ưu như những loại này , và hạn chế trong khả năng thực hiện những tác vụ đặc biệt phức tạp , tuy vậy FPGA ưu việt hơn ở chỗ có thể tái cấu trúc lại khi sử dụng, công đoạn thiết kế đơn giản do vạy chi phí giảm, rút ngắn thời gian đưa sản phẩm vào sử dụng.

Còn nếu so sánh với các dạng vi mạch bán dẫn lập trình được cấu trúc mảng phần tử logic như PLA, PAL, CPLD thì FPGA ưu việt hơn các điểm:

- ✓ Tác vụ tái lập của FPGA thực hiện đơn giản hơn.
- ✓ Khả năng lập trình linh động hơn
- ✓ Kiến trúc của FPGA cho phép nó có khả năng chứa khối lượng lớn cổng logic (logic gate), so với các vi mạch bán dẫn lập trình được có trước nó .
- ✓ Thiết kế hay lập trình cho FPGA được thực hiện chủ yếu bằng ngôn ngữ mô tả phần cứng HDL như VHDL, Verilog, AHDL, các hãng sản suất FPGA lớn như Xilinx, Altera thường cung cấp các gói phần mềm và thiết bị phụ trợ cho quá trình thiết kế , cũng như có một hãnh thứ ba cung cấp các gói phần mềm này như Synosys, Synplify… Các gói phần mềm này có khả năng thực hiện tất cả các bước của toàn bộ quy trình thiết kế IC chuẩn với đầu vào là thiết kế trên HDL (còn gọi là mã RTL)

1.1.2 Lịch sử ra đời của FPGA

FPGA được thiết kế đầu tiên bởi Ross Freeman, người sáng lập công ty Xilinx vào năm 1984, kiến trúc mới của FPGA cho phép tích hợp số lượng tương đối lớn các phần tử bán dẫn vào một vi mạch. So với kiến trúc trước đó là CPLD, FPGA có khả năng chứa tới từ 100.000 đến hàng vài tỷ cổng logic, trong khi CPLD chỉ chứa từ 10.000 đến 100.000 cổng logic; con số này đối với PAL, PLA còn thấp hơn nữa chỉ đạt vài nghìn đến 10.000.

CPLD được cấu trúc từ số lượng nhất định các khối SPLD (Simple programable logic device) thuật ngữ chung chỉ PAL, PLA. SPLD thường là một mảng logic AND/OR lập trình được có kích thước xác định và chứa một số lượng hạn chế các phần tử nhớ đồng bộ (clocked register). Cấu trúc này hạn chế khả năng thực hiện những hàm phức tạp và thông thường hiệu suất làm việc của vi mạch phụ thuộc vào cấu trúc cụ thể của vi mạch hơn là vào yêu cầu bài toán.

Kiến trúc của FPGA là kiến trúc mảng các khối logic, mỗi khối này nhỏ hơn nhiều nếu đem so sánh với một khối SPLD, ưu điểm này giúp FPGA có thể chứa nhiều hơn các phần tử logic và phát huy tối đa khả năng lập trình của các phần tử logic và hệ thống mạch kết nối, để đạt được mục đích này thì kiến trúc của FPGA phức tạp hơn nhiều so với CPLD.

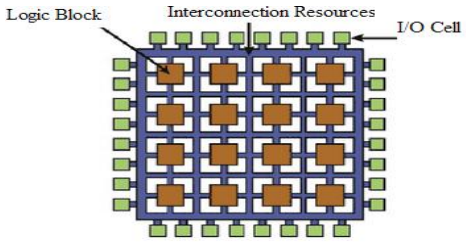
Một điểm khác biệt nữa với CPLD là trong những FPGA hiện đại được tích hợp nhiều bộ logic số học đã được tối ưu hóa, hỗ trợ RAM, ROM, tốc độ cao, hay các bộ nhân, cộng dùng cho những ứng dụng xử lý tín hiệu số.Ngoài khả năng cấu trúc lại vi mạch ở mức toàn cục, một số FPGA hiện đại còn hỗ trợ cấu trúc lại ở mức cục bộ, tức là khả năng cấu trúc lại một bộ phận riêng lẻ trong khi vẫn đảm bảo hoạt động bình thường cho các bộ phận khác.

Khái niệm cơ bản và cấu trúc của FPGA, Vi mạch FPGA được cấu thành từ các bộ phận

a.Khái niệm cơ bản và cấu trúc của FPGA

FPGA (Field Programable Gate Arrays - Ma trận cổng lập trình được theo hàng) là một thiết bị bán dẫn bao gồm các khối logic lập trình được gọi là "Logic Block", và các kết nối khả trình. Các khối logic có thể được lập trình để thực hiện các chức năng của các khối logic cơ bản như AND, XOR, hoặc các chức năng kết hợp phức tạp hơn như decoder hoặc các phép tính toán học.

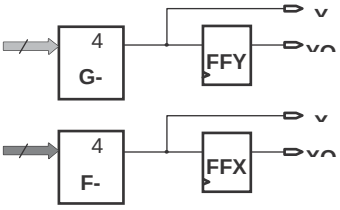
Trong hầu hết các kiến trúc FPGA, các khối logic cũng bao gồm cả các phần tử nhớ. Đó có thể là các Flip-Flop hoặc những bộ nhớ hoàn chỉnh hơn.



Hình 1.1 Cấu trúc tổng quan của FPGA

b.Vi mạch FPGA được cấu thành từ các bộ phận:

- Các khối logic cơ bản lập trình được (logic block):Phần tử chính của FPGA là các khối logic (logic block). Khối logic được cấu thành từ LUT và một phần tử nhớ đồng bộ flip-flop. LUT (Look up table) là khối logic có thể thực hiện bất kì hàm logic nào từ 4 đầu vào, kết quả của hàm này tùy vào mục đích mà gửi ra ngoài khối logic trực tiếp hay thông qua phần tử nhớ flip-flop.



Hình 1.2 Khối logic lập trình được của FPGA

Trong tài liệu hướng dẫn của các dòng FPGA của Xilinx còn sử dụng khái niệm SLICE, 1 Slice gồm 4 khối logic tạo thành, số lượng các Slices thay đổi từ vài nghìn đến vài chục nghìn tùy theo loại FPGA.

- Hệ thống mạch liên kết lập trình được :Mạng liên kết trong FPGA được cấu thành từ các đường kết nối theo hai phương ngang và đứng, tùy theo từng loại FPGA mà các đường kết nối được chia thành các nhóm khác nhau, ví dụ trong XC4000 của Xilinx có 3 loại kết nối: ngắn, dài và rất dài. Các đường kết nối được nối với nhau thông qua các khối chuyển mạch lập trình được (programable switch), trong một khối chuyển mạch chứa một số lượng nút chuyển lập trình được, đảm bảo cho các dạng liên kết phức tạp khác nhau.

- Khối vào/ra (IO Pads) :Khối vào/ra nhiều hay ít là tuỳ thuộc vào từng loại FPGA. Chúng có thể được kết nối với các thiết bị bên ngoài như LED, USB, RS232, RAM....tuỳ theo mục đích sử dụng
- Các phần tử tích hợp sẵn:Ngoài các khối logic, tùy theo các loại FPGA khác nhau mà có các phần tử tích hợp thêm khác nhau, ví dụ để thiết kế những ứng dụng SoC, trong dòng Virtex 4, 5 của Xilinx có chứa nhân xử lý PowerPC, hay cho những ứng dụng xử lý tín hiệu số trong FPGA được tích hợp các DSP Slice là bộ nhân, cộng tốc độ cao, thực hiện hàm A*B+C, ví dụ dòng Virtex của Xilinx chứa từ vài chục đến hàng trăm DSP slices với A, B, C 18-bit.

1.2 NGÔN NGỮ VHDL

1.2.1 Giới thiệu về VHDL

VHDL là ngôn ngữ mô tả phần cứng cho các mạch tích hợp tốc độ rất cao, là một loại ngôn ngữ mô tả phần cứng được phát triển dung cho trương trình VHSIC(Very High Speed Itergrated Circuit) của bộ quốc phòng Mỹ. Mục tiêu của việc phát triển VHDL là có được một ngôn ngữ mô phỏng phần cứng tiêu chuẩn và thống nhất cho phép thử nghiệm các hệ thống số nhanh hơn cũng như cho phép dễ dàng đưa các hệ thống đó vào ứng dụng trong thực tế. Ngôn ngữ VHDL được ba công ty Intermetics, IBM và Texas Instruments bắt đầu nghiên cứu phát triển vào tháng 7 năm 1983. Phiên bản đầu tiên được công bố vào tháng 8-1985. Sau đó VHDL được đề xuất để tổ chức IEEE xem xét thành một tiêu chuẩn chung. Năm1987 đưa ra tiêu chuẩn về VHDL (tiêu chuẩnIEEE-1076-1987).

VHDL được phát triển để giảiquyết các khó khăn trong việc phát triển, thay đổi và lập tài liệu cho các hệ thống số. VHDL là một ngôn ngữ độc lập không gắn với bất kỳ một phương pháp thiết kế, một bộ mô tả hay công nghệ phần cứng nào. Người thiết kế có thể tự do lựa chọn công nghệ, phương pháp thiết kế trong khi chỉ sử dụng một ngôn ngữ duy nhất.

1.2.2 Các ưu điểm VHDL

Chương trình trong VHDL có thể được viết theo nhiều cấu trúc khác nhau: Ngẫu nhiên, tuần tự, nối chân, định thời chỉ rõ, ngôn ngữ sinh dạng sóng.

- VHDL là một ngôn ngữ phân cấp, hệ thống số có thể được mô phỏng như một kết nối các khối mà các khối này được thực hiện bởi các khối con khác nhỏ hơn.

- Cung cấp một cách mềm dẻo các phương thức thiết kế trên xuống, dưới lên, hoặc tổ hợp cả hai.
- Cung cấp cả hai mode đồng bộ và không đồng bộ.
- Linh hoạt trong kĩ thuật mô phỏng số như sử dụng biểu đồ trạng thái, thuật toán, các hàm Boolean.
- Có tính đại chúng: VHDL được phát triển dưới sự bảo trợ của chính phủ Mỹ và hiện nay là một tiêu chuẩn của IEEE. VHDL được sự hỗ trợ của nhiều nhà sản xuất

thiết bị cũng như nhiều nhà cung cấp công cụ thiết kế mô phỏng hệ thống.

- VHDL cung cấp 3 kiểu mẫu viết khác nhau: structural, dataflow và behavioral.
- Không giới hạn về độ lớn của thiết kế khi sử dụng ngôn ngữ.
- VHDL hoàn toàn độc lập với công nghệ chế tạo phần cứng. Một mô tả hệ thống dùng VHDL thiết kế ở mức cổng có thể được chuyển thành các bản tổng hợp mạch khác nhau tuỳ thuộc công nghệ chế tạo phần cứng mới ra đời nó có thể được áp dụng ngay cho các hệ thống đã thiết kế .
- Khả năng định nghĩa kiểu dữ liệu mới cung cấp một công cụ hữu hiệu cho thiết kế và mô phỏng công nghệ mới với một mức rất cao

1.2.3 Cấu trúc một mô hình hệ thống sử dụng ngôn VHDL

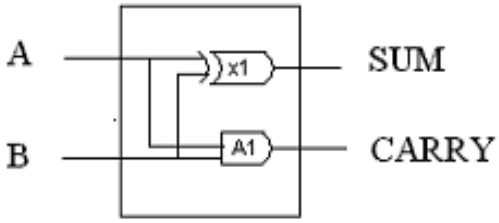
VHDL là ngôn ngữ mô tả phần cứng do vậy mà nó có thể được sử dụng để làm mô hình của một hệ thống số. Hệ thống số có thể đơn giản là các cổng logic hay phức tạp như một hệ thống hoàn chỉnh. Các khối xây dựng nên ngôn ngữ VHDL gọi là các khối thiết kế. Có 3 khối thiết kế chính:

- Khai báo Entity (Thực thể)
- Khai báo Architecture (Kiến trúc)
- Khai báo Configuration (Cấu hình)
- Đôi khi ta sử dụng các gói (Packages) và mô hình kiểm tra hoạt động của hệ thống (Testbench).

a.Entity(Thựcthể)

Khai báo thực thể trong VHDL là phần định nghĩa các chỉ tiêu phía ngoài của một phần tử hay một hệ thống. Khai báo Entity là chỉ ra tên của

Entity và liệt kê các cổng vào/ra. Các cổng là các (dây) tín hiệu mà qua đó entity giao tiếp với môi trường bên ngoài. Ví dụ, một mạch bán tổng được chỉ ra ở hình dưới đây : :



Hình 1.3 Mạchbántổng

Khai báo Entity như sau:

```
entity HALF-ADDER is
    port ( A, B : in BIT;
           SUM, CARRY : out BIT);
end HATF-ADDER;
```

Bộ bán cộng này gồm có hai đầu vào là A và B; và hai đầu ra là SUM và CARRY, BIT là một kiểu cấu trúc ngôn ngữ được định nghĩa trước của FPGA

b.Architecture(Kiếntrúc)

Phần thứ 2 trong mã nguồn VHDL là khai báo Architecture. Mỗi một khai báo Entity đều phải đi kèm với ít nhất một Architecture tương ứng. Khai báo Architecture trong chương trình phải kết hợp tên của Architecture và một Entity trong chương trình đó. Phần thân Architecture có thể bao gồm các khai báo về các tín hiệu bên trong, các phần tử bên trong hệ thống, hay các hàm và thủ tục mô tả hoạt động của hệ thống. Tên của Architecture là nhãn được đặt tuỳ theo người viết chương trình. Cấu trúc bên trong của Architecture có thể được viết theo một trong số các kiểu mẫu sau:

-Tập hợp kết nối bên trong của các thiết bị.

-Tập các câu lệnh ngẫu nhiên

-Tập các câu lệnh tuần tự.

-Kết hợp của ba dạng trên.

Các kiểu mô hình này sẽ được mô tả cụ thể như sau:

 Kiểu kiến trúc

Kiểu này được xây dựng dựa trên một tập các thành phần được kết nối. Ví dụ như bộ bán tổng được chỉ ra sau đây:

```
architecture HA-STRUCTURE of HALF-ADDER is
        component XOR2
            port (X,Y : in BIT; N: 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;
```

 Kiểu luồng dữ liệu

Trong kiểu này, luồng dữ liệu qua Entity trước tiên được biểu diễn bằng các phép gán đồng thời. Kiểu luồng dữ liệu của bộ bán cộng được chỉ ra trong ví dụ sau:

```
        Architecture DATAFLOW of HALF-ADDER is
        Begin
            SUM <= A xor B after 8ns;
            CARRY <= A and B after 4ns;
        End DATAFLOW;
```

Trong ví dụ này kiểu luồng dữ liệu sử dụng hai phép gán tín hiệu đồng thời (hoặc gán nối tiếp). Trong phép gán cho tín hiệu thì ký hiệu gán là "<=". Giá trị của biểu biểu thức bên phải được gán cho tín hiệu bên phía tay trái. Một phép gán đồng thời được thực hiện chỉ khi có bất kỳ tín hiệu trong biểu thức phía phải có sự thay đổi, tức là giá trị tín hiệu thay đổi. Thông tin trễ cũng có thể được thêm vào phép gán bằng cách sử dụng mệnh đề "after" .

 Kiểu behavior

Kiểu behavior chỉ ra cách thức hoạt động của một entity như là một tập hợp lệnh được thực hiện theo kiểu nối tiếp bằng cách sử dụng process. Chúng không chỉ ra rõ ràng cấu trúc của entity mà chỉ ra chức năng của nó. Ví dụ sau xem xét kiểu behavior của bộ bán tổng.

```
Architecture BEHAVIOR of HALF-ADDER is
Begin
        Process (A,B)
                Variable  X,Y : BIT;
        Begin
                X:=A;
                Y:=B;
                SUM<=X xor Y;
                CARRY <= X and Y;
        End process;
End BEHAVIOR;
```

Một process cũng có một phần để khai báo (trước từ khóa "begin") và một phần để trình bày (giữa từ khóa "begin" và "process"). Các lệnh bên trong phần trình bày này được thực hiện theo kiểu nối tiếp. Danh sách các tín hiệu được chỉ ra trong dấu ngoặc sau từ khóa "process" tạo thành một danh sách "nhạy". Tức là, khi có sự thay đổi của bất kỳ giá trị nào trong danh sách này thì mới thực hiện các lệnh trong process. Tuy nhiên, tất cả các process trong một chương trình thì đều thực hiện đồng thời.

Khai báo biến (bắt đầu bằng từ khóa "variable"), trong ví dụ này có hai biến X và Y. Các biến được gán với ký hiệu là ":=" và giá trị của vế phải gán cho giá trị biến bên trái.

- Kiểu hỗn tạp.

Kiểu hỗn tạp là kiểu kết hợp cả ba kiểu trên. Tức là, bên trong một architecture, chúng ta có thể sử dụng cả ba cách trình bày trên.

c.Configuration(Cấuhình)

Khai báo Configuration dùng để lựa chọn một trong các thân Architecture có sẵn mà một Entity có hoặc để gắn các khối vào Entity.

Nếu cho dạng cấu trúc, Configuration có thể được xem như liệt kê các thành phần cho khối mô hình. Cho mỗi khối thì Configuration chỉ rõ Architecture nào cho Entity từ nhiều Architecture. Khi Configuration cho tổng hợp Entity- Architecture thì được biên dịch vào thư viện và một thực thể mô phỏng được tạo ra. Ví dụ khai báo Configuration trong bộ bán tổng như sau:

```
Library CMOS-LIB, MY-LIB;
Configuration CONFIG of HALF-ADDER is
For HA-STRUCTURE
For  X1:XOR2
        Use entity CMOS-LIB.XOR-GATE (DATAFLOW);
End for;
For A1 : AND2
        Use configuration MY-LIB.AND-CONFIG;
End for;
End for; End CONFIG;
```

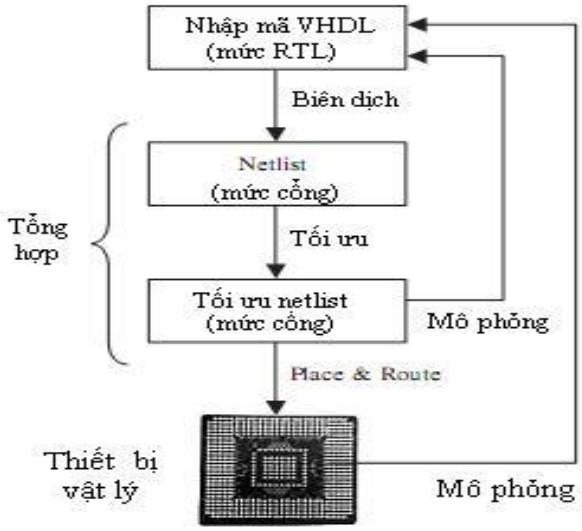
d.Package(Gói)

Mục đích cơ bản của Package là gói gọn các phần nhỏ có thể được sử dụng trong nhiều thiết kế. Package là một biện pháp thường dùng để lưu dữ thông tin có thể được sử dụng trong nhiều Entity. Mối quan hệ trong Package cho phép dữ liệu có thể được tham chiếu bởi những Entity khác. Vì thế dữ liệu có thể được chia sẻ.

Một Package gồm hai phần: Phần khai báo và phần thân (Body). Phần khai báo định nghĩa giao diện cho Package, bằng một cách tương tự như định nghĩa của Entity. Thân của Package chỉ rõ sự biến đổi quan hệ trong Package giống như trong Architecture.

VHDL là không giống như cách thực thi chương trình một cách tuần tự như chương trình của PC, các lệnh của VHDL được thực hiện một cách đồng thời. Vì lí do này, người ta thường gọi là "mã VHDL" chứ không gọi là "chương trình VHDL". Trong VHDL, chỉ các lệnh nằm trong PROCESS, FUNCTION hoặc PROCEDURE mới được thực thi một cách tuần tự. Như đã đề cập ở trên, một trong những ưu điểm của VHDL là nó cho phép tổng hợp một mạch hay một hệ thống trong một thiết bị khả trình (như PLD hoặc

FPGA) hoặc trong một chip ASIC. Các bước thực hiện một project được chỉ ra trong hình dưới đây.



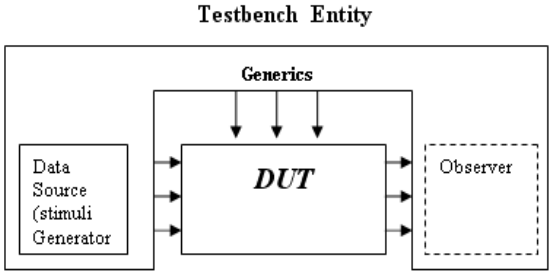
Hình 1.4 Các bước thực hiện một project

Thiết kế được bắt đầu bằng việc viết mã VHDL và lưu vào file có đuôi ".vhd" có cùng tên với tên của ENTITY. Bước đầu tiên trong quá trình tổng hợp là biên dịch. Biên dịch là quá trình chuyển từ ngôn ngữ VHDL bậc cao (mô tả mạch ở mức RTL – mức chuyển thanh ghi) sang dạng danh sách kết nối (netlist) ở mức gate. Bước thứ hai là tối ưu, được thực hiện trên danh sách kết nối mức gate để đạt được sự tối ưu về tốc độ hoặc tối ưu về diện tích sắp đặt. Ở giai đoạn này, thiết kế có thể được mô phỏng. Cuối cùng một phần mềm Place-và-route sẽ tạo ra sự sắp đặt (layout) vật lý cho một thiết bị PLD/FPGA hoặc sẽ tạo ra mặt nạ (mask) cho chip ASIC.

e.Mô hình kiểm tra hoạt động(Testbench)

Một trong các nhiệm vụ rất quan trọng là kiểm tra bản mô tả thiết kế. Kiểm tra một mô hình VHDL được thực hiện bằng cách quan sát hoạt động của nó trong khi mô phỏng và các giá trị thu được có thể đem so sánh với yêu cầu thiết kế.Môi trường kiểm tra có thể hiểu như một mạch kiểm tra ảo. Môi trường kiểm tra sinh ra các tác động lên bản thiết kế và cho phép quan sát hoặc so sánh kết quả hoạt động của bản mô tả thiết kế. Thông thường thì các bản mô tả đều cung cấp chương trình thử. Nhưng ta cũng có thể tự xây dựng chương trình thử (testbench). Mạch thử thực chất là sự kết hợp của tổng hợp nhiều thành phần. Nó gồm ba thành phần. Mô hình VHDL đã qua kiểm tra, nguồn dữ liệu và bộ quan sát. Hoạt động của mô hình VHDL được kích

thích bởi các nguồn dữ liệu và kiểm tra tính đúng đắn thông qua bộ quan sát.



Hình 1.5 Sơ đồ tổng quát của một chương trình thử(Testbench)

Trong đó: DUT: (device under test) mô hình VHDL cần kiểm tra

Observer: khối quan sát kết quả

Data source: nguồn dữ liệu (khối tạo ra các tín hiệu kích thích)

1.2.4 Các đối tượng và các kiểu dữ liệu trong VHDL

a.Đối tượng trong VHDL

Trong ngôn ngữ VHDL gồm có 3 đối tượng là: tín hiệu - signal, biến - variable, hằng - constant, mỗi đối tượng được khai báo dựa vào từ khóa tương ứng và chúng có mục đích sử dụng như sau:

+ Tín hiệu – Signal: là đối tượng để biểu diễn đường kết nối các giữa các cổng vào/ra của thực thể, giữa các cổng vào/ra của các khối thành phần phần cứng xuất hiện trong thực thể… Chúng là phương tiện truyền dữ liệu động giữa các thành phần của thực thể.

Tín hiệu có tính toàn cục rất cao, chúng có thể được khai báo trong package (tín hiệu toàn cục, được sử dụng bởi một số thực thể), khai báo trong thực thể - Entity (tín hiệu nội bộ dùng trong thực thể, có thể được tham chiếu bởi bất kỳ kiến trúc nào của thực thể đó), khai báo trong kiến trúc – Architecture (tín hiệu nội bộ dùng trong kiến trúc, có thể được sử dụng trong bất cứ cấu trúc lệnh nào trong kiến trúc). Các tín hiệu có thể được sử dụng nhưng không được khai báo trong tiến trình – process, trong chương trình con. Vì tiến trình và chương trình con là thành phần cơ sở của mô hình và chúng được coi như các hộp đen. Cú pháp khai báo tín hiệu như sau:

Signal tên_tín_hiệu {,tên_tín_hiệu}:kiểu_dữ_liệu [:=giá_trị_khởi_tạo];

Ví dụ: Signal a,b,c: Bit:=’1’;

Signal y, reg: std_logic_vector(3 downto 0):=”0000”;

+ Biến – Variable: là đối tượng cục bộ được sử dụng để chứa các kết quả trung gian. Biến chỉ được khai báo và sử dụng trong process và trong chương trình con. Cú pháp khai báo của biến cũng tương tự như khai báo tín hiệu:

Variable tên_biến {,tên_biến}: kiểu_dữ_liệu [:=giá_trị_khởi_tạo];

Ví dụ: variable x: Bit:=’1’;

variable Q: std_logic_vector(3 downto 0);

Nếu không được khởi tạo giá trị ban đầu biến sẽ nhận giá trị khởi tạo ban đầu là giá trị thấp nhất trong các giá trị thuộc miền xác định của kiểu dữ liệu. Tín hiệu cũng có thể chứa dữ liệu nhưng chúng lại không được sử dụng vì những lý do sau:

Việc sử dụng biến hiệu quả hơn vì giá trị của biến được gán ngay lập tức trong process khi tín hiện chỉ được lập kế hoạch để thực hiện và chỉ được cập nhật toàn bộ sau khi kết thúc process. Biến chiếm ít bộ nhớ hơn trong khi tín hiệu cần nhiều thông tin để có thể lập kế hoạch thực hiện cũng như để chứa các thuộc tính của tín hiệu. Sử dụng tín hiệu yêu cầu có lệnh wait để thực hiện đồng bộ phép gán tín hiệu với phép lặp thực hiện theo cách sử dụng quen thuộc.

+ Hằng –constant: là đối tượng hằng được gán cho các giá trị cụ thể của một kiểu khi được tạo ra và không đổi trong toàn bộ quá trình thực hiện. Hằng cũng có tính toàn cục giống như tín hiệu và có thể được khai báo trong package, entity, architecture, proceduce, process… Cú pháp khai báo hằng:

constant tên_hằng {,tên_hằng} : kiểu_dữ_liệu := giá_trị_khởi_tạo;

Ví dụ: constant GND: std_logic:=’0’;

constant PI: real:=3.1414;

Tóm lại: Các đối tượng trong VHDL có mục đích sử dụng, phạm vi sử dụng khác nhau, nhưng chúng có cú pháp khai báo chung như sau:

Đối_tượng tên_đối_tượng: kiểu_dữ_liệu {:=giá_trị_khởi_tạo}

Các đối tượng khi khai báo phải được xác định kiểu dữ liệu tương ứng. VDHL định nghĩa nhiều kiểu dữ liệu khác nhau để phù hợp với việc mô tả, thiết kế, mô phỏng các hệ thống số khác nhau trong thực tế.

b.Kiểu dữ liệu trong VHDL

- **Trong VHDL có 4 dạng dữ liệu:**

Vô hướng: gồm các dữ liệu có giá trị đơn như bit, boolean, integer, real, physical, character, std_logic và std_ulogic, enumerated (kiểu liệt kê)... Kiểu ghép: các dữ liệu dưới dạng một nhóm các thành phần như mảng, bảng ghi (record). Bit_logic_vector, std_logic_vector và String đều là những dạng dữ liệu ghép đã được định nghĩa sẵn. 2-D Arrays: các dữ liệu có dạng mảng 2

chiều, được tạo nên từ 1 mảng của một mảng 1 chiều (hay một bản ghi). VHDL Subtypes: dạng dữ liệu con do người dùng tự định nghĩa dựa trên những dạng có sẵn.

Các kiểu dữ liệu đã được định nghĩa trong gói Standard chứa trong thư viện chuẩn Standard Library của VHDL là: bit, boolean, integer, real, physical, character, std_logic and std_ulogic, Bit_logic_vector, std_logic_vector và String và một số kiểu dữ liệu con. Cú pháp chung định nghĩa kiểu dữ liệu như sau:

Type Tên_kiểu is giới_hạn_giá_trị_của_kiểu

✓ Kiểu vô hướng

- Kiểu Bit: Kiểu liệt kê với 2 giá trị '0' và '1'. Kiểu Bit đã được định nghĩa như sau:

Type Bit is ('0', '1');

- Kiểu Boolean: Kiểu liệt kê với 2 giá trị false và true. Kiểu Boolean đã được định nghĩa như sau: Type Boolean is (false, true);

- Kiểu Integer: Kiểu số nguyên với những giá trị dương hoặc âm, độ lớn mặc định là 32 bit với giới hạn giá trị: từ -2147483647 đến +2147483647. Khi sử dụng có thể giới hạn miền xác định theo giới hạn giảm dần dùng từ khóa downto hoặc tăng dần dùng từ khóa to:

signal A: integer range 0 to 7; -- A số nguyên 3 bit

variable B: integer range 15 downto 0; -- B số nguyên 4 bit

signal B: integer range 15 downto -15; -- B số nguyên 5 bit

Các cách biểu diễn số nguyên dạng thập phân:

+ digit[underline]digit, ví dụ: 0, 1, 123_456_789 , -123_5678…

+ digit(E)digit, ví dụ: 987E6 (=987.106-)…

Các cách biểu diễn dưới dạng cơ số xác định:

+ base#based_integer#[exponent], ví dụ: 2#1100_0100#, 16#C4#, 4#301#E1, (=196)

- Kiểu Real: Kiểu số thực có giới hạn từ -1.0E+38 đến 1.0E+38, khác với kiểu integer kiểu số thực khi sử dụng thường được định nghĩa thành kiểu dữ liệu riêng và có giới hạn miền xác định:

signal a: Real:=-123E-4;

type CAPACITY is range -25.0 to 25.0 ;

signal Sig_1: CAPACITY:= 3.0 ;

type PROBABILITY is range 1.0 downto 0.0;

constant P: PROBABILITY:= 0.5 ;

Các cách biểu diễn số thực:

+ Biểu diễn dưới dạng thập phân: integer[.integer][exponent],

ví dụ: 0.0, 0.5, 1.1234_5678, 12.4E-9…

+ Biểu diễn dưới dạng cơ số xác định:

base#based_integer[.based_integer]#[exponent]

Ví dụ: 2#1.111_1111_111#E+11, 16#F.FF#E2 (=4095.0)

- Kiểu Character: Kiểu kiểu ký tự, liệt kê với miền xác định là tập hợp các ký tự ASCII. Biểu diễn của giá trị Character: ‘A’, ‘a’, ‘*’, ‘ ‘, NUL, ESC…

- Kiểu Vật lý – Physical: được sử dụng để biểu diễn các đại lượng vật lý như khoảng cách, điện trở, dòng điện, thời gian… Kiểu vật lý cung cấp đơn vi cơ bản và các đơn vị kế tiếp được định nghĩa theo đơn vị cơ bản, đơn vị nhỏ nhất có thể biểu diễn được là đơn vị cơ bản. Trong thực việc chuẩn Time (kiểu dữ liệu thời gian) là kiểu vật lý duy nhất đã được định nghĩa.

type Time is range <xác_định giới hạn>

Ví dụ sử dụng:constant Tpd: time:= 3ns ;...Z <= A after Tpd ; units

fs; -- Đơn vị cơ bản

ps = 1000 fs; ns = 1000 ps; us = 1000 ns;ms = 1000 us;sec = 1000 ms;

min = 60 sec; hr = 60 min;

End Units;

- Kiểu std_logic và std_ulogic: kiểu dữ liệu logic nhiều mức đã được định nghĩa trong gói std_logic_1164, so với kiểu Bit thì chúng có thể mô tả chính xác và chi tiết hơn cho các phần cứng số, chúng còn xác định được cường độ khác nhau của các tín hiệu.

type std_ulogic is (‘U’,-- UninitializeX’, -- Forcing Unknown‘0’, -- Forcing Zero‘1’, -- Forcing One‘Z’, -- High Impedance‘W’, -- Weak Unknown‘L’, -- Weak Zero‘H’, -- Weak One‘-‘ -- Don’t Care) ;

type std_logic is (‘U’, -- UninitializeX’, -- Forcing Unknown‘0’, -- Forcing Zero‘1’, -- Forcing One‘Z’, -- High Impedance‘W’, -- Weak Unknown‘L’, -- Weak Zero‘H’, -- Weak One‘-‘ -- Don’t Care) ;

Hai kiểu dữ liệu std_logic và std_ulogic tương tự nhau, chúng chỉ khác nhau ở chỗ là kiểu std_ulogic không có hàm phân dải (unresolved) – hàm quyết định giá trị tín hiệu, do đó sẽ có lỗi khi các tín hiệu kiểu std_ulogic được nối chung vào 1 điểm. Thư viện cũng cung cấp hàm phát hiện lỗi này của các tín hiệu kiểu std_ulogic.

signal A,B,C,Res_Out: std_logic ;signal Out_1: std_ulogic ;Out_1 <= A ;Out_1 <= B ;Out_1 <= C ;CBARes_Out <= A;Res_Out <= B;Res_Out <= C;Res_OutCBAOut_1XCó lỗiThực hiện được

(Ký hiệu “<=” dùng ở trên là lệnh gán tín hiệu, lệnh gán tín hiệu thực hiện được với 2 dữ liệu cùng kiểu, cùng độ lớn, giá trị của tín hiệu bên phải sẽ được gán cho tín hiệu bên trái).

- Kiểu dữ liệu liệt kê tự định nghĩa: Kiểu dữ liệu liệt kê, do người sử dụng tự định nghĩa, cho phép mô tả rất sáng sủa, và linh hoạt cho các mô hình phần cứng số với mức độ trừu tượng cao. Kiểu dữ liệu này dùng nhiều mô tả đồ hình trạng thái, các hệ thống phức tạp…

Ví dụ:

type My_State is(RST, LOAD, FETCH, STOR, SHIFT) ;

signal STATE, NEXT_STATE: My_State ;

✓ Kiểu dữ liệu ghép

Tương tự các ngôn ngữ lập trình, VHDL cũng có các kiểu dữ liệu ghép là nhóm các phần tử dữ liệu theo dạng mảng (array) hoặc bảng ghi (record).

+ Mảng – Array:

Mảng là nhóm nhiều phần tử có cùng kiểu dữ liệu với nhau thành đối tượng duy nhất. Mỗi phần tử của mảng có thể được truy cập bằng một hoặc nhiều chỉ số của mảng. Cú pháp định nghĩa kiểu dữ liệu mảng như sau:

Type tên_mảng is array (khoảng _của _chỉ số) of kiểu_của_phần_tử;

Ví dụ một số cách khai báo và sử dụng dữ liệu mảng:

type WORD is array (3 downto 0) of std_logic ;

signal B_bus: WORD ;

type DATA is array (3 downto 0) of integer range 0 to 9 ;

signal C_bus: DATA ;

Các kiểu dữ liệu mảng đã được định nghĩa trong thư viện chuẩn của VHDL là: Bit_logic_vector (mảng dữ liệu kiểu Bit), std_logic_vector (mảng dữ liệu kiểu std_logic) và String (mảng dữ liệu kiểu Chacracter). Một số ví dụ sử dụng các kiểu dữ liệu này như sau:

signal My_BusA, My_BusB: bit_vector (3 downto 0);

signal My_BusC: bit_vector (0 to 3) ;

signal Data_Word: std_logic_vector (11 downto 0);

variable Warning2: string(1 to 30):= “Unstable, Aborting Now”;

constant Warning3: string(1 to 20):= “Entering FSM State2”;

Một số phép toán thao tác với phần tử mảng:

- Phép gán cho mảng: 2 mảng phải cùng kiểu, cùng độ lớn, phép gán sẽ thực hiện gán theo từng phần tử theo thứ tự từ trái sang phải:

Data_Word <= ”101001101111” ;

Data_Word <= X”A6F”;

Data_Word <= O”5157”;

Data_Word <= B”1010_0110_1111” ;

Cách biểu diễn số liệu bit_vector và std_logic_vector: B|O|X ”giá_trị” (dùng dấu nháy kép). Trong đó B: Binary -Kiểu nhị phân, O: Octal - kiểu bát phân, X: hexadecimal.

X”1AF”=B”0001_1010_1111”= B”000_110_101_111”=O”0657”

- Phép gộp ( ): cho phép nhóm cả dữ liệu vô hướng và dữ liệu mảng để thuận tiện cho các phép gán cho mảng:

signal H_BYTE, L_BYTE: std_logic_vector (0 to 7);

signal Q_Out: std_logic_vector (31 downto 0);

signal A, B, C, D : std_logic;

signal WORD: std_logic_vector (3 downto 0);

(A,B,C,D)<=WORD;

Chú ý: Phép gộp ở vế bên trái chỉ dùng với kiểu dữ liệu vô hướng.

WORD <= (2 => '1', 3 => D, others => '0') ;

Q_Out <= (others => '0') ;

WORD <= (A, B, C, D) ;

H_Byte <= (7|6|0=>'1', 2 to 5 => '0');

L_Byte <= (3=>'1', 1 to 2 => '0', 4 to 7 => '1');

Chú ý: "others" có thể được sử dụng khi gán mặc định, nó có ý nghĩa là các tất cả các phần tử còn lại được gán bằng một giá trị nào đó) .

+ Bảng ghi – Record:

Bảng ghi là nhóm nhiều phần tử có kiểu dữ liệu khác nhau thành đối tượng duy nhất.

Mỗi phần tử của bản ghi được truy nhập tới theo tên trường. Các phần tử của bản ghi có thể nhận mọi kiểu của ngôn ngữ VHDL kể cả mảng và bảng ghi.

3012My_BusAMy_BusBMy_BusB<=My_BusA ;30123012My_BusAMy_BusBMy_BusC <= My_BusA ;0;2

Ví dụ định nghĩa kiểu dữ liệu bảng ghi như sau:

type OPCODE is record

PARITY : bit;

ADDRESS: std_logic_vector (0 to 3);

DATA_BYTE: std_logic_vector (7 downto 0);

NUM_VALUE: integer range 0 to 6;

STOP_BITS: bit_vector (1 downto 0);

end record ;

signal TX_PACKET, RX_PACKET : OPCODE;

PARITYADDRESSDATA_BYTENUM_VALUESTOP_BITS;Cách truy nhập và gán dữ liệu cho các trường của bản ghi: Các phần tử của bản ghi được

truy nhập theo tên bản ghi và tên trường, 2 thành phần này được ngăn cách bởi dấu '.'

TX_PACKET <= ('1',"0011","11101010",5,"10") ;

TX_PACKET.ADDRESS <= ("0011");

TX_PACKET <= RX_PACKET;

TX_PACKET.ADDRESS <= RX_PACKET.ADDRESS;

✓ Kiểu dữ liệu mảng 2 chiều (2-D Array)

Mảng 2 chiều là kiểu dữ liệu mảng của các phần tử mạng một chiều hay bảng ghi. Một số ví dụ định nghĩa và khai báo kiểu dữ liệu mảng 2 chiều như sau:

type Mem_Array is array (0 to 3) of std_logic_vector (7 downto 0);

type Data_Array is array (0 to 2) of OPCODE ;

signal My_Mem:Mem_Array ;

signal My_Data:Data_Array ;

Ví dụ ứng dụng dùng mảng 2 chiều khởi tạo một vùng nhớ ROM

constant My_ROM: REM_Array:= (0 => (others=>'1'),

1 => "10100010",

2 => "00001111",

3 => "11110000");

✓ Kiểu dữ liệu con

Là một tập hợp con của các kiểu dữ liệu đã được định nghĩa khác. Phép khai báo kiểu dữ liệu con có thể nằm ở mọi vị trí cho phép khai báo kiểu dữ liệu. Cú pháp khai báo chung:

Subtype Tên_kiểu_dữ_liệu_con is xác_định_kiểu_dữ_liệu_con;

Ví dụ: subtype My_Int is integer range 0 to 255 ;

subtype My_Small_Int is My_Int range 5 to 30 ;

subtype word is bit_vector(31 downto 0)

CHƯƠNG 2 GIỚI THIỆU VỀ SPARTAN-3E KIT BOARD VÀ MÔI TRƯỜNG LẬP TRÌNH ISE 8.2I

2.1 SPARTAN -3E KIT BOARD

2.1.1 Các thành phần của kit Spartan-3E

- Xilinx XC3S500E Spartan-3E FPGA : con chíp chính của KIT
- Xilinx 4 Mbit Platform Flash configuration PROM
- Xilinx 64-macrocell XC2C64A CoolRunner CPLD
- 64 MByte (512 Mbit) of DDR SDRAM, x16 data interface, 100+ MHz
- 16 MByte (128 Mbit) of parallel NOR Flash (Intel StrataFlash)
- 16 Mbits of SPI serial Flash (STMicro)
- 2-line, 16-character LCD screen
- PS/2 mouse or keyboard port
- VGA display port
- 10/100 Ethernet PHY (requires Ethernet MAC in FPGA)
- Two 9-pin RS-232 ports (DTE- and DCE-style)
- On-board USB-based FPGA/CPLD download/debug interface
- 50 MHz clock oscillator
- SHA-1 1-wire serial EEPROM for bitstream copy protection
- Hirose FX2 expansion connector
- Three Digilent 6-pin expansion connectors
- Four-output, SPI-based Digital-to-Analog Converter (DAC)
- Two-input, SPI-based Analog-to-Digital Converter (ADC) with programmable-gain
- pre-amplifier
- ChipScope™ SoftTouch debugging port
- Rotary-encoder with push-button shaft
- Eight discrete LEDs
- Four slide switches

2.1.2Các thông số kỹ thuật và một số hình ảnh

Spartan-3E là họ FPGA mới nhất của Xilinx với nhiều ưu điểm nổi bật. Đầu tiên phải kể đến là khả năng tích hợp của spartan-3E từ 100,000 gates đến 1,6 triệu gates. Ngoài ra, còn một số đặc điểm chính của Spartan-3E là:

Dễ sử dụng , giá thành thấp, tiêu thụ điện ít.

Mật độ tích hợp nhiều phần tử logic(Đây là ưu điểm so với họ Spartan 3).

Tốc độ xung nhịp hệ thống từ 5-300 Mhz.

Năm mức tiêu thụ điện năng (3.3V;2.5V;1.8V;1.5V;1.2V)

Tích hợp tới 376 chân I/O hay 156 cặp tín hiệu khác nhau.

Truyền dữ liệu với tốc độ khá cao.

Hình 2.1 Spartan-3E Starter Kit Board

2.1.3 Cấu trúc Spartan-3E

Các thành phần:

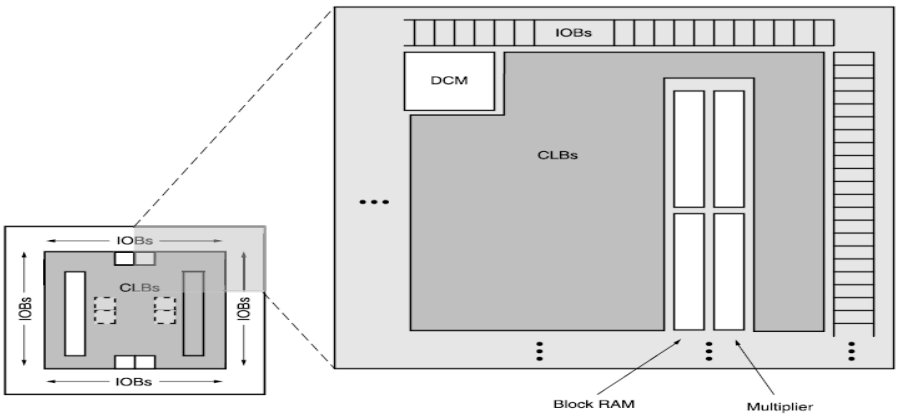
Input/Output Blook (Ios): các khối vào ra

Configurable Logic Blocks (CLBs): được cấu tạo từ look-Up Table(LUTs).

Block RAM: Hỗ trợ 16 Kb RAM trên mỗi Block RAM, số lượng các Block RAM tùy thuộc vào mỗi chip, với XC3S500E có 20 Block 18 bit.

Digital Clock Manager(DCM) Blocks: khối điều khiển xung clk.

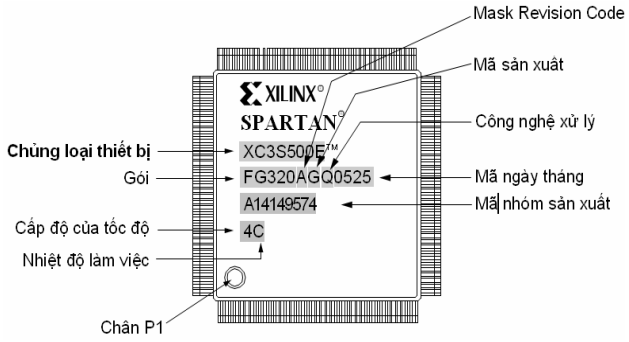
Interconnect: các kết nối.



Hình 2.2Cấu trúc các thành phần của Spartan 3E

Spartan-3E Starter kit board là một công cụ hữu hiệu cho bất kỳ ai đang có ý định thiết kế các sản phẩm dựa trên công nghệ FPGA. Đây là một giải pháp cơ bản cho nhằm tối ưu thời gian và chi phí ban đầu. Nó cho phép chế tạo ngay với giá thành sản phẩm thấp. Bộ kit này là một thiết bị cấu trúc logic có thể người sử dùng lập trình trực tiếp mà không sử dụng bất kỳ một công cụ chế tạo mạch tích hợp nào.

2.1.4 Mã số Chip và ý nghĩa của nó

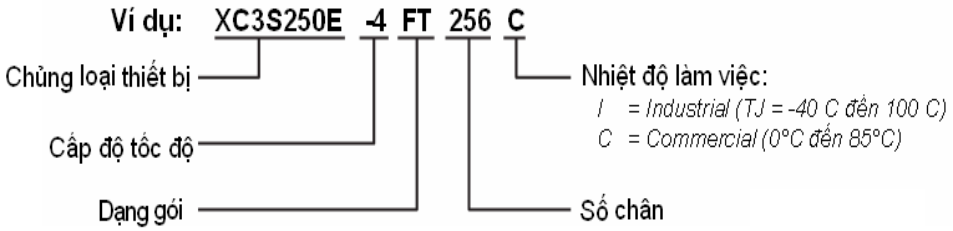


Hình 2.3 Chíp Spartan-3E Xilink với các thông số

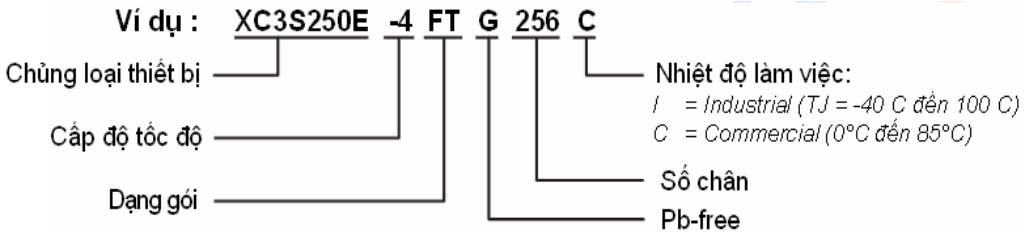
Trên bề mặt chíp được in các mã số, dựa vào các mã số này, người thiết kế mạch có thể biết được khả năng làm việc của bo mạch và lựa chọn để mua thiết bị phù hợp với nhu cầu sử dụng.

Các bo Kit phát triển Spartan-3E được sản xuất ở hai dạng gói cả tiêu chuẩn và Pb-free cho tất cả các thiết bị sản xuất. Các gói Pb-free có chứa thêm ký tự ‘G’ trong mã gói

Standard Packaging



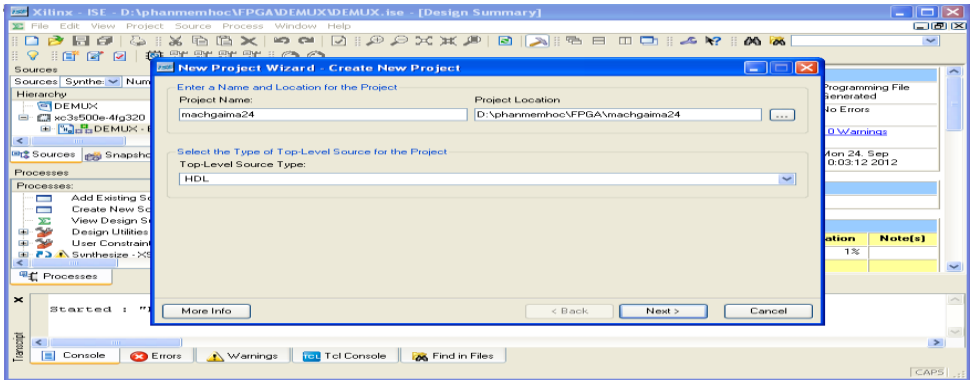
Pb-Free Packaging



2.2 SƠ LƯỢC VỀ ISE 8.2

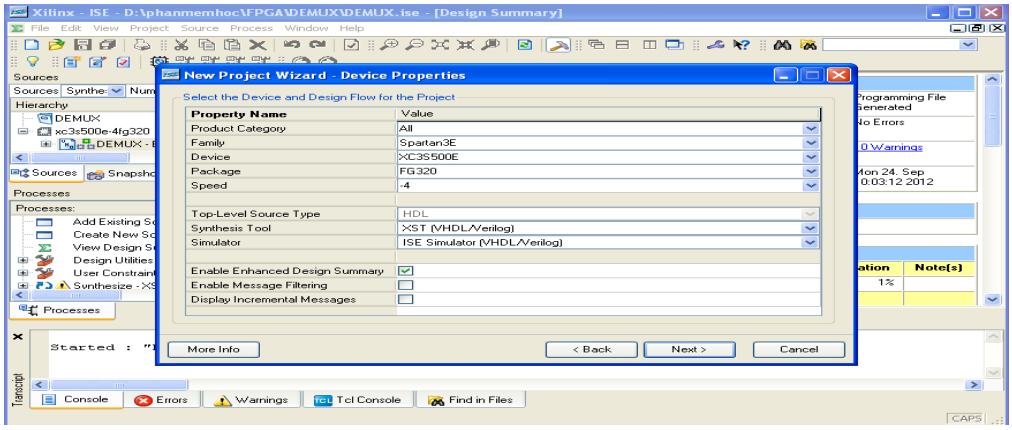
2.2.1Tạo một Project

Vào Start > All Programs > Xilinx ISE 9.2i > Project Navigator để khởi động chương trình. Vào File > New Project cửa sổ hướng dẫn hiện ra như hình 2.4 ở bên dưới:



Hình 2.4 Tạo project mới

Project Name: Đặt tên project. Project location : Nơi chứa project. Click Next, cửa sổ mới hiện ra như hình 2.5 ở bên dưới:



Hình 2.5 Lựa chọn thiết bị cho chương trình

Ô Family : chọn Spartan3E .

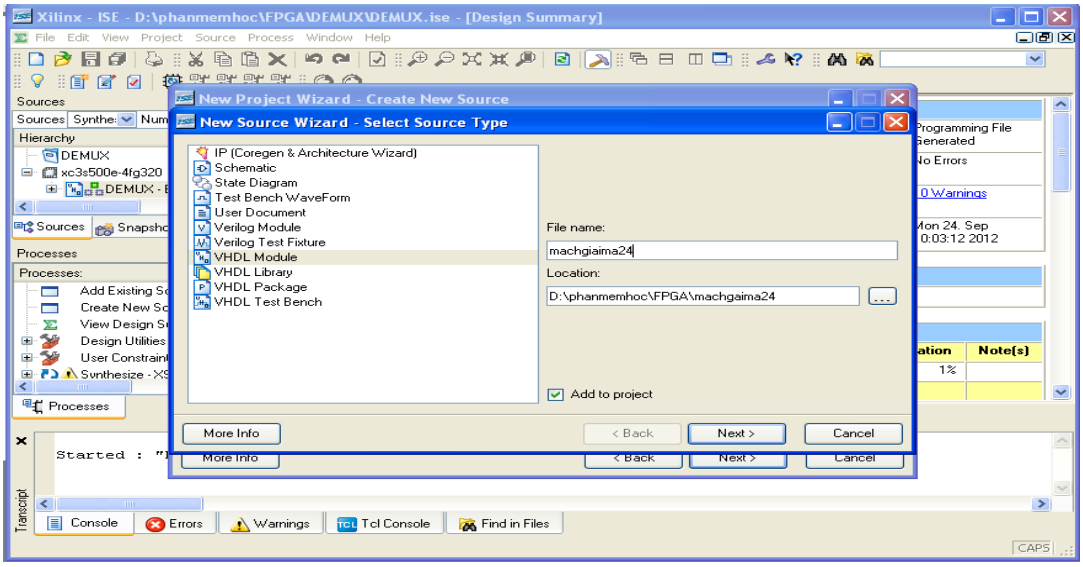
Ô Device : chọn XC3S500E.

Ô Package : chọn FG320.

Tiếp tục click Next , Next cửa mới hiện ra, chọn thanh : New Source.

Cửa sổ mới hiện ra, chọn VHDL Module để viết code vhdl, nếu viết bằng verilog

thì chọn : Verilog Module. cửa sổ hướng dẫn hiện ra như hình 2.6 ở bên dưới:



Hình 2.6 Thêm Module vào chương trình

Chọn tên file vhdl ở ô File name. (ở đây ta đang tạo bộ đếm nên chọn tên làcounter).

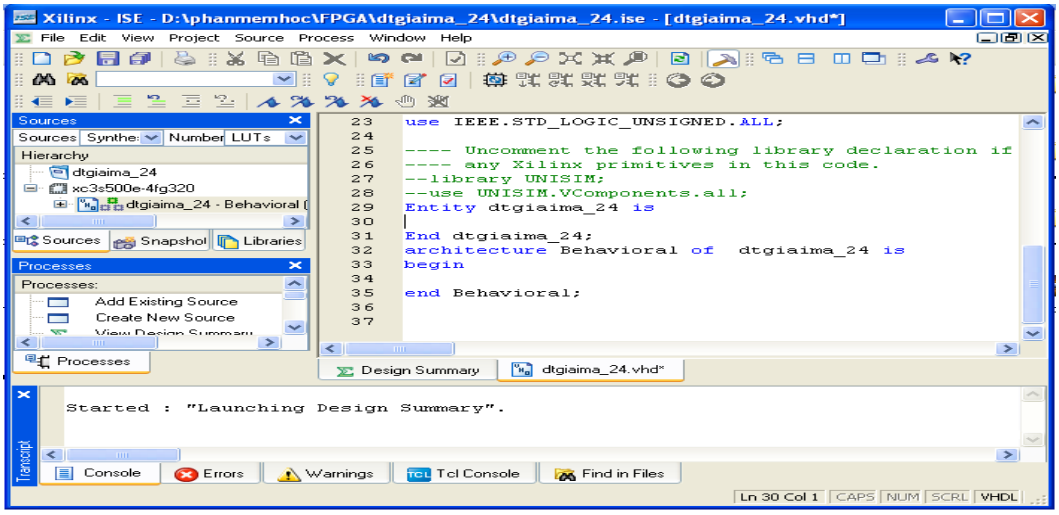
Tiếp tục click Next . Cửa sổ mới hiện ra , ta sẽ chọn giao diện cho vào ra cho khối counter:

Cột Port Name để chọn tên cổng

Cột Direction để chọn chân là lối vào, lối ra hay cả hai vào/ra

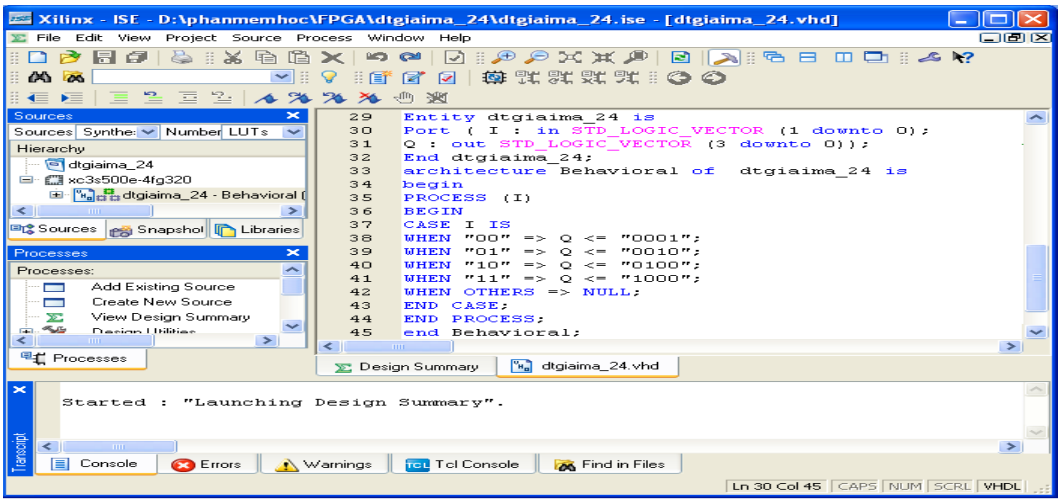
Cột Bus : nếu dùng bus thì tréo vào ô này. Ở đây, bộ đếm của ta có ngõ ra là một port 4 bit nên ta chéo ô này.

Tiếp tục ấn Next -> Next -> Finish , cuối cùng ta được kết quả như hình 2.7 như sau:



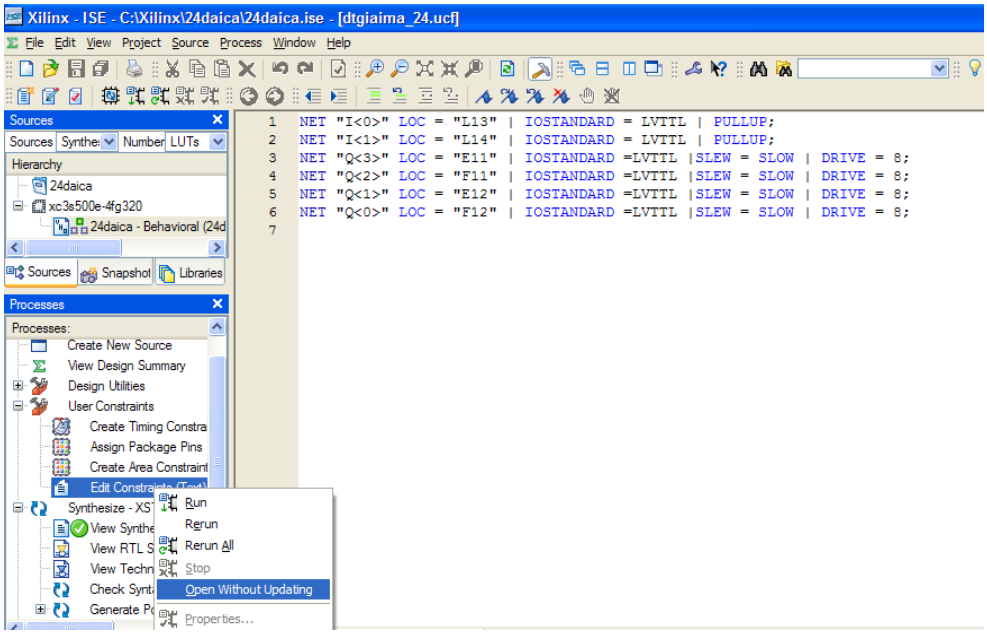
Hình 2.7 Khung chương trình

Sau đó ta viết code vào ta sẽ được như hình 2.8 dưới đây :



Hình 2.8 viết chương trình

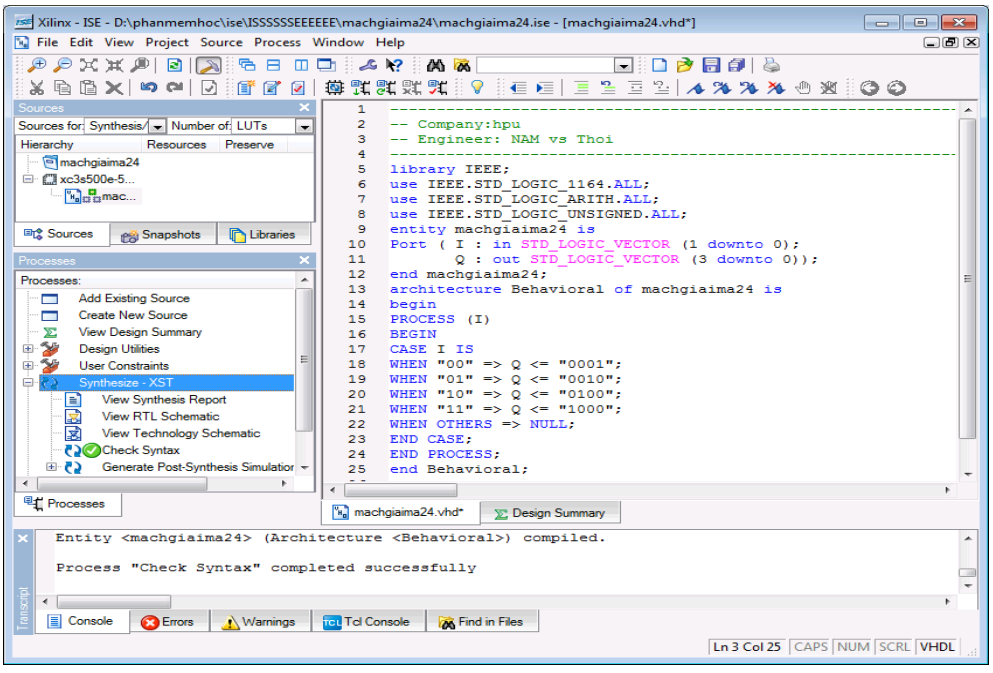
Chú ý việc gắn chân : Chọn User Constraints Edit Constraints(Text) (kích chuột phải vào vào chọn open without updating sau đó gắn chân như hình 2.9 dưới:



Hình 2.9 Gắn chân

✓ Kiểm tra mã nguồn

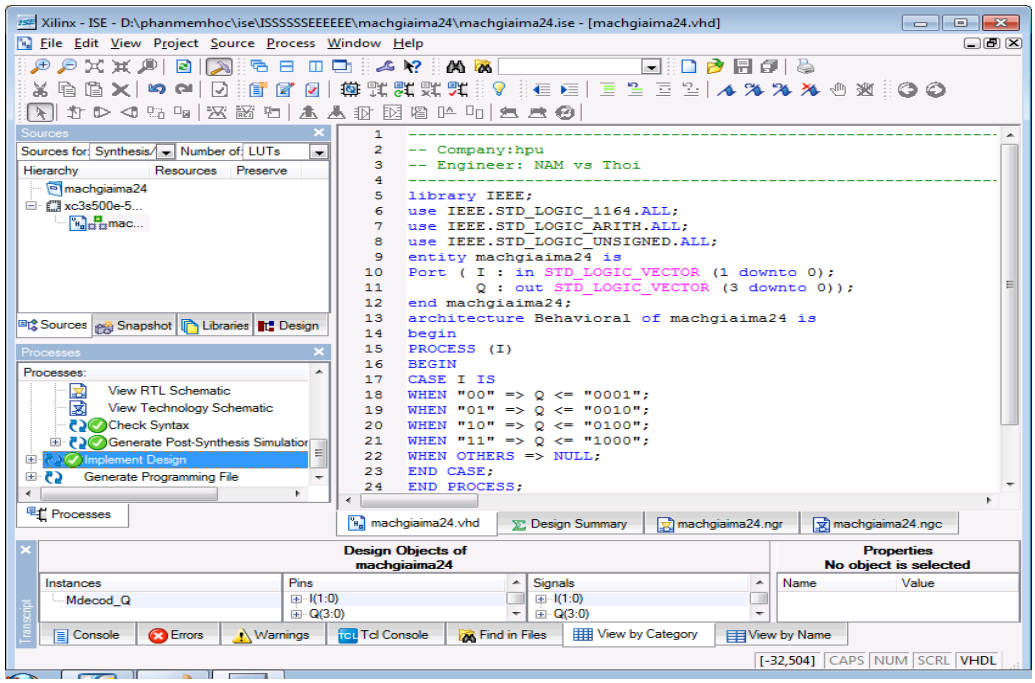
Tại cửa sổ process: Symthesis_XST→ Check Syntax được kết quả như hình 2.10 dưới



Hình 2.10 kiểm tra mã nguỗn

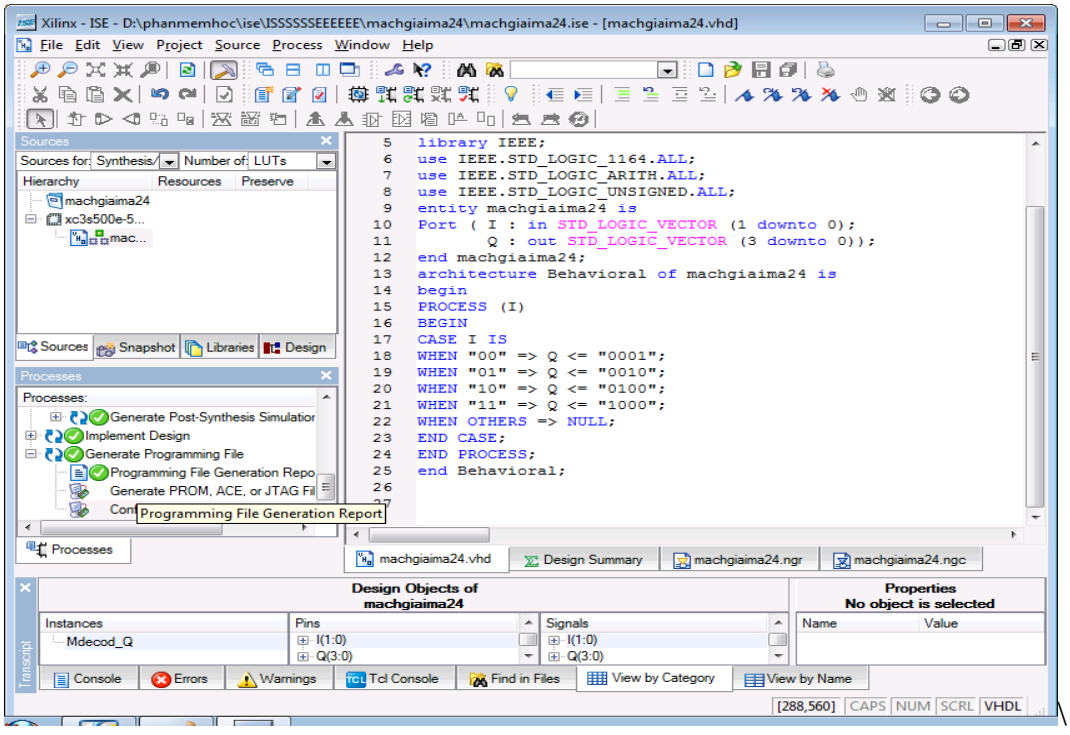
✓ Kết nối với FPGA

Chọn Implement Design được kết quả như hình 2.11 sau :



Hình 2.11 Kiểm tra việc gắn chân

✓ Nạp vào FPGA như hình 2.12 bên dưới:



Hình 2.12 Thực hiện kết nối và nạp chương trình vào kit

Tại Generate Programming File→Configure Deveice

Xuất hiện của sổ ISE iMPACT gán vào khối hình ROM đầu tiên và quan sát kết quả đầu ra trên Kit Spartan 3E

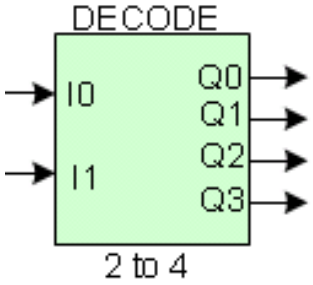
CHƯƠNG 3 THIẾT KẾMẠCH LOGIC VÀ MỘT SỐ ỨNG DỤNG KẾT NỐI CỦA FPGA TRÊN KIT SPARTAN 3E

3.1 Thiết kế mạch logic

a.Thiết kế mạch giải mã 2 đường sang 4 đường với ngõ ra tích cực cao

Bảng trạng thái sơ đồ khối

Ngõ vào		Ngõ ra			
I1	I2	Q3	Q2	Q1	Q0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Chương trình code:

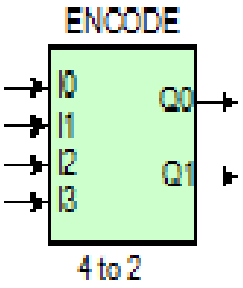
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity machgiaima24 is
Port ( I : in  STD_LOGIC_VECTOR (1 downto 0);
Q : out STD_LOGIC_VECTOR (3 downto 0));
end machgiaima24;
architecture Behavioral of machgiaima24 is
begin
PROCESS (I)
BEGIN
CASE I IS
WHEN "00" => Q <= "0001";
WHEN "01" => Q <= "0010";
```

```
WHEN "10" => Q <= "0100";
WHEN "11" => Q <= "1000";
WHEN OTHERS =>NULL;
END CASE;
END PROCESS;
end Behavioral;
```

b. Thiết kế mạch mã hóa 4 đường sang 2 đường với ngõ vào tích cực cao

Bảng trạng thái: Sơ đồ khối:

Ngõ vào				Ngõ ra	
I3	I2	I1	I0	Q1	Q0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



Chương trình code:

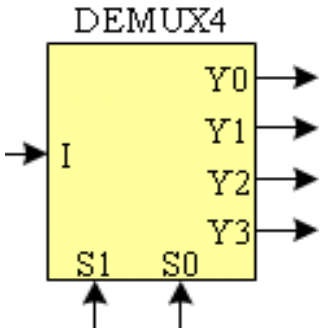
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity machmahoa42 is
Port ( I : in  STD_LOGIC_VECTOR (3 downto 0);
Q : out  STD_LOGIC_VECTOR (1 downto 0));
end machmahoa42;
architecture Behavioral of machmahoa42 is
begin
PROCESS (I)
BEGIN
```

```
CASE I IS
WHEN "0001"  =>  Q <= "00";
WHEN "0010"  =>  Q <= "01";
WHEN "0100"  =>  Q <= "10";
WHEN "1000"  =>  Q <= "11";
WHEN OTHERS => NULL;
END CASE;
END PROCESS;
end Behavioral;
```

c.Thiết kế mạch giải mã đa hợp 1 ngõ vào 4 ngõ ra 2 lựa chọn

Bảng nguyên lý **Sơ đồ khối**

Ngõ vào			Ngõ ra			
I	S1	S0	Y3	Y2	Y1	Y0
I	0	0	0	0	0	I
I	0	1	0	0	I	0
I	1	0	0	I	0	0
I	1	1	I	0	0	0



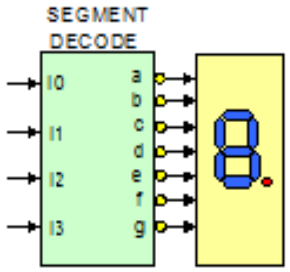
Chương trình code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity madahop is
Port ( I : in  STD_LOGIC;
S : in  STD_LOGIC_VECTOR (1 downto 0);
Q : out  STD_LOGIC_VECTOR (3 downto 0));
```

```
end madahop;
architecture Behavioral of madahop is
begin
PROCESS (I,S)
BEGIN
CASE S IS
WHEN "00" => Q(0) <= I;
WHEN "01" => Q(1) <= I;
WHEN "10" => Q(2) <= I;
WHEN "11" => Q(3) <= I;
WHEN OTHERS => NULL;
END CASE;
END PROCESS;
end Behavioral;
```

d.Thiết kế mạch giải mã led 7 đoạn loại anode chung

Sơ đồ khối :



Bảng trạng thái:

Số	Ngõ vào				Ngõ ra							Số
Tp	I3	I2	I1	I0	g	f	E	d	c	B	A	Hex
0	0	0	0	0	1	0	0	0	0	0	0	40
1	0	0	0	1	1	1	1	1	0	0	1	79
2	0	0	1	0	0	1	0	0	1	0	0	24
3	0	0	1	1	0	1	1	0	0	0	0	30

4	0	1	0	0	0	0	1	1	0	0	1	19
5	0	1	0	1	0	0	1	0	0	1	0	12
6	0	1	1	0	0	0	0	0	0	1	0	02
7	0	1	1	1	1	1	1	1	0	0	0	78
8	1	0	0	0	0	0	0	0	0	0	0	00
9	1	0	0	1	0	0	1	0	0	0	0	10
Tắt	1	0	1	0	1	1	1	1	1	1	1	7F
Tắt	1	0	1	1	1	1	1	1	1	1	1	7F
Tắt	1	1	0	0	1	1	1	1	1	1	1	7F
Tắt	1	1	0	1	1	1	1	1	1	1	1	7F
Tắt	1	1	1	0	1	1	1	1	1	1	1	7F
Tắt	1	1	1	1	1	1	1	1	1	1	1	7F

Chương trình code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity giaimaled7doan is
Port ( I : inSTD_LOGIC_VECTOR (3 downto 0);
Y : outSTD_LOGIC_VECTOR (6 downto 0));
end giaimaled7doan;
architecture Behavioral of giaimaled7doan is
begin
PROCESS (I)
BEGIN
CASE I IS
when “0000” => Y <= “1000000”;
```

when “0001” => Y <= “1111001”;

when “0010” => Y <= “0100100”;

when “0011” => Y <= “0110000”;

when “0100” => Y <= “0011001”;

when “0101” => Y <= “0010010”;

when “0110” => Y <= “0000010”;

when “0111” => Y <= “1111000”;

when “1000” => Y <= “0000000”;

when “1001” => Y <= “0010000”;

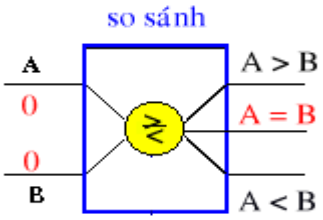
when others => Y <= “1111111”;

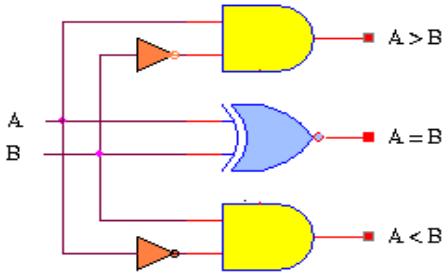
END CASE;

END PROCESS;

end Behavioral;

e.Thiết kế mạch so sánh 2 số 1 bit





Ngõ vào		Ngõ ra so sánh		
A	B	A > B	A = B	A < B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Chương trình:

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sosanh is

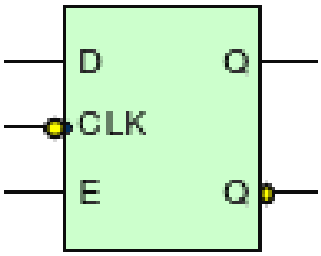
```
port ( I: in STD_LOGIC_VECTOR (1 downto 0);
Q: out STD_LOGIC_VECTOR (2 downto 0));
end sosanh;
architecture Behavioral of sosanh is
begin
process(I)
begin
case I is
when"00"=> Q <="010";
when"01"=>Q <="001";
when "10"=>Q <="100";
when"11"=>Q <="010";
when others => null;
end case;
end process;
end Behavioral;
```

f. Thiết kế Flip Flop D

Bảng chân lý

Ngõ vào			Ngõ ra	
E	Clk	D	Q	QD
0	X	X	Qo	Qdo
1	0	0	Qo	Qdo
1	↓	0	0	1
1	↓	1	1	0

Sơ đồ khối



Chương trình:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity D is
   Port ( D : in  STD_LOGIC;
          E : in  STD_LOGIC;
          CLK : in  STD_LOGIC;
          Q : out  STD_LOGIC;
          QD : out  STD_LOGIC);
end D;
architecture Behavioral of D is
SIGNAL QT: STD_LOGIC;
begin
PROCESS(D,E,CLK)
BEGIN
IF E='1' THEN
IF CLK='0' AND CLK'EVENT THEN QT <= D;
END IF;
END IF;
Q <= QT;
QD <= NOT QT;
END PROCESS;
end Behavioral;
```

3.2MỘT SỐ ỨNG DỤNG KẾT NỐI CỦA FPGA TRÊN KIT SPARTAN 3E

a. LCD kết nối với Spartan_3E

chương t rình:

```
entity lcd3 is
```

```
port( clk    : in std_logic;
rst    : in std_logic;
SF_D   : out std_logic_vector(11 downto 8);
LCD_E  : out std_logic;
LCD_RS : out std_logic;
LCD_RW : out std_logic;
SF_CE0 : out std_logic);
end lcd3 ;
architecture rtl of lcd3 is
type istate_t is (istep_one, istep_two, istep_three, istep_four, istep_five,
istep_six, istep_seven, istep_eight, istep_nine,
function_set, entry_mode, control_display, clear_display,
init_done);
type dstate_t is (didle, set_start_address, write_data_D, write_data_T,
write_data_1, write_data_3, --return_home,
address_digit, write_digit);
signal istate, next_istate   : istate_t;--state and next state of the init. sm
signal dstate, next_dstate   : dstate_t;--state and next state of the display sm
signal idone, next_idone     : std_logic;--initialization done
signal count, next_count     : integer range 0 to 750000;
signal nibble                : std_logic_vector(3 downto 0);
signal enable, next_enable   : std_logic;--register enable signal put out to
LCD_E
signal regsel, next_regsel   : std_logic;--register select signal put out to
LCD_RS
signal byte                  : std_logic_vector(7 downto 0); --data to pass to SF_D
signal timer_15ms            : std_logic;
signal timer_4100us          : std_logic;
signal timer_100us           : std_logic;
signal timer_40us            : std_logic;
signal timer_1640us          : std_logic;
signal txdone, next_txdone   : std_logic;
signal txcount, next_txcount : integer range 0 to 2068;
signal selnibble             : std_logic;
signal next_selnibble        : std_logic;
signal digit, next_digit     : std_logic_vector(3 downto 0);
signal cnt, next_cnt         : integer range 0 to 50000000;
```

```
begin
SF_CE0 <= '1'; --disable intel strataflash memory.
LCD_RW <= '0'; --write LCD (LCD accepts data).
SF_D   <= nibble;
LCD_E  <= enable;
LCD_RS <= regsel;
case istate is
when istep_two | istep_four | istep_six =>
byte <= X"30";
when istep_eight =>
byte <= X"20";
when function_set =>
byte <= X"28";
when entry_mode =>
byte <= X"06";
when control_display =>
byte <= X"0C";
when clear_display =>
byte <= X"01";
when others =>
byte <= (others => '0');
end case;
if istate = init_done then
case dstate is
when set_start_address =>
byte <= X"80"; -- first char of first line
when write_data_D =>
byte <= X"44";
when write_data_T =>
byte <= X"54";
when write_data_1 =>
byte <= X"31";
when write_data_3 =>
byte <= X"33";
when address_digit =>
byte <= X"CF"; -- last char of the second line
when write_digit =>
```

```
byte <= "0011" & digit;
when others =>
byte <= (others => '0');
end case;
end if;
end process data_selector;
digit_incr: process (dstate, txdone, digit, cnt)
begin
next_digit <= digit; -- hold the value
next_cnt   <= cnt;
if (cnt = 50000000) then
if (dstate = address_digit and txdone = '1') then
if digit = X"9" then
next_digit <= (others => '0');
else
next_digit <= digit + 1;
end if;
next_cnt <= 0;
end if;
else
next_cnt <= cnt + 1;
end if;
end process digit_incr;
nibble_select: process (selnibble, byte)
begin
case selnibble is
when '0' => -- pass lower nibble
nibble <= byte(3 downto 0);
when '1' => -- pass upper nibble
nibble <= byte(7 downto 4);
when others => -- nothing to do
end case;
end process nibble_select;
init_sm: process (istate, idone, timer_15ms, timer_4100us, timer_100us,
timer_40us, timer_1640us, txdone )
begin
next_istate     <= istate;
```

```
next_idone      <= idone;
case istate is
when istep_one => -- wait here for 15 ms
if (timer_15ms = '1') then
next_istate    <= istep_two;
end if;
when istep_two => -- write nibble (0x3)
if (txdone = '1') then
next_istate <= istep_three;
end if;
when istep_three => -- wait here for 4100 us
if (timer_4100us = '1') then
next_istate    <= istep_four;
end if;
when istep_four => -- write nibble (0x3)
if (txdone = '1') then
next_istate  <= istep_five;
end if;
when istep_five => -- wait here for 100 us
if (timer_100us = '1') then
next_istate   <= istep_six;
end if;
when istep_six => -- write nibble (0x3)
if (txdone = '1') then
next_istate <= istep_seven;
end if;
when istep_seven => -- wait here for 40 us
if (timer_40us = '1') then
next_istate   <= istep_eight;
end if;
when istep_eight => -- write nibble (0x2)
if (txdone = '1') then
next_istate  <= istep_nine;
end if;
when istep_nine => -- wait here for 40 us
if (timer_40us = '1') then
next_istate   <= function_set;
```

```
end if;
when function_set => -- istep 10:
if (txdone = '1') then
next_istate <= entry_mode;
end if;
when entry_mode => -- istep 11
if (txdone = '1') then
next_istate <= control_display;
end if;
when control_display => -- istep 12
if (txdone = '1') then
next_istate <= clear_display;
end if;
when clear_display => -- istep 13
if (txdone = '1') then
next_istate <= init_done; -- init. done
end if;
when init_done => -- istep 14
if (timer_1640us = '1') then
next_idone     <= '1';
end if;
when others => -- nothing to do
end case;
end process init_sm;
time_m: process(istate, count, idone)
begin
next_count   <= count;
timer_15ms   <= '0'; -- combinational output
timer_4100us <= '0'; -- combinational output
timer_100us  <= '0'; -- combinational output
timer_40us   <= '0'; -- combinational output
timer_1640us <= '0'; -- combinational output
case istate is
when istep_one   =>
next_count <= count + 1;
if (count = 750000) then
next_count  <= 0;
```

```
timer_15ms  <= '1';
end if;
when istep_three =>
next_count <= count + 1;
if (count = 205000) then
next_count   <= 0;
timer_4100us <= '1';
end if;
when istep_five  =>
next_count <= count + 1;
if (count = 5000) then
next_count  <= 0;
timer_100us <= '1';
end if;
when istep_seven | istep_nine =>
next_count <= count + 1;
if (count = 2000) then
next_count  <= 0;
timer_40us  <= '1';
end if;
when init_done =>
if (idone = '0') then
next_count <= count + 1;
end if;
if (count = 82000) then
next_count   <= 0;
timer_1640us <= '1';
end if;
when others => -- nothing to do
end case;
end process time_m;
tx_m: process(istate, txcount, byte, selnibble, enable, txdone,
idone, dstate)
begin
next_selnibble <= selnibble;
next_txdone    <= txdone;
next_txcount   <= txcount;
```

```
next_enable    <= enable;
case istate is
when istep_one | istep_three | istep_seven | istep_nine =>
next_selnibble <= '1'; -- pass hign nibble
when istep_two | istep_four | istep_six | istep_eight =>
next_txcount <= txcount + 1;
if (txcount = 1) then
next_enable <= '1';
end if;
if (txcount = 10) then
next_enable <= '0';
next_txdone <= '1';
end if;
if (txcount = 11) then
next_txcount    <= 0;
next_txdone     <= '0';
when function_set | entry_mode | control_display |
clear_display | init_done =>
if (istate /= init_done or
(istate = init_done and
(dstate = set_start_address or
dstate = write_data_D or
dstate = write_data_T or
dstate = write_data_1 or
dstate = write_data_3 or
dstate = address_digit or
dstate = write_digit
))) then

next_txcount <= txcount + 1;
if (txcount = 1) then
next_enable <= '1';
end if;
if (txcount = 10) then
next_enable <= '0';
end if;
if (txcount = 11) then
```

```
-- next we could pass zeros on the SF_D bus
end if;
if (txcount = 58) then    -- 10 + 1 + 50 - 2 = 58
next_selnibble <= '0'; -- pass lower nibble
end if;
if (txcount = 60) then
next_enable <= '1';
end if;
if (txcount = 69) then
next_enable <= '0';
end if;
if(txcount = 70) then -- done with the lower nibble data
if (txcount = 2067) then
next_txdone <= '1';
end if;
if (txcount = 2068) then -- 69 + 1 + 2000 - 2 =
next_txcount   <= 0;
next_txdone    <= '0';
next_selnibble <= '1'; -- pass upper nibble
end if;
end if;
when others => --nothing to do
end case;
end process tx_m;
display_sm: process(dstate, txdone, idone, regsel, txcount)
begin
next_dstate <= dstate;
next_regsel <= regsel;
if txcount = 11 then
next_regsel <= '0';
end if;
if txcount = 58 then
next_regsel <= idone; --high for active write dstates, low for istates
end if;
if txcount = 70 then
next_regsel <= '0';
end if;
```

```
case dstate is
when didle =>
next_regsel <= '0'; -- must be low for active istates
if (idone = '1') then
next_dstate  <= set_start_address;
next_regsel  <= '0'; -- must be low for address commands
end if;
when set_start_address => -- start the text at the first
-- location of the first line
-- of the LCD (0x80)
next_regsel <= '0';
if (txdone = '1') then
next_dstate <= write_data_D;
next_regsel <= '1'; --must be high for write commands
end if;
when write_data_D => -- D = 0x44
if (txdone = '1') then
next_dstate <= write_data_T;
next_regsel <= '1';
end if;
when write_data_T => -- T = 0x54
if (txdone = '1') then
next_dstate <= write_data_1;
next_regsel <= '1';
end if;
when write_data_1 => -- 1 = 0x31
if (txdone = '1') then
next_dstate <= write_data_3;
next_regsel <= '1';
end if;
when write_data_3 => -- 3 = 0x33
if (txdone = '1') then
next_dstate <= address_digit;
next_regsel <= '0';
end if;
when address_digit => -- 0x80
next_regsel <= '0';
```

```
if (txdone = '1') then
next_dstate <= write_digit;
next_regsel <= '1';
end if;
when write_digit => -- the digit running from 0 to 9
-- 0x30, 0x31, 0x32, 0x33, 0x34,
-- 0x35, 0x36, 0x37, 0x38, 0x39
if (txdone = '1') then
next_dstate <= address_digit; --return_home;
next_regsel <= '0';
end if;
when others => -- nothing to do;
end case;
end process display_sm;
registers: process(rst, clk)
begin
if rst = '1' then
istate    <= istep_one;
dstate    <= didle;
idone     <= '0';
count     <=  0;
txcount   <=  0;
selnibble <= '1'; -- upper nibble
enable    <= '0';
txdone    <= '0';
regsel    <= '0';
digit     <= (others => '0');
cnt       <= 0;
elsif clk = '1' and clk'event then
istate    <= next_istate;
dstate    <= next_dstate;
idone     <= next_idone;
count     <= next_count;
txcount   <= next_txcount;
selnibble <= next_selnibble;
enable    <= next_enable;
txdone    <= next_txdone;
```

```
regsel    <= next_regsel;
digit     <= next_digit;
cnt       <= next_cnt;
end if;
end process registers;
end rtl;
```

b. VGA kết nối với Spartan_ 3E

Chương trình :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity vga is
Port (mclk : in  STD_LOGIC;
red : out  STD_LOGIC_VECTOR(2 downto 0);
grn : out  STD_LOGIC_VECTOR(2 downto 0);
blu : out  STD_LOGIC_VECTOR(1 downto 0);
hs : out  STD_LOGIC;
vs : out  STD_LOGIC);
end vga;
architecture Behavioral of vga is
signal clk: STD_LOGIC;
signal horz_scan: STD_LOGIC_VECTOR (9 downto 0);
signal vert_scan: STD_LOGIC_VECTOR (9 downto 0);
signal vinc_flag: STD_LOGIC;
signal start_red: STD_LOGIC_VECTOR (5 downto 0);
signal delta_red: STD_LOGIC_VECTOR (2 downto 0);
signal delta_green: STD_LOGIC_VECTOR (2 downto 0);
signal green_y: STD_LOGIC_VECTOR (9 downto 0) := "0100000000";
```

```
signal green_dy: STD_LOGIC;
signal blue_x: STD_LOGIC_VECTOR (9 downto 0) := "0100000000";
signal blue_y: STD_LOGIC_VECTOR (9 downto 0) := "0100000000";
signal blue_dx: STD_LOGIC;
signal blue_dy: STD_LOGIC;
begin
-- Clock divide by 1/2
process(mclk)
begin
if mclk = '1' and mclk'Event then
clk <= not clk;
end if;
end process;
-- horizonal clock
process(clk)
begin
if clk = '1' and clk'Event then
if horz_scan = "1100100000" then
horz_scan <= "0000000000";
else
horz_scan <= horz_scan + 1;
end if;
if horz_scan(3 downto 0) = "0000" then
if horz_scan(9 downto 0) < 70 then
delta_red <= start_red(3 downto 1);
else
delta_red <= delta_red + 1;
end if;
```

```
end if;
end if;
end process;
-- vertial clock (increments when the horizontal clock is on the front porch
process(vinc_flag)
begin
if vinc_flag = '1' and vinc_flag'Event then
if vert_scan = "1000001001" then
vert_scan <= "0000000000";
delta_green <= "000";
start_red <= start_red + 1;
if green_dy = '1' then
green_y <= green_y + 1;
if green_y = 320 then
green_dy <= '0';
end if;
else
if green_y = 42 then
green_dy <= '1';
end if;
green_y <= green_y - 1;
end if;
if blue_dx = '1' then
blue_x <= blue_x + 1;
if blue_x >= 700 then
blue_dx <= '0';
end if;
else
```

```
blue_x <= blue_x - 1;
if blue_x <= 145 then
blue_dx <= '1';
end if;
end if;
if blue_dy = '1' then
blue_y <= blue_y + 1;
if blue_y = 470 then
blue_dy <= '0';
end if;
else
if blue_y = 42 then
blue_dy <= '1';
end if;
blue_y <= blue_y - 1;
end if;
else
vert_scan <= vert_scan + 1;
delta_green <= delta_green + 1;
end if;
end if;
end process;
-- horizontal sync for 96 horizontal clocks (96 pixels)
hs <= '1' when horz_scan < 96 else '0';
-- vertial sync for 2 scan lines
vs <= '1' when vert_scan(9 downto 1) = "000000000" else '0';
red <= delta_red when vert_scan > 42 and
vert_scan < 520 and
```

```
horz_scan >= 144 and
horz_scan < 784
else "000";
grn <= delta_green when vert_scan >= green_y and
vert_scan < green_y+200 and
horz_scan >= 144 and
horz_scan < 784
else "000";
blu <= vert_scan(1 downto 0) when vert_scan >= blue_y and
vert_scan < blue_y+50 and
horz_scan >= blue_x and
horz_scan < blue_x+50
else "00";
vinc_flag <= '1' when horz_scan = "1100011000" else '0';
end Behavioral;
```

c.Mouse kết nối với Spartan -3E

Chương trình :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity key is
port( data: in std_logic;
pclk: in std_logic;
l1 : out std_logic;
l2 : out std_logic;
l3 : out std_logic;
l4 : out std_logic;
```

```
l5 : out std_logic;
l6 : out std_logic;
l7 : out std_logic;
l8 : out std_logic);
end key;
architecture Behavioral of key is type state is
(state1,state2,state3,state4,state5,state6,state7,state8,state9,state10,state11);
signal ps,ns : state;
signal store : std_logic_vector(7 downto 0):="00000000";
signal start,parity,stop : std_logic;
begin
process(pclk,data)
begin
if pclk'event and pclk = '1' then   ps <= ns;
end if;
if pclk'event and pclk = '0' then if ps = state1 then stop <= data;
ns <= state2;
elsif ps = state2 then store(0) <= data;
ns <= state3;
elsif ps = state3 then store(1) <= data;
ns <= state4;
elsif ps = state4 then store(2) <= data;
ns <= state5;
elsif ps = state5 then store(3) <= data;
ns <= state6;
elsif ps = state6 then store(4) <= data;
ns <= state7;
elsif ps = state7 then store(5) <= data;
```

```
ns <= state8;
elsif ps = state8 then store(6) <= data;
ns <= state9;
elsif ps = state9 then store(7) <= data;
ns <= state10;
elsif ps = state10 then parity <= data;
ns <= state11;
elsif ps = state11 then stop <= data;
ns <= state1;
end if;
end if;
end process;
process(store) begin l1 <= store(0);
l2 <= store(1);
l3 <= store(2);
l4 <= store(3);
l5 <= store(4);
l6 <= store(5);
l7 <= store(6);
l8 <= store(7);
end process;
end Behavioral;
```

KẾT LUẬN:

Sau một quá trình nghiên cứu học hỏi , được sự giúp đỡ tận tình của thầy cô trong khoa Điện Tử -Viễn Thông nói chung , thầy Đoàn Hữu Chức nói riêng trong việc thực hiển đồ án của em và sau đây là kết quả em đã đạt được trong quá trình làm đồ án :

- ✓ Hiểu rõ về tổng quan FPGA và ngôn ngữ VHDL,Kit Spartan _3E, sử dụng thành thạo phần mềm ISE 8.2.
- ✓ Thiết kế một sốmạch logic trên kit Spartan_3E:Thiết kế mạch giải mã 2 đường sang 4 đường với ngõ ra tích cực cao; thiết kế mạch mã hóa 4 đường sang 2 đường với ngõ vào tích cực cao; thiết kế mạch giải mã đa hợp 1 ngõ vào 4 ngõ ra 2 lựa chọn; thiết kế mạch giải mã led 7 đoạn loại anode chung; thiết kế mạch so sánh 2 số 1 bit; thiết kế Flip Flop D.
- ✓ Một số ứng dụng kết nối của FPGA trên kit Spartan_3E: LCD kết nối với Spartan_3E; VGA kết nối với Spartan_3E, kết nối với mouse.

Mặc dù em đã nỗ lực và cố gắng để hoàn thiện đồ án một cách tốt nhất, nhưng em vẫn không thể tránh khỏi những sai sót trong việc xây dựng hệ thống và trình bày đồ án,… Em rất mong được các thầy cô hết sức thông cảm cho những sai sót đó của em.

Cuối cũng, Em xin chân thành cảm ơn các thầy cô trong trường Đại học Dân lập Hải Phòng đã dạy bảo em trong suốt quá trình học tập tại trường, đặc biệt là thầy Đoàn Hữu Chức và các thành viên trong tập thể lớp ĐT1301 đã giúp em hoàn thành tốt đồ án này.

Em xin chân thành cảm ơn !

Hải Phòng, ngày 29 tháng 6 năm 2013

Tác giả:

Hoàng Văn Thơi

TÀI LIỆU THAM KHẢO

1. Trịnh Quang Kiên, Lê Xuân Bằng (HĐ: PGS TS Đỗ Xuân Tiến) Thiết kế logic số - HVKTQS 2011

2. IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE StandardNo. 754. American National Standards Institute – Washington, DC - 1985.

3.Douglas L.Perry, VHDL Programming by Example McGraw-Hill,Fourth Edition

4.Volnei A.Pedroni, Circuit Design With VHDL,MIT Press,2004

5.Jan Van Der Spiegel,VHDL tutorial

7.Tống Văn On, Thiết kế mạch số sử VHDL và Verilog, Nhà xuất bản lao động xã hội, 2007.

8. Nguyễn Thúy Vân - Thiết kế logic mạch số - NXB Khoa học kỹ thuật – Năm2005

9. Nguyễn Linh Giang - Thiết kế mạch bằng máy tính –NXB Khoa học kỹ thuật Năm 2005

10. www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

11. http://www.doko.vn

12.http://www.fpga4fun.com/

13. http://www.dientuvietnam.net/

14.https://code.google.com/p/plasmacpu/source/browse/trunk/vhdl/spartan3e.ucf?r=17

15. http://luanvan.net.vn/luan-van/tong-quan-ve-fpga-6373/

16. http://www.kilobooks.com