HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

NGUYỄN NGỌC MINH
NGUYỄN TRUNG HIẾU

BÀI GIẢNG
HỆ THỐNG NHÚNG

HÀ NỘI – 12.2014

Mục lục

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 kĩ 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:

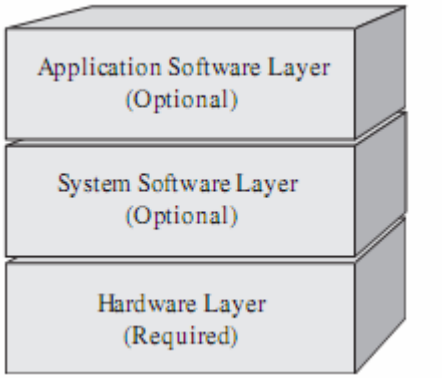
- Cấu trúc theo dạng module: Theo dạng này, các phần tử là các thành phần có chức năng khác nhau của hệ thống (VD như các phần cứng hoặc phần mềm căn bản để cho hệ thống hoạt động được) trong một thiết bị nhúng. Cấu trúc này

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 cần 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 nhúng 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 kĩ 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 nhão (firmware).

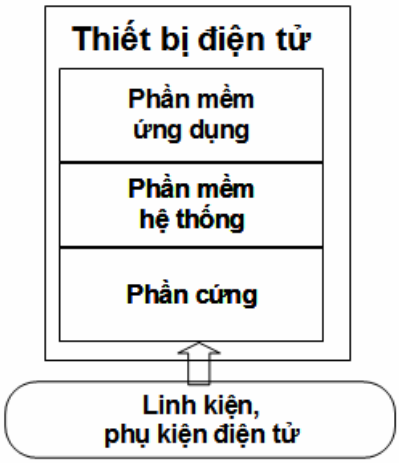
Cách thiết kế cổ điển là các chức năng phần mềm (SW) và phần cứng (HW) được xác định trước rồi sau đó các bước thiết kế chi tiết được tiến hành một cách độc lập ở hai khối . Hiện nay đa số các hệ thống tự động hoá thiết kế (CAD) thường dành cho thiết kế phần cứng. Các hệ thống nhúng sử dụng đồng thời 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 kĩ 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 kĩ năng đã đặt ra. Tập các kĩ 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 vậy lý.Ví dụ như với một PC, phần cứng gồm các thành phần bo mạch chủ, Ram, ở 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 kĩ 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 kết 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 kĩ 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 kĩ 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.

- Kĩ 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 kĩ năng trong thiết kế, phát triển phần mềm ứng dụng là không thể thiếu đối với thiết kế hệ thống nhúng hiện nay. Các kĩ 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...
- Kĩ năng thực thi các ứng trên các ngôn ngữ lập trình : yêu cầu có kĩ 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 kĩ 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 kĩ thuật ứng dụng trong kiến trúc máy tính như pipeline, DMA, caching, buffering,...
- Kĩ 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 kĩ thuật lập trình tiên tiến trong các phần mềm... nên đây cũng là một kĩ 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 kĩ năng yêu cầu bắt buộc đối với các sinh viên điện tử viễn thông. Nó gồm các kĩ 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 mĩ. 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

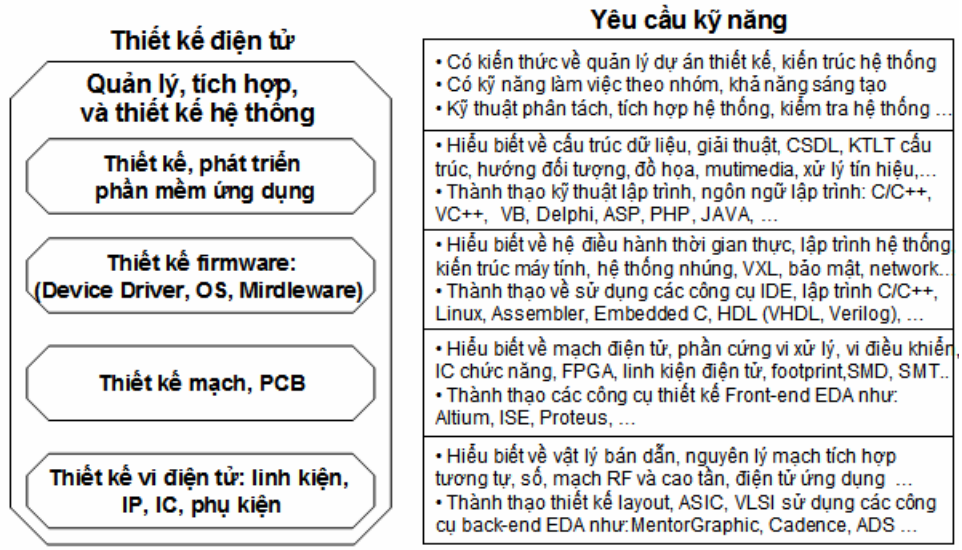
Vi điện tử là các vấn đề liên quan đến nghiên cứu, chế tạo các linh kiện điện tử. Những linh kiện này làm từ các chất bán dẫn. Chúng là những thành phần cơ bản trong các thiết kế điện tử như transistor, tụ điện, điện trở, diode, IC, ...

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 kĩ 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 kĩ năng yêu cầu đối với từng mặt trong thiết kế hệ thống nhúng như trong bảng 1-1.



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

Câu hỏi ôn tập

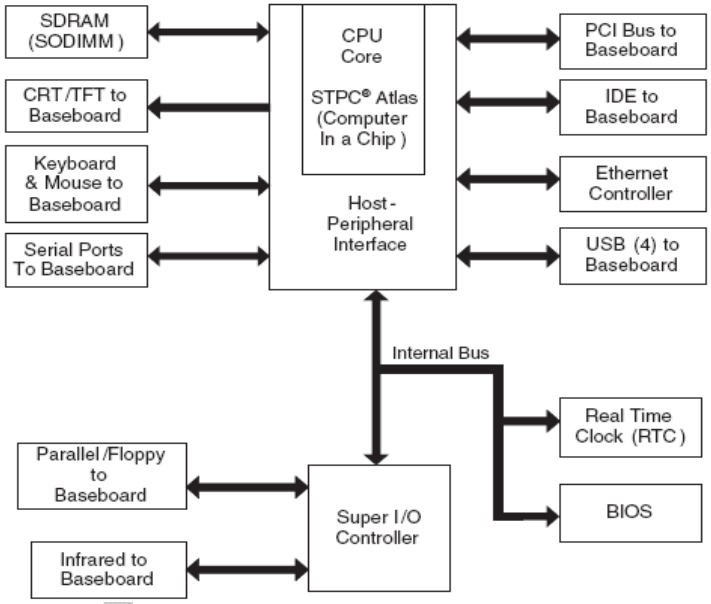
1. Lấy ví dụ 3 thị trường sử dụng hệ thống nhúng
   Mỗi thị trường lấy 5 ví dụ thiết bị.
2. Các đặc tính của hệ thống nhúng bao gồm những đặc tính nào
3. Nêu các thành phần căn bản của kiến trúc hệ thống nhúng
4. Các kiến trúc CPU nhúng căn bản bao gồm những kiến trúc nào
5. Liệt kê các giao tiếp sử 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 lí 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 chi tiết hơn dưới 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.

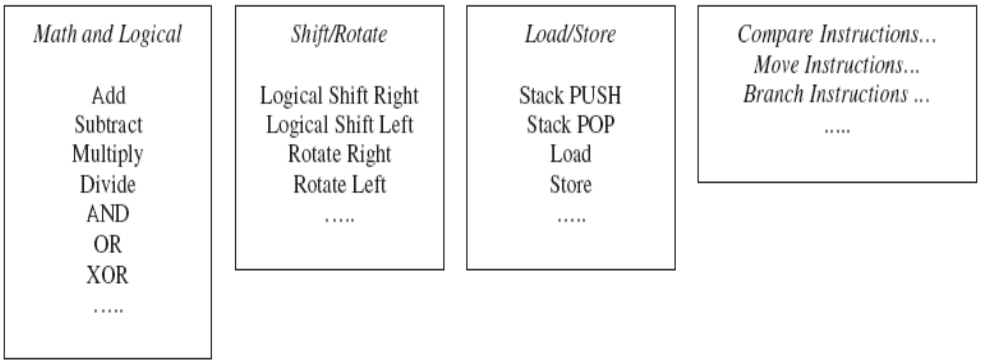
Các thao tác được tạo ra từ một hay nhiều chỉ lệnh. Các bộ xử lý khác nhau có thể thực thi các thao tác tương tự nhau nhờ sử dụng một số lượng và kiểu chỉ lệnh khác nhau.

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).



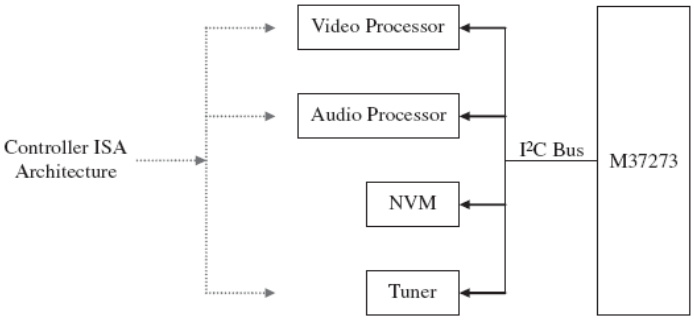
Hình 2-2: Các thao tác ISA đơn giản

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 nhúng 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 nhúng, các mô hình phổ biến nhất là:

Mô hình điều khiển – Controller model

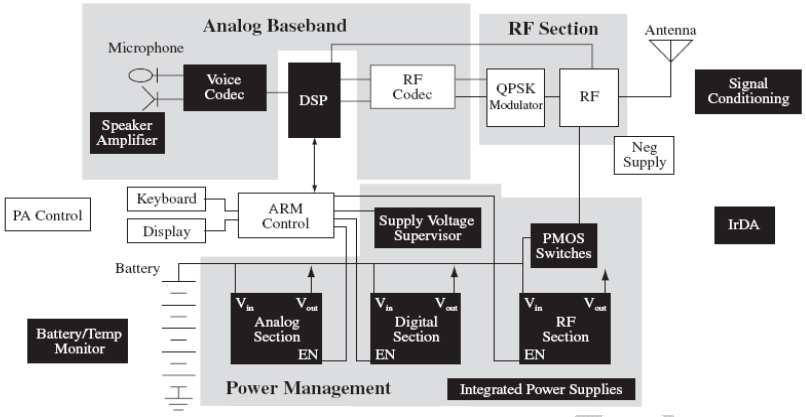
Các ISA controller được thực hiện trong các bộ xử lý mà không yêu cầu các thao tác dữ liệu phức tạp. Ví dụ như bộ xử lý audio và video được sử dụng như các slave processors trong bảng mạch TV, xem Hình 2-3.



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.



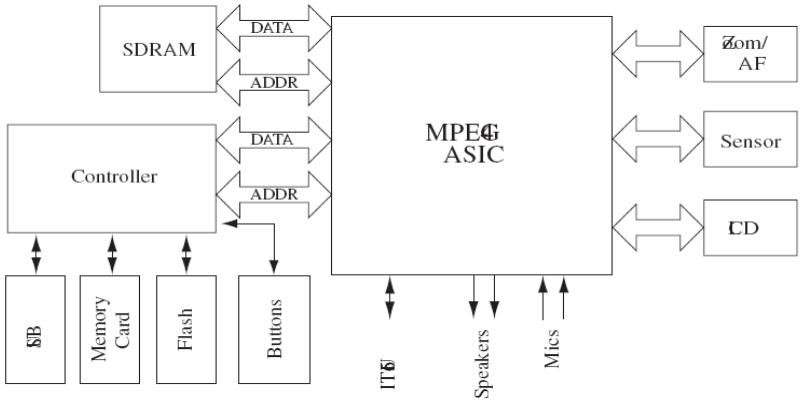
Hình 2-4: Board ví dụ với điện thoại di động thực hiện kỹ thuật số ISA đường dữ liệu

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:

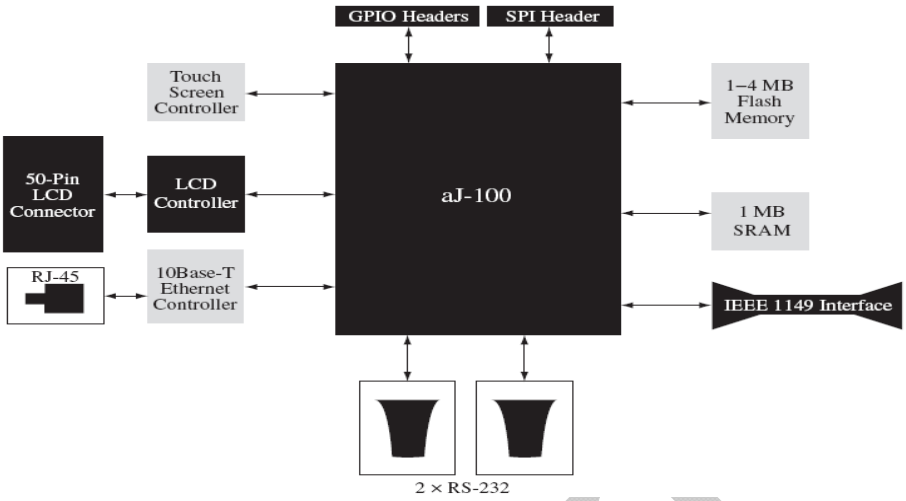
Mạch tích hợp ASICs thể hiện trên hình 2-5, các thiết bị logic khả trình (PLD), FPGA



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.



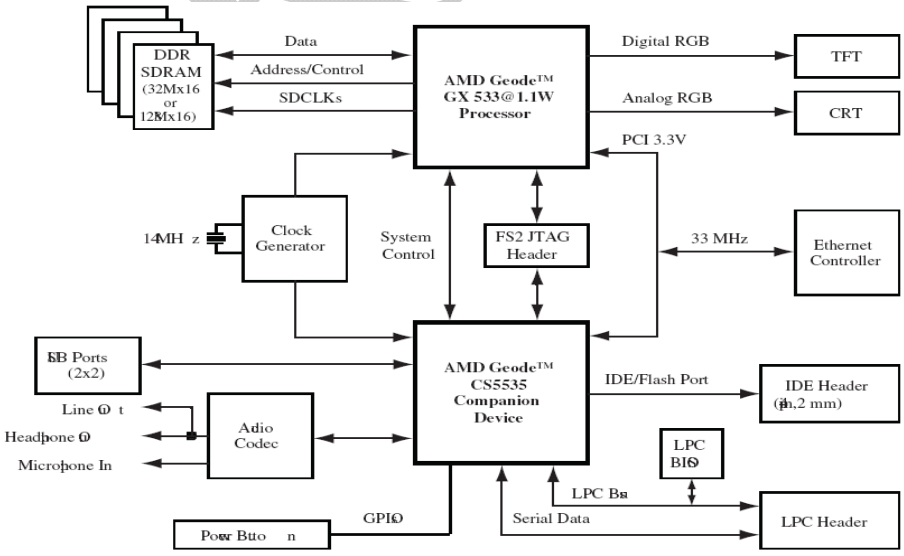
Hình 2-6: ví dụ thực hiện JVM ISA

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 nhúng. Các dạng phổ biến nhất của General-Purpose ISA models trong các bộ xử lý nhúng là:

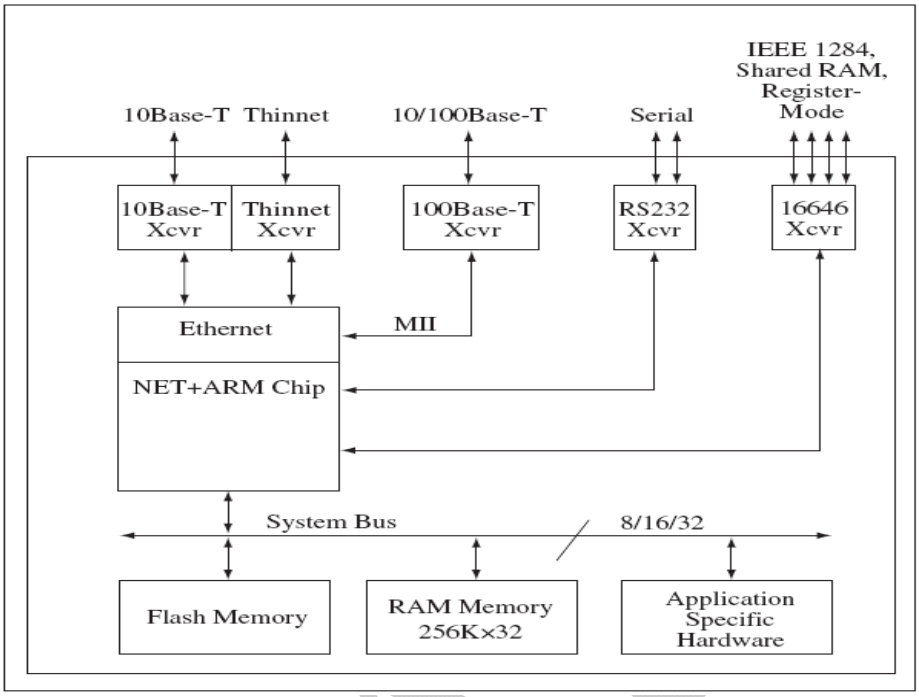
Mô hình tính toán với tập lệnh phức tạp (Complex Instruction Set Computing: CISC)

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 kì hoạt động.

- Các bộ xử lý RISC chỉ có một chu kì hoạt động trong khi CISC thường có nhiều chu kì.

- 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 gianh 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.

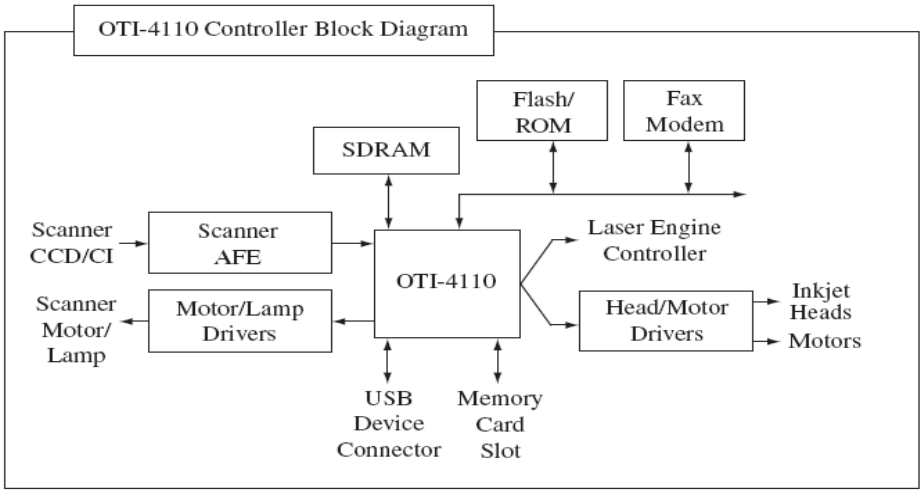
2.1.1.4 Các mô hình ISA song song mức lệnh (Instruction-Level Parallelism ISA Models)

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 kì 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)

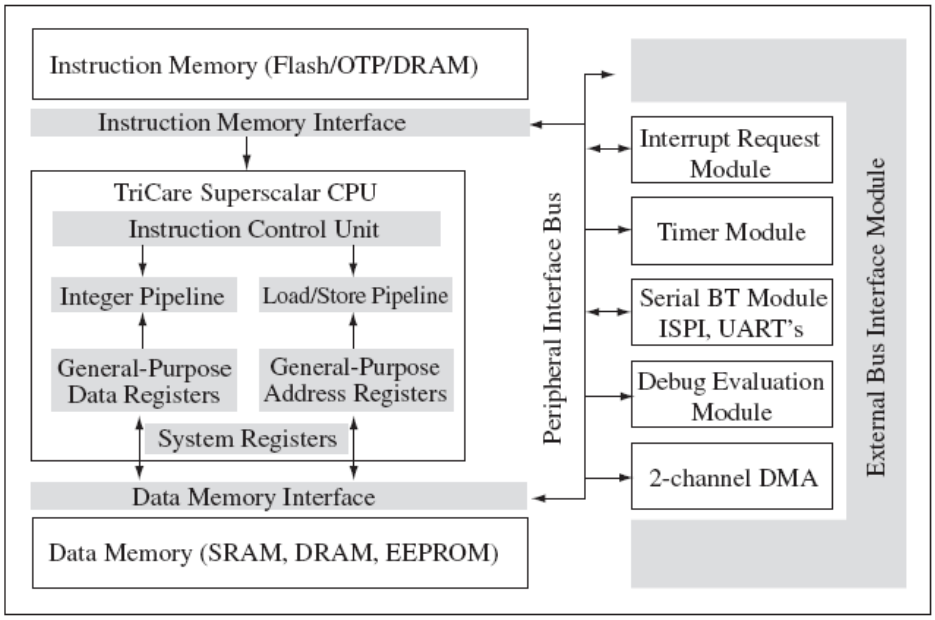
SIMD Machine ISA được thiết kế để xử lý một lệnh đồng thời trên nhiều phần dữ liệu.



Hình 2-9: ví dụ thực hiện SIMD ISA

Mô hình máy siêu vô hướng (Superscalar Machine Model)

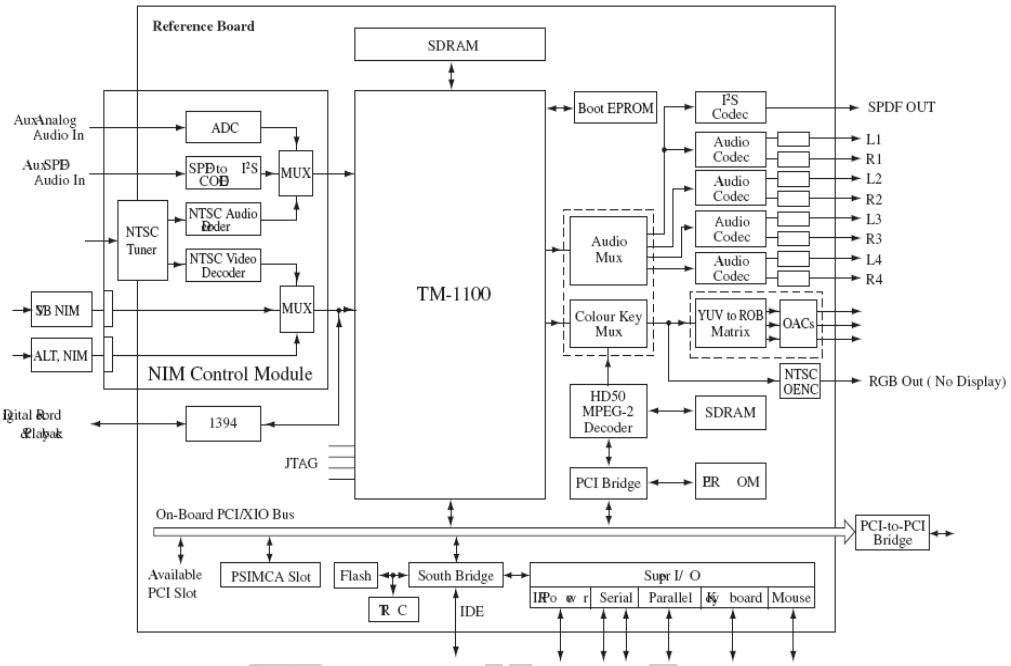
Superscalar ISA có thể thực hiện nhiều lệnh đồng thời trong một chu kì xung nhịp thông qua việc thực hiện nhiều hàm chức năng trong xử lý



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 xuất 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 kì xung nhịp, là nghịch đảo của tốc độ xung nhịp, được gọi là chu kì của bộ xử lý. Tốc độ và chu kì xung nhịp của bộ xử lý thường có trong tài liệu kĩ thuật của bộ xử lý.

Nhìn vào các lệnh, chỉ số CPI (trung bình số chu kì 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 per instruction * 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 nhúng 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.

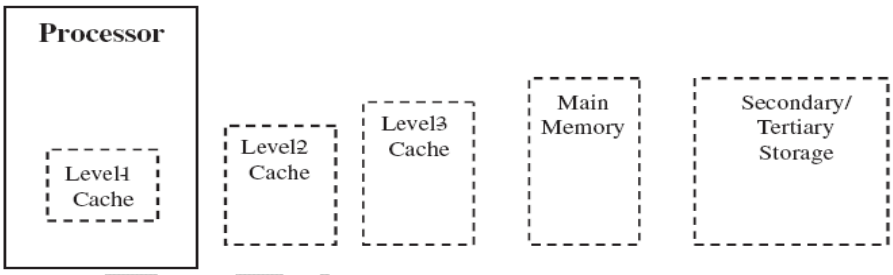
Datasheet của bộ xử lý cung cấp các thành phần chính của bộ xử lý và các thông tin để sử dụng bộ xử lý.

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 kĩ 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

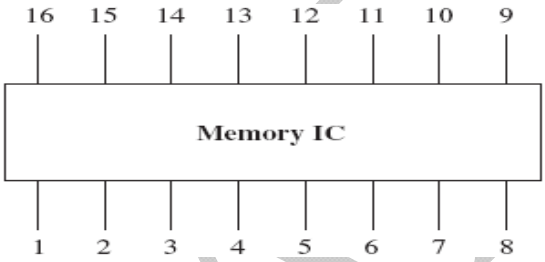
Thông thường, một IC nhớ gồm 3 bộ phận: mảng nhớ, bộ giải mã địa chỉ, và giao diện dữ liệu. Mảng nhớ thực tế là bộ nhớ vật lý để lưu trữ các bit dữ liệu. Trong khi bộ xử lý chủ, và các hệ lập trình, thiết lập bộ nhớ như mảng một chiều, mỗi ô trong mảng là một hàng các byte và số bit trên một hàng có 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.

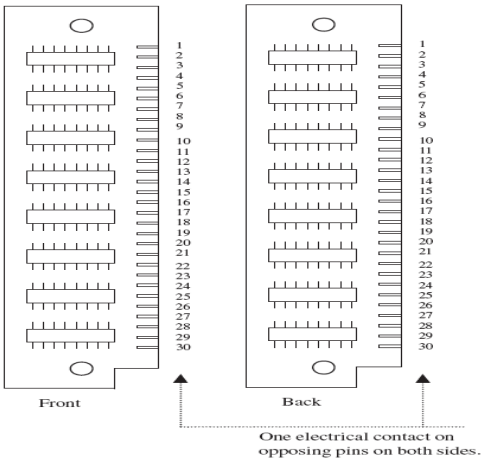
Các IC nhớ có thể kết nối tới bản mạch của nhiều loại gói khác nhau, phụ thuộc vào từng loại bộ nhớ. Các loại gói bao gồm loại vỏ 2 hàng chân (DIP), môđun nhớ từng dòng đơn lẻ (SIMM), và môđun nhớ 2 dòng (DIMM). Như chỉ dẫn ở Hình 2-16a, DIP là các gói bao quanh IC, được làm từ gốm hoặc chất dẻo. Số 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 mođun (256KB, 1MB, …) Ví dụ, 256K x 8 SIMM là một môđun yêu cầu 256K (256*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...

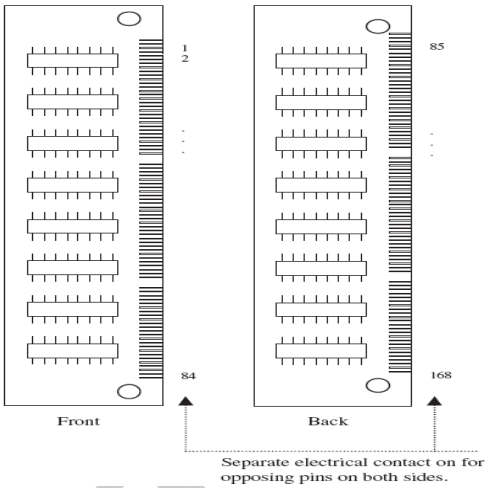


Hình 2-13a: Ví dụ DIP

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 x32) sẽ thay thế cho 4 SIMM 30 chân (256K x8) 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



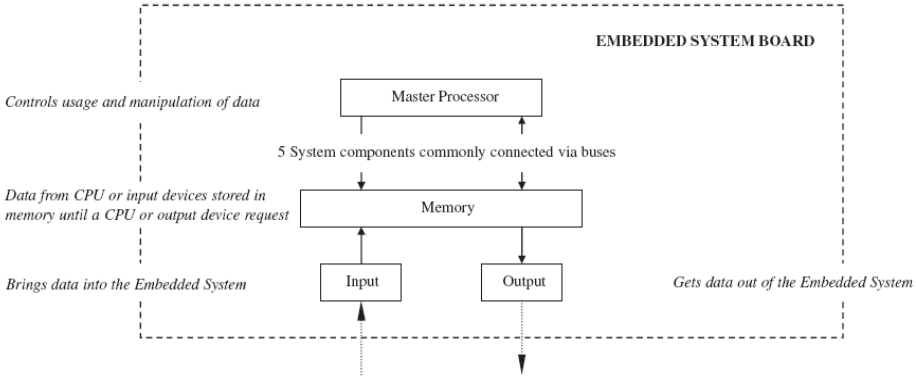
Hình 2-13c: Ví dụ DIMM 168 chân

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 phần 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 khiểu từ xa, hoặc có thể định vị ở trên chính bảng mạch nhúng ví dụ như đèn LED.



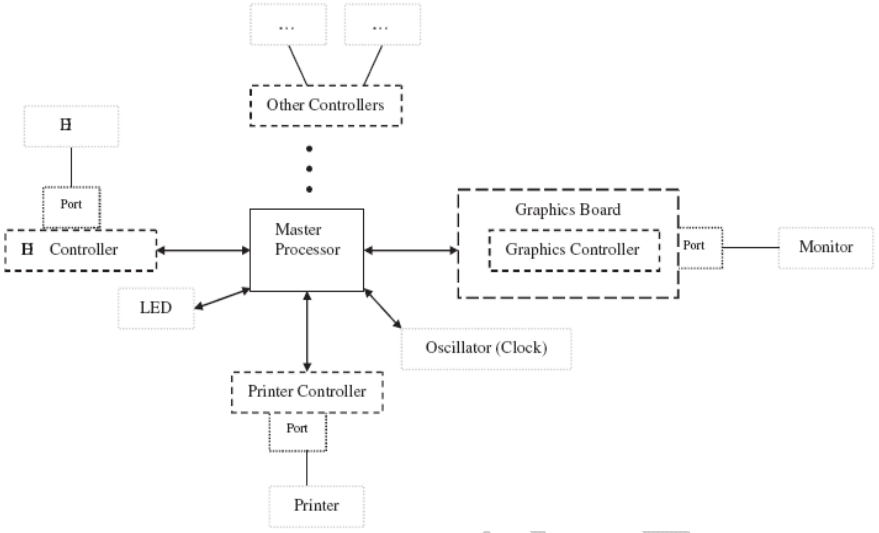
Hình 2-14: Sơ dồ khối I/O cơ bản của kiến trúc Von Newman

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…)

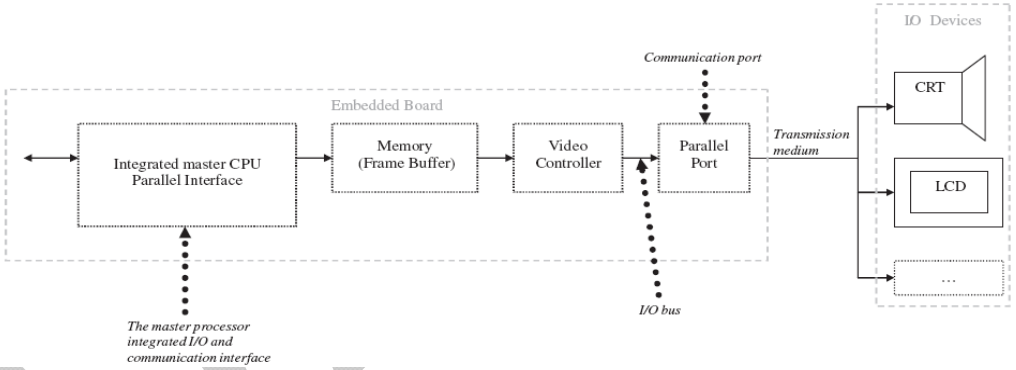
Ban đầu, bảng mạch vào/ra rất đơn giản chủ yếu là mạch điện kết nối với bộ xử lý chính, cổng vào/ra của bộ xử lý chính như là cổng xung nhịp hay đèn LED được định thời trên bảng. Phần cứng của I/O điển hình được làm từ tất cả hoặc một số kết hợp của 6 bộ logic chính sau:

- 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.

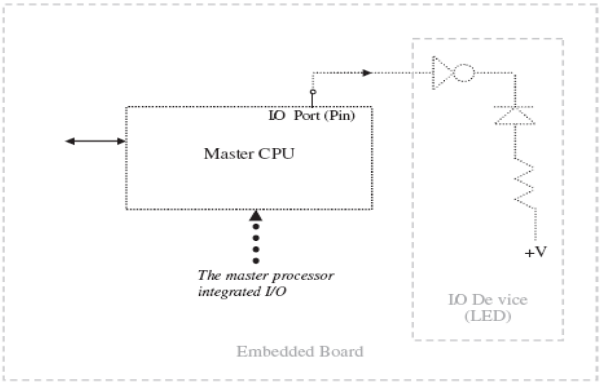


Hình 2-15: Các cổng và bộ điều khiển thiết bị điều khiển trên một bảng mạch nhúng

Bảng mạch vào/ra có thể giới hạn từ thành phần hệ thống phức 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 phân chính sau khi thiết kế hệ thống vào ra chính là đường bao quanh thiết bị vào/ra - giới hạn mục đích -chất lượng.

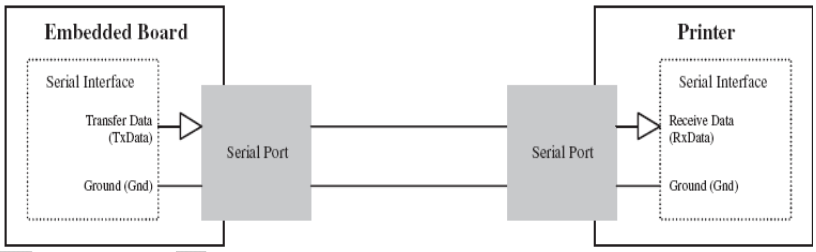
2.1.3.1 Quản lý dữ liệu nối tiếp

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.

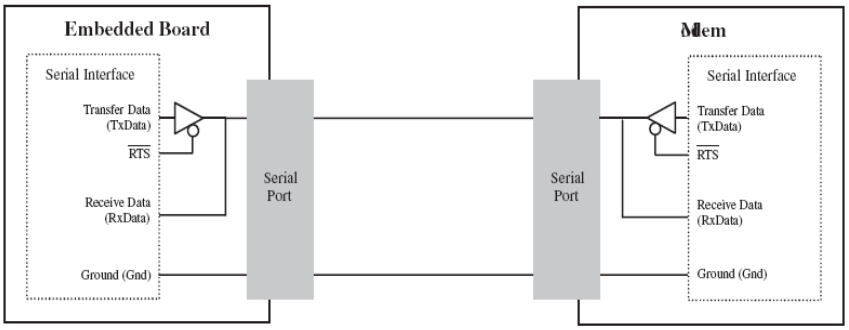
Giao diện nối tiếp là chuỗi dữ liệu được truyền và nhận giữa CPU chính và một số thiết bị I/O hoặc bộ điều khiển của nó. Chúng bao gồm nhận và truyền dữ liệu từ bộ đệm tới bộ lưu trữ và giải mã hoặc mã hoá dữ liệu và chúng có trách nhiệm truyền dữ liệu tới CPU chính khác hoặc thiết bị vào ra khác. Như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

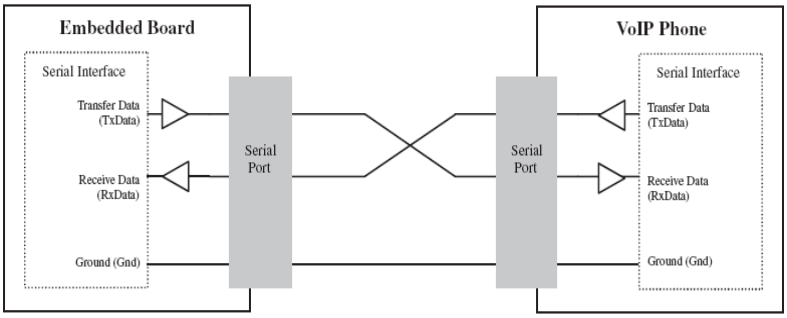
Sơ đồ bán song công mô tả chuỗi dữ liệu có thể truyền hoặc nhận ở một bộ vi xử lý khác nhưng 1 bộ xử lý chỉ truyền 1 lần (Hình 2-17b). Sơ đồ chuỗi song công mô tả chuỗi dữ liệu có thể truyền hoặc nhận ở bộ vi xử lý khác tương thích (Hình 2-17c)



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



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

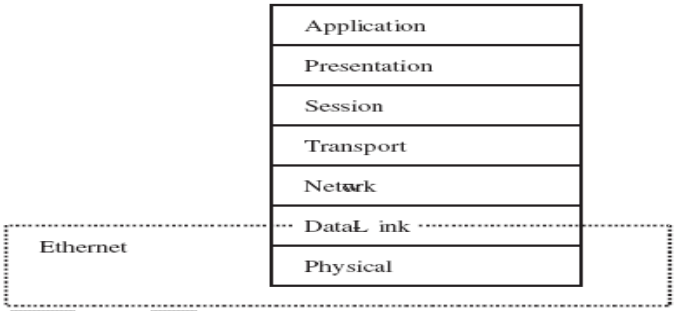


Hình 2-17c: Ví dụ sơ đồ truyền song công

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.

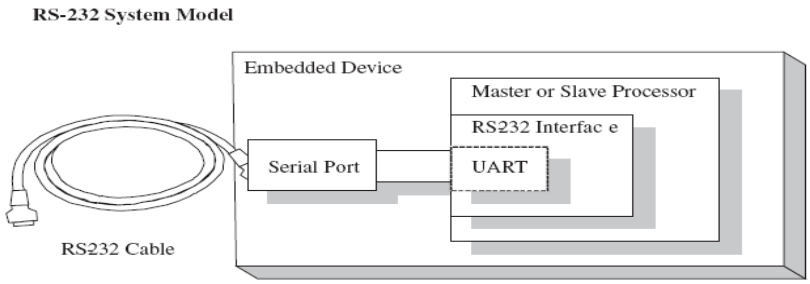


Hình 2-18: Mô hình OSI

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

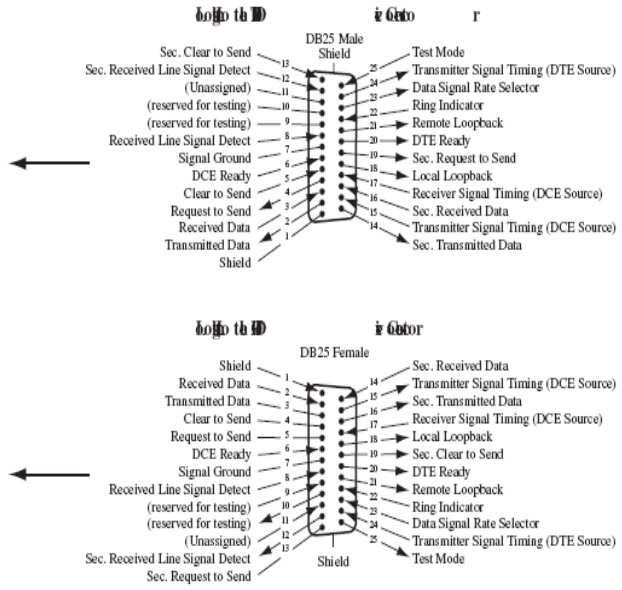
Bản chất đặc tính kỹ thuật của RS232 :*RS232 interface* (xem Hình 2-20). RS232 interface là định nghĩa chi tiết của cổng nối tiếp và những tín hiệu với những mạch điện được bổ sung là những tín hiệu ánh xạ từ 1 cổng giao diện đồng bộ hoặc không đồng bộ (ví dụ UART) được mở rộng bởi thiết bị vào/ra. RS232 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).



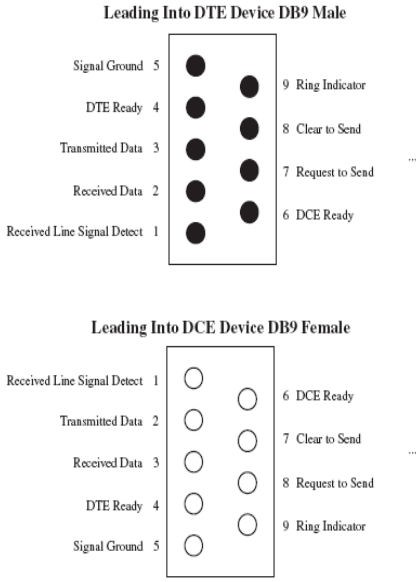
Hình 2-20: Sơ đồ khối những thành phần nối tiếp

Dùng số tín hiệu để phân biệt giữa chuẩn EIA 232 và chuẩn RS232. Chuẩu 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.

Pin	Name	Signal	Description	Voltage	DTE	DCE
1		FG	Frame Ground/Shield		Out	In
2	BA	TxD	Transmit Data	−12	In	Out
3	BB	RxD	Receive Data	−12	Out	In
4	CA	RTS	Request To Send	+12	In	Out
5	CB	CTS	Clear To Send	+12	In	Out
6	CC	DSR	Data Set Ready	+12		
7	AB	SG	Signal Ground			
8	CF	DCD	Data Carrier Detect	+12	In	Out
9			Positive Test Voltage			
10			Negative Test Voltage			
11			Not Assigned			
12		sDCD	Secondary DCD	+12	In	Out
13		sCTS	Secondary CTS	+12	In	Out
14		sTxD	Secondary TxD	−12	Out	In
15	DB	TxC	DCE Transmit Clock		In	Out
16		sRxD	Scondary RxD	−12	In	Out
17	DD	RxC	Receive Clock		In	Out
18	LL		Local Loopback			
19		sRTS	Secondary RTS	+12	Out	In
20	CD	DTR	Data Terminal Ready	+12	Out	In
21	RL	SQ	Signal Quality	+12	In	Out
22	CE	RI	Ring Indicator	+12	In	Out
23		SEL	Speed Selector DTE		In	Out
24	DA	TCK	Speed Selector DCE		Out	In
25	TM	TM	Test Mode	+12	In	Out

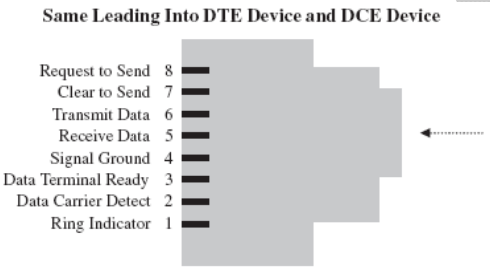


Hình 2-21a: Các tín hiệu RS-232 và đầu nối DB25



DB9 Pin	Name	Signal	Description	Voltage	DTE	DCE
1	109	DCD	Data Carrier Detect	+12	In	Out
2	104	RxD	Receive Data	–12	Out	In
3	103	TxD	Transmit Data	–12	In	Out
4	108	DTR	Data Terminal Ready	+12	Out	In
5	102	SG	Signal Ground			
6	107	DSR	Data Set Ready	+12		
7	105/13	RTS	Request To Send	+12	In	Out
8	106	CTS	Clear To Send	+12	In	Out
9	125	RI	Ring Indicator	+12	In	Out

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



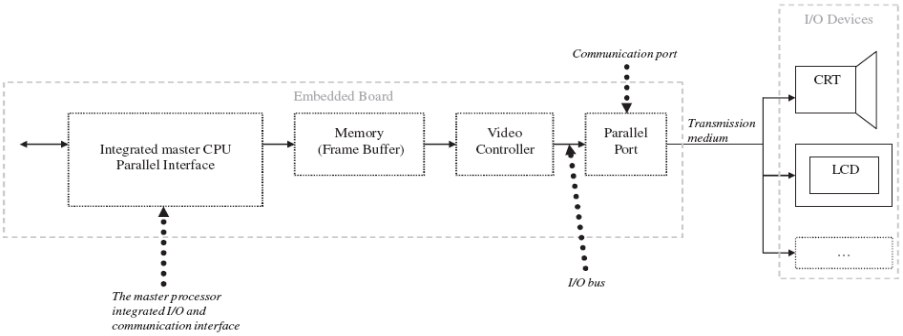
DB9 Pin	Name	Signal	Description	Voltage	DTE	DCE
1	125	RI	Ring Indicator	+12	In	Out
2	109	DCD	Data Carrier Detect	+12	In	Out
3	108	DTR	Data Terminal Ready	+12	Out	In
4	102	SG	Signal Ground			
5	104	RxD	Receive Data	+12	Out	In
6	103	TxD	Transmit Data	+12	In	Out
7	106	CTS	Clear To Send	+12	In	Out
8	105/13	RTS	Request To Send	+12	In	Out

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

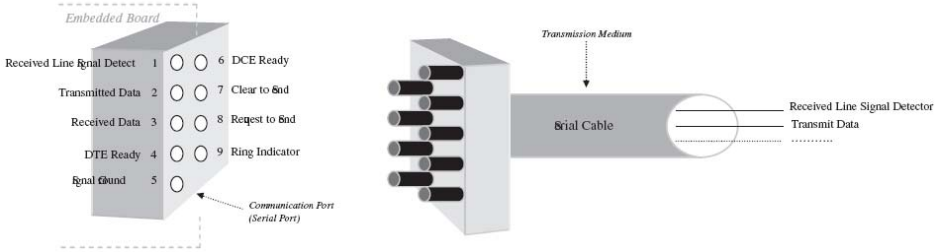
Phần cứng vào/ra được làm từ tất cả hoặc kết hợp của bộ xử lý chính vào/ra hoặc bộ điều khiển vào/ra, bus vào/ra và môi trường truyền dữ liệu. Xem Hình 2-22.



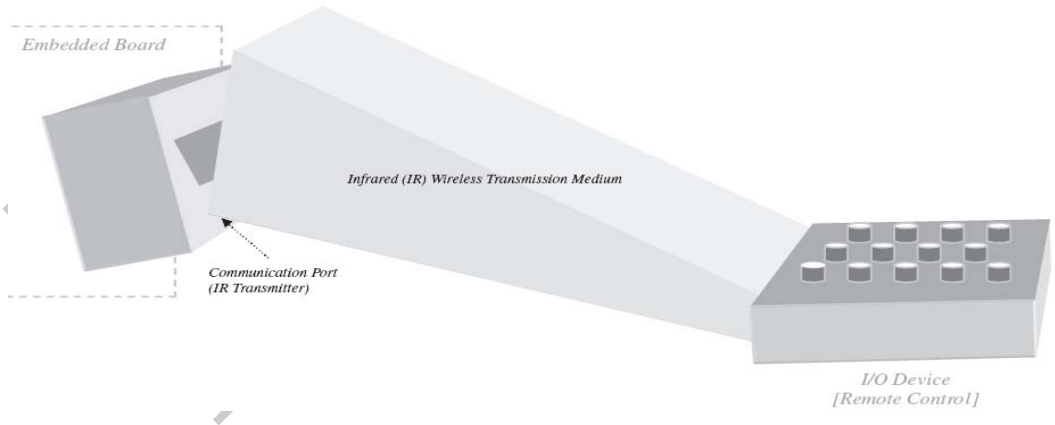
Hình 2-22: Hệ thống con I/O mẫu

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

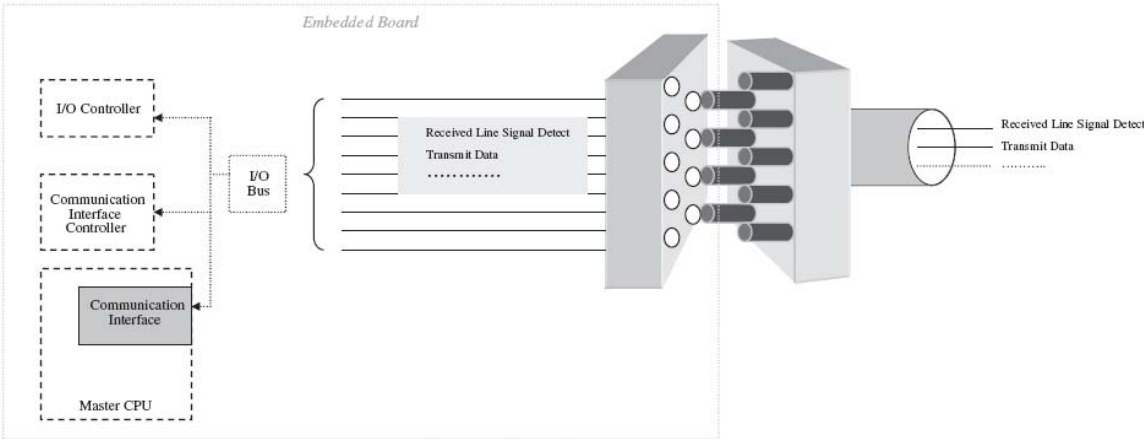


Hình 2-23b: Phương tiện truyền tin không dây

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 kiển 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 1giao 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 khiên thẻ 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:

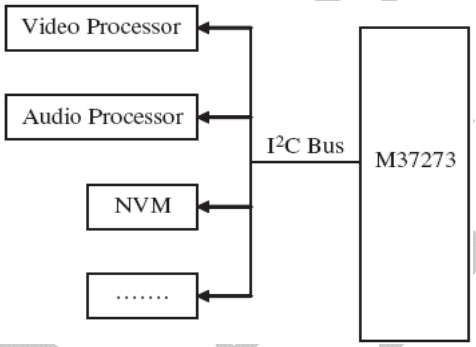
- Tốc độ truyền dữ liệu của thiết bị vào/ra
- Tốc độ của bộ xử lý chính
- Cách đồng bộ hóa giới hạn của bộ xử lý chính và giới hạn của thiết bị vào/ra.
- Cách để vào/ra và bộ xử lý chính liên lạc

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.

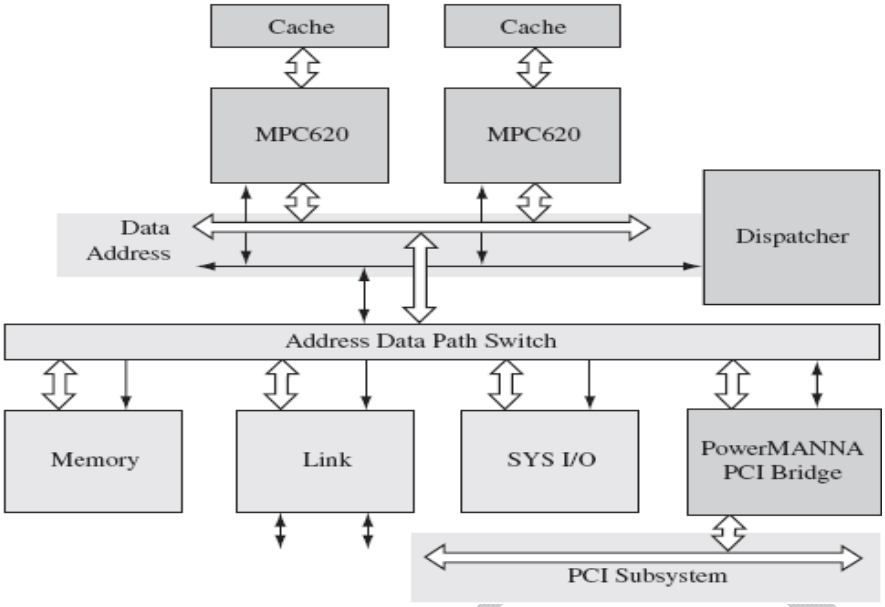
2.1.5 Hệ thống Bus

Tất cả các bộ phận chính khác nhau tạo nên một bảng nhúng- bộ xử lí 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 lí 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 nhúng 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 giữ 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 lí 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 nhụ 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.

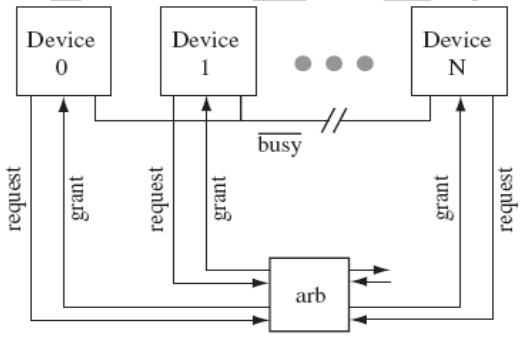
2.1.5.1 TRỌNG TÀI BUS VÀ ĐỊNH THỜI

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.

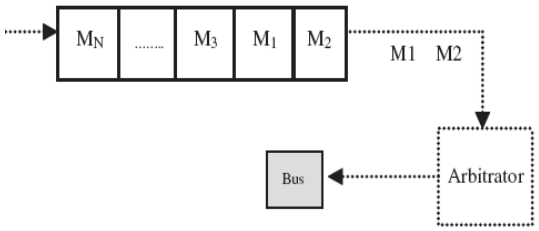
Bảng các thiết bị được truy cập vào một bus đang dùng một hệ thống trọng tài bus. Trọng tài Bus được dựa trên các thiết bị đang được phân loại như là một trong hai thiết bị chủ (các thiết bị có thể bắt đầu một giao dịch bus - transaction) hoặc các thiết bị phụ (là các thiết bị mà có thể truy nhập vào 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 nhúng đang được phổ biến nhất song song tập trung động, tập trung theo từng chuỗi (daisy-chain) và tự lựa chọn phân phối.

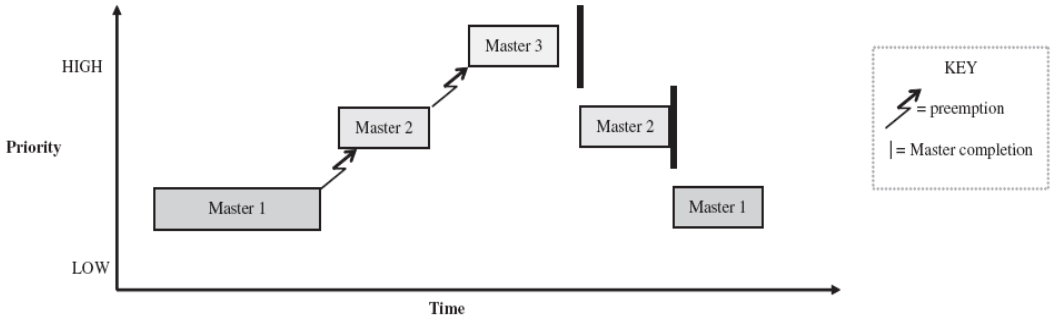
Sự phân xử song song tập trung động là sơ đồ trong đó trọng tài ở vị trí trung tâm. Tất cả các bus chủ kết nối với trọng tài trung tâm. Trong sơ đồ này, các máy chủ sau khi được cấp quyền truy cập đến bus thông qua FIFO hoặc được ưu tiên trên cơ sở hệ thống. Thuật toán FIFO thực hiện một số loại hàng 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



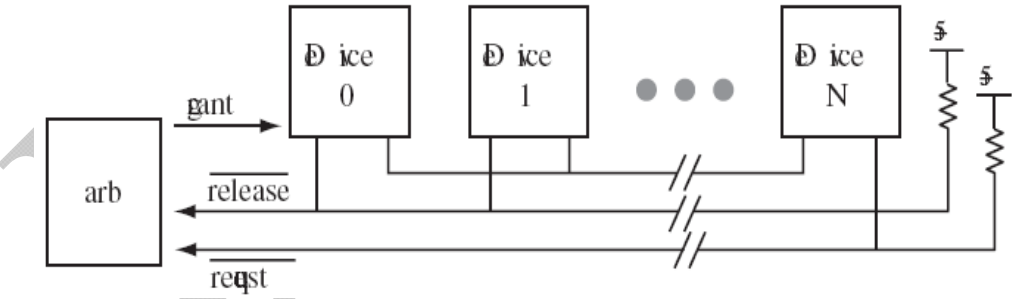
Hình 2-28b: Sự phân xử trên nền tảng FIFO



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

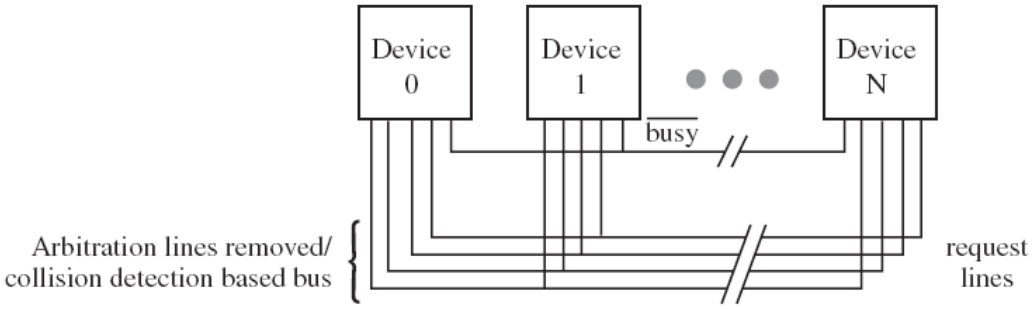
Hệ thống phân xử ưu tiên phân biệt với máy chủ dựa trên tầm quan trọng tương đối với nhau và hệ thống. Về cơ bản thì mỗi thiết bị chủ được ưu tiên gán giá trị, hành động như một chỉ thị của thứ tự ưu tiên trong hệ thống. Nếu bộ phân xử thực hiện quyền ưu tiên trên cơ sở hệ thống, thiết bị chủ với quyền 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 lí (một là máy chủ, một là điều khiển I/O) hoặc bộ xử lí 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 tùng bus) và dữ liệu được truyền trên xung cao hoặc thấp của chu kì xung nhịp. Thứ tự làm việc của một hệ thống, các thành phần hoặc phải đóng lân cận 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. Điển 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 lí 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

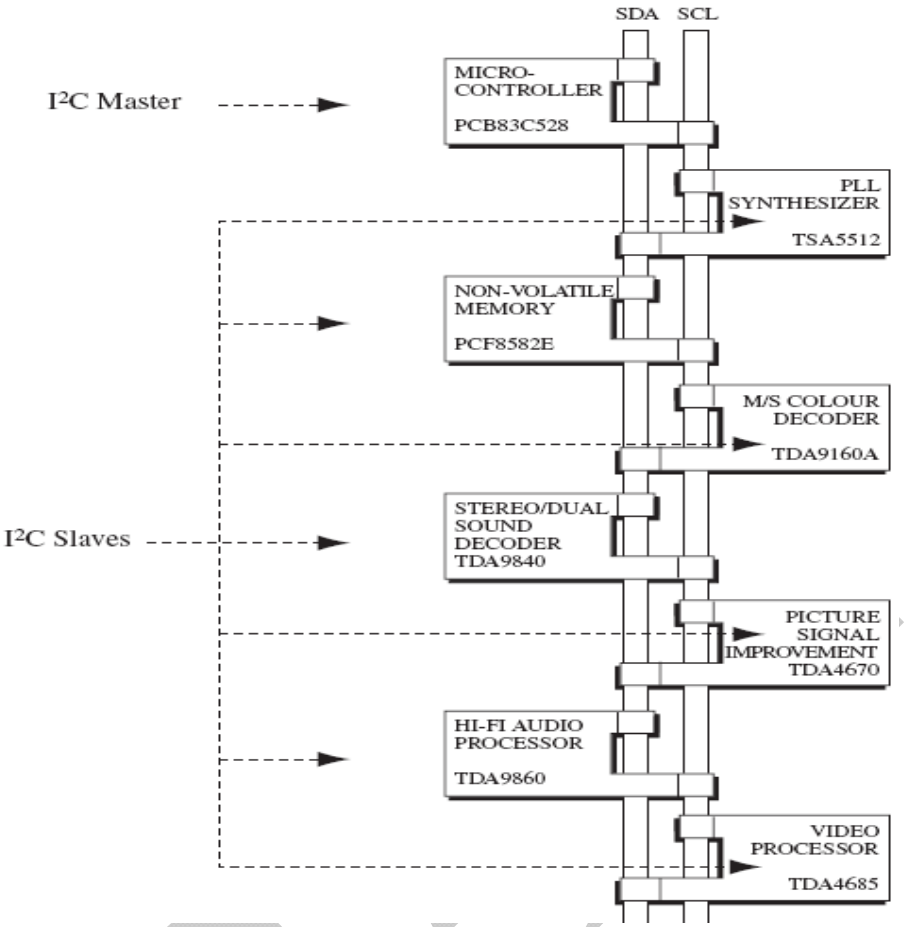
tiếp hoặc yêu cầu (ví dụ như một sự thừa nhận ACK hoặc yêu cầu ENQ). Cơ sở của hai giao thức này là điều khiển truyền tín hiệu thông qua điều khiển một dải bus riêng hoặc qua một dải dữ liệu. Dù nó là một yêu cầu dữ liệu tại bộ nhớ hay là giá trị của bộ điều khiển I/O điều khiển hoặc trạng thái đăng 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 để 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 lí bất kì 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 lí có kết hợp một I2C trên giao diện chíp, cho phép trực tiếp truyền thông tin giữa các bộ xử lí trên bus. Một máy chủ/ máy phụ thuộc có mối quan hệ với các bộ xử lí 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 lí được kết nối qua I2C định đỉa chỉ bởi một địa chỉ duy nhất là một phần của một luồng dữ liệu được truyền giữa các thiết bị.

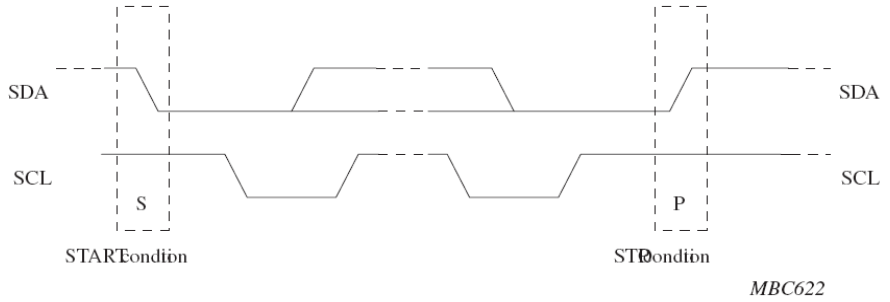


Hình 2-31: Bảng mạch mẫu TV

Máy chủ I2C bắt đầu truyền dữ liệu và tạo ra các xung nhịp cho phép truyền. Về cơ bản SCL có chu kì là mức cao hay thấp .

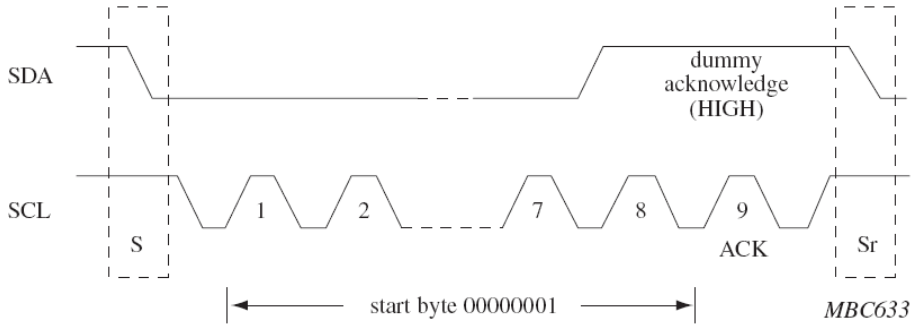
Một máy chủ sau khi sử dụng dải SDA (như SCL là chu kì) để chuyển dữ liệu đến các máy phụ, một phiên được bắt đầu và kết thúc như Hình 2-32. "START" là bắt đầu khi máy chủ kéo cổng SDA ở mức thấp trong khi tín hiệu SCL ở mức cao, nhưng trái lại điều kiện "STOP" bắt đầu khi máy chủ kéo SDA ở mức cao khi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

Đối với việc truyền tải dữ liệu, bus I2C là một dãy 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 bít dữ liệu " đọc" mỗi lần tín hiệu SCLchuyển từ mức cao xuống mức thấp, sườn đến sườn.

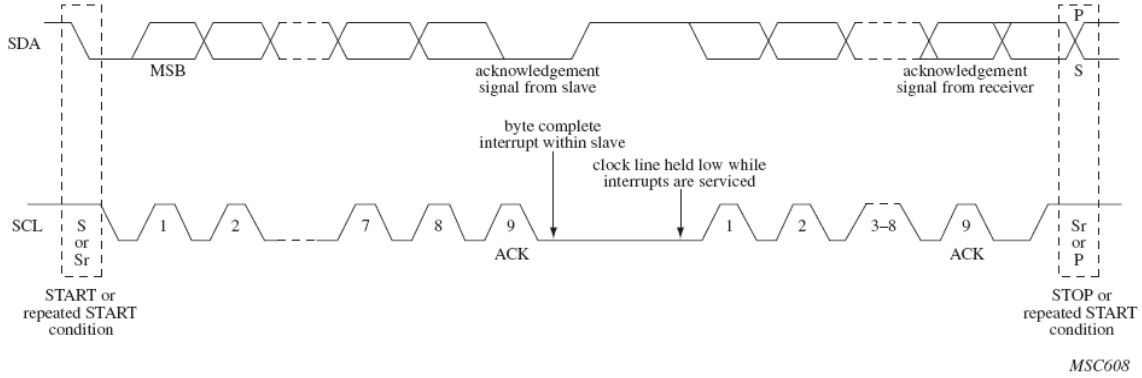


Hình 2-32: Các điều kiện START và STOP của I2C

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 bít 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



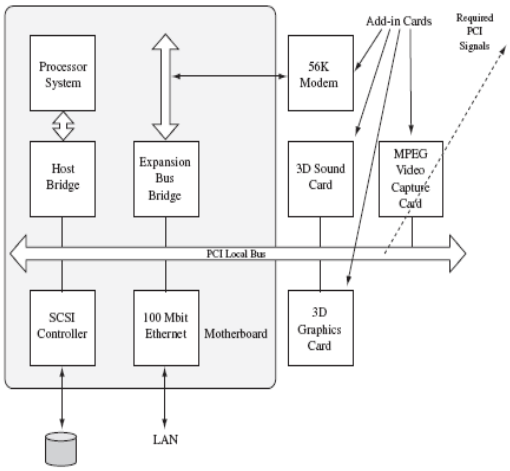
Hình 2-33b: Sơ đồ truyền dữ liệu hoàn chỉnh trên I2C

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 lí 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.

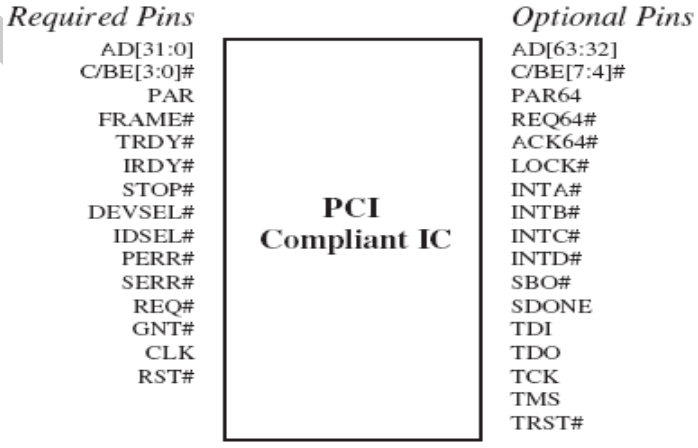


Signal Name	Driven by	Description
CLK	Master	Bus Clock (normally 33MHz; DC okay)
FRAME#	Master	Indicates start of a bus cycle
AD[31:0]	Master/Target	Address/Data bus (multiplexed)
C/BE#[3:0]	Master	Bus command (address phase) Byte enables (data phases)
IRDY#	Master	Ready signal from master
TRDY#	Target	Ready signal from target
DEVSEL#	Target	Address recognized
RST#	Master	System Reset
PAR	Master/Target	Parity on AD, C/BE#
STOP#	Target	Request to stop transaction
IDSEL		Chip select during initialization transactions
PERR#	Receiver	Parity Error
SERR#	Any	Catastrophic system error
REQ#		Request Bus
GNT#		Bus Grant

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^2C.

Bus I2C 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 I2C. Do I2C 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 I2C, 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 lí 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 lí 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 trển đường là các yêu 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 dải 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 kì bus, đây gọi là độ rộng của bus. Các bus thường có băng thông là số mũ nhị phân 2, chẳng hặn như 1(2^0) cho các bus với độ rộng bus nối tiếp, 8(2^3) bít, 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 bít bất kể lúc nào cũng có thể được truyền. Độ rộng của bus giới hạn băng thông của bus vì nó giới hạn số bít dữ liệu có thể được truyền trong bất kì một giao dịch nào. Độ trễ có thể sảy 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 truya 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 rỗi 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 kìm hãm 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 Unistall, 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:

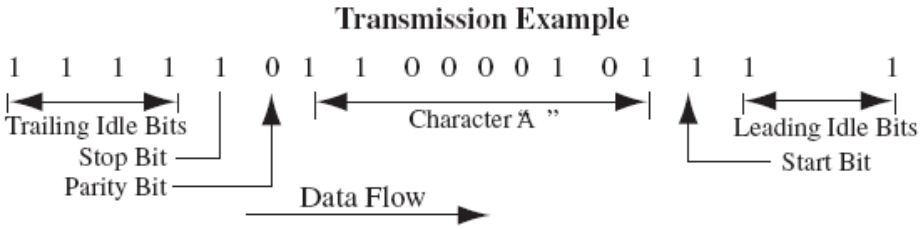
Một trong những chuỗi giao thức I/O không đồng bộ được thiết lập là RS-232 hoặc EIA-232 , dựa trên nền tảng chuẩn của hội công nghiệp điện tử. Các chuẩn này xác minh những bộ phận chính của bất cứ hệ thống gốc RS-232 nào được thực thi trong phần cứng.

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 Jave không chú í 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 lí đơn). Trong các giai đoạn, các ngắt bị vô hiệu hóa có thể hoàn toàn ngắn để tránh sự chậm trễ không thể dự đoán được trong việc sử lý các biến cố quan trọng.

Đối với RTOS thực hiện thệ 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 lí 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 kĩ 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 lí chính này là nguồn được tìm thấy trong mỗi hệ thống, bao gồm bộ xử lí, 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.

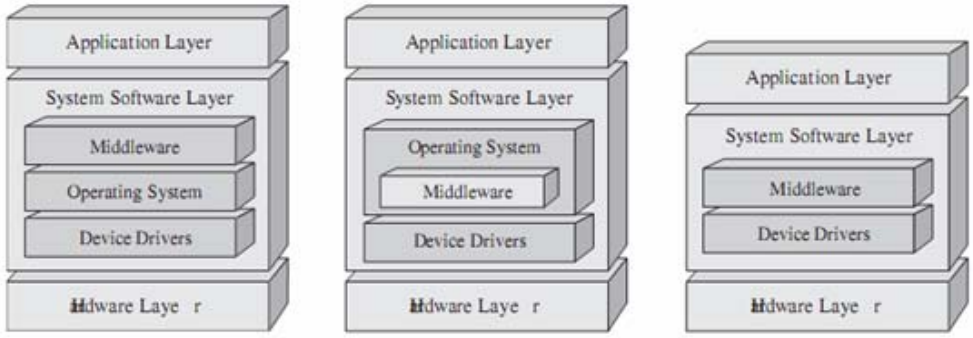
Trong khi một số RTOS được thiết kế cho các ứng dụng gán nói chung, các ứng dụng khác tập trung trên một khu vực cụ thể. Ví dụ, OSEK/VDX OS tập trung điều khiển tự động. Do tập trung này, nó là môt hệ điều hành khá nhỏ.

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 biển phát triển nhiều ở nhật bản. Nhiều loại RT chính của OS có riêng API. ITRON, được đề cặp 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.

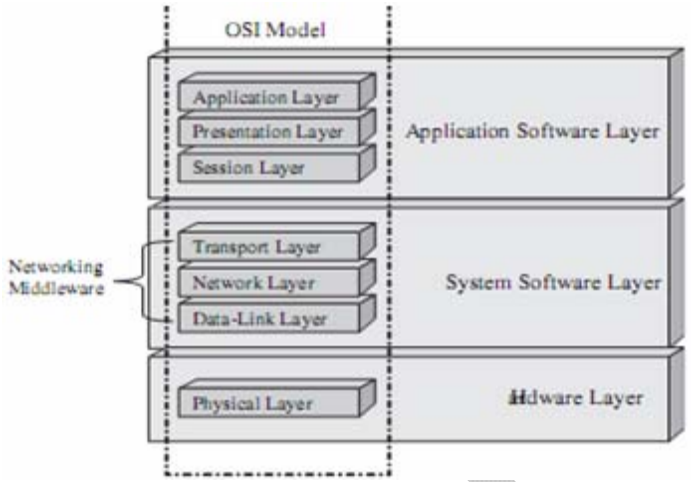
+ ***Market-specific*:** nghĩa là chúng là độc nhất trong một gia đình riêng biệt của các hệ thống nhúng như một phần mềm trên nền tiêu chuẩn TV số đặt trên hệ điều hành hay trên JVM.

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.

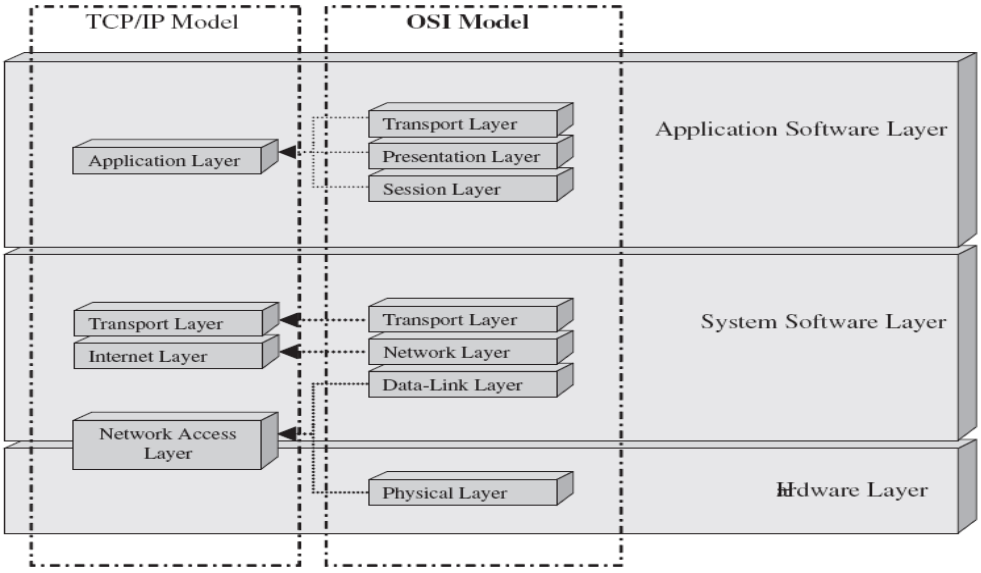
Ví dụ middleware driver mạng:

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.

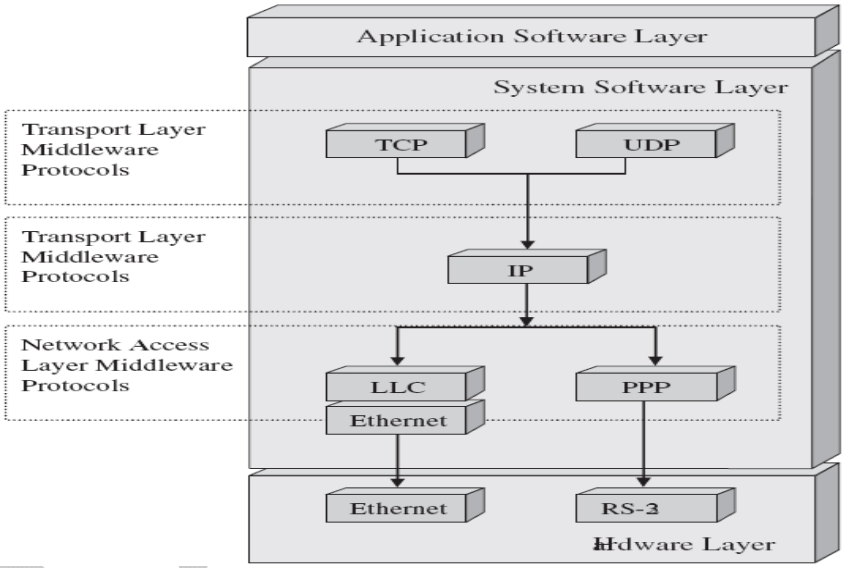


Hình 2-38: Mô hình OSI và middleware

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 lí 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

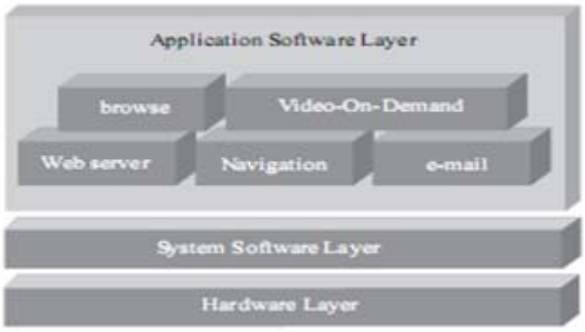


Hình 2-39b: Sơ đồ khối mô hình TCP/IP và các giao thức

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 lí 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 lí 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 Havard 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 gắn với hệ thống thời gian thực:

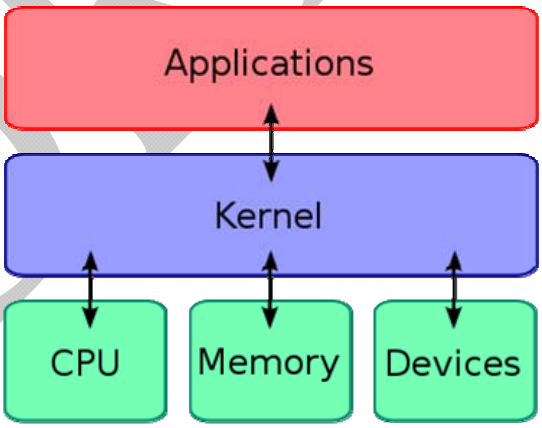
- Đa nhiệm/Đa luồng (Multitasking/multithreading): Một hệ điều hành thời gian thực phải có khả năng xử lý đa nhiệm và đa luồng.

- Cơ chế ưu tiên (Priorities): Hệ điều thành 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 thành 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.

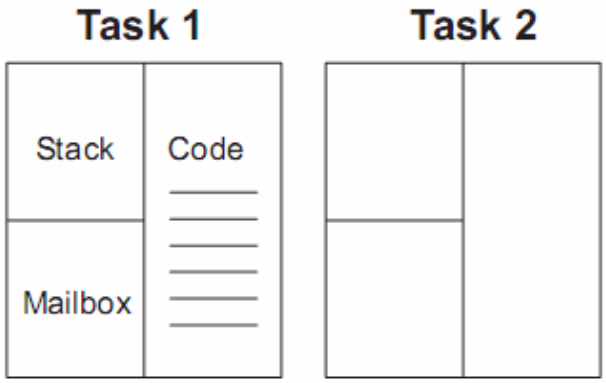
Vấn đề thứ ba là số lượng tác vụ. Nếu số lượng tác vụ trong hệ thống quá lớn, hệ thống sẽ không thể xử lý kịp các yêu cầu, thậm chí mỗi tác vụ chỉ chiếm một lượng thời gian hạn chế.

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 vụs 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.

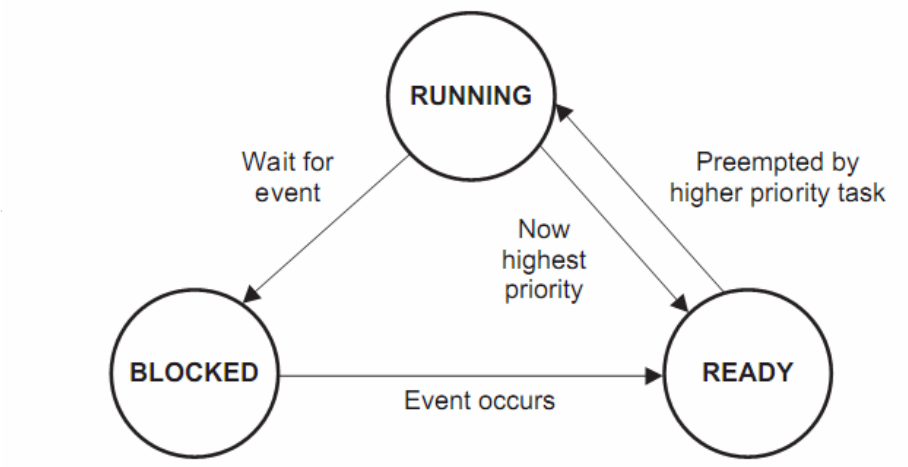
3.3.3 Lập lịch thời gian thực (Real-time Scheduling)

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 điệu 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 họa 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 tụ 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 cáo 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 căn 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

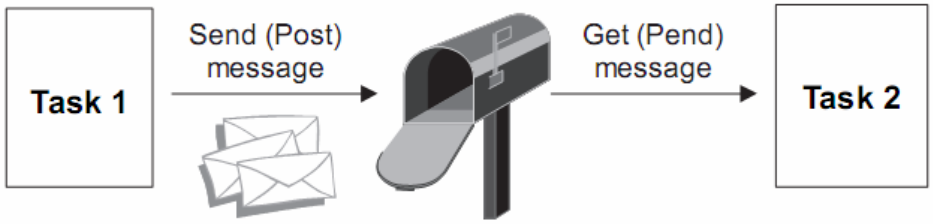
- Cờ sự kiện (Event Flag): Dùng để thể hiện một hay nhiều sự kiện xảy ra. Cờ sự kiện cho phép đồng bộ khóa trong trường hợp có nhiều sự kiện kết hợp.
- Hòm thư (mailbox), hàng đợi (queue) hay đường ống (pipe): Các cơ chế này dùng để chuyển dữ liệu giữa các tác vụ với nhau.

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 tin hiệu cho các sự kiện, các tác vụ thường xuyên cần chia sẻ dữ liệu. Việc chia sẻ này được thực hiện bừng 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 dùng 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 nên 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.

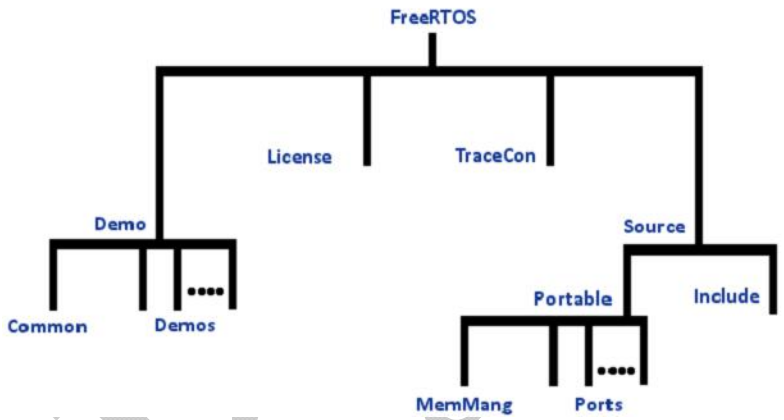
Các dòng vi điều khiển đã được thử nghiệm với FreeRTOS:

- 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 bao gồm mã nguồn cho các bản port trên các bộ vi xử lý và các ứng dụng demo. 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 nhỏ 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_AtmelSAM7S64_IAR Ứng dụng mẫu cho AtmelSAM7S64 tool IAR.
 ¦   ¦
 ¦   +-ARM7_AT91SAM7X256_Eclipse Ứng dụng mẫu cho AT91SAM7X256, tool
Eclipsse
 ¦   ¦
 ¦   +-.....
 ¦
 +-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 bản đồ 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()*.

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

- *xTaskGetTicksCount:* 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 dach sách */
    volatile struct xLIST_ITEM * pxNext;
    /* Con trỏ đến phần tử sau nó trong dach 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-rountine 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 biết.
 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*/
#ifdef 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 công 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?