SỞ LAO ĐỘNG - THƯƠNG BINH VÀ XÃ HỘI HÀ NỘI

TRƯỜNG TRUNG CẤP CÔNG NGHỆ VÀ DU LỊCH HÀ NỘI

GIÁO TRÌNH

MÔN ĐUN: VI MẠCH SỐ LẬP TRÌNH

NGHỀ: CÔNG NGHỆ KỸ THUẬT ĐIỆN – ĐIỆN TỬ

TRÌNH ĐỘ: TRUNG CẤP

(Ban hành kèm theo Quyết định số: 32/QĐ-CNDL ngày 28 tháng 02 năm 2023 của Hiệu trưởng Trường Trung cấp Công nghệ và Du lịch Hà Nội)

Hà Nội, năm 2023

LỜI NÓI ĐẦU

Cùng với các mô đun của ngành Công nghệ kỹ thuật điện - điện tử, mô đun Vi mạch số lập trình là mô đun kỹ thuật chuyên ngành quan trọng của ngành điện tử, hiện nay mô đun được ứng dụng rộng rãi trong ngành kỹ thuật và các lĩnh vực điều khiển khác.

Mô đun được ứng dụng cho tất cả sinh viên ngành Công nghệ kỹ thuật điện - điện tử. Bởi vậy để tạo điều kiện cho việc học tập và nghiên cứu mô đun của học viên được thuận lợi trong quá trình học tập. Bộ môn Điện tử thuộc Khoa Kỹ thuậ điện - Công nghệ Trường trung cấp Công nghệ và Du lịch Hà Nội tổ chức biên soạn tài liệu: " ***Vi mạch số lập trình*** " làm bài giảng lưu hành hội bộ. Trong quá trình biên soạn chắc chắn sẽ không tránh khỏi những thiếu sót, bởi vậy tôi mong nhận được sự thông cảm và góp ý chân thành của các bạn đồng nghiệp để cho giáo trình ngày càng hoàn thiện hơn.

Xin chân thành cảm ơn!

MỤC LỤC

BÀI 1: GIỚI THIỆU CHUNG VỀ PLDs

1. LỊCH SỬ PHÁT TRIỂN

Trước thời kỳ vi mạch số lập trình (Programmable Logic Device) ra đời, thiết kế logic số truyền thống thường dùng nhiều vi mạch TTL loại MSI và SSI kết hợp lại để tạo ra các hàm logic mong muốn. Những nhà thiết kế dựa vào những sách tra cứu các vi mạch số để tìm hiểu chức năng và các thông số kỹ thuật, sau đó mới quyết định sử dụng các vi mạch số cần thiết cho yêu cầu thiết kế của họ. Điều bất lợi của việc thiết kế này là sử dụng nhiều vi mạch dẫn đến nhiều khuyết điểm như: Kích thước board mạch lớn,

công suất tiêu thụ cao, dễ hư hỏng, thi công khó khăn, tốn kém…Nói chung là không kinh tế nhất là với những yêu cầu điều khiển phức tạp.

Vào năm 1975,công ty SIGNETICS đã giới thiệu vi mạch số lập trình không có bộ nhớ đầu tiên 82S100 (hiện nay là PLS100) gọi là mảng logic lập trình trường (Field- Programmable Logic Array). Napoleon Cavlan, người được gọi là cha đẻ của mạch logic lập trình, lúc bấy giờ là nhà quản lý những ứng dụng PLA của Signetics đã thực sự hiểu rằng sử dụng PLA là phương pháp tốt hơn để thiết kế và thay đổi hệ thống số. Trong khi đó, công ty Harris đã sớm giới thiệu PROM, họ trình bày triển vọng của PROM và đã ứng dụng vào trong một số mạch logic.

Công ty National Semiconductor đã chế tạo mặt nạ lập trình cho PLA, cấu tạo của nó gồm một mảng AND lập trình kèm với mảng OR lập trình, cho phép thực hiện tổ hợp tổng các tích số của hàm logic tiêu chuẩn. Bằng cách kết hợp công nghệ PROM sử dụng nguyên tắc cầu chì với khái niệm PLA, Cavian đã thuyết phục được các nhà quản lý công ty Signetics để đưa dự án PLAvào sản xuất.

Vi mạch PLA đầu tiên 82S100, là thành viên đầu tiên của họ vi mạch IFL (Intergrated Fuse Logic) có hình dạng 28 chân. Cấu trúc của PLA gồm một mảng AND lập trình và một mảng OR lập trình, nó cho phép thực hiện tổ hợp logic tổng của các tích số đơn giản .

Kỹ sư John Martin Birkner là một người quan tâm đến PLA, vì ông ấy hiểu rằng nhiều phương pháp thiết kế logic được học trong trường thì không áp dụng được nhiều trong công việc hiện tại. Do đó, vào năm 1975 ông ấy đã rời thung lũng Silicon để đến công ty Monolithic Memories (MMI), đây là công ty chế tạo PROM và các vi mạch logic tiêu chuẩn. Vì vậy, Birkner có điều kiện hơn trong việc tìm hiểu PLA và công nhận những ưu điểm của mạch logic lập trình nhưng đồng thời ông cũng nhận ra khuyết điểm của PLA là có hai mảng lập trình. Sau đó, Birkner đã đưa ra khái niệm mới về vi mạch số lập trình, vi mạch này cũng tương tự FLA nhưng thay vì có hai mảng lập trình thì PAL (Programmable Array Logic) chỉ có một mảng AND lập trình và theo sau là mảng OR được giữ cố định (không lập trình). Như vậy mỗi cổng OR sẽ có một tích số cố định được nối với ngõ vào của nó, do vậy sẽ giảm được kích thước của vi mạch và cho phép tín hiệu được truyền nhanh hơn trong khi vẫn cho phép thực hiện các tổ hợp logic. PAL được đóng vỏ

20 chân. Sau một thời gian thuyết phục các nhà quản lý của công ty MMI thấy rõ những lợi điểm của PAL và đồng ý sản xuất. Vi mạch đầu tiên thuộc họ PAL được phổ biến là PAL 16L8, PAL 16R4, PAL 16R6, PAL 16R8. Các vi mạch này có thời gian truyền trì hoãn 35ns. Mỗi vi mạch có 8 ngõ ra và 16 ngõ vào, trong đó ký tự L trong ký hiệu của vi mạch biểu thị 8 tổ hợp ngõ ra tác động ở mức thấp, ký tự R cho biết có 4, 6 hay 8 thanh ghi ở ngõ ra tương ứng.

Sau một thời gian khởi đầu chậm, cuối cùng PAL đã được thiết kế trong hệ thống thực. Những công ty máy tính mini đã nhận thấy được ưu điểm của PAL là cho phép họ giảm số board cần thiết để thực hiện tốt những yêu cầu thiết kế, công ty MMI đã chọn phương pháp sản xuất PAL công đoạn mặt nạ chế tạo theo yêu cầu khách hàng. Vào lúc này MMI lại giới thiệu một họ vi mạch mới HAL (Hard Array Logic) và để sản xuất những chi tiết này cho hãng Data General and Digital Equipment. MMI đã thay đổi cách sắp xếp công đoạn mặt nạ cầu chì và thay vào đó là lớp liên kết kim loại phù hợp yêu cầu thiết kế của khách hàng. Những chi tiết này có nhiều lợi ích gồm mang lại những kết quả tốt và kiểm tra dễ dàng hơn. Đồng thời khách hàng cũng được lợi hơn bởi không phải quan tâm đến lập trình và kiểm tra các chi tiết. Điều này đã mang lại sự cải tiến về phương pháp chế tạo PAL, và được sự chấp nhận của thị trường. Vào năm 1978, MMI đã xuất bản sách hướng dẫn PAL đầu tiên. Đó là một bước khởi đầu để PAL mở rộng thế giới của những người thiết kế mạch logic. Ngoài ra trong sách hướng dẫn còn trình bày danh sách chương trình gốc của ngôn ngữ lập trình FORTRAN cho PALASM (PAL Assembler) đó là phần mềm dành cho việc thiết kế mạch logic PAL. PALASM có thể biên soạn, định nghĩa logic cho một khuôn thức. Ngoài ra PALASM cũng có khả năng mô phỏng sự vận hành trên phương trình mạch logic theo nguyên tắc PAL. Trong việc liên kết với những nhà thiết kế để định rõ những "vector kiểm tra", PALASM có thể là một sự thật phù hợp. Tất cả những đặc điểm của PAL bao gồm việc khắc phục những khuyết điểm của PLA kết hợp với việc thúc đẩy sử dụng PAL đã mang đến kết quả tốt đẹp. PAL đã nhanh chóng vượt qua họ vi mạch IFL của công ty Signetics và được phổ biến trên thị trường, thuật ngữ PAL đã trở nên đồng nghĩa với PLD

Trong lúc ấy, công ty Signetics tiếp tục phát triển họ IFL, và vào năm

1977 Signetics giới thiệu họ vi mạch FPGA (Field Programmable Gate Array) 82S103, vào năm 1979 là họ FPLS (Field Programmable Logic Sequencer). Họ FPGA có cấu tạo một mảng AND ở mức đơn với ngõ vào lập trình được và cực tính ngõ ra cũng vậy cho phép thực hiện các hàm logic cơ bản (AND, OR, NAND, NOR, INVERT), cấu trúc của họ FPLS có chức các FlipFlop để thực hiện các trạng thái của hàm tuần tự. Đồng thời Signetics cũng giới thiệu AMAZE (Automated Map and Zap Equations) là chương trình biên dịch để hỗ trợ cho những vi mạch của họ. Tương tự, những công ty chế tạo PLD khác đã lần lược giới thiệu những phần mềm hỗ trợ của họ.

Cả 2 công ty Signetics và MMI tiếp tục giới thiệu những PLD mới để đáp ứng tính đa dạng theo các yêu cầu thiết kế. Vào giữa năm 1980, mạch logic lập trình đã được thừa nhận cùng với sự phát triển tính đa dạng của IFL và PAL đã có nhiều giá trị cho những người thiết kế. Mặc dù sự khởi đầu thành công của PLD, tuy nhiên chỉ một số ít các nhà thiết kế quen với việc dùng PLD, một số trường đại học đã đưa vi mạch logic lập trình vào những khóa học thiết kế của họ.

Tuy thế, kĩ thuật logic lập trình tiếp tục cải tiến và những vi mạch phát triển ở giai đoạn thứ hai được giới thiệu vào năm 1983. Công ty Advance Micro Devices (AMD) đã giới thiệu PAL22V10 với những đặc điểm đặc biệt là sự linh động của những cổng PLD ở 10 ngõ vào. Mỗi cổng PLD có khả năng tổ hợp hoặc với thanh ghi ở ngõ ra hoặc một ngõ vào. Cổng đệm ngõ ra ba trạng thái được điều khiển bởi một tích số riêng cho phép vận hành hai chiều. Tất cả thanh ghi đều được reset tự động trong quá trình tắt hay mở và mỗi thanh ghi có khả năng "đặt trước", đó là đặc điểm đặc biệt cho việc kiểm tra sau này.

Với những vi mạch mới, được giới thiệu thường xuyên trên thị trường đã dẫn đến việc cần thiết phải có một phần mềm hỗ trợ trong quá trình sử dụng PLD để đạt hiệu quả cao. Bob Osann đã nhận thấy được sự cần thiết của một chương trình biên dịch PLD vạn năng dùng cho tất cả PLD của những công ty chế tạo khác nhau.

Vào tháng 9/1983, Công ty Assisted Technology đã đưa ra phiên bản 1.01a của chương trình biên dịch PLD có tên là CUPL(Universal Compiler for Programmable). Chương trình này hỗ trợ cho 29 loại vi mạch, sự ra đời của CUPL đã gây được sự chú ý của nhiều công ty chế tạo. Công ty Data

I/O, nhà chế tạo các vi mạch lập trình lớn nhất trên thế giới (EPROM, PROM, PLD), đã quyết định phát triển phần mềm hỗ trợ cho riêng họ. Năm 1984, Data I/O giới thiệu ABEL (Advanced Boolean Expression Language), đó là chương trình biên dịch PLD có đặc điểm tương tự như CUPL nhưng nó được đầu tư tiếp thị nên được các nhà thiết kế chấp nhận. Vì vậy, ABEL đã sớm theo kịp CUPL trên thị trường.

Sự ra đời của chương trình biên dịch vạn năng cho PLD đã thúc đẩy nền công nghiệp thiết kế số sẵn sàng cho việc áp dụng PLD cho những thiết kế mới. Những chương trình biên dịch vạn năng này đã được cải tiến hơn so với các chương trình biên dịch PALASM và AMAZE, nó được cung cấp cho các nhà thiết kế để thực hiện các mạch logic và mô phỏng những thiết bị. Đó là những đặc điểm tiêu chuẩn của hai bộ biên dịch vạn năng CUPL và ABAL. JEDEC (the Joint Electron Device Engineering Council) dự định sản xuất một bộ biên dịch PLD tạo ra một tiêu chuẩn để sử dụng cho tất cả các công ty chế tạo PLD hiện nay và tương lai. Vào 10/1983, the JEDEC Solid State Products Engineering Council đưa ra tiêu chuẩn JEDEC thứ 3".

Tiêu chuẩn khuôn thức chuyển đổi giữa hệ thống tạo dữ liệu và thiết bị lập trình cho PLD". Tháng 5/1986, JEDEC tiếp tục đưa ra tiêu chuẩn 3-A, tiêu chuẩn này trở thành tiêu chuẩn chung cho công nghiệp PLD.

Tháng 7/1984, công ty Altera giới thiệu EP300. Đó là vi mạch sử dụng công nghệ CMOS của EPROM, nó có đặc tính là công suất tiêu thụ thấp, có thể xóa được (dùng tia cực tím) cùng một số đặc tính mở rộng khác. Năm 1985, một họ PLD mới được công ty Lattice Semiconductor giới thiệu là GAL (Generic Array Logic). Lattice dùng công nghệ CMOS của EEPROM, có các đặc tính kỹ thuật như công suất thấp, có thể lập trình nhiều lần (xóa bằng điện áp với thời gian xóa khoảng vài giây). Vi mạch đầu tiên của họ GAL được kí hiệu là GAL16V8 có khả năng thay thế hoạt động của PAL (đối với vi mạch cùng loại).

Ngày càng nhiều công ty tham gia vào thị trường PLD để tạo ra những vi mạch đặc biệt và sử dụng nhiều công nghệ chế tạo khác nhau. Vào năm 1985, công ty Xilen tạo ra một họ mới là LCA (Logic Call Array). Cấu trúc của LCA có 3 đoạn: một ma trận của khối logic được bao quanh là khối vào ra và một mạng đường dữ liệu nối gián tiếp. Đặc biệt của LCA là PLD đầu tiên sử dụng tế bào RAM động cho chức năng logic. Ưu điểm của cấu trúc

này là khách hàng có thể kiểm tra được chương trình của vi mạch, do bản chất dễ xóa của LCA, nên cần phải lưu trữ cấu hình của LCA ở bộ nhớ ngoài. Vì vậy, LCA không được sử dụng ở những trường hợp đòi hỏi sự hoạt động ngay lập tức khi khởi động máy. Đi kèm với LCA là chương trình soạn thảo XACT và bộ mô phỏng giúp cho việc sửa lỗi cho những thiết kế trên LCA được thuận tiện.

Năm 1985, công ty Signetics với một khái niệm mới là PML (Programmable Macro Logic). Vi mạch PML đầu tiên của Signetics PMLS 501, vi mạch này sử dụng công nghệ lưỡng cực, và được đóng vỏ 52 chân .

Vào năm 1986, công ty ExMicroelectronic giới thiệu họ ERASIC (Erasable Application Specific 7C) sử dụng công nghệ EEPROM CMOS. Vi mạch đầu tiên là XL78C00 có dạng 24 chân và điều đặc biệt là XL78C00 có thể thay thế chức năng cho PAL và EPLA cùng loại (không tính đến tốc độ), đi kèm là một phần mềm hỗ trợ ERASIC.

Vào năm 1986, công ty Signetics quyết định thay đổi họ IFL thành họ PLS (Programmable Logic From Signetics). Ví dụ như từ 82S100 thành PLS100, từ 82S157 thành PLS157. Sau đó 2 năm, công ty Actel đã cải tiến khuyết điểm họ LCA là vi mạch có thể hoạt động không nhất thiết phải có bộ nhớ ngoài. Đồng thời công ty Gazelle Microcircuit đã công bố phát minh công nghệ GaAs (Gallium Arsenide). Đặc điểm của công nghệ này là cải tiến tốc độ , công suất của các vi mạch trên nền tảng là công nghệ silicon, cho phép vi mạch làm việc với tốc độ nhanh hơn công suất tiêu tán khi ở mức trung bình.

Ứng dụng đầu tiên của công nghệ GaAs được công ty Gazelle đưa ra là phiên bản của PAL 22V10. Ưu điểm của mạch này là cho phép vi mạch GaAs có thể tương hợp với các vi mạch TTL, do đó công nghệ GaAS đã được ứng dụng rộng rãi. Sau một thời gian cải tiến không ngừng, những PLD thế hệ sau đã được ứng dụng rộng rãi trong kỹ thuật phần cứng, nó trở thành công cụ cần thiết cho những kỹ sư thiết kế.

Sự phát triển trong công nghiệp PLD nói riêng và với công nghiệp bán dẫn nói chung đã tạo nên sự cạnh tranh của các công ty chế tạo PLD trên thế giới. Do đó, đã có nhiều xung đột xảy ra giữa các công ty trong việc cạnh tranh thị trường.

Vào năm 1986 công ty MMI đã kiện hai công tyAltera và Lattic vì đã vi phạm bản quyền PAL. Kết quả là hai công ty này đã chấp nhận thua kiện

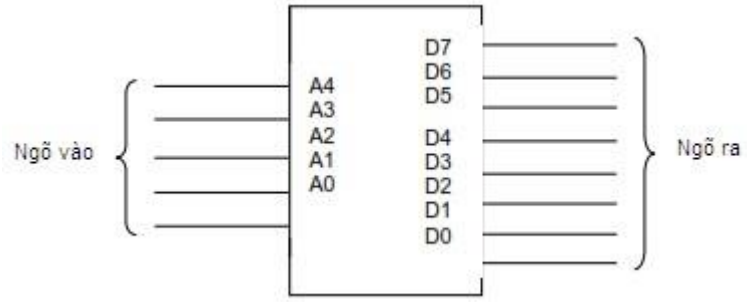
và phải mua bản quyền. Sau đó công ty MMI mua cổ phần trong công ty Xilin và sở hữu bản quyền họ LCA. Sau đó 1 năm công ty MMI hợp với AMD trở thành một tập đoàn sản xuất các linh kiện bán dẫn hàng đầu trên thế giới. Tuy đã hợp nhất hai công ty nhưng họ vẫn tiếp tục phát triển các họ vi mạch hiện có vì những họ PLD này đã trở nên phổ biến trên thị trường. Vào năm 1987, công ty National Semiconductor đã mua lại công ty Fairchild và tiếp tục phát triển họ PAL FASTPLA trên thị trường

2. CẤU TRÚC CƠ BẢN CỦA PLD

Vi mạch số lập trình trải qua thời gian dài phát triển và cải tiến đã thực sự mở ra một hướng đi mới cho những nhà thiết kế. Ưu điểm của PLD là giải quyết được vô số những vấn đề thiết kế nhờ vào nhiều họ PLD khác nhau. Những họ vi mạch này có cấu trúc và công nghệ chế tạo khác nhau, do đó chúng có những đặc điểm riêng để ứng dụng vào nhiều lĩnh vực trong công nghiệp. Mặc khác người thiết kế còn quan tâm đến các thông số kỹ thuật của vi mạch như tốc độ, công suất tiêu thụ, nguồn cung cấp và công cụ hỗ trợ để lập trình.

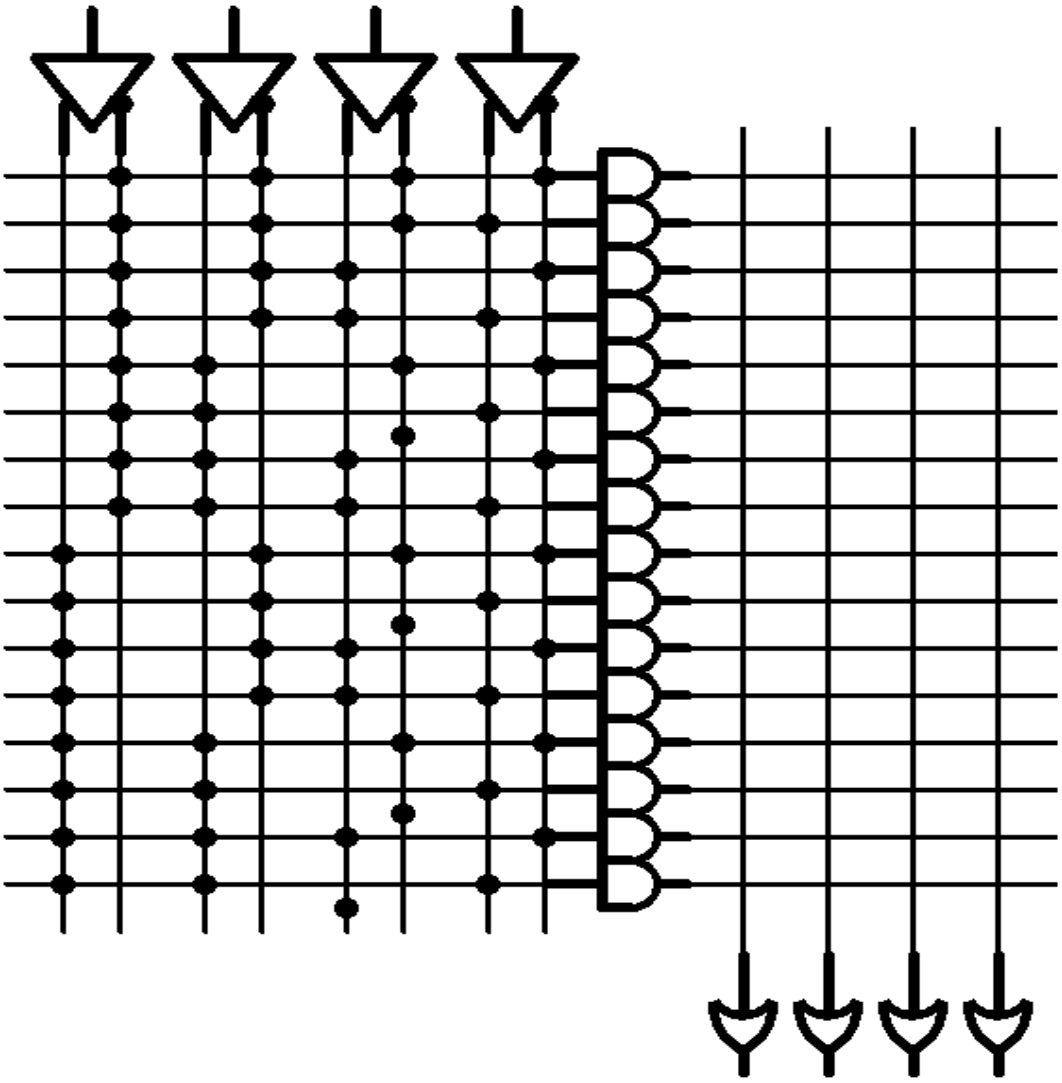
2.1 Họ vi mạch PROM

PROM gọi là bộ nhớ chỉ đọc lập trình được. Đây là họ vi mạch đầu tiên được sử dụng như là những vi mạch số lập trình theo quan điểm của vi mạch số. Cấu trúc của PROM rất đơn giản bao gồm một mảng tế bào nhớ với những đường địa chỉ ngõ vào và những đường dữ liệu ngõ ra. Số đường địa chỉ và dữ liệu cho biết ma trận nhớ của PROM. Một PROM đơn giản được trình bày ở hình 1.1



Hình 1.1. Cấu tạo PROM đơn giản

PROM có 5 đường điều khiển ngõ vào cho phép tạo ra 32 tổ hợp logic và 8 đường dữ liệu ra tạo thành một ma trận nhớ 32x8, vì vậy có tổng cộng 256 tế bào nhớ. Cấu trúc của PROM gồm một mảng AND cố định theo sau là mảng OR lập trình, được minh họa ở hình 1.2



Hình 1.2. Sơ đồ logic của PROM

Ghi chú:

- Dấu x là những điểm nối lập trình được (kết nối bằng một cầu chì)
- Dấu z là những điểm nối cố định

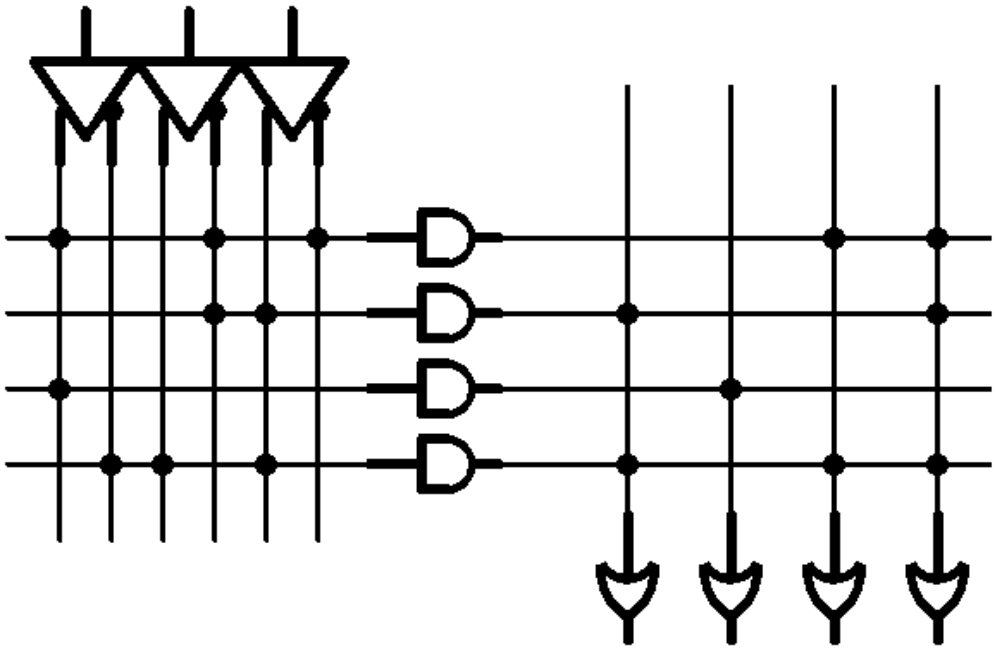
Ở mảng AND cố định có 16 biến được chọn và liên kết với 4 tín hiệu ngõ vào mảng OR. Do đó bất kì một liên kết nào bị loại bỏ (nghĩa là cầu chì ở đó bị đứt, thì biến đó sẽ không có mặt ở biểu thức ngõ ra). Các hàm ở ngõ ra thay đổi tùy thuộc vào sự kết nối của các biến ở ngõ vào. PROM thường được sử dụng để giải mã địa chỉ và ứng dụng để lưu trữ dữ liệu. Khi thiết kế các PROM, người thiết kế phải chú ý đến sự thay đổi mức logic ngõ vào (xảy ra trong thời gian ngắn) khi địa chỉ ngõ vào thay đổi. Phương thức ghi của PROM là khi có một tín xung clock đồng bộ thì mạch ngõ ra chuyển sang trạng thái khác. Đặc điểm này sẽ giúp khắc phục được vấn đề tạp nhiễu ở PROM.

Khi khảo sát PROM, người ta thường quan tâm đến tốc độ truy xuất dữ liệu. Thông thường các loại PROM có thời gian truy xuất dưới 60 ns. Các loại PROM thường sử dụng công nghệ lưỡng cực là nguyên tắc cơ bản để chế tạo. Tuy nhiên, khoa học tiến bộ đã phát minh ra công nghệ CMOS cho phép rút ngắn thời gian truy xuất. Công nghệ CMOS được dùng để chế tạo EPROM, đó là một dạng PROM có thể xóa được bằng tia cực tím. Nó đã tạo ra một bước tiến đáng kể như: EPROM WS57C256F của công ty WaferScale Integration có dung lượng 32Kx8 với thời gian truy xuất là 55 ns, công ty Cypress Semicondutor giới thiệu PROM CY7C245 có dung lượng là 2048x8 với thời gian truy xuất là 25 ns.

Trên đây là một vài ví dụ cho thấy công nghệ CMOS được chấp nhận cho những ứng dụng thiết kế mạch.

2.2 Họ vi mạch FPLA (Field Progammable Logic Array)

Họ vi mạch FPLA đầu tiên được công ty Signetics giới thiệu vào năm 1975. Cấu trúc của FPLA là một mảng AND – OR đơn giản, được trình bày như sau.



Hình 1.3. Sơ đồ biểu thức ngõ ra của FPLA

Từ kết quả trên cho thấy ngõ ra của cổng AND luôn ở mức thấp, điều này không có lợi. Tuy nhiên nếu ta lập trình cho 4 cầu chì trên, ví dụ ta chọn A x B, lúc này giá trị của 2 biến này sẽ không có trong biểu thức.

Biểu thức ngõ ra của cổng lúc này là; K = A.B

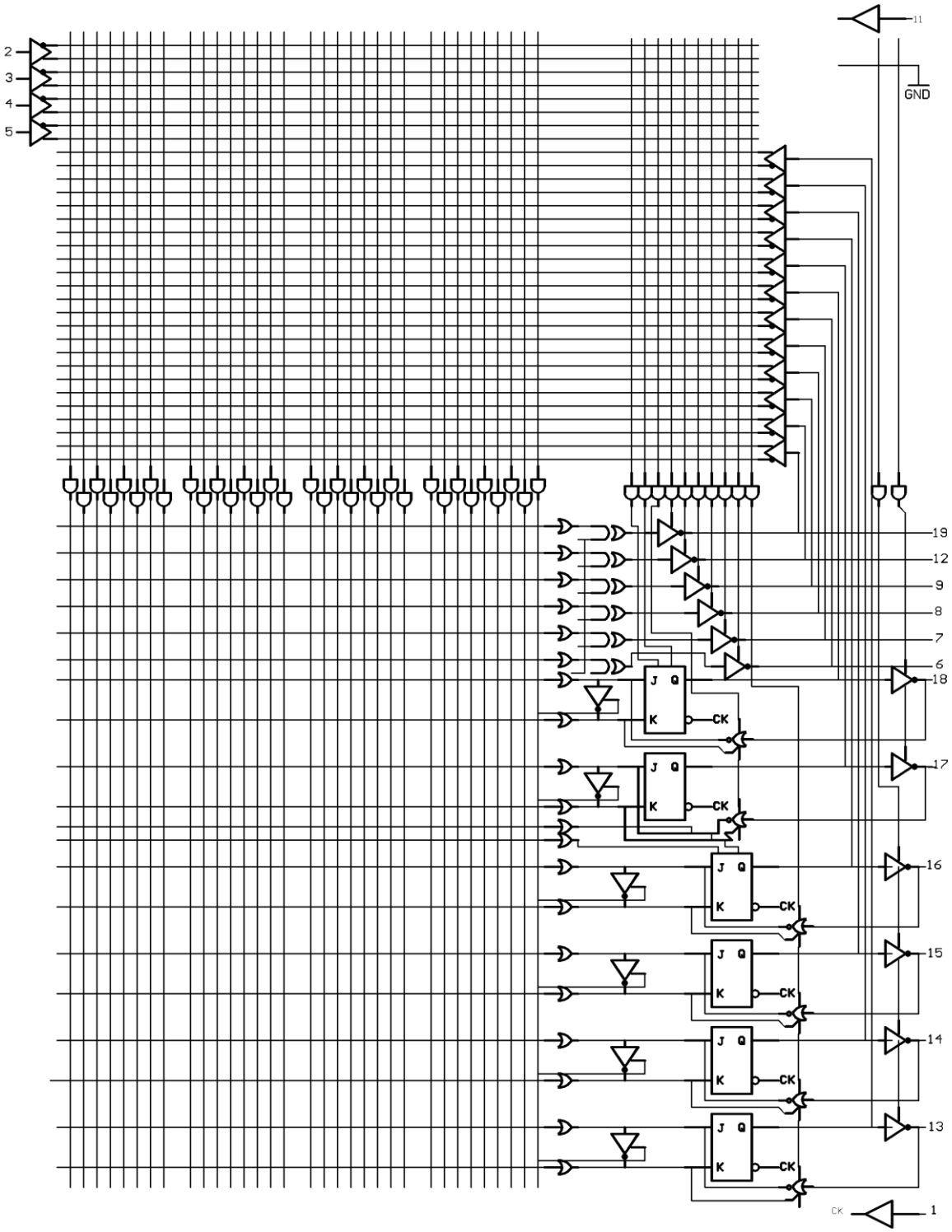
Nguyên tắc ở đây là lựa chọn những giá trị để lập trình, khi một cầu chì được chọn nghĩa là giá trị của nó sẽ không có mặt trong biểu thức.

Lưu ý mảng OR trong mạch ở hình 1.3. Mỗi ngõ ra cổng AND được nối tới 1 ngõ vào cổng OR thông qua một cầu chì và một Diode. Xét biểu thức F1 giả sử các cầu chì đều thông, ta có : F1= K + L+ M + N

Với K, L, M, N là những tích số của AXB, F1 là tổng các tích số của hai biến A và B. Bây giờ ta sẽ lập trình bằng cách làm đứt các cầu chì thì các số hạng ứng với những cầu chì bị đứt sẽ không có mặt trong biểu thức. Bằng cách lập trình các cầu chì ở mảng AND – OR (nghĩa là loại bỏ giá trị giá trị của nó trong biểu thức) FPLA có thể tạo ra các hàm logic khác nhau theo mạch thiết kế chỉ với hai biến ngõ vào. Lưu ý những Diode trong mảng OR được dùng để bảo vệ ngắn mạch.

2.3 Họ vi mạch FPLS (Field Programable Logic Sequencer)

Họ FPLS được giới thiệu vào năm 1979, FPLS có cấu trúc mô phỏng theo cấu trúc của FPLA nhưng được bổ sung thêm những thanh ghi cho phép "preloading" trạng thái của thiết bị. Một vài thanh ghi ở ngõ ra được đưa hồi tiếp về mảng AND lập trình và một số khác có những thanh ghi ngầm (những thanh ghi được bổ sung trên chíp và không nối với chân của ngõ vào hay ngõ ra) bổ sung với thanh ghi ngõ ra, nó có thể hồi tiếp hoặc không hồi tiếp.



Hình 1.4. Sơ đồ logic FPLS PLS 157 **2.4**

Ho vi mạch FPGA (Field Progammable Gate Array).

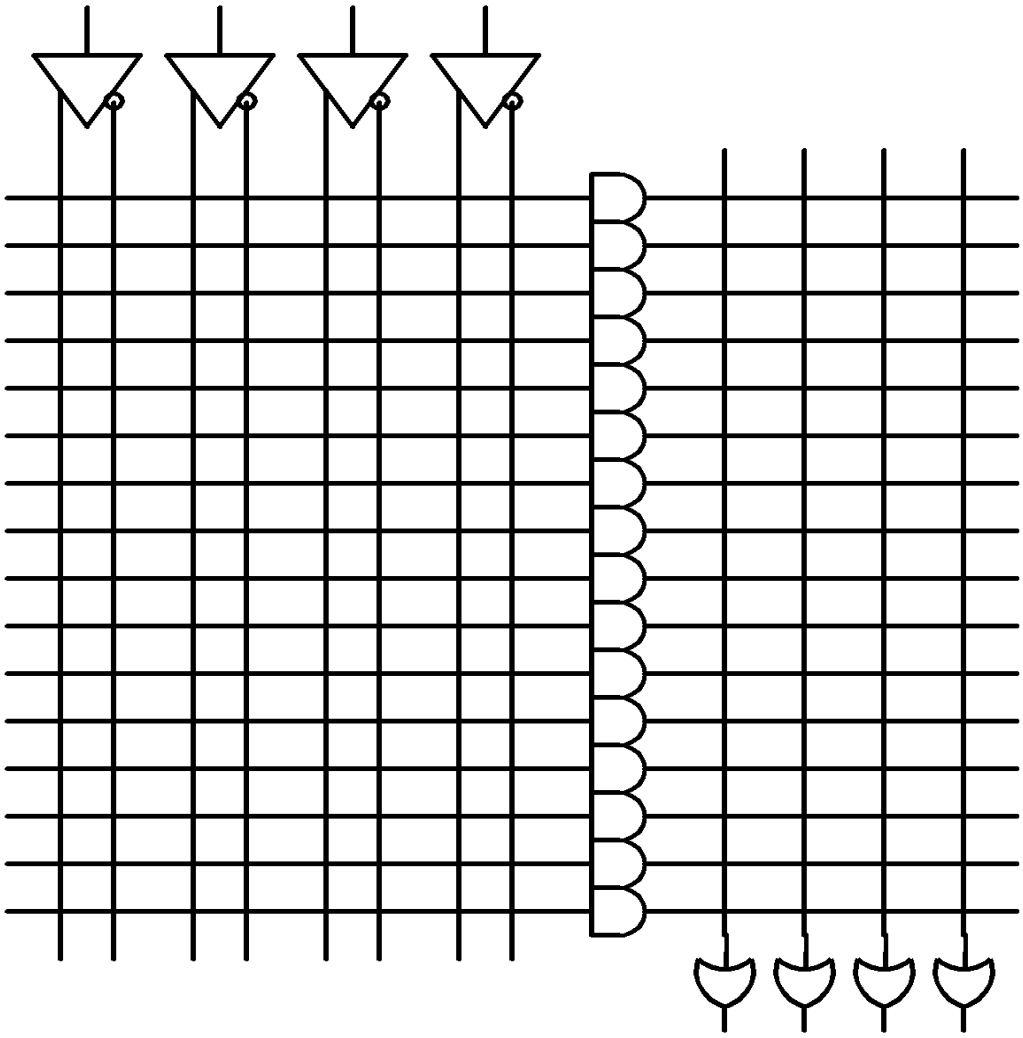
Họ FPGA được Signetics giới thiệu vào năm 1977 được sử dụng để thay thế cho những cổng nhiều ngõ vào tiêu chuẩn, cấu trúc của nó bao gồm một mảng AND lập trình, với lập trình cực tính ở ngõ ra. Chỉ với một cổng AND có thể biến đổi thành cổng NAND, NOR hay cổng OR. Mỗi cổng AND trong FPGA có thể biến đổi thành các cổng logic khác nhau.

FPGA cũng được bổ sung linh động hơn những cổng tiêu chuẩn khác. Vi mạch đại diện cho họ FPGA là PLS151, có hình dáng 20 chân. PLS151 có 6 ngõ vào, 12 ngõ ra và có tín hiệu hồi tiếp đưa về mảng AND được sử dụng như những ngõ vào. Có thêm 3 tích số được tạo ra bởi 3 đường điều khiển, các tín hiệu này điều khiển những cổng đệm ngõ ra 3 trạng thái. FPGA thích hợp trong các thiết kế để giải mã địa chỉ và được thêm vào các chức năng khác.

2.5 Ho vi mạch PAL (Programmable Array Logic).

PAL là một họ phổ biến nhất trong họ PLD được MONOLITHIC MEMORIES INC giới thiệu vào năm 1978. PAL được đăng ký bản quyền về cấu trúc của công ty MMI. Cấu trúc của PAL bao gồm một mảng AND lập trình theo sau là một mảng OR cố định, cấu trúc này được cải tiến từ những khuyết điểm của họ FPLA. Do loại bỏ việc sử dụng cầu chì ở mảng OR, do đó số lượng tinh thể Silicon được sử dụng giảm, dẫn đến giá thành của PAL thấp hơn so với FPLA. Mặt khác thời gian trì hoãn của PAL ngắn hơn so với FPLA do giảm được sự trì hoãn khi truyền qua mảng OR.

Khảo sát PAL16L8 có hình dáng 20 chân sơ đồ logic được trình bày ở hình 1.5. Vi mạch này có 8 tổ hợp ngõ ra, mỗi ngõ ra được đảo với 7 tích số của ngõ vào, 6 trong 8 ngõ ra được hồi tiếp về mảng AND, cho phép những chân này được sử dụng với chức năng I/O. Do PAL16L8 có ngõ ra tác động ở mức thấp nên nó có thể kết hợp với các IC khác cùng một mức tác động.



Hình 1.5. Sơ đồ logic PAL

PAL16L8 được ứng dụng trong lĩnh vực giải mã địa chỉ, nó thuận tiện trong việc kết hợp với các bộ vi xử lý và thiết bị ngoại vi vì cùng một mức tác động. Với những đặc tính như tốc độ tương đối cao, giá thành thấp, thời gian truyền trì hoãn khoảng 7,5ns nên PAL16L8 rất phổ biến trong công nghiệp PLD. Ngoài ra PAL16L8 có một đặc điểm mới so với các họ trước là có cầu chì bảo vệ, nó dùng để chống sự sao chép, giúp bảo vệ nội dung bên trong. Ngoài PAL16L8 công ty MMI còn giới thiệu các loại vi mạch khác như PAL16R4, PAL16R6, PAL16R8. Các vi mạch này có cấu tạo giống như PAL16L8 nhưng ở ngõ ra sử dụng thêm các FF D để chốt tín hiệu ngõ ra.

Một thế hệ vi mạch PAL được công ty AMD giới thiệu là PAL22V10 với hình dáng 24 chân được chế tạo bằng công nghệ CMOS thay thế cho công nghệ lưỡng cực. Đặc trưng của vi mạch này là ở ngõ ra được cho qua cổng PLD.

Ngoài việc tăng số biến ngõ vào vi mạch này còn có một số đặc điểm nữa là trong hàm logic các thành phần tích số có thể thay đổi từ 8 đến 16 biến. Điều này sẽ giúp cho vi mạch thực hiện nhiều phương trình phức tạp. Nhờ vào cấu tạo ở ngõ ra các cổng PLD nên các ngõ ra hoặc vào của vi mạch có đặc tính giao tiếp 2 chiều, điều này làm tăng khả năng xử lý của vi mạch và tạo sự thuận lợi cho việc thiết kế. Do những đặc điểm đã được cải tiến nên các thế hệ vi mạch PAL được phổ biến rộng rãi (đặc biệt là nhóm vi mạch 20 chân) và PAL được xem là họ vi mạch đại diện cho họ vi mạch số lập trình.

Ngoài ra các công ty chế tạo PAL có chọn lựa trong việc ký hiệu các số trên một vi mạch. Điều này cung cấp cho người sử dụng những thông tin cần thiết có liên quan đến ứng dụng của vi mạch. Các ký hiệu trong việc đánh số của họ PAL nói chung bao gồm 2 số đếm được tách rời nhau bởi 1 hay 2 ký tự. Số đầu tiên trong tên vi mạch cho biết số ngõ vào của vi mạch (đây chính là số biến ngõ vào của mảng AND). Số thứ hai biểu thị số ngõ ra của vi mạch. Ký tự nằm giữa 2 số chỉ ra ý nghĩa các thuộc tính của ngõ ra. Một số mã ký tự có ý nghĩa là:

H tác động mức thấp.

L tác động mức cao.

P tác động ngõ ra có thể lập trình.

C phần bổ sung các ngõ ra.

S bộ tuần tự.

Các ký hiệu của vi mạch họ PAL được xem là những hướng dẫn cơ bản của vi mạch. Ngoài ra các công ty chế tạo còn cung cấp bản thông số kỹ thuật và sơ đồ logic của vi mạch kèm theo để làm tài liệu tham khảo cho các nhà thiết kế.

2.6 Họ vi mạch GAL (Generic Array Logic).

GAL là một nhóm của công nghệ EEPLD, nó được giới thiệu và phát triển bởi công ty Lattice Semiconductor Comp. Công ty này đã đưa ra một khái niệm về cổng PLD có ký hiệu là OLMCs (Output Logic Macrocells).

Vi mạch này cũng có những đặc điểm là có thể xóa bằng điện và lập trình lại bằng các phần mềm và công cụ hỗ trợ. GAL16V8 có hình dạng 20 chân là một vi mạch phổ biến trong họ GAL.

Mỗi một OLMC có 8 ngõ vào tương đương với 8 tích số trong một biểu thức. Ngoài ra OLMC cũng có tín hiệu hồi tiếp đưa về để điều khiển, tín hiệu xung đồng hồ, tín hiệu hồi tiếp về mảng AND. Các vi mạch GAL đều có hỗ trợ những thanh ghi "Preload", điều này có ích trong việc kiểm tra vi mạch. Mặt khác một thế hệ vi mạch mới được phát triển là vi mạch lập trình hệ thống ký hiệu là ispEELD (In-system Progammable).

Vi mạch đầu tiên là ispGAL16Z8, cấu trúc của nó gần giống với GAL16V8 nhưng được thêm vào 4 chân để điều khiển lập trình. Trong hệ thống ispGAL16Z8 cho phép chu kỳ lập trình là 10000 lần và dữ liệu được giữ cố định trong khoảng thời gian 20 năm. Đó cũng là quy định của những vi mạch theo nguyên tắc EPROM. Cấu trúc của họ GAL là sự lặp lại cấu trúc của họ PAL và những đặc điểm của họ GAL được thiết kế để kết hợp với những vi mạch họ PAL. Điều này được thể hiện qua việc ký hiệu các vi mạch họ GAL và cấu trúc tế bào bảo vệ của nó.

2.7 Họ vi mạch PEEL (Progammable Electrially Erasable Logic).

Họ PEEL được công ty International Cmos Technology INC giới thiệu. Nó được chế tạo với công nghệ EEPROM. Cấu trúc của PEEL cũng tương tự như PAL và GAL, nó được xóa bằng điện và lập trình cũng nhờ vào phần mềm hỗ trợ.

Vi mạch có 20 chân với 8 ngõ ra được cấu tạo bởi cổng PLD, mỗi ngõ ra có 8 tích số trong một hàm của biểu thức và có một tích số riêng để điều khiển cổng đệm ngõ ra. Cực tính ngõ ra cũng được lập trình các thanh ghi ở ngõ ra của vi mạch được Reset không đồng bộ, ngoài ra các thanh ghi có thể được chốt bên trong khi ngõ ra được điều khiển bởi một biểu thức của tổng các số hạng của ngõ vào. Đặc điểm này được cải tiến hơn số với các vi mạch PAL16V10 hay GAL16V8.

Công ty Altera lần đầu tiên giới thiệu thuật ngữ xóa các PLD bằng tia cực tím và nó đã trở thành thuật ngữ chung cho công nghệ PLD để tham khảo cho các vi mạch lập trình xóa bằng tia cực tím. Từ khi khởi đầu, công ty Altera thay đổi công nghệ chế tạo PLD từ công nghệ lưỡng cực sang công nghệ CMOS vì công nghệ CMOS đạt được hiệu suất cao về không gian (mật độ tích hợp cao hơn). Như PAL16L8 có mật độ tích hợp từ 100 lên 150 cổng,

PAL22V10 có 500 đến 600 cổng và EP310 (là vi mạch đại diện cho họ EPLD) có trên 1000 cổng.

Cấu trúc của cổng PLD bao gồm cả khối điều khiển cấu trúc I/O. Cấu hình của ACB giống như cấu trúc của cổng PLD của vi mạch PAL và GAL nhưng có chức năng hoạt động đơn giản hơn. Trong đó mỗi cổng có 8 biến ngõ vào cùng với một biến để điều tổ hợp ngõ ra tác động ở mức cao hoặc thấp hay các tín hiệu được ghi cũng tác động ở mức cao hoặc thấp. Đối với tín hiệu hồi tiếp về mảng AND được đưa về từ thanh ghi ở ngõ ra. Các cổng đệm ngõ ra được điều khiển bằng các biến riêng cho phép các chân của vi mạch có thể hoạt động hai chiều. Ngoài ra EP310 cũng có cầu chì bảo vệ chống sao chép và giờ đây cầu chì bảo vệ trở thành một tiêu chuẩn cho các thế hệ PLD mới. Một số vi mạch tiêu biểu cho họ EPLD là EP900, có cấu tạo 40 chân, bên trong có 24 khối ACB, mật độ tích hợp hơn 1000 cổng với các tổ hợp ngõ ra có lựa chọn. Nhưng trong tương lai kỹ thuật ngày càng phát triển thì mật độ tích hợp có thể lên đến hơn 10000 cổng logic trong một chip.

Ngoài vi mạch EP900 thì công ty Altera còn giới thiệu vi mạch EP1800 có 68 chân với các chức năng được mở rộng hơn so với EP900 vì số cổng logic trong IC được tăng gấp đôi và số ngõ vào cũng vậy. Vi mạch EP1800 có thể thực hiện đồng thời 4 chức năng khác nhau, có thể xem như đó là 4 vi mạch rời. Những vi mạch số lập trình đang hướng đến mật độ tích hợp trên 1000 cổng logic trong một chip thì đang gây ảnh hưởng đến các PLD có mật độ tích hợp thấp. Công ty Cypress Semicondutor đang sản xuất các sản phẩm ứng dụng công nghệ CMOS có tốc độ cao. Sản phẩm cạnh tranh của họ chủ yếu là các họ PAL thông thường và PAL 22V10 đã tạo ra thế hệ PAL có công suất thấp, tốc độ cao nên được ứng dụng rộng rãi vào các lĩnh vực công nghiệp khác nhau. Những vi mạch phổ biến của công ty Cypress Semicondutor như CY7C330, CY7C331 và CY7C332

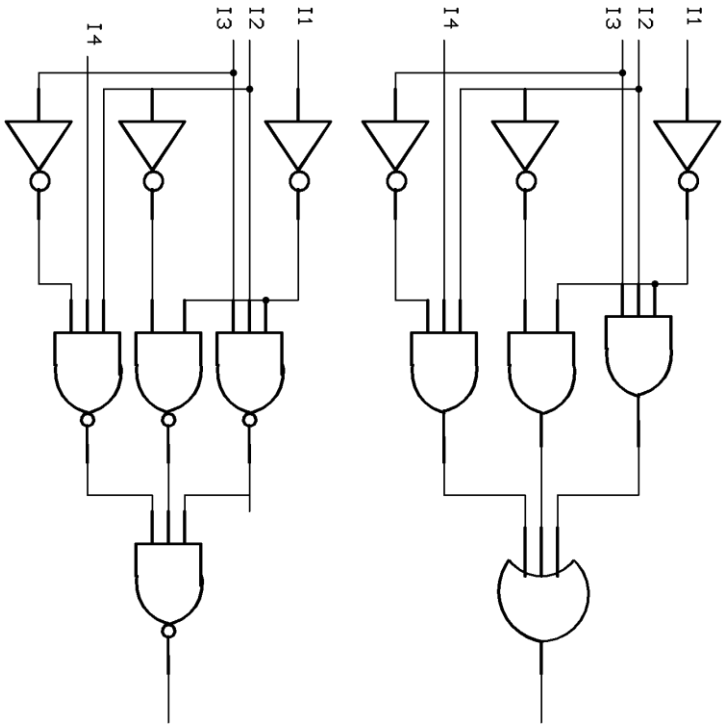
2.8 Họ vi mạch PML (Programmable Macro Logic).

Họ vi mạch được công ty Signetics sử dụng cấu trúc mới gọi là "foldback" (gấp về). Mạch logic "foldback" sử dụng một cổng NAND đơn hay mảng NOR kết hợp với một cấu trúc liên kết lập trình trung tâm cho phép thực hiện nhiều mức logic khác nhau để liên kết với macro ngõ vào và

ngõ ra. Như trong họ vi mạch PML, một mảng NAND được sử dụng vì cổng NAND có tốc độ truyền nhanh nhất trong công nghệ lưỡng cực.

Từ khóa macro để tham khảo một khối chức năng và có thể xác định một tín hiệu ngõ vào, một cổng đệm ngõ ra hay bất cứ một hàm logic nào như FF, mạch đếm hay mạch tổ hợp. Công ty Signetics phân loại các macro như sau: ngõ vào là macro ngõ vào, macro ngõ ra và những khối chức năng khác như thanh ghi hay mạch tổ hợp thì gọi là macro chức năng. So với cấu trúc mảng AND – OR của các họ IC PAL và FPLA thì cấu trúc mảng NAND phức tạp hơn

Các cổng đệm ngõ ra 3 trạng thái được điều khiển bằng từng cổng NAND riêng để tạo nên tính linh hoạt trong thiết kế.



Hình 1.6. Mạch logic cấu trúc Flodback

2.9 Họ vi mạch ERASIC(Erasable Programmable Application Specific IC).

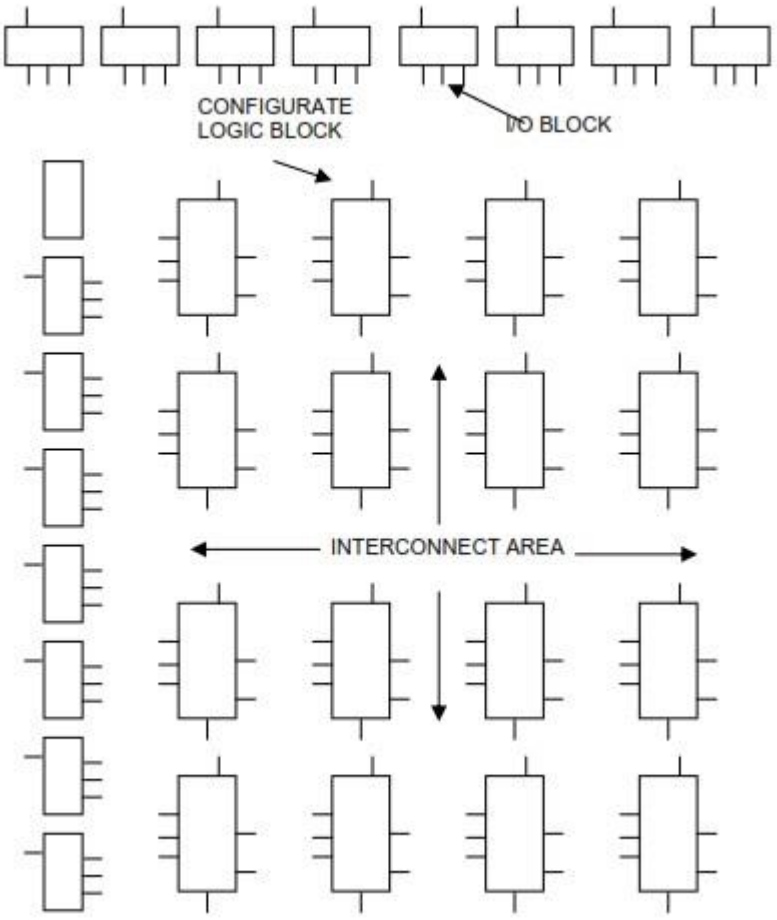
Họ vi mạch ERASIC được giới thiệu bởi công ty Exel Microeletronics có cấu trúc tương tự như họ PML nhưng được chế tạo bằng công nghệ CMOS EEPROM khác với họ PML dùng công nghệ lưỡng cực . Một đặc điểm khác biệt nữa là họ ERASIS sử dụng cấu trúc mảng NOR , vì trong công nghệ CMOS cổng NOR có thời gian truyền nhanh nhất. Vi mạch đầu tiên của họ này là XL 78C800 có 24 chân với mật độ thích hợp khoảng 800 cổng.

XL78C800 có 12 ngõ vào và 10 chân I/O được liên kết với các cổng lập trình. Chân số 1 là đường cung cấp tín hiệu xung clock cho FF JK, chân 13 dùng để điều khiển các cổng đệm ngõ ra cho các cổng lập trình, 8 ngõ vào được đưa vào mảng NOR thông qua các mạch lật, 2 cổng NOR được sử dụng để điều khiển mạch lật. Ngõ ra của cổng NOR được cấu tạo bằng các khối PCE (Polarity Control Element) để tăng tính ling hoạt.

XL78C800 có 32 biến ở ngõ vào cổng NOR, hai biến dùng để điều khiển mạch lật và có 30 biến dùng cho cổng lập trình.

2.10 Họ vi mạch LCA (Logic Cell Array)

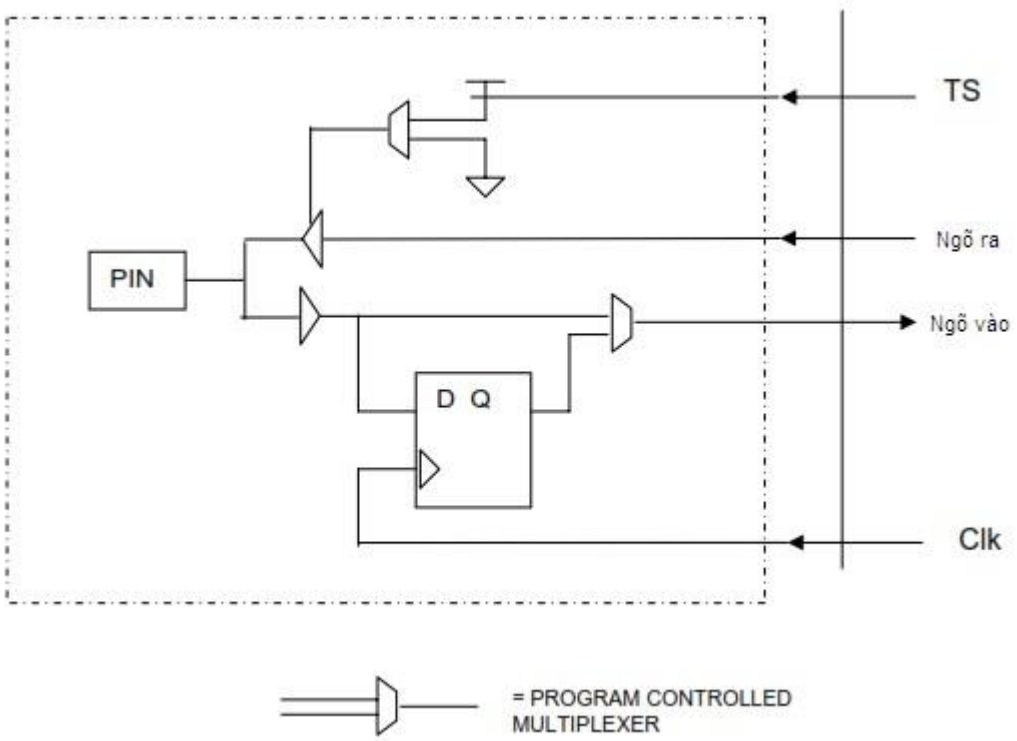
Họ LCA được công ty Xillinx giới thiệu dựa theo các cấu trúc của công ty MMI, đã trình bày một cấu trúc độc đáo trong các họ của PLD. Cấu trúc truyền thống của các họ vi mạch PAL và FPLA là các mảng AND – OR. Các cổng lập trình có cấu trúc của họ LCA gọi là cấu trúc lập trình cho người sử dụng. Đặc biệt là trong cấu tạo của LCA, họ dùng RAM động để tạo ra các chức năng logic theo yêu cầu thiết kế. Nhược điểm của các tế bào RAM động thường không ổn định. Do đó các chức năng sẽ trở lại trạng thái ban đầu khi mất điện. Để hỗ trợ cho vấn đề này họ sử dụng thêm phương pháp lưu trữ mới có chức năng tương tự như ROM. Cấu trúc của LCA được mô tả ở hình 1.7, bao gồm một khối IOB bao quanh ma trận của khối LCB.



Hình 1.7. Cấu trúc LCA

Liên kết các đường tín hiệu dọc và ngang giữa 2 khối giúp cho việc kết nối giữa 2 khối thêm thuận tiện. Vi mạch đầu tiên của họ LCA là XC 2064, có mật độ thích hợp khá phức tạp khoảng 1200 cổng logic, 58 khối IOB cùng một ma trận 8x8 hàng và cột tạo ra 64 khối CLB. Tạo ra một vi mạch khác là XC 2018 có mật độ tích hợp khoảng 1800 cổng, có 74 khối IOB cùng một ma trận 10x10 tạo ra 100 khối CLB. Vi mạch có các đường tín hiệu xung clock, tín hiệu reset đặc biệt và mạch tạo dao động thạch anh bên trong IC dùng để kết nối với các phần tử dao động bằng thạch anh bên ngoài.

Hình 1.9 trình bày sơ đồ của khối IOB, bao gồm 1 cổng đệm ngõ vào, bộ đa hợp IN – MUX và FFD. Mức điện áp ngưỡng ở ngõ vào cổng đệm thích hợp cho cả hai họ TTL và CMOS. Ngõ ra của FFD được nối với ngõ vào của bộ đa hợp và ở ngõ ra của bộ đa hợp có thể nối 1 hay nhiều khối CLB. Ngõ ra của khối IOB gồm 1 cổng đệm 3 trạng thái được nối thẳng tới chân IC



Hình 1.9. Cấu trúc khối vào/ra (IOB) của LCA

3. PHẦN MỀM HỖ TRỢ PLD

Các phần mềm hỗ trợ cho các vi mạch lập trình được các công ty phát triển liên tục, ngày càng có nhiều tính đa dạng, có thể hỗ trợ cho nhiều loại vi mạch khác nhau nên có tính cạnh tranh mạnh mẽ trong thị trường vi mạch lập trình.

3.1. Phần mềm PALASM 2 (PAL Assembler)

PALASM 2 của công ty MMI là phần mềm tiêu chuẩn cho các vi mạch lập trình. Đây là bộ biên dịch thế hệ thứ 2 hỗ trợ cho các vi mạch hoạt động không đồng bộ, như các vi mạch họ PAL của công ty MMI, vi mạch họ PLA và các vi mạch của công ty AMD.

3.2. Phần mềm AMAZE.

Phần mềm AMAZE được công ty Signetics phát triển và nó được cung cấp cho các khách hàng sử dụng vi mạch lập trình của công ty. Module chính của phần mềm AMAZE là BLAST (Boolean logic & State Transfer) dùng để biên dịch các thông tin ngõ vào chuyển đổi sang các file chương trình chuẩn của Signetics (các file có phần mở rộng là ‘STD ’). AMAZE hỗ trợ để mô phỏng các vectơ kiểm tra để thiết kế theo yêu cầu của người sử dụng.

3.3. Phần mềm PLAN (Programmable Logic Analysis).

Phần mềm PLAN được công ty National Semiconductor giới thiệu hỗ trợ cho các vi mạch lập trình cỡ vừa và nhỏ. PLAN là một ngôn ngữ đơn

giản, dùng để thực hiện các biểu thức của đại số Boolean và có khả năng giao tiếp với các công cụ lập trình để lập trình cho vi mạch.

3.4. Phần mềm HELD (Harris Enhanced Language for Programmable Logic).

Công ty Harris phát triển phần mềm HELD để hỗ trợ cho các khách hàng sử dụng vi mạch lập trình của họ. HELD sử dụng giao diện tương tự như phần mềm PLAN nhưng cũng có những điểm khác biệt. HELD không có khả năng lựa chọn các vi mạch lập trình nhưng có khả năng kiểm tra lỗi tổng quát. Ngoài ra HELD còn yêu cầu các phương trình ngõ vào ở dạng tổng các tích (SOP)

3.5. Phần mềm PLPL (Programmable Logic Programming Language).

PLPL được công ty Avanced Micro Devices giới thiệu vào năm 1984. Đây là phần mềm tiến bộ nhất so với các phần mềm trước, có những đặc điểm mới và khả năng cài đặt được mở rộng hơn so với phần mềm AMAZE. Những đặc điểm mới như cho phép định nghĩa và sử dụng các chân của vi mạch cho một nhóm tín hiệu cũng như sử dụng các phương trình của đại số Boolean. PLPL cũng hỗ trợ các phương trình phức tạp có nhiều cấp logic khác nhau. Ngoài ra bộ biên dịch này cũng để ứng dụng nguyên lí Demorgan, các hàm của đại số Boolean nhưng không bắt được ở dạng tổng của các tích do đó cho phép cú pháp linh hoạt hơn.

3.6. Phần mềm APEEL (Assembler for Programmable Electrically Erasable Logic).

Vào năm 1987, Công ty International Cmos Technology giới thiệu trình biên dịch APEEL. APEEL là một trình biên dịch đơn giản phù hợp với các yêu cầu thiết kế vừa và nhỏ và có chức năng mô phỏng. APEEL gồm một chương trình soạn thảo toàn màn hình và ở ngõ ra theo tiêu chuẩn của JEDEC. Nhưng khuyết điểm của bộ biên dịch này là không hỗ trợ để tối giản các biểu thức logic. Phần mềm APEEL cài đặt trên các máy tính cá nhân của công ty IBM và các công ty khác thích hợp với nó.

3.7. Phần mềm IPLDS II (Intel Programmable Logic Devolopment System II).

Phần mềm IPLDS II được công ty Intel giới thiệu để hỗ trợ cho các vi mạch họ EPLD. Điều cơ bản của phần mềm này là cho phép thiết kế theo 2 phương pháp là phương pháp dùng phương trình đại số Boolean và phương

pháp liệt kê các lệnh. Để tối giản các biểu thức logic IPLDS II sử dụng thuật giải đơn giản ESPRESSO II – MV. Đó là thuật giải được phát triển bởi đại học California, nó được dùng để thực hiện việc rút gọn các tích số trong các hàm logic của các vi mạch do công ty Intel sản xuất. Tương tự như các phần mềm trước, IPLDS II cài đặt được trong các máy tính của công ty IBM và các máy tính khác có cấu hình thích hợp, được sử dụng kèm với công cụ lập trình cho vi mạch.

3.8. Phần mềm CUPL (Universal Compiler for Programmable Logic).

CUPL được công tyAssited Technology giới thiệu vào năm 1983. Đây là bộ biên dịch vạn năng được hỗ trợ cho 29 loại vi mạch các loại kể cả PROM và các công ty chế tạo vi mạch lập trình khác. CUPL là một ngôn ngữ mạnh hỗ trợ cho các phương trình của đại số Boolean , bảng sự thật và thiết kế sơ đồ trạng thái, CUPL được sử dụng hầu hết các máy vi tính cá nhân trên các hệ điều hành khác nhau như trên máy vi tính của công ty IBM hay CP/M, VAX/ UNIX và VAX/ VMS.

3.9. Phần mềm ABEL (Advanced Boolean Expression Language).

ABEL là phần mềm của công ty Data I/0, nó được sử dụng hầu hết các loại vi mạch lập trình khác nhau kể cả EPROM. Đây là bộ biên dịch vạn năng có nhiều chức năng hỗ trợ tương tự như CUPL.

Trên đây là giới thiệu sơ lược các phần mềm hỗ trợ cho vi mạch lập trình để soạn thảo và lập trình cho các vi mạch. Ngoài ra còn nhiều phần mềm của các công ty khác được sản xuất để hỗ trợ cho các vi mạch lập trình của họ.

BÀI 2: MẢNG LOGIC LẬP TRÌNH

1. GIỚI THIỆU CHUNG

Các nhà sản xuất IC đã rất thành công trong việc tích hợp hàng triệu transistor trên một bản mạch. Nhưng công nghệ này có thể giúp giảm số lượng thiết bị trong hệ thống số ?

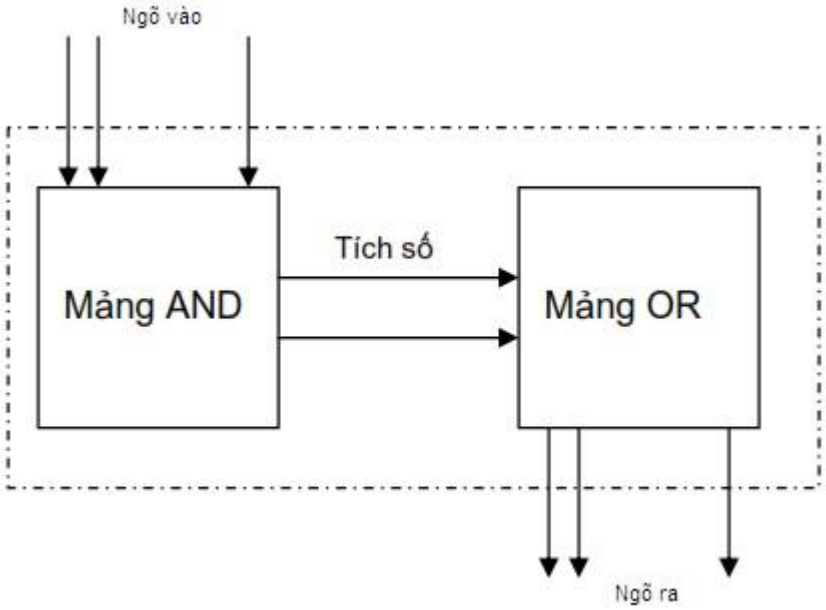
Vấn đề cơ bản ở đây là phải tìm ra các khối logic vạn năng có thể áp dụng được cho nhiều thiết kế logic khác nhau. Ví dụ như các cổng AOI có thể dùng trong nhiều thiết kế song các khối này cũng không giúp nhiều trong việc làm giamr các số lượng thiết bị, điều này có thể thực hiện bằng cách đưa vào sử dụng một khối chứa hàng trăm cổng.

Nhưng cấu trúc các khối này phải như thế nào ?

Một giải pháp không khéo cho tình huống này là sắp xếp các cổng AND và OR (hoặc NAND và NOR) theo một cấu trúc mảng được tổng quát hóa mà các điểm nối giữa chúng với nhau có thể lập trình được để thực hiện một hàm logic xác định. Những khối logic mục đích chung như vậy được gọi là PLA hoặc PAL.

2. PLA và PAL

Hình 2.1 biểu diễn sơ đồ khối của một thiết bị mảng logic lập trình. Đó là các thiết bị nhiều đầu vào và nhiều đầu ra được tổ chức thành một mảng con các phần tử AND và một mảng con các phần tử OR. Mảng con AND sẽ chuyển các đầu vào thành các tích số (product term) phụ thuộc vào các điểm nối được lập trình. Mảng con OR sẽ cộng các tích số này với nhau để tạo ra biểu thức dạng tổng các tích cuối cùng.



Hình 2.1. cấu trúc mảng logic lập trình

Một thiết bị PLA có thể thực hiện một tập hợp các hàm khá phức tạp. Sự phức tạp này được xác định bởi số lượng đầu vào (input), số lượng các tích số (chính là số lượng các cổng AND) và số lượng các đầu ra (chính là số lượng các cổng OR) mà PLA có thể cung cấp.

Chẳng hạn một FPLA tiêu biểu kiểu TTL có thể có 16 đầu vào, 48 tích số và 8 đầu ra được đóng gói trong một vỏ 24 chân dữ liệu. Nó tương đương với 48 cổng AND 16 đầu vào và 8 cổng OR 48 đầu vào. Nếu xét một vi mạch SSI 12 chân dữ liệu thì nó chỉ có 4 cổng hai đầu vào, từ đó có thể thấy tác dụng thực sự của các mảng logic. Ví dụ để thực hiện các hàm boole sau đây:

$F0 = A\overline{BC}$

$F1 = A.\overline{C} + A.B$

$F2 = \overline{B}C + A.B$

$F3 = B.C + A$

Các phương trình trên có thể được mô tả bằng tập hợp các biến (A, B, C), các tích số (A, BC , A.$\overline{C}$, AB, B.$\overline{C}$) và các hàm (F0, F1, F2, F3). Những chữ này tương đương với số đầu vào của mảng AND, các đầu ra từ mảng AND sẽ tương đương với số đầu vào của mảng OR và các đầu ra của mảng OR là các hàm. Để thực hiện các hàm này phải cần một PLA có ít nhất 3 đầu vào, 5 tích số và 4 đầu ra.

Một cách tiện lợi để mô tả các hàm là dùng một ma trận đặc tính được vẽ trong hình 2.2. Nó mô tả đầu vào nào phải nối với cổng AND để hình thành tích số mong muốn (1 = Biến không đảo; 0 = Biến đảo; - = Không nối) và những tích số nào phải được OR với nhau để tạo thành các đầu ra cuối cùng (1 = Nối; 0 = Không nối). các hàng trong bảng xác định các tích số, các cột biểu diễn các đầu vào và đầu ra. Một tích số tham gia trong nhiều hàm nếu như có nhiều giá trị 1 trên hàng của nó trong các cột ra của ma

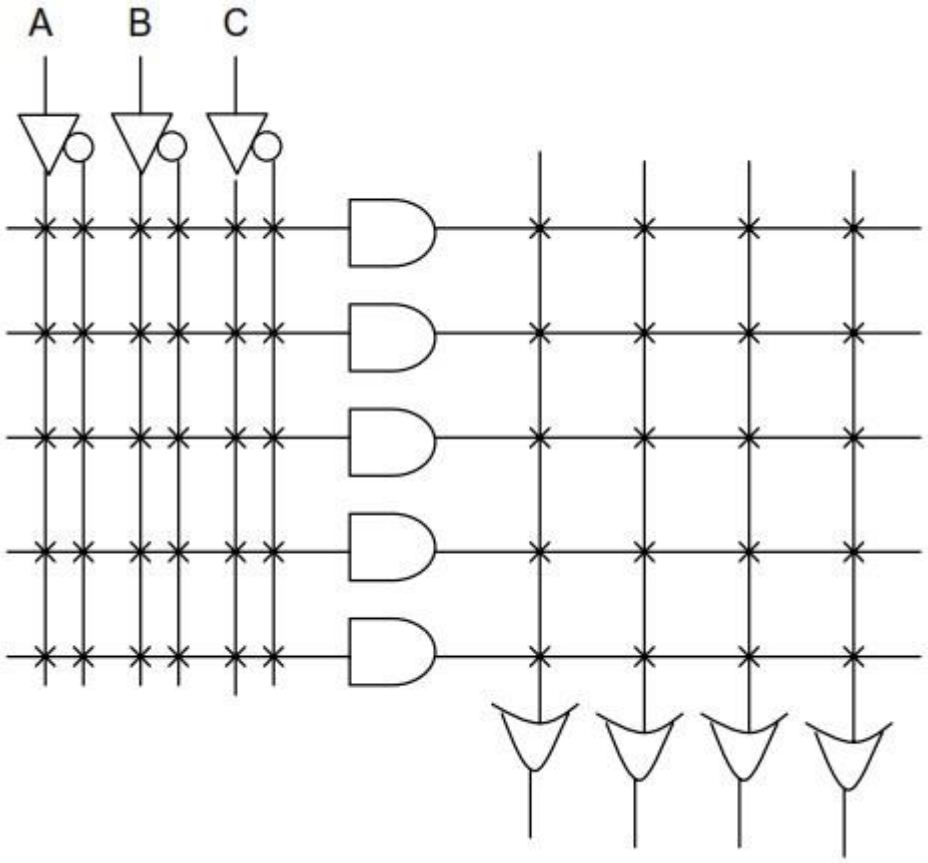
trận đặc tính. Trong hình 2.2 dễ dàng nhận thấy rằng các tích số A, AB, $\overline{B}.\overline{C}$ được dùng trong nhiều hàm

Tích số	A	B	C	F0	F1	F2	F3
AB	1	1	-	0	1	1	0
BC	-	0	1	0	0	0	1
AC	1	-	0	0	1	0	0
BC	-	0	0	1	0	1	0

A	1	-	-	1	0	0	1

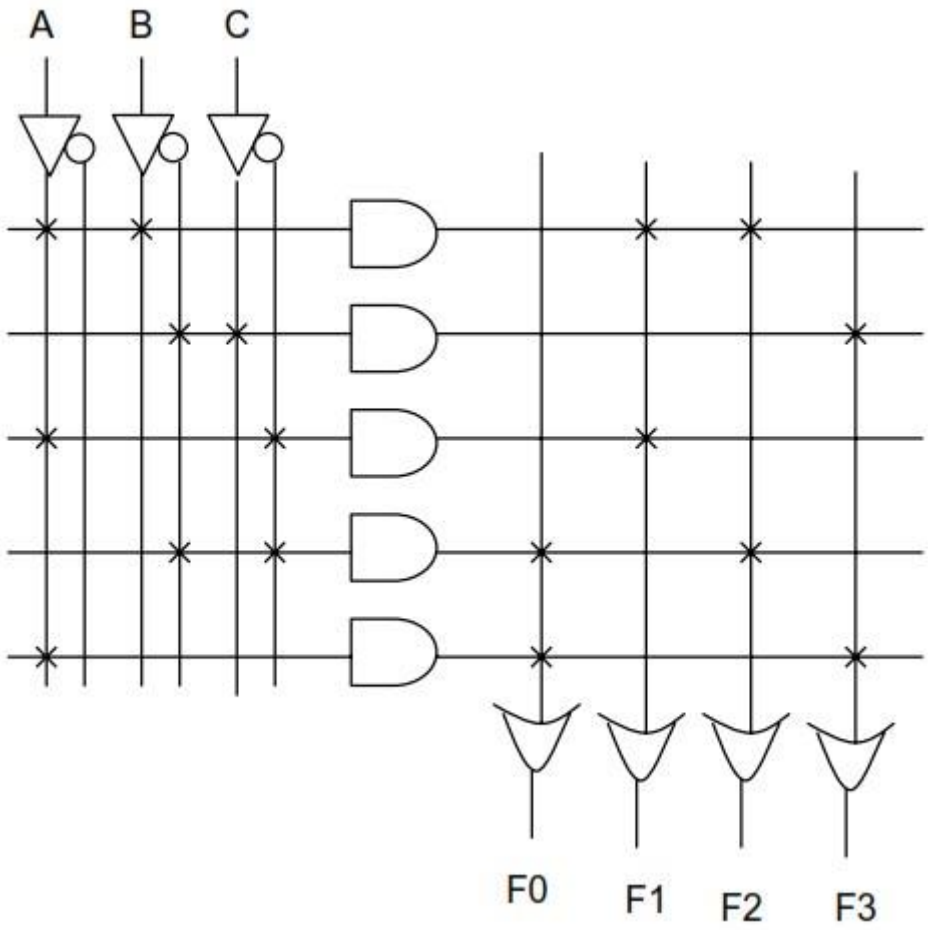
Hình 2.2. Bảng ma trận đặc tính

Hình 2.3 và 2.4 vẽ các sơ đồ thực hiện các hàm ở trên dưới dạng các cổng rời rạc tương đương với mảng logic lập trình. Hình 2.3 là mảng trước khi lập trình với tất cả các điểm nối giữa đầu vào với các cổng. Có thể đặc tính hóa mảng bằng cách sử dụng một thiết bị phần cứng gọi là bộ lập trình.



Hình 2.3. FLA trước khi lập trình

Những chi tiết của quá trình lập trình phụ thuộc vào các vi mạch cụ thể. Một kỹ thuật thường hay được sử dụng là đặt các cầu chì giữa tất cả các điểm nối ở đầu vào và đầu ra của một cổng. Một cầu chì là một tiếp giáp điện được thiết kế đặc biệt để có thể bị đứt dưới dòng điện lớn. Bằng cách cho một dòng điện cao chảy qua các cầu chì được chọn, phần cứng lập trình sẽ phá hỏng các tiếp giáp này. Phần mềm lập trình sẽ phân tích các phương trình boole để xác định những cầu chì nào cần phá hỏng những cầu chì nào cần để lại. Hình 2.4 trình bày mảng logic sau khi đã lập trình.



Hình 2.4. PLA sau khi lập trình

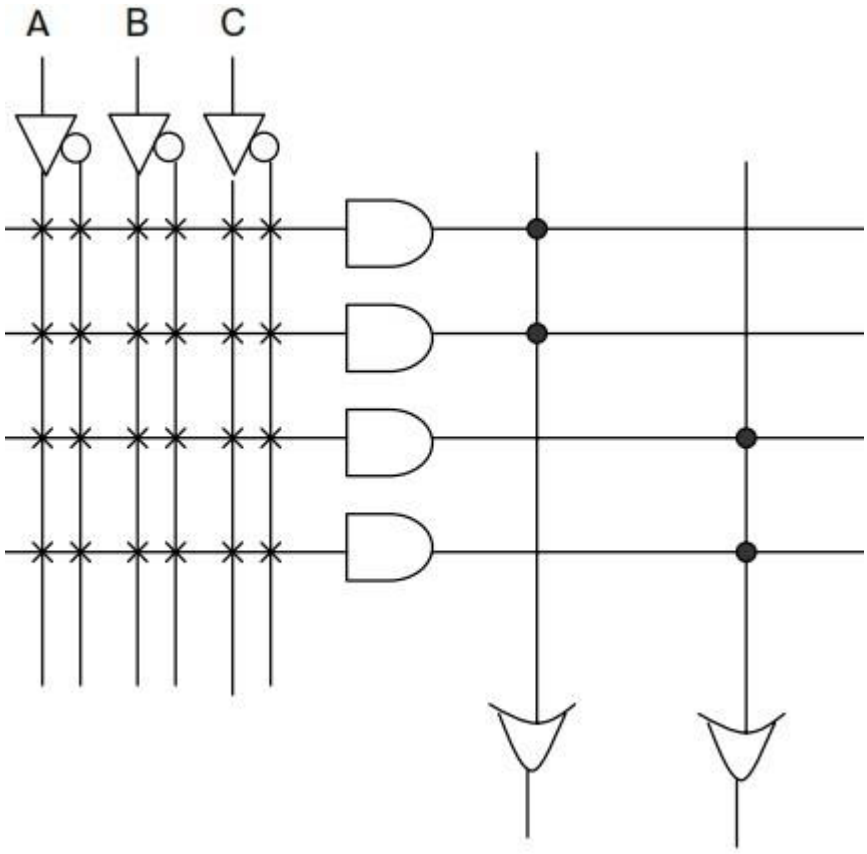
Sự khác nhau giữa PLA và PAL

Trong hình 2.3 cho thấy cả hai mảng con AND và OR có thể đặc tính hóa theo bất kỳ cách nào mà người thiết kế mong muốn. Những thiết bị như thế được gọi là mảng logic chương trình hóa (Programmable Logic Array – PLA).

Trong thực tế có những mảng logic không cung cấp khả năng lập trình đầy đủ. Ví dụ các thiết bị PAL (Programmable Array Logic) có mảng AND lập trình được còn mảng OR thì được xác định trước. Số lượng tích số đi vào một cổng OR thường được giới hạn là: 2, 4, 8 hoặc 16. Có một sự bù trừ trong các thiết bị PAL giữa độ phức tạp của các hàm với số lượng tích số trên một cổng OR và số lượng các hàm độc lập mà thiết bị có thể thực hiện được. Cổng OR có độ chịu tải đầu vào (fan – in) càng cao thì thiết bị PAL càng ít đầu ra.

Ví dụ một họ PAL có thể tồn tại dưới 3 dạng, mỗi dạng đều có 16 đầu vào, 16 tích số nhưng khác nhau về tổ chức của mảng OR: 4 cổng OR mỗi cổng 4 đầu vào, 2 cổng OR mỗi cổng 8 đầu vào hoặc 1 cổng OR 16

đầu vào. Còn mảng AND thì vẫn hoàn toàn có thể lập trình. Hình 2.5 biểu diễn một thiết bị PAL có 3 đầu vào, 4 tích số và 2 đầu ra còn mảng OR đã được xác định sẵn, trong trường hợp này các cổng OR bị giới hạn 2 tích số mỗi cổng.



Hình 2.5. Cấu trúc PAL

Điểm khác chính giữa PLA với PAL là: PLA có thể lợi dụng các tích số chung để chia sẽ cho nhiều đầu ra còn thiết bị PAL không thể làm điều này. Như vậy nên dùng một PLA khi cần thực hiện một tập hợp các hàm mà có nhiều tích số chung để có thể chia sẻ cho các đầu ra.

Mặt khác một thiết bị PLA sẽ chậm hơn so với PAL do sự khác nhau về trở kháng của tiếp giáp, một có thể lập trình còn một thì đã được định vị cứng. Các tiếp giáp có thể lập trình được cấu tạo từ các cầu chì có trở kháng cao hơn là các tiếp giáp đã được định vị cứng. Vì vậy tín hiệu đi qua hai tiếp giáp lập trình trong PLA bị trễ hơn khi chỉ đi qua một tiếp giáp lập trình trong PAL.

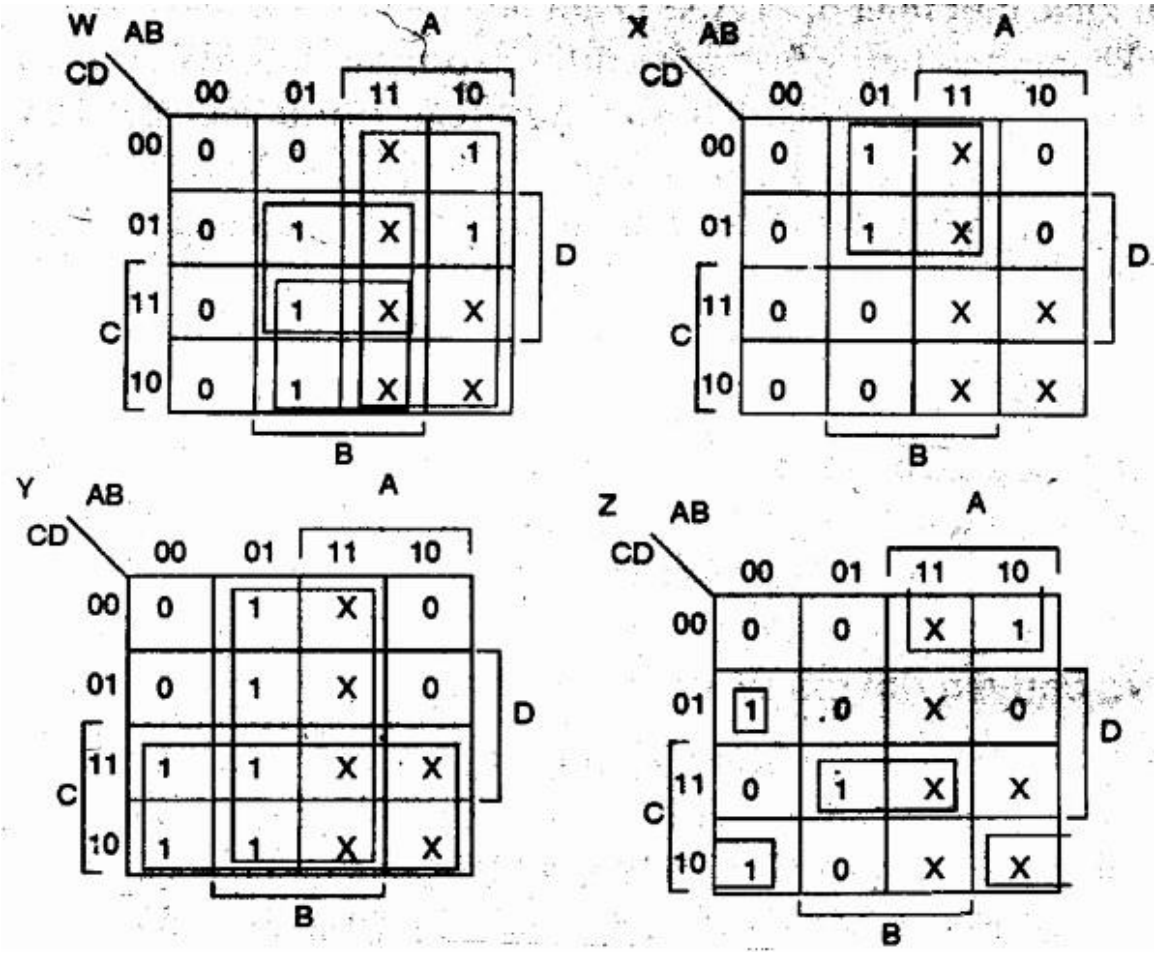
3. CÁC VÍ DỤ THIẾT KẾ

Phần dưới đây sẻ trình bày hai ví dụ thiết kế khác nhau đó là một bộ chuyển đổi mã và một bộ so sánh để minh họa cách thực hiện mạch logic tổ hợp bằng cách dùng PLA hoặc PAL **3.1. Bộ chuyển mã BCD sang Gray**

Nhiệm vụ của bộ chuyển mã này là chuyển một số BCD 4 bít thành mã gray 4 bít. Mỗi số trong mã gray chỉ khác so với số cạnh nó 1 bít. Mạch có 4 đầu vào A, B, C và D biểu diễn số BCD và 4 đầu ra W, X, Y, Z biểu diễn mã gray 4 bít

A	**B**	**C**	**D**	**W**	**X**	**Y**	**Z**
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Hình 2.6. Bảng sự thật bộ chuyển mã BCD sang Gray



Hình 2.7. Bảng Kanaugh bộ chuyển mã

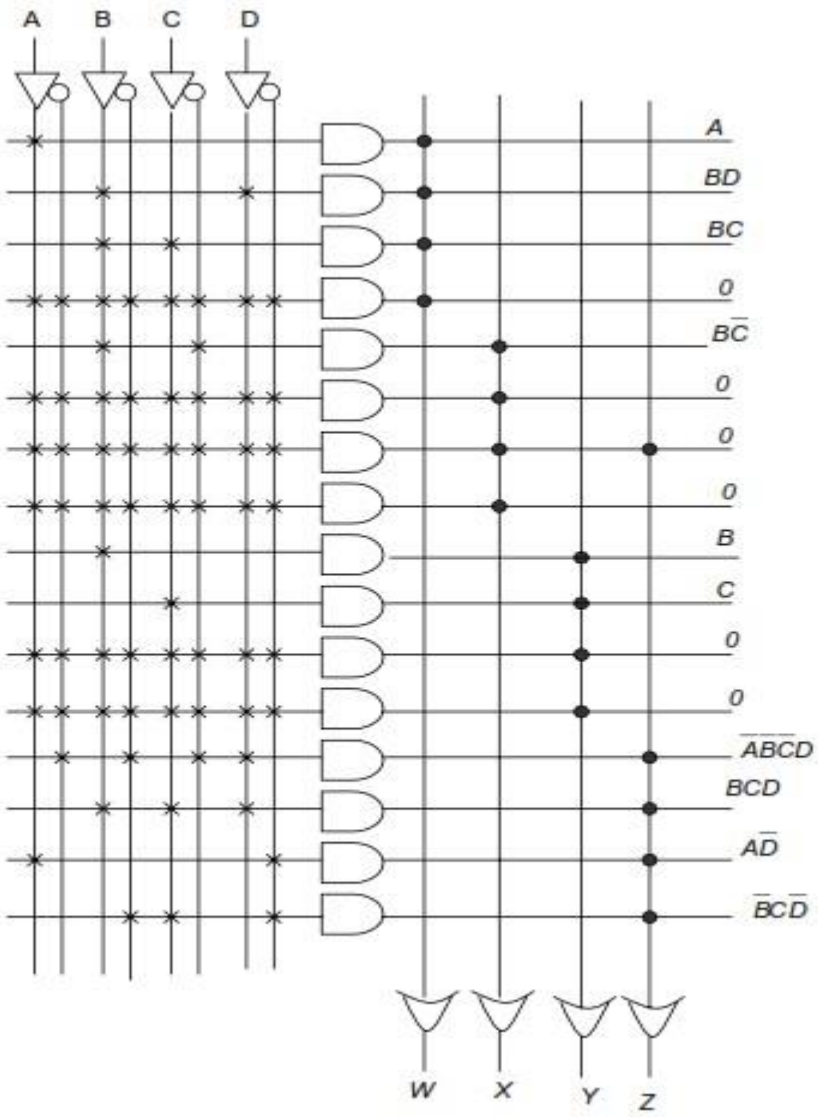
Kết quả sau khi tối thiểu hóa như sau

$$W = A + BD + BC$$

$$Y = B + C$$

$$X = B.\overline{C}$$

$$Z = ABC.D \square BCD \square A.\overline{D} \square \overline{B}.C.\overline{D}$$



Hình 2.8. Thực hiện bộ chuyển mạch bằng thiết bị PAL

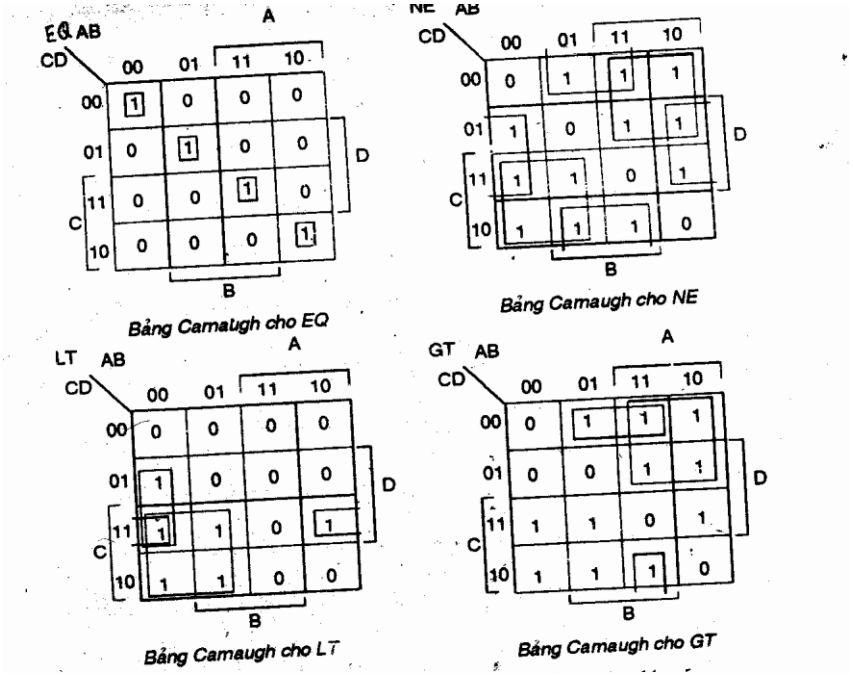
Vì tập hợp các hàm này không có tích số nào chung để chia sẻ nên chúng thích hợp với cách thực hiện bằng thiết bị PAL (hình 2.8). Thiết bị này gồm 4 cổng OR 4 đầu vào nhưng có nhiều cổng AND không dùng đến, bài này có thể thực hiện bằng thiết bị PLA đơn giản hơn nhưng tốc độ sẽ chậm hơn.

3.2. Bộ so sánh hai bít

Đầu vào của mạch là hai số nhị phân hai bít ký hiệu là AB và CD. Đầu ra là 4 hàm xác định bởi giá trị logic của các mệnh đề.

AB = CD (EQ: Bằng nhau) AB ≠ CD (NE: Không bằng nhau)

AB < CD (LT: Nhỏ hơn) AB > CD (GT: Lớn hơn)



Hình 2.9. Bản đồ Kanaugh bộ so sánh

Các phương trình logic sau khi tối thiểu hóa

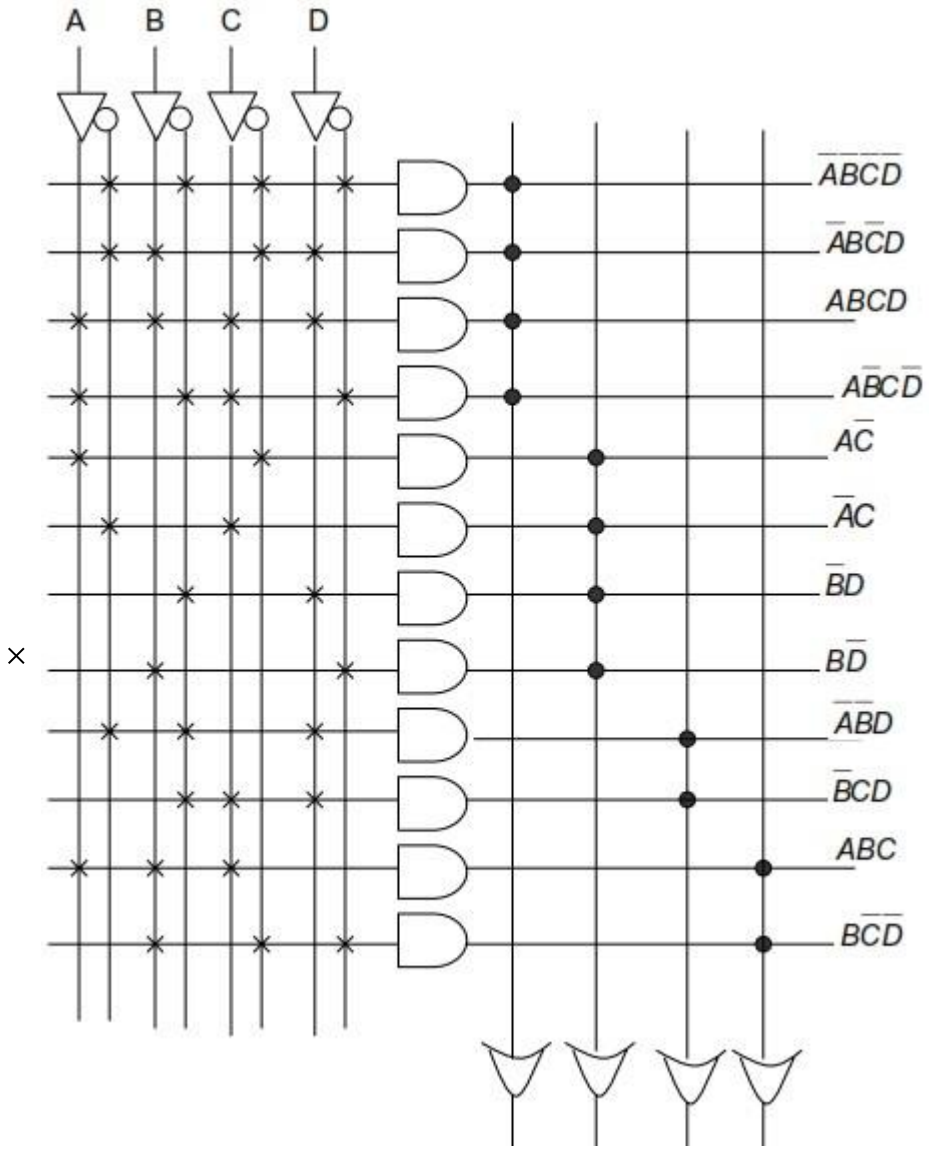
$$EQ = \overline{A}\overline{B}CD + A\overline{B}\overline{C}D + AB\overline{C}\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D}$$

$$NE = A\overline{C} + \overline{A}C + B\overline{D} + \overline{B}D$$

$$LT = \overline{A}C + \overline{A}BD + \overline{B}CD$$

$$GT = AC + ABD + BCD$$

Các hàm sử dụng 14 tích số trong đó có 2 tích số ($A\overline{C}$ và $\overline{A}C$) được dùng 2 lần nên dùng PLA có lợi hơn. Một PAL hoặc PLA có thể thay thế từ 5 đến 15 vi mạch số rời. Đó là lý do hiện nay chúng rất được dùng để thực hiện các hệ logic.



Hình 2.10. Thực hiện bộ so sánh bằng PLA

4. CÁC MẢNG LOGIC LẬP TRÌNH THÔNG DỤNG

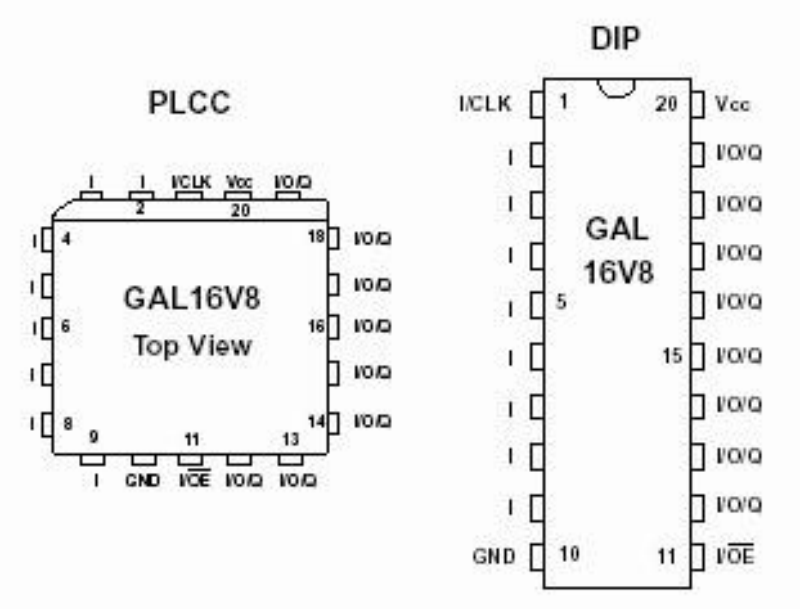
4.1. GAL16V8C

Là một vi mạch được sản xuất bởi hảng LATTICE có trì hoãn truyền tối đa 5 nS, là một kết hợp giữa công nghệ CMOS hiệu năng cao với công nghệ ghi xóa được bằng điện nhằm tạo ra các PLD tốc độ cao trên thị trường. Do thời gian xóa cao (< 100 mS) cho phép thiết bị được lập trình lại một cách nhanh chóng và hiệu quả.

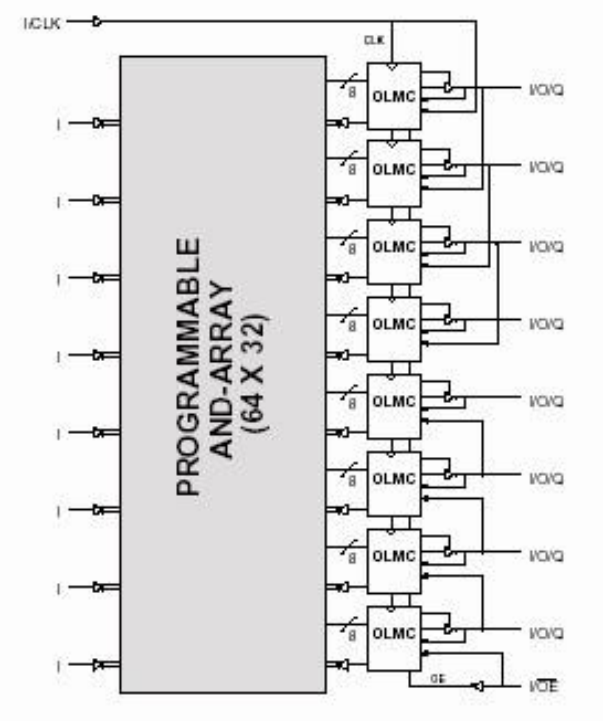
Cấu trúc tổng quát của thiết bị cung cấp khả năng thiết kế linh động nhất nhờ vào tính năng cấu hình ngõ ra OLMC bởi người dùng, một tập hợp phụ nhiều khả năng cấu hình của GAL16V8 là các cấu trúc PAL được liệt kê trong một bảng ở phần mô tả macrocell. Các thiết bị GAL16V8 cho phép mô

phỏng một cấu trúc PAL bất kỳ với sự tương thích hoàn toàn về chức năng/bản đồ cầu chì/thông số.

Mạch thử đồng nhất và các tế bào lập trình lại cho phép thử toàn bộ đặc tính AC, DC và chức năng trong khi sản xuất. Kết quả là công ty LATTICE bảo đảm 100% chức năng và khả năng lập trình tất cả các sản phẩm GAL. Thêm vào đó còn bảo đảm 100 lần ghi/xóa và dữ liệu được lưu trữ trên 20 năm.



Hình 2.11. Hình dạng vỏ GAL 16V8



Hình 2.12. Sơ đồ khối GAL 16V8

Đặc tính kỹ thuật

* Công nghệ CMOS và E^2
    - Trì hoãn truyền tối đa 5 nS
    - Fmax = 166 MHz
    - Trễ tối đa 4 nS kể từ lúc có xung đồng hồ cho đến khi nhận được dữ liệu ra
* Công suất giamr từ 50% đến 75% so với loại lưỡng cực
    - Dòng Icc điển hình 75 mA đối với loại công suất thấp
    - Dòng Icc điển hình 45 mA đối với loại ¼ công suất
* Có điện trở kéo lên ở tất cả các chân

* Công nghệ tế bào E^2
    - Cho phép logic cấu hình lại
    - Các tế bào lập trình lại được
    - Bảo đảm kiểm tra 100%
    - Tốc độ xóa cao (< 100 mS)
    - Khả năng lưu trữ dữ liệu 20 năm * 8 OLMC
    - Rất linh hoạt đối với các thiết kế logic phức tạp
    - Lập trình được cực tính ngõ ra
    - Mô phỏng được các thiết bị PAL 20 chân với chức năng/bản đồ cầu chì/thông số tương thích
* Nạp trước và reset khi mở nguồn tất cả các thanh ghi
    - Khả năng kiểm tra chức năng 100%
* Ứng dụng điển hình
    - Điều khiển DMA
    - Điều khiển máy trạng thái
    - Xử lý đồ họa tốc độ cao
    - Nâng cấp tốc độ logic chuẩn
* Có ký hiệu nhận dạng điện tử

4.1.1. Ngõ ra OLMC

Trong phần này giới thiệu về cách cấu hình ngõ ra macrocell. Trong thực tế công việc này được thực hiện bằng các công cụ phần mềm cũng như phần cứng và hoàn toàn dễ dàng sử dụng.

Có 3 chế độ cấu hình chung đó là: Simple, complex và registered. Hai bít toàn cục SYN và AC0 xác định cấu hình cho tất cả các ngõ ra macrocell. Bít XOR của mỗi macrocell xác định cực tính ngõ ra của một trong 3 chế độ. Trong khi bít AC1 của mỗi macrocell có nhiệm vụ điều khiển cấu hình I/O. Hai bít toàn cục kết hợp với 16 bít cấu trúc riêng định nghĩa tất cả các cấu hình của GAL16V8. Thông tin cung cấp bởi các bít cấu trúc này chỉ nhằm làm hiểu rõ hơn về thiết bị. Chương trình dịch sẽ thiết lập các bít cấu trúc này một cách rõ ràng thông qua việc định nghĩa chân linh kiện, do đó người dùng không cần thao tác trực tiếp đến chúng.

4.1.2. Trình dịch hỗ trợ OLMC

Phần mềm dịch hỗ trợ 3 chế độ OLMC toàn cục khác nhau khi các kiểu thiết bị khác nhau. Các kiểu thiết bị này được liệt kê trong bảng dưới đây. Hầu hết các trình dịch đều có khả năng tự động chọn kiểu thiết bị thường dựa vào việc sử dụng thanh ghi và chân cho phép ra (OE). Việc dùng thanh ghi sẽ buộc phần mềm chọn chế độ thanh ghi. Tất cả các ngõ ra tổ hợp với OE được điều khiển bởi thành phần tích số sẽ buộc phần mềm chọn chế độ complex. Phần mềm chỉ chọn chế độ simple khi tất cả các ngõ ra đều là dạng tổ hợp không có điều khiển OE. Các kiểu thiết bị khác nhau được liệt kê trong bảng có thể được dùng để hủy bỏ việc tự động lựa chọn thiết bị của phần mềm. Để biết thêm chi tiết có thể tham khảo tài liệu của chương trình dịch.

Khi dùng phần mềm để cấu hình thiết bị cần phải lưu ý đến những giới hạn cho mỗi chế độ như sau

	Registered	**Complex**	**Simple**	**Tự động chọn chế độ**
ABEL	P16V8R	P16V8C	P16V8AS	P16V8
CUPL	G16V8MS	G16V8MA	G16V8AS	G16V8
LOG/iC	GAL16V8_R	GAL16V8_C7	GAL16V8_C8	GAL16V8
OrCAD-PLD	“Registered” [1]	“Complex” [1]	“Simple” [1]	GAL16V8A
PLDesigner	P16V8R[2]	P16V8C[2]	P16V8C[2]	P16V8A

TANGO-PLD	G16V8R	G16V8C	G16V8AS[3]	G16V8

4.1.3. Chế độ thanh ghi

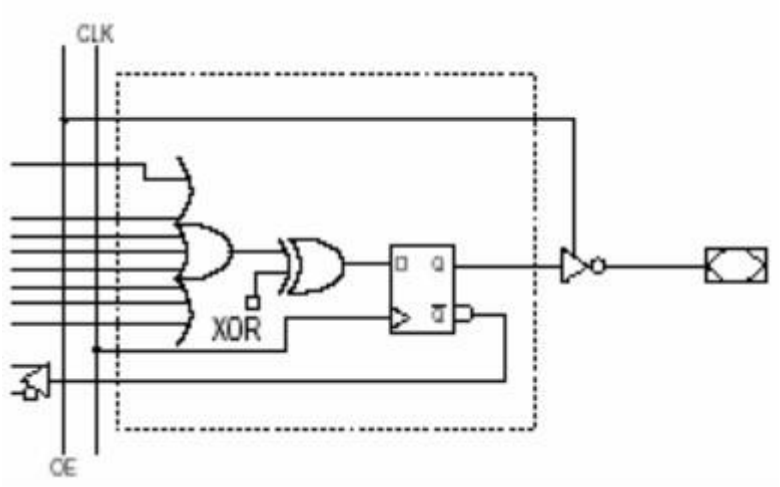
Trong chế độ này các macrocell được cấu hình thành các ngõ ra dạng thanh ghi hoặc chức năng I/O.

Các cấu hình kiến trúc trong chế độ này tương tự các thiết bị 16R8 và 16RP4 với sự bố trí I/O, thanh ghi và cực tính thay đổi.

Tất cả các macrocell kiểu thanh ghi chia sẻ chung chân xung đồng hồ và chân cho phép ra. Một macrocell bất kỳ có thể được cấu hình là thanh ghi hoặc I/O. Chế độ này cho phép đến 8 thanh ghi hoặc 8 I/O. các chức năng vào ra định sẵn có thể được thực hiện như một tập hợp phụ của chức năng I/O.

Mỗi ngõ ra kiểu thanh ghi có 8 thành phần tích số, mỗi ngõ ra kiểu I/O có 7 thành phần tích số.

Số lượng cầu chì JEDEC bao gồm cầu chì UES (User Electronic Signature) và PTD (Product Term Disable) được trình bày trong sơ đồ logic sau đây



Cấu hình thanh ghi trong chế độ thanh ghi

SYN = 0

AC0 = 0

XOR = 0 Ngõ ra tích cực mức thấp

XOR = 1 Ngõ ra tích cực mức cao

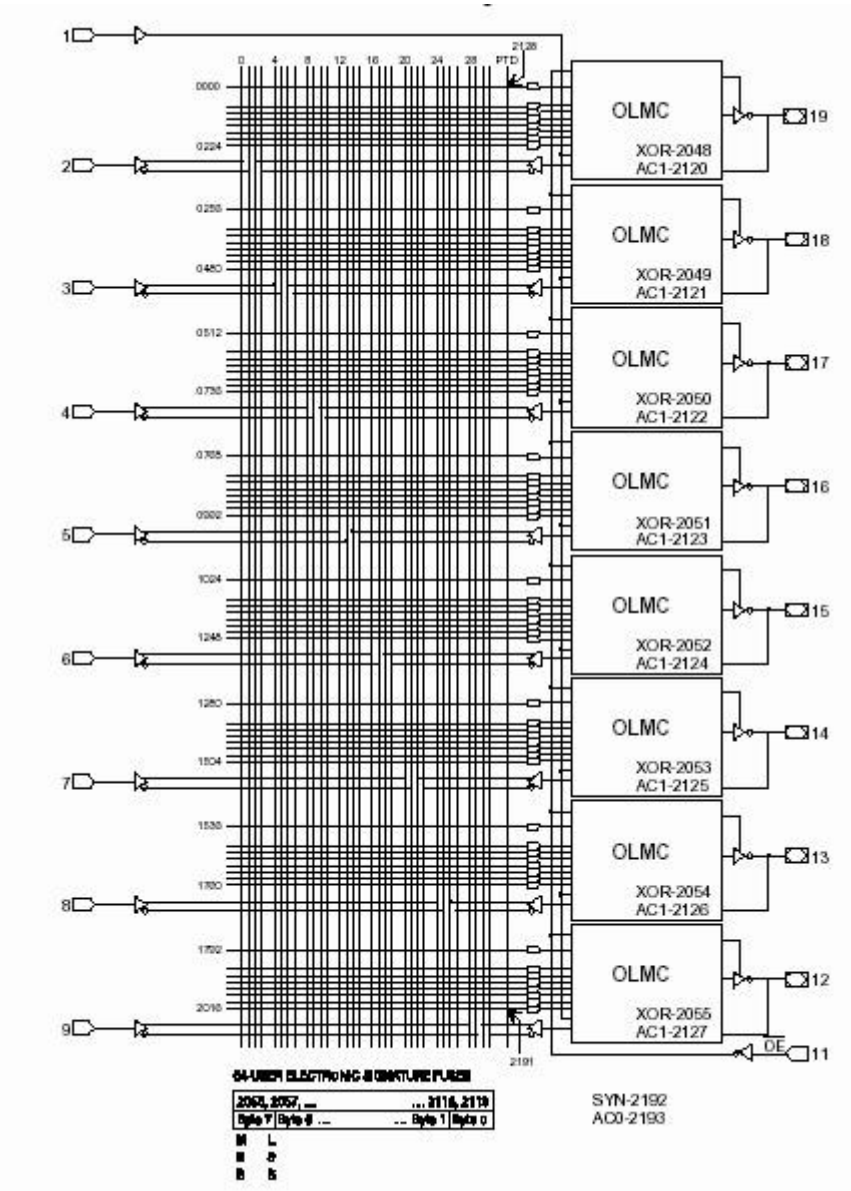
AC1 = 1 Cấu hình ra

Chân1 là đồng hồ chung cho các ngõ ra thanh ghi

Chân 11 là $\overline{OE}$ cho các ngõ ra thanh ghi

Chân 1 và 11 luôn được cấu hình là CLK và $\overline{OE}$

Ghi chú: Phần mềm phát triển tự động cấu hình tất cả các bít điều khiển kiến trúc và kiểm tra việc sử dụng các chân tương ứng



Hình 2.13: Sơ đồ khối chế độ thanh ghi

4.1.4. Chế độ complex

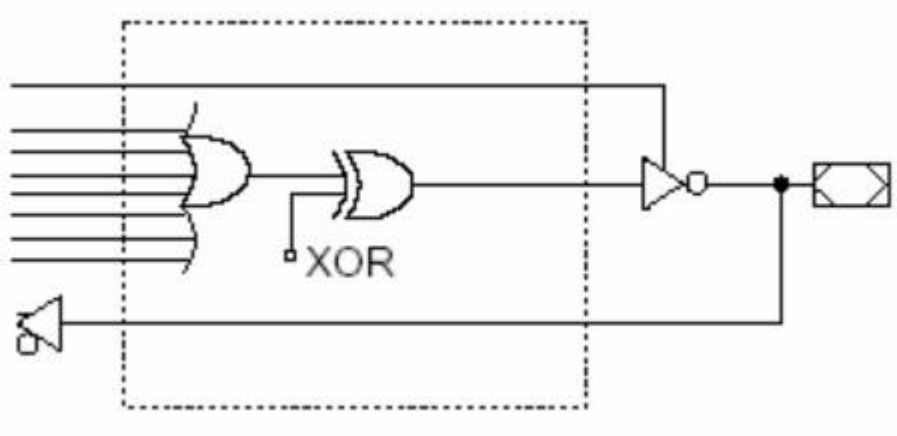
Trong chế độ này các macrocell chí được cấu hình là ngõ ra hoặc chức năng I/O. các cấu hình kiến trúc trong chế độ này tương tự các thiết bị 16L8 và 16P8 với cực tính lập trình được cho mỗi macrocell

Cho phép lên đến 6 I/O, các ngõ vào và ra định sẵn có thể được thực hiện như các tập hợp phụ của chức năng I/O. Hai macrocell ngoài cùng (chân 12 và 19) không thể là ngõ vào. Các thiết kế cần 8 I/O có thể thực hiện ở chế độ thanh ghi.

Mỗi macrocell có 7 thành phần tích số, một tích số được dùng để điều khiển chức năng cho phép ra. Chân 1 và 11 luôn là ngõ vào dữ liệu cho mảng

AND

Số lượng cầu chì JEDEC bao gồm các cầu chì UES và PTD được trình bày trong sơ đồ logic sau đây



Cấu hình I/O tổ hợp trong chế độ complex

SYN = 1

AC0 = 1

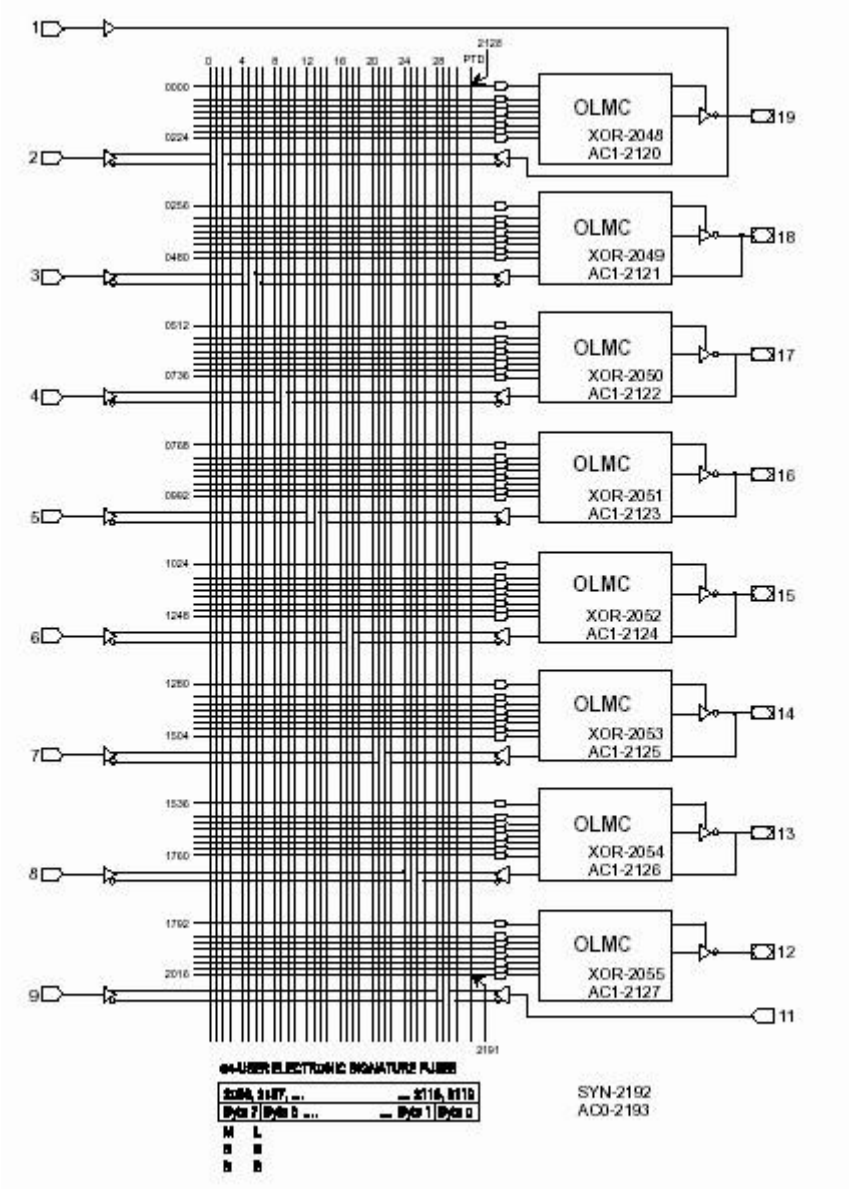
XOR = 0 Ngõ ra tích cực mức thấp

XOR =1 Ngõ ra tích cực mức cao

AC1 = 1

Chân 13 đến 18 được cấu hình chức năng này

Ghi chú: Phần mềm phát triển tự động cấu hình tất cả các bít điều khiển kiến trúc và kiểm tra việc sử dụng các chân tương ứng



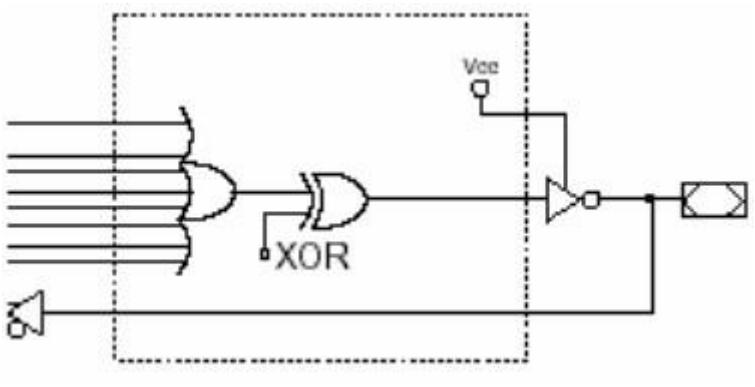
Hình 2.14. Sơ đồ khối chế độ Complex

4.1.5. Chế độ simple

Trong chế độ này các macrocell được cấu hình như các ngõ vào hoặc các ngõ ra tổ hợp định sẵn. các cấu trúc kiến trúc trong chế độ này tương tự như các thiết bị 10L8 và 12P6 với nhiều sự hoán vị của cực tính ra hoặc việc lựa chọn vào.

Tất cả ngõ ra trong chế độ simple có tối đa 8 tích số có thể điều khiển logic. Thêm vào đó cực tính của mỗi ngõ ra có thể lập trình được.

Chân 1 và 11 luôn là ngõ vào dữ liệu cho mảng AND, hai macrocell trung tâm (chân 15 và 16) không thể dùng làm ngõ vào hoặc chân I/O và chỉ là các ngõ ra định sẵn



Ngõ ra tổ hợp có hồi tiếp trong chế độ simple

SYN = 1

AC0 = 0

XOR = 0 Ngõ ra tích cực mức thấp

XOR =1 Ngõ ra tích cực mức cao

AC1 = 0 Xác định cấu hình này

Tất cả OLMC trừ chân 15 và 16 đều được cấu hình chức năng này

Ký hiệu nhận dạng

Một ký hiệu nhận dạng điện tử được cung cấp cho mỗi thiết bị GAL16V8, nó gồm 64 bít nhớ lập trình được. Các bít này có thể chứa dữ liệu định nghĩa bởi người dùng. Trong một số ứng dụng bao gồm mã nhận dạng người dùng, các số đọc lại hoặc điều khiển kiểm kê. Dữ liệu nhận dạng luôn cho phép người dùng không phụ thuộc vào trạng thái bít bảo mật.

Ghi chú: Ký hiệu nhận dạng điện tử được bao gồm trong phép tính kiểm tra tổng. Việc thay đổi ký hiệu nhận dạng sẽ làm thay đổi kiểm tra tổng.

Bít bảo mật

Một bít bảo mật được cung cấp trong GAL16V8 để ngăn việc sao chép không bản quyền nội dung các mảng. Mỗi khi được lập trình bít này sẽ ngăn không cho phép đọc các bít chức năng trong thiết bị. Bít này chỉ có thể xóa bằng cách lập trình lại thiết bị. Vì vậy cấu hình nguyên thủy không bao giờ được kiểm tra mỗi khi bít này được lập trình. Nhưng dữ liệu nhận dạng thì luôn được phép bất chấp trạng thái của bít này.

Chống điện tích khóa

Các thiết bị GAL16V8 được thiết kế có tính năng nạp trong hệ thống với nền được phân cực âm. Sự phân cực âm giúp làm giamr điện tích khóa gây ra bởi các nhiễu xung âm ở ngõ vào. Thêm vào đó, ngõ ra còn được thiết

kế với kênh N kéo lên thay vì thông thường là kênh P kéo lên để triệt hiện tượng khóa gây ra bởi sự vọt lố ở ngõ ra.

Lập trình thiết bị

Các thiết bị GAL được nạp chương trình bằng một thiết bị lập trình logic phù hợp chuẩn của LATTICE. Chương trình được nạp hoàn tất chỉ trong vòng vài giây, quá trình xóa dễ dàng đối với người dùng và được thực hiện tự động trong thời gian lập trình.

Nạp trước thanh ghi ra

Khi kiểm tra kết quả thiết kế máy trạng thái phải kiểm lại tất cả các trạng thái và sự chuyển trạng thái của hệ. Điều này là cần thiết vì trong các tình huống hoạt động nhất định có thể tạo ra tín hiệu logic tại trạng thái không hợp lệ (bật nguồn, nhiễu lưới điện…). Để kiểm tra thiết kế trong những điều kiện như thế phải cung cấp một phương pháp cho phép cách ly đường hồi tiếp và áp đặt một trạng thái bất kỳ vào thanh ghi, sau đó hệ tiếp tục hoạt động và giá trị ra được kiểm tra tại các trạng thái thích hợp tiếp theo.

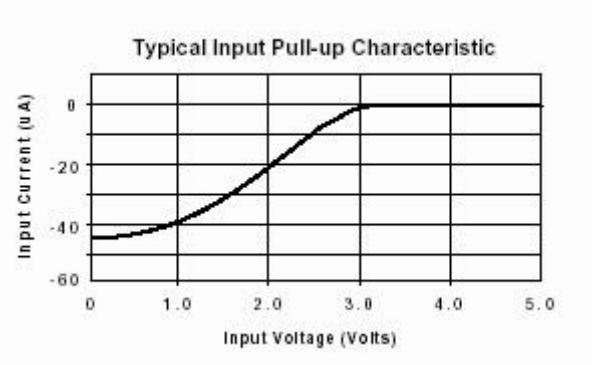
Sơ đồ mạch của GAL16V8 cho phép mỗi ngõ ra thanh ghi được thiết lập đồng bộ với một trong hai mức cao hoặc thấp. Do đó, một trạng thái hiện tại bất kỳ có thể được áp đặt để kiểm tra hoạt động tuần tự. Khi cần thiết các thiết bị nạp GAL có khả năng chạy các vectơ thử tự động thực hiện việc nạp trước các thanh ghi ra.

Bộ đệm ngõ vào

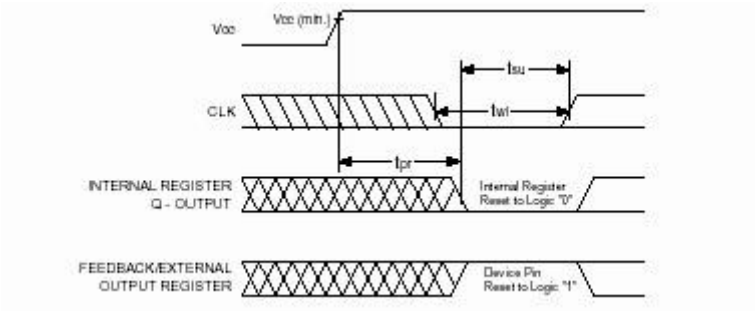
Thiết bị Gal16V8 được thiết kế với các bộ đệm vào tương thích với mức TTL. Các bộ đệm này có trở kháng vào cao nên làm giamr dòng tải hơn nhiều so với các thiết bị TTL lưỡng cực.

Ngõ vào của GAL16V8 và các chân I/O có đặt sẵn các điện trở kéo lên, nên các ngõ vào và các chân I/O không dùng sẽ được treo lên mức cao (logic 1). LATTICE khuyến cáo nên nối các ngõ không dùng này đến các ngõ vào khác, Vcc hoặc GND. Việc làm này nhằm giảm nhiễu và dòng

Icc của thiết bị

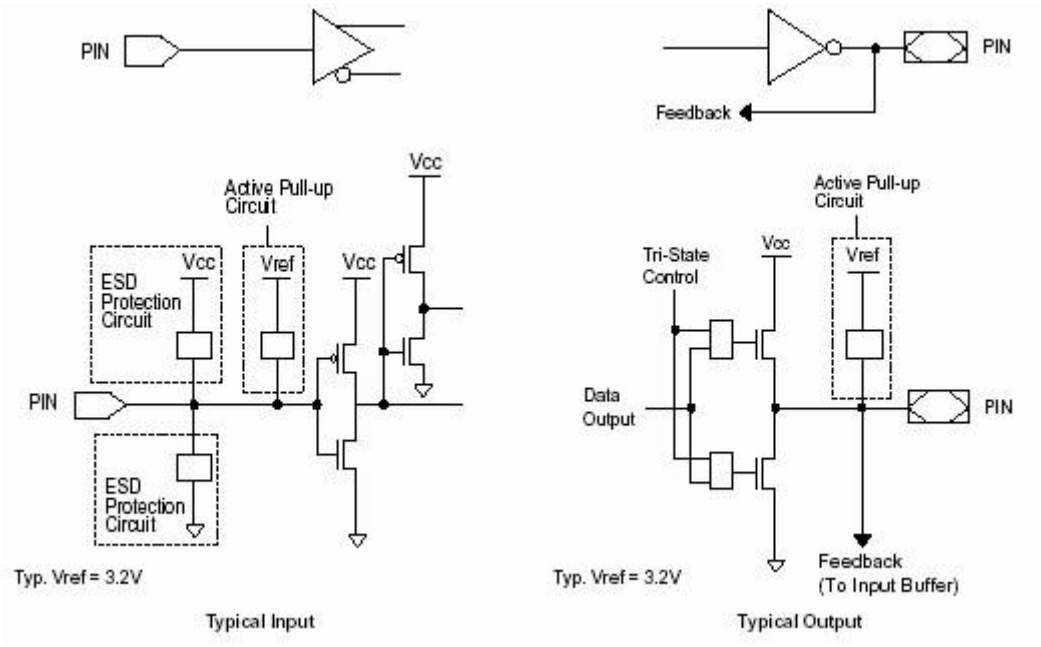


RESET khi mở nguồn



Mạch điện GAL16V8 cung cấp một tín hiệu reset cho tất cả các thanh ghi trong suốt thời gian mở nguồn, ngõ ra Q của tất cả các thanh ghi nội bộ được thiết lập mức thấp sau một thời gian xác định (tpr, 1 µS MAX). Kết quả là trạng thái của các chân ra kiểu thanh ghi (nếu chúng được cho phép) sẽ luôn ở mức cao khi mở nguồn bất chấp cực tính đã lập trình ở các chân ra. Đặc tính này có thể làm đơn giản hóa rất nhiều quá trình thiết kế máy trạng thái bằng cách tạo ra một trạng thái đã biết khi mở nguồn. Vì việc mở nguồn xảy ra ngẫu nhiên nên phải có một vài điều khiện để bảo đảm thiết bị được chắc chắn rsset. Thứ nhất điện áp Vcc phải tăng đều, thứ hai xung đồng hồ ngõ vào phải cố định ở mức TTL như trình bày trên đồ thị trong suốt thời gian mở nguồn. các thanh ghi sẽ reset trong khoảng thời gian tối đa là tpr.

Khi hệ thống hoạt động bình thường nên tránh kích thích thiết bị cho đến khi tất cả các thời gian thiết lập ngõ vào và các đường hồi tiếp hoàn tất. Xung đồng hồ cũng cần phải có bề rộng tối thiểu.



Hình 2.15. Sơ đồ tương đương vào/ra

4.2. Isp GAL22V10

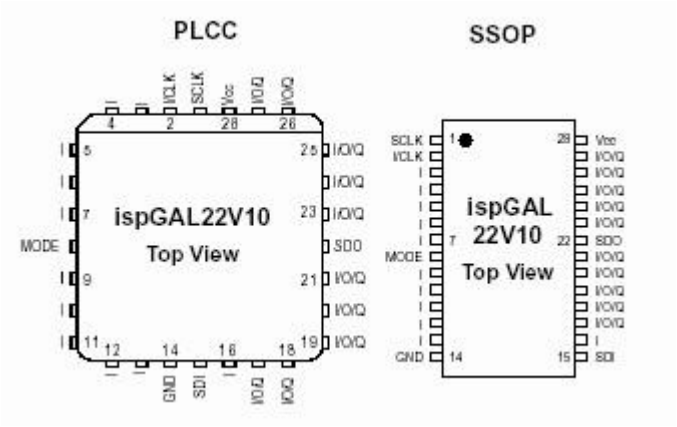
Thiết bị ispGAL22V10 có thời gian trì hoãn truyền tối đa là 7,5 nS.

Kết hợp giữa công nghệ CMOS hiệu năng cao với công nghệ cổng E^2 (Electrical Erasable floating gate) đã tạo ra sản phẩm 22V10 đầu tiên lập trình được trong hệ thống cho công nghiệp. Công nghệ E^2 có thời gian xóa nhanh (< 100 mS) cung cấp khả năng lập trình hoặc cấu hình lại thiết bị một cách nhanh chóng và hiệu quả.

Cấu trúc tổng quát tạo khả năng thiết kế linh động một cách tối đa nhờ vào sự cho phép cấu hình của OLMC bởi người dùng.

Thiết bị ispGAL22V10 có chức năng/bản đồ cầu chì/thông số tương thích với các thiết bị GAL22V10 lưỡng cực và CMOS.

Mạch điện kiểm tra đồng nhất và các tế bào lập trình lại cho phép kiểm tra đầy đủ chức năng, đặc tính AC, DC trong khi sản xuất. Thêm vào đó thiết bị còn có khả năng ghi xóa 10.000 lần và dữ liệu được lưu trữ trên 20 năm.



Hình 2.16. Sơ đồ chân

Đặc tính kỹ thuật

* Lập trình trong hệ thống (5 V)

- 4 đường giao tiếp lập trình nối tiếp
- Tối thiểu 10.000 lần ghi/xóa
- Mạch kéo xuống bên trong chân SDI làm giảm điện trở ngoài (chỉ đối với ispGAL22v10C)

* Công nghệ CMOS và E^2

- Trì hoãn truyền tối đa 7,5 nS
- Fmax = 111 MHz
- Trễ tối đa 5 nS từ khi có xung đồng hồ cho đến khi có dữ liệu ra

* Có mạch kéo lên ở tất cả các chân vào và chân I/O

* Tương thích với các thiết bị 22V10 chuẩn

- Chức năng/bản đồ cầu chì/thông số các loại 22v10 lưỡng cực và CMOS

* Công nghệ tế bào E2

- Lập trình trong hệ thống
- Bảo đảm được kiểm tra 100%
- Tốc độ xóa bằng điện cao (< 100 mS)
- Dữ liệu lưu trữ trong vòng 20 năm

* 10 OLMC

- Đạt độ linh hoạt tối đa với các thiết kế logic phức tạp

* Ứng dụng điển hình

- Điều khiển DMA
- Điều khiển máy trạng thái
- Xử lý đồ họa tốc độ cao

- Driver cấu hình phần cứng

* Nhận dạng bằng ký hiệu điện tử

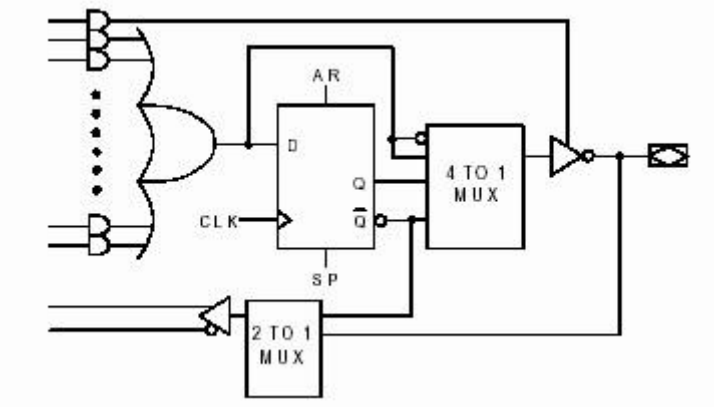
4.2.1 OLMC

Mỗi OLMC trong ispGAL22V10 có số thành phần tích số thay đổi. Hai trong 10 OLMC có 8 tích số (chân 17 và 27), hai OLMC có 10 tích số (chân 18 và 26), hai OLMC 12 tích số (chân 19 và 25), hai OLMC 14 tích số (chân 20 và 24) và hai OLMC 16 tích số (chân 21 và 23). Thêm vào đó mỗi OLMC còn có 1 tích số dành sẵn để điều khiển chức năng cho phép ngõ ra.

Cực tính ra của mỗi OLMC có thể được lập trình riêng rẽ ở một trong hai chế độ tổ hợp hoặc thanh ghi. Điều này cho phép mỗi ngõ ra có thể được cấu hình tích cực mức thấp hoặc mức cao một cách độc lập.

Thiết bị ispGAL22V10 có một tích số reset không đồng bộ (AR) và một tích số preset đồng bộ (SP). Hai tích số này dùng chung cho tất cả các OLMC kiểu thanh ghi. AR sẽ thiết lập tất cả các thanh ghi về 0 bất cứ khi nào tích số này được xác định. SP sẽ thiết lập tất cả các thanh ghi lên 1 tại cạnh lên của xung đồng hồ kế tiếp sau khi tích số này được xác định.

Lưu ý: Hai tích số AR và SP sẽ bắt buộc ngõ ra Q của FF vào cùng trạng thái bất chấp cực tính ngõ ra. Do đó, một thao tác reset sẽ làm ngõ ra các thanh ghi trở về 0, dẫn đến kết quả chân ra có thể cao hoặc thấp là tùy thuộc vào việc chọn cực tính của chân.



Hình 2.17. Cấu tạo OLMC

4.2.2. Cấu hình OLMC

Mỗi OLMC trong ispGAL22V10 có hai mode chức năng cơ bản: Thanh ghi và I/O tổ hợp. các chế độ và cực tính ra được thiết lập bởi hai bít (S0 và S1). Thông thường các bít này được điều khiển bởi chương trình dịch.

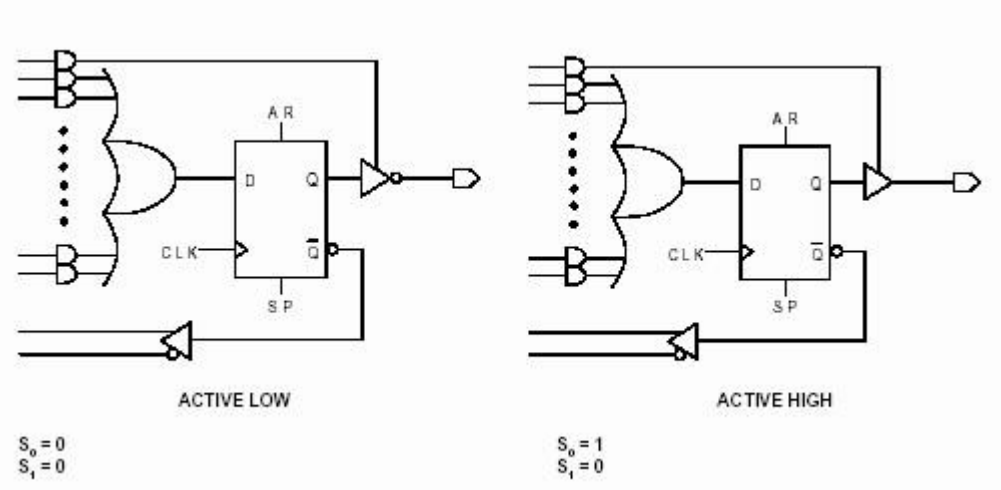
Chế độ thanh ghi

Trong chế độ này, chân ra được kết hợp với một OLMC được lái bởi ngõ ra Q của FF-D trong OLMC đó. Cực tính tín hiệu ra tại chân này có thể được chọn bằng cách xác định bộ đệm ra của nó tích cực mức cao hoặc mức thấp. Việc điều khiển ngõ ra 3 trạng thái có hiệu lực như một tích số riêng của mỗi OLMC và vì vậy có thể được định nghĩa bằng một phương trình logic. Ngõ ra $\overline{Q}$ của FF-D được hồi tiếp về mảng AND.

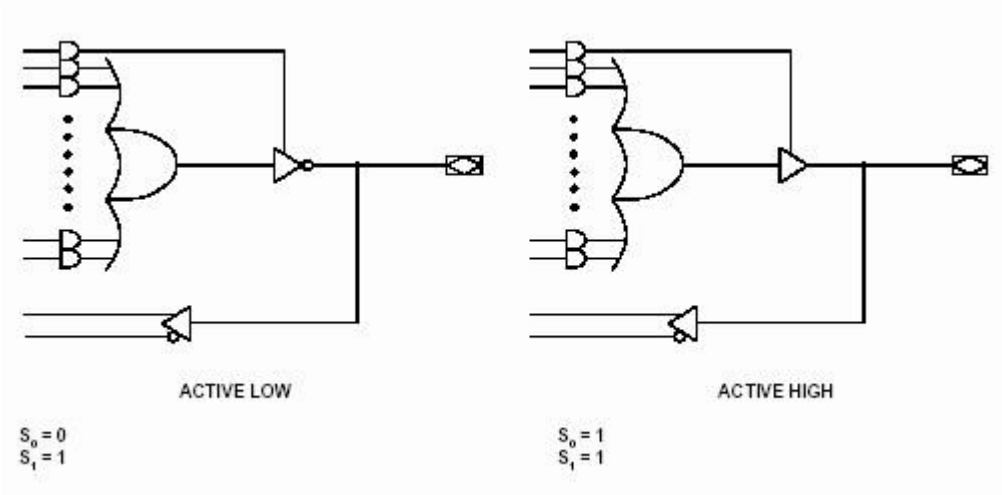
Lưu ý: Trong chế độ thanh ghi, tín hiệu hồi tiếp lấy từ ngõ ra $\overline{Q}$ của thanh ghi chứ không phải từ chân thiết bị, do đó một chân khi đã được định nghĩa là thanh ghi thì sẽ chỉ là ngõ ra và không thể dùng làm I/O động như các chân tổ hợp.

Chế độ I/O tổ hợp

Trong chế độ này một chân được liên kết với một OLMC riêng được điều khiển từ ngõ ra của một tổng số. Cực tính tín hiệu ra tại chân này có thể được chọn bằng cách xác định bộ đệm ra điều khiển tích cực mức cao hoặc mức thấp, việc điều khiển ngõ ra 3 trạng thái có khả năng như một thành phần tích số riêng cho mỗi ngõ ra và có thể được thiết lập bởi chương trình dịch là ON (ngõ ra định sẵn), OFF (ngõ vào định sẵn) hoặc tích số điều khiển (I/O động). Tín hiệu hồi tiếp đưa về mảng AND lấy từ chân của bộ đệm cho phép ra, cả hai cực tính của chân (đảo và không đảo) đều được hồi tiếp về mảng AND.



Hình 2.18. Chế độ thanh ghi



Hình 2.19. Chế độ tổ hợp

Ký hiệu nhận dạng

Mỗi ispGAL22V10 được cung cấp một ký hiệu nhận dạng điện tử (ES) gồm 64 bít nhớ cho phép lập trình lại. các bít này có thể chứa dữ liệu được định nghĩa bởi người dùng. Một số ứng dụng bao gồm mã nhận dạng người dùng, số phiên bản hoặc điều khiển kiểm kê. Dữ liệu nhận dạng luôn được phép sử dụng không phụ thuộc vào trạng thái của tế bào bảo mật.

Ký hiệu nhận dạng điện tử là một đặc tính bổ sung không có hỗ trợ trong các thiết bị 22V10 của các nhà sản xuất khác. Để sử dụng tính năng này phải chọn loại 22V10 của LATTICE khi dịch một tập hợp các phương trình logic. Thêm vào đó nhiều máy lập trình có hai khả năng lựa chọn: Một ispGAL22V10 và một ispGAL22V10-UES (User Electronic Signature) hoặc ispGAL22V10-ES, điều này cho phép người dùng duy trì sự tương thích với các thiết kế 22V10 hiện có trong khi vẫn còn khả năng tùy chọn sử dụng tính năng mở rộng của các thiết bị GAL.

Bản đồ JEDEC của ispGAL22V10 bao gồm 64 cầu chì bổ xung cho ký hiệu nhận dạng điện tử trong tổng số 5892 cầu chì. Tuy nhiên, ispGAL22V10 vẫn còn có thể được lập trình theo bản đồ JEDEC 22V10 chuẩn (5828 cầu chì) với máy lập trình bất kỳ.

Tế bào bảo mật

Mỗi ispGAL22V10 có một tế bào bảo mật để chống sao chép nội dung của mảng. Mỗi khi được lập trình tế bào này sẽ ngăn việc đọc các bít chức năng của thiết bị. Tế bào này chỉ có thể bị xóa khi lập trình lại nên cấu hình nguyên thủy có thể không bao giờ được kiểm tra. Ký hiệu nhận dạng điện tử vẫn cho phép sử dụng bất chấp trạng thái của tế bào này.

Chống nghẽn mạch

Thiết bị ispGAL22V10 được thiết kế với một bơm điện tích trên mạch để phân cực âm khối nền, biên độ phân cực âm đủ để ngăn các nhiễu xung âm ở ngõ vào có thể làm nghẽn mạch. Ngoài ra, các ngõ ra được thiết kế với mạch kéo lên kênh N thay vì thông thường là kênh P để giảm thiểu mọi khả năng nghẽn mạch cảm ứng SCR.

Lập trình thiết bị

Các thiết bị ispGAL22V10 sử dụng một file bản đồ cầu chì JEDEC 22V10 chuẩn để mô tả thông tin lập trình cho thiết bị. các chương trình dịch của hãng thứ ba có thể tạo ra file JEDEC cho thiết bị này.

Khả năng lập trình trong hệ thống

Thiết bị ispGAL22V10 có tính năng lập trình được trong hệ thống. Bằng cách tích hợp tất cả mạch lập trình điện áp cao trên chip, việc lập trình được thực hiện đơn giản bằng cách dịch chuyển dữ liệu vào thiết bị. Mỗi khi được lập trình dữ liệu trong các tế bào E^2CMOS vẫn duy trì ngay cả khi tắt nguồn.

Tất cả các điều kiện cần thiết cho việc lập trình được thực hiện thông qua 4 tín hiệu giao tiếp mức TTL. Các tín hiệu này được đưa đến mạch lập trình trên chip, nơi đó một máy trạng thái sẽ điều khiển việc lập trình. Các tín hiệu giao tiếp là: Dữ liệu vào nối tiếp (SDI), dữ liệu ra nối tiếp (SDO), xung đồng hồ nối tiếp (SCLK) và tín hiệu điều khiển chế độ (MODE).

Nạp trước thanh ghi ra

Khi kiểm tra các thiết kế máy trạng thái cần thiết phải kiểm tra tất cả các trạng thái có thể và các quá trình chuyển trạng thái. Đây là những tình huống có thể xảy ra khi máy đang hoạt động có thể xuất hiện nhiễu gây ra bởi việc tắt/mở nguồn, nhiễu trên lưới điện…Để kiểm tra một thiết kế phù hợp với các điều kiện này cần phải cung cấp một phương pháp cách ly đường hồi tiếp và áp đặt một trạng thái bất kỳ vào các thanh ghi. Sau đó máy trạng thái có thể hoạt động tiếp tục và các ngõ ra được kiểm tra đùng theo các điều kiện của trạng thái kế tiếp.

Thiết bị ispGAL22V10 bao gồm mạch điện cho phép mỗi ngõ ra kiểu thanh ghi được thiết lập đồng bộ với mức cao hoặc mức thấp. Do đó một trạng thái hiện tại bất kỳ có thể được áp đặt để kiểm tra hoạt động của máy.

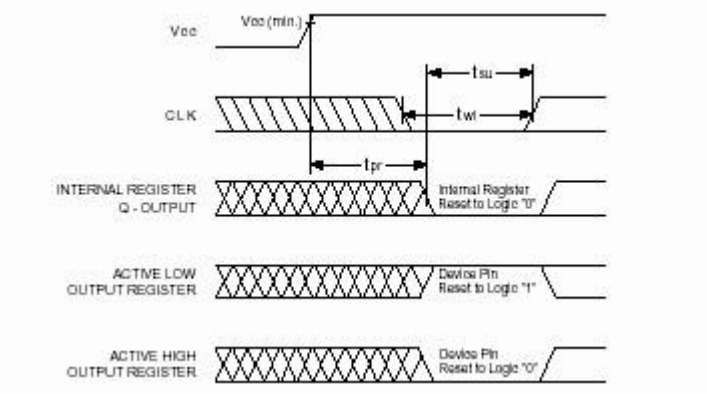
Khi cần thiết máy nạp GAL có khả năng tiến hành các véc tơ thử bằng cách tự động thực hiện việc nạp trước các thanh ghi ra.

Đệm ngõ vào

Thiết bị ispGAL22V10 được thiết kế với các bộ đệm vào tương thích TTL. Các bộ đệm này có trở kháng vào cao nên mức tải tương đương nhỏ hơn nhiều so với các tải TTL thông thường khác.

Tất cả các chân vào và chân I/O (ngoại trừ chân SDI của ispGAL22V10C) đều có mạch kéo lên bên trong, do đó các ngõ vào để hở sẽ có mức cao TTL (mức 1), chân SDI có mạch kéo xuống bên trong để giữ cho thiết bị thoát khỏi trạng thái lập trình khi chân này không được điều khiển tích cực. Tuy nhiên, công ty LATTICE khuyên nên nối các ngõ vào không dùng, các chân I/O 3 trạng thái với các ngõ vào tích cực gần đó, với Vcc hoặc với GND nhằm cải thiện khả năng chống nhiễu và giảm dòng Icc cho thiết bị.

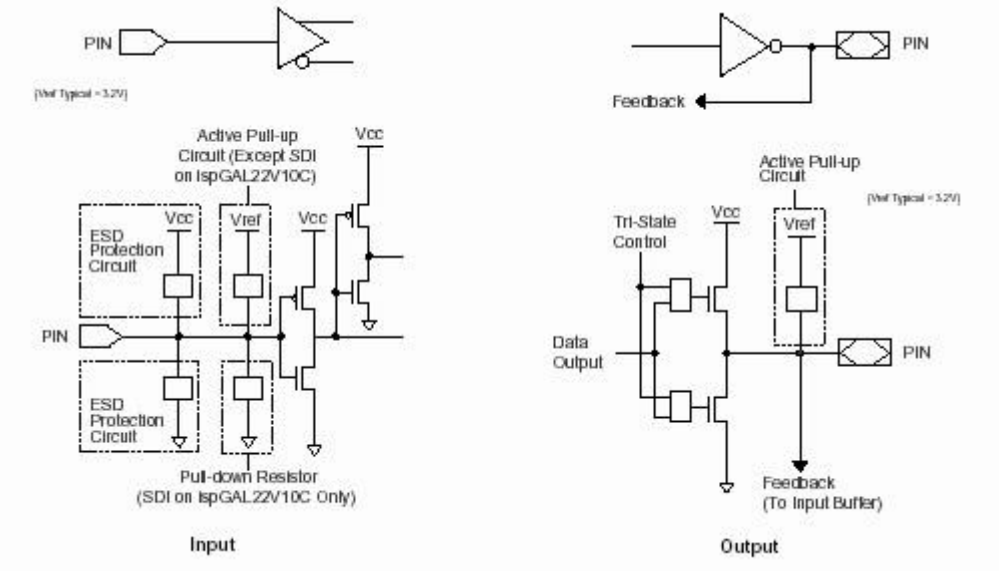
Reset khi mở nguồn



Mạch điện của ispGAL22V10 cung cấp một tín hiệu reset cho tất cả các thanh ghi trong suốt thời gian mở nguồn. Ngõ ra Q của các thanh ghi được đưa về mức thấp sau một thời gian xác định (tpr, 1 μS MAX). Kết quả là trạng thái của các chân ra kiểu thanh ghi (nếu chúng được cho phép) sẽ ở một trong hai mức cao hoặc thấp khi mở nguồn phụ thuộc vào cực tính đã được lập trình của các chân ra. Đặc tính này làm đơn giản việc thiết kế máy trạng thái qua việc tạo ra một trạng thái biết trước khi mở nguồn. Vì thời điểm mở nguồn là ngẫu nhiên nên phải thỏa mãn một số điều khiện để đạt được quá trình reset hợp lệ. Trước tiên, Vcc phải tăng lên, thứ hai xung đồng hồ phải đạt mức TTL trong suốt thời gian mở nguồn như trình bày trong đồ thị. Các thanh ghi sẽ reset trong khoảng thời gian tối đa tpr. Trong chế độ hoạt động bình thường tránh kích thiết bị cho đến khi tất cả các đường vào

và đường hồi tiếp đã hoàn tất thời gian thiết lập, xung đồng hồ cũng phải có bề rộng tối thiểu.

Sơ đồ tương đương vào/ra



BÀI 3: NGÔN NGỮ ABEL

1. GIỚI THIỆU

ABEL (Advanced Boolean Equation Language) là một ngôn ngữ lập trình rất mạnh cho PLDs bao gồm cả phần lập trình, mô phỏng và tạo file cầu chì để đốt PLDs. Nó cho phép nhập những mô tả tương tự hành vi của một mạch logic. ABEL là một ngôn ngữ mô tả phần cứng (HDL) chuẩn công nghiệp được phát triển bởi công ty Data I/O cho các thiết bị PLD. Trên thực tế còn có nhiều ngôn ngữ mô tả phần cứng khác như VHDL và Verilog. ABEL đơn giản hơn VHDL là một ngôn ngữ có khả năng mô tả các hệ thống có độ phức tạp cao hơn.

ABEL có thể được dùng để mô tả hành vi của một hệ thống trong một loại các định dạng bao gồm các phương trình logic, các bảng sự thật và các sơ đồ trạng thái bằng cách dùng các câu lệnh giống như ngôn ngữ C.
Chương trình dịch ABEL cho phép mô phỏng và thực hiện các thiết kế trên PLDs như PALs, CPLDs, FPGAs.

2. CẤU TRÚC FILE NGUỒN ABEL

Một file nguồn ABEL gồm có các thành phần sau đây Phần

mở đầu: Bao gồm module, các tùy chọn và tiêu đề.

Phần mô tả: Chân linh kiện, hằng số, chân ẩn, tập hợp, trạng thái, thư viện.

Phần mô tả logic: Phương trình, bảng sự thật, sơ đồ trạng thái.

Phần véc tơ thử: các véc tơ thử.

Kết thúc.

Các từ khóa (các từ được nhậ dạng bởi ABEL như các lệnh, VD: goto, if, then, module…) không phân biệt chữ hoa và chữ thường. các tên do người dùng định nghĩa, các nhãn (từ định danh) có thể được viết bằng chữ hoa, chữ thường hoặc lẫn lộn. Một mẫu điển hình được trình bày như sau:

module *module name*
[**title** *string*]
[deviceID **device** deviceType;]
pin declarations other declarations **equations** *equations*
[**Test_Vectors**] *test vectors*
end *module name*

Ví dụ sau đây là file nguồn của một mạch cộng bán phần

```
module my_first_circuit;  title 'ee200 assignment 1'
EE200XY device 'XC4003E';
" input pins
A, B pin 3, 5;
" output pins
SUM, Carry_out pin 15, 18 istype 'com';
equations
SUM = (A & !B) # (!A & B) ;  Carry_out = A & B;
end my_first_circuit;
```

Khi sử dụng ABEL với phần mềm CAD Xilinx người dùng có thể dựa trên mẫu hướng dẫn có sẵn của ABEL và thêm vào một số từ khóa, cũng tương tự như trong phần trợ giúp của trình soạn thảo ABEL cung cấp các trợ giúp trực tuyến. Vào thực đơn TOOL > LANGUAGE ASSISTANT. Các biểu mẫu có sẵn cung cấp một mô tả của hầu hết các lệnh ABE, cú pháp, cấu trúc…Trong khi các mẫu tổng hợp trình bày các ví dụ mạch điển hình.

3. CÁC MÔ TẢ

Module: Mỗi file nguồn bắt đầu với một câu lệnh module và theo sau là một tên module (nhận dạng). các file nguồn lớn thường bao gồm nhiều module với các thành phần riêng của chúng như: Tiêu đề, phương trình, lệnh end…

Title: Là tùy chọn và có thể được dùng để nhận dạng dự án, tên tiêu đề phải được đặt giữa hai dấu móc đơn. Dòng chứa tiêu đề được chương trình dịch bỏ qua nhưng được dùng để tạo tài liệu cho chương trình.

String: Là một chuỗi các ký tự ASCII được bao bởi hai dấu móc đơn, string được dùng cho tiêu đề, các câu lệnh tùy chọn, mô tả chân linh kiện, chân ẩn và thuộc tính.

Device: Mô tả này là tùy chọn và tướng với một định danh thiết bị PLD xác định. Câu lệnh device phải được kết thúc bằng dấu chấm phẩy.

Khi dùng hệ thống CAD xilinx để dịch thiết kế tốt hơn là không nên đặt câu lệnh device trong file nguồn để giữ cho thiết kế không phụ thuộc vào linh kiện. Khi cần tạo một dự án mới trong xilinx thì cần phải xác định kiểu thiết

bị (cũng có thể thay đổi trong cửa sổ quản lý dự án bằng cách chọn nút thông tin dự án). Dạng thức như sau:

device_id device 'real_device';

Ví dụ: MY_DECODER device ‘XC4003E’ ;

Comment: Có thể được chèn vào bất kỳ nơi nào trong file nguồn và được bắt đầu bằng dấu móc kép và kết thúc bằng dấu móc kép khác hoặc khi đến cuối dòng.

Pin: Mô tả pin cho báo cho chương trình dịch tên các ký hiệu tương ứng với các chân bên ngoài của thiết bị. Dạng thức như sau:

[!]pin_id pin [pin#] [istype 'attributes'] ;

One can specify more than one pin per line:

[!]pin_id , pin_id, pin_id pin [pin#, [pin#, [pin#]]] [istype 'attributes'];

Ví dụ:

IN1, IN2, A1 pin 2, 3, 4;

OUT1 pin 9 istype 'reg';

ENABLE pin;

!Chip_select pin 12 istype 'com';

!S0..!S6 pin istype 'com';

Lập trình viên không cần xác định chân. Số chân có thể được xác định sau đó bằng cách sử dụng một file người dùng trong khi đang dịch bằng CAD xilinx. Điều này giúp kết quả thiết kế được tổng quát và linh động hơn. Ký hiệu ! biểu thị mức tích cực là mức thấp (tín hiệu bị đảo) istype là một thuộc tính tùy chọn của chân linh kiện ví dụ ‘com’ báo cho biết tín hiệu ra là kiểu tổ hợp hoặc ‘reg’ là một tín hiệu tuần tự (được ghi vào một FF). Thuộc tính này chỉ áp dụng cho các chân ra.

Node: Mô tả node có cùng dạng thức như mô tả pin. Node là các tín hiệu bên trong linh kiện chúng không được nối đến các chân ngoài.

Ví dụ:

tmp1 node [istype 'com'

Các mô tả khác cho phép người dùng định nghĩa các hằng số, tập hợp, macro, biểu thức nhằm mục đích đơn giản hóa chương trình. Như ví dụ sau đây là một mô tả hằng số

id [, id],... = expr [, expr].. ;

Ví dụ:

A = 21; C=2*7;

ADDR = [1,0,11];

LARGE = B & C;

D = [D3, D2, D1, D0];

D = [D3..D0];

Hai phương trình cuối cùng là tương đương nhau. Ký hiệu ".." được sử dụng để định nghĩa một khoảng. Ví dụ sau cùng dùng cho chú thích véc tơ, khi D được sử dụng trong một phương trình thì nó sẽ tham chiếu đến véc tơ [D3, D2, D1, D0].

4. SỐ

Giá trị số có thể được nhập vào theo 4 dạng khác nhau: Nhị phân, bát phân, thập phân và thập lục phân. Hệ thống số mặc định là thập phân. Hệ thống số được xác định bằng một trong các ký hiêu sau đây (chấp nhận chữ hoa và chữ thường). Khi không có ký hiệu xác định thì số được hiểu là thập phân.

Có thể thay đổi hệ thống số mặc định bằng chỉ dẫn "radix"

Hệ thống số	Cơ số	Ký hiệu
Nhị phân	2	^b
Bát phân	8	^o
Thập phân	10	^d(mặc định)
Thập lục phân	16	^h

Ví dụ:

Định nghĩa bởi Abel	Giá trị thập phân tương ứng
35	35
^h35	53
^b101	5

5. CÁC CHỈ DẪN

Các chỉ dẫn cho phép thao tác và xử lý các file nguồn và chúng có thể được đặt bất kỳ nơi nào khi cần trong file nguồn.

5.1 @ALTERNATE

Cú pháp

@alternate

@ALTERNATE cho phép một tập hợp luân phiên các toán tử. Việc sử dụng tập hợp toán tử luân phiên để tránh việc dùng các toán tử ABEL-HDL công (+), nhân (*) và chia (/) bởi vì chúng biểu diễn các toán tử logic AND, OR và NOT trong tập hợp luân phiên. Toán tử cũ vẫn hoạt động khi @ALTERNATE có hiệu lực. Toán tử luân phiên vẫn duy trì tác dụng cho đến khi @ALTERNATE được sử dụng hoặc kết thúc module.

@RADIX

Cú pháp

@radix *expr* ;

Expr là một biểu thức hợp lệ cho ra các giá trị 2, 8, 10, 16 để báo một cơ số mặc định mới.

Chỉ dẫn @RADIX thay đổi hệ thống số mặc định, cơ số mặc định duy trì hiệu lực cho đến khi một chỉ dẫn @radix khác được sử dụng hoặc khi kết thúc module. Lưu ý là khi một @radix mới được tạo ra thì đặc điểm của cơ số mới phải ở trong dạng thức của cơ số hiện hành.

Ví dụ: @radix 2; " thay đổi cơ số mặc định là nhị phân

@radix 1010; " chuyển từ nhị phân sang thập phân

5.2 @STANDARD

Cú pháp

@standard

Chỉ dẫn @standart phục hồi các toán tử về chuẩn ABEL-HDL. Tập hợp luân phiên được chọn với chỉ dẫn @alternate

6. TẬP HỢP

Một tập hợp là một bộ các tín hiệu hoặc hằng số được dùng để tham chiếu đến một nhóm tín hiệu thông qua một tên. Tập hợp rất tiện lợi để làm đơn giản các biểu thức logic. Một toán tử bất kỳ áp dụng cho tập hợp sẽ tác dụng đến từng phần tử trong tập hợp đó.

Tập hợp là một danh sách các hằng số hoặc tín hiệu phân cách nhau bởi dấu phẩy hoặc toán tử khoảng (..) đặt giữa các ngoặc vuông.

Ví dụ:

[D0, D1, D2, D3, D4, D5]	
[D0..D5]	" Khoảng tăng
[b6..b0]	" Khoảng giảm
[D7..D15]	

[b1, b2, a0..a3]	“ Khoảng trong một tập hợp lớn hơn
[!s7..!s0]	“ Khoảng giảm của các tín hiệu tích cực mức thấp

Tuy nhiên, không được phép viết [D0, x] ;

Trong đó x cũng là một tập hợp x = [x3..x0], do đó phải viết là [D0, x3..x0]

6.1. Chỉ số hoặc truy xuất một tập hợp

Chỉ số cho phép truy xuất các phần tử trong tập hợp, các giá trị số được dùng để biểu thị chỉ số của một tập hợp. Chỉ số tham chiếu đến vị trí bít trong tập hợp và được bắt đầu là 0 tương ứng với bít thấp nhất trong tập hợp. Sau đây là một số ví dụ.

D1 = [D15..D0] ;“ mô tả tập hợp

X2 = [X3..X0] ; “ mô tả tập hợp

X2 := D1[3..0] ; “ làm cho X2 bằng với [D3, D2, D1, D0]

X2 := D1[7..4] ; “ làm cho X2 bằng với [D7, D6, D5, D4]

Cú pháp sau đây dùng để truy xuất một phần tử trong tập hợp

OUT = (X[2] == 1) ;

Toán tử so sánh == được dùng để biến đổi phần tử (X[2]) ra giá trị bít chính là X2. Toán tử == cho ra kết quả 1 hoặc 0 là tùy thuộc vào phép so sánh true hoặc false.

6.2. Các toán tử trên tập hợp

Hầu hết các toán tử có thể áp dụng cho tập hợp và được thực hiện trên từng phần tử theo đúng quy luật đại số Boolean. Các toán tử được thực hiện theo thứ tự ưu tiên. Các toán tử có cùng mức ưu tiên được thực hiện từ trái sang phải (trừ trường hợp dùng dấu ngoặc đơn).

Ví dụ 1:

Signal = [D2, D1, D0] ; “ mô tả tập hợp các tín hiệu

Signal = [1, 0, 1] & [0, 1, 1] ; “ kết quả là signal sẽ bằng [0, 0, 1]

Ví dụ 2: [A, B]
= C & D ;

tương đương với hai câu lệnh A = C & D ; B = C & D ;

Ví dụ 3:

[A1, B1] = [D1, D2] & [C3, C2] ;

tương đương với

[A1, B1] = [D1 & C3, D2 & C2] ;

Kết quả là

A1 = D1 & C3 và B1 = D2 & C2.

Ví dụ 4:

X & [A, B, C] ;

Tương đương với

[X & A, X & B, X & C] ;

Tuy nhiên, hãy xem biểu thức sau đây

2 & [A, B, C] ;

Trước tiên, giá trị 2 được biến đổi thành nhị phân và được thêm vào các số 0 nếu cần thiết (0010), phương trình trên trở thành.

[0 & A, 1 & B, 0 & C] ;

Ví dụ 5:

A = [A2, A1, A0] ; " thiết lập mô tả

B = [B2, B1, B0] ; " thiết lập mô tả

A # B ; tương đương với [A2 # B2, A1 # B1, A0 # B0]

; !A ; tương đương với [!A2, !A1, !A0] ; Ví dụ 6:

[b3, b2, b1, b0] = 2 ; tương đương với b3 = 0, b2 = 0, b1 = 1, b0 = 0. Giá trị 2 được chuyển thành nhị phân và thêm vào các số zero.

Ví dụ 7:

Tập hợp cũng rất thuận tiện đẻ xác định các phương trình logic

Chip_sel = !A7 & A6 & A5 ;

Yêu cầu này có thể được thực hiện bằng cách dùng tập hợp. Trước tiên, định nghĩa một tập hợp hằng Addr.

Addr = [A7, A6, A5] ; " mô tả tập hợp hằng Addr

Sau đó có thể dùng phương trình sau đây để xác định địa chỉ. Chip_sel = Addr == [0, 1, 1] ; kết quả tương đương

Chip_sel = !A7 & A6 & A5 ;

Nếu A7 = 0, A6 = 1, A5 = 1, biểu thức Addr == [0, 1, 1] là true (hoặc 1) nên Chip_sel cũng là true (hoặc 1). Phương trình trên có thể viết cách khác.

Chip_sel = Addr == 3 ; " trị thập phân 3 tương đương với 011

Phương pháp trên rất thông dụng khi làm việc với các biến giá trị lớn

(ví dụ địa chỉ 16 bít)

Ví dụ 8:

Cũng như ví dụ trên, biểu thức

3 & Addr ; tương đương với

[0, 1, 1] & [A7, A6, A5] ;

[0 & A7, 1 & A6, 1 & A5] ; [0, A6, A5]. Tuy

nhiên, phát biểu sau thì lại khác.

3 & (Addr == 1) ;

Biểu thức tương đương

3 & [!A7, !A6, A5] ;

Tuy nhiên, toán tử so sánh chỉ cho kết quả 1 bít nên giá trị tính cũng được đánh giá là 1 bít và giá trị 3 bị cắt thành 1. Phương trình trên trở thành 1 & !A7 & !A6 & A5

7. TOÁN TỬ

Có 4 kiểu toán tử cơ bản: Logic, số học, so sánh và gán

7.1. Toán tử logic

Bảng sau đây trình bày các toán tử logic. Chúng được thực hiện từng bít một. Với chỉ dẫn @alternate người dùng có thể chuyển tập hợp toán tử như trình bày trong bảng

Toán tử (mặc định)	Mô tả	Toán tử luân phiên
!	NOT (bù một)	/
&	AND	*
#	OR	+
$	XOR	:+:
!$	XNOR	:*:

7.2. Toán tử số học

Bảng sau đây trình bày các toán tử số học, trong đó 4 toán tử sau cùng không áp dụng được cho tổ hợp, dấu trừ có thể có ý nghĩa khác: Là phép trừ (hoặc phép cộng số bù hai) nếu dùng với hai toán hạng và là số bù hai khi dùng với 1 toán hạng

Toán tử	Ví dụ	Mô tả
-	- D1	Số bù 2 (đảo)
-	C1 – C2	Phép trừ
+	A + B	Phép cộng

Các toán tử sau đây không dùng với tập hợp		
*	A * B	Phép nhân
/	A / B	Phép chia số nguyên không dấu
%	A % B	Modulus số dư của phép chia
<<	A << B	A được dịch trái B bít
>>	A >> B	A được dịch phải B bít

7.3 Toán tử so sánh

Bốn toán tử này cho kết quả là giá trị boolean: True (-1) hoặc falsse (0). Giá trị là -1 khi tất cả các bít đều bằng 1. Ví dụ với số 16 bít -1 tương đương với 1111 1111 1111 1111

Toán tử	Ví dụ	Mô tả
==	A == B 3= = 5 (false)	Bằng nhau
!=	A != B 3 != 5 (true)	Khác nhau
<	A < B 3 < 5 (true)	Nhỏ hơn
<=	A <= B 3 <= 5 (true)	Nhỏ hơn hoặc bằng
>	A > B -1 > 5 (true)	Lớn hơn
>=	A >= B !0 >= 0 (true)	Lớn hơn hoặc bằng

Toán tử so sánh là không dấu. Chú ý: !0 là bổ túc của 0 hoặc 1111 1111 (dữ liệu 8 bít) có giá trị là 255. Vì vậy 10 > 9 là true và biểu thức -1 > 5 cũng là true

Giá trị -1 hoặc 0 có thể được thay thế tùy thuộc vào kết quả logic

Ví dụ:

A = B !$ (C==D) ;

A bằng B nếu C = D (true hoặc 1111…B XNOR 1 bằng B). Ngược lại A sẽ bằng bổ túc của B (nếu C khác B (false hoặc 0).

7.4. Toán tử gán

Các toán tử này được dùng trong phương trình để gán giá trị một biểu thức cho tín hiệu ra. Có hai kiểu toán tử gán: Tổ hợp và thanh ghi. Trong một

toán tử tổ hợp phép gán xảy ra tức thời không có trì hoãn. Trong khi phép gán kiểu thanh ghi xảy ra tại xung đồng hồ kế tiếp kết hợp với ngõ ra.

Ví dụ một FF có thể được định nghĩa bằng các dòng lệnh sau đây. Q1

pin istype ‘reg’ ; Q1 := D ;

Dòng lệnh đầu tiên định nghĩa FF Q1 bằng mô tả istype ‘reg’. (registered output). Dòng lệnh thứ hai cho biết rằng ngõ ra của FF nhận giá trị của ngõ vào D tại thời điểm chuyển tiếp xung đồng hồ kế tiếp.

Toán tử	Mô tả
=	Gán tổ hợp
:=	Gán thanh ghi

7.5 Thứ tự ưu tiên

Mức ưu tiên của mỗi toán tử được trình bày trong bảng dưới đây, với 1 là mức ưu tiên cao nhất và 4 là thấp nhất. Những toán tử có cùng mức ưu tiên thì được thực hiện từ trái sang phải.

Mức ưu tiên	Toán tử	Mô tả
1	-	Phủ định (bù 2)
1	!	Đảo
2	&	AND
2	<<	Dịch trái
2	>>	Dịch phải
2	*	Nhân
2	/	Chia không dấu
2	%	Modulus
3	+	Cộng
3	-	Trừ
3	#	OR
3	$	XOR
3	!$	XNOR
4	==	Bằng nhau

4	!=	Khác nhau
4	<	Nhỏ hơn
4	<=	Nhỏ hơn hoặc bằng
4	>	Lớn hơn
4	>=	Lớn hơn hoặc bằng

8. MÔ TẢ LOGIC

Một thiết kế logic có thể được mô tả theo các phương pháp sau đây:

- Phương trình.
- Bảng sự thật
- Sơ đồ trạng thái

8.1. Phương trình

Mô tả phương trình logic được bắt đầu bằng từ khóa equations. Phương trình xác định các biểu thức logic bằng cách dùng các toán tử như đã trình bày ở phần trên hoặc câu lệnh "When – Then – Else".

Câu lệnh "When – Then – Else" được dùng trong phương trình để mô tả một chức năng logic (Lưu ý: "If – Then –Else" được dùng trong phần sơ đồ trạng thái để mô tả quá trình chuyển đổi trạng thái).

Dạng thức của câu lệnh "When – Then – Else" như sau: ***When***
điều kiện ***Then*** phần tử = Biểu thức ;
Else phương trình ; Hoặc: ***When***
điều kiện ***Then*** phương trình ;

Ví dụ:

```
SUM = (A & !B) # (!A & B) ;
A0 := EN & !D1 & D3 & !D7;
WHEN (A == B) THEN D1_out = A1;
ELSE WHEN (A == C) THEN D1_out = A0;
WHEN (A>B) THEN { X1 :=D1; X2 :=D2; }
```

Cũng có thể dùng dấu ngoặc móc để nhóm các đoạn lại với nhau thành một khối. Đoạn văn bản trong khối có thể được viết trên một dòng hoặc rải trên nhiều dòng. Các khối được dùng trong phương trình, sơ đồ trạng thái và các chỉ dẫn.

8.2. bảng sự thật

Từ khóa là truth_table và cú pháp là

```
TRUTH_TABLE ( in_ids -> out_ids )
inputs -> outputs ;
```

Hoặc

```
TRUTH_TABLE ( in_ids :> reg_ids )
inputs :> reg_outs ;
```

Hoặc

```
TRUTH_TABLE
( in_ids :> reg_ids -> out_ids )
Inputs :> reg_outs -> outputs ;
```

Ký hiệu -> là ngõ ra tổ hợp và :> là ngõ ra kiểu thanh ghi. Dòng đầu tiên của bảng sự thật (trong cặp ngoặc đơn) định nghĩa các tín hiệu vào và ra. Các dòng tiếp theo cho biết giá trị của tín hiệu vào và ra, mỗi dòng phải kết thúc bằng dấu chấm phẩy. các ngõ vào và ra có thể là tín hiệu đơn hoặc tập hợp. Khi là tập hợp thì các tín hiệu phải được bao bằng các ngoặc vuông và cách nhau bằng các dấu phẩy. Giá trị không xác đinh được biểu diễn bằng ký hiệu ".X.".

Ví dụ 1: mạch cộng bán phần

```
TRUTH_TABLE ( [ A, B] -> [Sum, Carry_out] )
[ 0, 0 ] -> [0, 0 ] ;
[ 0, 1 ] -> [1, 0 ] ;
[ 1, 0 ] -> [1, 0 ] ;
[ 1, 1 ] -> [1, 1 ] ;
```

Tuy nhiên, nếu định nghĩa một tập hợp IN = [A, B] hoặc OUT = [sum, carry_out] ; lúc này bảng sự thật sẽ trở nên đơn giản hơn

```
TRUTH_TABLE (IN -> OUT )
0 -> 0;
1 -> 2;
2 -> 2;
3 -> 3;
```

Ví dụ 2: Mạch EXOR có hai ngõ vào và một ngõ cho phép, ví dụ này cho thấy cách dùng các trị không xác định

```
TRUTH_TABLE ([EN, A, B] -> OUT )
[ 0, .X.,.X.] -> .X. ;
[ 1, 0 , 0 ] -> 0 ;
[ 1, 0 , 1 ] -> 1 ;
```

[1, 1 , 0] -> 1 ;

[1, 1 , 1] -> 0 ;

Bảng sự thật cũng có thể được dùng để định nghĩa hệ tuần tự ví dụ một mạch đếm lên 3 bít từ 000, 001 đến 111 và trở về 000. Gọi QA, QB và QC là ngõ ra của các FF và ngõ ra OUT khi bộ đếm đạt đến giá trị 111, bộ đếm bị xóa về 000 khi tín hiệu reset ở mức cao.

MODULE CNT3;

CLOCK pin; " input signal

RESET . pin; " input signal

OUT pin istype 'com'; " output signal (combinational)

QC,QB,QA pin istype 'reg'; " output signal (registered)

[QC,QB,QA].CLK = CLOCK; "FF clocked on the CLOCK input

[QC,QB,QA].AR = RESET; "asynchronous reset by RESET

TRUTH_TABLE) [QC, QB, QA] :> [QC,QB,QA] -> OUT)

[0 0 0] :> [0 0 1] -> 0;

[0 0 1] :> [0 1 0] -> 0;

[0 1 0] :> [0 1 1] -> 0;

[0 1 1] :> [1 0 0] -> 0;

[1 0 0] :> [1 0 1] -> 0;

[1 0 1] :> [1 1 0] -> 0;

[1 1 0] :> [1 1 1] -> 0;

[1 1 1] :> [0 0 0] -> 1;

END CNT3;

8.3. Mô tả trạng thái

Phần sơ đồ trạng thái bao gồm việc mô tả trạng thái một thiết kế logic. Trong phần này dùng cú pháp state_diagram với các câu lệnh "If – Then – Else", "Goto", "case" và "With". Thông thường tên trạng thái được đặt trong phần mô tả để giúp việc đọc chương trình dễ hơn.

Cú pháp: state_id [, state_id ...] STATE ;

Ví dụ: SREG = [Q1, Q2] ; kết hợp tên trạng thái SREG với các trạng thái được định nghĩa là Q1 và Q2.

Cú pháp phần sơ đồ trạng thái như sau:

State_diagram *state_reg*

STATE *state_value* : [equation;] [equation;] trans_stmt ; ...

Từ khóa state_diagram báo cho biết vị trí bắt đầu phần mô tả máy trạng thái. Từ khóa STATE và các câu lệnh theo sau trình bày một trạng thái của sơ đồ trạng thái trong đó gồm có giá trị và tên trạng thái, câu lệnh chuyển trạng thái và một phương trình ngõ ra tùy chọn. Trong cú pháp trình bày ở trên:

- State_reg: Là một định danh định nghĩa các ký hiệu xác định trạng thái của máy. Đây có thể là một thanh ghi trạng thái đã được mô tả trước trong phần mô tả.
- State_value: Có thể là một biểu thức, một giá trị hoặc tên trạng thái hiện hành.
- Equation: Phương trình định nghĩa các ngõ ra của máy trạng thái.
- Trans_stmt: các câu lệnh "If – Then – Else", "Case" hoặc "Goto"

dùng để xác định các trạng thái kế tiếp, được theo sau bởi các phương trình chuyển tiếp WITH tùy chọn.

"If – Then – Else"

Câu lệnh này được dùng trong phần sơ đồ trạng thái để mô tả trạng thái kế tiếp và xác định các điều kiện chuyển tiếp Cú pháp:

IF expression THEN state_exp

[ELSE state_exp] ;

State_exp có thể là một biểu thức logic hoặc tên trạng thái. Lưu ý là câu lệnh If – Then – Else chỉ có thể được dùng trong phần sơ đồ trạng thái (dùng "When – If – Then" để mô tả các chức năng logic). Mệnh đề ELSE là tùy chọn. Câu lệnh If – Then – Else có thể được kế tiếp bởi các câu lệnh Goto, Case và With.

Ví dụ:

Trong phần mô tả, trước tiên chúng ta định nghĩa các thanh ghi trạng thái

SREG = [Q1, Q0]; "definition of state registers

S0 = [0, 0]; S1 = [1, 1];

State_diagram

SREG State S0:

OUT1 = 1; if A then

S1 else

S0; state S1: OUT2 =1;

if A then S0 else

S1;

Các câu lệnh If – Then – Else có thể lồng nhau như ví dụ sau đây. Giả sử rằng các trạng thái và các thanh ghi đã được định nghĩa trong phần mô tả.

State_diagram MAK

State INIT: if RESET then INIT else LOOK;
State LOOK: if REST than INIT
else if (X == LASTX) then OK
else LOOK;
state OK: if RESET than
INIT else if Y then OK else if
(X == LASTX) then OK
else LOOK; state
OK: goto INIT;

Câu lênh "With"

Cú pháp:

trans_stmt state_exp WITH equation
[equation] ... ;

trans_stmt có thể là câu lệnh "If – then – else", "Goto" hoặc "Case", **state_exp** là trạng thái kế tiếp và **equation** là một phương trình ra của máy.

Câu lệnh này có thể được dùng với "If – then – else", "Goto" hoặc "Case" tại vị trí của một biểu thức trạng thái đơn. Câu lệnh "With" cho phép các phương trình ra được ghi vào các số hạng của sự chuyển tiếp. Ví dụ 1:

if X#Y==1 then S1 with Z=1 else S2;

Trong ví dụ trên ngõ ra Z sẽ được xác định ngay khi biểu thức sau câu lệnh **If** có giá trị là 1 (hoặc true). Biểu thức sau từ khóa WITH có thể là một phương trình và được định trị ngay khi điều kiện If là true như trong ví dụ 2.

Ví dụ 2: if X&!Y then S3 with Z=X#Y else S2
with Z=Y;

Câu lệnh "With" cũng thường được dùng để mô tả hành vi ra của các ngõ ra kiểu thanh ghi. Do các ngõ ra kiểu thanh ghi chậm sau một chu kỳ

đồng hồ. Nó cũng cho phép thể hiện để xác định ngõ ra thanh ghi có một giá trị xác định sau một chuyển tiếp đặc biệt như ở ví dụ [1]

Ví dụ 3[1]

```
state S1:
if RST then S2 with { OUT1 := 1; Error-Adrs := ADDRESS; }
else if (ADDRESS <= ^hC101)
then     S4
else S1;
```

Lưu ý là có thể dùng cặp ngoặc nghoéo để nhóm chung các ngõ ra và các phương trình như ví dụ trên. Ví dụ 3 state S1: if (A & B) then S2 with TK = 1 else S0 with TK = 0 ;

Cần phải quan tâm đến thời gian khi dùng câu lênh "With" đối với các ngõ ra tổ hợp hoặc không đồng bộ (như trong hệ Mealy). Hệ Mealy thay đổi các ngõ ra của nó ngay khi các ngõ vào thay đổi. Điều này có thể làm cho ngõ ra thay đổi quá nhanh và sẽ gây ra nhiễu logic.Các ngõ ra của Mealy phải ổn định tại thời điểm cuối của trạng thái (có nghĩa là ngay trước khi xung đồng hồ thay đổi). Về điểm này thì một ngõ ra Moore sẽ ít gây lỗi hơn.

Câu lệnh "Case"

Cú pháp:

```
CASE expression : state_exp;
[ expression : state_exp; ]
:
ENDCASE ;
```

expression là một biểu thức ABEL bất kỳ và state_exp là một biểu thức cho biết trạng thái kế tiếp (có thể được theo sau bởi câu lệnh "With")

Ví dụ:

```
State S0:
case ( A == 0) : S1;
( A == 1) : S0;
endcase;
```

Câu lệnh case được dùng để liệt kê một chuỗi các điều kiện chuyển tiếp loại trừ nhau và các trạng thái kế tiếp tương ứng. Các điều kiện của câu lệnh case phải loại trừ lẫn nhau (khônmg thể có hai điều kiện chuyển tiếp cùng bằng true) hoặc kết quả trạng thái kế tiếp không thể đoán được.

8.4. Dấu chấm (.)

Dấu chấm có thể được dùng để mô tả hành vi mạch điện một cách chi tiết hơn. Việc mở rộng tín hiệu rất thuận lợi và cung cấp một phương pháp tham chiếu đến tín hiệu bên trong và các chân nội kết hợp với một tín hiệu sơ cấp. Cú pháp signal_name.ext

Một số dấu chấm mở rộng được cho trong bảng sau. Các mở rộng không phân biệt chữ hoa chữ thường, một số dấu chấm mở rộng có công dụng chung (cũng còn được gọi là độc lập cấu trúc hoặc điểm – điểm) và có thể được dùng với nhiều kiểu cấu trúc thiết bị. Các dấu chấm mở rộng khác được dùng cho các cấu trúc thiết bị xác định và được gọi là phụ thuộc cấu trúc hoặc dấu chấm mở rộng chi tiết.

Dấu chấm mở rộng	**Mô tả**
.ACLR	Reset không đồng bộ
.ASET	Preset không đồng bộ
.CLK	Ngõ vào đồng hồ FF tác động cạnh
.CLR	Reset đồng bộ
.COM	Ngõ vào dữ liệu hồi tiếp kiểu tổ hợp từ FF
.FG	Hồi tiếp thanh ghi
.OE	Cho phép ra
.PIN	Hồi tiếp chân thiết bị
.SET	Preset đồng bộ
Các mở rộng xác định (phụ thuộc cấu trúc)	
.D	Ngõ vào dử liệu của D – FF
.J	Ngõ vào J của JK – FF
.K	Ngõ vào K của JK – FF
.S	Ngõ vào S của SR – FF
.R	Ngõ vào R của SR – FF
.T	Ngõ vào T của T – FF
.Q	Hồi tiếp thanh ghi
.PR	Preset thanh ghi

.RE	Reset thanh ghi
.AP	Preset thanh ghi không đồng bộ
.AR	Reset thanh ghi không đồng bộ
.SP	Preset thanh ghi đồng bộ
.SR	Reset thanh ghi đồng bộ

Ví dụ 1 :

[S6..S0].OE = ACTIVE;

Truy cập đến tín hiệu điều khiển 3 trạng thái của bộ đệm ra của các tín hiệu S6…S0. Khi ACTIVE ở mức cao, các tín hiệu được cho phép.
Ngược lại, các ngõ ra sẽ ở trạng thái Z cao.

Ví dụ 2 :

Q.AR = reset;

[Z.ar, Q.ar] = reset;

Ngõ ra của các thanh ghi (FF) trở về 0 khi reset ở mức cao

8.5. Các véc tơ thử

Các véc tơ thử là tùy chọn nhằm cung cấp một phương pháp để kiểm tra hoạt động của máy trạng thái. Các véc tơ xác định hoạt động logic mong muốn của một thiết bị logic bằng cách cho các ngõ ra là hàm của các ngõ vào một cách rõ ràng.

Cú pháp :

Test_vectors [note]

(input [, input].. -> output [, output] ..) [invalues -> outvalues ;]

:

:

Ví dụ :

Test_vectors

([A, B] -> [Sum, Carry]) [0, 0] -> [0, 0];

[0, 1] -> [1, 0];

[1, 0] -> [1, 0];

[1, 1] -> [1, 1];

Cũng có thể xác định các giá trị của tập hợp với các hằng số như chỉ ra sau đây :

([A, B] -> [Sum, Carry])

0 -> 0;

1 -> 2;

2 -> 2;

3 -> 3;

Trị bất định (.X.), ngõ vào đồng hồ (.C.) được dùng như các hằng số, xem ví dụ sau test_vectors

([CLK, RESET, A, B] -> [Y0, Y1, Y3])

[.X., 1, .X.,.X.]->[S0, 0, 0];

[.C., 0, 0, 1] -> [S0, 0, 0];

[.C., 1, 1, 0] -> [S0, 0, 1];

8.6. Các câu lệnh thuộc tính

ABEL cho phép cung cấp các câu lệnh xác định thiết bị bằng cách dùng lệnh thuộc tính. Lệnh này được bỏ qua trong quá trình dịch. Thuộc tính của các thiết bị CPLD gồm có:

- Độ dốc
- Tối ưu logic
- Sắp xếp logic
- Thiết lập công suất
- Giá trị nạp trước

8.7. Linh tinh

Mô tả tích cực mức thấp

Các tín hiệu tích cực mức thấp được định nghĩa với toán tử "!" như sau:

!OUT pin istype 'com' ;

Khi tín hiệu được sử dụng trong sự mô tả thiết kế tiếp theo sau, nó sẽ được tự động đảo lại như trong ví dụ sau đây module EXAMPLE

A, B pin ;
!OUT pin istype 'com';
equations
OUT = A & !B # !A & B ; end

Trong ví dụ trên, tín hiệu OUT là kết quả EXOR giữa A với B có nghĩa là OUT sẽ bằng 1 (High hoặc ON) khi chỉ một trong các ngõ vào bằng 1. Mặt khác OUT sẽ bằng 0. Tuy nhiên, vì chân ra được định nghĩa là !OUT có nghĩa là tín hiệu tích cực mức thấp. Điều này cho thấy rằng chân ra sẽ bằng 0 (tích cực mức thấp hoặc ON) chỉ khi một trong hai ngõ vào bằng 1. Cũng có thể

nhận được cùng kết quả bằng cách đảo tín hiệu trong các phương trình hoặc bằng cách mô tả chân ra là OUT.

Ví dụ:

```
module EXAMPLE
A, B pin ;
OUT pin istype 'com';
equations
!OUT = A & !B # !A & B ; end
```

Tích cực mức thấp cũng có thể áp dụng cho tổ hợp, như ví dụ sau đây định nghĩa các tổ hợp A, B và C

```
A = [A2,A1,A0]; "set declaration
B = [B2,B1.B0]; "set declaration
X = [X2,X1.X0]; "set declaration
!X = A & !B # !A & B;
```

Phương trình cuối cùng tương đương với cách viết sau

```
!X0 = A0 & !B0 # !A0 & B0;
!X1 = A1 & !B1 # !A1 & B1;
!X2 = A2 & !B2 # !A2 & B2;
```

9. CHƯƠNG TRÌNH MẪU

Chương trình 1:

```
1.  module Alarm_Circuit
2.  title ’Alarm Circuit Example
3.  J. Wakerly, Micro Systems Engineering’
4.  ALARMCKT device ’P16V8C’;
5.  " Input pins
6.  PANIC, ENABLEA, EXITING pin 1, 2, 3;
7.  WINDOW, DOOR, GARAGE pin 4, 5, 6;
8.  " Output pins
9.  ALARM pin 11 istype ’com’;
10. " Constant definition
11. X = .X.;
12. " Intermediate equation
13. SECURE = WINDOW & DOOR & GARAGE;
14. equations
15. ALARM = PANIC # ENABLEA & !EXITING &
!(WINDOW & DOOR & GARAGE);
```

16. test_vectors
17. ([PANIC,ENABLEA,EXITING,WINDOW,DOOR,GARAGE] ->
[ALARM])
18. [1, .X., .X., .X., .X., .X.] -> [1];
19. [0, 0, .X., .X., .X., .X.] -> [0];
20. [0, 1, 1, .X., .X., .X.] -> [0];
21. [0, 1, 0, 0, .X., .X.] -> [1];
22. [0, 1, 0, .X., 0, .X.] -> [1];
23. [0, 1, 0, .X., .X., 0] -> [1];
24. [0, 1, 0, 1, 1, 1] -> [0];
25. end Alarm_Circuit

Giải thích chương trình:

-Dòng 1 khai báo module tên là Alarm_Circuit

-Dòng 2 và 3 khai báo tiêu đề của module

-Dòng 4 khai báo kiểu thiết bị là P16V8C (chế độ tổ hợp)

-Dòng 5 là chú thích cho biết bắt đầu khai báo các chân vào

-Dòng 6 và 7 khai báo các chân vào

-Dòng 8 là chú thích cho biết bắt đầu khai báo các chân ra

-Dòng 9 cho biết tín hiệu ra kiểu tổ hợp ALARM được kết nối với chân 11

-Dòng 10 chú thích định nghĩa hằng số

-Dòng 11 định nghĩa X là giá trị không xác định

-Dòng 12 và 13 chú thích và định nghĩa phương trình trung gian SECURE

-Dòng 14 từ khóa equations cho biết bắt đầu phần phương trình

-Dòng 15 khai báo phương trình của tín hiệu ALARM

-Dòng 16 từ khóa test_vectors báo bắt đầu phần véc tơ thử kiểm tra chức năng hoạt động của mạch

-Dòng 17 khai báo các tín hiệu vào và ra cần kiểm tra

-Dòng 18 đến 24 là các véc tơ thử

-Dòng 25 kết thúc module Alarm_Circuit bằng từ khóa end

Chương trình 2: cách dùng các trị bất định module DontCare title 'Dont Care Examples'

@DCSET

```
" Input and output pins
N3..N0, A, B pin;
F, Y pin istype 'com'; NUM = [N3..N0];
X = .X.;
truth_table (NUM->F)
0->0;
1->1;
2->1;
3->1;
4->0;
5->1;
6->0;
7->1;
8->0;    9-
>0;
truth_table ([A,B]->Y) [0,0]->0;
[0,1]->X;
[1,0]->X;
[1,1]->1;
end
DontCare
```

Từ chương trình trên cho thấy có thể xác định các tổ hợp bất định một cách rõ ràng như ở bảng sự thật thứ hai.

Chương trình 3: Dùng phương pháp máy trạng thái thiết kế mạch đếm lên đồng bộ 16 dùng GAL 16V8 có ngõ CL đồng bộ tích cực mức thấp như sơ đồ mạch sau

Từ sơ đồ trạng thái kèm theo và lưu ý là các ngỏ ra Q3…Q0 cũng chính là các biến trạng thái.

```
1. module Count4
2. title ‘bộ đếm lên 16’
3. U4 device ‘P16V8R’ ;
4. Ck, cl  pin      1, 2 ;
5. Q3, Q2, Q1, Q0          pin 19, 18, 17, 16 ;
6. Count = [Q3,Q2,Q1,Q0] ;
7. c, h, l = .C., 1, 0 ;
8. “định nghĩa các trạng thái
```

9. s0 = ^b0000 ; s1 = ^b0001 ; s2 = ^b0010 ;
10. s3 = ^b0011 ; s4 = ^b0100 ; s5 = ^b0101 ;
12. s9 = ^b1001 ; s10 = ^b1010 ; s11 = ^b1011 ;
13. s12 = ^b1100 ; s13 = ^b1101 ; s14 = ^b1110 ;
14. s15 = ^b1111 ;
15. state_diagram [Q3,Q2,Q1,Q0]
17. state s1 : if cl then s2 else s0 ;
18. state s2 : if cl then s3 else s0 ;
19. state s3 : if cl then s4 else s0 ;
21. state s5 : if cl then s6 else s0 ;
22. state s6 : if cl then s7 else s0 ;
23. state s7 : if cl then s8 else s0 ;
24. state s8 : if cl then s9 else s0 ;
25. state s9 : if cl then s10 else s0 ;
26. state s10 : if cl then s11 else s0 ; 27. state s11 : if cl then s12 else s0 ; 28. state s12 : if cl then s13 else s0 ; 29. state s13 : if cl then s14 else s0 ;
30. state s14 : if cl then s15 else s0 ;
31. state s15 : goto s0 ;
32. test_vectors ([ck,cl] -> count)
33. [c,h] -> ^b0000 ; 34. [c,h] -> ^b0001 ; 35. [c,h] -> ^b0010 ; 36. [c,h] -> ^b0011 ;
37. [c,h] -> ^b0100 ;
38. [c,l] -> ^b0000 ;
39. end Count

Giải thích chương trình

Từ dòng 9 đến 14 định nghĩa các trạng thái s0 đến s15 bằng toán tử gán "=", trong chương trình này dùng mã nhị phân "^b"

Dòng 15 định nghĩa 4 biến trạng thái Q3Q2Q1Q0 bằng từ khóa state_diagram

Từ dòng 16 đến 30 là các câu lệnh if – then mô tả quá trình chuyển trạng thái

Dòng 31 câu lệnh goto đưa trạng thái bộ đếm về 0 vô điều kiện

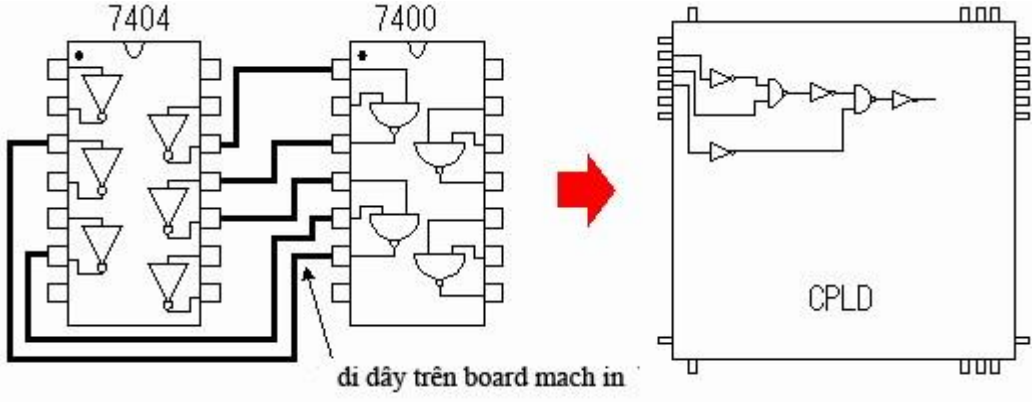
Dòng 32 bắt đầu phần véc tơ thử bằng từ khóa test_vectors Từ dòng 33 đến 38 là các véc tơ thử Dòng 39 từ khóa end kết thúc module.

BÀI 4: HỌ CPLD

1. GIỚI THIỆU CHUNG

Trong một vi mạch logic đơn lẻ được Ví dụ trong trường hợp của IC 7400 gồm 4 cổng NAND được thiết kế ở bên trong. Trong trường hợp IC 7406, 6 cổng đảo được thiết kế ở bên trong. Đây là những IC riêng biệt nên để sử dụng cần phải kết nối chúng với nhau trên board mạch và vì thế chúng chiếm nhiều khoảng trống trên board. Trong trường hợp CPLD, nó kết nối các khối chức năng logic này với nhau trong cùng một IC (sơ đồ kết nối này được xác định bởi phần mềm) nên không chiếm nhiều khoảng trống và nhờ thế mạch thiết kế sẽ gọn hơn.. Ngoài ra CPLD còn có các ưu điểm khác hơn hẳn so với khi dùng các họ vi mạch truyề thống như :

- Công suất tiêu thụ thấp
- Độ tin cậy cao
- Kich thước board mạch nhỏ gọn, sơ đồ nối dây bên ngoài rất đơn giản dẫn đến thi công đơn giản
- Quá trình thiết kế dể dàng nhờ có sự trợ giúp của các công cụ phần mềm phát triển hệ thống
- Giá thành rẻ



Tuy nhiên, Khả năng của CPLD cũng có giới hạn, đó là sự giới hạn về số chân vì vậy đừng trông mong quá nhiều vào CPLD. Mặc khác vì được chế tạo theo dạng flash nên CPLD có thể tái lập trình khoảng 10.000 lần đó là con số mà nhà sản xuất cung cấp

2. VI MẠCH ispLSI 1016

2.1. Đặc tính

❋ *Vi mạch logic mật độ cao*

- Khả năng kết nối tốc độ cao
- Gồm 2000 cổng PLD
- 32 chân I/O, 4 ngõ vào định sẵn
- 96 thanh ghi
- Cổng vào dải rộng cho chức năng đếm tốc độ cao, máy trạng thái, giãi mã địa chỉ…
- Khả năng bảo mật chống sao chép

❋ *Công nghệ E^2CMOS*

- Tần số tốI đa fmax = 110 KHz
- Tần số fmax = 60 MHz đốI với thiết bị công nghiệp và quân sự
- Trì hoãn truyền tpd = 3 nS
- Ngõ vào và ra tương thích TTL
- Cho phép lập trình lại và xóa bằng điện
- Công nghệ bộ nhớ không mất dữ liệu E^2CMOS
- Được kiểm tra 100%

❋ *Lập trình trong hệ thống*

- Khả năng lập trình 5V ngay trong hệ thống
- Tăng lĩnh vực sản xuất, giảm thời gian thương mại hóa và cải thiện chất lượng sản phẩm
- Tìm lỗi nhanh

❋ *Kết hợp dể dàng giữa việc ứng dụng các hệ PLD tốc độ cao với mật độ và tính linh hoạt của FPGA*

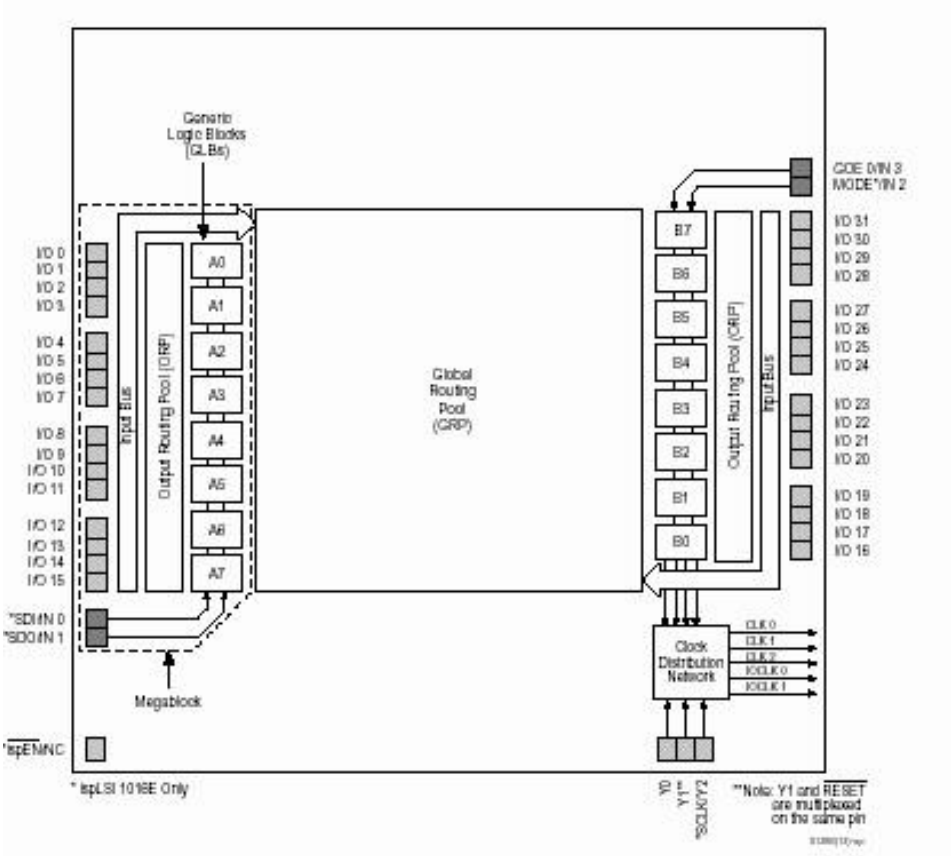
- Thiết bị lập trình toàn bộ có thể kết hợp giữa logic tổ hợp với thiết kế có cấu trúc

- Có 3 ngõ vào đồng hồ định sẵn
- Xung đồng hồ đồng bộ và không đồng bộ
- Vị trí chân linh hoạt

2.2. Mô tả

IspLSI 1016 là vi mạch số lập trình mật độ cao bao gồm 96 thanh ghi, 32 ngõ I/O đa năng, 4 ngõ vào định sẵn, 3 ngõ vào đồng hồ định sẵn và một khối kết nối toàn cục (global routing pool). IspLSI 1016 cho phép lập trình ngay trên hệ thống với mức điện áp 5V và cũng là thiết bị logic đầu tiên có bộ nhớ lập trình lại được mà không mất dữ liệu.

Khối logic cơ sở trong ispLSI 1016 là khối logic tông quát GLB (generic logic block), các khối này được đặt tên là A0, A1,…,B7 (hình 4.1), có tất cả 16 GLB : Mỗi GLB có 18 ngỏ vào, một mảng AND/OR/XOR lập trình được và 4 ngõ ra. Các ngỏ ra có thể được cấu hình theo kiểu tổ hợp hoặc thanh ghi. Tín hiệu đưa vào GLB lấy từ các GRB hoặc từ các ngõ vào định sẵn. Tất cả ngõ ra của GLB được hồi tiếp về GRB để có thể kết nối với ngõ vào của các GLB khác trong thiết bị.



Hình 4.1: Sơ đồ khối Isp LSI 1016

Các thiết bị cũng có 32 đơn vị I/O được nối trực tiếp đến các chân

I/O bên ngoài. Mỗi đơn vị I/O có thể được lập trình riêng rẽ theo các kiểu : Ngõ vào tổ hợp, ngõ vào thanh ghi, ngõ vào chốt, ngõ ra hoặc chân I/O hai chiều 3 trạng thái. Các mức tín hiệu tương thích TTL, dòng nguồn ngõ ra 4 mA và dòng nhận ngõ ra là 8 mA. Mỗi ngõ ra cho phép lập trình độc lập với tốc độ tăng điện áp ra nhanh hoặc chậm nhằm giảm thiểu nhiễu phát sinh khi ngõ ra chuyển trạng thái.

Tám khối GLB, 16 đơn vị I/O, hai ngõ vào định sẵn và một ORP được nối với nhau tạo thành một khối mega (hình 4.1), các ngõ ra của 8 GLB được nối đến một tập hợp gồm 16 đơn vị I/O đa năng bởi ORP. Mỗi vi mạch ispLSI 1016 gồm có hai khối mega

Khối GRP có các ngõ ra cũng như các ngõ vào được cung cấp từ các GLP, các ngõ vào của đơn vị I/O hai chiều. Tất cả các tín hiệu này có thể dùng làm tín hiệu vào cho các GLP

Xung đồng hồ trong họ ispLSI 1016 được chọn bởi khối phân phối xung. Ba chân đồng hồ định sẵn Y0, Y1 và Y2 được đưa vào khối này và năm ngõ ra đồng hồ CLK 0, CLK 1, CLK 2, IOCLK 0 và IOCLK 1 được cung cấp đến các GLB và các đơn vị I/O. Khối phân phối xung cũng có thể được điều khiển từ một GLB đồng hồ đặc biệt (B0 trong ispLSI 1016). Chức năng GLB này cho phép người dùng tạo ra xung đồng hồ bên trong qua việc kết hợp các tín hiệu trong thiết bị

2.3. Thông số giới hạn

Điện áp nuôi Vcc……..-0,5 đến +7 V

Điện áp ngỏ vào ……...-2,5 đến Vcc + 1V

Điện áp ra ở trạng thái tắt……-2,5 đến Vcc + 1V

Nhiệt độ lưu trử…….-65 đến + 150^0C

Nhiệt độ vỏ khi cấp nguồn……-65 đến + 125^0C

Nhiệt độ mối nối tối đa (TJ) khi cấp nguồn..150^0C

2.4. Điều kiện hoạt động DC

Ký hiệu	Thông số		Min	Max	Đơn vị
Vcc	Điện áp nuôi	Thương mại TA = 0^0C	4,75	5,25	V

		Công nghiệp TA = -40^{0}C	4,5	5,5	V
VIL	Điện áp vào mức thấp		0	0,8	V
VIH	Điện áp vào mức cao		2	Vcc + 1	V

2.5. Điện dung (TA = 25^{0}C, f = 1 MHz)

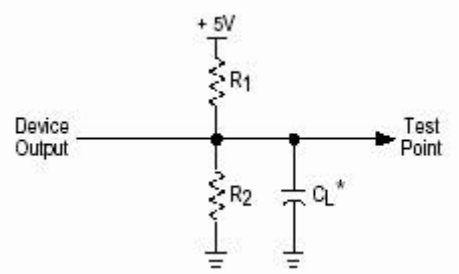
Ký hiệu	**Thông số**	**Điển hình**	**Đơn vị**	**Điều kiện thử**
C1	Ngõ vào định sẵn, I/O, Y1, Y2, Y3, (thương mại, công	8	PF	Vcc = 5V, VPIN = 2V
C2	nghiNgõ đồngp) hồ Y0	12	PF	Vcc = 5V, VPIN = 2V

2.6. Đặc tính lưu trữ dữ liệu

Thông số	**Min**	**Max**	**Đơn vị**
Thời gian lưu trữ	20	-	Năm
Số lần lập trình lại	10000	-	Lần

2.7. Điều kiện thử chuyển mạch

Mức xung vào	**GND đến 3V**	
Thời gian tăng và giảm ngõ vào 10% đến 90%	-125	□ 2ns
	-100, -80	□ 3ns
Mức tham chiếu thời gian vào	1,5V	
Mức tham chiếu thời gian ra	1,5V	
Tải ngõ ra	Hình 4.2	



Hình 4.2. Thử tải

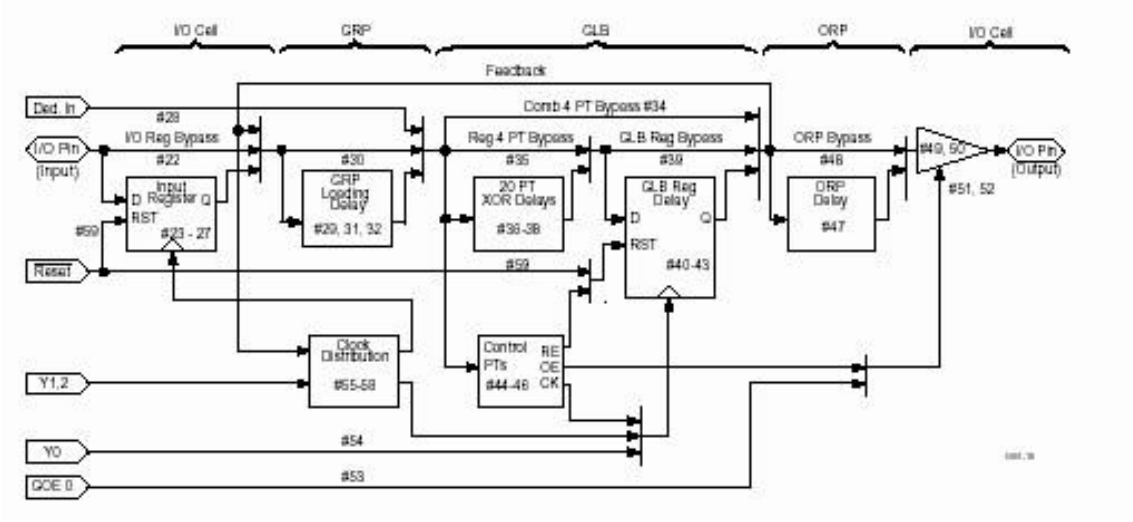
Điều kiện tải ngõ ra

Điều kiện thử		R1	R2	C_L
A		470□	390□	35pF
B	Tích cực mức cao	□	390□	35pF
	Tích cực mức thấp	470□	390□	35pF
C	Tích cực H đến Z tại V_{0H} - 0,5V	□	390□	5pF
	Tích cực L đến Z tại V_{0L} + 0,5V	470□	390□	5pF

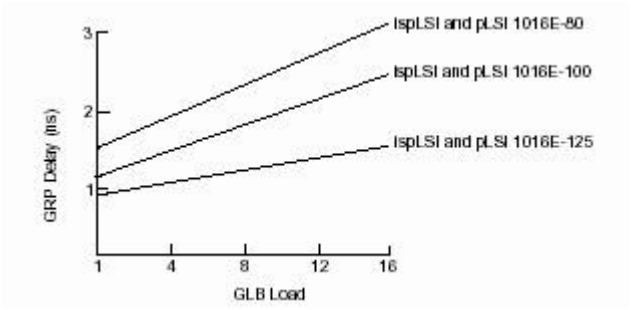
2.8. Đặc tính điện DC

Ký hiệu	Thông số	Điều kiện	Min	Typ	Max	Đơn vị
VOL	Điện áp ra mức thấp	IOL = 8 mA	-	-	0,4	V
VOH	Điện áp ra mức cao	IOH = -4 mA	2,4	-	-	V
IIL	Dòng rỉ mức thấp ngõ vào hoặc I/O	0≤VIN≤VIL (max)	-	-	-10	µA
IIH	Dòng rỉ mức cao ngõ vào hoặc I/O	3,5 V≤VIN≤VCC	-	-	10	µA
IIL-isp	Dòng rỉ ngõ vào ispEN	0≤VIN≤VIL	-	-	-150	µA
IIL-PU	Dòng kéo lên I/O	0≤VIN≤VIL	-	-	-150	µA
IOS	Dòng ngắn mạch ngõ ra	VCC = 5 V, VOUT =	-	-	-200	mA
ICC	Dòng nuôi	VIL = 0,5V VIH = 3V fCLOCK = 1	-	90	-	mA

2.9. Mô hình thời gian Isp LSI 1016MHz



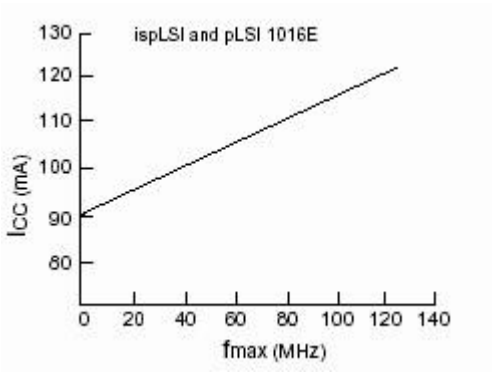
2.10. Thời gian trì hoãn tối đa



Hình 4.3. Thời gian trì hoãn của GRB

2.11. Công suất tiêu thụ

Công suất tiêu thụ của ispLSI 1016 phụ thuộc vào hai yếu tố chính: Tốc độ làm việc và số lượng tích số đã sử dụng. Hình 4.4 cho thấy quan hệ giữa công suất với tốc độ làm việc



Hình 4.4. Công suất tiêu thụ với f_{max}

ICC (mA) = 23 + (# of PTs * 0,52) + (# of Nets * max freq * 0,004)

Trong đó :

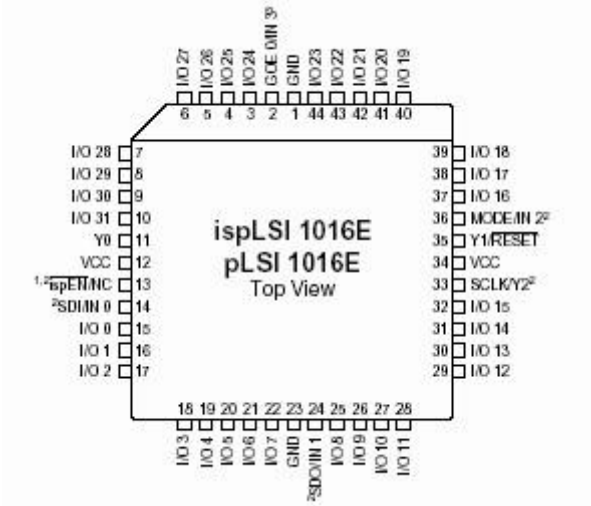
# of PTs : Số tích số đã dùng

# of nets : Số tín hiệu sử dụng max

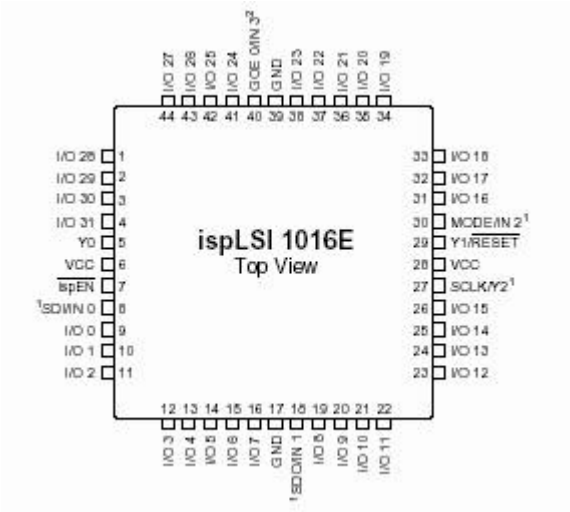
freq : Tần số làm việc cao nhất

Việc tính toán ICC dựa trên các điều kiện chuẩn (VCC = 5 V, 25^0C) với tải là 4 khối GLB và cấu hình thiết bị là 4 bộ đếm 16 bít. Do giá trị của ICC dễ bị ảnh hưởng với điều kiện làm việc và chương trình trong thiết bị nên trị số thực tế của ICC có thể thay đổi.

2.12. Sơ đồ chân

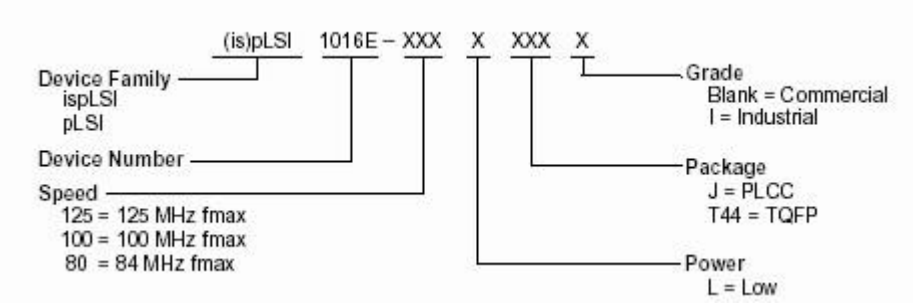


Hình 4.5. Sơ đồ chân PLCC 44



Hình 4.6. Sơ đồ chân TQFP 44

2.13. Ý nghĩa tên linh kiện



BÀI 5: PHẦN MỀM ISP SYNARIO

1. GIỚI THIỆU

Với Synario navigator người dùng có thể thực hiện các yêu cầu thiết kế bao gồm việc truy cập các thành phần trong thiết kế, theo dõi các bước xử lý. Người dùng có thể mở sơ đồ mạch và trình soạn thảo văn bản thông qua các cấp thiết kế. Trong phần sau sẽ hướng dẫn việc ứng dụng phần mềm thiết kế ISP synario qua các đề mục sau :

- Yêu cầu hệ thống
- Khởi động ISP synario
- Nhập một modul VHDL vào dự án thiết kế
- Nhập một sơ đồ vào dự án thiết kế
- Hoàn tất dự án
- Bổ sung thuộc tính
- Tạo véc tơ thử
- Dịch sơ đồ và véc tơ thử
- Thực hiện chức năng mô phỏng và dạng sóng ra
- Tạo một ký hiệu
- Chế độ nhập hỗn hợp
- Tạo file nguồn ABEL-HDL
- Dịch ABEL-HDL - Mô phỏng thiết kế
- Nạp kết quả thiết kế vào thiết bị CPLD

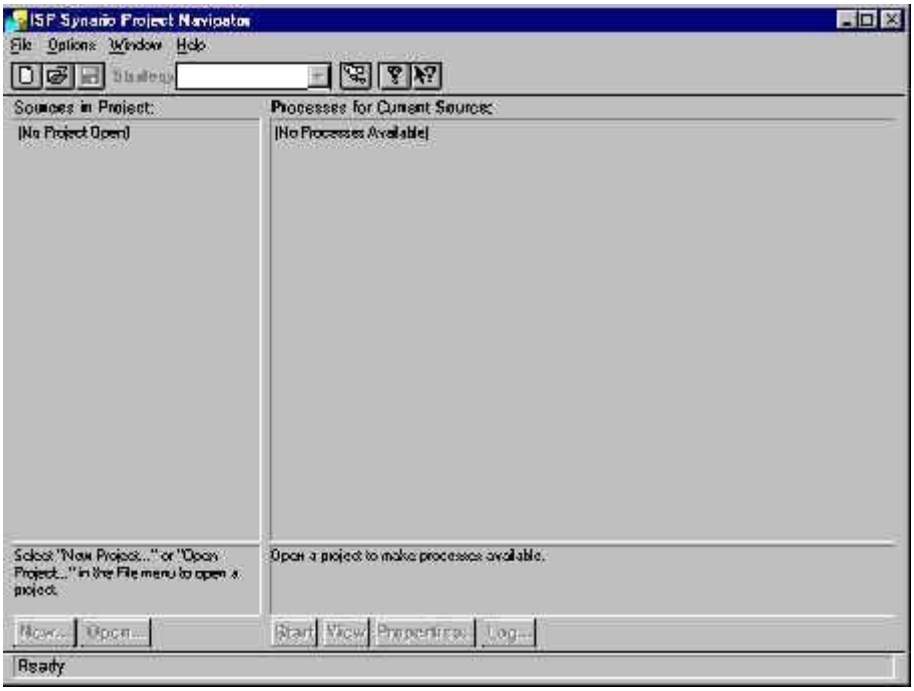
2. YÊU CẦU HỆ THỐNG

Những phần mềm cần thiết cần phải cài đặt trong hệ thống máy tính trước khi bắt đầu thiết kế

- ISP Synario Entry/Simulation
- ISP Synario VHDL Entry
- IspDS + Filter

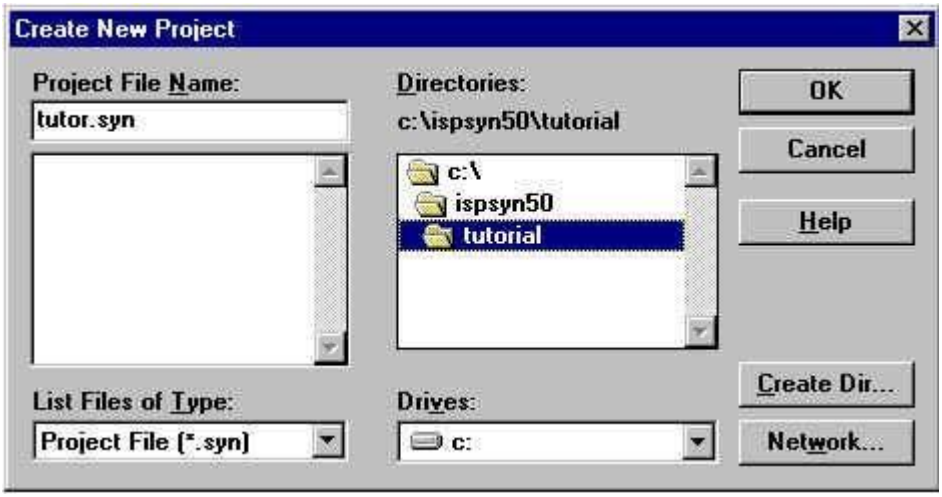
3. KHỞI ĐỘNG SYNARIO

1. Nhấn kép chuột vào biểu tượng ISP Synario, cửa sổ quản lý dự án ISP Synario xuất hiện như ở hình 5.1. Cửa sổ được chia làm hai vùng : Vùng bên trái (sources in project) liệt kê các file nguồn của dự án và vùng bên phải (processes for curent source) cho thấy các quá trình xử lý liên kết với các file nguồn này



Hình 5.1. Cửa sổ quản lý dự án

2. Trong hộp sources in project chọn **File** ⇒ **New Project**, hộp thoại **Create New Project** xuất hiện (h.5.2)



Hình 5.2. Hộp thoại Create New Project

3. Trong ổ đĩa C (mặc định) kích đúp chuột để mở **ispsyn50** (thư mục cài đặt mặc định)

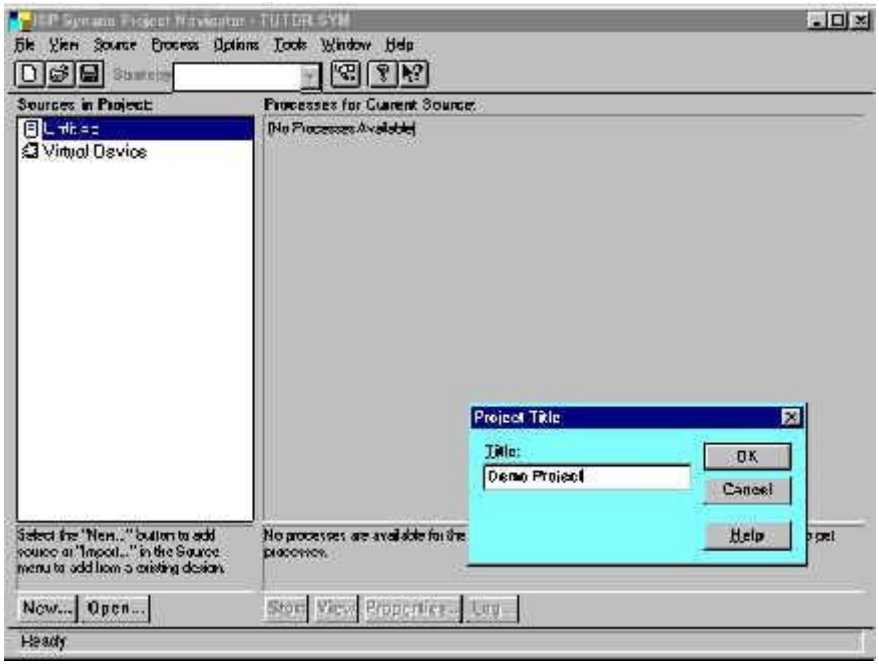
a. Nếu đang dùng hệ ISP Synario thì phải tạo thư mục tutorial. Kích

vào nút **Create Dir** (h.5.2), hộp thoại Create Directory xuất hiện nhập vào **tutorial**

b. Nếu đang dùng hệ ispVHDL Synario thì thư mục tutorial đã được tự động tạo ra trong khi cài đặt. Kích đúp để mở nó ra, trong hộp

Project File Name nhập tên **tutor.syn** và ấn OK

4. Trong cửa sổ quản lý dự án có hai muc Untitled và Virtual Device. Để đặt tên dự án, kích đúp lên Untitled. Một hộp thoại Project Title xuất hiện (h.5.3) nhập Demo Project vào vùng **Title** sau đó ấn OK



Hình 5.3. Cửa sổ quản lý dự án và hộp Project Title

5. Kích chọn mục Virtual Device để xổ xuống cửa sổ Choose Device. Chọn danh sách thiết bị ISP Synario Device List trong vùng Device Kit.

Một danh sách thiết bị xuất hiện (h.5.4)

Hình 5.4. Cửa sổ chọn thiết bị

6. Cuộn qua danh sách thiết bị đến mục ispLSI 2032-150 TQFP44. Chiếu sáng thiết bị này và ấn OK. Bạn sẽ được hỏi để xác nhận sự chọn lựa này vì nó sẽ làm thay đổi môi trường thiết kế dự án. Ấn YES

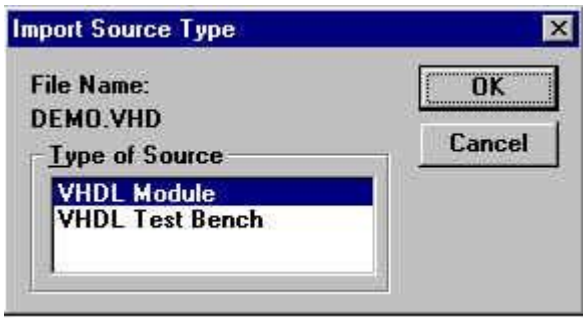
4. NHẬP MODUL VHDL VÀO DỰ ÁN

Thiết kế dự án là sự kết hợp từ một hoặc nhiều nguồn có thể là sơ đồ mạch, file ABEL-HDL, file VHDL (chỉ đối với hệ ispVHDL Synario), file véc tơ thử hoặc các file tài liệu.

Bốn bước sau đây sẽ trình bày chi tiết phương pháp dùng một modul VHDL để thiết kế dự án. Nếu không muốn dùng tính năng VHDL thì có thể bỏ qua bốn bước này

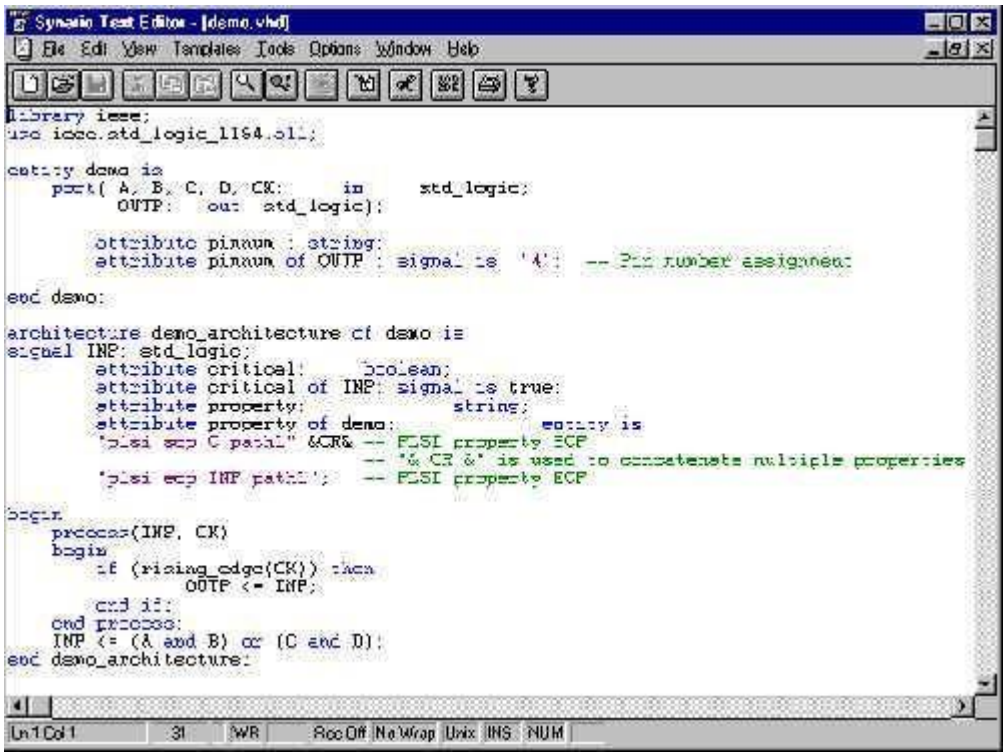
7. VớI thiết bị ispLSI 2032 đã chọn, ghi lại các phương thức xử lý được liên kết với nguồn này. Từ thanh menu chọn **Source** ⇒ **Import**

8. Trong danh sách thư mục của hộp thoại Import File chọn đường dẫn drive :\ispsyn50\tutorial. Chiếu sáng tên file demo.vhd trong vùng file name. Ấn **OK**, hộp thoại Import Source Type xuất hiện, chọn **VHDL Module** và ấn **OK** như ở hình 5.5



Hình 5.5. Hộp thoại Import Source Type

File VHDL demo.vhd xuất hiện trong phần Source của danh sách dự án trong cửa sổ quản lý dự án. Chiếu sáng biểu tượng VHDL và lưu ý phương pháp xử lý liên kết với nó. Có thể xem cú pháp bằng cách dùng trình soạn thảo văn bản (h.5.6)

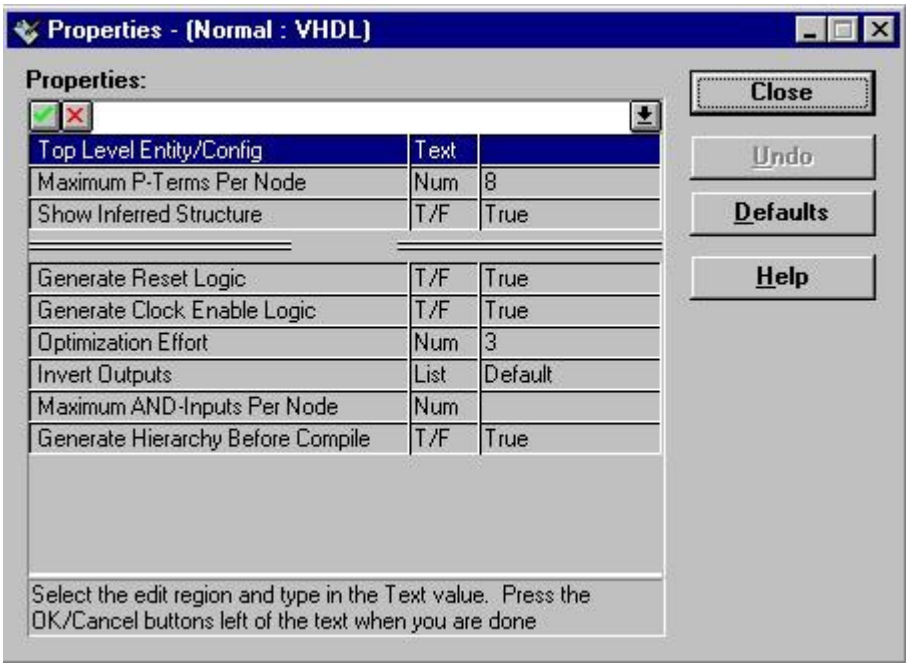


Hình 5.6. Trình soạn thảo văn bản với file demo.vhd

Phải dùng thuộc tính thiết kế LOCK để gán số “4” cho chân I/O, dòng “scp C path1” cho biết đường dẫn khởi động và path1 bao gồm tên duy nhất cho thuộc tính đường dẫn

9. Với file VHDL được chiếu sáng, mục chọn Synthezise Logic trong bảng danh sách xử lý cũng được chiếu sáng. Chọn nút nhấn Properties bên dưới để làm xuất hiện hộp thoại Properities (Normal : VHDL)

10. Các thuộc tính nhìn thấy trên hộp thoại là mặc định của modul VHDL. Chiếu sáng tùy chọn thuộc tính Generate Reset Logic và dùng phím mũi tên để thay đổi từ False thành True. Kích vài dấu kiểm để chấp thuận thay đổi. Hộp thoại hiển thị như ở hình 5.7. Đóng hộp thoại



Hình 5.7. Hộp thoại Properties (Normal. VHDL)

Khi một thuộc tính bất kỳ được thay đổi, nút Default nổi lên cho phép trả về giá trị mặc định nếu cần

5. NHẬP SƠ ĐỒ MẠCH VÀO DỰ ÁN

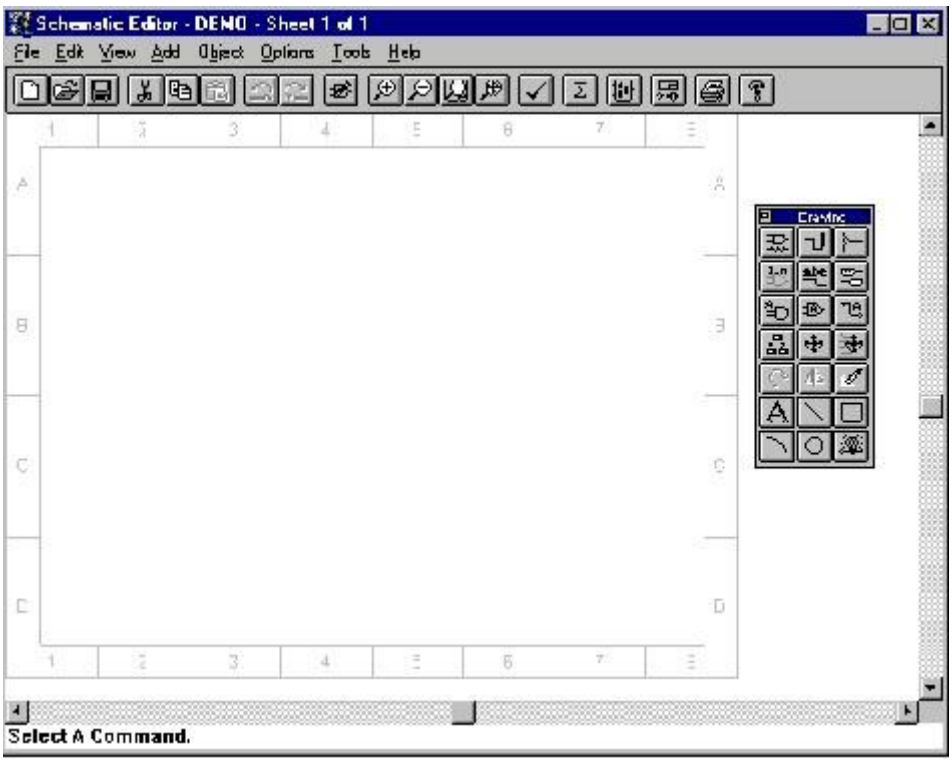
Các bước sau đây hướng dẫn cách nhập một sơ đồ trống vào dự án, tiếp theo là nhập các ký hiệu và kết nối chúng với nhau

11. Với thiết bị 2032 đã được chọn ở trên, ghi lại các bước xử lý tương ứng với nguồn này. Từ thanh menu chọn **Source** ⇒ **New**

12. Trong hộp thoại New Source chọn Schematic và kích **OK** hoặc ấn **Enter**. Một hộp thoại xuất hiện yêu cầu nhập tên cho sơ đồ. Chọn đường dẫn c:\ispsyn50\tutorial và nhập tên file demo.sch trong vùng text. Kích OK hoặc ấn Enter

Lưu ý : Để tránh rắc rối có thể xảy ra tốt nhất nên đặt tên cho file nguồn và file dự án khác nhau

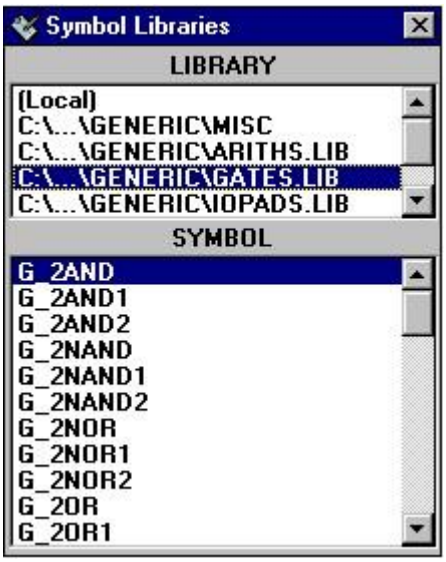
13. Một sơ đồ trống sẽ xuất hiện trong cửa sổ soạn thảo (h.5.8). Từ thanh menu chọn **Add** ⇒ **Symbol**



Hình 5.8. Cửa sổ soạn thảo sơ đồ

Mẹo : Dùng chức năng Zoom trong menu thả xuống ở mục View hoặc trên thanh công cụ Schematic Editor để dễ quan sát

14. Trong hộp thoại Symbol Libraries chọn C:\...\generic\gates.lib từ danh sách thư viện sau đó chiếu sáng ký hiệu G_2AND (h.5.9)



Hình 5.9. Cửa sổ Symbol Libraries

15. Chuyển con trỏ trở về Schematic Editor, lưu ý là cổng AND được

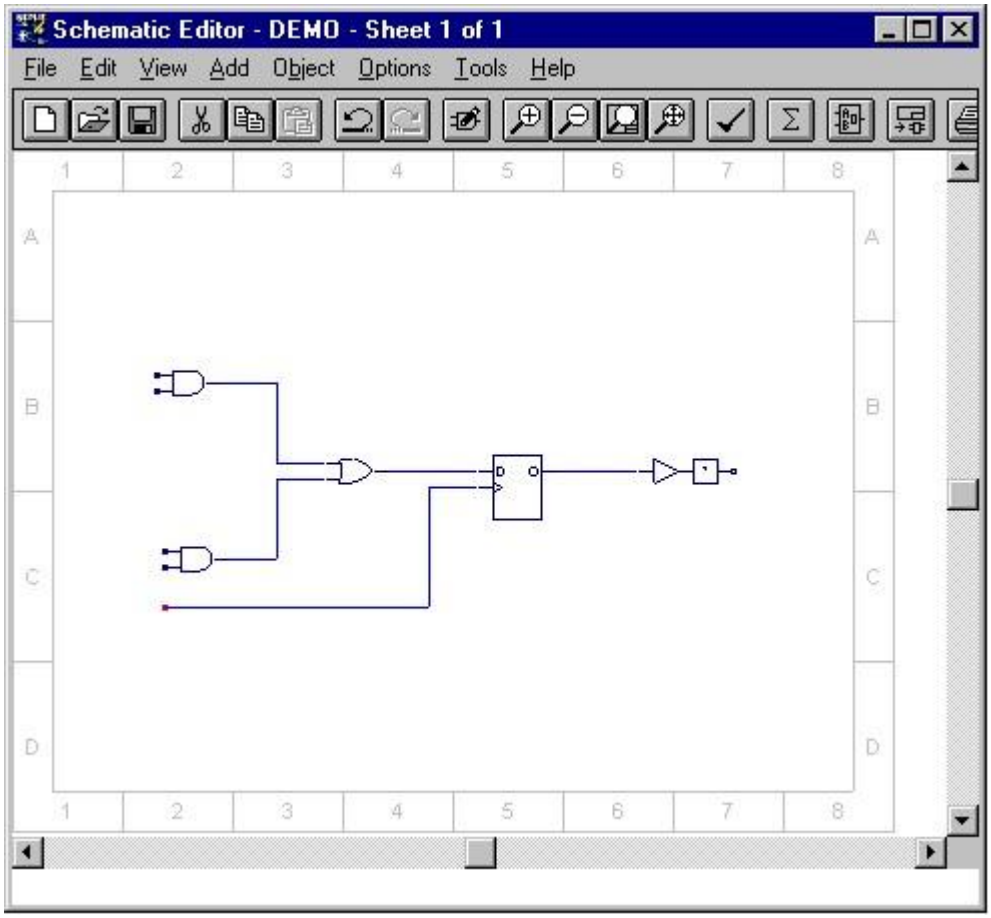
dính kèm theo con trỏ, đặt cổng bằng cách kích lên sơ đồ, đặt thêm một cổng AND khác bên dưới cổng thứ nhất.

16. Đưa con trỏ trở vào hộp thoại Symbol Libraries và chọn ký hiệu G_2OR, đặt cổng này về bên phải của hai cổng AND vừa rồi

17. Từ thanh menu trong Schematic Editor chọn **Add** ⇒ **Wire**, kích vào chân ra của cổng AND phía trên để bắt đầu nối dây. Kích đơn để bẻ cong dây nối và kích đúp để kết thúc dây nối. Kích đơn để nối đến ngõ vào của cổng OR. Lặp lại các thao tác vừa rồi với cổng AND bên dưới

18. Thực hiện lại các bước như trên để nhập vào một thanh ghi g_d từ thư viện REGS.LIB và một G_OUTPUT từ thư viện IOPADS.LIB, nối tất cả chúng lại theo sơ đồ ở hình 5.10

19. Từ thanh menu trong cửa sổ Schematic Editor chọn **File** ⇒ **Save** để lưu lại các thiết kế vừa rồi



Hình 5.10. Xây dựng sơ đồ

6. HOÀN TẤT THIẾT KẾ

Thiết kế được hoàn tất bằng cách thêm tên mạng và các dấu I/O, tên mạng được nhập bằng tính năng của ISP Synario, tính năng này cho phép

nhập cùng lúc mạng và tên mạng. Dấu I/O là các ký hiệu đặc biệt cho biết các tín hiệu biểu diễn chân của linh kiện.

Các ký hiệu này chấp nhận tên mạng mà chúng thâm nhập và khác với các ký hiệu chân I/O

20. Trên thanh menu của Schematic Editor chọn **Add** ⇒ **Net Name**. Thanh trạng thái dưới đáy màn hình sẽ hiện câu nhắc nhập tên mạng. Nhập **A** và ấn Enter. Tên mạng sẽ dính vào con trỏ

21. Di chuyển con trỏ đến ngõ vào trên cùng, kích và giữ tại điểm chưa nối của mạng (chấm màu đỏ bên trái của mạng) drag về bên trái và thả ra. Thao tác này đồng thời đặt tên mạng và tạo ra một mạng vào. Tên mạng nên được đặt tại điểm cuối của mạng

22. Lặp lại các bước trên để nhập tên mạng "B", "C", "D" và "CK" cho các ngõ vào và "OUTP" cho ngõ ra

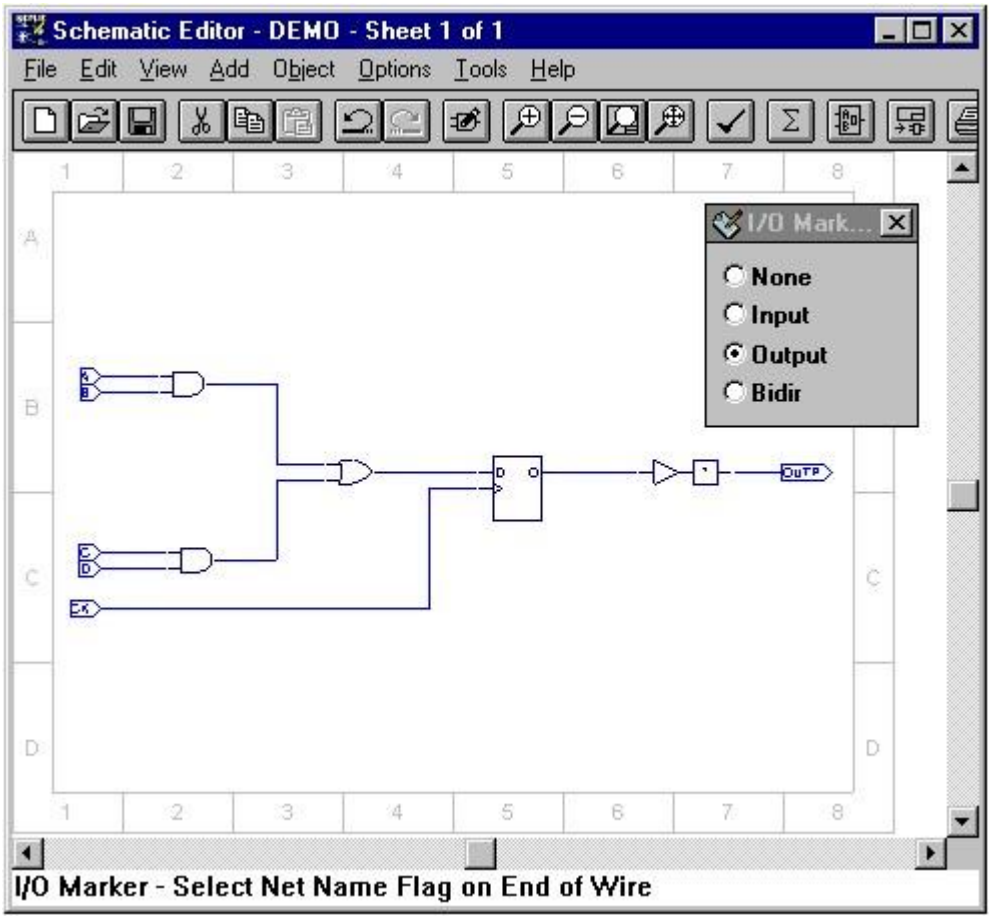
Mẹo : Đối với các ngõ vào còn lại sau khi nhập tên và ấn Enter, mỗi lần kích vào điểm cuối của mạng, chiều dài mạng sẽ bằng với lần đặt đầu tiên bằng cách kích giữ và rê chuột

23. Từ thanh menu trong Schematic Editor chọn **Add** ⇒ **I/O Marker**. Hộp thoại I/O Marker xuất hiện, chọn **Input**

24. Đưa con trỏ kích vào điểm cuối của một mạng vào (ở giữa điểm cuối và tên của mạng). Một dấu I/O xuất hiện với tên mạng chứa bên trong, di chuyển đến ngõ vào kế tiếp và kích chuột. Lặp lại thao tác trên cho đến khi tất cả các ngõ vào đều có dấu I/O

Mẹo : Để nhập một lần tất cả các dấu input, kích và giữ chuột sau đó rê một khung bao tất cả các mạng vào và nhả ra. Thao tác này cũng áp dụng tốt với các ngõ ra

25. Chọn **Output** từ hộp thoại I/O Marker, kích tại điểm cuối của mạng ra. Sơ đồ hoàn tất được trình bày trong hình 5.11 Chọn lưu sơ đồ



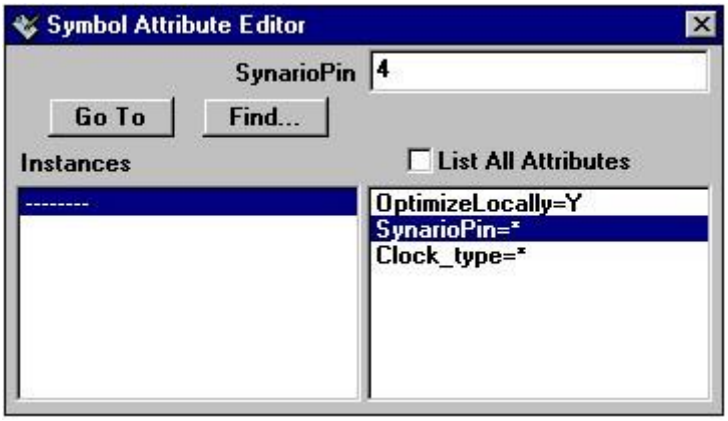
Hình 5.11. Sơ đồ hoàn tất

7. NHẬP THUỘC TÍNH

Thuộc tính có thể được nhập vào ký hiệu hoặc mạng. Ví dụ sau sẽ nhập một thuộc tính LOCK vào ký hiệu chân ra và một cặp thuộc tính SCP/ECP (đường dẫn giới hạn Start/End) vào đường dẫn. Lưu ý là trong ISP Synario các thuộc tính pin được nhập vào các ký hiệu chân I/O thực chứ không phải các dấu I/O và cũng cần nhớ rằng các ký hiệu chân I/O chỉ thực sự cần thiết khi muốn nhập các thuộc tính cho các pin. Nếu không thì chỉ cần dùng các dấu I/O

26. Từ thanh menu trong Schematic Editor chọn **Add** ⇒ **Symbol Attribute**, hộp thoại Symbol Attribute Editor xuất hiện. Trên sơ đồ kích vào chân I/O dính với mạng OUTP. Một danh sách thuộc tính liên quan xuất hiện trong hộp thoại

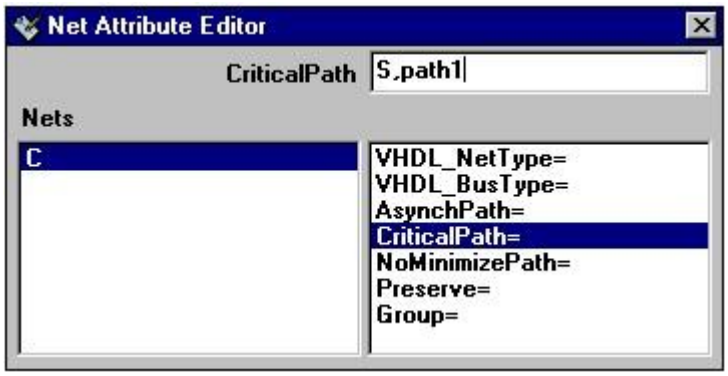
27. Chọn thuộc tính ISP SynarioPin=* và thay thế dấu * bằng số 4 như chỉ ở hình 5.12. Đóng hộp thoại, chú ý : Số 4 đã được nhập vào ký hiệu chân I/O trong sơ đồ



Hình 5.12. Symbol Attribute Editor

28. Thực hiện tương tự để nhập thuộc tính cho mạng hoặc đường dẫn. Từ thanh menu của Schematic Editor chọn **Add** ⇒ **Net Attribute**. Hộp thoại Net Attribute Editor xuất hiện. Trên sơ đồ kích vào mạng nối với ngõ vào "C" của cổng AND, một danh sách thuộc tính liên quan xuất hiện trong hộp thoại

29. Chọn thuộc tính CriticalPath= và nhập **S,path1** vào hộp văn bản như chỉ ở hình 5.13 **S** cho biết vị trí bắt đầu của đường dẫn và **path 1** bao gồm tên duy nhất cho thuộc tính đường dẫn. Thuộc tính này không được hiển thị trên sơ đồ



Hình 5.13. Net Attribute Editor

30. Trên sơ đồ, kích vào mạng nối cổng OR với D-FF. Trong hộp thoại Net Attribute Editor nhập **E,path1** để xác định vị trí kết thúc của đường dẫn giới hạn có tên "path 1". Đóng hộp thoại

31. Kiểm tra lỗi cho sơ đồ bằng cách chọn **File** ⇒ **Consistency Check**. Một cửa sổ thông báo lỗi được hiện lên với câu thông báo No Symbol for this schematic

Nếu sơ đồ là mới và chưa đươc lưu. Nếu là sơ đồ cũ, chọn **File** ⇒ **Matching Symbol**. ISP Synario sẽ hỏi có muốn thay thế file *design.sym*.

Chọn **Replace**

Lưu ý : Nếu các dấu I/O đã được nhập hoặc tháo bỏ thì phải thay thế ký hiệu bằng cách chọn **File** ⇒ **Matching Symbol**

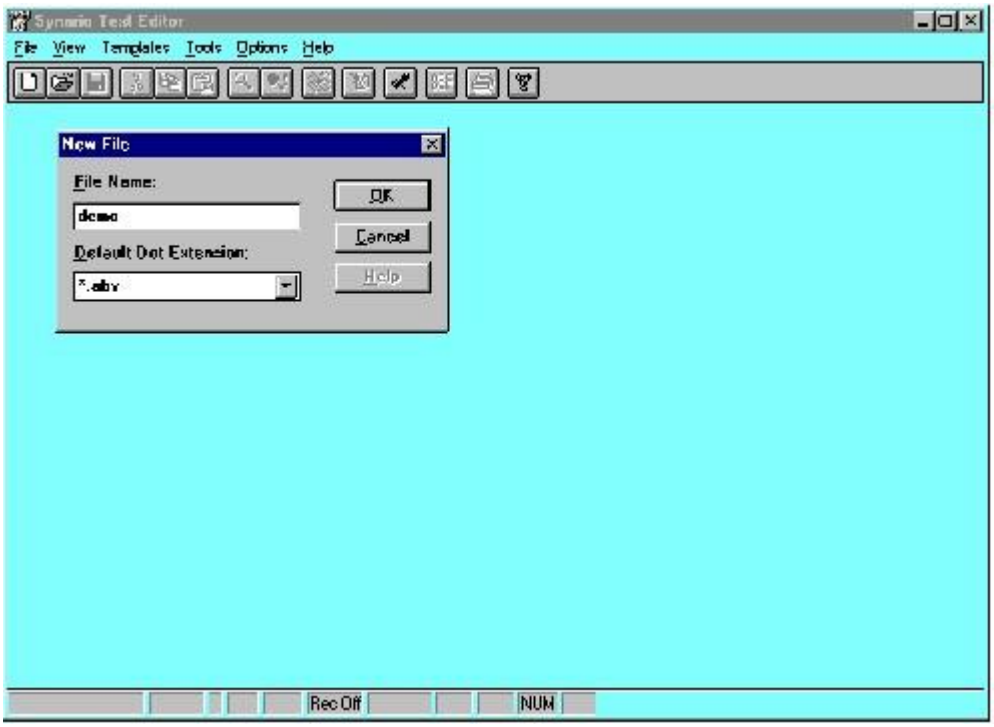
32. Từ thanh menu của Schematic Editor chọn **File** ⇒ **Save** để lưu kết quả thiết kế, chọn **File** ⇒ **Exit** để đóng cửa sổ Schematic Editor. Các thao tác trình bày ở trên cũng có thể được dùng để khóa một pin vào

8. TẠO VÉC TƠ THỬ

Trong ISP Saynario cho phép áp dụng một hoặc nhiều tập các véc tơ thử để mô phỏng chức năng của thiết bị. Trong phần này sẽ giới thiệu cách tạo một tập các véc tơ thử đơn giản

33. Trong ISP Synario Project Navigator, chiếu sáng thiết bị ispLSI 2032-150 TQFP44. Từ thanh menu chọn **Source** ⇒ **New**. Một hộp thoại New Source xuất hiên, chiếu sáng mục **ABEL Test Vectors** và kích **OK** hoặc ấn **Enter**.

34. Hộp thoại New File và một cửa sổ ISP Synario Text Editor còn trống xuất hiện. Nhập tên file **demo** như trong hình 5.14 cho tên file véc tơ thử. Kích OK hoặc ấn Enter

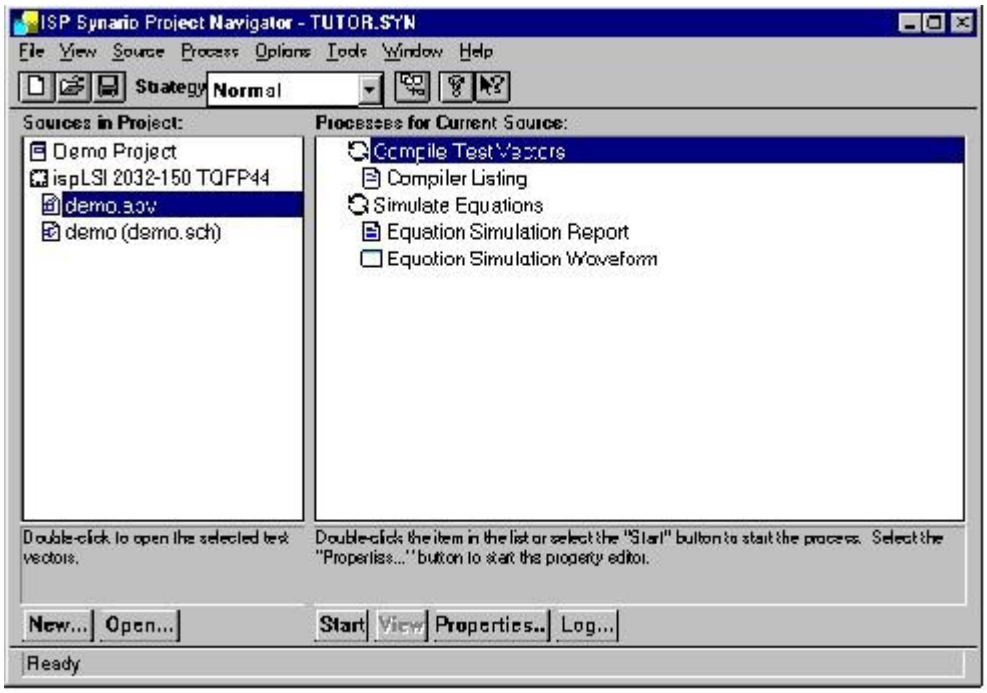


Hình 5.14. ISP Synario Text Editor với hộp thoại New File

35. Trong cửa sổ ISP Synario Text Editor nhập các dòng sau, chú ý chữ thường và chữ hoa (giống ngôn ngữ “C”) module demo;

c,x = .c.,.x.;
CK,A,B,C,D,OUTP PIN;
TEST_VECTORS
([CK,A,B,C,D]->[OUTP])
[c , 0 , 0 , 0 , 0]->[x];
[c , 0 , 0 , 1 , 0]->[x];
[c , 1 , 1 , 0 , 0]->[x];
[c , 0 , 1 , 0 , 1]->[x];
END

36. Sau khi nhập xong chọn **File** ⇒ **Save** từ thanh menu trong ISP Synario Text Editor để lưu file véc tơ thử. Chọn File ⇒ Exit. ISP Synario sẽ trở về cửa sổ Project Navigator như trình bày ở hình 5.15



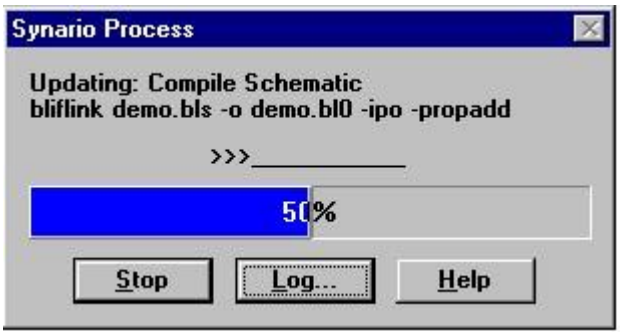
Hình 5.15. ISP Synario Project Navigator

9. BIÊN DỊCH FILE VHDL, SƠ ĐỒ VÀ VÉC TƠ THỬ

Bây giờ với các file nguồn đã được tạo ra trong dự án, bước kế tiếp là thực hiện quá trình xử lý liên hệ với từng file nguồn. Các quá trình xử lý khác nhau này được hiển thị và thực hiện bằng cách chọn từng file nguồn. Trong phần này sẽ trình bày việc biên dịch file sơ đồ và file véc tơ thử độc lập nhau

37. Trong cửa sổ ISP Synario Project Navigator, chiếu sáng file sơ đồ có tên demo (demo.sch) trong bảng danh sách Sources in Project, kích đúp vào mục Compile Schematic trong bảng Processes. Một hộp thoại trạng

thái (h.5.16) xuất hiện trong khoảng thời gian ngắn. Khi quá trình xử lý kết thúc, trên mục Compile Schematic process xuất hiện dấu kiểm màu xanh lá ở kế bên cho biết kết quả biên dịch thành công



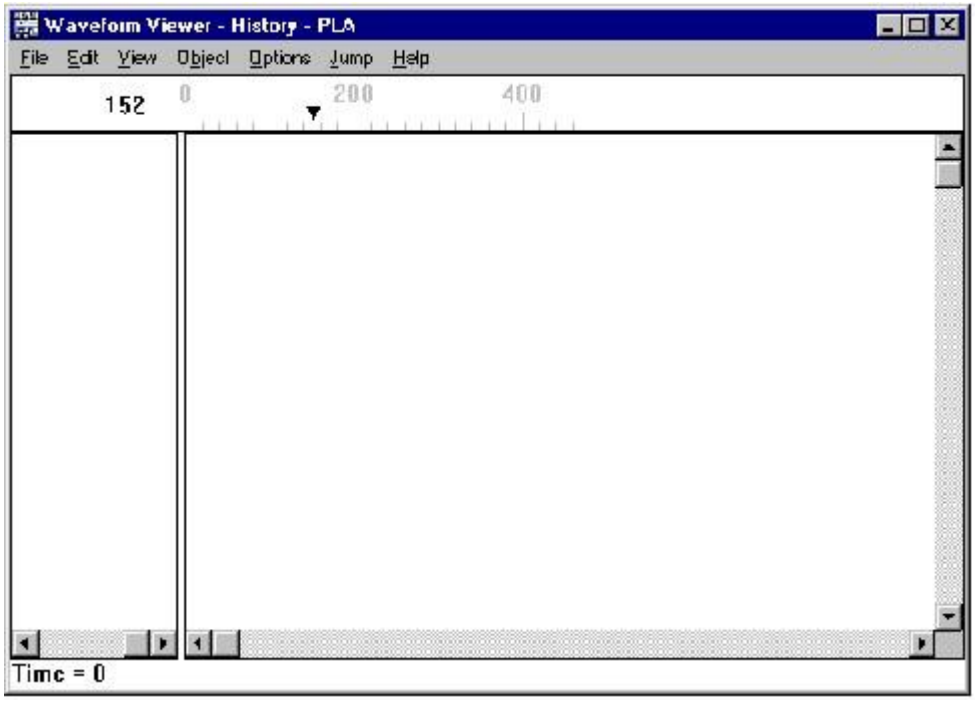
Hình 5.16. Hộp thoại trạng thái ISP Synario Process

38. Trong cửa sổ ISP Synario Project Navigator, chiếu sáng file véc tơ thử demo.abv. Kích đúp vào mục Compile Test Vectors trong bảng Processes. Một dấu kiểm xanh lá xuất hiện kế bên mục Compile Test Vectors khi hoàn tất.

10. MÔ PHỎNG CHỨC NĂNG VÀ DẠNG SÓNG RA

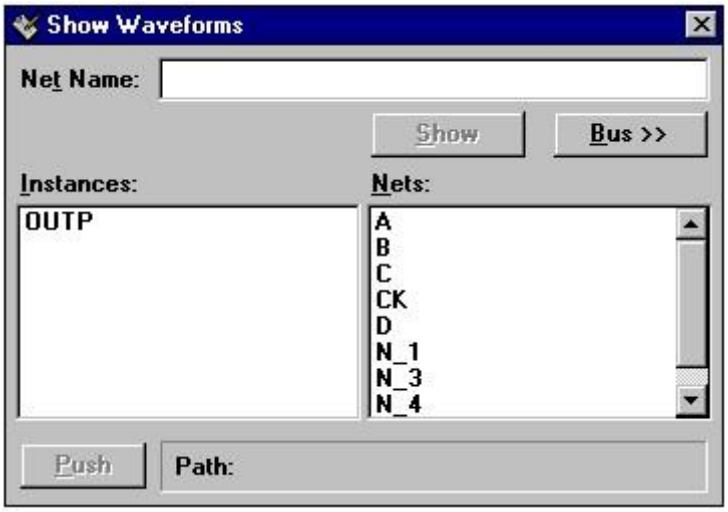
Hệ ISP Synario bao gồm tính năng xem dạng sóng ra và mô phỏng chức năng. Các bước sau đây trình bày phương pháp thực hiện để đạt được kết quả

39. Với file véc tơ thử demo.abv đã được chiếu sáng, kích đúp vào mục Equation Simulation Waveform trong danh sách Processes. ISP Synario sẽ mô phỏng các phương trình và mở cửa sổ Waveform Viewer như trong hình 5.17



Hình 5.17. Cửa sổ Waveform Viewer

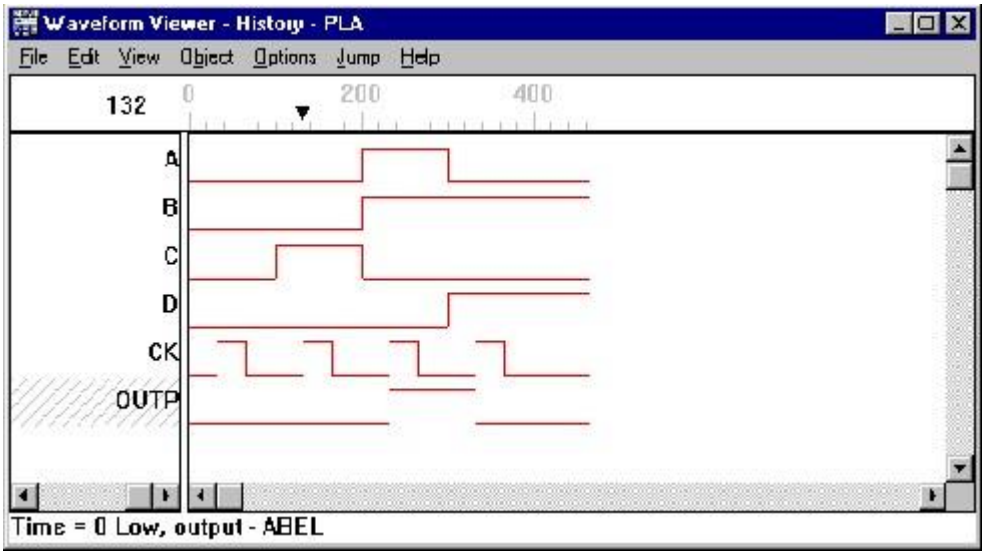
40. Để xem dạng sóng, chọn **Edit** ⇒ **Show** từ thanh menu của Waveform Viewer. Hộp thoại Show Waveforms xuất hiện như trên hình 5.18



Hình 5.18. Hộp thoại Show Waveforms

41. Tại từng thời điểm, chiếu sáng tên tín hiệu cần xem và ấn **Show**, dạng sóng được chọn sẽ xuất hiện trên cửa sổ Waveform Viewer theo thứ tự chọn. (h.5.19). Chọn **File** ⇒ **Exit** để đóng Waveform Viewer, một hộp thoại xuất hiện yêu cầu xác nhận thay đổi dạng sóng PLA

Mẹo : Chọn nhiều tên tín hiệu cùng lúc bằng cách giữ và rê chuột để chiếu sáng chúng. Hoặc ấn giữ phím CTRL và kích vào từng tên để xem



Hình 5.19. Dạng sóng đầy đủ

11. TẠO MỘT KÝ HIỆU

Một tính năng rất hữu dụng của ISP Synario là cho phép tạo nhanh một ký hiệu trong sơ đồ. Với tính năng này có thể tạo các macro có thể dùng lại và đặt vào các bản sơ đồ cấp cao hơn

42. Mở file sơ đồ bằng cách kích đúp vào schematic source, demo.sch trong ISP Synario Project Navigator

43. Trong thanh menu của Schematic Editor chọn **File** ⇒ **Matching Symbol**.

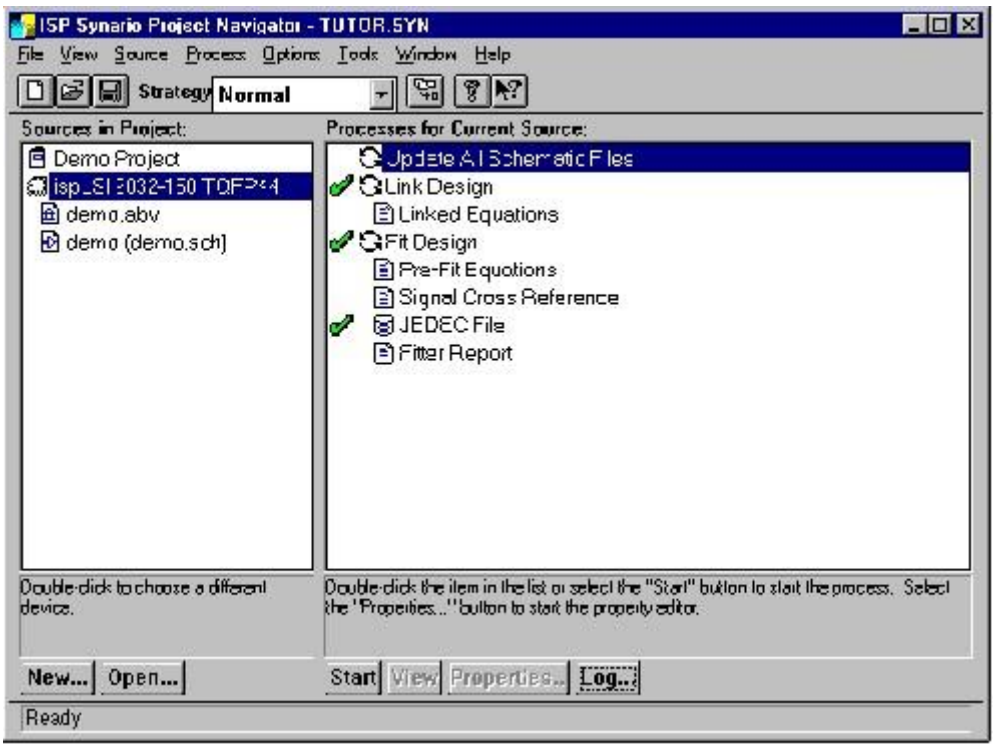
44. Chọn **File** ⇒ **Exit** để đóng sơ đồ

45. Ký hiệu đã được tạo ra và nhập vào bảng symbol sẵn sàng để xử dụng cho lần sau

12. THÍCH ỨNG THIẾT KẾ VỚI THIẾT BỊ CỦA LATTICE SEMICONDUCTOR

Bây giờ ví dụ thiết kế đã được hoàn tất và mô phỏng. Nếu dự định kế tiếp là biên dịch VHDL, ABEL-HDL và sơ đồ thì không cần thiết phải hoàn tất các bước còn lại trong phần trình bày này. Ngược lại, quá trình xử lý cuối cùng là phải làm cho tương thích giửa kết quả thiết kế với họ ispLSI của công ty LATTICE

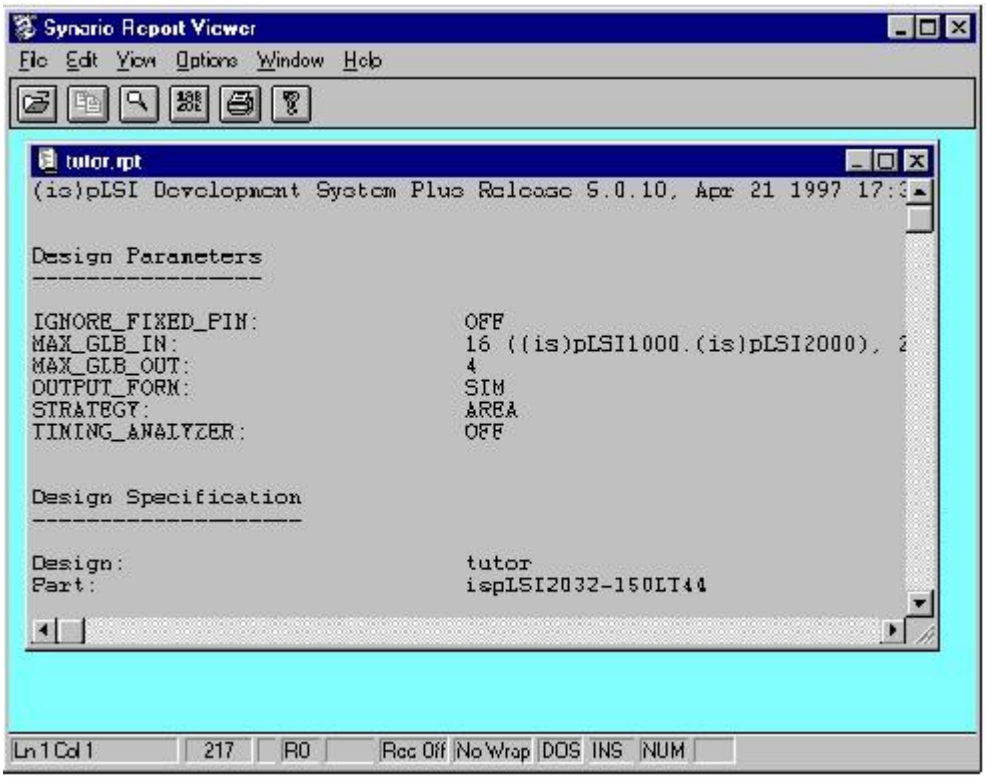
46. Chọn ispLSI 2032-150 TQFP44 và quan sát các xử lý liên quan. Kích đúp vào mục xử lý Fit Design. Một hộp thoại ISP Synario Process xuất hiện. Khi biên dịch xong file nguồn tiếp theo đó ISP Synario sẽ liên kết các file này lại với nhau. Cuối cùng là phân vùng chương trình và tương thích hóa thiết kế vào phần LSC. Khi xử lý hoàn tất ISP Synario quay về cửa sổ Project Navigator (hình 5.20). Chú ý dấu kiểm được thêm vào mục Fit Design báo cho biết việc xử lý thành công



Hình 5.20. ISP Synario Project Navigator sau khi xử lý Pit Design

Lưu ý : Dấu chấm than màu vàng cho biết có cảnh báo được tạo ra và dấu X đỏ báo có lỗi xảy ra

47. Kích đúp vào Fitter Report trong danh sách Processes để xem thông tin về quá trình tương thích hóa kết quả thiết kế



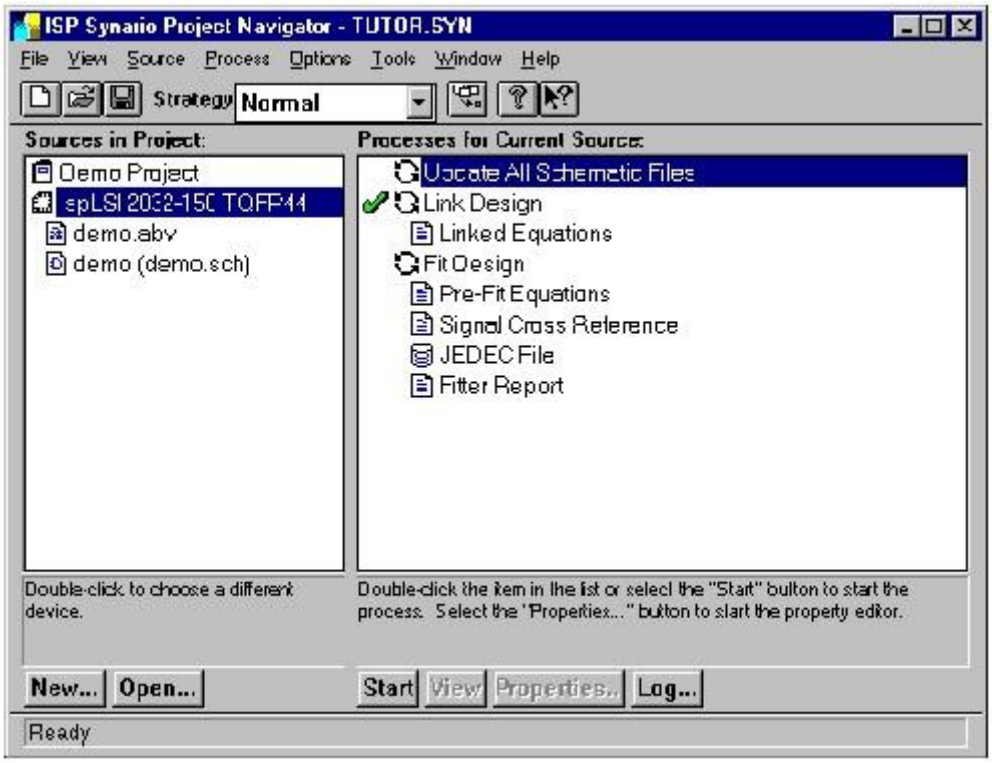
Hình 5.21. Báo cáo sau khi tương thích hóa

13. CHẾ ĐỘ NHẬP HỖN HỢP

Có thể kết hợp phần mềm ispVHDL và ISP Synario để tạo một bản thiết kế VHDL tốt hơn là thiết kế ABEL-HDL. Tuy nhiên, để làm quen với công cụ này trước tiên sẽ tạo ra một thiết kế ABEL-HDL đơn giản sau đó liên kết nó với sơ đồ từ các bước trước trên một sơ đồ cấp đỉnh. Bản thiết kế hoàn tất này sẽ được mô phỏng và biên dịch vào một thiết bị ispLSI

48. Nếu đã thoát ra ISP Synario. Khởi động lại bằng cách kích đúp vào biểu tượng trong nhóm chương trình ISP Synarrio. Màn hình Navigator xuất hiện như hình

5.22. Nếu không giống thì phải bảo đảm tất cả các bước ở phần trên đã được hoàn tất một cách đúng đắn

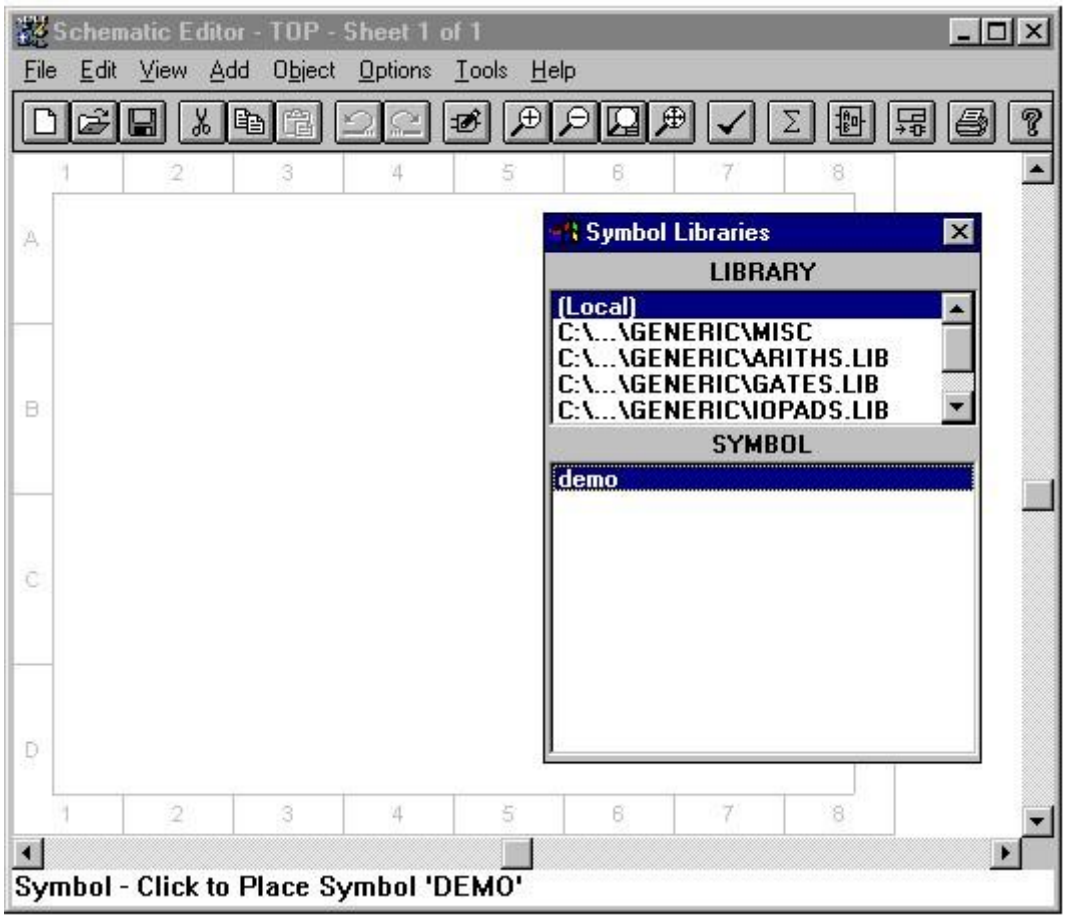


Hình 5.22. ISP Synario Project Navigator

49. Với thiết bị 2032 đã được chọn, chọn mục **Source** ⇒ **New** từ thanh menu của ISP Synario Project Navigator. Trong hộp thoại chọn Schematic và kích **OK**. Chọn đường dẫn c:\ispsyn50\tutorial và nhập tên file top.sch vào hôp File Name. Kích **OK** để vào Schematic Editor

50. Thêm ký hiệu vào sơ đồ đã tạo ở phần trên. Chọn **Add** ⇒ **Symbol**. Hộp thoại Symbol Libraries xuất hiện với Local library được chọn.

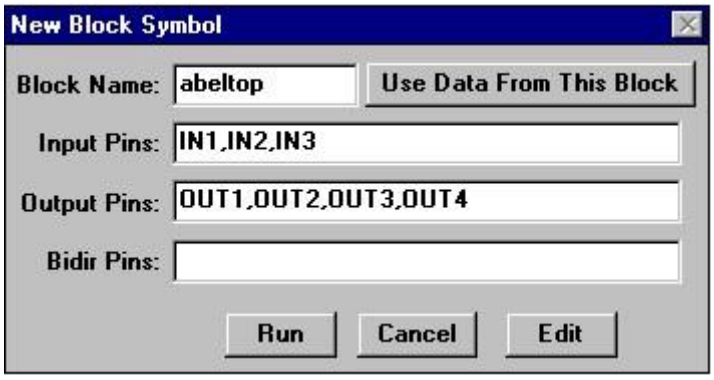
Chú ý, ký hiệu demo nằm trong text box phía dưới (h.5.23). Chọn ký hiệu demo và đặt nó vào sơ đồ



Hình 5.23. Schematic Editor với hộp thoại Symbol Libraries

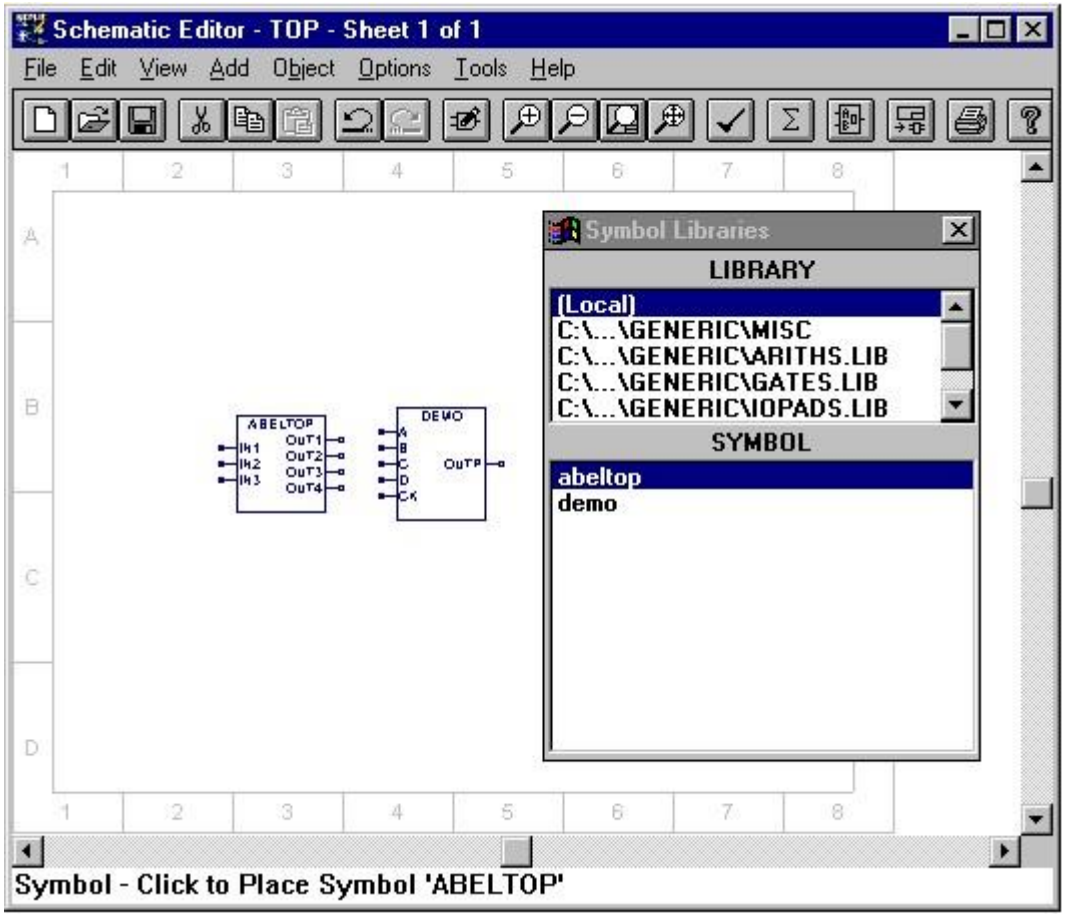
Bước kế tiếp là tạo ra một ký hiệu cấp đỉnh cho file thiết kế ABELHDL. Một ký hiệu có thể được tạo ra cho modul thiết kế với có mức thấp tùy ý khi biết được giao tiếp của nó. File ABEL-HDL thực tế cho dự án thiết kế sẽ được hoàn tất trong một bước tiếp theo sau đây

51. Trong của sổ Schematic Editor chọn **Add** ⇒ **New Block Symbol**, trong hộp thoại hiện ra nhập vào abeltop trong text box Block Name, IN1, IN2, IN3 trong hộp văn bản Input Pins và OUT1, OUT2, OUT3, OUT4 trong hộp văn bản Output Pins (h.5.24). Kích **RUN**



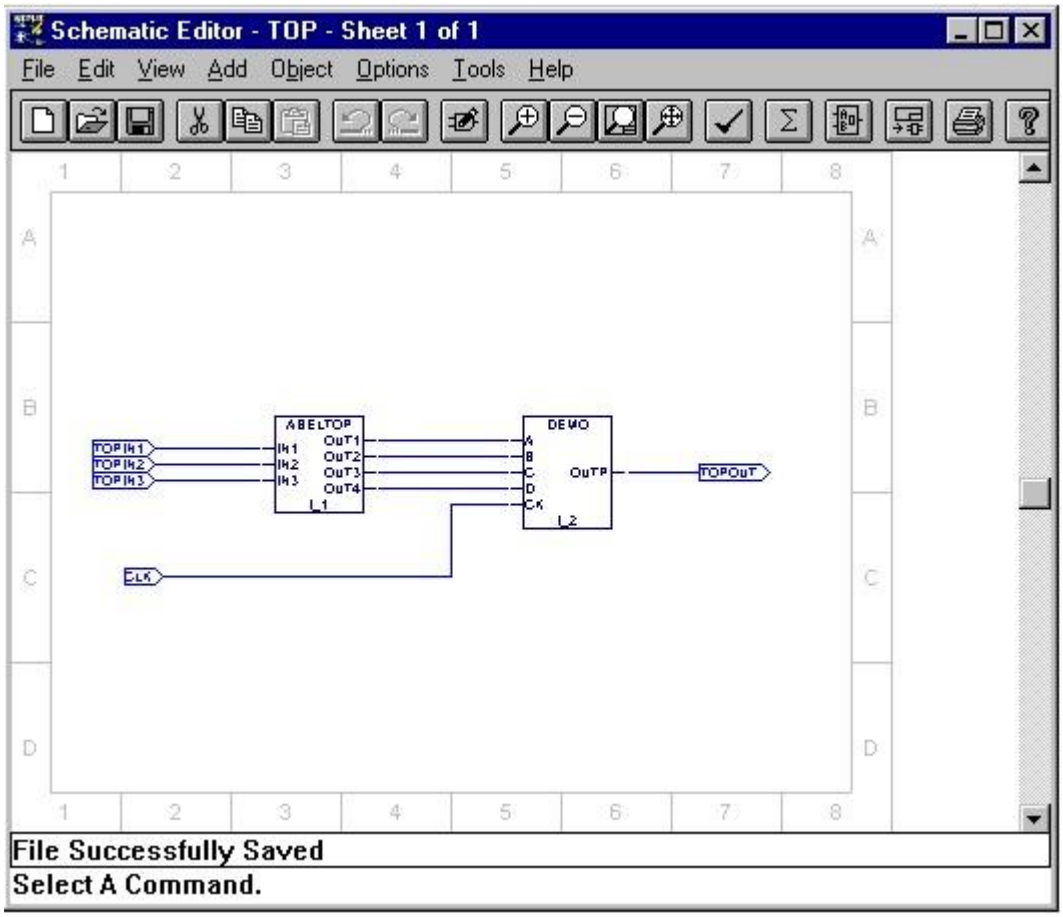
Hình 5.24. Hộp thoại New Block Symbol

52. Một ký hiệu sẽ được thêm vào local library và ký hiệu này dính kèm theo con trỏ, đặt ký hiệu về bên trái của ký hiệu demo. Kích chuột phải để hiển thị hộp thoại Symbol Libraries (h.5.25). Chú ý ký hiệu abeltop trong Local library. Đóng hộp thoại



Hình 5.25. Thêm ký hiệu từ Local Library

53. Hoàn tất sơ đồ cấp đỉnh bằng cách thêm vào các dây nối, tên mạng và các dấu I/O cần thiết để kết thúc thiết kế (h.5.26), thực hiện kiểm tra tính đồng nhất và tạo một ký hiệu tương thích. Lưu thiết kế và thoát khỏi Schematic Editor



Hình 5.26. Hoàn tất thiết kế Abeltop

Nếu thích chỉnh sửa cho đúng và đồng nhất bản thiết kế cấp đỉnh, lập trình viên có thể chuyển qua các mức thiết kế dùng tính năng Hierarchy Navigator. Một số chức năng soạn thảo có hiệu lực qua Navigator

54. Trong các file nguồn trong danh sách Project của cửa sổ ISP Synario Project Navigator, chiếu sáng sơ đồ cấp đỉnh (top.sch). Trong danh sách Processes kích đúp lên Navigate Hierarchy, một hộp thông điệp Building Hierarchy xuất hiện trong một khoảng thời gian ngắn. Sau đó cửa sổ Hierarchy Navigator hiện ra với bản thiết kế cấp đỉnh

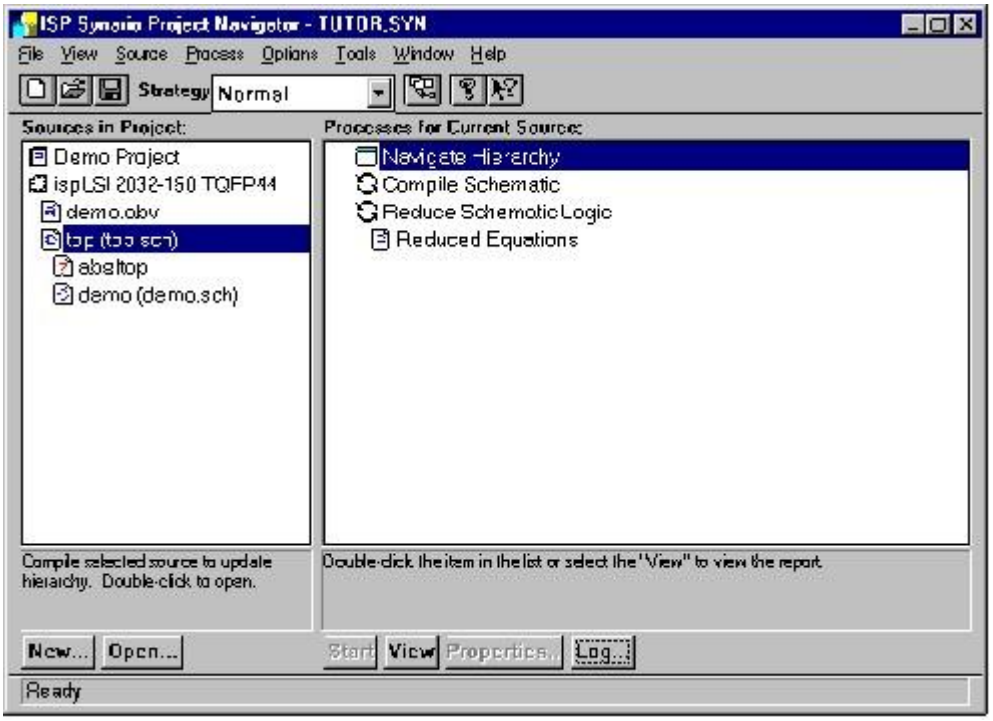
55. Chọn **View** ⇒ **Push/Pop**, con trỏ biến thành hình chữ thập. Kích lên ký hiệu mong muốn, Hierarchy Navigator sẽ mở trang của ký hiệu đó tại cấp kế tiếp. Nếu kích lên ký hiệu gốc sẽ xuất hiện một thông báo nhắc nhở ở dưới đáy cửa sổ Navigator

56. Chọn **File** ⇒ **Exit** để đóng Hierarchy Navigator và hiện lên câu hỏi save changes you made nếu có thay đổi nào đó

14. TẠO FILE NGUỒN ABEL-HDL

Bây giờ cần phải tạo file nguồn ABEL-HDL và liên kết nó với ký hiệu trên sơ đồ cấp đỉnh. Project Navigator thực hiện việc này một cách dễ dàng

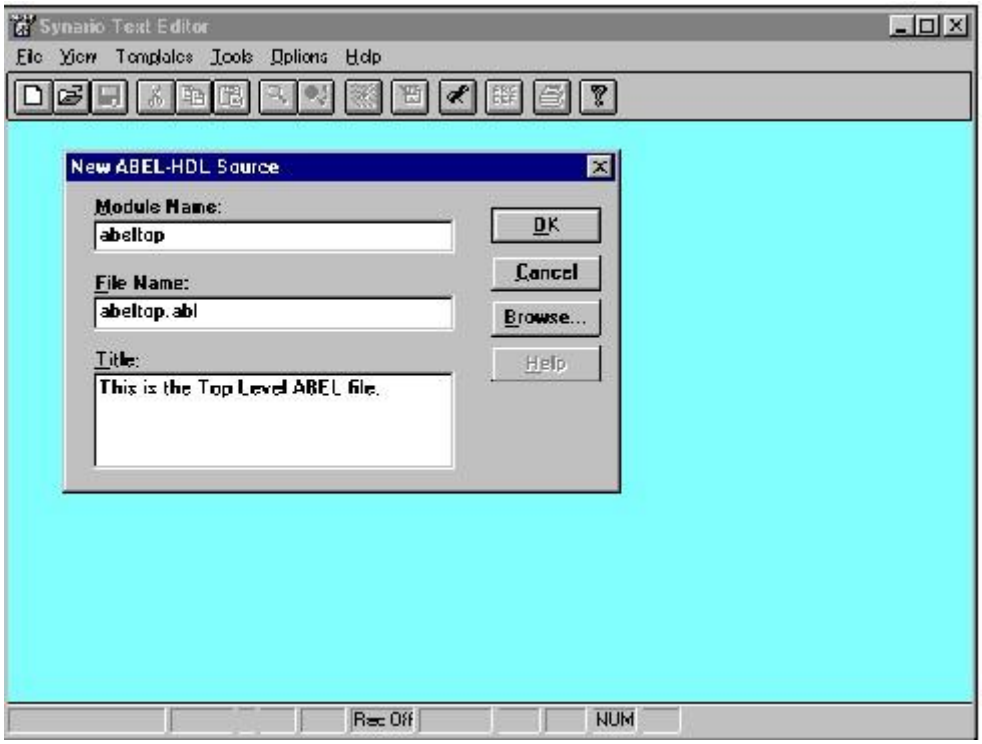
57. Cửa sổ Navìgator giống như trong hình 5.27 Biểu tượng "?" của abeltop có nghĩa là file này không biết do đã chưa tạo ra nó. Lưu ý đến khoảng thụt vào của abeltop và file demo so với sơ đồ cấp đỉnh (sơ đồ chính)



Hình 5.27. ISP Synario Project Navigator

58. Để tạo file nguồn, chiếu sáng mục abeltop sau đó chọn **Source** ⇒ **New**. Trong hộp thoại New Source chọn ABEL-HDL Module và kích **OK**.

Một trình soạn thảo văn bản ISP Synario hiện ra với hộp thoại New ABEL-HDL Source (h.5.28)



Hình 5.28. Hộp thoại New Abel – HDL Source

Để file được liên kết với ký hiệu phải đặt tên modul giống với tên ký hiệu. Tên file không cần giống tên ký hiệu nhưng để cho đơn giản cũng nên đặt tên giống nhau. Điền vào các vùng văn bản như sau : Module Name: abeltop

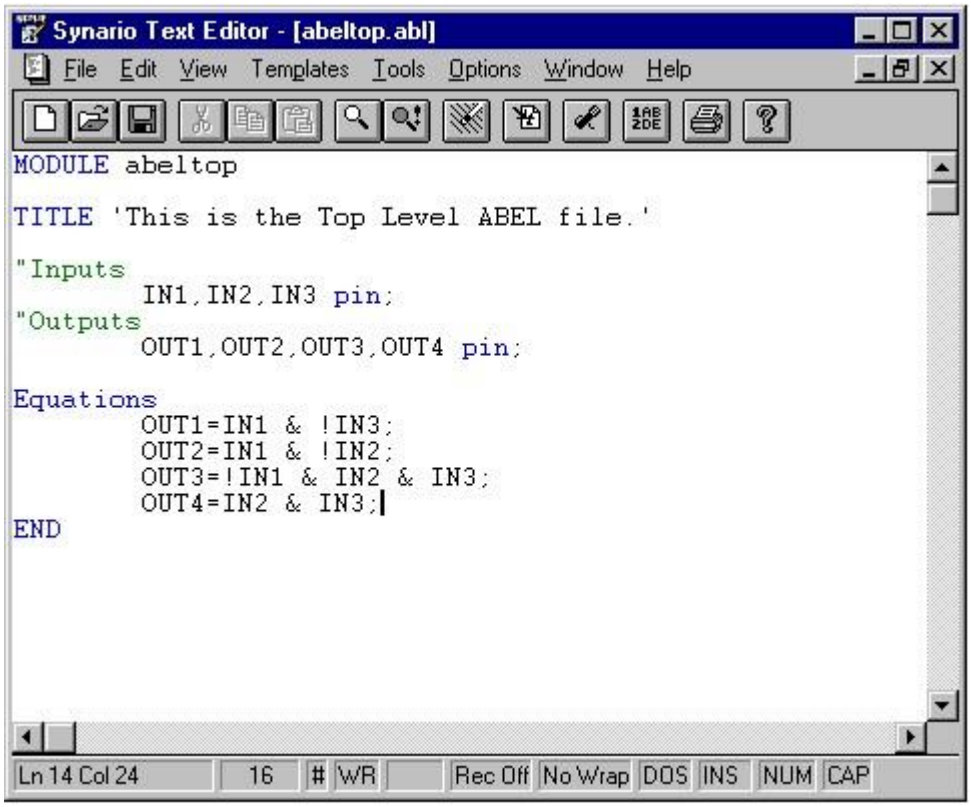
File Name: abeltop.abl

Title: This is the Top Level ABEL file

Kích **OK** để vào trình soạn thảo văn bản ISP Synario và sẽ thấy khung làm việc được khởi tạo sẵn

59. Nhập vào mã lệnh như bên dưới phải bảo đảm nhập đúng ở khoảng giữa hai câu lệnh TITLE và END như trong hình 5.29

```
"Inputs
IN1,IN2,IN3 pin;
"Outputs
OUT1,OUT2,OUT3,OUT4  pin;
Equations
OUT1=IN1 & !IN3;
OUT2=IN1 & !IN2;
OUT3=!IN1 & IN2 & IN3;
OUT4=IN2 & IN3;
```

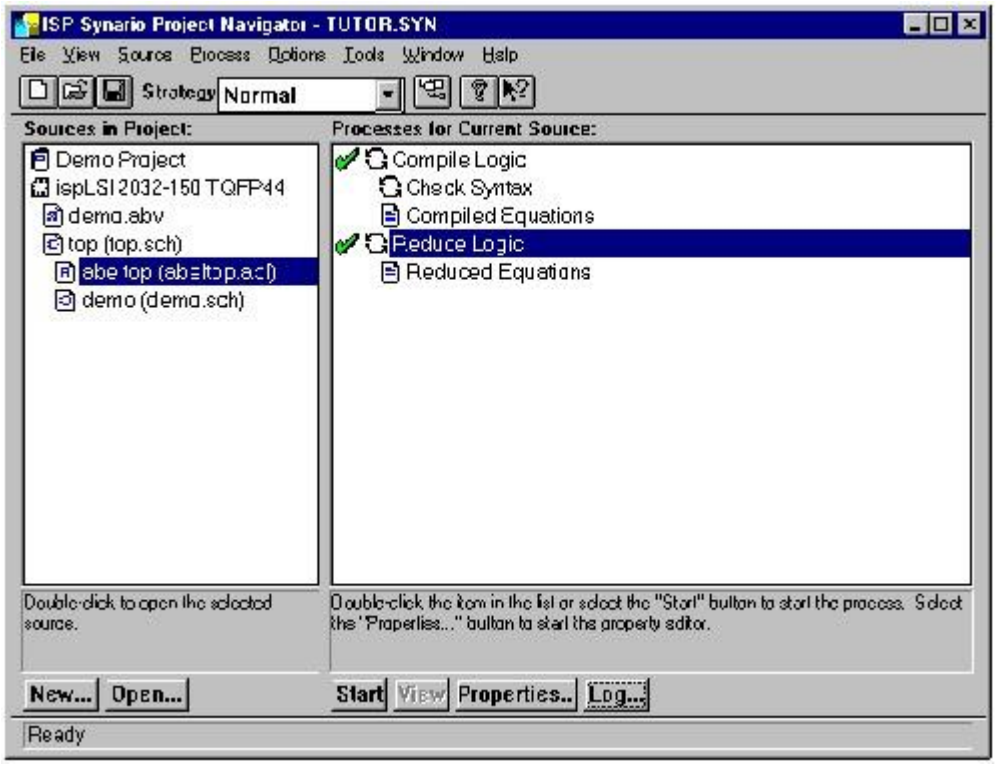


Hình 5.29. Trình soạn thảo văn bản ISP Synario

60. Chọn **File** ⇒ **Save** sau đó **File** ⇒ **Exit**. Chú ý trong cửa sổ Project Navigator biểu tượng kế bên abeltop đã thay đổi, có nghĩa là đã có một file ABEL-HDL tương ứng và nó sẽ được liên kết chính xác.

15. BIÊN DỊCH ABEL-HDL

61. Trong các file nguồn trong vùng Project của Project Navigator chọn abeltop (abeltop.abl), trong cửa sổ Processes for Current Source kích đúp vào mục Reduce Logic. Navigator sẽ biên dịch cẩn thận trước khi thực hiện yêu cầu Reduce Logic (tối thiểu hóa). Khi kết thúc màn hình Navigator giống như hình 5.30



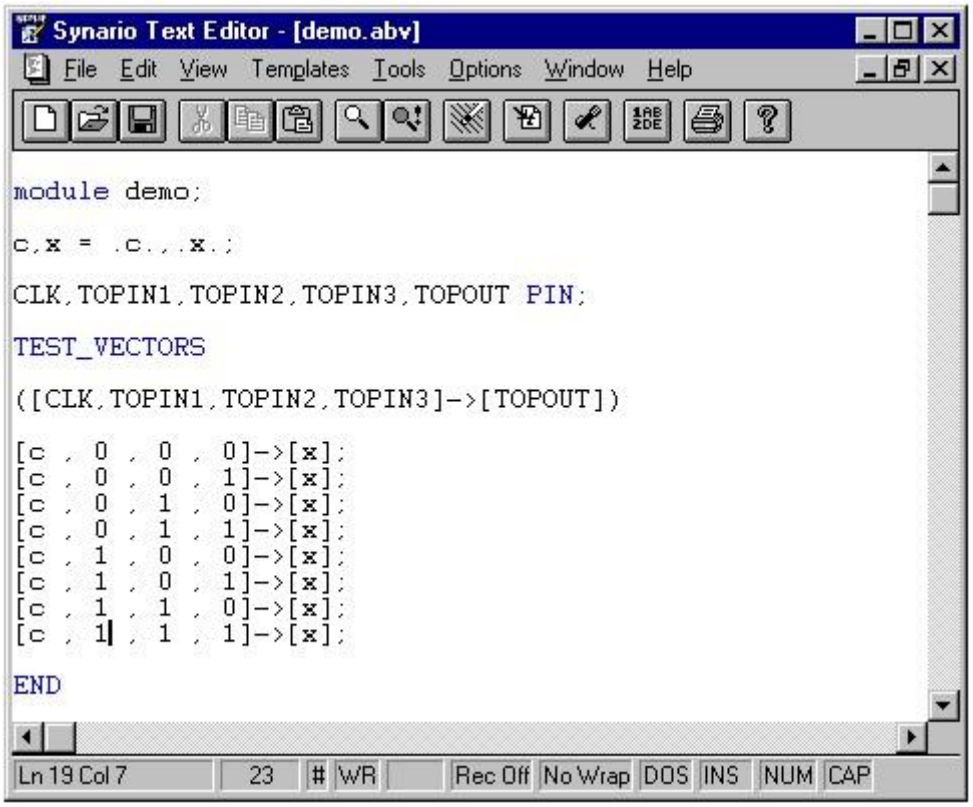
Hình 5.30. ISP Synario Project Navigator sau khi biên dịch ABEL – HDL

16. MÔ PHỎNG KẾT QUẢ THIẾT KẾ

Trong phần này sẽ mô phỏng toàn bộ kết quả thiết kế, để thực hiện cần phải có một file véc tơ thử (.abv) trong phần này sẽ chỉnh sửa file véc tơ thử đã được tạo ra ở các phần trên.

62. Kích đúp lên file demo.abv trong cửa sổ Project Navigator, trình soạn thảo văn bản ISP Synario xuất hiện với cú pháp đã được nhập ở bước 39. Chỉnh sửa như sau :

```
module demo;
c,x = .c.,.x.;
CLK,TOPIN1,TOPIN2,TOPIN3,TOPOUT PIN;
TEST_VECTORS
([CLK,TOPIN1,TOPIN2,TOPIN3]->[TOPOUT]) [c , 0 , 0 , 0]->[x];
[c , 0 , 0 , 1]->[x];
[c , 0 , 1 , 0]->[x];
[c , 0 , 1 , 1]->[x];
[c , 1 , 0 , 0]->[x];
[c , 1 , 0 , 1]->[x];
[c , 1 , 1 , 0]->[x];
[c , 1 , 1 , 1]->[x];
END
```



Hình 5.31. File vector thử trong trình soạn thảo văn bản ISP Synario

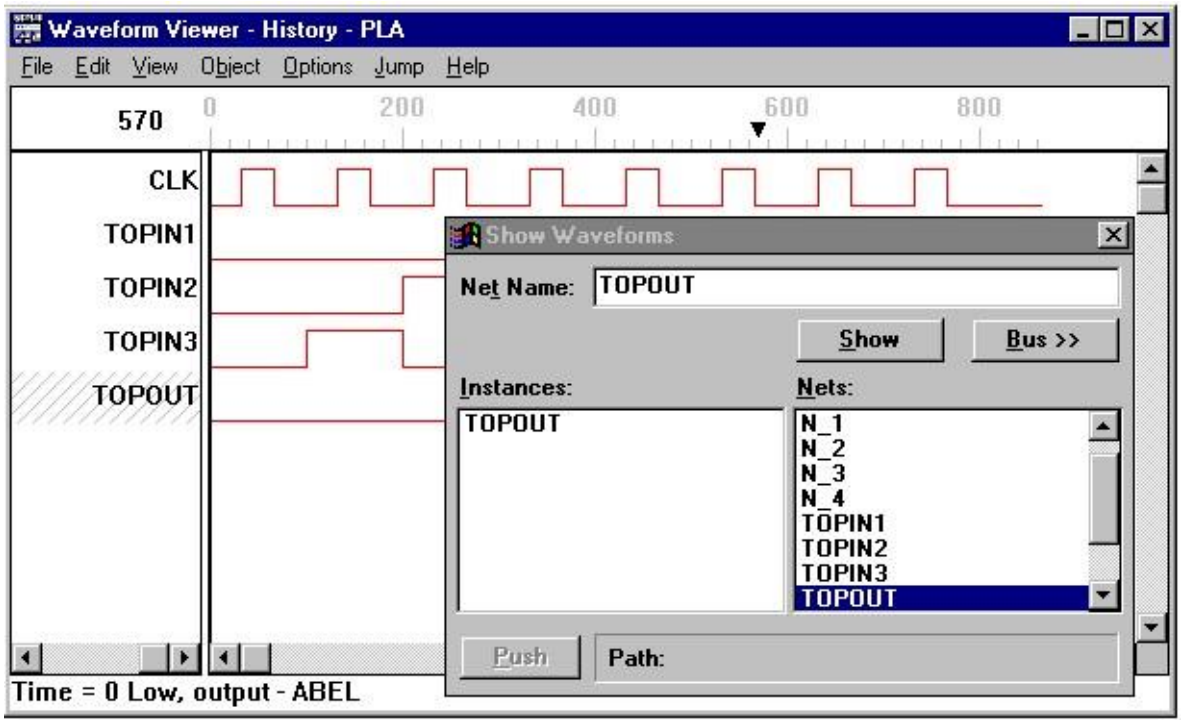
63. Sau khi chỉnh sửa xong chọn **File** ⇒ **Save**, tiếp theo chọn **File** ⇒ **Exit**

64. Trong Navigator với véc tơ thử vẫn còn được chọn, kích đúp lên Equation Simulation Waveform process, cửa sổ Waveform Viewer xuất hiện

65. Để xem dạng sóng chọn **Edit** ⇒ **Show**. Hộp thoại Show Waveforms hiện ra với đầy đủ tên các tín hiệu

66. Để xem cùng lúc chọn TOPIN1, TOPIN2, TOPIN3, TOPOUT và CLK và kích **Show**. Từng tín hiệu sẽ hiển thị trong Waveform Viewer

(h.5.32). Đóng hộp thoại Show Waveforms, chọn **File** ⇒ **Save** trong cửa sổ Waveform Viewer, tiếp theo chọn **File => Exit**



Hình 5.32. Hộp thoại Show Waveforms và Waveforms Viewer

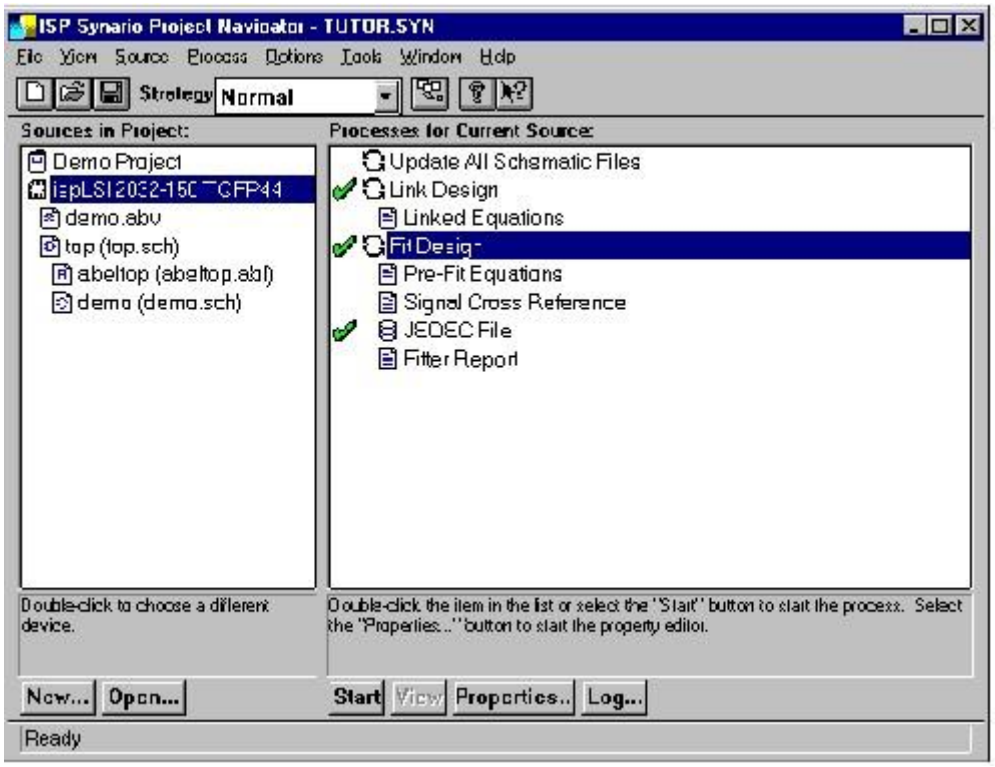
17. THÍCH ỨNG THIẾT KẾ VỚI THIẾT BỊ LATTICE

Dự án thiết kế hỗn hợp giữa sơ đồ với ABEL-HDL đã hoàn tất và đã chạy mô phỏng, chỉ còn bước cuối cùng là tương thích hóa thiết kế với họ ispLSI của LATTICE

67. Từ cửa sổ Sources in Project trong ISP Synario Project Navigator chọn ispLSI 2032-150 TQFP44 và quan sát các quá trình xử lý liên quan được tạo ra

68. Trong danh sách Processes for Current Source kích đúp lên mục Fit Design.

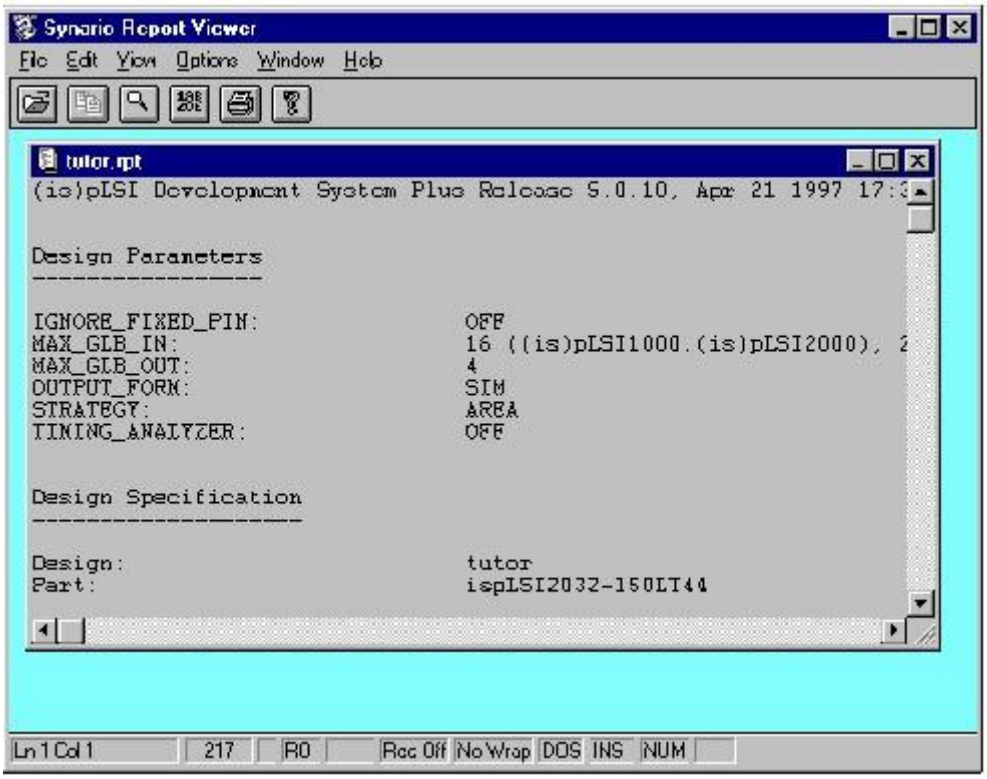
Việc làm này sẽ bắt Navigator phải kết thúc quá trình dịch và liên kết các file nguồn lại với nhau trước khi phân vùng và tương thích thiết kế vào một phần LSC. ISP Synario Project Navigator sẽ thêm dấu kiểm khi nhiệm vụ thành công (h.5.33)



Hình 5.33. ISP Synario Project Navigator sau khi tương thích thành công

69. Phần mềm ispDS+ có một số điều khiển người dùng có thể được truy cập từ Navigator, chiếu sáng Fit Design và kích **Properties** tại phía dưới đáy của cửa sổ Navigator, một hộp thoại chứa danh sách các thuộc tính xuất hiện

70. Để xem tất cả các thông tin liên quan đến việc tương thích hóa kích đúp vào ISP Synario Fitter Report, cửa sổ tutor.rpt hiện ra trong ISP Synario Report Viewer (h.5.34), có thể cuốn cửa sổ report đối với dữ liệu thiết kế



Hình 5.34. ISP Synario Report Viewer với các chức năng chính của ISP VHDL và ISP Synario

TÀI LIỆU THAM KHẢO

1. Đề cương môđun/môn học nghề Sửa chữa thiết bị điện tử công nghiệp " Dự án Giáo dục kỹ thuật và Dạy nghề (VTEP), Tổng cục Dạy Nghề, Hà Nội, 2003

2. Kỹ thuật điện tử số - *Đặng văn Chuyết - NXB KHKT 2003*

3. Kỹ thuật điều khiển- *NXB Lao Động 2004*

4. Mạch điện tử- *NXB Lao Động 2002*

5. Mạch điện tử trong công nghiệp - *Nguyễn Tấn Phước - NXBKhoa học kỹ thuật 2004*

6. Baugruppen de Mikroelektronik III - *Pflaum Verlag Muenchen.*

7. Xilinx - ABEL Design Software Refernce Mannual - Data I/O Corp..1993

8. D. Vanden Bout, ‘ Xilinx FPGA Student Manua” - *Prentice Hall, Englewoods Cliff, NJ, 1997*