

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**NGUYỄN NGỌC MINH**

**NGUYỄN TRUNG HIẾU**

**BÀI GIẢNG  
HỆ THỐNG NHÚNG**

**HÀ NỘI – 12.2014**

DRAFT

# Mục lục

Mục lục .....	3
CHƯƠNG 1- GIỚI THIỆU CHUNG VỀ HỆ THỐNG NHÚNG .....	5
1.1 Khái niệm Hệ thống nhúng (Embedded system) .....	5
1.2 Lịch sử phát triển của hệ thống nhúng .....	5
1.3 Các đặc điểm hệ thống nhúng .....	6
1.3.1 Giao diện .....	7
1.3.2 Kiến trúc CPU .....	7
Kiến trúc điển hình của hệ thống nhúng .....	9
Phân loại các hệ thống nhúng .....	12
1.4 Phạm vi ứng dụng của hệ thống nhúng .....	12
1.5 Các yêu cầu về kĩ năng trong thiết kế hệ thống nhúng .....	13
1.5.1 Quản lý, tích hợp, thiết kế hệ thống: .....	15
1.5.2 Thiết kế, phát triển phần mềm ứng dụng .....	16
1.5.3 Thiết kế firmware .....	16
1.5.4 Thiết kế mạch, PCB: .....	17
1.5.5 Thiết kế vi điện tử : linh kiện, IP, IC, phụ kiện .....	17
Câu hỏi ôn tập .....	19
CHƯƠNG 2: CÁC THÀNH PHẦN HỆ THỐNG .....	20
2.1 Các thành phần phần cứng .....	20
2.1.1 Bộ xử lý nhúng .....	20
2.1.2 Bộ nhớ .....	29
2.1.3 Bảng mạch Vào/Ra .....	31
2.1.5 Hệ thống Bus .....	40
2.2 Các thành phần phần mềm của hệ thống: .....	50
2.2.1. Trình điều khiển thiết bị .....	50
2.2.2. Hệ điều hành thời gian thực .....	51
2.2.3. Middleware .....	52
2.2.4 Phần mềm ứng dụng .....	55
Câu hỏi ôn tập .....	57
CHƯƠNG 3 - HỆ ĐIỀU HÀNH THỜI GIAN THỰC DÙNG CHO CÁC HỆ THỐNG NHÚNG .....	58
3.1 Yêu cầu chung cho các hệ điều hành thời gian thực .....	58
3.2 Các chức năng chính của phần lõi trong hệ điều hành thời gian thực .....	59
3.2.1. Kernel .....	59
3.2.2 Tác vụ và đa nhiệm .....	60
3.3.3 Lập lịch thời gian thực (Real-time Scheduling) .....	62
3.3.4 Đồng bộ .....	64
3.2.5 HAL (Hardware Abstraction Layer) .....	66
3.3 Giới thiệu các hệ điều hành thời gian thực .....	66
3.3.1 FreeRTOS: .....	66
3.3.3 Hệ điều hành Embedded Linux: .....	76
3.3.4 Hệ điều hành uCLinux: .....	77
Câu hỏi ôn tập .....	78
CHƯƠNG 4: THIẾT KẾ VÀ CÀI ĐẶT CÁC HỆ THỐNG NHÚNG .....	79
4.1 Thiết kế hệ thống .....	79

4.1.1 Xác định yêu cầu.....	79
4.1.2 Đặc tả .....	80
4.1.3 Phân hoạch phần cứng - phần mềm .....	84
4.1.4 Thiết kế hệ thống .....	92
4.2 Cài đặt và thử nghiệm hệ thống nhúng .....	109
Câu hỏi ôn tập .....	112
<b>CHƯƠNG 5 PHÁT TRIỂN HỆ THỐNG NHÚNG DỰA TRÊN VXL ARM.....</b>	<b>113</b>
5.1 Giới thiệu chung.....	113
5.2 Kiến trúc của hệ vi xử lý nhúng ARM.....	113
5.2.1 Lõi ARM.....	113
5.2.2 Thanh ghi và các chế độ hoạt động.....	114
5.2.3 Pipeline .....	116
5.2.4 Cấu trúc bus .....	117
5.2.5 Bản đồ bộ nhớ.....	119
5.2.6 Tập lệnh ARM .....	122
5.2.7 Tập lệnh Thumb.....	130
5.3 Thiết kế các thành phần cơ bản của bộ xử lý nhúng.....	130
5.3.1 Lập trình các thanh ghi .....	130
5.3.2 Thiết kế điều khiển I/O .....	131
5.3.3 Lập trình điều khiển bộ định thời (Timer) và bộ đếm (Counter).....	135
5.3.4 Đồng hồ thời gian thực .....	138
5.3.5 Bộ điều chế độ rộng xung .....	141
5.3.6 Điều khiển ngắt.....	143
5.3.7 Giao tiếp UART.....	144
5.3.8 Giao tiếp I2C.....	148
5.3.9 Giao tiếp SPI.....	151
5.3.10 Thiết kế điều khiển giao tiếp với thiết bị tương tự: ADC, DAC.....	154
5.3.11 Vòng khóa pha.....	158
5.4 Thiết lập hệ điều hành nhúng trên nền ARM.....	159
5.4.1 Firmware và Bootloader .....	159
5.4.2 Hệ thống file (Filesystem).....	161
5.4.3 Thiết lập nhân (kernel).....	161
Tài liệu tham khảo .....	166

# CHƯƠNG 1- GIỚI THIỆU CHUNG VỀ HỆ THỐNG NHÚNG

## 1.1 Khái niệm Hệ thống nhúng (Embedded system)

Hệ thống nhúng (Embedded system) là một thuật ngữ để chỉ một hệ thống có khả năng tự trị được nhúng vào trong một môi trường hay một hệ thống mẹ. Đó là các hệ thống tích hợp cả phần cứng và phần mềm phục vụ các bài toán chuyên dụng trong nhiều lĩnh vực công nghiệp: điện tử, viễn thông, công nghệ thông tin, tự động hoá điều khiển, quan trắc và truyền tin. Đặc điểm của các hệ thống nhúng là hoạt động ổn định và có tính năng tự động hoá cao.

Hệ thống nhúng thường được thiết kế để thực hiện một chức năng chuyên biệt nào đó. Khác với các máy tính đa chức năng, chẳng hạn như máy tính cá nhân, một hệ thống nhúng chỉ thực hiện một hoặc một vài chức năng nhất định, thường đi kèm với những yêu cầu cụ thể và bao gồm một số thiết bị máy móc và phần cứng chuyên dụng mà ta không tìm thấy trong một máy tính đa năng nói chung. Vì hệ thống chỉ được xây dựng cho một số nhiệm vụ nhất định nên các nhà thiết kế có thể tối ưu hóa nó nhằm giảm thiểu kích thước và chi phí sản xuất. Các hệ thống nhúng thường được sản xuất hàng loạt với số lượng lớn.

Hệ thống nhúng rất đa dạng, phong phú về chủng loại. Đó có thể là những thiết bị cầm tay nhỏ gọn như đồng hồ kỹ thuật số và máy chơi nhạc MP3, hoặc những sản phẩm lớn như đèn giao thông, bộ kiểm soát trong nhà máy hoặc hệ thống kiểm soát các máy năng lượng hạt nhân. Xét về độ phức tạp, hệ thống nhúng có thể rất đơn giản với một vi điều khiển hoặc rất phức tạp với nhiều đơn vị, các thiết bị ngoại vi và mạng lưới được nằm gọn trong một lớp vỏ máy lớn.

Các thiết bị PDA hoặc máy tính cầm tay cũng có một số đặc điểm tương tự với hệ thống nhúng như các hệ điều hành hoặc vi xử lý điều khiển chúng nhưng các thiết bị này không phải là hệ thống nhúng thật sự bởi chúng là các thiết bị đa năng, cho phép sử dụng nhiều ứng dụng và kết nối đến nhiều thiết bị ngoại vi.

Cho đến nay, khái niệm hệ thống nhúng được nhiều người chấp nhận nhất là: hệ thống thực hiện một số chức năng đặc biệt có sử dụng vi xử lý. Không có hệ thống nhúng nào chỉ có phần mềm.

## 1.2 Lịch sử phát triển của hệ thống nhúng

Hệ thống nhúng đầu tiên là Apollo Guidance Computer (Máy tính Dẫn đường Apollo) được phát triển bởi Charles Stark Draper tại phòng thí nghiệm của trường đại học MIT. Hệ thống nhúng được sản xuất hàng loạt đầu tiên là máy hướng dẫn cho tên lửa quân sự vào năm 1961. Nó là máy hướng dẫn Autonetics D-17, được xây dựng sử dụng

những bóng bán dẫn và một đĩa cứng để duy trì bộ nhớ. Khi Minuteman II được đưa vào sản xuất năm 1996, D-17 đã được thay thế với một máy tính mới sử dụng mạch tích hợp. Tính năng thiết kế chủ yếu của máy tính Minuteman là nó đưa ra thuật toán có thể lập trình lại sau đó để làm cho tên lửa chính xác hơn, và máy tính có thể kiểm tra tên lửa, giảm trọng lượng của cáp điện và đầu nối điện.

Từ những ứng dụng đầu tiên vào những năm 1960, các hệ thống nhúng đã giảm giá và phát triển mạnh mẽ về khả năng xử lý. Bộ vi xử lý đầu tiên hướng đến người tiêu dùng là Intel 4004, được phát minh phục vụ máy tính điện tử và những hệ thống nhỏ khác. Tuy nhiên nó vẫn cần các chip nhớ ngoài và những hỗ trợ khác. Vào những năm cuối 1970, những bộ xử lý 8 bit đã được sản xuất, nhưng nhìn chung chúng vẫn cần đến những chip nhớ bên ngoài.

Vào giữa thập niên 80, kỹ thuật mạch tích hợp đã đạt trình độ cao dẫn đến nhiều thành phần có thể đưa vào một chip xử lý. Các bộ vi xử lý được gọi là các vi điều khiển và được chấp nhận rộng rãi. Với giá cả thấp, các vi điều khiển đã trở nên rất hấp dẫn để xây dựng các hệ thống chuyên dụng. Đã có một sự bùng nổ về số lượng các hệ thống nhúng trong tất cả các lĩnh vực thị trường và số các nhà đầu tư sản xuất theo hướng này. Ví dụ, rất nhiều chip xử lý đặc biệt xuất hiện với nhiều giao diện lập trình hơn là kiểu song song truyền thống để kết nối các vi xử lý. Vào cuối những năm 80, các hệ thống nhúng đã trở nên phổ biến trong hầu hết các thiết bị điện tử và khuynh hướng này vẫn còn tiếp tục cho đến nay.

### **1.3 Các đặc điểm hệ thống nhúng**

Hệ thống nhúng thường có một số đặc điểm chung như sau:

- Các hệ thống nhúng được thiết kế để thực hiện một số nhiệm vụ chuyên dụng chứ không phải đóng vai trò là các hệ thống máy tính đa chức năng. Một số hệ thống đòi hỏi ràng buộc về tính hoạt động thời gian thực để đảm bảo độ an toàn và tính ứng dụng; một số hệ thống không đòi hỏi hoặc ràng buộc chặt chẽ, cho phép đơn giản hóa hệ thống phần cứng để giảm thiểu chi phí sản xuất.
- Một hệ thống nhúng thường không phải là một khối riêng biệt mà là một hệ thống phức tạp nằm trong thiết bị mà nó điều khiển.
- Phần mềm được viết cho các hệ thống nhúng được gọi là firmware và được lưu trữ trong các chip bộ nhớ chỉ đọc (read-only memory) hoặc bộ nhớ flash chứ không phải là trong một ổ đĩa. Phần mềm thường chạy với số tài nguyên phần cứng hạn chế: không có bàn phím, màn hình hoặc có nhưng với kích thước nhỏ, bộ nhớ hạn chế. Sau đây, ta sẽ đi sâu, xem xét cụ thể đặc điểm của các thành phần của hệ thống nhúng.

### 1.3.1 Giao diện

Các hệ thống nhúng có thể không có giao diện (đối với những hệ thống đơn nhiệm) hoặc có đầy đủ giao diện giao tiếp với người dùng tương tự như các hệ điều hành trong các thiết bị để bàn. Đối với các hệ thống đơn giản, thiết bị nhúng sử dụng nút bấm, đèn LED và hiển thị chữ cỡ nhỏ hoặc chỉ hiển thị số, thường đi kèm với một hệ thống menu đơn giản.

Còn trong một hệ thống phức tạp hơn, một màn hình đồ họa, cảm ứng hoặc có các nút bấm ở lề màn hình cho phép thực hiện các thao tác phức tạp mà tối thiểu hóa được khoảng không gian cần sử dụng, ý nghĩa của các nút bấm có thể thay đổi theo màn hình và các lựa chọn. Các hệ thống nhúng thường có một màn hình với một nút bấm dạng cần điều khiển (joystick button). Sự phát triển mạnh mẽ của mạng toàn cầu đã mang đến cho những nhà thiết kế hệ nhúng một lựa chọn mới là sử dụng một giao diện web thông qua việc kết nối mạng. Điều này có thể giúp tránh được chi phí cho những màn hình phức tạp nhưng đồng thời vẫn cung cấp khả năng hiển thị và nhập liệu phức tạp khi cần đến, thông qua một máy tính khác. Điều này là hết sức hữu dụng đối với các thiết bị điều khiển từ xa, cài đặt vĩnh viễn. Ví dụ, các router là các thiết bị đã ứng dụng tiện ích này.

### 1.3.2 Kiến trúc CPU

Các bộ xử lý trong hệ thống nhúng có thể được chia thành hai loại: vi xử lý và vi điều khiển. Các vi điều khiển thường có các thiết bị ngoại vi được tích hợp trên chip nhằm giảm kích thước của hệ thống. Có rất nhiều loại kiến trúc CPU được sử dụng trong thiết kế hệ nhúng như ARM, MIPS, Coldfire/68k, PowerPC, x86, PIC, 8051, Atmel AVR, Renesas H8, SH, V850, FR-V, M32R, Z80, Z8 ... Điều này trái ngược với các loại máy tính để bàn, thường bị hạn chế với một vài kiến trúc máy tính nhất định. Các hệ thống nhúng có kích thước nhỏ và được thiết kế để hoạt động trong môi trường công nghiệp thường lựa chọn PC/104 và PC/104++ làm nền tảng. Những hệ thống này thường sử dụng DOS, Linux, NetBSD hoặc các hệ điều hành nhúng thời gian thực như QNX hay VxWorks. Còn các hệ thống nhúng có kích thước rất lớn thường sử dụng một cấu hình thông dụng là hệ thống on chip (System on a chip – SoC), một bảng mạch tích hợp cho một ứng dụng cụ thể (an application-specific integrated circuit – ASIC). Sau đó nhân CPU được mua và thêm vào như một phần của thiết kế chip. Một chiến lược tương tự là sử dụng FPGA (field-programmable gate array) và lập trình cho nó với những thành phần nguyên lý thiết kế bao gồm cả CPU.

#### *Thiết bị ngoại vi*

Hệ thống nhúng giao tiếp với bên ngoài thông qua các thiết bị ngoại vi, ví dụ như:

- Serial Communication Interfaces (SCI): RS-232, RS-422, RS-485...
- Synchronous Serial Communication Interface: I2C, JTAG, SPI, SSC và ESSI
- Universal Serial Bus (USB)
- Networks: Controller Area Network, LonWorks...
- Bộ định thời: PLL(s), Capture/Compare và Time Processing Units
- Discrete IO: General Purpose Input/Output (GPIO)

### ***Công cụ phát triển***

Tương tự như các sản phẩm phần mềm khác, phần mềm hệ thống nhúng cũng được phát triển nhờ việc sử dụng các trình biên dịch (compilers), chương trình dịch hợp ngữ (assembler) hoặc các công cụ gỡ rối (debuggers). Tuy nhiên, các nhà thiết kế hệ thống nhúng có thể sử dụng một số công cụ chuyên dụng như:

- Bộ gỡ rối mạch hoặc các chương trình mô phỏng (emulator).
- Tiện ích để thêm các giá trị checksum hoặc CRC vào chương trình, giúp hệ thống nhúng có thể kiểm tra tính hợp lệ của chương trình đó.
- Đối với các hệ thống xử lý tín hiệu số, người phát triển hệ thống có thể sử dụng phần mềm workbench như MathCad hoặc Mathematica để mô phỏng các phép toán.
- Các trình biên dịch và trình liên kết (linker) chuyên dụng được sử dụng để tối ưu hóa một thiết bị phần cứng.
- Một hệ thống nhúng có thể có ngôn ngữ lập trình và công cụ thiết kế riêng của nó hoặc sử dụng và cải tiến từ một ngôn ngữ đã có sẵn.

Các công cụ phần mềm có thể được tạo ra bởi các công ty phần mềm chuyên dụng về hệ thống nhúng hoặc chuyển đổi từ các công cụ phát triển phần mềm GNU. Đôi khi, các công cụ phát triển dành cho máy tính cá nhân cũng được sử dụng nếu bộ xử lý của hệ thống nhúng đó gần giống với bộ xử lý của một máy PC thông dụng.

### ***Độ tin cậy***

Các hệ thống nhúng thường nằm trong các cỗ máy được kỳ vọng là sẽ chạy hàng năm trời liên tục mà không bị lỗi hoặc có thể khôi phục hệ thống khi gặp lỗi. Vì thế, các phần mềm hệ thống nhúng được phát triển và kiểm thử một cách cẩn thận hơn là phần mềm cho máy tính cá nhân. Ngoài ra, các thiết bị rời không đáng tin cậy như ổ đĩa, công tắc hoặc nút bấm thường bị hạn chế sử dụng. Việc khôi phục hệ thống khi gặp lỗi có thể



được thực hiện bằng cách sử dụng các kỹ thuật như watchdog timer – nếu phần mềm không đều đặn nhận được các tín hiệu watchdog định kì thì hệ thống sẽ bị khởi động lại.

### ***Một số vấn đề cụ thể về độ tin cậy như:***

- Hệ thống không thể ngừng để sửa chữa một cách an toàn, ví dụ như ở các hệ thống không gian, hệ thống dây cáp dưới đáy biển, các đèn hiệu dẫn đường,... Giải pháp đưa ra là chuyển sang sử dụng các hệ thống con dự trữ hoặc các phần mềm cung cấp một phần chức năng.
- Hệ thống phải được chạy liên tục vì tính an toàn, ví dụ như các thiết bị dẫn đường máy bay, thiết bị kiểm soát độ an toàn trong các nhà máy hóa chất,... Giải pháp đưa ra là lựa chọn backup hệ thống.
- Nếu hệ thống ngừng hoạt động sẽ gây tổn thất rất nhiều tiền của ví dụ như các dịch vụ buôn bán tự động, hệ thống chuyển tiền, hệ thống kiểm soát trong các nhà máy ...

### **Kiến trúc điển hình của hệ thống nhúng**

Kiến trúc của một hệ thống nhúng là một sự trừu tượng hóa thiết bị nhúng, điều đó có nghĩa là một sự tổng quát hóa của một hệ thống mà không chỉ rõ các thông tin thực thi chi tiết của hệ thống như mã nguồn hoặc thiết kế mạch phần cứng.

Các thành phần phần cứng và phần mềm ở mức kiến trúc trong một hệ thống nhúng được đại diện bởi các phần tử có tác động lẫn nhau. Các phần tử là đại diện của phần cứng hoặc phần mềm nhưng chi tiết đã được trừu tượng hóa. Do đó, chỉ có thông tin về các mối quan hệ qua lại và các hoạt động của chúng. Các phần tử này có thể được tích hợp bên trong thiết bị nhúng hoặc tồn tại bên ngoài hệ thống nhúng và tương tác với các phần tử bên trong. *Tóm lại, một kiến trúc hệ thống nhúng bao gồm các phần tử của hệ thống nhúng, các phần tử tương tác với một hệ thống nhúng, các tính chất của mỗi phần tử riêng biệt và mối quan hệ tương tác giữa các thành phần.*

Các thông tin ở mức kiến trúc được mô tả theo dạng cấu trúc. Một cấu trúc sẽ bao gồm tập hợp của các phần tử, các tính chất và thông tin về các mối quan hệ qua lại. Do đó, một cấu trúc là một hình ảnh của phần cứng và phần mềm của hệ thống tại thời điểm thiết kế hoặc thời điểm chạy.

Do hệ thống thường có cấu trúc phức tạp, một kiến trúc thường là sự kết hợp của nhiều cấu trúc khác nhau. Tất cả các cấu trúc trong một kiến trúc có mối quan hệ thừa kế qua lại với nhau. Một số kiểu cấu trúc như sau:

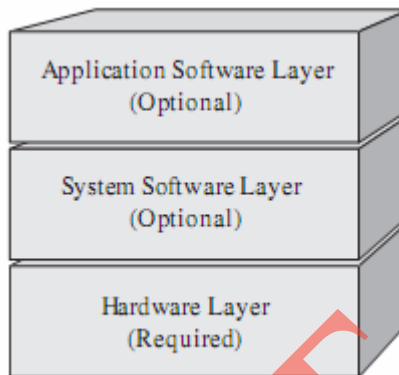
- Cấu trúc theo dạng module: Theo dạng này, các phần tử là các thành phần có chức năng khác nhau của hệ thống (VD như các phần cứng hoặc phần mềm căn bản để cho hệ thống hoạt động được) trong một thiết bị nhúng. Cấu trúc này

thường được trình bày theo dạng lớp (layers), theo các phần mềm dịch vụ cho nhân (kernel services)...

- Cấu trúc theo dạng thành phần và kết nối: Cấu trúc này là sự kết hợp của các thành phần (VD phần cứng, phần mềm, CPU, ) và các kết nối như bus phần cứng, các bản tin của phần mềm, các process trong hệ thống.

### Mô hình hệ thống nhúng

Trên thực tế, có nhiều cấu trúc kiến trúc được sử dụng. Tuy nhiên, ở mức căn bản cao nhất, mô hình hệ thống nhúng thường được trình bày như hình sau:



Hình 1-1. Mô hình chung hệ thống nhúng

Từ mô hình trên, có thể thấy rằng tất cả hệ thống nhúng đều có chung sự tương tự ở mức cao nhất. Cụ thể, chúng đều có các lớp (phần cứng, phần mềm hệ thống và phần mềm ứng dụng). Lớp phần cứng bao gồm các thành phần vật lý trên bo mạch nhúng. Lớp phần mềm hệ thống và phần mềm ứng dụng bao gồm các phần mềm trên hệ thống nhúng.

### Một số kiến trúc phần mềm hệ thống nhúng:

#### *Vòng lặp kiểm soát đơn giản*

Theo thiết kế này, phần mềm được tổ chức thành một vòng lặp đơn giản. Vòng lặp gọi đến các chương trình con, mỗi chương trình con quản lý một phần của hệ thống phần cứng hoặc phần mềm.

#### *Hệ thống ngắt điều khiển*

Các hệ thống nhúng thường được điều khiển bằng các ngắt. Có nghĩa là các tác vụ của hệ thống nhúng được kích hoạt bởi các loại sự kiện khác nhau. Ví dụ, một ngắt có thể được sinh ra bởi một bộ định thời sau một chu kỳ được định nghĩa trước, hoặc bởi sự kiện khi cổng nối tiếp nhận được một byte nào đó.

Loại kiến trúc này thường được sử dụng trong các hệ thống có bộ quản lý sự kiện đơn giản, ngắn gọn và cân độ trễ thấp. Hệ thống này thường thực hiện một tác vụ đơn giản trong một vòng lặp chính. Đôi khi, các tác vụ phức tạp hơn sẽ được thêm vào một cấu trúc hàng đợi trong bộ quản lý ngắt để được vòng lặp xử lý sau đó. Lúc này, hệ thống gần giống với kiểu nhân đa nhiệm với các tiến trình rời rạc.

#### *Đa nhiệm tương tác*

Một hệ thống đa nhiệm không ưu tiên cũng gần giống với kỹ thuật vòng lặp kiểm soát đơn giản ngoại trừ việc vòng lặp này được ẩn giấu thông qua một giao diện lập trình

API. Các nhà lập trình định nghĩa một loạt các nhiệm vụ, mỗi nhiệm vụ chạy trong một môi trường riêng của nó. Khi không cần thực hiện nhiệm vụ đó thì nó gọi đến các tiến trình con tạm nghỉ (bằng cách gọi “pause”, “wait”, “yeild” ...).

Ưu điểm và nhược điểm của loại kiến trúc này cũng giống với kiểm vòng lặp kiểm soát đơn giản. Tuy nhiên, việc thêm một phần mềm mới được thực hiện dễ dàng hơn bằng cách lập trình một tác vụ mới hoặc thêm vào hàng đợi thông dịch (queue-interpreter).

#### *Đa nhiệm ưu tiên*

Ở loại kiến trúc này, hệ thống thường có một đoạn mã ở mức thấp thực hiện việc chuyển đổi giữa các tác vụ khác nhau thông qua một bộ định thời. Đoạn mã này thường nằm ở mức mà hệ thống được coi là có một hệ điều hành và vì thế cũng gặp phải tất cả những phức tạp trong việc quản lý đa nhiệm.

Bất kỳ tác vụ nào có thể phá hủy dữ liệu của một tác vụ khác đều cần phải được tách biệt một cách chính xác. Việc truy cập tới các dữ liệu chia sẻ có thể được quản lý bằng một số kỹ thuật đồng bộ hóa như hàng đợi thông điệp (message queues), semaphores ... Vì những phức tạp nói trên nên một giải pháp thường được đưa ra đó là sử dụng một hệ điều hành thời gian thực. Lúc đó, các nhà lập trình có thể tập trung vào việc phát triển các chức năng của thiết bị chứ không cần quan tâm đến các dịch vụ của hệ điều hành nữa.

#### *Vi nhân (Microkernel) và nhân ngoại (Exokernel)*

Khái niệm vi nhân (microkernel) là một bước tiếp cận gần hơn tới khái niệm hệ điều hành thời gian thực. Lúc này, nhân hệ điều hành thực hiện việc cấp phát bộ nhớ và chuyển CPU cho các luồng thực thi. Còn các tiến trình người dùng sử dụng các chức năng chính như hệ thống file, giao diện mạng lưới, ... Nói chung, kiến trúc này thường được áp dụng trong các hệ thống mà việc chuyển đổi và giao tiếp giữa các tác vụ là nhanh.

Còn nhân ngoại (exokernel) tiến hành giao tiếp hiệu quả bằng cách sử dụng các lời gọi chương trình con thông thường. Phần cứng và toàn bộ phần mềm trong hệ thống luôn đáp ứng và có thể được mở rộng bởi các ứng dụng.

#### *Nhân khối (monolithic kernels)*

Trong kiến trúc này, một nhân đầy đủ với các khả năng phức tạp được chuyển đổi để phù hợp với môi trường nhúng. Điều này giúp các nhà lập trình có được một môi trường giống với hệ điều hành trong các máy để bàn như Linux hay Microsoft Windows và vì thế rất thuận lợi cho việc phát triển. Tuy nhiên, nó lại đòi hỏi đáng kể các tài nguyên phần cứng làm tăng chi phí của hệ thống.

Một số loại nhân khối thông dụng là Embedded Linux và Windows CE. Mặc dù chi phí phần cứng tăng lên nhưng loại hệ thống nhúng này đang tăng trưởng rất mạnh, đặc biệt là trong các thiết bị nhúng mạnh như Wireless router hoặc hệ thống định vị GPS. Lý do của điều này là:

- Hệ thống này có cổng để kết nối đến các chip nhúng thông dụng
- Hệ thống cho phép sử dụng lại các đoạn mã sẵn có phổ biến như các trình điều khiển thiết bị, Web Servers, Firewalls, ...

- Việc phát triển hệ thống có thể được tiến hành với một tập nhiều loại đặc tính, chức năng còn sau đó lúc phân phối sản phẩm, hệ thống có thể được cấu hình để loại bỏ một số chức năng không cần thiết. Điều này giúp tiết kiệm được những vùng nhớ mà các chức năng đó chiếm giữ.
- Hệ thống có chế độ người dùng để dễ dàng chạy các ứng dụng và gỡ rối. Nhờ đó, qui trình phát triển được thực hiện dễ dàng hơn và việc lập trình có tính linh động hơn.
- Có nhiều hệ thống nhưng thiếu các yêu cầu chặt chẽ về tính thời gian thực của hệ thống quản lý. Còn một hệ thống như Embedded Linux có tốc độ đủ nhanh để trả lời cho nhiều ứng dụng. Các chức năng cần đến sự phản ứng nhanh cũng có thể được đặt vào phần cứng.

## Phân loại các hệ thống nhúng

Hiện nay có nhiều cách phân loại hệ thống nhúng khác nhau. Tùy theo cách phân chia có thể phân loại các hệ thống nhúng như sau:

- Hệ thống phân phối và hệ thống không phân phối:

Các hệ thống không phân phối thường hoạt động riêng biệt. Ngược lại, hệ thống phân phối phối kết các thiết bị được kết nối với nhau, ví dụ như các vi điều khiển nhúng, các thiết bị mạng, các máy tính nhúng, các hệ thống cảm biến không dây. Sự kết hợp các thiết bị nhúng này thường được gặp trong các thiết bị điều khiển ô tô hoặc hàng không, các thiết bị giám sát môi trường hoặc các thiết bị quản lý dây chuyền sản xuất tự động.

- Hệ thống dữ liệu và hệ thống điều khiển.

Các hệ thống dữ liệu dùng để xử lý dữ liệu, xử lý hoặc cung cấp các dữ liệu thông tin cần thiết khi có yêu cầu. Còn các hệ thống điều khiển dùng để điều khiển hệ thống, điều khiển các quy trình trong sản xuất hoặc trong các thiết bị.

### 1.4 Phạm vi ứng dụng của hệ thống nhúng

Ngày nay, hệ thống nhúng được ứng dụng rộng rãi ở khắp mọi nơi. Hầu hết những thiết bị điện tử đều là các hệ thống nhúng.

Một số ví dụ như sau:

- Các hệ thống điều khiển ô tô, tàu, máy bay
- Các hệ thống y tế
- Các hệ thống quân sự
- Các hệ thống giám sát, cảnh báo.
- Các hệ thống cảm ứng
- Các thiết bị điện tử dân dụng

Ngoài ra, theo ước tính, mỗi năm lượng phần mềm nhúng được phát triển lớn gấp năm lần lượng phần mềm thường. Đối với phần cứng, đa số CPU được sản xuất là cho thị trường nhúng. Chỉ có một phần nhỏ CPU là dùng trong các hệ thống máy tính.

## 1.5 Các yêu cầu về kỹ năng trong thiết kế hệ thống nhúng

### Tổng quan về thiết kế các hệ nhúng

Thiết kế các hệ thống nhúng là thiết kế phần cứng và phần mềm phối hợp bao gồm những bước sau:

- Mô hình hoá hệ thống: Mô tả các khối chức năng với các đặc tính và thuật toán xử lý.
- Chi tiết hoá các khối chức năng
- Phân bố chức năng cho phần cứng và mềm (HW-SW)
- Đồng bộ hoạt động của hệ thống
- Cài đặt các chức năng thiết kế vào phần cứng (hardware) và phần mềm (software) hoặc phần nhào (firmware).

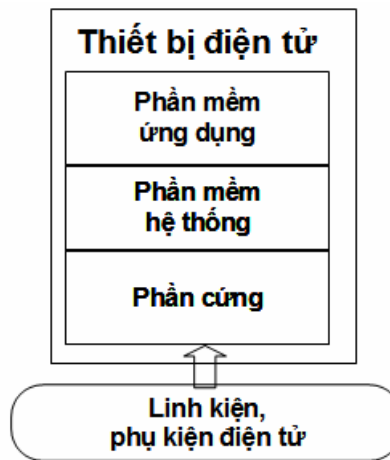
Cách thiết kế cổ điển là các chức năng phần mềm (SW) và phần cứng (HW) được xác định trước rồi sau đó các bước thiết kế chi tiết được tiến hành một cách độc lập ở hai khối. Hiện nay đa số các hệ thống tự động hoá thiết kế (CAD) thường dành cho thiết kế phần cứng. Các hệ thống nhúng sử dụng đồng thời nhiều công nghệ như vi xử lý, DSP, mạng và các chuẩn phối ghép, protocol, do vậy xu thế thiết kế các hệ nhúng hiện nay đòi hỏi có khả năng thay đổi mềm dẻo hơn trong quá trình thiết kế 2 phần HW và SW. Để có được thiết kế cuối cùng tối ưu quá trình thiết kế SW và HW phải phối hợp với nhau chặt chẽ và có thể thay đổi sau mỗi lần thử chức năng hoạt động tổng hợp

Thiết kế các hệ nhúng đòi hỏi kiến thức đa ngành về điện tử, xử lý tín hiệu, vi xử lý, thuật điều khiển và lập trình thời gian thực.

Đối với mỗi một ngành nghề đào tạo, mục đích cơ bản đó là xây dựng tập các kỹ năng cần phát triển cho đối tượng được đào tạo, để sau khi kết thúc khóa đào tạo, đối tượng được đào tạo về cơ bản phải đạt được những kỹ năng đã đặt ra. Tập các kỹ năng này được xây dựng dựa trên tính chất, đặc trưng của nội dung đào tạo.

Trong thiết kế hệ thống nhúng, nội dung đào tạo liên quan đến việc thiết kế các hệ thống nhúng hay các thiết bị điện tử thông minh hoặc còn được gọi tắt là hệ thống được điều khiển bởi các linh kiện bán dẫn khả trình như bộ vi xử lý – microprocessor, bộ vi điều khiển – microcontroller, FPGA, CPLD, ....

Để đảm bảo tính đồng nhất, từ giờ chúng ta sẽ gọi tên gọi chung cho các thiết bị điện tử hay các hệ thống nhúng là các hệ thống nhúng.



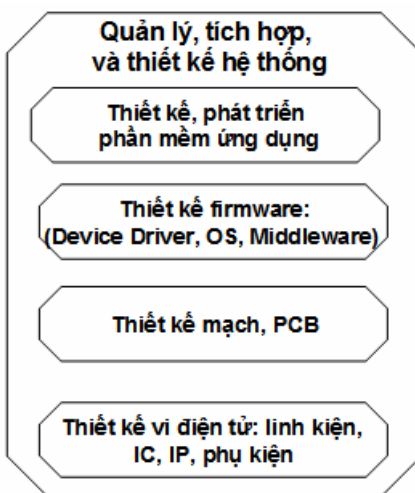
Hình 1-2 : Cấu trúc của thiết bị điện tử

Cấu trúc của một hệ thống nhúng về cơ bản được mô tả như trong hình 1, nó gồm các thành phần:

- Phần mềm ứng dụng : là các phần mềm được thiết kế để thực thi một tác vụ thực tế dựa trên tài nguyên do nền phần cứng cung cấp. Ví dụ như các phần mềm chơi nhạc trên các máy MP3, ứng dụng game trên các máy PS2, bộ công cụ Microsoft Office trên các PC, ..
- Phần mềm hệ thống : ví dụ như các hệ điều hành Windows, Linux, Unix, và các chương trình hỗ trợ như trình biên dịch, loader, linker, debugger, .. giúp quản lý các tài nguyên phần cứng ở mức thấp. Về cơ bản nó cho phép các phần của hệ thống làm việc với nhau, cấp phát các tài nguyên cho các phần mềm ứng dụng.
- Phần cứng : chỉ các thành phần vật lý của hệ thống, được cấu tạo về cơ bản từ các linh kiện vật lý. Ví dụ như với một PC, phần cứng gồm các thành phần bo mạch chủ, Ram, ổ cứng, nguồn nuôi, ... Các bo mạch chủ thì được cấu tạo từ các linh kiện bán dẫn, các linh kiện thụ động như điện trở, tụ điện, cuộn cảm,...

Từ cấu trúc trên của hệ thống nhúng, có thể phân chia việc thiết kế một hệ thống thành các mảng công việc như trong hình 1-3:





Hình 1-3 : Các mảng công việc trong thiết kế hệ thống nhúng

Nó gồm các mảng khác nhau đó là :

- Quản lý, tích hợp và thiết kế hệ thống
- Thiết kế, phát triển phần mềm ứng dụng
- Thiết kế firmware (device driver, OS, middleware)
- Thiết kế mạch, PCB
- Thiết kế vi điện tử : linh kiện, IC, IP, Phụ kiện

Đối với từng mảng công việc, đòi hỏi người thực hiện cần có các kỹ năng tương ứng.

### 1.5.1 Quản lý, tích hợp, thiết kế hệ thống:

Đây là một trong những mặt có vai trò quan trọng đối với sự thành công của một dự án thiết kế một hệ thống nhúng. Nó bao gồm các công việc:

- Hoạch định các yêu cầu của hệ thống, từ đó xây dựng kết cấu chung của hệ thống.
- Xác định các tài nguyên có sẵn bao gồm nhân lực và vật lực.
- Lên kế hoạch các bước thực hiện.
- Giám sát quá trình thực hiện.

.....

Để đảm nhiệm được công việc này, yêu cầu các kỹ năng sinh viên cần có:

- Có kiến thức về quản lý dự án thiết kế, kiến trúc hệ thống : bao gồm kiến thức các mô hình trong phân tích hệ thống và triển khai các dự án thiết kế; kiến trúc hệ thống nhúng, có cái nhìn tổng quan với các khía cạnh của hệ thống nhúng gồm phần mềm, phần cứng; ..
- Có kỹ năng quản lý nhóm, đây là một yêu cầu quan trọng có tính chất quyết định
- Có khả năng sáng tạo, ứng dụng các phương thức quản lý tiên tiến nhằm đạt hiệu năng cao trong thiết kế hệ thống.

- Kỹ năng về phân tách, tích hợp hệ thống, kiểm tra hệ thống,... đảm bảo hệ thống hoạt động ổn định, đáp ứng được các yêu cầu về hiệu năng, giá thành, tuổi thọ,....

### 1.5.2 Thiết kế, phát triển phần mềm ứng dụng

Đối với các hệ thống nhúng, điều quan trọng là ứng dụng thực tiễn của nó trong đời sống, quan trọng hơn, chức năng được quyết định bởi phần mềm ứng dụng được cài đặt trên hệ thống. Vì vậy, các kỹ năng trong thiết kế, phát triển phần mềm ứng dụng là không thể thiếu đối với thiết kế hệ thống nhúng hiện nay. Các kỹ năng yêu cầu gồm:

- Có kiến thức đối với khoa học lập trình : bao gồm kiến thức về xây dựng cấu trúc dữ liệu và giải thuật; cơ sở dữ liệu; các phương thức lập trình hướng cấu trúc , hướng đối tượng; đồ họa; đa phương tiện; kiến thức liên quan đến xử lý tín hiệu...
- Kỹ năng thực thi các ứng dụng trên các ngôn ngữ lập trình : yêu cầu có kỹ năng về các ngôn ngữ lập trình ứng dụng như C/C++, VC++, VB, Delphi, ASP, PHP, JAVA.
- Có kiến thức lập trình trên các nền tảng khác nhau : PC, smartphone, ...

### 1.5.3 Thiết kế firmware

Điều quan trọng của các ứng dụng hệ thống nhúng đó là tận dụng được các tài nguyên của hệ thống : CPU, memory, các ngoại vi, các giao diện,... Hầu hết quá trình xử lý bên trong của hệ thống là sự giao tiếp giữa các thành phần phần cứng bên trong. Vì vậy đòi hỏi xây dựng firmware để quản lý các tài nguyên này một cách hữu hiệu, cung cấp các giao diện truy cập các tài nguyên này cho lớp cao hơn (lớp ứng dụng).

Nó gồm các lĩnh vực về thiết kế trình điều khiển thiết bị - Device Driver, thiết kế OS – hệ điều hành, thiết kế phần mềm Middleware.

Để thực hiện được, yêu cầu các kỹ năng sinh viên cần có:

- Hiểu biết về hệ điều hành, hệ điều hành thời gian thực,.. như khái niệm hệ điều hành, các thành phần hệ điều hành, các khái niệm về scheduling, process, thread, inter-process,....
- Kiến thức về kiến trúc máy tính, các kỹ thuật ứng dụng trong kiến trúc máy tính như pipeline, DMA, caching, buffering,...
- Kỹ năng lập trình hệ thống, hệ thống nhúng, lập trình vi xử lý, bảo mật, mạng...
- Thành thạo các ngôn ngữ lập trình hệ thống như assembly, C/C++; các ngôn ngữ mô tả phần cứng như VHDL, Verilog.
- Sử dụng thành thạo các công cụ phần mềm hỗ trợ lập trình firmware trình biên dịch, IDE, ...
- Với xu thế hiện nay, các kiến thức về phần mềm mã nguồn mở, hệ điều hành mã nguồn mở sẽ đem lại những lợi ích to lớn trong phát triển phần mềm hệ thống nhúng, ngoài vấn đề về giá thành, khi sử dụng các phần mềm mã nguồn mở, sinh viên sẽ được sự hỗ trợ to lớn từ cộng đồng mã nguồn mở, các kỹ thuật lập trình tiên tiến trong các phần mềm... nên đây cũng là một kỹ năng sinh viên nên chú ý phát triển.



#### 1.5.4 Thiết kế mạch, PCB:

Phần cứng có thể coi như phần xác của hệ thống, cũng là một phần không thể thiếu của hệ thống nhúng, và là kỹ năng yêu cầu bắt buộc đối với các sinh viên điện tử viễn thông. Nó gồm các kỹ năng về:

- Hiểu biết về mạch điện tử, phần cứng vi xử lý, vi điều khiển, IC chức năng, FPGA, CPLD ... , khả năng kết hợp các thành phần để tạo lên 1 hệ thống hoàn chỉnh.
- Các khái niệm trong thiết kế mạch PCB như footprint, layer, path, SMD, SMT..
- Có kiến thức về thiết kế các mạch ổn định, khả năng chống nhiễu, bố trí hợp lý, khoa học, và thẩm mỹ. Đặc biệt là kinh nghiệm làm việc với các mạch hoạt động ở tần số cao.
- Sử dụng thành thạo các công cụ hỗ trợ thiết kế mạch , mô phỏng mạch như Altium, ISE, proteus,..

#### 1.5.5 Thiết kế vi điện tử : linh kiện, IP, IC, phụ kiện

Vi điện tử là các vấn đề liên quan đến nghiên cứu, chế tạo các linh kiện điện tử. Những linh kiện này làm từ các chất bán dẫn. Chúng là những thành phần cơ bản trong các thiết kế điện tử như transistor, tụ điện, điện trở, diode, IC, ...

Có thể nói các linh kiện, IP, IC, phụ kiện,... là những “viên gạch” xây lên “ngôi nhà” phần cứng hệ thống. Kiến thức thiết kế ở mặt này là những kiến thức cơ bản, nền tảng cho các thiết kế nâng cao hơn trong thiết kế phần cứng.

Nó gồm các kỹ năng được yêu cầu :

- Kiến thức về vật lý bán dẫn, nguyên lý mạch tích hợp tương tự, số, mạch RF và cao tần, điện tử ứng dụng...
- Các công nghệ chế tạo chất bán dẫn như NMOS, MOSFET, CMOS, ....
- Thành thạo thiết kế layout, ASIC, VLSI .. sử dụng các công cụ như MentorGraphic, Cadence, ADS, ...

Tổng kết lại, có thể tóm tắt các kỹ năng yêu cầu đối với từng mặt trong thiết kế hệ thống nhúng như trong bảng 1-1.

Thiết kế điện tử	Yêu cầu kỹ năng
Quản lý, tích hợp, và thiết kế hệ thống	<ul style="list-style-type: none"> <li>• Có kiến thức về quản lý dự án thiết kế, kiến trúc hệ thống</li> <li>• Có kỹ năng làm việc theo nhóm, khả năng sáng tạo</li> <li>• Kỹ thuật phân tách, tích hợp hệ thống, kiểm tra hệ thống ...</li> </ul>
Thiết kế, phát triển phần mềm ứng dụng	<ul style="list-style-type: none"> <li>• Hiểu biết về cấu trúc dữ liệu, giải thuật, CSDL, KTLT cấu trúc, hướng đối tượng, đồ họa, multimedia, xử lý tín hiệu,...</li> <li>• Thành thạo kỹ thuật lập trình, ngôn ngữ lập trình: C/C++, VC++, VB, Delphi, ASP, PHP, JAVA, ...</li> </ul>
Thiết kế firmware: (Device Driver, OS, Mirdleware)	<ul style="list-style-type: none"> <li>• Hiểu biết về hệ điều hành thời gian thực, lập trình hệ thống kiến trúc máy tính, hệ thống nhúng, VXL, bảo mật, network...</li> <li>• Thành thạo về sử dụng các công cụ IDE, lập trình C/C++, Linux, Assembler, Embedded C, HDL (VHDL, Verilog), ...</li> </ul>
Thiết kế mạch, PCB	<ul style="list-style-type: none"> <li>• Hiểu biết về mạch điện tử, phần cứng vi xử lý, vi điều khiển, IC chức năng, FPGA, linh kiện điện tử, footprint, SMD, SMT...</li> <li>• Thành thạo các công cụ thiết kế Front-end EDA như: Altium, ISE, Proteus, ...</li> </ul>
Thiết kế vi điện tử: linh kiện, IP, IC, phụ kiện	<ul style="list-style-type: none"> <li>• Hiểu biết về vật lý bán dẫn, nguyên lý mạch tích hợp tương tự, số, mạch RF và cao tần, điện tử ứng dụng ...</li> <li>• Thành thạo thiết kế layout, ASIC, VLSI sử dụng các công cụ back-end EDA như: MentorGraphic, Cadence, ADS ...</li> </ul>

Bảng 1-1 : Các yêu cầu về kỹ năng tương ứng

### **Câu hỏi ôn tập**

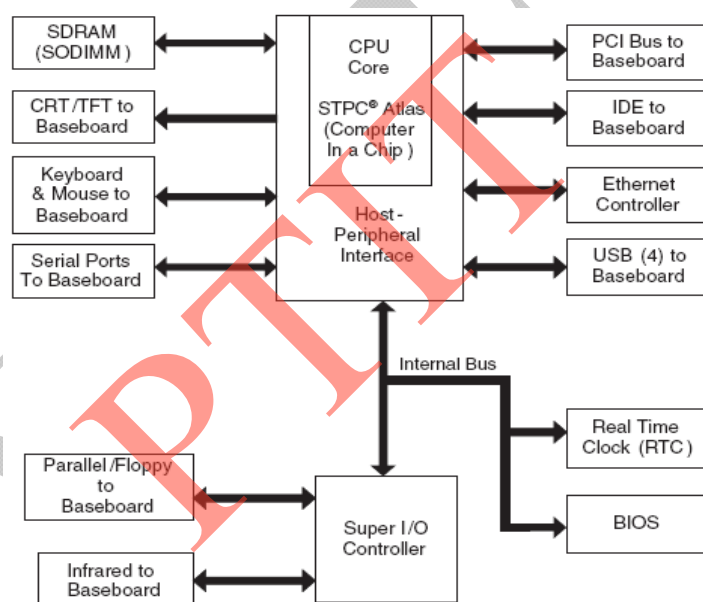
1. Lấy ví dụ 3 thị trường sử dụng hệ thống nhúng  
Mỗi thị trường lấy 5 ví dụ thiết bị.
2. Các đặc tính của hệ thống nhúng bao gồm những đặc tính nào
3. Nêu các thành phần căn bản của kiến trúc hệ thống nhúng
4. Các kiến trúc CPU nhúng căn bản bao gồm những kiến trúc nào
5. Liệt kê các giao tiếp sử dụng trong hệ thống nhúng và giải thích hoạt động của chúng
6. Liệt kê các loại thiết bị lưu trữ thường sử dụng trong hệ thống nhúng.
7. So sánh các loại thiết bị lưu trữ thường sử dụng trong hệ thống nhúng.
8. Để phát triển một hệ thống nhúng, cần những kỹ năng nào?

## CHƯƠNG 2: CÁC THÀNH PHẦN HỆ THỐNG

### 2.1 Các thành phần phần cứng

#### 2.1.1 Bộ xử lý nhúng.

Bộ xử lý là đơn vị chức năng chính của một hệ thống nhúng, và chịu trách nhiệm trong việc xử lý lệnh và dữ liệu. Một thiết bị điện tử có chứa ít nhất một bộ xử lý chủ (master processor), có thể bổ sung các bộ xử lý tớ (slave processors) cùng làm việc và điều khiển bởi bộ xử lý chủ. Slave processors có thể tham gia và các chỉ lệnh của master processors hoặc thi hành quản lý bộ nhớ, các bus và các thiết bị vào ra. Trong sơ đồ khối của một board X86, Atlas STPC là một master processor còn bộ điều khiển I/O, bộ điều khiển ethernet là các Slave processor.



Hình 2-1: Bảng mạch Encore 400 của Ampro

Hệ thống nhúng được thiết kế xung quanh bộ xử lý chủ. Các bộ xử lý chủ phức tạp thường được xác định cho dù nó được phân loại là vi xử lý hay vi điều khiển. Thông thường các bộ vi xử lý chứa một lượng nhỏ bộ nhớ tích hợp và thành phần vào ra (I/O), trong khi các bộ vi điều khiển có phần lớn bộ nhớ hệ thống và các thành phần vào ra tích hợp trên chip. Tuy nhiên hãy nhớ rằng các định nghĩa truyền thống này có thể không còn chính xác cho các thiết kế bộ xử lý thời gian gần đây. Ví dụ các bộ xử lý hiện nay ngày càng được tích hợp nhiều lên.

Các bộ xử lý nhúng có thể tách thành các nhóm dựa trên kiến trúc. Khác biệt giữa các nhóm kiến trúc này là tập hợp các chỉ lệnh mã máy mà các bộ xử lý trong các nhóm kiến trúc có thể thực thi. Bộ xử lý được xem như kiến trúc tương tự nhau khi chúng có

thể thực thi cùng tập lệnh. Dưới đây là bảng vài ví dụ về các bộ xử lý thực tế và họ kiến trúc của chúng.

Architecture	Processor	Manufacturer
AMD	Au1xxx	Advanced Micro Devices, ...
ARM	ARM7, ARM9, ...	ARM, ...
C16X	C167CS, C165H, C164CI, ...	Infineon, ...
ColdFire	5282, 5272, 5307, 5407, ...	Motorola/Freescale, ...
I960	I960	Vmetro, ...
M32/R	32170, 32180, 32182, 32192, ...	Renesas/Mitsubishi, ...
M Core	MMC2113, MMC2114, ...	Motorola/Freescale
MIPS32	R3K, R4K, 5K, 16, ...	MTI4kx, IDT, MIPS Technologies, ...
NEC	Vr55xx, Vr54xx, Vr41xx	NEC Corporation, ...
PowerPC	82xx, 74xx, 8xx, 7xx, 6xx, 5xx, 4xx	IBM, Motorola/Freescale, ...
68k	680x0 (68K, 68030, 68040, 68060, ...), 683xx	Motorola/Freescale, ...
SuperH (SH)	SH3 (7702, 7707, 7708, 7709), SH4 (7750)	Hitachi, ...
SHARC	SHARC	Analog Devices, Transtech DSP, Radstone, ...
strongARM	strongARM	Intel, ...
SPARC	UltraSPARC II	Sun Microsystems, ...
TMS320C6xxx	TMS320C6xxx	Texas Instruments, ...
x86	X86 [386, 486, Pentium (II, III, IV)...]	Intel, Transmeta, National Semiconductor, Atlas, ...
TriCore	TriCore1, TriCore2, ...	Infineon, ...

Bảng 2-1: Các bộ xử lý và kiến trúc thực tế

#### 2.1.1.1 Các mô hình kiến trúc tập lệnh (ISA architecture models)

ISA (instruction set architecture) định nghĩa các tính năng như các thao tác (operations) có thể sử dụng bởi người lập trình để tạo chương trình, các toán hạng (operands), lưu trữ (storage), các kiểu định địa chỉ để truy xuất và xử lý toán hạng và xử lý các ngắt. Các tính năng được mô tả chi tiết hơn dưới đây, bởi vì thực thi ISA là một nhân tố xác định đặc điểm của một hệ thống nhúng như là hiệu suất, thời gian thiết kế, chức năng và giá thành.

-Các thao tác.

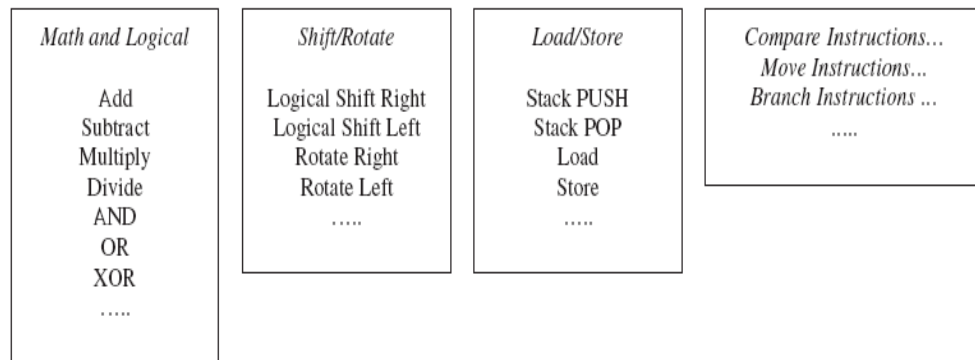
Các thao tác được tạo ra từ một hay nhiều chỉ lệnh. Các bộ xử lý khác nhau có thể thực thi các thao tác tương tự nhau nhờ sử dụng một số lượng và kiểu chỉ lệnh khác nhau.

*Các kiểu thao tác.*

Các thao tác là các chức năng có thể thực hiện trên dữ liệu, chúng thường bao gồm các phép toán số học, các thao tác chuyển dữ liệu từ bộ nhớ hoặc thanh ghi đến các vị trí khác, các rẽ nhánh (có điều kiện hoặc không điều kiện), hoạt động truyền nhận dữ liệu, chuyển đổi phép toán..

Chỉ lệnh trên một bộ vi xử lý cấp thấp phổ biến 8051, bao gồm hơn 100 chỉ lệnh toán học, truyền dữ liệu, phép toán logic, thao tác bit, rẽ nhánh và điều khiển. Trong khi đó ở cấp cao hơn MPC 823 có tập chỉ lệnh lớn hơn 8051, nhưng cùng với nhiều kiểu hoạt động tương tự bao gồm trong 8051 thiết lập cùng với một bổ sung, bao gồm phép toán số nguyên, dấu chấm động, thao tác nạp, lưu trữ, các thao tác rẽ nhánh và điều khiển, thao

tác điều khiển bộ xử lý, đồng bộ bộ nhớ. Ví dụ về các thao tác phổ biến quy định trong một ISA (Hình 2-2).



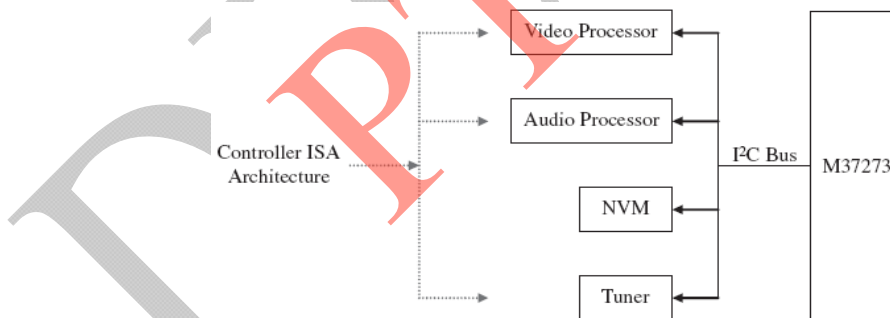
Hình 2-2: Các thao tác ISA đơn giản

### 2.1.1.2 Ứng dụng cụ thể của mô hình ISA

Ứng dụng cụ thể của mô hình ISA xác định các bộ xử lý dành cho các ứng dụng những cụ thể, ví dụ các bộ xử lý cho TV. Một số ứng dụng cụ thể của mô hình ISA được thực hiện bởi các bộ xử lý những, các mô hình phổ biến nhất là:

#### Mô hình điều khiển – Controller model

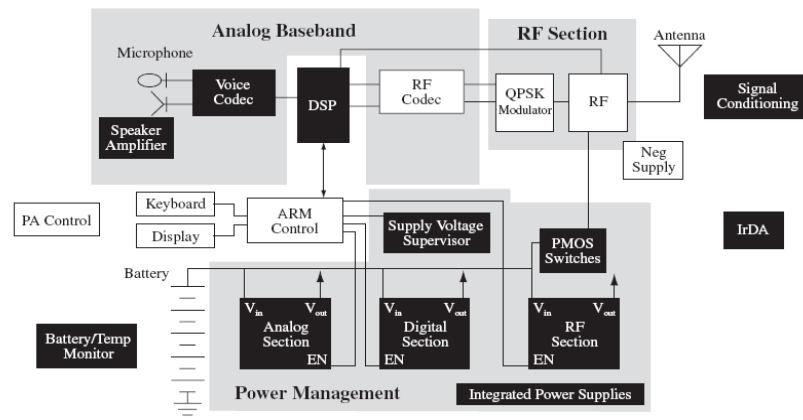
Các ISA controller được thực hiện trong các bộ xử lý mà không yêu cầu các thao tác dữ liệu phức tạp. Ví dụ như bộ xử lý audio và video được sử dụng như các slave processors trong bảng mạch TV, xem Hình 2-3.



Hình 2-3: Ví dụ bảng mạch TV tương tự với sự thi hành bộ điều khiển ISA

#### Mô hình đường dữ liệu – Datapath model

Datapath model được thực hiện trong các bộ xử lý với mục đích thực hiện nhiều lần các phép toán cố định trên các bộ dữ liệu khác nhau. Ví dụ điển hình là bộ xử lý tín hiệu số (DSP), thể hiện trong Hình 2-4.



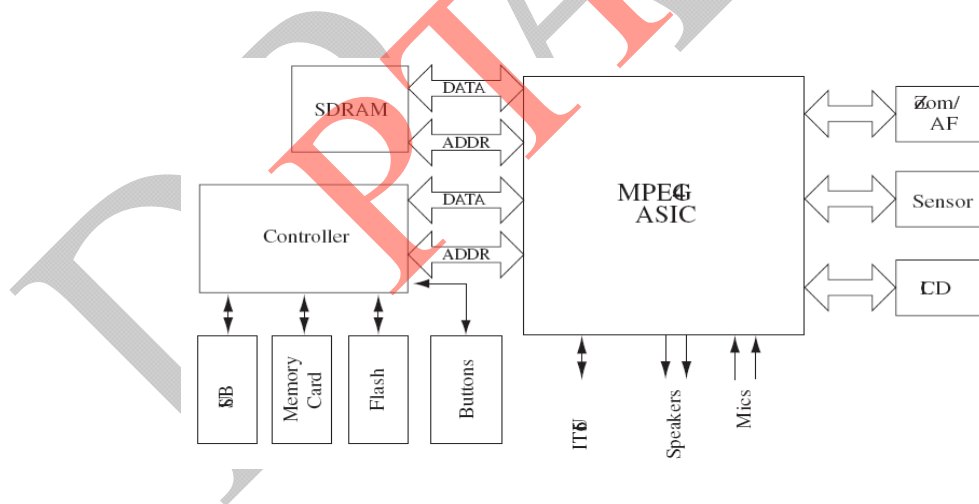
Hình 2-4: Board ví dụ với điện thoại di động thực hiện kỹ thuật số ISA đường dữ liệu

### Finite State Machine with Datapath (FSMD) Model

FSDM ISA là sự kết hợp của Datapath ISA và Controller ISA cho các bộ xử lý, không yêu cầu thực hiện các thao tác dữ liệu phức tạp và lặp lại việc tính toán các phép toán cố định trên các bộ dữ liệu khác nhau.

Ví dụ của FSMD thực hiện các ứng dụng cụ thể:

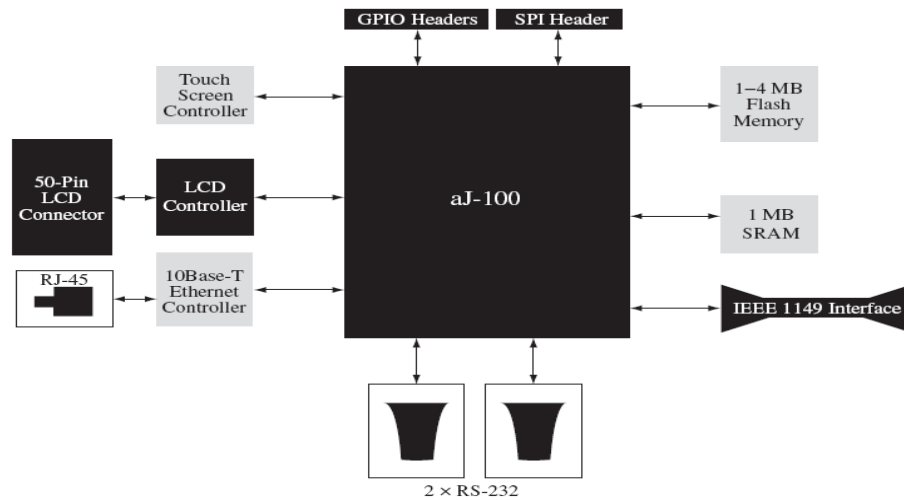
Mạch tích hợp ASICs thể hiện trên hình 2-5, các thiết bị logic khả trình (PLD), FPGA



Hình 2-5: Board ví dụ với máy quay kỹ thuật số FSMD ISA

### Mô hình máy ảo Java (Java Virtual Machine: JVM)

ISA JVM dựa trên một trong các tiêu chuẩn Java Virtual Machine. Trong thế giới thực, JVM có thể được thực hiện trong một hệ thống nhúng thông qua phần cứng. Ví dụ trong Hình 2-6.



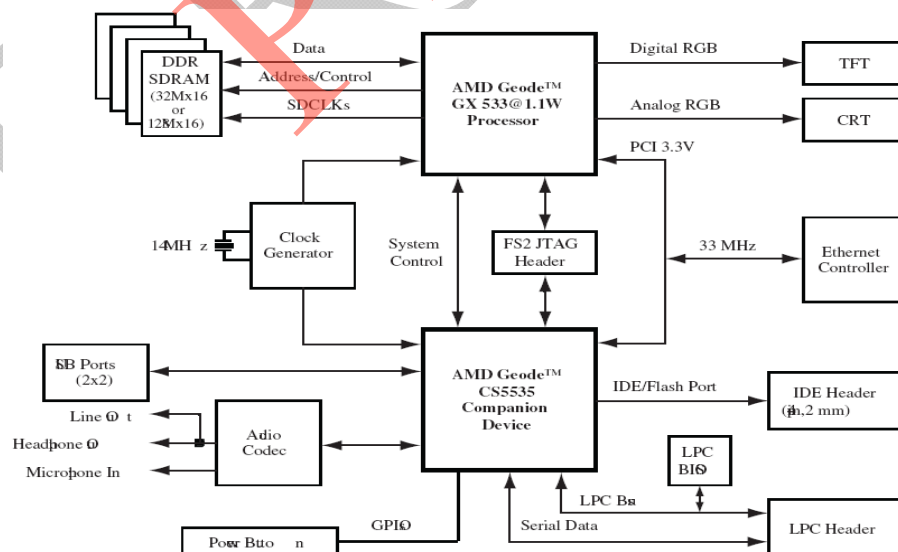
Hình 2-6: ví dụ thực hiện JVM ISA

### 2.1.1.3 Các mô hình ISA đa năng (General-Purpose ISA models)

General-Purpose ISA models thường được thực hiện trong các bộ xử lý với mục đích sử dụng rộng rãi trong nhiều hệ thống chứ không chỉ trong một dạng hệ thống nhưng. Các dạng phổ biến nhất của General-Purpose ISA models trong các bộ xử lý nhúng là:

#### Mô hình tính toán với tập lệnh phức tạp (Complex Instruction Set Computing: CISC)

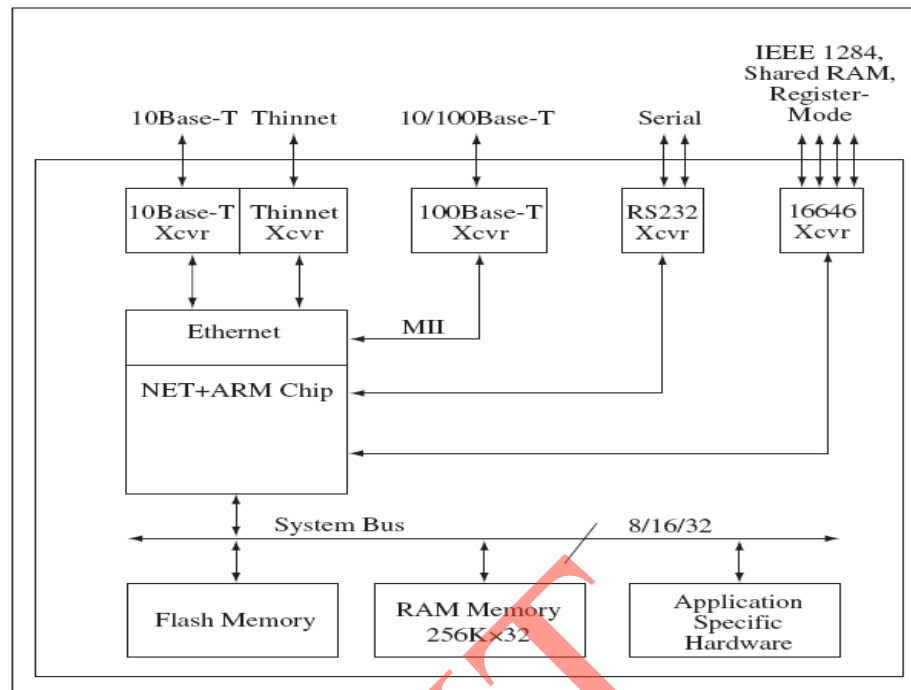
CISC ISA, như tên của nó xác định các hoạt động phức tạp tạo ra từ nhiều lệnh. Ví dụ phổ biến về kiến trúc thực hiện CISC ISA là các họ bộ xử lý intel x86 và Motorola/Freescale 68000.



Hình 2-7: Ví dụ thực hiện CISC ISA



## Mô hình tính toán với tập lệnh rút gọn (Reduced Instruction Set Computing : RISC)



Hình 2-8: Ví dụ thực hiện RISC ISA

Trái ngược với CISC, RISC ISA thường định nghĩa:

- Một kiến trúc đơn giản và/hoặc hoạt động ít hơn gồm ít lệnh hơn.
- Một kiến trúc giảm số chu kỳ hoạt động.
- Các bộ xử lý RISC chỉ có một chu kỳ hoạt động trong khi CISC thường có nhiều chu kỳ.
- ARM, PowerPC, SPARC, MIPS là một vài ví dụ về kiến trúc RISC cơ bản.

Trong lĩnh vực tính toán đa năng, lưu ý rằng nhiều mẫu thiết kế bộ vi xử lý hiện nay thuộc thể loại chip CISC hoặc RISC chủ yếu là do tính kế thừa của chúng. Các bộ xử lý RISC đã trở nên phức tạp hơn, trong khi các bộ xử lý CISC đã trở nên hiệu quả hơn để cạnh tranh với chip RISC đối thủ tương ứng của chúng, do đó làm mờ đi các ranh giới giữa các định nghĩa của một kiến trúc RISC so với một kiến trúc CISC. Về mặt kỹ thuật, những bộ xử lý này có cả hai thuộc tính RISC và CISC, bất kể những định nghĩa của chúng.

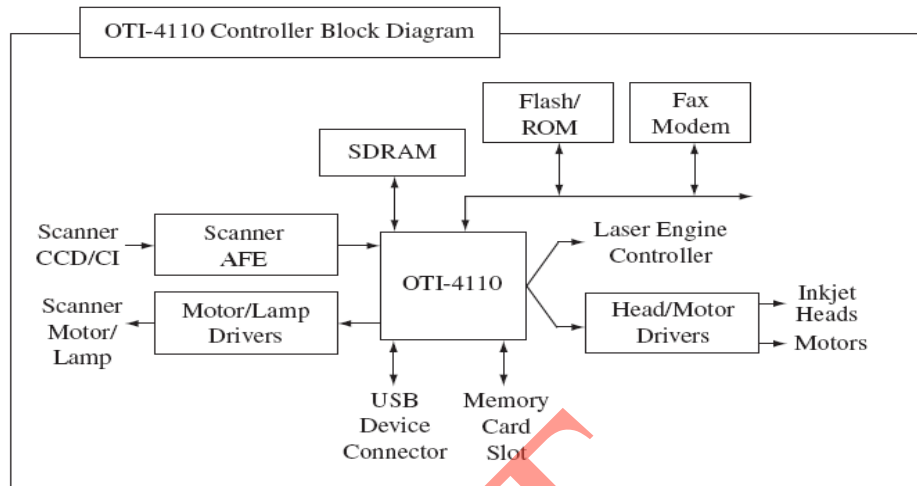
### 2.1.1.4 Các mô hình ISA song song mức lệnh (Instruction-Level Parallelism ISA Models)

Kiến trúc Instruction-level Parallelism ISA tương tự như kiến trúc chung của ISA ngoại trừ việc thực hiện nhiều lệnh song song. Trong thực tế, Instruction-level Parallelism ISA được xem như phát triển từ RISC ISA, thường chỉ có một chu kỳ hoạt

động, một trong những lý do chính khiến RISC là cơ sở của việc thực hiện lệnh song song. Ví dụ về Instruction-level Parallelism ISA bao gồm

### Mô hình SIMD (Single Instruction Multiple Data)

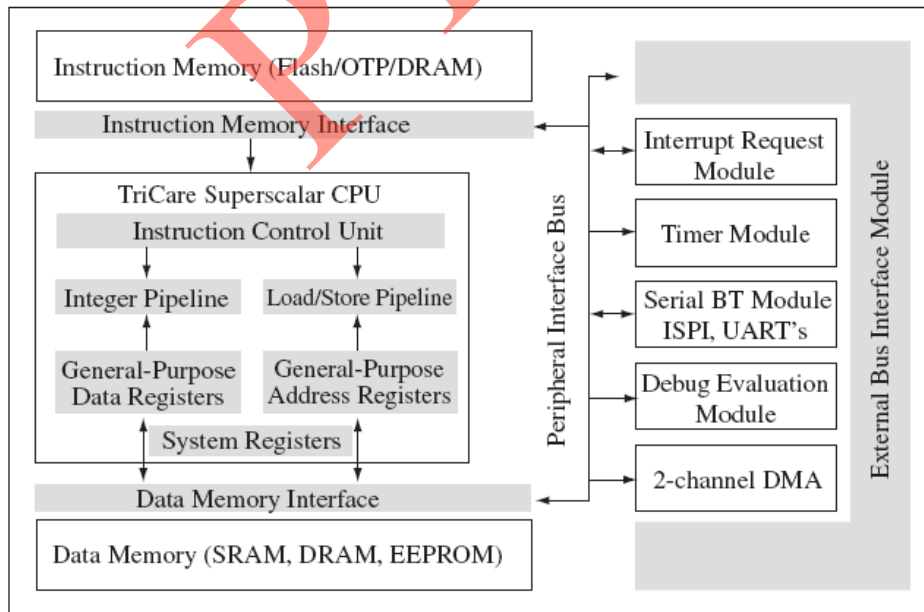
SIMD Machine ISA được thiết kế để xử lý một lệnh đồng thời trên nhiều phần dữ liệu.



Hình 2-9: ví dụ thực hiện SIMD ISA

### Mô hình máy siêu vô hướng (Superscalar Machine Model)

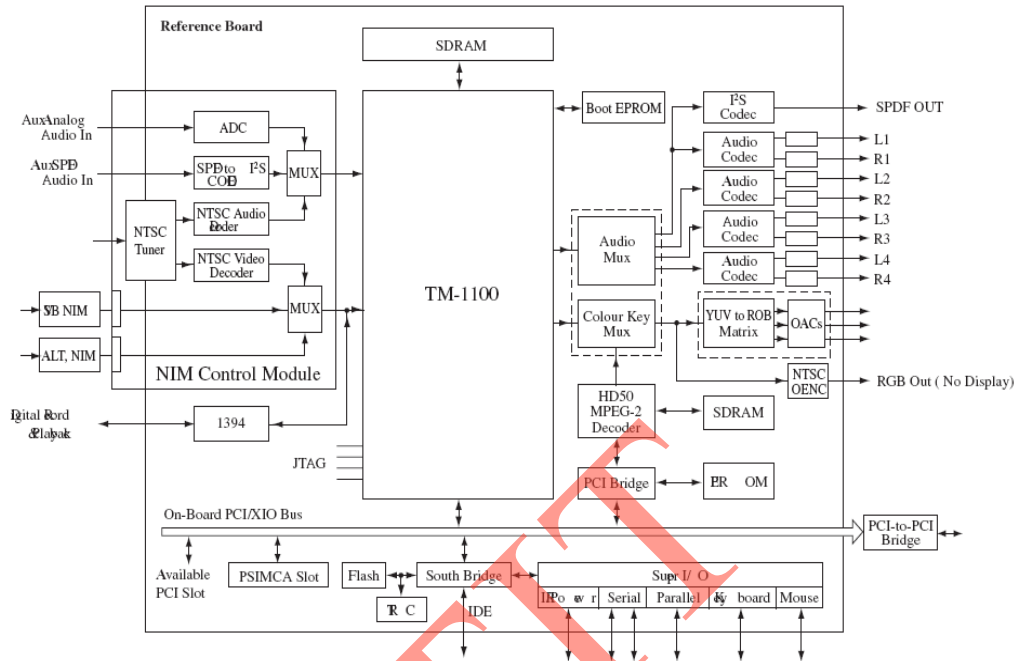
Superscalar ISA có thể thực hiện nhiều lệnh đồng thời trong một chu kỳ xung nhịp thông qua việc thực hiện nhiều hàm chức năng trong xử lý



Hình 2-10: Ví dụ thực hiện siêu vô hướng ISA

## Mô hình tính toán với từ lệnh rất dài (Very Long Instruction Word Computing: VLIW)

The VLIW ISA là kiến trúc trong đó một từ lệnh dài yêu cầu thực thi nhiều hoạt động. Các hoạt động này được chia nhỏ và xử lý song song bởi nhiều đơn vị thực thi trong bộ xử lý.



Hình 2-11: Ví dụ thực hiện VLIW ISA

### 2.1.1.5 Hiệu suất bộ xử lý

Có nhiều cách để đo hiệu suất của một bộ xử lý, nhưng tất cả đều dựa trên hoạt động của bộ xử lý trong một khoảng thời gian nhất định. Một trong những định nghĩa chung nhất về hiệu suất của bộ xử lý đó chính là lưu lượng thông tin đi qua bộ xử lý, số lượng công việc mà CPU hoàn thành trong một khoảng thời gian. Như đã nói đến ở phần 2.1, một sự thực thi của bộ xử lý được đồng bộ bởi một hệ thống bên ngoài hoặc xung nhịp chủ trên bảng mạch.

Sử dụng tốc độ xung nhịp, thời gian thực thi của CPU, là tổng thời gian bộ xử lý xử lý một số chương trình có thể tính toán được. Từ tốc độ xung nhịp, khoảng thời gian CPU hoàn thành một chu kỳ xung nhịp, là nghịch đảo của tốc độ xung nhịp, được gọi là chu kỳ của bộ xử lý. Tốc độ và chu kỳ xung nhịp của bộ xử lý thường có trong tài liệu kỹ thuật của bộ xử lý.

Nhìn vào các lệnh, chỉ số CPI (trung bình số chu kỳ xung nhịp trên một lệnh) có thể xác định theo nhiều cách. Một cách là lấy chỉ số CPI của mỗi lệnh nhân với tần số của lệnh đó.

$$CPI = (CPI \text{ per instruction} * \text{instruction frequency})$$

Tổng thời gian thực thi của các CPU có thể được xác định bởi:

thời gian CPU thực thi mỗi chương trình (tính bằng giây) = (Tổng số chỉ lệnh của mỗi chương trình hay tổng số chỉ lệnh đếm được) \* (CPI trong số lượng các chu kỳ/chỉ lệnh) \* (số giây trên một chu kỳ xung nhịp) = ((tổng số chỉ lệnh đếm được) \* (CPI trong số lượng các chu kỳ/chỉ lệnh)) / (tốc độ xung nhịp (tính bằng MHz))

{CPU execution time in seconds per program = (total number of instructions per program or instruction count) \* (CPI in number of cycle cycles/instruction) \* (clock period in seconds per cycle) = ((instruction count) \* (CPI in number of cycle cycles/ instruction)) / (clock rate in MHz) }

Tỉ lệ thực thi bình quân của bộ xử lý còn gọi là thông lượng hay băng thông, cho biết số lượng công việc CPU thực hiện trong một chu kì thời gian và bằng nghịch đảo của thời gian thực thi của CPU

CPU throughput (in bytes/sec or MB/sec) = 1 / CPU execution time = CPU performance

Các định nghĩa khác về hiệu suất bên cạnh thông lượng bao gồm:

Độ đáp ứng của bộ xử lý hay độ trễ là khoảng thời gian một bộ xử lý cần để đáp ứng một số sự kiện.

Sự sẵn sàng của bộ xử lý, thể hiện qua thời gian bộ xử lý hoạt động bình thường mà không gặp sự cố, hay là độ tin cậy, thời gian trung bình xảy ra sự cố, thời gian CPU cần để khắc phục sự cố.

Thiết kế bên trong của bộ xử lý quyết định xung nhịp và chỉ số CPI của bộ xử lý. Do đó hiệu suất của bộ xử lý phụ thuộc vào mô hình ISA nào được thực hiện và thực hiện như thế nào. Hiệu suất có thể được cải thiện bởi những hiện thực vật lý thực tế của ISA bên trong bộ xử lý, chẳng hạn như sự thực hiện kỹ thuật đường ống trong ALU.

Khoảng cách ngày càng tăng giữa hiệu suất của bộ xử lý và bộ nhớ có thể cải thiện bằng các thuật toán cache thực hiện lệnh và tìm nạp dữ liệu trước (đặc biệt là các thuật toán sử dụng các dự đoán rẽ nhánh để giảm thời gian trì hoãn) và giải phóng bộ nhớ đệm. Về cơ bản bất kì thiết kế tính năng cho phép tăng xung nhịp đồng hồ hoặc giảm chỉ số CPI sẽ tăng hiệu suất tổng thể của một bộ xử lý.

#### 2.1.1.6 Những chuẩn đánh giá (Benchmarks)

Một trong những hiệu suất chung sử dụng bộ xử lý nhưng là hàng triệu lệnh thực hiện trong một giây (MIPS)

$MIPS = Instruction\ Count / (CPU\ execution\ time * 10^6) = Clock\ Rate / (CPI * 10^6)$

MIPS khiến chúng ta cho rằng bộ xử lý nhanh hơn thì có giá trị MIPS cao hơn, do công thức MIPS tỉ lệ nghịch với thời gian thực hiện của CPU. Tuy nhiên MIPS có thể gây hiểu lầm ví một số lý do:

Các lệnh và hàm phức tạp không được xem xét trong MIPS, do đó MIPS không thể so sánh khả năng của bộ xử lý với các ISA khác nhau.

MIPS có thể biến đổi trên cùng một bộ xử lý khi chạy các chương trình khác nhau. (Với sự thay đổi cách tính các lệnh và các loại lệnh khác nhau)

Chương trình phần mềm chuẩn có thể chạy trên các bộ xử lý để đo hiệu suất của chúng.

### 2.1.1.7 Đọc datasheet của một bộ xử lý.

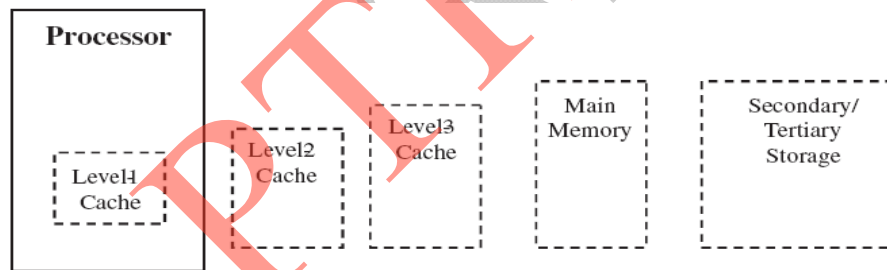
Datasheet của bộ xử lý cung cấp các thành phần chính của bộ xử lý và các thông tin để sử dụng bộ xử lý.

Lưu ý: Đừng cho rằng những gì đọc từ nhà cung cấp là chính xác 100% cho đến khi tự mình chứng kiến bộ xử lý hoạt động và tự mình xem xét các tính năng của chúng.

Datasheet tồn tại ở hầu hết các thành phần, cả phần cứng lẫn phần mềm và thông tin từ các nhà cung cấp khác nhau là khác nhau. Một số datasheet chỉ dài vài trang và chỉ xuất hiện ở những thành phần chính của hệ thống. Trong khi một số khác lại không dưới 100 trang thông số kỹ thuật.

### 2.1.2 Bộ nhớ

Nền tảng nhúng có sự phân cấp bộ nhớ, 1 tập hợp gồm các loại bộ nhớ khác nhau, mỗi loại đều có tốc độ, kích cỡ và cách sử dụng riêng biệt ( xem Hình 2-12). Một vài bộ nhớ có thể được tích hợp sẵn trong bộ xử lý, như các thanh ghi và các loại bộ nhớ sơ cấp đã biết, là nơi mà bộ nhớ có thể kết nối trực tiếp hay tích hợp trong bộ xử lý như ROM, RAM và level-1 cache. Trong chương này, ta nói đến loại bộ nhớ điển hình bên ngoài bộ xử lý, hay có thể là cả hai cùng được tích hợp bên trong bộ xử lý hay cùng bên ngoài bộ xử lý, đó là vấn đề cần thảo luận. Chương này bao gồm các loại bộ nhớ sơ cấp, như ROM, level-2+ cache, và bộ nhớ chính, và bộ nhớ cấp2, cấp3, đó là những loại bộ nhớ có thể kết nối tới bản mạch nhưng không tới bộ xử lý **chủ** trực tiếp được, như CD-ROM, ổ mềm, ổ cứng và băng từ.



Hình 2-12: sự phân cấp bộ nhớ

Bộ nhớ sơ cấp là một phần đặc trưng của hệ thống con của bộ nhớ, gồm 3 phần:

- IC nhớ
- Bus địa chỉ
- Bus dữ liệu

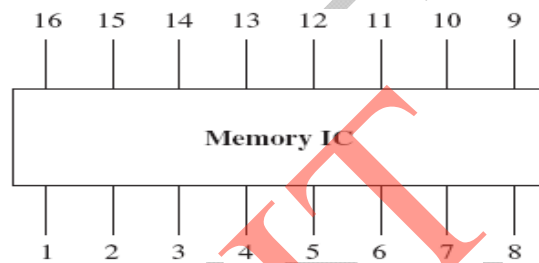
Thông thường, một IC nhớ gồm 3 bộ phận: mảng nhớ, bộ giải mã địa chỉ, và giao diện dữ liệu. Mảng nhớ thực tế là bộ nhớ vật lý để lưu trữ các bit dữ liệu. Trong khi bộ xử lý chủ, và các hệ lập trình, thiết lập bộ nhớ như mảng một chiều, mỗi ô trong mảng là một hàng các byte và số bit trên một hàng có thể biến thiên, trong bộ nhớ vật lý thực tế là mảng 2 chiều được tạo thành bởi các ô nhớ định địa chỉ bởi mỗi hàng và cột duy nhất, mỗi ô có thể lưu trữ 1 bit.

Bộ phận chính còn lại của IC nhớ, bộ giải mã địa chỉ, định vị địa chỉ của dữ liệu trong mảng nhớ, làm nền tảng cho những thông tin nhận được qua bus địa chỉ, và giao diện của dữ liệu sẽ cung cấp các dữ liệu tới bus dữ liệu trong quá trình vận chuyển. Bus

địa chỉ và bus dữ liệu đều nhận địa chỉ và dữ liệu từ bộ giải mã địa chỉ và giao diện địa chỉ của IC nhớ.

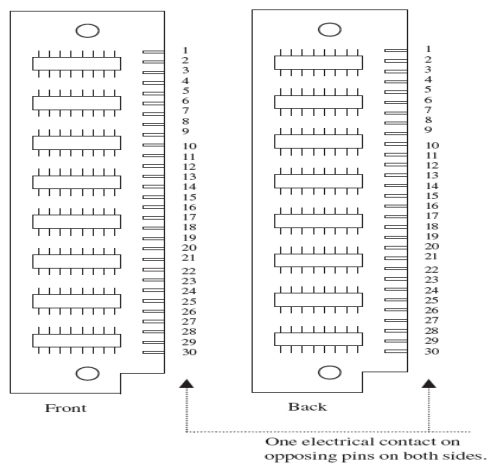
Các IC nhớ có thể kết nối tới bản mạch của nhiều loại gói khác nhau, phụ thuộc vào từng loại bộ nhớ. Các loại gói bao gồm loại vỏ 2 hàng chân (DIP), môđun nhớ từng dòng đơn lẻ (SIMM), và môđun nhớ 2 dòng (DIMM). Như chỉ dẫn ở Hình 2-16a, DIP là các gói bao quanh IC, được làm từ gốm hoặc chất dẻo. Số chốt có thể đa dạng giữa các IC nhớ, nhưng sơ đồ chân thực tế của các IC nhớ khác nhau đều tuân theo chuẩn JEDEC để đơn giản lệnh giao diện bên ngoài của các IC nhớ tới bộ xử lý.

SIMM và DIMM (được nêu ở Hình 2-16b và c) là các môđun nhỏ (PCB) chứa một vài các IC nhớ. SIMM và DIMM có các chân nhô ra từ 1 phía (cả 2 cùng nằm phía trước hoặc sau) của môđun để kết nối tới bản mạch nhúng chính. Cấu hình của SIMM và DIMM có thể cùng thay đổi trong kích cỡ của IC nhớ trên môđun (256KB, 1MB, ...) Ví dụ, 256K x 8 SIMM là một môđun yêu cầu 256K ( $256 \times 1024$ ) địa chỉ cho mỗi byte. Để hỗ trợ cho bộ xử lý chủ 16-bit, 2 trong số các SIMM sẽ cần dùng; để hỗ trợ cho cấu trúc 32-bit, 4 SIMM của cấu hình cần dùng đến, v.v...

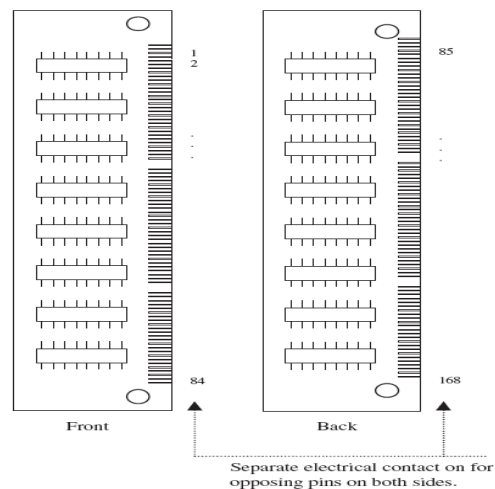


Hình 2-13a: Ví dụ DIP

Số chân nhô ra từ các SIMM và DIMM có thể khác nhau (30 chân, 72 chân, 168 chân, v.v...). Ưu điểm của SIMM và DIMM là có nhiều chân hơn để nó cấp phát cho ít môđun cần đến hỗ trợ cấu trúc rộng hơn. Do vậy, ví dụ như, 1 SIMM 72 chân ( $256K \times 32$ ) sẽ thay thế cho 4 SIMM 30 chân ( $256K \times 8$ ) cho cấu trúc 32 bit. Vậy, sự khác nhau lớn nhất giữa SIMM và DIMM là đặc trưng của các chân trên môđun thế nào: ở SIMM, loại 2 chân trong cùng bản mạch được kết nối, tạo một tiếp xúc, trái lại, ở DIMM các chân ngược lại đều có các tiếp xúc độc lập (xem Hình 2-16b và c).



Hình 2-13b: Ví dụ SIMM 30 chân



Hình 2-13c: Ví dụ DIMM 168 chân

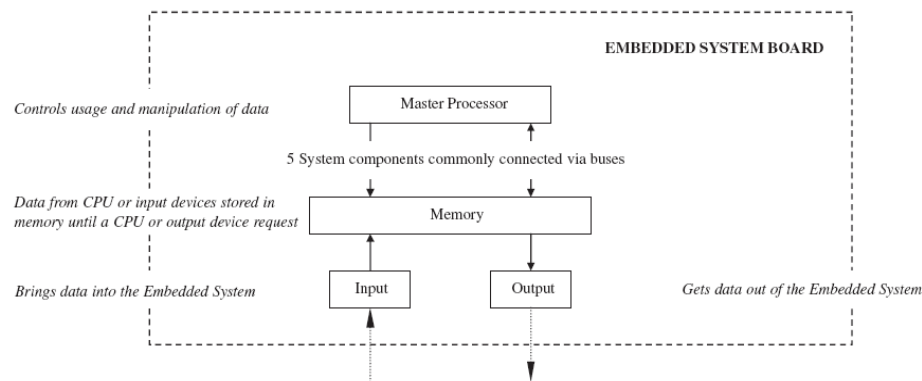
Ở mức cao nhất, cả bộ nhớ sơ cấp và cấp 2 đều có thể được chia thành 2 nhóm, cố định và khả biến. Bộ nhớ không bốc hơi là bộ nhớ có thể lưu trữ dữ liệu sau khi nguồn điện chính cấp cho bảng mạch đã được tắt (thường là do nhỏ, gắn trên bảng mạch, có pin nguồn cấp lâu dài). Bộ nhớ khả biến sẽ làm mất các bit dữ liệu của nó khi nguồn điện chính trên bản mạch bị tắt. Trên mạch nhúng, có 2 loại họ bộ nhớ không thay đổi- bộ nhớ chỉ đọc (ROM) và bộ nhớ ngoài- và một họ bộ nhớ thay đổi, bộ nhớ truy cập tức thời (RAM).

### 2.1.3 Bảng mạch Vào/Ra

Các bộ phận vào/ra trên bảng mạch là thiết bị chịu trách nhiệm di chuyển thông tin đến và đi của bảng mạch đến thiết bị vào/ra kết nối tới một hệ thống nhúng. Bảng mạch vào/ra có thể gồm những bộ phận đầu vào, thiết bị chỉ đưa thông tin từ một thiết bị đầu vào đến bộ xử lý chính, bộ phận đầu ra là cái lấy thông tin ra của bộ xử lý chính đưa đến 1 thiết bị đầu ra. Hoặc là gồm cả thiết bị vào và thiết bị ra.

Bất kỳ hệ thống điện cơ nào, được nhúng hay không nhúng, cho dù thông thường hay không thông thường, có thể được kết nối tới một bảng mạch nhúng và hoạt động như một thiết bị vào/ra. Vào/ra ở mức cao có thể được chia nhỏ ra thành những bộ nhỏ hơn của thiết bị ra, thiết bị vào, và cả thiết bị mà bao gồm cả thiết bị ra và thiết bị vào. Thiết bị ra nhận dữ liệu từ những bộ phận bảng mạch I/O và hiển thị dữ liệu đó theo một số cách thức, ví dụ : in dữ liệu ra giấy, in ra đĩa, hiển thị lên màn hình hoặc là nhấp nháy đèn LED... Thiết bị vào ví dụ như chuột, bàn phím hoặc là điều khiển từ xa truyền dữ liệu đến bảng mạch I/O. Một số thiết bị vào ra có thể làm cả 2, vừa có thiết bị kết nối vừa có thể truyền dữ liệu đi. Ví dụ một thiết bị vào/ra có thể kết nối tới 1 bảng mạch nhúng thông qua dây nối hoặc môi trường truyền dữ liệu không dây, ví dụ 1 bàn phím hay điều khiển từ xa, hoặc có thể định vị ở trên chính bảng mạch nhúng ví dụ như đèn LED.





Hình 2-14: Sơ đồ khối I/O cơ bản của kiến trúc Von Newman

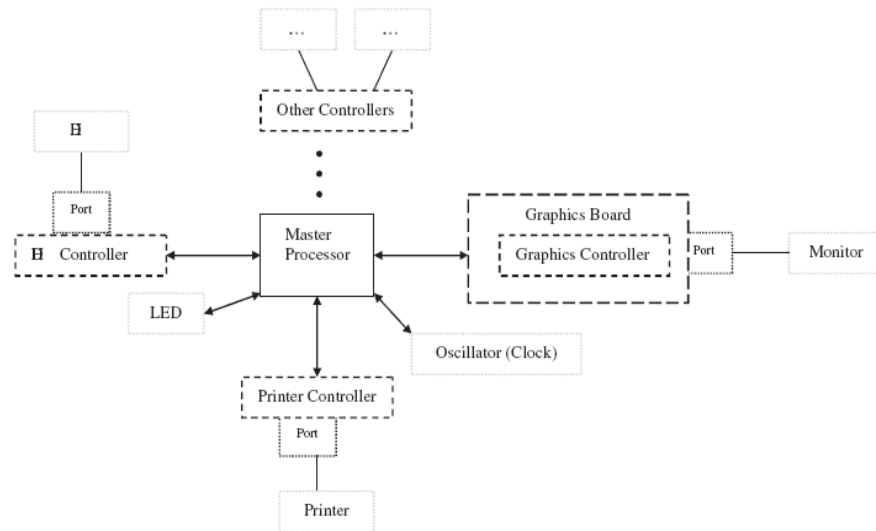
Bởi vì các thiết bị vào/ra rất đa dạng, khoảng cách từ một mạch đơn giản đến hệ thống nhúng hoàn thành, linh kiện của bảng mạch vào ra có thể chênh lệch một hoặc nhiều loại khác nhau. Những điều giống nhau nhất gồm :

- Liên kết mạng và kết nối vào/ra
- Thiết bị vào
- Sơ đồ và thiết bị ra
- Thiết bị điều chỉnh vào /ra
- Thời gian thực và đa dạng vào/ra( bộ thời gian/đếm,chuyển từ tương tự sang số, chuyển từ số sang tương tự...)

Ban đầu, bảng mạch vào/ra rất đơn giản chủ yếu là mạch điện kết nối với bộ xử lý chính, cổng vào/ra của bộ xử lý chính như là cổng xung nhịp hay đèn LED được định thời trên bảng. Phần cứng của I/O điển hình được làm từ tất cả hoặc một số kết hợp của 6 bộ logic chính sau:

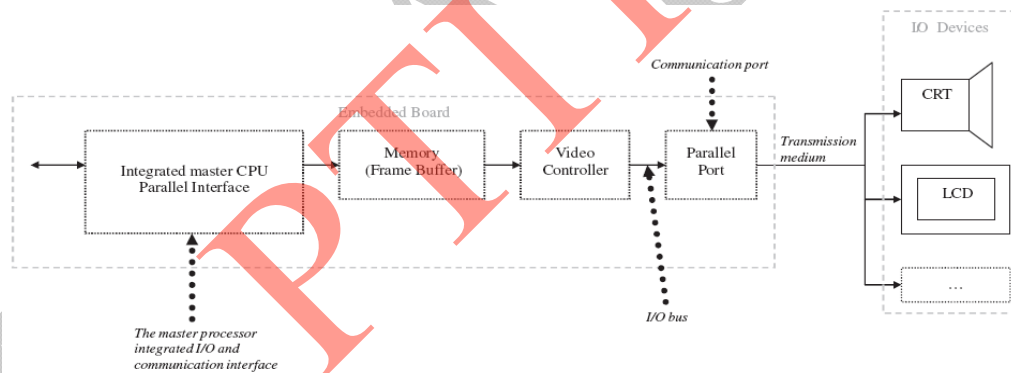
- Phương tiện truyền thông, đường truyền không dây hoặc có dây kết nối với thiết bị vào/ra đến dữ liệu thông tin và chuyển đổi của bảng mạch nhúng.
- Cổng thông tin là phương tiện truyền thông kết nối tới bảng mạch hoặc nếu có hệ thống không dây thì nhận tín hiệu không dây.
- Giao diện truyền thông, bộ phận quản lý dữ liệu thông tin giữa CPU chủ và thiết bị vào/ra hoặc điều khiển vào/ra và nó chịu trách nhiệm mã hoá và giải mã dữ liệu và từ mức logic của một IC và mức logic của một cổng vào/ra. Giao diện này có thể được tích hợp ở trong bộ xử lý chính hoặc có thể là một IC riêng rẽ.
- Một điều khiển vào/ra xử lý quản lý thiết bị vào/ra.
- Đường truyền dẫn vào/ra là cái kết nối giữa bảng mạch vào/ra và bộ xử lý chính
- Bộ xử lý chính tích hợp vào /ra.



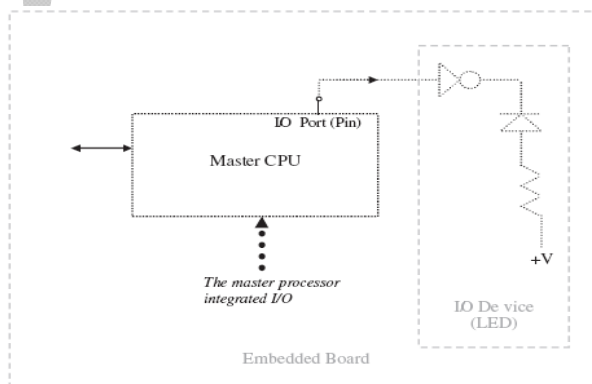


Hình 2-15: Các cổng và bộ điều khiển thiết bị điều khiển trên một bảng mạch nhúng

Bảng mạch vào/ra có thể giới hạn từ thành phần hệ thống phức tạp Hình 2-16a. Một số thành phần bảng mạch vào/ra tích hợp xem Hình 2-16b.



Hình 2-16a: Bảng mạch I/O phức tạp



Hình 2-16b: Bảng mạch I/O đơn giản

Hiện tại sự lắp ráp của 1 hệ thống cài đặt vào/ra ở trên 1 bảng mạch nhúng, có thể dùng kết nối và cổng hoặc dùng 1 thiết bị điều khiển vào/ra, là phụ thuộc của thiết bị kết nối vào/ra định vị trên bảng mạch nhúng. Nó có nghĩa là trong khi một số chỉ số như là an toàn và mở rộng rất là quan trọng trong việc thiết kế một hệ thống vào/ra. Đọc các thông số của bộ phận chính sau khi thiết kế hệ thống vào ra chính là đường bao quanh thiết bị vào/ra - giới hạn mục đích -chất lượng.

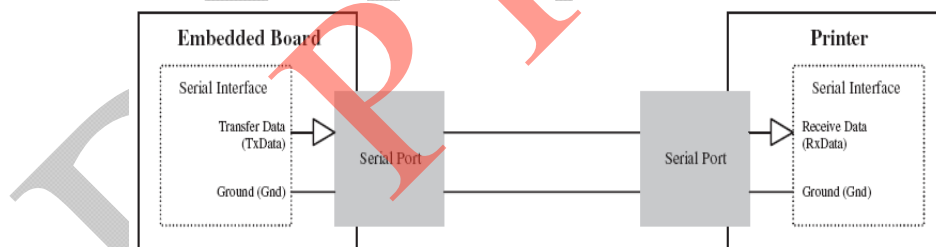
### 2.1.3.1 Quản lý dữ liệu nối tiếp

Bảng mạch I/O này có thể truyền dữ liệu và nhận dữ liệu nối tiếp. Phần cứng I/O nối tiếp là kiểu được tạo sẵn của hệ thống được kết hợp 6 phần logic chính đã được nói qua ở trên. Sự truyền thông nối tiếp bao gồm hệ thống con I/O bên trong một cổng nối tiếp và một giao diện nối tiếp.

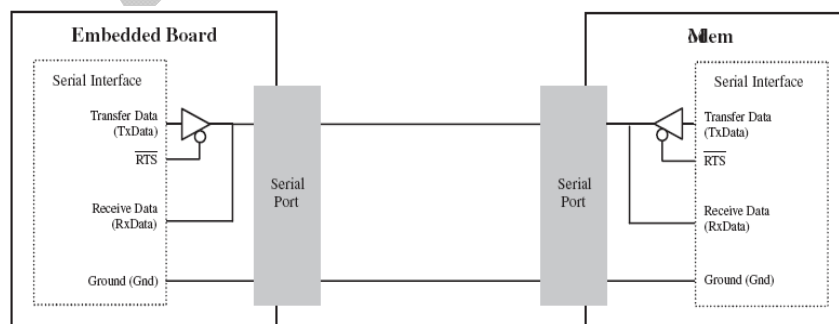
Giao diện nối tiếp là chuỗi dữ liệu được truyền và nhận giữa CPU chính và một số thiết bị I/O hoặc bộ điều khiển của nó. Chúng bao gồm nhận và truyền dữ liệu từ bộ đệm tới bộ lưu trữ và giải mã hoặc mã hoá dữ liệu và chúng có trách nhiệm truyền dữ liệu tới CPU chính khác hoặc thiết bị vào ra khác. Như vậy là sự thật là truyền và nhận dữ liệu trên một từ, dữ liệu truyền và nhận giới hạn bởi ghép nối tiếp.

Dữ liệu có thể truyền giữa 2 thiết bị trong 1 của 3 hướng: trong 1 chiều hướng, trong cả 2 hướng trong thời gian riêng, bởi vì nó có thể chia sẻ những đường dữ liệu giống nhau và cả 2 hướng tương thích. Chuỗi dữ liệu thông tin vào/ra dữ liệu nối tiếp vào/ra có thể dùng sơ đồ đơn giản mô tả dữ liệu nối tiếp có thể truyền hoặc nhận trong 1 bộ xử lý (Hình 2-17a)

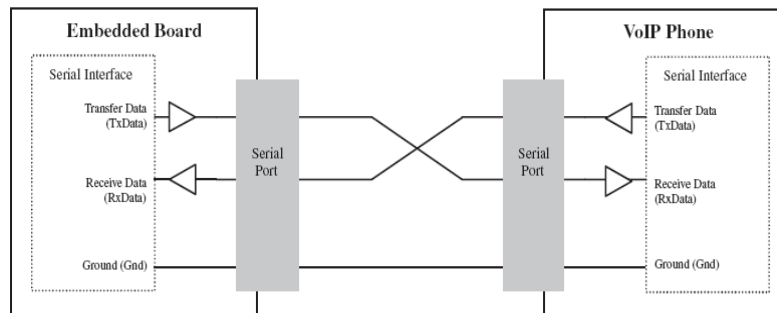
Sơ đồ bán song công mô tả chuỗi dữ liệu có thể truyền hoặc nhận ở một bộ vi xử lý khác nhưng 1 bộ xử lý chỉ truyền 1 lần (Hình 2-17b). Sơ đồ chuỗi song công mô tả chuỗi dữ liệu có thể truyền hoặc nhận ở bộ vi xử lý khác tương thích (Hình 2-17c)



Hình 2-17a: Ví dụ sơ đồ truyền đơn giản



Hình 2-17b: Ví dụ sơ đồ truyền bán song công

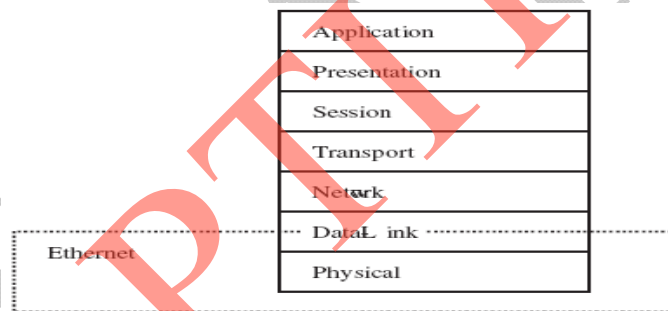


Hình 2-17c: Ví dụ sơ đồ truyền song công

**VÍ DỤ : CỔNG NỐI TIẾP VÀO/RA :** liên kết mạng và truyền thông RS232.

Giao thức vào/ra nối tiếp có thể truyền đồng bộ hoặc không đồng bộ: RS232, EIA232. EIA hiệp hội công nghiệp điện tử. Một số chuẩn định nghĩa những thành phần chính của hệ thống RS\_232, những cái được thực hiện trong phần cứng.

Thành phần của phần cứng là tất cả lớp ánh xạ vật lý của OSI ( xem Hình 2-18) phần mềm yêu cầu khởi động chức năng ánh xạ RS232 đến những đoạn dưới của đường dữ liệu, nhưng chúng ta sẽ không thảo luận ở phần này.



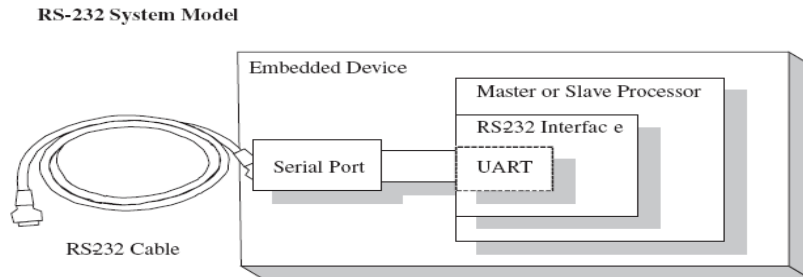
Hình 2-18: Mô hình OSI

Sự phù hợp của chuẩn EIA232, thiết bị thích hợp của RS232( xem Hình 2-19) được gọi là DTE và DCE. Thiết bị DTE là sự khởi đầu của thông tin nối tiếp như là PC hay hệ thống nhúng. DCE là thiết bị DTE muốn truyền qua. Ví dụ như thiết bị vào ra kết nối với hệ thống nhúng.



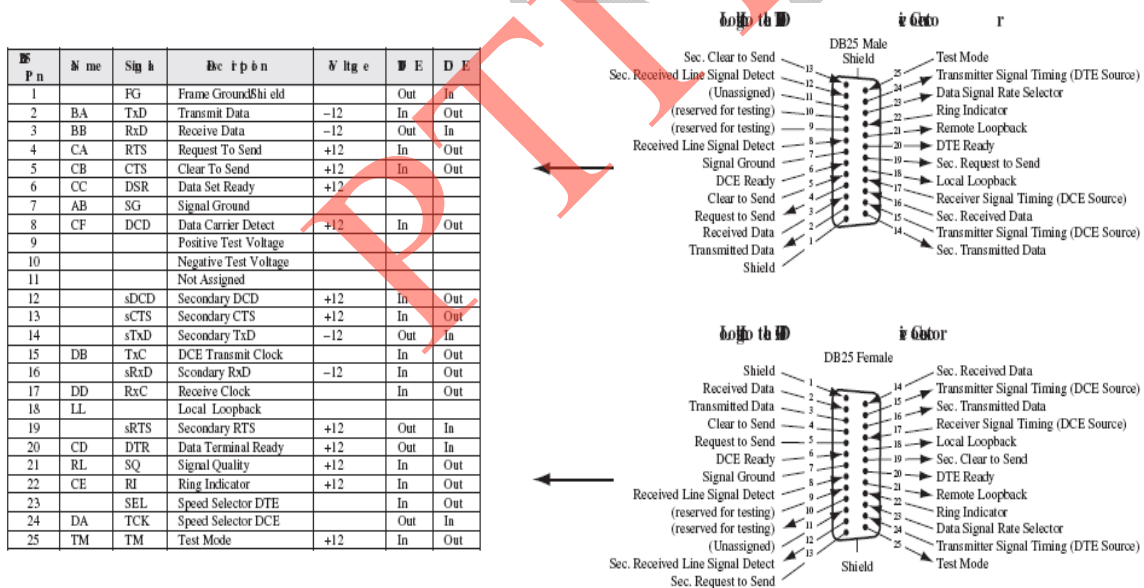
Hình 2-19: Sơ đồ mạng nối tiếp

Bản chất đặc tính kỹ thuật của RS232 :*RS232 interface* (xem Hình 2-20). RS232 interface là định nghĩa chi tiết của cổng nối tiếp và những tín hiệu với những mạch điện được bổ sung là những tín hiệu ánh xạ từ 1 cổng giao diện đồng bộ hoặc không đồng bộ (ví dụ UART) được mở rộng bởi thiết bị vào/ra. RS232 được định nghĩa như phương tiện truyền dữ liệu là những dây nối tiếp. Kết nối RS232 phải có cả hai cổng truyền thông tin (DTE và DCE hoặc hệ thống nhúng và thiết bị vào/ra).

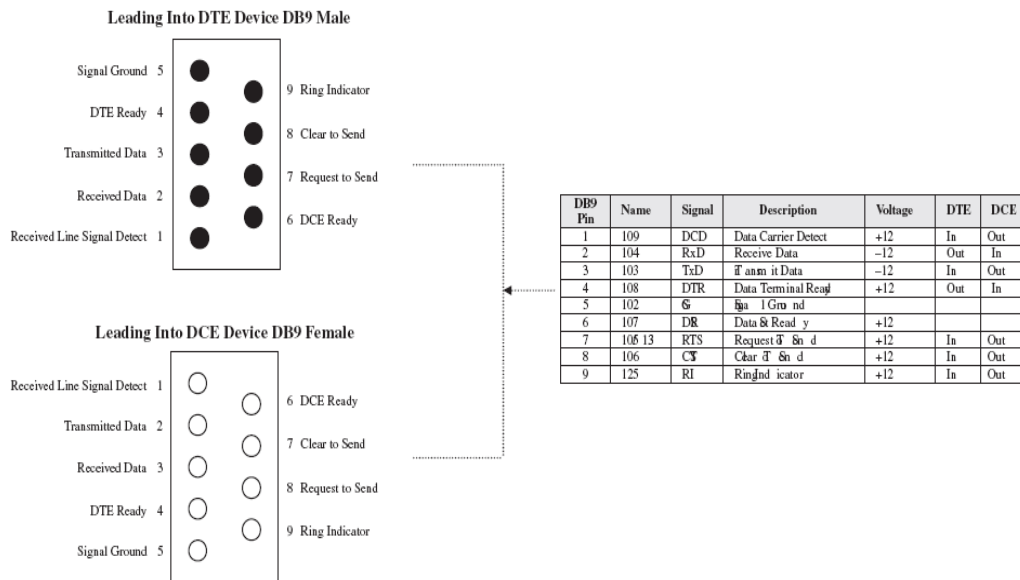


Hình 2-20: Sơ đồ khối những thành phần nối tiếp

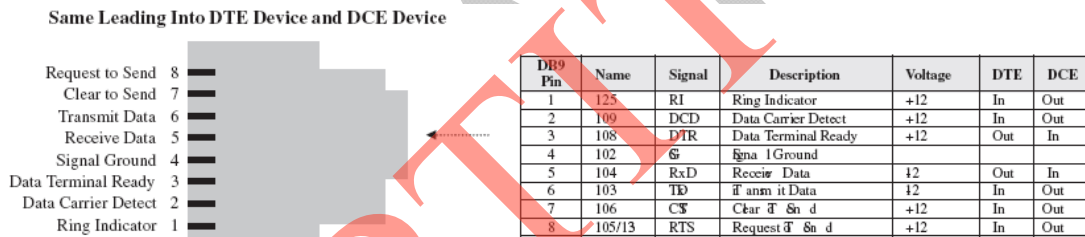
Dùng số tín hiệu để phân biệt giữa chuẩn EIA 232 và chuẩn RS232. Chuẩn RS 232 dùng tất cả 25 tín hiệu gọi là kết nối DB25 xem Hình 2-30a. Chuẩn EIA có 8 tín hiệu phù hợp với kết nối DB9 xem Hình 2-21b. Chuẩn EIA 561 có 8 tín hiệu phù hợp với kết nối RJ45 xem Hình 2-21c.



Hình 2-21a: Các tín hiệu RS-232 và đầu nối DB25



Hình 2-21b: Các tín hiệu RS-232 và đầu nối DB9

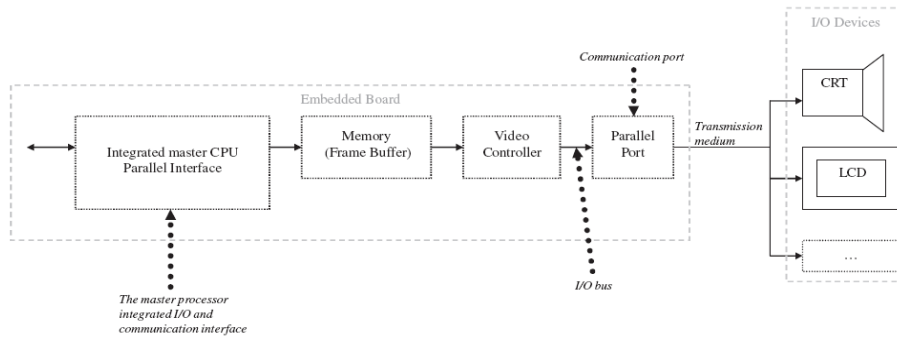


Hình 2-21c: Các tín hiệu RS-232 và đầu nối RJ45

Hai thiết bị DTE có thể kết nối với nhau thông qua *null modem*. DTE có thể truyền và nhận dữ liệu trên 1 chân. Những chân null modem trao đổi nhận và truyền kết nối trên mỗi thiết bị DTE được điều phối.

### 2.1.3.2 Giao diện các thành phần I/O

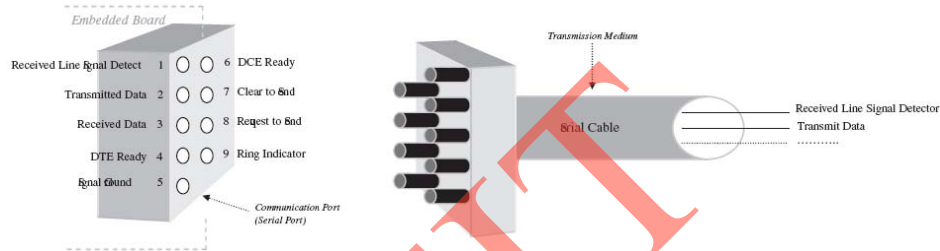
Phần cứng vào/ra được làm từ tất cả hoặc kết hợp của bộ xử lý chính vào/ra hoặc bộ điều khiển vào/ra, bus vào/ra và môi trường truyền dữ liệu. Xem Hình 2-22.



Hình 2-22: Hệ thống con I/O mẫu

### Giao diện thiết bị I/O với bảng mạch nhúng

Thiết bị vào/ra ví dụ như bàn phím, LCD, máy in kết hợp bảng mạch nhúng thông qua cổng thông tin. Môi trường không dây Hình 2-23b hoặc môi trường dây Hình 2-23a.



Hình 2-23a: Phương tiện truyền tin dùng dây

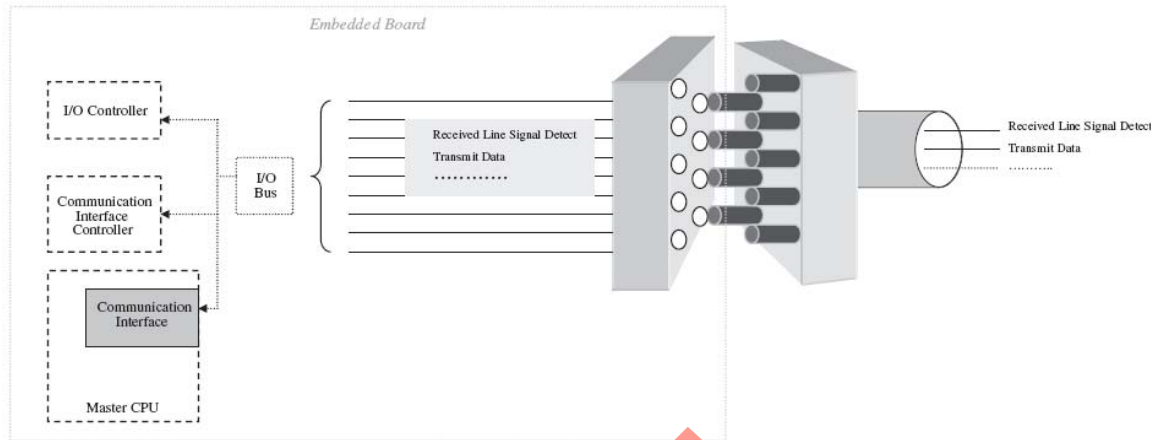


Hình 2-23b: Phương tiện truyền tin không dây

Trong hình 2-23a, môi trường truyền dữ liệu giữa thiết bị vào/ra và bảng mạch nhúng bằng dây dẫn. Trong hình 2-23b, môi trường truyền dữ liệu là không dây.

Giao diện điều khiển thông tin, bộ xử lý chính thông qua bus vào/ra trên hệ thống nhúng (xem Hình 2-36) bus vào/ra về bản chất là tập hợp truyền dữ liệu bằng dây dẫn.

Thiết bị vào/ra có thể kết nối bộ xử lý chính thông qua cổng vào/ra. Nếu thiết bị vào/ra định vị trên bảng mạch hoặc có thể kết nối gián tiếp dùng 1 giao diện truyền thông trong bộ xử lý chính hoặc trên 1 IC riêng biệt trên bảng mạch và 1 cổng giao tiếp. Giao diện thông tin kết nối trực tiếp với thiết bị vào/ra hoặc con trỏ điều khiển thiết bị vào/ra. Ngoài tấm mạch của thiết bị vào/ra các thành phần bảng mạch vào/ra có liên quan thông qua bus vào/ra.



Hình 2-24: Giao diện cổng truyền tin tới I/O tấm bảng mạch khác

### Giao diện một bộ điều khiển I/O và CPU chủ

Hệ con bao gồm 1 bộ điều khiển quản lý thiết bị vào/ra, thiết kế giao diện giữa bộ điều khiển I/O và CPU chủ thông qua 1 giao diện thông tin – 4 yêu cầu cơ bản sau :

- Năng lực của máy chủ CPU khởi tạo và điều chỉnh thiết bị vào/ra. Bộ điều khiển vào/ra có thể đặc trưng cho cấu hình thông qua bộ điều khiển bộ ghi và bộ điều chỉnh thông qua trạng thái bộ ghi. Những bộ ghi được định vị trên bộ điều khiển vào/ra. Dữ liệu bộ ghi được máy chủ xử lý có thể điều chỉnh cấu hình trên bộ điều khiển vào/ra. Trạng thái bộ đếm chỉ đọc trên bộ đếm, máy chủ có thể xử lý thông tin phụ thuộc vào trạng thái của bộ điều khiển vào/ra. Máy chủ CPU dùng trạng thái và bộ điều khiển để nhớ thông tin đến thiết bị vào/ra thông qua bộ điều khiển vào/ra.
- Đường đi của bộ xử lý chính đến yêu cầu thiết bị vào/ra: bộ xử lý chính yêu cầu thiết bị vào/ra thông qua bộ điều khiển vào/ra.
- Đường đi của thiết bị vào/ra kết nối với máy chủ CPU: bộ điều khiển vào ra kết nối với máy chủ CPU thông qua các ngắt.
- Bộ dẫn động thay đổi dữ liệu: trong 1 chương trình bộ xử lý chính nhận dữ liệu từ bộ điều khiển vào/ra trong bộ đếm và CPU truyền dữ liệu đến bộ nhớ.

#### 2.1.3.3 I/O và hiệu suất

Gồm các đặc trưng:

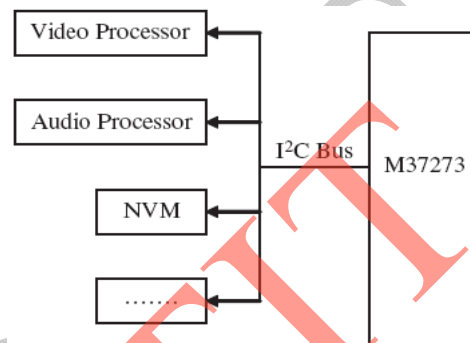
- Tốc độ truyền dữ liệu của thiết bị vào/ra
- Tốc độ của bộ xử lý chính
- Cách đồng bộ hóa giới hạn của bộ xử lý chính và giới hạn của thiết bị vào/ra.
- Cách để vào/ra và bộ xử lý chính liên lạc

Những thành phần để đo đặc trưng của thiết bị vào/ra gồm :

- Lưu lượng của các thành phần vào/ra khác nhau
- thời gian thực hiện của các thành phần vào/ra
- thời gian phản hồi hoặc là độ trễ của thiết bị vào/ra.

### 2.1.5 Hệ thống Bus

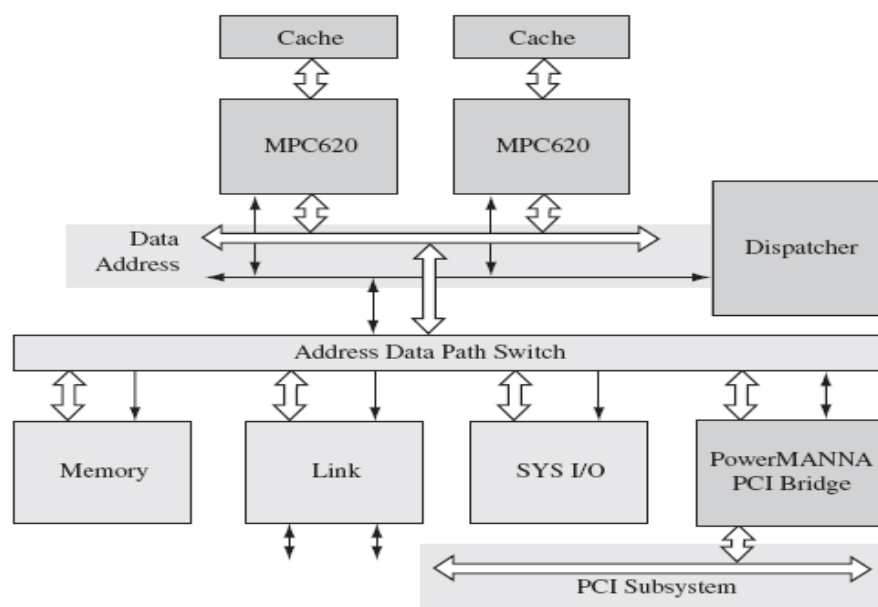
Tất cả các bộ phận chính khác nhau tạo nên một bảng nhúng- bộ xử lý tổng thể, bộ phận vào ra I/O, bộ nhớ - được kết nối với nhau thông qua các bus trên bảng nhúng. Định nghĩa ban đầu, bus là một kết nối đơn giản của tập hợp các dây mang các tín hiệu khác nhau, địa chỉ, và kiểm soát tín hiệu (tín hiệu đồng hồ, yêu cầu, nhận biết tín hiệu, ...) giữa tất cả các thành phần chính khác trên bảng nhúng, trong đó bao gồm các hệ thống con vào ra I/O, hệ thống bộ nhớ con và bộ xử lý tổng thể. Trên bảng nhúng có ít nhất 1 bus liên kết với các thành phần chính khác trong hệ thống (Hình 2-26).



Hình 2-26: Cấu trúc bus chung

Trên một bản phức tạp, nhiều bus có thể được tích hợp trên một bảng. Một bảng nhúng với một vài kết nối các thành phần để liên lạc, cầu (bridge) trên các bảng kết nối bus khác nhau và mang thông tin từ bus khác. Trong Hình 2-27 cầu PCI là một trong những ví dụ. Một cầu có thể tự động cung cấp một cách minh bạch bản đồ của địa chỉ thông tin khi dữ liệu được di chuyển từ một bus khác, thực hiện điều khiển tín hiệu yêu cầu cho các bus nhận chu kỳ, ví dụ như: cũng như sửa đổi các dữ liệu được truyền đi nếu chuyển đổi giao thức khác từ bus đến bus. Nếu một byte sắp xếp khác, cầu có thể xử lý việc trao đổi byte.





Hình 2-27: Kiến trúc MPC620 với cầu

Bảng bus đặc trưng bởi một trong 3 loại: hệ thống bus, bus đa năng, hoặc bus vào ra I/O. Hệ thống bus ( cũng được xem là chính, cục bộ, bộ xử lí, bộ nhớ bus) kết nối bên ngoài bộ nhớ chính và giữ CPU chỉ và/hoặc bất kì bridge đến bus khác. Hệ thống bus thường ngắn hơn, tốc độ cao hơn, tùy chỉnh bus. Backplane Bus đa năng thường nhanh hơn bus mà ở bộ nhớ kết nối, bộ xử lí tổng thể, I/O tất cả trên 1 bus. Các bus I/O cũng được gọi là bus “ mở rộng”, “ bên ngoài” hoặc “chủ”, kết quả hoạt động như phần mở rộng của hệ thống bus để kết nối các thành phần còn lại cho CPU với nhau, đến hệ thống bus qua bridge và/hoặc đến các hệ thống riêng của riêng mình, qua cổng thông tin I/O. Các Bus I/O là các bus đặc trưng tiêu chuẩn hóa có thể ngắn hơn, tốc độ bus cao hơn, như PCI và USB hoặc bus dài hơn, chậm hơn ví dụ như SCSI.

Sự khác biệt chính giữa hệ thống bus và bus I/O là có IRQ điều khiển tín hiệu trên bus I/O. Có nhiều cách khác nhau I/O và bộ xử lý chủ có thể giao tiếp và ngắt là một trong những phương thức phổ biến nhất. Một IRQ vào ra trên I/O trên một bus để cho phép đến bộ xử lý trung tâm một sự kiện đã xảy ra hoặc một hành động đã được hoàn thành bởi tín hiệu trên bus IRQ. Khác I/O bus có thể có tác động khác trên sơ đồ ngắt. Một bus ISA, ví dụ như yêu cầu mỗi thẻ mà tạo ra ngắt phải chỉ định của nó giá trị IRQ duy nhất. Bus PCI cho phép hai hoặc nhiều thẻ I/O phân chia giá trị giống như IRQ.

Trong mỗi loại bus, nhiều bus có thể được chia thành hoặc bus là mở rộng hoặc là không mở rộng. Một bus mở rộng (PCMCIA, PCI, IDE, SCSI, USB....) là một trong những thành phần bổ sung có thể cắm trong bảng on-the-fly, nhưng trái lại một **bus không mở rộng** (DIB, VME, I2C là những ví dụ) là một trong những thành phần bổ sung không thể cắm đơn giản trong bảng và sau đó liên lạc trên bus đến các thành phần khác.

Trong khi thực hiện hệ thống bus được mở rộng linh hoạt hơn bởi vì các thành phần có thể được thêm đặc biệt cho các bus và làm việc “ ngoài khung”, mở rộng các bus có xu hướng đắt hơn khi thực hiện. Nếu bảng không được thiết kế ban đầu với tất cả

các loại có thể có của các thành phần có thể được thêm vào trong bộ nhớ sau này, chất lượng có thể bị ảnh hưởng xấu bởi việc bổ xung quá nhiều “đường dẫn” hoặc các thành phần được thiết kế kém trên bus mở rộng.

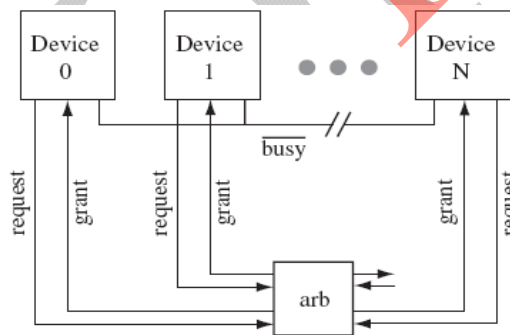
### 2.1.5.1 TRỌNG TÀI BUS VÀ ĐỊNH THỜI

Được liên kết với bus là một số loại giao thức được định nghĩa cách các thiết bị truy nhập vào bus (trọng tài) các qui định kèm theo các thiết bị phải tuân để giao tiếp trên bus (bắt tay) và các tín hiệu liên kết với các dòng bus khác nhau.

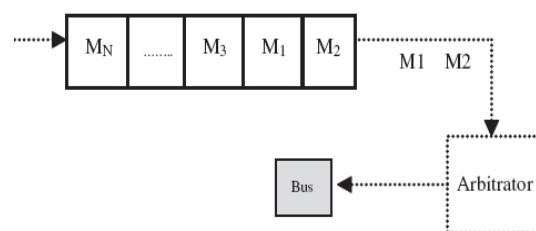
Bảng các thiết bị được truy cập vào một bus đang dùng một hệ thống trọng tài bus. Trọng tài Bus được dựa trên các thiết bị đang được phân loại như là một trong hai thiết bị chủ (các thiết bị có thể bắt đầu một giao dịch bus - transaction) hoặc các thiết bị phụ (là các thiết bị mà có thể truy nhập vào một bus để đáp ứng với yêu cầu của thiết bị chủ). Một hệ thống trọng tài bus đơn giản nhất là chỉ có một thiết bị ở trên bảng (board) – xử lý chủ để được cho phép tới máy chủ, trong khi tất cả các thiết bị khác là thiết bị phụ. Trong trường hợp này, không cần có trọng tài khi có thể chỉ là một máy chủ.

Nhiều bus có thể cho phép nhiều máy chủ, có một số phân xử (phân chia sơ đồ mạch phần cứng) xác định tổng thể một máy chủ được điều khiển của bus. Có một vài sơ đồ trọng tài bus được sử dụng cho bus nhưng đang được phổ biến nhất song song tập trung động, tập trung theo từng chuỗi (daisy-chain) và tự lựa chọn phân phối.

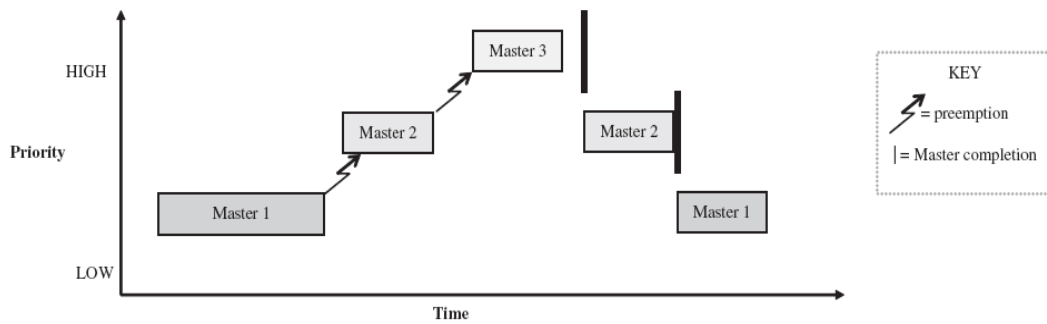
Sự phân xử song song tập trung động là sơ đồ trong đó trọng tài ở vị trí trung tâm. Tất cả các bus chủ kết nối với trọng tài trung tâm. Trong sơ đồ này, các máy chủ sau khi được cấp quyền truy cập đến bus thông qua FIFO hoặc được ưu tiên trên cơ sở hệ thống. Thuật toán FIFO thực hiện một số loại hàng đợi lưu trữ trong danh sách của các thiết bị chủ sẵn sàng sử dụng bus theo thứ tự của yêu cầu bus. Các thiết bị chủ được thêm ở cuối hàng đợi và được phép truy nhập đến bus từ đầu hàng đợi. Một nhược điểm chính là khả năng của trọng tài không can thiệp nếu một máy chủ ở trước hàng đợi duy trì kiểm soát của bus không hoàn thành hoặc không cho phép máy chủ truy cập bus.



Hình 2-28a: Sự phân xử song song tập trung động



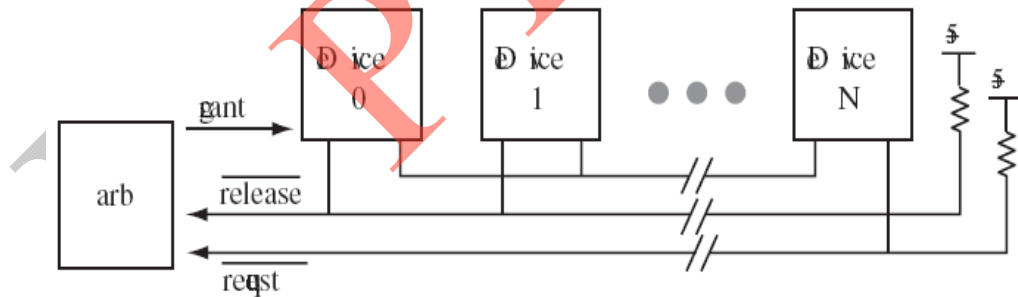
Hình 2-28b: Sự phân xử trên nền tảng FIFO



Hình 2-28c: Sự phân xử trên nền quyền ưu tiên

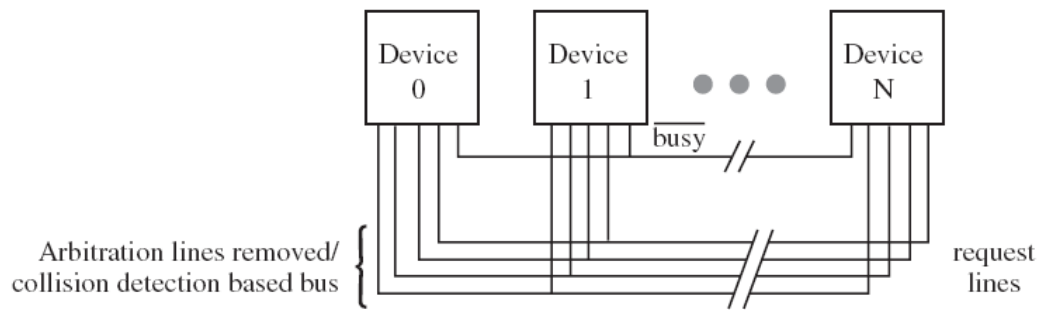
Hệ thống phân xử ưu tiên phân biệt với máy chủ dựa trên tầm quan trọng tương đối với nhau và hệ thống. Về cơ bản thì mỗi thiết bị chủ được ưu tiên gán giá trị, hành động như một chỉ thị của thứ tự ưu tiên trong hệ thống. Nếu bộ phân xử thực hiện quyền ưu tiên trên cơ sở hệ thống, thiết bị chủ với quyền ưu tiên cao nhất luôn luôn có thể chặn trước những thiết bị chủ có ưu tiên thấp hơn khi chúng muốn truy cập đến bus, điều đó có nghĩa là một máy chủ đang truy cập vào bus có thể bị buộc phải giải phóng bởi trọng tài nếu muốn ưu tiên máy chủ hơn bus. Hình 2-28c đã thể hiện 3 thiết bị chủ (1, 2, 3, thiết bị 1 là ưu tiên thấp nhất, ưu tiên cao nhất là thiết bị 3).

Thứ tự phân xử trung tâm, cũng được xem như chuỗi phân xử, là một hệ thống trong đó bộ phân xử được kết nối với các máy chủ và các máy chủ được kết nối thành chuỗi. Không kể đến việc máy chủ tạo yêu cầu cho bus, máy chủ đầu tiên ở trong chuỗi được cấp bus và chuyển sang “cấp bus” cho máy chủ tiếp theo trong chuỗi nếu/khi bus không cần thiết hơn nữa (Hình 2-29).



Hình 2-29: Sự phân xử tuần tự/chuỗi tập trung

Các hệ thống phân bố sự phân xử, điều đó có nghĩa là không có bộ phân xử trung tâm, không có mạch bổ xung (Hình 2-30). Trong các hệ thống, máy chủ tự phân xử bằng cách trao đổi ưu tiên thông tin để quyết định nếu máy chủ ưu tiên cao hơn là tạo yêu cầu đến bus, hoặc đều nhau bằng cách loại bỏ tất cả các dòng phân xử và chờ xem nếu có sự xung đột trên bus, có nghĩa là bus đang làm việc với hơn một máy chủ đang cố sử dụng nó.



Hình 2-30: Sự phân xử phân tán qua sự tự chọn

Mặt khác tùy thuộc vào bus, các bộ trọng tài bus có thể cấp 1 bus đến một máy chủ *atomically* (cho đến khi máy chủ hoàn thành việc truyền dẫn với nó) hoặc cho phép phân chia truyền dẫn, nơi mà bộ phân xử có thể tránh các thiết bị ở giữa việc thực hiện, chuyển đổi giữa các máy chủ để cho phép các máy chủ khác truy cập bus.

Một thiết bị chủ được cấp bus, chỉ có 2 thiết bị một là máy chủ và thiết bị khác phụ thuộc vào kiểu giao tiếp qua bus tại bất cứ thời điểm nào. Chỉ có 2 kiểu thực hiện một kiểu thiết bị có thể được (hoặc nhận) và/hoặc viết (truyền). Các giao dịch này có thể diễn ra giữa một trong 2 bộ xử lý (một là máy chủ, một là điều khiển I/O) hoặc bộ xử lý và bộ nhớ (một máy chủ và bộ nhớ). Trong mỗi loại giao dịch, dù đọc hay viết cũng có một vài qui tắc cụ thể mà mỗi thiết bị cần làm theo thứ tự để hoàn thành một giao dịch. Các qui tắc có thể thay đổi nhiều giữa các kiểu thiết bị thông tin, cũng như từ bus đến bus. Bộ qui tắc, thông thường được gọi là *bắt tay bus*, dạng cơ sở của bất kì giao thức bus nào.

Cơ sở của bất kì *bắt tay bus* nào là giới hạn sau cùng bởi bus định thời. Các bus dựa trên một hoặc một vài sự kết hợp đồng thời hoặc không đồng thời của hệ thống bus định thời cho phép các thành phần gắn liền với bus đồng bộ truyền hóa nó. Một bus đồng bộ bao gồm một tín hiệu đồng hồ giữa các tín hiệu khác nó truyền tải, chẳng hạn như dữ liệu, địa chỉ hoặc các thông tin điều khiển khác. Các thành phần sử dụng bus đồng bộ tất cả đều chạy giống như nhịp đồng hồ như bus, (tùy thuộc vào từng bus) và dữ liệu được truyền trên xung cao hoặc thấp của chu kỳ xung nhịp. Thứ tự làm việc của một hệ thống, các thành phần hoặc phải đồng bộ cho tốt độ xung tăng lên, hoặc tốc độ xung phải chậm cho bus lâu hơn. Một bus quá dài với 1 tốc độ xung quá nhanh (hoặc quá nhiều thành phần gắn liền với bus) sẽ gây ra độ lệch trong đồng bộ truyền đi, bởi vì việc truyền đi trong một hệ thống như vậy sẽ không đồng bộ với xung nhịp. Diễn hình các bus nhanh hơn sử dụng đồng bộ hệ thống bus định thời.

Một bus không đồng bộ truyền tín hiệu không có xung đồng hồ, nhưng thay vào đó là truyền tín hiệu “bắt tay”, chẳng hạn như yêu cầu và nhận tín hiệu. Trong khi hệ thống không đồng bộ phức tạp hơn cho các thiết bị có yêu cầu phân phối lệnh, trả lời lệnh, và như vậy một bus không đồng bộ không có vấn đề gì với độ dài của bus hoặc một số lớn hơn của các thành phần giao tiếp trên bus, bởi vì một xung nhịp không phải là cơ sở để giao tiếp đồng bộ. Tuy nhiên một bus không đồng bộ cần một “bộ đồng bộ” khác để quản lý sự trao đổi thông tin và để khóa các thông tin liên lạc.

Có hai giao thức phổ biến nhất để bắt đầu bất cứ *bắt tay bus* nào là chỉ dẫn hoặc yêu cầu một giao tiếp (hoặc đọc hoặc viết) và phụ thuộc vào đáp ứng đến chỉ dẫn giao

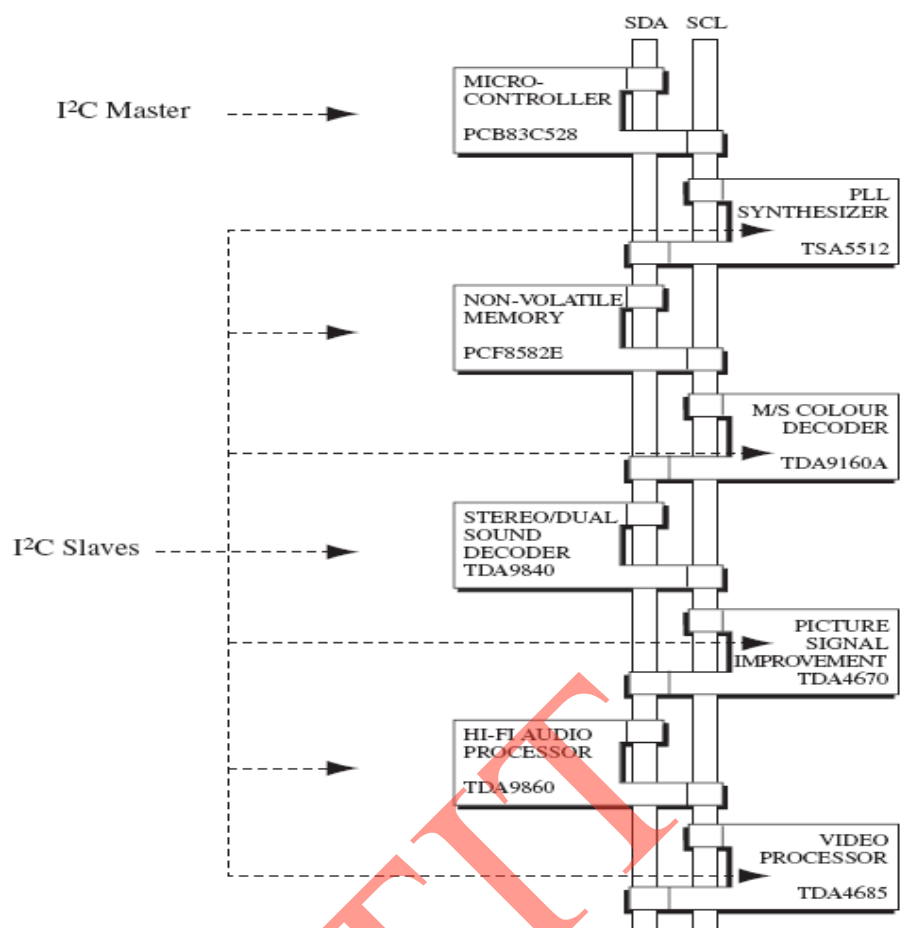
tiếp hoặc yêu cầu (ví dụ như một sự thừa nhận ACK hoặc yêu cầu ENQ). Cơ sở của hai giao thức này là điều khiển truyền tín hiệu thông qua điều khiển một dải bus riêng hoặc qua một dải dữ liệu. Dù nó là một yêu cầu dữ liệu tại bộ nhớ hay là giá trị của bộ điều khiển I/O điều khiển hoặc trạng thái đăng kí, nếu phụ thuộc vào đáp ứng trong việc xác định đến yêu cầu truyền của các thiết bị chủ, sau đó hoặc là một địa chỉ của dữ liệu liên quan đến việc truyền được trao đổi qua một dải bus riêng hoặc dải dữ liệu hoặc là địa chỉ này được truyền như một phần của việc truyền giống như yêu cầu truyền ban đầu. Nếu địa chỉ này hợp lệ, sau đó trao đổi dữ liệu trên đường truyền dữ liệu (thêm hoặc bớt một tập các nhận biết qua một dải khác hoặc ghép vào cùng một chuỗi). Chú ý giao thức bắt tay biến đổi khác với bus. Ví dụ, một bus yêu cầu truyền tải của yêu cầu và/hoặc nhận biết với mọi đường truyền, các bus khác chỉ đơn giản là cho phép truyền quảng bá của máy chủ đến tất cả các thiết bị bus, và chỉ phụ thuộc các thiết bị liên quan đến giao dịch truyền dữ liệu lại cho nơi gửi. Một ví dụ khác về sự khác nhau giữa các giao thức bắt tay, thay vì trao đổi phức tạp của điều khiển tín hiệu thông tin đang được yêu cầu một xung đồng hồ có thể là cơ sở của tất cả bắt tay.

Các bus có thể hợp thành một tập hệ thống truyền, ra lệnh như thế nào để bus truyền dữ liệu. Các hệ thống phổ biến nhất là đơn lẻ, nơi mà một địa chỉ truyền đi trước thông tin truyền của dữ liệu và chặn, nơi địa chỉ được truyền một lần cho nhiều thông tin của dữ liệu. Một hệ thống truyền bị chặn có thể tăng băng thông của bus (không có thêm không gian và thời gian cho việc truyền cùng một địa chỉ) và đôi khi cho phép dễ tách rời truyền hệ thống. Nó được dùng phổ biến trong các loại giao tiếp bộ nhớ, ví dụ như giao tiếp vùng nhớ đệm. Tuy nhiên, một hệ thống bị chặn, tác động tiêu cực đến hiệu suất của bus trong đó các thiết bị khác có thể phải chờ lâu hơn để truy cập vào bus. Một trong những thế mạnh của hệ thống truyền đơn lẻ bao gồm các thiết bị không phụ thuộc vào yêu cầu để có bộ đệm để lưu trữ địa chỉ và các thông tin của dữ liệu được liên kết với địa chỉ cũng như không sử dụng bất kỳ vấn đề gì nảy sinh với nhiều thông tin hoặc đến theo thứ tự hoặc không trực tiếp liên kết với một địa chỉ.

### **Bus không mở rộng:**

#### **Ví dụ: I2C Bus**

Các bus I2C kết nối với các bộ xử lý có kết hợp một I2C trên giao diện chip, cho phép trực tiếp truyền thông tin giữa các bộ xử lý trên bus. Một máy chủ/ máy phụ thuộc có mối quan hệ với các bộ xử lý tồn tại ở tất cả các giai đoạn, với máy chủ hoạt động như máy chủ nhận hoặc truyền. Hình 2-31, bus I2C là bus hai dây với một chuỗi dải dữ liệu ( SDA) và một chuỗi dải Clock (SCL). Bộ xử lý được kết nối qua I2C định địa chỉ bởi một địa chỉ duy nhất là một phần của một luồng dữ liệu được truyền giữa các thiết bị.



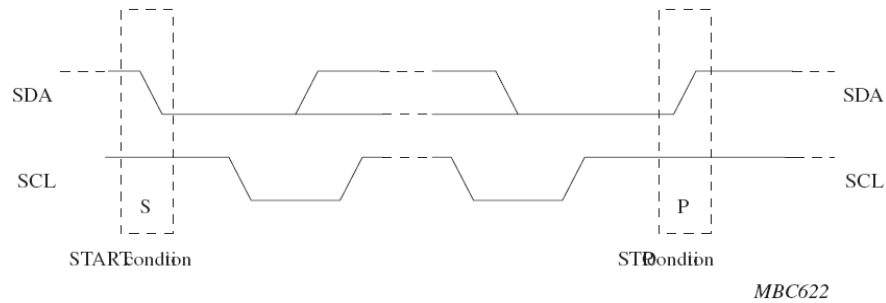
Hình 2-31: Bảng mạch mẫu TV

Máy chủ I<sup>2</sup>C bắt đầu truyền dữ liệu và tạo ra các xung nhịp cho phép truyền. Về cơ bản SCL có chu kỳ là mức cao hay thấp.

Một máy chủ sau khi sử dụng dải SDA (như SCL là chu kỳ) để chuyển dữ liệu đến các máy phụ, một phiên được bắt đầu và kết thúc như Hình 2-32. “START” là bắt đầu khi máy chủ kéo cổng SDA ở mức thấp trong khi tín hiệu SCL ở mức cao, nhưng trái lại điều kiện “STOP” bắt đầu khi máy chủ kéo SDA ở mức cao khi SCL ở mức cao.

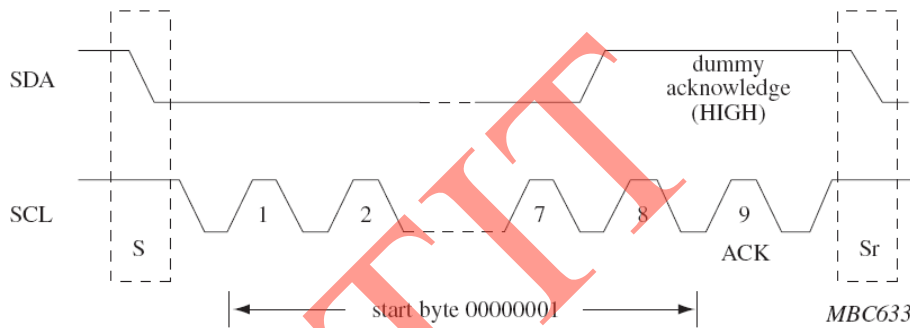
Đối với việc truyền tải dữ liệu, bus I<sup>2</sup>C là một dây bus 8 bit. Điều này có nghĩa là trong khi không có giới hạn về số lượng byte có thể được truyền đi trong một phiên, chỉ có 1 byte (8 bit) của dữ liệu được truyền đi cùng một lúc, 1 bit tại một thời điểm. Cách dịch vào trong sử dụng các tín hiệu SDA và SCL là một bit dữ liệu “đọc” mỗi lần tín hiệu SCL chuyển từ mức cao xuống mức thấp, sườn đến sườn.



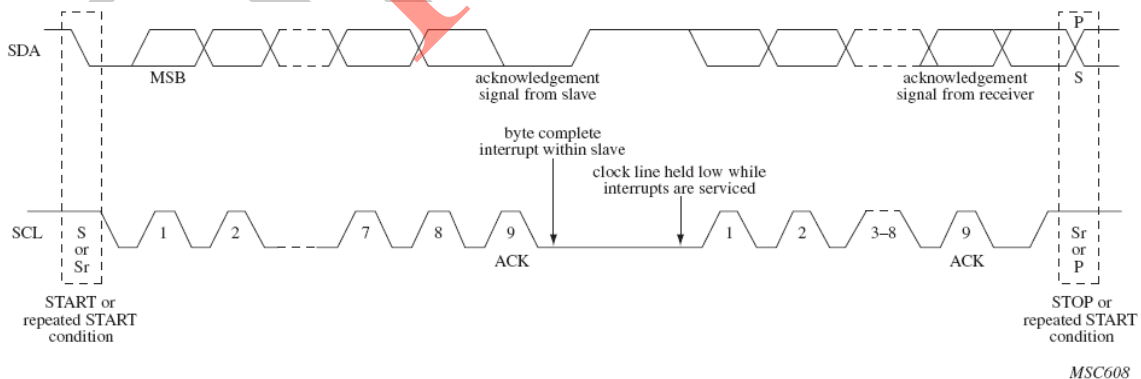


Hình 2-32: Các điều kiện START và STOP của I2C

Nếu tín hiệu SDA ở mức cao tại điểm của một sườn, sau đó bit dữ liệu sẽ được đọc là “1”. Nếu SDA ở mức thấp tại điểm của một sườn, sau đó bit dữ liệu được đọc là “0”. Một ví dụ về byte “00000001” truyền, Hình 2-33a, trong Hình 2-33b cho thấy ví dụ về hoàn thành một phiên truyền.



Hình 2-33a: Ví dụ truyền dữ liệu trên I2C



Hình 2-33b: Sơ đồ truyền dữ liệu hoàn chỉnh trên I2C

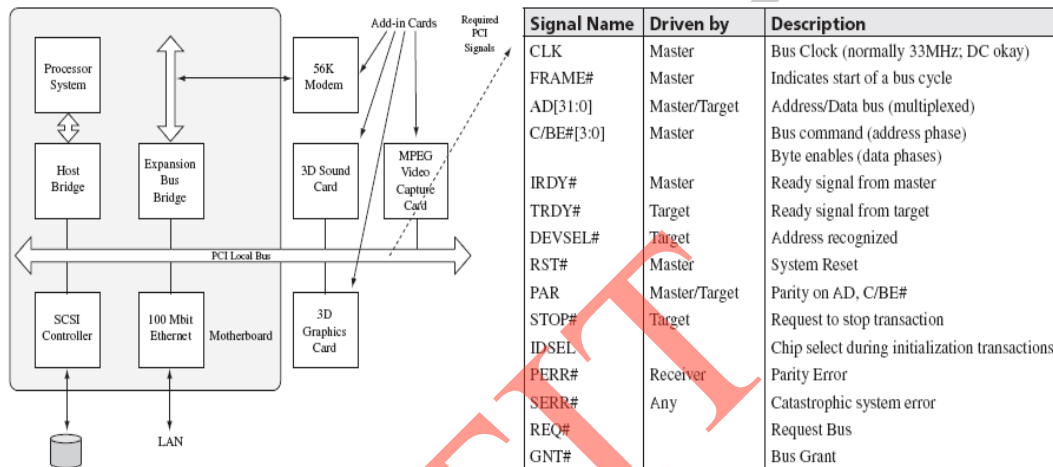
### Ví dụ bus có thể mở rộng : PCI(Peripheral Component Interconnect):

PCI Local Bus Specification Revision 2.1, xác định yêu cầu (điện, thời gian, giao thức ...) của việc xử lý một bus PCI. PCI là một bus đồng bộ, điều đó có nghĩa là nó đồng bộ hóa thông tin liên lạc bằng cách sử dụng xung clock. Tiêu chuẩn định nghĩa thiết kế một



bus PCI với ít nhất 33MHz xung nhịp (lên đến 66 MHz) và độ rộng của một bus ít nhất là 32 bit (lên đến 64 bit) có thể đưa ra thông lượng tối thiểu xấp xỉ khoảng 132 Mbytes/giây và lên đến 528 Mbytes/giây, lớn nhất với 64bit truyền cho 66 MHz xung, PCI chạy ở một trong những tốc độ xung, không kể đến tốc độ xung nhịp mà tại đó các thành phần kèm theo nó đang hoạt động.

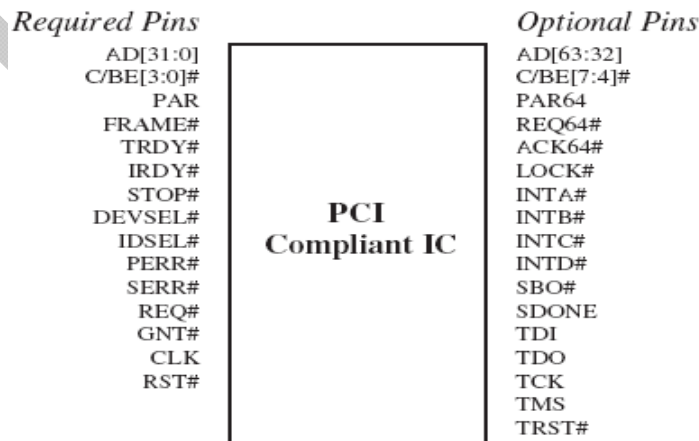
Ở Hình 2-34 bus PCI có 2 giao diện kết nối, một nội bộ giao diện PCI kết nối với bảng chính (để xử lí...) qua kênh EIDE và mở rộng giao diện PCI, bao gồm các khe bên trong PCI để cắm thẻ (audio, video...). Các giao diện mở rộng làm cho PCI mở rộng bus, nó cho phép các phần cứng cắm vào bus, cho toàn bộ hệ thống tự động điều chỉnh và hoạt động chính xác.



Hình 2-34: Bus PCI

### 2.1.5.2 GHÉP BUS VỚI CÁC THÀNH PHẦN BẢN MẠCH KHÁC

Các bus thay đổi trong các đặc tính vật lí của nó, các đặc tính này tương ứng với các thành phần mà bus kết nối, chủ yếu là các sơ đồ chân của bộ xử lí và các chip bộ nhớ, nó phản ánh các tín hiệu mà bus có thể truyền (xem Hình 2-35).



Hình 2-35: IC tương thích chuẩn PCI

Bên trong một kiến trúc, cũng có hỗ trợ chức năng giao thức bus. Như ví dụ, MPC860 bao gồm bus điều khiển tích hợp I<sup>2</sup>C.

Bus I<sup>2</sup>C là bus với 2 tín hiệu: SDL và SCL, cả hai tín hiệu này được thể hiện bên trong khối sơ đồ của bộ điều khiển PowerPC I<sup>2</sup>C. Do I<sup>2</sup>C là bus đồng bộ, một thiết bị tạo ra tốc độ baud bên trong bộ điều khiển cung cấp xung nhịp nếu PowerPC hoạt động như một máy chủ, theo 2 bộ (nhận và truyền) bao phủ xử lý và điều hành sự truyền tải bus.

Trong bộ điều khiển tích hợp I<sup>2</sup>C, địa chỉ và thông tin dữ liệu được truyền qua bus thông qua thanh ghi truyền dữ liệu và ra thanh ghi dịch. Khi MPC860 nhận dữ liệu, dữ liệu được truyền vào bên trong thanh ghi nhận dữ liệu qua một thanh ghi dịch.

### 2.1.5.3 HIỆU SUẤT BUS.

Năng suất của bus được tính qua băng thông của nó, số lượng dữ liệu mà bus có thể truyền trong một khoảng thời gian, một cấu trúc của bus cả cấu trúc vật lý và giao thức liên kết của nó sẽ tác động đến hiệu suất của nó. Trong giới hạn của giao thức, ví dụ, hệ thống bắt tay đơn giản băng thông cao hơn. Thực tế các thiết kế vật lý của bus (độ dài, số dòng, số lượng các thiết bị được hỗ trợ...) giới hạn hoặc tăng hiệu suất của nó. Các bus ngắn hơn, các thiết bị được kết nối ít, nhiều dữ liệu hơn thường nhanh hơn các bus và băng thông cao hơn.

Số lượng các dải bus và cách sử dụng các dải bus – Ví dụ, cho dù có nhiều đường dây riêng cho mỗi tín hiệu hay có nhiều tín hiệu đa bội ít được chia sẻ trên đường là các yếu tố tác động bổ xung cho băng thông của bus. Càng nhiều dải bus thì càng có nhiều dữ liệu có thể được truyền qua cùng một thời điểm. Ít dải hơn có nghĩa là nhiều dữ liệu có thể chia sẻ vào dải để truyền, kết quả là có ít dữ liệu được truyền trong cùng một thời gian hơn. Liên quan đến chi phí, hãy lưu ý sự gia tăng trong chất dẫn điện trên bảng mạch, trong trường hợp dây của bus làm tăng chi phí trên bảng. Tuy nhiên, chú ý rằng, sự dồn dãi sẽ đưa ra độ trễ trên cuối mỗi đường truyền, các yêu cầu ở cuối bus đến tín hiệu dồn kênh và phân kênh được tạo thành từ các tín hiệu thông tin khác.

Các yếu tố khác góp phần vào băng thông của bus là số bit dữ liệu mà một bus có thể truyền trong một chu kỳ bus, đây gọi là độ rộng của bus. Các bus thường có băng thông là số mũ nhị phân 2, chẳng hạn như  $1(2^0)$  cho các bus với độ rộng bus nối tiếp,  $8(2^3)$  bit,  $16(2^4)$  bit,  $32(2^5)$  bit... Ví dụ, cho dữ liệu 32 bit cần được truyền, nếu một bus nói riêng có độ rộng là 8bit, sau đó dữ liệu được chia và gửi trong 4 lần truyền riêng; nếu bus có độ rộng 16 bit, thì có 2 gói dữ liệu riêng để truyền đi, bus có 32 bit dữ liệu truyền một gói, một bit bất kể lúc nào cũng có thể được truyền. Độ rộng của bus giới hạn băng thông của bus vì nó giới hạn số bit dữ liệu có thể được truyền trong bất kỳ một giao dịch nào. Độ trễ có thể xảy ra trong mỗi phiên truyền bởi vì giao thức bắt tay, lưu lượng bus và tần số xung nhịp khác các thành phần giao tiếp, đặt các thành phần này trong hệ thống trễ, chẳng hạn như trạng thái chờ. Độ trễ tăng vì số gói dữ liệu cần được truyền cũng tăng. Vì vậy độ rộng của băng thông lớn hơn, độ trễ ít hơn và băng thông lớn hơn.

Đối với bus có giao thức bắt tay phức tạp hơn, hệ thống truyền thực hiện có thể ảnh hưởng lớn đến hiệu suất. Một khối hệ thống truyền tải cho phép băng thông lớn hơn qua hệ thống truyền tải đơn lẻ, bởi vì có ít sự trao đổi giao thức bắt tay trên mỗi khối với những từ đơn lẻ, các byte dữ liệu. Truyền tải khối có thể thêm thời gian chờ để các thiết bị chờ lâu hơn cho việc truy cập bus. Từ khi một khối truyền tải dựa trên chuyển giao

một khối kéo dài lâu hơn là truyền tải đơn lẻ cơ sở chuyển giao. Một giải pháp phổ biến cho loại thời gian chờ là bus cho phép chia chuyển giao, là nơi mà bus được hoạt động trong suốt quá trình bắt tay, chẳng hạn như trong khi chờ đợi xác nhận trả lời. Điều này cho phép các giao dịch khác diễn ra và cho phép các bus rồi còn lại chờ thiết bị chuyển giao. Tuy nhiên, nó gắn với độ trễ của các giao dịch bằng cách yêu cầu bus có được nhiều hơn một lần giao dịch duy nhất.

## **2.2 Các thành phần phần mềm của hệ thống:**

### **2.2.1. Trình điều khiển thiết bị**

#### ***Các ví dụ về bộ điều khiển cổng I/O***

Các bộ phận của hệ thống con I/O yêu cầu một số dạng quản lý phần mềm bao gồm các thành phần được tích hợp sẵn trong bộ xử lý chủ, như bộ điều khiển thụ động I/O. Các bộ điều khiển I/O đều gồm có một bộ trạng thái và các thanh ghi điều khiển được dùng để điều khiển bộ xử lý và kiểm soát trạng thái của nó. Phụ thuộc vào hệ thống con I/O, thông thường tất cả hoặc một vài hệ thống trong 10 đặc trưng từ danh sách chức năng bộ điều khiển thiết bị đã được giới thiệu ở phần đầu của chương sẽ thực thi trong bộ điều khiển I/O, bao gồm:

- I/O Startup, là phần khởi động của nguồn cấp cho I/O hoặc khởi động lại.
- I/O Shutdown, thiết lập I/O trở về trạng thái tắt nguồn của nó
- I/O Disable, cấp phát chương trình khác để ngăn chặn hoạt động của I/O
- I/O Enable, cấp phát chương trình khác để kích hoạt hoạt động của I/O
- I/O Acquire, cấp phát chương trình khác để khuếch đại truy cập đơn lẻ (chốt) tới I/O
- I/O Release, cấp phát chương trình khác để nghỉ (mở chốt) I/O
- I/O Read, cấp phát chương trình khác để đọc dữ liệu từ I/O
- I/O Write, cấp phát chương trình khác để viết dữ liệu lên I/O
- I/O Install, cấp phát chương trình khác để cài đặt hoạt động mới của I/O
- I/O Uninstall, cấp phát chương trình khác để gỡ bỏ hoạt động của I/O đã được cài đặt.

Các chương trình con khởi động Ethernet và RS232 I/O cho PowerPC và các cấu trúc ARM được cung cấp như trong ví dụ của hệ thống thiết bị khởi động (thiết lập) I/O. Các ví dụ đó chỉ ra cách I/O được thực thi trong các cấu trúc liên hợp, như PowerPC và ARM, và nó có thể được dùng như một bản chỉ dẫn để nắm bắt cách viết các hệ thống I/O lên các bộ xử lý phức hợp hoặc kém phức hợp hơn PowerPC và cấu trúc ARM.

#### **Ví dụ khởi tạo Driver RS-232:**

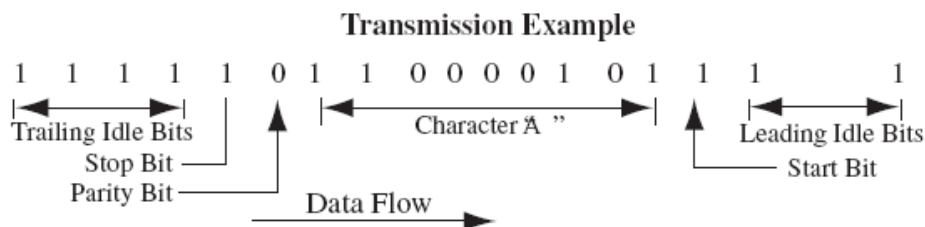
Một trong những chuỗi giao thức I/O không đồng bộ được thiết lập là RS-232 hoặc EIA-232, dựa trên nền tảng chuẩn của hội công nghiệp điện tử. Các chuẩn này xác minh những bộ phận chính của bất cứ hệ thống gốc RS-232 nào được thực thi trong phần cứng.

Firmware (software) yêu cầu kích hoạt ánh xạ đặc trưng tới phiên thấp hơn của lớp liên kết dữ liệu vật lý. Các bộ phận phần cứng có thể được ánh xạ hết tới lớp vật lý của mô hình OSI, nhưng nó sẽ không được nói đến trong phần này.

Bộ phận RS-232 được tích hợp sẵn trong bộ xử lý chủ được gọi là giao diện RS-232, nó có thể được lập trình truyền dẫn đồng bộ hay không đồng bộ. Ví dụ, loại dẫn tuyến

không đồng bộ, chỉ có firmware (software) được thực thi RS-232 nằm trong bộ phận UART, nó sẽ thực thi việc truyền dẫn dữ liệu lần lượt.

Dữ liệu được truyền không đồng bộ qua RS-232 nằm trong một chuỗi các bit mà được chuyển với tốc độ không đổi. Khung truy cập bởi UART nằm trong khuôn mẫu được nêu trong Hình 2-36.



Hình 2-36: Sơ đồ khung RS-232

### 2.2.2. Hệ điều hành thời gian thực

Hệ điều hành thời gian thực là một hệ điều hành hỗ trợ cấu trúc của hệ thống thời gian thực.

#### Trạng thái định thời của OS nên được dự đoán trước:

Đối với mỗi dịch vụ của hệ điều hành (OS), ràng buộc trong thời gian thực hiện cần được đảm bảo. Trên thực tế có các mức độ dự đoán khác nhau. Ví dụ, có thể có bộ của dịch vụ hệ điều hành gọi mà ràng buộc được biết và sự thay đổi của thời gian thực hiện là không đáng kể. Các lệnh gọi như là “đưa cho tôi thời gian trong ngày” có thể ở trong lớp này. Đối với các lệnh gọi khác có thể thay đổi rất lớn. Các lệnh gọi như “đưa cho tôi 4MB bộ nhớ độc lập” có thể rơi ở trong lớp thứ hai này. Đặc biệt, bản liệt kê danh sách của RTOS cần phải được xác định (tiêu chuẩn Java không chú ý về mặt này, như là không có thứ tự thực hiện đối với thực thi “chuỗi” – các luồng được qui định). Một trường hợp đặc biệt khác, chúng ta đề cập đến việc thu thập dữ liệu rác. Trong phạm vi Java, có nhiều cố gắng đã thực hiện để cung cấp thu gom dữ liệu rác.

Cũng có thể trong suốt thời gian lệnh ngắt có thể không có khả năng gây trở ngại phá hủy giữa các nhiệm vụ (nó là một cách rất đơn giản đảm bảo loại trừ lẫn nhau trên một hệ thống bộ vi xử lý đơn). Trong các giai đoạn, các ngắt bị vô hiệu hóa có thể hoàn toàn ngăn để tránh sự chậm trễ không thể dự đoán được trong việc xử lý các biến cố quan trọng.

Đối với RTOS thực hiện hệ thống tập tin, nó có thể cần thiết để thực hiện các tập tin kề nhau( file được lưu trữ kề nhau trong đĩa) để hạn chế sự chuyển động của đầu đĩa không dự đoán trước được.

#### OS cần phải được quản lý thời gian và lịch trình các nhiệm vụ.

Lịch trình được định nghĩa như sự ánh xạ từ tập các nhiệm vụ đến các khoảng thời gian thực hiện. Bao gồm ánh xạ để bắt đầu như là một trường hợp đặc biệt. Ngoài ra, OS có thể nhận biết được nhiệm vụ đúng thời hạn để OS có thể áp dụng lịch trình kỹ thuật thích hợp (trong đó lịch trình là hoàn toàn thực hiện gián tiếp và OS chỉ cần cung cấp các dịch vụ để bắt đầu nhiệm vụ ở thời điểm cụ thể hoặc các mức độ ưu tiên). OS cần

phải cung cấp các dịch vụ thời gian chính xác với độ chính xác cao. Thời gian dịch vụ được yêu cầu, ví dụ, để phân biệt giữa bản gốc và các lỗi con. Ví dụ, nó có thể giúp xác định năng lượng của máy, để chịu trách nhiệm cho việc mất nguồn.

### **OS cần phải nhanh**

Ngoài việc dự đoán được, OS cần phải có khả năng hỗ trợ các ứng dụng với thời hạn trên tỉ lệ giây. Mỗi RTOS bao gồm một thời gian thực gọi là hệ thống điều hành chính. Bộ phận quản lý chính này là nguồn được tìm thấy trong mỗi hệ thống, bao gồm bộ xử lý, bộ nhớ, và hệ thống giờ. Bộ phận bảo vệ các thiết bị máy (loại từ an toàn, an toàn, hay lí do bảo mật) không cần có mặt.

Chức năng chính trong bộ phận chính bao gồm các quản lí nhiệm vụ, nhiệm vụ đồng bộ và truyền dữ liệu, quản lí thời gian, quản lí bộ nhớ.

Trong khi một số RTOS được thiết kế cho các ứng dụng gán nói chung, các ứng dụng khác tập trung trên một khu vực cụ thể. Ví dụ, OSEK/VDX OS tập trung điều khiển tự động. Do tập trung này, nó là một hệ điều hành khá nhỏ.

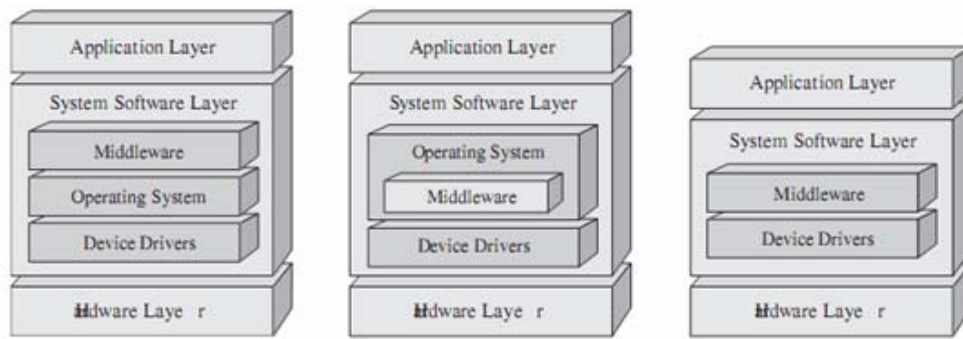
Tương tự, trong một số RTOS cung cấp API chuẩn những cái khác đến cùng với đặc điểm riêng của chúng, API độc quyền. Ví dụ, một vài RTOSs phù hợp với POSIX RT mở rộng cho UNIX, với OSEK/VDX OS hoặc với ITRON đặc biệt phát triển nhiều ở nhật bản. Nhiều loại RT chính của OS có riêng API. ITRON, được đề cập trong trường hợp này là một RTOS hoàn chỉnh sử dụng liên kết cấu hình.

### **2.2.3. Middleware**

Khái quát chung nhất, middleware là phần mềm hệ thống bất kì, nó không chỉ là một bộ phận chính của hệ điều hành, các bộ điều khiển thiết bị, hay các phần mềm ứng dụng. Chú ý rằng một vài OS có thể tích hợp middleware vào trong các bộ thực thi của hệ điều hành. Các middleware hệ thống nhúng thông thường được đặt bên trong các bộ điều khiển thiết bị hay được đặt ở phần trên cùng của hệ điều hành, và thi thoảng có thể được hợp nhất lại bên trong chính hệ điều hành.

Middleware thường là phần mềm trung gian giữa các phần mềm ứng dụng với bộ phận chính hay là phần mềm điều khiển thiết bị. Middleware cũng là phần mềm trung gian và điều khiển các phần mềm ứng dụng khác. Đặc biệt, middleware là một lớp cấu trúc chung được sử dụng trên hầu hết các thiết bị nhúng với 2 hoặc nhiều ứng dụng với mục đích để đáp ứng được tính phức tạp, tính bảo mật, tính linh động, tính liên kết, truyền thông với nhau, tính tương thích giữa các ứng dụng. Một trong các thế mạnh chính khi sử dụng middleware đó là nó cho phép giảm thiểu được sự phức tạp của các ứng dụng bởi cơ sở hạ tầng phần mềm trung tâm. Tuy nhiên, khi giới thiệu về middleware hay giới thiệu về một hệ thống, cái cần giới thiệu đầu tiên là cái nhìn tổng thể, nó có thể tác động rất lớn đến tính mở rộng và hiệu năng. Trong một thời gian ngắn, middleware tác động tới hệ thống nhúng tại tất cả các lớp.





Hình 2-37: Middleware trong mô hình hệ nhúng

Trên đây có thể kể ra các thành phần middleware khác nhau như phần middleware định hướng thông báo (MOM-message oriented middleware), các thực thể trung gian yêu cầu đối tượng (ORBs-object request brokers), gọi thủ từ xa (remote procedure call-RPCs), cơ sở dữ liệu/truy nhập cơ sở dữ liệu, và các giao thức mạng trên tầng driver thiết bị. Tuy thế, hầu hết các loại middle nhìn chung đều rơi vào một trong 2 loại sau:

+ **General –purpose**: nghĩa là chúng được thực thi trên các thiết bị khác nhau như giao thức mạng nằm bên trên lớp điều khiển thiết bị và nằm bên dưới lớp ứng dụng trong mô hình OSI, các hệ thống tập tin hay trong một vài thiết bị ảo như JVM.

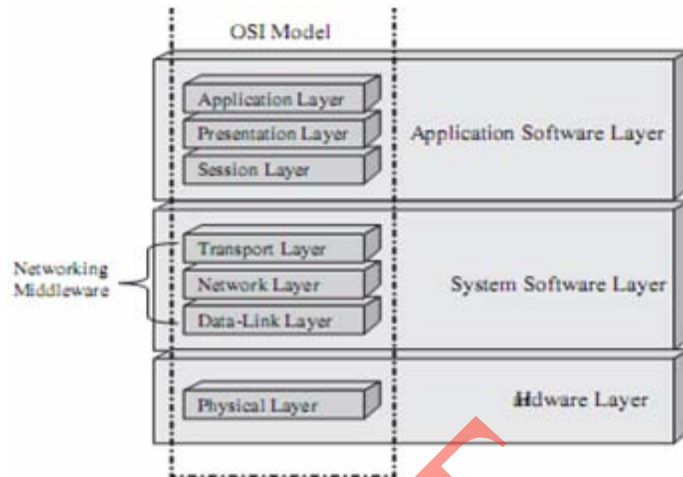
+ **Market-specific**: nghĩa là chúng là độc nhất trong một gia đình riêng biệt của các hệ thống nhúng như một phần mềm trên nền tiêu chuẩn TV số đặt trên hệ điều hành hay trên JVM.

Dù chúng là general purpose hay market –specific, thì thành phần middleware có thể được phân chia rõ hơn theo “**đóng**”, nó là các phần mềm được cung cấp bởi một công ty với bản quyền sử dụng cho phép ta có thể sử dụng nó, hoặc “**mở**”, nghĩa là nó là một chuẩn qui định bởi các hội đồng công nghiệp và chúng được cài đặt hay cấp phép bởi bất kì bên nào có liên quan.

Khi người ta tìm ra một công nghệ có thể đáp ứng được tất cả các yêu cầu của ứng dụng riêng biệt nào đó, thì lúc này các hệ thống nhúng trở nên phức tạp hơn và đòi hỏi phải có thêm nhiều hơn các linh kiện nhúng. Trong trường hợp đó, các linh kiện middleware cá nhân được chọn lựa tùy thuộc vào khả năng thao tác với các linh kiện khác, để tránh được các vấn đề sau này khi chúng được tích hợp lại. Trong một vài trường hợp, các gói middleware tích hợp của thành phần middleware đã mang sẵn tính thương mại, sử dụng cho hệ thống nhúng như các giải pháp nhúng Java của SUN, Microsoft’s, Framework, và CORBA từ nhóm quản lí đối tượng (OMG), để đặt tên cho một vài hệ thống nhúng. Nhiều nhà cung cấp hệ điều hành nhúng cũng cung cấp các gói middleware tích hợp chạy theo kiểu “out of box” với hệ điều hành tương ứng và nền tảng phần cứng của họ.

**Ví dụ middleware driver mạng:**

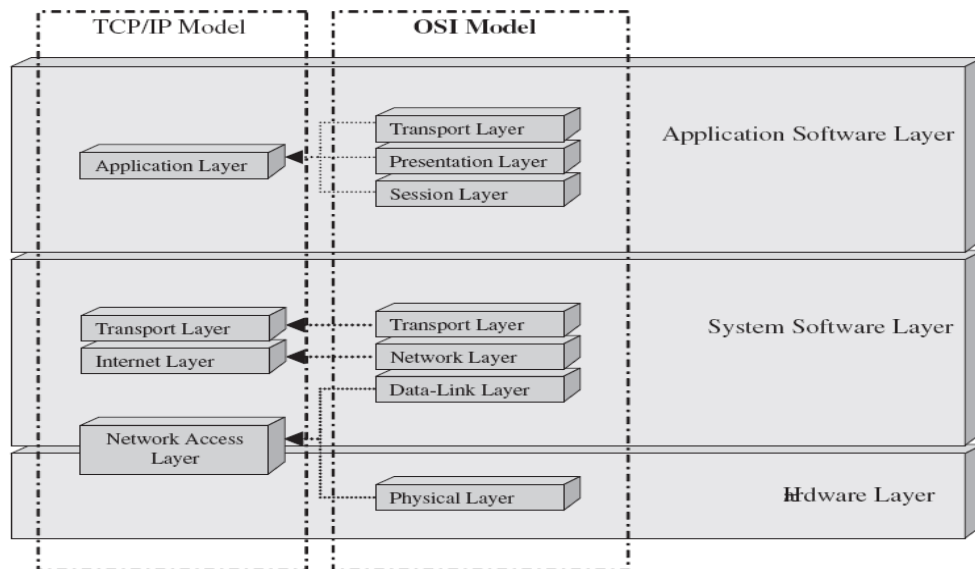
Một trong các cách đơn giản nhất để có thể hiểu về các thành phần cần có để lắp đặt một mạng trong một thiết bị nhúng để hiển thị hóa các thành phần của mạng theo mô hình OSI và liên quan tới mô hình hệ thống nhúng. Trong Hình 2-38, phần mềm được đặt nằm trên lớp liên kết dữ liệu và lớp phiên, nó có thể được xem như thành phần phần mềm của middleware mạng.



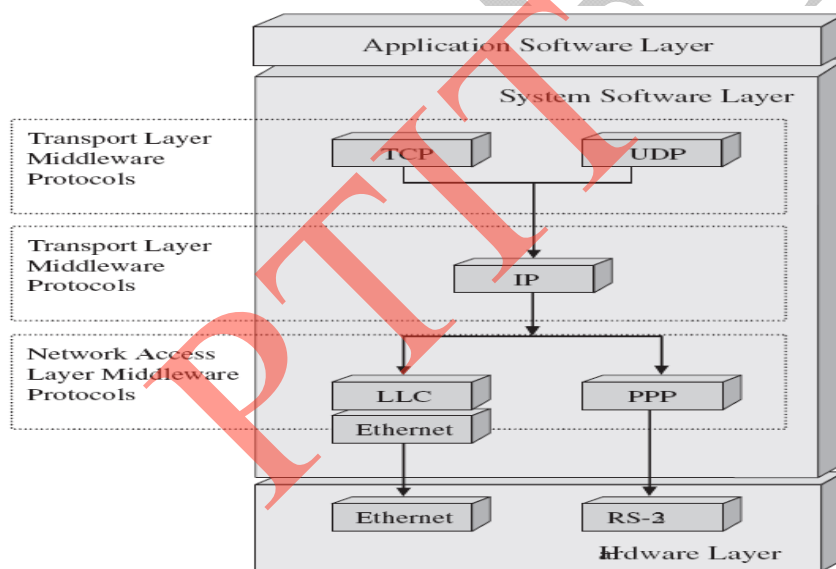
Hình 2-38: Mô hình OSI và middleware

Trong ví dụ ở phần này, ta sẽ đề cập tới hai giao thức UDP và IP (Hình 2-40a và b), đây là các giao thức trong chồng giao thức TCP/IP và được thực thi như middleware. Như đã giới thiệu trong chương 2, mô hình TCP/IP được tạo nên từ 4 lớp: lớp truy nhập mạng, lớp mạng, lớp giao vận, và lớp ứng dụng. Lớp ứng dụng của mô hình TCP/IP hợp nhất chức năng của 3 lớp trên cùng trong mô hình OSI (lớp ứng dụng, lớp trình diễn, và lớp phiên), và lớp truy nhập hợp nhất chức năng của hai lớp là lớp vật lý và lớp liên kết dữ liệu. Lớp internet tương đương với lớp mạng trong mô hình OSI và lớp giao vận thì giống nhau cho cả hai mô hình. Điều đó có nghĩa là, khi xem xét mô hình TCP/IP cần chú ý, middleware mạng nằm bên dưới lớp giao vận, lớp internet và nằm bên trên lớp truy nhập mạng (Hình 2-39a).





Hình 2-39a: Sơ đồ khối mô hình OSI, TCP/IP và mô hình hệ nhúng

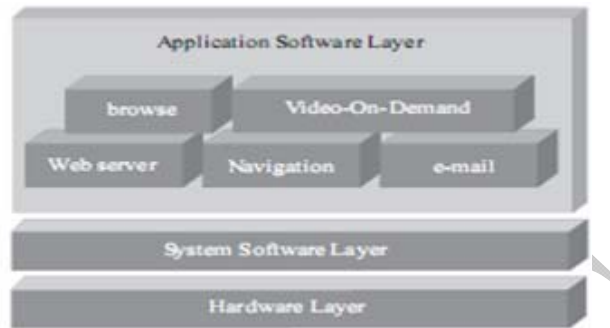


Hình 2-39b: Sơ đồ khối mô hình TCP/IP và các giao thức

## 2.2.4 Phần mềm ứng dụng

Là loại phần mềm lớp trên cùng trong hệ thống nhúng (***application software***). Trong hình 2-40, phần mềm ứng dụng nằm ở phần trên cùng của lớp phần mềm hệ thống và phụ thuộc, chịu sự quản lý và chạy các ứng dụng tùy thuộc vào phần mềm hệ thống. Nó là một phần mềm nằm bên trong lớp ứng dụng, qui định loại thiết bị nào được dùng trong hệ thống nhúng, bởi vì chức năng của một lớp ứng dụng là đặc tả mục đích cao nhất của hệ thống nhúng đó và làm tất cả các công việc liên quan gần nhất với người sử dụng hoặc quản lý thiết bị đó nếu cái thiết bị nào tồn tại. Khi người sử dụng bấm nút thì có thể kích hoạt chức năng điều khiển của một cách trực tiếp bằng cách đóng/ngắt theo

tuần tự, tốt hơn so với một ứng dụng phụ thuộc vào một người lập trình viên làm công việc đó bằng tay.



Hình 2-40: Lớp ứng dụng và mô hình hệ nhúng

Giống như các tiêu chuẩn nhúng, các ứng dụng nhúng có thể được chia theo market specific và general purpose.

- + marker specific (thực thi chỉ trên duy nhất một kiểu thiết bị, như các ứng dụng yêu cầu video trong một TV số tương tác).

- + general purpose (có thể chỉ được thực thi trên nhiều loại thiết bị khác nhau, như một trình duyệt web).

## Câu hỏi ôn tập

1. Sự khác biệt giữa von Neumann model và Harvard model?
2. Vẽ sơ đồ phân cấp bộ nhớ trong hệ thống và giải thích.
3. Bản đồ bộ nhớ là gì?
4. Các loại bộ nhớ nào thường được sử dụng trong hệ thống nhúng.
5. Loại bộ nhớ nào tích hợp trên chip, trên board.
6. Việc quản lý bộ nhớ ngoài có thể thực hiện bằng các phương pháp nào?
7. Mục đích của I/O trên bảng mạch là gì?
8. Định nghĩa bus?
9. Mục đích của bus là gì?
10. Lấy ví dụ một số loại bus phổ biến.
11. Sự khác biệt giữa bus mở rộng được và không mở rộng được
12. Sự khác biệt giữa bus master và slave
13. Tại sao lại cần trọng tài bus?
14. Trọng tài bus làm nhiệm vụ gì?
15. Nêu 3 loại trọng tài bus phổ biến?
16. Vẽ sơ đồ các cơ chế của trọng tài bus và giải thích.
17. Sự khác biệt giữa serial và parallel I/O.
18. Sự khác biệt giữa UART và SPI.
19. Vẽ sơ đồ hoạt động của I2C và giải thích các tín hiệu.
20. Vẽ biểu đồ hoạt động theo thời gian của I2C và giải thích.
21. Vẽ sơ đồ hoạt động của SPI và giải thích các tín hiệu.
22. Vẽ biểu đồ hoạt động theo thời gian của SPI và giải thích.
23. Trình điều khiển thiết bị (device driver) là gì?
24. Vẽ sơ đồ các vị trí có thể của lớp trình điều khiển thiết bị trong kiến trúc hệ thống nhúng.
25. Liệt kê và mô tả chức năng của các hàm trong trình điều khiển thiết bị.
26. Hệ điều hành nhúng làm chức năng gì?
27. Vẽ sơ đồ vị trí có thể của hệ điều hành nhúng trong phân lớp hệ thống nhúng.
28. Nhân hệ điều hành (kernel) làm những nhiệm vụ chính nào?
29. Middleware là gì?
30. Vẽ sơ đồ vị trí có thể của middleware trong phân lớp hệ thống nhúng.
31. Liệt kê các loại middleware
32. Định nghĩa phần mềm ứng dụng?
33. Phần mềm ứng dụng nằm ở vị trí nào trong phân lớp hệ thống nhúng.
34. Phân loại phần mềm ứng dụng?

## CHƯƠNG 3 - HỆ ĐIỀU HÀNH THỜI GIAN THỰC DÙNG CHO CÁC HỆ THỐNG NHÚNG

### 3.1 Yêu cầu chung cho các hệ điều hành thời gian thực

Thời gian thực là một khái niệm rất khó định nghĩa. Theo nghĩa đen, hệ thống phải phản ứng tức thì, thích hợp theo yêu cầu. Theo đúng ý nghĩa, thời gian thực của một sự kiện nghĩa là nó xảy ra ngay lập tức. Một ví dụ khi ta chuyển khoản tiền giữa các tài khoản, đối với một hệ thống thời gian thực, chúng ta sẽ được cập nhật tài khoản ngay lập tức. Tuy nhiên, đối với một hệ thống không phải thời gian thực, việc xử lý sẽ được thực hiện vào cuối mỗi ngày, do đó việc cập nhật có thể phải đợi đến ngày hôm sau. Một ví dụ khác là một hệ thống đa phương tiện, khi chơi các file media, hệ thống này phải đủ khả năng xử lý video, audio đúng theo thời gian của chuẩn dữ liệu, nếu không, sẽ xảy ra hiện tượng giật hình, trễ tiếng, trễ hình hoặc các lỗi khác.

Ngoài việc phản ứng đủ nhanh, hệ thống thời gian thực cần phải tin cậy và chính xác. Đối với các hệ thống điều khiển như trong ô tô, trong dây chuyền sản xuất, độ chính xác rất quan trọng.

Tùy theo mức độ của yêu cầu về thời gian, hệ thống thời gian thực có thể phân thành thời gian thực cứng (hard real-time) hay thời gian thực mềm (soft real-time):

- Thời gian thực cứng (hard real-time): Một hệ thống thời gian thực cứng cần một sự đảm bảo về thời gian đáp ứng trong trường hợp xấu nhất. Cả hệ thống bao gồm hệ điều hành, ứng dụng, phần cứng, .... phải được thiết kế đảm bảo các yêu cầu về thời gian đáp ứng. Thời gian đáp ứng có thể là ms, ns nhưng chúng phải luôn được đảm bảo. Nếu không có thể sẽ có những hậu quả nghiêm trọng đối với hệ thống. Ví dụ như các hệ thống quốc phòng, các hệ thống điều khiển xe cộ hoặc máy bay, các hệ thống thu thập dữ liệu, các thiết bị y tế...
- Thời gian thực mềm (soft real-time): Đối với hệ thống thời gian thực mềm, việc đáp ứng yêu cầu về thời gian không phải lúc nào cũng cần thiết. Ví dụ như hệ thống xem phim, thỉnh thoảng có thể hệ thống bị treo, hoặc giật hình. Những sự cố này không có hậu quả lớn. Tuy nhiên nếu hệ thống không phản ứng đúng thời gian quá nhiều lần, hệ thống có thể coi như không đạt yêu cầu. Ví dụ như thoại VoIP, hệ thống audio, video,...

POSIX 1003.1b định nghĩa thời gian thực cho hệ điều hành là khả năng của hệ điều hành để cung cấp mức độ của dịch vụ trong một thời gian giới hạn.

Các đặc tính sau có thể được gán với hệ thống thời gian thực:

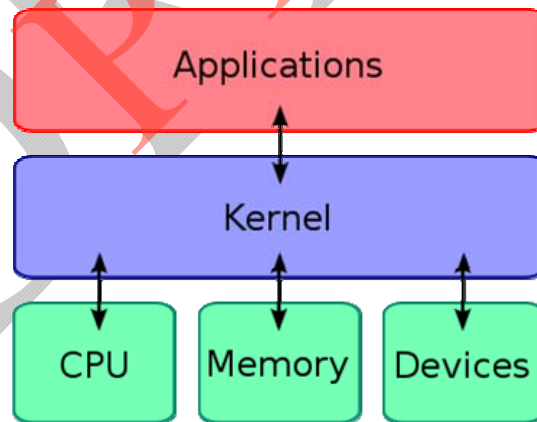
- Đa nhiệm/Đa luồng (Multitasking/multithreading): Một hệ điều hành thời gian thực phải có khả năng xử lý đa nhiệm và đa luồng.

- Cơ chế ưu tiên (Priorities): Hệ điều hành thời gian thực phải có cơ chế ưu tiên cho các tác vụ. Các chức năng có yêu cầu giới hạn về thời gian cao cần được ưu tiên xử lý.
- Thừa kế ưu tiên (Priority inheritance): Hệ điều hành thời gian thực phải có cơ chế thừa kế ưu tiên.
- Chiếm quyền (Preemption): Hệ điều hành thời gian thực cần có tính năng chiếm quyền, có nghĩa là khi một tác vụ có mức ưu tiên cao hơn sẵn sàng chạy, nó có thể chiếm quyền của một tác vụ có mức ưu tiên thấp hơn.
- Trễ ngắt (Interrupt latency): Thời gian trễ ngắt là thời gian từ khi một ngắt từ phần cứng xảy ra đến khi hàm xử lý ngắt chạy. Một hệ điều hành thời gian thực cần có trễ ngắt có thể đoán được và thường càng nhỏ càng tốt.
- Trễ lập lịch (Scheduler latency): Đây là thời gian bắt đầu từ khi tác vụ có khả năng chạy đến khi tác vụ thực sự chạy. Một hệ điều hành thời gian thực cần có trễ lập lịch có thể xác định được.
- Đồng bộ và thông tin giữa các process (Interprocess communication and synchronization): Kiểu thông tin phổ biến nhất giữa các tác vụ trong một hệ thống nhúng là gửi các bản tin. Một hệ điều hành thời gian thực cần có một cơ chế xử lý bản tin ở thời gian không đổi.
- Phân phối bộ nhớ động (Dynamic memory allocation): Một hệ điều hành thời gian thực cần cung cấp các hàm cung cấp bộ nhớ với thời gian cố định.

### 3.2 Các chức năng chính của phần lõi trong hệ điều hành thời gian thực

#### 3.2.1. Kernel

Trong một hệ điều hành bất kỳ, kernel là thành phần trung tâm nhất của hệ điều hành, nó là cầu nối giữa các ứng dụng và việc xử lý dữ liệu ở mức phần cứng.



Hình 3-1. Kernel trong hệ thống

Nhiệm vụ chính của kernel là quản lý tài nguyên hệ thống, các tài nguyên này thường bao gồm:

- CPU: Kernel chịu trách nhiệm quyết định chương trình nào sẽ được chạy trên CPU hoặc các CPU tại một thời điểm nhất định.
- Bộ nhớ: Kernel chịu trách nhiệm quyết định mỗi tác vụ sẽ được sử dụng phần bộ nhớ nào, và xác định sẽ làm gì khi không đủ bộ nhớ.

- Các thiết bị vào/ra (I/O devices): Kernel cung cấp các yêu cầu từ các ứng dụng để thực hiện các tính năng truy cập vào/ra cho các thiết bị tương ứng.

### 3.2.2 Tác vụ và đa nhiệm

#### Polling và ngắt

Các hệ thống thời gian thực được hiểu là được điều khiển tùy theo sự kiện, có nghĩa là chức năng chính của hệ thống là phản ứng theo sự kiện xảy ra. Có hai cách tiếp cận vấn đề này: dùng polling hoặc ngắt.

Ví dụ vòng lặp polling như sau:

```
int main (void)
{
    sys_init();
    while (TRUE)
    {
        if (event_1)
            service_event_1();
        if (event_2)
            service_event_2();
    }
}
```

Ở đây, chương trình bắt đầu chạy cài đặt cho hệ thống, sau đó vào một vòng lặp vô hạn. Đối với mỗi sự kiện (event) có một hàm (một tác vụ) tương ứng. Mỗi tác vụ được quyền điều khiển lần lượt. Khi tác vụ một hoàn thành, nó chuyển quyền điều khiển cho tác vụ hai, lần lượt như thế đến hết, sau đó lại lặp lại. Phương pháp này còn gọi là lập lịch nối tiếp (sequential scheduling hoặc một tên khác là round-robin scheduling).

Mặc dù phương pháp này hoạt động tốt với các hệ thống đơn giản, tuy nhiên nó cũng có một số mặt hạn chế. Thời gian hệ thống phản ứng lại sự kiện phụ thuộc vào vị trí mà chương trình chạy vòng lặp. Ví dụ nếu sự kiện event\_1 xảy ra ngay trước câu lệnh `if(event_1)` thì thời gian phản ứng sẽ rất ngắn. Tuy nhiên, nếu nó xảy ra ngay sau khi chương trình kiểm tra câu lệnh đó, hệ thống sẽ phải đợi một chu kỳ vòng lặp để thực hiện phản ứng đối với event\_1 đó.

Đối với một số hệ thống đơn giản, phương pháp này chấp nhận được. Tuy nhiên đối với các hệ thống phức tạp, khi vòng lặp quá lớn, nếu hệ thống phản ứng không kịp với sự kiện xảy ra, hệ thống có thể bị sự cố. Ví dụ trong một hệ thống điều khiển dây chuyền lắp ráp, nếu hệ thống không phản ứng kịp lại một sự kiện nào quá lâu, cả dây chuyền có thể bị trục trặc.

Một vấn đề nữa là tất cả các tác vụ đều có ưu tiên ngang nhau. Trên thực tế trong một hệ thống bao giờ cũng có những tác vụ có quyền ưu tiên cao hơn các tác vụ khác.

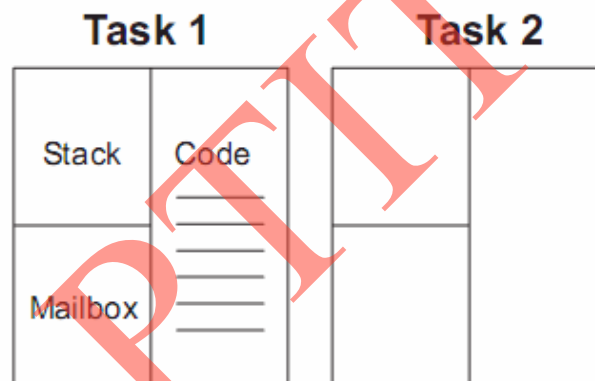
Vấn đề thứ ba là số lượng tác vụ. Nếu số lượng tác vụ trong hệ thống quá lớn, hệ thống sẽ không thể xử lý kịp các yêu cầu, thậm chí mỗi tác vụ chỉ chiếm một lượng thời gian hạn chế.

Phương pháp thứ hai là ngắt. Phương pháp này hiệu quả hơn và được sử dụng rộng rãi trong các hệ điều hành thời gian thực hiện nay. Khi có một sự kiện xảy ra, sự kiện này sẽ ngắt chương trình đang chạy hiện tại và gọi hàm xử lý ngắt. Hàm này sẽ phản ứng lại sự kiện vừa xảy ra. Sau khi hoàn thành, chương trình lại chạy về đoạn chạy dở. Các sự kiện xảy ra sẽ được đáp ứng ngay và không phải chờ đợi cả vòng lặp polling.

## Tác vụ

Để hệ thống có thể phản ứng kịp với các sự kiện bên ngoài, các hệ điều hành nhúng đều phải có khả năng đa nhiệm (multi-tasking). Các vấn đề lớn sẽ được chia thành các vấn đề nhỏ hơn, đơn giản hơn. Mỗi một vấn đề con này trở thành một tác vụ. Mỗi tác vụ chỉ làm một thứ và phải đơn giản. Các tác vụ này có thể được cho rằng chạy song song với nhau. Nhưng trên thực tế, trên hệ thống có một bộ VXL, các tác vụ chia nhau tài nguyên của bộ VXL. Mỗi tác vụ sẽ chiếm dùng bộ VXL một khoảng thời gian. Các khoảng thời gian này rất nhỏ, cho nên ta có thể coi như các tác vụ chạy song song.

Giống như một chương trình bất kỳ, một tác vụ chứa mã lệnh để thực hiện tính năng mà tác vụ được thiết kế cho. Mã lệnh này được chứa trong các hàm tương tự hàm main() trong một chương trình C bình thường. Tuy nhiên, sự khác biệt là mỗi tác vụ có ngữ cảnh (context) trong ngăn xếp (stack) của nó.



Hình 3-2. Cấu trúc của tác vụ

Mỗi tác vụ bao gồm:

- Đoạn mã thực hiện chức năng
- Ngăn xếp đựng ngữ cảnh
- Hộp thư (mailbox) để liên hệ với các tác vụ khác

Các tác vụ khác nhau có thể độc lập với nhau, tuy nhiên chúng thường được tạo ra để thực hiện một nhiệm vụ nào đó theo thiết kế. Do đó, trong tác vụ cần có một cơ chế thông tin để nó có thể liên lạc và đồng bộ với các tác vụ khác. Cơ chế thường được dùng là một hòm thư (mailbox).

Ví dụ một đoạn code tác vụ.

```
void tác_vụ (void *data)
{
    init_task();
```



```

while (TRUE)
{
    Wait for message at task mailbox();
    switch (message.type)
    {
        case MESSAGE_TYPE_X:
            ...
            break;
        case MESSAGE_TYPE_Y:
            ...
            break;
    }
}

```

Trong đoạn code trên, tác vụ được bắt đầu bằng một hàm khởi tạo. Sau đó, nó sẽ đi vào một vòng lặp vô hạn (VD `while (TRUE)`). Trong vòng lặp đó, tác vụ hầu như không sử dụng tài nguyên, mà chỉ đợi một sự kiện nào đó xảy ra.

Khi có sự kiện xảy ra, tác vụ sẽ thức dậy, và khi nhận được một bản tin, tác vụ sẽ giải mã bản tin đó và xử lý bằng lệnh `switch` như ví dụ trên. Sau khi xử lý bản tin xong, tác vụ sẽ quay lại để đợi sự kiện tiếp theo.

Hệ điều hành thời gian thực theo dõi các tác vụ thông qua khối điều khiển tác vụ (tác vụ control block hoặc TCB). Đây là nơi hệ điều hành thời gian thực lưu các thông tin về các tác vụ. Một khối TCB thường lưu các thông tin sau:

- Tác vụ ID: Thường là số của tác vụ.
- Tình trạng của tác vụ: thường là sẵn sàng, bị chặn, ..
- Thứ tự ưu tiên của tác vụ: thường là số hoặc mức độ ưu tiên.
- Địa chỉ của tác vụ: Nơi đoạn mã của tác vụ được lưu trong bộ nhớ
- Con trỏ ngăn xếp của tác vụ

### **Đa nhiệm (Multi-tasking)**

Từ trên có thể thấy các tác vụ giành phần lớn thời gian chờ đợi sự kiện. Thời gian xử lý thường rất ít so với thời gian chờ đợi. Đây là lý do để có thể dùng đa nhiệm (multi-tasking).

Đa nhiệm là cách mà nhiều tác vụ có thể chia sẻ tài nguyên xử lý chung (VD như một CPU). Trong trường hợp hệ thống chỉ có một CPU, tại một thời điểm chỉ có một tác vụ chạy. Điều đó có nghĩa là CPU chỉ phục vụ tác vụ đó. Để có thể chạy đa nhiệm, hệ thống cần có cơ chế lập lịch để cho các tác vụ có thể chia sẻ tài nguyên của CPU. Trong khi một tác vụ đang chạy, tác vụ khác sẽ xếp hàng chờ đến lượt. Thông tin chi tiết sẽ được trình bày ở phần sau.

### **3.3.3 Lập lịch thời gian thực (Real-time Scheduling)**

Trong một hệ điều hành thời gian thực, việc lập lịch được thực hiện bởi một bộ phận của nhân (kernel) gọi là bộ lập lịch (scheduler). Bộ lập lịch sẽ quản lý việc phân chia tài nguyên của hệ thống cho các tác vụ khác nhau.

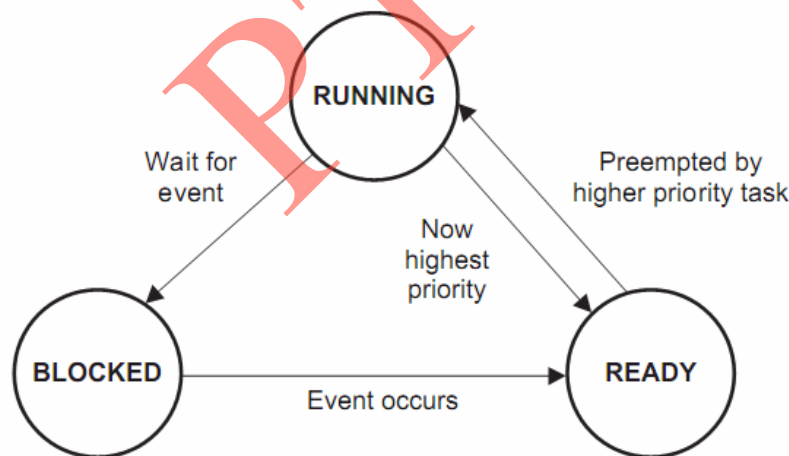
Do yêu cầu nghiêm ngặt về thời gian trong các hệ điều hành thời gian thực, bộ lập lịch phải đảm bảo các tác vụ phải đáp ứng về thời hạn. Do đó, bộ lập lịch phải phân chia mức độ ưu tiên của các tác vụ và đảm bảo rằng chỉ có duy nhất tác vụ với mức ưu tiên cao nhất đang được chiếm dụng tài nguyên CPU.

Trên thực tế, bộ lập lịch xem xét các tác vụ như một hệ thống máy trạng thái (state machine). Các trạng thái như sau:

- **Đang hoạt động (running):** Đây là trạng thái hoạt động của tác vụ. Khi được nhân hệ điều hành trao quyền, tác vụ sẽ chuyển từ trạng thái sẵn sàng sang trạng thái hoạt động. Ở trạng thái này, tác vụ sẽ thực hiện các công việc của mình. Tác vụ có thể chuyển từ trạng thái hoạt động sang khóa trong trường hợp chờ đợi một sự kiện nào đó, hoặc sang trạng thái sẵn sàng nếu có một tác vụ khác với mức độ ưu tiên đã trở thành sẵn sàng. Quá trình này gọi là chiếm quyền thực thi (preemption).
- **Sẵn sàng (ready):** Đây là trạng thái chuẩn bị hoạt động của tác vụ. Khi ở trạng thái sẵn sàng, tác vụ sẽ chuyển sang trạng thái hoạt động khi được cung cấp quyền ưu tiên cao nhất.

Một tác vụ có thể chuyển sang trạng thái sẵn sàng từ một trong hai trường hợp :

- Khi một tác vụ đang bị khóa thì có sự kiện xuất hiện, ví dụ như có một ngắt hoặc một bản tin gửi đến hộp thư.
  - Khi tác vụ đang ở trạng thái hoạt động thì có một tác vụ khác chiếm quyền thực thi.
- **Khóa (blocked):** Đây là trạng thái không hoạt động của tác vụ. Tác vụ chuyển sang trạng thái này trong trường hợp đang chờ đợi một sự kiện xảy ra



Hình 3-3. Các trạng thái của tác vụ

### Lập lịch theo chu kỳ:

Nhiều tác vụ thường cần thức dậy theo chu kỳ và thực thi công việc, sau đó, tác vụ quay lại trạng thái ngủ. Đối với các tác vụ này, việc lập lịch được tính theo chu kỳ. VD: Cứ đến một thời điểm nhất định, tác vụ phải kiểm tra một số tính năng của hệ thống.

Để thực thi, trong các hệ điều hành nhúng thường có hàm trễ `Delay()`, các tác vụ được quản lý để thức dậy và hoạt động trong khoảng thời gian trễ này.

Một phương pháp khác là khai báo các tác vụ có tính chất chu kỳ. Trong trường hợp này, bộ lập lịch sẽ đánh thức các tác vụ thức dậy trong mỗi khoảng thời gian mà không phụ thuộc vào thời gian chạy của tác vụ (VD cứ mỗi 4s lại gọi một tác vụ).

Một tác vụ có tính chất chu kỳ gọi một hàm tương tự như `WaitUntilNext()`, hàm này sẽ chặn tác vụ đến thời điểm chạy tiếp theo.

### Lập lịch không theo chu kỳ:

Khác với các tác vụ có tính chu kỳ, một số tác vụ chỉ chạy ở một thời điểm ngẫu nhiên để phản ứng lại một sự kiện xảy ra. Sự kiện đó có thể là một công tắc đã được bật, một bộ cảm ứng đã đọc dữ liệu. Thường thì các sự kiện này được kết nối với máy tính qua các bộ ngắt (interrupt). Các chương trình con dịch vụ ngắt (ISR) sẽ thông tin về việc xảy ra ngắt với các tác vụ chịu trách nhiệm cho việc xử lý sự kiện.

### Lập lịch chiếm quyền thực thi:

Trong một hệ thống thông thường, khi có một ngắt xảy ra, hệ thống gọi chương trình con ISR để phục vụ cho ngắt. Nếu có một tác vụ với mức ưu tiên thấp đang chạy và một tác vụ có mức ưu tiên đang ở trạng thái khóa (block) để chờ sự kiện xảy ra, chương trình con dịch vụ ngắt ISR sẽ chuyển tác vụ với mức ưu tiên cao hơn chuyển sang trạng thái sẵn sàng (ready). Sau đó, bộ lập lịch xác định tác vụ có mức ưu tiên cao hơn đang ở trạng thái sẵn sàng và chuyển nó lên trạng thái hoạt động (running). Như vậy, tác vụ với mức ưu tiên thấp hơn đã bị chiếm quyền thực thi.

Các hệ thống cho phép chiếm quyền thực thi cung cấp cho ta hệ thống với thời gian phản ứng có thể dễ dàng đoán được hơn bởi các sự kiện với mức độ ưu tiên cao được xử lý sớm hơn. Đây là một điều cần bản của hệ thống thời gian thực, nó sẽ có thể đảm bảo thời gian tối đa để xử lý một sự kiện. Trong trường hợp hệ thống không chiếm quyền thực thi, không có sự đảm bảo về thời gian trước khi tác vụ đang chạy sẽ thoát.

### 3.3.4 Đồng bộ

Trong hệ thống, các tác vụ có thể độc lập với nhau. Tuy nhiên, chức năng của cả hệ thống cho một công việc yêu cầu các tác vụ phải có sự liên hệ, phối hợp. Việc đồng bộ giữa các tác vụ trong hệ điều hành thời gian thực được thực hiện bằng tập hợp các dịch vụ truyền thông giữa các tác vụ.

Một số cơ chế thông tin và đồng bộ hóa thường dùng như sau

- Semaphore: được sử dụng cho việc đồng bộ và khóa tài nguyên

- Cờ sự kiện (Event Flag): Dùng để thể hiện một hay nhiều sự kiện xảy ra. Cờ sự kiện cho phép đồng bộ khóa trong trường hợp có nhiều sự kiện kết hợp.
- Hòm thư (mailbox), hàng đợi (queue) hay đường ống (pipe): Các cơ chế này dùng để chuyển dữ liệu giữa các tác vụ với nhau.

Ngoài ra, trên thực tế có thể sử dụng một số cơ chế khác.

### Semaphore:

Về mặt chức năng, semaphore hoạt động giống như một chiếc chìa khóa cho việc truy cập tài nguyên. Tác vụ vào giữ chiếc chìa khóa này mới có quyền sử dụng tài nguyên hệ thống.

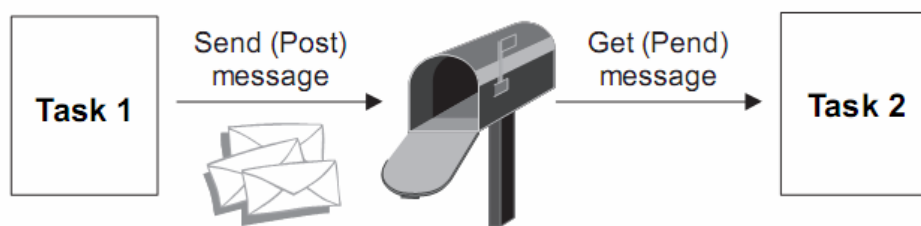
VD: Trong trường hợp có 2 tác vụ cùng truy cập một chiếc máy in. Nếu không có cơ chế đồng bộ giữa 2 tác vụ, máy in sẽ in bất cứ dữ liệu nào mà nó nhận được. Khi đó các bản tin sẽ bị trộn lẫn.

Trên thực tế, để có thể sử dụng tài nguyên hệ thống, tác vụ cần yêu cầu chìa khóa (semaphore) bằng cách gọi tới một dịch vụ thích hợp. Nếu chìa khóa ở trạng thái sẵn sàng (có nghĩa là tài nguyên hiện tại đang không được sử dụng bởi tác vụ nào), tác vụ có thể được phép sử dụng tài nguyên, đồng thời chìa khóa sẽ được giữ lại. Sau khi sử dụng xong, tác vụ đó phải trả lại chìa khóa (semaphore) để các tác vụ khác có thể sử dụng tài nguyên hệ thống.

Ngược lại, nếu tài nguyên đang được sử dụng bởi tác vụ khác, tác vụ đó sẽ bị khóa cho đến khi semaphore được trả lại. Trong trường hợp cùng một lúc có nhiều tác vụ yêu cầu semaphore khi tài nguyên hệ thống đang được sử dụng thì tất cả tác vụ đó sẽ bị khóa lại. Các tác vụ bị khóa sẽ được xếp hàng theo thứ tự về mặt ưu tiên hay về mặt thời gian theo yêu cầu semaphore.

### Hộp thư (mailbox):

Không những chỉ tạo các tín hiệu cho các sự kiện, các tác vụ thường xuyên cần chia sẻ dữ liệu. Việc chia sẻ này được thực hiện bằng cơ chế hộp thư (mailbox). Một tác vụ (là người gửi) sẽ gửi một bản tin (message) tới hộp thư trong khi tác vụ khác (người nhận) chờ bản tin tại hòm thư. Nếu không có bản tin trong hộp thư khi tác vụ nhận chờ ở hộp thư, tác vụ đó sẽ bị khóa đến khi bản tin được đưa tới.



Hình 3-4. Sơ đồ hộp thư

### Hàng đợi (queue) hay đường ống (pipe):

Hàng đợi thường là hộp thư có khả năng chứa nhiều thư cùng một lúc. Khi tạo ra một hàng đợi, cần xác định kích thước của hàng, số bản tin hàng đợi có thể lưu. Trong hàng đợi ngoài các lệnh gửi và chờ như hộp thư, một hàng đợi cần có các dịch vụ để chuyển bản tin có ưu tiên cao hơn lên đầu hàng hoặc xóa các bản tin trong cả hàng.

Trong cơ chế đường ống, việc sắp xếp dữ liệu lại khác một chút. Đối với hộp thư và hàng đợi, việc dịch chuyển dữ liệu được thực hiện theo khối (bản tin). Còn trong cơ chế đường ống, một luồng dữ liệu (byte stream) liên tục được nối giữa hai tác vụ. Một tác vụ sẽ đọc đường ống, còn một tác vụ khác ghi lên đường ống.

### 3.2.5 HAL (Hardware Abstraction Layer)

Để việc triển khai hệ điều hành thời gian thực trên các phần cứng khác nhau được dễ dàng hơn, cần có việc đặt ra các giao diện chuẩn cho việc lập trình. Khi lập trình viên chuyển sang một nền tảng (platform) mới, các giao diện lập trình này được giữ nguyên mặc dù phần cứng phía dưới thay đổi.

VD: Hai nền tảng khác nhau có thể có các bộ đếm khác nhau. Mỗi bộ đếm cần một phiên bản mã mới để khởi tạo và thiết lập cấu hình thiết bị. Nếu sử dụng HAL, giao diện lập trình sẽ không thay đổi mặc dù cả phần cứng và phần mềm đều thay đổi.

HAL là một lớp phần mềm cung cấp một tập hợp các giao diện lập trình xác định, che đi cấu trúc phần cứng. HAL được thực thi bằng cách ảo hóa nền tảng phần cứng, làm cho các trình điều khiển có thể chuyển giữa các phần cứng khác nhau.

Phần mềm HAL giao tiếp với một thiết bị ngoại vi chuyên biệt được gọi là trình điều khiển thiết bị (device driver). Một trình điều khiển thiết bị cung cấp giao diện lập trình ứng dụng chuẩn (API) để đọc và ghi tới thiết bị ngoại vi đó.

Trên thực tế, khi phát triển các bo mạch nhúng, các hãng thường cung cấp gói phần mềm phát triển kèm theo (BSP). Các gói phần mềm cho việc phát triển này tương đương với HAL.

HAL thường hỗ trợ các thành phần phần cứng sau:

- CPU, cache, và MMU.
- Thiết lập bản đồ bộ nhớ.
- Hỗ trợ xử lý ngắt và các trường hợp đặc biệt.
- Truy cập bộ nhớ trực tiếp (DMA).
- Các bộ đếm (Timers).
- Console của hệ thống.
- Quản lý giao diện bus.
- Quản lý nguồn.

## 3.3 Giới thiệu các hệ điều hành thời gian thực

### 3.3.1 FreeRTOS:

FreeRTOS là một hệ điều hành thời gian thực miễn phí. Hệ điều hành này được Richard Barry công bố rộng rãi từ năm 2003, đến nay, hệ điều hành này đang được phát triển mạnh mẽ và được cộng đồng mạng mã nguồn mở ủng hộ. FreeRTOS có tính khả chuyển, mã nguồn mở, lõi có thể được tải miễn phí và nó có thể dùng cho các ứng dụng thương mại. Nó phù hợp với những hệ nhúng thời gian thực nhỏ. Hầu hết các code được viết bằng ngôn ngữ C nên nó có tính phù hợp cao với nhiều nền khác nhau.

Ưu điểm của nó là dung lượng nhỏ và có thể chạy trên những nền phần cứng mà nhiều hệ điều hành khác không chạy được. Có thể port cho nhiều kiến trúc vi điều khiển và những công cụ phát triển khác nhau.

FreeRTOS được cấp giấy phép bởi bản đã được chỉnh sửa bởi GPL (*General Public License*) và có thể sử dụng trong các ứng dụng thương mại với giấy phép này. Ngoài ra liên quan đến FreeRTOS có OpenRTOS và SafeRTOS. OpenRTOS là bản thương mại của FreeRTOS và không liên quan gì đến GPL. SafeRTOS là về cơ bản dựa trên FreeRTOS nhưng được phân tích, chứng minh bằng tài liệu và kiểm tra nghiêm ngặt với chuẩn IEC61508. Và chuẩn IEC61508 SIL3 đã được tạo ra và phát triển độc lập để hoàn thiện tài liệu cho SafeRTOS.

So sánh giữa FreeRTOS và OpenRTOS:

	FreeRTOS	OpenRTOS
Miễn phí	Có	Không
Có thể sử dụng trong ứng dụng thương mại?	Có	Có
Miễn phí trong ứng dụng thương mại?	Có	Có
Phải đưa ra mã nguồn của code ứng dụng?	Không	Không
Phải đưa ra thay đổi mã nguồn của lõi?	Có	Không
Phải đưa vào báo cáo nếu sử dụng FreeRTOS?	Có, đường dẫn	Không
Phải cung cấp mã FreeRTOS cho người sử dụng?	Có	Không
Có thể nhận hỗ trợ thương mại?	Không	Có

Bảng 3.1: So sánh giữa FreeRTOS và OpenRTOS

*Các đặc điểm chính của FreeRTOS:*

- Lõi FreeRTOS hỗ trợ cả preemptive, cooperative hoặc kết hợp giữa 2 kiểu lập lịch này.
- SafeRTOS là sản phẩm dẫn xuất, cung cấp mã nguồn riêng ở mức độ cao.
- Được thiết kế nhỏ, đơn giản và dễ sử dụng.
- Cấu trúc mã nguồn rất linh động được viết bằng ngôn ngữ C.
- Hỗ trợ cả task và co-routine.
- Có lựa chọn nhận biết tràn ngăn xếp.
- Không giới hạn số task có thể tạo ra, phụ thuộc vào tài nguyên của hệ thống.
- Không giới hạn số mức ưu tiên được sử dụng.
- Không giới hạn số task cùng một mức ưu tiên.
- Hỗ trợ truyền thông và đồng bộ giữa các tác vụ hoặc giữa tác vụ và ngắt thông qua nhiều chiến lược khác nhau.



- Các công cụ phát triển miễn phí, port cho Cortex-M3, ARM7, PIC, MSP430, H8/S, AMD, AVR, x86 và 8051...
- Miễn phí mã nguồn phần mềm nhúng.
- Miễn phí trong ứng dụng thương mại.
- Tiền cấu hình cho các ứng dụng thử nghiệm, từ đó dễ dàng tìm hiểu và phát triển.

*Các dòng vi điều khiển đã được thử nghiệm với FreeRTOS:*

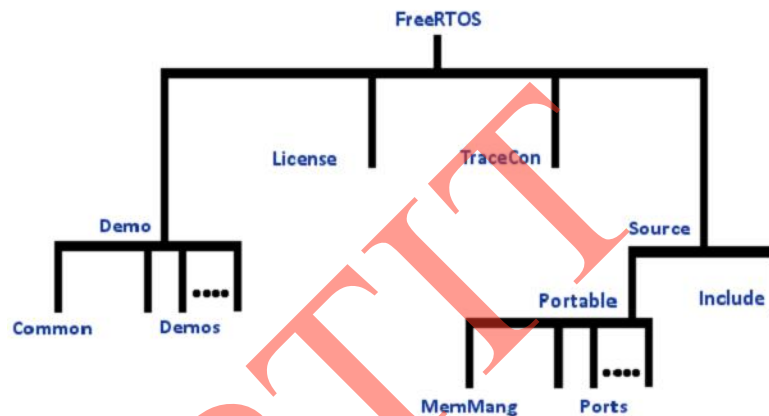
- Vi điều khiển ST STM32 Cortex-M3.
- ARM Cortex-M3 dựa trên vi điều khiển sử dụng ARM Keil (RVDS), IAR, Rowley và công cụ GCC.
- Atmel AVR32 AT32UC3A: vi điều khiển flash sử dụng GCC và IAR.
- Các vi điện tử ST: STR71x (ARM7), STR75x( ARM7), STR9 (ARM9) (STR711F, STR712F, ... ).
- LPC2106, LPC2124 và LPC2129 (ARM7). Gồm mã nguồn cho I2C driver.
- H8S2329 (Hitachi H8/S) với EDK2329 demo.
- Atmel AT91SAM7 family (AT91SAM7X256, AT91SAM7X128, AT91SAM7S32, AT91SAM7S64, AT91SAM7S128, AT91SAM7S256). Bao gồm mã nguồn USB driver cho IAR Kickstart, uIP và lwIP nhúng vào Ethernet TCP/IP.
- AT91FR40008 với Embest ATEB40X demo.
- MSP430 với demo cho LCD driver. MSPGCC and Rowley CrossWorks được hỗ trợ.
- HCS12 (MC9S12C32 loại bộ nhớ nhỏ và MC9S12DP256B kiểu bank nhớ)
- Fujitsu MB91460 series (32bit) and MB96340 series (16FX 16bit) sử dụng trình dịch Softune và Euroscope debugger.
- Cygnal 8051 / 8052.
- Microchip PICMicro PIC18 (8 bit), PIC24 (16bit MCU) và dsPIC (16bit DSC) và PIC32 (32bit).
- Atmel AVR (MegaAVR) với STK500 demo.
- Vi điều khiển RDC8822 với demo cho Flashlite 186 SBC.



- PC (chạy ở FreeDOS hoặc DOS khác).
- ColdFire.
- Zilog Z80.
- Xilinx Microblaze chạy trên Virtex4 FPGA.
- Xilinx PowerPC (PPC405) chạy trên Virtex4 FPGA.

Ngoài ra các trình dịch đã hỗ trợ cho các bản thử nghiệm FreeRTOS trên các vi điều khiển : Rowley CrossWorks, Keil, CodeWarrior, IAR, GNU GCC (nhiều loại), MPLAB, SDCC, Open Watcom, Paradigm và Borland.

### **Cấu trúc thư mục các file của FreeRTOS:**



Hình 3-5. Sơ đồ cấu trúc thư mục của FreeRTOS

Các bản download FreeRTOS trên [www.freertos.org](http://www.freertos.org) bao gồm mã nguồn cho các bản port trên các bộ vi xử lý và các ứng dụng thử nghiệm. Cấu trúc thư mục khá đơn giản, và nhân FreeRTOS chỉ bao gồm đúng 3 files(hay 4 file với tính năng co-routines – đây là một dạng rút gọn của task thường được dùng cho các hệ thống với bộ nhớ rất giới hạn).

Từ thư mục gốc, bản download FreeRTOS gồm 2 thư mục con :

```

FreeRTOS
|
|--Demo      Chứa các ứng dụng thử nghiệm.
|
|--Source    Chứa mã nguồn của nhân.

```

Phần lớn nhân freeRTOS (cụ thể hơn là mã nguồn của bộ lập lịch phân bổ - scheduler ) chứa trong 3 file chung với mọi kiến trúc vi xử lý là **tasks.c**, **queue.c** và **list.c** (hoặc thêm file croutine.c thực thi chức năng co-routines)trong thư mục **Source**.

Mỗi kiến trúc vi xử lý sẽ yêu cầu 1 phần mã nhân nhò cho riêng kiến trúc đó. Mã riêng cho từng kiến trúc được nằm trong thư mục **/Source/Portable**.

```
FreeRTOS
|
|--Demo      Chứa các ứng dụng thử nghiệm.
|
|--Common   Các file ứng dụng chung cho tất cả các port.
|
|--ARM7_AtmeISAM7S64_IAR  Ứng dụng mẫu cho AtmeISAM7S64 tool IAR.
|
|--ARM7_AT91SAM7X256_Eclipse  Ứng dụng mẫu cho AT91SAM7X256, tool
Eclipse
|
|--.....
|
|--Source   Chứa các file mã nguồn của bộ lập lịch (sheduler)
|
|   3 file mã nguồn của bộ lập lịch chung cho tất cả các port
|   (4 với chức năng co-routines)
|--include      Các file header.
|
|--portable     Lớp port cho các bộ vi xử lý khác nhau.
|
|   |--MemMang   Bộ cấp phát bộ nhớ mẫu.
|   |--GCC       Lớp port cho các vi xử lý sử dụng tool GCC compiler
|   |--RVDS      Lớp port cho các vi xử lý sử dụng tool RVDS
|   |--.....
```

5 file header trong thư mục **/Source/Include** chính chứa các khai báo các tham số, định nghĩa các kiểu dữ liệu và các khai báo prototype của các API của FreeRTOS đó là :

- FreeRTOS.h : chứa khai báo các file header và các định nghĩa được yêu cầu cho vi xử lý sẽ được sử dụng và kiểm tra các marco cần thiết phải viết cho riêng từng vi xử lý đã được định nghĩa chưa. Các marco này phải được định nghĩa trong FreeRTOSConfig.h trong thư mục demo của từng port.
- Task.h : chứa các khai báo về các hàm và các marco liên quan đến các thủ tục tạo, xóa, thiết lập và chỉnh sửa các tham số của các task(stack, trạng thái(running, ready, block), mức ưu tiên,...), các tác vụ liên quan đến bộ scheduler.
- List.h : chứa các khai báo về các hàm và các marco xây dựng cấu trúc danh sách để scheduler dùng để quản lý các task.
- Croutine.h: chứa các khai báo về các hàm và các marco xây dựng các co-routine.
- Portables.h : chứa khai báo các header phù hợp với từng port và cho phép tạo tính linh động đối với từng cấu hình port.

Ngoài ra còn một số các header chứa các khai báo về các API hỗ trợ các thuật toán cho phép truyền thông giữa các task như queues.h, semphr.h.

a) FreeRTOS.h:

Gồm 2 phần:

- Chứa các khai báo về các file header viết riêng cho từng port:

```
/* Chứa các định nghĩa cơ bản */
#include "projdefs.h"
/* Chứa các định nghĩa về các tham số cấu hình */
/* cho bộ scheduler */
#include "FreeRTOSConfig.h"
/* chứa các định nghĩa cho từng port */
#include "portable.h"
```

- Kiểm tra các khai báo cần thiết về các tham số cấu hình và các marco, thiết lập các giá trị mặc định với các tham số chưa được thiết lập giá trị cụ thể.

```
/*kiểm tra cấu hình cho chính sách sử dụng trong bộ scheduler */
#ifndef configUSE_PREEMPTION
    #error Missing definition: configUSE_PREEMPTION should be
    defined in FreeRTOSConfig.h as either 1 or 0. See the
    Configuration section of the FreeRTOS API documentation for
    details.
#endif
.....
/*kiểm tra cấu hình sử dụng co-routine hay không*/
#ifndef configUSE_CO_ROUTINES
    #error Missing definition: configUSE_CO_ROUTINES should
    be defined in FreeRTOSConfig.h as either 1 or 0. See the
    Configuration section of the FreeRTOS API documentation for
    details.
#endif
.....
/*kiểm tra cấu hình tham số cho semaphores và thiết lập*/
/*giá trị mặc định khi cần thiết*/
#ifndef configUSE_COUNTING_SEMAPHORES
    #define configUSE_COUNTING_SEMAPHORES 0
#endif
```

b) Task.h:

Gồm các khai báo về các API liên quan đến task, chia theo 5 chức năng:

#### ***Các API liên quan đến tạo và xóa task:***

Gồm 2 task chính thực hiện 2 nhiệm vụ quan trọng là tạo mới và xóa task.

- API tạo task : *xTaskCreate()*.
- API xóa task: *vTaskDelete()*.

Khai báo đầy đủ của *xTaskCreate()* là:

```
portBASE_TYPE xTaskCreate(
                                pdTASK_CODE pvTaskCode,
                                const char * const pcName,
                                unsigned short usStackDepth,
                                void *pvParameters,
                                unsigned portBASE_TYPE uxPriority,
                                xTaskHandle *pvCreatedTask
                                );
```

API này sẽ tạo ra 1 một task mới và thêm nó vào danh sách các task sẵn sàng thực thi. Task được tạo ra có quyền truy cập toàn bộ đối với toàn bộ bộ nhớ của vi xử lý. Đối với các hệ thống mà có thêm hỗ trợ MPU, có thể tạo ra một task với các tham số ràng buộc bởi MPU thông qua API *xTaskCreateRestricted()*.

Thực chất thì khai báo của *xTaskCreate()* là một marco gọi đến thủ tục *xTaskGenericCreate()* trong tasks.c. Đây là một hàm cho phép tạo ra các task với nhiều tùy chọn hơn, phù hợp với các cấu hình khác nhau của hệ thống, cụ thể trong trường hợp này, nó tạo ra một task trong cấu hình không sử dụng khối MPU.

```
/*định nghĩa API xTaskCreate()*/
#define xTaskCreate( pvTaskCode, pcName, usStackDepth, pvParameters,
uxPriority, pxCreatedTask ) xTaskGenericCreate( ( pvTaskCode ),
(pcName ), ( usStackDepth ), ( pvParameters ), ( uxPriority ),
(pxCreatedTask ), ( NULL ), ( NULL ))
```

**Các API điều khiển task:** tạo ra các hàm điều khiển API cụ thể là các nhiệm vụ như sau:

- Gây trễ task với các điều kiện khác nhau: *vTaskDelay()* và *vTaskDelayUntil()*. *vTaskDelay()* dùng để tạo trễ trong một khoảng thời gian tương đối bắt đầu từ thời điểm gọi API này, còn *vTaskDelayUntil()* tạo trễ trong một khoảng thời gian xác định.
- Các tác vụ liên quan đến ức ưu tiên: *xTaskPriorityGet()* và *vTaskPrioritySet()*. Hai API này làm nhiệm vụ xác định mức ưu tiên đang có và đặt mức ưu tiên cho task.
- Các API liên quan đến trạng thái task như chặn, khôi phục task: *vTaskSuspend()* nhằm để chặn bất kỳ task nào. *vTaskResume()* được gọi sau khi task bị dừng muốn quay về trạng thái sẵn sàng. Muốn gọi hàm *vTaskResume()* từ ngắt thì sử dụng *xTaskResumeFromISR()*.

**Các API tiện ích cho task:** chứa các API cung cấp các tiện ích hệ thống cho task như:

- *xTaskGetTickCount*: trả lại giá trị các tick đếm được từ khi *vTaskStartScheduler* bị hủy bỏ.
- *uxTaskGetNumberOfTasks(void)*: trả lại số lượng các task mà kernel đang quản lý. Hàm này còn bao gồm cả các task đã sẵn sàng, bị khóa hoặc bị treo. Task đã bị delete mà chưa được giải phóng bởi idle cũng được tính vào.

- *vTaskList()*: hàm này sẽ không cho phép ngắt trong khoảng thời gian nó làm việc. Nó không tạo ra để chạy các ứng dụng bình thường nhưng giúp cho việc debug.
- *vTaskStartTrace()*: đánh dấu việc bắt đầu hoạt động của kernel. Việc đánh dấu chia ra để nhận ra task nào đang chạy vào lúc nào. Đánh dấu file được lưu trữ ở dạng nhị phân. Sử dụng những tiện ích độc lập của DOS thì gọi convtrce.exe để chuyển chúng sang kiểu text file dạng mà có thể được xem và được vẽ.
- *ulTaskEndTrace()*: Dừng đánh dấu kernel hoạt động, trả lại số byte mà đã viết vào bộ đệm đánh dấu.
- .....

**Các API điều khiển nhân:** gồm các API cung cấp các giao diện đối với bộ scheduler như:

- *vTaskStartScheduler()* : khởi động bộ scheduler, cụ thể là kích hoạt hoạt động của freeRTOS.
- *vTaskEndScheduler()* : chấm dứt hoạt động của scheduler.
- *taskDISABLE\_INTERRUPTS()/taskENABLE\_INTERRUPTS()* : cấm/cho phép ngắt.
- *taskENTER\_CRITICAL()/taskEXIT\_CRITICAL()* : báo hiệu các đoạn mã quan trọng cần có những xử lý phù hợp.
- .....

**Các API liên quan đến MPU:** gồm các API mà ứng dụng khối MPU trong các cấu hình port có hỗ trợ MPU:

- *xTaskCreateRestricted()* : tạo 1 task với những ràng buộc về quyền truy cập đến tài nguyên bộ nhớ của vi xử lý. Nó cũng như *xTaskCreate()*, là một Macro tham chiếu đến *xTaskGenericCreate()* tuy nhiên nó có thêm các tham số về MPU.  

```
#define xTaskCreateRestricted( x, pxCreatedTask )
xTaskGenericCreate( ((x)->pvTaskCode), ((x)->pcName), ((x)->usStackDepth), ((x)->pvParameters), ((x)->uxPriority),
(pxCreatedTask), ((x)->puxStackBuffer), ((x)->xRegions) )
```
- *vTaskAllocateMPURegions()* : thay đổi hoặc cấp phát lại các vùng nhớ ràng buộc của các task được tạo bởi *xTaskCreateRestricted()*.
- ....

c) List.h:

File này chứa các khai báo về các hàm và các marco liên quan đến các thủ tục tạo, xóa, thiết lập và chỉnh sửa các tham số của các task.

FreeRTOS xây dựng một cấu trúc danh sách các task, mỗi task là một phần tử có cấu trúc:

```
/*khai báo kiểu của các phần tử trong danh sách*/
struct xLIST_ITEM
{
```

```

/*kết hợp với mức ưu tiên dùng trong sắp xếp danh sách */
portTickType xItemValue;
/*Con trỏ đến phần tử trước nó trong danh sách */
volatile struct xLIST_ITEM * pxNext;
/* Con trỏ đến phần tử sau nó trong danh sách */
volatile struct xLIST_ITEM * pxPrevious;
/*con trỏ đến task, cụ thể là Task Control Block(TCB)*/
void * pvOwner;
/*con trỏ đến danh sách mà nó nằm trong đó*/
void * pvContainer;
};
/*Định nghĩa danh sách*/
typedef struct xLIST
{
    /*kích thước của danh sách*/
    volatile unsigned portBASE_TYPE uxNumberOfItems;
    /*pxIndex: dùng để chỉ đến phần tử hiện thời của danh sách*/
    volatile xListItem * pxIndex;
    /*đánh dấu kết thúc danh sách*/
    volatile xMiniListItem xListEnd;
} xList;

```

Đi kèm cấu trúc danh sách được sử dụng trong bộ scheduler là các API thao tác trên các phần tử của danh sách và danh sách:

- Khởi tạo danh sách : *vListInitialise()*.
- Đặt đối tượng sở hữu các phần tử của danh sách.
- Đặt giá trị của phần tử danh sách. Trong hầu hết trường hợp giá trị đó được dùng để sắp xếp danh sách theo một thứ tự nhất định nào đó.
- Để lấy giá trị của phần tử danh sách. Giá trị này có thể biểu thị bất cứ cái gì, ví dụ như mức ưu tiên của tác vụ hoặc thời gian mà task có thể bị khoá.
- Xác định xem danh sách còn chứa phần tử nào không, chỉ có giá trị true nếu danh sách rỗng.
- Kiểm tra số phần tử trong danh sách.
- Xác định phần tử tiếp theo của danh sách.
- Tìm chương trình chủ của phần tử đầu tiên trong danh sách.
- Kiểm tra xem phần tử có nằm trong danh sách không.
- Thêm phần tử vào danh sách.
- Loại bỏ phần tử từ danh sách.

Ví dụ:

```
.....
/*API xác định độ dài hiện tại của danh sách*/
#define listCURRENT_LIST_LENGTH( pxList )      ( ( pxList )->uxNumberOfItems )
.....
/*API khởi tạo danh sách*/
void vListInitialise( xList *pxList );
```

d) Croutine.h:

Chứa các khai báo về các hàm và các marco xây dựng các co-routine. Trong Croutine, cần chú ý một số các API quan trọng tương ứng với các thao tác trên task khai báo trong task.h:

- xCoroutineCreate() : tạo mới một co-routine và đưa nó vào danh sách các co-routine sẵn sàng chạy.
- crDELAY() : trễ một co-routine() trong một chu kì thời gian xác định.
- crSTART()/crEND() : khởi chạy/kết thúc một co-routine.
- ...

Ví dụ về cấu trúc chung của một co-routine:

```
void vACoRoutine( xCoRoutineHandle xHandle, unsigned
portBASE_TYPE uxIndex )
{
    // Khai báo các biến.
    static long ulAVariable;
    // bắt đầu co-routine;
    crSTART( xHandle );

    for( ;; )
    {
        // các câu lệnh của co-routine.
    }
    // kết thúc co-routine
    crEND();
}
```

e) Portables.h:

Khai báo file header tương ứng với port sẽ được sử dụng. File header được đưa vào sẽ tương ứng với marco về bản port được sử dụng:

```
.....
/*kiểm tra có phải là port cho avr, tool gcc*/
#ifndef GCC_MEGA_AVR
    #include "../portable/GCC/ATMega323/portmacro.h"
#endif
/*kiểm tra có phải là port cho avr, tool IAR*/
```



```

#ifdef IAR_MEGA_AVR
    #include "../portable/IAR/ATMega323/portmacro.h"
#endif

/*kiểm tra có phải là port cho PIC24, tool MPLAB*/
#ifdef MPLAB_PIC24_PORT
    #include "../..\\Source\\portable\\MPLAB\\PIC24_dsPIC\\portmacro.h"
#endif

....

```

Thêm nữa, file này cũng bao gồm các khai báo về các thủ tục quản lý bộ nhớ đối với từng port:

```

/*
 * Map to the memory management routines required for the port.
 */
void *pvPortMalloc( size_t xSize ) PRIVILEGED_FUNCTION;
void vPortFree( void *pv ) PRIVILEGED_FUNCTION;
void vPortInitialiseBlocks( void ) PRIVILEGED_FUNCTION;
size_t xPortGetFreeHeapSize( void ) PRIVILEGED_FUNCTION;
....

```

Thêm vào các file của nhân cần quan tâm 2 file *FreeRTOSConfig.h* và *portmacro.h*. Trong đó *FreeRTOSConfig.h* chứa các tham số cấu hình phù hợp với từng port cụ thể còn *portmacro.h* chứa các API riêng đối với từng port như cấu hình timer, các thủ tục sao lưu và khôi phục context.

### 3.3.3 Hệ điều hành Embedded Linux:

Thực chất của hệ điều hành embedded Linux là một phiên bản của hệ điều hành Linux được dùng trong các hệ thống nhúng như mobile phone, PDA, set-top box, các thiết bị điện tử gia dụng, các thiết bị mạng, các thiết bị điều khiển,... Theo thống kê hiện có, Linux hiện được sử dụng bởi khoảng 18% các kỹ sư nhúng.

Không giống như phiên bản dành cho desktop và server của Linux, phiên bản embedded Linux được thiết kế cho các thiết bị với tài nguyên giới hạn. Bởi những hạn chế về giá thành và kích thước, các thiết bị nhúng thường có ít RAM và bộ nhớ thứ cấp hơn so với desktop, và thường sử dụng bộ nhớ flash thay vì các ổ đĩa cứng. Bởi vì các thiết bị nhúng thực hiện các chức năng chuyên dụng nên các nhà phát triển đã tối ưu các phiên bản nhúng của Linux cho các cấu hình phần cứng của các hệ thống nhúng. Những tối ưu này giúp giảm thiểu các ứng dụng phần mềm và các trình điều khiển thiết bị, và cho phép thay đổi nhân Linux để nó trở thành 1 hệ điều hành thời gian thực.

Thiết kế của Linux phân biệt giữa các mã phụ thuộc và bộ xử lý cần thay đổi cho mỗi kiến trúc và mã có thể được khả chuyển đơn giản chỉ bằng việc biên dịch lại nó. Vì vậy, Linux đã được khả chuyển cho nhiều các kiến trúc bộ vi xử lý khác nhau như:

- Motorola 68k và các biến thể của nó.
- Alpha.
- Power PC.

- ARM.
- SPARC.
- MIPS.
- AVR32.
- Blackfin.
- ...

Việc sử dụng Linux trong các thiết bị nhúng có ưu điểm là bản quyền miễn phí, nhân ổn định, và được hỗ trợ lớn từ cộng đồng mã nguồn mở, nhà phát triển được tự do thay đổi và phân phối mã nguồn. Tuy nhiên nhược điểm đó là nó yêu cầu bộ nhớ tương đối lớn cho nhân và hệ thống file gốc, sự phức tạp trong việc truy nhập bộ nhớ ở chế độ nhân và chế độ người sử dụng cũng như nền tảng trình điều khiển thiết bị phức tạp.

Các đặc điểm quan trọng của Linux gồm:

- Đa nhiệm : bộ lập lịch phân bổ trong Linux sử dụng thuật toán lập lịch preemptive trong đó một tiến trình với mức ưu tiên cao hơn sẽ được ưu tiên là tiến trình tiếp theo được CPU thực hiện khi xảy ra một sự kiện không đồng bộ.
- Đa người sử dụng : Linux phát triển từ Unix là một hệ thống chia sẻ theo thời gian cho phép nhiều người sử dụng chia sẻ chung một máy tính. Vì vậy có rất nhiều các chức năng hỗ trợ việc bảo vệ dữ liệu và tính riêng tư.
- Đa xử lý: Linux cung cấp hỗ trợ mở rộng cho đa xử lý đối xứng SMP(Symmetric Multi-Processing).
- Bộ nhớ được bảo vệ: Mỗi tiến trình trong Linux hoạt động trong không gian nhớ riêng của nó và không được cho phép truy cập trực tiếp đến không gian nhớ của một tiến trình khác. Điều này tránh những con trỏ lỗi trong một tiến trình gây phá hoại không gian nhớ của tiến trình khác. Các truy cập sai được bẫy bởi phần cứng bảo vệ bộ nhớ của bộ xử lý và tiến trình sẽ bị ngắt với cảnh báo tương ứng.
- Hệ thống file phân cấp.

Hiện nay, với sự yêu cầu giảm thiểu thời gian phát triển, embedded Linux đang là một xu hướng phát triển mạnh mẽ trong thế giới nhúng.

### 3.3.4 Hệ điều hành uCLinux:

uClinux hay micro-controller Linux là một phiên bản của Linux được thiết kế cho các bộ vi xử lý mà không có khối quản lý bộ nhớ MMU(Memory Management Unit). Việc không có MMU là một đặc điểm khá chung đối với các bộ vi xử lý giá thành thấp. uClinux là một giải pháp mã nguồn mở và miễn phí bản quyền, mục đích là nhằm tương thích với Linux.

Dự án uClinux được bắt đầu năm 1997 với mục đích cho ra một phiên bản của nhân Linux 2.0 dành cho các vi điều khiển giá thành thấp do Jeff Dionne, Kenneth Albanowski và một nhóm các nhà phát triển khác đề xuất trong quá trình họ thảo luận về việc nhúng Linux và các bộ điều khiển mạng không có MMU nhằm giải quyết bài toán truyền thông trong hệ thống truyền thông. Phiên bản đầu tiên của uClinux là được phân bổ với bộ xử lý Motorola 68000, dựa trên bộ vi xử lý MC68328 được triển khai trong một bộ điều khiển SCADA.

Ngày nay, uClinux đã được port cho rất nhiều dòng vi điều khiển như ColdFire, Axis ETRAX, ARM, Atari 68k,...

## **Câu hỏi ôn tập**

1. Hệ điều hành thời gian thực là gì?
2. Nhiệm vụ của hệ điều hành thời gian thực?
3. Các đặc điểm của hệ điều hành thời gian thực.
4. Vẽ sơ đồ vị trí hệ điều hành thời gian thực trong model hệ thống nhúng.
5. Liệt kê một số hệ điều hành thời gian thực.
6. Kernel là gì?
7. Liệt kê và mô tả các tính năng chính của kernel.
8. Các model của hệ điều hành bao gồm những model nào?
9. Sự khác biệt giữa process và tác vụ?
10. Sự khác biệt giữa chiếm quyền và không chiếm quyền
11. Các trạng thái của tác vụ bao gồm các trạng thái nào?
12. Việc đồng bộ được thực hiện như thế nào?
13. Liệt kê các hàm chính trong FreeRTOS và mô tả các chức năng của các hàm?

## CHƯƠNG 4: THIẾT KẾ VÀ CÀI ĐẶT CÁC HỆ THỐNG NHÚNG

### 4.1 Thiết kế hệ thống

#### 4.1.1 Xác định yêu cầu

Một đặc điểm của hệ thống nhúng là cả phần cứng và phần mềm phải được tính toán, cân nhắc trong thời gian thiết kế chúng. Bởi vậy, loại thiết kế này còn được gọi là đồng thiết kế phần cứng/phần mềm (hardware/software codesign). Mục đích cuối cùng là tìm được sự kết hợp thích hợp của phần cứng và phần mềm để sản phẩm tạo ra có đầy đủ các đặc điểm kỹ thuật đã đề ra. Bởi vậy, các hệ thống nhúng không thể được thiết kế bởi một quá trình tổng hợp chỉ ghi chép lại các đáp ứng kỹ thuật vào bản kê khai. Tốt hơn, các phần tử cấu thành sẵn có để dùng phải được liệt kê cho hệ thống. Cũng có các lý do khác cho điều bắt buộc này: giảm thiểu việc tăng độ phức tạp của hệ thống nhúng, các yêu cầu nghiêm ngặt về thời gian đưa sản phẩm tới khách hàng, khả năng dùng lại của sản phẩm. Điều này dẫn đến thuật ngữ platform-based design (thiết kế trên nền):

*Một platform (nền) là một họ kiến trúc thoả mãn một tập hợp các ràng buộc áp đặt để cho phép dùng lại các phần tử phần cứng và phần mềm. Tuy nhiên, một nền phần cứng là chưa đủ. Các thiết kế nhanh, tin cậy, có tính dẫn xuất cao thường yêu cầu sử dụng một nền giao diện lập trình ứng dụng (API) để từ đó phát triển, mở rộng để đạt tới phần mềm ứng dụng. Nhìn chung, một nền (platform) là một lớp khái niệm trừu tượng trong đó bao gồm nhiều sàng lọc khả dĩ để đưa tới một mức thấp hơn. Thiết kế trên cơ sở nền là một cách tiếp cận meet-in-the-middle. Theo dòng thiết kế từ đỉnh xuống, người thiết kế sẽ lập bản đồ các nền từ cao tới thấp, và công bố những ràng buộc thiết kế giữa chúng [Sangiovanni-Vincentelli, 2002].*

Việc lập bản đồ là một quá trình lặp đi lặp lại, các công cụ đánh giá hoạt động trong quá trình này được hướng dẫn ở phần tiếp theo. Hình 4-1 thể hiện cách tiếp cận này.

Hoạt động thiết kế phải tính đến sự tồn tại của những platform sẵn có và ghi chúng vào bảng kê khai. Trên thực tế có một lượng lớn các hoạt động thiết kế, nhưng chỉ một vài hoạt động trong số chúng được trình bày ở đây và việc tham chiếu đến các platform sẵn có không phải luôn luôn được thể hiện. Các hoạt động thiết kế bao gồm:

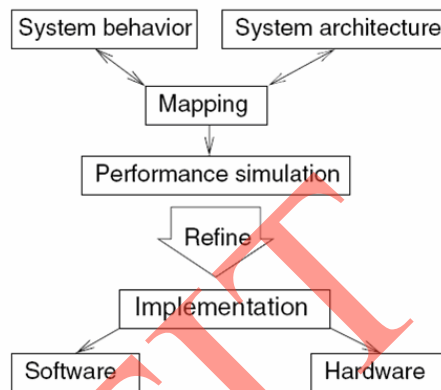
**Quản lý các nhiệm vụ cùng mức:** Hoạt động này có liên quan với việc xác định các nhiệm vụ phải được hiện diện trong hệ thống nhúng cuối cùng. Những nhiệm vụ này có thể khác với những nhiệm vụ đã bao gồm trong các đặc điểm kỹ thuật, từ đó có những lý do tốt để cho việc sát nhập và chia tách các nhiệm vụ.

**Các biến đổi cấp cao:** Người ta đã nhận thấy rằng có nhiều biến đổi cấp cao tối ưu có thể được ứng dụng trong kỹ thuật. Ví dụ, các vòng lặp có thể được thay đổi để truy xuất đến các phần tử mảng ở nhiều vùng khác nhau. Ngoài ra, các phép toán số học dấu phẩy động có thể thường xuyên được thay thế bởi các phép toán số học dấu phẩy cố định mà không có bất kỳ tổn thất đáng kể trong chất lượng. Những biến đổi cấp cao

thường vượt ra ngoài khả năng của trình biên dịch có sẵn và phải được áp dụng trước khi bắt đầu bất kỳ một quá trình biên dịch nào.

**Phân hoạch phần cứng/phần mềm:** Chúng ta thấy rằng trong trường hợp chung, một số chức năng phải được thực hiện bởi phần cứng đặc biệt để đáp ứng yêu cầu tăng tốc độ tính toán của hệ thống [De Man, 2002]. Phân hoạch phần cứng/phần mềm là hoạt động phân chia chức năng cần thực hiện cho phần cứng hoặc phần mềm.

**Biên dịch:** Những phần của đặc điểm kỹ thuật được ánh xạ tới phần mềm phải được biên dịch. Hiệu quả của mã tạo ra được cải thiện nếu trình biên dịch khai thác tốt kiến thức về bộ vi xử lý (và có thể bộ nhớ) nằm bên dưới phần cứng. Bởi vậy, có những chương trình biên dịch "nhận thức phần cứng" đặc biệt cho hệ thống nhúng.



Hình 4-1. Thiết kế trên nền (platform)

**Lập lịch trình:** Lập lịch trình (sắp đặt thời gian bắt đầu các hoạt động) phải được thực hiện trong một vài bối cảnh. Lịch trình phải được ánh chừng (gần chính xác) trong thời gian phân vùng phần cứng/phần mềm, trong thời gian quản lý các nhiệm vụ đồng mức và cũng có thể trong thời gian biên dịch. Lịch trình chính xác có thể nhận được cho mã chương trình cuối cùng.

**Khảo sát không gian thiết kế:** Trong hầu hết các trường hợp, sẽ có một vài thiết kế đáp ứng được các thông số kỹ thuật. Khảo sát không gian thiết kế là quá trình phân tích chi tiết tập các thiết kế khả dĩ (có thể thực hiện). Trong số các thiết kế đáp ứng được các thông số kỹ thuật, chỉ duy nhất một thiết kế được chọn.

Những luồng thiết kế riêng biệt có thể sử dụng các hoạt động này theo trình tự khác nhau. Không có một tập tiêu chuẩn của các hoạt động thiết kế.

#### 4.1.2 Đặc tả

Có thể vẫn còn những trường hợp mà đặc tả kỹ thuật của các hệ thống nhúng được thu thập trong một ngôn ngữ tự nhiên, như tiếng Anh. Tuy nhiên, cách tiếp cận này là hoàn toàn không thích hợp vì nó thiếu yêu cầu quan trọng cho các kỹ thuật đặc tả: cần thiết để kiểm tra đặc điểm kỹ thuật đầy đủ, không có mâu thuẫn và nó phải được triển khai có thể xuất phát từ đặc điểm kỹ thuật theo một cách có hệ thống. Vì vậy, đặc tả kỹ

thuật nên được thu thập trong các ngôn ngữ máy chính thức có thể đọc được. Ngôn ngữ đặc tả kỹ thuật cho các hệ thống nhúng phải có khả năng đại diện cho các tính năng sau đây:

**Sự phân cấp:** Con người thường không có khả năng thấu hiểu hệ thống có chứa nhiều đối tượng (các trạng thái, các thành phần) có quan hệ phức tạp với nhau. Việc mô tả tất cả các hệ thống thực tế cần nhiều hơn các đối tượng mà con người có thể hiểu được. Phân cấp là cơ chế duy nhất giúp giải quyết tình trạng khó xử này. Phân cấp có thể được đưa vào mà như vậy con người chỉ cần phải xử lý một số nhỏ các đối tượng tại bất kỳ thời điểm nào.

Có hai loại phân cấp:

- **Các phân cấp theo hành vi** : phân cấp theo hành vi là các phân cấp chứa các đối tượng cần thiết để mô tả hành vi hệ thống. Các trạng thái, các sự kiện và các tín hiệu đầu ra là những ví dụ cho các đối tượng như vậy.

- **Các phân cấp cấu trúc**: các phân cấp cấu trúc mô tả các hệ thống được bao gồm những thành phần vật lý như thế nào.

Ví dụ, các hệ thống nhúng có thể được bao gồm bộ vi xử lý, bộ nhớ, các cơ cấu chấp hành và các cảm biến. Các bộ xử lý, theo trình tự, lại bao gồm các thanh ghi, các bộ dồn kênh và các bộ cộng. Các bộ dồn kênh lại bao gồm trong nó là các cổng logic.

**Thời gian-hành vi**: một khi các yêu cầu tính toán thời gian rõ ràng (chính xác) là một trong những đặc trưng của hệ thống nhúng, các yêu cầu tính toán thời gian phải được thu thập trong đặc tả kỹ thuật.

**Hành vi định hướng trạng thái**: Nó đã được đề cập trong chương 1 mà các thiết bị tự động cung cấp một cơ chế tốt cho mô hình hóa các hệ thống phản ứng. Vì vậy, hành vi định hướng trạng thái cung cấp bởi các thiết bị tự động sẽ dễ mô tả. Tuy nhiên, các mô hình thiết bị tự động cổ điển là không đủ, vì chúng không thể mô hình hoá thời gian và vì sự phân cấp là không được hỗ trợ.

**Xử lý-sự kiện** : Do tính chất phản ứng tự nhiên của các hệ thống nhúng, các cơ chế cho việc mô tả các sự kiện phải tồn tại. Các sự kiện như vậy có thể là các sự kiện bên ngoài (gây ra bởi môi trường) hoặc các sự kiện nội bộ (gây ra bởi các thành phần của hệ thống).

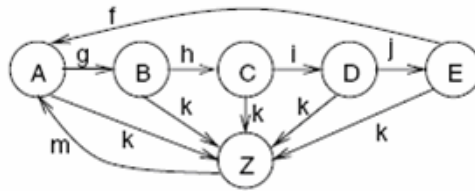
**Không có những trở ngại cho việc tạo ra những thực thi hiệu quả**: vì các hệ thống nhúng phải hiệu quả, không có những trở ngại ngăn cản việc tạo ra những thực thi hiệu quả cần phải thể hiện trong đặc tả.

**Hỗ trợ cho việc thiết kế các hệ thống đáng tin cậy**: Các kỹ thuật đặc tả cần phải cung cấp sự hỗ trợ cho việc thiết kế những hệ thống đáng tin cậy. Ví dụ, ngôn ngữ đặc tả kỹ thuật nên có ngữ nghĩa rõ ràng, tạo điều kiện thuận lợi cho sự xác minh hình thức và có khả năng mô tả bảo mật và những yêu cầu an toàn.

**Hành vi hướng ngoại lệ**: Trong nhiều trường hợp thực tế, những ngoại lệ hệ thống xuất hiện. Để thiết kế hệ thống đáng tin cậy, phải thật khả dĩ để mô tả những hoạt động để xử lý những ngoại lệ một cách dễ dàng. Không chấp nhận được rằng những ngoại lệ phải được chỉ thị cho mỗi và mọi trạng thái (giống như trong trường hợp những sơ đồ trạng thái cổ điển). Ví dụ: Trong Hình 4-2, đầu vào **k** có thể tương ứng tới một ngoại lệ.



Việc chỉ rõ ngoại lệ này tại mỗi trạng thái làm cho sơ đồ rất phức tạp. Tình hình sẽ tồi tệ hơn cho sơ đồ trạng thái lớn hơn với nhiều sự chuyển tiếp. Chúng ta sau đó sẽ biểu thị, làm thế nào tất cả các quá trình chuyển tiếp có thể được thay thế bằng một chuyển tiếp duy nhất.



Hình 4-2. Sơ đồ trạng thái với ngoại lệ **k**

**Truy cập đồng thời:** những hệ thống thực là những hệ thống phân tán, tồn tại đồng thời. Do đó, cần thiết để có thể xác định đồng thời thuận tiện.

**Đồng bộ hóa và truyền thông:** các hoạt động đồng thời phải có khả năng liên lạc và nó phải khả dĩ phù hợp với việc sử dụng các tài nguyên. Chẳng hạn, cần thiết biểu thị sự loại trừ lẫn nhau.

**Sự hiện diện của các yếu tố lập trình:** các ngôn ngữ lập trình thông thường đã được chứng minh là một phương tiện thuận tiện để thể hiện các tính toán cần phải được thực hiện. Do đó, các yếu tố ngôn ngữ lập trình nên có sẵn trong kỹ thuật đặc tả được sử dụng. Những sơ đồ trạng thái cổ điển không đáp ứng yêu cầu này.

**Có thể thực hiện được:** Những đặc tả không tự động phù hợp với những ý tưởng trong đầu con người. Thực hiện các đặc tả kỹ thuật là một phương tiện kiểm tra đáng tin cậy. Các đặc tả kỹ thuật sử dụng ngôn ngữ lập trình có một lợi thế rõ ràng trong ngữ cảnh này.

**Hỗ trợ cho việc thiết kế các hệ thống lớn:** Có một xu thế hướng tới các chương trình phần mềm nhúng lớn và phức tạp. Công nghệ phần mềm đã tìm ra những cơ chế để thiết kế những hệ thống lớn như vậy. Chẳng hạn, hướng đối tượng là một cơ chế như vậy. Nó cần phải sẵn sàng trong phương pháp đặc tả kỹ thuật.

**Hỗ trợ lĩnh vực chuyên biệt:** Tất nhiên sẽ là tốt hơn nếu cùng một kỹ thuật đặc tả có thể được áp dụng cho tất cả các loại khác nhau của hệ thống nhúng, vì điều này sẽ giảm thiểu các nỗ lực để phát triển các kỹ thuật đặc tả kỹ thuật và công cụ hỗ trợ. Tuy nhiên, do phạm vi rộng các lĩnh vực ứng dụng, có rất ít hy vọng rằng một ngôn ngữ có thể được sử dụng để đại diện hiệu quả cho các đặc tả kỹ thuật trong mọi lĩnh vực. Chẳng hạn, những lĩnh vực ứng dụng tập trung và phân tán, thiên về điều khiển, thiên về xử lý dữ liệu đều có thể hưởng lợi từ các tính năng ngôn ngữ chuyên dụng đối với các lĩnh vực đó.

**Có thể đọc được:** Tất nhiên, những đặc tả kỹ thuật phải đọc được bởi con người. Tốt nhất là, chúng cũng có thể đọc được bằng máy trong trình tự để xử lý chúng trong một máy tính.

**Tính khả chuyển và linh hoạt:** Các đặc tả kỹ thuật phải được độc lập với các nền tảng phần cứng cụ thể để chúng có thể dễ dàng sử dụng cho một loạt các nền tảng



mục tiêu. Chúng cần phải linh hoạt sao cho những thay đổi nhỏ của tính năng hệ thống cũng chỉ yêu cầu những thay đổi nhỏ trong đặc tả.

**Điểm kết thúc:** Nó phải có tính khả thi để xác định quá trình sẽ hoàn thành từ đặc tả kỹ thuật.

**Hỗ trợ các thiết bị I/O không tiêu chuẩn:** Nhiều hệ thống nhúng sử dụng các thiết bị I/O khác với các thiết bị thông thường được tìm thấy trên máy PC. Cần phải có khả năng mô tả những đầu vào và những đầu ra cho những thiết bị đó thuận tiện.

**Những thuộc tính không hoạt động:** các hệ thống thực tế phải thể hiện một số thuộc tính không hoạt động, chẳng hạn như sai hỏng cho phép, kích thước, tính co giãn, thời gian sống dự kiến, công suất tiêu thụ, trọng lượng, thân thiện với người sử dụng, tương thích điện từ (EMC) v.v. Không có hy vọng rằng tất cả những thuộc tính này có thể được định nghĩa một cách chính thức.

**Mô hình tính toán thích hợp:** Để mô tả tính toán, các mô hình tính toán là bắt buộc. Các mô hình như vậy sẽ được mô tả trong phần tiếp theo.

Từ danh sách các yêu cầu, đã thể hiện rõ ràng rằng sẽ không có bất kỳ ngôn ngữ chính thức nào có khả năng đáp ứng tất cả các yêu cầu này. Bởi vậy, trong thực tế, chúng ta phải sống với những thỏa hiệp. Việc lựa chọn ngôn ngữ được sử dụng cho một thiết kế thực tế sẽ phụ thuộc vào các miền ứng dụng và môi trường mà trong đó thiết kế sẽ được thực hiện. Trong phần sau, chúng ta sẽ trình bày một khảo sát các ngôn ngữ có thể được sử dụng cho các thiết kế thực tế.

#### **a. Mô hình tính toán**

Những ứng dụng công nghệ thông tin đã tồn tại cho đến nay rất nhiều dựa vào mô hình máy tính von Neumann của tính toán tuần tự. Mô hình này là không thích hợp cho các hệ thống nhúng, đặc biệt là những hệ thống có yêu cầu các câu thời gian thực, vì không có khái niệm về thời gian trong máy tính von Neumann. Các mô hình tính toán khác đầy đủ hơn.

#### **b. Biểu đồ trạng thái (StateCharts)**

Ngôn ngữ thực tế đầu tiên sẽ được trình bày là StateCharts. StateCharts đã được giới thiệu vào năm 1987 bởi David Harel và sau đó mô tả chính xác hơn. StateCharts mô tả việc truyền thông trong những máy trạng thái hữu hạn. Nó dựa trên khái niệm bộ nhớ chia sẻ truyền thông.

#### **c. Những đặc trưng ngôn ngữ khái quát**

Phần trước cung cấp cho chúng ta một số ví dụ đầu tiên về các thuộc tính của các ngôn ngữ đặc tả kỹ thuật. Những ví dụ này giúp chúng ta hiểu được một cuộc thảo luận tổng quát hơn các thuộc tính ngôn ngữ trong phần này trước khi chúng ta tiếp tục thảo luận về ngôn ngữ trong các phần tiếp theo. Có một số đặc điểm mà trên đó chúng ta có thể so sánh những thuộc tính của các ngôn ngữ. Thuộc tính đầu tiên là liên quan đến việc phân biệt giữa các mô hình xác định và không xác định đã được đề cập đến trong cuộc thảo luận của chúng ta về StateCharts.

### 4.1.3 Phân hoạch phần cứng - phần mềm

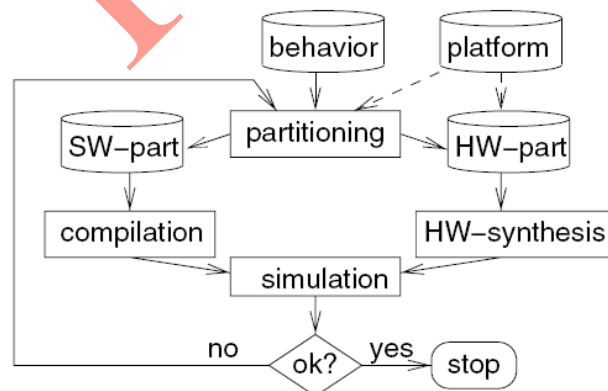
Trong quá trình thiết kế, chúng ta phải giải quyết vấn đề thực hiện các đặc tả kỹ thuật hoặc trong phần cứng hoặc ở dạng của các chương trình chạy trên bộ vi xử lý. Phần này mô tả một số kỹ thuật để lập bản đồ này. Áp dụng các kỹ thuật này, chúng ta sẽ có thể quyết định những phần phải được thực hiện trong phần cứng và những phần sẽ được thực bởi phần mềm.

Bởi phân hoạch phần cứng/phần mềm, có nghĩa là chúng ta ánh xạ các nút biểu đồ nhiệm vụ cho một trong hai phần cứng hoặc phần mềm. Một thủ tục tiêu chuẩn cho việc nhúng phân vùng phần cứng/phần mềm vào tiến trình thiết kế tổng thể được hiển thị trong Hình 4-3. Chúng ta bắt đầu từ một đại diện chung của đặc tả kỹ thuật, ví dụ ở dạng đồ thị nhiệm vụ và thông tin về nền (platform).

Đối với mỗi nút của đồ thị nhiệm vụ, chúng ta cần thông tin liên quan đến các nỗ lực cần thiết và những lợi ích nhận được từ việc lựa chọn một sự thực hiện nào đó các nút này. Ví dụ, thời gian thực hiện phải được dự đoán. Rất khó để dự đoán thời gian cần thiết cho truyền thông. Tuy nhiên, hai nhiệm vụ đòi hỏi một băng thông truyền thông rất cao tốt nhất nên được ánh xạ tới các thành phần giống nhau. Phương pháp lập được sử dụng trong nhiều trường hợp. Một giải pháp ban đầu cho vấn đề phân vùng được tạo ra, phân tích và sau đó được cải thiện.

Một số phương pháp tiếp cận cho phân hoạch được giới hạn để ánh xạ các nút đồ thị nhiệm vụ tới hoặc phần cứng chức năng chuyên biệt hoặc phần mềm chạy trên một bộ xử lý đơn. Việc phân hoạch như vậy có thể được thực hiện với các thuật toán bipartitioning (phân đôi) cho các đồ thị nhiệm vụ.

Những thuật toán phân hoạch phức tạp hơn có khả năng lập bản đồ các nút đồ thị cho hệ thống đa xử lý và phần cứng. Trong phần sau, chúng ta sẽ mô tả cách làm việc của thuật toán này bằng cách sử dụng một kỹ thuật tối ưu hóa tiêu chuẩn từ các hoạt động nghiên cứu, lập trình số nguyên. Trình bày của chúng ta được dựa trên một phiên bản đơn giản hóa của các đề xuất tối ưu hóa cho công cụ thiết kế COOL (codesign tool).



Hình 4-3. Tổng quan về phân hoạch phần cứng/phần mềm

#### a. COOL

Đối với COOL, đầu vào bao gồm ba phần:

**Công nghệ mục tiêu:** Phần này của đầu vào cho COOL bao gồm những thông tin về các thành phần nền phần cứng sẵn có. COOL hỗ trợ hệ thống đa xử, nhưng

yêu cầu tất cả các bộ xử lý là cùng loại, vì nó không bao gồm sự lựa chọn bộ vi xử lý tự động hay bằng tay. Tên của bộ xử lý được sử dụng (cũng như các thông tin về trình biên dịch tương ứng) phải được bao gồm trong phần này của đầu vào cho COOL. Đối với phần cứng ứng dụng chuyên biệt, các thông tin phải có đủ để bắt đầu tổng hợp phần cứng tự động với tất cả các thông số cần thiết. Đặc biệt, thông tin về các thư viện công nghệ phải được cung cấp.

**Những ràng buộc thiết kế:** Phần thứ hai của đầu vào bao gồm những ràng buộc thiết kế như thông lượng yêu cầu, độ trễ, kích thước bộ nhớ tối đa, hoặc diện tích tối đa cho phần cứng ứng dụng chuyên biệt.

**Hành vi:** Phần thứ ba của đầu vào mô tả hành vi tổng thể được yêu cầu. Đồ thị nhiệm vụ phân cấp được sử dụng cho việc này. Chúng ta có thể nghĩ ra, ví dụ bằng cách sử dụng đồ thị nhiệm vụ phân cấp cho việc này.

COOL sử dụng hai loại giới hạn (edge): giới hạn truyền thông và giới hạn về phân định thời gian. Giới hạn truyền thông có thể chứa thông tin về số lượng thông tin được trao đổi. Giới hạn về phân định thời gian cung cấp những sự ràng buộc về phân định thời gian. COOL đòi hỏi cách xử lý của mỗi nút trong các nút lá (leaf node) của phân cấp đồ thị đã biết. COOL cho là cách xử lý này đã được quy định trong VHDL.

Đối với phân vùng, COOL sử dụng các bước sau:

**1 Chuyển đổi** hành vi thành một mô hình đồ thị nội bộ.

**2 Chuyển đổi** hành vi của mỗi nút từ VHDL vào C.

**3 Biên dịch tất cả các chương trình C** cho bộ xử lý mục tiêu đã chọn, tính toán kích thước của chương trình kết quả, dự toán thời gian thực hiện kết quả. Nếu mô phỏng được sử dụng sau đó, thì dữ liệu đầu vào mô phỏng phải được sẵn sàng.

**4 Tổng hợp các thành phần phần cứng:** Đối với mỗi nút lá, phần cứng ứng dụng chuyên biệt phải được tổng hợp. Từ đó một số lớn thành phần phần cứng có thể phải được tổng hợp, tổng hợp phần cứng không nên quá chậm. Có thể nhận thấy rằng các công cụ tổng hợp thương mại tập trung tổng hợp ở mức cổng có thể là quá chậm để có ích cho COOL. Tuy nhiên, các công cụ tổng hợp cao cấp làm việc tại mức-chuyển đổi-thanh ghi (sử dụng các bộ cộng, các thanh ghi, và bộ dồn kênh như là các thành phần, thay vì các cổng) cung cấp tốc độ tổng hợp đầy đủ. Ngoài ra, các công cụ như vậy có thể cung cấp đầy đủ các giá trị chính xác cho những thời gian trì hoãn và diện tích silic yêu cầu. Trong thực hiện thực tế, công cụ tổng hợp cấp cao OSCAR [Landwehr and Marwedel, 1997] được sử dụng.

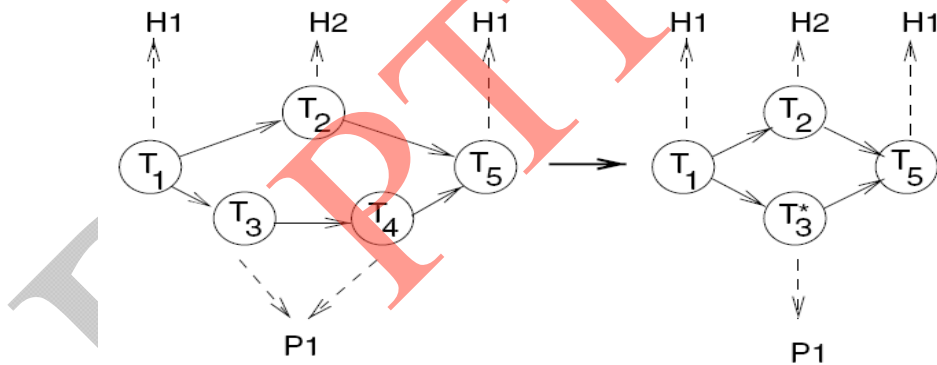
**5 Trải phẳng sự phân cấp:** Bước tiếp theo sẽ khai triển một đồ thị nhiệm vụ phẳng từ lưu đồ có phân cấp. Khi không có việc sát nhập hoặc chia tách các nút được thực hiện, đồ thị tiết được sử dụng bởi nhà thiết kế được duy trì. Chi phí và thông tin thực hiện thu được từ quá trình biên tập và tổng hợp phần cứng được bổ sung vào các nút. Đây thực sự là một trong những ý tưởng chính của COOL: các thông tin cần thiết cho phân vùng phần cứng/phần mềm được tính toán trước và nó được tính với độ chính xác tốt. Thông tin này thiết lập cơ sở để tạo ra các thiết kế chi phí tối thiểu thỏa mãn các ràng buộc thiết kế.

**6 Tạo ra và giải quyết một mô hình toán của vấn đề tối ưu hóa:** COOL sử dụng lập trình số nguyên (IP) để giải quyết vấn đề tối ưu hóa. Một thiết bị giải IP thương

mại được dùng để tìm những giá trị cho những biến quyết định đến việc tối thiểu hoá chi phí. Giải pháp là tối ưu đối với các hàm chi phí bắt nguồn từ những thông tin có sẵn. Tuy nhiên, chi phí này chỉ bao gồm một sự xấp xỉ thô của thời gian truyền thông. Thời gian truyền thông giữa hai nút bất kỳ của đồ thị nhiệm vụ phụ thuộc vào việc ánh xạ các nút này tương ứng tới các bộ xử lý và phần cứng. Nếu cả hai nút được ánh xạ tới cùng một bộ xử lý, truyền thông sẽ là cục bộ và do đó khá nhanh. Nếu các nút được ánh xạ tới các thành phần phần cứng khác nhau, truyền thông sẽ là không-cục bộ và có thể bị chậm hơn. Mô hình hóa các chi phí truyền thông cho tất cả các ánh xạ có thể có của các nút đồ thị nhiệm vụ sẽ làm cho mô hình rất phức tạp và do đó được thay thế bởi các cải tiến lặp của giải pháp ban đầu. Chi tiết hơn về bước này sẽ được trình bày dưới đây.

**7 Các cải tiến lặp:** Để làm việc với các ước lượng tốt về thời gian truyền thông, các nút liền kề được ánh xạ tới cùng một thành phần phần cứng bây giờ được hợp nhất. Sự hợp nhất này được thể hiện trên Hình 4-4.

Chúng ta giả định rằng các nhiệm vụ T1, T2 và T5 được ánh xạ tới các thành phần phần cứng H1 và H2, trong khi đó T3 và T4 được ánh xạ tới bộ xử lý P1. Tương ứng, truyền thông giữa T3 và T4 là truyền thông cục bộ. Vì vậy, chúng ta hợp nhất T3 và T4, và thừa nhận rằng việc giao tiếp giữa hai nhiệm vụ không đòi hỏi một kênh truyền thông. Thời gian truyền thông bây giờ có thể được ước lượng với độ chính xác được cải thiện. Đồ thị kết quả sau đó được sử dụng như đầu vào mới cho việc tối ưu hóa toán học. Các bước trước đó và hiện tại được lặp lại cho đến khi không còn nút đồ thị nào là có thể được sát nhập.



Hình 4-4. Hợp nhất các nút nhiệm vụ được ánh xạ đến cùng một thành phần phần cứng

**8 Tổng hợp giao diện:** Sau khi phân vùng, theo trình tự logic đòi hỏi phải tạo ra giao diện giữa các bộ xử lý, phần cứng ứng dụng chuyên biệt và bộ nhớ.

Tiếp theo, chúng ta sẽ mô tả bước 6 chi tiết hơn. Các mô hình IP cung cấp một phương pháp tiếp cận chung để mô hình hóa những vấn đề tối ưu hóa. Các mô hình IP bao gồm hai phần: một hàm chi phí và một tập các ràng buộc. Cả hai phần cùng bao hàm các tham chiếu đến một tập  $X = \{x_i\}$  của những biến giá trị nguyên. Những hàm chi phí phải là những hàm tuyến tính của những biến đó. Vì vậy, chúng phải có dạng chung:

$$C = \sum_{x_i \in X} a_i x_i, \text{ with } a_i \in \mathbb{Z}, x_i \in \mathbb{Z} \quad (4.1)$$

Tập J của các ràng buộc cũng phải chứa các hàm tuyến tính của các biến giá trị nguyên. Chúng phải có dạng:

$$\forall j \in J: \sum_{x_i \in X} b_{i,j} x_i \geq c_j, \text{ with } b_{i,j}, c_j \in \mathbb{Z} \quad (4.2)$$

Lưu ý rằng  $\geq$  có thể được thay thế bởi  $\leq$  trong phương trình (4.2) nếu các hằng số  $b_{i,j}$  được sửa đổi phù hợp.

Định nghĩa: Vấn đề lập trình số nguyên (IP-) là vấn đề của việc tối thiểu hoá hàm chi phí (4.1) lệ thuộc vào những ràng buộc nhận được trong biểu thức (4.2). Nếu tất cả các biến bị cưỡng bức là 0 hoặc 1, thì mô hình tương ứng được gọi là một mô hình lập trình số nguyên 0/1. Trong trường hợp này, các biến cũng được biểu thị như những biến quyết định (nhị phân).

Ví dụ, giả định rằng  $x_1$ ,  $x_2$  và  $x_3$  không âm và phải là số nguyên, tập các phương trình sau đại diện cho một mô hình 0/1-IP:

$$C = 5x_1 + 6x_2 + 4x_3 \quad (4.3)$$

$$x_1 + x_2 + x_3 \geq 2 \quad (4.4)$$

$$x_1 \leq 1 \quad (4.5)$$

$$x_2 \leq 1 \quad (4.6)$$

$$x_3 \leq 1 \quad (4.7)$$

Vì những ràng buộc, tất cả các biến đều chỉ có thể là 0 hoặc 1 (số nguyên không âm). Có bốn giải pháp khả dĩ. Các giải pháp này được liệt kê trong Bảng 4-1. Giải pháp với một chi phí bằng 9 là tối ưu.

Các ứng dụng đòi hỏi phải tối đa hóa một vài lợi ích thì hàm  $C'$  có thể được thay vào dạng thức ở trên bằng cách đặt  $C = -C'$ .

$x_1$	$x_2$	$x_3$	$C$
0	1	1	10
1	0	1	9
1	1	0	11
1	1	1	15

Bảng 4-1. Các giải pháp khả dĩ của vấn đề IP đang được trình bày

Các mô hình IP có thể được giải quyết tối ưu bằng cách sử dụng các kỹ thuật lập trình toán học. Thật không may, lập trình số nguyên là NP-đầy đủ và thời gian thực hiện có thể trở nên rất lớn. Tuy nhiên, nó rất hữu ích cho việc giải quyết các vấn đề tối ưu hóa miễn là kích thước mô hình không phải là vô cùng lớn. Thời gian thực hiện phụ thuộc vào số lượng các biến, cấu trúc và số lượng các ràng buộc. Các bộ giải IP tốt (như lp\_solve hay CPLEX) có thể giải quyết các vấn đề được cấu trúc tốt có chứa một vài nghìn biến trong thời gian tính toán chấp nhận được (ví dụ: vài phút). Để biết thêm thông tin về lập trình số nguyên và lập trình tuyến tính liên quan, hãy tham khảo các cuốn sách cùng chủ đề (ví dụ: to Wolsey). Mô hình hoá những vấn đề tối ưu hóa như vấn đề lập



trình số nguyên làm cho có cảm giác bất chấp sự phức tạp của vấn đề: nhiều vấn đề có thể được giải quyết trong thời gian thực hiện chấp nhận được và nếu chúng không thể, các mô hình IP cũng cung cấp một điểm khởi đầu tốt cho chẩn đoán.

Tiếp theo, chúng ta sẽ mô tả việc phân vùng có thể được mô hình bằng cách sử dụng một mô hình 0/1-IP như thế nào. Các tập chỉ số sau sẽ được sử dụng trong việc mô tả về mô hình IP:

- \* Tập chỉ số  $I$  biểu thị những nút đồ thị nhiệm vụ. Mỗi  $i \in I$  tương ứng với một nút biểu đồ nhiệm vụ.
- \* Tập chỉ số  $L$  biểu thị những kiểu nút đồ thị nhiệm vụ. Mỗi  $l \in L$  tương ứng với một kiểu nút đồ thị nhiệm vụ. Ví dụ, có thể có các nút mô tả căn bậc hai, các tính toán biến đổi Cosine rời rạc (DCT: Discrete Cosine Transform) hoặc biến đổi Fourier nhanh rời rạc (DFT: Discrete Fast Fourier Transform). Mỗi loại trong chúng được tính là một kiểu.
- \* Tập chỉ số  $KH$  biểu thị các kiểu thành phần phần cứng. Mỗi  $k \in KH$  tương ứng với một kiểu thành phần phần cứng. Ví dụ, có thể có các thành phần phần cứng đặc biệt cho DCT hoặc DFT. Có một giá trị chỉ số cho các thành phần phần cứng DCT và một cho các thành phần phần cứng FFT.
- \* Đối với mỗi thành phần phần cứng, có thể có nhiều bản sao, hay những "trường hợp". Mỗi trường hợp được xác định bởi một chỉ số  $j \in J$ .
- \* Tập chỉ số  $KP$  biểu thị những bộ xử lý. Mỗi  $k \in KP$  xác định một trong những bộ xử lý (tất cả đều là cùng loại).

Các biến quyết định sau đây được yêu cầu bởi mô hình:

- \*  $X_{i,k}$ : biến này sẽ là 1, nếu nút  $v_i$  được ánh xạ tới kiểu thành phần phần cứng  $k \in KH$  và 0 nếu không.
- \*  $Y_{i,k}$ : biến này sẽ là 1, nếu nút  $v_i$  được ánh xạ tới bộ xử lý  $k \in KP$  và 0 nếu không.
- \*  $NY_{l,k}$ : biến này sẽ là 1, nếu ít nhất một nút loại  $l$  được ánh xạ tới bộ xử lý  $k \in KP$  và 0 nếu không.
- \*  $T$  là một ánh xạ  $I \rightarrow L$  từ các nút đồ thị nhiệm vụ tới các kiểu tương ứng của chúng.

Trong trường hợp cụ thể của chúng ta, hàm chi phí tích lũy tổng chi phí của tất cả các đơn vị phần cứng:

$C$  = các chi phí cho bộ xử lý + các chi phí cho bộ nhớ + các chi phí của phần cứng ứng dụng chuyên biệt

Chúng ta rõ ràng sẽ giảm thiểu tổng chi phí nếu không có các bộ xử lý, bộ nhớ và phần cứng ứng dụng chuyên biệt đã được bao gồm trong "thiết kế". Do những ràng buộc, đây không phải là một giải pháp pháp lý. Bây giờ chúng ta có thể trình bày một mô tả ngắn gọn của một số các ràng buộc của mô hình IP:

\* **Những ràng buộc ấn định hoạt động:** Những ràng buộc này bảo đảm rằng mỗi hoạt động được thực hiện hoặc trong phần cứng hoặc trong phần mềm. Những ràng buộc tương ứng có thể được công thức hóa như sau:

$$\forall i \in I : \sum_{k \in KH} X_{i,k} + \sum_{k \in KP} Y_{i,k} = 1$$

Trong văn bản gốc, điều này có nghĩa như sau: với mọi nút đồ thị nhiệm vụ  $i$ , ràng buộc sau đây phải được duy trì:  $i$  được thực hiện hoặc trong phần cứng (thiết lập một trong những biến  $X_{i,k}$  tới 1, cho giá trị  $k$  nào đó) hoặc nó được thực hiện trong phần mềm (thiết lập một trong những biến  $Y_{i,k}$  tới 1, cho giá trị  $k$  nào đó).

Tất cả các biến được giả định là các số nguyên không âm:

$$X_{i,k} \in \mathbb{N}_0, \quad (4.8)$$

$$Y_{i,k} \in \mathbb{N}_0 \quad (4.9)$$

Những sự ràng buộc bổ sung bảo đảm rằng những biến quyết định  $X_{i,k}$  và  $Y_{i,k}$  có 1 như một giới hạn trên và, do đó, là các biến thực tế có giá trị 0/1:

$$\forall i \in I : \forall k \in KH : X_{i,k} \leq 1$$

$$\forall i \in I : \forall k \in KP : Y_{i,k} \leq 1$$

Nếu chức năng của một nút nhất định của kiểu  $l$  được ánh xạ tới bộ xử lý  $k$  nào đó, thì bộ nhớ chương trình của bộ xử lý này phải bao gồm một bản sao của phần mềm cho chức năng này:

$$\forall l \in L, \forall i : T(v_i) = c_l, \forall k \in KP : NY_{l,k} \geq Y_{i,k}$$

Trong văn bản gốc, điều này có nghĩa là: với mọi kiểu  $l$  thuộc các nút đồ thị nhiệm vụ và với mọi nút  $i$  thuộc kiểu này, ràng buộc sau đây phải được duy trì: nếu  $i$  được ánh xạ tới bộ xử lý  $k$  nào đó (biểu thị bởi  $Y_{i,k}$  là 1), thì phần mềm tương ứng với chức năng  $l$  phải được cung cấp bởi bộ xử lý  $k$ , và phần mềm tương ứng phải tồn tại trên bộ xử lý đó (biểu thị bởi  $NY_{l,k}$  là 1).

Những sự ràng buộc bổ sung bảo đảm rằng những biến quyết định  $NY_{l,k}$  cũng là những biến giá trị 0/1:

$$\forall l \in L : \forall k \in KP : NY_{l,k} \leq 1$$

**Những ràng buộc tài nguyên:** Tập tiếp theo của các ràng buộc đảm bảo rằng "không quá nhiều" các nút được ánh xạ tới cùng một thành phần phần cứng tại cùng một thời điểm. Chúng ta giả định rằng, cứ mỗi chu kỳ đồng hồ, nhiều nhất là một hoạt động có thể được thực hiện trên mỗi thành phần phần cứng. Thật không may, điều này có nghĩa rằng thuật toán phân vùng cũng phải tạo ra một lịch trình để thực hiện các nút đồ thị nhiệm vụ. Việc lập lịch tự nó đã là một vấn đề NP-đầy đủ cho hầu hết các trường hợp vấn đề có liên quan.

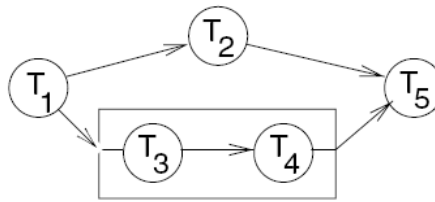
**Những ràng buộc mức ưu tiên:** Các ràng buộc này đảm bảo rằng lịch trình để thực hiện các hoạt động phù hợp với các ràng buộc mức ưu tiên trong đồ thị nhiệm vụ.

**Những ràng buộc thiết kế:** Những ràng buộc này đặt một giới hạn về chi phí của các thành phần phần cứng nào đó, chẳng hạn như bộ nhớ, các bộ xử lý hoặc khu vực dành cho phần cứng ứng dụng chuyên biệt.

**Những ràng buộc về phân định thời gian:** Những ràng buộc về phân định thời gian, nếu hiện diện trong đầu vào cho COOL, thì được chuyển đổi thành các ràng buộc IP.

Một số sự ràng buộc bổ sung, nhưng ít quan trọng hơn không được bao gồm trong danh sách này.





Hình 4-5. Đồ thị nhiệm vụ

Ví dụ: Trong phần sau, chúng ta sẽ trình bày về việc các ràng buộc này có thể được tạo ra như thế nào cho đồ thị nhiệm vụ trong Hình 4-5.

Giả sử rằng chúng ta có một thư viện thành phần phần cứng có chứa ba kiểu thành phần là H1, H2 và H3 với chi phí tương ứng là 20, 25 và 30 đơn vị chi phí. Hơn nữa, giả sử rằng chúng ta cũng có thể sử dụng một bộ xử lý P với phí là 5. Ngoài ra, chúng ta giả định rằng Bảng 4-2 mô tả thời gian thực hiện của các nhiệm vụ của chúng ta trên các thành phần này.

T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

Bảng 4-2. Thời gian thực hiện của các nhiệm vụ từ T1 đến T5 trên các thành phần

Các nhiệm vụ T1 đến T5 chỉ có thể được thực hiện trên bộ xử lý hoặc trên một đơn vị phần cứng ứng dụng chuyên biệt. Rõ ràng, bộ xử lý được coi là rẻ nhưng chậm trong việc thực hiện các nhiệm vụ T1, T2, và T5.

Những sự ràng buộc ấn định hoạt động sau đây phải được tạo ra, giả thiết rằng tối đa một bộ xử lý (P1) sẽ được sử dụng:

$$X_{1,1} + Y_{1,1} = 1 \text{ (Nhiệm vụ 1 được ánh xạ hoặc tới H1 hoặc tới P1)}$$

$$X_{2,2} + Y_{2,1} = 1 \text{ (Nhiệm vụ 2 được ánh xạ hoặc tới H2 hoặc tới P1)}$$

$$X_{3,3} + Y_{3,1} = 1 \text{ (Nhiệm vụ 3 được ánh xạ hoặc tới H3 hoặc tới P1)}$$

$$X_{4,3} + Y_{4,1} = 1 \text{ (Nhiệm vụ 4 được ánh xạ hoặc tới H3 hoặc tới P1)}$$

$$X_{5,1} + Y_{5,1} = 1 \text{ (Nhiệm vụ 5 được ánh xạ hoặc tới H1 hoặc tới P1)}$$

Hơn nữa, giả định rằng các kiểu của các nhiệm vụ T1 đến T5 là  $l = 1, 2, 3, 3$  và 1, tương ứng. Tiếp theo, những ràng buộc tài nguyên bổ sung sau đây được yêu cầu:

$$NY_{1,1} \geq Y_{1,1} \quad (4.10)$$

$$NY_{2,1} \geq Y_{2,1}$$

$$NY_{3,1} \geq Y_{3,1}$$

$$NY_{3,1} \geq Y_{4,1}$$

$$NY_{1,1} \geq Y_{5,1} \quad (4.11)$$

Phương trình (4.10) có nghĩa là: nếu nhiệm vụ 1 được ánh xạ tới bộ xử lý, thì chức năng  $l = 1$  phải được thực hiện trên bộ xử lý đó. Chức năng tương tự cũng phải được thực hiện trên bộ xử lý này nếu nhiệm vụ 5 được ánh xạ tới bộ xử lý (Phương trình (4.11)).

Chúng ta không bao gồm những ràng buộc về phân định thời gian. Tuy nhiên, rõ ràng là bộ xử lý chậm trong thực hiện một số nhiệm vụ và phần cứng ứng dụng chuyên biệt là cần thiết cho những ràng buộc thời gian nhỏ hơn 100 đơn vị thời gian.

Hàm chi phí là:

$$C = 20 * \#(H1) + 25 * \#(H2) + 30 * \#(H3) + 5 * \#(P)$$

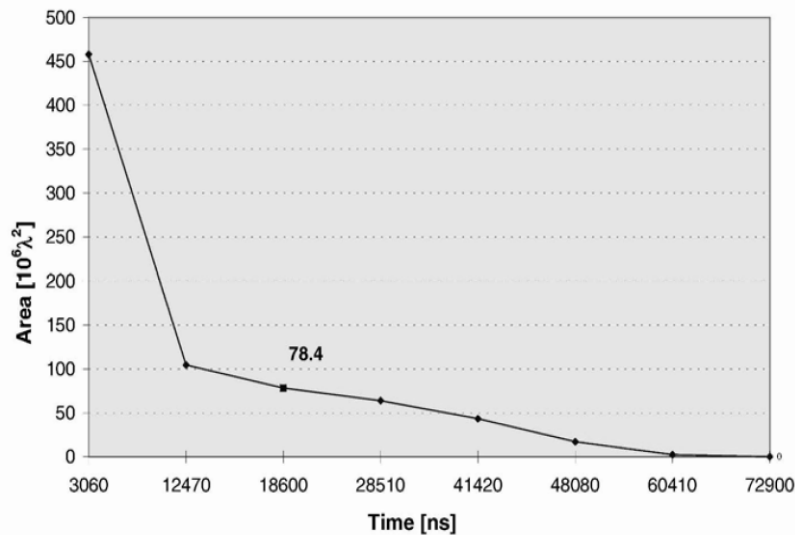
Trong đó  $\#()$  biểu thị số lượng các trường hợp sử dụng của các thành phần phần cứng. Số này có thể được tính toán từ những biến được giới thiệu cho đến lúc này nếu lịch trình cũng được đưa vào bản kê khai. Với một ràng buộc thời gian của 100 đơn vị thời gian, thiết kế chi phí tối thiểu bao gồm các thành phần H1, H2 và P. Điều này có nghĩa rằng nhiệm vụ T3 và T4 được thực hiện trong phần mềm và tất cả những nhiệm vụ khác được thực hiện trong phần cứng.

Nói chung, do sự phức tạp của việc phân hoạch kết hợp và các vấn đề lập lịch, chỉ những trường hợp vấn đề nhỏ của vấn đề kết hợp có thể được giải quyết trong thời gian chạy có thể chấp nhận được. Vì vậy, vấn đề là sự suy nghiệm được tách thành vấn đề lập lịch và phân hoạch: một phân hoạch ban đầu được dựa trên thời gian thực hiện dự tính và lập lịch cuối cùng được thực hiện sau khi phân hoạch. Nếu nó chỉ ra rằng lịch trình đã quá lạc quan, thì toàn bộ quá trình phải được lặp lại với những sự ràng buộc về phân định thời gian chặt hơn. Thí nghiệm đối với các ví dụ nhỏ đã cho thấy rằng chi phí cho các giải pháp suy nghiệm chỉ là 1 hoặc 2% lớn hơn chi phí của các kết quả tối ưu.

Phân hoạch tự động có thể được sử dụng để phân tích không gian thiết kế. Trong phần sau, chúng ta sẽ trình bày các kết quả cho một phòng thí nghiệm âm thanh, bao gồm các khối mixer, fader, echo, equalizer và balance. Ví dụ này sử dụng công nghệ nhằm mục tiêu trước đó để chứng minh tác dụng của phân hoạch. Phần cứng mục tiêu gồm có một bộ xử lý SPARC (chậm), bộ nhớ ngoài, và phần cứng ứng dụng chuyên biệt sẽ được thiết kế từ một thư viện 1 $\mu$  ASIC (Lỗi thời). Tổng thời gian trễ cho phép được thiết lập là 22675 ns, tương ứng với tốc độ lấy mẫu là 44,1 kHz, như được sử dụng trong đĩa CD. Hình 4-6 cho thấy những điểm thiết kế khác nhau mà có thể được tạo ra bằng cách thay đổi ràng buộc về thời gian trễ.

Đơn vị  $\lambda$  tham chiếu tới một đơn vị chiều dài phụ thuộc công nghệ. Nó về bản chất là một nửa của khoảng cách gần nhất giữa các tâm của hai dây kim loại trên chip (còn gọi là *half-pitch*). Điểm thiết kế ở bên trái tương ứng với một giải pháp thực hiện hoàn toàn trong phần cứng, điểm thiết kế ở bên phải là một giải pháp phần mềm. Những điểm thiết kế khác sử dụng một sự pha trộn của phần cứng và phần mềm. Điểm tương ứng với diện tích 78,4  $\lambda^2$  là điểm rẻ nhất thoả mãn vạch cấm.

Rõ ràng, ngày nay công nghệ đã tiến bộ để cho phép một thiết kế phòng thí nghiệm âm thanh trên nền phần mềm 100%. Tuy nhiên, ví dụ này chứng tỏ phương pháp thiết kế tiềm ẩn trong đó cũng có thể được sử dụng cho nhiều ứng dụng đòi hỏi khắt khe hơn, đặc biệt là trong lĩnh vực đa phương tiện tốc độ cao, như MPEG-4.



Hình 4-6. Không gian thiết kế cho phòng thí nghiệm âm thanh

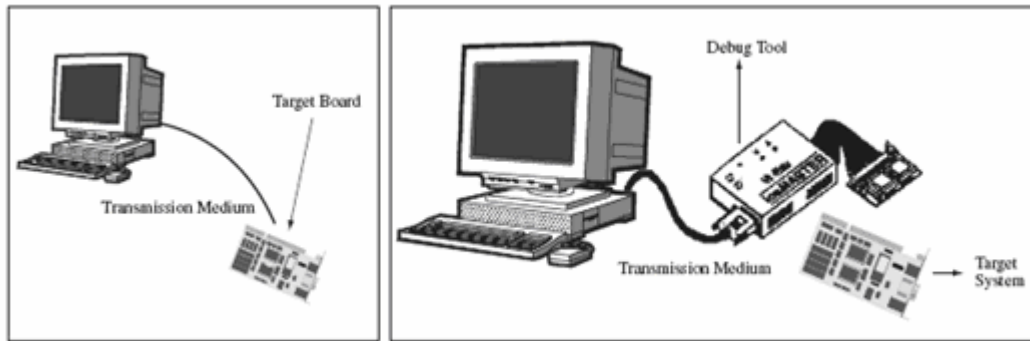
#### 4.1.4 Thiết kế hệ thống

Việc có tài liệu kiến trúc rõ ràng sẽ giúp các kỹ sư và lập trình viên của đội phát triển thực hiện hệ thống nhúng phù hợp với các yêu cầu. Trong suốt tài liệu này, các đề xuất thực tế đã được làm để thực hiện các thành phần khác nhau của một thiết kế đáp ứng các yêu cầu này. Ngoài sự hiểu biết các thành phần và khuyến nghị này, điều quan trọng là hiểu những gì mà các công cụ phát triển sẵn sàng trợ giúp trong việc thực hiện một hệ thống nhúng. Việc phát triển và tích hợp các thành phần phần cứng và phần mềm khác nhau của một hệ thống nhúng được thực hiện có thể thông qua các công cụ phát triển cung cấp mọi thứ từ việc nạp phần mềm vào phần cứng đến việc cung cấp điều khiển hoàn toàn qua những thành phần hệ thống khác nhau.

Các hệ thống nhúng thường không được phát triển trên một hệ thống đơn lẻ (ví dụ, bo mạch phần cứng của hệ thống nhúng) nhưng thường cần ít nhất một hệ thống máy tính khác kết nối với nền tảng nhúng để quản lý sự phát triển của nền tảng đó. Ngắn gọn hơn, một môi trường phát triển thường hình thành một **đích** (hệ thống nhúng đang được thiết kế) và một máy chủ (một PC, Sparc Station, hoặc một số hệ thống máy tính khác nơi mà mã thực sự được phát triển). Đích và máy chủ được kết nối bởi phương tiện truyền dẫn nào đó như nối tiếp, Ethernet, hoặc phương pháp khác. Nhiều công cụ khác, chẳng hạn như các công cụ tiện ích để ghi EPROM hoặc các công cụ gỡ lỗi, có thể được sử dụng trong môi trường phát triển cùng với máy chủ và đích. (Xem Hình 4-7)

Các công cụ phát triển quan trọng trong thiết kế nhúng có thể được đặt trên máy chủ, trên đích, hoặc có thể tồn tại độc lập. Những công cụ này thường thuộc một trong ba loại: **tiện ích**, **dịch thuật**, và các công cụ **gỡ lỗi**. Các công cụ tiện ích là các công cụ chung mà hỗ trợ trong phát triển phần mềm hoặc phần cứng, chẳng hạn như trình soạn thảo (cho việc viết mã nguồn), VCS (điều khiển phiên bản phần mềm) các phần mềm quản lý tập tin, bộ ghi ROM cho phép phần mềm được đưa vào ROM... Các công cụ dịch thuật chuyển đổi mã phát triển dự định cho đích thành dạng mà đích có thể thực thi, và các công cụ gỡ lỗi có thể được sử dụng để theo dõi và sửa lỗi trong hệ thống. Tất cả các loại công cụ phát triển là quan trọng đối với một dự án như thiết kế kiến trúc, bởi vì nếu

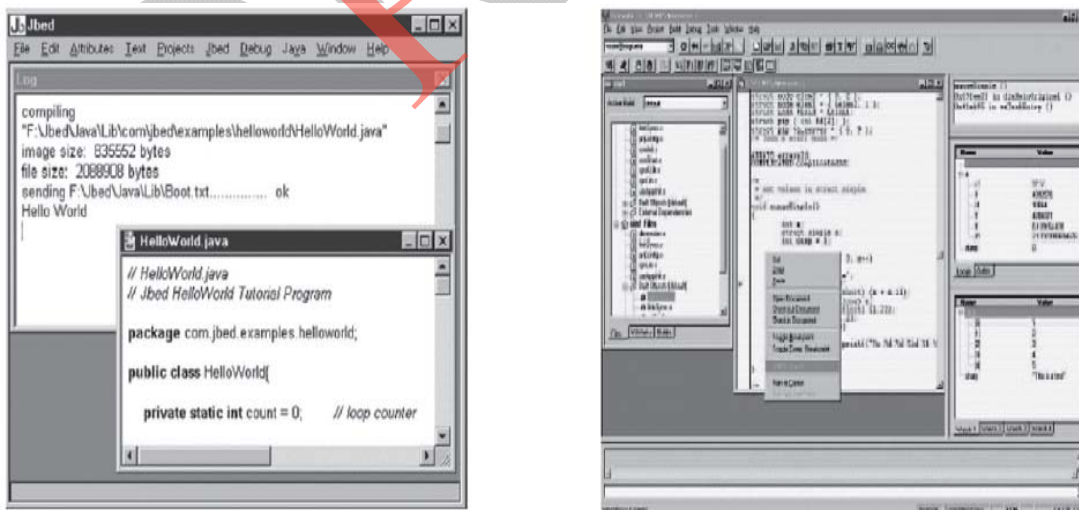
không có các công cụ đúng, việc thực hiện và gỡ lỗi hệ thống sẽ rất khó khăn, nếu không phải là không thể.



Hình 4-7. Môi trường phát triển

**Công cụ phần mềm tiện ích chính: viết mã trong một trình soạn thảo (Editor) hoặc môi trường phát triển tích hợp (IDE)**

Mã nguồn thường là được viết với một công cụ như một trình soạn thảo văn bản chuẩn ASCII, hoặc một **môi trường phát triển tích hợp (IDE)** nằm trên nền máy chủ (phát triển), như trong Hình 4-8. Một IDE là một tập hợp các công cụ, bao gồm một trình soạn thảo văn bản ASCII, tích hợp vào một ứng dụng giao diện người dùng. Trong khi trình soạn thảo văn bản ASCII bất kỳ có thể được sử dụng để viết bất kỳ loại mã nào, độc lập với ngôn ngữ và nền tảng, một IDE là cụ thể dành cho một nền tảng và thường được cung cấp bởi nhà cung cấp của IDE, một nhà sản xuất phần cứng (trong một bộ starter kit thường bao gồm bảng mạch phần cứng với các công cụ như một IDE hoặc trình soạn thảo văn bản), nhà cung cấp hệ điều hành, hoặc nhà cung cấp ngôn ngữ (Java, C, vv).

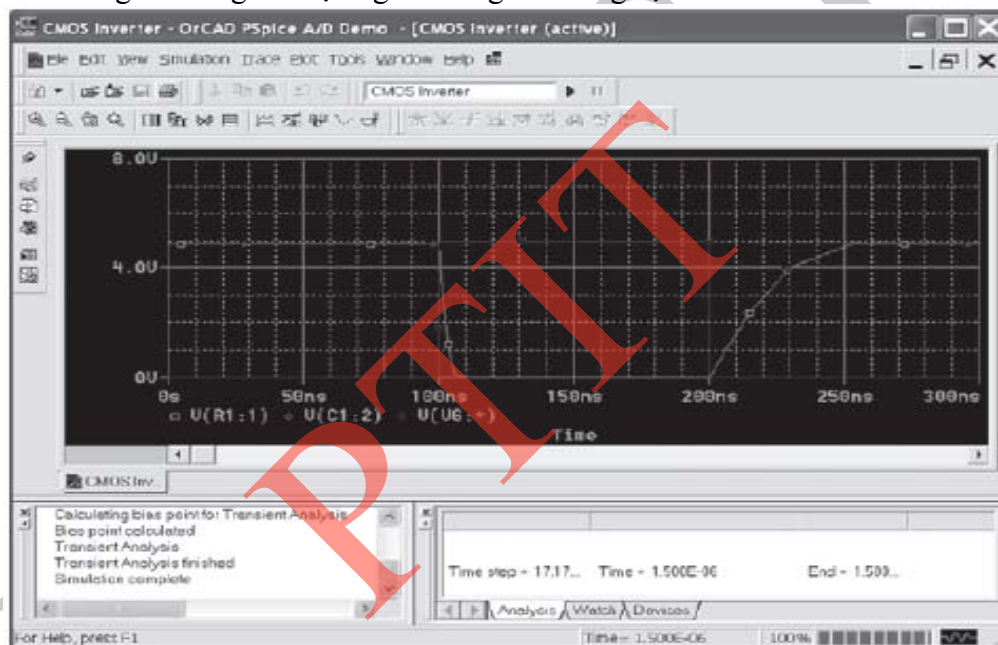


Hình 4-8. IDE

**Thiết kế trợ giúp máy tính (CAD) và phần cứng**

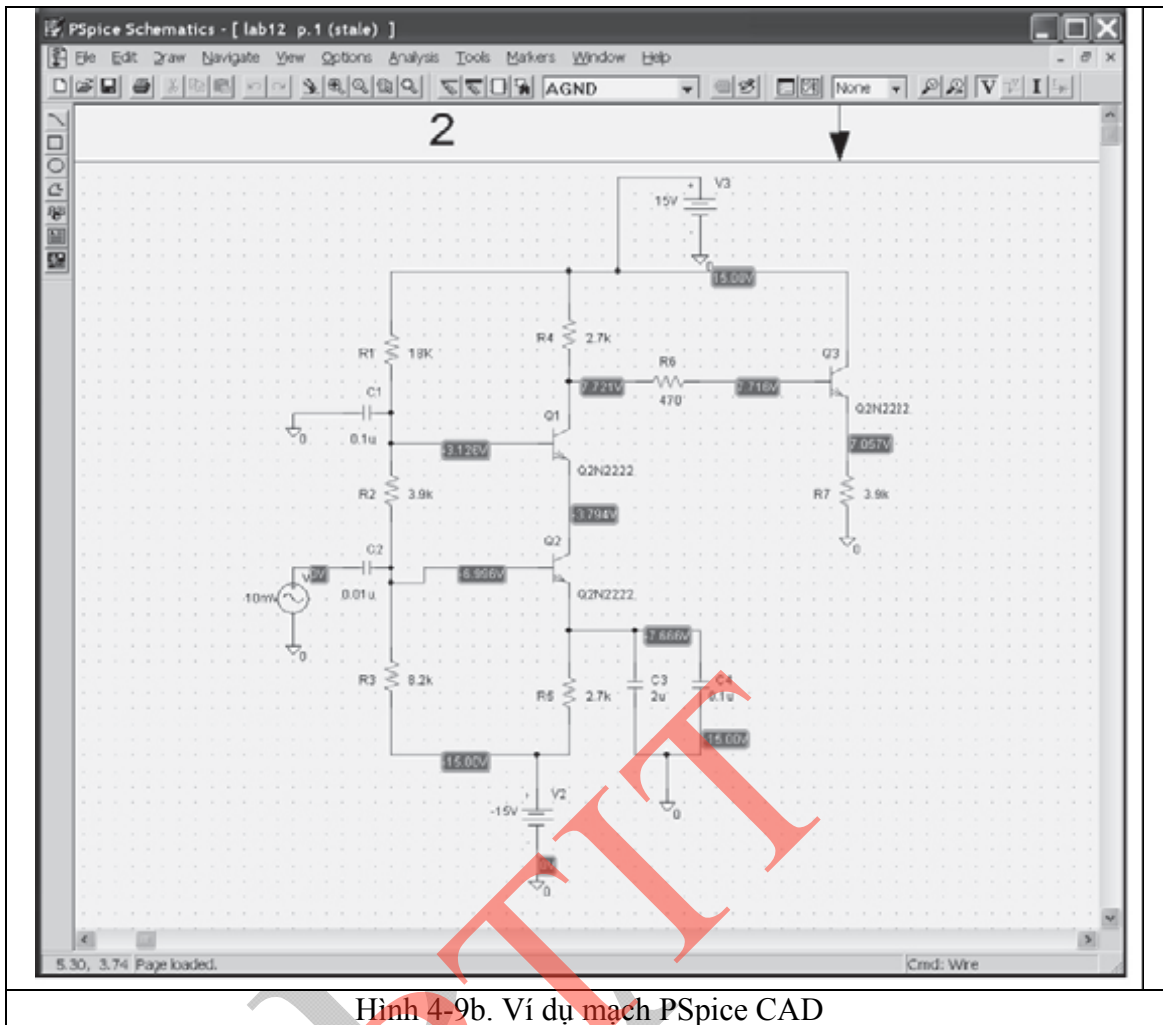
Các công cụ Thiết kế trợ giúp máy tính (CAD) thường được sử dụng bởi các kỹ sư phần cứng để mô phỏng mạch điện ở cấp độ điện để nghiên cứu hành vi của một mạch trong những điều kiện khác nhau trước khi họ thực sự xây dựng các mạch.

Hình 4-9a là một bản chụp của một trình mô phỏng mạch phổ biến tiêu chuẩn, được gọi là PSpice. Phần mềm mô phỏng mạch này là một biến thể của một trình mô phỏng mạch khác mà đã được phát triển tại Đại học California, Berkeley gọi là SPICE (Simulation Program with Integrated Circuit Emphasis: Chương trình mô phỏng với Mạch Tích Hợp Quan Trọng). PSpice là phiên bản PC của SPICE, và là một ví dụ của một trình mô phỏng mà có thể làm một số loại phân tích mạch, chẳng hạn như phi tuyến ngắn hạn, DC phi tuyến, AC tuyến tính, nhiễu, và sự méo dạng. Như trong Hình 4-9b, các mạch được tạo ra trong trình mô phỏng này có thể được tạo thành từ một loạt các phần tử tích cực và/hoặc thụ động. Nhiều công cụ mô phỏng mạch điện thương mại sẵn có nói chung là tương tự như PSpice về mục đích tổng thể, và chỉ khác nhau chủ yếu là về những phân tích có thể được thực hiện, các thành phần mạch có thể được mô phỏng, hoặc về bên ngoài của giao diện người dùng của công cụ.



Hình 4-9a. Ví dụ trình mô phỏng PSpice CAD





Hình 4-9b. Ví dụ mạch PSpice CAD

Do tầm quan trọng của thiết kế phần cứng và các chi phí liên quan, có nhiều ngành kỹ thuật công nghiệp mà trong đó các công cụ CAD được sử dụng để mô phỏng mạch. Cho một tập phức tạp các mạch trong một bộ xử lý hoặc trên một bảng mạch, rất là khó khăn, nếu không phải là không thể, để thực hiện một mô phỏng trên toàn bộ thiết kế, do đó một hệ thống phân cấp các mô phỏng và mô hình thường được sử dụng. Trong thực tế, việc sử dụng các mô hình là một trong những yếu tố quan trọng nhất trong thiết kế phần cứng, bất kể hiệu quả hoặc tính chính xác của mô phỏng này.

Ở cấp độ cao nhất, một mô hình hành vi của toàn bộ mạch được tạo ra cho cả hai mạch tương tự và số, và được sử dụng để nghiên cứu hành vi của toàn bộ mạch. Mô hình hành vi có thể được tạo ra với một công cụ CAD mà cung cấp tính năng này, hoặc có thể được viết bằng một ngôn ngữ lập trình tiêu chuẩn. Sau đó, phụ thuộc vào kiểu và cấu thành của mạch, các mô hình bổ sung được tạo ra xuống đến các thành phần tích cực và thụ động riêng lẻ của mạch, cũng như cho bất kỳ yếu tố phụ thuộc môi trường nào (ví dụ: nhiệt độ) mà mạch có thể có.

Ngoài việc sử dụng một số phương pháp cụ thể để viết các phương trình mạch cho một giả lập cụ thể, chẳng hạn như các phương pháp tiếp cận hoạt cảnh hoặc sửa đổi phương pháp nút, có các kỹ thuật mô phỏng để xử lý các mạch phức tạp bao gồm một hoặc một sự kết hợp nào đó của:

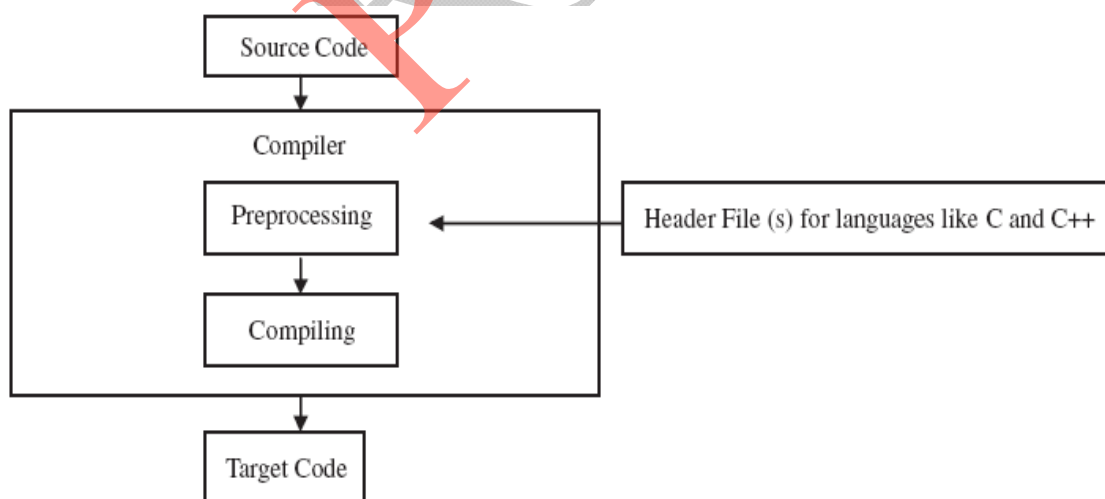
- phân chia các mạch phức tạp hơn thành các mạch nhỏ hơn, và sau đó kết hợp các kết quả.
- sử dụng các đặc tính đặc biệt của một số loại mạch nhất định.
- sử dụng các máy tính vector và/hoặc máy tính song song tốc độ cao.

**\* Những công cụ biên dịch: các bộ tiền xử lý, các trình thông dịch, các trình biên dịch, và các trình liên kết.**

Việc dịch mã đã được giới thiệu đầu tiên trong Chương 2, cùng với một lời giới thiệu ngắn gọn tới một số những công cụ được dùng trong việc Dịch mã, bao gồm các bộ tiền xử lý, các trình biên dịch, các trình biên dịch, và các trình kết nối. Như một tổng quát, sau khi mã nguồn đã được viết, nó cần phải được dịch sang mã máy, vì mã máy là ngôn ngữ duy nhất mà phần cứng có thể trực tiếp thực hiện. Tất cả các ngôn ngữ khác cần công cụ phát triển để tạo ra mã máy tương ứng mà phần cứng có thể hiểu. Cơ chế này thường bao gồm một hoặc sự kết hợp nào đó của các kỹ thuật phát mã máy như tiền xử lý, biên dịch, và/hoặc thông dịch. Các cơ chế này được thực hiện trong một loạt các công cụ phát triển phục vụ cho việc dịch.

Tiền xử lý là một bước tùy chọn có thể xuất hiện trước khi dịch hoặc thông dịch mã nguồn, và chức năng này thường được thực hiện bởi một bộ tiền xử lý. Vai trò của bộ tiền xử lý là tổ chức và cấu trúc lại mã nguồn để thực hiện dịch hoặc thông dịch mã này dễ dàng hơn. Bộ tiền xử lý có thể là một thực thể riêng biệt, hoặc có thể được tích hợp bên trong khối biên dịch, hoặc thông dịch.

Nhiều ngôn ngữ chuyển đổi mã nguồn, hoặc trực tiếp hoặc sau khi đã được tiền xử lý, tới mã đích (mã máy) thông qua việc sử dụng một trình biên dịch, một chương trình tạo ra một số ngôn ngữ đích, chẳng hạn như mã máy, mã Java byte, v.v., từ ngôn ngữ nguồn, chẳng hạn như hợp ngữ, C, Java, v.v. (xem Hình 4-10).



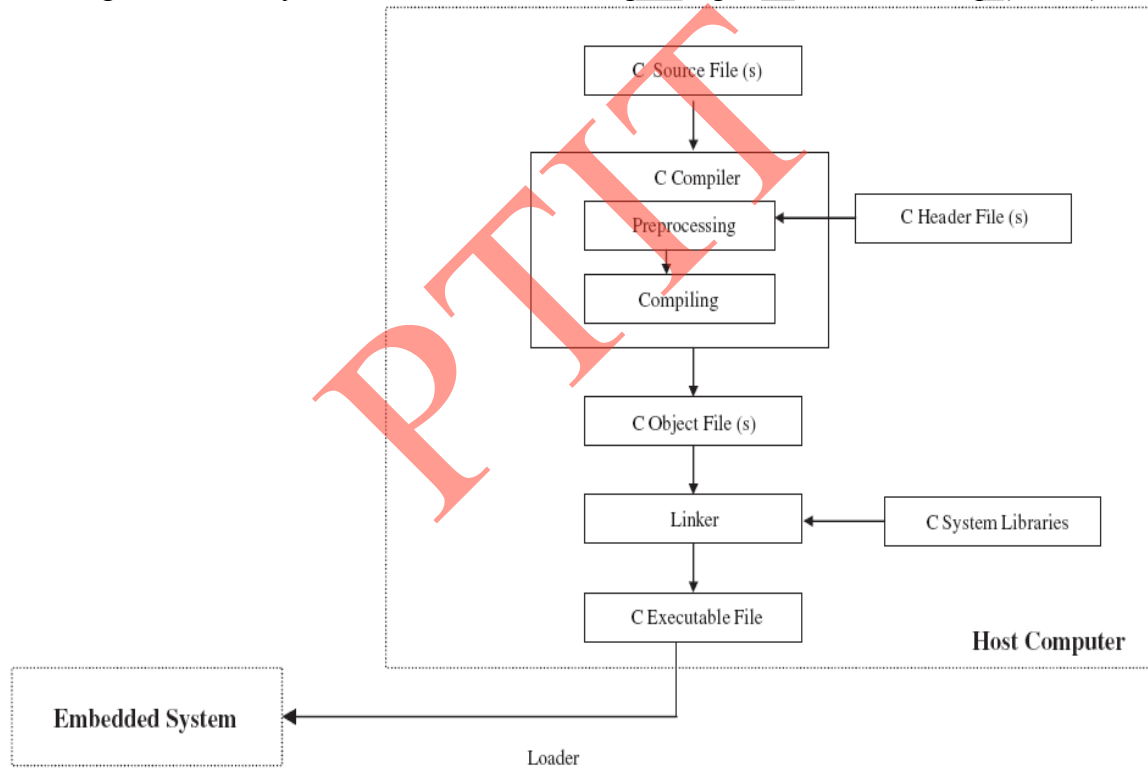
Hình 4-10. Sơ đồ biên dịch

Một trình biên dịch thông thường dịch tất cả các mã nguồn tới mã đích trong một lần. Như thường là trường hợp trong các hệ thống nhúng, hầu hết các trình biên dịch được đặt trên máy chủ của lập trình viên và tạo ra các mã đích cho các nền tảng phần



cứng khác biệt với nền tảng mà trình biên dịch thực sự đang chạy trên đó. Các trình biên dịch này thường được gọi là trình biên dịch chéo. Trong trường hợp hợp ngữ, một trình biên dịch hợp ngữ là một trình biên dịch chéo đặc biệt gọi là **trình dịch hợp ngữ (assembler)**, và nó sẽ luôn luôn tạo ra mã máy. Các trình biên dịch ngôn ngữ bậc cao khác thường được gọi bằng tên ngôn ngữ cộng với "trình biên dịch" (ví dụ: trình biên dịch Java (Java compiler), trình biên dịch C (C compiler)). Các trình biên dịch ngôn ngữ bậc cao có thể thay đổi rộng về những gì được tạo ra. Một số tạo ra mã máy, trong khi những cái khác tạo ra các ngôn ngữ cấp cao khác, mà sau đó yêu cầu những gì được tạo ra phải được chạy thông qua ít nhất một trình biên dịch nữa. Vẫn còn các trình biên dịch khác tạo ra mã hợp ngữ, mà mã sau đó phải được chạy thông qua một trình dịch hợp ngữ (assembler).

Sau khi tất cả việc biên dịch trên máy tính chủ của lập trình viên được hoàn thành, các tập tin mã đích còn lại thường được gọi là một tập tin đối tượng (object file), và có thể chứa bất cứ điều gì từ mã máy đến mã byte Java, tùy thuộc vào ngôn ngữ lập trình được sử dụng. Như trong Hình 4-11, một **trình liên kết (linker)** kết hợp tập tin đối tượng này với bất kỳ thư viện hệ thống cần thiết khác, để tạo ra tập tin thường được gọi là một tập tin nhị phân có thể thực thi, hoặc trực tiếp đưa vào bộ nhớ của bảng mạch hoặc sẵn sàng để được chuyển tới bộ nhớ của hệ thống nhúng đích bởi một bộ nạp (loader).



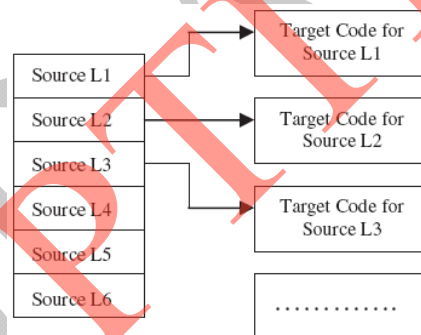
Hình 4-11: các bước biên dịch/liên kết và tập tin đối tượng kết quả, thực hiện trong C

Một trong những điểm mạnh cơ bản của một quy trình dịch là dựa trên khái niệm về sự sắp đặt phần mềm (còn gọi là sự sắp đặt đối tượng), khả năng phân chia phần mềm thành các module và tái định vị các module mã và dữ liệu này ở bất cứ nơi nào trong bộ nhớ. Đây là một tính năng đặc biệt hữu ích trong các hệ thống nhúng, bởi vì: (1) các thiết kế nhúng có thể chứa một vài loại khác nhau của bộ nhớ vật lý, (2) chúng thường có một số lượng hạn chế bộ nhớ so với các loại hệ thống máy tính khác; (3) bộ nhớ có thể

thường trở nên rất phân mảnh và chức năng chống phân mảnh là không có sẵn out-of-the-box hoặc quá đắt tiền, và (4) một số loại phần mềm nhúng có thể cần phải được thực hiện từ một vị trí bộ nhớ cụ thể(xác định).

Khả năng sắp đặt phần mềm theo vị trí này có thể được hỗ trợ bởi bộ xử lý chủ(master), trong đó cung cấp các chỉ dẫn chuyên dụng mà có thể được sử dụng để tạo ra "mã độc lập vị trí", hoặc nó có thể được chia cách bởi các công cụ dịch phần mềm riêng. Trong cả hai trường hợp, khả năng này phụ thuộc vào liệu có trình hợp dịch/trình biên dịch chỉ có thể xử lý các địa chỉ tuyệt đối, nơi mà các địa chỉ bắt đầu được cố định bởi phần mềm trước khi sự hợp dịch xử lý mã, hoặc liệu có sự hỗ trợ một sơ đồ định địa chỉ tương đối mà trong đó địa chỉ bắt đầu của mã có thể được chỉ rõ sau và nơi module mã được xử lý liên quan đến sự bắt đầu của module. Trường hợp một trình biên dịch/trình hợp dịch tạo ra các mô-đun tái định vị, các định dạng chỉ dẫn quá trình, và có thể thực hiện một số biên dịch liên quan đến các địa chỉ vật lý (tuyệt đối), ví dụ, dịch các địa chỉ tương đối còn lại thành địa chỉ vật lý, về cơ bản việc sắp đặt phần mềm, được thực hiện bằng trình liên kết.

Trong khi các IDE, các trình tiền xử lý, các trình biên dịch, các trình liên kết vẫn nằm trên hệ thống phát triển chủ, một số ngôn ngữ, chẳng hạn như Java và các ngôn ngữ kịch bản, có trình biên dịch hoặc trình thông dịch nằm trên đích. Một trình thông dịch tạo ra (phiên dịch ra) mã máy từ một dòng mã nguồn tại một thời điểm từ mã nguồn hoặc mã đích được tạo ra bởi một trình biên dịch trung gian trên hệ thống máy chủ (xem Hình 4-12 dưới đây).



Hình 4-12: Sơ đồ sự biên dịch

Một người phát triển nhúng có thể làm một tác động lớn trong việc lựa chọn các công cụ dịch cho một dự án bởi sự hiểu biết làm thế nào trình biên dịch làm việc và, nếu có những tùy chọn, bằng việc chọn trình biên dịch mạnh nhất có thể. Điều này là do trình biên dịch, trong phần lớn, xác định kích thước của các mã thực thi bởi cách thức nó dịch mã như thế nào.

Điều này không chỉ có nghĩa là lựa chọn một trình biên dịch dựa trên sự hỗ trợ của bộ vi xử lý chủ, phần mềm hệ thống đặc biệt, và các công cụ còn lại (một trình biên dịch có thể được mua riêng rẽ, như một phần của một starter kit từ một nhà cung cấp phần cứng, và/hoặc tích hợp trong một IDE). Nó cũng có nghĩa là lựa chọn một trình biên dịch dựa trên một tập hợp tính năng tối ưu hóa tính đơn giản, tốc độ, và kích cỡ của mã. Những tính năng này có thể, tất nhiên, khác nhau giữa các trình biên dịch của các ngôn ngữ khác nhau, hoặc thậm chí các trình biên dịch khác nhau của cùng một ngôn ngữ, nhưng như là một ví dụ sẽ bao gồm việc cho phép đặt các dòng lệnh hợp ngữ (in-line

assembly) bên trong mã nguồn và các hàm thư viện chuẩn mà làm cho việc lập trình mã nhúng dễ dàng hơn một chút. Việc tối ưu hóa mã cho hiệu suất có nghĩa là trình biên dịch hiểu và sử dụng các tính năng khác nhau của một ISA cụ thể, chẳng hạn như hoạt động toán học, tập thanh ghi, nhận biết được các loại ROM và RAM tích hợp trên chip (on-chip), số lượng các chu kỳ đồng hồ cho các các loại truy cập, v.v. Bởi việc hiểu làm thế nào trình biên dịch dịch mã, một người phát triển có thể nhận ra những gì hỗ trợ được cung cấp bởi trình biên dịch và tìm hiểu, ví dụ, làm thế nào để chương trình trong ngôn ngữ bậc cao được hỗ trợ bởi trình biên dịch một cách hiệu quả ("mã thân thiện trình biên dịch"), và khi để mã trong một ngôn ngữ bậc thấp hơn, nhanh hơn, chẳng hạn như hợp ngữ.

#### *Trình biên dịch nhúng lý tưởng*

Các hệ thống nhúng có các yêu cầu khác thường và những ràng buộc không điển hình của thế giới không-nhúng của các máy tính và các hệ thống lớn hơn. Trong nhiều cách, những tính năng và kỹ thuật thực hiện trong nhiều thiết kế trình biên dịch nhúng phát triển từ các thiết kế của trình biên dịch không nhúng. Các trình biên dịch này làm việc tốt cho sự phát triển hệ thống không nhúng, nhưng không giải quyết các yêu cầu khác biệt của sự phát triển các hệ thống nhúng, chẳng hạn như giới hạn về tốc độ và không gian. Một trong những lý do chính mã hợp ngữ vẫn còn rất thịnh hành trong các thiết bị nhúng sử dụng các ngôn ngữ bậc cao hơn là các nhà phát triển không thể nhìn thấy được những gì các trình biên dịch làm với mã bậc cao hơn. Nhiều trình biên dịch nhúng không cung cấp thông tin về cách mã được tạo ra. Vì vậy, các nhà phát triển đã không có cơ sở để đưa ra quyết định lập trình khi sử dụng một ngôn ngữ ở bậc cao hơn để cải thiện về kích thước và hiệu suất. Các tính năng trình biên dịch mà sẽ giải quyết một số yêu cầu, chẳng hạn như các yêu cầu về kích thước và tốc độ liên quan đến thiết kế hệ thống nhúng, bao gồm:

- Một tập tin danh sách trình biên dịch mà đánh mỗi dòng mã với những dự toán về số lần thực hiện dự kiến, phạm vi dự kiến của thời gian thực hiện, hoặc một số loại công thức được dùng để tính toán (thu thập từ các thông tin mục tiêu cụ thể của các công cụ khác được tích hợp với trình biên dịch).

- Một công cụ biên dịch cho phép nhà phát triển xem một dòng mã dưới hình thức biên dịch của nó, và đánh dấu bất kỳ vùng vấn đề tiềm tàng nào.

- Cung cấp thông tin về kích thước của mã thông qua một bản đồ kích thước chính xác, cùng với một trình duyệt cho phép lập trình viên xem bao nhiêu bộ nhớ đang được sử dụng bởi các thủ tục con riêng biệt.

Nhớ đến các đặc tính hữu ích này khi thiết kế hoặc khi mua một trình biên dịch nhúng.

#### **Công cụ gỡ lỗi**

Bên cạnh việc tạo ra kiến trúc, việc gỡ lỗi mã có lẽ là nhiệm vụ khó khăn nhất của chu kỳ phát triển. Gỡ lỗi chủ yếu là nhiệm vụ định vị và cố định lỗi trong hệ thống. Công việc này được thực hiện đơn giản khi lập trình viên là quen thuộc với các loại công cụ gỡ lỗi sẵn có và cách thức họ có thể sử dụng (loại thông tin được hiển thị trong Bảng 4-3).

Như đã thấy từ một số các mô tả trong Bảng 4-3, các công cụ gỡ lỗi cư trú và kết nối trong một vài sự kết hợp của những thiết bị độc lập, trên máy chủ, và/hoặc trên bảng mạch đích.

#### **Một nhận xét nhanh trên việc đo lường hiệu suất hệ thống với các chuẩn đánh giá**

Ngoài các công cụ gỡ lỗi, mỗi khi bảng mạch được khởi động và chạy, chuẩn đánh giá là các chương trình phần mềm thường được sử dụng để đo lường hiệu quả hoạt động (độ trễ, hiệu quả, vv) của các tính năng riêng biệt trong một hệ thống nhúng, như bộ vi xử lý chủ, hệ điều hành, hoặc JVM. Trong trường hợp của một hệ điều hành, ví dụ, hiệu suất được đo bằng hiệu quả của việc bộ vi xử lý chủ được sử dụng bởi các chương trình lập lịch của hệ điều hành. Bộ lập lịch cần ấn định định lượng thời gian thích hợp - thời gian một quá trình được tiếp cận với CPU- cho một quá trình, vì nếu định lượng thời gian là quá nhỏ, sự xung đột sẽ xuất hiện.

Mục tiêu chính của một ứng dụng chuẩn đánh giá là đại diện cho một khối lượng công việc thực tế cho hệ thống. Có rất nhiều ứng dụng chuẩn đánh giá có sẵn. Chúng bao gồm các chuẩn đánh giá EEMBC (Embedded Microprocessor Benchmark Consortium), tiêu chuẩn công nghiệp để đánh giá khả năng của các bộ xử lý nhúng, các trình biên dịch, và Java; Whetstone, trong đó mô phỏng các ứng dụng khoa học số học chuyên sâu; và Dhrystone, trong đó mô phỏng các ứng dụng lập trình hệ thống, thường bắt nguồn từ MIPS được giới thiệu trong Phần II. Các hạn chế của các chuẩn đánh giá là chúng có thể không được thực tế hoặc có thể tái sản xuất trong một thiết kế thế giới thực có liên quan đến nhiều hơn một tính năng của một hệ thống. Vì vậy, thường là tốt hơn khi sử dụng các chương trình nhúng thực sự mà sẽ được triển khai trên hệ thống để xác định không chỉ hiệu suất của phần mềm, mà toàn bộ hiệu suất hệ thống.

Tóm lại, khi làm sáng tỏ các chuẩn đánh giá, đảm bảo bạn hiểu chính xác những gì phần mềm đã chạy và những gì các chuẩn đánh giá đã đo hoặc đã không đo.

Loại công cụ	Công cụ gỡ lỗi	Mô tả	Ví dụ về sử dụng và những hạn chế
Phân cứng	Bộ mô phỏng trên mạch(In-Circuit Emulator: ICE)	Thiết bị hoạt động thay thế bộ vi xử lý trong hệ thống	<ul style="list-style-type: none"> <li>* giải pháp gỡ lỗi đắt nhất điển hình, nhưng có nhiều khả năng gỡ lỗi</li> <li>* có thể hoạt động ở tốc độ đầy đủ của bộ xử lý (phụ thuộc vào ICE) và tới phần còn lại của hệ thống đó là bộ vi xử lý</li> <li>* cho phép có thể xem và có thể thay đổi được nội dung bộ nhớ trong , các thanh ghi, các biến, v.v..trong thời gian thực</li> <li>* tương tự như những trình gỡ rối, cho phép đặt những điểm dừng, thực hiện từng bước,...</li> <li>* thường có lớp che bộ nhớ để mô phỏng bộ nhớ ROM</li> <li>* bộ xử lý phụ thuộc</li> <li>.....</li> </ul>
	Bộ mô phỏng ROM	Công cụ hoạt động thay thế ROM với các cáp kết nối với RAM công kép bên trong bộ mô	<ul style="list-style-type: none"> <li>* cho phép sửa đổi nội dung trong ROM (không giống như một trình gỡ lỗi)</li> <li>* có thể đặt breakpoint trong mã ROM, và xem mã ROM thời gian thực</li> <li>* thường không hỗ trợ ROM on-chip,</li> </ul>

		phòng ROM, mô phỏng ROM. Nó là một thiết bị phần cứng trung gian kết nối với các đích thông qua một số cáp (tức là BDM), và kết nối với máy chủ thông qua cổng khác	ASIC tùy chỉnh, v.v.. * có thể tích hợp với những trình gỡ lỗi .....
Chế độ gỡ lỗi nền (Background Debug Mode: BDM)	Phần cứng BDM trên bảng mạch (cổng và bộ giám sát gỡ lỗi tích hợp vào CPU chủ), và trình gỡ lỗi trên máy chủ, kết nối thông qua một cáp nối tiếp đến cổng BDM. Đầu nối trên cáp đến cổng BDM, thường được gọi là wiggler. Gỡ lỗi BDM đôi khi được gọi là gỡ lỗi On-Chip (On-Chip Debugging: OCD)	* thường rẻ hơn so với ICE, nhưng không linh động như ICE * quan sát sự thực hiện phần mềm một cách kín đáo trong thời gian thực * có thể đặt breakpoint để dừng sự thực hiện phần mềm * cho phép đọc và ghi tới các thanh ghi, RAM, các cổng I/O, v.v.. * phụ thuộc bộ xử lý/đích, giao diện gỡ lỗi độc quyền của Motorola .....	
IEEE 1149.1 Joint Test Action Group (JTAG)	phần cứng tương thích JTAG trên bảng mạch	* tương tự như BDM, nhưng không độc quyền cho kiến trúc cụ thể (là một tiêu chuẩn mở) .....	
IEEE-ISTO Nexus 5001	Các tùy chọn của cổng JTAG, cổng tương thích Nexus, hoặc cả hai, một số lớp phù hợp (phụ thuộc độ phức tạp của bộ xử lý chủ, lựa chọn kỹ thuật, v.v..)	* cung cấp khả năng mở rộng chức năng gỡ lỗi tùy theo mức độ tương thích của phần cứng .....	
Oscilloscope (máy hiện sóng)	Thiết bị tương tự thụ động vẽ đồ thị điện áp (trên	* giám sát tới 2 tín hiệu đồng thời * có thể đặt một điện áp kích hoạt để bắt giữ điện áp trong những điều kiện	



		trục thẳng đứng) so với thời gian (trên trục ngang), cho phép tìm ra điện áp chính xác tại một thời điểm nhất định	cụ thể * sử dụng như là vôn mét (mặc dù đắt hơn nhiều) * có thể kiểm tra mạch đang làm việc bằng cách xem tín hiệu trên bus hoặc các cổng I/O * bắt giữ những thay đổi trong một tín hiệu trên cổng I/O để kiểm tra các đoạn của phần mềm đang chạy, tính toán thời gian từ mỗi thay đổi tín hiệu tới thay đổi tiếp theo, v.v.. * độc lập với bộ xử lý
Logic Analyzer (bộ phân tích logic)	Thiết bị thụ động này có thể chụp và theo dõi đồng thời nhiều tín hiệu và có thể vẽ đồ thị chúng	 * có thể đắt * thường chỉ có thể theo dõi 2 mức điện áp (VCC và GND); các tín hiệu nằm ở khoảng giữa được vẽ như VCC hay GND * có thể lưu trữ dữ liệu * 2 chế độ hoạt động chính (thời gian, trạng thái) cho phép kích hoạt trên những thay đổi trạng thái của tín hiệu (tức là cao-xuống-thấp hoặc thấp-lên-cao) * bắt giữ những thay đổi trong một tín hiệu trên cổng I/O để kiểm tra các đoạn của phần mềm đang chạy, tính toán thời gian từ một thay đổi tín hiệu đến một thay đổi tín hiệu tiếp theo, v.v.. (chế độ thời gian) * có thể được kích hoạt để bắt giữ dữ liệu từ một sự kiện xung nhịp bên trong đích hoặc một xung nhịp bên trong bộ phân tích logic * có thể kích hoạt nếu bộ xử lý truy cập vào vùng cấm của bộ nhớ, ghi dữ liệu không hợp lệ vào bộ nhớ, hoặc truy cập một loại lệnh đặc biệt (chế độ trạng thái) * một số sẽ hiển thị mã hợp ngữ, nhưng thường không có thể đặt breakpoint và chạy bước đơn thông qua mã sử dụng bộ phân tích * bộ phân tích logic chỉ có thể truy cập dữ liệu được truyền từ bên ngoài đến và từ bộ xử lý, chứ không phải bộ nhớ trong, các thanh ghi, v.v.	

			<ul style="list-style-type: none"> <li>* bộ xử lý độc lập và cho phép xem hệ thống thực hiện trong thời gian thực với rất ít sự xâm nhập</li> </ul> <p>....</p>
	Voltmeter	Đo điện áp chênh lệch giữa 2 điểm trên mạch	<ul style="list-style-type: none"> <li>* để đo các giá trị điện áp đặc biệt</li> <li>* để xác định sự hiện diện của nguồn tại tất cả các điểm trên mạch</li> <li>* rẻ hơn những công cụ phần cứng khác</li> </ul> <p>.....</p>
	Ohmmeter	Đo điện trở giữa 2 điểm trên mạch	<ul style="list-style-type: none"> <li>* rẻ hơn so với các công cụ phần cứng khác</li> <li>* để đo những thay đổi về dòng điện/điện áp trong mối quan hệ với điện trở (dùng định luật Ôm <math>V=IR</math>)</li> </ul> <p>* ....</p>
	Multimeter	Đo cả điện áp lẫn điện trở	<ul style="list-style-type: none"> <li>* giống như volt và ohm meter</li> </ul> <p>* .....</p>
Phần mềm	Debugger (trình gỡ rối)	Công cụ gỡ lỗi chức năng	<p>Phụ thuộc vào trình gỡ rối - nói chung:</p> <ul style="list-style-type: none"> <li>* nạp/chạy từng bước/giám sát mã trên đích</li> <li>* thực hiện những điểm breakpoint để dừng sự thực hiện phần mềm</li> <li>* thực hiện những điểm breakpoint có điều kiện để dừng nếu điều kiện đặc biệt xuất hiện trong thời gian thực hiện</li> <li>* có thể sửa đổi nội dung của RAM, thường không thể sửa đổi nội dung của ROM</li> </ul> <p>.....</p>
	Profiler	Thu thập giá trị theo thời gian của các biến, các thanh ghi... được lựa chọn	<ul style="list-style-type: none"> <li>* thời gian bắt giữ phụ thuộc (khi) hành vi của phần mềm đang thực hiện</li> <li>* để bắt mẫu thực hiện (ở đâu) của phần mềm đang thực hiện</li> </ul> <p>....</p>
	Monitor	Giao diện gỡ lỗi tương tự như ICE, với phần mềm gỡ lỗi chạy trên đích và máy chủ. Một phần của monitor nằm trong ROM của bảng mạch đích (thường được gọi là tác nhân gỡ lỗi hoặc tác nhân	<ul style="list-style-type: none"> <li>* tương tự như in phát biểu nhưng nhanh hơn, ít xâm nhập, làm việc tốt hơn cho giới hạn thời gian thực mềm, nhưng không tốt cho thời gian thực cứng</li> <li>* chức năng tương tự tới trình gỡ rối</li> <li>* Hệ điều hành nhúng có thể bao gồm monitor cho những kiến trúc đặc biệt</li> </ul> <p>.....</p>



		đích), và một hạt nhân gỡ lỗi trên máy chủ. Phần mềm trên máy chủ và đích thường giao tiếp nối tiếp hoặc thông qua Ethernet (phụ thuộc vào những gì có sẵn trên đích).	
	Trình mô phỏng tập lệnh	Chạy trên máy chủ và mô phỏng bộ xử lý chủ và bộ nhớ (chương trình nhị phân có thể thực thi được nạp vào trình mô phỏng như là nó sẽ được nạp vào đích) và bắt chước phần cứng	<ul style="list-style-type: none"> <li>* thường không chạy ở cùng một tốc độ chính xác như đích thực tế, nhưng có thể đánh giá phản ứng và thông qua thời gian bằng cách xem xét sự khác biệt giữa tốc độ máy chủ và đích</li> <li>* thường không mô phỏng phần cứng khác mà có thể tồn tại trên đích, nhưng có thể cho phép thử nghiệm các thành phần được xây dựng trong bộ xử lý</li> <li>* có thể mô phỏng hành vi ngắt</li> <li>* bắt giữ giá trị của biến, bộ nhớ và các thanh ghi</li> <li>* sẽ không mô phỏng chính xác hành vi của các phần cứng thực tế trong thời gian thực</li> <li>* thường phù hợp hơn để thử nghiệm các thuật toán chứ không phải là phản ứng đối với các sự kiện bên ngoài của một kiến trúc hay bảng mạch (dạng sóng và như vậy cần được mô phỏng thông qua phần mềm)</li> <li>* thường rẻ hơn so với đầu tư vào phần cứng và các công cụ thực</li> </ul> <p>.....</p>
Manual (thủ công)	Luôn sẵn có, miễn phí hoặc rẻ hơn so với các giải pháp khác, hiệu quả, đơn giản để sử dụng nhưng thường xâm nhập cao hơn các loại công cụ khác, không đủ kiểm soát đối với sự lựa chọn sự kiện, cách ly, hoặc lặp lại. Khó khăn để gỡ lỗi hệ thống thời gian thực nếu phương pháp thủ công phải mất quá lâu để thực thi.		
	In các báo cáo	Công cụ gỡ lỗi chức năng, việc in các báo cáo được đưa vào bên trong mã để in thông tin thay đổi, vị trí trong mã thông	<ul style="list-style-type: none"> <li>* để xem đầu ra của các biến, giá trị các thanh ghi, v.v.. trong khi các mã đang chạy</li> <li>* để kiểm tra đoạn mã đang được thực hiện</li> <li>* có thể làm chậm đáng kể thời</li> </ul>

		tin, v.v.	gian thực hiện chương trình * có thể dẫn đến mất thời hạn trong hệ thống thời gian thực. .....
	Dumps	Công cụ gỡ lỗi chức năng kết xuất dữ liệu vào một số loại cấu trúc lưu trữ trong thời gian chạy	* giống như in báo cáo nhưng cho phép thời gian thực hiện nhanh hơn trong việc thay thế một số báo cáo in ấn (đặc biệt là nếu có một bộ lọc xác định những loại cụ thể của thông tin để kết xuất hoặc những điều kiện cần phải được đáp ứng để kết xuất dữ liệu vào cấu trúc) * xem nội dung của bộ nhớ tại thời gian chạy để xác định nếu có bất kỳ sự tràn stack/heap .....
	Counters/Timers	Công cụ gỡ lỗi hiệu suất và hiệu quả mà trong đó các bộ đếm hoặc các bộ định thời thiết được thiết lập lại và tăng lên tại các điểm khác nhau của mã	* thu thập thông tin thời gian thực hiện chung bằng cách giảm bớt xung nhịp hệ thống hoặc đếm các chu kỳ bus, v.v.. * một số xâm nhập .....
	Fast Display	Công cụ gỡ lỗi chức năng trong đó các đèn LED được bật/tắt hoặc các màn hình LCD đơn giản được dùng để biểu diễn một số dữ liệu	* tương tự như in báo cáo nhưng nhanh hơn, ít xâm nhập, làm việc tốt cho các giới hạn thời gian thực * cho phép xác nhận các phần đặc biệt của mã đang chạy .....
	Ouput ports	Công cụ gỡ lỗi chức năng hiệu suất, hiệu quả trong đó các cổng đầu ra bật/tắt tại các điểm khác nhau trong phần mềm	* với một máy hiện sóng hoặc máy phân tích logic, có thể đo khi cổng bị bật/tắt và nhận được thời gian thực hiện giữa các lần bật/tắt của cổng * giống như trên nhưng có thể thấy trên máy hiện sóng mã đang được thực hiện tại vị trí đầu tiên * trong hệ thống đa nhiệm/đa luồng gán những cổng khác nhau tới mỗi luồng/nhiệm vụ để nghiên cứu hành vi .....

Bảng 4-1: Những công cụ gỡ lỗi

Một số trong những công cụ này là các công cụ gỡ lỗi tích cực và xâm nhập được vào các hoạt động của hệ thống nhúng, trong khi các công cụ gỡ lỗi khác thụ động nắm bắt các hoạt động của hệ thống không có sự xâm nhập khi hệ thống đang chạy. Gỡ lỗi một hệ thống nhúng thường đòi hỏi một sự kết hợp của những công cụ này để xác định tất cả các loại khác nhau của các vấn đề có thể phát sinh trong quá trình phát triển.

### *Cách rẻ nhất để gỡ lỗi*

Thậm chí với tất cả các công cụ có sẵn, các nhà phát triển vẫn còn phải cố gắng để giảm thời gian gỡ lỗi và chi phí, bởi vì 1) chi phí lỗi tăng đến gần hơn với thời gian sản xuất và tiến độ triển khai được, và 2) chi phí của một lỗi là logarit (nó có thể tăng mười lần khi được phát hiện bởi một khách hàng so với nếu nó đã được tìm thấy trong quá trình phát triển của thiết bị). Một số các phương tiện hiệu quả nhất của việc giảm thời gian gỡ lỗi và chi phí bao gồm:

- Không phát triển quá nhanh và cầu thả. Cách rẻ nhất và nhanh nhất để gỡ lỗi là không chèn bất kỳ lỗi nào ở vị trí đầu tiên. Phát triển nhanh và cầu thả thực sự làm chậm trễ tiến độ với phần lớn thời gian tiêu tốn cho việc gỡ lỗi những sai lầm.

- Kiểm tra hệ thống. Điều này bao gồm các kiểm tra phần cứng và phần mềm trong suốt quá trình phát triển mà đảm bảo rằng các nhà phát triển đang thiết kế theo các đặc tả kỹ thuật kiến trúc, và bất kỳ tiêu chuẩn nào khác được yêu cầu của các kỹ sư. Mã hay phần cứng không đáp ứng các tiêu chuẩn sẽ phải được "sửa lỗi" sau đó nếu các kiểm tra hệ thống không được sử dụng để đưa ra chúng ra nhanh chóng và rẻ (liên quan đến thời gian tiêu tốn cho việc gỡ lỗi và định vị tất cả lỗi mà với phần cứng và code nhiều hơn sau đó).

- Không sử dụng phần cứng bị lỗi hoặc mã được viết tồi. Một thành phần thường là sẵn sàng để được thiết kế lại khi các kỹ sư chịu trách nhiệm lo sợ việc thực hiện bất kỳ thay đổi nào đối với thành phần lỗi đó.

- Theo dõi các lỗi trong một tập tin văn bản chung hoặc sử dụng một trong nhiều các công cụ phần mềm theo dõi lỗi có sẵn. Nếu các thành phần (phần cứng hoặc phần mềm) đang tiếp tục gây ra những vấn đề, thì có thể dành thời gian để thiết kế lại thành phần đó.

- Đừng tiết kiệm với các công cụ gỡ lỗi. Một công cụ gỡ lỗi tốt (mặc dù đắt hơn) sẽ cắt giảm thời gian gỡ lỗi thì có trị giá hơn một tá các công cụ rẻ hơn mà, với không tốn nhiều thời gian và nhức đầu, chỉ có thể theo dõi các loại lỗi gặp phải trong quá trình thiết kế một hệ thống nhúng.

Và cuối cùng, một trong những phương pháp tốt nhất để giảm bớt thời gian và chi phí gỡ lỗi là đọc tài liệu được cung cấp bởi các nhà cung cấp và/hoặc các kỹ sư chịu trách nhiệm đầu tiên, trước khi cố gắng chạy hoặc sửa đổi bất cứ điều gì. Tôi đã nghe nói nhiều, nhiều lời bào chữa trong những năm qua, từ "Tôi không biết đọc cái gì" đến "Có tài liệu hướng dẫn không?" - Là tại sao một kỹ sư đã không đọc những tài liệu hướng dẫn. Các kỹ sư này đã dành nhiều giờ, nếu không phải nhiều ngày, về các vấn đề riêng lẻ với việc cấu hình phần cứng hay việc nhận được một phần của phần mềm chạy đúng. Tôi biết rằng nếu các kỹ sư này đọc tài liệu ở ngay thời điểm đầu tiên, vấn đề sẽ được giải quyết trong vài giây hoặc vài phút - hoặc có thể chẳng có vấn đề khó nào xuất hiện.

*Nếu bạn đang quá tải với tài liệu và không biết những gì để đọc đầu tiên, bất cứ tiêu đề nào đọc theo dòng như “Getting Started...”, “Booting up the system...”, hoặc “README” là các chỉ thị tốt của một nơi để bắt đầu. Hơn nữa, dành thời gian để đọc tất cả các tài liệu được cung cấp với bất kỳ phần cứng hay phần mềm để trở nên quen thuộc với những loại thông tin này, sẽ giúp ích trong trường hợp cần thiết sau này.*

---Dựa trên bài viết "Firmware Basics for the Boss" của Jack Ganssle,  
Embedded Systems Programming, tháng 2 năm 2004

### Khởi động (Boot-Up) hệ thống

Với những công cụ phát triển sẵn sàng để thử, và hoặc một bảng mạch tham chiếu hoặc một bảng mạch phát triển kết nối với máy chủ phát triển, đó là thời điểm để khởi động hệ thống và xem những gì sẽ xảy ra. Khởi động hệ thống có nghĩa là một vài loại cấp nguồn hoặc khởi động lại nguồn, chẳng hạn như một khởi động lại cứng bên trong/bên ngoài (tức là, tạo ra bởi một lỗi kiểm tra dừng, các cơ quan giám sát phần mềm, mất khóa của PLL, bộ gỡ rối, v.v. ), hoặc một khởi động lại mềm bên trong/bên ngoài (tức là, tạo ra bởi một trình sửa lỗi, mã ứng dụng, v.v.), xuất hiện. Khi nguồn được cấp tới một bảng mạch nhúng (bởi một khởi động lại), mã khởi động (start-up code), cũng được gọi là **boot code**, **bootloader**, mã **bootstrap**, hoặc **BIOS** (hệ thống vào ra cơ sở) tùy thuộc vào kiến trúc, trong ROM của hệ thống được nạp và thực hiện bởi bộ xử lý chủ. Một số kiến trúc nhúng (master) có một bộ đếm chương trình nội bộ được cấu hình tự động với một địa chỉ trong ROM mà trong đó sự bắt đầu của mã boot-up (hoặc bảng) được định vị, trong khi những cái khác được nối cứng để bắt đầu thực hiện tại một địa điểm cụ thể trong bộ nhớ.

Mã Boot khác ở chiều dài và chức năng tùy thuộc vào thời điểm trong chu kỳ phát triển bảng mạch, cũng như các thành phần của nền tảng thực tế cần sự khởi tạo. Các chức năng chung (tối thiểu) được thực hiện bởi mã khởi động trên các nền tảng khác nhau, những cái khởi tạo chức năng cơ bản phần cứng, bao gồm vô hiệu hóa ngắt, khởi tạo các bus, thiết lập các bộ xử lý chủ và tở trong một trạng thái cụ thể, và khởi tạo bộ nhớ. Phần khởi tạo phần cứng đầu tiên này của mã boot-up về cơ bản là thực hiện các trình điều khiển thiết bị khởi tạo, như được thảo luận trong Chương 2. Làm thế nào khởi tạo là thực sự được thực hiện, đó là-thứ tự mà các trình điều khiển được thực hiện-thường được phác thảo bởi tài liệu kiến trúc tổng thể hay trong tài liệu được cung cấp bởi các nhà sản xuất bảng mạch. Sau chuỗi khởi tạo phần cứng, thực hiện thông qua khởi tạo các trình điều khiển thiết bị, phần mềm hệ thống còn lại, nếu có, sau đó được khởi tạo. Mã bổ sung này có thể tồn tại trong ROM, cho một hệ thống đang được vận chuyển ra khỏi nhà máy, hoặc nạp từ một nền tảng máy chủ bên ngoài (xem hộp lời thoại với bootcodeExample).

```
bootcodeExample ()
{
...
// Serial Port Initialization Device Driver
initializeRS232 (UART,BAUDRATE,DATA_BITS,STOP_BITS,PARITY);
// Initialize Networking Device Driver
initializeEthernet(IPAddress,Subnet, GatewayIP, ServerIP);
//check for host development system for down loaded file of
rest of code to RAM
```

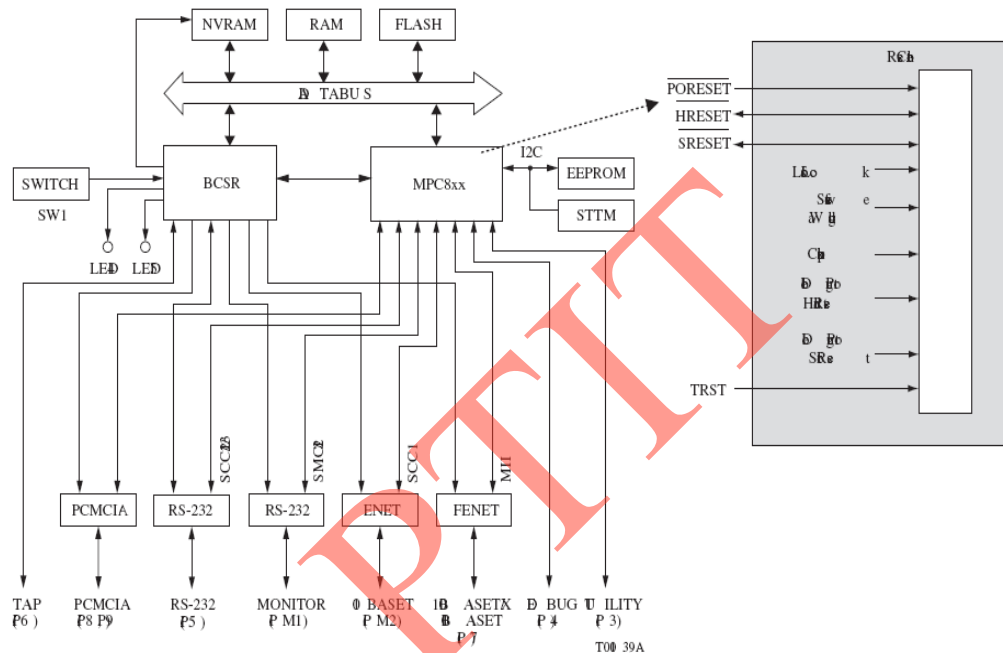
```

// through ethernet
// start executing rest of code(i.e. define memory map,
load OS, etc.)
...
}

```

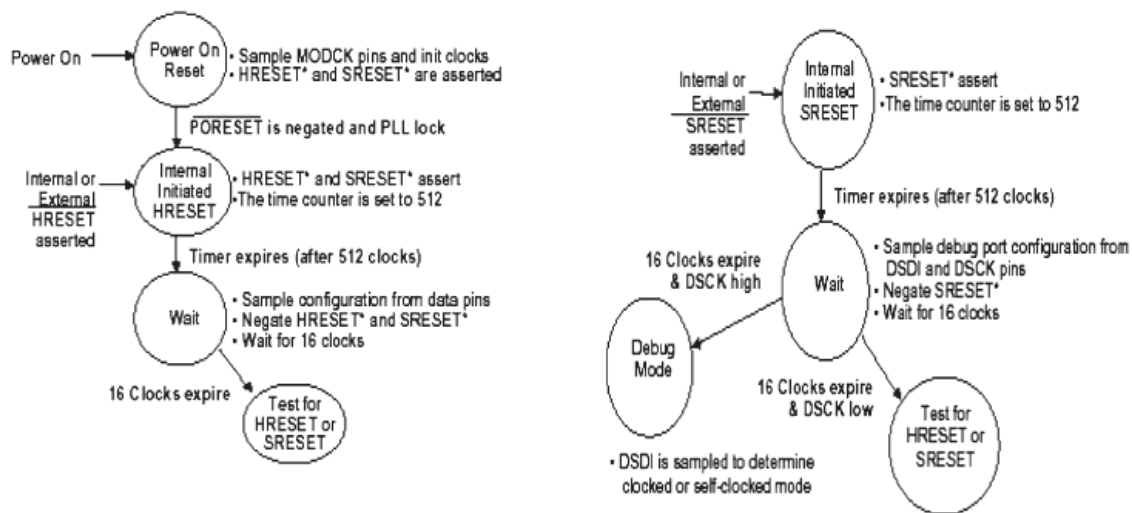
### Ví dụ khởi động bằng mạch dùng MPC823

Bộ xử lý MPC823 chứa một bộ điều khiển khởi động lại mà có trách nhiệm đáp ứng lại tới mọi nguồn khởi động lại. Những hành động được thực hiện bởi bộ điều khiển khởi động lại khác nhau tùy theo nguồn gốc của một sự kiện khởi động lại, nhưng nhìn chung quá trình bao gồm Cấu hình lại phần cứng, và sau đó lấy mẫu các chân dữ liệu hoặc sử dụng một hằng số mặc định nội bộ để xác định các giá trị khởi động lại ban đầu của các thành phần hệ thống.



Hình 4-13a: Sơ đồ giải thích





Hình 4-13b: Sơ đồ giải thích

## 4.2 Cài đặt và thử nghiệm hệ thống nhúng

Trong số các mục tiêu của việc thử nghiệm và đảm bảo chất lượng của một hệ thống là tìm lỗi trong thiết kế và theo dõi xem các lỗi được cố định. Bảo đảm chất lượng và thử nghiệm tương tự như gỡ lỗi, đã thảo luận ở trên, ngoại trừ các mục tiêu của gỡ lỗi là thực sự để sửa lỗi được phát hiện. Một khác biệt chính giữa gỡ lỗi và thử nghiệm hệ thống là gỡ lỗi thường xảy ra khi nhà phát triển gặp một vấn đề trong cố gắng để hoàn thành một phần của thiết kế, và sau đó thường thử nghiệm để thông qua những sửa chữa lỗi (có nghĩa là các thử nghiệm chỉ để đảm bảo hệ thống tối thiểu làm việc dưới trường hợp thông thường). Với thử nghiệm, mặt khác, lỗi được phát hiện như là kết quả của sự cố gắng để phá vỡ hệ thống, bao gồm cả *testing-to-pass* và *testing-to-fail*, nơi mà những yếu kém trong hệ thống được thăm dò.

Dưới thử nghiệm, các lỗi thường xuất phát từ hoặc hệ thống không trung thành với đặc tả kiến trúc hoặc không có khả năng kiểm tra hệ thống. Các loại lỗi gặp phải trong thử nghiệm phụ thuộc vào loại thử nghiệm đang được thực hiện. Nhìn chung, các kỹ thuật thử nghiệm thuộc một trong bốn mô hình: thử nghiệm *hộp đen tĩnh*, thử nghiệm *hộp trắng tĩnh*, thử nghiệm *hộp đen động*, hoặc thử nghiệm *hộp trắng động* (xem ma trận trong Hình 4-14). Thử nghiệm hộp đen xảy ra với một bộ thử nghiệm mà không có khả năng hiển thị các hoạt động nội bộ bên trong của hệ thống (không có sơ đồ nguyên lý, không có mã nguồn, v.v.). Thử nghiệm hộp đen được dựa trên tài liệu các yêu cầu sản phẩm nói chung, trái ngược với thử nghiệm hộp trắng (còn gọi là thử nghiệm hộp trong suốt hoặc thử nghiệm hộp thủy tinh) trong đó bộ thử nghiệm có thể truy cập vào mã nguồn, sơ đồ nguyên lý v.v. Thử nghiệm tĩnh được thực hiện trong khi hệ thống không hoạt động, trong khi thử nghiệm động được thực hiện khi hệ thống đang chạy.

	Thử nghiệm hộp đen (Black Box Testing)	Thử nghiệm hộp trắng (White Box Testing)
Thử nghiệm tĩnh	Thử nghiệm các đặc tả kỹ thuật của sản phẩm bởi: 1. tìm kiếm các vấn đề cơ bản cấp	Quy trình xem xét lại cẩn thận phần cứng và mã cho các lỗi mà không thực hiện nó.

	<p>cao, các trường hợp sơ suất, bỏ sót (ví dụ, giả vờ là khách hàng, nghiên cứu các hướng dẫn/tiêu chuẩn hiện hành, xem xét và thử nghiệm phần mềm tương tự, vv.)</p> <p>2. thử nghiệm đặc tả kỹ thuật cấp thấp bởi đảm bảo đầy đủ, chính xác, sự tinh tế, nhất quán, phù hợp, khả thi, v.v.</p>	
Thử nghiệm động	<p>Yêu cầu định nghĩa những gì phần mềm và phần cứng thực hiện, bao gồm:</p> <ul style="list-style-type: none"> <li>* <i>thử nghiệm dữ liệu</i>, đó là việc kiểm tra thông tin các đầu vào và đầu ra người sử dụng</li> <li>* <i>thử nghiệm điều kiện biên</i>, đó là thử nghiệm các trạng thái tại cạnh của các giới hạn hoạt động dự kiến của phần mềm</li> <li>* <i>thử nghiệm đầu vào</i>, đó là thử nghiệm với dữ liệu vô giá trị, không hợp lệ.</li> <li>* <i>thử nghiệm trạng thái</i>, đó là thử nghiệm các phương thức và quá trình chuyển đổi giữa các chế độ phần mềm với các biến trạng thái tức là, các điều kiện chạy đua, thử nghiệm sự lặp lại (lý do chính là để phát hiện rò rỉ bộ nhớ), ứng suất (phần mềm đói = bộ nhớ thấp, cpu chậm = mạng chậm), tải (nguồn cấp dữ liệu phần mềm = kết nối nhiều thiết bị ngoại vi, xử lý một lượng lớn dữ liệu, web server có nhiều khách hàng truy cập vào nó, v.v.), ...</li> </ul>	<p>Thử nghiệm hệ thống đang chạy trong khi theo dõi mã, các sơ đồ nguyên lý, v.v.</p> <p>Trực tiếp thử nghiệm ở mức độ thấp và mức độ cao dựa trên sự hiểu biết hoạt động chi tiết, truy cập vào các biến và các kết xuất bộ nhớ. Tìm kiếm các lỗi tham chiếu dữ liệu, các lỗi khai báo dữ liệu, lỗi tính toán, các lỗi so sánh, lỗi lưu đồ điều khiển, các lỗi tham số cho chương trình con, các lỗi I/O, v.v.</p>

Bảng 4-2: Ma trận mô hình thử nghiệm

Bên trong mỗi mô hình (như trong Hình 4-14), thử nghiệm có thể được tiếp tục chia nhỏ để bao gồm các *thử nghiệm unit/module* (thử nghiệm gia tăng các yếu tố riêng lẻ trong hệ thống), *thử nghiệm tính tương thích* (thử nghiệm rằng các phần tử không gây ra các vấn đề với các phần tử khác trong hệ thống), *thử nghiệm sự tích hợp* (thử nghiệm gia tăng các yếu tố tích hợp), *thử nghiệm hệ thống* (thử nghiệm toàn bộ hệ thống nhúng với tất cả các yếu tố tích hợp), *thử nghiệm hồi quy* (quay lại các thử nghiệm đã được thông qua trước đó sau khi sửa đổi hệ thống), và *thử nghiệm sản xuất* (thử nghiệm để đảm bảo rằng việc sản xuất hệ thống không đưa ra lỗi).



Từ các loại thử nghiệm này, một tập các hiệu quả của các trường hợp thử nghiệm có thể nhận được từ việc kiểm tra rằng một yếu tố và/hoặc hệ thống đáp ứng các đặc tả kỹ thuật kiến trúc, cũng như xác nhận rằng các yếu tố và/hoặc hệ thống đáp ứng các yêu cầu thực tế, có thể hoặc có thể không được phản ánh một cách chính xác hoặc ở tất cả trong tài liệu. Một khi các trường hợp thử nghiệm đã được hoàn thành và các thử nghiệm được chạy, các kết quả được xử lý có thể thay đổi tùy thuộc vào sự tổ chức như thế nào, nhưng thường khác nhau giữa những cái không chính thức, nơi thông tin được trao đổi mà không cần bất kỳ quy trình cụ thể nào theo sau, và sự xem xét lại thiết kế chính thức, hoặc sự xem xét ngang hàng nơi mà các nhà phát triển thành viên trao đổi các yếu tố để thử nghiệm, walkthroughs nơi các kỹ sư chịu trách nhiệm chính thức duyệt các sơ đồ nguyên lý và mã nguồn, kiểm tra nơi mà ai đó khác các kỹ sư chịu trách nhiệm sẽ thực hiện duyệt thiết kế... Các phương pháp thử nghiệm cụ thể và các mẫu cho các trường hợp thử nghiệm, cũng như toàn bộ quá trình thử nghiệm, đã được định nghĩa trong một số các tiêu chuẩn thử nghiệm và đảm bảo chất lượng công nghiệp thông dụng, bao gồm các tiêu chuẩn đảm bảo chất lượng ISO9000, Capability Maturity Model (CMM), và ANSI / IEEE 829.

Cuối cùng, như với gỡ lỗi, có nhiều loại tự động hóa, công cụ thử nghiệm và kỹ thuật mà có thể trợ giúp trong tốc độ, tính hiệu quả, và tính chính xác của việc thử nghiệm các yếu tố khác nhau. Chúng bao gồm các công cụ tải, công cụ ứng suất, máy phun nhiễu, máy phát tiếng ồn, các công cụ phân tích, ghi và phát lại macro, và macro được lập trình, bao gồm các công cụ được liệt kê trong Bảng 4-3.

### ***Câu hỏi ôn tập***

1. Liệt kê các bước trong quá trình thiết kế hệ thống nhúng
2. Mô tả các bước trong quá trình thiết kế hệ thống nhúng
3. Liệt kê các bước trong quá trình cài đặt và thử nghiệm hệ thống nhúng
4. Mô tả các bước trong quá trình cài đặt và thử nghiệm hệ thống nhúng

DRAFT

# CHƯƠNG 5 PHÁT TRIỂN HỆ THỐNG NHÚNG DỰA TRÊN VXL ARM

## 5.1 Giới thiệu chung

Trong các hệ thống nhúng hiện nay, vi xử lý lõi ARM được sử dụng rộng rãi nhất. Trong chương này, các kiến thức căn bản về kiến trúc vi xử lý lõi ARM và tập lệnh ARM được giới thiệu. Sau đó, các kiến thức căn bản về việc thiết kế các thành phần căn bản của hệ thống nhúng được đề cập. Phần cuối sẽ tập trung vào thiết lập hệ điều hành nhúng trên nền ARM.

## 5.2 Kiến trúc của hệ vi xử lý nhúng ARM

### 5.2.1 Lõi ARM

Kiến trúc của ARM được thiết kế chuyên dụng cho các ứng dụng nhúng. Do đó, hiện thực hóa chip ARM được thiết kế để cho các ứng dụng nhỏ nhưng có hiệu năng cao, tiêu thụ ít năng lượng.

Lõi ARM được thiết kế theo kiến trúc RISC, nó chứa các kiến trúc RISC chung

- Các thanh ghi đồng dạng.
- Kiến trúc dạng Load-Store. Các địa chỉ Load/Store chỉ được xác định từ nội dung thanh ghi và các chỉ lệnh.
- Các kiểu đánh địa chỉ đơn giản.
- Các chỉ lệnh có độ dài cố định và đồng dạng, do đó đơn giản hóa việc giải mã các câu lệnh.
- Thay vì chỉ dùng 1 chu kỳ xung nhịp cho tất cả các chỉ lệnh, ARM thiết kế để sao cho tối giản số chu kỳ xung nhịp cho một chỉ lệnh, do đó tăng được sự phức tạp cho các chỉ lệnh đơn lẻ.

Ngoài ra, kiến trúc ARM có thể cung cấp:

- Điều khiển cả khối logic số học (ALU) và bộ dịch chuyển (shifter) trong các lệnh xử lý dữ liệu để tối đa hóa việc sử dụng ALU và bộ dịch chuyển.
- Các chế độ địa chỉ tự tăng hoặc tự giảm để tối ưu hóa các lệnh vòng lặp.
- Các lệnh nhân Load/Store để tối đa dữ liệu truyền qua.

Nhờ các tối ưu trên nền kiến trúc RISC căn bản, lõi ARM có thể đạt được một sự cân bằng giữa hiệu năng cao, kích thước mã nguồn ít, công suất tiêu thụ thấp.

Lỗi ARM có 37 thanh ghi trong đó có 31 thanh ghi đa dụng. Tuy nhiên tại một thời điểm chỉ có 16 thanh ghi đa dụng và 2 thanh ghi trạng thái hiện thị. Các thanh ghi khác ở dạng ẩn, chỉ hiện thị ở một số chế độ hoạt động riêng.

Trong các thanh ghi đã dụng trên, có 3 thanh ghi còn được dùng để các chức năng hoặc nhiệm vụ đặc biệt riêng: r13, r14, r15.

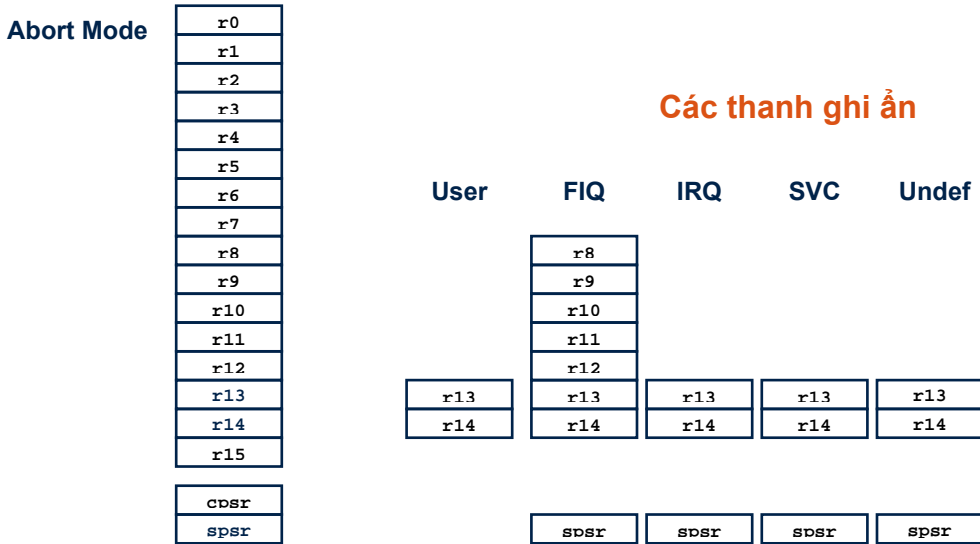
- Hai thanh ghi trạng thái bao gồm thanh ghi trạng thái chương trình hiện tại (cpsr) dùng để giám sát các trạng thái hoạt động hiện tại và thanh ghi trạng thái chương trình lưu (spsr) dùng để lưu trữ giá trị của cpsr khi có một trường hợp ngoại lệ xảy ra.

[illegible]

Các cờ của cpsr như sau:

- 114

## Các thanh ghi hiện



### Hình 5-2 Các thanh ghi của lõi ARM

Chế độ hoạt động của bộ VXL sẽ xác định thanh ghi nào hoạt động và quyền truy cập tới thanh ghi *cpsr*. Mỗi chế độ hoạt động của bộ VXL sẽ là chế độ đặc quyền và không đặc quyền: chế độ đặc quyền cho phép đọc và ghi tới thanh ghi *cpsr*. Người lại chế độ không đặc quyền chỉ cho phép đọc trường điều khiển của *cpsr* nhưng vẫn cho phép đọc và ghi tới các cờ điều kiện.

Có bảy (7) chế độ hoạt động của bộ VXL: sáu chế độ đặc quyền (abort, fast interrupt request, interrupt request, supervisor, system, and undefined) và một chế độ không đặc quyền (user). Sơ đồ các thanh ghi các chế độ như hình dưới.

User	FIQ	IRQ	SVC	Undef	Abort
r0	User mode r0-r7, r15, và cpsr	User mode r0-r12, r15, và cpsr	User mode r0-r12, r15, và cpsr	User mode r0-r12, r15, và cpsr	User mode r0-r12, r15, và cpsr
r1					
r2					
r3					
r4					
r5					
r6					
r7					
r8	r8				
r9	r9				
r10	r10				
r11	r11				
r12	r12				
r13	r13				
r14	r14				
r15					
cpsr					
	spsr	spsr	spsr	spsr	spsr

Hình 5-3 Các chế độ hoạt động và các thanh ghi

Hoạt động của các chế độ như sau:

- Bộ VXL hoạt động ở chế độ Abort khi bộ VXL không thể truy cập bộ nhớ.
- Bộ VXL hoạt động ở chế độ *interrupt request* (IRQ) và *fast interrupt request* (FIQ) tương ứng với hai mức ngắt của chip ARM.
- Bộ VXL hoạt động ở chế độ Supervisor khi sau khi hệ thống khởi động (reset) và khi nhân của hệ điều hành hoạt động.
- Bộ VXL hoạt động ở chế độ System khi hệ thống có thể truy cập và đọc, ghi toàn bộ thanh ghi *cpsr*. Đây là một chế độ đặc biệt của chế độ User.
- Bộ VXL chuyển sang chế độ Undefined khi bộ VXL gặp một lệnh không xác định hoặc không được hỗ trợ.
- Bộ VXL hoạt động ở chế độ User là để chạy các chương trình và các ứng dụng thông thường.

Đối với từng chế độ, có thể có các thanh ghi riêng cho từng chế độ đó.

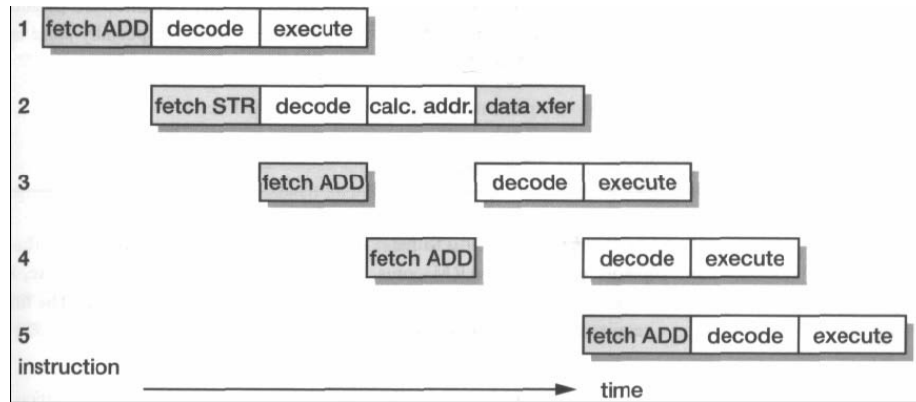
### 5.2.3 Pipeline

Cách tổ chức của nhân ARM không thay đổi nhiều trong khoảng 1983-1995: đến ARM7-sử dụng dòng chảy lệnh sử dụng 3 tác vụ. Từ 1995 trở về sau, đã xuất hiện một vài nhân ARM mới được giới thiệu có dòng chảy lệnh sử dụng 5 tác vụ. Các dòng ARM sau này có thể có dòng chảy lệnh 6 tác vụ (ARM10), 9 tác vụ (ARM11) hoặc 13 tác vụ (ARM Cortex).



### Dòng chảy lệnh 3 tác vụ:

Bao gồm các tác vụ sau: Fetch-decode-Execute (nhận lệnh, giải mã, thực thi)



Hình 5-4: Dòng chảy lệnh 3 tác vụ áp dụng trong trường hợp 1 lệnh có nhiều chu kỳ máy

### Dòng chảy lệnh 5 tác vụ:

Bao gồm các tác vụ : Fetch-decode-execute-buffer/data-write back

### Dòng chảy lệnh 6 tác vụ:

- Fetch và đoán nhánh
- Issue
- Decode và đọc thanh ghi
- Execute shift và ALU, hoặc tính địa chỉ, hoặc nhân
- Truy cập bộ nhớ, hoặc nhân
- Ghi vào thanh ghi

### Dòng chảy lệnh 9 tác vụ:

- Ba tác vụ Fetch.
- Một tác vụ Decode
- Một tác vụ Issue
- Bốn tác vụ integer execution pipeline

## 5.2.4 Cấu trúc bus

Khi muốn thiết kế riêng một vi điều khiển, cũng như thiết kế một hệ thống nhúng, ngoài quan tâm đến các chức năng các khối, bus kết nối các khối lại với nhau cũng là một vấn đề nên được xem xét kỹ lưỡng. Các hệ thống nhúng sử dụng các kỹ thuật bus khác với thiết kế trên các PC dựa trên họ x86. Một dạng bus PC thông dụng nhất đó là bus PCI kết nối các thiết bị như card đồ họa, bộ điều khiển đĩa cứng,... đến bus bộ vi xử lý x86. Đây là loại bus ngoài(off-chip), nghĩa là nó kết nối các thiết bị bên ngoài tới chip. Ngược lại, các thiết bị nhúng sử dụng bus on-chip nằm ở bên trong chip và cho phép các thiết bị ngoại vi được kết nối với lõi ARM.

Có hai loại thiết bị khác nhau gắn với bus : là master và slave. Trong đó lõi ARM là master – có khả năng điều khiển quá trình truyền dữ liệu với thiết bị khác trên cùng bus, và các thiết bị ngoại vi – các slave chỉ có thể đáp ứng với sự điều khiển một thiết bị master.

Kiến trúc bus vi điều khiển tiên tiến gọi tắt là AMBA (Advanced Microcontroller Bus Architecture) được giới thiệu năm 1996 và đã được sử dụng làm kiến trúc bus on-chip dành cho các bộ xử lý ARM. Các bus AMBA đầu tiên được giới thiệu là ARM System Bus (ASB) và ARM Peripheral Bus (APB). Một thiết kế bus khác của ARM được giới thiệu sau đó là ARM High Performance Bus (AHB). Với AMBA, các nhà thiết kế ngoại vi có thể sử dụng lại cùng thiết kế trên nhiều project khác nhau.

Bởi vì có nhiều các ngoại vi được thiết kế với giao diện AMBA nên các nhà thiết kế phần cứng có nhiều lựa chọn đối với các ngoại vi đã được kiểm thử nhằm sử dụng trong thiết kế của họ.

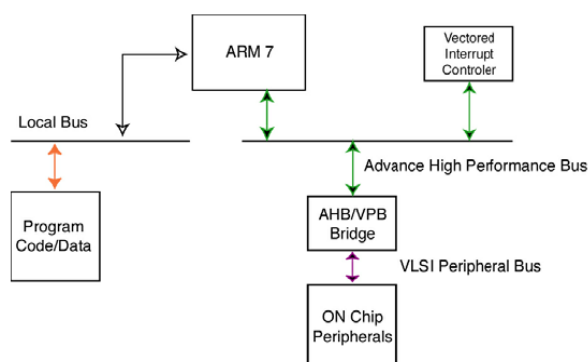
AHB cung cấp băng thông dữ liệu cao hơn so với ASB bởi nó được thiết kế dựa trên lược đồ bus ghép kênh tập trung hơn là thiết kế bus hai chiều ASB. Sự thay đổi này cho phép bus AHB chạy ở tốc độ clock cao hơn và là bus ARM đầu tiên hỗ trợ độ rộng bus lên tới 64 và 128bit. ARM cũng đã giới thiệu 2 biến thể của bus AHB là Multi-layer AHB và AHB-Lite. Ngược lại với bus AHB ban đầu chỉ cho phép một master tích cực trên bus tại mọi thời điểm, Multi-layer AHB cho phép nhiều master cùng tích cực một lúc. AHB-Lite là một tập con của bus AHB, nó bị giới hạn chỉ có một master trên bus. Bus này được phát triển cho các thiết kế không yêu cầu đầy đủ chức năng của bus AHB chuẩn.

Đối với lập trình viên, có thể coi LPC2378 gồm 1 bus duy nhất có độ rộng 32bit tức cho phép hỗ trợ 1 dải bộ nhớ liên tục lên tới 4Gbyte. Tuy nhiên, thực tế vi điều khiển này gồm nhiều bus.

Lõi ARM7 được kết nối đến bus AHB. Như vậy thường các ngoại vi nào có tốc độ hoạt động cao sẽ được kết nối trực tiếp với bus này. Ví dụ điển hình trong LPC2378 là khối điều khiển ngắt và một cầu kết nối đến 1 bus khác là VPB – VLSI peripheral bus. Tất cả các ngoại vi người sử dụng được kết nối đến VPB. Cầu VPB gồm 1 bộ chia tần cho phép bus VPB có thể hoạt động ở tốc độ thấp hơn tốc độ lõi ARM7 và bus AHB. Kỹ thuật này đem lại 2 lợi ích : thứ nhất, giúp tiết kiệm công suất tiêu thụ; thứ hai, cho phép tùy chọn tích hợp thêm các ngoại vi tốc độ thấp hơn lõi ARM7 mà không gây thất nút cổ chai trên bus AHB.

Chú ý, sau khi reset, bộ chia tần VPB được thiết lập hệ số chia là 4 nên các thiết bị ngoại vi on-chip sẽ chạy ở tốc độ bằng  $\frac{1}{4}$  tần số của CPU.

Một bus thứ 3 nữa trong vi điều khiển LPC2378 dùng để kết nối bộ nhớ Flash và RAM on-chip tới lõi ARM7. Mặc dù có thể sử dụng bus AHB để kết nối bộ nhớ chương trình và dữ liệu tới lõi ARM7, tuy nhiên do nhiều ngoại vi khác cũng sử dụng bus này nên sẽ gây ra vấn đề cạnh tranh quyền sử dụng bus. Bus thứ 3 này giúp giải quyết vấn đề này và cải thiện đáng kể hiệu năng hoạt động của lõi ARM7.

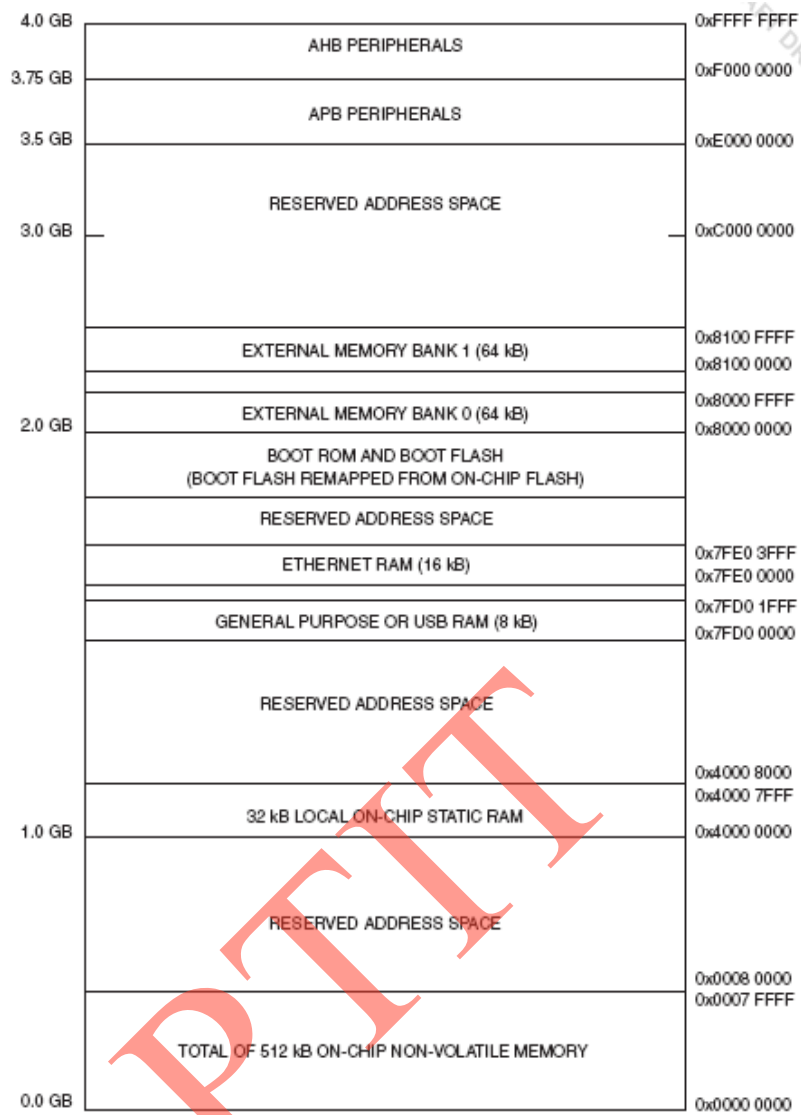


Hình 5-5 . Cấu trúc bus LPC2378

### 5.2.5 Bản đồ bộ nhớ:

Dữ liệu và chương trình được lưu trữ trong bộ nhớ. Bộ nhớ này có thể tích hợp trên cùng vi xử lý hoặc có thể là một chip tách biệt. Bộ nhớ được đặt trong không gian nhớ của vi xử lý và bộ xử lý giao tiếp với bộ nhớ thông qua tập các bus dữ liệu và bus địa chỉ. Để đọc(ghi) vào một vị trí cụ thể trong bộ nhớ, bộ xử lý đầu tiên phải ghi địa chỉ của vị trí cần đọc(ghi) lên bus địa chỉ. Một mạch logic(hoặc tích hợp trên cùng vi xử lý hoặc là một mạch ngoài) được gọi là bộ giải mã địa chỉ sẽ biên dịch các bit địa chỉ này trên bus địa chỉ và chọn bộ nhớ hoặc ngoại vi phù hợp. Dữ liệu sau đó sẽ được truyền trên bus dữ liệu.

Thêm nữa, cũng có các tín hiệu cho việc đọc và ghi đến nhiều thiết bị trong không gian bộ nhớ của bộ vi xử lý mà thường được gọi là bus địa chỉ. Những tín hiệu bus điều khiển này gồm tín hiệu đọc, tín hiệu ghi, tín hiệu chọn chip (chip-select hoặc chip-enable). Tín hiệu đọc và ghi thường kết hợp cùng nhau thành tín hiệu đọc/ghi(read/write). Tín hiệu chọn chip sẽ được thiết lập đến mức tích cực của nó khi địa chỉ trên bus địa chỉ rơi vào dải địa chỉ của một thiết bị nào đó. Ví dụ giả sử có một bộ nhớ RAM chiếm dải địa chỉ từ 0x0000 đến 0x0FFF, khi lệnh phần mềm truy cập giá trị ở địa chỉ 0x02F2, tín hiệu chọn chip cho RAM sẽ được tích cực.



Hình 5-6 . Bản đồ bộ nhớ của LPC2378

APB Peripheral	Base Address	Peripheral Name
0	0xE000 0000	Watchdog Timer
1	0xE000 4000	Timer 0
2	0xE000 8000	Timer 1
3	0xE000 C000	UART0
4	0xE001 0000	UART1
5	0xE001 4000	Not used
6	0xE001 8000	PWM1
7	0xE001 C000	I <sup>2</sup> C0
8	0xE002 0000	SPI
9	0xE002 4000	RTC
10	0xE002 8000	GPIO
11	0xE002 C000	Pin Connect Block
12	0xE003 0000	SSP1
13	0xE003 4000	ADC
14	0xE003 8000	CAN Acceptance Filter RAM <sup>[1]</sup>
15	0xE003 C000	CAN Acceptance Filter Registers <sup>[1]</sup>
16	0xE004 0000	CAN Common Registers <sup>[1]</sup>
17	0xE004 4000	CAN Controller 1 <sup>[1]</sup>
18	0xE004 8000	CAN Controller 2 <sup>[1]</sup>
19 to 22	0xE004 C000 to 0xE005 8000	Not used
23	0xE005 C000	I <sup>2</sup> C1
24	0xE006 0000	Not used
25	0xE006 4000	Not used
26	0xE006 8000	SSP0
27	0xE006 C000	DAC
28	0xE007 0000	Timer 2
29	0xE007 4000	Timer 3
30	0xE007 8000	UART2
31	0xE007 C000	UART3
32	0xE008 0000	I <sup>2</sup> C2
33	0xE008 4000	Battery RAM
34	0xE008 8000	I <sup>2</sup> S
35	0xE008 C000	SD/MMC Card Interface <sup>[2]</sup>
36 to 126	0xE009 0000 to 0xE01F BFFF	Not used
127	0xE01F C000	System Control Block

◀Bảng5-1 : Địa chỉ của các ngoại vi kết nối với VPB

Khi tìm hiểu về một vi điều khiển hay vi xử lý, nên thiết lập một bảng biểu diễn tên và dải địa chỉ của mỗi thiết bị nhớ và ngoại vi đặt trong không gian bộ nhớ. Bảng này được gọi là bản đồ bộ nhớ - memory map. Thường bản này được cung cấp bởi nhà sản xuất trong tài liệu kỹ thuật đi kèm với vi điều khiển/vi xử lý.

Flash on-chip đặt ở địa chỉ 0x00000000 và RAM đặt ở 0x40000000. Các vi điều khiển LPC2xxx được lập trình trước trong quá trình sản xuất bởi một bootloader FLASH

và một chương trình giám sát thời gian thực của ARM, chúng được đặt dưới địa chỉ 0x80000000. Khoảng địa chỉ từ 0x80000000 đến 0xE0000000 dành cho bộ nhớ ngoài. Các ngoại vi người sử dụng kết nối với VPB được ánh xạ vào vùng 0xE0000000 và 0xF0000000 như trong bảng 1.

### 5.2.6 Tập lệnh ARM

VXL ARM sử dụng cấu trúc load-store. Điều đó có nghĩa là: tất cả các lệnh đều được thực hiện trên thanh ghi. Các lệnh ARM thường có 2 đến 3 toán tử.

Mặc dù các phiên bản kiến trúc ARM khác nhau hỗ trợ các tập lệnh khác nhau, các phiên bản mới thường tương thích ngược với các tập lệnh cũ.

#### Danh sách các lệnh của ARM

##### Các lệnh xử lý dữ liệu

Các lệnh xử lý dữ liệu thực thi các phép tính đối với dữ liệu trên các thanh ghi. Các lệnh đó bao gồm chuyển dữ liệu, các phép tính số học, logic, các phép so sánh và phép nhân.

Nếu các lệnh này có thêm S ở cuối, nó sẽ cập nhật cờ trên thanh ghi CPRS. Các lệnh dịch chuyển và phép toán logic cập nhật cờ Carry C, cờ Negative N, và cờ Zero Z.

##### Các lệnh dịch chuyển

MOVE là lệnh đơn giản nhất trong các lệnh của ARM. Lệnh thay copy giá trị  $N$  đến thanh ghi đích  $Rd$ .

Cú pháp : <instruction>{<cond>} {S}  $Rd, N$

Có 2 lệnh dịch chuyển

MOV: Chuyển một giá trị 32 bit đến thanh ghi ( $Rd=N$ )

MVN: Chuyển giá trị đảo 32 bit đến thanh ghi ( $Rd= \sim N$ )

Ví dụ:

Trước:	$r5=5$ $r7=8$
	MOV $r7, r5$ ; let $r7 = r5$
Sau	$r5=5$ $r7=5$

MOVCS  $R0, R1$  ; carry SET thì  $R0:=R1$

MOVS  $R0, \#0$  ;  $R0:=0$   $Z=1, N=0$   
;C, V không thay đổi

##### Các lệnh số học

Các lệnh số học thực hiện cộng và trừ các giá trị 32 bit có dấu và không có dấu. Kết quả là 32 bit và được đặt trong một thanh ghi. Cấu trúc của lệnh có 3 địa chỉ.

Cú pháp : <instruction>{<cond>} {S}  $Rd, Rn, N$



ADD R0, R1, R2	; R0 = R1+R2
ADC R0, R1, R2	; R0 = R1+R2+C
SUB R0, R1, R2	; R0 = R1-R2
SBC R0, R1, R2	; R0 = R1-R2+C-1
RSB R0, R1, R2	; R0 = R2-R1
RSC R0, R1, R2	; R0 = R2-R1+C-1

Ví dụ:

Trước:

```
r0 = 0x00000000
r1 = 0x00000002
r2 = 0x00000001
SUB r0, r1, r2
```

Sau:

```
r0 = 0x00000001
```

Trước:

```
r0 = 0x00000000
r1 = 0x00000077
RSB r0, r1, #0 ; Rd = 0x0 - r1 (giá trị âm của r1)
```

Sau:

```
r0 = -r1 = 0xfffff8
```

### Các lệnh logic

Các lệnh logic thực hiện các phép toán logic theo bit trên các thanh ghi

Cú pháp: <instruction> {<cond>} {S} Rd, Rn, N

AND R0, R1, R2	; R0 = R1 and R2
ORR R0, R1, R2	; R0 = R1 or R2
EOR R0, R1, R2	; R0 = R1 xor R2
BIC R0, R1, R2	; R0 = R1 and (~R2)

Ví dụ:

Trước;

```
r0 = 0x00000000
r1 = 0x02040608
r2 = 0x1030507056
ORR r0, r1, r2
```

Sau:

```
r0 = 0x12345678
```

Trước:

```
r1 = 0b1111
r2 = 0b0101
BIC r0, r1, r2
```

Sau:

r0 = 0b1010

Trước:

r1=0x11111111

r2=0x01100101

BIC r0, r1, r2

Sau:

R0=0x10011010

### Các lệnh so sánh

Các lệnh so sánh dùng để so sánh hoặc kiểm tra một thanh ghi với một giá trị 32 bit. Các lệnh này không ảnh hưởng đến các thanh ghi, chỉ thay đổi cập nhật các bit cờ trên thanh ghi CPRS. Các lệnh này không cần thêm S

Cấu trúc: <instruction> {<cond>} Rn, N

CMP R1, R2 ; set cờ cho kết quả R1-R2  
CMN R1, R2 ; set cờ cho kết quả R1+R2  
TST R1, R2 ; set cờ cho kết quả R1 and R2  
TEQ R1, R2 ; set cờ cho kết quả R1 xor R2

### Các lệnh nhân

Các lệnh nhân thực hiện phép nhân giá trị trên hai thanh ghi và có thể thực hiện cộng dồn với một thanh ghi khác. Kết quả cuối cùng được ghi vào thanh ghi đích hoặc hai thanh ghi nếu kết quả là 64 bits.

Cấu trúc:

MLA {<cond>} {S} Rd, Rm, Rs, Rn

MUL {<cond>} {S} Rd, Rm, Rs

MUL R0, R1, R2 ; R0 = (R1xR2)[31:0] (32 bits)

MLA R4, R3, R2, R1 ; R4 = R3xR2+R1

### Các lệnh rẽ nhánh

Các lệnh rẽ nhánh thay đổi chu trình chạy của chương trình, hoặc dùng để gọi một hàm con khác. Việc này làm thay đổi giá trị thanh ghi chương trình PC làm thanh ghi PC trở đến một địa chỉ mới

Cấu trúc:

B {<cond>} label

BL {<cond>} label

BX {<cond>} Rm

BLX {<cond>} label | Rm

B label ;chuyển đến địa chỉ label, pc=label

BL label: rẽ nhánh có liên kết, pc=label, địa chỉ quay lại được lưu vào thanh ghi lr (R14)  
 BX Rm; rẽ nhánh trao đổi, pc = Rm & 0xffffffe, T = Rm & 1 (T là Thumb bit trên CPSR)  
 BLX

Ví dụ:

```

B label
...

label: ... ; chương trình sẽ chuyển đến địa chỉ label

MOV R0, #0
loop: ...
      ADD R0, R0, #1
      CMP R0, #10
      BNE loop ; vòng lặp lại loop nếu R0 khác 10
  
```

Ví dụ:

```

BL sub ; gọi sub
CMP R1, #5 ; quay lại địa chỉ này
MOVEQ R1, #0
...
sub: ... ;
     ...
     MOV PC, LR ; quay lại
  
```

### Các lệnh chuyển dữ liệu Load-Store

Các lệnh chuyển dữ liệu load-store chuyển dữ liệu giữa bộ nhớ và các thanh ghi trên CPU. Có ba loại lệnh load-store : chỉ dùng một thanh ghi, dùng nhiều thanh ghi hoặc trao đổi giữa thanh ghi và bộ nhớ

#### Load-store dùng một thanh ghi

Các lệnh load-store dùng một thanh ghi dùng để chuyển dữ liệu vào và ra khỏi thanh ghi. Các lệnh này hỗ trợ các kiểu dữ liệu có dấu và không có dấu, kích cỡ có thể là 32 bit, 16 bit hoặc 8 bit (byte)

Cấu trúc:

```

<LDR|STR>{<cond>} {B} Rd, addressing
LDR{<cond>} SB|H|SH Rd, addressing
STR{<cond>} H Rd, addressing
  
```

```

LDR R0, [R1] ; R0 := mem32[R1]
STR R0, [R1] ; mem32[R1] := R0
  
```

LDR, LDRH, LDRB lần lượt cho 32, 16, 8 bit  
 STR, STRH, STRB lần lượt cho 32, 16, 8 bit

Ví dụ:

; Nạp thanh ghi r0 với nội dung ở địa chỉ do r1 trỏ đến

LDR r0, [r1] ; = LDR r0, [r1, #0]  
 ; Lưu nội dung của thanh ghi r0 vào địa chỉ do r1 trỏ đến  
 STR r0, [r1] ; = STR r0, [r1, #0]

### Các chế độ đánh địa chỉ

VXL ARM hỗ trợ ba chế độ đánh địa chỉ:

- Pre-indexed addressing (**LDR R0, [R1, #4]**) without a writeback

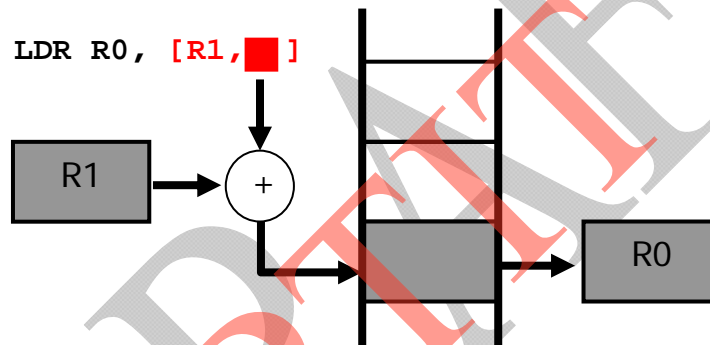
Ví dụ:

Trước

r0 = 0x00000000  
 r1 = 0x00090000  
 mem32[0x00090000] = 0x01010101  
 mem32[0x00090004] = 0x02020202

LDR R0, [R1, #4] ; R0=mem[R1+4]  
 ; R1 unchanged

r0 = 0x02020202  
 r1 = 0x00090000



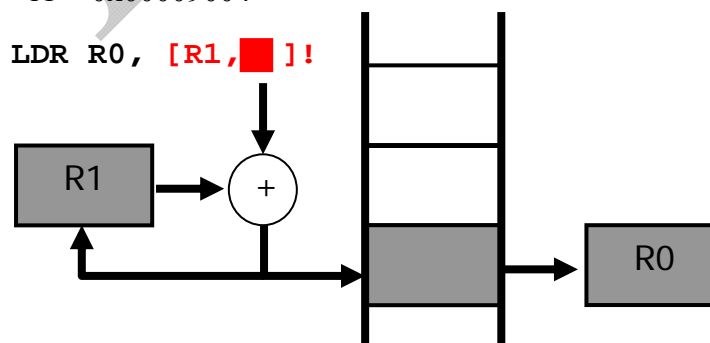
Hình 5-7. Sơ đồ hoạt động lệnh LDR (Pre-indexed addressing)

- Auto-indexing addressing (**LDR R0, [R1, #4]!**) calculation before accessing with a writeback

Ví dụ:

LDR R0, [R1, #4]! ; R0=mem[R1+4]  
 ; R1=R1+4

r0 = 0x02020202  
 r1 = 0x00090004



Hình 5-8. Sơ đồ hoạt động lệnh LDR (Auto-indexing addressing)

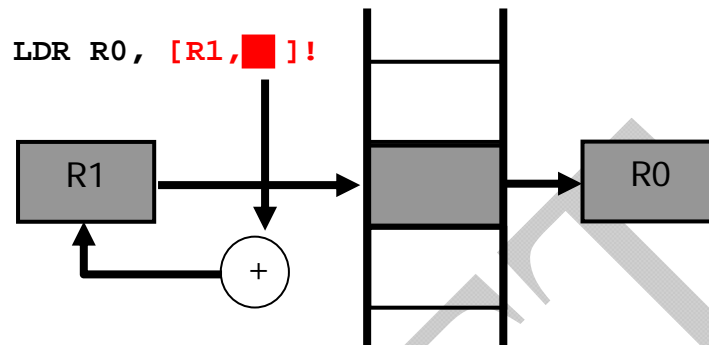
- Post-indexed addressing (**LDR R0, [R1], #4**) calculation after accessing with a writeback

Ví dụ :

LDR R0, R1, #4 ; R0=mem[R1]  
; R1=R1+4

r0 = 0x01010101

r1 = 0x00009004



Hình 5-9. Sơ đồ hoạt động lệnh LDR (Post-indexing addressing)

### Load-store dùng nhiều thanh ghi

Khi cần chuyển dữ liệu giữa nhiều thanh ghi và bộ nhớ hoặc CPU, các lệnh load-store cho nhiều thanh ghi có thể được dùng. Quá trình chuyển dữ liệu từ địa chỉ gốc do thanh ghi Rn chỉ đến. Việc sử dụng các lệnh này sẽ có hiệu quả hơn khi cần chuyển một khối dữ liệu giữa bộ nhớ và thanh ghi hoặc giữa các vị trí trong bộ nhớ

Cấu trúc:

<LDM|STM> {<cond>} <addressing mode> Rn{!}, <registers> {^}

LDM Nạp nhiều thanh ghi

STM Lưu nhiều thanh ghi

Chế độ đánh địa chỉ:

Đuôi ý nghĩa

IA tăng địa chỉ sau khi thực hiện

IB tăng địa chỉ trước khi thực hiện

DA giảm địa chỉ sau khi thực hiện

DB giảm địa chỉ trước khi thực hiện

IA	increment after	$Rn$	$Rn + 4 * N - 4$	$Rn + 4 * N$
IB	increment before	$Rn + 4$	$Rn + 4 * N$	$Rn + 4 * N$
DA	decrement after	$Rn - 4 * N + 4$	$Rn$	$Rn - 4 * N$
DB	decrement before	$Rn - 4 * N$	$Rn - 4$	$Rn - 4 * N$

Ví dụ :

Trước:

mem32[0x80018] = 0x03

mem32[0x80014] = 0x02

mem32[0x80010] = 0x01

r0 = 0x00080010

r1 = 0x00000000

r2 = 0x00000000

r3 = 0x00000000

Address pointer	Memory		
	address	Data	
	0x80020	0x00000005	
	0x8001c	0x00000004	
	0x80018	0x00000003	r3 = 0x00000000
	0x80014	0x00000002	r2 = 0x00000000
r0 = 0x80010 →	0x80010	0x00000001	r1 = 0x00000000
	0x8000c	0x00000000	

LDMIA r0!, {r1-r3}

Sau:

r0 = 0x0008001c

r1 = 0x00000001

r2 = 0x00000002

r3 = 0x00000003

Address pointer	Memory		
	address	Data	
	0x80020	0x00000005	
r0 = 0x8001c →	0x8001c	0x00000004	
	0x80018	0x00000003	r3 = 0x00000003
	0x80014	0x00000002	r2 = 0x00000002
	0x80010	0x00000001	r1 = 0x00000001
	0x8000c	0x00000000	

Nếu dùng lệnh LDMIB



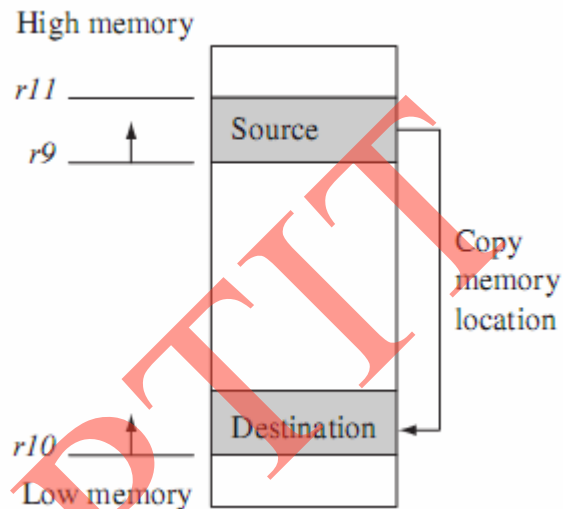
Address pointer	Memory address	Data	
	0x80020	0x00000005	
$r0 = 0x8001c \rightarrow$	0x8001c	0x00000004	
	0x80018	0x00000003	$r3 = 0x00000003$
	0x80014	0x00000002	$r2 = 0x00000002$
	0x80010	0x00000001	$r1 = 0x00000001$
	0x8000c	0x00000000	

Ví dụ : Ứng dụng các lệnh trên để copy một khối dữ liệu

R9: địa chỉ nguồn

R10: địa chỉ đích

R11: địa chỉ cuối của nguồn



```

loop: LDMIA R9!, {R0-R7}; nạp 32 byte từ nguồn và cập nhật con trỏ R9
      STMIA R10!, {R0-R7}; lưu 32 byte từ nguồn và cập nhật con trỏ R10
      CMP R9, R11; so sánh địa chỉ copy và địa chỉ cuối cùng
      BNE loop
  
```

Các lệnh này có thể ứng dụng trong stack

### Trao đổi dữ liệu của thanh ghi với bộ nhớ (swap)

Lệnh swap đổi chỗ nội dung của bộ nhớ và thanh ghi.

Cấu trúc:

SWP {B} {<cond>} Rd,Rm,[Rn]

SWP đổi nội dung 32 bits giữa thanh ghi và bộ nhớ

tmp = mem32[Rn]

mem32[Rn] = Rm

Rd = tmp

SWPB đổi nội dung 8 bits giữa thanh ghi và bộ nhớ

```

tmp = mem8[Rn]
mem8[Rn] = Rm
Rd = tmp

```

Ví dụ

Trước:

```

mem32[0x9000] = 0x12345678
r0 = 0x00000000
r1 = 0x11112222
r2 = 0x00009000
SWP r0, r1, [r2]

```

Sau:

```

mem32[0x9000] = 0x11112222
r0 = 0x12345678
r1 = 0x11112222
r2 = 0x00009000

```

## 5.2.7 Tập lệnh Thumb

Một tính năng quan trọng của ARM là hỗ trợ tập lệnh Thumb. Tập lệnh Thumb bao gồm các lệnh 16 bit. Do đó chúng chiếm ít bộ nhớ hơn và có hiệu năng cao hơn đối với các dữ liệu 16 bit. Đối với các ứng dụng nhúng cần tối ưu bộ nhớ, mật độ lệnh rất quan trọng.

## 5.3 Thiết kế các thành phần cơ bản của bộ xử lý nhúng

### 5.3.1 Lập trình các thanh ghi

Đối với mỗi ngoại vi sẽ có một bộ điều khiển, bộ điều khiển này giao diện với chương trình phần mềm qua 1 hoặc nhiều thanh ghi. Bộ xử lý giao tiếp với các bộ điều khiển bằng việc đọc và ghi các bit lên các thanh ghi này thông qua các lệnh truyền 1 byte hay 1 từ tới 1 địa chỉ cổng I/O. Các lệnh I/O này sẽ kích khởi các đường bus để chọn đúng thiết bị và chuyển các bit tới hoặc đọc các bit từ một thanh ghi thiết bị. Các thanh ghi này được ánh xạ vào không gian địa chỉ của bộ vi xử lý. Kỹ thuật này được gọi là kỹ thuật I/O ánh xạ bộ nhớ - **memory-mapped I/O**.

Một thiết bị điển hình gồm 4 thanh ghi được gọi là thanh ghi trạng thái, thanh ghi điều khiển, thanh ghi dữ liệu vào (**data-in**), thanh ghi dữ liệu ra (**data-out**):

- **Thanh ghi dữ liệu vào – data-in register** : được đọc bởi CPU để đọc đầu vào
- **Thanh ghi dữ liệu ra – data-out register** : được ghi bởi CPU để gửi dữ liệu tới thiết bị
- **Thanh ghi trạng thái – status register** : chứa các bit có thể được đọc bởi CPU. Những bit này chỉ trạng thái của thiết bị như yêu cầu hiện thời đã được thực thi xong chưa, một byte đã sẵn để đọc trong thanh ghi data-in chưa, có lỗi xảy ra không,...
- **Thanh ghi điều khiển – control register** : có thể được ghi bởi CPU để bắt đầu 1 yêu cầu hoặc để thay đổi mode hoạt động của thiết bị.

Các thanh ghi dữ liệu điển hình có kích thước 1 đến 4 byte. Một vài bộ điều khiển có các chip FIFO – một bộ nhớ sử dụng nguyên lý first in first out – để chứa nhiều byte dữ

liệu đầu vào hoặc ra nhằm tăng khả năng của bộ điều khiển so với kích thước của thanh ghi dữ liệu. Một chip FIFO có thể chứa 1 lượng dữ liệu cho đến khi thiết bị hoặc CPU có thể nhận những dữ liệu này, hay có thể gọi đây là bộ đệm.

Thường để thuận tiện sử dụng các thanh ghi này, chúng ta sử dụng khai báo các định danh đến các thanh ghi trong một file header giúp dễ dàng nhớ tên các định danh thay vì phải nhớ địa chỉ của các thanh ghi. Ví dụ khai báo các thanh ghi cho LPC2378, trong file lpc23xx.h đi kèm bộ công cụ KeilC.

```
/* Vectored Interrupt Controller (VIC) */
#define VIC_BASE_ADDR 0xFFFFF000
#define VICIRQStatus (*(volatile unsigned long *) (VIC_BASE_ADDR + 0x000))
.....
/* Pin Connect Block */
#define PINSEL_BASE_ADDR 0xE002C000
#define PINSEL0 (*(volatile unsigned long *) (PINSEL_BASE_ADDR + 0x00))
.....
/* General Purpose Input/Output (GPIO) */
#define GPIO_BASE_ADDR 0xE0028000
#define IOPIN0 (*(volatile unsigned long *) (GPIO_BASE_ADDR + 0x00))
.....
/* Memory Accelerator Module (MAM) */
#define MAMCR (*(volatile unsigned long *) (SCB_BASE_ADDR + 0x000))
.....
```

### 5.3.2 Thiết kế điều khiển I/O

Vi điều khiển LPC2300 có lên tới 5 port vào ra đa dụng gọi tắt là GPIO – General Purpose IO, mỗi port gồm 32 đường IO được ánh xạ vào 32 bit của các thanh ghi điều khiển, trạng thái.

Đề tương thích với các các dòng LPC21xx, PORT0 và PORT1 có thể được điều khiển bởi tập các thanh ghi điều khiển kế thừa từ các dòng cũ được giao tiếp thông qua bus APB, tuy nhiên, việc điều khiển thông qua các thanh ghi này có tốc độ khá chậm. Để cải thiện vấn đề này trong LPC2300 có thêm 1 tập các thanh ghi điều khiển cho các port GPIO đặt ở bus cục bộ được gọi là các thanh ghi Fast GPIO để có được tốc độ điều khiển nhanh hơn.

Giống như các loại vi điều khiển khác, mỗi pin của LPC2378 bao gồm nhiều chức năng cụ thể là mỗi chân có từ 1 đến 4 chức năng. Việc chọn chức năng của mỗi pin thông qua các thanh ghi PINSELn mà trong datasheet của nhà sản xuất sẽ nói rõ chi tiết. Khi reset, mặc định các pin là các GPIO.

Về cơ bản mỗi port GPIO được điều khiển thông qua tập gồm 4 thanh ghi với các chức năng:

- IOPIN : thanh ghi giá trị các pin của Port. Trạng thái hiện tại của các pin của Port có thể được đọc từ thanh ghi này. Giá trị của pin tương ứng được biểu diễn qua bit tương ứng của thanh ghi. Bit thứ i sẽ mô tả trạng thái của pin thứ i của port

- IOSET: Thanh ghi set đầu ra Port. Thanh ghi này cùng với thành ghi IOCLR được dùng để điều khiển trạng thái đầu ra của các pin của port. Viết 1 vào một bit nào đó thì tương ứng sẽ tạo ra mức cao ở pin tương ứng với bit đó. Viết 0 vào sẽ không tác động gì

Ví dụ:

$\text{IOSET} = 1 \gg 10$  ; // **Đưa pin 10 lên mức cao**

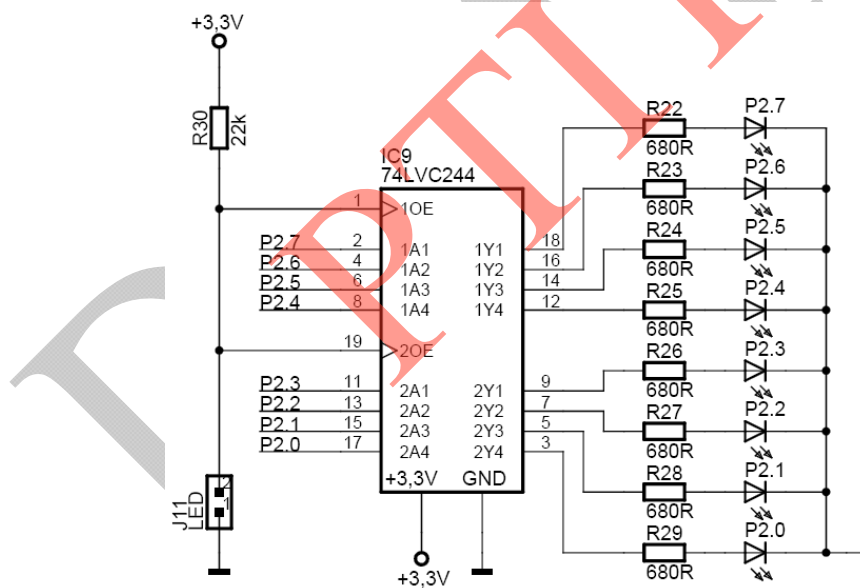
- IOCLR : tương tự như thanh ghi set, tuy nhiên khi viết 1 bit nào đó tương ứng với đưa pin đó về mức thấp.

Ví dụ :  $\text{IOCLR} = (1 \ll 1) | (1 \ll 2)$ ; // **Đưa pin 1 và pin 2 về mức thấp**

- IODIR : thanh ghi chọn hướng của pin là input hay output. Nếu bit tương ứng với pin trong thanh ghi bằng 1 thì pin đó là pin output còn bit đó bằng 0 thì pin tương ứng là pin input.

Đây chính là tập thanh ghi điều khiển với thời gian đáp ứng chậm. Như đã giới thiệu LPC2378, các port còn có thể được điều khiển với đáp ứng thời gian nhanh hơn thông qua các thanh ghi fast với chức năng tương tự là FIODIR, FIOSET, FIOCLR, FIOPIN.

Chú ý khi làm việc với các thanh ghi fast IO, nó còn có thêm một thanh ghi khác là FIOMASK, việc thiết lập bit tương ứng là 0 của mỗi pin sẽ cho phép việc set, clear, đọc trạng thái tương ứng của pin đó qua các thanh ghi FIOSET, FIOCLR, FIOPIN, khi giá trị bit nào đó là 1 thì pin tương ứng với chân đó sẽ không đáp ứng lại với sự thay đổi do tác động của các thanh ghi FIOSET, FIOCLR, FIOPIN.



Hình 5-10. Sơ đồ kết nối các chân IO với các led trên board MCB2300

Ví dụ 1 : Điều khiển nháy led

Sau đây là chương trình điều khiển tắt mở các led kết nối với các Pin 0 đến 7 của port 2 gồm 1 viết bằng assembly và 1 viết bằng C

Chương trình assembly:

```
AREA RESET, CODE, READONLY
ARM
```

## ENTRY

```

start      ;bat dau chuong trinh
           ;khoi tao cac chan io tu p2.0-->p2.7 lam output

           ldr r1,PINSEL10      ;tat chuc nang ETM tren port 2
           mov r0,#0
           str r0,[r1]

           ldr r0,FIO2DIR      ;chon chan 2.0-->2.7 lam chan output
           mov r3,#0xff
           str r3,[r0]

           ldr r0,FIO2MASK      ;cho phép điều khiển các pin
           mov r3,#0x00
           str r3,[r0]

           mov r3,#0xff
           ; vong lap chinh,nhay cac led

led_blinky
           ldr r0,FIO2SET      ;bat led
           str r3,[r0]
           bl delay            ;delay trong khoang 1/2 sec

           ldr r0,FIO2CLR      ;tat led
           str r3,[r0]
           bl delay            ;delay trong khoang 1/2 sec
           b led_blinky

delay      ;ham delay
           ldr r0,DELAY_VAL    ;khoi tao gia tri dem

loop_delay
           subs r0,r0,#0x01
           bne loop_delay

           bx lr              ;tro ve chuong trinh chinh

```

; khai bao cac thanh ghi dac biet cho viec dieu khien cac io

```

FIO_BASE_ADDR    EQU    0x3FFFC000
PINSEL10         DCD    0xe002c028
FIO2DIR          DCD    (FIO_BASE_ADDR + 0x40)
FIO2MASK         DCD    (FIO_BASE_ADDR + 0x50)
FIO2PIN          DCD    (FIO_BASE_ADDR + 0x54)
FIO2SET          DCD    (FIO_BASE_ADDR + 0x58)
FIO2CLR          DCD    (FIO_BASE_ADDR + 0x5C)
DELAY_VAL        DCD    50000

```

END

Chương trình viết bằng C:

```
#include <LPC23xx.H>
//ham tao tre
void delay(unsigned long n)
{
    while(n--) {}
}

int main()
{
    //khởi tạo các pin io từ P2.0→P2.7 là output
    PINSEL10 = 0x00;
    FIO2DIR = 0xff;
    FIO2MASK = 0x00;
    FIO2CLR = 0xff;

    while(1)
    {
        //Set các pin lên mức cao
        FIO2SET=0xff;
        delay(500000);

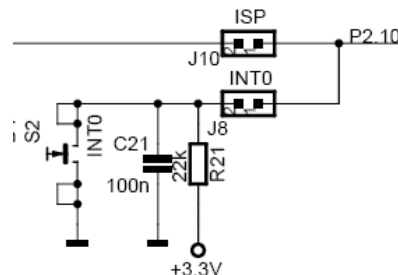
        //Clear các pin về mức thấp
        FIO2CLR =0xff;
        delay(500000);
    }
    return 0;
}
```

**Ví dụ 2: Đọc phím bấm:**

Trên board MCB2300, phím S2 được kết nối tới P2.10.

Về hoạt động:

- Ở trạng thái bình thường: P2.10 ở mức cao qua điện trở kéo lên 3.3V
- Khi phím S2 được nhấn: P2.10 ở trạng thái thấp



Hình 5-11. Sơ đồ kết nối của button S2

Chương trình của chúng ta sẽ thực hiện đọc phím, khi phím được nhấn nó sẽ thể hiện thông qua việc sáng Led trong 1 khoảng thời gian



### Chương trình:

```
#include <LPC23xx.H>

void delay()
{
    int i;
    for (i=0;i<=4000000;i++) {}
}

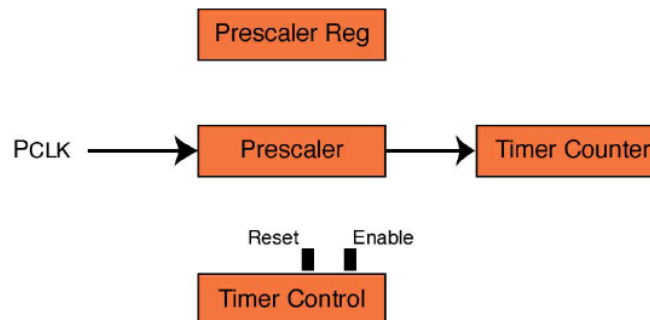
int main()
{
    unsigned int status=0;
    //inits Port 2
    PINSEL10 = 0;
    FIO2MASK = 0;
    FIO2DIR = 0xff; //pin 0..7 are output pins
                  //pin 10 is input pin

    //clear pins 0..7 to turn off all leds
    FIO2CLR = 0xff;

    while(1)
    {
        //checks button status
        if ((FIO2PIN & (1<<10)) == 0)
        {
            FIO2CLR = 0xff;
            FIO2SET = (1<<status);
            status++;
            status %= 8;
            delay();
        }
    }
}
```

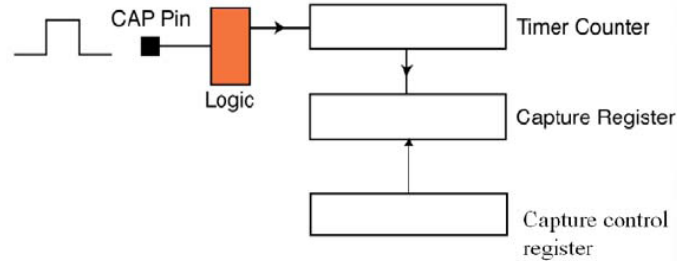
### 5.3.3 Lập trình điều khiển bộ định thời (Timer) và bộ đếm (Counter)

Timer/Counter là các thiết bị ngoại vi được thiết kế để đếm số chu kỳ của 1 clock bên trong hệ thống hoặc từ một nguồn clock ngoài. Trong LPC2378 có bốn bộ Timer/Counter. Tất cả đều giống nhau về cấu trúc và cách sử dụng. Các timer đều hoạt động dựa trên các bộ đếm timer 32 bit. Nguồn clock bên trong cho tất cả bộ định thời là clock ngoại vi PCLK được lấy từ clock dành cho lõi CCLK



Hình 5-12. Mô hình bộ timer

Tốc độ tick của bộ timer được điều khiển bởi giá trị được lưu trong thanh ghi Prescaler Reg. Giá trị bộ đếm Prescaler sẽ tăng theo mỗi xung của PCLK cho đến khi nó đạt đến giá trị trong Prescaler Reg. Khi nó đạt tới giá trị này thì thanh ghi Timer Counter được tăng lên 1 và thanh ghi Prescaler reset về 0 và bắt đầu đếm tiếp. Thanh ghi điều khiển Timer Control chỉ gồm 2 bit mà được sử dụng để cho phép/cấm bộ timer và reset bộ đếm của nó.



Hình 5-13 Sơ đồ hoạt động các thanh ghi Timer

Ngoài chức năng cơ bản của một bộ timer, mỗi timer còn có thêm 4 kênh capture. Các kênh capture cho phép bạn chụp lại giá trị của bộ timer counter khi có một sự chuyển mức của tín hiệu ở 1 kênh đầu vào.

Mỗi một kênh capture có một pin tương ứng được cho phép thông qua khối kết nối pin. Thanh ghi **capture control register** có thể cấu hình sườn nào đó kích hoạt 1 sự kiện capture hoặc cả 2 sườn. Khi sự kiện capture xảy ra, giá trị trong timer counter sẽ được ghi vào thanh ghi capture tương ứng và nếu cần thiết, một ngắt có thể được tạo ra.

Mô tả các pin dành cho bộ Timer/Counter:

Pin	Type	Mô tả
Cap0[1:0] Cap1[1:0] Cap2[1:0] Cap3[1:0]	Input	Capture Signals – 1 sự chuyển mức trên 1 pin capture có thể được cấu hình để copy vào thanh ghi Capture tương ứng giá trị hiện thời của bộ đếm thời gian TC (Timer Counter) và tùy chọn có thể tạo ra các ngắt
MAT0[1:0] MAT1[1:0] MAT2[1:0] MAT3[1:0]	Output	External Match Output – Khi 1 thanh ghi match(MR3:0) bằng với giá trị trong bộ đếm thời gian TC (Timer Counter) đầu ra này có thể được cấu hình để thay đổi mức, thiết lập mức cao, thiết lập mức thấp.

Mô tả các thanh ghi liên quan đến bộ Timer/Counter:

Thanh ghi	Mô tả	Kiểu truy cập
IR	Interrupt Register : IR có thể được ghi vào để xóa các ngắt hoặc có thể được đọc để nhận diện một trong tám nguồn ngắt có thể đang chờ được xử lý	Read/Write(R/W)
TCR	Timer Control Register : được sử dụng để điều khiển các chức năng của	R/W

	bộ Timer/Counter. Nó có thể dừng hoặc reset bộ Timer/Counter	
TC	Timer Counter : là thanh ghi 32 bit tăng theo mỗi chu kì PR+1 của PCLK. Được điều khiển bởi thanh ghi TCR	R/W
PR	Prescale Register : khi thanh ghi Prescale Counter bằng giá trị trong thanh ghi này thì ở chu kì clock tiếp theo nó tăng TC và clear PC	R/W
PC	Prescaler Counter : là thanh ghi 32 bit, thực hiện chức năng đếm giá trị clock và so sánh với PR	R/W
MCR	Match Control Register : được sử dụng để điều khiển nếu một ngắt được tạo ra và nếu TC reset khi có một sự kiện Match xảy ra	R/W
MR0/1/2/3	Match Register 0/1/2/3 : MRn có thể được cho phép qua MCR, khi giá trị trong thanh ghi MRn bằng với giá trị trong TC nó có thể tạo ra các đáp ứng như reset TC, dừng cả TC và PC hoặc tạo ra 1 ngắt hoặc không làm gì	R/W
CCR	Capture Control Register: CCR điều khiển sườn nào của các đầu vào Capture để copy giá trị các Capture Register và có hoặc không tạo ra một ngắt khi có một sự kiện capture xảy ra	R/W
CR0/1/2/3	Capture Register 0/1/2/3 : CR0/1/2/3 sẽ được load vào giá trị của TC khi có một sự kiện capture xảy ra ở pin capture tương ứng	ReadOnly(RO)
EMR	External Match Register : điều khiển các chân ngoài MATn.0-3	R/W
CTCR	Count Control Register: chọn mode hoạt động là Timer hay Counter và từ đó chọn tín hiệu clock nguồn	R/W

### 1.1. Chương trình điều khiển:

Ví dụ bên dưới sẽ sử dụng bộ TIMER0 để tạo ra trễ chính xác 0.5s. Cứ sau mỗi 0.5s, các led sẽ được cập nhật.  
 Chú ý trong ví dụ chúng ta thiết lập tốc độ PCLK cho TIMER0 là 15MHz.

#### Chương trình:

```
#include <LPC23xx.H>

// PCLK for TIMER0
```

```

#define TIMER_CLOCK 15000000

int main()
{
    int status=0;
    /* inits timer0 */

    /* reset timer */
    T0TCR = 0x02;
    /* Set value for PR to TC increments every 0.5s */
    T0PR = (TIMER_CLOCK/2)-1;

    /* init leds */
    PINSEL10 = 0;
    FIO2MASK = 0;
    FIO2DIR = 0xFF;

    while(1)
    {
        /* update leds */
        FIO2CLR=0xFF;
        FIO2SET=(1<<status);
        status++;
        status%=8;

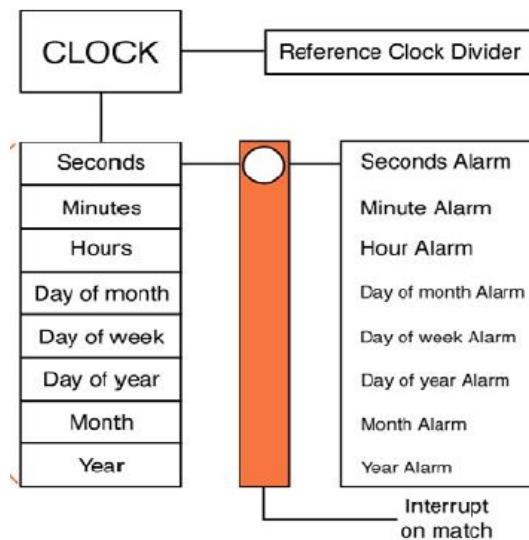
        /* delay 0.5ms */
        T0TCR=0x02; //Timer0 reset
        T0TCR=0x01; //TIMER0 run
        while(T0TC!=1) {}
        T0TCR=0; //Timer0 stop
    }
    return 0;
}

```

### 5.3.4 Đồng hồ thời gian thực

Đồng hồ thời gian thực là một bộ lịch biểu cho phép thực hiện tính toán quản lý thời gian trong lịch biểu như ngày , tháng, năm, tuần lễ,.. và đặc biệt là tạo ra giờ chính xác theo giờ phút giây.

Bộ RTC trong LPC2378 tạo ra lịch biểu chính xác cho đến năm 2099 và tùy chọn có thể tạo ra các ngắt khi thời gian hiện thời phù hợp với những thông số thời gian được chọn lựa trong các thanh ghi Alarm. Bộ RTC có thể chọn hoạt động trên thạch anh 32KHz ngoài hoặc từ PCLK bên trong. Nó cũng có bộ nhớ SRAM công suất thấp 2K bên trong được gọi là battery SRAM. RTC và battery SRAM được cung cấp bởi nguồn riêng từ pin Vbat. Chúng được thiết kế để tiêu tốn công suất thấp và có thể chạy từ nguồn pin.



Hình 5-14. Sơ đồ RTC

Phía trên là hình vẽ mô tả hoạt động chung của bộ RTC, tín hiệu clock đi vào bộ RTC qua khối **Reference Clock Divider** sẽ được chia ra để tạo ra nguồn clock phù hợp. Tín hiệu clock này vào bộ **Clock** tạo ra tín hiệu điều khiển cho các bộ đếm Second, Minute, Hours,... Khi được tùy chọn, nếu thời gian hiện tại phù hợp với các giá trị thời gian trong các thanh ghi Alarm thì nó sẽ tạo ra các ngắt theo ý muốn.

Mô tả các pin dành cho chức năng RTC:

Pin	Type	Mô tả
RTCX1	I	Đầu vào mạch dao động RTC
RTCX2	O	Đầu ra từ mạch dao động RTC
Vbat	I	Nguồn cung cấp cho bộ RTC

Mô tả các thanh ghi dành cho chức năng RTC:

Register	Size	Mô tả	Kiểu truy cập
ILR	2	Interrupt Location Register : đọc thanh ghi này để nhận diện nguồn ngắt và ghi 1 và bit tương ứng sẽ clear ngắt tương ứng	R/W
CTC	15	Clock Tick Counter : được reset thông qua thanh ghi CCR. Nó chứa các bit điều khiển bộ chia clock	RO
CCR	4	Clock Control Register : điều khiển hoạt động của bộ chia clock như chọn nguồn clock, reset,...	R/W
CIIR	4	Counter Increment Interrupt Register : chọn lựa bộ đếm nào trong các bộ đếm giây, phút, giờ, ... sẽ tạo ra ngắt mỗi khi các bộ đếm này tăng	R/W
AMR	8	Alarm Mask Register : được dùng để	R/W

		chọn lựa tổ hợp các thanh ghi nào trong các thanh ghi Alarm được so sánh với thời gian hiện tại để tạo ra ngắt	
CTIME0/1/2	32	Consolidated Time Register 0/1/2 : chứa tổ hợp các giá trị thời gian bậc thấp : giây, phút, giờ, và ngày trong tuần.	R/W
SEC MIN HOUR DOM DOW DOY MONTH YEAR		Là các thanh ghi chứa giá trị thời gian hiện thời	R/W
ALSEC ALMIN ALHOUR ALDOM ALDOW ALDOY ALMON ALYEAR		Các thanh ghi Alarm chứa giá trị để so sánh với giá trị thời gian hiện thời, thực hiện các tác vụ tương ứng	
PREINT	13	Prescale Value, integer portion - cùng với thanh ghi PREFRAC nó được sử dụng để tạo ra giá trị chia phù hợp khi sử dụng nguồn clock là PCLK	R/W
PREFRAC	15	Prescale Value, fractional portion - cùng với thanh ghi PREINT nó được sử dụng để tạo ra giá trị chia phù hợp khi sử dụng nguồn clock là PCLK	R/W

Thiết lập giá trị PREINT và PREFRAC khi sử dụng PCLK:

Như đã giới thiệu, bộ RTC có thể hoạt động hoặc từ bộ dao động RTC 32.768 kHz bên ngoài hoặc từ PCLK cung cấp bởi clock hệ thống sau khi đã được thay đổi bởi bộ **Reference Clock Divider**

Bộ **Reference Clock Divider** cho phép tạo ra tín hiệu clock 32.768 từ bất cứ nguồn clock nào lớn hơn 65.536 kHz(= 2x32.768 kHz). Điều này cho phép bộ RTC luôn chạy ở tốc độ chính xác bất chấp tốc độ của tín hiệu cung cấp. Bộ chia trong **Reference Clock Divider** cho phép chia tín hiệu vào từ PCLK với 1 giá trị chứa cả phần nguyên và phần phân số. Nên tốc độ đầu ra có thể biến thiên theo thời gian xong tổng hợp thì nó luôn đếm được 32,768 lần trên 1 giây.

Phần nguyên của bộ chia (chứa trong thanh ghi PREINT) là giá trị 13 bit và phần phân số( chứa trong PREFRAC) là giá trị 15 bit. Cụ thể tính toán PREINT và PREFRAC:

$$\begin{aligned} \text{PREINT} &= \text{phần nguyên}(\text{PCLK}/32768) - 1 \\ \text{PREFRAC} &= \text{PCLK} - ((\text{PREINT} + 1) \times 32768) \end{aligned}$$



### 5.3.5 Bộ điều chế độ rộng xung

Bộ điều chế độ rộng xung PWM có chức năng tạo ra các tín hiệu chu kỳ với độ rộng xung có thể điều khiển thông qua các thanh ghi điều khiển với giá trị chu kỳ, độ rộng xung được tính toán trước theo yêu cầu cụ thể của ứng dụng. Các bộ PWM hoạt động dựa trên các khối Timer và kế thừa các chức năng của nó.

Như ta biết các bộ Timer được thiết kế để đếm các chu kỳ của đồng hồ ngoại vi PCLK và tùy chọn có thể tạo ra các ngắt hoặc thực hiện các yêu cầu nào đó khi 1 giá trị thời gian xác định thiết lập trên các thanh ghi Match bằng với giá trị trong thanh ghi TC. Chức năng PWM dựa trên các chức năng này của bộ Timer.

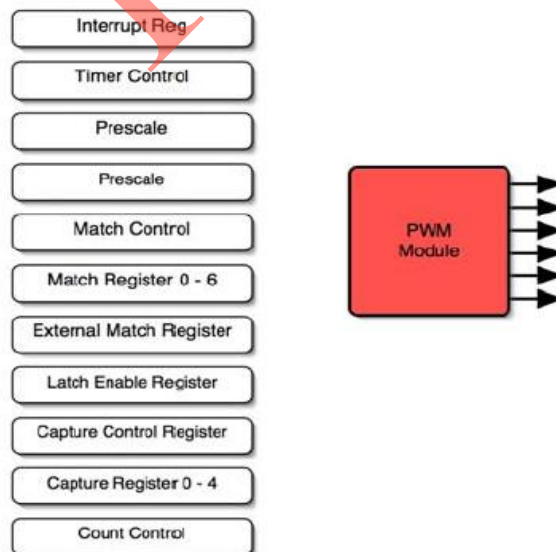
Với khả năng điều khiển riêng các thời điểm sườn dương và sườn âm của tín hiệu PWM nên các bộ PWM được sử dụng với nhiều ứng dụng hơn. Ví dụ điển hình là các bộ điều khiển motor nhiều phase.

Với mỗi tín hiệu đầu ra của bộ điều khiển độ rộng xung được điều khiển thông qua 2 hoặc 3 thanh ghi Match:

- Trường hợp sử dụng 2 thanh ghi Match: được sử dụng để tạo tín hiệu ra được điều khiển theo 1 sườn :
  - o Một thanh ghi (PWMMR0) điều khiển chu kỳ tín hiệu
  - o Một thanh ghi khác dùng để điều khiển vị trí của sườn
- Trường hợp sử dụng 3 thanh ghi Match: được sử dụng để tạo ra tín hiệu ra được điều khiển theo cả 2 sườn:
  - o Một thanh ghi (PWMMR0) điều khiển chu kỳ tín hiệu
  - o Hai thanh ghi còn lại điều khiển các vị trí của 2 sườn.

Do sử dụng chung PWMMR0 nên chu kỳ của các tín hiệu đầu ra giống nhau

Hình bên dưới mô tả các thành phần của bộ PWM dưới quan điểm lập trình gồm tập các thanh ghi điều khiển và thanh ghi trạng thái, cùng các tín hiệu đầu ra.



Hình 5-15. Sơ đồ bộ PWM

Mô tả các pin sử dụng cho bộ PWM:

Pin	Type	Mô tả
PWM1[1]	Output	Đầu ra từ bộ PWM kênh 1
PWM1[2]	Output	Đầu ra từ bộ PWM kênh 2
PWM1[3]	Output	Đầu ra từ bộ PWM kênh 3
PWM1[4]	Output	Đầu ra từ bộ PWM kênh 4
PWM1[5]	Output	Đầu ra từ bộ PWM kênh 5
PWM1[6]	Output	Đầu ra từ bộ PWM kênh 6
PCAP1[1:0]	Input	Các đầu vào Capture : sự chuyển mức trên pin capture có thể được cấu hình để copy giá trị của TC vào thanh ghi Capture tương ứng và tùy chọn có thể tạo ra 1 ngắt

Mô tả các thanh ghi sử dụng cho bộ PWM:

Tập các thanh ghi của PWM giống với bộ Timer và thêm một tập các thanh ghi riêng

Thanh ghi	Mô tả	Kiểu truy cập
IR	Interrupt Register : IR có thể được ghi vào để xóa các ngắt hoặc có thể được đọc để nhận diện một trong tám nguồn ngắt có thể đang chờ được xử lý	R/W
TCR	Timer Control Register : được sử dụng để điều khiển các chức năng của bộ Timer/Counter. Nó có thể cấm hoặc reset bộ Timer/Counter	R/W
TC	Timer Counter : là thanh ghi 32 bit tăng theo mỗi chu kỳ PR+1 của PCLK. Được điều khiển bởi thanh ghi TCR	R/W
PR	Prescale Register : khi thanh ghi Prescale Counter bằng giá trị trong thanh ghi này thì ở chu kỳ clock tiếp theo nó tăng TC và clear PC	R/W
PC	Prescaler Counter : là thanh ghi 32 bit, thực hiện chức năng đếm giá trị clock và so sánh với PR	R/W
MCR	Match Control Register : được sử dụng để điều khiển nếu một ngắt được tạo ra và nếu TC reset khi có một sự kiện Match xảy ra	R/W
MR0/1/2/3/4/5/6	Match Register 0/1/2/3/4/5/6 : MRn có thể được cho phép qua MCR, khi giá trị trong thanh ghi MRn bằng với giá trị trong TC nó có thể tạo ra các đáp ứng như reset TC, dừng cả TC và PC hoặc tạo ra một ngắt hoặc không làm gì. Thêm nữa nó được sử dụng để điều khiển các đầu ra bộ PWM	R/W
CCR	Capture Control Register: CCR điều khiển sườn nào của các đầu vào Capture để copy giá trị các Capture Register và có hoặc	R/W

	không tạo ra một ngắt khi có một sự kiện capture xảy ra	
CR0/1/2/3	Capture Register 0/1/2/3 : CR0/1/2/3 sẽ được load vào giá trị của TC khi có 1 sự kiện capture xảy ra ở pin capture tương ứng	RO
PCR	PWM Control Register : cho phép các đầu ra PWM và chọn loại của kênh là điều khiển đơn sườn hay cả 2 sườn	R/W
CTCR	Count Control Register: chọn mode hoạt động là Timer hay Counter và từ đó chọn tín hiệu clock nguồn	R/W
LER	Load Enable Register : cho phép sử dụng các giá trị Match mới trong bộ PWM. Chỉ khi bit này được cho phép thì đầu ra mới đáp ứng với các giá trị trong thanh ghi Match	R/W

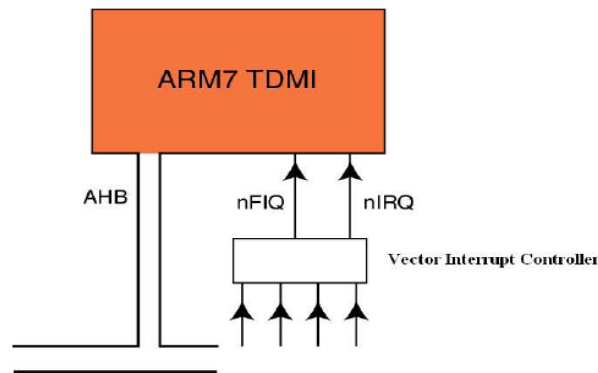
### 5.3.6 Điều khiển ngắt

Có hai loại bộ điều khiển ngắt được sử dụng trong bộ vi xử lý ARM là bộ điều khiển ngắt chuẩn và bộ điều khiển ngắt vector VIC – *vector interrupt controller*.

Bộ điều khiển chuẩn gửi một tín hiệu ngắt đến lõi vi xử lý khi một thiết bị ngoài yêu cầu phục vụ. Nó có thể được lập trình để bỏ qua yêu cầu ngắt từ một hoặc nhiều thiết bị. Trình xử lý ngắt xác định thiết bị yêu cầu phục vụ bằng việc đọc một thanh ghi của thiết bị trong bộ điều khiển ngắt.

VIC mạnh hơn bộ điều khiển ngắt chuẩn bởi nó cho phép thiết lập các mức ưu tiên cho các ngắt và đơn giản quá trình xác định nguồn ngắt. Sau khi gán một mức ưu tiên và địa chỉ trình xử lý ngắt với mỗi ngắt, VIC chỉ cần gửi một tín hiệu ngắt tới lõi nếu mức ưu tiên của ngắt mới cao hơn trình xử lý ngắt đang thực thi. Dựa vào tín hiệu ngắt này, VIC có thể gọi trình xử lý ngoại lệ ngắt chuẩn tại địa chỉ của trình phục vụ ngắt cho thiết bị từ VIC hoặc tạo một lệnh nhảy cho lõi đến trình xử lý ngắt trực tiếp. Cụ thể, mỗi khi có một ngắt xảy ra, VIC sẽ OR các yêu cầu ngắt, và tạo ra một tín hiệu ngắt đến một trong hai ngắt FIQ và IRQ tùy theo các thông số của các thanh ghi cấu hình cho VIC, và nó sẽ gửi địa chỉ của thường trình phục vụ ngắt vào thanh ghi VICVectAddr. Trình xử lý ngắt sẽ bắt đầu bằng việc đọc thanh ghi này và nhảy tới địa chỉ được cung cấp bởi thanh ghi này. Và khi kết thúc chương trình phục vụ ngắt, ngoài việc clear cờ ngắt của nguồn ngắt tương ứng, VICVectAddr phải được ghi vào một giá trị bất kỳ để update phần cứng mức ưu tiên của VIC.

CPU ARM có 2 đường phục vụ cho ngắt ngoài là FIQ cho ngắt nhanh và IRQ cho ngắt thông thường. Một hệ thống ARM7 chỉ nên có một nguồn ngắt tạo ra FIQ, các ngắt khác còn lại sẽ được kết nối tới IRQ.



Hình 5-16. Cấu trúc hệ thống xử lý ngắt trong LPC2378

### 5.3.7 Giao tiếp UART

Máy tính truyền dữ liệu theo hai phương pháp : song song hoặc nối tiếp. Truyền dữ liệu song song thường sử dụng nhiều đường dây dẫn để truyền dữ liệu đến một khoảng cách xa vài mét ví dụ như trong giao tiếp với máy in và ổ đĩa cứng. Phương pháp này cho phép truyền với tốc độ cao nhưng khoảng cách truyền bị hạn chế. Để truyền dữ liệu đi xa thì cần phương pháp truyền nối tiếp. Theo phương pháp này, dữ liệu được truyền đi theo từng bit.

Để tổ chức truyền tin nối tiếp, trước hết byte dữ liệu được chuyển thành các bit nối tiếp nhờ thanh ghi dịch vào song song –ra nối tiếp. Tiếp đó dữ liệu được truyền qua một đường dữ liệu đơn, ở đầu thu, nhờ một thanh ghi dịch vào nối tiếp-ra song song, dữ liệu được nhận nối tiếp và được gói thành từng byte một. Khi tín hiệu được truyền đi xa, yêu cầu có một modem để điều chế tín hiệu và sau đó giải điều chế ở bộ thu.

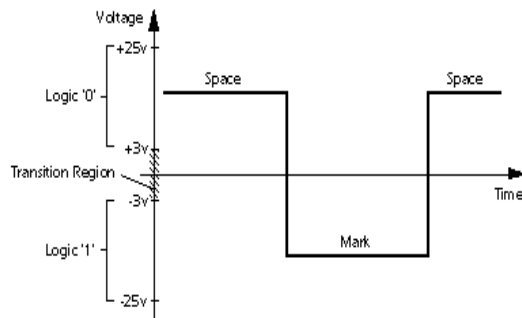
Truyền tin nối tiếp có hai phương pháp: đồng bộ và bất đồng bộ

- Đồng bộ: truyền mỗi lần một khối dữ liệu
- Bất đồng bộ: chỉ truyền từng byte một.

#### Chuẩn RS-232

Trong truyền tin nối tiếp, RS-232 là một chuẩn cho kết nối tín hiệu dữ liệu nhị phân nối tiếp giữa một DTE (data terminal equipment) và một DCE (data circuit-terminating equipment) được xây dựng năm 1960 nhằm tương thích dữ các thiết bị truyền dữ liệu nối tiếp giữa các hãng khác nhau. Ngày nay nó là một chuẩn được sử dụng khá rộng rãi. Tuy nhiên do chuẩn này ra đời khá lâu trước khi có họ mạch điện tử TTL vì vậy mức điện áp của nó không tương thích với TTL.

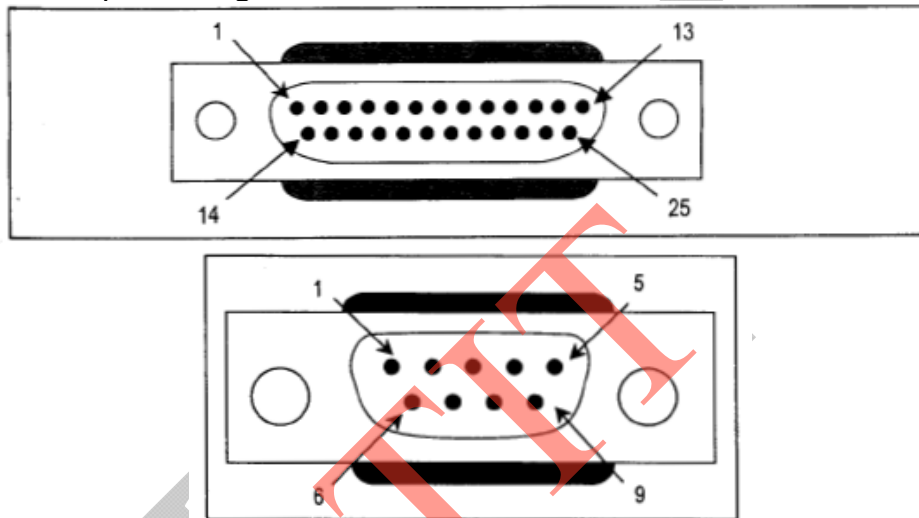
RS-232 định nghĩa mức điện thế mà tương ứng với các mức logic 0 và 1. Tín hiệu có mức điện thế như sau : mức logic 0: từ +3V đến +25V, mức logic 1 : từ -25V đến -3V.



Hình 5-17. Mức điện thế biểu diễn tín hiệu nhị phân

### Về bố trí chân của RS232

RS232 có 2 phiên bản giắc đầu nổi là DB-25 và DB-9

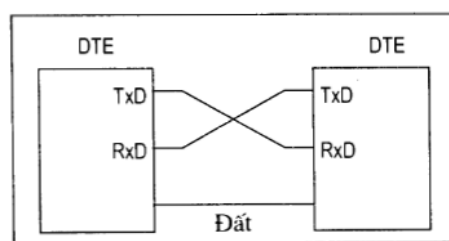


Hình 5-18. DB-25 và DB-9

Hiện tại DB-25 ít được sử dụng nên ta chỉ xem xét cách bố trí chân đầu nối của DB-9 thông qua bảng dưới:

Chân	Mô tả	Ý nghĩa
1	Data carrier detect(DCD)	Tách tín hiệu mang dữ liệu
2	Received Data(RxD)	Dữ liệu nhận được
3	Transmitted Data(TxD)	Dữ liệu được gửi
4	Data terminal ready(DTR)	Đầu cuối dữ liệu sẵn sàng
5	Signal Ground(GND)	Đất của tín hiệu
6	Data set ready(DSR)	Dữ liệu sẵn sàng
7	Request to send(RTS)	Yêu cầu gửi
8	Clear to send(CTS)	Xóa để gửi
9	Ring indicator(RL)	Báo chuông

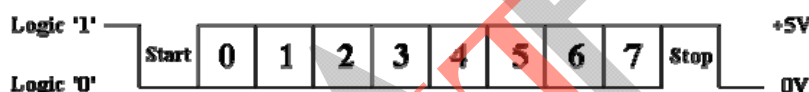
Một kết nối đơn giản nhất giữa PC và bộ vi điều khiển yêu cầu tối thiểu 3 chân TxD, RxD, GND như trong hình vẽ dưới đây :



Hình 5-19. Kết nối tối thiểu giữa PC với vi điều khiển

#### Định dạng bản tin:

Trong truyền bất đồng bộ, đường liên kết không bao gồm một đường clock, vì trong mỗi đầu cuối thiết bị đều có một clock. Vì vậy hai clock phải có chung tần số, cho phép sai khác nhỏ. Mỗi một byte được truyền gồm một bit Start để đồng bộ các clock, và một hoặc nhiều bit Stop để báo hiệu kết thúc một từ truyền. Một đường truyền bất đồng bộ sử dụng nhiều định dạng, nhưng định dạng phổ biến nhất là 8-N-1 ở đó bộ truyền sẽ gửi mỗi byte dữ liệu với 1 bit Start, theo sau 8 bit dữ liệu, bắt đầu với bit 0 (LSB) và kết thúc với 1 bit Stop như mô tả hình dưới



Hình 5-20. Định dạng 8-N-1

#### Mô tả các chân UART:

Pin	Type	Mô tả
RXD0/2/3	Input	Serial Input : dữ liệu nhận nối tiếp
TXD0/2/3	Output	Serial Output: dữ liệu truyền nối tiếp

#### Mô tả các thanh ghi:

Thanh ghi	Mô tả	Kiểu truy cập
RBR	Receiver Buffer Register : chứa byte đầu trong bộ đệm nhận FIFO của cổng UART cũng là byte được nhận trước nhất. Bit DLAB trong thanh ghi LCR phải là 0 để có thể truy cập RBR.	RO
THR	Transmit Holding Register : là byte đầu của bộ đệm truyền FIFO của cổng UART cũng là kí tự mới nhất trong bộ đệm này. Bit DLAB trong thanh ghi LCR phải là 0 để truy cập THR. Byte này sẽ được truyền khi nó tiến đến cuối của bộ đệm	WO(WriteOnly)
DLL	Divisor Latch LSB Kết hợp với nhau chúng chứa giá trị hệ số chia để chia tần hiệu PCLK nhằm tạo ra tốc độ	R/W



DLM	Divisor Latch MSB	baude phù hợp. Giá trị chia là giá trị 16 bit với DLL là byte cao và DLM là byte thấp. Chúng chỉ được truy cập khi bit DLAB bằng 1	
IER	Interrupt Enable Register	: được sử dụng để cho phép hoặc cấm các chức năng ngắt trong UART . Chỉ được truy cập khi bit DLAB bằng 0	R/W
IIR	Interrupt Identification Register	: thanh ghi nhận diện nguồn ngắt chứa trạng thái các ngắt đang yêu cầu xử lý liên quan đến chức năng UART	RO
FCR	FIFO Control Register	: điều khiển hoạt động của các bộ đệm FIFO nhận và truyền như : cho phép bộ đệm hoạt động, reset bộ đệm..	R/W
LCR	Line Control Register	: dùng để xác định định dạng của kí tự sẽ được truyền hoặc nhận như : độ dài từ, số lượng bit stop, tùy chọn kiểm tra chẵn lẻ..	R/W
LSR	Line Status Register	: chứa thông tin về trạng thái trên các khối TX và RX của cổng UART như đã nhận dữ liệu chưa, lỗi,...	RO

Và còn 1 số thanh ghi nữa tuy nhiên sẽ không đề cập chi tiết trong tài liệu này, tham khảo trong datasheet nhà sản xuất.

Tính toán tốc độ baud:

$$\text{Baudrate} = \frac{PCLK}{16 \times (256 \times DLM + DLL)}$$

Chương trình ví dụ:

```
#include <LPC23xx.H> /* LPC2300 definitions */

void init_serial (void) {
    /* Initialize Serial Interface */

    U0FDR = 0; /* Fractional divider not used */
    U0LCR = 0x83; /* 8 bits, no Parity, 1 Stop bit */
    U0DLL = 94; /* 9600 Baud Rate @ 14.4 MHZ PCLK */
    U0DLM = 0; /* High divisor latch = 0 */
    U0LCR = 0x03; /* DLAB = 0 */
}

/* Implementation of putchar */
int sendchar (int ch) { /* Write character to Serial Port */
    if(ch=='\n') ch = 0x0d; /*new-line character*/
    while (!(U0LSR & 0x20));
    return (U0THR = ch);
}

int getkey (void) { /* Read character from Serial Port */
```

```

while (!(U0LSR & 0x01));
return (U0RBR);
}

int sendstring(char * s) {    /* write a string */
    while(*s!='\0')
    {
        sendchar(*s);
        s++;
    }
}

int main()
{
    init_serial();
    while(1)
    {
        sendstring("hello \n");
    }
}

```

### 5.3.8 Giao tiếp I2C

I2C, viết tắt của Inter-Integrated Circuit, là một loại bus nối tiếp được phát triển bởi hãng sản xuất linh kiện điện tử Philips. Ban đầu, loại bus này chỉ được dùng trong các linh kiện điện tử của Philips. Sau đó, do tính ưu việt và đơn giản của nó, I2C đã được chuẩn hóa và được dùng rộng rãi trong các module truyền thông nối tiếp trên các vi mạch tích hợp ngày nay.

I2C sử dụng chỉ 2 đường 2 chiều là đường dữ liệu – SDA (Serial Data) và đường cung cấp xung nhịp – SCL (Serial Clock), nên không yêu cầu khối logic chọn và phân quyền bus, việc thiết kế phần cứng trở nên đơn giản hơn nhiều. Mọi thiết bị được gắn lên bus đều có địa chỉ duy nhất. Mỗi thiết bị này hoạt động như 1 bộ thu hoặc 1 bộ nhận hoặc cả thu và nhận. Ví dụ như 1 LCD thì chỉ nhận, trong khi 1 bộ nhớ thì thực hiện cả 2 chức năng.

Sử dụng chỉ 2 đường SDA và SCL trong I2C cho phép truyền dữ liệu nối tiếp dưới dạng các byte địa chỉ 8-bit với không gian địa chỉ 7-bit cho phép hỗ trợ lên tới xấp xỉ 128 thiết bị sử dụng trên cùng 1 bus.

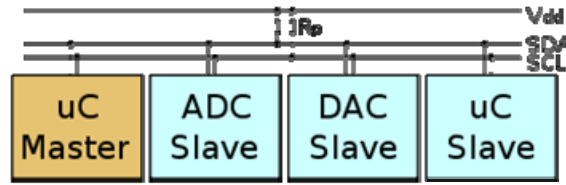
Trong một hệ thống sử dụng bus, thiết bị khởi tạo 1 quá trình trao đổi dữ liệu được gọi là master, còn thiết bị được địa chỉ hóa bởi master hay điều khiển bởi master được gọi là slave. Thông thường master điều khiển tín hiệu clock. I2C là một bus multi-master, có nghĩa là có thể có hơn 1 IC có khả năng khởi tạo 1 quá trình trao đổi dữ liệu.

#### Tốc độ:

Tốc độ giao tiếp của I2C có thể lên đến 3.4Mbps với nhiều tùy chọn tốc độ:

- Standard : 100 kbps
- Fast : 400 kbps
- Fast mode plus : 1 Mbps

- High speed mode : 3.4 Mbps



Hình 5-21. Ví dụ bus I2C

### Hoạt động:

Khi bắt đầu 1 quá trình trao đổi dữ liệu, trước hết master khởi tạo quá trình truyền bằng cách gửi 1 start bit theo sau là địa chỉ 7-bit của slave mà nó muốn giao tiếp, tiếp đó là 1 bit đơn mô tả quá trình là ghi tới (bit 0) hay đọc (bit 1) từ slave.

Nếu slave tồn tại trên bus thì nó sẽ đáp ứng lại 1 bit ACK với địa chỉ trên. Master sau đó sẽ tiếp tục hoặc truyền hoặc gửi.

Nếu master muốn tiếp tục gửi dữ liệu đến slave thì nó tiếp tục gửi 1 byte và slave gửi lại 1 bit ACK

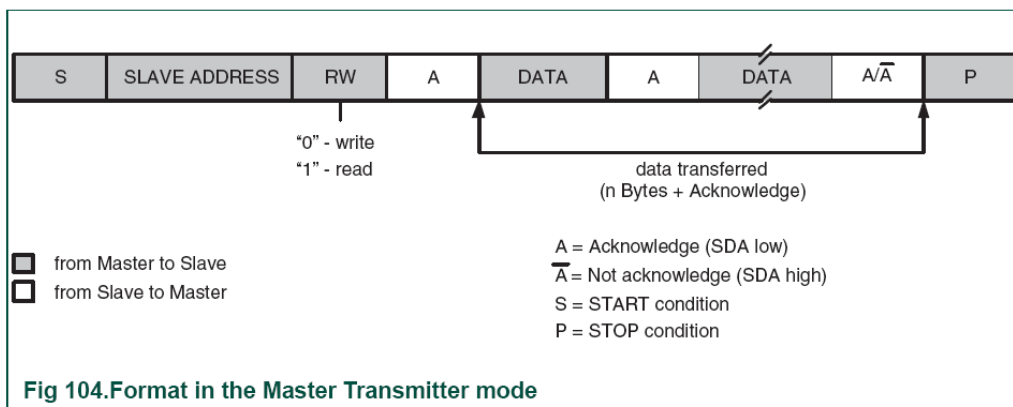
Nếu master muốn tiếp tục đọc từ slave thì nó tiếp tục nhận 1 byte từ slave và gửi 1 bit ACK sau mỗi byte nhận được.

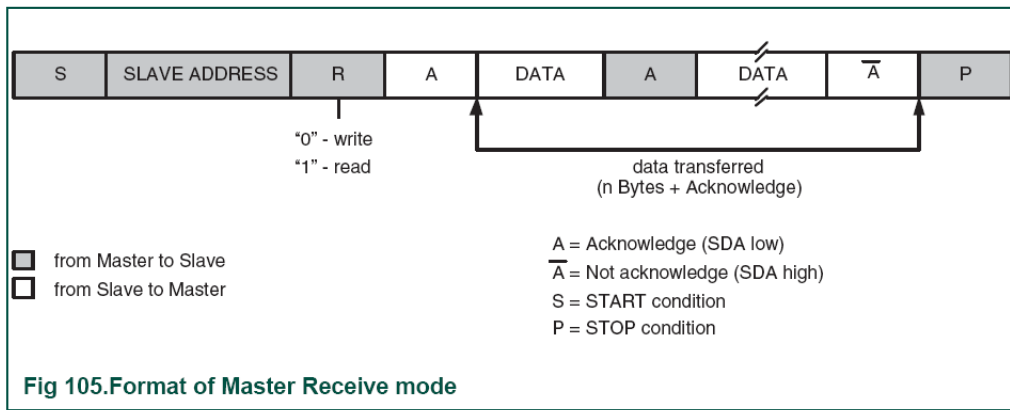
Master muốn kết thúc trao đổi thì nó sẽ gửi 1 stop bit lên bus.



Hình 5-22. Lược đồ thời gian

Lược đồ trên mô tả timing của quá trình truyền. Mỗi quá trình truyền bắt đầu với 1 bit Start(S) khi SDA bị kéo xuống thấp trong khi SCL vẫn ở mức cao. Sau đó SDA bắt đầu thiết lập bit được truyền trong khi SCL mức thấp và dữ liệu được lấy mẫu (nhận) khi SCL chuyển mức. Khi quá trình truyền kết thúc, 1 bit Stop(P) được truyền bằng cách giải phóng đường SDA để nó được kéo lên mức cao trong khi SCL vẫn ở mức cao.





Hình 5-23. Ví dụ qua trình truyền tin

### I2C trong LPC2378:

LPC2378 hỗ trợ 3 cổng giao tiếp I2C.

Mô tả các pin I2C:

Pin	Loại	Mô tả
SDA0/1/2	I/O	Đường dữ liệu SDA
SCL0/1/2	I/O	Đường tín hiệu đồng hồ SCL

Mô tả các thanh ghi I2C:

Register	Mô tả	Kiểu truy cập
I2CONSET	I2C Control Set Register : Thanh ghi thiết lập điều khiển điều khiển I2C. Khi 1 được ghi vào 1 bit của thanh ghi này thì bit tương ứng trong thanh ghi điều khiển I2C sẽ được set. Ghi 0 vào sẽ không có hiệu ứng gì	R/W
I2STAT	I2C Status Register : thanh ghi trạng thái I2C. Trong hoạt động của I2C, thanh ghi này cung cấp các mã trạng thái chi tiết cho phép phần mềm xác định đáp ứng cần thiết tiếp theo	RO
I2DAT	I2C Data Register : thanh ghi dữ liệu I2C. Trong quá trình truyền dữ liệu, dữ liệu để được truyền sẽ được ghi vào đây, còn trong quá trình đọc dữ liệu, dữ liệu đã nhận được sẽ đọc từ thanh ghi này	R/W
I2ADR	I2C Slave Address Register: thanh ghi dữ liệu slave, chứa địa chỉ 7-bit của slave khi thiết bị hoạt động ở mode slave. Biết trọng số nhỏ nhất xác định rằng slave có đáp ứng tới địa chỉ gọi chung hay không	R/W
I2SCLH	SCH Duty Cycle Register High Half Word : nửa từ cao thanh ghi chu kỳ hoạt động SCH : xác định thời gian mức cao của clock I2C	R/W

I2SCLL	SCH Duty Cycle Register Low Half Word : nửa từ thấp thanh ghi chu kỳ hoạt động SCH cùng với I2SCLH xác định tần số clock được tạo bởi 1 I2C master và thời gian phù hợp được sử dụng trong chế độ slave. Nó xác định thời gian mức thấp của clock I2C	R/W
I2CONCLR	I2C Control Clear Register : thanh ghi xóa điều khiển. Khi 1 được ghi vào một bit của thanh ghi này, bit tương ứng trong thanh ghi điều khiển sẽ được clear, ghi 0 vào sẽ không có hiệu ứng gì	R/W

#### Xác định tần số hoạt động của bus I2C:

Phần mềm sẽ phải thiết lập các thông số phù hợp cho I2SCLH và I2SCLL để thiết lập tần số hoạt động của bus I2C. Trong đó I2SCLH là số chu kỳ của clock ngoại vi cung cấp cho module I2C PCLK mà SCL ở mức cao còn I2SCLL là số chu kỳ của PCLK mà SCL ở mức thấp. Nên tần số clock của SCL hay của I2C là:

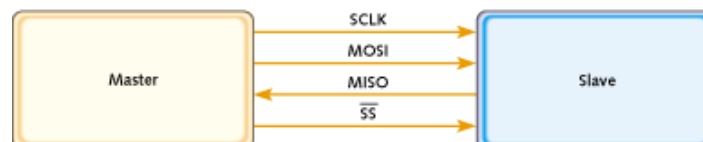
$$I^2C_{bitfrequency} = \frac{f_{PCLK}}{I2CSCLH + I2CSCLL}$$

### 5.3.9 Giao tiếp SPI

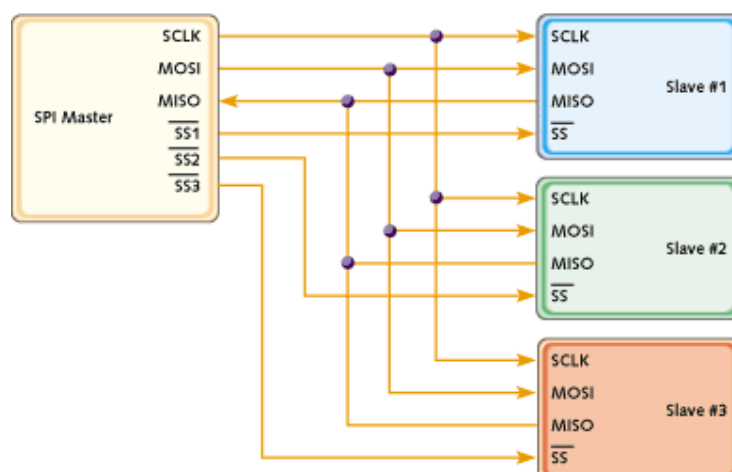
Giao diện ngoại vi nối tiếp SPI (Serial Peripheral Interface) là một giao tiếp nối tiếp song công. Nó có thể có nhiều master và slave cùng được kết nối trên một bus xác định. Chỉ một master và một slave có thể truyền dữ liệu tại mỗi thời điểm. Trong suốt quá trình truyền dữ liệu master luôn luôn gửi từ 8 đến 16 bit dữ liệu tới slave và slave luôn gửi một byte dữ liệu tới master.

Cả SPI và I2C đều hỗ trợ tốt cho các ngoại vi tốc độ chậm được truy cập không liên tục như EEPROM và Real-Time Clock nhưng SPI có tốc độ dữ liệu cao hơn I2C, thường lên tới hàng chục Mbps, phù hợp cho các ứng dụng truyền dữ liệu theo luồng. Nó còn hiệu quả hơn nhờ giao diện song công đầy đủ, sử dụng trong các ứng dụng giữa 1 bộ codec và 1 bộ xử lý tín hiệu số.

Cũng như trong I2C, master khởi tạo quá trình trao đổi dữ liệu. Khi master tạo ra clock và chọn thiết bị slave, dữ liệu có thể được truyền theo 2 hướng đồng thời. Thực tế thì dữ liệu được truyền theo cả hai hướng, nên master và slave phải biết khi nào byte nhận có ý nghĩa hay không. Nên thiết bị phải vứt bỏ byte nhận được khi nó đang truyền và tạo ra 1 byte giả để truyền khi nó đang nhận.



Hình 5-24. Cấu hình 1 master, 1 slave



Hình 5-25. Cấu hình 1 master nhiều slave

Giao tiếp SPI gồm 4 đường tín hiệu : clock (SCLK), dữ liệu ra từ master vào slave MOSI (Master Out Slave In), dữ liệu vào master từ slave MISO (Master In Slave Out), và tín hiệu chọn slave SS (Slave Select).

Khi giao tiếp sử dụng SPI cần để ý cặp thông số cực clock CPOL (Clock Polarity) và phase clock CPHA (Clock Phase), cặp thông số này được dùng để xác định sườn của tín hiệu clock mà dữ liệu được lái và lấy mẫu. Mỗi thông số có 2 trạng thái mức cao – mức thấp nên tổng hợp lại có 4 tổ hợp có thể mà mỗi tổ hợp đều không tương thích với các tổ hợp còn lại. Nên cặp giao tiếp master/slave phải sử dụng đúng cặp thông số để giao tiếp.

Các tổ hợp CPOL và CPHA xác định:

- Khi nào bit dữ liệu đầu tiên được thiết lập
- Khi nào các bit dữ liệu khác được thiết lập
- Khi nào dữ liệu được lấy mẫu

Mối quan hệ này được biểu diễn ở bảng dưới

Tổ hợp CPOL và CPHA	Bit đầu tiên được thiết lập	Các bit khác được thiết lập	Dữ liệu được lấy mẫu
CPOL = 0, CPHA = 0	Trước sườn dương SCK đầu tiên	Sườn âm SCK	Sườn dương SCK
CPOL = 0, CPHA = 1	Sườn dương SCK đầu tiên	Sườn dương SCK	Sườn âm SCK
CPOL = 1, CPHA = 0	Trước sườn âm SCK đầu tiên	Sườn dương SCK	Sườn âm SCK
CPOL = 1, CPHA = 1	Sườn âm SCK đầu tiên	Sườn âm SCK	Sườn dương SCK

Mô tả các thanh ghi SPI:

Register	Mô tả	Kiểu truy cập
S0SPCR	SPI Control Register : thanh ghi điều khiển hoạt động SPI	R/W
S0SPSR	SPI Status Register : thanh ghi trạng thái SPI,	RO

	thông báo các trạng thái của SPI	
S0SPDR	SPI Data Register : đây là thanh ghi 2 chiều cung cấp dữ liệu truyền và nhận cho SPI. Dữ liệu truyền được cung cấp tới SPI0 được ghi vào thanh ghi này còn dữ liệu nhận được bởi SPI0 có thể đọc từ thanh ghi này	R/W
S0SPCCR	SPI Clock Counter Register : thanh ghi điều khiển tần số của master SCK0	R/W
S0SPINT	SPI Interrupt Flag : chứa cờ ngắt cho giao diện SPI	R/W

Ví dụ sau gồm các hàm cơ bản cho phép thực hiện truyền và nhận dữ liệu trên SPI. Các hàm này có thể sử dụng như là thư viện cho các ứng dụng lớn hơn.

```
#include "LPC23xx.h"

void SPIInit( void )
{
    PCONP |= (1 << 8);
    S0SPCR = 0x00;
    /* enable ports */
    PINSEL0 |= 0xC0000000;
    PINSEL1 |= 0x0000003C;
    IODIR0 = 1 << 16;
    IOSET0 = 1 << 16;

    /* Setting SPI0 clock*/
    S0SPCCR = 0x8;
    S0SPCR = 0x00000020;
}

/* Send a block of data to the SPI port, the first */
void SPISend( unsigned char *buf, int Length )
{
    int i;
    unsigned char Dummy;

    if ( Length == 0 )
        return;
    for ( i = 0; i < Length; i++ )
    {
        S0SPDR = *buf;

        while ( !(S0SPSR & SPIF) );
        Dummy = S0SPDR;      /* Flush the Rx FIFO */
        buf++;
    }
    return;
}

/*the module will receive a block of data from SPI*/
void SPIReceive( BYTE *buf, int Length )
{
    int i;
```



```

for ( i = 0; i < Length; i++ )
{
    *buf = SPIReceiveByte();
    buf++;
}
return;
}
/*Receive one byte of data from the SPI port*/
unsigned char SPIReceiveByte( void )
{
    unsigned char data;

    /* wrtie dummy unsigned char out to generate
    clock, then read data from MISO */
    S0SPDR = 0xFF;
    /* Wait for transfer complete, SPIF bit set */
    while ( !(S0SPSR & SPIF) );
    data = S0SPDR;
    return ( data );
}

```

### 5.3.10 Thiết kế điều khiển giao tiếp với thiết bị tương tự: ADC, DAC

#### Giao tiếp với ADC

Các bộ ADC như tên gọi đã chỉ rõ là các thiết bị thực hiện chuyển đổi 1 tín hiệu liên tục thành các xung rời rạc. Cụ thể trong điện tử chúng chuyển đổi 1 đầu vào tương tự thành 1 giá trị số hóa tương đương với mức của biên độ của tín hiệu đầu vào hoặc ở dạng dòng điện hoặc ở dạng điện thế trên một dải giá trị biên độ của tín hiệu đầu vào.

Khi xét đến bộ ADC cần quan tâm đến 2 tham số là độ chính xác và tốc độ chuyển đổi.

Độ chính xác mô tả số các mức lượng tử rời rạc mà bộ ADC có thể tạo ra trên dải các giá trị tín hiệu đầu vào tương tự. Các giá trị thông thường được lưu trong các thiết bị số ở dạng nhị phân nên độ chính xác cũng được biểu diễn dưới dạng số bit. Ví dụ một ADC với độ chính xác là 8 bit có khả năng mã hóa một đầu vào tương tự sang 1 trong 256 mức khác nhau. Giá trị có thể là từ 0 đến 256 (số nguyên không dấu) hoặc từ -128 đến 127 (số nguyên có dấu) tùy thuộc vào ứng dụng.

Tốc độ chuyển đổi là thời gian đáp ứng từ khi bộ ADC bắt đầu chuyển đổi cho đến khi bộ ADC đưa ra được mức lượng tử của đầu vào. Độ chính xác càng cao và thời gian chuyển đổi càng nhanh thì bộ ADC có chất lượng càng tốt.

*Bộ ADC trong LPC2378:*

Gồm có 6 kênh đầu vào được ghép kênh với độ chính xác là 10 bit, dải điện áp tính toán cho phép là từ 0 đến 3 V với tốc độ chuyển đổi  $\geq 2.44 \mu\text{S}$ . Ngoài ra nó còn được thiết kế để cho phép điều khiển chuyển đổi thông qua xung khi có một sự dịch chuyển mức từ các pin đầu vào hoặc khi có 1 sự kiện Match trong bộ Timer. Tín hiệu clock được cung cấp bởi PCLK dành cho bộ ADC, đồng thời bên trong bộ ADC còn có bộ chia clock khả trình cho phép tạo ra tín hiệu clock phù hợp

*Mô tả các pin:*

Pin	Type	Mô tả
AD0[5:0]	Input	Đầu vào tương tự
VREF	Tham chiếu	Đầu vào điện áp tham chiếu : nó cung cấp các mức điện áp tham chiếu cho quá trình chuyển đổi

*Mô tả các thanh ghi ADC*

Tên	Mô tả	Kiểu truy cập
AD0CR	A/D Control Register. AD0CR phải được viết vào để lựa chọn chế độ hoạt động trước khi việc chuyển đổi AD có thể diễn ra	R/W
AD0GDR	A/D Global Data Register. AD0GDR chứa các kết quả của quá trình chuyển đổi gần nhất	R/W
AD0STAT	AD Status Register. AD0STAT chứa cờ DONE và cờ OVERUN cho tất cả các kênh AD, cũng như cờ ngắt của AD	RO
AD0INTEN	A/D Interrupt Enable Register. Thanh ghi này chứa các bit kích hoạt mà cho phép cờ DONE của mỗi kênh A/D được bao gồm hoặc loại trừ ra từ việc đóng góp hoặc tạo ra của một ngắt A/D	R/W
AD0DR0-7	A/D Channel 0-7 Data Register. Các thanh ghi này chứa kết quả của quá trình chuyển đổi gần nhất được hoàn thành trên kênh 0-7	R/W

Ví dụ sử dụng ADC. Trong ví dụ này, chương trình sẽ liên tục đọc giá trị đầu vào ADC và lưu trong biến `value`. Có thể mở rộng chương trình dựa trên xử lý giá trị này để tạo các tính năng mở rộng hơn cho từng ứng dụng cụ thể.

```
#include "LPC23xx.h"

void ADC_init( void )
{
    /* Enable CLOCK into ADC controller */
    PCONP |= (1 << 12);
    /* select channel AD0.0 on P0.23 */
    PINSEL1 = (1<<14);
    /*set operation mode*/
    AD0CR = ( 0x01 << 0 ) | /* SEL=1 */
            ( 4 << 8 ) | /* CLKDIV = 4 */
            ( 0 << 16 ) | /* BURST = 0 */
            ( 0 << 17 ) | /* CLKS = 0 */
            ( 1 << 21 ) | /* PDN = 1 */
            ( 0 << 22 ) | /* TEST1:0 = 00 */
            ( 0 << 24 ) | /* START = 0 */
            ( 0 << 27 ); /* EDGE = 0 */

    return;
}
```

```

void start_convert() {
    AD0CR |= (1<<24);
}

void stop_convert() {
    AD0CR &= ~(1<<24);
}

int check_done() {
    if (AD0STAT & 1 ==0) return 0;
    return 1;
}

int get_value() {
    return (AD0DR0>>6)&0x3ff;
}
/*****
**   Main Function   main()
*****/

int main (void)
{
    int value;
    ADC_init();
    while(1) {
        start_convert();
        while (!check_done()) {}
        value=get_value();
        stop_convert();
    }
}

```

## Giao tiếp DAC

Bộ chuyển đổi số sang tương tự DAC là thiết bị thực hiện chuyển đổi tín hiệu rời rạc dạng số sang tín hiệu liên tục tương tự. Như vậy nó thực hiện nhiệm vụ ngược với bộ ADC.

*Bộ DAC trong LPC2378:*

- Bộ chuyển đổi số sang tương tự 10 bit
- Kiến trúc chuỗi điện trở
- Đầu ra được đệm
- Chế độ power down để tiết kiệm công suất
- Tốc độ và công suất có thể lựa chọn được

*Mô tả chân:*

Tên	Kiểu chân	Mô tả
AOUT	Output	Analog Output. Sau khi thời gian quá độ được lựa chọn sau khi DACR được viết một giá trị mới, điện áp trên chân này là $VALUE/1024 \cdot VREF$

VRFE	Reference	Voltage Reference. Chân này cung cấp điện áp tham khảo cho bộ chuyển đổi D/A
V <sub>DDA</sub> , V <sub>SSA</sub>	Power	Analog power and Ground. Những chân này trên danh nghĩa có cùng điện áp là V <sub>DD</sub> (3V3) và V <sub>SS</sub> , nhưng chúng phải được cách ly để tối thiểu nhiễu và lỗi.

**Lưu ý:** khi DAC không được sử dụng. V<sub>DD</sub> và V<sub>REF</sub> phải được kết nối tới nguồn cung cấp và chân V<sub>SSA</sub> phải được kết nối với đất. Những chân này không được để thả nổi.

#### Mô tả thanh ghi

Thanh ghi R/W chứa giá trị số để được chuyển đổi thành analog và một bit mà trao đổi giữa hiệu năng và công suất. Các bit 5:0 được dành trước cho tương lai, cho các bộ D/A có độ phân giải cao hơn.

Bit	Ký hiệu	Giá trị	Mô tả	Giá trị Reset
5:0	-		Dành trước. Các phần mềm phải không được viết các bit 1 vào các bit này. Giá trị đọc từ các bit dành trước không có ý nghĩa	NA
15:6	VALUE		Sau khi thời gian quá độ sau khi trường này được viết một giá trị mới, điện áp trên chân Aout là VALUE/1024xVREF	0
16	BIAS	0	Thời gian quá độ của DAC tối đa là 1μs, và dòng tối đa là 700 μA	0
		1	Thời gian quá độ của DAC tối đa là 2,5μs, và dòng tối đa là 350 μA	
31:17	-		Dành trước. Các phần mềm phải không được viết các bit 1 vào các bit này. Giá trị đọc từ các bit dành trước không có ý nghĩa	NA

Chương trình ví dụ sau thực hiện tạo 1 xung tam giác ở đầu ra bộ DAC. Tín hiệu thể hiện trên speaker của bộ kit MCB2300 là một âm đơn kéo dài.

```
#include "LPC23xx.h"

void DACInit( void )
{
    /* setup the related pin to DAC output */
    /* set p0.26 to DAC output */
    PINSEL1 = 0x00200000;
    return;
}

/*****
**   Main Function main()
*****/
```

```

int main (void)
{
    unsigned long i = 0;

    /* Initialize DAC */
    DACInit();
    while ( 1 )
    {
        DACR = (i << 6) | 0x00010000;
        i++;
        if ( i == 1024 )
        {
            i = 0;
        }
    }
    return 0;
}

```

### 5.3.11 Vòng khóa pha

PLL thực chất là một bộ nhân tần, với đầu vào PLL là tín hiệu dao động ngoài có tần số từ 32kHz đến 50MHz từ thạch anh, tín hiệu này được nhân lên và tạo tín hiệu đầu ra có tần số từ 275MHz cho đến 550 MHz, kết hợp cùng khối chia tần trong LPC2378 sẽ cung cấp tín hiệu xung clock cho lõi ARM7 và các ngoại vi. Sử dụng PLL cho phép LPC2378 hoạt động ở tần số tối đa ngay cả với bộ dao động ngoài với tần số thấp, giảm ảnh hưởng khi có tín hiệu cao tần trong mạch.

Tần số đầu ra của PLL có thể thay đổi được trong quá trình CPU đang hoạt động, cho phép tiết kiệm công suất trong trạng thái idle.

Trong PLL có 2 tham số xác định tần số đầu ra là N và M.

Giá trị tần số đầu ra được tính như sau:

$$F_{CCO} = (2 \times M \times F_{IN}) / N$$

Trong đó

- $F_{IN}$  – tần số tín hiệu đầu vào bộ PLL
- $F_{CCO}$  – tần số tín hiệu đầu ra PLL
- N – hệ số chia của PLL, lấy từ các bit NSEL trong thanh ghi PLLCFG của PLL (có giá trị bằng trường NSEL+1,  $N=1 \div 32$ ).
- M – hệ số nhân của PLL, lấy từ các bit MSEL trong thanh ghi PLLCFG của PLL (có giá trị bằng trường MSEL+1, chỉ có 1 số giá trị trong dải trường MSEL được phù hợp).
- Mô tả các thanh ghi I2C:

Register	Mô tả	Kiểu truy cập
PLLCON	PLL Control Register. Thanh ghi chứa các bit điều khiển việc cập nhật PLL. Các giá trị được ghi vào	R/W

	thanh ghi này sẽ không được áp dụng ngay tại đầu ra PLL mà phải chờ cho đến khi chuỗi giá trị trợ giúp PLL được sử dụng	
PLLSTAT	PLL Status Register. Thanh ghi này chứa thông tin về cấu hình và điều khiển PLL. Nếu PLLCON và PLLCFG được cập nhật giá trị mới nhưng chuỗi giá trị trợ giúp PLL chưa được sử dụng thì giá trị mới này sẽ không phản ánh trạng thái hiện tại của PLL. Đọc thanh ghi này sẽ cung cấp các giá trị thực đang điều khiển PLL cũng như trạng thái PLL	RO
PLLFEED	PLL Feed Register: thanh ghi này cho phép cập nhật các giá trị thông tin cấu hình và điều khiển PLL từ PLLCON và PLLCFG vào các thanh ghi ẩn mà thực sự đang điều khiển hoạt động của PLL	WO

Chuỗi trợ giúp PLL phải được ghi vào thanh ghi PLLFEED để thay cho các thay đổi trong PLLCON và PLLCFG được cập nhật thực sự vào các thanh ghi đang điều khiển PLL. Chuỗi này gồm:

- (1) Ghi giá trị 0xAA vào PLLFEED
- (2) Ghi giá trị 0x55 vào PLLFEED

Ví dụ dưới giải thích cách cấu hình PLL cho PLL

```
void init_PLL(void)
{
    // Set multiplier and divider of PLL
    PLLCFG = 0x00000024;

    // Enable the PLL
    PLLCON = 0x00000001;

    // Update PLL registers with feed sequence
    PLLFEED = 0x000000AA;
    PLLFEED = 0x00000055;

    // test Lock bit
    while (!(PLLSTAT & 0x00000400));

    // Connect the PLL
    PLLCON = 0x00000003;

    // Update PLL registers
    PLLFEED = 0x000000AA;
    PLLFEED = 0x00000055;
}
```

## 5.4 Thiết lập hệ điều hành nhúng trên nền ARM

### 5.4.1 Firmware và Bootloader

Firmware thường là phần mã nguồn được chạy đầu tiên trong một hệ thống nhúng khi khởi động. Do đó, nó là một trong những thành phần quan trọng nhất. Firmware rất đa dạng, có thể chỉ là một đoạn code khởi động hoặc cả một phần mềm nhúng.

Có nhiều định nghĩa về firmware, chúng ta dùng định nghĩa sau:

Firmware là phần mềm nhúng cấp thấp cung cấp giao diện giữa phần cứng và các phần mềm ứng dụng hoặc hệ điều hành. Firmware thường được lưu trên ROM và chạy khi hệ thống nhúng được khởi động.

Đối với một hệ thống có hệ điều hành, khi khởi động lên, hệ thống cần chạy một chương trình từ ROM để nạp hệ điều hành và dữ liệu để sau đó có thể chạy trên RAM. Do đó, người ta thường dùng định nghĩa sau:

Bootloader là một ứng dụng để nạp hệ điều hành hoặc ứng dụng của một hệ thống. Bootloader thường chỉ tồn tại tới thời điểm hệ điều hành hoặc ứng dụng chạy. Bootloader thường được tích hợp cùng với firmware.

Chu trình chạy thường gặp của một hệ thống nhúng như sau

- Bước đầu tiên là thiết lập nền tảng của hệ thống (hay chuẩn bị môi trường để khởi động một hệ điều hành).
  - Bước này bao gồm đảm bảo hệ thống được khởi tạo đúng (VD như thay đổi bản đồ bộ nhớ hoặc thiết lập các giá trị thanh ghi..).
  - Sau đó, firmware sẽ xác định chính xác nhân chip và hệ thống. Thường thì thanh ghi số 0 của bộ đồng xử lý lưu loại VXL và tên nhà sx.
  - Tiếp theo, phần mềm chuẩn đoán hệ thống sẽ xác định xem có vấn đề gì với các phần cứng cơ bản không.
  - Thiết lập giao diện sửa lỗi: khả năng sửa lỗi hỗ trợ cho việc sửa lỗi phần mềm chạy trên phần cứng. Các sự hỗ trợ này bao gồm
    - Thiết lập breakpoint trong RAM.
    - Liệt kê và sửa đổi bộ nhớ.
    - Hiện thị nội dung của các thanh ghi.
    - Disassemble nội dung bộ nhớ thành các lệnh ARM và Thumb.
  - Thông dịch dòng lệnh (Command Line Interpreter hoặc CLI) : Tính năng thông dịch dòng lệnh cho phép người sử dụng thay đổi hệ điều hành được khởi động bằng cách thay đổi cấu hình mặc định thông qua các lệnh tại command prompt. Đối với hệ điều hành nhúng, CLI được điều khiển từ một ứng dụng chạy trên máy host. Việc liên lạc giữa host và target thường qua cáp nối tiếp hoặc kết nối mạng
- Bước thứ hai là trừu tượng hóa phần cứng. Lớp trừu tượng hóa phần cứng (HAL) là một lớp phần mềm giấu chi tiết về phần cứng phía dưới bằng cách cung cấp một tập hợp giao diện lập trình đã được định sẵn trước. Khi chuyển sang một nền tảng khác, các giao diện lập trình này được giữ nguyên nhưng phần cứng phía dưới có thể thay đổi. Đối với mỗi một phần cứng cụ thể, phần mềm HAL để giao tiếp với phần cứng đó gọi là trình điều khiển thiết bị (device driver). Trình điều khiển thiết bị cung cấp giao diện lập trình ứng dụng (API) chuẩn để đọc và ghi tới một thiết bị riêng.
- Bước thứ ba là nạp một ảnh khởi động được (bootable image). File này có thể lưu ở trong ROM, network hoặc thiết bị lưu trữ khác. Đối với chip ARM, format phổ biến nhất cho image file là Executable and Linking Format (ELF).



- Bước thứ tư là từ bỏ quyền điều khiển. Đến đây khi hệ điều hành đã bắt đầu được nạp lên và chạy, firmware sẽ chuyển quyền điều khiển hệ thống cho hệ điều hành. Trong một số trường hợp đặc biệt, firmware vẫn có thể giữ quyền điều khiển hệ thống.
  - Trong hệ thống sử dụng chip ARM, trao quyền điều khiển có nghĩa là cập nhật bảng vector và thay đổi thanh ghi PC. Thay đổi bảng vector sẽ làm chúng chỉ đến các hàm điều khiển các ngắt và các ngoại lệ. Thanh ghi PC được cập nhật để chỉ đến địa chỉ ban đầu của hệ điều hành.

### 5.4.2 Hệ thống file (Filesystem)

Hệ thống file là cách thức để lưu trữ và tổ chức các file và dữ liệu trên máy tính.

Khác với các hệ thống lưu trữ trên máy tính hay máy chủ, các hệ thống nhúng thường sử dụng các thiết bị lưu trữ thể rắn như flash memory, flash disk. Các thiết bị này phải được thiết lập cấu hình hệ thống file hoàn chỉnh cho hệ thống.

Có nhiều loại hệ thống file khác nhau, sau đây là một số hệ thống file phổ biến cho hệ thống nhúng:

#### **ROMFS (ROM file system)**

ROMFS là hệ thống file đơn giản nhất, lưu các dữ liệu có kích thước nhỏ, chỉ đọc được. Thường các dữ liệu này phục vụ quá trình khởi tạo RAM disk.

#### **RAMdisk**

Ramdisk (còn gọi là ổ nhớ RAM ảo hoặc ổ nhớ RAM mềm) là một ổ đĩa ảo được thiết lập từ một khối RAM. Hệ thống máy tính sẽ làm việc với khối RAM này như một ổ đĩa.

Hiệu năng của RAMdisk thường cao hơn các dạng lưu trữ khác nhiều, tuy nhiên các dữ liệu lưu trên RAM disk sẽ bị mất khi mất nguồn.

#### **CRAMFS (Compressed RAM file system)**

CRAMFS là một hệ thống file tiện dụng cho các hệ thống lưu trữ thể rắn. Đây là một hệ thống file chỉ đọc ra, và có khả năng lưu dữ liệu lưu trên hệ thống có dạng nén. CRAMFS dùng thư viện zlib để nén dữ liệu.

Công cụ làm việc với hệ thống file này là `mkcramfs`.

#### **Journaling Flash File System (JFFS và JFFS2)**

JFFS là hệ thống file cổ điển cho hệ thống nhúng. JFFS hỗ trợ bộ nhớ flash NOR. Phiên bản cập nhật của JFFS là JFFS2 có thêm nhiều tính năng cải tiến, hỗ trợ bộ nhớ flash NAND. Đồng thời, JFFS2 cũng hỗ trợ nén với một trong ba thuật toán : zlib, rubin và rtime.

### 5.4.3 Thiết lập nhân (kernel)

Nhân của hệ điều hành là thành phần phần mềm trung tâm của hệ thống nhúng. Khả năng của nhân cũng là chỉ số của khả năng của cả hệ thống.

Kiến trúc, cấu trúc của nhân hệ điều hành nhúng tùy theo từng hệ điều hành có thể từ đơn giản đến phức tạp. Các tài liệu về kiến trúc, cấu trúc, lập trình, sử dụng nhân rất phong phú và phổ biến. Do phạm vi vượt quá nội dung liên quan, ở đây chúng ta chỉ đề cập đến vấn đề chuẩn bị một nhân cho hệ điều hành nhúng. Các ví dụ được lấy là nhân Linux.

Cụ thể, chúng ta sẽ đề cập đến việc lựa chọn nhân, cấu hình, dịch và cài đặt.

## Lựa chọn nhân

Hiện nay có nhiều hệ điều hành nhúng khác nhau, mỗi hệ điều hành nhúng có nhiều phiên bản khác nhau. Trên thực tế, nhiều phiên bản chỉ dành riêng cho một số bo mạch nhúng, một số phiên bản có thể hoạt động trên nhiều bo mạch nhúng, nhưng cần được sửa đổi phù hợp lại với cấu hình.

Đối với hệ điều hành Linux, các phiên bản nhân cho chip ARM được lưu ở trang web <http://www.arm.linux.org.uk/developer/>. Ngoài ra, các phiên bản có thể được lưu ở các trang mã nguồn khác, hoặc đi kèm theo kit phát triển.

Sau khi download hoặc copy phiên bản nhân dự định sử dụng. Ta có thể cần phải dùng các bản vá cho nhân (patch). Các bản vá này có thể sửa chữa các lỗi hoặc thay đổi một số tính năng trong nhân.

## Cấu hình nhân

Thiết lập cấu hình nhân là bước đầu tiên để xây dựng nhân cho bo mạch. Có nhiều cách để thiết lập cấu hình nhân, trong quá trình thiết lập cấu hình cũng có nhiều lựa chọn khác nhau. Tuy nhiên, không phụ thuộc vào cách dùng hoặc cách lựa chọn các options, nhân sẽ tạo ra một file `.config` và tạo ra các liên kết symbolic và các file headers sẽ được dùng trong quá trình còn lại của việc xây dựng nhân.

**Các lựa chọn:** Có rất nhiều lựa chọn khác nhau, các lựa chọn này sẽ được dùng để thiết lập nên nhân. Tùy theo yêu cầu của bo mạch và chip mà ta lựa chọn cấu hình thích hợp. Ở đây ta chỉ liệt kê một số lựa chọn chính:

- Code maturity level options

- Loadable module support

- General setup

- Memory technology devices

- Block devices

- Networking options

- ATA/IDE/MFM/RLL support

- SCSI support

- Network device support

- Input core support

Character devices

Filesystems

Console drivers

Sound

Kernel hacking

**Các cách thiết lập cấu hình:** có 4 cách chính

*make config*

Sử dụng giao diện dòng lệnh (command-line interface) cho lần lượt từng lựa chọn

*make oldconfig*

Sử dụng cấu hình trong một file *.config* sẵn có và chỉ hỏi những lựa chọn chưa được thiết lập.

*make menuconfig*

Thiết lập cấu hình dùng giao diện thực đơn trên terminal.

*make xconfig*

Thiết lập cấu hình dùng giao diện X Window

Để xem thực đơn cấu hình nhân, có thể dùng dòng lệnh. VD đối với dòng chip ARM:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- menuconfig
```

Sau đó, ta có thể chọn các tùy chọn cấu hình tương ứng cho bo mạch. Nhiều tính năng và trình điều khiển thường có sẵn theo dạng các module và có thể được chọn để tích hợp trong nhân hoặc kèm theo dạng module.

Sau khi hoàn thành việc cấu hình nhân, thoát và lưu cấu hình nhân. File cấu hình sẽ được lưu lại trong một file *.config*.

## **Biên dịch nhân**

Biên dịch nhân bao gồm ba bước chính sau: Xây dựng các file phụ thuộc, xây dựng ảnh nhân (kernel image), xây dựng các module nhân. Mỗi bước đều dùng các lệnh *make* và được giải thích dưới đây. Tuy nhiên, các bước này có thể được làm gộp bằng cách dùng một lệnh duy nhất.

### **Xây dựng các file phụ thuộc:**

Phần lớn các files trong mã nguồn của nhân phụ thuộc vào nhiều file header. Để xây dựng nhân, file Makefiles cần biết tất cả các phụ thuộc này. Đối với mỗi thư mục trong cây thư mục nhân, một file *.depend* được tạo trong quá trình xây dựng các phụ thuộc. File này chứa danh mục các header file mà các file trong thư mục phụ thuộc vào.

Từ thư mục gốc (root directory) của mã nguồn nhân, để xây dựng các phụ thuộc của nhân dùng lệnh sau:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- clean dep
```

Ở đây ARCH (kiến trúc) là nhân ARM. CROSS\_COMPILE là biên dịch chéo, dùng trong việc biên dịch trên kiến trúc khác với ARM (VD kiến trúc x86). CROSS\_COMPILE sẽ được dùng để lựa chọn công cụ xây dựng kernel. Trong trường hợp chạy trên Linux, trình biên dịch là *gcc*, do đó nếu bo mạch đích sử dụng chip ARM, trình biên dịch sẽ là *arm-linux-gcc*.

### **Xây dựng file ảnh nhân (kernel image):**

File ảnh của nhân được tạo ra bằng lệnh sau:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- zImage
```

Ở đây, file đích là *zImage*, có nghĩa là ảnh của nhân được nén dùng thuật toán gzip. Ngoài thuật toán này ra có thể dùng *vmlinux* để tạo một ảnh không nén.

### **Xây dựng các module nhân:**

Sau khi ảnh của nhân đã được xây dựng xong, các module nhân có thể được xây dựng bằng dòng lệnh sau:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- modules
```

Tùy theo các tùy chọn nhân đã được lựa chọn xây dựng theo dạng module trong quá trình thiết lập cấu hình nhân (thay vì tích hợp trong nhân), thời gian cho quá trình này có thể dài ngắn khác nhau.

Sau khi ảnh của nhân và các module của nhân đã được xây dựng, ta có thể cài vào trong bo mạch đích. Trước khi cài, chúng ta cần backup file cấu hình nhân và dọn sạch mã nguồn của nhân bằng câu lệnh sau:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- distclean
```

### **Cài đặt nhân**

Bước cuối cùng là cài đặt nhân vào bo mạch đích. Đối với mỗi cấu hình nhân, ta cần copy bốn file: file ảnh nhân đã nén, file ảnh nhân không nén, bản đồ symbol nhân và file cấu hình. File ảnh nhân đã nén ở trong thư mục *arch/arm/boot/zImage*, ba file còn lại lần lượt là *vmlinux*, *System.map*, và *.config*.

Để dễ dàng nhận dạng các file, chúng ta cần đặt tên file theo phiên bản mã nhân. Ví dụ nếu nhân được tạo từ mã nguồn 2.4.18-rmk5, chúng ta copy các file như sau:

```
$ cp arch/arm/boot/zImage ${PRJROOT}/images/zImage-2.4.18-rmk5
$ cp System.map ${PRJROOT}/images/System.map-2.4.18-rmk5
$ cp vmlinux ${PRJROOT}/images/vmlinux-2.4.18-rmk5
$ cp .config ${PRJROOT}/images/2.4.18-rmk5.config
```

File Makefile của nhân kèm theo một file *modules\_install* cho việc cài đặt các module. Theo mặc định, các module được cài trong thư mục */lib/modules*. Bởi vì các module nhân được dùng với ảnh của nhân tương ứng, các module này nên được cài trong thư mục với tên tương tự như ảnh của nhân. Trong ví dụ trên nhân sử dụng là 2.4.18-rmk5 được cài trong thư mục *\${PRJROOT}/images/modules-2.4.18-rmk5*. Toàn bộ nội dung của thư mục này sẽ được copy vào hệ thống file gốc của bo mạch đích để dùng với nhân tương ứng, do đó để cài các module ta dùng lệnh:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- \
> INSTALL_MOD_PATH=${PRJROOT}/images/modules-2.4.18-rmk5 \
> modules_install
```

Sau khi đã hoàn thành việc copy các modules, nhân sẽ thực hiện việc xây dựng sự phụ thuộc của các module cần cho các tiện ích trong quá trình chạy thực. Để làm việc này, chúng ta cần công cụ xây dựng đi kèm với gói phần mềm BusyBox. Download BusyBox về, copy vào thư mục *\${PRJROOT}/sysapps* và giải nén tại đây. Từ thư mục BusyBox, copy file Perl script *scripts/depmod.pl* sang thư mục *\${PREFIX}/bin*.

Ta dùng lệnh sau để xây dựng các phụ thuộc module cho bảng mạch đích:

```
$ depmod.pl \
> -k ./vmlinux -F ./System.map \
> -b ${PRJROOT}/images/modules-2.4.18-rmk5/lib/modules > \
> ${PRJROOT}/images/modules-2.4.18-rmk5/lib/modules/2.4.18-
rmk5/modules.dep
```

Ở trên đây tùy chọn *-k* để xác định ảnh của nhân là không nén, *-F* dùng để xác định bản đồ hệ thống, *-b* dùng để xác định thư mục căn bản.

## Tài liệu tham khảo

- [1]. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*, Tammy Noergaard, Newnes, 2005.
- [2]. *Embedded Systems Design*, Steve Heath,,Second Edition, Newnes, 2002 .
- [3]. *Embedded Systems- Architecture, Programming and Design*, Raj Kamal, McGraw Hill, 2003
- [4]. *Embedded Microprocessor system*,2<sup>nd</sup> ed., Stuart R. Ball, Newnes, 2001.
- [5]. *Embedded\_Microcomputer Systems: Real Time Interfacing*, 2nd Edition, ISBN 0534551629, Thomson 2006, by J. W. Valvano.
- [6]. *Embedded System Design: A unified Hardware/Software Introduction*, Vahid/Givargis, John Wiley & Sons INC, 2002.
- [7].*Co-Synthesis of Hardware and Software for Digital Embedded Systems*, G.D. Micheli, Illinois University at Urbana Champaign, 2000.
- [8].*Co-Synthesis of Hardware and Software for Digital Embedded Systems*, G.D. Micheli, Illinois University at Urbana Champaign, 2000.
- [9].*Embedded Linux System design and development*, TP. Raghavan , Amol Lad ,Sriram Neelakandan, Auerbach Publications, 2006.
- [10]. *The Insider's Guide To The Philips ARM®7 Based Microcontrollers*, Hitek, 2008.
- [11]. *Embedded Linux Primer: A Practical, Real-World Approach*, Christopher Hallinan, Prentice Hall, 2006.
- [12]. *Building Embedded Linux Systems*, Karim Yaghmour, O'Reilly, 2003