HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Khoa Công nghệ thông Tin

Bộ môn Khoa học máy Tính

XÂY DỰNG CÁC HỆ THỐNG NHÚNG

Hà Nội, tháng 11 năm 2013

(bản sửa và bổ sung)

Lời nói đầu

Dạy và học hệ thống nhúng là đề cập tới một chủ đề có phạm vi rộng bao gồm thiết kế, môi trường ứng dụng, loại hình công nghệ, qui tắc cần thiết để tiếp cận một cách có hệ thống. Lĩnh vực thiết kế và ứng dụng các hệ thống nhúng bao gồm : các hệ thống vi điều khiển (micro-controller) nhỏ và đơn giản, các hệ thống điều khiển, hệ thống nhúng phân tán, hệ thống trên chip, mạng máy tính (có dây và không dây), các hệ thống PC nhúng, các hệ thống ràng buộc thời gian, robotic, các thiết bị ngoài của máy tính, xử lý tín hiệu, hệ thống lệnh và điều khiển… Nền tảng công nghệ hiện đại là kỹ thuật vi điện tử với mật độ tích hợp lớn và rất lớn. Khi muốn thiết kế hệ thống nhúng, có nhiều yếu tố cần tuân thủ giống như khi thiết kế máy tính, nhưng lại bị ràng buộc bởi đặc thù ứng dụng. Thêm vào đó là sự đan chéo của các kỹ năng rất cần thiết cho thiết kế hệ thống nhúng, độc lập vận hành, thiết kế với tiêu chí tiêu hao năng lượng thấp, công nghệ phần cứng, công nghệ phần mềm (hệ thống và ứng dụng), hệ thời gian thực, tương tác người máy và đôi khi cả vấn đề an ninh hệ thống. Như vậy đào tạo và học hệ thống nhúng cần một khối lượng kiến thức tập hợp ít nhất từ các bộ môn khác như khoa học máy tính (*computer science*), khoa học truyền thông (*communication*), kỹ thuật thiết kế điện tử: các mạch tương tự và số, sử dụng tốt các phần mềm thiết kế bo mạch (như Protel, Proteus, DXP…), kiến thức về chế tạo bán dẫn. Vì là bộ môn công nghệ có tính ứng dụng cao với bài toán cụ thể, nên lại cần có chuyên môn của ngành nghề, mà ở đó hệ thống nhúng sẽ ứng dụng. Tóm lại đây là một chủ đề hợp nhất và việc thực hiện chủ đề này thực không dễ dàng.

Với lượng thời gian nhất định, môn học XÂY DỰNG CÁC HỆ THỐNG NHÚNG sẽ mang lại cho người học những vấn đề cơ bản nhất về hệ thống nhúng ở Chương 1. Chương 2 đề cập tới kiến trúc phần cứng hệ thống, cách thiết kế một số khối chức năng cơ sở có tính thực tế cao. Chương 3 chủ yếu giới thiệu về phần mềm cài đặt trên hệ thống nhúng, bao gồm các trình điều khiển thiết bị, các phần mềm trung gian, và phần mềm hệ thống được cài đặt . Đặc biệt nhắc lại một số yêu cầu về khái niệm của các hệ thống thời gian thực và hệ điều hành thời gian thực. Chương 4 giới thiệu các tiêu chí và phương pháp thiết kế hệ thông nhúng. Cuối chương là một số các bài tập lớn kiểu Dự án thiết kế, có thể lựa chọn cho thực hành với các kiểu kiến trúc hệ thống nhúng khác nhau.

Như đã nêu, đây là chủ đề rộng, mang tính kỹ thuật và kiến thức lại được tổng hợp từ các môn khác, nên tài liệu này chắc không thể thật sự đầy đủ. Các phần kiến thức nào không được đề cập sâu ở đây, người học cần tham khảo thêm các tài liệu khác, hay từ các môn học liên quan.

Tác giả xin chân thành cám ơn các cán bộ, giảng viên Khoa Công nghệ thông tin và bộ môn Khoa học máy tính, Học viện Công nghệ BCVT Hà Nội đã góp ý để tác giả hoàn thành giáo trình. Tác giả cũng xin đón nhận các ý kiến đóng góp, phê bình từ người đọc, người học, sao cho tài liệu này có ích hơn. Địa chỉ theo e-mail: htcuoc@ioit.ac.vn.

Hà Nội, tháng 10 năm 2013

Huỳnh Thúc Cước,
Viện Công nghệ thông tin, VAST, 18, Hoàng Quốc Việt, Hà Nội.

Một số chữ viết tắt

CPU	Central Processing Unit	Đơn vị xử lý trung tâm
ROM	Read Only Memory	Bộ nhớ chỉ đọc
EPROM	Erasable programmable read-only memory	Bộ nhớ chỉ đọc, xóa và lập trình lại được
RAM	Random Access Memory	bộ nhớ truy cập ngẫu nhiên
FLASH	non-volatile computer storage (memory cards, USB flash drives, solid-state drives -SSD)	Bộ nhớ bán dẫn không bị mất nội dung ngay cả khi không cung cấp nguồn nuôi
OS	Operating System	Hệ điều hành
RTOS	Real Time Operating System	Hệ điều hành thời gian thực
ES	Embedded System	Hệ thống nhúng
HTN	Embedded System	Hệ thống nhúng
OS hay HĐH	Operating System	Hệ Điều Hành
TĐKTB	Device Driver	Trình điều khiển thiết bị
PLC	Programmable Logic Controller	bộ điều khiển logic khả trình
PIC	Programmable Intelligent Computer	Máy tính khả trình thông minh
PSoC	*Programmable System - on - Chip*	Hệ thống khả trinh trên vi mạch
ASIC	Application-Specific Integrated Circuit	ASIC là một vi mạch được thiết kế dành cho một ứng dụng cụ thể theo yêu cầu cá biệt
MCU	Microcontroller Unit	Vi điều khiển
CICS	Complex Instruction Set	Tập lệnh đầy đủ
RISC	Reduced Instruction Set	Tập lệnh rút gọn
SPI	Serial Peripheral Interface	Đường liên kết dữ liệu nối tiếp, đồng bộ, hoạt động theo kiểu Chủ/tớ (Master/Slave)
I^2C	Inter-Integrated Circuit	Bus dùng để nối giữa các vi mạch điện tử …

USART	Universal Serial Aynchronous Receiver/Transmitter	Bộ thu/phát nối tiếp di bộ đa năng
ISR	Interrupt Service Routine	Chương trình con xử lí ngắt hay Dịch vụ xử lí ngắt
MAC	Media Access Control	Điều khiển truy nhập môi trường (mạng máy tính). Ví dụ: *MAC address*: Địa chỉ vật lí của thiết bị mạng.
MIPS	Million instructions per second	Triệu lệnh máy trong một giây
IDE	Integrated Development Environment, hoặc: *Integrated Design Environment* *hoặc:* *Integrated Debugging Environment*	Là tập các phần mềm hỗ trợ các công cụ, tiện ích đê phát triển phần mềm máy tính, bao gồm: Soạn thảo mã nguồn, trình thông dịch, trình biên dịch, trình gở rối
ICE	In-Circuit Emulator	Là loại thiết bị phần cứng dùng để gở rối khi phát triển phần cứng và phần mềm hợp nhất, như HTN. Ví dụ như Logic anlyzer, phần mềm MPLAB của Microchip

Danh sách các hình vẽ

Chương 1. GIỚI THIỆU CHUNG VỀ CÁC HỆ THỐNG NHÚNG

1.1 KHÁI NIỆM VỀ HỆ THỐNG NHÚNG (HTN)

Nhìn lại những năm 70 thế kỹ trước, xử lý thông tin thường phải sử dụng các máy tính lớn hay máy tính mini (ví dụ dòng máy mini PDP 11 của hãng DEC một hệ thống thống trị trong truyền thông). Cho tới những năm 80, khi vi xử lý và máy tính cá nhân (PC để bàn và xách tay) ra đời, máy tính trở thành công cụ đắc dụng cho xử lý thông tin bởi khả năng tính toán nhanh, gọn nhẹ và di động linh hoạt. Giai đoạn tiếp theo là khả năng chế tạo vi mạch kích thước vài trăm micro mét và nano mét của những năm chín mươi, đã thúc đẩy xu hướng nhỏ hóa (*miniaturization*) và đa dạng các dòng vi xử lý, phát triển mạnh mẽ. Các bộ vi xử lý đa năng và sự xuất hiện các vi xử lý chuyên biệt (ASIC- *application-specific integrated circuit*) được chế tạo với số lượng lớn chưa từng thấy. Việc sử dụng các bộ vi xử lý chuyên biệt để tạo ra các thiết bị chuyên xử lý một hay một vài bài toán kĩ thuật, tạo ra một nghành công nghệ mới, gọi là công nghệ nhúng. Sản phẩm của công nghệ nhúng này là các *hệ thống nhúng*.

Vậy hệ thống nhúng (HTN- *Embedded system*) là gì?

Có nhiều định nghĩa về HTN, nhưng nếu ta lấy tiêu chí mô tả HTN làm cái gì và sử dụng nó như thế nào, thì có thể nói về HTN như sau:

Hệ thống nhúng là một thuật ngữ để chỉ một hệ thống có khả năng hoạt động tự trị được nhúng vào trong một môi trường hay một hệ thống khác qui mô phức tạp hơn. Đó là các hệ thống tích hợp cả phần cứng (là một hệ thống máy tính được xây dựng trên cơ sở sử dụng vi xử lý - *microprocessor-based system*) và phần phềm nhúng trong phần cứng đó, để thực hiện các bài toán chuyên biệt.

Hay theo định nghĩa của tổ chức IEEE thì hệ thống nhúng là một hệ tính toán (*máy tính số*) nằm trong (hay *được nhúng vào*) sản phẩm khác lớn hơn và rằng thông thường ẩn đối với người sử dụng. Nói rộng ra, và đơn giản hơn, khi một hệ tính toán (có thể là PC, IPC, PLC, vi xử lý, vi hệ thống (microcontroller), DSP v.v...) được nhúng vào trong một sản phẩm hay một hệ thống nào đó và thực hiện một số chức năng cụ thể của hệ thống đó, thì ta gọi hệ tính toán đó là một hệ thống nhúng. Tuy nhiên thật không dễ gì định nghĩa cho thật đúng về HTN, định nghĩa trên rất ít nói tới công nghệ và cũng rất đơn giản. Hiện nay chưa có định nghĩa nào thật thỏa đáng về HTN, ví dụ nếu lấy chức năng xử lý thông tin, thì HTN là một phần xử lý thông tin nhúng trong các hệ thống lớn hơn và phức tạp hơn, hay cũng có thể là một hệ thống độc lập vận hành tự động. Ví dụ gần gũi ta có: máy tính cá nhân, hay máy chủ, là một hệ thống phức tạp được xây dựng từ các thành phần hoạt động độc lập nhưng được đồng bộ với nhau. Vĩ điều khiển đồ họa, có vi điều khiển rất mạnh xử lý đồ họa, vĩ điều khiển trên đĩa cứng có vi điều khiển chuyên dụng để xử lý tín hiệu, ghi/đọc dữ liệu từ đĩa từ tính theo yêu cầu của hệ điều hành, vĩ mạng cũng là một vi điều khiển tinh vi xử lý tín hiệu… Đó là các hệ thống con được nhúng trong hệ thống máy tính nói chung. Mặc dù vậy ta cũng sẽ nêu ra đây một số điểm chung về HTN.

1.2 ĐẶC ĐIỂM CỦA HTN

Để hiểu rõ hơn về HTN, ta nêu ra một số đặc điểm để nhận biết về một hệ thống nhúng:

- *Là một kiểu máy tính ứng dụng đặc biệt, rất giới hạn về phần cứng và phần mềm khi* so sánh với các máy tính đa năng, như máy tính cá nhân, máy chủ, siêu máy tính. Điều đó nói lên rằng hiệu năng xử lý, năng lượng tiêu thụ, bộ nhớ, các phần cứng khác đều hạn chế. Còn phần mềm hạn chế, hay phần mềm là cố định, có nghĩa hệ điều hành được thiết kế phù hợp với các xử lý đã định. Hiện nay hệ điều hành thường sử dụng là hệ điều hành đa nhiệm (như DOS 6.X hỗ trợ đa nhiệm trên các loại HTN dòng PC 104), hay hệ điều hành thời gian thực. Nếu không có hệ điều hành, thì cũng là một kiểu chương trình điều khiển chung (*monitor*) nào đó. Phần mềm viết ra không có các phần mã có mức độ trừu tượng hay có cũng ở mức thấp. Mã thực thi (gồm hệ điều hành và các ứng dụng) được nạp vào bộ nhớ ROM. Nhìn chung mã thực thi có kích thước nhỏ và tối ưu vì ROM có dung lượng nhỏ. Tuy nhiên với sự phát triển nhanh chóng của công nghệ, cách nêu trên có thể thay đổi, bởi sẽ có các HTN rất tinh xảo và mức độ phức tạp rất cao, bộ nhớ có thể đến vài chục mega bytes.

- *HTN được thiết kế để thực hiện một hay vài ứng dụng xác định, chuyên biệt (Application specific)*, ví dụ các thiết bị nhúng công nghiệp như robot thuộc loại này. Tuy nhiên có những thiết bị nhúng khác như các PDA, điện thoại di động là các HTN có khả năng thực hiện nhiều chức năng hơn. Hay các Tivi kỹ thuật số lại có thể thực hiện các ứng dụng tương tác với màn hình cảm ứng, v.v… Tuy nhiên xu hướng hiện nay là tạo ra các *HTN khả trình* có giao diện kết nối với một hệ phát triển khác để nâng cấp phần mềm.

- HTN tương tác với môi trường ứng dụng qua nhiều phương thức:
  - ✓ Qua các bộ cảm biến (sensor), ghép nối vào HTN bằng dây dẫn, hay không dây;
  - ✓ Phát triển các giao thức truyền tin riêng biệt, hay theo các giao thức chuẩn để trao đổi thông tin với các thiết bị khác, có thể có hỗ trợ nối mạng LAN;
  - ✓ HTN thuộc loại thiết bị thông minh *tự phản ứng* (*reactive*), bị động nhưng tương tác liên tục với môi trường và có đáp ứng kịp thời với những tiến triển (sự kiện) mà môi trường đó xác lập.
  - ✓ Tương tác người-máy rất đơn giản nếu có vì HTN chạy độc lập và thông tin với hệ thống lớn hơn là chính. Ngày nay xu hướng WEB hóa giao diện tương tác là phổ biến, ví dụ các thiết bị kết nối mạng Internet như ADSL dòng SOHO (Small Office-Home Office), có WEB để làm cấu hình và quản trị. Tương tự, các HTN công nghiệp cũng phát triển theo xu hương này để dễ quản trị từ trung tâm điều khiển.
- HTN *hoạt động độc lập,* do đó các đặc điểm sau đây:
  - ✓ Độ tin cậy, lí tưởng là không có sự cố hỏng hóc.
  - ✓ Bảo trì: thời gian bảo trì nhanh chóng.
  - ✓ Có tính sẵn sàng cao, là kết quả của sự tin cậy và bảo trì.

- ✓ An toàn: nếu có sự cố xảy ra, HTN không gây ra những tác hại khác của toàn hệ thống.
- ✓ An ninh: dữ liệu của HTN được bảo mật, truy nhập phải có xác nhận (ví dụ, HTN là các thiết bị truyền thông, SOHO).

▪ *HTN là một kiểu máy tính có yêu cầu về chất lượng và độ tin cậy rất cao, hoạt động được trong các môi trường khắc nghiệt* về nhiệt độ (cao, hay rất thấp), độ ẩm cao, độ rung động lớn, nhiễu sóng điện từ v.v… Ví dụ các máy tính trong công nghiệp, các thiết bị truyền thông, trạm BTS chẳng hạn, hay các máy tính điều khiển trên máy bay (fly by wire)…

▪ Phần lớn các hệ thống nhúng hoạt động với *sự ràng buộc thời gian*: yêu cầu có thời gian cho (đáp ứng) đầu ra nhanh, đúng thời điểm, trong mối tương quan với thời điểm xuất hiện của (sự kiện) đầu vào. Kiểu hoạt động như vậy gọi là tạo đáp ứng theo *thời gian thực*. Thời gian thực có thể chia ra làm hai kiểu:

- ✓ Nhạy cảm với thời gian (*time- sensitive*): sự kiện chỉ được xử lý trong một khung thời gian nhất định;
- ✓ Thời gian tới hạn (*time critical*): khi có sự kiện, hệ thống phải phản ứng ngay, chuyển nhanh nhất đến mã chương trình ứng với sự kiện đó để xử lý, nói cách khác trong một của sổ thời gian cho phép, xử lý phải được thực hiện và phải có đáp ứng đầu ra. Ví dụ như vòng phản hồi trong các hệ có điều khiển ở đó HTN là bộ điều khiển chạy giải thuật điều khiển không đủ nhanh (xử lý, tính toán quá lâu), hệ thống trở nên không ổn định.

▪ Có hiệu năng cao. Các số đo sau đây se phản ánh đặc tính này:

- ✓ Sử dụng năng lượng thấp và hiệu quả. Có thể thấy điểm này ở các thiết bị di động.
- ✓ Mã phần mềm có kích thước rất tối ưu, vì mã phải cài toàn bộ trên HTN.
- ✓ Thời gian xử lý tác vụ (run-time) phải nhanh, sử dụng ít tài nguyên phần cứng (vì liên quan tới tiêu hao năng lượng).
- ✓ Trọng lượng nhỏ. Đây là một trong những lực chọn khi mua một HNT.
- ✓ Giá thành rất cạnh tranh. Muốn vậy thiết kế và sử dụng phần cứng, phần mềm cần quan tâm tới hiệu quả.

Dưới đây là các ví dụ về các HTN nhìn ở góc độ sử dụng:

Lĩnh vực ứng dụng	Thiết bị nhúng
Ô tô	Đánh lửa điện tử, Điều khiển động cơ, hệ thống phanh, hộ số …
Điện tử tiêu dùng	Tivi analog, Tivi số, CD, DVD, VCR,… PDA, Điện thoại di động, CAMCODER, GPS, tủ điều hòa, tủ

	lạnh, là vi sóng …
Công nghiệp	Robot, dây chuyền sản xuất tự động, SCADA agents…
Y tế	Máy thẩm tách, máy pha-lọc, máy thở, máy trợ tim, máy quét cắt lát, rất nhiều thiết bị y tế hiện đại…
Mạng thông tin (WAN, LAN, thoại)	Bộ định tuyến, gateway, chuyển mạch mạng, các thiết bị truyền thông-mạng, trạm chuyển tiếp, BTS di động …
Văn phòng	Máy Fax, máy potocopy, máy in (kim, laser, phun), máy quét, màn hình LCD…
Các lĩnh vực khác: an ninh, quốc phòng, hàng không, hàng hải	Được hợp nhất trong rất nhiều khí tài hiện đại của tất cả các binh chủng .

1.3 CÁC YÊU CẦU VỚI HTN

HTN thực tế là một loại máy tính dùng để xử lý thông tin đã ở dạng số. HTN có thể là một hệ thống độc lập như một thiết bị tích cực trong mô hình điều khiển, tức HTN là một *regulator số*, thực hiện các chức năng của *PID regulator*, khi các chức năng này được thể hiện bởi thuật toán và chuyển hóa ở dạng mã chương trình trong HTN. Trong khi đó HTN lại là một phần của một qui trình công nghệ trong công nghiệp. Như trên đã liệt kê các đặc điểm chung mà các hệ thống nhúng thường có, tự nhiên ta có thể rút ra được những yêu cầu cần có trên một hệ thống nhúng.

1. *Khả năng đáp ứng* với sự kiện bên ngoài (từ các tác nhân bị kiểm soát) phải nhanh nhạy, kịp thời, tức là khả năng theo *thời gian thực:*
   - ✓ *Các tác vụ có đáp ứng ràng buộc bởi thời hạn chót (deadline);*
   - ✓ *Thời gian phát hiện lỗi phải rất ngắn (tối thiểu);*
   - ✓ *Khi chạy các chu trình vòng lặp điều khiển bằng phần mềm phải có đáp ứng đầu ra đúng thời hạn;*
2. Có khả năng làm ở *môi trường khắc nghiệt*.
3. Có *giá thành* thấp hay *hiệu quả hoạt động/giá thành hợp lí*.
4. *Kích thước nhỏ gọn*, *nhẹ*, dễ mang dễ vận chuyển, lắp đặt.
5. *Tiêu thụ năng lượng thấp*, khả năng sử dụng nguồn pin, ắc qui (tất nhiên phụ thuộc vào dung lượng của pin, ắc qui).
6. *Hoạt động tin cậy, chịu lỗi cao*
   - ✓ *Tin cậy: đáp ứng dịch vụ yêu cầu đúng thời hạn sau thời gian từ t_0 đến t;*

- ✓ *Cho hằng số của tỉ số sự cố trong thời gian 1 giờ đồng hồ là e (e/h), thì biểu diễn tỉ số sự cố là* $R(t)=exp^{(-e(t-t0))}$*, sau đó thời gian giữa 2 lần sự cố (Mean Time To Failure- MTTF) sẽ là 1/R(t);*
- ✓ *Độ tin cậy cao nếu đạt ~*10^{-9} *sự cố /giờ.*
- ✓ *Tính sẵn sàng cao: Nếu thời gian sửa chữa sự cố trung bình là MTTR, thì tính sẵn sàng (availability) A=MTTF/(MTTF+MTTR).*

7. *An toàn và bảo mật.*
8. *Khả năng nâng cấp phần mềm* và *dự phòng nâng cấp phần cứng* (mở rộng qua khe cắm dự trữ, ví dụ các thiết bị mạng thường có tính năng này).

1.4 MÔ HÌNH TỔNG THỂ HTN

Như đã nêu trong định nghĩa của HTN, thì HTN có kiến trúc của một máy tính số, do vậy sẽ không có gì khác biệt khi mô tả mô hình kiến trúc của HTN nói chung. Có khác chăng là ở chi tiết của từng HTN cụ thể.

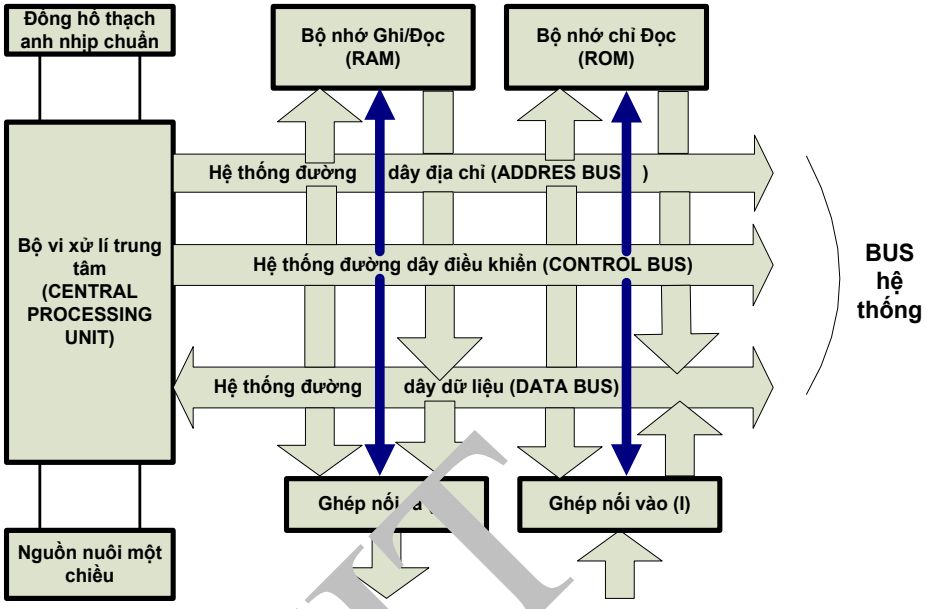
1.4.1 Mô hình cấu trúc phần cứng của máy tính

Dưới đây ta nêu ra mô hình tổng quát của máy tính theo nguyên lí Von Neumman. HTN cũng chia sẻ kiến trúc này trong một số trường hợp. Mô hình cho thấy các khối chức năng cơ bản cần có. Trong thực tế có những CPU chúng có kiến trúc cụ thể khác nhau, nhưng khi mô tả các khối chức năng, thì hoàn toàn thống nhất.

Mô hình lấy đơn vị xử lý trung tâm (Central Processing Unit – CPU) là khối chức năng cơ bản nhất để tạo nên một hệ vi xử lý, MNT hay máy tính cá nhân (*Personal Computer – PC*).

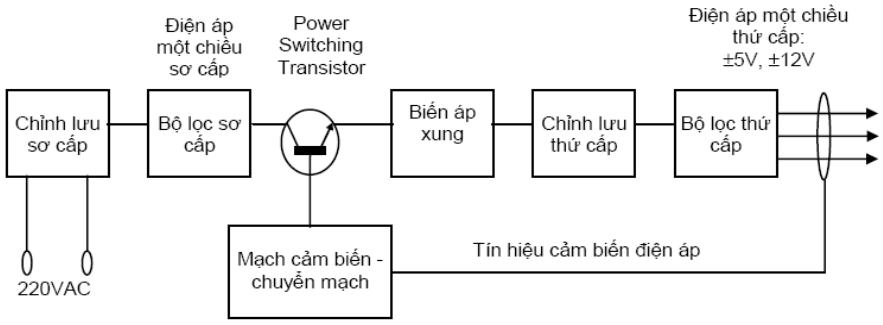
- CPU thực hiện chức năng xử lý dữ liệu theo nguyên tắc thực hiện chương trình máy tính cài trong bộ nhớ ROM hay nạp vào bộ nhớ RAM. Việc thực hiện như sau: CPU đọc mã lệnh (OPCODE) từ bộ nhớ (ROM hay RAM), sau đó,
  - ✓ Giải mã lệnh, tạo các tín hiệu (xung) điều khiển tương ứng với mã lệnh để điều khiển hoạt động của các khối chức năng khác trong CPU và bên ngoài CPU.
  - ✓ Tập các tín hiệu tạo ra BUS của CPU. Khi kết hợp với các vi mạch dồn /tách kênh và khuyếch đại, các mạch giải mã,… sẽ tạo ra BUS hệ thống. BUS hệ thống cung cấp các tín hiệu cho các vi mạch tích hợp vào CPU (ROM, RAM, I/O module) tạo thành bo mạch chính (CPU board).
  - ✓ Thực hiện từng bước các thao tác xử lý dữ liệu đã định nghĩa trong mã lệnh.
  - ✓ Ghi lại kết quả thực hiện lệnh.
- Bộ nhớ chính (ROM/RAM) được tổ chức từ các từ nhớ đơn, kép. Theo chuẩn của IBM/PC *từ nhớ đơn (có sở) có độ dài 1 byte* (8 bits). Bộ nhớ này gồm các chip nhớ chỉ đọc ROM (*Read Only Memory*) và các chip nhớ truy xuất ngẫu nhiên RAM (*Random Access Memory*) có tốc độ truy cập nhanh. Bộ nhớ được sử dụng để chứa

các chương trình và các dữ liệu cần xử lý. Các chương trình ứng dụng và dữ liệu có thể được chứa ở ROM hoặc RAM, các kết quả trung gian hay kết quả cuối cùng của các thao tác xử lý có thể được chứa trong các thanh ghi đa dụng hoặc trong RAM.



Hình 1.1- Mô hình tổng quát bo mạch chủ

- Các mạch ghép nối vào/ra (*I/O interface*) là các mạch điện tử, thực hiện các giải pháp kĩ thuật thích ứng tín hiệu, tốc độ… cho phép CPU trao đổi dữ liệu với các thiết bị ngoại vi như bàn phím, màn hình, máy in, các bộ chuyển đổi số-tương tự DAC (*Digital/Analog Converter*), chuyển đổi tương tự-số ADC (*Analog/Digital Converter*), các mạch vào/ra dữ liệu dạng số DO (*Digital Outputs*), DI (*Digital Inputs*)…
- Hệ vi xử lý còn có một mạch tạo xung nhịp gọi là CPU Clock. Bộ tạo xung này được điều khiển bằng một mạch thạch anh có tần số thích hợp và đảm bảo tần số làm việc ổn định, với tần số chính xác cao. CPU có một chân phát xung cho các vi mạch còn lại, gọi là đồng hồ hệ thống (*System Clock*), đồng hồ này nối tới tất cả các vi mạch điều khiển chính trên bo mạch, đồng bộ toàn bộ hoạt động với CPU.
- Một khối nguồn nuôi (*Power Supply*) cung cấp năng lượng cho hệ thống từ mạng điện lưới, hay sử dụng pin. Bộ nguồn của các hệ vi xử lý thông thường là bộ nguồn xung với kỹ thuật đóng-ngắt dùng linh kiện bán dẫn công suất (*Switching Power Supply*), vừa gọn nhẹ, công suất đủ lớn, hiệu suất cao.



Hình 1.2-Nguồn nuôi cho hệ máy tính

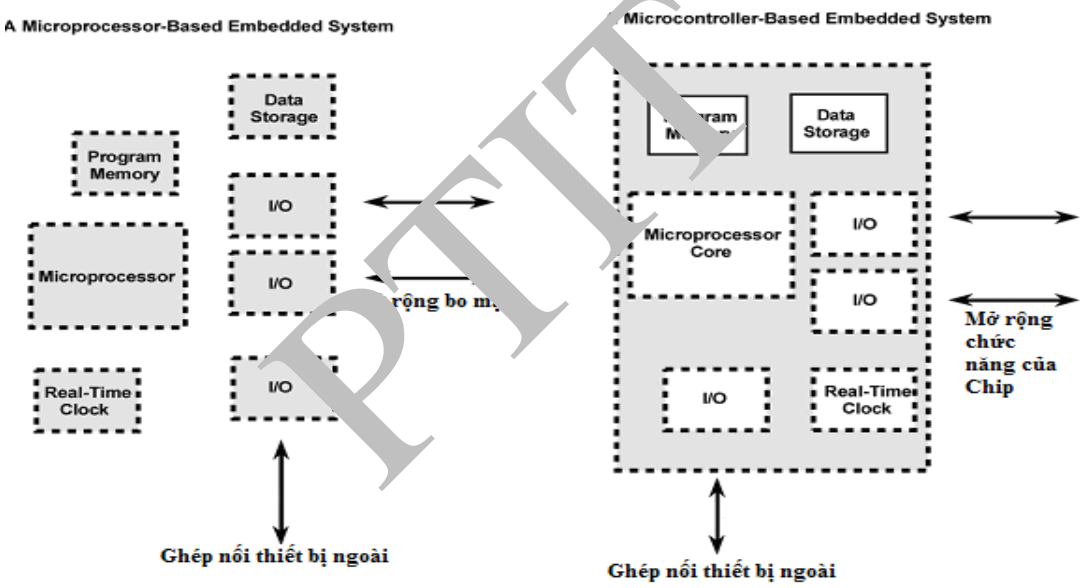
- Phân loại trên cơ sở vi mạch

  Tuy nhiên khi đề cập tới cấu trúc của HTN được xây dựng trên công nghệ vi mạch, có thể có hai loại khác biệt:

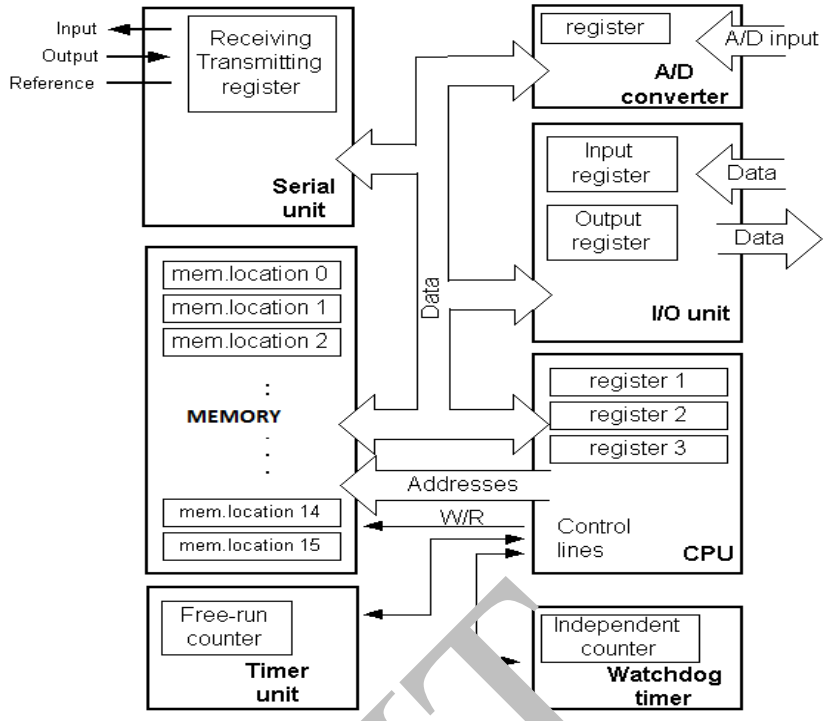
Vi điều khiển (Microcontroller) và **vi xử lí (microprocessor)** khác nhau theo vài cách, **trước hết là chức năng.** Để sử dụng một vi xử lí, cần hợp nhất nhiều thành phần vào đó: bộ nhớ, I/O, COM … và do vậy vi xử lí là trung tâm của cả hệ thống. Trong khi đó vi điều khiển được thiết kế theo tiêu chí: tất cả được tích hợp trong một vi mạch, đặc biệt là các giao tiếp I/O, tức là sẽ không cần thêm thành phần nào vào đó. Lợi thế là ở chỗ **tiết kiệm thời gian thiết kế** và **kích thước sẽ nhỏ gọn** rất nhiều. Tuy nhiên do phần cứng hạn chế, nên **lĩnh vực ứng dụng** cũng phải được xác định.

Hệ thống nhúng dựa trên bộ vi xử lý trên bo mạch (*Microprocessor-based Embedded System*)	**Hệ thống nhúng với vi điều khiển trên bo mạch (*Microcontroller-based Embedded System*)**
▪ Có CPU độc lập, có thể là CPU đa năng phổ biến (Intel 8080/8085, Motorola 6800…), ▪ Có RAM, ROM, định thời , I/O độc lập, ▪ Khả năng mở rộng RAM, ROM, I/O tùy ý, ▪ Đa năng, đắt tiền. ▪ Kiến trúc gần như một máy tính	▪ CPU dạng lõi chuyên biệt, RAM, ROM, định thời, I/O trong một vi mạch đơn, ▪ RAM, ROM có dung lượng cố định, I/O đủ cho mục đích sử dụng, ▪ Đơn mục đích, ứng dụng xác định, tiêu hao ít năng lượng, giá cả hợp lí cho ứng dụng nhúng. ▪ Kích thước nhỏ, gọn.

nhưng có kích thước nhỏ. *Thiết kế chức năng:* ▪ Cần phải có các vi mạch RAM, ROM hợp thành từ bên ngoài vi mạch. ▪ Không thể kết nối với ngoại vi ngoại vi, cần có thêm các vi mạch hỗ trợ cho chức năng này. ▪ Tuy nhiên năng lực tính toán mạnh.	*Thiết kế chức năng:* ▪ **Được thiết kế để có tất cả trong một Chip !** ▪ Năng lực tính toán được thiết kế tối ưu cho ứng dụng xác định, các ứng dụng chuyên biệt. ▪ Rất phù hợp để xây dựng các HTN ▪ Tiết kiệm gian thiết kế.



Hình 1.3 HTN xây dựng từ xây dựng từ vi xử lý(Microprocessor-based) và vi điều khiển (microcontroller based)



Hình 1.4 Chip Microcontroller và các thành phần cơ bản, BUS kết nối bên trong.Tất cả trong một chip.

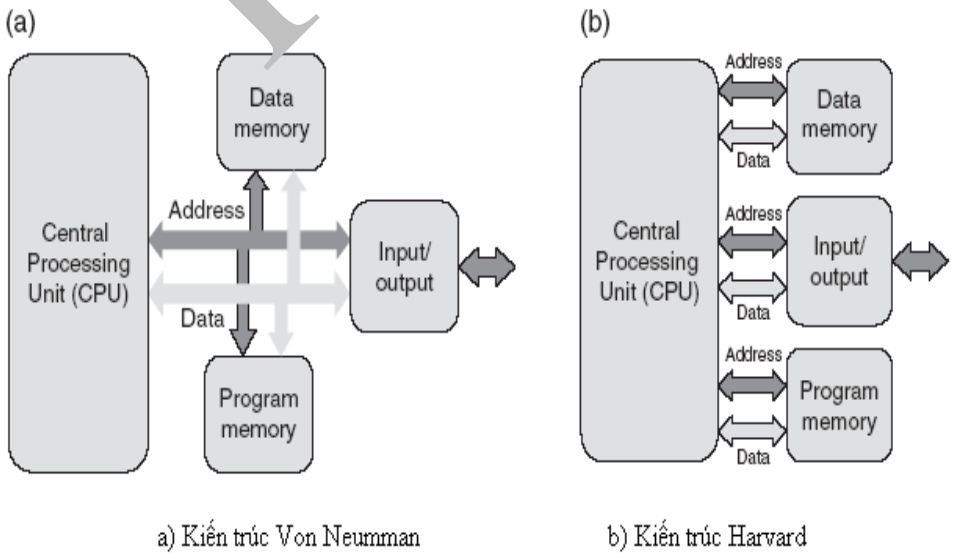
1.4.2 Kiến trúc của CPU

Phần nhiều các tài liệu khi đề cập tới thiết kế HTN, đều dành một số chủ đề về kiến trúc và cách thiết kế chế tạo CPU. Đây là một vấn đề chuyên và rất sâu, tài liệu này sẽ không dẫn xuất. Tài liệu chỉ giới hạn ở giới thiệu các kiểu CPU có thể sử dụng để thiết kế HTN. Các kiểu CPU này rất phổ biến trên thị trường, rất đa dạng với năng lực xử lý khác nhau và phù hợp cho mỗi loại ứng dụng nhúng. Dưới đây là một trong các quan điểm nhìn nhận CPU:

- ✓ Tập lệnh: có thể là CISC hay RISC, trong đó RISC là phổ biến.
- ✓ Hoạt động theo kiểu **Von Neumman**, ví dụ điển hình như hình vẽ trên, trong đó Hệ thống BUS địa chỉ và BUS dữ liệu, BUS điều khiển chung cho toàn bộ hệ thống, bộ nhớ chia sẻ chung cho toàn hệ thống với vùng mã lệnh (code) và dữ liệu (data) trên cùng không gian địa chỉ bộ nhớ và BUS dữ liệu không thể truyền đồng thời mã lệnh và dữ liệu cùng một thời điểm. Quá trình thực hiện một lệnh máy như sau:
    1) Đọc mã lệnh từ ROM/RAM qua BUS dữ liệu vào CPU, giả mã để xác định làm gì tiếp theo;
    2) Đọc dữ liệu tiếp theo là một phần của lệnh (operands) nếu có;
    3) Thực hiện lệnh khi đã đọc hết các operands của lệnh;
    4) Lưu kết quả vào thanh ghi hay ra RAM.

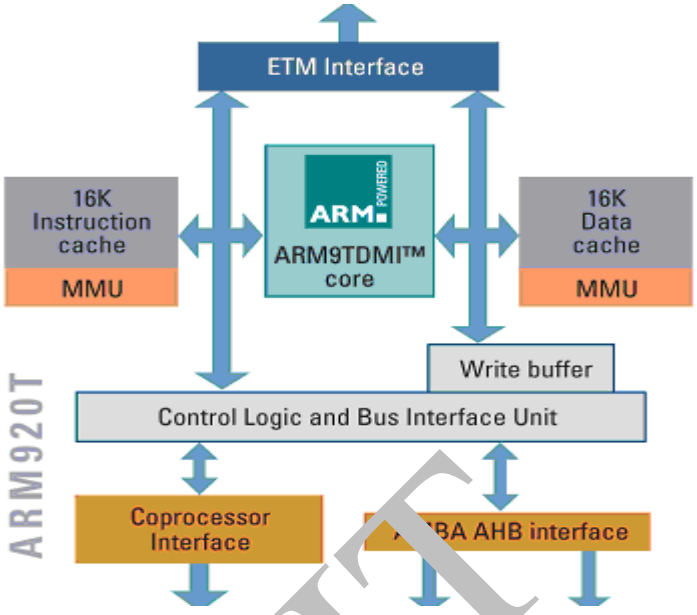
Trong qua strình thực hiện lệnh, BUS dữ liệu là kênh duy nhất để trao đổi dữ liệu, do vậy ta nói BUS dữ liệu có thể bị "bảo hòa", hiệu năng tính toán bị hạn chế. Với các CPU hiện đại BUS dữ liệu được cải tiến rất nhiều, đặc biệt là giao thức BUS và đồng hồ BUS được nâng cao để cải thiện hạn chế nói trên.

- ✓ Và hoạt động kiểu **Harvard** với một số đặc điểm khác biệt :
  - ✓ Howard Aiken (1900-1973) khi xây dựng máy tính với các relé đã tách các bộ nhớ dữ liệu (RAM) và bộ nhớ chương trình (NVM *Non Volatile Memory*: ROM, FLASH) với các bus riêng rẽ để truy cập vào bộ nhớ dữ liệu (RAM) và bộ nhớ chương trình chứa phần mềm nhúng (hệ điều hành, Device drivers, ứng dụng nhúng).
  - ✓ Bus cho lệnh và Bus cho dữ liệu tách biệt, điều hành độc lập nên lệnh và dữ liệu có thể được đưa ra cùng một lúc, cải thiện tốc độ so với thiết kế với chỉ một bus.
  - ✓ Phân biệt rõ ràng bộ nhớ dữ liệu và bộ nhớ chương trình, CPU có thể vừa đọc một lệnh, vừa truy cập dữ liệu từ bộ nhớ cùng lúc.
  - ✓ Do các BUS độc lập, CPU có khả năng tìm trước các lệnh (*instruction prefetch*), nên với kiến trúc Harvard chương trình chạy nhanh hơn, bởi vì nó có thể thực hiện ngay lệnh tiếp theo khi vừa kết thúc lệnh trước đó.
  - ✓ Tuy nhiên về kiến trúc có phần phức tạp hơn trong phần cứng, với 2 BUS độc lập gây khó khăn nhất định cho các ngôn ngữ lập trình cấp cao (ví dụ như C,…) nhưng cho hiệu quả hơn cho các ứng dụng nhúng. Là loại phổ biến để thiết kế các HTN.



Hình 1.5 Hai kiểu HTN với 2 loại kiến trúc CPU

Ví dụ Havard CPU ARM 920T, là loại CPU cải tiến với một nhớ cache cho lệnh và một cache cho dữ liệu.

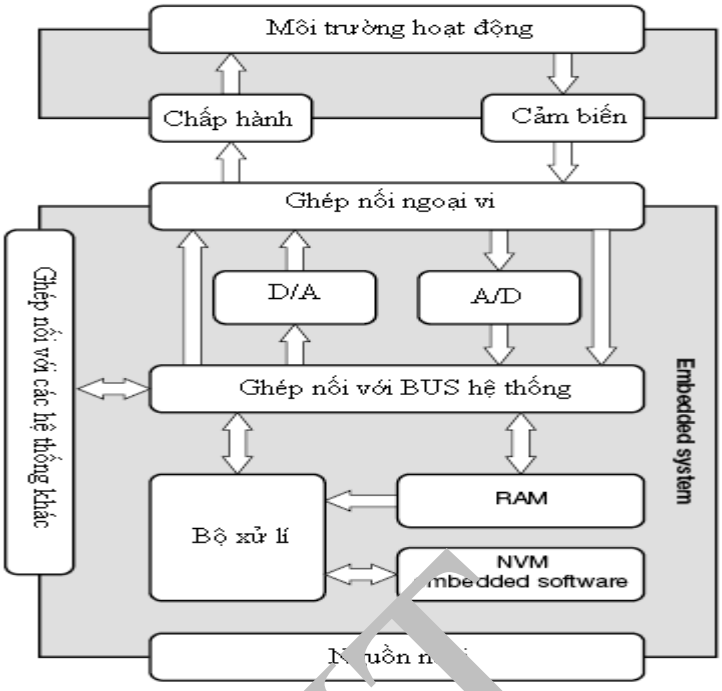


Hình1.6 Havard CPU ARM 920T của Amtel

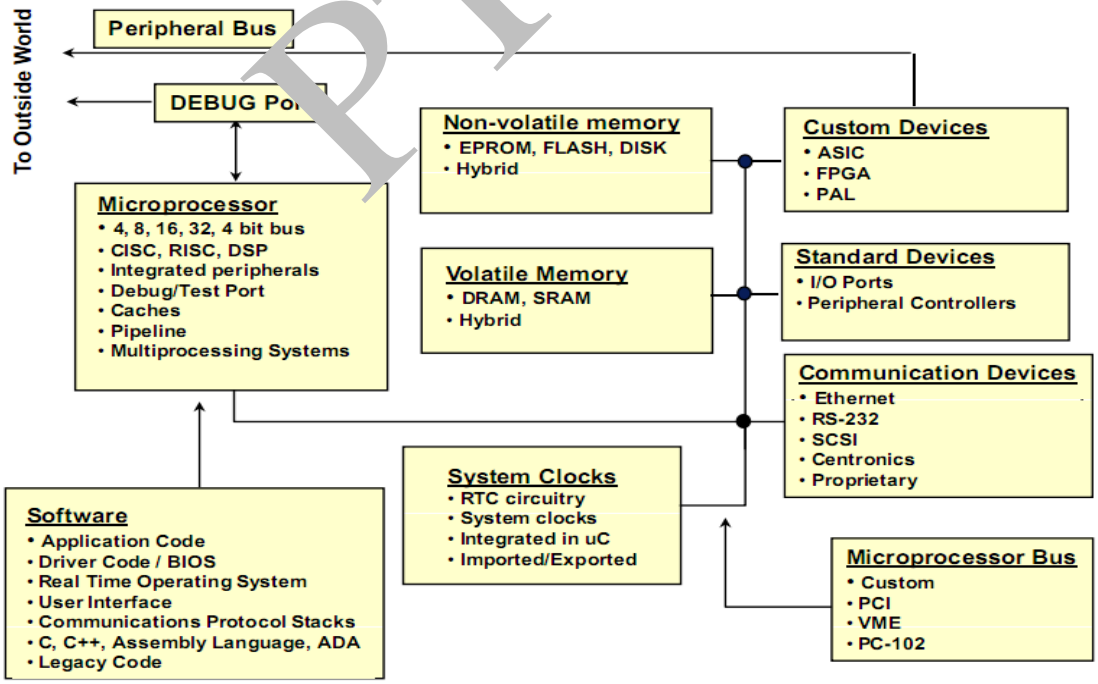
1.4.3 Mô hình tổng quát của một HTN

✓ *Các khối chức năng:*

- Môi trường hoạt động: nơi sử dụng HTN,
- Chấp hành: là các thiết bị công nghệ,
- Cảm biến: thiết bị đặc biệt ghi nhận thông tin công nghệ (vị trí, vòng quay, tốc độ, nhiệt độ, áp suất, kích thước (cao, dài, sâu) …) ,
- Ghép nối: là các thiết bị phối hợp, chuyển hóa các thông tin từ cảm biến thành tín hiệu điện để số hóa,
- Các bộ số hóa (A/D) và tương tự hóa (D/A),
- Ghép nối với các hệ thống khác: liên kết các HTN khác, mạng dữ liệu, Trung tâm điều khiển SCADA, …
- Ghép nối BUS hệ thống
- CPU, RAM, ROM (FLASH),



Hình 1.7 Mô hình tổng quát HTN- Mô hình với các khối chức năng



Hình 1.8 Một cách nhìn khác về mô hình tổng quát HTN:Với các khối ngoại vi và phần mềm

✓ *Kiến trúc trừu tượng: lớp xếp chồng*

Khi nói về kiến trúc một hệ thống, thường ta đề cập tính tổng quát và những chức năng cơ bản. Như vậy ở mức độ tổng quát, các lớp phần cứng và phần mềm được đề cập như các thành phần (*element*) hợp thành. Các thành phần kiến trúc có thể hợp nhất bên trong thiết bị nhúng hoặc tồn tại bên ngoài và tương tác với các thành phần bên trong theo một cách nào đó. Ở cách nhìn kiến trúc, thì kiến trúc được biểu diễn bởi các cấu trúc. Mỗi cấu trúc bao gồm một tập hợp các thành phần đặc trưng, các thuộc tính và những đặc tả về mối quan hệ bên trong các thành phần đó.

Kiến trúc lớp xếp chồng có *đặc tính là mỗi lớp chỉ sử dụng chức năng(hay dịch vụ) của tầng dưới nó, đồng thời chỉ cho phép tầng trên sử dụng các chức năng (dịch vụ) của mình.* Kiến trúc này có lợi thế về an ninh, bền vững, đơn giản về thiết kế, dễ nâng cấp (các dịch vụ), thực hiện mỗi lớp và khả năng nâng cấp "nóng" ngay cả khi hệ thống đang hoạt động. Ví dụ nguyên lí này ta thấy ở mô hình mạng chuẩn OSI (*Open Systems Interconnection),* kiến trúc của hệ điều hành Unix/Linux.

Mô hình một NTH cũng được đặc tả theo [illegible] kiến trúc để đảm bảo về độ tin cậy, đơn giản khi hoạt động:

Hình 1.9 Kiến trúc trừu tượng HTN

- Lớp phần cứng: Như ở *Các khối chức năng*.
- Lớp phần mềm hệ thống: Hệ điều hành hay Monitor
- Lớp ứng dụng: Là một số chương trình ứng dụng xác định mà HTN thi hành.

1.4 PHÂN LOẠI HTN

Phân loại HTN có thể theo nhiều tiêu chí khác nhau và có thể không hoàn toàn giống nhau (giống như khi nêu định nghĩa về HTN). Tuy nhiên có thể nêu ra đây một số tiêu chí để phân loại HTN.

- HTN hoạt động ở đâu:
  - ✓ Hoạt động độc lập: nhận đầu vào từ các tác nhân bị điều khiển, xử lý và cho đầu ra. Thời gian có đầu ra (đáp ứng) phải trong một khung thời gian nhất định theo ý đồ khi thiết kế.

- ✓ Hoạt động có liên kết với nhau giữa các HTN và các trung tâm kiểm soát khác. Loại này gọi là HTN mạng. Ví dụ các HTN cục bộ tại các thiết bị chấp hành đầu cuối của một qui trình công nghệ phức tạp liên kết qua mạng cục bộ của nhà máy hay của một cổ máy phức tạp. Hệ thống mạng điện thoại di động là một ví dụ kiểu HTN mạng: máy người dùng <–> các trạm BTS <–> tổng đài <–> tổng đài <–> BTS <–> máy người dùng. Tên chung của HNT lại này là HTN di động.

- Lĩnh vực ứng dụng:
  - ✓ Công cụ tính toán như các máy tính nhưng chỉ để chạy các bài toán nhất định.
  - ✓ Xử lý tín hiệu: các thiết bị video thời gian thực, DVD player, thiết bị y tế…
  - ✓ Truyền thông, mạng: thiết bị mạng như router, chuyển mạch (switch), firewall….
  - ✓ Hệ thống điều khiển và thu thập dữ liệu.
- Kiến trúc và qui mô:
  - ✓ HTN qui mô nhỏ (Small Scale Embedded Systems) với các xác định như sau:
  - Phần cứng ít phức tạp, thiết kế với CPU dạng loại 4, 8 bits;
  - Phần mềm đơn giản, dùng một monitor để kiểm soát hoạt động;
  - Công cụ phát triển phần mềm soạn thảo chương trình, hợp ngữ và hợp ngữ chéo (asembler, cross asembler) môi trường phát triển hợp nhất (integrated developememt enviroment) sử dụng với vi điều khiển hay CPU đã chọn. Ngôn ngữ phát triển là C, mã C được dịch ra mã phân tổ hợp, định vị mã thực thi trong bộ nhớ ROM, dung lượng bộ nhớ giới hạn.
  - Tiêu thụ năng lượng rất ít.

  Ví dụ : HTN đơn giản, chỉ có một vài phím bấm để đưa thông tin vào, một vài đèn LED hiện đầu ra (trạng thái nào đó). Ví dụ: HTN máy điều hòa nhiệt độ, lò nhiệt v.v.
  - ✓ HTN qui mô phức tạp:
  - Phần cứng phức tạp: Thiết kế với CPU 8,16 hay32 bits, hay sử dụng vi điều khiển;
  - Hệ thống có cấu trúc với BUS mở rộng để ghép nối với các thiết bị ngoại vi;
  - Phần mềm nhúng tinh vi, có hệ điều hành để thực hiện các nhiệm vụ, thao tác đồng thời. Có thể là loại RTOS.
  - Công cụ lập trình: C/C++/Visual C++/Java, RTOS, mã nguồn, công cụ ki thuật: Simulator, Debugger. Môi trương phát triển hợp nhât (Integrated Development Envirinment-IDE. Công cụ soft để xây dựng phần cứng phức hợp.

    Ví dụ: các HTN trên các máy gia công (kim loại, khuôn nhựa v.v).
  - ✓ HTN tinh vi (Sophisticated Embedded Systems)
  - Phần cứng và phần mềm rất đặc biệt;

- Nhiều CPU và có thể mở rộng, hay các CPU có thể cấu hình được (configurable CPUs), hay mảng logic lập trình được (programable logic array-PLA);
- Phát triển cho các lớp ứng dụng mới nhất khi các ứng dụng loại này cần phải 8ểcó quá trình thiết kế đồng thời giữa phần cứng và phần mềm, hợp nhất các linh kiện ở hệ thống cuối cùng, sử dụng công nghệ ASIC để chế tạo CPU, vi mạch đồng xử lý (còn gọi là ChipSet hay Co-processor).

Ví dụ các hệ thống hàng không quân sự mới nhất trên các máy bay (military/civil avionic), các thiết bị mạng cao cấp…. Các HTN kiểu này bị chế ngự bởi tốc độ xử lý của phần cứng (CPUs), Các chức năng phần mềm như các giải thuật mã hóa/giải mã, giải thuật chuyển đổi tín hiệu số (*Fourrier/Z transformation*), giao thức TCP/IP stack, các hàm chức năng mạng nhúng trong phần cứng để tăng tốc xử lý; Một số chức năng phần mềm được cứng hóa (như DSP). Công cụ phát triển thường không có sẵn vì đắt tiền, do đó phải phát triển riêng khi dự án được chấp nhận.

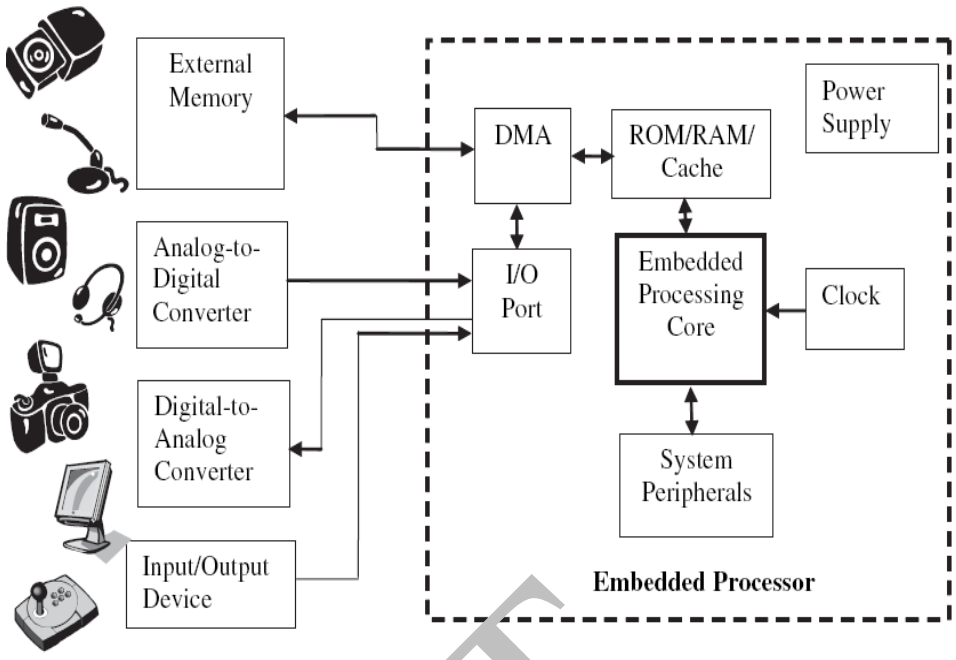
- ✓ HTN phần cứng hay HTN phần mềm;
- ✓ HTN theo An toàn sự cố (fail-safe), hay t[illegible]n toàn (fail-safe operational);
- ✓ HTN đáp ứng được bảo đảm hay đáp ứng với nỗ lực tối đa;
- ✓ HTN với nguồn tài nguyên đầy đủ hay nguồn tài nguyên hạn chế;
- ✓ HTN phản ứng ngay với sự kiện hay phản ứng với sự kiện có thời hạn.

- **Tại sao các HTN lại có sự khác nhau ?**

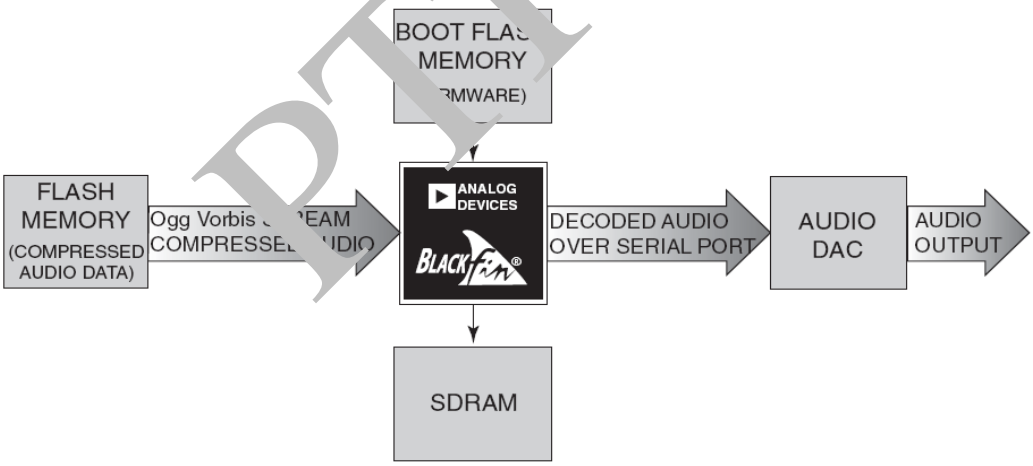
Có thể trả lời đơn giản đó là do các ứng dụng khác nhau và hệ thống phải hoạt động hiệu quả. Ví dụ:

1. HTN là dành để thực hiện các tác vụ riêng biệt. Các tác vụ riêng biệt ở đây phần lớn liên quan tới *các xử lý khác nhau chuyên biệt*, các sự kiện, các trạng thái của một qui trình công nghệ, ví dụ qui trình điều khiển máy công cụ, robot … Khi thay đổi qui trình, thường dẫn tới thay đổi hay thiết kế lại cả hệ thống. Như vậy có thể thấy cần có một loại bộ xử lý thích hợp cho lại tác vụ đã nêu. Bộ xử lý như vậy gọi là bộ xử lý chuyên biệt (*dedicated microprocessor*), nó không mạnh như bộ xử lý đa năng ta sử dụng trong máy tính, như máy tính PC chẳng hạn. Ví dụ điển hình là bộ xử lý tín hiệu số DSP (*Digital Signal Processor*) dòng TMS320 (*TMS320C6000*™ Multicore DSPs, TMS320DM6446 DaVinci™ Video Processor …) của Texas Instruments, hay các bộ xử lý MP3 (xử lý dữ liệu âm thanh đã nén và giải mã đưa vào khuyếch đại âm thanh rồi ra loa).

   Ví dụ sơ đồ khối chức năng của CPU DSP:



Hình 1.10 Sơ đồ khối CPU DSP-MP3.

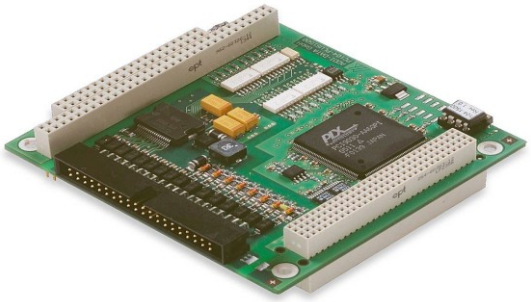


Hình 1.11 Bộ MP3 với CPU BlackFin của ANALOG DEVICES

2. Như điểm 1. nêu trên, về phần cứng, các HTN được thiết kế từ rất nhiều loại CPU nhúng và các CPU nhúng bản thân chúng lại có kiến trúc khác nhau. Hiện trên thị trường có thể liệt kê các kiểu CPU nhúng như: CPU vạn năng rút gọn phù hợp cho ứng dụng nhúng, các vi điều khiển (*microcontroller, PIC*), các kiến trúc kiểu hệ thống trên một vi mạch (*PSoC-Programmable System on Chip*)…
3. Về phần mềm cở sở có thể từ đơn giản cho tới tinh xảo, hệ điều hành thời gian thực (*RTOS-Real Time Operating System*).

4. Ví dụ một số HTN

- Phần cứng thương mại là các sản phẩm như bo mạch HTN (*Microprocessor-based)* đầy đủ, thiết kế hoàn chỉnh và đi cùng đã có phần mềm hệ thống (hệ điều hành, hay monitor) cài đặt trong EEPROM, flash, có RAM hay còn có đĩa cứng bán dẫn (DiskOnChip dung lượng 32MB hay lớn hơn), ví dụ như dòng PC 104 dưới đây:



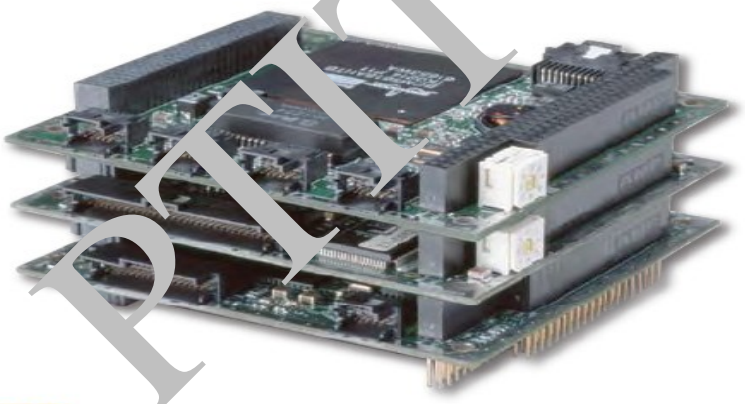


Figure 3 Cisco 3200 Series Wireless and Mobile Routers for mobile applications, based on PC/104 stacking technology, run the company's IOS software and allow the networking of multiple wireless devices that run a variety of communications links.

TMZ104
PC/104 Computer with Transmeta Crusoe TM5500 CPU
TMZ104 Photo Features:
Low power fanless x86 compatible
Embedded BIOS
Linux OS
Watchdog timers
Dual EIDE & floppy support
USB, parallel port,
PS2 keyboard & Mouse
Dual RS-232 serial

Hình 1.12 Một số HTN thương mại

- Các loại vi điều khiển dạng bo mạch.
- Các loại vi xử lý rời để thiết kế THN theo yêu cầu ứng dụng:

  CPU : Intel 80X51, PIC 12F 675, Amtel 8051, PSoC (Programmable System on Chip) : CY8C29466… Đặc điểm chung của các loại này là có kiến trúc đầy đủ chỉ trong một vi mạch có mức độ tích hợp cỡ lớn và rất lớn (VLSI: *Very Large Scale Integration*).

  Bảng dưới đây liệt kê số loại CPU cho HTN từ các hãng khác nhau (chưa đầy đủ):

Architecture	Processor	Manufacturer
AMD	Au1xxx	Advanced Micro Devices,...
ARM	ARM7, ARM9, ...	ARM, ...
C16X	C167CS, C165H, C164CI,...	Infineon,...
ColdFire	5282, 5272, 5307, 5407, ...	Motorola,...
I960	I960	Vmetro, ...
M32/R	32170, 32180, 32182, 32192, ...	Renesas/Mitsubishi, ...
M Core	MMC2113, MMC2114, ...	Motorola
MIPS32	R3K, R4K, 5K, 16, ...	MTI4kx, IDT, MIPS Technologies, ...
NEC	Vr55xx, Vr54xx, Vr41xx	NEC Corporation, ...
PowerPC (PPC)	82xx, 74xx, 8xx, 7xx, 6xx, 5xx, 4xx	IBM, Motorola,...
68k	680x0 (68K, 68030, 68040, 68060 ... 683xx	Motorola, ...
SuperH (SH)	SH3 (7702, 7707, 7708, 7709 ... H4 (7750)	Hitachi, ...
SHARC	SHARC	Analog Devices, Transtech DSP, Radstone, ...
strongARM	strongARM	Intel, ...
SPARC	UltraSPAR... II	Sun Microsystems, ...
TMS320C6xxx	TMS320C...	Texas Instruments, ...
x86	X86 [386, 486, P... n (II, III, IV)...]	Intel, Transmeta, National Semiconductor, Atlas, ...
TriCore	TriCore1, TriCore2, ...	Infineon,...
MicroChip	PIC10 .... PIC32 , 8 bits , 16 bits and 32 bits	MicroChio Technology

1.5 KẾT CHƯƠNG

Chương này giới thiệu về HTN từ định nghĩa, mô hình, phân loại, các đặc thù và kiến trúc của HTN. Chương cũng nêu ra những lĩnh vực công nghệ mà HTN được sử dụng. Từ đó cũng bật ra những yêu cầu kỷ thuật trên HTN nói chung và HTN cho ứng dụng riêng biệt.

Tham khảo một số định nghĩa HTN từ các nguồn tài liệu:

"HTN là một hệ thống có phần cứng được xây dựng trên nền tảng phần cứng máy tính chuyên biệt với phần mềm được nhúng trong phần cứng đó, như một trong các thành phần quan trọng nhất của HTN. HTN do đó có thể là một hệ độc lập hay là một phần của một hệ thống lớn hơn."

Một số định nghĩa về HTN:

Wayne Wolf: " what is an embedded computing system ? Loosly defined, it is any device that includes a programmable computer but is not itself intended to be general-purpose computer" [Computers as Components – Principal of Embedded Computer System Design].

Told D. Morton: "Embedded Systems are electronic systems that contain a microprocessor or microcontroller, but we do not think od them as computers – the computer is hidden or embedded in the system" [Embedded Microcontrollers]

Tim Wilmshurst: " Embedded system is a system whose primcipal function is not computational, but which is controlled by a computer embedded within it. The computer is liklly to be a microprocessor or micro controller. The word ***embedded*** *implies that it lies inside the overal system, hidden from view, forming an integral part of greater whole" [An Introduction to the Design of Small Scale Embedded System with PIC, 80c51 and 68HC05/08 Microcontroller]*

1.6 CÂU HỎI CUỐI CHƯƠNG

1) Định nghĩa tương đối về HTN.
2) Những thách thức nào phải đối mặt khi thiết kế một HTN ?
3) Những cách để nêu mô hình kiến trúc của một HTN ?
4) Thế nào là HTN kiểu vi xử lý HTN kiểu vi điều khiển ?
5) Nêu các khối chức năng dựng hình 1.7 mô tả về mô hình một HTN.
6) Có bao nhiêu loại kiến trúc CPU được sử dụng khi xây dựng HTN ? Mỗi loại khác nhau ở điểm nào chủ yếu ?
7) Nêu các thành phần phần cứng thường có trong một HTN ?
8) Các thành phần như biến đổi tương tự-số (ADC), số-tương tự (DAC), định thời (timer), cổng (port) nhất thiết cần có trên một HTN ? Tại sao ?
9) Có các loại phần mềm nào trên một HTN ?

Chương 2. CÁC THÀNH PHẦN PHẦN CỨNG CỦA HỆ THỐNG NHÚNG

Chương 2 đề cập tới các thành phần phần cứng, nền tảng cơ sở của HTN. Bao gồm: bộ xử lý trung tâm (CPU) với kiến trúc Von Neumman và kiến trúc Harvard, BUS của CPU và BUS hệ thống, bộ nhớ, cổng. Tiếp theo là kĩ thuật ghép nối các thiết bị ngoại vi vào với CPU, các chương trình điều khiển ghép nối. Đối với CPU, tài liệu nêu nguyên lí kiến trúc, các đặc tính kĩ thuật và biểu đồ thời gian hoạt động của CPU, giúp cho việc thiết kế phần cứng sau này. Riêng về tập lệnh không đề cập tới, do vậy khi sử dụng một CPU nào đó cần nắm được tập lệnh của CPU đó để có thể lập trình, viết các trình điều khển bằng hợp ngữ.

Nội dung chương cung cấp các kiến thức phần cứng và kĩ năng thiết kế, đặc biệt là thiết kế ghép nối với thiết bị. Cuối chương là một số bài tập thiết kế đơn giản như thiết kế bộ nhớ ROM, RAM, cổng với CPU, tạo thành một bo mạch như một HTN chưa có các thiết bị ngoài.

2.1 BỘ XỬ LÍ TRUNG TÂM (Central Processing Unit-CPU)

Các hệ thống sử dụng kĩ thuật tính toán số để xử lý thông tin đều cần một tổ hợp các mạch số để tạo ra một hệ thống có khả năng: thực hiện các phép tính số học, luận lí (logic), các phép toán số học, các quyết định chuyển hướng thực hiện có hay không có điều kiện … và quan trọng là hoạt động theo một hệ mã vi lệnh (*micro-instruction code*) theo một trình tự nhất định. Ý tưởng tạo ra một hệ thống điện tử số như vậy chính là tạo ra một bộ xử lý trung tâm. Bộ xử lý trung tâm ngày nay rất tinh xảo, kích thước rất nhỏ (chỉ lớn hơn 1 cm^2) nhưng chứa vài trăm triệu transistor, hoạt động với tần số từ vài MHz tới vài GHz, công suất tiêu tán từ vài Watt tới vài chục Watt , ví dụ: *Intel Pentium G0.50 công nghệ: Clarkdale (32 nm), tần số:2.8 GHz,Powerdissipation:73 W (http://en.wikipedia.org/wiki/List_of_CPU_power_dissipation #Intel_Pentium_Dual-Core*). Đổi lại, tốc độ tính toán đạt hơn 1 tỉ lệnh máy trong một giây (*MIPS: milions Instruction per second*). Ví dụ: *Intel Core i7 Extreme Edition i980EE :147.600 MIPS at 3,3 GHz , 44.7 lệnhmáy/chu kì xung đồng hồ (với f=3,3GHz, Tchukì=0,303 ns).*
(http://en.wikipedia.org/wiki/Instructions_per_second).
Đấy là nói tới các bộ xử lý vạn năng, dùng để chế tạo các máy tính (để bàn, máy chủ). Đối với các CPU dùng trong thiết kế các HTN, tần số làm việc của CPU có thấp hơn, từ vài chục, vài trăm MHz trở lên. Tại sao vậy ? Đơn giản không phải lúc nào cũng cần tốc độ tính toán thật nhanh, và còn tùy vào ứng dụng nhúng đó là gì. Tất nhiên càng nhanh càng tốt, nhưng đổi lại giá thành cao, môi trường hoạt động không thể thỏa mãn (ví dụ nhiệt độ môi trường cao, bụi, rung động cơ học …).

2.2.1 Các loại CPU và nguyên lí hoạt động

Hiện nay trên thị trường rất nhiều nhà sản xuất CPU với nhiều chủng loại phù hợp cho các ứng dụng. Có thể liệt kê một số nét để phân biệt:

a) Cách tổ chức và thực hiện lệnh máy:
   - Với tập lệnh đầy đủ (CICS).
   - Với tập lệnh rút gọn (RISC).

b) Cách xử lý thông tin và truy nhập bộ nhớ:
   - Von Neumman: bộ nhớ chung, truy cập tuần tự theo từng lệnh máy.
   - Harvard: bộ nhớ lệnh và bộ nhớ dữ liệu độc lập, truy cập đồng thời.

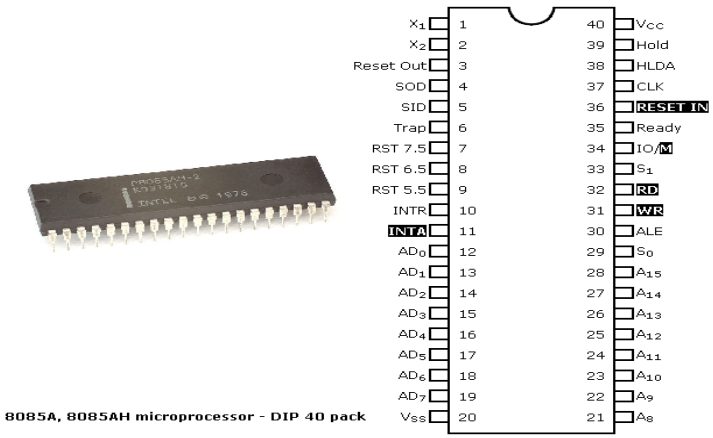
c) Công nghệ chế tạo hướng ứng dụng:
   - CPU đa năng: chế tạo máy tính đa năng, HTN tinh vi, hệ điều khiển thông minh.
   - CPU chuyên dụng: các ứng dụng đặc thù (như cho ứng dụng nhúng). Trong phạm trù này lại có một vài công nghệ tiêu biểu:
     - Vi điều khiển (microcontroller và PIC-*Programmable Intelligent Computer" (Máy tính khả trình thông minh*)).
     - Hệ thống trong một vi mạch lập trình được (PSoC- *Programmable System on Chip: integrating configurable analog and digital peripheral functions, memory and a microcontroller on a single chip).*

     Lớp công nghệ này được sử dụng phổ biến trong các HTN có đòi hỏi mức tinh xảo trung bình hay thấp.

2.2.2 Ví dụ về một CPU và nguyên lí hoạt động

Để có thể thực hiện thiết kế một HTN, cần tìm hiểu chi tiết về cấu trúc, cách làm việc và lập trình cho một CPU đã chọn. Phần này giới thiệu dòng **Intel CPU 808X, 8 bits hay 16 bits**, đa năng của Intel được sử dụng rất phổ biến trên thế giới cũng như ở Việt Nam. Ưu điểm nổi bật của CPU này là tính phổ biến, đa năng, dễ triển khai, công cụ phát triển rất đa dạng và sẵn có.

a) Sơ đồ hình thức bên ngoài:

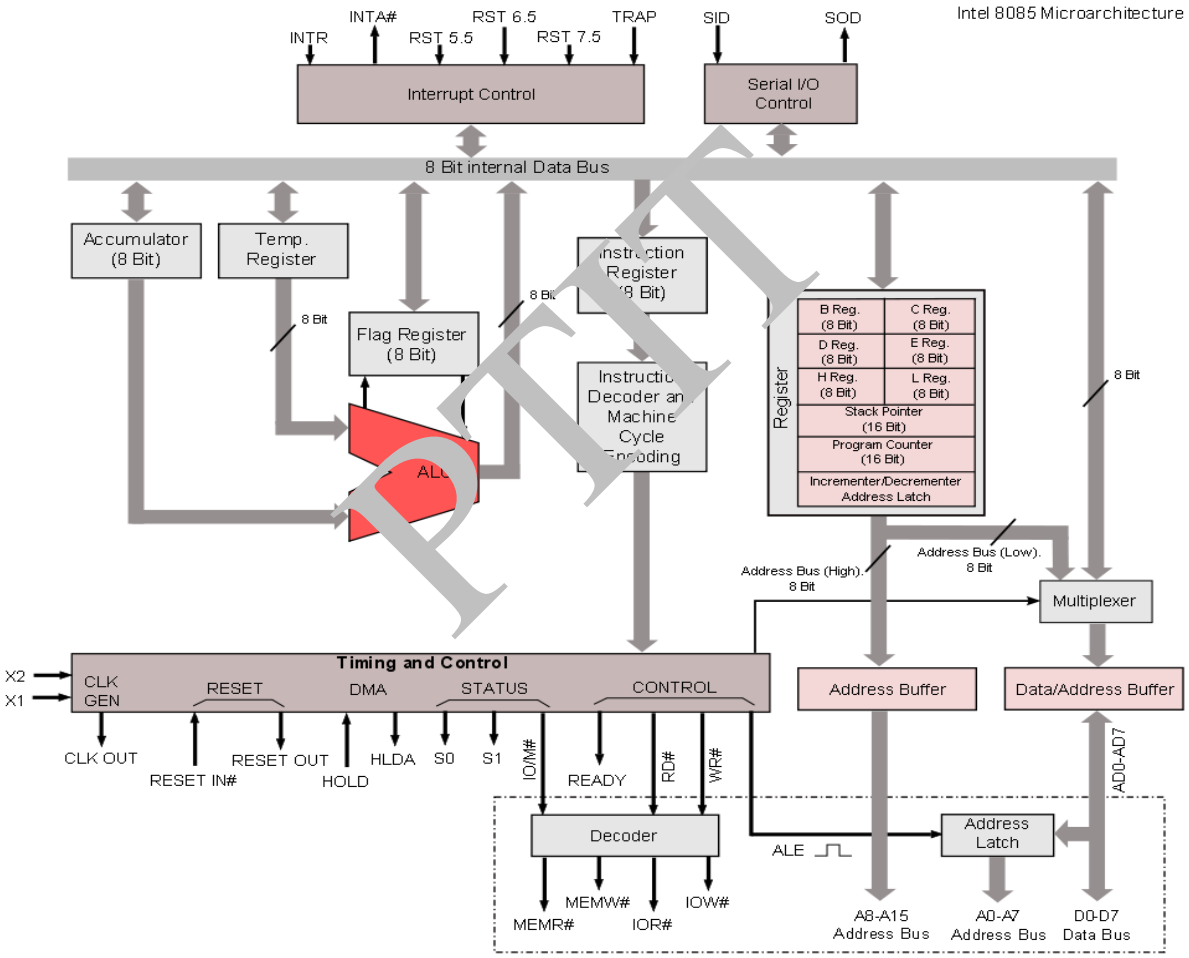


Hình 2.1 Intel CPU 8085

b) Kiến trúc: Mô hình chức năng bên trong CPU, ví dụ Intel 8080/8085

8085 là 8-bit microprocessor, trong đó dữ liệu xử lý là 8 bits, không gian địa chỉ được xác định bởi 16 bits, cho dung lượng địa chỉ là 65.535 (gọi là 64K) ô nhớ. Các thành phần chức năng bao gồm:

- Tập các thanh ghi (Register).
- Đơn vị thực thi các phép tính số học và luận lí (Arithmetic logic unit- ALU).
- Hệ thống các dây nối giữa các vi mạch chức năng (BUS).
- Khối định thời và điều khiển (Timing & Control unit).
- Khối điều phối, kiểm soát ngắt (Interrupt Control)
- Khối truyền thông nối tiếp (Serial I/O Control)



Hình 2.2 Các khối chức năng của CPU 8080/8085

1) Tập các thanh ghi (Registers):

Các thanh ghi sử dụng để chứa dữ liệu và địa chỉ. Co hai loại thanh ghi:

Thanh ghi đa năng được dùng như chức năng nhớ dữ liệu tạm thời hay đị chỉ tức thời qui chiếu tới bộ nhớ (ROM/RAM). Các thanh ghi 8 bits đó là B, C, D, E, H và L. Khi ghép lại sẽ thành thanh ghi 16 bits với tên kép: BC, DE or HL.
Thanh ghi đặc biệt là các thanh ghi gán cho chức năng đặc biệt (hay chuyên dụng):

- **Thanh ghi tích lũy ACC (Accumulator)** hay **A**, là thanh ghi 8-bit, đa dụng cho các thao tác và các phép tính số học, luận lí, I/O, nạp, nhớ dữ liệu, toán hạng, kết quả phép toán khi thực hiện lệnh.
- **Thanh ghi chỉ số** (**Index register**): được dùng làm chỉ số cho qui chiếu địa chỉ, cách dùng phụ thuộc vào chế độ địa chỉ hóa.
- **Thanh ghi đa năng (General registers**) 16 bit chia phần thấp và phần cao:
  - 8-bit B và 8-bit C hay kết hợp thành cặp BC 16-bit.
  - 8-bit D và 8-bit E hay kết hợp thành cặp DE 16-bit.
  - 8-bit H và 8-bit L hay kết hợp thành cặp HL 16-bit.

  Các thanh ghi này dùng độc lập hay kết hợp chứa dữ liệu hay địa chỉ qui chiếu vào bộ nhớ (chế độ dịa chỉ hóa gián tiếp qua thanh ghi).
- **Thanh ghi trạng thái** (Status register) hay Cờ (**Flag),** là thanh ghi 8-bit, chứa các bit có ý nghĩa sau:
  - Sign, lên 1 nếu bit lớn nhất của kết quả phép tính có giá trị = 1.
  - Zero, lập giá trị = 0, nếu kết quả phép tính = 0.
  - Auxiliary carry, với phép tính 4 bit (D3-D0), lên 1 nếu kết quả phép tính có số mang từ D4 chuyển sang D4.
  - Parity, lên 1 nếu số parity (là tổng của các bit trong kết quả) là số chẵn.
  - Carry, lên 1 nếu có số mang khi thực hiện phép cộng số học hay borrow khi thực hiện phép trừ số học hay phép so sánh giá trị.
- **Thanh ghi ngăn xếp** (**Stack pointer**): 16 bit. Thanh ghi này mỗi lần tăng hay giảm 2 (+/-2).
- **Thanh ghi lệnh máy** (**Instruction register**): 8 bit chứa mã lệnh (OPCODE) từ ROM/RAM, đầu vào cho khối Timing&Control để giải mã thành các tín hiệu điều khiển của CPU**.**
- **Thanh đếm chương trình (Program counter),** 16-bit, trỏ trực tiếp vào bộ nhớ nơi chứa mã lệnh của mỗi lệnh. Địa chỉ qui chiếu từ 0000h đến FFFFh (0 đến 65.535).

2) ***Đơn vị thực thi các phép tính số học và luận lí (ALU):***
Thực hiện các phép tính: Cộng, trừ, nhân, chia, logis AND, OR, XOR, NOT, dịch trái/phải, quay vòng trái/phải.
3) ***Hệ thống các dây nối giữa CPU với các vi mạch chức năng, thiết bị ngoài*** **(BUS)** (*Hình 1.1.1- Mô hình tổng quát bo mạch chủ)*
Tập hợp các tín hiệu phát ra từ CPU và nối tới các vi mạch chức năng trên bo mạch chính. Để mô tả ý nghĩa của từng tín hiệu ta nhóm lại theo chức năng như sau:

BUS địa chỉ (Address bus): Mang thông tin về dịa chỉ qui chiếu tới ROM/RAM, bộ giải mã chọn vi mạch. Với CPU 8080/8085 có tất cả 16 đường hay 16 bit. BUS này chỉ có một hướng (chiều) đi ra từ CPU.

BUS dữ liệu (Data bus) : Dữ liệu trao đổi giữa CPU và các vi mạch bên ngoài, các thiết bị ngoài sử dụng BUS này. Tùy loại CPU có thể là 8 bit, 16 bit, 32 bit, 64 bit. Số bit này thường dùng để nói tới loại CPU. Đặc điểm cơ bản của BUS là hai chiều.

BUS điều khiển (Control bus): Các tín hiệu điều khiển phát ra từ CPU tới các vi mạch chức năng khác nhau trên bo mạch chủ, các thiết bị ngoài nối với CPU. Các tín hiệu này được dùng để đồng bộ mọi bước hoạt động của máy tính.

4) Định thời và điều khiển

Khối này tạo ra tất cả các tín hiệu đồng hồ, các tín hiệu điều khiển bên trong CPU, CPU với bên ngoài qua Bus điều khiển (Control bus).

c) Bộ nhớ (Memory)

Chương trình ứng dụng, hệ điều hành, dữ liệu, ngăn xếp đều dùng chung không gian nhớ. Với họ 8080/8085 có 16 dây địa chỉ (A15 ….. A0) cho dung lượng nhớ tối đa là 65.535 địa chỉ ô nhớ. Nếu mỗi ô nhớ là 1 byte sẽ có 65.535 byte hay 64 K.

Kiến trúc sử dụng 64 bytes đầu tiên (000F-0000) để đặt các vector ngắt của các lệnh RST.

d) Ngắt (Interrupts)

CPU 8085 có 5 đầu vào tín hiệu ngắt (*interrupt*) trình bày theo thứ tự ưu tiên từ thấp đến cao:

INTR, che được. Khi có ngắt xuất hiện, CPU sẽ đưa lệnh trên BUS, lệnh đó có thể là trong các lệnh RESTART (đều có hiệu lực khởi động lại CPU) loại RST (RST 5.5, RST 6.5, RST 7.5 và TRAP). CPU bảo lưu giá trị của PC vào ngăn xếp, chuyển tới ô nhớ có giá trị N*8, trong đó N có giá trị từ 0 đến 7 mà lệnh RST cung cấp. Ô nhớ này chứa địa chỉ của chương trình xử lý cho ngắt đó như sau:

Tên của ngắt	Địa chỉ chương trình khởi động tại:
TRAP	24 Hex
RST 5.5	2C Hex
RST 6.5	34 Hex
RST 7.5	3C Hex (ưu tiên cao nhất, tác dụng sườn lên của xung ngắt)

Lệnh gọi CALL (lệnh có 3 byte). CPU "gọi" một chu trình con có địa chỉ xác định ở byte thứ 2 và thứ 3 của lệnh này.

RST5.5, che được. Khi có ngắt này, CPU bảo lưu giá trị của PC vào ngăn xếp, nhảy tới địa chỉ cố định 002Ch (h-hexadecimal).

RST6.5, che được. Khi có ngắt này, CPU bảo lưu giá trị của PC vào ngăn xếp, nhảy tới địa chỉ cố định 0034h.

RST7.5 che được. Khi có ngắt này, CPU bảo lưu giá trị của PC vào ngăn xếp, nhảy tới địa chỉ cố định 003Ch.

Trap, không che. Khi có ngắt này, CPU bảo lưu giá trị của PC vào ngăn xếp, nhảy tới địa chỉ cố định 0024h.

Tất cả các ngắt che được có thể lập không che hay che lập trình qua lệnh EI và DI. RST 5.5, RST6.5 và RST7.5 lập trình bằng lệnh SIM.

e) Cổng vào/ra (I/O ports)

Họ 8080/8085 có tất cả 256 cổng vào và ra, 256 cổng ra chạy theo lệnh IN hay OUT.

f) Tập lệnh (Instruction Set)

Tập lệnh của CPU Intel 8085 gồm các nhóm sau đây:

- Lệnh chuyển dữ liệu.
- Lệnh số học – cộng, trừ, tăng, giảm.
- Logic - AND, OR, XOR và quay vòng.
- Chuyển điều khiển đi có/không điều kiện, gọi chu trình, trở về chỗ cũ khi thoát khỏi chu trình, khởi động lại.
- Lệnh vào/ra (I/O).
- Các lệnh thao tác bit, cờ, cho phép/không cho phép ngắt, ngăn xếp, …

g) Chế độ địa chỉ (Addressing modes)

- Kiểu thanh ghi: qui chiếu dữ liệu trong 1 thanh ghi hay đôi thanh ghi.
- Gián tiếp qua thanh ghi: Thanh ghi chứa địa chỉ chỗ nơi có dữ liệu.

Qui chiếu trực tiếp – Dữ liệu 8 hay 16 bit.

i) Các nhóm tín hiệu trong CPU 8080/8085

A8 – A15. Nhóm tín hiệu ra: 8 bit cao của địa chỉ. Các chân này là các chân được nối với bên ngoài qua mạch 3 trạng thái. Các phần tử 3 trạng thái sẽ được đặt ở trạng thái trạng thái trở kháng cao (còn gọi là trạng thái không kết nối) trong các trường hợp một trong các tín hiệu HOLD hay HALT là tích cực.

AD0 – AD7. Nhóm tín hiệu 8 kênh cho các tín hiệu địa chỉ và tín hiệu dữ liệu theo chia sẻ thời gian, 3 trạng thái. Ở giai đoạn đầu của chu kỳ máy, T1 của M1, sẽ là byte thấp của 16 bit địa chỉ từ A0 đến A7.

ALE (Address Latch Enable). Tín hiệu ra qua mạch 3 trạng thái. Được sử dụng để chốt byte thấp của tín hiệu địa chỉ (A0 – A7) từ nhóm AD0-AD7. Tín hiệu này được tạo ra trong giai đoạn đầu tiên của chu kỳ máy, T1 của M1, và cũng được dùng để chốt các tín hiệu trạng thái S0 và S1 khi cần thiết.

S0, S1 (Data BUS Status). Là các tín hiệu chỉ trạng thái của các chân thuộc BUS dữ liệu trong mỗi chu kỳ máy. Tổ hợp của hai tín hiệu này cũng cho biết trạng thái của CPU như sau:

S1	*S0*	*Trạng thái hoạt động của BUS dữ liệu*
0	0	Trạng thái HALT
0	1	CPU đang thực hiện thao tác WRITE
1	0	CPU đang thực hiện thao tác đọc (READ)
1	1	CPU đang thực hiện thao tác nhận lệnh (Instruction Fetch)

RD (Read). Chân ra 3 trạng thái. Nằm trong nhóm tín hiệu điều khiển. Tín hiệu tích cực khi CPU tiến hành đọc dữ liệu từ bộ nhớ hoặc từ thiết bị ngoại vi. Trong chế độ HALT hoặc DMA, chân ra này ở trạng thái trạng thái trở kháng cao.

WR (Write). Chân ra 3 trạng thái. Nằm trong nhóm tín hiệu điều khiển. Tín hiệu tích cực khi CPU tiến hành ghi dữ liệu vào bộ nhớ hoặc đưa dữ liệu ra thiết bị ngoại vi. Trong các chế độ HALT hoặc DMA, chân ra này ở trạng thái trạng thái trở kháng cao.

IO/M. Trạng thái logic của đầu ra này cho biết CPU đang làm việc với thiết bị ngoại vi hay với bộ nhớ. Nếu là logic 1, CPU đang truy cập thiết bị vào/ra, còn nếu là 0, CPU đang truy cập bộ nhớ. Kết hợp với hai đầu ra RD và WR để tạo ra các tín hiệu I/OR, I/OW, MEMR, và MEMW trong trường hợp sử dụng địa chỉ tách biệt đối với thiết bị vào/ra. Nằm trong nhóm tín hiệu điều khiển, IO/M cũng là đầu ra 3 trạng thái.

Interrupts. P8085 có ngắt đa mức. Có 5 chân ngắt tất cả: (INTR, RST5.5, RST6.5, RST7.5 và TRAP). Ngoài chân ngắt không che được là TRAP, các chân khác đều có thể che hoặc không che nhờ lập trình phần mềm.

- ***INTR:*** Chân nhận yêu cầu ngắt từ bên ngoài được đáp ứng theo nguyên tắc quay vòng (polling) hoặc vector thông qua lệnh RST
- ***Các yêu cầu ngắt RST:*** Có 3 đầu vào yêu cầu ngắt với các mức ưu tiên khác nhau là RST7.5, RST6.5 và RST5.5. Khi yêu cầu ngắt xuất hiện tại các chân này, CPU tự động chuyển đến các vector ngắt tương ứng. Cụ thể như sau:
  - ***RST5.5*** là mức ưu tiên thấp nhất, phản ứng theo mức điện áp trên chân yêu cầu ngắt, địa chỉ vector ngắt này nằm ở ô nhớ có địa chỉ 2CH.
  - ***RST6.5:*** Ngắt có mức ưu tiên thấp thứ 2, phản ứng theo mức điện áp trên chân yêu cầu ngắt, địa chỉ vector ngắt này nằm ở ô nhớ 34H
  - ***RST7.5****:* Mức ưu tiên cao nhất. Phản ứng theo sườn lên của xung yêu cầu ngắt. Sườn lên của xung này tác động lên một Flip-Flop, mạch này giữ lại yêu cầu ngắt cho đến khi được xoá nhờ tín hiệu đáp ứng nhận biết yêu cầu ngắt (Acknowledge). Địa chỉ của vector ngắt này nằm ở ô nhớ 3CH

TRAP: Là chân nhận yêu cầu ngắt không che được (dĩ nhiên là nó có mức ưu tiên cao nhất). Địa chỉ của vector ngắt này ở ô nhớ 24H.

INTA. Tín hiệu ra nhận biết yêu cầu ngắt tại chân INTR. Các yêu cầu ngắt RST5.5, RST6.5, RST7.5 và TRAP không tác động đến INTA.

HOLD. Trạng thái logic 1 ở chân này là yêu cầu của thao tác DMA. Các đầu ra RD, WR, IO/M và ALE sẽ được đưa về trạng thái trở kháng ra cao.

HLDA. Tín hiệu nhận biết yêu cầu HOLD.

RESET IN. Logic thấp 0 ở đầu vào của chân này yêu cầu tái khởi động hệ vi xử lý. Do tác động của tín hiệu RESET IN tích cực, giá trị của thanh đếm chương trình **PC sẽ được nạp lại là 0000h**. Các mặt nạ ngắt và tín hiệu HLDA cũng được tái thiết lập về giá trị mặc định.

RESET OUT. Đầu ra nhận biết hệ vi xử lý được tái khởi động. Dùng tín hiệu này để tái khởi động toàn bộ hệ thống.

READY. Logic 1 ở đầu vào này thông báo trạng thái sẵn sàng cung cấp dữ liệu cho CPU hoặc nhận dữ liệu từ CPU của các thiết bị ngoại vi.

SID (Serial Input Data). Là cổng vào của dữ liệu nối tiếp của hệ Vi xử lý. Bit hiện diện tại cổng này được đọc vào CPU nhờ lệnh RIM, bit sẽ được đưa vào bit cao của Acc (MSB).

SOD (Serial Output Data). Bit cao (MSB) của Acc được truyền ra ngoài chân này khi sử dụng lệnh SIM.

X1, X2. Lối nối thạch anh hoặc một mạch dao động để tạo xung nhịp cho CPU. Có thể sử dụng thạch anh có tần số dao động trong khoảng từ 0.5 đến 3MHz.

CLK. Đầu ra của xung nhịp, có thể làm xung nhịp cho các thành phần chức năng khác trong hệ vi xử lý.

Vcc, Vss. Lối nối nguồn +5V và GND cho CPU 8085. Cũng cần nhắc lại rằng, CPU 8085 chỉ cần một nguồn nuôi duy nhất là +5V, khả năng cung cấp dòng của nguồn cần được thiết kế tuỳ theo nhu cầu của toàn hệ vi xử lý.

j) Biểu đồ thời gian (system timing)

Khi thiết kế phần cứng của một HTN nói riêng hay một thiết bị kĩ thuật số nói chung, khái niệm về biểu đồ thời gian là hết sức quan trọng. Nắm bắt được ý nghĩa của biểu đồ thời gian sẽ có ích khi tiến hành hiệu chỉnh và tìm lỗi phần cứng. Lỗi phần cứng thường xảy ra khi các tín hiệu hoạt động không đứng thời điểm ở đầu vào mạch số, gây ra *lỗi*, đặt biệt hay xảy ra ở các mạch tổ hợp. Dưới đây là một số định nghĩa của các bộ xử lý:

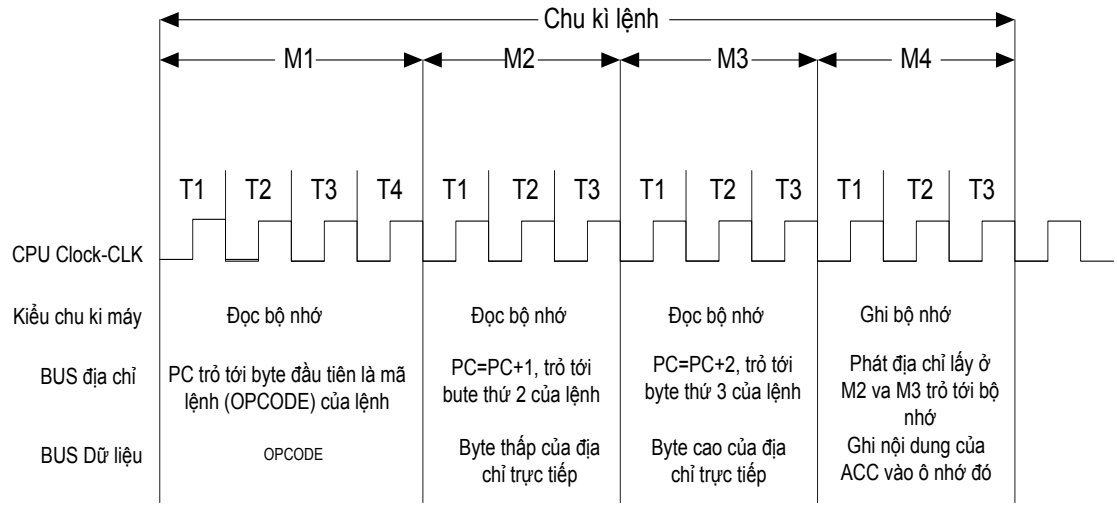
- Trạng thái máy: T (machine State): được định nghĩa là thời gian của một chu kì xung đồng hồ hệ thống (CPU Clock-out). Ví du nếu Clock-out=10 Mhz, thì T=200ns. Các sườn xung lên/xuống được sử dụng bên trong CPU cho các thao tác khác nhau.
- Chu kì máy (hay chu kì BUS): M (machine cycle): Là tập hợp của một số các T để CPU hay một vi mạch (như DMAC 8237) khi nắm quyền kiểm soát BUS hệ thống, thực hiện xong một thao tác (một phần của quá trình gởi ra hay đọc vào một dữ liệu) trên BUS hệ thống.
- Chu kì lệnh: (Instrution cycle): là tập các M cần thiết để hoàn thành một lệnh máy.

Hình sau đây mô tả thực thi của lệnh STA cua CPU Intel 8085: Cất nội dung trong thanh ghi ACC của CPU và ô nhớ trỏ trực tiếp bởi 2 byte tiếp theo của lệnh: ([byte 3], [byte 2]) <- ACC.

Ví dụ: cú pháp hợp ngữ như sau: STA 0610h, giả định trong ACC có một giá trị nào đó, ta sẽ cất (STore) giá trị đó vào ô nhớ 1006:

Trong đó STA=00110010: OPCODE
Địa chỉ thấp của ô nhớ RAM: 00000110
Địa chỉ cao của ô nhớ RAM: 00010000

Biểu đồ thời gian thực hiện như sau:



Mi: Chu kì máy , Ti: trạng thái máy

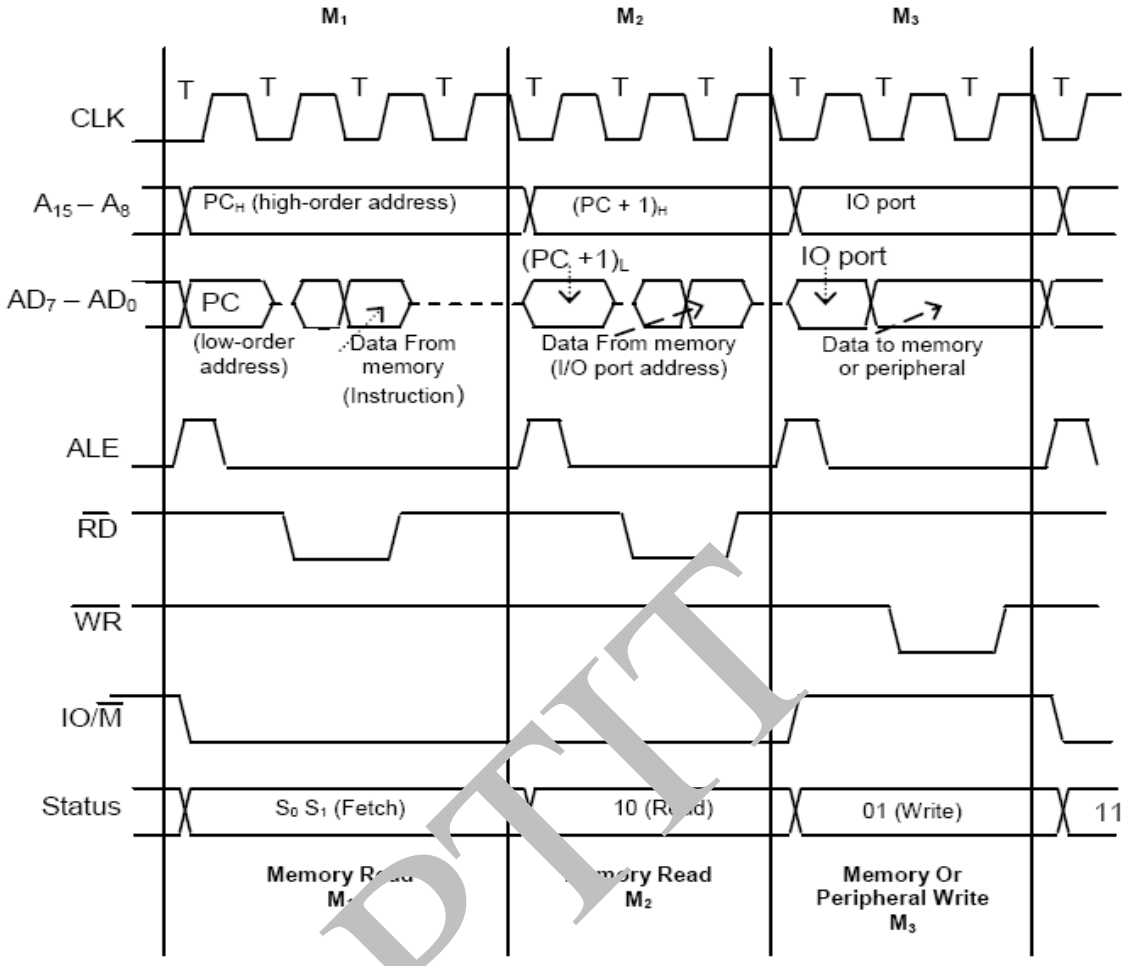
Ví dụ thực hiện lệnh có độ dài 3 byte, cất nội dung của ACC vào ô nhớ trỏ bởi địa chỉ trong lệnh: STA [địa-chỉ-thấp địa-chỉ-cao]

Hình 2.3 Các khái niệm đối chiếu theo CPU Clock

Ta có thể theo dõi các xung điện này khi sử dụng máy hiện sóng (OSCILOSCOPE) với ít nhất 2 tia cặp vào vị trí thích hợp trên bo mạch, lấy xung đồng hồ CPU CLK làm chuẩn để đồng bộ các tín hiệu.

Việc thực hiện một lệnh trong CPU P8085 thực tế là một chuỗi các thao tác READ và WRITE. Mỗi thao tác READ hay WRITE tương ứng với một chu kỳ máy M. Mỗi lệnh được thực hiện qua 1 đến 5 chu kỳ máy. Mỗi chu kỳ máy cần từ 3 đến 5 nhịp đồng hồ. Ví dụ lệnh STA nói trên có 5 chu kì máy, 13 trạng thái máy.

Sau đây là mô tả hoạt động của CPU 8085 với các chu kì đồng hồ hệ thống và các thao tác khác. Ở chu kỳ máy thứ nhất, CPU thực hiện nạp mã lệnh (*Instruction Code Fetch*) trong RAM, còn gọi là chu kỳ *Opcode Fetch*. Hình dưới cho thấy rằng việc thực hiện chu kỳ máy M1 (Opcode Fetch), CPU gửi ra các tín hiệu IO/M, S1 và S0 (tương ứng 0, 1, 1 trên biểu đồ thời gian) xác định thao tác của chu kỳ.

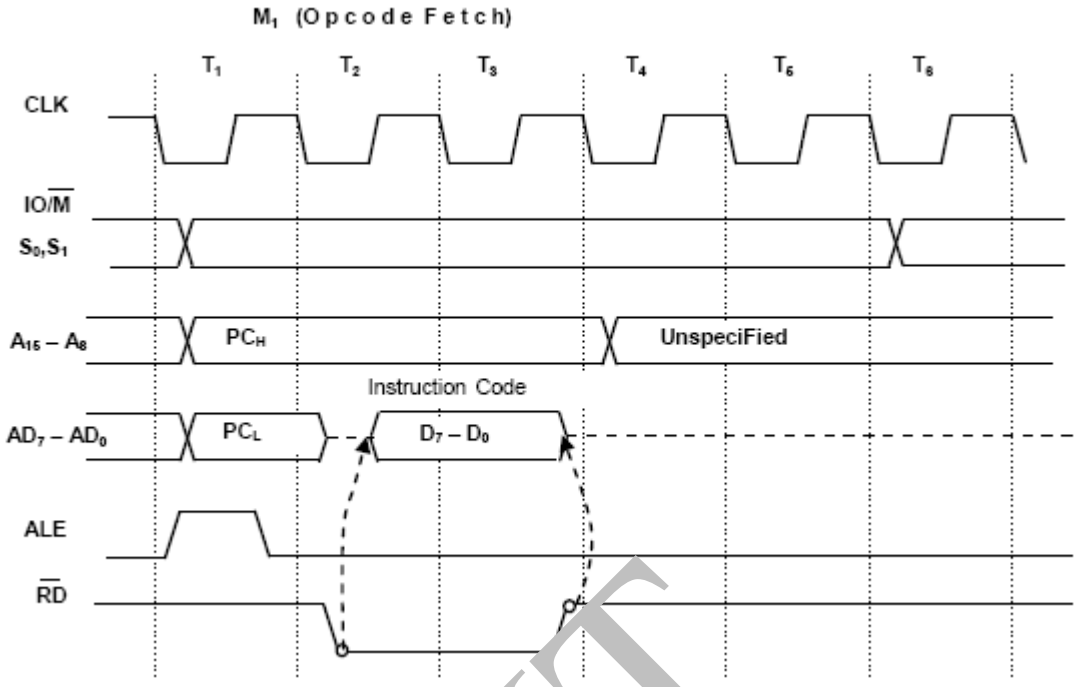


Hình 2.4 Lưu đồ thời gian cơ sở của CPU 8085 (Theo tài liệu của hãng Intel)

CPU cũng đồng thời gửi 16 bit địa chỉ ra ở chu kỳ máy đầu tiên, ngay từ nhịp đầu tiên (T1) để xác định ô nhớ hay thiết bị I/O. Phần địa chỉ byte thấp đặt trên AD7-AD0 (Program Counter Low byte-PCL) chỉ tồn tại trong thời gian 1 nhịp nên cần phải được chốt lại nhờ tín hiệu ALE ở mức cao. Còn phần địa chỉ byte cao đặt trên A16-A8.

Khi D7 – D0 đã ổn định trên các dây dữ liệu, CPU gửi tín hiệu RD. Khi đã nhận được dữ liệu, RD chuyển lên mức cao để cấm vị trí ô nhớ hay thiết bị I/O.

Số lượng chu kỳ máy và trạng thái cần cho thực hiện một lệnh là cố định, song số lượng này khác nhau đối với các lệnh khác nhau, tuỳ theo độ dài của từ lệnh (1 byte, 2 bytes, 3 bytes). Số lượng chu kỳ máy phụ thuộc vào số lần CPU phải liên lạc với các phần tử khác trong hệ thống, chủ yếu là với các chip khác.



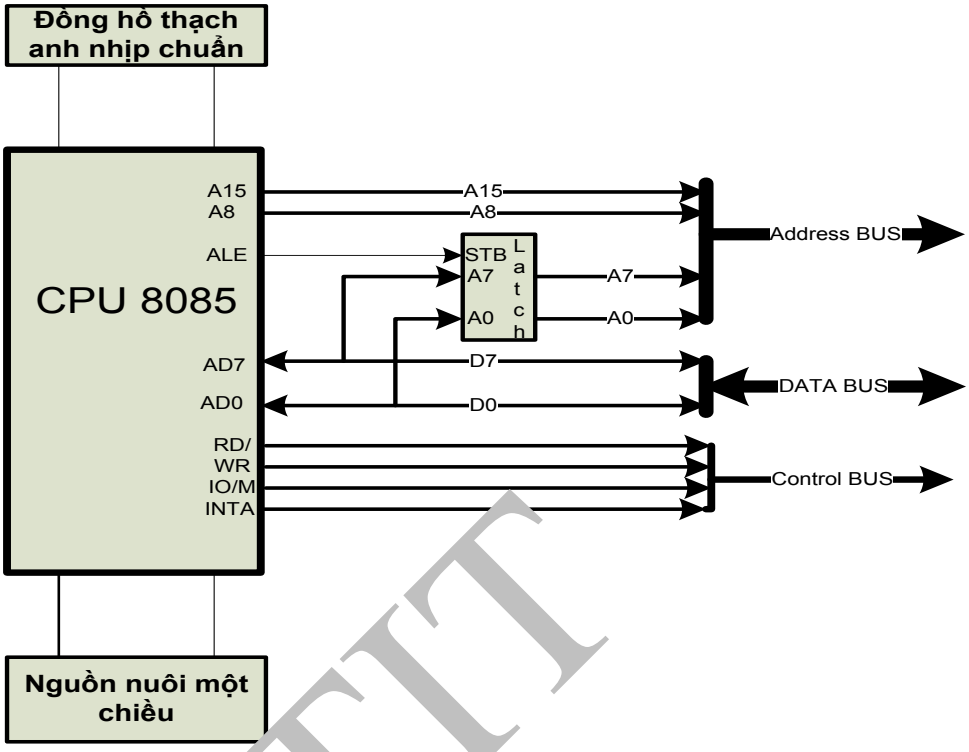
Hình 2.5 Biểu đồ thời gian của chu kì tìm lệnh.

2.2 CPU 8085 VÀ HỆ THỐNG BUS

Việc đầu tiên trong thiết kế một HTN là lựa chọn CPU và hình thành BUS hệ thống trước khi mở rộng với các thành phần khác như ROM, RAM và các cổng ghép nối. Tiếp tục với tư duy sử dụng họ Intel 808X, ta sẽ thiết kế hệ thống BUS như sau:

Đặc điểm của các chân/tín hiệu phát ra từ CPU: Như trên đã liệt kê, CPU đưa ra 3 tập hợp tín hiệu chính, bao gồm:

- Tập tín hiệu địa chỉ trên các chân AD7 – AD0 và A8 – A15, trong đó AD7 – AD0 là các chân 2 chức năng : Lúc ở (M1, T1) là địa chỉ phần thấp, sau đó là dữ liệu. Do đó để tạo ra 16 đường địa chỉ ta cần chốt các giá trị này lại với hỗ trợ của tín hiệu ALE (*Address Latch Enable*) với vi mạch SN74373. Tập hợp này tạo thành **BUS địa chỉ**.
- Tập các chân dữ liệu AD7 – AD0 ở các chu ki M/T tiếp theo của chu kì lệnh. Tập hợp này tạo thành BUS dữ liệu. BUS dữ liệu có độ rộng khác nhau theo loại CPU, có thể là 4, 8, 16, 32, 64, hay trên một loại BUS đặc biệt chung ghép nối vào vi mạch đồng xử lí (ChipSet), gọi là BUS phía trước (Front Side Bus-FSB) ở các mát PC hiện đại, độ rông co thể lên 128, 256 hat 1024 bit.
- Tập các tín hiệu điều khiển phát ra từ CPU, là **BUS điều khiển**.
- Do cần ghép nối với các thiết bị bên ngoài, cần một BUS khác gọi là **BUS hệ thống mở rộng**, ví du: ISA, EISA, … và phổ biến là PCI. Các BUS này cần được định nghĩa tường minh và có giao thức hoạt động rất phức tạp.



Hình 2.[illegible] Cấu hình tối thiểu: CPU 8085 và tạo BUS hệ thống

Do chức năng của BUS là để truyền thông tin giữa các thành phần hợp thành của máy tính và do đó có vai trò rất quan trọng ảnh hưởng tới hiệu năng của máy tính, nên đôi khi phải xem xét tới một khái niệm về thông lượng của BUS (như ở bất kì loại thiết bị truyền thông tin nào). Đó là giá trị về lượng dữ liệu tối đa được chuyển qua BUS trong một khoản thời gian nào đó, thông thường qui theo CPU clock (hay chu kì lệnh):

$$\textit{Tốc độ truyền tối đa} = \frac{\textit{Tốc độ BUS (MHz) x số byte một lần truyền}}{\textit{Số chu kì Clock cho một lần truyền}}$$

2.2.1 Khái niệm và bản chất vật lý của các BUS

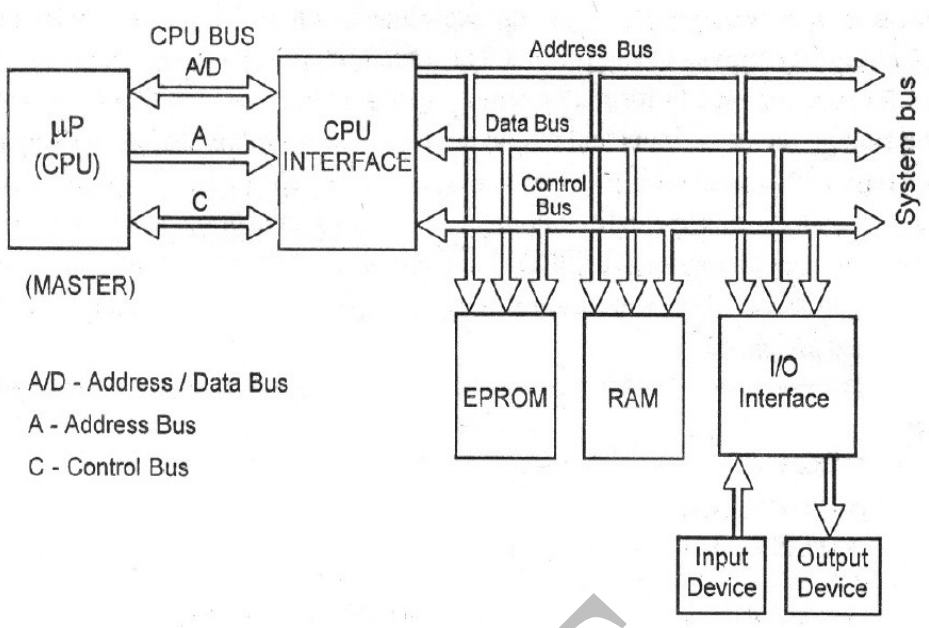
Hoạt động của một hệ kĩ thuật số thực chất là việc trao đổi và xử lý các giá trị nhị phân giữa các thành phần, các khối và các mạch vi điện tử trong toàn bộ hệ thống. Như đã biết, các giá trị nhị phân (hoặc 0 hoặc 1) được thể hiện qua mức điện áp so với một chuẩn nhất định, ví dụ chuẩn TTL (*transistor-transistor logic*) giá trị 0 tương ứng với mức điện áp thấp (từ 0V đến +0,8V) và giá trị 1 tương ứng với mức điện áp từ khoảng +3V đến +5V. Để biểu diễn một số liệu nhị phân, các phần tử mang thông tin được liên kết kề nhau theo nhóm (ví dụ 1byte là 8 bits). Để đảm nhận công việc di chuyển các dữ liệu này trong toàn bộ hệ thống, có các đường dây truyền dẫn điện

được ghép song song thành hệ thống, mỗi dây truyền dẫn dành riêng cho 1 bit. Tập các đường truyền dẫn dành riêng cho các tín hiệu có cùng chức năng (dữ liệu, địa chỉ, điều khiển và trọng số (2^n)) được gọi là BUS. BUS giải quyết một vấn đề kĩ thuật cơ sở quan trọng là: nối được hai đầu ra của hai mạch điện tử khác nhau mà không làm cho đầu ra nào bị hỏng. Yếu tố cơ bản ở đây là các đầu ra phải được điều khiển bằng chương trình máy tính để đưa đầu ra của vi mạch vào một trong hai chế độ làm việc đặc biệt (hay gọi là trạng thái) sau: hoạt động bình thường, tải năng lượng lên dây kết nối và ở trạng thái không hoạt động và có trở kháng đầu ra đủ cao (*Tri-state*) để không đoản mạch đầu ra của vi mạch kia. Làm được như vậy tín hiệu sẽ đi được cả hai chiều trên cùng một dây nối. Và hệ quả là giảm đi một nửa các chân nối ra/vào của vi mạch số.

Như vậy, trong một hệ máy tính BUS có một số tiêu chí sau đây:

- BUS phải tuân theo 1 chuẩn nào đó. Tập các quy tắc của chuẩn còn được gọi là giao thức bus (*bus protocol*)
- Có thể có nhiều loại bus khác nhau được sử dụng, các bus này nói chung là không tương thích với nhau.
- Bus thường phân loại theo 3 cách sau:
  - Theo tổ chức phần cứng
  - Theo giao thức truyền thông (bus đồng bộ và không đồng bộ)
  - Theo loại tín hiệu truyền trên bus :
    - ✓ BUS dữ liệu, là BUS hai chiều, có trạng thái trở kháng cao.
    - ✓ BUS địa chỉ: là BUS một chiều vì thông thường địa chỉ phát sinh ra từ CPU. Tuy nhiên phải có khả năng ở trạng thái trở kháng cao khi cần thiết.
    - ✓ BUS điều khiển, các tín hiệu điều khiển từ CPU hay vi mạch chức năng khác..
    - ✓ Các BUS này hợp lại thành BUS của CPU.
    - ✓ BUS của CPU thường có tải đầu ra yếu, nên BUS này được khuyếch đại, còn có CPU Clock_out (hay **BUS Clock**) tạo thành **BUS hệ thống**. Sự khác nhau là ở chỗ:

BUS của CPU:	**BUS hệ thống:**
▪ Đi ra trực tiếp từ CPU ▪ BUS dồn kênh	▪ Không nối trực tiếp vào CPU, mà qua khuyếch đại BUS ▪ Không còn dồn kênh, các BUS tách biệt ▪ Phụ tải lớn hơn ▪ Có BUS Clock trên BUS



Hình 2.7 CPU Bus và BUS hệ thống

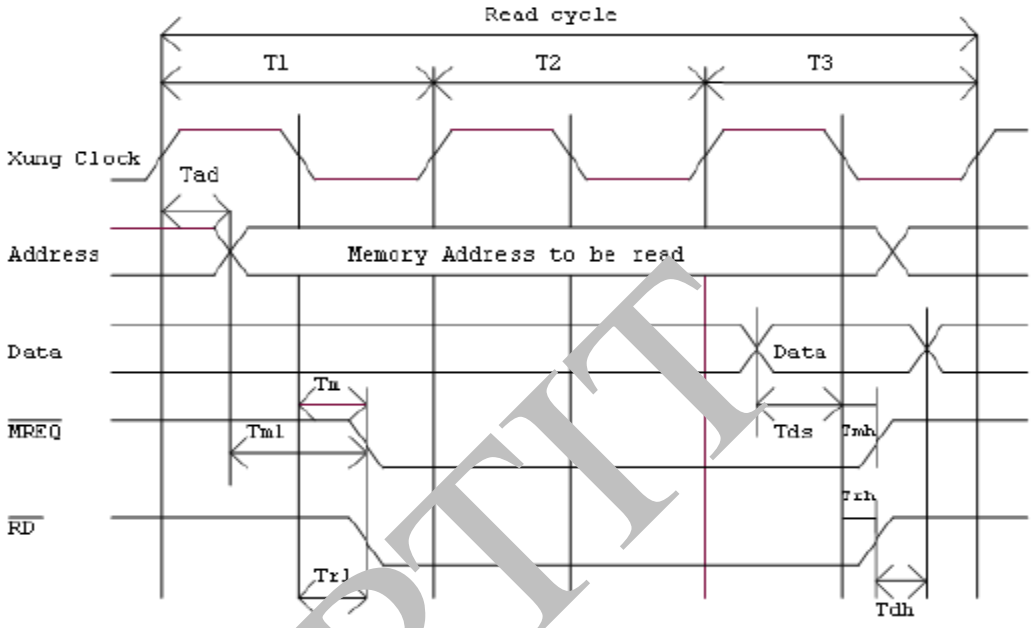
Từ khái niệm trên, dễ dàng suy ra bản chất vật lý của các BUS trong một hệ máy tính: đó là các *đường truyền dẫn điện, có thể dưới các dạng cáp nhiều sợi, đường dẫn trong các bảng mạch in v.v*... Khả năng và chất lượng dẫn điện của các đường truyền dẫn này đóng vai trò quan trọng và quyết định đối với hoạt động của một hệ máy tính. Đường truyền dẫn kém, điện trở thuần cao có thể gây ra sự suy giảm của tín hiệu, điện dẫn gây ra các hiện tượng mất hoặc sai dữ liệu. BUS là đường dẫn điện nội bộ mà theo đó các tín hiệu được truyền từ bộ phận này đến các bộ phận khác trong hệ máy tính.

2.2.2 Khuyếch đại BUS (bus driver)

Tín hiệu phát sinh từ CPU thường có công suất thấp chỉ đủ cho một số tải danh định (*fan-out*), không đủ để mở rộng BUS, nhất là khi bus khá dài và có nhiều thiết bị nối với nó. Chính vì thế mà hầu hết các BUS được nối một số vi mạch khuyếch bus (*bus driver)*, về cơ bản đó là các vi mạch khuyếch đại tín hiệu số. Hầu hết vi mạch khác nối với BUS qua vi mạch đầu vào (*bus receiver)*. Đối với các thiết bị khi thì đóng vai trò đưa tín hiệu lên BUS (*master*), khi thì đóng vai trò nhận tín hiệu từ BUS (*slave*), người ta sử dụng một vi mạch kết hợp có khả năng phát ra và nhận về, gọi là vi mạch phát và thu tín hiệu (*bus transceiver)*. Các chip này đóng vai trò ghép nối và là thiết bị 3 trạng thái, cho phép nó có thể ở trạng thái thứ 3 trở kháng cao. Các vấn đề quan trọng nhất liên quan đến thiết kế bus là: xung đồng hồ bus (clock bus: sự phân chia theo thời gian, hay còn gọi là *bus blocking*), cơ chế trọng tài bus (*bus arbitration*), xử lý ngắt và xử lý lỗi. Các bus có thể được chia theo giao thức truyền thông thành hai loại riêng biệt là bus đồng bộ và bus không đồng bộ phụ thuộc vào việc sử dụng clock bus.

2.2.3 Bus đồng bộ (Synchronous bus):

Bus đồng bộ có một tín hiệu trên đường dây *BUS clock* dạng sóng vuông, với tần số ví dụ, trong khoảng vài MHz ÷ GHz. Mọi hoạt động bus xảy ra đều qui chiếu vào BUS Clock, trong một số nguyên lần chu kỳ này và được gọi là chu kỳ bus. Hình sau giản đồ thời gian của một bus đồng bộ với tần số xung BUS clock là 4MHz, như vậy chu kỳ bus là 250ns. Giả sử đọc 1 byte từ bộ nhớ chiếm 3 chu kỳ bus (750 ns), tương ứng với T1, T2, T3 như hình vẽ. Vì tất cả các bóng bán dẫn khi chuyển mức không phải là tức thời, mà có quá độ và mất một khoản thời gian, nên trên hình vẽ có các sườn xung, ta giả sử các sườn xung kéo dài 10ns. Con số này rất quan trọng khi thiết kế mạch kĩ thuật số !



Hình2.8 Chu kì đọc đồng bộ

• *T1* bắt đầu bằng cạnh dương của xung clock, trong một phần thời gian của T1, vi xử lý đặt địa chỉ byte cần đọc lên bus địa chỉ. Sau khi tín hiệu địa chỉ được xác lập, vi xử lý đặt các tín hiệu MREQ và RD tích cực mức thấp, tín hiệu MREQ (Memory Request) - xác định truy xuất bộ nhớ chứ không phải thiết bị I/O, còn tín hiệu RD - chọn đọc chứ không phải ghi dữ liệu.

• *T2*: thời gian cần thiết để bộ nhớ giải mã địa chỉ và đưa dữ liệu lên bus dữ liệu.

• *T3*: tại cạnh âm của T3, vi xử lý nhận dữ liệu trên bus dữ liệu, chứa vào thanh ghi bên trong vi xử lý và chốt dữ liệu. Sau đó vi xử lý đảo các tín hiệu MREQ v à RD. Như vậy thao tác đọc đã hoàn thành, tại chu kỳ máy tiếp theo vi xử lý có thể thực hiện thao tác khác. Các giá trị cụ thể về thời gian của hình vẽ trên có thể được giải thích chi tiết như sau:

• *TAD*: TAD <110ns, nghĩa là nhà sản xuất vi xử lý đảm bảo rằng trong mọi chu kỳ đọc toán hạng từ bộ nhớ, vi xử lý sẽ đưa ra tín hiệu địa chỉ không nhiều hơn 110 ns tính từ thời điểm cạnh dương của T1.

• *TDS*: giá trị nhỏ nhất là 50ns, có nghĩa là nhà sản xuất bộ nhớ phải đảm bảo rằng dữ liệu đã ổn định trên bus dữ liệu ít nhất là 50ns trước điểm giữa cạnh âm của T3. Yêu cầu này đảm bảo cho vi xử lý đọc dữ liệu tin cậy. Khoảng thời gian bắt buộc đối với TAD và TDS xác định rằng trong trường hợp xấu nhất, bộ nhớ chỉ có 250 + 250 + 125 – 110 – 50 = 465 ns tính từ thời điểm có tín hiệu địa chỉ cho tới khi tạo ra dữ liệu trên bus dữ liệu. Nếu bộ nhớ không có khả năng đáp ứng đủ nhanh, nó phát tín hiệu WAIT trước cạnh âm

của T2. Thao tác này đưa thêm các trạng thái chờ – wait state (tức là đưa thêm vào 1 chu kỳ bus), khi bộ nhớ đã đưa ra tín hiệu ổn định, nó sẽ đảo WAIT thành WAIT

• *TML*: đảm bảo tín hiệu địa chỉ sẽ được xác lập trước tín hiệu MREQ ít nhất 60ns. Khoảng thời gian này sẽ quan trọng nếu tín hiệu MREQ điều khiển quá trình tạo tín hiệu chon chip CS hay CE do một số chip nhớ đòi hỏi phải nhận được tín hiệu địa chỉ trước tín hiệu chọn chip. Như vậy, không thể chọn chip nhớ với thời gian thiết lập 75ns.

• *TM, TRL* cho phép hai tín hiệu MREQ và RD tích cực trong khoảng thời gian 85ns tính từ thời điểm xuống của xung clock T1. Trong trường hợp xấu nhất, chíp nhớ chỉ có 250 + 250 – 85 – 50 = 365ns sau khi 2 tín hiệu trên tích cực để đưa dữ liệu ra bus dữ liệu. Sự bắt buộc về thời gian này bổ sung thêm sự bắt buộc thời gian với tín hiệu clock.

• *TMH, TRH* : thời gian để các tín hiệu MREQ và RD được đảo sau khi dữ liệu đã được vi xử lý nhận vào.

• *TDH:* Thời gian bộ nhớ cần giữ dữ liệu trên bus sau khi tín hiệu RD đã đảo.

Giản đồ thời gian một chu kỳ đọc trên bus đồng bộ đã được đơn giản hoá so với thực tế, trong đó các tín hiệu cần sử dụng lớn hơn nhiều. Giá trị tới hạn của các thông số cho trong bảng sau:

Ký hiệu	***Tham số***	***Min (ns)***	***Max (ns)***
TAD	Thời gian trễ của địa chỉ		100
TML	Thời gian địa chỉ ổn định trước MREQ	60	
TM	Thời gian trễ của MREQ so với cạnh âm của T1		85
TRL	Thời gian trễ của RD bù so sườn xuống của tín hiệu đồng bộ T1		85
TDS	Thời gian thiết lập dữ liệu trước sườn xuống của tín hiệu xung clock(tín hiệu đồng hồ)	50	
TMH	Thời gian trễ của MREQ bù so với sườn xuống của tín hiệu đồng hồ T3		85
TRH	Thời gian trễ của RD bù so với sườn xuống của tín hiệu đồng hồ T3		85
TDH	Thời gian lưu trữ dữ liệu từ lúc đảo tín hiệu RD	0	

Truyền theo khối:

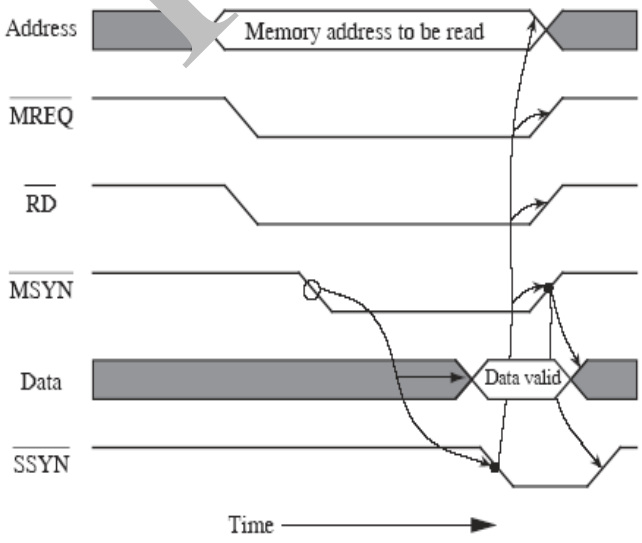
Ngoài các chu kỳ đọc/ghi, một số bus truyền dữ liệu đồng bộ còn hỗ trợ truyền dữ liệu theo khối. Khi bắt đầu thao tác đọc khối, vi mạch làm chủ bus (bus master) báo cho vi mạch khác (slave) biết số byte cần được truyền đi, thí dụ truyền con số này đi trong chu kỳ T1, sau đó đáng lẽ truyền đi 1 byte, slave đưa ra

trong mỗi chu kỳ 1 byte cho tới khi đủ số byte được thông báo. Như vậy, khi đọc dữ liệu theo khối, n byte dữ liệu cần n+2 chu kỳ clock chứ không phải 3n chu kỳ.

Một cách khác để cho truyền dữ liệu nhanh hơn là giảm chu kỳ. Ở ví dụ trên: 1 byte được truyền đi trong 750ns, vậy bus có tốc độ truyền 1.33MBps. Nếu xung clock có tần số 8MHz, thời gian 1 chu kỳ chỉ còn một nửa, tốc độ sẽ là 2.67MBps. Tuy nhiên, giảm chu kỳ bus dẫn đến khó khăn về mặt kỹ thuật, các tín hiệu truyền trên các đường khác nhau không phải luôn có cùng tốc độ, dẫn đến hiệu ứng méo dạng tín hiệu. Điều quan trọng là thời gian chu kỳ phải dài hơn so với sự tồn tại của méo dạng tín hiệu để tránh việc những khoảng thời gian được số hoá lại trở thành các đại lượng biến thiên liên tục.

2.2.4 Bus không đồng bộ (Asynchronous bus)

Bus không đồng bộ không sử dụng xung BUS clock, chu kỳ của nó có thể kéo dài tuỳ ý và có thể khác nhau đối với các cặp thiết bị khác nhau, gọi là đối thoại tuần tự bởi các tín hiệu điều khiển. Làm việc với các bus đồng bộ dễ dàng hơn do nó được định thời một cách gián đoạn , tuy vậy chính đặc điểm này cũng dẫn đến nhược điểm. Mọi công việc được tiến hành trong khoảng thời gian là bội số của xung clock, nếu 1 thao tác nào đó của vi xử lý hay bộ nhớ hoàn thành trong 3,1 chu kỳ thì nó cũng sẽ phải kéo dài trong 4 chu kỳ. Khi đã chọn chu kỳ bus và đã xây dựng bộ nhớ, I/O card cho bus này thì khó có thể tận dụng những tiến bộ của công nghệ. Chẳng hạn sau khi đã xây dựng bus với sự định thời như trên, công nghệ mới đưa ra các vi xử lý và bộ nhớ có thời gian chu kỳ là 100ns chứ không còn là 750ns như cũ, thì chúng vẫn chạy với tốc độ thấp như các vi xử lý, bộ nhớ loại cũ, bởi vì giao thức bus đòi hỏi bộ nhớ phải đưa được dữ liệu ra và ổn định trước thời điểm cạnh âm của . Nếu có nhiều thiết bị khác nhau cùng nối với 1 bus, trong đó có thể có một số thiết bị hoạt động nhanh hơn hơn các thiết bị khác thì cần phải đặt bus hoạt động phù hợp với thiết bị có tốc độ thấp nhất. Bus không đồng bộ ra đời nhằm khắc phục những nhược điểm của bus đồng bộ.



Hình 2.9 BUS không đồng bộ, hoạt động đồng bộ bởi "đối thoại" giữa các tín hiệu điều khiển.

Trước hết master phát ra địa chỉ nhớ mà nó muốn truy cập, sau đó phát tín hiệu MREQ bù tích cực để xác định cần truy xuất bộ nhớ. Tín hiệu này cần thiết khi bộ nhớ và các cổng I/O sử dụng chung miền địa chỉ. Sau khi phát địa chỉ, bên master cũng phải phát tín hiệu RD tích cực để bên slave biết rằng master sẽ thực hiện thao tác đọc chứ không phải ghi. Các tính hiệu MREQ bù và RD bù được đưa ra sau tín hiệu địa chỉ một khoảng thời gian phụ thuộc tốc độ hoạt động của master. Sau khi 2 tín hiệu này đã ổn định, master sẽ phát ra tính hiệu MSYN (master synchrization) ở mức tích cực để báo cho slave biết rằng các tín hiệu cần thiết đã sẵn sàng trên bus, slave có thể nhận lấy. Khi slave nhận được tín hiệu này, nó sẽ thực hiện công việc với tốc độ nhanh nhất có thể được, đưa dữ liệu của ô nhớ được yêu cầu lên bus dữ liệu. Khi hoàn thành slave sẽ phát tín hiệu SSYN (slave synchronization) tích cực. Master nhận được tín hiệu SSYN tích cực thì xác định được dữ liệu của slave đã sẵn sàng nên thực hiện việc chốt dữ liệu, sau đó đảo các đường địa chỉ cũng như các tín hiệu MREQ, RD, và SSYN. Khi slave nhận được tín hiệu MSYN không tích cực, nó xác định kết thúc chu kỳ và đảo tín hiệu SSYN làm bus trở lại trạng thái ban đầu, mọi tín hiệu đều không tích cực, chờ bus master mới. Trên giản đồ thời gian của bus bất đồng bộ, ta sử dụng mũi tên để thể hiện nguyên nhân và kết quả MSYN tích cực dẫn đến việc truyền dữ liệu ra bus dữ liệu và đồng thời cũng dẫn đến việc slave phát ra tín hiệu SSYN tích cực, đến lượt mình tín hiệu SSYN lại gây ra sự đảo mức của các đường địa chỉ, MREQ bù, RD bù, và SSYN. Cuối cùng sự đảo mức của MSYN lại gây ra sự đảo mức tín hiệu SSYN và kết thúc chu kỳ. Tập các tín hiệu điều khiển phối hợp với nhau như vậy được gọi là kết nối hoàn toàn (*full handshake*), chủ yếu gồm 4 tín hiệu sau:

- MSYN tích cực.
- SSYN bù tích cực để đáp lại tín hiệu MSYN.
- MSYN được đảo để đáp lại tín hiệu SSYN bù (tích cực).
- SSYN bù được đảo để đáp lại tín hiệu MSYN không tích cực.

Ta có thể nhận thấy bắt tay toàn phần là độc lập thời gian, mỗi sự kiện được gây ra bởi 1 sự kiện trước đó chứ không phải bởi xung BUS clock. Nếu 1 cặp master-slave nào đó hoạt động chậm thì cặp master-slave kế tiếp không hề bị ảnh hưởng. Ưu điểm của bus bất đồng bộ rất rõ ràng, nhưng trong thực tế phần lớn các bus đang sử dụng là loại đồng bộ. Nguyên nhân là các hệ thống sử dụng bus đồng bộ với các vi mạch đồng bộ về tiêu chuẩn kĩ thuật, tốc độ nhanh, khối lượng bit lớn, khoản cách BUS ngắn (trên một bo mạch chủ) và dễ thiết kế hơn. CPU chỉ cần chuyển các mức tín hiệu cần thiết sang trạng thái tích cực là bộ nhớ đáp ứng ngay, không cần tín hiệu phản hồi. Chỉ cần các lựa chọn phù hợp thì mọi hoạt động đều trôi chảy, không cần phải bắt tay.

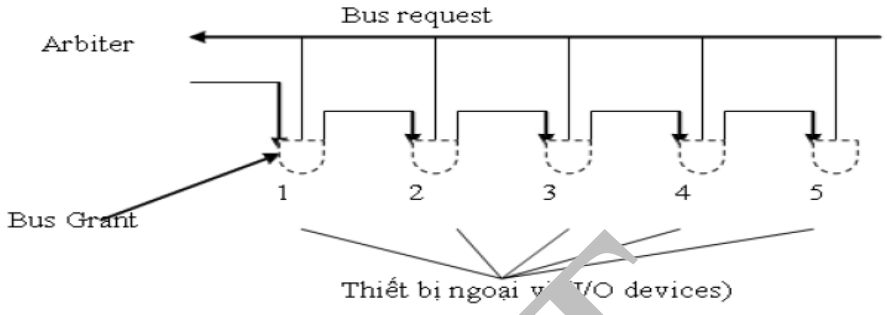
2.2.5 Trọng tài BUS (bus arbitration)

Trong hệ thống máy tính không phải chỉ có CPU làm bus master, các chip I/O cũng có lúc làm bus master để có thể đọc hay ghi bộ nhớ và gọi ngắt. Các bộ đồng xử lý cũng có thể làm bus master. Như vậy nảy sinh ra vấn đề: điều gì sẽ xảy ra khi 2 thiết bị trở lên đồng thời cần làm bus master? Từ đó cần có một cơ chế phân xử để tránh sự hỗn loạn của hệ thống. Cơ chế phân xử có thể là tập trung hay không tập trung.

Trọng tài BUS tập trung

Nhiều vi xử lý có đơn vị phân xử được chế tạo nằm ngay trong chip CPU, trong một số máy tính mini, đơn vị này nằm ngoài chíp CPU. Theo cơ chế này thì bộ trọng tài (*arbiter*) chỉ có thể biết có yêu cầu chiếm dụng bus hay không mà không biết có bao nhiêu đơn vị muốn chiếm dụng bus. Khi *arbiter* nhận

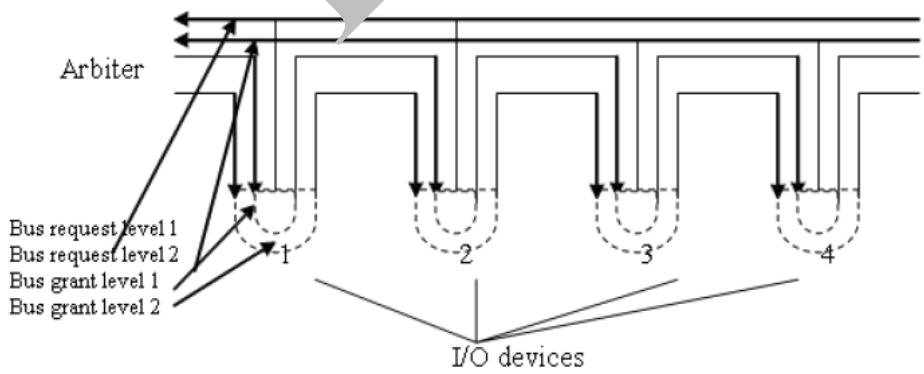
được yêu cầu, nó sẽ phát ra 1 tín hiệu cho phép trên đường dây (bus grant: cho phép sử dụng bus). Đường dây này nối qua tất cả các thiết bị I/O theo kiểu nối tiếp. Khi thiết bị nằm gần *arbiter* nhất nhận được tín hiệu cho phép, nó kiểm tra xem có phải chính nó đã phát ra yêu cầu hay không. Nếu có thì nó sẽ chiếm lấy bus và không truyền tiếp tín hiệu cho phép trên đường dây. Nếu không thì nó sẽ truyền tín hiệu cho phép tới thiết bị kế tiếp trên đường dây, với thiết bị này sự việc xảy ra giống thiết bị trước nó, quá trình cứ tiếp diễn cho đến khi có một thiết bị chiếm lấy bus. Sơ đồ xử lý như vậy có tên gọi là chuỗi quay vòng (*daisy chaining*). Điểm nổi bật của sơ đồ này là các thiết bị được gán thứ tự ưu tiên tuỳ thuộc vào vị trí của nó so với arbiter, thiết bị gần hơn thì mức ưu tiên cao hơn.



Hình 2.10 BUS chuỗi quay vòng (daisy chaining)

Trọng tài BUS

Một số loại bus có nhiều mức độ ưu tiên, với mỗi mức độ ưu tiên có đường yêu cầu bus (*bus request*) và đường dây cho phép bus (*bus grant*). Ví dụ: giả sử 1 bus có 2 mức ưu tiên 1 và 2 (các bus thực tế có 4, 8 hay 16 mức). Mỗi thiết bị trong hệ thống máy tính nối với 1 trong các mức yêu cầu bus, các đường thường được sử dụng nhiều hơn được gắn với đường dây có mức ưu tiên cao hơn. Ở ví du, các thiết bị 1, 2 sử dụng mức ưu tiên 1, còn các thiết bị 3, 4 sử dụng mức ưu tiên 2.



Hình 2.11 Trọng tài BUS

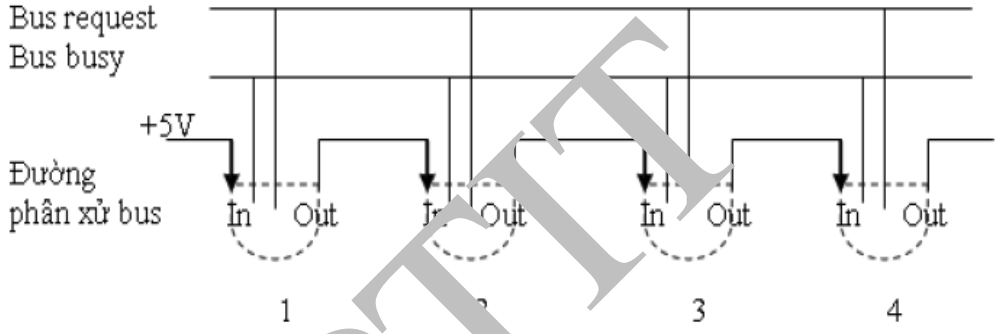
Nếu có một số thiết bị ở các mức ưu tiên khác nhau cùng yêu cầu, arbiter chỉ phát ra tín hiệu *grant* đối với yêu cầu có mức ưu tiên cao nhất. Trong số các thiết bị có cùng mức ưu tiên, thiết bị nào gần *arbiter* hơn sẽ ưu tiên hơn. Về mặt kỹ thuật, không cần nối đường grant level 2 giữa các thiết bị vì chúng không bao giờ đòi hỏi bus ở mức 2. Tuy nhiên, trong thực tế để thuận tiện cho việc lắp đặt người ta hay làm như sau:

nối tất cả các đường grant thông qua tất cả các thiết bị, như vậy sẽ dễ dàng hơn là nối các đường grant một cách riêng biệt, và từ đó căn cứ vào thiết bị nào có quyền ưu tiên cao hơn.

Một trọng tài BUS có đường dây thứ 3 nối tới các thiết bị để các thiết bị xác nhận đã nhận được tín hiệu grant và chiếm dụng bus – đường tín hiệu xác nhận ACK (*acknowledgement*). Ngay sau 1 thiết bị phát tín hiệu tích cực trên đường dây ACK, có thể đảo tín hiệu trên các đường dây request và grant xuống mức không tích cực. Các thiết bị khác có thể yêu cầu bus khi thiết bị đầu tiên đang dùng bus. Khi sự truyền thông kết thúc, bus master kế tiếp sẽ được lựa chọn. Cách làm việc như vậy làm tăng hiệu quả sử dụng bus, nhưng cần thêm 1 đường truyền tín hiệu và cấu trúc thiết bị cũng phức tạp hơn. Các chip trong máy tính PDP-11 và các chip Motorola làm việc với các bus như vậy.

Trọng tài BUS không tập trung

Trong Multibus, người ta cho phép có thể lựa chọn sử dụng phân xử bus tập trung hay không tập trung, cơ chế phân xử bus không tập trung được thực hiện như sau:



Hình 2.12 Trọng tài Bus không tập trung trong multibus

Cơ chế sử dụng 3 đường dây, không phụ thuộc vào số lượng thiết bị nối với bus:

- Bus request: yêu cầu chiếm dụng bus.
- Bus busy: đường báo bận, được bus master đặt ở mức tích cực khi có thiết bị đang chiếm dụng bus
- Bus arbitration: được mắc nối tiếp thành 1 chuỗi xích qua tất cả các thiết bị ngoại vi. Đầu của chuỗi này được gắn với mức điện áp 5V của nguồn nuôi.

Khi không có đơn vị nào yêu cầu chiếm dụng bus, đường dây phân xử bus truyền mức tích cực tới tất cả các thiết bị trong chuỗi xích. Khi 1 đơn vị nào đó muốn chiếm dụng bus, đầu tiên nó kiểm tra xem bus có rảnh không và đầu vào In của đường trọng tài bus có mức tích cực hay không. Nếu không (not active) thì nó không trở thành bus master. Ngược lại, nó sẽ đảo đầu Out thành không tích cực, làm cho các thiết bị đứng sau nó trong chuỗi xích có đầu In không tích cực. Khi trạng thái có thể hiểu lầm (khoảng thời gian tín hiệu trên đầu In và Out đang thay đổi) qua đi, chỉ còn lại duy nhất 1 thiết bị có đầu In tích cực và Out không tích cực. Thiết bị này trở thành bus master, nó sẽ đặt bus busy tích cực và bắt đầu truyền thông tin trên bus.

2.2.6 Bus mở rộng (Expansion bus) EISA, MCA, Bus cục bộ, PCI

Bus mở rộng cho phép bo mạch chủ liên lạc được với các thiết bị ngoại vi, các thiết bị này được cài đặt qua các khe cắm mở rộng (*expansion slot*). Các thông số chính của bus mở rộng: tốc độ truyền tối đa giữa các thiết bị với nhau và giữa các thiết bị với bộ nhớ chính, số đường địa chỉ (số lượng ô nhớ có thể được truy xuất bởi 1 thiết bị), số đường ngắt cứng, v.v.

- ***Bus theo kiến trúc chuẩn công nghiệp (ISA - Industry Standard Architecture)***

Bus ISA dùng cho hệ thống chỉ được điều khiển bởi 1 CPU trên bảng mạch chính, tức là tất cả các chương trình và thiết bị đều chỉ được điều khiển bởi CPU đó. Tần số làm việc cực đại là 8.33 MHz (tốc độ chuyển tải cực đại là 16.66 MBps với số liệu 2 bytes). Bề rộng dữ liệu là 8 hay 16 bits. ISA có 24 đường địa chỉ nên quản lý được 16 MB bộ nhớ. Bus ISA tương thích 90% với bus AT.

- ***Bus ISA* mở rộng (EISA)** và kiểu kiến trúc vi kênh (MCA- *Micro Channel Architecture)*

Sử dụng cho các CPU 32 bits (số liệu và đường địa chỉ) từ 80386 trở đi.

- *Bus EISA:*

Đây là chuẩn mở rộng của ISA để bố trí các dữ liệu 32 bits nhưng vẫn giữ được sự tương thích với mạch nối ghép cũ. Bus EISA có 2 nấc, các tín hiệu ISA được gửi qua nấc trên và các tín hiệu phụ trợ EISA qua nấc dưới. Các đặc trưng của EISA như sau:

• Về mặt cơ khí: có nhiều chân cắm hơn nhưng vẫn tương thích với ISA

• Độ rộng dữ liệu: có thể truy xuất 2 đường 8 bits (tương thích với ISA), hay 2 đường 16 bits Do. đó, đơn vị quản lý bus 32 bits có thể chuyển tải 4 byte với bộ nhớ hoặc thiết bị ngoại vi. Điều này góp phần tăng tốc độ truyền tải lên khoảng 33 MBps so với 16.66 MBps của ISA.

• Độ rộng địa chỉ: ngoài 24 đường như ISA còn thêm 8 đường bổ sung nữa, do đó có thể định địa chỉ trong 4 GB bộ nhớ.

• Phần cứng được thiết kế theo hệ thống EISA phức tạp hơn ISA vì nó cũng phải thực hiện các chu kỳ bus tương thích với ISA. EISA có thể thực hiện phân xử bus, nó cho phép vi xử lý nằm ngoài bo mạch chính có thể điều khiển toàn bộ bus. Điều này rất hiệu quả trong các hệ thống đa xử lý (multiprocessor). Hãng Intel đã phát triển 4 chip điện tử phục vụ cho bus EISA như sau:

  - o ISP (Intergrated system peripheral)
  - o BMIC (Bus master interface controller)
  - o EBC (EISA bus controller)
  - o EBB (EISA bus buffer)

- ***Bus MCA:***

Phục vụ cho hệ thống IBM PS/2 không tương thích với bus ISA, có thể hoạt động với 16 hay 32 bits dữ liệu. Nó có nhiều đường dẫn hơn ISA, thiết kế phức tạp cho phép giảm bớt các nhiễu cao tần của PC tới các thiết bị xung quanh. Tốc độ truyền dữ liệu có thể lên tới 160 MBps.

- ***Bus cục bộ (Local Bus):***

Nhược điểm của các bus chuẩn trên là mặc dù xung clock của CPU rất cao nhưng cũng chỉ làm việc với các ngoại vi với tốc độ truyền tải không quá 33MBps. Điều này không thể đáp ứng được tốc độ của các card đồ hoạ cắm vào khe cắm của bus mở rộng trong chế độ đồ họa. Chuẩn các *bus cục bộ* tạo thêm *các*

khe cắm mở rộng nối trực tiếp vào bus cục bộ (bus nối giữa CPU và các bộ đệm). Do vậy, bus mở rộng loại này cho phép truy xuất lên trên 32 bit cũng như tận dụng được tốc độ xung clock của chính CPU, tránh được rào cản 8.33MHz của bus hệ thống. Theo hướng giải quyết này, Intel đã phát triển bus PCI và Uỷ ban VESA (Video Electronics Standards Association) đã phát triển bus VL.

- ***Bus PCI (Peripheral Component Interconnect)***

Là loại BUS với nhiều cải tiến và dùng phổ biến trên PC hiện đại. Bus PCI là bus trong đó dữ liệu và địa chỉ được gởi đi theo cách thức dồn kênh (multiplexing), các đường địa chỉ và dữ liệu được dồn chung trên các đường của PCI. Cách này tiết kiệm được số chân PCI nhưng lại hạn chế tốc độ vì phải cần 2 xung clock cho một quá trình truyền dữ liệu (1 cho địa chỉ và 1 cho dữ liệu). Việc nối giữa CPU, bộ nhớ chính, và bus PCI được thực hiện bằng *cầu PCI* (PCI bridge), qua đó bus PC sẽ phục vụ cho tất cả các đơn vị của bus PCI. Tối đa là 10 thiết bị có thể được nối tới bus PCI, trong đó cầu PCI được coi là một. PCI có thể hoạt động với độ rộng 32 bits dữ liệu và tốc độ 33MHz : *PCI = 33Mhz x 4 bytes (32 bits) = 133MB/s*. Hiện tại PCI đã phát triển PCI-X 2.0, chạy 64 bit với BUS clock=266/533 MHz cho thông băng tới 2,15 GB/s và 4,3 GB/s. Một điểm mạnh của PCI là dữ liệu được truyền tải theo kiểu cụm (burst), trong đó địa chỉ chỉ truyền đi 1 lần, sau đó nó sẽ được hiểu ngầm bằng cách cho các đơn vị phát hoặc thu đếm lên trong mỗi xung clock. Do đó, bus PCI hầu như được lấp đầy bởi dữ liệu.

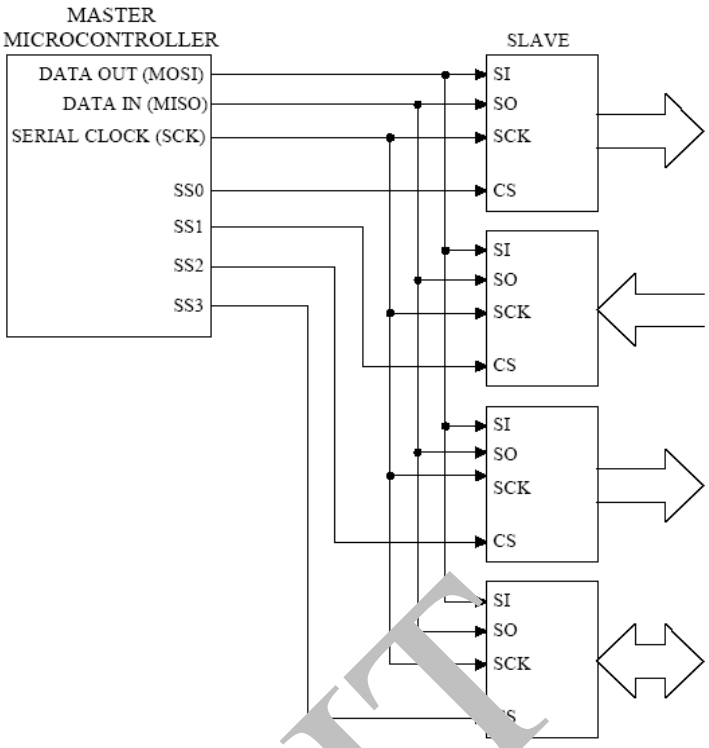
Khi thiết kế cần xem lại chi tiết các tiêu chuẩn/định nghĩa chân của từng loại BUS sẽ sử dụng cho bo mạch.

2.2.7 Bus SPI (Serial Peripheral Interface)

Là loại bus liên kết dữ liệu nối tiếp, đồng bộ, hoạt động theo kiểu chủ/tớ (Master/Slave). Về kết nối bao gồm một thiết bị làm chủ (master) để điều phối và đồng bộ hoạt động, cung cấp xung đồng hồ (clock) chung cho toàn bus và một hoặc vài thiết bị khác hoạt động thụ động (tớ), nhận xung đồng hồ. Ngoài ra còn có các dây dữ liệu. Bus có 4 dây nối. Bus được sử dụng phổ biến trong các hệ có vi điều khiển nối với các vi mạch khác, tốc độ cao, sử dụng ít dây nối liên kết. Giao thức vận hành do phần mềm hay đã cứng hóa trong các chip sử dụng SPI. Một số kết nối điển hình như sau:

Chân vi mạch	Ý nghĩa	Ứng dụng
MOSI	Master out/Slave in	µP → ASIC
MISO	Master in /Slave out	ASIC → µP
SCK	Serial clock	µP → ASIC
SSn/CS	Chip select (low active)	µP → ASIC

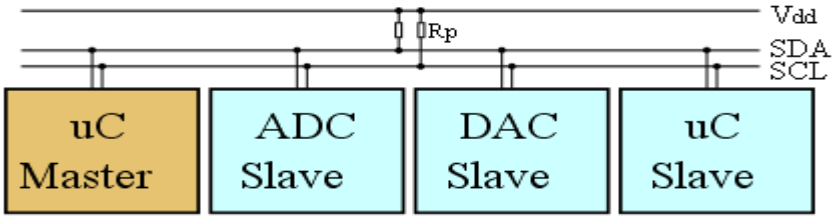
Xem thêm http://www.vti.fi/midcom-serveattachmentguid-b469d17c67e60d4e3dbb25b0d099ad68/TN15_SPI_Interface_Specification.pdf



Hình 2.13 Liên kết qua bus SPI

2.2.8 Bus I²C (Inter-Integrated Circuit)

Là BUS nối tiếp chỉ có 2 dây nối, hoạt động kết hợp cả phần cứng và phần mềm (Master/Slave và protocol), có thể nối vào nhiều thiết bị ngoài vào nhau, cũng như hỗ trợ nhiều BUS có cơ chế điều khiển (master mode). Bus cấu thành từ 2 thành phần là SDA và SCL.



Hình 2.14 Liên kết qua bus I2C

Trong đó: **SCL** (*Serial Clock line*) Là xung đồng hồ phát ra tư chip làm chủ (Master).

SDA (*Serial Data Line*) là đường dữ liệu với 7 bit địa chỉ các vi mạch tham gia (vi mạch chủ và tớ).

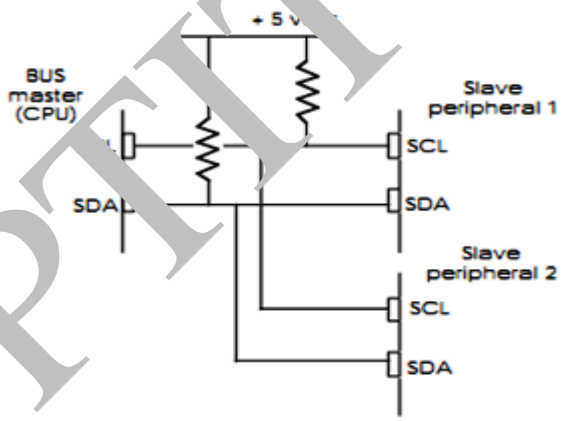
R_p là điện trở nối lên nguồn nuôi VDD.

Các thiết bị loại chủ hay tớ (Master/Slave), các thiết bị nhỏ có khoản cách gần , như bộ nhớ USB, điện thoại di động, có thể nối trực vào BUS. Khi nối xa hơn cần bộ khuyếch đại (P82B715) tăng dòng điện cho các thiết bị. Về cơ bản số thiết bị nối sẽ phụ thuộc vào: chiều dài của BUS, xung đồng hồ BUS, dung kháng phụ tải không quá 400 pF. Với tần số thấp BUS có thể dài tới vài mét.
Có 4 chế độ làm việc, tuy nhiên 2 chế độ sau là phổ biến:

- master phát dữ liệu cho slave
- master nhận dữ liệu từ slave
- slave phát dữ liệu tới master
- slave nhận dữ liệu từ master

Thủ tục qua thông điệp:

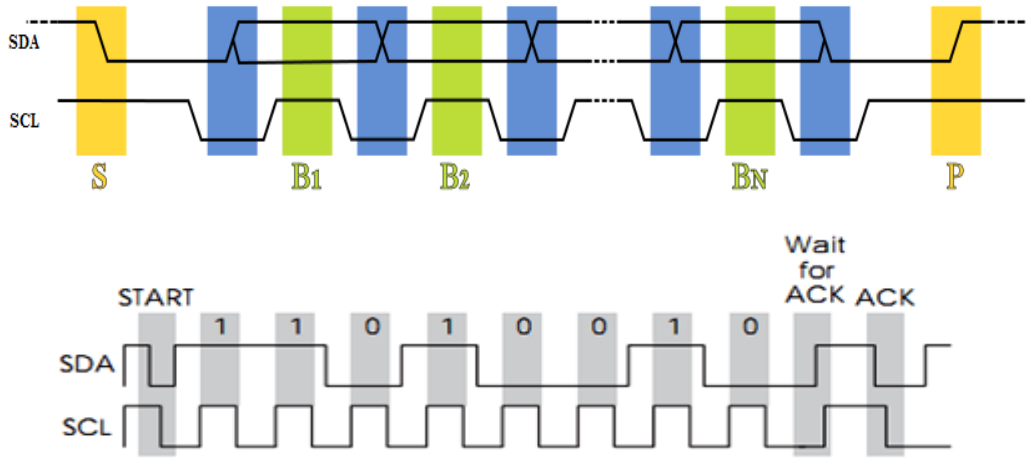
- Thông điệp đơn ở đó master ghi dữ liệu vào slase
- Thông điệp đơn ở đó master đọc dữ liệu từ slave
- Thông điệp phối hợp nơi master sẽ thực hiện ít nhất 2 lần đọc/ghi tới 1 hay vài slave. Khi đó mỗi lần đọc/ghi đều bắt đầu bằng START cùng địa chỉ của slave. Sau tín hiệu START đầu tiên trong thông điệp phối hợp là các bit gọi là START nhắc lại (*repeated START* bits), các bit này không đặt trước các bit STOP sao cho slave nhận biết rằng tiếp theo là một phần của thông điệp. Các slave sẽ phản ứng (trả lời) cho từng thông điệp riêng biệt



Hình 2.15 nguyên lí nối BUS I²C

Bus I²C với các thiết bị nối vào: Các mạch đầu ra/vào hoạt động kiểu cả hai chiều (bidirectional), nên cần các điện trở treo, cho nên khi rút thiết bị khỏi BUS, đảm bảo đầu ra sẽ có mức điện áp cao. Nếu không có điện trở, các đường nối sẽ rơi vào trạng thái không xác định, gây lỗi dữ liệu.

Hoạt động của BUS I²C:



Hình 2.16 ghi/đọc trên BUS I²C

Trình tự và logic thay đổi được định nghĩa bởi 3 thông điệp:

Message	1st event	2nd event
START	**SDA H to L**	**SCL H to L**
STOP	**SCL L to H**	**SDA L to H**
ACK (**ACK**NOWLEDGE)	**SDA H to L**	**SCL H to L**

Tín hiệu **ACK**NOWLEDGE và START tuy tương tự nhau, song có khác là: START do Master thực hiện, trong khi ACK từ Slave cả hai tạo ra qui tắc đối thoại kiểu bawdng phần cứng (*hardware-handshak).*

Việc truyền dữ liệu được bắt đầu với START (S) khi SDA chuyển xuống mức thấp (H to L) trong khi SCL giữ ở mức cao (H). Sau đó SDA đặt bit được truyền đi trong khi SCL ở mức thấp. Bit dữ liệu sẽ được tiếp nhận (lấy mẫu) khi SCL chuyển lên mức cao (L to H). Khi dữ liệu truyền đã hết, bit STOP được gởi lên trên SDA trong khi SCL vẫn ở mức cao liên tục. Để tránh lỗi khi việc phát hiện các bit đánh dấu xảy ra, mức của SDA thay đổi ở sườn xuống (H to L) và sẽ được lấy mẫu khi SCL chuyển từ thấp lên cao (L to H).

Dưới đây là code cho quá trình kiểm soát được mô tả quá trình nói trên của giao thức I²C ở phía thiết bị master (clock, arbitration, start/stop bit, ack/nack):

```
// Hardware-specific support functions that MUST be customized:
#define I2CSPEED 100
void I2C_delay() { volatile int v; int i; for (i=0; i < I2CSPEED/2; i++) v; }
bool read_SCL(void); // Set SCL as input and return current level of line, 0 or 1
bool read_SDA(void); // Set SDA as input and return current level of line, 0 or 1
void clear_SCL(void); // Actively drive SCL signal low
void clear_SDA(void); // Actively drive SDA signal low
```

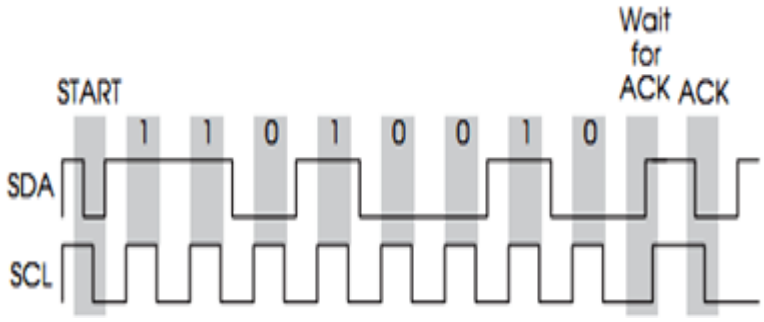
```
void arbitration_lost(void);

bool started = false; // global data
void i2c_start_cond(void) {
  if (started) { // if started, do a restart cond
    // set SDA to 1
    read_SDA();
    I2C_delay();
    while (read_SCL() == 0) {  // Clock stretching
      // You should add timeout to this loop
    }
    // Repeated start setup time, minimum 4.7us
    I2C_delay();
  }
  if (read_SDA() == 0) {
    arbitration_lost();
  }
  // SCL is high, set SDA from 1 to 0.
  clear_SDA();
  I2C_delay();
  clear_SCL();
  started = true;
}

void i2c_stop_cond(void){
  // set SDA to 0
  clear_SDA();
  I2C_delay();
  // Clock stretching
  while (read_SCL() == 0) {
    // add timeout to this loop.
  }
  // Stop bit setup time, minimum 4us
  I2C_delay();
  // SCL is high, set SDA from 0 to 1
  if (read_SDA() == 0) {
    arbitration_lost();
  }
  I2C_delay();
```

```
  started = false;
}

// Write a bit to I2C bus
void i2c_write_bit(bool bit) {
  if (bit) {
    read_SDA();
  } else {
    clear_SDA();
  }
  I2C_delay();
  while (read_SCL() == 0) { // Clock stretching
    // You should add timeout to this loop
  }
  // SCL is high, now data is valid
  // If SDA is high, check that nobody else is driving SDA
  if (bit && read_SDA() == 0) {
    arbitration_lost();
  }
  I2C_delay();
  clear_SCL();
}

// Read a bit from I2C bus
bool i2c_read_bit(void) {
  bool bit;
  // Let the slave drive data
  read_SDA();
  I2C_delay();
  while (read_SCL() == 0) { // Clock stretching
    // You should add timeout to this loop
  }
  // SCL is high, now data is valid
  bit = read_SDA();
  I2C_delay();
  clear_SCL();
  return bit;
}
```

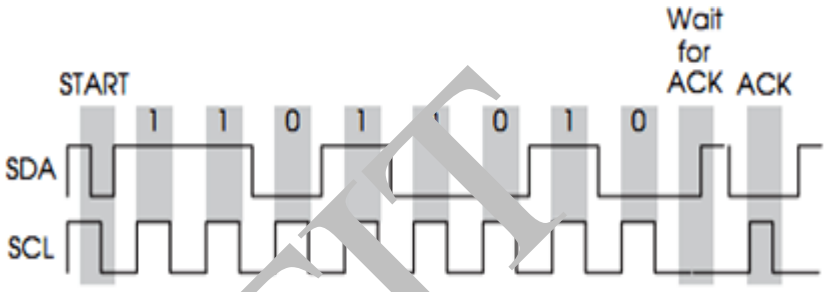
```
// Write a byte to I2C bus. Return 0 if ack by the slave.
bool i2c_write_byte(bool send_start,
                    bool send_stop,
                    unsigned char byte) {
  unsigned bit;
  bool nack;
  if (send_start) {
    i2c_start_cond();
  }
  for (bit = 0; bit < 8; bit++) {
    i2c_write_bit((byte & 0x80) != 0);
    byte <<= 1;
  }
  nack = i2c_read_bit();
  if (send_stop) {
    i2c_stop_cond();
  }
  return nack;
}

// Read a byte from I2C bus
unsigned char i2c_read_byte(bool nack, bool send_stop) {
  unsigned char byte = 0;
  unsigned bit;
  for (bit = 0; bit < 8; bit++) {
    byte = (byte << 1) | i2c_read_bit();
  }
  i2c_write_bit(nack);
  if (send_stop) {
    i2c_stop_cond();
  }
  return byte;
}
```

Ghi dữ liệu:

Byte dữ liệu ghi truyền đi cùng với ACK.



Đọc dữ liệu: Byte dữ liệu đọc truyền cùng ACK.



Hình 2.17 Việc dữ liệu thu/phát trên BUS I^2C

Chi tiết giao thức hoạt động xem thêm tài liệu: http://en.wikipedia.org/wiki/I2C

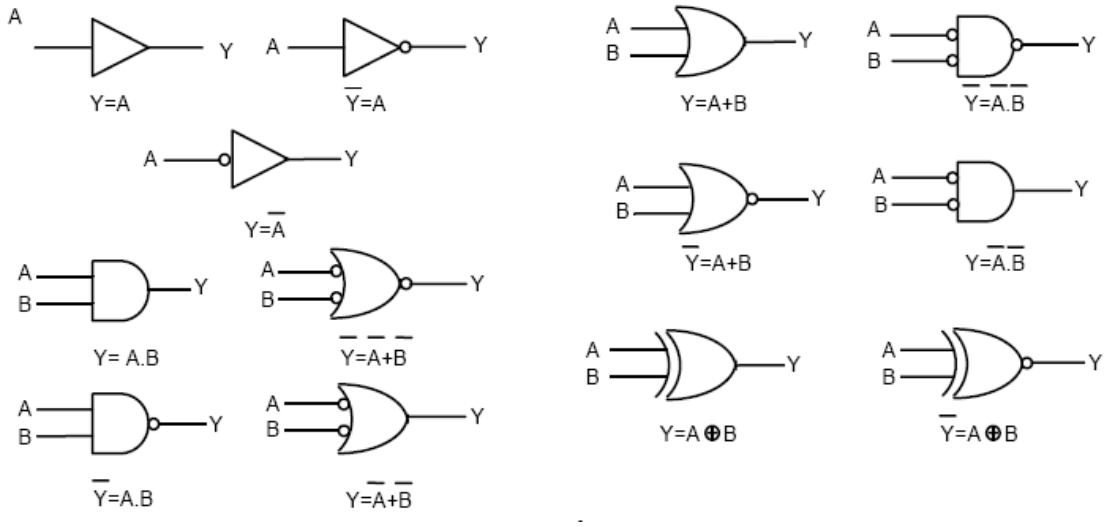
2.2.9 Thực hiện kĩ thuật của BUS

Các mạch 3 trạng thái, mạch chốt và mạch khuyếch đại BUS 2 chiều.

Trước tiên, cũng cần nhắc lại một số linh kiện điện tử số cơ bản sử dụng trong máy vi tính. Nhờ công nghệ cao, các linh kiện có độ tích hợp lớn và rất lớn đã ra đời, nhưng không thể không nhìn lại một số mạch tổ hợp thực hiện những hàm logic cơ bản nhất.

a) Các cổng logic

Ký hiệu các mạch được chỉ ra trên Hình II.2, cùng biểu thức hàm logic gồm: *mạch đệm (buffer), mạch đảo (NOT), mạch và (AND), mạch NAND, mạch hoặc (OR), mạch NOR và mạch XOR.*

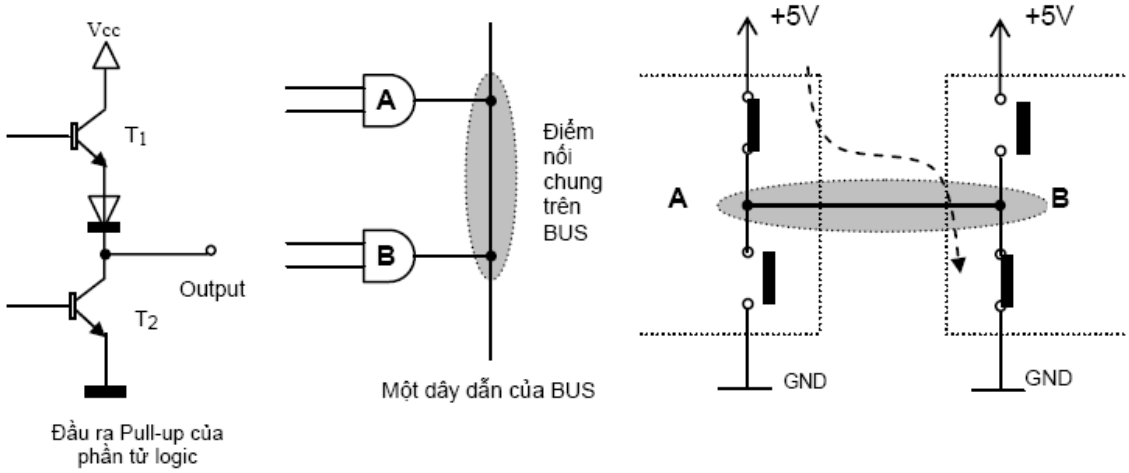


Hình 2.18 Các mạch logic thường dùng trong thiết kế kĩ thuật số

Các loại mạch này thường được sử dụng để tạo nên các mạch tổ hợp logic thực hiện các chức năng lập mã, giải mã, dồn kênh và phân kênh. Cũng cần lưu ý rằng, một số mạch chức năng như giải mãm dồn kênh và phân kênh đã được các hãng tích hợp dưới dạng các mạch MSI. Một số mạch có thể kể ra như mạch giải mã 3/8 SN74138 để tạo tín hiệu chọn port (Chip Select hay CS), mạch dồn kênh 74151-74257, mạch cộng và mạch nhân v.v…

b) Mạch 3 trạng thái (Tri-state Component)

Trong hệ máy tính số, có nhiều khối chức năng cần thông tin, nhưng tại một thời điểm, bao giờ cũng chỉ có một khối đưa tín hiệu ra (dữ liệu) và một số hạn chế các khối thu nhận tín hiệu được điều khiển (đồng bộ) bằng chương trình máy tính. Thay vì nối dây dẫn liên kết các khối qua từng đôi phần tử một, các tín hiệu này được đưa lên BUS. Với các cổng logic thông thường, không thể nối trực tiếp chúng lên cùng một đường dây vì sẽ xảy ra tranh chấp BUS vì đoản mạch. Ví dụ đầu ra của phần tử A là 1 trong lúc đầu ra của phần tử B là 0. Các đầu ra của loại mạch này đều theo cấu trúc đẩy kéo (*pull-up),* nghĩa là có hai transistor được nối nối tiếp với nhau (xem hình vẽ), emitter của transistor này qua một diode rồi đến đầu ra, đến collector của transistor kia. Với hai trạng thái logic 1 và 0, tương ứng sẽ là T1 mở, T2 đóng và ngược lại, T2 mở và T1 đóng. Trên hình vẽ hiện tượng nguy hiểm xảy ra khi lối ra của phần tử logic A là 1, các khoá mở hay đóng tương đương việc transistor thông bão hoà hay ngắt, lối ra của phần tử logic B là 0 và hiện tượng đoản mạch xảy ra.

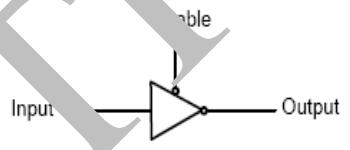


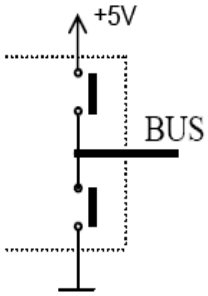
Hình 2.19 Các kiểu nối đầu ra, đầu ra trở kháng cao

Để tránh hiện tượng này, một loại cổng logic gọi là cổng 3 trạng thái (*tri-state gate HZ, hay 3 trạng thái: 0, 1 logic và trở kháng cao*) được sử dụng cho lối ra của các khối nối chung vào BUS. Hình sau là một phần tử đảo (*invertor*) với đầu ra 3 trạng thái.

Input	Enable	Output
0	1	HZ
1	1	HZ
0	0	1
1	0	0

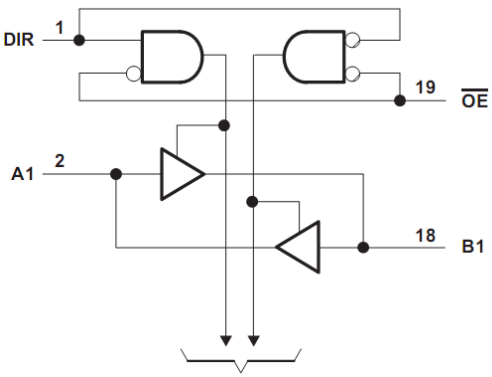
Bảng chân lý
mạch 3 trạng thái





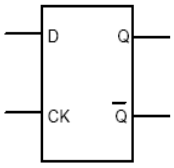
FUNCTION TABLE

INPUTS		OPERATION
$\overline{OE}$	DIR	
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation



Hình 2.20 Vi mạch 3 trạng thái: hai trạng thái logic và trạng thái thứ 3 HZ: đầu ra bị "tách" khỏi BUS.

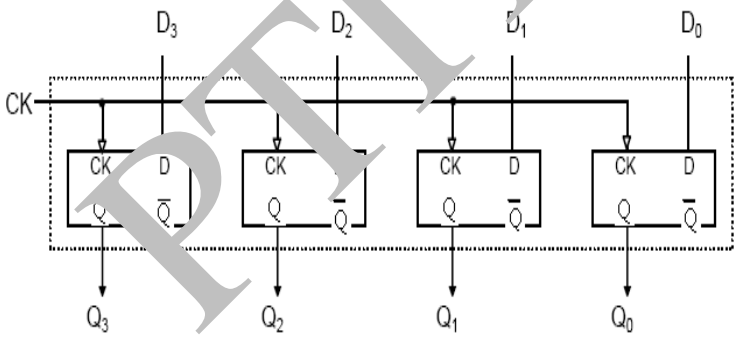
c) Mạch chốt, thanh ghi: Mạch chốt là một mạch gồm các phần tử có khả năng lưu giữ các giá trị 0 hoặc 1 ở lối ra. Có thể dùng D Flip-Flop làm một mạch chốt với tín hiệu để chốt dữ liệu CK tại đầu ra Q theo bảng giá trị chân lý sau: D là đầu vào, Q là đầu ra, CK là đồng hồ điều khiển.
Biết rằng Qn+1 = D với tín hiệu điều khiển là sự xuất hiện sườn dương của xung nhịp CK. Như vậy, giá trị logic (0 hoặc 1) tại D đã được chuyển sang đầu ra Q (chốt). Nếu CK giữ nguyên trạng thái bằng 1, thì trạng thái đầu ra Q được giữ nguyên. Như vậy, giá trị logic của D đã được lưu giữ ở Q (nhớ).



D	CK	Q	$\overline{Q}$
X	0	Q	Q_n
0	1	0	1
1	1	1	0

Hình 2.21 Mạch chốt (hay nhớ, gữi lại) kiểu D,
làm việc theo mức hay sườn lên của xung đồng hồ CK. (Xem thêm chi tiết mach SN 7474).

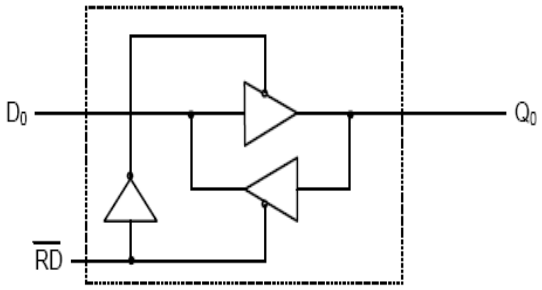
Để chốt 4 bit ta sử dụng 2 vi mạch 7474 và có sơ đồ như sau:



Hình2.22 Chốt 4 bit với D-Flip/flop

Với mạch SN 74374 sẽ "chốt" được 8 bit D7-D0. Với 2 mạch SN 74374 sẽ chốt 16 bit. Mạch này thường dùng cho BUS địa chỉ.

d) Mạch khuyếch đại BUS 2 chiều



Hình 2.23- Cổng khuyếch đại (driver) chốt hai chiều

Trên cơ sở của các mạch 3 trạng thái, các mạch khuyếch đại BUS hai chiều được xây dựng theo nguyên lý sau: Hai phần tử 3 trạng thái sẽ được ghép ngược với nhau, chân điều khiển sẽ dùng tín hiệu chọn chiều và đảo tín hiệu đó, ví dụ tín hiệu đọc RD. Khi xuất hiện tín hiệu RD, dữ liệu được phép đi từ Q0 sang D0, ngược lại, tín hiệu chỉ được phép đi từ D0 sang Q0 và cho phép CPU đưa tín hiệu ghi dữ liệu ra ngoài. Ghép nối đủ số phần tử cho tất cả các dây dữ liệu, ta có mạch khuyếch đại BUS hai chiều. Trong thực tế, mạch có chức năng trên đã được tích hợp theo chuẩn của TTL, được ký hiệu là SN 74244 hay SN 74245 (hoặc Intel 8288- Octal BUS Transceiver).

2.3 BO MẠCH một HTN VỚI CẤU HÌNH TỐI THIỂU

Tuy chưa đề cập chính xác về thiết kế phần cứng của một HTN, nhưng ví dụ sau đây hoàn toàn có thể được sử dụng như một HTN. Hệ được xây dựng theo kiểu HTN duqja trên bo vi xử lý và các linh kiện không cùng trong một chip đã nêu ở Chương 1.

Dưới đây là một mô hình tối thiểu xây dựng với CPU Intel 8085, có ROM Monitor và chương trình ứng dụng, RAM dữ liệu và vi mạch ghép nối với thiết bị ngoài.

Một thiết kế tối thiểu với CPU, ROM. RAM và Cổng I/O

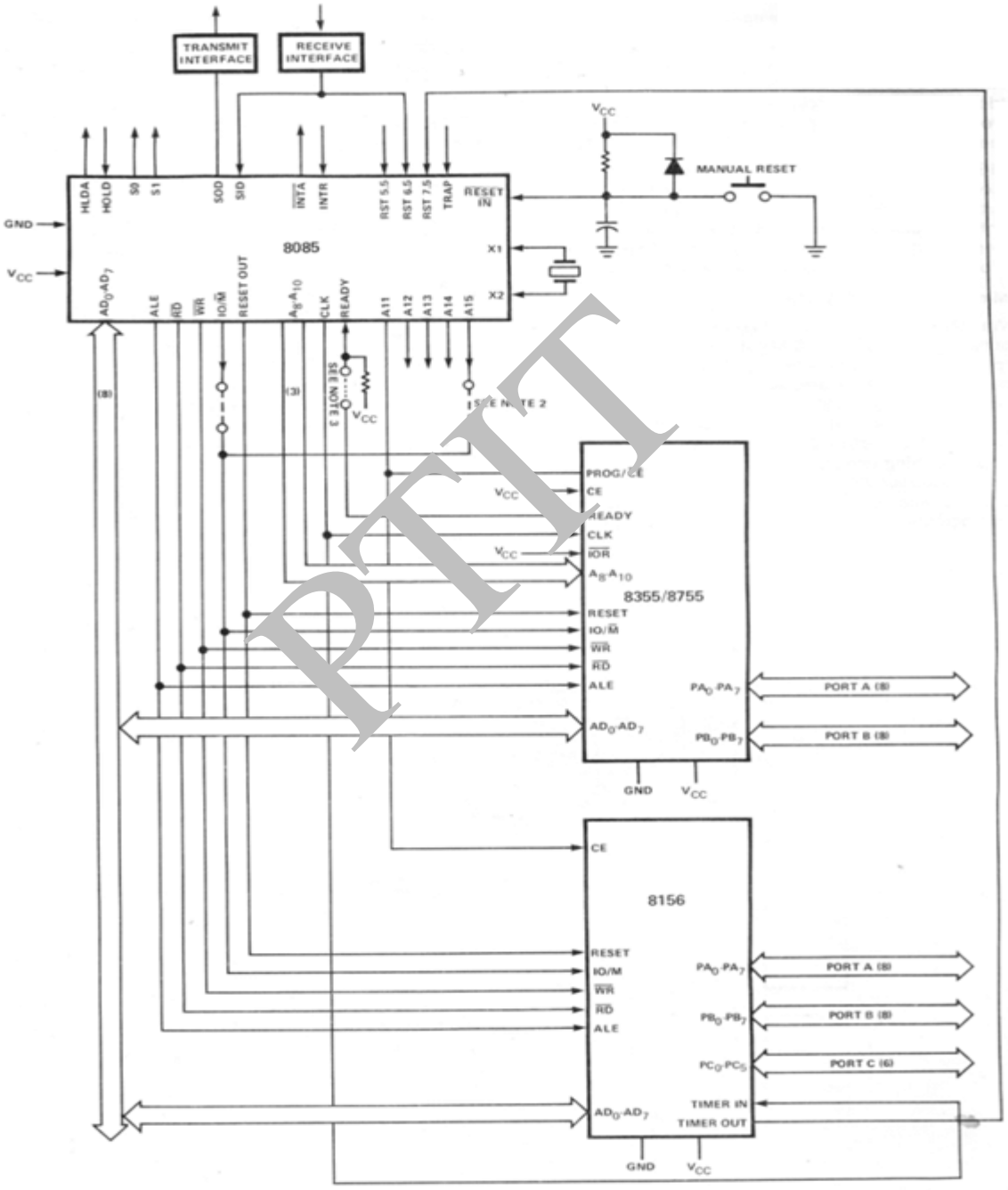
Các linh kiện cho bo mạch MSC-85:

✓	1	CPU 8085A	Xtal cho f=3.0 Mhz
✓	1	8755	2048 byte PROM
✓	1	8156	256 bye RAM Programmable Timer/Counter x Port 8 bit programmable 1 x Port 8 bit programmable. Các cổng sử dụng để ghép các thiết bị ngoài vào bo mạch.

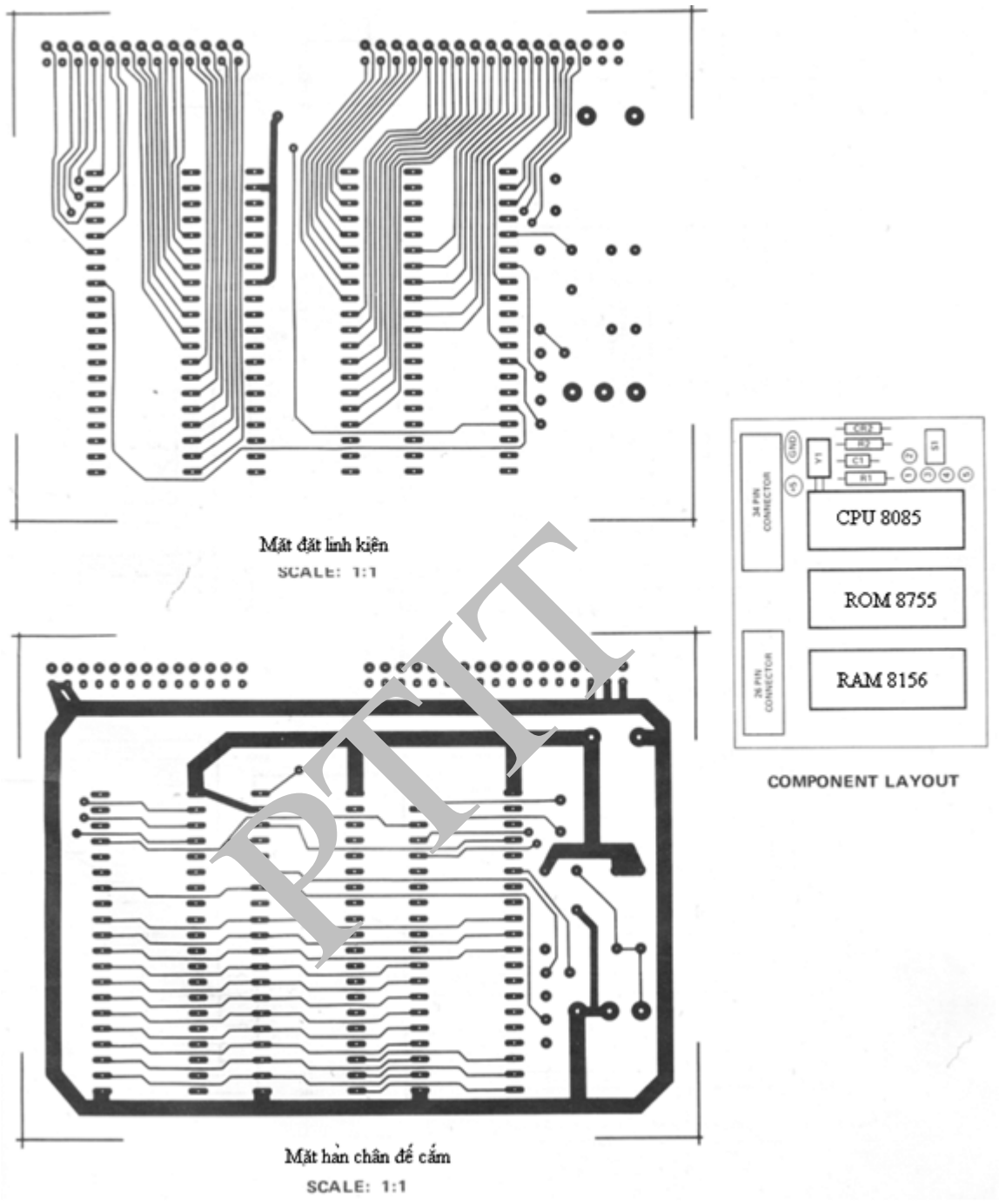
✓ Phần mềm hệ thống giám sát (*Monitor*) nạp vào ROM

Một thiết kế tối thiểu với CPU, ROM, RAM và Cổng I/O. Các cổng sẽ dùng khi thiết kế ghép nối với các thiết bị ngoại vi. Bộ nhớ có thể mở rộng khi có ý tưởng phân hoạch không gian nhớ cho ROM và RAM để mở rộng phần mềm, bao gồm HĐH và ứng dụng. CPU 8085 là CPU rất mạnh, họ 8 bít.

Sơ đồ chuẩn từ nguồn của Intel:



Hình 2.24 Cấu hình tối thiểu bo mạch CPU 8085, RAM/ROM/Ports



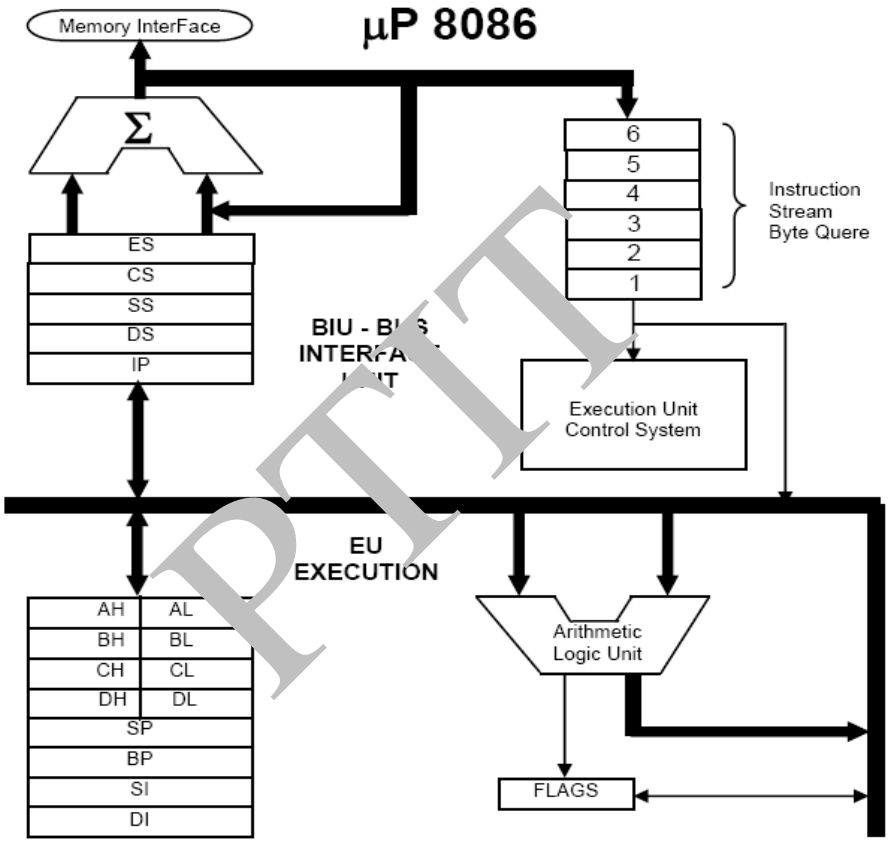
Hình 2.25 Mạch in cho hình 2.24

Khi xây dựng xong phần cứng (mạch in, hàn đế cắm, vi mạch …), tạo phần mềm để nạp vào ROM.

2.4 HTN VỚI CÁC CPU KHÁC NHAU

2.4.1 CPU đa năng 16 bit

Khi nhu cầu xử lý số liệu đòi hỏi có độ chính xác cao và thực hiện các chức năng bởi các giải thuật tinh xảo và đòi hỏi xử lý nhanh, thì các CPU 8 bit có thể không đáp ứng được, đặc biệt khi HTN có sử dụng các hệ điều hành như RTOS. Trong trường hợp này cần bộ xử lý với số bit biểu diễn số liệu lên đến 16, 32 bit. Với họ CPU đa năng, có thể chọn Intel 8086/8088. Dưới đây là mô hình kiến trúc của Intel 8986:



Hình 2.26 CPU Intel x86

Các thanh ghi bên trong:

Accumulator	AX
Base register	BX
Counter register	CX
Data register	DX
Source index	SI
Destination index	DI
Stack pointer	SP
Base pointer	BP
Code segment	CS
Data segment	DS
Stack segment	SS
Extra segment	ES
Instruction pointer	IP
Status flags	FL

15 0

Các CPU họ 80x86 được phát triển trên cơ sở công nghệ chế tạo CHMOS với mật độ tích hợp rất cao (VLSI), có độ tiêu hao công suất rất nhỏ. Sơ đồ khối chức năng của CPU 8086 được thể hiện trên như hình trên, gồm hai thành phần chủ yếu là *đơn vị ghép nối BUS* (*BIU – BUS Interface Unit*), *đơn vị thực thi lệnh* (*EU – Execute Unit*). Tất cả các thanh ghi và đường truyền dữ liệu trong EU đều có độ dài 16 bits. BIU thực hiện tất cả các nhiệm vụ giao tiếp với BUS bên ngoài: thiết lập khâu liên kết với BUS dữ liệu, BUS địa chỉ và BUS điều khiển. Dữ liệu được trao đổi giữa CPU với bộ nhớ khi EU có yêu cầu, song không được truyền trực tiếp tới EU mà thông qua một vùng nhớ RAM dung lượng nhỏ (6 bytes) được gọi là *hàng nhận lệnh trước* (*Instruction Stream Byte Queue PQ - Prefetch Quere*) rồi mới được truyền cho hệ *thống thực thi lệnh EU.* BUI bao gồm các thanh ghi đoạn mã, con trỏ lệnh, và bộ điều khiển BUS.
EU chính là "lõi" của họ CPU X86 và gần giống như CPU 8085 đã đề cập.
Có thể mô tả cách làm việc đơn giản như sau: Khi EU đang thực hiện một lệnh thì BUI đã tìm và lấy lệnh sau từ RAM ngoài và đặt sẵn vào PQ. Đây là *cơ chế đường ống* (*pipeline*), một kỹ thuật tăng tốc độ cho CPU. Kỹ thuật đường ống sử dụng một vùng nhớ RAM cực nhanh (PQ), làm tăng đáng kể tốc độ của bộ xử lý thông qua việc truy tìm lệnh ở PQ thay vì tìm từ bộ nhớ RAM bên ngoài. Nếu so với cách thức truyền thống, có thể coi thực tế thời gian lấy lệnh bằng 0. Có thể tìm hiểu thêm về CPU loại này từ các tài liệu chuyên môn khác.

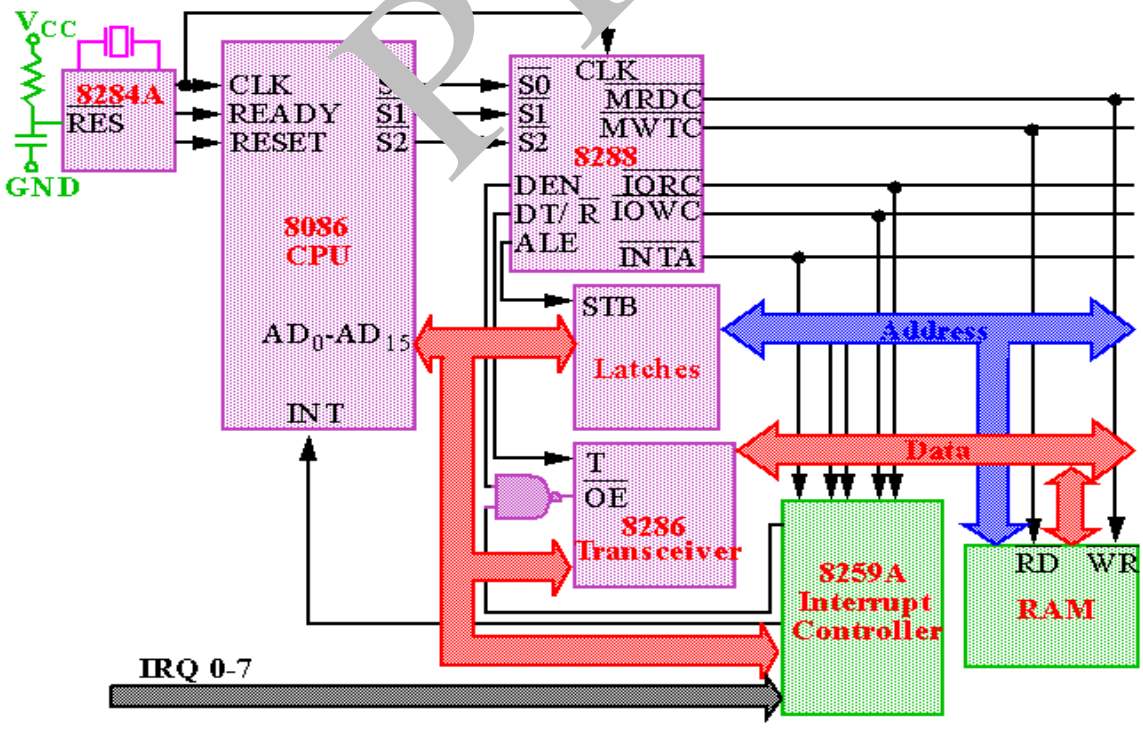
Cho dù CPU hoạt động bên trong có khác, nhưng những nguyên lí máy tính thì không có gì thay đổi khi thiết kế. Khi tìm hiểu kĩ tài liệu thiết kế với CPU cụ thể ta có thể lên được một sơ đồ cho bo mạch. Với CPU 8086, một thiết kế với qui mô tối thiểu sẽ như sau:

- CPU
- Mạch tạo xung nhịp đồng hồ 8284
- Mạch tạo các tín hiệu cho BUS điều khiển 8288
- Mạch chốt địa chỉ latch 16 bit với xung ALE

- Mạch khuyếch đại BUS dữ liệu 3 trang thái 8286

Mạch điều khiển ngắt (*sử dụng vi mạch chuyên dụng 8259 cho 8 ngắt, mở rộng tới 16 ngắt, tổ chức theo vector, khác với CPU 8085 !). Các vector ngắt được định nghĩa như sau:*

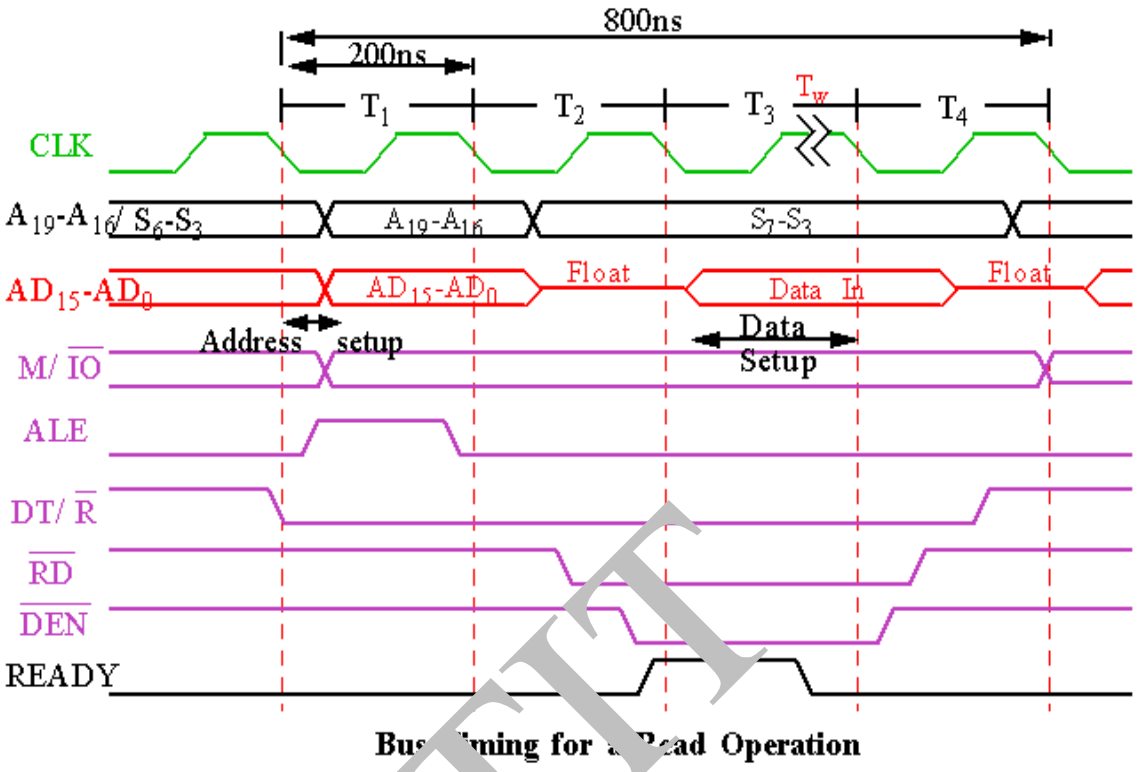
Số của vector	*Ý nghĩa của vector*
0	Divide error
1	Debug exception
2	Non-masked interrupt NMI
3	One byte interrupt INT
4	Interrupt on overflow INTO
S	Array bounds check BOUND
6	Invalid opcode
7	Device not available
8	Double fault
9	Coprocessor segment overrun
10	Invalid TSS
11	Segment not present
12	Stack fault
13	General protection fault
14	Page fault
15	Reserved
16	Coprocessor [illegible]
17-32	Reserved
33-255	INT [illegible] uctions



Hình 2.27 Bo mạch với tối thiểu với CPU 8086:BUS controller, Ngắt controller, RAM

Biểu đồ thời gian.

Để biết chức năng các tín hiệu trong sơ đồ, lấy ví dụ về chu kì đọc bộ nhớ sau đây:



Bus timing for a Read Operation

Hình 2.28 CPU 8086 timing: lệnh đọc

Ở T 1 :

- *Địa chỉ đặt lên BUS và kênh AD Address/Data bus.*
- *Tín hiệu điều khiển M/* IO *, ALE and DT/* R *sẽ cho biết thao tác là ttruy nhập bộ nhớ hay thao tác vào/ra (do mã lệnh OPCODE). Chốt địa chỉ lại, lạp hướng truyề dữ liệu trên BÚ dũ liêu.*

Ở T2 :

- *8086 phát tín hiệu* RD *hay* WR, DEN. Để ghi dua liệu,DEN *cho phép bộ nhớ hay thiết bị IO nhận dữ liệu để ghivà CPU nhận dữ liêu đọc.*

Ở T_3 :

- *Bộ nhớ nhận dữ liệu.*
- *READY dùng để kéo dài số trạng thái chờ nếu sử dụng RAM có thời gian truy nhập lâu (access time). READ được CPU ghi nhận ở cuối T*
  - *Nếu READ=low, T_3* thành T đợi (kéo dài trạng thái truy nhập RAM RD hay WR.
  - *READY=high, dữ liệu được ghi nhận cuối T3.*

Ở T4 :

- *BUS treo, chuẩn bị cho chu kì tiếp theo.*

Lưu ý: Khi thực hành đọc kĩ tài liệu CPU 8086 !

2.4.2 Bo mạch với CPU HARVARD (microcontroller Unit-MCU) họ Intel 8051/8052/8xC251

Intel phát triển chip đơn vi điều khiển (microcontroller - µC, VĐK) 8051 với kiến trúc Harvard vào năm 1980 và phổ biến cho tới những năm 1990. Những năm sau đó một số hãng khác trên thế giới tiếp tục phát triển và nâng cao, đa dạng, phổ biến rất rộng rải, dùng hầu như trong mọi lĩnh vực ở đâu cần tự động hóa xử lí dữ liệu.

Vi điều khiển có một số đặc điểm như sau:

3 Về phần cứng, VĐK giống như một máy vi tính cự nhỏ, được chế tạo chỉ trong một vi mạch (chip đơn). Như vậy có nghĩa các thành phần cấu trúc cho máy vi tính đều có ở đây. Nhưng còn hơn thế nữa, là các vi mạch chức năng rất đa dạng, bao gồm: các bộ biến đổi tương tự-số (Analog-Dital converter), số-tương tự (Digital-Analog converter), rất nhiều cổng ghép nối cho tín hiệu vào/ra ở các chế độ song song, nối tiếp, đồng thời, một vài bộ định thời (Time counter), các bộ nguồn áp chuẩn, có ROM (hay EPROM, hay FLASH), RAM …ngay trong vi mạch.

4 Cách khai thác ROM, RAM cũng khác, đó là tỉ lệ dung lượng ROM/RAM cao hơn (tức là ngược với máy vi tình, máy tính). Đó là vì chương trình xử lí cần có trong ROM, trong khi RAM chỉ là nơi "nháp" số liệu. ROM ở đây là các bộ nhớ không bị mất dữ liệu (non-volatile memory) với các công nghệ khác nhau, có dung lượng rất lớn, có thể chứa cả các hệ điều hành (ví dụ các router, chuyển mạch SW có IOS, ADSL hợp nhất các lớp 1, 2, 3 công nghệ mạng, …).

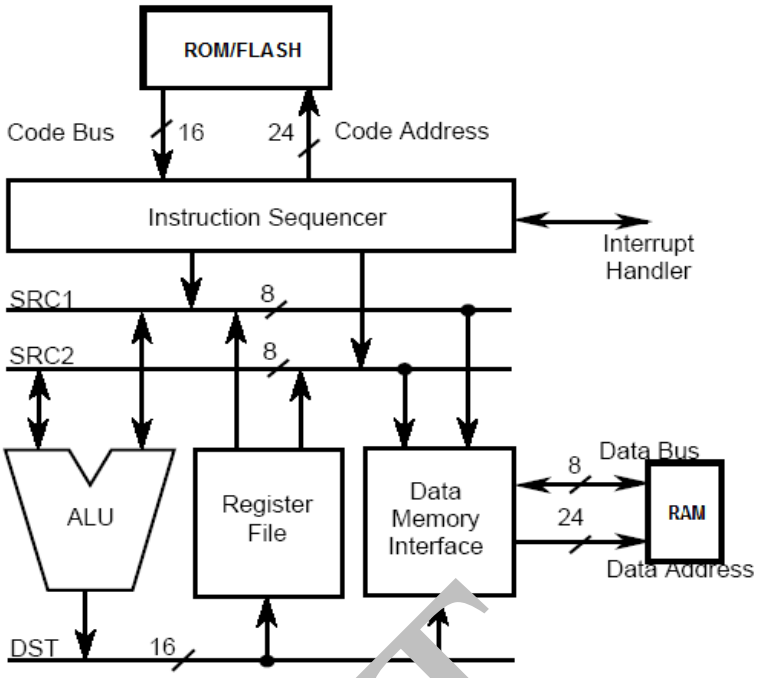
5 Tập lệnh của các VĐK thường không đa chế độ như CPU đa năng, nhưng vừa đủ cho các lớp ứng dụng, sao cho hiệu năng xử lí là cao nhất, đặc biệt là xử lí cho đáp ứng đầu ra nhanh nhất (tạm gọi là *đáp ứng kiểu thời gian thực*).

6 Do có mật độ tích hợp cao như vậy, và xuất phát từ thực tế, với công nghệ chế tạo chip ở từng thời kì phát triển, tần số làm việc của CPU thường không cao như các vi xử lí đa năng.

7 Do có cấu trúc như vậy nên thời gian phát triển cho lớp ứng dụng ngắn hơn, giá thành sản phẩm thấp hơn, công suất tiêu thụ thấp hơn, hiệu suất ứng dụng cao hơn, độ tin cậy cao hơn. Lẽ tự nhiên là chu kì phát tiển mới, cải tiến, nâng cấp, đa dạng sản phẩm ngắn hơn. Thậm chí trong khi không nâng cấp phần cứng, chỉ nâng cấp phần mềm, cũng mang lại sự thay đổi đáng kể của sản phẩm.

8 Kiến trúc của CPU cũng có khác, kiến trúc có tên là kiểu **Havard,** với các đặc điểm như sau:

Mô hình kiến trúc Havard như sau:



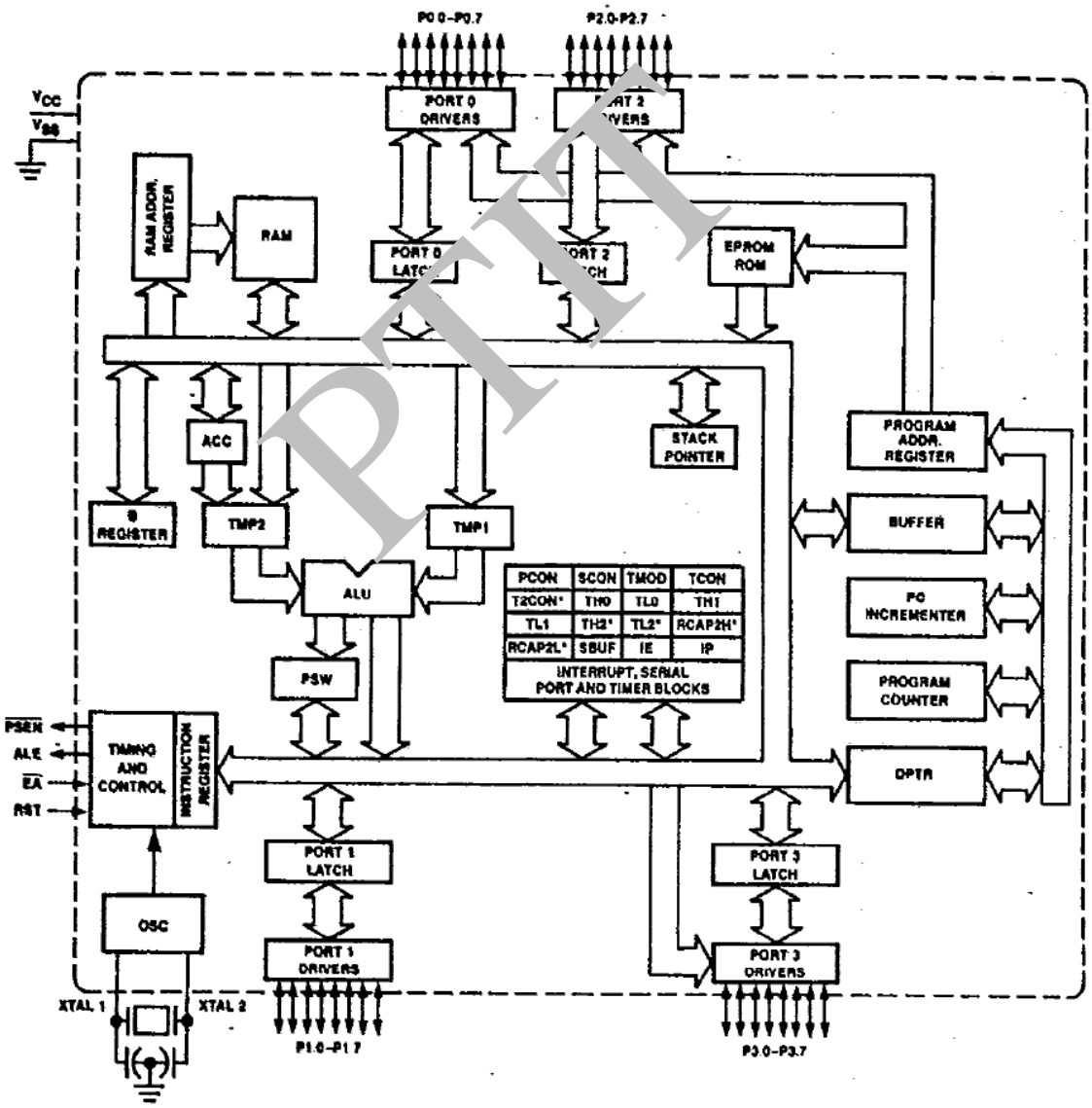
Hình 2.29 Mô hình kiến trúc Havard:
BUS cho bộ nhớ chương trình: Code Bus và Code Address;
BUS cho RAM dữ liệu: Data Bus và Data Address;
SRC1, SRC2:nguồn, DST: đích là các Bus nội bộ.

- ✓ Tách riêng các bộ nhớ dữ liệu (RAM) và bộ nhớ chương trình với các bus riêng rẽ để truy cập vào bộ nhớ dữ liệu (RAM) và bộ nhớ chương trình (NVM *Non Volatile Memory*: ROM, FLASH) chứa phần mềm nhúng (hệ điều hành, Device drivers, ứng dụng nhúng).
- ✓ Các bus điều hành độc lập, các chỉ dẫn chương trình và dữ liệu có thể được đưa ra cùng một lúc, cải thiện tốc độ so với thiết kế với chỉ một bus.
- ✓ Phân biệt rõ ràng bộ nhớ dữ liệu và bộ nhớ chương trình, CPU có thể vừa đọc một lệnh, vừa truy cập dữ liệu từ bộ nhớ cùng lúc.
- ✓ Do các BUS độc lập, CPU có khả năng tìm trước (*instruction prefetch*), nên với kiến trúc Harvard chương trình chạy nhanh hơn, bởi vì nó có thể thực hiện ngay lệnh tiếp theo khi vừa kết thúc lệnh trước đó.
- ✓ Tuy nhiên về kiến trúc có phần phức tạp hơn trong phần cứng, nhưng cho hiệu quả hơn cho các ứng dụng nhúng. Là loại phổ biến để thiết kế các HTN.

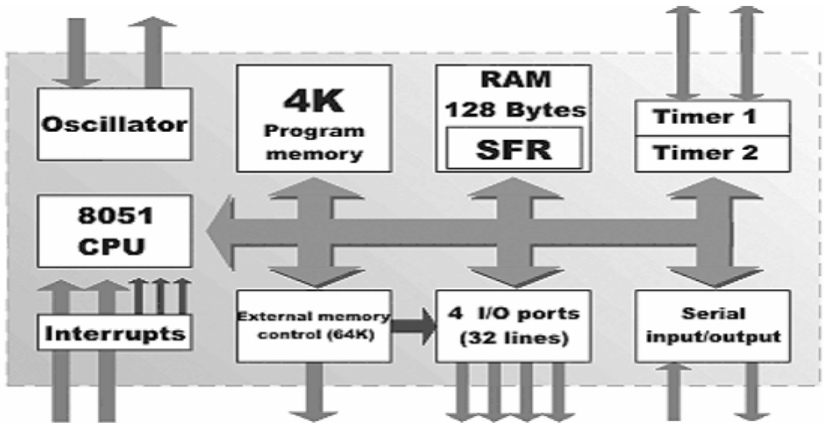
Hiện nay Intel không còn cung cấp các loại Vi điều khiển họ MCS-51 nữa, thay vào đó các nhà sản xuất khác như Atmel, Philips/signetics, AMD, Siemens,

Matra&Dallas, Semiconductors được cấp phép làm nhà cung cấp thứ hai cho các chip của họ MSC-51. Chip Vi điều khiển được sử dụng rộng rải trên thế giới cũng như ở Việt Nam hiện nay là Vi điều khiển của hãng Atmel với nhiều chủng loại vi điều khiển khác nhau. Atmel có các chip Vi điều khiển có tính năng tương tự như chip Vi điều khiển MCS-51 của Intel, các mã số chip được thay đổi chút ít khi được Atmel sản xuất. M. số 80 chuyển thành 89, chẳng hạn 80C52 của Intel khi sản xuất ở Atmel thành 89C52 (Mã số đầy đủ: AT89C52) với tính năng chương tr.nh tương tự như nhau. Tương tự 8051,8053,8055 có m. số tương đương ở Atmel là 89C51,89C53,89C55. Vi điều khiển Atmel sau này ngày càng được cải tiến và được bổ sung thêm nhiều chức năng tiện lợi hơn cho người dùng.

Ví dụ bo mạch xây dựng trên **CPU 8051** có tên chung là họ µC **MCS 51**.



Hình 2.30 Các khối chức năng của CPU 8051/8052



Hình 2.31 CPU 8051: EEPROM, RAM bên trong
và khả năng mở rộng bộ nhớ tới 128 KB (64 KB code+64 KB data)

Một HTN: Bo mạch vi điều khiển MCU (hay µC) với CPU 8052 và các vi mach hổ trợ cho ứng dụng nhúng: (http://www.keil.com/dd/docs/datashts/silabs/c8051f52x.pdf)

Hình 2.32 Bo mạch với CPU 8051/8052

Dưới đây là các thông số cho thấy các thành phần của bo mạch HTN hợp nhất SBS 8051F530:

Ngoại vi tương tự:

- 12-Bit ADC

Sai số ±1 bit nhỏ nhất (LSB INL) (C8051F52x/C8051F53x);

• Lập trình lấy mẫu với tốc độ tối đa đến 200.000 mẫu trong một giây (2000 ksps)

• Số kênh đầu vào: 6/16

• Giao diện kết nối dùng ngắt

• Có cảm biến nhiệt bên trong vi mạch

- Bộ so sánh

• Lập trình được với trễ đáp ứng

• Cói thể cấu hình để làm thiết bị đánh thức hay nguồn tạo tín hiệu RESET

• Dòng điện tiêu thụ thấp

- Khả năng tái khởi động khi nguồn nuôi thay đổi theo ngưỡng nhất định (POR/Brownout Detector)

- Điện áp chuẩn 1 .5 đến 2.2 V (lập trình được)

- Gởi rối ngay trên chip (On-Chip Debug)

Nguồn nuôi cho hoạt động 2.7 to 5.25 V

- Có ổn áp ở giá trị thấp (LDO regulator)

Nhân 8051 có tốc độ xử lý cao

- Cơ chế đường ống **(**Pipelined instruction architecture**)**, 70 % các lệnh thực hiện ngay ở 1 hay 2 đồng hồ hệ thống;

- Tốc độ xử lý tới 25 MIPS ở xung đồng hồ hệ thống 25 MHz

- Có khả năng mở rộng ngắt

Bộ nhớ

- 8/4/2 kB Flash; In-system byte programmable in 512 byte sectors

Ngoại vi số

- 16/6 cổng I/O; đầu ra có chế độ đẩy kéo, 5 V

- Cổng SPI™, và UART

- Giao tiếp mạng tốc độ thấp **qua bú LIN (**Hardware LIN - **L**ocal **I**nterconnect **N**etwork, có hai chế độ chủ/tớ, tương thích với chuẩn V1.3 và V2.0)

- Có 3 bộ đếm/định thời đa năng 16-bit khả trình, khả so sánh theo module, làm chó canh cổng (WDT-Watch Dog Timer)

Nguồn đồng hồ

- Đồng trong với bộ giao động 24.5 MHz , độ chính xác ±0.5% hỗ trợ cho các giao hoạt động của UART và LIN-Master

- Bộ giao động ngoài dùng thạch anh hay bộ giao động RC, hay xung đồng hồ nối vào qua kiểu nối 1 hay 2 chân đầu vào

- Có thể chuyển đổi nguồn đồng hồ ngay khi đang hoạt động (on-the-fly)

Cách đóng gói:

- 10-chân hàn dán QFN (3 x 3 mm)
- 20-chân hàn dán QFN (4 x 4 mm)
- 20-chân hàn dán TSSOP

Thang nhiệt môi trường làm việc: –40 to +125 °C

Có thể tìm hiểu thêm bo mạch Bo mạch 8052.com SBC tai www.8052.com

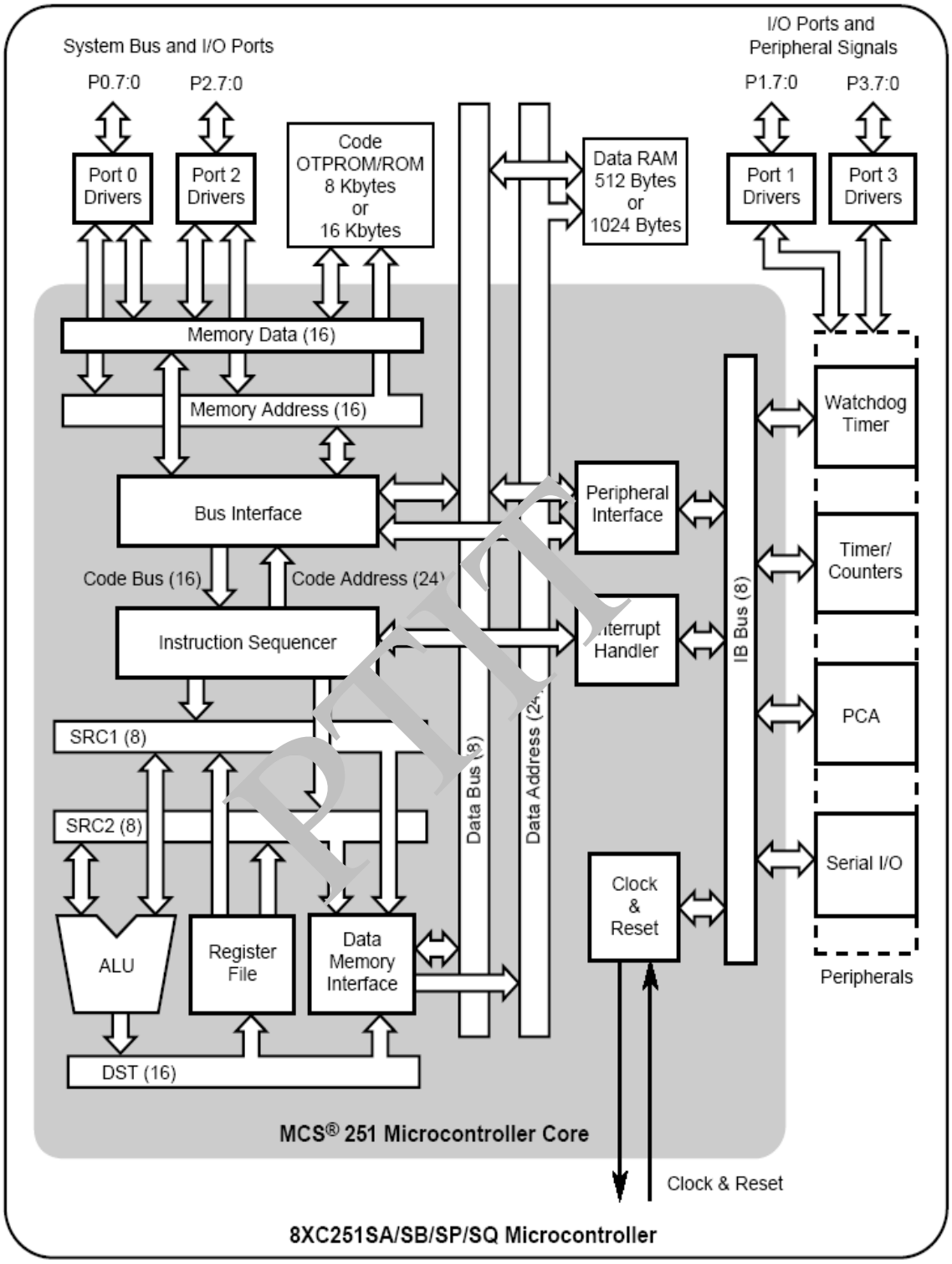
Bo mạch 8052.com Single Board Computer (SBC) sử dụng CPU Atmel AT89S8252/AT89S8253 và Dallas DS89C420, nhưng có thể cắm bất kì chip 40-pin 8052 tương thích (8052, 8051, 8032, 8031, etc.); SBC cũng sủ dụng cả với CPU mới AT89S8253 (loại thay thê chip AT89S8252). Có thể sử dụng SBC cho nghiên cứu, thiết kế hay triển khai các dự án ứng dụng HTN.

Hiện tại µCs **MCS 51** đã nâng cấp lên với **CPU 8XC251S*x*** và gọi là họ **MCS® 251.**

Họ MCS 251 có một số đặc tả sau đây:

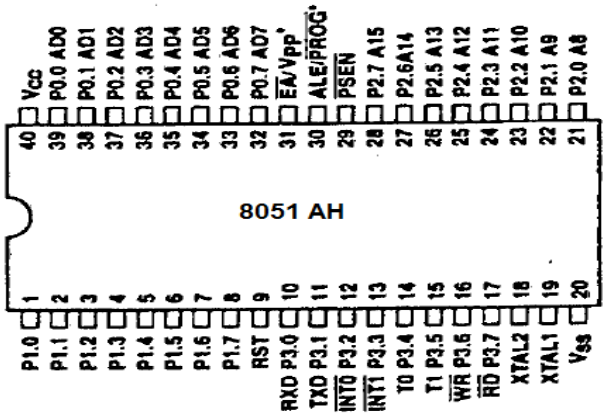
- Không gian địa chỉ hóa tuyến tính 24 bit cho bộ nhớ tới 16MB,
- CPU được xây dựng trên cơ sở các thanh ghi và các thanh ghi truy nhập được theo kiểu byte, từ (2 byte), từ kép (4 byte),
- Truy nhập bộ nhớ lệnh theo chế độ trang, tăng tốc chu kì tìm lệnh,
- Thực hiện lệnh kiểu đường ống (pipeline),
- Tập lệnh mạnh, thực hiện lệnh số học và logic 16 bit,
- Mở rộng ngăn xếp tới 64 K,
- Tực hiện lệnh (chu kì lệnh) trong 2 clock (trong khi MSC51 cần 12 clock),
- 3 giải pháp cho trạng thái đợi (wait state): real-time, RD#/WR#/PSEN#, and ALE,
- **Mã nhị phân tương thích hoàn toàn với MSC 51 do không gian địa chỉ của MSC51 được ánh xạ vào không gian địa chỉ của MSC251.**
- Với bộ nhớ lớn, hỗ trợ chương trình có mã lớn (Hệ điều hành + ứng dụng phức tạp),
- Chạy rất hiệu quả với mã viết từ ngôn ngữ C
- Điều khiển BUS động qua sử dụng các thao tác trạng thái đợi thời gian thực (real-time wait state).

CPU tiên tiến họ *8XC251Sx:*



Hình 2.33 Các khối chức năng của nhân 8XC251Sx

a) Tổng quan về CPU Intel 8051



Hình 2.34 CPU 8051

Ngoài các vi xử lý họ x86, Intel® còn thiết kế và sản xuất các vi xử lý chuyên dụng phục vụ các mục đích đo lường và điều khiển tự động, mà thực chất là các hệ thống nhúng. Các chip vi xử lý loại này đã vượt ra ngoài khuôn khổ của một *vi xử lý đơn thuần,* trở thành một máy tính kiểu *vi điều khiển* (*MicroController- µC*). Bỏ qua cách đề cập về kiến trúc bên trong (là kiểu *Harvard*), và nhìn theo kiến trúc xây dựng bên ngoài và để dễ mô tả và lập trình, ta có thể coi như hoạt động như một máy vi tính, thì "giống như" kiến trúc máy tính của *Von Neumann*. CPU được trang bị thêm bộ nhớ chương trình (ROM hoặc EPROM) và bộ nhớ dữ liệu, cũng như các cổng vào/ra nối tiếp, vào/ra song song, ngay trong vi mạch.

Vi mạch chủ yếu của họ MCS - 51 là chip µC 8051/8052, linh kiện đầu tiên của họ này được đưa ra thị trường. Chip µC8051 có các đặc trưng được tóm tắt như sau:

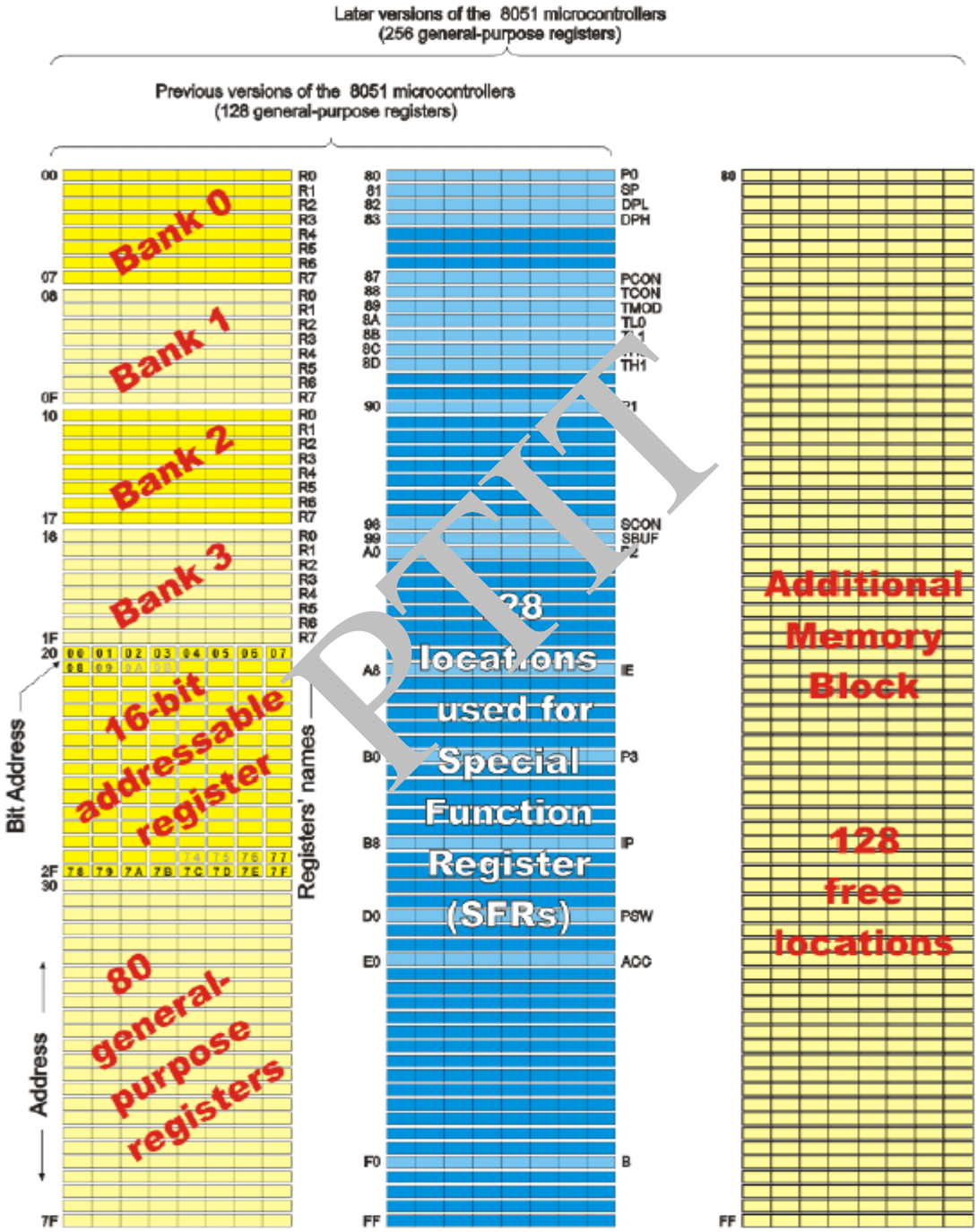
- 4 KB ROM và 128 byte RAM
- 4 *port* 8- bít, 32 lối vào/ra
- 2 bộ định thời (Timmer) 16 bít
- Mạch giao tiếp nối tiếp
- Không gian nhớ chương trình ngoài (mở rộng) 64K
- Không gian nhớ dữ liệu ngoài 64K
- Bộ xử lý bít (thao tác trên các bít riêng rẽ)
- 210 vị trí bit nhớ được định địa chỉ

Các thành viên khác của họ MCS-51 có các tổ hợp ROM (EPROM), RAM trên chip với dung lượng khác nhau, bộ biến đổi tín hiệu tương tự-số và số-tương tự, và có thể có thêm bộ định thời thứ ba. Mỗi một chíp của họ MCS-51 đều có phiên bản CMOS tiêu thụ công suất thấp.

Khi thiết kế HTN với CPU 8051, cần tham khảo tài liệu kĩ thuật. Một số lưu ý cần quan tâm bao gồm:

b) Phân hoạch địa chỉ

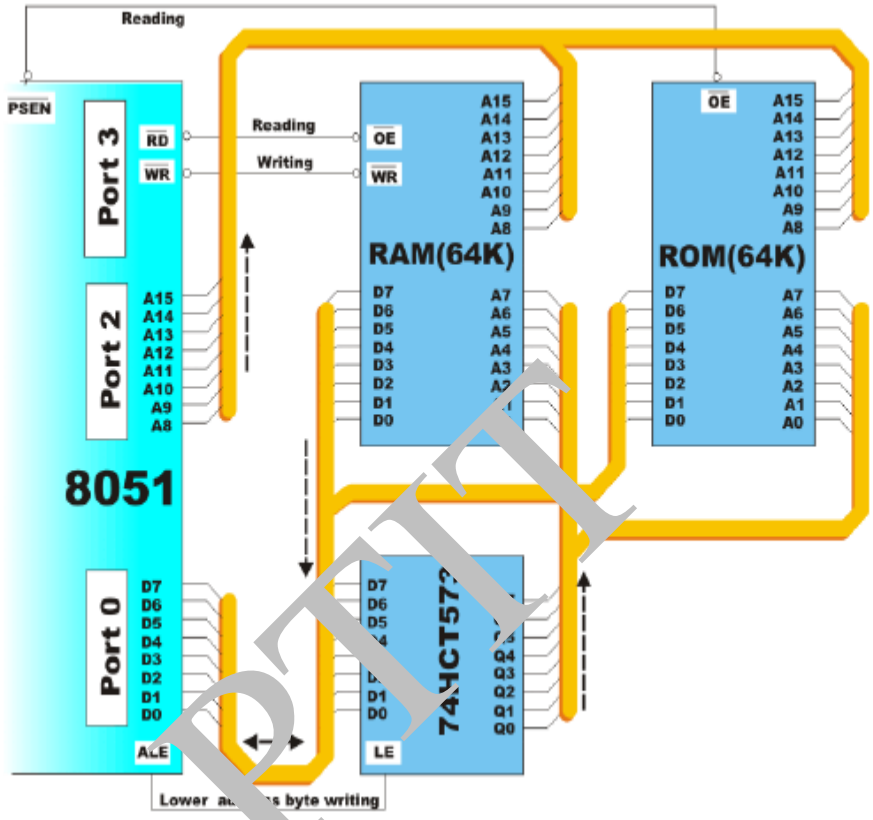
Do thu nhỏ kiến trúc để tạo ra vi điều khiển cho các ứng dụng đặc biệt, nên 8051 sử dụng cách địa chỉ hóa khác thường (so với các CPU đa năng) để mã hóa cho các thanh ghi và không gian bộ nhớ. Dưới đây là mô hình diễn đạt cách địa chỉ hóa đó:



Hình 2.35 Phân hoạch địa chỉ trong CPU 8051

c) Ví dụ thiết kế mở rộng bộ nhớ với CPU 8051

Vì dung lượng bộ nhớ trong Chip nhỏ đôi khi không đủ để chứa mã chương trình, nên cần mở rộng bộ nhớ qua một thiết kế mở rộng. Dưới đây là một thiết kế tạo ra 64KB ROM và 64KB RAM qua các cổng từ CPU 8051:



Hình 2.36 Bo mạch với CPU Intel 8051 và RAM, ROM mở rộng bên ngoài.

Truy xuất bộ nhớ chương trình ngoài

Bộ nhớ chương trình ngoài là bộ nhớ chỉ đọc, được cho phép truy xuất bởi tín hiệu PSEN. Khi có một EPROM ngoài được sử dụng, cả hai *port* 0 và *port* 2 không được sử dụng cho mục đích vào/ra. Kết nối 8051với bộ nhớ ngoài EPROM được trình bày ở hình trên.

Một chu kỳ máy của 8051 có 12 xung nhịp. Nếu bộ giao động trên chip có tần số 12MHz, một chu kỳ máy dài 1µsec. Trong một chu kỳ máy điển hình, ALE có 2 xung và 2 byte của lệnh được đọc từ bộ nhớ chương trình (nếu lệnh chỉ có một byte, byte thứ hai được loại bỏ).

Bộ nhớ dữ liệu ngoài là bộ nhớ đọc/ghi được cho phép bởi các tín hiệu RD và WR ở các chân của P3. Lệnh dùng để truy xuất bộ nhớ dữ liệu ngoài là: MOVX, sử dụng hoặc con trỏ dữ liệu 16-bít DPTR hoặc R0, R1 làm thanh ghi chứa địa chỉ.

RAM có thể giao tiếp với 8051 theo cách như EPROM ngoại trừ đường RD nối với đường cho phép xuất (OE) của RAM và WR nối với đường ghi (WR) của RAM. Các kết nối với bus dữ liệu

và bus địa chỉ giống như EPROM. Bằng cách sử dụng các *port* 0 và *port* 2 và các chip nhớ, ta có dung lượng RAM và ROM ngoài lên đến 64K

d) Lập trình cho µC8051

Để lập trình cho 8051, người lập trình cần nắm thật vững cách tổ chức rất hữu hiệu nhưng tương đối phức tạp của bộ nhớ RAM tích hợp trong chip. Không đơn thuần đóng vai trò bộ nhớ dữ liệu trong MCS51, nó còn sử dụng một phần bộ nhớ RAM để làm thanh ghi đa năng và thanh ghi với các chức năng đặc biệt.

Tồn tại chương trình Assembler riêng cho họ MCS51, lập trình hợp ngữ tương đương như lập trình hợp ngữ cho họ 80x86. Điểm mạnh tương ứng là tồn tại một phiên bản ngôn ngữ C cho MCS51, tạo điều kiện cho những ai đã quen với lập trình C có thể tạo các phần mềm ứng dụng để cài đặt vào trong bộ nhớ ROM của MCS 51 cho những ứng dụng thực tế.

Có thể tham khảo tài liệu

1. *Nguyễn Tăng Cường, Phan Quốc Khánh: Cấu trúc và lập trình họ Vi điều khiển 8051. NXB KH&KT Hà Nội-2004* về lập trình cho µC8051 được nêu ở cuối cuốn giáo trình này.
2. Bộ Kit 8051/251 Evalution Kit của KEIL Software là tài liệi được sử dụng phổ biến với các công cụ phát triển.

e) Các khả năng ứng dụng của µC8051 Thông thường, các trung tâm vi xử lý được dùng để xây dựng nên các máy tính. Riêng các trung tâm của Single Chip Microcomputer, do những cấu trúc đặc trưng và tính năng kỹ thuật, được ứng dụng nhiều trong các thiết kế nhỏ, đặc biệt là trong các hệ thống nhúng với số thành phần phụ trợ thêm vào tối thiểu nhất. Nhờ cấu trúc và khả năng cài đặt các chương trình ứng dụng ngay trong bộ nhớ ROM hoặc bộ nhớ Flash tích hợp sẵn, các hướng và các ứng dụng cụ thể của họ vi xử lý này chủ yếu tập trung vào các mục đích gia dụng và dân dụng. Thống kê một số lĩnh vực ứng dụng của các trung tâm vi xử lý họ này được liệt kê trong bảng sau.

CPU 8051 và các lĩnh vực ứng dụng:

- Điều khiển trong công nghiệp (System Supervision)
- Điều khiển động cơ
- Hàng không
- Máy y tế
- Truyền thông, truyền dữ liệu
- Mạng:
    - Vĩ mạng
    - Thiết bị không dây,

- Điện thoại di động,
- Bộ lặp lại (repeater),
- Chuyển mạch (switch)

- Video game, đồ chơi (toy)
- Thiết bị:
  - Thiết bị cầm tay, thiết bị di động
  - Thiết bị đầu cuối truy nhập dữ liệu
  - Hệ thống đèn hiệu đường sắt
  - Trạm vệ tinh mặt đất,
  - Hệ thống giám sát không dây.
  - Điện thoại Máy tính Hệ thống an toàn
  - Máy in Laze, máy in màu, máy nhắn tin
- Trong gia đình:
  - Đồ điện gia dụng Thiết bị đàm thoại Điện thoại Hệ thống an toàn Mở đóng cửa Trả lời tự động Máy FAX,
  - Máy tính gia đình TV Truyền hình cáp VCR Camera Điều khiển từ xa Trò chơi điện tử
  - Nhạc cụ điện tử, điều khiển ánh sáng

2.4.3 Vi mạch Hệ thống khả trình trong một Chip (*Programmable System on chip-PsoC*) và Máy tính thông minh khả trình (*Programmable Intelligent Computer-PIC*)

Là một công nghệ phát triển rất mạnh hiện nay, mà nền tảng cơ sơ là : hợp nhất tất cả các thành phần một máy vi tính, các vi mạch điện tử chuyên dụng như ADC/DAC, DSP … vào một vi mạch duy nhất với khả năng lập trình ứng dụng ngay trên vi mạch đó. Lĩnh vực ứng dụng là tạo ra các HTN với phổ cấu hình từ đơn giản tới phức tạp. Ở nước ta đây là xu hướng phát triển rất mạnh bởi dễ thực hiện và đặc biệt giá thành rất hợp lí.

PIC: PIC bắt nguồn là chữ viết tắt của "Programmable Intelligent Computer" (Máy tính khả trình thông minh) là một sản phẩm của hãng General Instruments đặt cho dòng sản phẩm đầu tiên của họ là PIC1650. Lúc này, PIC1650 được dùng để giao tiếp với các thiết bị ngoại vi cho máy chủ 16bit CP1600, vì vậy, người ta cũng gọi PIC với cái tên "Peripheral Interface Controller" (Bộ điều khiển giao tiếp ngoại vi). CP1600 là một CPU tốt, nhưng lại kém về các hoạt động xuất nhập, và vì vậy PIC 8-bit được phát triển vào khoảng năm 1975 để hỗ trợ hoạt động xuất nhập cho CP1600. PIC sử dụng microcode đơn giản đặt trong ROM, và mặc dù, cụm từ RISC chưa được sử dụng thời bây giờ, nhưng PIC thực sự là một vi điều khiển với kiến trúc RISC, chạy một lệnh một chu kỳ máy (4 chu kỳ của bộ dao động).

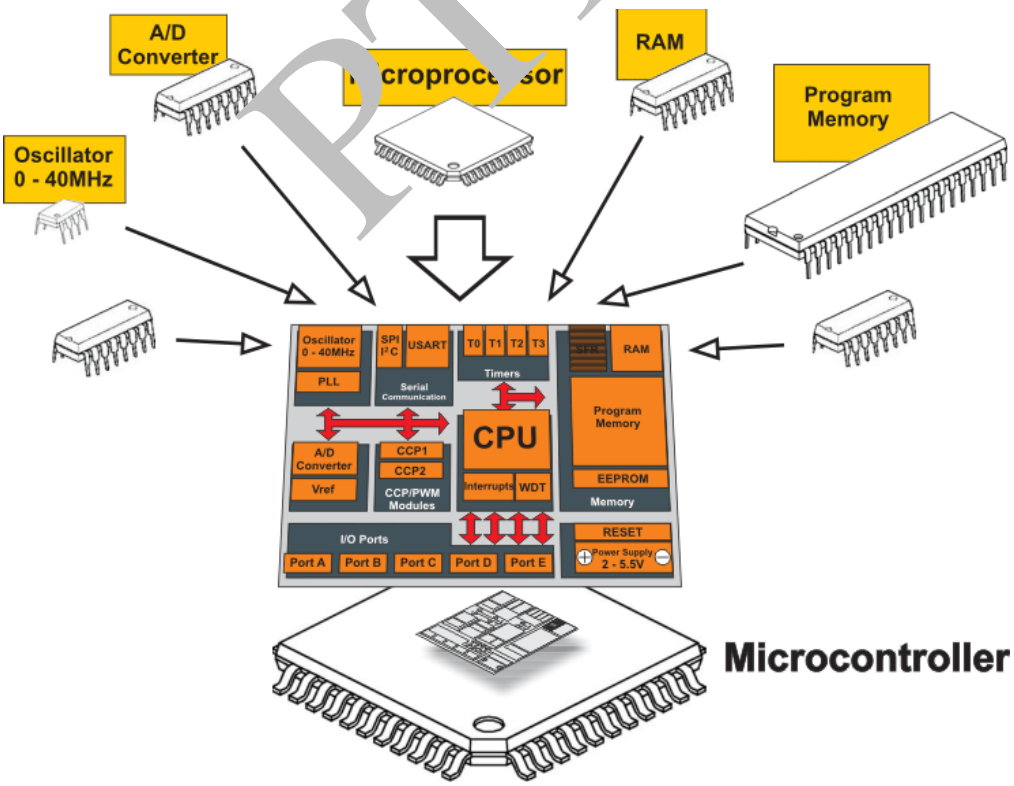
Năm 1985 General Instruments bán bộ phận vi điện tử của họ, và chủ sở hữu mới hủy bỏ hầu hết các dự án - lúc đó đã quá lỗi thời. Tuy nhiên PIC được bổ sung EEPROM để tạo thành 1 bộ điều

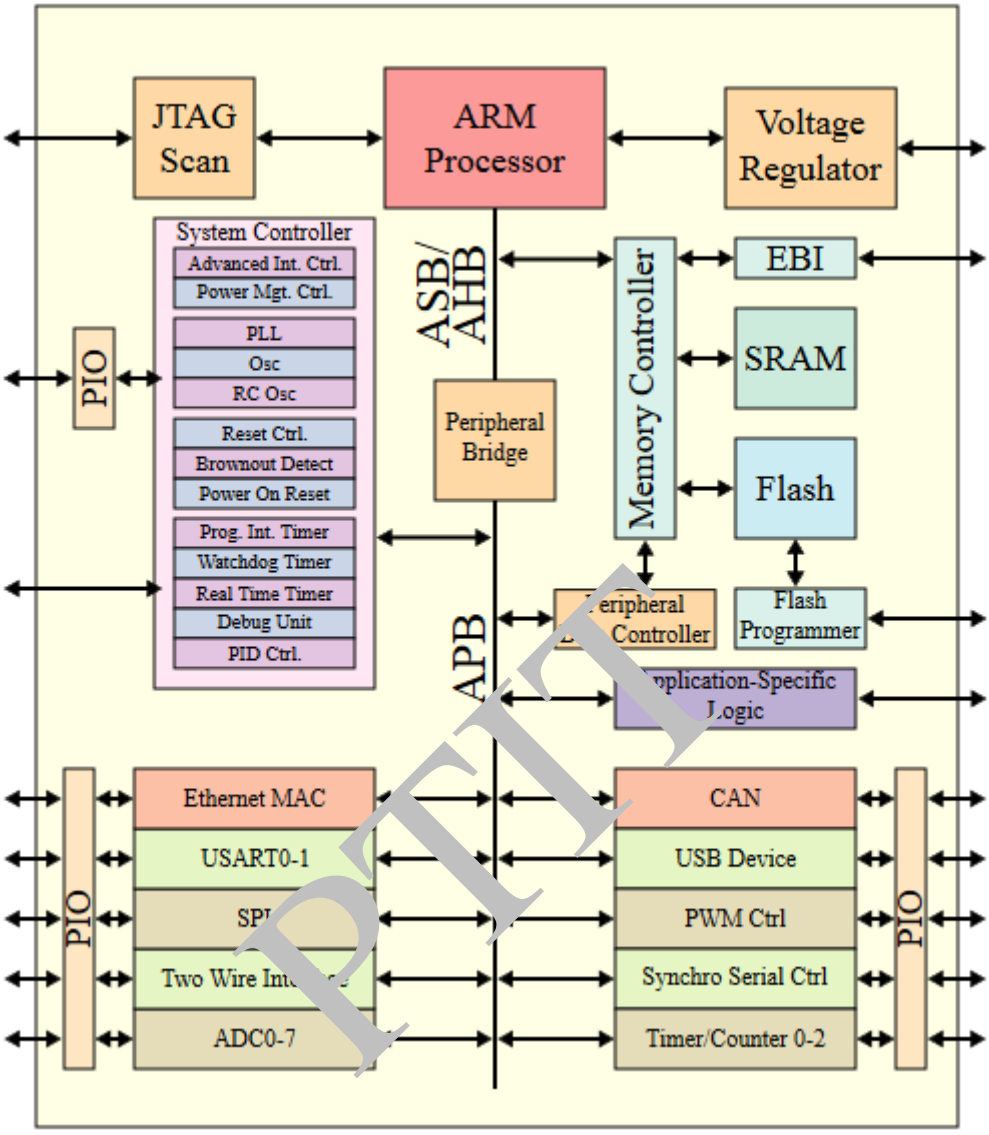
khiển vào ra khả trình. Ngày nay rất nhiều dòng PIC được xuất xưởng với hàng loạt các module ngoại vi tích hợp sẵn (như USART, PWM, ADC...), với bộ nhớ chương trình từ 512 word đến 32K word (1word=16 bít).

Về cấu trúc của PsoC/PIC: sự tích hợp nhiều thành phần trong một vi mạch

- Có một hay vài vi điều khiển (microcontroller), hay vi xử lý (microprocessor) và bộ xử lý tín hiệu số (Digital Signal Processor-DSP).
- Các khối bộ nhớ tùy chọn ROM, RAM, EEPROM và Flash.
- Nguồn đồng hồ chuẩn gồm bộ giao động thạch anh và mạch phản hồi dò chốt pha (phase-locked loops).
- Ngoại vi gồm bộ đếm định thời (counter-timers), đếm thời gian thực (real-time timers) và mạch tự động khởi động lại hệ thống (power-on reset).
- Ghép nối ngài theo chuẩn công nghiệp USB, FireWire, Ethernet, USART, SPI(*Serial Peripheral Interface)*.
- Ghép nối với tín hiệu tương tự ADC và DAC 8, [illegible] 12 bit.
- Bộ nguồn chuẩn, có độ chính xác cao.

  Ví dụ các thành phần hợp thành của một vi điều khiển hoàn chỉnh:





Các khối chức năng điển hình của CPU ARMv8-A (British company ARM Holdings)

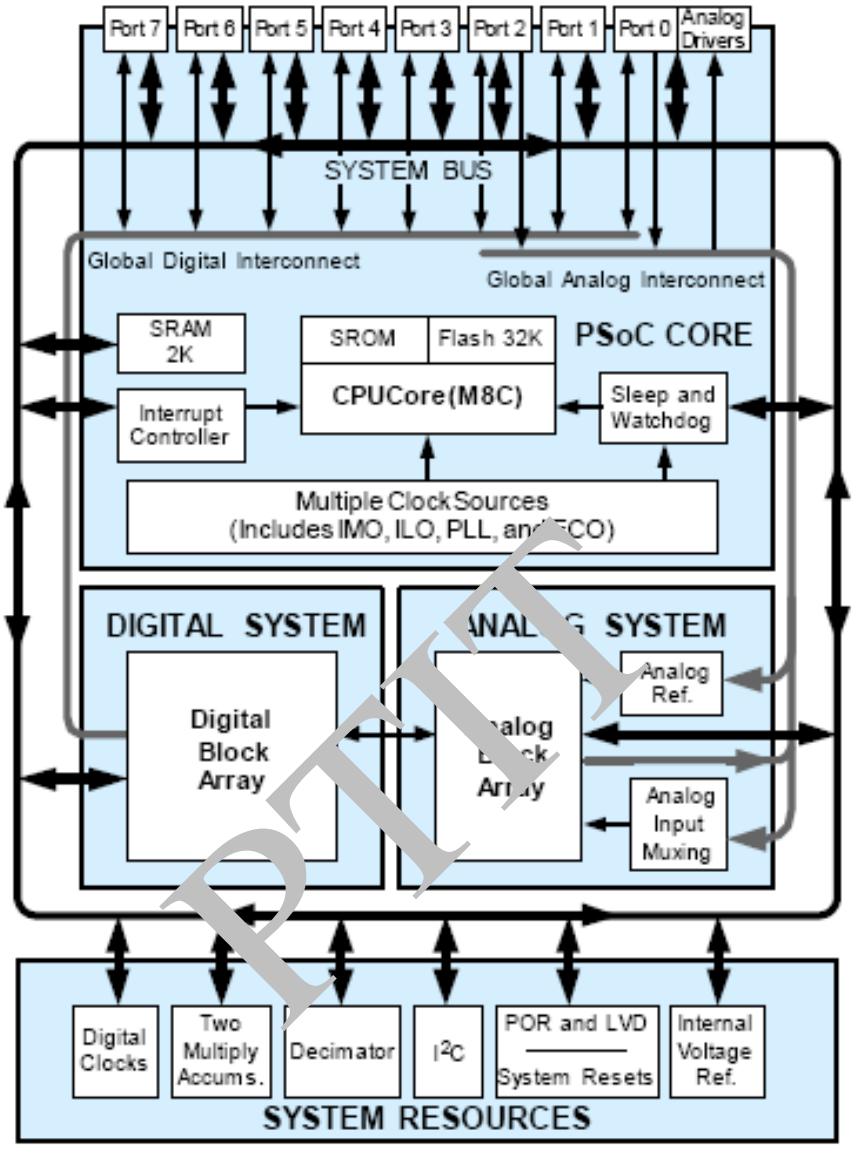
Hình 2.37 Mô hình một vi điều khiển kiểu PSoC hay PIC kiểu Vi xử lí trong một Chip (Microprocessor-based system on a chip)

✓ **Một số vi mạch PSoC là sảm phẩm của CYPRESS**

Ví dụ CY8C29466:

■ Bộ xử lý kiểu Harvard rất mạnh
* Tốc độ tới 24 MHz
* Hai bộ nhân 8x8, Thanh ghi tích lũy ACC 32-bit

* Tiêu thụ nguồn thấp ngay cả ở tốc độ xử lí cao
* Nguồn nuôi : 3.0V to 5.25V
* Môi trường nhiệt độ : -40°C to +85°C
* Hội đồng Điện tử và Tự động hóa (AEC) đã chứng nhận sản phẩm
■ Hệ thống các thiết bị ngoại vi tiên tiến (PSoC® Blocks)
* 12 Khối liên kết thiết bị tương tự cung cấp:
• Các ADC tới 14 bit
• Các DACs tới 9 bit
• Các bộ khuyếch đại khả trình
• Các bộ lọc và bộ so sánh khả trình
* 16 khối kỉ thuật số cung cấp:
• các bộ đếm và định thời 8 tới 32 bit, và bộ điều chế theo độ rộng của xung (PWM)
• các module CRC và PRS
• Có 4 UART chạy song công
• Đa bus SPI™ Masters or Slaves
* Bằng cách tổ hợp các khối cao ra thiết bị ngoại vi phức hợp
■ Đồng hồ khả trình, độ chính xác cao
* Đồng hồ trong 24 tới 48 MHz , ±5.0%
* Có thêm đồng hồ thạch anh ở tần số 32.768 kHz
* Tùy chọn đồng hồ ngoài tới 24 MHz
* Giao động nội cho Watchdog và Sleep
■ Bộ nhớ linh hoạt trên chip
* 32K Bytes Flash cho chương trình
* 2K Bytes SRAM cho Dữ liệu
* Lập trình trực tiếp qua liên kết nối tiếp (In-System Serial Programming -ISSP)
* Nâng cấp Flash từng phần
* Có chế độ bảo vệ linh hoạt (Protection Modes)
■ Các chân nối khả trình chức năng
* Dòng đầu ra 25 mA Sink, 10 mA Drive trên tất cả các chân (GPIO) của chip
* Đẩy kéo, điện trở treo, trở kháng cao, hở mạch đầu ra ở tất cả các chân của chip.
* Có tới 12 đầu vào tương tự (Analog Inputs on GPIO)
* 4 đầu ra tương tự cung caaso tới 40 mA (Analog Outputs on GPIO)
* Các ngắt khả cấu hình trên tất cả các chân ngắt của chip
■ Các nguồn tài nguyên hệ thống khác:
* Kết nối giữa các IC kiểu chủ/tớ với tốc độ 400 kHz (I2C Slave, Master, and Multi-Master to 400 kHz)
* Watchdog and Sleep Timers
* Khả cấu hình để phát hiện nguồn nuôi thấp
* Hợp nhất mạch giám sát (Integrated Supervisory Circuit)
* Có nguồn điện áp chuẩn chính xác trong chip
■ Công cụ phát triển đầy đủ
* Phần mềm phát triển tự do (Free Development Software (PSoC Designer™))
* Đầu đủ công cụ mô phỏng hiajt động kiểu ICE (Full-Featured, In-Circuit Emulator and Programmer)
* Mô phỏng chạy với tần số tối đa (Full Speed Emulation)
* Ccó cấu trúc tìm lỗi hoàn hảo (Complex Breakpoint Structure)
* 128K Bytes nhớ cho lưu dấu vết hoạt động (Trace Memory)
* Ghi nhận sự kiện đầy đủ
* C Compilers, Assembler, and Linker



Hình 2.38 Vi điều khiển PSoC CY8C29466

- ✓ **Vi mạch PIC của hãng Microchip Technology**

http://www.best-microcontroller-projects.com/pic-microcontroller.html

Một PIC giống như một thiết bị mạnh được thiết kế và tích hợp nhiều thành phần kiểu module như sau:

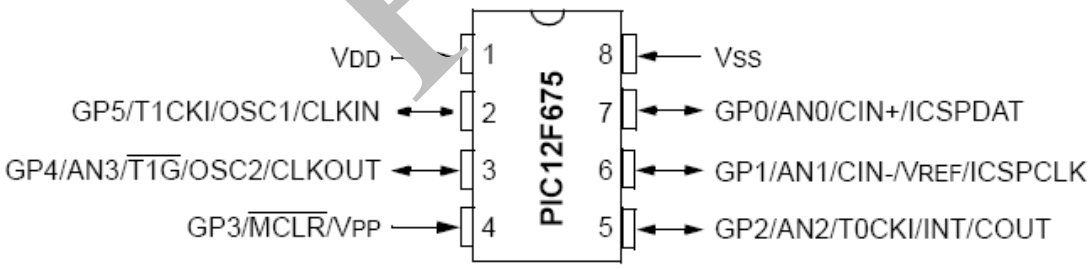
- EEPROM.
- Timers (các bộ định thời).
- Analogue comparators (các bộ so sánh tương tự).
- UART (bộ thu phát dị bộ vạn năng).
- Các cổng kết nối ngoại vi, mở rộng bộ nhớ

Với các module như vậy, có thể triển khai các dự án ứng dụng như:
*** Frequency counter** – Đếm tần số, sử dụng các bộ định thời bên trong, kết quả thông báo ra ngoài bằng cách truyền thông đi xa UART (RS232), hay ra LCD tại chỗ.
*** Capacitance meter** - Điện dung kế, bằng các bộ giao động so sánh tương tự.

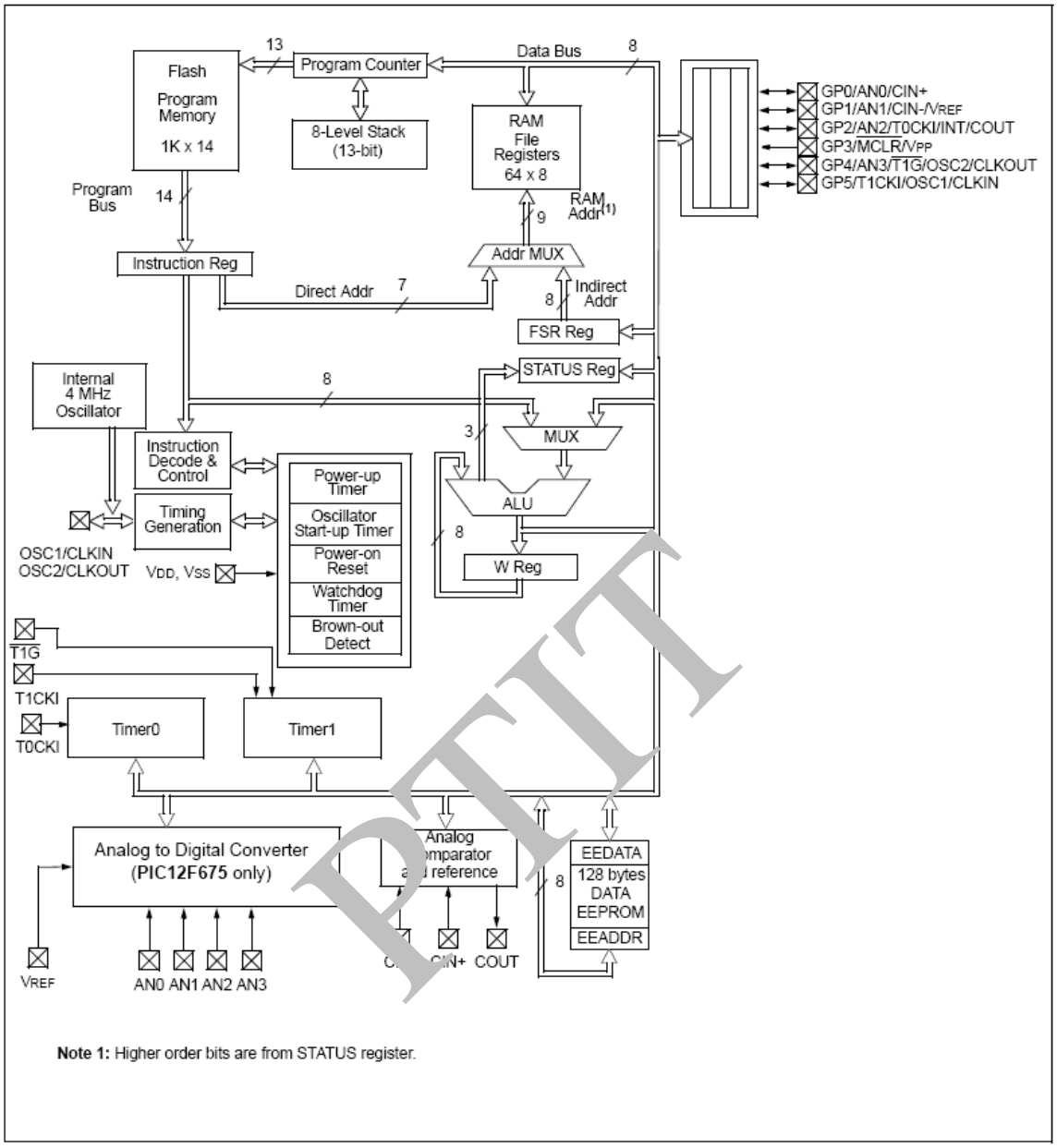
*** Event timer –** Định thời sự kiện với các bộ định thời bên trong.

*** Event data logger** – Thu thập dữ liệu theo định thời và số hóa bằng DAC bên trong, lưu dữ liệu trong EEPROM hay lưu trong bộ nhớ ngoài qua tuyến truyền dữ liệu tốc độ cao I2C.
*** Servo controller –** Ứng dụng tự động hóa kiểu điều khiển servo sử dụng bộ điều chế độ rộng xung PWM với phần mềm điều khiển servo, kết hợp truyền thông qua UART đến thiết bị đầu cuối.

Ví dụ PIC 12F675: 8 chân (Dual In Line) có các ngoại vi tích hợp như sau:



Hình 2.39 Bố trí võ-chân PIC 12F675



Hình 2.40 Mô hình khối chức năng **PIC12F629/675**

Đặc trưng:

- Two timers.
- One 10 bit ADC with 4 selectable inputs.
- An internal oscillator (or you can use an external crystal).
- An analogue comparator.
- 1024 words of program memory.
- 64 Bytes of RAM.

- 128 Bytes of EEPROM memory.
- External interrupt (as well as interrupts from internal peripherals).
- External crystal can go up to 20MHz.
- ICSP : PIC standard programming interface.

Các PIC hạng trung bình có không gian địa chỉ từ 1 KB đến 8 KB (ví dụ họ 18FXYZ). Bộ nhớ xem ra không nhiều, nhưng với tập lệnh hiệu quả có thể xây dựng các ứng dụng rất hữu ích. (Xem ví dụ ở PHỤ LỤC : LM35 temperature sensing project).

Lập trình: Sử dụng giao diện nối tiếp ICSP để lập trình, có thể lập trình bằng hợp ngữ, hay C.

Vào/Ra - I/O: PIC có các cổng kết nối sử dụng cho nhiều mục đích, ví dụ cho điều khiển motor bước, đóng cắt các rele, đọc các nút ấn đóng/mở, hiển thị ra LCD/7--segment/LED, đo tần số, sử dụng ADC với nhiều mục đích khác nhau …

Ngoại vi (Peripherals): Các thiết bị tích hợp bên trong rất nhiều và hữu hiệu cho các dữ án ứng dụng, tùy vào yêu cầu có thể chọn dòng PIC với các đặc trưng khác nhau. Các đặc trưng tổng quát gồm có:

PIC microcontroller Feature	**PIC microcontroller Feature description**
Flash memory	Re-programmable program storage.
RAM	Memory storage for variables.
EEPROM	Long term stable memory : Electrically Erasable Programmable Read Only Memory.
I/O ports	High current Input/Output ports (with pin direction change).
Timers/Counters	Typically 3.
USART	Built in RS232 protocol (only needs level translator chip).

CCP	Capture/Compare/PWM module.
SSP	I2C and SPI Interfaces.
Comparator	An analogue comparator and internal voltage reference.
ADC	Analogue to digital converter.
PSP	Parallel Slave Port (for 8 bit microprocessor systems).
LCD	LCD interface.
Special features	ICSP,WDT,BOR,MCLR,PWRT,OST,SLEEP
ICSP	Simple programming using In Circuit Serial Programming.

Flash memory: Bộ nhớ cho chương trình.

ICSP (In Circuit Serial Programming): Cổng giao diện với hệ phát triển, lập trình trực tiếp cho PIC đã nằm trên bo mạch thiết kế. Qúa trình lập trình/thử nghiệm được đơn giản hóa theo kiểu **in the circuit** cho tới khi hoàn thiện phần mềm.

Cổng I/O: Sử dụng để ghép nối với các thiết bị ngoài.

Các chân nối: Hầu hết các chân nối có thể là IN hay OUT đơn lẽ hay nhóm qua lập trình. 1 chân nối có thể làm nhiều chức năng theo từng thời điểm khác nhau, ví dụ chân RA0 của cổng PORTA: out-đưa dữ liệu cho 7-segment LED, sau đó là cổng in-đọc dữ liệu vào. Công suất các cổng ra có thể cho tới 25 mA.

Định thời (Timer / Counters): Thường có 3 bộ đếm dùng để làm định thời, dung như bộ đếm hay định thời.

Timer 0: 8 bit, có khả năng đặt trước một giá trị với tín hiệu đầu vào có từ bên trong (Fosc/4) hay từ bên ngoài. Khi đếm tràn giá trị sẽ tạo ra một ngắt, được dùng như một watch dog.

Timer 1 : 16 bit, với khả năng đặt trước giá trị. Đếm tràn sẽ tạo ra một ngắt.
Timer 2 : 8 bit, với khả năng đặt trước giá trị. Sử dụng đặt giá trị gốc thờii gian cho module PWM.
USART: Cổng nối tiếp RS 232 dùng kết hợp với mạch tạo ra V24, ví dụ vi mạch MAX232, hay SP202ECP.
Baud Rates: đặt tốc độ cho UART, có thể hoạt động tới 38.4kbaud.
CCP (Capture/Compare/PWM) điều chế PWM với 3 chế độ:
Capture – ghi nhận sự kiện
Compare – So sánh khi Timer 1 đạt giá trị đặt trước.
PWM - Pulse Width Modulation.
Capture : Lấy dữ liệu của Timer 2 khi chân CCP xuống lên cao hay xuống thấp do lập trình.
Compare: So sánh khi Timer 1 đạt giá trị, và đưa vào CCPR1. Dùng để khởi động ADC.
PWM: Điều chế PWM với độ phân giải 10 bit.
SSP (Synchronous Serial Port)**:** kết nối với thiết bị theo giao thức SPI SPI (Serial Peripheral Interface) hay I2C(Inter IC communication). (project I2C here, more info I2C here).
Comparator and comparator voltage reference: Module so sánh có 2 bộ so sánh tín hiệu tương tự, có thể lập 1 trong 8 cách làm việc khác nhau: đầu vào có thể là tương tự hay số, so sánh với các mức điện áp chuẩn. Điện áp chuẩn được tạo ra bên trong chip, nối vào 1 đầu vào của cả 2 bộ so sánh, với cách dồn kênh có thể ứng dụng cho 4 tín hiệu vào/bộ so sánh. Đầu ra của bộ so sánh có thể gởi ra ngoài qua một chân của vi mạch. Mức tín hiệu vào nằm trong giới hạn Vdd và Vss.
ADC : Phân giải cấp độ 10 bit, 10 đầu vào tương tự dồn kênh.
PSP: Cổng song song 8 bit, đọc/ghi, chế độ slave, cho phép một hệ thống khác giao tiếp trực tiếp vào CPU từ ngoài với tín hiệu chọn chip (CS-chip select). Hệ bên ngoài nhìn CPU như thiết bị kiểu ánh xạ bộ nhớ (memory mapped I/O), như vậy PIC CPU là một hệ con (slave) của hệ kia, và có thể lạp trình cho PIC thực hiện các tác vụ theo yêu cầu.
LCD: Giao diện với LCD, ví dụ với LCD module HD44780.
Các đặc tả đặc biệt:

ICSP	In Circuit Serial Programming	click here (jumps to ICSP section).

WDT	Watch dog timer	This is a software error protector.
BOR	Brown Out reset	This detects if the power supply dips slightly and resets the device if so.
POR	Power on reset	This starts microcontroller initialization.
PWRT	PoWeR up Time	A time delay to let Vdd rise.
OST	Oscillator start up timer	Wait for 1024 cycles after PWRT.
SLEEP	PIC microcontroller sleep mode	Enter low power mode.

WDT (Watch dog timer): sử dụng khi phần mềm chạy có vấn đề, sẽ RESET cứng CPU về trạng thái ban đầu. Để cấu reset cứng CPU, khi phần mềm làm việc bình thường, cần phát sinh lệnh CLRWDT cài trong chu trình, và nạp lại giá trị định thời cho WDT. WDT chạy theo đồng hồ riêng.

POR (Power On Reset): Khởi động PIC microcontroller khi có sườn lên của tín hiệu MCLR.

PWRT: nếu cho phép PIC microcontroller sẽ khởi động sau 72 ms.

OST (Oscillator Startup Timer): độ trể cần 1024 chu kì để có tần số ổn định.

SLEEP (Sleep mode -or low power consumption mode) hoạt động khi thực hiện lệnh 'SLEEP'. PIC sẽ "thức dậy" bởi các sự kiện như: RESET ngoài, Watch Dog Timer, INT - peripheral interrupt.

Một số PIC với các dòng khjasc nhau: 12F, 16F:

PIC	PIC	PIC

microcontroller Device	microcontroller No. Pins	microcontroller Flash memory WORDS
12F675	8	1k
16F88	18	4k
16F877A	40	8k

Dung lượng bộ nhớ Flash của PIC Microcontroller

Bộ nhớ của PIC thường không lớn, nhưng với tập lệnh RICS vẫn có thể chạy nhiều ứng dụng, tuy nhiên nếu cần bộ nhớ lớn hơn, hãy chọn PIC 16F với 4 - 8Kword (hay 8-16KB) hay mở rroojng qua liên kết I2C.

Dung lượng bộ nhớ RAM và EEPROM của PIC microcontroller

RAM chứa biến và các dữ liệu tức thì, hay tạm thời, lưu ý khi ứng dụng với độ dài dữ liệu không quá lớn. Không nên dùng dấu phẩy động nên dùng số nguyên dài dấu phẩy cố định

PIC microcontroller EEROM cho dữ liệu hằng số.

PIC hiện nay rất phổ biến ở nước ta trong giảng dạy và thực hành ứng dụng. Có thể tham khảo tại: http://www.picvietnam.com/forum/showthread.php?t=10

Dưới đây là ví dụ loại PIC 16F882/883/886, đang sử dụng khá rộng rải ở Việt nam:

28-pin PDIP, SOIC, SSOP

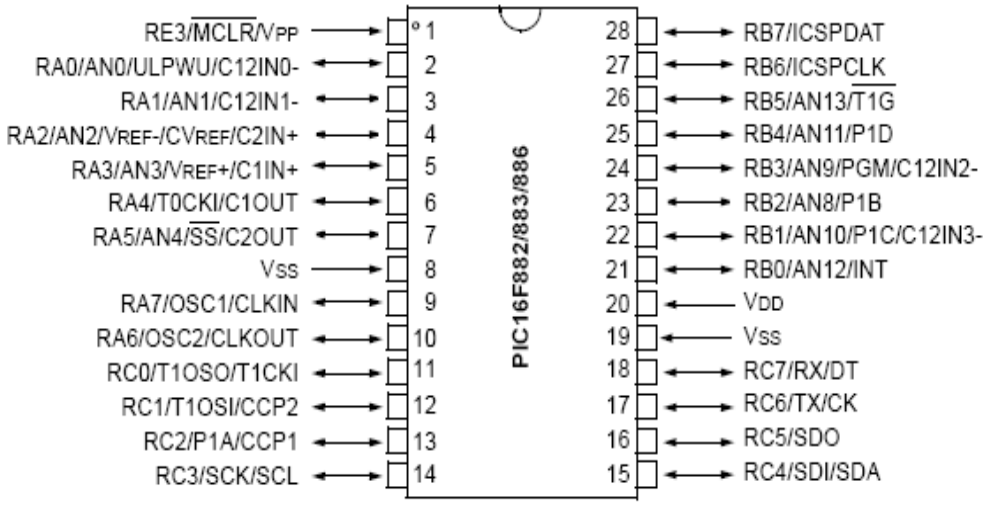
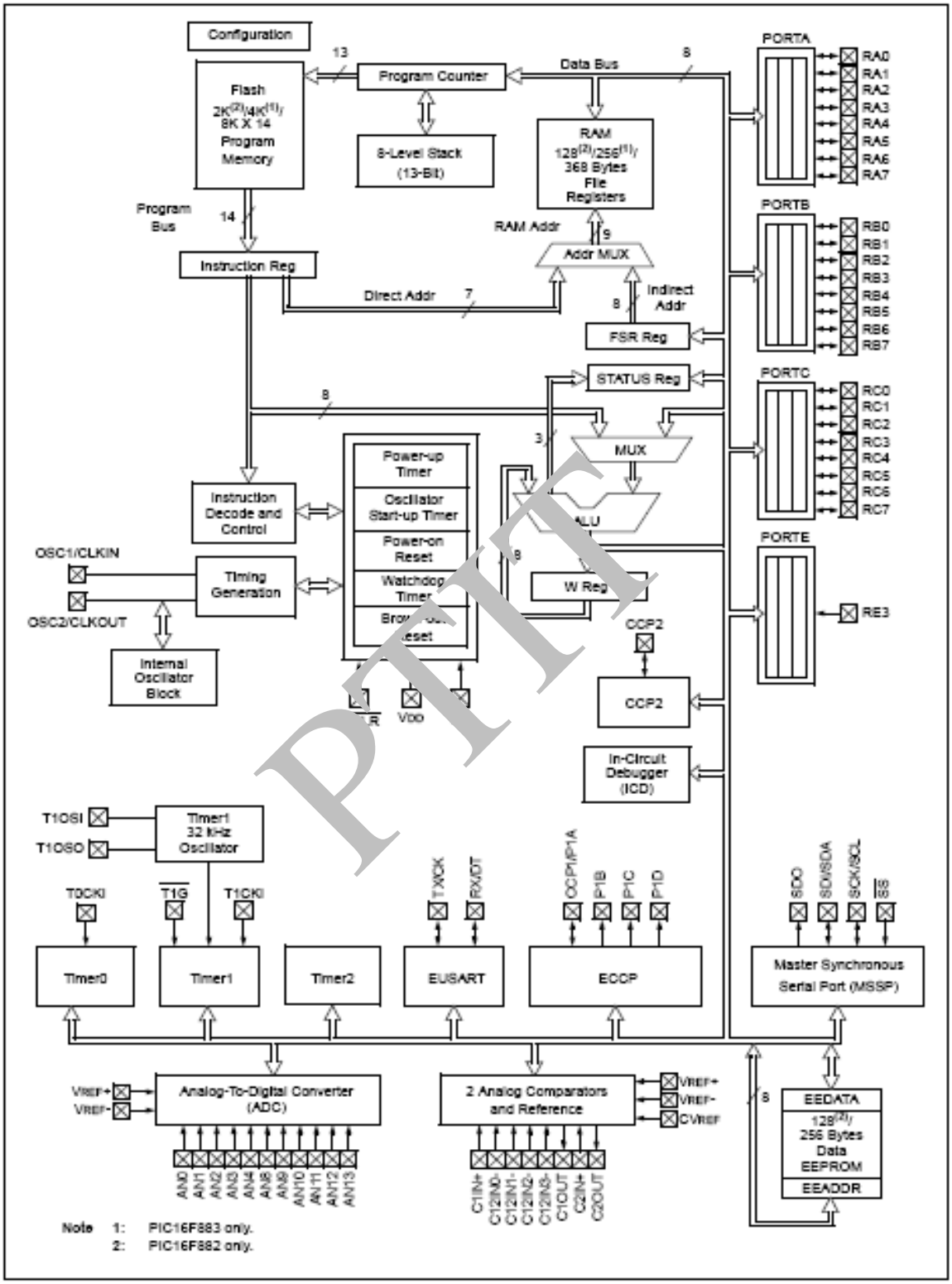


FIGURE 1-1: PIC16F882/883/886 BLOCK DIAGRAM



Hình 2.41 Vi điều khiển PIC 16F882/883/88

CPU RISC hiệu năng cao:
• Chỉ cần 35 lệnh máy :
- Tất cả các lệnh đều thực hiện chỉ trong 1 chu kì lệnh, trừ các lệnh nhảy
• Tốc độ hoạt động:
- DC – 20 MHz tần số đầu vào (clock input)
- DC – 200 ns cho một chu kì lệnh
• Ngắt
• 8-Level Deep Hardware Stack
• Csac chế độ địa chỉ: Direct, Indirect và Relative
Các đặc điểm đặc biệt của vi điều khiển:
• Giao động nội chính xác lập bởi nhà máy ±1%
- Khả trình lập tần số từ 8 MHz xuống đến 31 kHz
- Tinh chỉnh bằng phần mềm
- Hai tốc độ khởi động chọn khi khởi động
- Phát hiện sự cố ở thạch anh cho các ứng dụng tới hạn
- Khả chuyển đồng hồ ngay khi đang hoạt động để tiết kiệm năng lượng
• Có chế độ "ngủ" tiết kiệm năng lượng
• Thang nguồn nuôi rộng :2.0V-5.5
• Chụi đựng môi trường nhiệt công nghiệp cao
• Khởi động khi bật nguồn (Power-on Reset (POR))
• Định thời bật nguồn (Power-up Timer (PWRT)) và khởi động giao động (Oscillator Start-up Timer (OST))
• Khởi động với phần mềm
• Bảo vệ mã khả trình
• Flash/EEPROM Cell bền vững lâu dài với:
- 100,000 lần ghi vàoFlash
- 1,000,000 lần ghi vào EEPROM
- Flash/Data EEPROM sống tới hơn 40 years
• Gở rối trong chip (In-Circuit Debugger (on board))
Tiêu thụ năng lượng thấp:
• Dòng ở chế độ nghỉ (Standby Current):
- 50 nA @ 2.0V
• Dòng làm việc (Operating Current):
- 11μA @ 32 kHz, 2.0V,
- 220μA @ 4 MHz, 2.0V
• Dòng với Watchdog Timer:
- 1μA @ 2.0V, typical
Các ngoại vi:
• 24/35 chân vào/ra (I/O) khả năng kiểm soát hướng :
- Dòng tải trực toeesp cho đèn LED
- Thay đổi chức năng ngắt ở chân chip
- Khả trình ở từng chân chip
- Module với các bộ so sánh:
- 2 bộ so sánh tương tự
- Khả trình điện áp chuẩn (CVREF) module (% of VDD)
- Điện áp chuẩn cố định (0.6V)
- Có mạch lật kiểu SR làm thanh chốt
- Có công cho bộ đếm ngoài nối vào cho phép đếm
• Bộ biến đổi ADC:
- Độ phân giải 10-bit với 11/14 kênh
• Timer0: Bộ đếm 8-bit (Timer/Counter) khả trình giá trị tỉ lệ chia đặt trước (Programmable Prescaler)
• Timer1 tiên tiến:
- 16-bit timer/counter khả trình giá trị tỉ lệ chia đặt trước
- Có cổng chốt để nối tín hiệu từ bên ngoài
- Bộ giao động tinh xảo 32 kHz
• Timer2: 8-bit Timer/Counter with 8-bit Period Register, Prescaler and Postscaler
• Enhanced Capture, Compare, PWM+ Module:

- 16-bit Capture, max. resolution 12.5 ns
- Compare, max. resolution 200 ns
- 10-bit PWM with 1, 2 or 4 output channels, programmable “dead time”, max. frequency 20 kHz
- Đầu ra của điều chế PWM cho điều khiển lái

• Module nhận, so sánh, điều chế PWM:
- 16-bit Capture, max. resolution 12.5 ns
- 16-bit Compare, max. resolution 200 ns
- 10-bit PWM, max. frequency 20 kHz

• Module USART:
- Hỗ trợ kết nối RS-485, RS-232, và LIN 2.0
- Tự động phát hiện tốc độ truyền thông
- Tự động trở lại hoạt động khi phát hiện bit START

• Lập trình truyền thông nối tiếp qua 2 chân của chip (In-Circuit Serial ProgrammingTM (ICSPTM))

• Module truyền thông đồng bộ chủ (Master Synchronous Serial Port (MSSP) Module) hỗ trợ kết nối 3 dây SPI (với cả 4 chế độ làm việc) và kết nối I2C™ có Chủ/tớ với mặt nạ địa chỉ (
Master and Slave Modes with I2C Address Mask)

Do khuôn khổ giáo trình, các hướng dẫn khai thác và thiết kế với các dòng chip này không đưa vào tài liệu. Khi triển khai tìm đọc tài liệu và làm theo hướng dẫn của nhà cung cấp. Thông thường từ nhà cung cấp có các thiết kế mẫu và công cụ phát triển (Cách lập trình, công cụ để nạp chương trình vào chip) hỗ trợ nghiên cứu và triển khai ứng dụng.

2.5 BỘ NHỚ VÀ THIẾT KẾ BỘ NHỚ

Bộ nhớ là phương tiện lưu trữ thông tin bao gồm chương trình và số liệu trong hệ thống máy tính. Phần giới thiệu chung này đề cập tổng quan và những khái niệm cơ bản nhất về bộ nhớ. Bảng sau đây cho một số thông tin cơ bản về bộ nhớ trong máy tính:

Memory Type	Access Time	Cost /MB	Typical Amount Used	Typical Cost
Registers	1ns		1KB	
Cache	5-20 ns		1MB	
Main memory	60-80ns		MB	
Disk memory	10 ms		GB	

2.5.1 Một số thông số chính của mạch nhớ

Độ dài của ô nhớ: Độ dài của ô nhớ cho biết số bit chứa trong ô nhớ, có thể tính bằng bit, byte (8 bit), từ (16 bit), từ đúp (32 bit) hay từ kép (64 bit).

Dung lượng (*capacity*) của mạch nhớ xác định số bit hay byte hay từ cực đại mà mạch nhớ có thể chứa. Giả sử mạch nhớ có n bit địa chỉ đầu vào và mỗi địa chỉ (hay ô nhớ) độ dài là m bit, như vậy mạch nhớ có dung lượng 2^n x m (bit). Đơn vị đo dung lượng bộ nhớ thông thường nhất là: Byte(B), KiloByte (1KB=2^{10}B), MegaByte (1MB=2^{20}B), GigaByte (1GB=2^{30}B), TetraByte (1TB=2^{40}B)...

Thời gian thâm nhập (Acces Time) là thời gian từ thời điểm áp địa chỉ tới BUS địa chỉ tới khi khi nội dung của ô nhớ đó được đưa ra BUS số liệu, ký hiệu là t_A , thời gian này phụ thuộc vào công nghệ chế tạo và cấu trúc mạch nhớ.

Chu kỳ đọc (Read Cycle) là thời gian kể từ khi áp địa chỉ để đọc ô nhớ cho đến khi có thể áp địa để đọc ô nhớ tiếp theo, ký hiệu là t_{RC}. Đó là thời gian ngắn nhất giữa hai lần đọc một ô nhớ.

Chu kỳ ghi (Write Cycle) là thời gian kể từ khi áp địa chỉ để ghi ô nhớ cho đến khi có thể áp địa để ghi ô nhớ tiếp theo, ký hiệu là t_{WC}. Đó là thời gian ngắn nhất giữa hai lần ghi mạch nhớ.

Trạng thái đợi t_W (wait state) là số CPU Clock chen vào chu kì đọc/ghi để thích ứng với mạch nhớ do t_A của mạch quá dài. Nói cách khác nếu dùng một module nhớ chậm cho máy tính nhanh, cần đồng bộ với hoạt động của CPU để đảm bảo máy chạy ổn định thì cần thêm một vài t_W, mỗi t_W = 1 T clock của CPU, Thông số mỗi chip DRAM đều có ghi các thông số kĩ thuật cần cho thiết kế, ví du: DRAM Hitachi HM51V17400BS tổ chức như sau: 4M x 4 bits, 2K refresh, Extended Data Out (EDO), Speed (-6)= 60ns, Supply = 3.3 Volt. Có thể xem và điều chỉnh thông số này trong BIOS của các PC.

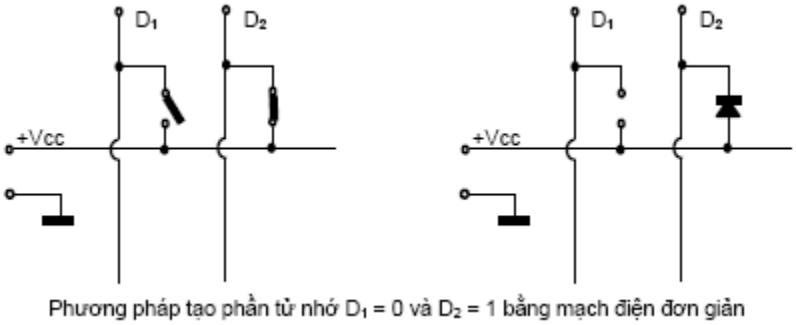
Tần số của mạch nhớ là lượng thông tin lớn nhất có thể đọc hay ghi vào mạch nhớ trong thời gian 1 giây.

$$f = 1/t_M$$

Trong đó $t_M = Max (t_{RC}, t_{WC})$

Phần tử nhớ

Phần tử nhớ thông thường là một mạch điện có thể ghi lại và lưu giữ một trong hai giá trị của một biến nhị phân, hoặc 0 hoặc 1, được gọi là **bit**. Trên mạch điện dưới đây, trên dây D1 sẽ không có điện áp (do công tắc mở), trong khi dây D2 có điện áp (vì công tắc đóng, hay thông qua diode mắc theo chiều thuận), gần bằng giá trị nguồn nuôi Vcc, tương ứng với bit D1 = 0 và bit D2 = 1.

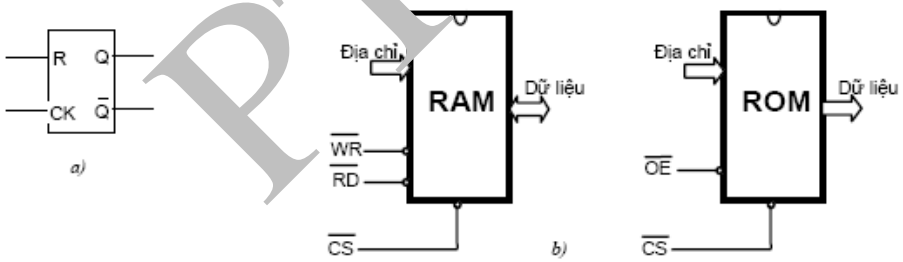


Hình 2.42 Cách tạo bit nhớ cố định bằng công tắc cơ học hay diode bán dẫn

Mạch lật (flip-flop) RS (còn gọi là triger RS) đồng bộ là một mạch có khả năng lưu giữ các giá trị 0 hoặc 1 ở lối ra. Có thể dùng RS Flip-Flop làm một mạch lưu giữ tín hiệu vào R bằng cách chốt dữ liệu đó lại tại đầu ra Q. Các hãng chế tạo thực hiện mạch này bằng công nghệ cao, nên kích thước vô cùng nhỏ, có thể có hàng nhiều triệu phần tử nhớ trên một diện tích 1mm2. Các vi mạch nhớ thông thường được chế tạo với độ dài từ nhớ và số lượng từ nhớ cố định.

Hình mô tả:

- Dùng Flip/flop RS, hay D nhớ 1 bit, thích hợp để chế tạo bộ nhớ tĩnh tĩnh (Static).
- Các đầu vào/ra cần thiết của một ô nhớ, hay một chip nhớ, hay một module nhớ (ROM hay RAM): Địa chỉ (tập các địa chỉ áp vào), các tín hiệu ghi (WR/)/đọc(RD/), chọn chip/module CS (Chip Select) và đầu ra/vào dữ liệu.



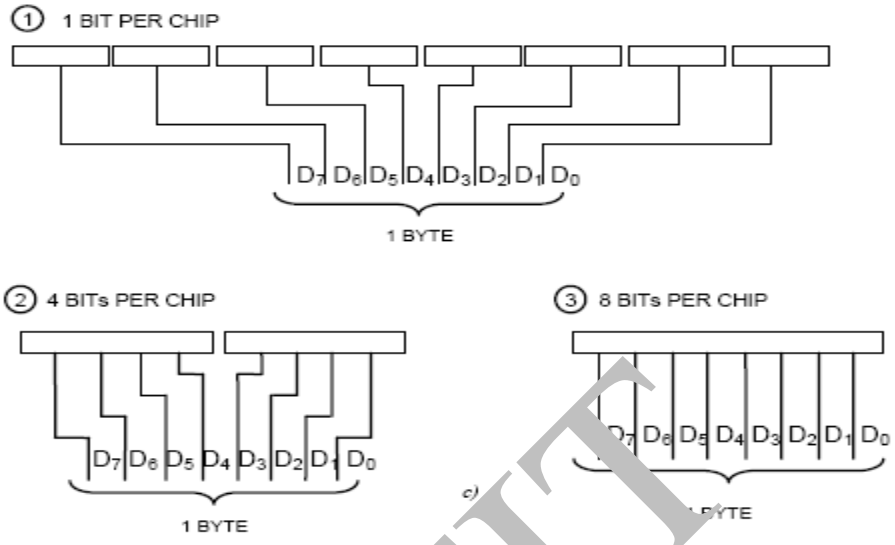
Hình 2.43 Mô hình đầu vào/ra của phần tử nhớ

Đơn vị nhớ

Đơn vị nhớ là các giá trị qui ước trong kỹ thuật máy tính, có thể có các loại như sau:

- Nhớ 1 bit,
- Nhớ vài bit : 4 bit hay 8 bit (còn gọi là 1 byte);
- Một từ (word) 16 bit
- Một từ đúp (double word) 32 bit
- Một từ dài 64 bit

Để tạo được một *từ nhớ của bộ nhớ*, tức là từ nhớ có độ dài (số bit trong một từ) chuẩn (theo chuẩn IBM là 8 bits), trong một số trường hợp nhất định cần phải tiến hành ghép các chip nhớ lại với nhau.
cho ta khái niệm về khả năng tạo một từ nhớ cơ bản (byte) khi từ nhớ của chip nhớ là 1bit, 2bits và 4 bits. Trong trường hợp độ dài từ nhớ của chip nhớ là 8 bits, việc liên kết là không cần thiết.



Hình 2.44 Cách tổ chức 1 đơn vị nhớ chuẩn (1 byte) từ các phần tử 1 bit, 4bit và 8 bit

Thông thường độ dài một đơn vị nhớ phụ thuộc vào loại CPU có khả năng xử lí, ví dụ CPU 8 bit như Intel 8085, Motorola 6800 xử lý tối đa 8 bit song song. Ngày nay các CPU đa năng trong PC xử lý 32 bit và 64 bit.

2.5.2 Phân loại bộ nhớ

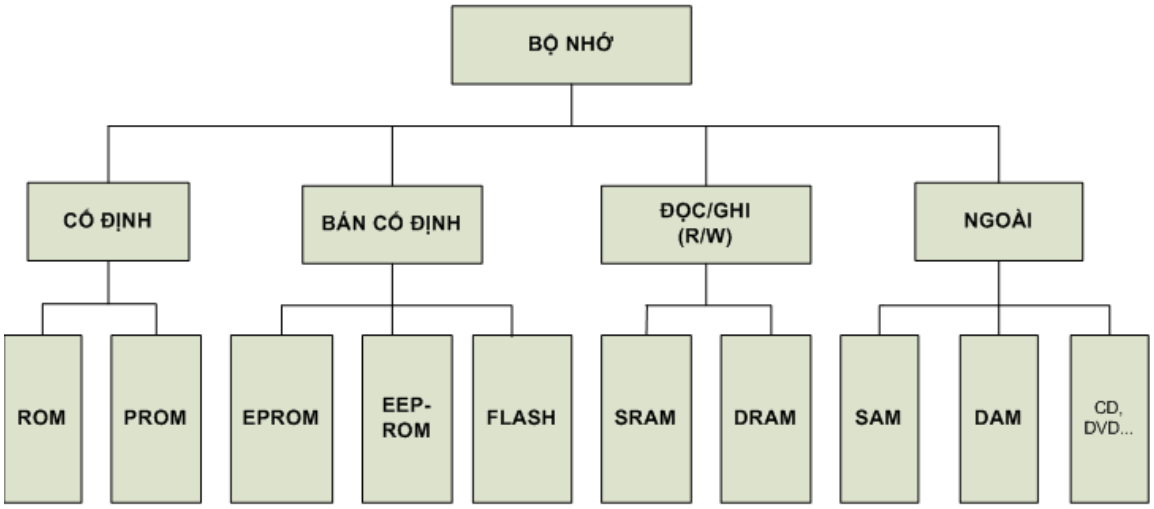
Nói chung, bộ nhớ được phân loại theo một vài thuộc tính. Sau đây là một số cách phân loại bộ nhớ:

Theo chức năng bộ nhớ được chia thành hai loại:

- Bộ nhớ trong (bộ nhớ chính)
- Bộ nhớ ngoài. (bộ nhớ phụ)

Dựa trên thời gian ghi và cách ghi bộ nhớ trong có thể chia thành:

- Bộ nhớ cố định
- Bộ nhớ bán cố định
- Bộ nhớ đọc/ ghi



Hình 2.45 Phân loại bộ nhớ

Bộ nhớ cố định nội dung

ROM (Read Only Memory):

Bộ nhớ có nội dung ghi sẵn một lần khi chế tạo được gọi là bộ nhớ cố định và được ký hiệu là ROM. Việc ghi được thực hiện bằng mặt nạ. Một phần tử nhớ trong ROM thường đơn giản hơn nhiều so với một mạch lật trong bộ nhớ đọc/ghi, vì trạng thái của nó cố định. Chương trình điều khiển của hầu hết các hệ vi tính được giữ trong ROM.

Bộ nhớ bán cố định nội dung

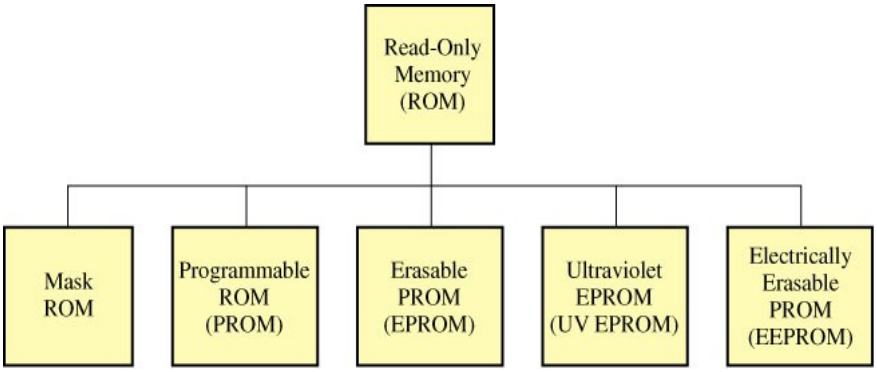
EPROM (Erasable Programmable Read Only Memory)

Đây là bộ nhớ xóa được bằng tia cực tím và ghi lại được. Số lần xóa và ghi lại không hạn chế. Chúng có thời gian ghi lớn hơn rất nhiều so với bộ nhớ đoc/ghi. Dưới tác dụng của tia cực tím tất cả các ô nhớ bị xóa cùng một lúc, trở về giá trị =1, khi xóa mạch nhớ phải được đưa ra khỏi hệ vi tính để xóa. Điều thuận tiện ở bộ nhớ bán cố định cũng như ROM, là bộ nhớ tuy bất biến khi dùng nhưng vẫn có thể ghi lại được. Một bộ nhớ bất biến là bộ nhớ có nội dung không bị mất khi nguồn điện bị ngắt.

EEPROM (Electrical Erasable Programmable Read Only Memory

EEPROM cũng tương tự EPROM, có thể ghi được nhiều lần, có nghĩa là ghi lại và sử dụng lại, nhưng EEPROM không xoá bằng tia cực tím mà bằng xung điện nên khi xóa vẫn để trong mạch điện.

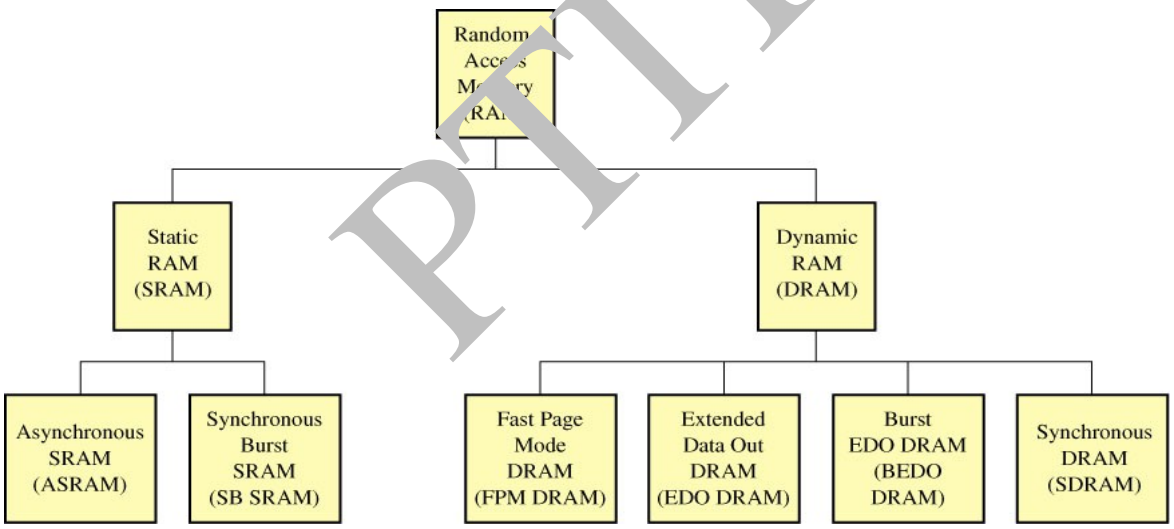
Các loại bộ nhớ ROM:



Hình 2.46 Các loại bộ nhớ ROM

Bộ nhớ đọc/ ghi- RAM

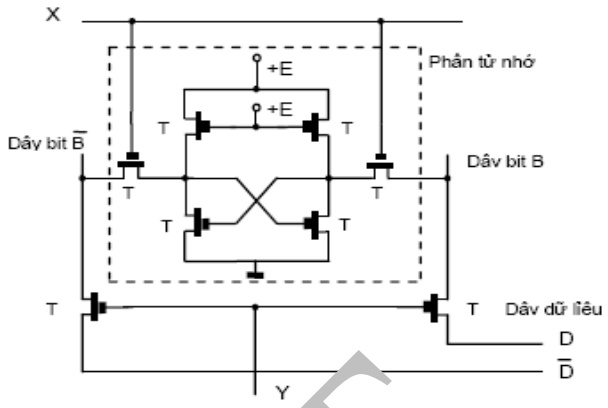
RAM (*Radom Access Memory*) kiểu truy nhập ngẫu nhiên, chỉ cần có địa chỉ áp đặt, vị trí nào không quan trọng, sẽ truy nhập được nội dung địa chỉ trỏ tới. Bộ nhớ có thể ghi và đọc nhiều lần, với thời gian ghi ngắn cỡ vài chục đến vài trăm nano giây. Trong các hệ vi tính, bộ nhớ đọc/ghi được sử dụng để cất giữ chương trình (hệ điều hành ứng dụng), kết quả trung gian.
Các loại RAM:



Hình 2.47 Các loại RAM

SRAM (*Static RAM*)- bộ nhớ tĩnh
SRAM là bộ nhớ đọc/ghi có nguyên lý hoạt động tĩnh: ghi vào một giá trị và tồn tại cho tới khi ghi giá trị mới. Khi mất điện sẽ mất nội dung. Bộ nhớ đọc/ghi tĩnh có các ô nhớ cấu tạo tương tự như mạch lật. SRAM không cần điều khiển phức tạp như DRAM, tốc độ nhanh hơn (tA = 10 ns), tin cậy hơn nhưng giá thành tính theo bit đắt hơn DRAM, được sử dụng làm bộ đệm cache trong các bộ vi xử lý hay máy vi tính. Bit nhớ được tạo bởi Flip-Flop.

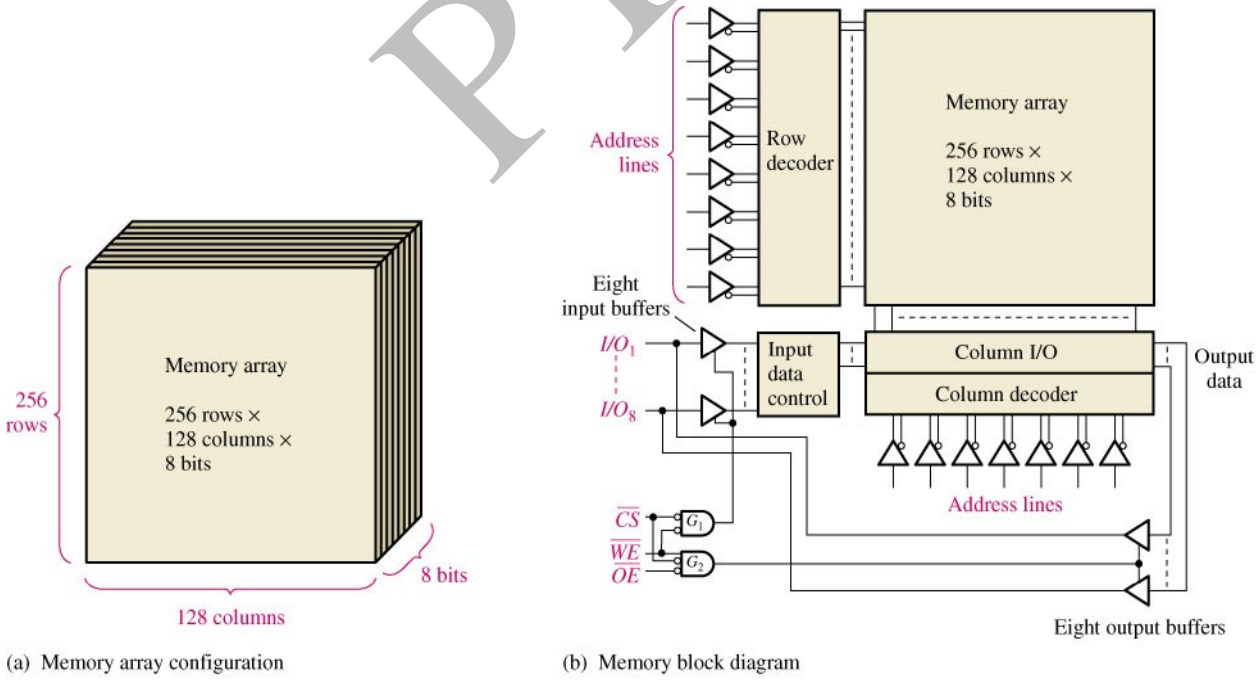
Ví dụ cho 1 bit SRAM: Trên hình bên là sơ đồ nguyên lí một một ô nhớ của RAM tĩnh, mạch nhớ sử dụng 6 transistor. RAM tĩnh được chế tạo theo công nghệ ECL (dùng trong CMOS và BiCMOS). Mỗi bit nhớ gồm có các cổng logic với 6 transistor MOS.



Hình 2.48 1 phần tử RAM tĩnh

Ví dụ tổ chức một Chip RAM 32Kx8 bao gồm:

- Ma trận các ô nhớ, hay byte nhớ (khối hàng x cột);
- Địa chỉ truy nhập vào ô nhớ hay byte nhớ (address lines);
- Bus và khuyếch đại bus dữ liệu đọc ra hay ghi vào các ô/byte nhớ (I/O lines)
- Tín hiệu điều khiển đọc/ghi (READ/WRITE) (CS/, WE/, OE/).



Hình 2.49 1 chip RAM 32K x 8 (32K byte)

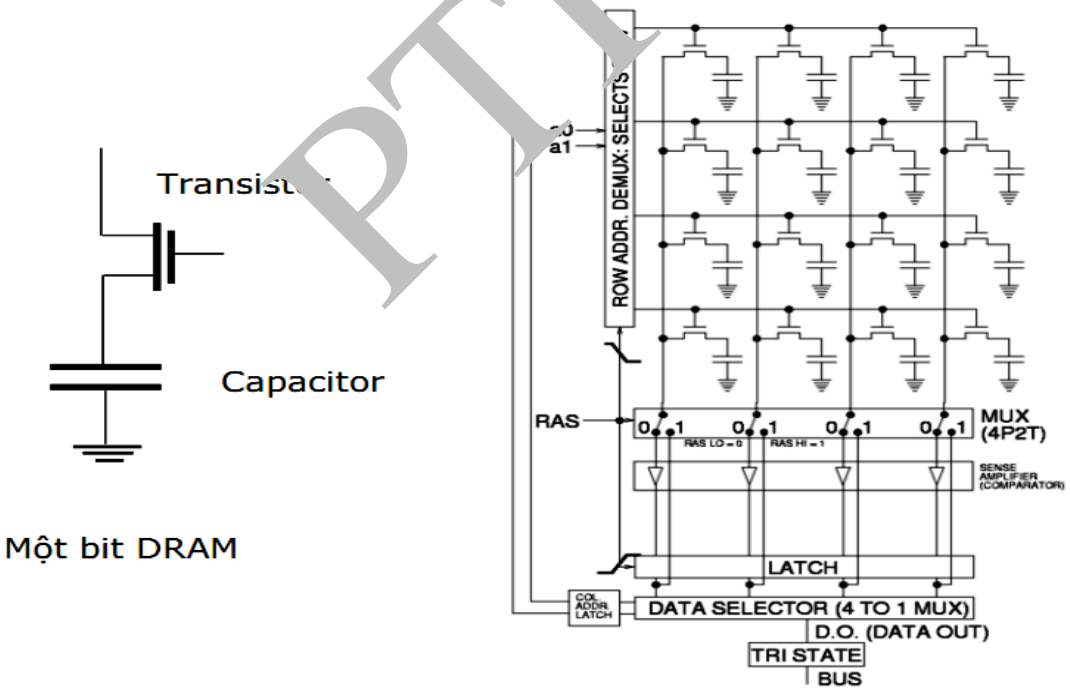
Lĩnh vực sử dụng SRAM:

Do giá thành cao, mật độ chế tạo (transistors/1 bit) cao, tốc độ nhanh, tin cậy, dễ thiết kế cho ứng dụng, nên

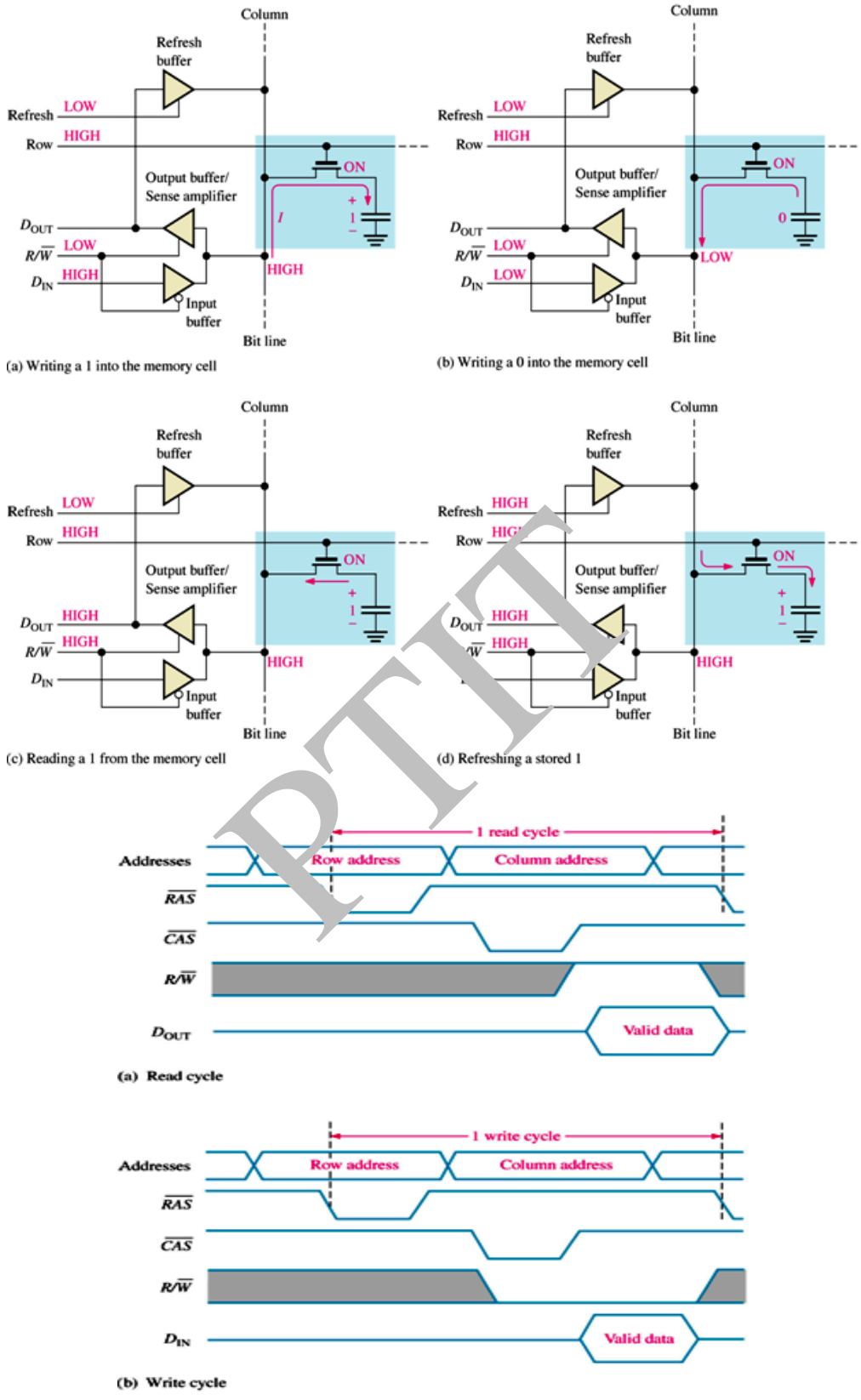
- Trong máy tính SRAM thường dùng như RAM cache.
- Trong các thiết bị điện tử số, đặc biệt trong các vi điều khiển của HTN , SRAM được sử dụng phổ biến bởi các tính năng nên trên. Hơn nữa các HTN không cần dung lượng bộ nhớ lớn như ở máy tính phổ thông.

Lĩnh vực sử dụng DRAM (*Dynamic RAM*) RAM động

Hình dưới là sơ đồ một ô nhớ RAM động được tạo từ 1 transistor và 1 tụ điện. RAM động dùng kỹ thuật MOS. Mỗi bit nhớ gồm một transistor và một tụ điện kí sinh. Việc ghi nhớ dữ liệu dựa vào việc duy trì điện tích nạp vào tụ điện và bị mất dần theo thời gian. Do vậy cần khôi phục lại dữ liệu, gọi là làm tươi (*refresh*). Qui trình làm tươi diễn ra liên tục, tự động theo chu kì (ví dụ khoản 2µs, hay khác tùy dung lượng và công nghệ chế tạo), vì thế gọi là RAM động. Việc làm tươi được thực hiện với tất cả các ô nhớ trong bộ nhớ. Công việc này được thực hiện tự động bởi một vi mạch làm tươi. Bộ nhớ DRAM chậm, dễ chế tạo mật độ cao/đơn vị diện tích, phù hợp cho chế tạo RAM, rẻ tiền hơn SRAM. Đổi lại điều khiển phức tạp.



Hình 2.50 Phần tử DRAM, 1 bit DRAM và ma trận DRAM



Hình 2.51 Các cách ghi/đọc/làm tươi của DRAM

Khi thiết kế DRAM dự vào đặc tả của CAS/ và RAS để thiết kế logic điều khiển

Các loại DRAM sử dụng trong máy vi tính PC

1. **SDRAM** (Viết tắt từ **S**ynchronous **D**ynamic **RAM**) được gọi là DRAM đồng bộ. SDRAM gồm 3 phân loại: SDR, DDR, và DDR2.

   o **SDR** SDRAM (**S**ingle **D**ata **R**ate SDRAM), thường được giới chuyên môn gọi tắt là "**SDR**". Có 168 chân. Được dùng trong các máy vi tính cũ, bus speed chạy cùng vận tốc với clock speed của memory chip, nay đã lỗi thời.

   o **DDR** SDRAM (**D**ouble **D**ata **R**ate SDRAM), thường được giới chuyên môn gọi tắt là "**DDR**". Có 184 chân. DDR SDRAM là cải tiến của bộ nhớ SDR với tốc độ truyền tải gấp đôi SDR nhờ vào việc truyền tải hai lần trong một chu kỳ bộ nhớ. Đã được thay thế bởi **DDR2**.

   o **DDR2** SDRAM (**D**ouble **D**ata **R**ate **2** SDRAM), Thường được giới chuyên môn gọi tắt là "**DDR2/DDR3**". Là thế hệ thứ hai của DDR với 240 chân, lợi thế lớn nhất của nó so với DDR là có bus speed cao gấp đôi clock speed.

2. **RDRAM** (Viết tắt từ **R**ambus **D**ynamic **RAM**), thường được giới chuyên môn gọi tắt là "**Rambus**". Đây là một loại DRAM được thiết kế kỹ thuật hoàn toàn mới so với kỹ thuật SDRAM. RDRAM hoạt động đồng bộ theo một hệ thống lặp và truyền dữ liệu theo một hướng. Một kênh bộ nhớ RDRAM có thể hỗ trợ đến 32 chip DRAM. Mỗi chip được ghép nối tuần tự trên một module gọi là RIMM (Rambus Inline Memory Module) nhưng việc truyền dữ liệu được thực hiện giữa các mạch điều khiển và từng chip riêng biệt chứ không truyền giữa các chip với nhau. Bus bộ nhớ RDRAM là đường dẫn liên tục đi qua các chip và module trên bus, mỗi module có các chân vào và ra trên các đầu đối diện. Do đó, nếu các khe cắm không chứa RIMM sẽ phải gắn một module liên tục để đảm bảo đường truyền được nối liền. Tốc độ Rambus đạt từ 400-800 MHz. Rambus tuy không nhanh hơn SDRAM là bao nhưng lại đắt hơn rất nhiều nên có rất ít người dùng. RDRAM phải cắm thành cặp và ở những khe trống phải cắm những thanh RAM giả (còn gọi là C-RIMM) cho đủ.

Bộ nhớ ngoài

SAM (Sequencial Access Memory**)**

SAM là bộ nhớ ngoài thâm nhập trình tự. Thời gian thâm nhập phụ thuộc vào vị trí của thông tin trên phương tiện mang tin. Ví dụ băng từ (Tap Cartridges) dùng như thiết bị lưu trữ dữ liệu của máy vi tính.

DAM (Direct Access Memory)

DAM cho phép thâm nhập tới bất cứ vùng dữ liệu nào cần. Thời gian thâm nhập phụ thuộc vào vị trí của thông tin trên phương tiên mang tin. Ví dụ thiết bị đĩa cứng, đĩa mềm thuộc loại thiết bị ngoại vi kiểu DAM.

CD-ROM (Compact Disk Read Only Memory)

CD-ROM là loại thiết bị đĩa dựa trên nguyên lý quang laser. Dung lượng của đĩa CD-ROM rất lớn có khả năng chứa tới 650 MByte.

WORM (Write Once Read Many)

WORM cho phép người sử dụng ghi thông tin một lần để đọc nhiều lần. Kiểu đĩa này rất thuận tiện cho việc sao chép phần mềm hay dự liệu khi triển khai các ứng dụng tin học.

DVD (Digital Versail Disk)

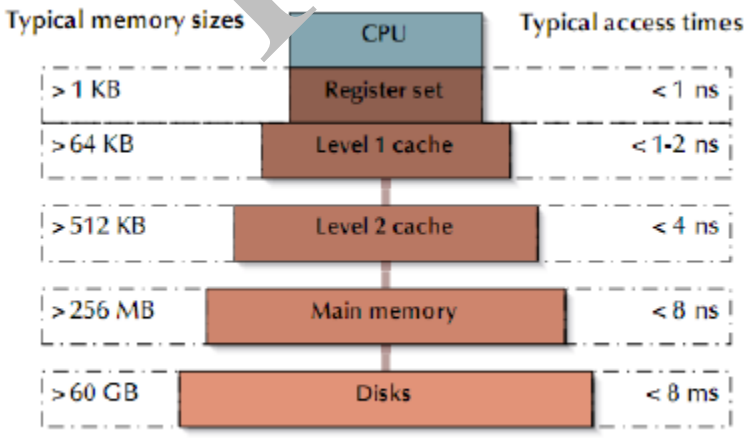
DVD cho phép ghi nhiều lớp thông tin trên một đĩa làm cho dung lượng của đĩa tăng lên đáng kể so với CD-ROM. Một đĩa DVD có thể lưu trữ lượng thông tin gấp 17 lần một đĩa CD-ROM.

Những mạch nhớ bán dẫn hiện nay có sức chứa giới hạn. Trong một vài ứng dụng, một bộ nhớ không những cần có dung lượng đủ lớn, mà còn cần được tổ chức để có số lượng từ và số lượng bit trong một từ như mong muốn. Nói chung, trong bộ nhớ có nhiều vi mạch nhớ được nối ghép lại để có độ dài từ và tổng số từ cần thiết. Những vi mạch nhớ bán dẫn được thiết kế sao cho có đầy đủ những chức năng của một bộ nhớ như:

- Một ma trận các phần tử nhớ, mỗi phần tử chứa một bit
- Phần mạch logic để giải mã địa chỉ cho ô nhớ
- Phần mạch lôgic cho phép đọc/ghi được nội dung của ô nhớ
- Bộ đệm vào, bộ kích ra
- Phần mạch mở rộng địa chỉ

2.5.3 Phân cấp bộ nhớ

Mô hình phân cấp bộ nhớ:



Hình 2.52 Phân cấp bộ nhớ

Phân cấp bộ nhớ được thể hiện trên :

Quan sát hệ thống nhớ từ CPU ra ngoài ta có các thành phần nhớ sau:

1. Các thanh ghi đa năng chứa một toán hạng hay kết quả trung gian, được điều khiển bằng phần cứng
2. Bộ nhớ đệm Cache chứa mảng lệnh và số liệu được sử dụng trong thời gian gần nhất, được điều khiển bằng phần cứng và chương trình. Bộ nhớ cache đặt giữa CPU và bộ nhớ chính.

- Cache trong CPU (cache L1), cache ngoài CPU (L2):

Bộ nhớ cache chứa một phần bản sao của bộ nhớ chính. Khi CPU thâm nhập vào dữ liệu nó đưa địa chỉ tới bộ điều khiển Cache, sau đó một trong hai quá trình sẽ xảy ra.

- Trúng (cache hit): nếu địa chỉ tìm thấy trong Cache
- Trượt (cache miss): nếu địa chỉ không có trong Cache

Khi trượt một khối nhớ từ bộ nhớ chính sẽ được đưa vào thay thế cho một đường (khối) của Cache. Đường nào sẽ được chọn để thay dựa trên hai nguyên lý sau:

- Cục bộ theo thời gian: nếu CPU thâm nhập vào một ô nhớ thì có xác suất cao nó sẽ thâm nhập ô nhớ đó trong tương lai.
- Cục bộ theo không gian: nếu CPU thâm nhập vào một ô nhớ thì có xác suất cao nó sẽ thâm nhập các lệnh và dữ liệu đặt sát các vị trí đó trong tương lai.

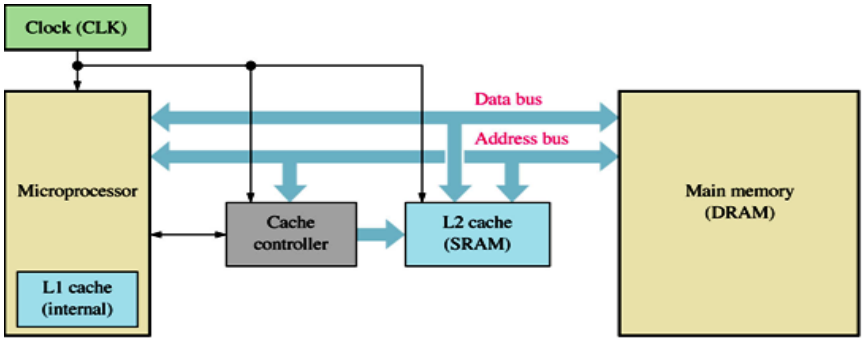
Trường hợp ghi vào Cache dữ liệu sẽ được ghi vào bộ nhớ chính, ta phân biệt hai trường hợp sau:

- Khi ghi vào Cache thì đồng thời ghi vào bộ nhớ chính, phương pháp này gọi là ghi xuyên (Write through)
- Khi ghi chỉ ghi vào bộ nhớ Cache, dữ liệu từ Cache sẽ được chuyển vào bộ nhớ chính tại một thời điểm thích hợp sau đó (ví dụ khi chuyển dữ liệu từ bộ nhớ chính ra thiết bị ngoại vi).

Việc ánh xạ giữa bộ nhớ Cache và bộ nhớ chính có thể tổ chức theo phương pháp khác nhau:

- Cache ánh xạ trực tiếp (Direct mapping cache)
- Cache ánh xạ liên kết toàn phần (Full associative mapping cache)
- Cache ánh xạ liên kết cụm (Set associative mapping cache)

Nội dung về bộ nhớ Cache được nghiên cứu kỹ hơn trong cấu trúc máy.



Hình 2.53 Mô hình hoạt động của RAM cache

3. Bộ nhớ trong (*Main memory*-bộ nhớ chính) chứa chương trình và số liệu đang thực hiện
4. Bộ nhớ ngoài lưu trữ chương trình và số liệu với khối lượng lớn. Nó cũng chứa phần nhớ ảo, khi máy tính chạy trong chế độ địa chỉ ảo.

Nếu đánh số phân cấp theo giá trị tăng dần từ trong CPU ra ngoài, ta có nhận xét sau:
- Thời gian thâm nhập của bộ nhớ có mức phân cấp càng thấp thì càng nhỏ

$t_{Ai} < t_{Ai+1}$

- Giá thành tính theo bit của bộ nhớ có mức phân cấp càng thấp thì càng cao

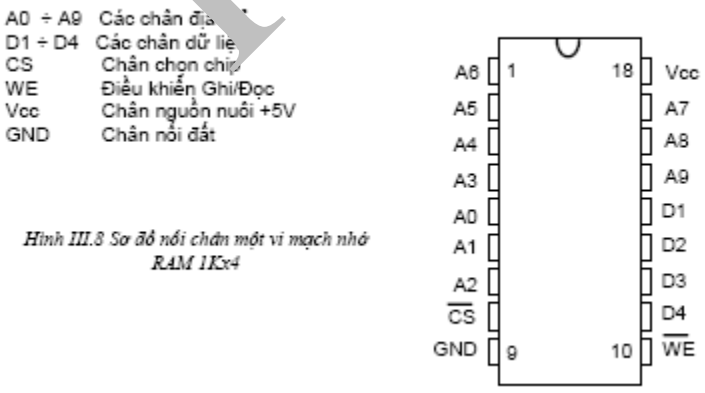
$c_i > c_{i+1}$

- Dung lương của bộ nhớ có mức phân cấp càng thấp thì càng nhỏ

$S_i < S_{i+1}$

2.5.4 Tổ chức bộ nhớ vật lý và thiết kế bộ nhớ

Tổ chức bộ nhớ cho một hệ máy vi tính phụ thuộc không chỉ vào một hệ cụ thể, mà còn phụ thuộc vào cách bố trí thuận lợi bên trong hệ thống. Trước hết, hãy làm quen với các khái niệm chip nhớ và từ nhớ để phân tích vấn đề tổ chức vật lý một bộ nhớ, sau đó mở rộng khái niệm tổ chức theo quan điểm của người lập trình (tổ chức logic). Các chip nhớ được sản xuất dưới nhiều kích cỡ và dung lượng nhớ, tổ chức từ nhớ khác nhau, phụ thuộc vào thể loại và công nghệ chế tạo. Chip nhớ là một vi mạch cụ thể, được bố trí các chân cơ bản như vẽ dưới. Các chân của một chip nhớ thông thường gồm các lối vào của BUS địa chỉ, lối dữ liệu, các chân điều khiển chọn chip, ghi/đọc và các chân nguồn.
Ví dụ một RAM tĩnh 1Kx4 (1K = 1024 "từ", mỗi từ có độ dài 4 bit:



Hình III.8 Sơ đồ nối chân một vi mạch nhớ RAM 1Kx4

Hình 2.54 Sơ đồ võ ngoài một vi mạch (chip) nhớ (pin-out)

Tuỳ theo từng chip, số lượng chân địa chỉ và số lượng chân dữ liệu có thể khác nhau phụ thuộc vào *độ dài từ nhớ* và *dung lượng* của chip nhớ. Độ dài từ nhớ của chip nhớ có thể là 1bit, 4 bits hoặc 8 bits, trong khi số chân địa chỉ có thể từ 10 trở lên tuỳ thuộc vào dung lượng của chip nhớ.

Trong trường hợp độ dài từ nhớ của chip là 1 bit, ta cần phải ghép song song 8 chip để tạo thành 1 byte, ghép song song 16 chip để tạo một từ word – 2 bytes).
Ngày nay, với công nghệ chip mật độ cao, thường 1 chip nhớ có thể có dung lượng rất lớn, ví dụ 256 x 1 Mb/Chip, 512 x 4 Mb/chip, 1 Gb/chip. Nhờ đó có thể tạo ra 1 thanh RAM có dung lượng lớn như PC hiện đang sử dụng.
Vấn đề tổ chức logic của bộ nhớ là một chương mục phức tạp vượt ngoài tầm của giáo trình, nên ở đây chỉ nêu lên một vài khái niệm đơn giản phù hợp cho người thiết kế.

Thiết kế vỉ nhớ là một việc rất quan trọng và rất cần thiết trong việc xây dựng một hệ vi tính. Các vỉ nhớ được thiết kế thông thường là EPROM, các loại vỉ nhớ RAM, từ các chip nhớ có sẵn. Thông thường, các chíp nhớ được chọn là những chip thông dụng trên thị trường, có các thông số kỹ thuật chủ yếu sau:

a. *Dung lượng nhớ của chip nhớ* tính theo đơn vị Kbyte
b. *Độ dài từ nhớ của chíp nhớ* tính theo số bits
c. *Một số thông số kỹ thuật khác như thời gian truy xuất, công suất tiêu tán của chip v.v...*Những thông số này không có ảnh hưởng lớn đến quá trình thiết kế và xây dựng vỉ nhớ.
d. Qui tắc tính toán khi thiết kế một vĩ (module) nhớ:
Xác định số chip nhớ, hoặc số chip liên thông để tạo được dung lượng nhớ theo yêu cầu. Trong trường hợp cụ thể của đề ra, cần 4 chip để tạo được dung lượng nhớ 32KB. Tính theo công thức:

M= Q/D, trong đó *Q* là dung lượng của vỉ nhớ.
D là dung lượng của mỗi chip
M là số chip nhớ hoặc số chip liên thông cần thiết.

Xác định số dây địa chỉ cơ sở (tức là số dây địa chỉ thấp được nối trực tiếp vào chip nhớ hoặc chip liên thông): Số dây địa chỉ *m* phụ thuộc vào dung lượng nhớ của chip nhớ hoặc chip liên thông theo biểu thức sau:

$2^m = D$, trong đó *D* là dung lượng của chip nhớ,
m là số dây địa chỉ cơ sở

Từ số chip hoặc số chip liên thông, xác định số dây địa chỉ cần thiết để tạo các dây chọn chip riêng biệt. Tính theo công thức:

$2^i = M$, trong đó *i* là số dây địa chỉ cần để giải mã *xác định các tín hiệu chọn chip (CSi) cho các chip nhớ hoặc chip liên thông.*
M là số lượng chip hoặc số lượng chip liên thông.

Các dây địa chỉ còn lại được sử dụng để tạo tín hiệu xác định vùng nhớ của vỉ nhớ trong không gian nhớ (được gán cho vỉ nhớ theo địa chỉ đầu của vỉ nhớ theo yêu cầu).

Ví dụ: Thiết kế RAM tĩnh (Static RAM)
Giả sử cần xây dựng một bộ nhớ kích thước 16Kbyte trên cơ sở các chíp SRAM loại 16Kx1bit. Băng nhớ SRAM 16Kbyte được xây dựng trên cơ sở 8 chip SRAM loại 16K x 1bit, để có được ô

nhớ có độ dài 8 bits (từ nhớ cơ bản). Để làm được điều này người ta sắp đặt 8 chip SRAM loại 16K x 1bit sao cho mỗi chip tại một vị trí xác định sẽ đảm nhiệm lưu trữ bit dữ liệu có trọng số tương ứng trong byte dữ liệu.

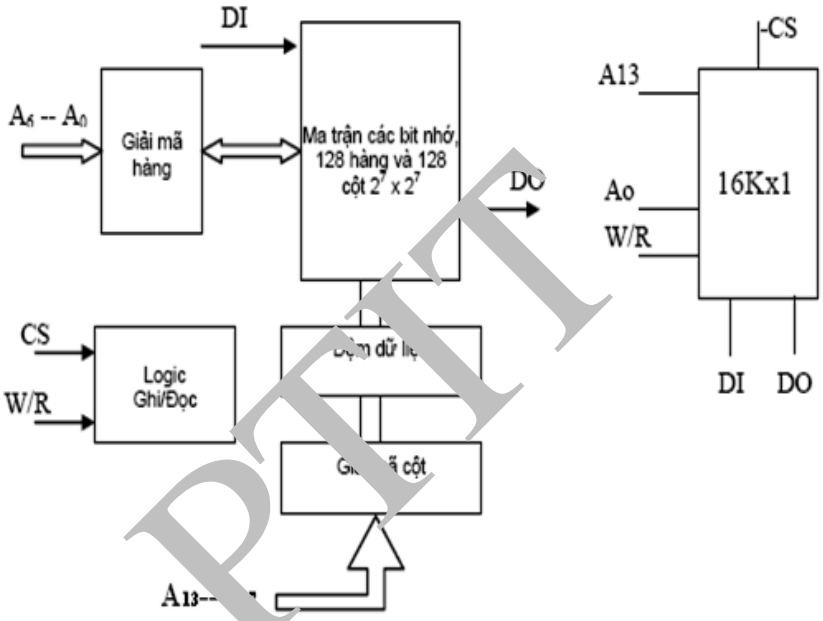
Các đường tín hiệu :

A13 - A0 BUS địa chỉ, đủ cho 16K (16383) địa chỉ

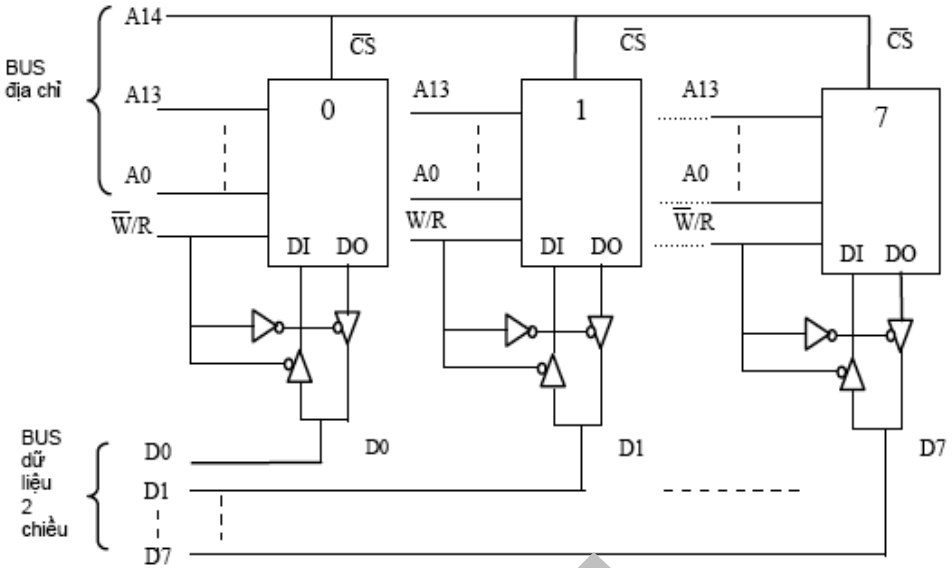
CS: Tín hiệu chọn chip. Nếu CS = 0 thì truy nhập được chip

W/R: Tín hiệu điều khiển ghi/đọc. W=0 (W/) điều khiển ghi, đồng thời điều khiển các vi mạch 3 trạng thái theo các chiều In/Out

Chọn băng nhớ cần thêm A14, A14=0 chọn băng thư nhất, (A14=1 chọn băng thứ 2).



Hình 2.55 Sơ đồ khối chức năng bên trong chip 16K x 1 bit

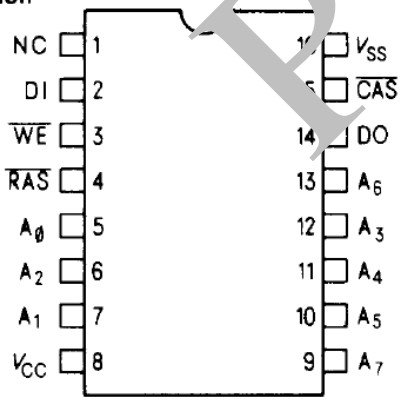


Hình 2.56 Sơ đồ thiết kế băng nhớ SRAM 16K x 8, với Chip 16Kx1

Ví dụ thiết kế RAM động (DRAM)

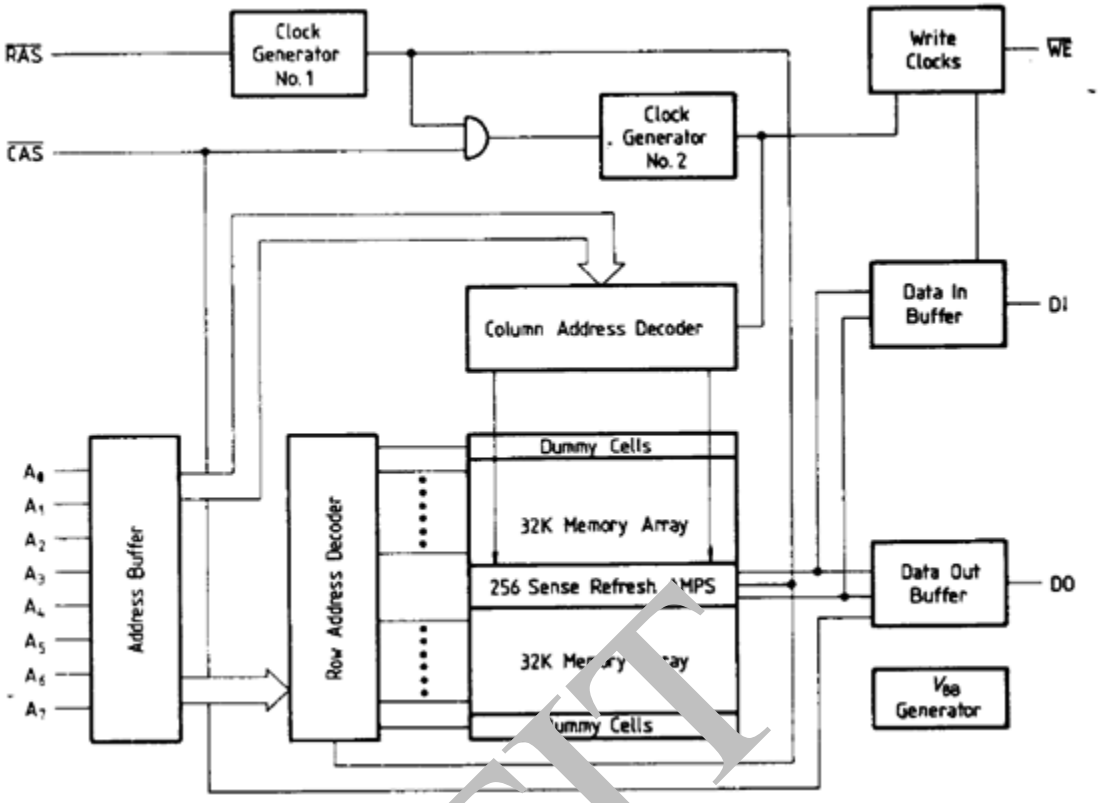
DRAM dùng phương pháp dồn kênh để nạp lần lượt (2 lần) địa chỉ hàng và địa chỉ cột vào đệm địa chỉ. Sơ đồ thời gian dưới đây cho thấy những lưu ý khi thiết kế với DRAM.

Pin Configuration



Pin Names

$A_0 - A_7$	Address Inputs
$\overline{CAS}$	Column Address Strobe
DI	Data In
NC	No Connection
DO	Data Out
$\overline{RAS}$	Row Address Strobe
$\overline{WE}$	Write Enable
V_{CC}	Power Supply (+5V)
V_{SS}	Ground (0V)



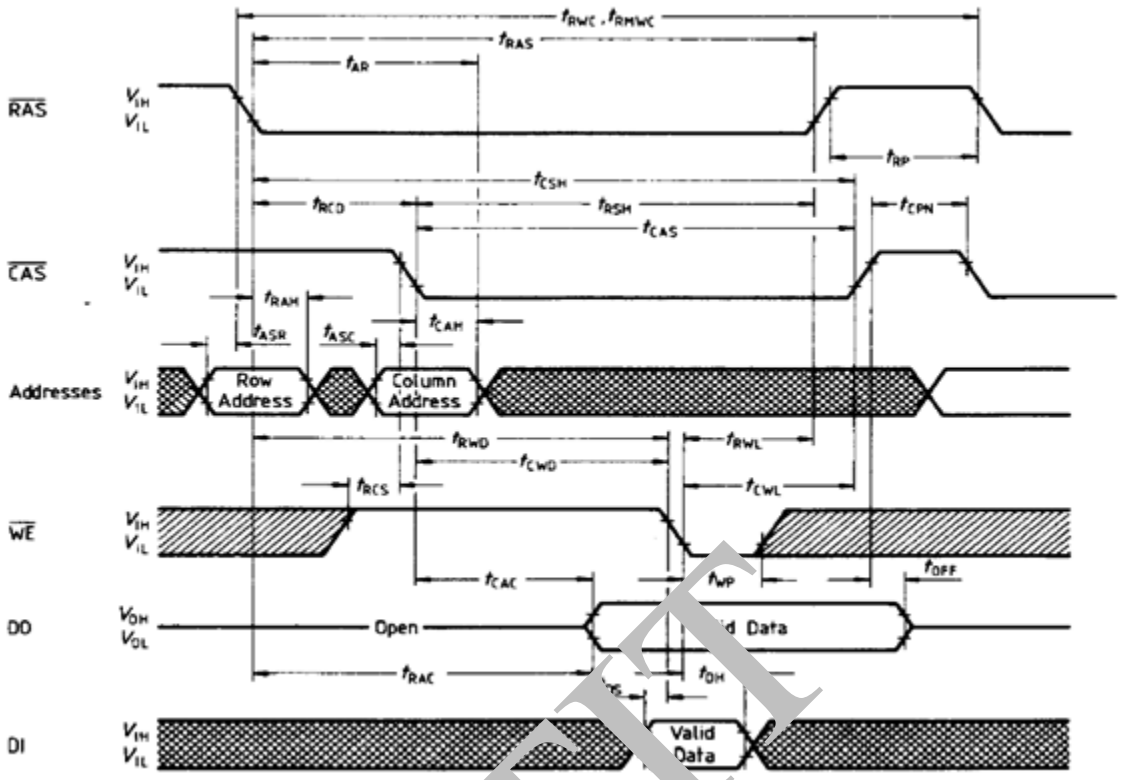
Hình 2.57 Sơ đồ khối chức năng của 1 chip DRAM thương mại 4164Kb

Tín hiệu điều khiển :

- RAS/: khi RAS (Row Address Strobe) tích cực thì địa chỉ hàng được nạp (chốt lại).
- CAS/: khi CAS (Column Address Strobe) tích cực thì địa chỉ cột được nạp (chốt lại).
- WE/: WE = 0 điều khiển ghi chip, WE = 1điều khiển đọc chip.

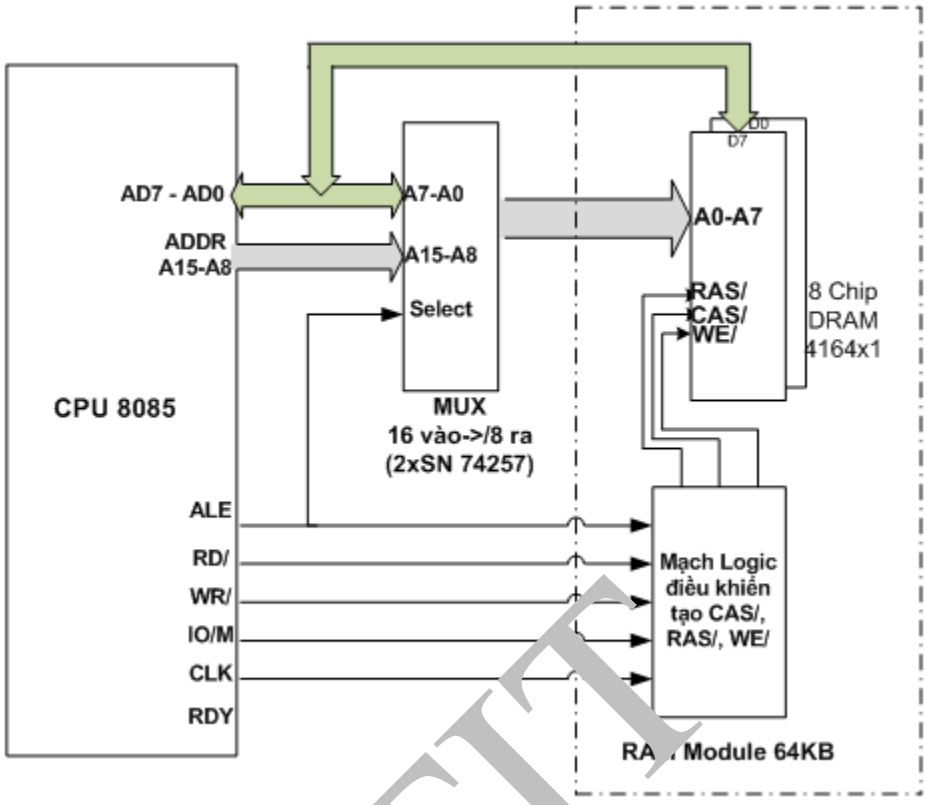
Việc xây dựng bộ nhớ từ các chip DRAM được thực hiện gần tương tự như với SRAM, tuy nhiên phải thiết kế một logic điều khiển để tạo các tín hiệu RAS/ và CAS/ bên ngoài, sau đó nối tới các chân RAS/ và CAS/ của chip.

Ví dụ lấy DRAM MK 4164 là DRAM 64K bit trong 1 Chip. Giả định sẽ thiết kế RAM cho CPU 8085 với RAM tối đa 64 KB, sẽ cần 8 Chip. Khi thiết kế cần tham khảo tài liệu đặc tả của Chip DRAM, như đã nói DRAM phức tạp hơn khi thiết kế.



Hình 2.58 Quan hệ các tín hiệu điều khiển DRAM 4164x1 thương mại.

DRAM 4164 có dung lượng 64 K bit/chip, được tổ chức kiểu ma trận gồm (256 hàng x 256 cột). Để chọn một bit trong ma trận cần có địa chỉ ở vị trí hàng và cột được hai bộ giải mã hàng và giải mã cột xác định. Đầu vào cho mỗi bộ giải mã cần 8 bit để có 256 đầu ra, vậy ta sẽ phối hợp các dây địa chỉ từ CPU 8085, bao gồm A7-A0 cho hàng và A15-A8 cho cột. CPU phát địa chỉ vào vi mạch dồn kênh SN 74257, đầu ra lần lược sẽ là A7-A0 chốt vào DRAM băng xung RAS/ A15-A8 bằng xung CAS/. Khi thiết kế Mạch logic điều khiển tạo RAS/ và CAS/ và WE cần sử dụng các tín hiệu điêu khiển tùa CPU phát ra. Để có được 64KB, cần 8 chip nói trên.



Hình 2.59 CPU 8088/8086 Module DRAM 64 KB toàn phần

d) Thiết kế ROM/EPROM

Trong tất cả các hệ máy vi tính, thành phần không thể thiếu là ROM/EROM/EPROM, với chức năng lưu các phần mềm không được mất đi khi mất điện. Các phần mềm này tùy vào lĩnh vực ứng dụng máy vi tính, có thể là :

- Nếu là máy tính vạn năng, như PC, sẽ là chương trình BIOS *(Basic Input Output system)* có chức năng lưu các *device driver*, các thông số khởi động các vi mạch điều khiển các thiết bị ghép nối vào PC trên bo mạch chủ hay qua các *slot* mở rộng, các *routine*, *subroutine*, … và phần *mã khởi động Hệ điều hành* từ thiết bị ngoài.
- Nếu là các thiết bị điện toán, ví dụ như HTN, … thì chứa phần mềm hệ thống điều khiển thiết bị và ứng dụng đặc thù mà thiết bị xử lý. Như đã đề cập các loại phần mềm ở đây rất đa dạng.

Ví dụ: Xây dựng module ROM có dung lượng 32KB, sử dụng Chip 2764 8K x 8 bit, địa chỉ đầu là 20000hex. Chương trình ứng dụng nạp vào module này.

Thiết kế một ROM, loại *2764 (8Kx8bit),* Vi mạch 2764 có thời gian truy xuất vào khoảng 250ns phù hợp với các bộ vi xử lý tốc độ cao như Intel 8MHz 8086-2 trong hệ thống này 2764 hoạt động không yêu cầu trạng thái "đợi" (Wait state). Vi mạch 2764 hoạt động trong chế độ dự

phòng (standby mode) cho phép giảm công suất tiêu tán mà không tăng thời gian truy cập . Dòng điện khi hoạt động là 150mA ,khi ở chế độ dự phòng là 35mA giảm 75%. Vi mạch 2764 được thiết kế chế tạo dựa trên công nghệ HMOS -E tốc độ cao, kênh N.

Mạch tổ hợp logic giải mã chọn địa chỉ vùng được thiết kế dựa vào thông số địa chỉ đầu của vùng nhớ, đó là các bit nhớ cao nhất, có giá trị cụ thể và không thay đổi trong toàn bộ quá trình truy xuất đến các vị trí nhớ trong vỉ nhớ.

Mạch tổ hợp logic giải mã tín hiệu chọn chip nhớ có đầu vào là các bit địa chỉ tiếp theo kể từ các bit địa chỉ độc lập (tức là các bit địa chỉ được nối trực tiếp vào các chân địa chỉ của chip nhớ). Số lượng bit địa chỉ là lối vào của mạch này phụ thuộc vào số lượng chip nhớ hoặc số lượng chip liên thông tạo nên từ nhớ cơ bản, tuân thủ công thức $2^i = M$.

Để truy nhập vào 8 K địa chỉ /chip cần 13 đường dịa chỉ, từ A12-A0

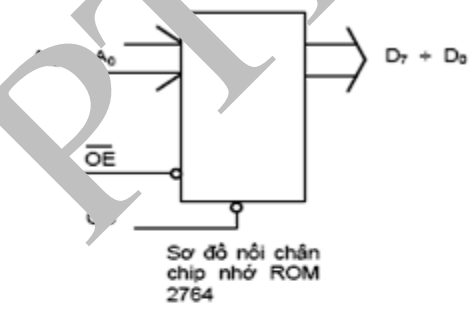
Để truy nhập vào mỗi chip trong 4 chip cần 2 dây địa chỉ: A14-A13

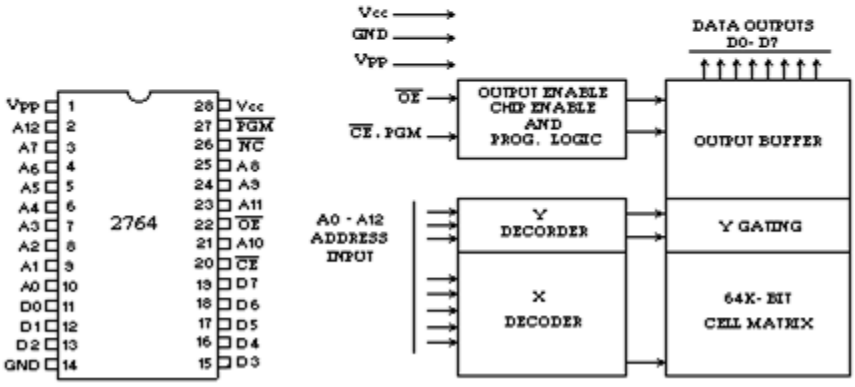
Để xác định vùng địa chỉ cần 5 đường địa chỉ: A14-A19 của 1MB đầu tiên

CPU x86 sử dụng 20 dây địa chỉ cho 1 MB đầu tiên: A19 – A0:

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	0	1	0	0	X	X	x	x	x	x	x	x	x	x	x	x	x	x	x

Giá trị =1 là cố định, giá trị =X thay đổi (0 hay 1)





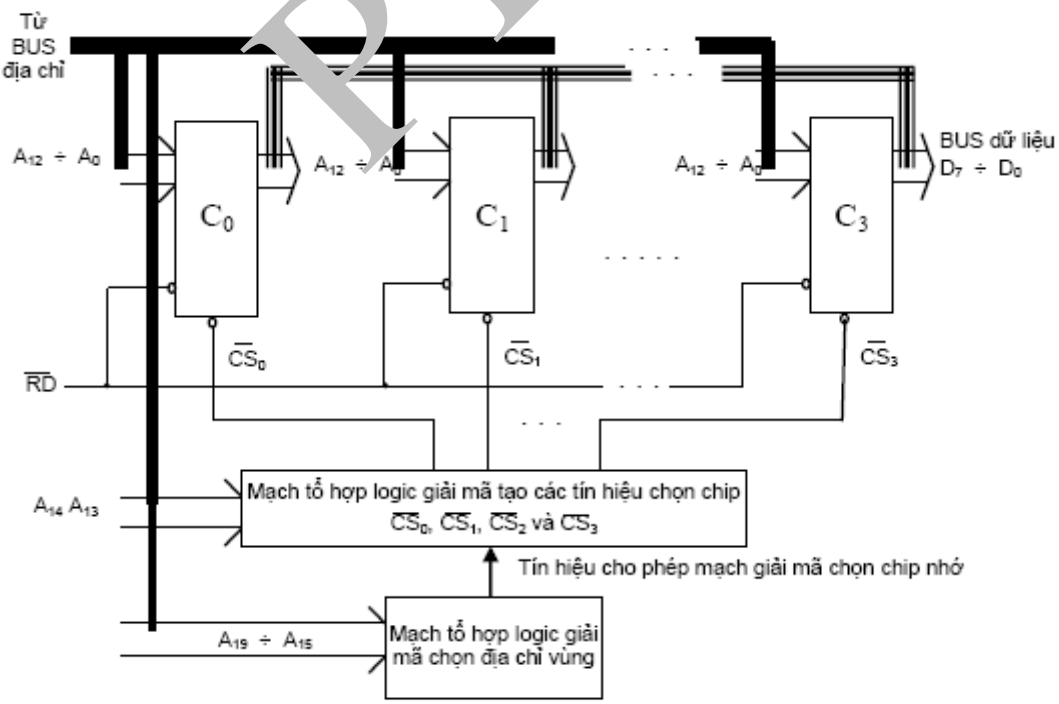
Hình 2.60 Chip ROM 2764 thương mại

Địa chỉ khi truy nhập module RAM này:

Chip #	Địa chỉ đầu	Địa chỉ cuối	Dung lượng
C0	20000	21FFF	8 K
C1	22000	23FFF	8 K
C2	24000	25FFF	8 K
C3	26000	27FFF	8 K

Giải mã thứ nhất có 5 đầu vào từ A19-A15 và 32 đâu ra chọn vùng địa chỉ. Với giá trị: 00100 , chọn vùng địa chỉ yêu cầu (20000-27FFF)hex = 32 K, sẽ có đầu ra cho phép giải mã thứ 2 hoạt động. Giải mã thứ hai dùng A13-A14 tạo ra 4 CS/ chọn 4 chip, trong khi A12-A0 chọn ô nhớ trong mỗi chip:

Chọn vùng A19.A18.A17.A16.A15	Chọn chip A14 A13	Chip được chọn theo A14 và A13	A12A11……. A0
0 0 1 0 0	0 0	C0(8K byte thứ 1)	0….
0 0 1 0 0	0 1	C1(8K byte thứ 2)	….
0 0 1 0 0	1 0	C2(8K byte thứ 3)	…..
0 0 1 0 0	1 1	C3(8K byte thứ 4)	. …

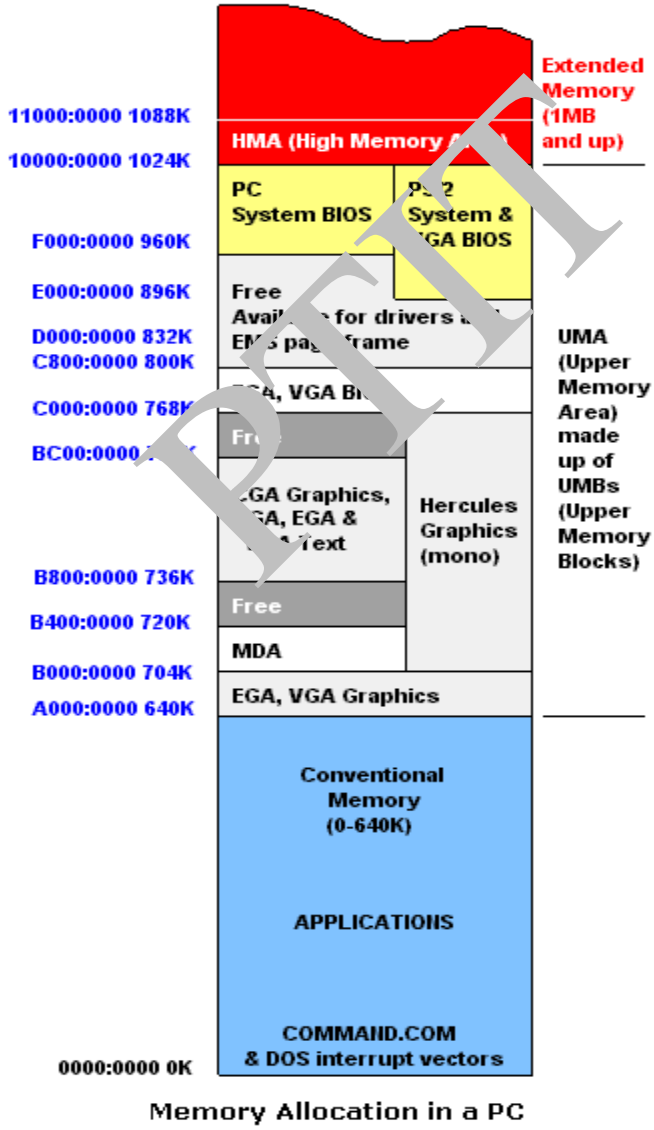


Hình 2.61 Sơ đồ thiết kế ROM 32KB từ 4 Chip 2764

Với các ý tưởng như trên có thể thiết kế các loại bộ nhớ cần thiết cho một HTN. Điều lưu ý chỉ là:

- ✓ Loại bộ nhớ nào (*ROM hay RAM*) sẽ dùng và dùng cho mục đích gì (*BIOS, HĐH hay Dữ liệu,Ứng dụng*).
- ✓ Không gian địa chỉ sẽ dùng từ đâu đến đâu trong toàn bộ không gian địa chị mà CPU cho phép.
- ✓ Cách truy nhập RAM của loại CPU sử dụng (tuyến tính, phân đoạn, phân trang, trỏ trực tiếp, trỏ gián tiếp qua các thanh ghi).

Ví dụ sau đây là phân hoạch địa chỉ bộ nhớ ở mega đầu tiên (0000:0000 – 10000:0000) trong PC chuẩn:



Hình 2.62 Ví dụ về cách phân bố bộ nhớ trong máy tính PC

Trong đó: F000:0000 đến F000:FFFF cho BIOS (64K cao nhất),
Địa chỉ sau RESET: CS:IP=FFFF:0000 (tức FFFF0) là lệnh đầu tiên thực hiện (JMP RESTART ở F000:E05B. Địa chỉ FFFF1 đến FFFFF dành riêng 16 byte cao nhất cho CPU.
Phần Extended (từ 10000:0000 trở lên) là ngoài MB đầu tiên với A20 kích hoạt !

PC-DOS and MS-DOS memory map

high memory			
segment	**offset**	**size**	**contents**
0000h	0000h	1024	interrupt vector table
	0400h	172	BIOS communication area
	04ACh	68	reserved by IBM
	04F0h	16	user communication area
	0500h	256	DOS communication area
	0600h	varies	*operating system:* IBMBIO.COM (DOS interface to the BIOS) IBMDOS.COM (DOS interrupt handlers and interrupt 21h service routines DOS buffers, control areas, and device drivers COMMAND.COM (resident portion), interrupt 22h, 23h, and 24h handlers, routine to reload transient portion of COMMAND.COM memory-resident utilities transient applications transient portion of COMMAND.COM
9000h	FFFFh		highest address of user memory
A000h	0000h	128K	reserved area of ROM
A000h	0000h		beginning of area used by EGA and PS/2 video systems
B000h	0000h	4000	monochrome video memory
	0800h	16K	color graphics adaptor (CGA) video memory
C000h	0000h	192K	ROM expansion and control area
	8000h		ROM for hard disk
D000h			ROM expansion and control area
E000h			ROM expansion and control area
F000h	0000h	16K	reserved ROM
	4000h	40K	base system ROM, ROM BASIC
	E000h	8K	ROM BIOS
low memory			

2.6 GHÉP NỐI VỚI THIẾT BỊ NGOẠI VI

Như các phần trên đã đề cập. Sau khi đã thiết kế phần trung tâm của HTN với CPU, RAM, ROM và vi mạch tạo các cổng đề giao tiếp với thiết bị ngoài, phần này đề cập tới các nguyên lí cơ bản của kỉ thuả ghép nối và cách tạo ra các module ghép nối hợp nhất trên bo mạch chính hoặc kéo dài qua BUS mở rộng.

2.6.1 Tổng quan

Trước khi đi vào chi tiết, cần biết một số khái niệm cơ bản của kỉ thuật ghép nối.

- **Các phương thức địa chỉ của CPU**:
  Các CPU thông thường có 2 phương thức sử dụng địa chỉ, đó là dùng địa chỉ để truy nhập vào bộ nhớ, thông qua các lệnh máy liên quan tới bộ nhớ, và một số rất ít lệnh đặc biệt để truy nhập vào các thiết bị (vi mạch khả trình) để gởi các mã điều khiển tới các thiết bị đó. Trong trường hợp này, CPU truy nhập vào thiết bị theo cổng (port).
- **Cổng (port)** được CPU gán cho một hay vài địa chỉ. CPU có thể " nhìn" cổng như bộ nhớ. Trong trường hợp này, người thiết kế phải qui hoạch không gian địa chỉ của bộ nhớ với một vùng địa chỉ dành riêng cho các thiết bị. Trường hợp thứ hai, CPU "nhìn" thiết bị qua cách gán địa chỉ danhf riêng cho cổng. Khi tiếp cận cổng, CPU có một vài lệnh đặc biệt để truy nhập. Địa chỉ dành cho cổng không nằm trong không gian địa chỉ cho bộ nhớ.
- **Cách địa chỉ hóa cổng**:
  - ✓ **Theo địa chỉ bộ nhớ** (*Memory mapped Imput/Output*). Qui hoạch bộ nhớ sẽ có một vùng địa chỉ dành cho thiết bị. Truy nhập cổng sử dụng tất cả lệnh qui chiếu vào bộ nhớ. Điểm yếu cơ bản của phương thức này là: không gian bộ nhớ bi co lại một phần, điều này đặc biệt bất lợi với các CPU có số địa chỉ cho bộ nhớ giới hạn, thời gian truy nhập cổng lâu hơn do các lệnh bộ nhớ cần nhiều CPU clock hơn.
    Ví dụ với CPU có 16 dây địa chỉ (CPU 8080/8085), không gian địa chỉ cho bộ nhớ tối đa là 65.535, hay 64KB, hãy xem phân hoạch sau đây để hình dung mô tả của *Memory mapped Imput/Output:*

Thiết bị	Phân bổ địa chỉ	Kích thước
RAM	0000 - 7FFF	RAM 32 KB
Cho các cổng sẽ dùng (I/O ports)	8000 - 80FF	256 bytes
Sound controller (I/O ports)	9000 - 90FF	256 bytes

Video controller/text-mapped display RAM	A000 - A7FF	2KB
ROM	C000 - FFFF	16 KB

✓ **Theo địa chỉ cho thiết bị** (*Imput/Output mapped Input/Output*). Phương thức này đơn giản và truy xuất nhanh. Các lệnh điển hình có cú pháp như sau:

Cho CPU 8080/8085: ***IN [địa_ chỉ_ port_in]***

OUT [địa_ chỉ_ port_out]

Khi thực hiện lệnh, [địa chỉ port] sẽ đặt trên cả AD7-AD0 và A15-A8 và đều giống như nhau (doublicated). 8080/8085 chỉ cho tối đa 256 cổng.

Với CPU x86, dùng 16 bit cho cổng, do đó số cổng sẽ rất lớn (65.535 cổng). Dữ liệu trao đổi có thể 8 bit/lần hay 16 bit /lần.

Lệnh đọc vào/ghi ra thường kết hợp với cặp đôi AX và DX (hoặc EAX và EDX). Trong đó ***src*** thường là DX chứa ***[địa_ chỉ_ port_in]*** hay ***[địa_ chỉ_ port_out]*** và ***des*** thường là AL hay AX chứa dữ liệu đọc vào hay ghi ra tương ứng cổng ghép nối. Trong chế độ bảo vệ (protected Mode hay Kernel Mode), người dùng bình thường không sử dụng được lệnh IN/OUT.

Lệnh thực thi: ***IN des, src***

OUT des, src

Ví dụ cổng chuẩn cho bo mạch và ngoại vi chuẩn sử dụng trong PC:

I/O address range **Device**

00 – 1f First DMA controller 8237 A-5

20 – 3f First Programmable Interrupt Controller, 8259A, Master

40 – 5f Programmable Interval Timer (System Timer), 8254

60 – 6f Keyboard, 8042

70 – 7f Real Time Clock, NMI mask

80 – 9f DMA Page Register, 74LS612

87 DMA Channel 0

83 DMA Channel 1

81 DMA Channel 2

82 DMA Channel 3

8b DMA Channel 5

89 DMA Channel 6

8a DMA Channel 7

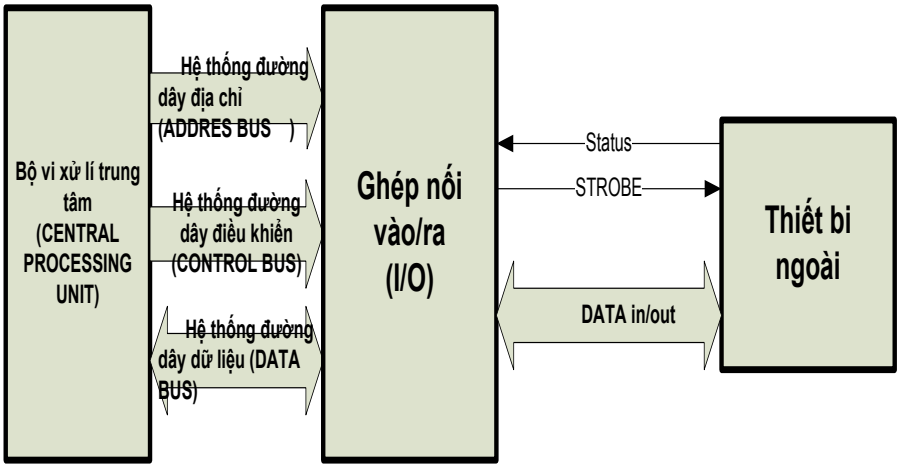
8f	Refresh
a0 – bf	Second Programmable Interrupt Controller, 8259A, Slave
c0 – df	Second DMA controller 8237 A-5
f0	Clear 80287 Busy
f1	Reset 80287
f8 – ff	Math coprocessor, 80287
f0 – f5	PCjr Disk Controller
f8 – ff	Reserved for future microprocessor extensions
100 – 10f	POS Programmable Option Select (PS/2)
110 – 1ef	System I/O channel
140 – 15f	Secondary SCSI host adapter
170 – 177	Secondary Parallel ATA Disk Controller
1f0 – 1f7	Primary Parallel ATA Hard Disk Controller
200 – 20f	Game port
210 – 217	Expansion Unit
220 – 233	Sound Blaster and most other sound cards
278 – 27f	LPT2 parallel port
280 – 29f	LCD on Wyse 2108 PC SM Elite default factory setting
2b0 – 2df	Alternate Enhanced Graphics Adapter (EGA) display control
2e8 – 2ef	COM4 serial port
2e1	GPIB/IEEE Adapter
2f8 – 2ff	COM2 serial port
2e2 – 2e3	Data acquisition
300 – 31f	Prototype Card
300 – 31f	Novell NE1000 compatible Ethernet network interfaces
300 – 31f	AMD Am7990 Ethernet network interface, irq=5.
320 – 323	ST-506 and compatible hard disk drive interface
330 – 331	MPU-401 UART on most sound cards
340 – 35f	Primary SCSI host adapter
370 – 377	Secondary floppy disk drive controller
378 – 37f	LPT1 parallel port
380 – 38c	Secondary Binary Synchronous Data Link Control (SDLC) adapter
388 – 389	AdLib Music Synthesizer Card
3a0 – 3a9	Primary Binary Synchronous Data Link Control (SDLC) adapter
3b0 – 3bb	Monochrome Display Adapter (MDA) display control
3bc – 3bf	MDA LPT parallel port
3c0 – 3cf	Enhanced Graphics Adapter (EGA) display control
3d0 – 3df	Color Graphics Adapter (CGA)

3e8 – 3ef	COM3 serial port
3f0 – 3f7	Primary floppy disk drive controller. Primary IDE controller (slave drive) (3F6–3F7h)
3f8 – 3ff	COM1 serial port
cf8 – cfc	PCI configuration space

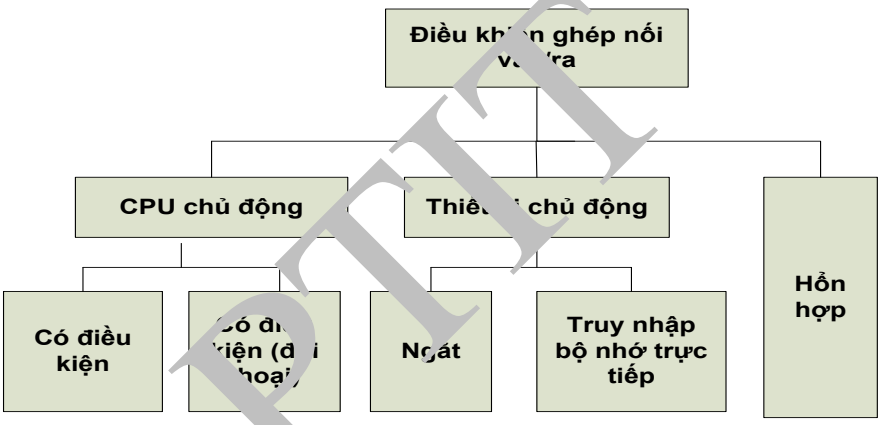
- **Ghi nhận trạng thái**

  Ghép nối là kỉ thuật *thích ứng* và *đồng bộ hoạt động trao đổi thông tin* giữa CPU và thiết bị ngoại vi, bao gồm thích hợp dạng tín hiệu (biến đổi, qui đổi mức năng lượng…) theo chuẩn TTL của máy tính. Quá trình này thuần túy là thiết kế điện tử (*số-số, analog-số-analog, số-analog*). Đồng bộ hoạt động có thể thực hiện theo cách diễn giải trình tự sự xuất hiện của các tín hiệu điều khiển theo thời gian, hay thực hiện kết hợp với phần mềm điều khiển. Việc khảo sát thông tin về trạng thái hoạt động của thiết bị là rất quan trọng. Thiết bị có thể *sẵn sàng* hoặc *không/chưa sẵn* sàng trao đổi dữ liệu. Để ghi nhận ta gọi đó là *thông tin trạng thái thiết bị* , được thể hiện bởi một loại tín hiệu mang thống tin trạng thái. Qui trình khảo sát trạng thái trước khi thực hiện trao đổi dữ liệu gọi là "*bắt tay*" ("*hand shaking*") và được thể hiện trong chương trình điều khiển thiết bị ("*device driver*").
- **Mô hình ghép nối vào/ra**

  Mô hình ghép nối bao gồm các thành phần sau đây:
  - Ghép nối thực hiện qua BUS hệ thống hay BUS mở rộng, gọi chung là BUS
  - Sử dụng các vi mạch thích ứng để ghép nối giữa BUS và thiết bị ngoài (latch SN74773, SN 74244/245, Flip/Flop Type D SN 7474, giải mã SN 74138, gate SN 7400, 3-state SN 74125, open collector SN 7403 …. . Các vi mạch này tạo thành "cổng" để dữ liệu trao đổi giữa CPU và thiết bị đi qua lại. CPU truy nhập vào các cổng để thực hiện đồng bộ qui trình trao đổi dữ liệu.
  - Các thiết bị ngoài, ghép vào bo mạch máy tính, cung cấp các tín hiệu cần thiết để thực hiện ghép nối kĩ thuật.



Hình 2.63 Mô hình kĩ thuật ghép nối



Hình 2.64 Các kiểu ghép nối

2.6.2 Ghép nối CPU chủ động

Vào/ra do CPU chủ động còn gọi là vào/ra điều khiển bằng chương trình được chia thành hai nhóm:

Vào/ra số liệu bằng chương trình không điều kiện

Vào/ra số liệu bằng chương trình có điều kiện (*handshaking*-đối thoại)

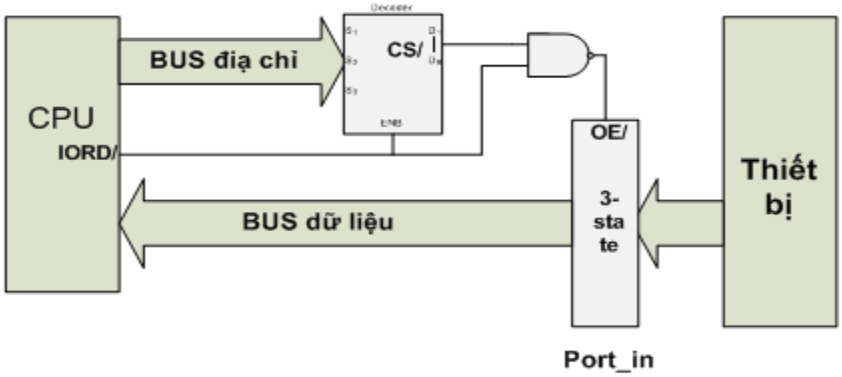
a) Vào/ra số liệu không điều kiện, điều khiển bằng chương trình

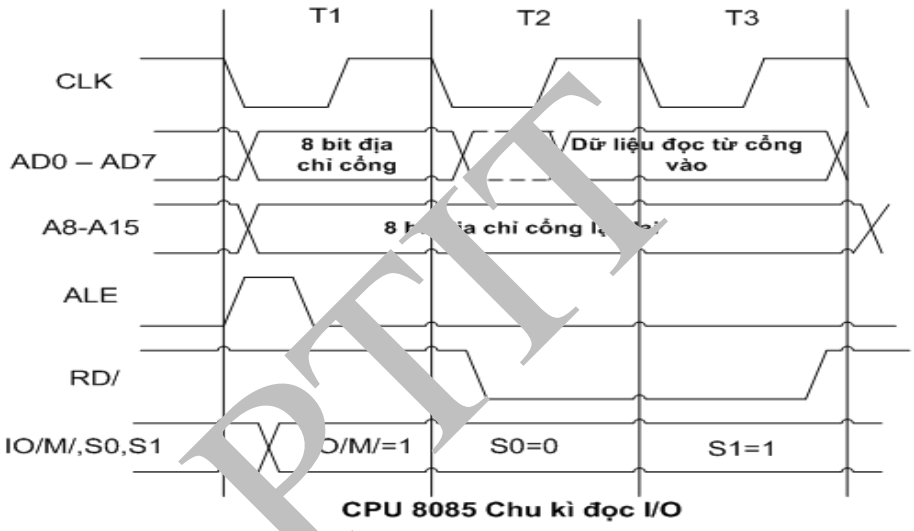
Vào/ra số liệu không điều kiện có các đặc điểm như:

- CPU chuyển số liệu thông qua chương trình
- CPU giả thiết TB vào/ra luôn sẵn sàng chuyển số liệu
- Dữ liệu đọc vào CPU qua cổng sau đó lưu lại o bộ nhớ cho các xử lý tiếp theo.
- Việc chuyển số liệu được thực hiện giữa các thanh ghi của CPU (ACC) và thanh ghi (cổng ghép nối) của TB vào/ra, sau đó lưu lại o bộ nhớ cho các xử lý tiếp theo.

Bộ nhớ ⇔ [CPU_ACC] ⇔ [Cổng] ⇔ [Thiết bị]

Đọc vào:





Hình 2.65 Đọc dữ liệu vào: Dữ liệu_từ thiết bị vào ACC sau đó vào RAM

Ví dụ ứng dụng: Khi kiểm soát nhiệt độ, thiết bị nhiệt luôn có số liệu để đo và xử lý. Trong trường hợp này ghép nối với thiết bị nhiệt trở nên đơn giản.

Tương tự cho mô hình đưa dữ liệu ra thiết bị.

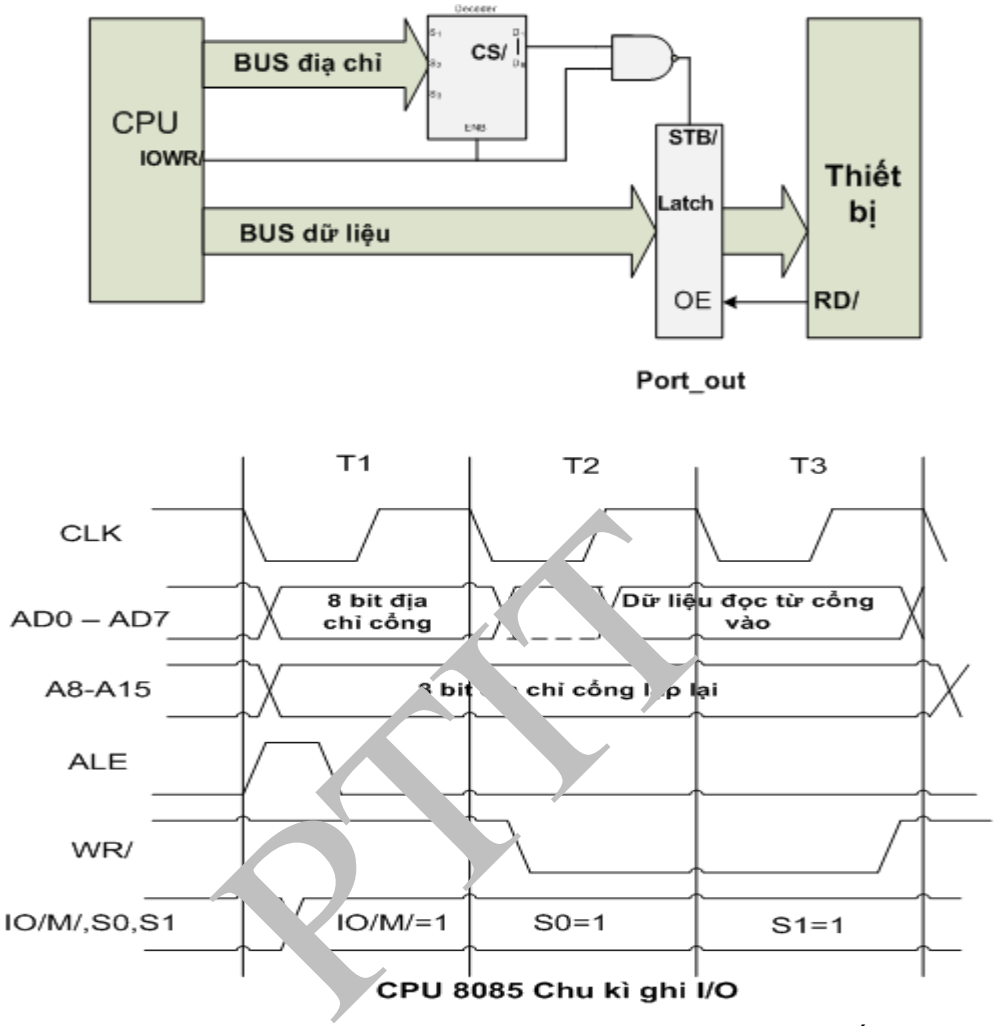
Lệnh thực hiện: ***IN [port_in]*** *hoặc* ***IN AL, [địa_ chỉ_ port_in]***

Các bước thực hiện:

- CPU đưa ra BUS địa chỉ địa chỉ cổng port_in cho giải mã, tạo CS/ mở port_in.
- CPU đưa ra BUS đ/k tín hiệu IORD
- Số liệu từ vi mạch 3-state chuyển vào BUS dữ liệu và dưới tác động của tín hiệu IORD/ và được đưa vào ACC của CPU qua port_in 3 trạng thái.
- Thực hiện chuyển ACC vào RAM.

Đưa ra:

Lệnh thực hiện: ***OUT [port_out]*** *hay* ***OUT [port_out], AL***



Hình 2.66 Đưa dữ liệu từ RAM vào ACC sau đó ACC ra thiết bị

Các bước thực hiện:

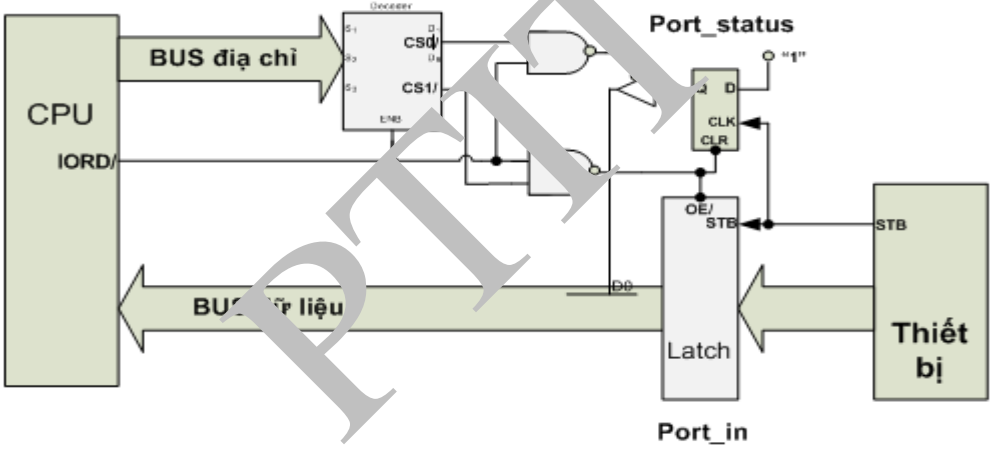
- CPU đưa ra BUS địa chỉ địa chỉ cổng port_out cho giải mã, tạo CS/ mở port_out.
- CPU đưa dữ liệu cần ghi ra BUS dữ liệu
- CPU đưa tín hiệu IOWR/ để chốt dữ liệu, dưới tác động của tín hiệu IOWR/ số liệu được ghi vào thanh ghi chốt, từ đó thiết bị nhận dữ liệu với tín hiệu RD/ của thiết bị.

Ghi nhận căn bản ở đây là CPU ở vị trí trung gian của qui trình nhận hay gởi dữ liệu. Chi phí một cổng cho mỗi cấu hình. Mỗi lần trao đổi 1 byte. Cơ chế này có hạn chế vì CPU ra lệnh đọc (IORD) hay ghi (IOWR) mà không kiểm tra xem thiết bị vào/ra có sẵn sàng gửi hay nhận số liệu hay chưa do đó khi ứng dụng cần phân tích hoạt động của thiết bị đẻ có giải pháp ghép nối đúng mục đích.

b) ***Vào/ra số liệu có điều kiện, điều khiển bằng chương trình***

Khi ghép nối với các thiết bị mà thiết bị đó cần có thời gian để hoàn thành một xử lý (tác vụ), trong qúa trình ghép nối, CPU cần có ghi nhận trạng thái sẵn sàng của thiết bị để đánh giá và ra quyết định tiếp theo.Quá trình kiểm tra trạng thái đôi khi phải thực hiện một vài lần trước khi quyết định tiếp tục hay từ bỏ trao đổi dữ liệu với thiết bị. Tại sao ? Nếu thiết bị làm việc tốt, mọi chuyện trôi chảy, nhưng nếu thiết bị hỏng hóc, chương trình điều khiển không thể cứ chờ thiết bị săn sàng mãi được, chu trình phải kết thúc. Qui tắc này gọi là đối thoại, hay móc nối có điều kiện là vậy. Về mặt thực hiện, ở đây cần ít nhất 2 cổng: 1 cổng để đọc trạng thái thiết bị, cổng kia để trao đổi dữ liệu vào hay ra. Trạng thái thông thường ở mức dộ đơn giản chỉ cần 1 bit để thể hiện, ví dụ STATUS=1, thiết bị sẵn sàng , và ngược lại. Tuy nhiên có nhiều trường hợp trạng thái có thể là tập hợp của vài bit. Ví dụ khi ghép với cổng truyền thông dị bộ (UART), có đến 5 bit phản ánh trạng thái của UART. Các bit trạng thái được nối vào BUS dữ liệu qua các mạch 3-trạng thái và được điều khiển bằng lệnh.

Đọc vào có đối thoại:

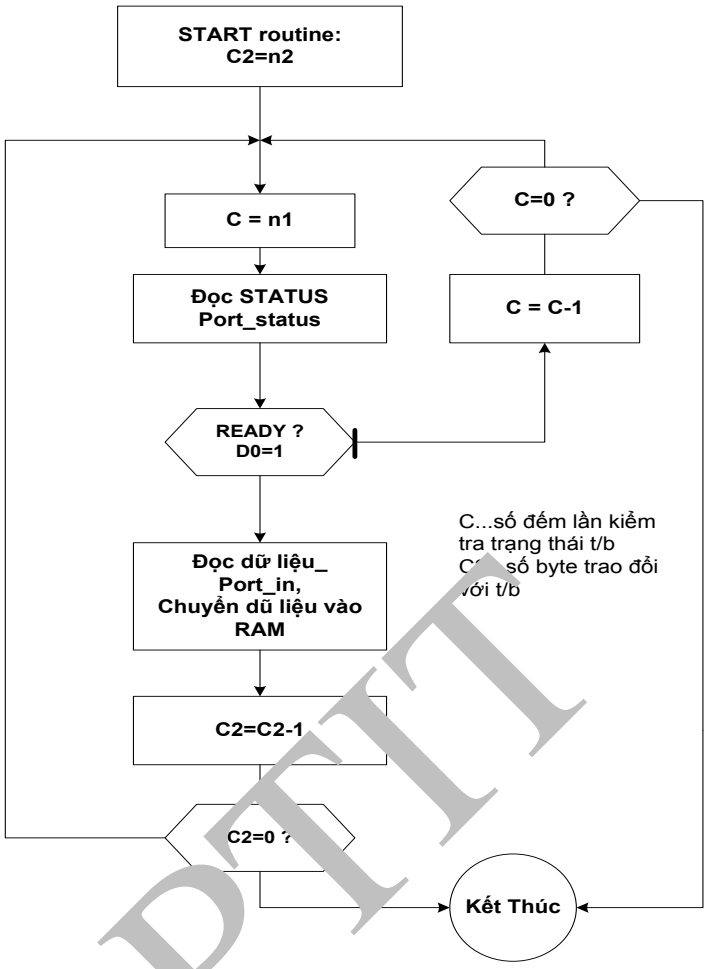


Hình 2.67 Trao đổi dữ liệu đọc vào có điều kiện

- Cần CS0/ để đọc trạng thái READY của thiết bị qua port_status, giả định nối vào D0 của BUS dữ liệu.
- Cần CS1/ để đọc dữ liệu qua port_in, hợp thành từ Flip/flop, cổng 3-state, nối vado bit D0 của BUS dữ liệu.

  Các bước thực hiện:
  - CPU đưa địa chỉ port_status ra BUS địa chỉ, vào giãi mã, tạo CS0, đọc STATUS, giá trị ở bit READY tại D0. (***IN port_status***)
  - CPU kiểm tra giá trị của READY.
  - Nếu READY=0, quay lại đọc STATUS
    NếuREADY=1, CPU đọc dữ liệu vào ACC (***IN port_in***)
  - Thực hiện lệnh cất dữ liệu vào RAM

- Lưu đồ điều khiển:



Hình 2.68 Lưu đồ điều khiển đọc dữ liệu và có điều kiện

BÀI TẬP:

1.Đưa ra có đối thoại:

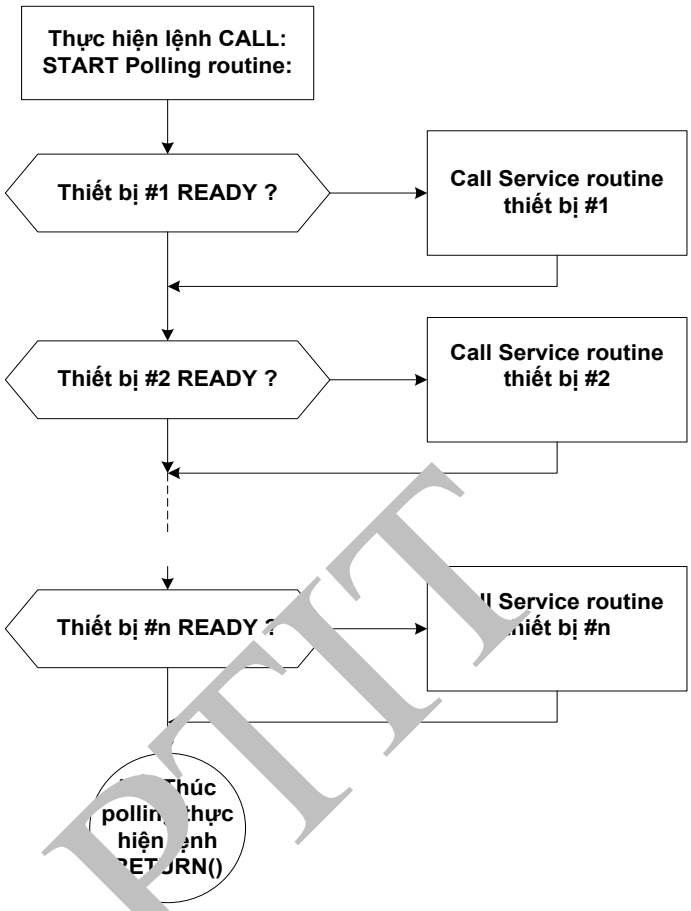
Thiết kế mạch ghép nốiđưa dữ liệu ra;

Viết chương trình điều khiển quá trình đưa dữ liệu ra.

c) Quay vòng (polling)

Trong phương pháp vào/ra có đối thoại, nếu thiết bị vào/ra chưa sẵn sàng nhận hay gửi số liệu thì CPU phải chờ cho thiết bị sẵn sàng. Nói cách khác phương pháp này gây lãng phí thời gian của CPU. Khi hệ thống có nhiều thiết bị ghép vào, cần có chiến thuật để giảm tối đa thời gian chờ trên một thiết bị. Cách đơn giản là dùng phương pháp quay vòng để hỏi trạng thái các thiết bị vào/ra. Về nguyên tức chỉ hỏi 1 thiết bị một lần, nếu thành công, thực hiện trao đổi dữ liệu, còn

không chuyển sang thiết bị khác. Chương trình sẽ tổ chức với những qui tắ tối ưu nào đó, ví dụ như bắt đầu với thiết bị có nức ưu tiên cao nhất và giảm dần.



Hình 2.69 Lưu đồ điều khiển đọc dữ liệu kiểu quay vòng

2.6.3 Ghép nối I/O chủ động

a) Ngắt (Interrupts)

Những hạn chế của phương pháp vào/ra bằng chương trình được khắc phục bằng phương pháp vào/ra bằng ngắt. Trong phương pháp này thiết bị vào/ra chủ động khởi động quá trình vào/ra số liệu. Khi sử dụng với cơ chế ngắt, phải quán triệt ý tưởng về ngắt khi gán cho thiết bị, đó là thiết bị sẽ có nhu cầu trao đổi dữ liệu, tuy nhiên thời điểm nhu cầu đó xuất hiện thì không thể biết trước được. Đó chính là ưu điểm vượt trội mà ngắt mang lại: Thiết bị chủ động vào bất cứ lúc nào, CPU không bị ràng buộc với thiết bị, do đó chi phí thời gian như phương pháp CPU chủ động giảm đi rất nhiều. Ngắt còn mang một ý nghĩa khác đó là tính tức thời nếu coi ngắt là biểu hiện của một sự kiện. Trong các HNT nhúng với xử lý theo thời gian thực, đây chính là điểm chủ yếu. Có hai hình thức ngắt, thông thường với thiết bị, ta nói đến ngắt cứng, tức là ngắt từ một tín hiệu phát sinh từ thiết bị. Ngoài ra còn có khái niệm ngắt mềm, sử dụng một nguyên lí mà hệ

điều hành hỗ trợ thông qua lệnh máy tính, giúp tạo ra sự chuyển xử lý (chương trình) tạm thời đến một xử lý đột xuất. Xét về xử lý thuần túy, ngắt cứng hay ngắt mềm đều có tác dụng như nhau.

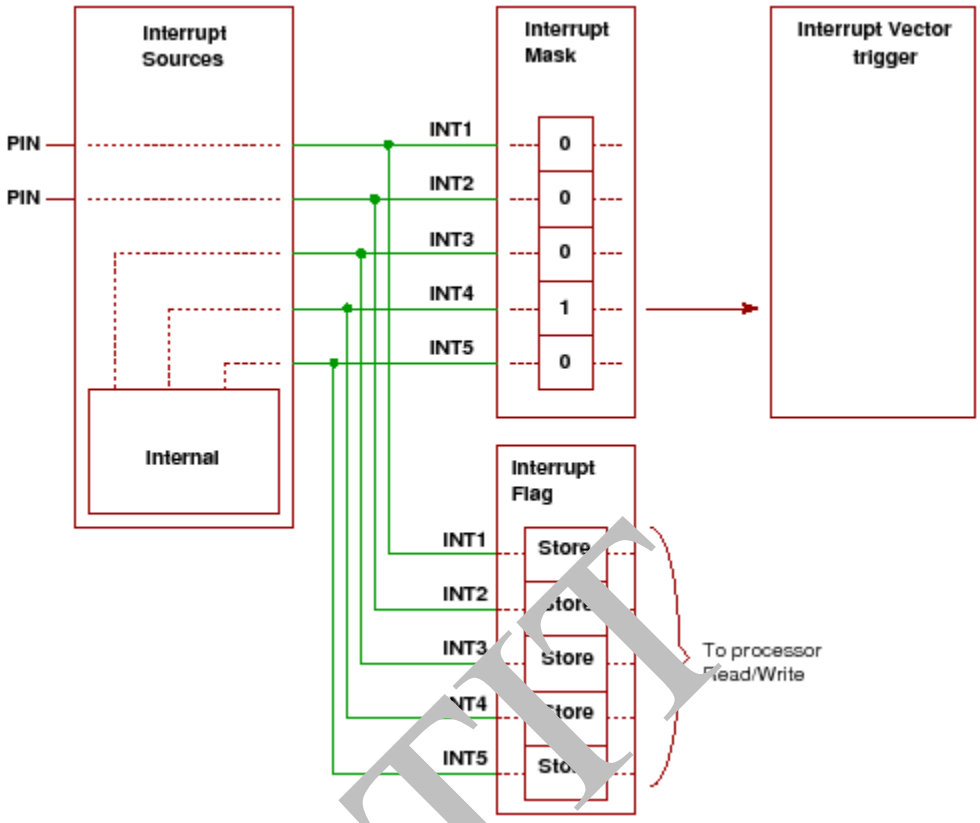
Ngắt cứng hay quay vòng ?

Lợi thế của ngắt cứng là thời gian sử dụng CPU rất hiệu quả, CPU phản ứng tức thì khi có ngắt, trong khi dùng hỏi đáp quay vòng (polling) chi phi ngiều thời gian của CPU. Ngắt cho nhiều giải pháp phản ứng của CPU với các sự kiện bên ngoài. Tuy nhiên, tạo và xử lí ngắt phức tạp hơn và, trong khi viết trình cho polling đơn giản hơn. Một trong các ứng dụng quan trọng của HTN là chế độ standby và đánh thức bằng wake-from-sleep interrupt.Another benefit of using interrupts is that in some processors you can use a wake-from-sleep interrupt, đặc biệt khi hệ nuôi bằng pin.

Các khái niệm với ngắt:

ISR	Interrupt Service Routine - Dịch vụ ngắt: mã xử lí cho ngắt.
Interrupt vector	Địa chỉ của ISR trong bộ nhớ hệ thống.
Interrupt mask	Mã lệnh kiểm soát cho phép hay cấm một ngắt hoạt động
NMI	Ngắt không thể kiểm soát được, luôn hoạt động.
Asynchronous event	Sự kiện có thể xảy ra bất kì thời điiểmm nào, có tính ngẫu nhiên.
Context switching	Qui trình bảo vệ / khôi phục nội dung các thanh ghi của CPU, hay dữ liệu trước / sau khi xử lí ngắt. Đoạn code này nằng trong code của ISR.

Một hệ ngắt cơ bản:



Hình 2.70: Mô hình hoạt động của ngắt

Nguồn phát sinh ngắt (Input source)

Là các tín hiệu ngắt từ các thiết bị trong hay ngoài bo mạch, ví dụ từ ADC/DAC, từ UART , từ các cổng ghép nối, DMAC … các tín hiệu có thể kích hoạt dạng mức (level) hay sườn lên hay xuống của tín hiệu ngắt (edge). CPU phản ứng với tín hiệu qua lập trình.

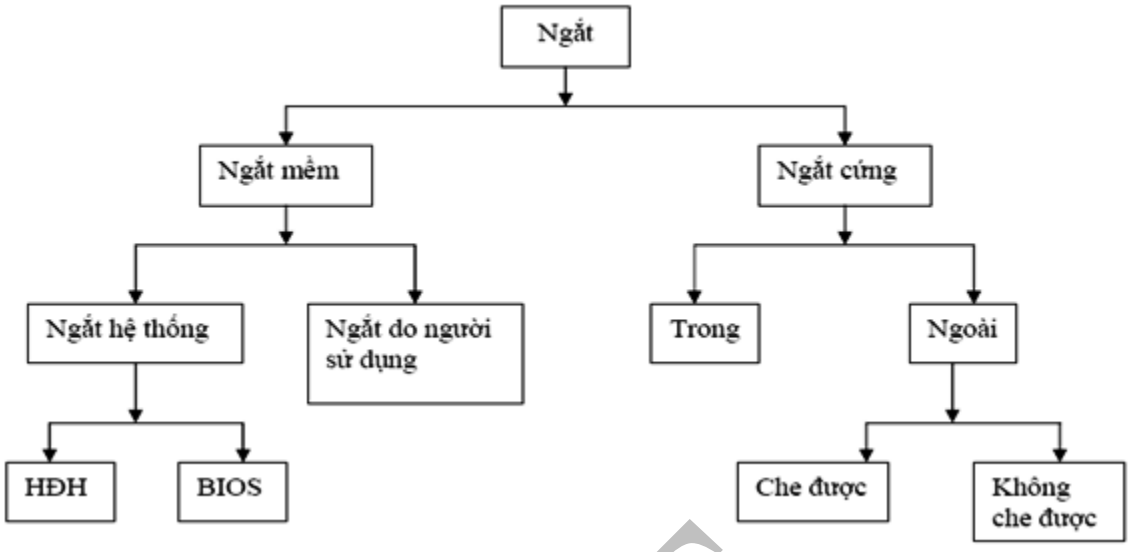
Cờ trạng thái ngắt (Interrupt flags)

Mỗi ngắt kết hợp với cờ, chỉ cần 1 bit, báo trạng thái cuiar ngắt đó. Các bit thường tập hợp trong 1 thanh ghi cờ ngắt (interrupt register hay interrupt flag) đọc/ghi được. CPU đọc để biết ngắt đã xuất hiện, và ghi (xóa) sau khi đã xử lí cho ngắt đó.

Mắt nạ ngắt (Interrupt mask)

Mỗi ngắt kết hợp với 1 bit của thanh ghi mặt nạ. Lập trình các bit của thanhg ghi sẽ cho phép hay cấm ngắt tương ứng kích hoạt.

Phân loại kiểu ngắt:



Hình 2.71 Các kiểu ngắt

- **Ngắt mềm**

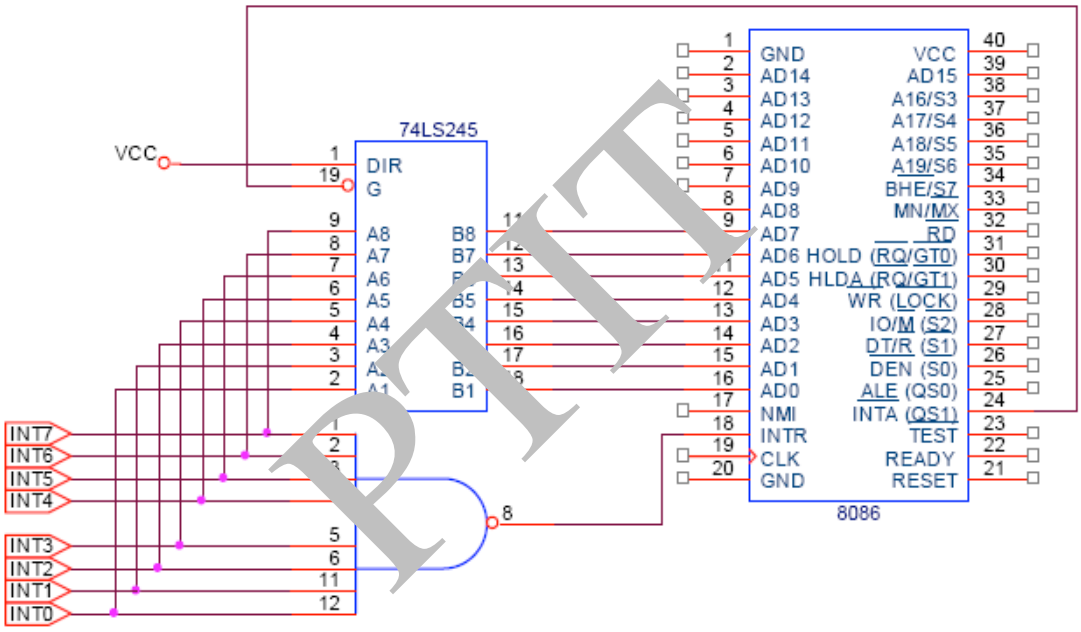
Ngắt mềm thực chất thực hiện một lời gọi hàm đặc biệt được kích hoạt bởi các nguồn ngắt là các sự kiện xuất hiện từ bên trong chương trình và ngoại vi tích hợp trên Chip. Ví dụ như ngắt thời gian, ngắt từ thiết bị như ADC/DAC, … Cơ chế ngắt này còn được hiểu là loại thực hiện đồng bộ với chương trình vì nó được kích hoạt và thực hiện tại các thời điểm xác định trong chương trình. Hàm được gọi sẽ thực thi chức năng tương ứng với yêu cầu ngắt. Các hàm đó thường được trỏ bởi một vector ngắt mà đã được định nghĩa và gán cố định bởi nhà sản xuất Chip. Ví dụ như hệ điều hành của PC sử dụng ngắt số 21hex để gán cho ngắt truy nhập đọc dữ liệu từ đĩa cứng và xuất dữ liệu ra máy in.

- **Ngắt cứng**

Ngắt cứng có thể được xem như là một lời gọi hàm đặc biệt trong đó nguồn kích hoạt là một sự kiện đến từ bên ngoài chương trình thông qua một cấu trúc phần cứng (thường được kết nối với thế giới bên ngoài qua các chân ngắt). Ngắt cứng thường được hiểu hoạt động theo cơ chế dị bộ vì các sự kiện ngắt kích hoạt từ các tín hiệu ngoại vi bên ngoài và tương đối độc lập với CPU, thường là không xác định được thời điểm kích hoạt. Khi các ngắt cứng được kích hoạt CPU sẽ nhận dạng và thực hiện lời gọi hàm thực thi chức năng phục vụ sự kiện ngắt tương ứng. Trong các cơ chế ngắt khoảng thời gian từ khi xuất hiện sự kiện ngắt (có yêu cầu phục vụ ngắt) tới khi dịch vụ ngắt được thực thi là xác định và tuỳ thuộc vào công nghệ phần cứng xử lý của Chip.

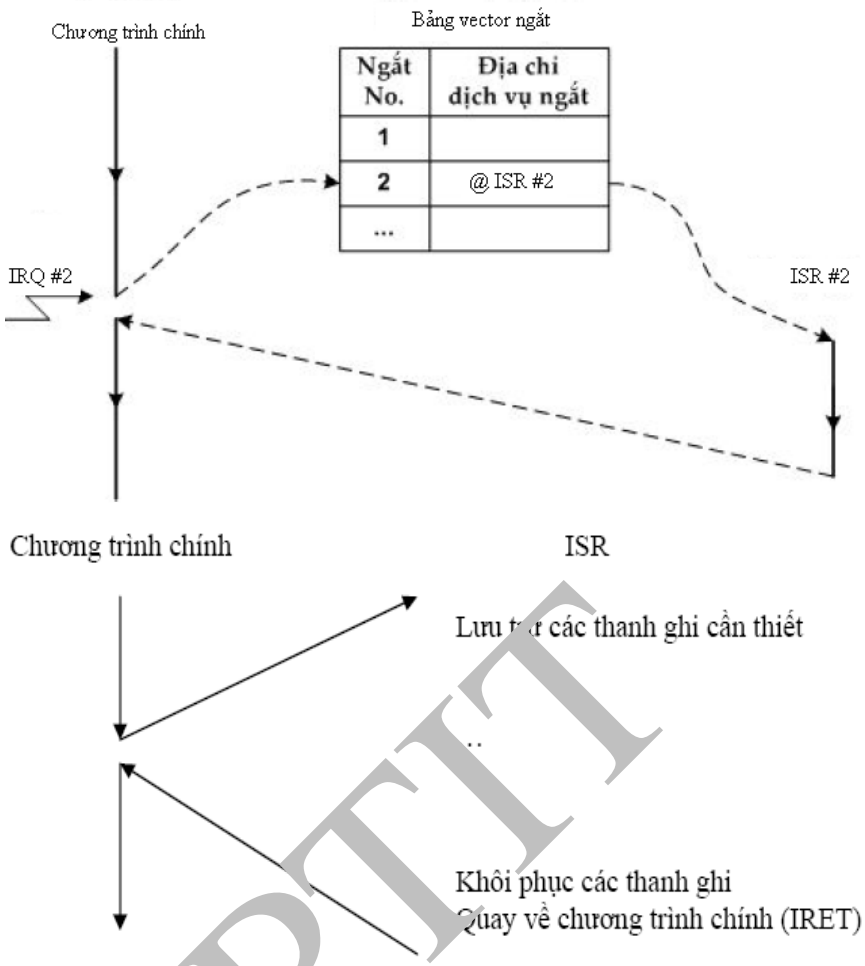
Tổ chức tạo ngắt có thể đơn giản nếu CPU hỗ trợ nhiều đầu vào ngắt và hệ thống không cần nhiều ngắt cứng. Ví dụ như trong CPU 8080/8085 có 4 ngắt (RST7.5, RST6.5, RST5.5 và

TRAP) và 8 ngắt mềm (RST0 đến RST7). Tuy nhiên số thiết bị nhiều, số tín hiệu ngắt cũng tăng lên, do đó có một cơ chế quản lí ngắt hieehu quả hơn, là cơ chế *vector*. Trong cơ chế này CPU chỉ cần 1 đầu vào (INTR) và một đầu ra (INTA) đe nhận và trả lời chấp nhận ngắt. Phần cứng do đó cần thêm một vi mạch hỗ trợ cho CPU, gọi là vi mạch điều khiển ngắt lập trình được(*Programmable Interrupt Controller*), ví dụ vi mạch Intel 8259 với 8 đầu vào ngắt và 1 đầu ra. Mỗi một ngắt (mỗi thiết bị) cần có một chương trình xử lý riêng, gọi tên chung là chương phục vụ ngắt (*Interrupt Service Routine* –ISR). Địa chỉ của mỗi ISR đặt trong một vùng bộ nhớ qui ước, ví dụ với Intel CPU, đó là vùng địa chỉ từ 00000-003FF (1024 byte), vùng bộ nhớ này gọi là bảng vector ngắt, cứ 4 byte cho một ngắt (CS:IP), nên có 256 ngắt. Bảng vector ngắt thuộc vùng RAM hệ thống. (Thử nghĩ xem tại sao để ở RAM, mà không để ở ROM ?).



Hình 2.72 Thiết kết với ngắt cứng che được INTR của CPU

Hãy xem xét tiến trình ngắt như sau ở chạy trình:
Khi cần trao đổi thông tin, thiết bị ngoại vi gửi tín hiệu *yêu cầu ngắt* (*Interrupt Request*-IRQ) tới đầu vào INTR của CPU. CPU sẽ thực hiện nốt lệnh hiện tại và trả lời bằng tín hiệu *nhận biết yêu cầu ngắt* (INTA). Chương trình chính lúc này bị tạm dừng (ngắt) và CPU chuyển sang thực hiện *chương trình con phục vụ ngắt (thực thi ISR của ngắt đó)*, tức là chương trình con trao đổi thông tin với thiết bị ngoại vi yêu cầu ngắt. Sau khi xong công việc phục vụ ngắt, CPU quay về thực hiện tiếp chương trình chính kể từ lệnh tiếp theo sau khi bị ngắt.



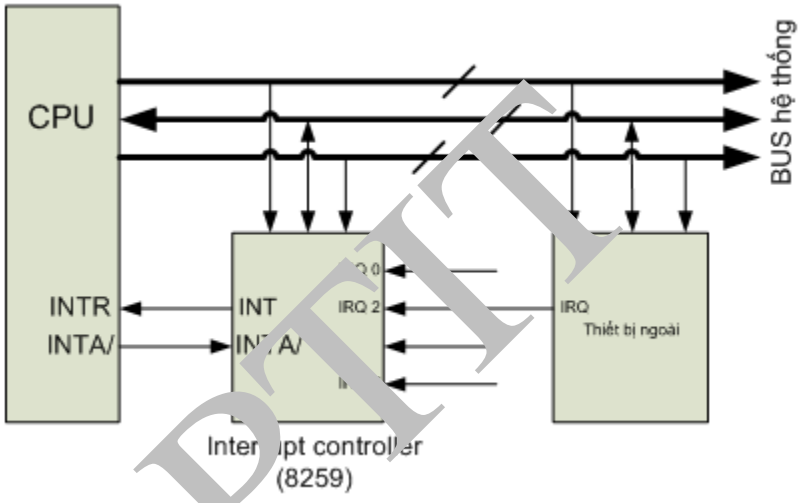
Hình 2.[illegible] Vector ngắt và chuyển xử lý tới ISR

Các tín hiệu yêu cầu phục vụ ngắt từ một thiết bị ngoại vi bất kỳ được gửi tới chân nhận yêu cầu ngắt của CPU có thể thông qua một khối điều khiển ngắt. Tuỳ theo người lập trình mà yêu cầu ngắt đó có được chuyển tới CPU hay không (thông qua chiến thuật nhận và xử lý ngắt với các lệnh cho phép ngắt (EI) hay cấm ngắt (DI). Trong trường hợp yêu cầu ngắt được gửi tới CPU, xử lý của CPU gồm các bước sau:

Quá trình thực hiện ngắt:

- CPU hoạt động bình thường
- Khi thiết bị vào/ra sẵn sàng chuyển số liệu sẽ gửi yêu ngắt tới CPU bằng tín hiệu IRQ tới dầu vào INTR (Interrupt Request) của CPU
- CPU thực hiện nốt lệnh đang thực hiện trước khi trả lời chấp nhận ngắt
- CPU nhận và tìm cách xác định ngắt và trả lời thiết bị vào/ra bằng tín hiệu INTA (*Interrupt Acknowledgement*)
- Đẩy PSW (*Program State Word*) và PC (*Program Counter*) vào ngăn xếp

- Xoá các cờ IF (*Interrupt Flag*) và cờ TF (*Trap Flag*)
- TB vào/ra thông qua bộ điều khiển ngắt cho biết địa chỉ của chương trình con phục vụ ngắt ISR của ngắt đó. CPU nạp địa chỉ này vào PC.
- CPU nhảy đến chương trình con ISR và thực hiện xử lý
- Chương trình ISR sẽ đẩy các thanh sẽ bị thay đổi trong chương trình con vào ngăn xếp.
- Chương trình ISR sẽ thực hiện việc chuyển số liệu giữa thiết bị vào/ra và bộ nhớ qua ACC của CPU.
- Sau khi chuyển số liệu xong, CPU khôi phục các thanh ghi
- Khôi phục PC và PSW từ ngăn xếp, trở về chương trình chính thực hiện tiếp nhiệm vụ trước khi có ngắt.



Hình 2.74 Tổ chức ngắt với điều khiển ngắt

Những điều lưu ý khi viết chương trình ISR: những lệnh đầu tiên của ISR thực hiện là:

- ✓ Đẩy các thanh ghi của CPU vào STACK,
- ✓ Thực hiện việc cấm ngắt để tránh đệ qui ngắt nếu cần, hoặc cấm các ngắt khác
- ✓ Mã xử lý của ISR …
- ✓ Khôi phục các thanh ghi của CPU
- ✓ Khôi phục lại khả năng chấp nhận ngắt cho các ngắt tạm cấm
- ✓ Quay về (RETURN) chương trình bị gián đoạn trước đó.

Lập trình ngắt và cài đặt vector ngắt là việc làm cần thận trọng. Các ngắt tổ chức theo vector đồng thời cũng là theo mức ưu tiên, nên khi thiết kế cần có ý tưởng rõ ràng.

Về mức ưu tiên và sử dụng mức ưu tiên:

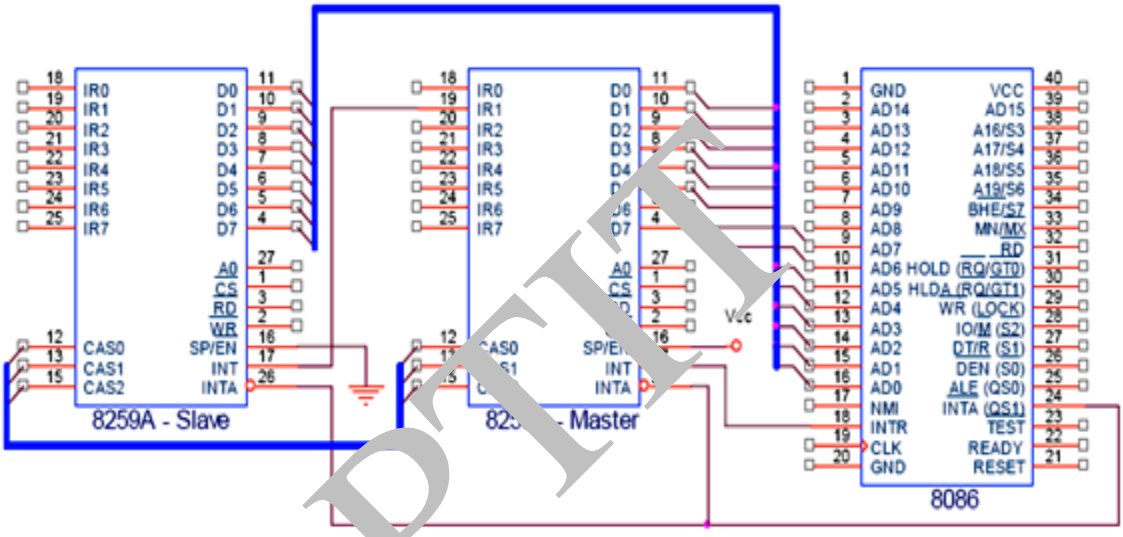
- ✓ NMI (*Non Maskable Interrupt*) là yêu cầu ngắt tức thời, không thể cấm hay cho phép băng chương trình.

- ✓ Ngắt theo Vector
- ✓ Ngắt mềm dùng lệnh (INT) để gọi các chương trình phục vụ ngắt của hệ thống.

Ví dụ các ngắt hệ thống: Đồng hồ thời gian thực, báo lỗi phần cứng, báo mất nguồn nuôi, báo lỗi trong truyền tin …

Thông thường ta hay quan tâm nhiều đến đáp ứng của CPU với sự kiện ngắt và thời gian thực hiện tác vụ ngắt. Ở đây thời gian đáp ứng phụ thuộc và quyết định bởi tốc độ và khả năng xử lý của phần cứng còn thời gian thực hiện tác vụ ngắt chủ yếu quyết định bởi tác vụ ngắt đó dài hay ngắn và do chương trình quyết định.

Ví dụ: Thiết kế với 8259, hỗ trợ 16 ngắt cứng:



Hình 2.75 Mở rộng số ngắt với 2 vi mạch 8259

Để sử dụng 8259, cần nghiên cứu đặc tả và phương pháp lập trình cho vi mạch này. Sơ đồ chưa có Chip select (CS/), còn tùy vào thiết kế cụ thể.

b) Truy nhập trực tiếp vào bộ nhớ (*Direct Memory Access-DMA*)

Trong các phương pháp vào/ra trình bày trên có các nhược điểm sau:

1) Sử dụng phương pháp vào/ra điều khiển bằng chương trình ta thấy:

- Đọc dữ liệu vào bằng chương trình phải chuyển số liệu giữa thiết bị vào/ra và bộ nhớ thông qua thanh ghi của CPU, ví dụ AX, AL (gọi chung là ACC):

Bộ nhớ ⇔ [CPU_ACC] ⇔ [Cổng] ⇔ [Thiết bị]

Trong đó có hai bước cần thực hiện:

1. [DATA_ghép nối_từ thiết bị] → ACC
2. [ACC] → MEM

- Tương tự khi ghi dữ liệu ra thiết bị phải đưa nội dung của ô nhớ tới TB vào/ra, cũng phải qua 2 bước:

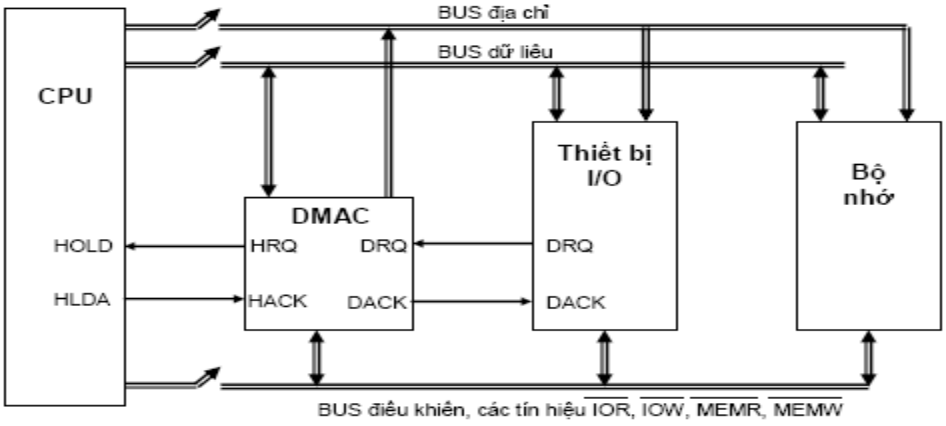
1. [MEM] → ACC
2. [ACC] → [DATA_ghép nối_ra thiết bị]

Như vậy việc chuyển số liệu giữa thiết bị ngoại vi và bộ nhớ cần hai bước, tốc độ chậm. Ở đây chứ tính thêm bước kiểm tra trạng thái READY của thiết bị.

2) Phương pháp vào/ra bằng ngắt bảo đảm thiết bị vào/ra được phục vụ gần như tức thời (trong thời gian ngắn). Nhưng trong chương trình con phục vụ ngắt, quá trình chuyển số liệu được thực hiện bằng lệnh chương trình nên tốc độ trao đổi cũng không thể cao được vì khâu này cũng không khác gì 1).

Tuy nhiên trong máy tính, việc phải trao đổi một khối lượng lớn dữ liệu giữa bộ nhớ RAM và thiết bị ngoài không thể sử dụng 2 cách nói trên được. Ví dụ khi ghi/đọc đĩa cứng, cập nhật cho video RAM để tạo ảnh trên màn hình là các loại ứng dụng kiểu như vậy. Kĩ thuật *vào/ra thâm nhập bộ nhớ trực tiếp sẽ* khắc phục các nhược điểm trên. Bản chất của kĩ thuật này như sau: khi cần trao đổi khối dữ liệu giữa RAM và thiết bị, thì tác vụ này phải xảy ra nhanh nhất có thể và chỉ phụ thuộc vào khả năng của thiết bị và phải thực hiện hoàn toàn bằng cơ chế điện tử, không sử dụng tới các lệnh máy qua chương trình. Nói vậy có mâu thuẫn ? Hay nói cách khác để làm được, " Hãy loại bỏ tạm thời CPU và nắm lấy quyền khống chế BUS hệ thống trong thời gian trao đổi dữ liệu !". Quá trình thiết kế các giải pháp này như sau:

Cần có một vi mạch khả trình có khả năng thay thế CPU, tạo ra các tín hiệu giống như CPU tạo (BUS địa chỉ, BUS dữ liệu, BUS điều khiển) và kiểm soát BUS hệ thống trong suốt thời gian thực hiện DMA. Có các tín hiệu đối thoại với CPU (HRQ, HLDA), các tín hiệu đối thoại với thiết bị ngoài (DREQx, DACKx). Vi mạch như vậy gọi là DMA Controller, hay DMAC, vi mạch phổ biến có tên Intel 8237A. Sơ đồ nguyên lí như sau:



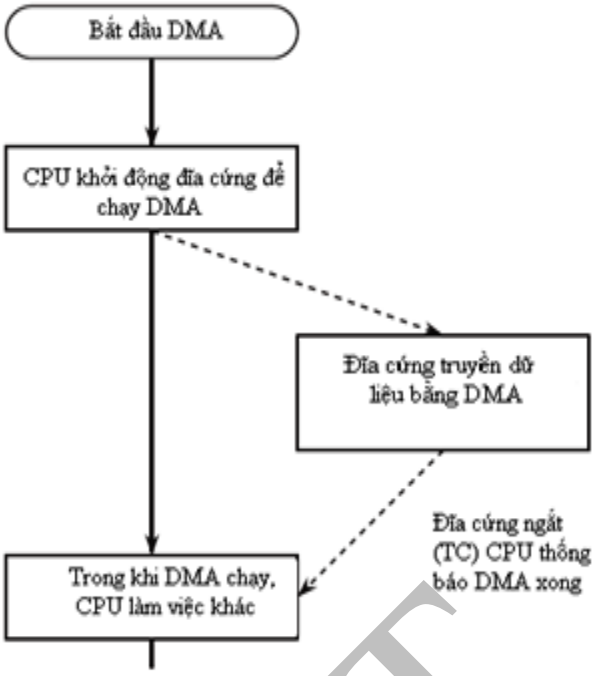
Hình 2.76 Nguyên lí DMA

Quá trình thực hiện DMA:

1) CPU làm việc bình thường
2) Khi thiết bị ngoại vi muốn chuyển số liệu trực tiếp với bộ nhớ thì gửi yêu cầu tới DMAC qua tín hiệu DRQx (DMA Reqest thứ x, mỗi DMAC có khả năng nhận 4 DRQ).
3) Bộ điều khiển DMAC chuyển yêu cầu này tới CPU qua tín hiệu HRQ: Yêu cầu CPU "tách ra " khỏi BUS hệ thống.
4) CPU thực hiện nốt chu kỳ máy đang thực hiện, treo BUS và trả lời DMAC bằng tín hiệu HLDA: Chấp nhận và đã treo BUS.
5) DMAC trả lời thiết bị vào/ra bằng tín hiệu DACKx, DMAC quản lí BUS hệ thống và phát sinh các tín hiệu địa chỉ lên BUS địa chỉ, hướng tới bộ nhớ,
6) Phát các tín hiệu điều khiển: MEMRD/, MEMWR/, IORD/, IOWR/, để thực hiện DMA vào: đọc thiết bị/ghi bộ nhớ hay DMA ra: dọc bọ nhớ/ghi ra thiết bị.
7) Điều khiển chuyển số liệu giữa bộ nhớ và thiết bị vào/ra, chạy đồng bộ theo BUS-clock. Số liệu chuyển giữa bộ nhớ và thiết bị vào/ra thường là cả một khối, có độ dài tùy ý qua lập trình.
8) Khi chuyển xong số liệu DMAC đưa tín hiệu TC hay EOP (Terminal Count, End of Operation-EOP) thành tích cực để báo một quá trình DMA kết thúc (tín hiệu TC đóng vai trò một ngắt khi sử dụng để thông báo cho CPU). DMAC treo BUS, DMAC hũy HRQ. CPU hũy HLDA, CPU trở lại quản lí BUS hệ thống. Chu kì DMA hoàn tất.

Qui trình chạy DMA xảy ra rất nhanh, chỉ phụ thuộc vào khả năng trao đổi của thiết bị. Có một vài thủ thuật DMA xảy ra trong máy tính, tham khảo thêm tài liệu, ví dụ Intel 8237 DMAC, để hiểu thêm.

DMA kết hợp với ngắt bằng tín hiệu TC, hay EOP khi kết thúc chu kì DMA. Ví dụ ứng dụng: ghi đọc dữ liệu RAM ⇔ Đĩa cứng, RAM ⇔ NIC.



Hình 2.77 DMA và hoạt động của CPU là độc lập

Ví dụ: Thiết kế với DMAC: Sử dụng Intel DMAC 8237

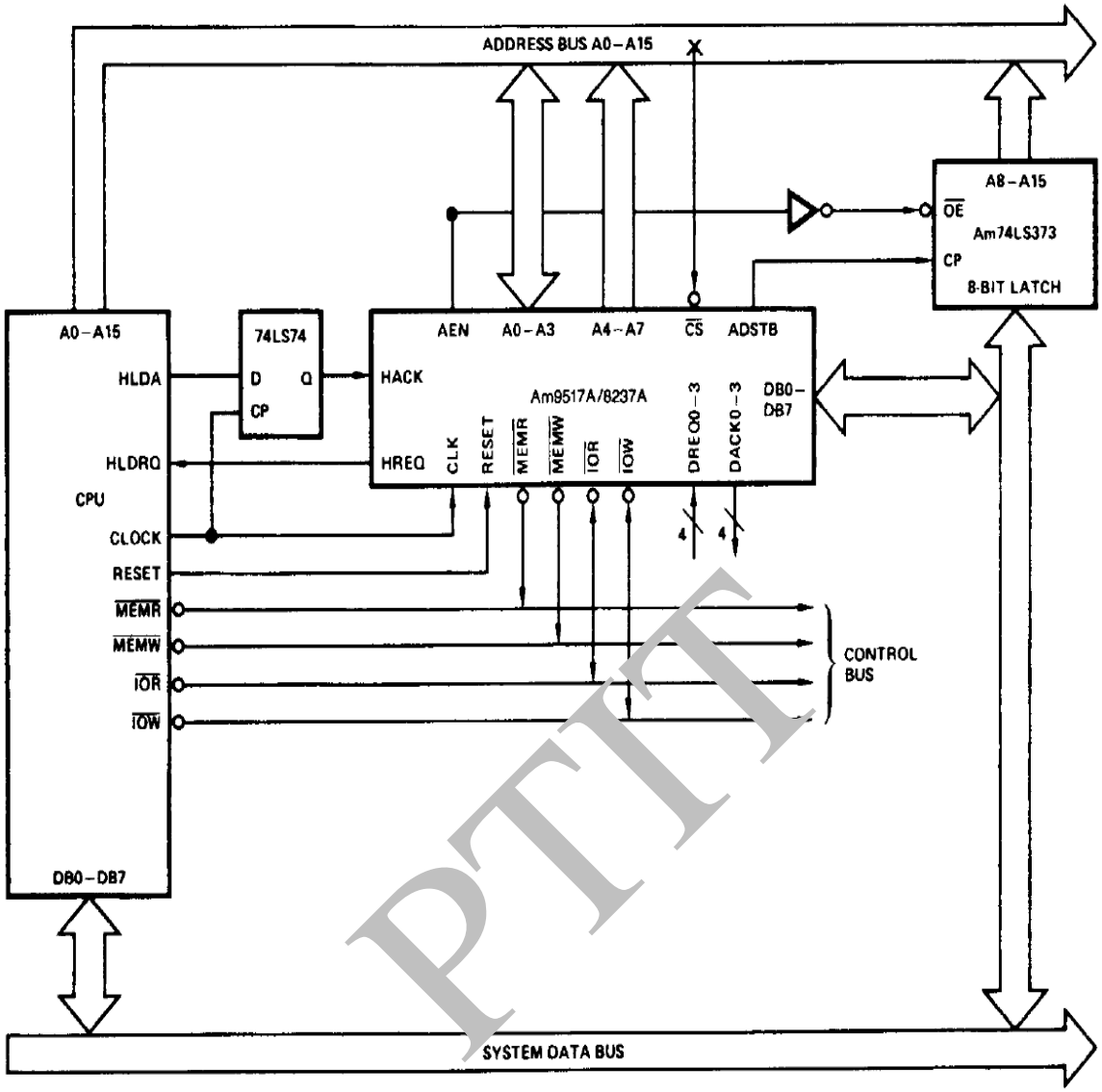
Có hai phương pháp thực hiện DMA:

- Cách như đã mô tả, CPU trao quyền sử dụng BUS hệ thống cho DMAC;
- Cách thứ hai gọi là DMAC lấy trên chu kì: DMAC tận dụng những khoản thời gian trong một chu kì BUS mà khi đó CPU không có truy nhập bộ nhớ, ví dụ các chu kì CPU phát địa chỉ cho các chu kì đọc hay ghi bộ nhớ).

Một khi DMA hoạt động, có một số cách trao đổi dữ liệu sau đây:

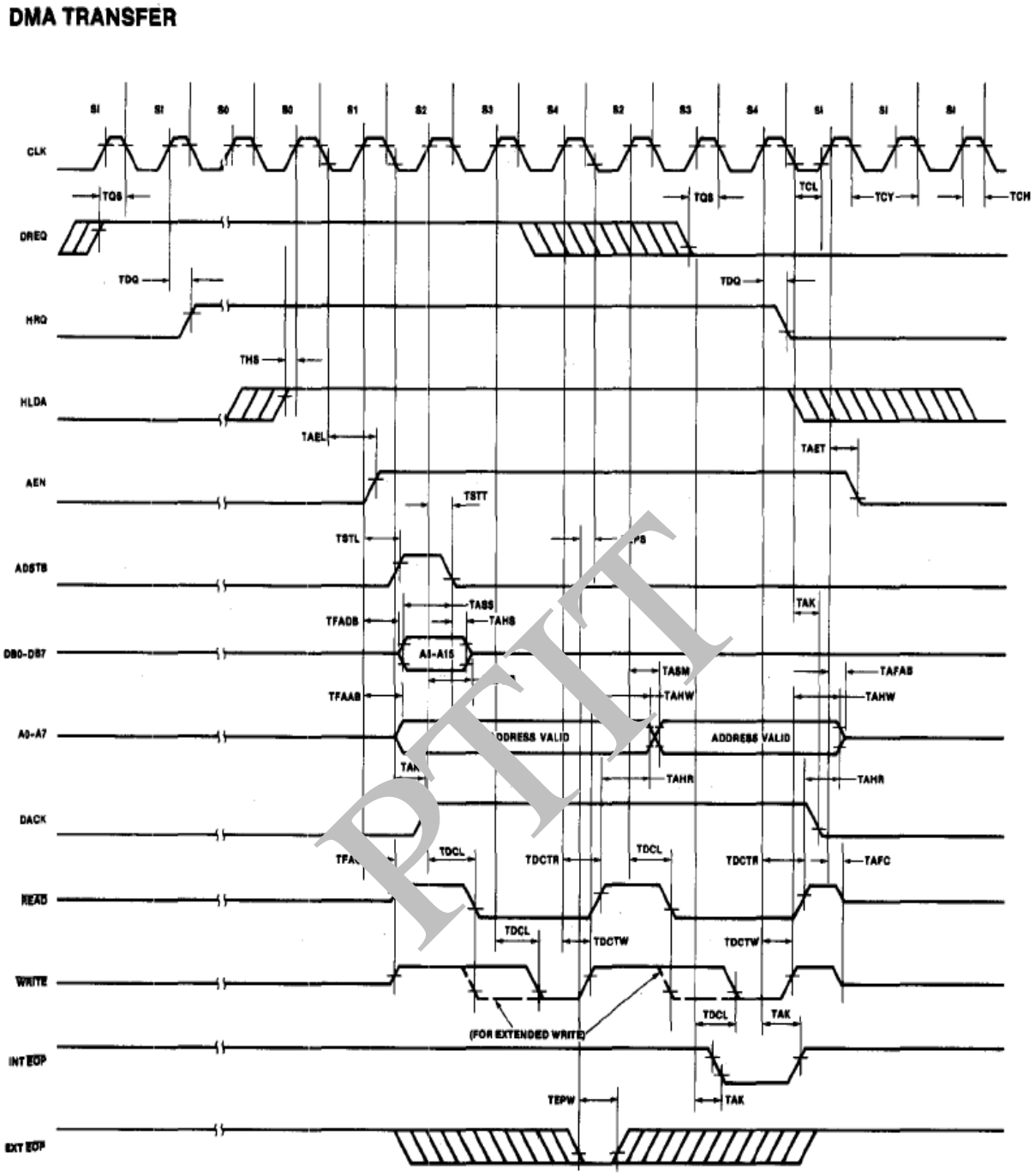
- Chế độ trao đổi đơn,
- Chế đọ theo yêu cầu,
- Chế độ mở rộng các vi mạch DMAC theo đa mức (cascade).

Khi thiết kế với DMAC xem chi tiết hướng dẫn cách ghép và lập trình cho DMAC.

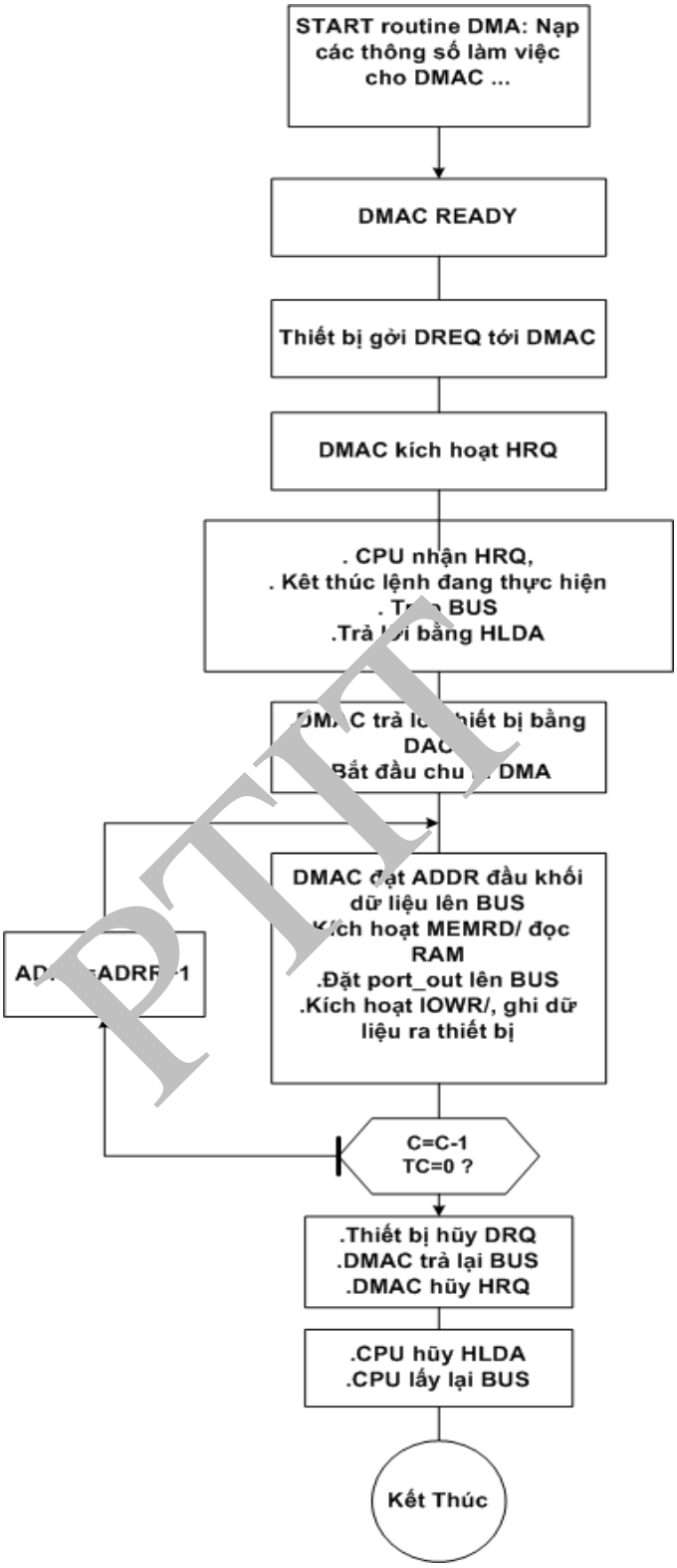


Hình 2.78- Ghép nối DMAC 8237 vào với CPU 8085

2. CPU 8080/8085
3. DMAC 8237A
4. Latch 74LS373 chốt địa chỉ A15-A8 do DMAC phát ra đưa lên BUS hệ thống



Lưu đồ thời gian của qui trình DMA



Hình 2.79 Lưu đồ DMA ghi dữ liệu từ RAM ra thiết bị ngoài

a) **Ghép nối hỗn hợp:** là phương pháp ở đó có qúa trình chuyển đổi vai trò kiểm soát quá trình trao đổi dữ liệu. Ví dụ thời điểm ban đầu CPU chủ động, nhưng có lúc sau đó thiết bị chủ động. cách hoạt động này thường phối hợp giữa phần cứng và thủ tục điều khiển phải đồng bộ.

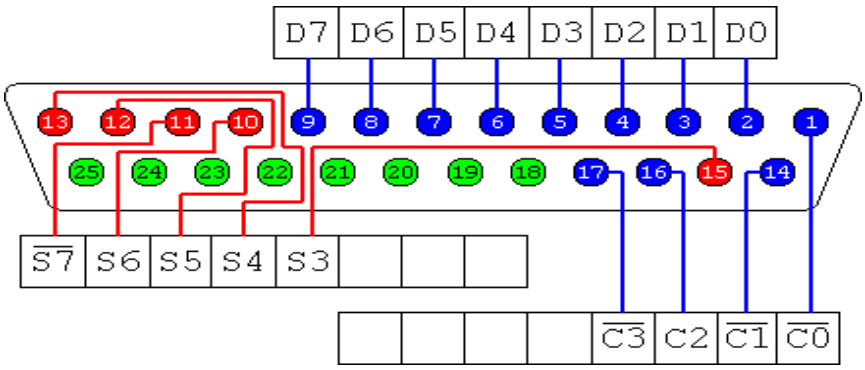
2.6.4 Cổng vào/ra

Khi nói về ghép nối ta hay đề cập tới khái niệm cổng. Thực ra cổng ở đây bao gồn hai phần: là sử dụng các mạch điện tử để làm thích ứng các tín hiệu mang thông tin trong thiết bị kĩ thuật số nói chung và phần mềm điều khiển hoạt động của cổng. Ở góc độ hợp nhất trong thiết kế, cổng phải do CPU quản lí bởi vì cổng được tạo thành dựa vào kiến trúc của CPU và thực hiện bằng lệnh máy tính. Như đã trình bày, CPU "nhìn" cổng theo hai cách: cổng như là một ô nhớ (*Memory mapped I/O*), như vậy cổng sẽ chiếm một phần trong không gian địa chỉ của CPU. Hay CPU "nhìn" cổng theo kiểu cổng thuần túy (*I/O mapped I/O*) và cổng có không gian địa chỉ riêng mà CPU dành cho các hoạt động vào/ra. Trường hợp này đã nêu ở phần ghép nối khi thiết kế các kĩ thuật ghép nối vào/ra. Các cổng như vậy gọi là cổng logic và dùng để điều khiển cổng vật lí mà nó ánh xạ tới.

Các cổng vật lí phải có mô tả kĩ thuật rất chi tiết vì nó dùng để ghép nối các tín hiệu vật lí (điện, quang, từ trường) vào máy tính. Các cổng này thông thường trong công nghiệp đều được chuẩn hóa để đảm bảo tính tương thích cho mọi hệ thống, mà mục đích là để hỗ trợ cho việc truyền dữ liệu giữa các thiết bị kĩ thuật số.

- *Cổng song song*. Cổng song song cho phép trao đổi thông tin giữa hai thiết bị số (PC, HTN) đồng thời nhiều bit một lần. Ví dụ đồng thời từ 2 bit trở lên. Việc truyền thông có thể đơn giản, nhưng cũng có thể rất phức tạp. Ví dụ, đơn giản chỉ cần thông qua một số tín hiệu đối thoại để đồng bộ phát/nhận dữ liệu. Một trong các cổng song song phổ biến trong PC là cổng nối với máy in.

  Dưới đây là cổng song song nối ra máy in trên PC (*Standard Parallel Port (SPP)*), tên đầu nối ra là DB25 với định nghĩa ý nghĩa các chân của cổng.

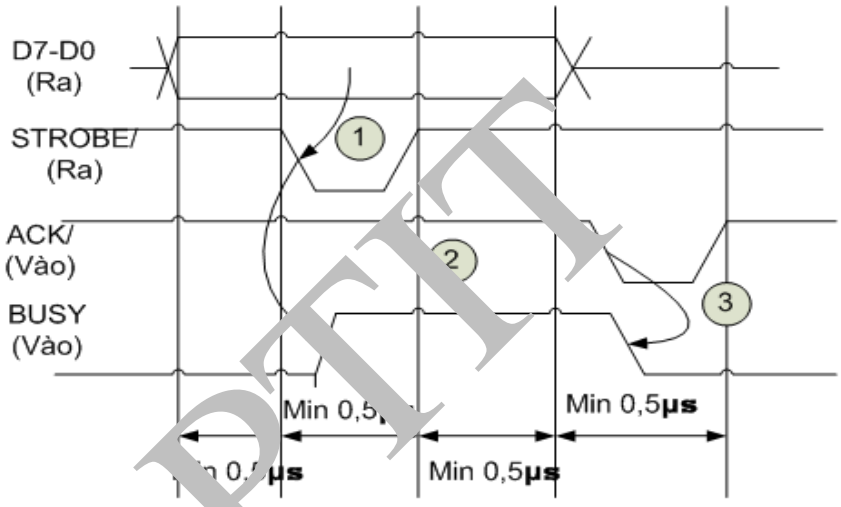


Hình 2.80 Định nghĩa các chân của cổng SSP

Chân	Tên gọi	Hướng dữ liệu	Mô tả
1	STROBE	Lối vào/ra, đảo	Byte được in
2	D0	Lối ra (Output)	Đường dữ liệu D0
3	D1	Lối ra	Đường dữ liệu D1
4	D2	Lối ra	Đường dữ liệu D2
5	D3	Lối ra	Đường dữ liệu D3
6	D4	Lối ra	Đường dữ liệu D4
7	D5	Lối ra	Đường dữ liệu D5
8	D6	Lối ra	Đường dữ liệu D6
9	D7	Lối ra	Đường dữ liệu D7
10	ACK	Lối vào (Input)	Acknowledge (Xác nhận)
11	BUSY	Lối vào, đảo	1: Máy in bận 0: Máy in không bận
12	PE	Lối vào	Báo hết giấy
13	SLCT	Lối vào	SELECT (Lựa chọn)
14	AF	Lối vào/ra, đảo	Auto Feed (Tự nạp)
15	ERROR	Lối vào	Error (Lỗi)
16	INIT	Lối vào/ra	0 : Reset lại trạng thái máy in
17	SLCTIN	Lối vào/ra, đảo	Select in
18	GND		Nối đất
19	GND		Nối đất
20	GND		Nối đất
21	GND		Nối đất

22	GND		Nối đất
23	GND		Nối đất
24	GND		Nối đất
25	GND		Nối đất

Lưu dồ thời gian mô tả nguyên tắc ghép nối với máy in, trong đó (Ra) có nghĩa tín hiệu ra từ cổng và (Vào) là tín hiệu từ máy in.

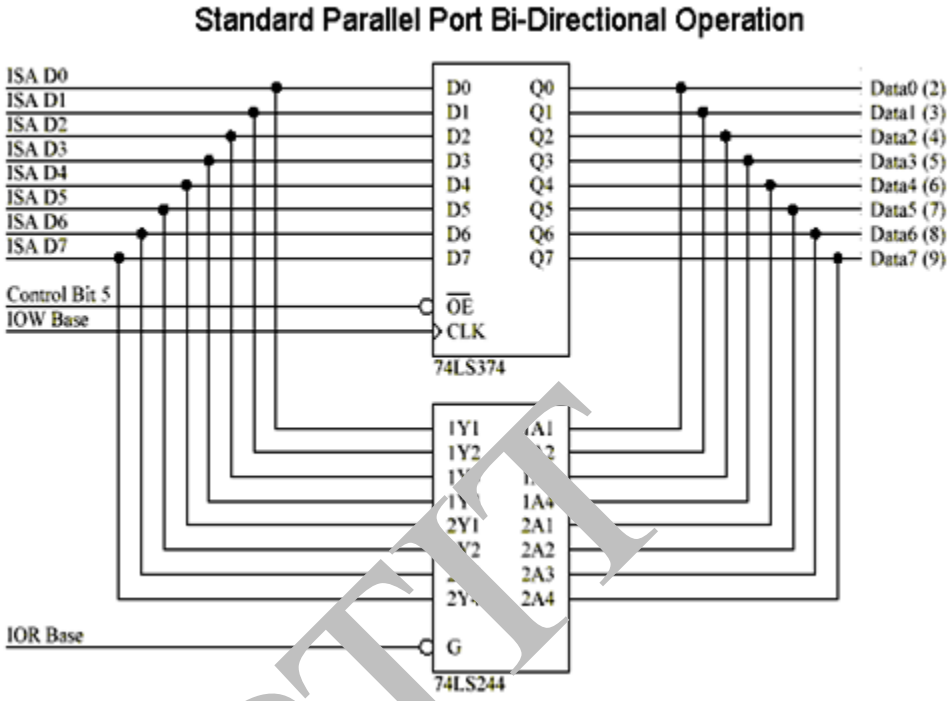


Hình 2.8 Lưu đồ các tín hiệu cổng song song

Địa chỉ truy nhập cổng:

Address	Notes:
3BCh - 3BFh	Địa chỉ này trước đay hay kết hợp trên bo mach video. Không hỗ trợ chế độ ECP addresses (Extended Capabilities Mode)
378h - 37Fh	Cho cổng LPT 1
278h - 27Fh	Cho cổng LPT 2

Khi sử dụng cổng song song để đưa dữ liệu vào, có thể dử dụng các tín hiệu trạng thái như ACK, PE, BUSY, SELECT, các cổng được dùng tùy biến theo sự khéo léo của người thiết kế. Hạn chế ở đay là chỉ có thể đọc vào 4 bit một lần. Ở các PC hiện đại cổng này vạn năng hơn gọi là cổng hai chiều:

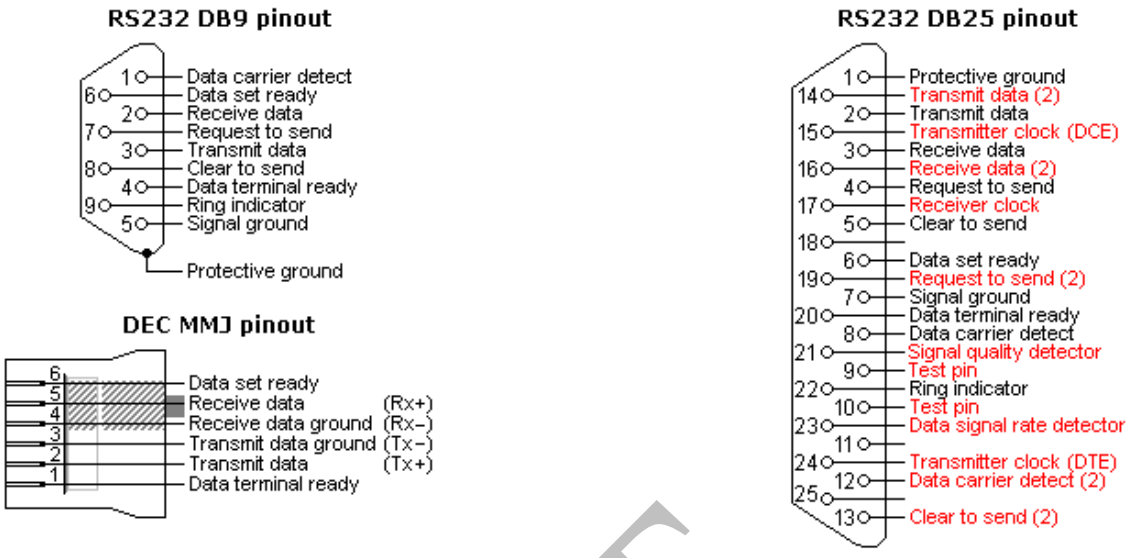


Hình 2.82 Cổng song song hai chiều

Xem thêm: http://www.beyondlogic.org/spp/parallel.htm#6

- *Cổng nối tiếp.* Trong khi cổng song song có ưu điểm về số bit truyền và có thể đạt tốc độ rất cao, thì điểm yếu của cổng là khoản cách kết nối. Để giải quyết vấn đề này phải sử dụng tới cổng nối tiếp. Bản chất nối tiếp là phải trao đổi giữa hai máy với nhau từng bit một và lĩnh vực ứng dụng của truyền thông nối tiếp lại rất phổ biến. Tuy nhiên truyền nối tiếp lại rất phức tạp, đòi hỏi thiết kế và đặc biệt là qui trình phát/thu, gọi chung là giao thức truyền thông. Do đó ở đây không đi vào chi tiết cụ thể của truyền thông nối tiếp, có thể thao tham khảo thêm các tài liệu khác:
  http://www.beyondlogic.org/serial/serial.htm#2 và
  http://www.lammertbies.nl/comm/cable/RS-232.html#pins. Trên PC có ít nhất 2 cổng với tên gọi COM1 và COM2 với chuẩn RS-232.

Chân cổng nối tiếp (Serial) D25 và DB9



Hình 2.83 Đầu nối RS 232 các loại DB9, DB 25 và DEC MMJ

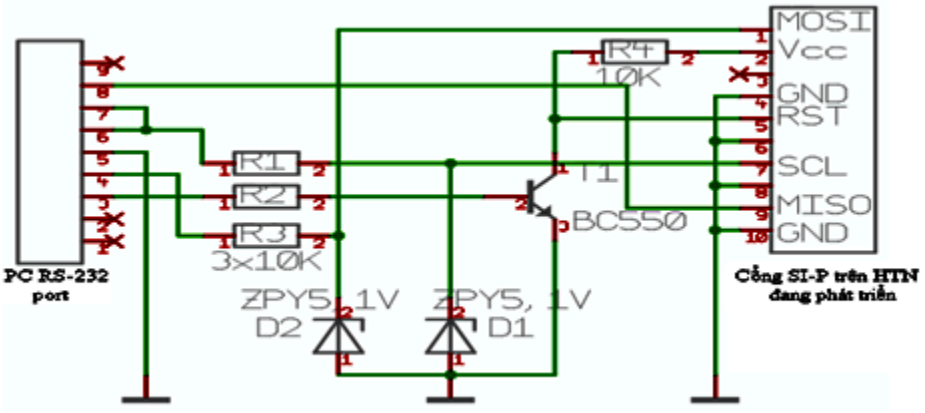
D-Type-25 Pin No.	D-Type-9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TxD (Ra)	(Transmit Data Serial). Cữ liệu phát ra từ DTE (ví duh PC, HTN)
Pin 3	Pin 2	RxD (Vào)	(Receive Data Serial). Nhận dữ liệu từ DCE về DTE
Pin 4	Pin 7	RTS (Ra)	(Request To Send). DTE sẵn sàng phát dữ liệu ra.
Pin 5	Pin 8	CTS (Vào)	(Clear To Send). DCE sẵn sàng nhận dữ liệu do DTE phát ra.
Pin 6	Pin 6	DSR (Vào)	(Data Set Ready). DTE sẵn sàng nhận dữ liệu từ DCE.
Pin 7	Pin 5	SG (Vỏ máy)	Signal Ground
Pin 8	Pin 1	DCD (Vào)	(Carrier Detect). Đã kết nối với DCE

			từ xa qua PSTN.
Pin 20	Pin 4	DTR (Ra)	Data Terminal Ready. DCE muốn phát dữ liệu
Pin 22	Pin 9	RI (Vào)	(Ring Indicator). DCE báo có chuông đến từ PSTN.

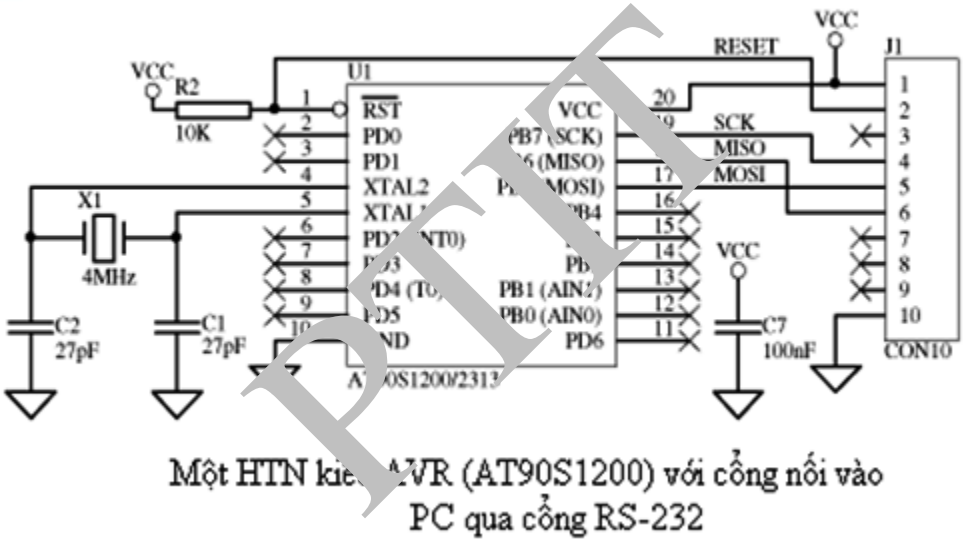
Một số cổng nói tiếp khác:
RS-422 (Hệ thống tốc độ cao tương tự RS-232 nhưng dùng tín hiệu vi sai)
RS-423 (Hệ thống tốc độ cao tương tự RS-422 nhưng dùng tín hiệu kiểu không cân bằng)
MIL-STD-188 (chuẩn quân sự, giống RS-232 nhưng chất lượng tố hơn (trở kháng, sườn lên dốc hơn)
EIA-530 (tốc độ cao sử dụng chuẩn các thuộc tính về điện cua EIA 423, chân nối của RS-422 hay RS-423,)
EIA/TIA-561 Kết nôi 8 vị trí, không đồng bộ giữa thiết bị dữ liệu đầu cuối (Data Terminal Equipment) và thiết bị mạch dữ liệu kết thúc (Data Circuit Terminating Equipment) dùng trao đổi dữ liệu nhị phân nối tiếp.
EIA/TIA-562 (Chuẩn qui định các giao diện kết nối sô không cân bằng (Electrical Characteristics for an Unbalanced Digital Interface (low-voltage version of EIA/TIA-232)
TIA-574 (standardizes the 9-pin D-subminiature connector pinout for use with EIA-232 electrical signalling, as originated on the IBM PC/AT)).

- Một số cổng hiện đại khác:
  - ✓ USB (Universal Serial Bus) là cổng phổ biến dùng trong các thiết bị nhúng, ví dụ bộ nhớ flash ngoài dùng với PC. Các thiết bị nhúng trong một tổ hợp thiết bị nối với nhau qua HUB để trao đổi dữ liệu…
  - ✓ Cổng 1394 (còn gọi là *FireWire)* cực nhanh đạt tới 800Mbps, cho phép kết hợp tới 63 thiết bị cùng lúc, "cắm là chạy" và "cắm/rút nóng". Các thiết bị như camcorder truyền video với PC đạt tốc đo rất cao.

- Cổng **SIP** (*Serial Interrface port - Programmer*): đây là một cổng rất đặc biệt phát triển để kết nối giữa một bo mạch HNT đang phát triển với PC qua công COM, trong đó PC là môi trường lập trình phần mềm cho HTN.
  Xem http://www.lancos.com/siprogsch.html#baseboard



Hình 2.84 PC làm hệ phát triển phần mềm cho HTN, phù hợp tín hiệu giữa RS-232 cua PC và cổng SI-P của HTN đang phát triển



Hình 2.85 Cổng SI-P đơn giản, dùng nguồn từ RS 232 của PC

2.6.5 Ghép nối với tín hiệu tương tự (analog signal)

Các thiết bị mà HTN phải kiểm soát, điều khiển sử dụng các tín hiệu để liên lạc với HTN thông qua các ghép nối. Các tín hiệu có thể đã ở dạng rời rạc (*discrete*), hay biến thiên liên tục theo thời gian, gọi là tín hiệu tương tự (*analog*). Các tín hiệu trong tự nhiên phần lớn ở dạng tương tự, biểu diễn sự biến thiên liên tục theo thời gian của một đại lượng vật lí nào đó. Trong khi dùng máy tính để xử lý các tín hiệu đó, cần một công đoạn nhất định để chuyển hóa tín hiệu tương tự thành tín hiệu rời rạc, gọi là số hóa (*digitalization*). Làm số hóa đôi khi giống như "mã hóa" tín hiệu analog vậy, bởi vì có nhiều cách để số hóa và đầu ra của số hóa lại có thể biểu diễn bởi các

tập hợp nhị phân khác nhau. Vi mạch điện tử thực hiện chức năng số hóa gọi là bộ số hóa (*analog-to-digital converter-ADC*).

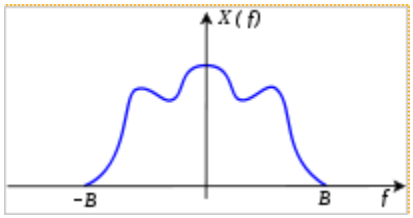
Ngược lại, sau khi xử lý, tính toán tín hiệu tương tự đã số hóa, HTN sẽ đưa ra các giá trị và chuyển tới các thiết bị. Nếu các thiết bị chỉ nhận tín hiệu tương tự, cần một quá trình chuyển đổi giá trị số thành giá trị tương tự. Vi mạch thực hiện gọi là bộ chuyển đổi số ra tương tự (*Digial Analog Converter –ADC*).

Một số khái niệm liên quan tới số hóa

- *Sai số lượng tử*: Quá trình số hóa luôn làm mất đi tính chính xác của thông tin chứa đựng trong tín hiệu gốc. Tuy nhiên với sai số nhất định thì kết quả vẫn được chấp nhận. Bảng dưới đây cho thấy sai số của mẫu được số hóa biểu diễn theo số bit thể hiện giá trị gốc:
  Ví dụ lấy mẫu audio:
  Đầu vào tương tự có giá trin từ 0 đến 1,
  Tần số lấy mẫu f_{sample} = 44,1 KHz.
  Resolution = f(số bit biểu diễn).
  Dung lượng bộ nhớ cần để lưu dữ liệu sau thời gian số hóa 1s, 60 s và 300 s

Size	Resolution	Storage (1s)	Storage (60s)	Storage (300s)
4 bit	0.0625000000	22050	1323000	6615000
6 bit	0.0156250000	33075	1984500	9922500
8 bit	0.0039062500	44100	2646000	13230000
10 bit	0.0009765625	55125	3307500	16537500
12 bit	0.0002441406	66150	3969000	19845000
16 bit	0.0000152588	88200	5292000	26460000
32 bit	0.0000000002	176400	10584000	52920000

- *Tốc độ lấy mẫu (tần số lấy mẫu)*, *độ trung thực số hóa*: phụ thuộc vào độ phân giải, vào thời gian lấy và biến đổi ($T_{CONVENSION}$) xong một mẫu của ADC. Sử dụng định luật Nyquist–Shannon có thể xác định tần sốn lấy mẫu phù hợp: .



$$X(f) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} x(t)\, e^{-i2\pi ft}\, dt.$$

Biểu diễn **Fourier transform** của x(t) như trên, và nếu X(f) = 0 cho mọi | f | > B, thì x(t) bị giới hạn bởi băng thông B. Từ đó tần số lấy mẫu đủ để tái tạo tín hiệu gốc là :

$f_{sample} > 2B$

Đơn giản hóa thì tần số lấy mẫu số hóa với ADC:

$f_{sample} = 1/ T_{CONVENSION}$, $f_{sample\ min} > 2\ f_{analog}$ là ít nhất.

Mẫu phải lấy và biến đổi liên tục để đảm bảo tính trung thực của tín hiệu. f càng lớn và số bit sử dụng càng cao tính trung thực càng tốt, tuy nhiên bộ nhớ dữ liệu lại là vấn đề. Có một số cách mã hóa đầu ra có thể bù trừ giữ 2 yêu cầu: chính xác và trung thực.

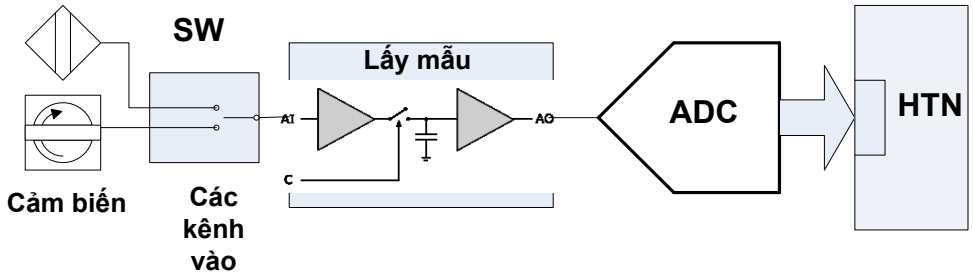
Ví dụ các thiết bị trang âm hi-fi có thang tần số từ 20 đến 20KHz, thì tần số lấy mẫu tối thiểu là 40 kHz.

- *Mã hóa (codecs: **Co**mpressor-**Dec**ompressor', '**Co**der-**Dec**oder', hoặc '**Co**mpression/**Dec**ompression algorithm):* Mã hóa dữ liệu là để xử lí bằng máy tính và đồng thời tối ưu độ dài dữ liệu, tức bài toán bộ nhớ, mà vẫn đảm bảo độ trung thực của tín hiệu gốc. Mã hóa mang lại nhiều lợi ích trong xủ lí tín hiệu và cho cả truyền thông. Mã hóa thường không chỉ là thuật toán mà còn được thực hiện ngay trên thiết bị điện tử, như ngay tại ADC/DAC. Một số định dạng mã hóa bao gồm:
  - Nhị phân**,**
  - Pulse code modulation (**PCM),**
  - Pulse width modulation (**PWM**).
  - Differential pulse coded modulation (**DPCM),**
  - Adaptive differential pulse code modulation (**ADPCM)**

  Là các biểu diễn định dạng số đầu ra của các ADC.

2.6.6 Biến đổi tương tự thành số (số hóa)

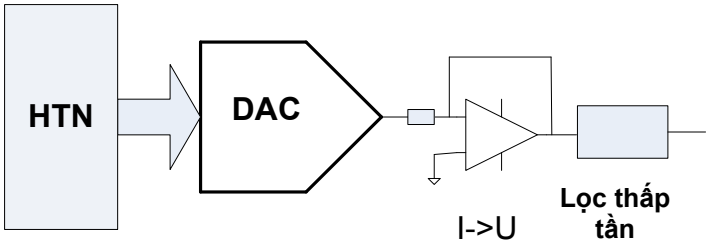
- Mô hình đầu vào của ADC và ghép nối với hệ thống
  - ✓ Bộ chọn kênh đầu vào: khi sử dụng một ADC có tính năng cao, thời gian biến đổi một mẫu nhanh, có thể dùng AD đó cho nhiều tín hiệu ở đầu vào.
  - ✓ Mạch trích mẫu: dùng để lấy một giá trị tương tự, " nhớ" trị số đó trong thời gian ADC chuyển thành giá trị số. Thời gian cần thiết để xong một biến đổi là $T_{CONVENSION}$.
- ADC ghép nối với máy tính (HTN)



Hình 2.86 ADC và ghép vào HTN

2.6.7 Biến đổi số thành tương tự (DAC)

Trong trường hợp thiết bị nhận giá trị tương tự từ HTN, cần có bộ biến đổi số-tương tự:



Hình 2.87 HTN và DAC

ADC và DAC là một chủ đề rất phức tạp, có thể tham khảo ơt các bài học khác. Khi sử dụng tham khảo tài liệu để chọn loại phù hợp với ứng dụng sẽ thiết kế. Tuy nhiên một số thông số sau đây được nêu ra khi chọn ADC/DAC:

- ✓ Độ phân giải của ADC: 8 bit, 10 bit, 12 bit;
- ✓ Thời gian chi phí cho một biến đổi (hay tốc độ biến đổi);
- ✓ Độ nhạy đầu vào tương tự (hay khoảng đo), giới (biên độ), cực tính tín hiệu đầu vào;
- ✓ Nguồn nuôi (đơn cực hay lưỡng cực:+/-), biên độ V;
- ✓ Cần hay không cần mạch lấy mẫu đầu vào (Sample-hold circuit);
- ✓ Ghép nối với vi tính (CPU), có tín hiệu đối thoại, ngắt;
- ✓ ….

2.7 KẾT CHƯƠNG

Chương này đề cập đến những kĩ thuật liên qua đến phần cứng, từ hoạt động của CPU, BUS (BUS của CPU và BUS hệ thống). Việc nắm bắt và hiểu các nguyên tắc làm việc của CPU giúp cho các phần tiếp theo, là những kĩ năng hỗ trợ cho việc thiết kế các hệ thống vi tính hay HTN, bởi về nguyên tắc không có gì khác biệt. Để tạo ra một hệ thống vi tính, cần phải thiết kế bộ nhớ và mở rộng ghép nối với các thiết bị ngoài. Ghép thiết bị ngoài vào HTN là khâu không thể thiếu, bởi HTN liên quan chính tới các thiết bị, là đối tượng HTN quản lí và điều khiển hoạt động của chúng. Các phương pháp ghép nối là chuẩn kĩ thuật, do đó hiểu được nguyên lí và áp dụng cho từng ứng dụng sẽ làm cho HTN hoạt động hiệu quả.

2.8 CÂU HỎI VÀ BÀI TẬP

2.8.1 Câu hỏi cuối chương

1. **BUS** hợp thành từ các tín hiệu nào của CPU, chức năng, chiều trao đổi dữ liệu mỗi BUS có khác nhau ?

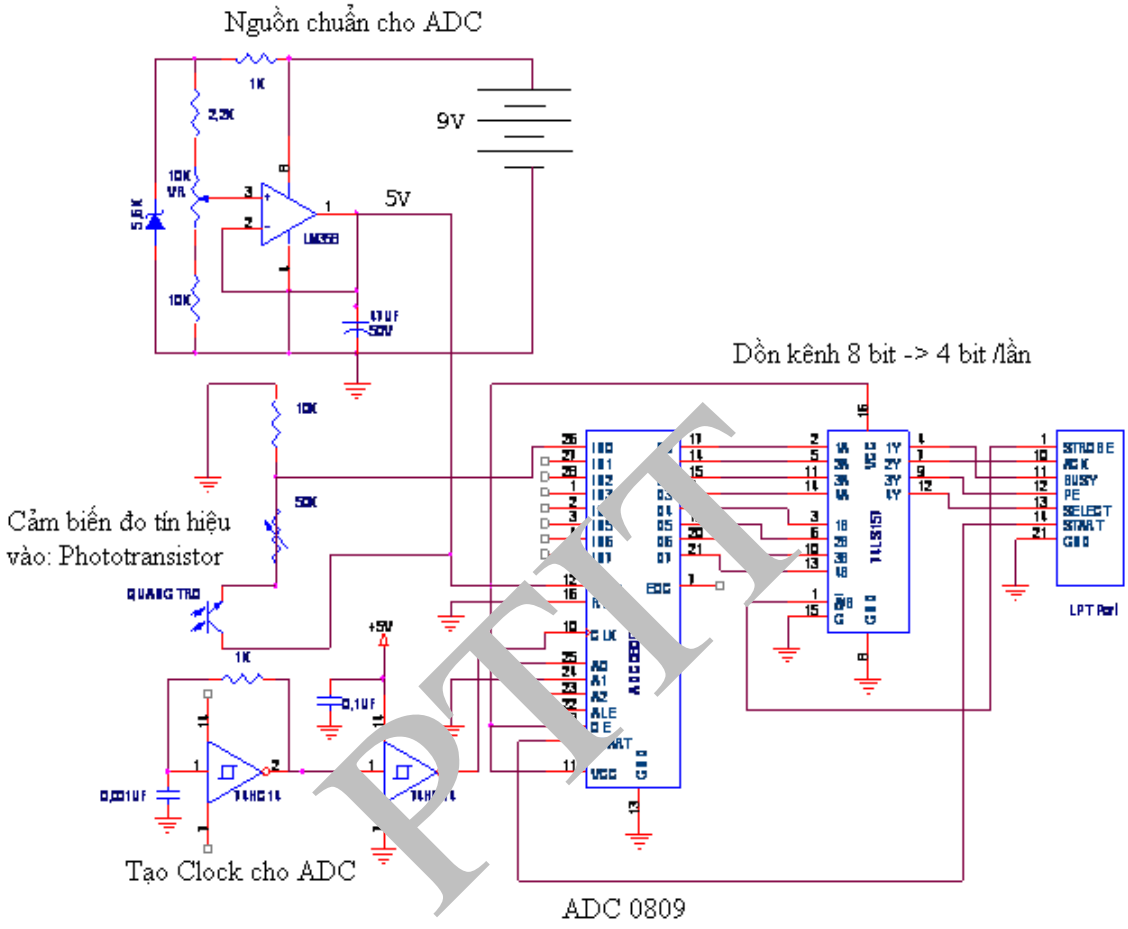
2. Thế nào là một vi mạch có 3 trạng thái và 3 trạng thái là những trạng thái nào ? Khi nào phải sử dụng vi mạch 3 trang thái ?
3. Thế nào là cổng (port), cổng dùng để là gì ?
4. Là thế nào để truy nhập vào cổng ?
5. Thế nào là "memory mapped I/O" ?
6. Thế nào là "I/O mapped I/O" ?
7. CPU thông thương có một tập các tín hiệu phát ra ở đầu ra theo kiểu dồn kênh, ví dụ các đương AD7-AD0 của CPU 8080/8085. Làm thế nào để tách các đường dây A7-A0 và các đường dây D7-D0 ? Thử thiết kế bài tập này ?
8. BUS của CPU và BUS hệ thống có khác nhau ? Tại sao ?
9. BUS mở rộng và BUS hệ thống có khác nhau ? Tạo sao ?
10. Khi thiết kế HNT, có cần tới BUS mở rộng và các chế độ hoạt động của BUS mở rộng ? Tạo sao ?
11. Mô tả mục đích của bảng vector ngắt.
12. Thế nào là ghép nối vào/ra không điều kiện và có điều kiện ?
13. Khi nào thi chọn giải pháp ghép nối không điều kiện ?
14. Khi nào thi chọn giải pháp ghép nối có điều kiện ?
15. Có những phương pháp ghép nối nào ? Mô tả nguyên tắc hoạt động của từng phương pháp.
16. Tại sao truyền cữ liệu bằng DMA lại nhanh hơn kiểu điều khiển bằng chương trình ?
17. Ưu và nhược điểm của của hai phương pháp vào/ra điều khiển bằng chương trình và DMA ?
18. Ngắt là gì ? Khi nào thì dùng ngắt ?
19. Có những kiểu ngắt nào, phạm trù sử dụng của các kiểu ngắt đó ?
20. Cho ví dụ về các thiết bị mà khi ghép nối sử dụng ngắt là hợp lí hơn các kiểu ghép nối khác.

2.8.2 Bài tập cuối chương

1. Thiết kế bo mạch với CPU 8085 với 4KB EPROM chứa nhân HTN, 8KB RAM cho dữ liệu, mỗi vi mạch RAM có dung lượng 8KB x 4 bit. Địa chỉ đầu cho EPROM=0000Hex, địa chỉ đầu cho RAM=8000Hex. Địa chỉ đầu vào cho vi mạch giải mã là các địa chỉ là ? (Bài giải ở PHỤ LỤC).
2. Dưới đây là sơ đồ thiết kế ghép nối một module với ADC vào PC qua cổng máy in. Tìm hiểu xem cách ghép nối như vậy có chạy không ? Thử viết chương trình điều khiển đọc dữ liệu tuwd ADC vào máy tính.

   Các vi mạch sử dụng:
   - Nguồn chuẩn LM 358

- Tạo dao động 74LS14, R=1K, C=0,001 µF cho f=700KHz (theo tính toán của ADC 0809)
- Dồn kênh 74LS157



Hình 2.88 Bài tập thiết kế ghép nối ADC, cổng LPT vào máy tính PC

3) Ghép nối với cổng khả trình 8255. Sử dụng ISA BUS trên PC.

Mục đích bài tập: sử dụng vi mạch đa năng 8255, ghép nối với BUS hệ thống mở rộng ISA:

-Sử dụng địa chỉ IO cho Prototype 300Hex- để thục hiện ghép nối vi mạch 8255 với PC qua BUS ISA. Cổng 301 Đọc trạng thái của 8255, Cổng 303 gởi lệnh cho 8255, mode 0.

-Sử dụng cổng của 8255 nối tới hiển thị là LED 7 thanh

- Viết chương trình thực hiện ghép nối, đưa ra các số làm cho đèn sáng.

Chương 3. CÁC THÀNH PHẦN PHẦN MỀM CỦA HỆ THỐNG NHÚNG

Chương 2 đã đề cập tới phần cứng của các hệ thống máy tính nói chung mà HTN cũng là một trong các hệ thống đó. Sự khác biệt phần cứng của HTN là để phù hợp với những yêu cầu đặt ra trên một HTN cũng đã được đề cập. Chương 3 sẽ xem xét đến phần mềm được triển khai trên HTN. Như trong mô hình kiến trúc trừu tượng đã nói ở chương 1, hình dưới đây sẽ cho thấy hai lớp (con) mới sẽ đưa thêm vào mô hình đó. Khi đề cập tới phần mềm, tổng quát, ta có thể chia ra làm hai lớp: phần mềm hệ thống và phần mềm ứng dụng. Phần mềm ứng dụng là các phần mềm ứng dụng nhúng, các phần mềm này định nghĩa chức năng cũng như mục đích hình thành một HTN cụ thể. Phần mềm hệ thống là phần mềm có chức năng quản lí hoạt động của phần cứng, cung cấp nguồn tài nguyên phần cứng và phần mềm trung gian khác cho phần mềm ứng dụng, thực thi mã phần mềm ứng dụng sao cho hiệu quả và ổn định. Tùy thuộc vào yêu cầu vận hành của một HTN, phần mềm hệ thống có thể đơn giản nhưng cũng có thể rất tinh xảo. Ví dụ rất phổ biến là phần mềm hỗ trợ đa nhiệm, và nếu chịu sự ràng buộc về thời gian xử lý, thì phải có tính thời gian thực, và phần mềm hệ thống sẽ là một *hệ điều hành thời gian thực* (*RTOS*). Còn có những phần mềm khác tạo ra liên kết giữa các lớp phần mềm, ví dụ để phần mềm hệ thống khai thác phần cứng, cần có một "phần mềm sụn" gọi là BIOS (*Basic Input Output System*), mà thực tế đó là tập hợp các module phần mềm điều khiển các phần cứng. Các module này còn có tên là các *module chương trình điều khiển thiết bị* (*aevice driver*). Có thể miêu tả vai trò này như sau: phần mềm ứng dụng có nhu cầu trao đổi dữ liệu với thiết bị bằng cách thực hiện *gọi* chức năng vào/ra của phần mềm hệ thống. Phần mềm hệ thống chuẩn bị các thông số vào/ra và gởi cho các device driver. Các device driver nhận các thông số đó ở đầu vào, và đầu ra sẽ là các lệnh điều khiển thiết bị cụ thể. Dữ liệu của quá trình này là kết quả mà thiết bị đã thi hành và trao đổi cho phần mềm hệ thống qua các phương thức ghép nối đã trình bày ở chương 2, để ở các vùng nhớ bộ đệm dữ liệu của phần mềm hệ thống. Sau đó phần mềm hệ thống sẽ chuyển dữ liệu cho phần mềm ứng dụng. Giữa phần mềm hệ thống và phần mềm ứng dụng còn có một cơ chế "liên lạc" gọi là *gọi hệ thống (systen call),* mà thực chất là phương thức mà phần mềm hệ thống hỗ trợ để phần mềm ứng dụng khai thác các chức năng của phần mềm hệ thống.

Đó cũng là chủ đề mà chương 3 sẽ đề cập.

3.1 TRÌNH ĐIỀU KHIỂN THIẾT BỊ (viết tắt: TĐKTB)

3.1.1 Tổng quan

Có thể nêu định nghĩa về *trình điều khiển thiết bị (TĐKTB)* như sau: là một phần mềm để khởi động phần cứng và phần mềm lớp cao hơn sử dụng để quản trị truy nhập vào phần cứng ghép nối vào máy tính. Phần mềm này tương tác trực tiếp và điều khiển phần cứng và được tổ chức ở dạng các thư viện phần mềm. Khi máy tính có hệ điều hành thì TĐKTB là cầu nối giữa phần cứng và hệ điều hành. Thông thường có nhiều thiết bị chuẩn ghép nối vào máy tính, ví dụ: đĩa cứng, đĩa mềm, CD, DVD bàn phím, màn hình, vĩ mạng … Vì là các thiết bị chuẩn, nên khi thiết kế hệ

điều hành (HĐH) thương mại (WINDOWS, UNIX, Linux. Mac OS, …), các TĐKTB đã được nhúng trong hệ điều hành như các thư viện. Tuy nhiên khi đưa thêm một thiết bị mới vào máy tính, mà HĐH không hỗ trợ, thì phải viết mã cho TĐKTB, lúc đó ta nói là viết trình điều khiển thiết bị, công việc sẽ làm là chuyển đổi các lệnh vào ra tổng quát của HĐH thành các chỉ thị dạng thông điệp sao cho các thiết bị có thể hiểu được và thực hiện các chỉ thị đó. HĐH thường không quản lí trực tiếp các thiết bị, mà thông qua một trình điều khiển thiết bị ảo hóa (*virtual device drivers*), và HĐH hành nhìn các thiết bị như là một lớp phần cứng trừu tượng (*hardware abstract layer – HAL*), còn các thiết bị thì được ảo hóa (*virtual device driver*) như vậy HĐH sẽ độc lập (tương đối) với các phần cứng. Các TĐKTB ảo sẽ xử lý các ngắt mềm từ HĐH thay vì xử lý các ngắt cứng từ các thiết bị cứng, vì TĐKTB ảo sẽ duy trì trạng thái của phần cứng và các thiết đặc của thiết bị lại có thể thay đổi được (*programmable settings*). Chúng ta đã đề cập tới ngắt cứng và ngắt mềm và tổ chức của vector ngắt của CPU. HĐH tận dụng cơ chế này của CPU để quan lí thiết bị qua TĐKTB ảo. Có thể chứng nghiệm điều này khi cài HĐH WINDOWS của Microsoft vào thời điểm nạp HAL để phát hiện các thiết bị ghép nối vào máy tính.

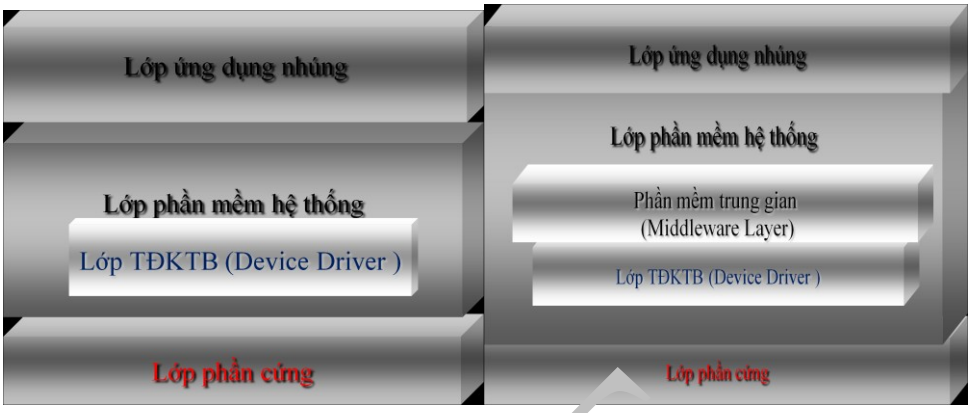
Mô hình chung kiến trúc các phần mềm máy tính như hình 3.1

Hình 3.1 Mô hình tổng quát các phần mềm trên máy tính

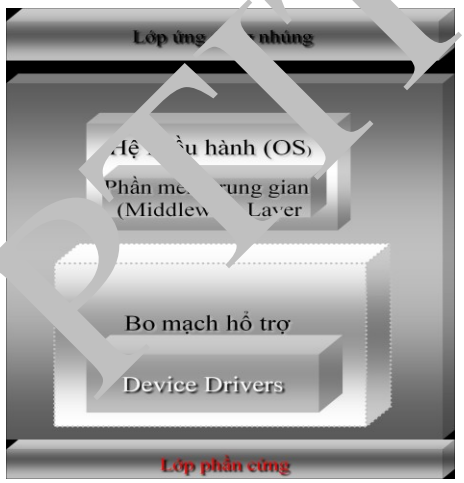
Trong đó phần mềm hệ thống đặt ở giữa phần mềm ứng dụng và phần cứng. Hãy xuất phát từ một ứng dụng: khi ứng dụng muốn sử dụng phần cứng của máy tính (truy nhập ROM, RAM, thiết bị ngoài như đĩa cứng, mạng …), ứng dụng sẽ chuyển yêu cầu đó theo một phương thức qui ước *gọi hệ thống (system calls-GHT).* GHT là một giao diện lập trình mà ứng dụng sử dụng để khai thác các dịch vụ của phần mềm hệ thống. Đến lượt mình, phần mềm hệ thống sẽ gởi lệnh tới

cho một nhóm các phần mềm điều khiển các thiết bị, đó là BIOS. BIOS sau đó ra lệnh để thiết bị thục hiện các vào/ra dữ liệu.

Khi triển khai mô hình trên, tùy thuộc vào kiến trúc tổng thể các phần mềm trên được triển khai theo một số cấu hình như sau:



a. b.



c.

Hình 3.2 Mô hình tổng quát các kiểu sắp xếp phần mềm trên máy tính

Hình 3.2 a. là cấu hình dạng đơn giản nhất. Các thiết bị nhúng với những chức năng điều khiển kiểu ON/OFF có thể sử dụng. Hình 3.2 *b.* là cấu hình nâng cao với phần mềm trung gian thực hiện các xử lý phức tạp hơn, theo các ưu tiên, đa xử lý hạn chế. Trong khi ở hình 3.2 *c.* Là các hệ thống tinh xảo hơn, việc quản lí ứng dụng và quản lí phần cứng cần có hệ điều hành, có phần mềm trung gian. Hệ có thể mở rộng bằng các bo mạch có TĐKTB nhúng trên bo và kết hợp với hệ điều hành qua cơ chế HAL.

Các TĐKTB trong HTN thường xếp vào hai lớp:

- Lớp xác định theo kiến trúc (*architecture-specific*) của HTN, các TĐKTB ở lớp này quản trị phần cứng hợp nhất với CPU. Các kiến trúc kiểu microcontroller, hay kiến trúc Havard với bộ nhớ trên chip, cổng, vi mạch quản trị bộ nhớ (*memory management Unit-MMU*), các phần cứng dấu phẩy động, các ADC/DAC hợp nhất trên bo mạch, thuộc lớp này.
- Lớp TĐKTB tổng quát (*generic*) điều khiển các thiết bị nằm trên bo mạch chính nhưng không hợp nhất vào chip với CPU. Tuy nhiên mã thực thi sẽ có một phần của lớp *architecture-specific* vì khi truy nhập bất kì thiết bị nào đều phải chạy qua CPU. Tuy nhiên TĐKTB lớp này còn dùng để điều khiển các thiết bị không nhất thiết xác định cho CPU cụ thể. Nói cách khác TĐKTB có thể làm cấu hình để hoạt động với các loại kiến trúc khác nhau mà lại có các phần cứng mà TĐKTB đã viết để điều khiển các phần cứng đó. Ví dụ các TĐKTB khởi động và truy nhập vào các thành phần của bo mạch như: BUS Inter-Integrated Circiut (I2C), PCI, PCMCIA, cache L2, Flash …

Bỏ qua kiểu TĐKTB, bỏ qua kiểu thiết bị, ta đưa ra đây các chức năng cơ bản của TĐKTB:

1) *Khởi động phần cứng (Hardware Startup):* Khởi động các phần cứng sau khi bật nguồn hay sau khi RESET máy. Phần lớn các phần cứng là các vi mạch ghép nối hay điều khiển các thiết bị và là khả trình. Cho nên bước này là lập trình cho các vi mạch chức năng của bo mạch, để đưa các vi mạch vào trạng thái đầu tiên: trạng thái sẵn sàng thực hiện các lệnh tiếp theo.
2) *Tắt máy (Hardware Shutdown):* Ngược lại với khởi động, điều khiển tắt máy sẽ cấm tuần tự các vi mạch, kết thúc kết nối với các thiết bị sao cho dữ liệu không bị xáo trộn hay bị hỏng. Cuối cùng là cắt nguồn điện nuôi máy.
3) *Cấm phần cứng hoạt động (Hardware Disable):* Cho phép phần mềm khác có thể tạm thời vô hiệu hóa một phần cứng khác.
4) *Cho phép hoạt động (Hardware Enable):* Cho phép phần mềm khác kích hoạt phần cứng.
5) *Dành lấy phần cứng (Hardware Acquire):* Cho phép phần mềm khác chiếm lấy phần cứng bằng cách khoá phần cứng đối với các phần mềm khác. Ví dụ sau khi HĐH giải quyết tranh chấp phần cứng, gán phần cứng cho một tiến trình tạm thời độc quyền sử dụng, các tiến trình khác phải chờ.
6) *Giải phóng phần cứng (Hardware Release):*Cho phép phần mềm khác giải khóa phần cứng.
7) *Đọc/ghi dữ liệu (Hardware Read/write):* Cho phép phần mềm khác thực hiện trao đổi dữ liệu với phần cứng.

8) *Cài và tháo dỡ phần cứng (Hardware Install/Uninstall):* Cho phép phần mềm khác cài mới hay tháo dỡ phần cứng.

TĐKTB còn có thể các chức năng khác hay có thể không nhất thiết phải có tất cả các chức năng này. Các phần cứng khi đã kích hoạt đều có ít nhất ba trạng thái mà TĐKTB phải giám sát: Không hoạt động (*inactive, idle*), đang hoạt động (*in active-busy*), kết thúc hoạt động (*finish*).

Vậy một TĐKTB chạy trong một hệ thống như thế nào ? Mỗi TĐKTB đều nhắm vào một thiết bị cụ thể, nhưng được hợp nhất với phần mềm hệ thống. Phần mềm ứng dụng tiếp cận TĐKTB qua giao diện lập trình GHT như đã nêu. Cách khai thác TĐKTB có khác nhau theo cách mà một hệ thống được xây dựng và có hay không có hệ điều hành, hệ hành là đơn nhiệm hay đa nhiệm. Ví dụ với MS DOS các phần mềm ứng dụng truy nhập TĐKTB dễ dàng và trực tiếp. Tuy nhiên với hệ điều hành đa nhiệm thì khác. Ở các hệ điều hành này có hai chế độ thực thi: *supervisor* hay *user*. Các chế độ này khác nhau ở chỗ các thành phần phần cứng nào của hệ thống, một phần mềm có thể truy nhập vào được. Chế độ *supervisor* cho phép phần mềm truy nhập không hạn chế phần cứng, tức tài nguyên hệ thống, trong khi ở chế độ *user* thì không. Các TĐKTB thường chạy trong chế độ *supervisor*. Lí do: đa nhiệm là tranh chấp tài nguyên, giải quyết tranh chấp cần giải pháp phần mềm hệ thống.

3.1.2 Các loại TĐKTB

Các thiết bị ghép nối vào máy tính nói chung hay HTN nói riêng có thể rất đa dạng, tuy nhiên có thể xếp lại như sau:

- Thiết bị cung cấp khối dữ liệu (block device drivers), do đó khi thực hiện vào/ra với loại này TĐKTB cần có đủ bộ nhớ đệm (buffer) cho mỗi lần thực hiện. Kích thước của khối có thể lớn hay nhỏ phụ thuộc vào khả năng của thiết bị. Như chương 2 đã đề cập ghép nối với thiết bị loại này sử dụng phương thức DMA. TĐKTB sẽ bao gồm lập trình xin cấp buffer trong RAM, gởi các thông số điều khiển tới DMAC, cho phép DMAC nhận yêu cầu DMA. Với HNT, các thiết bị loại này, ví dụ máy MP3, Camcorder, TV, dữ liệu từ các cảm biến radar v.v.
- Thiết bị xử lý kí tự (character device drivers): không cần sử dụng buffer lớn, thường thao tác 1 kí tự 1 lần. Một kí tự thường có độ dài 8 bit. Khi trao đổi dữ liệu có thể không giới hạn số byte cho mỗi lần trao đổi vào ra. Chương 2 đã đề cập tới các cách ghép nối và những nét chính để viết TĐKTB: có/không có điều kiện, có kiểm tra trạng thái hay không kiểm tra trạng thái thiết bị, cách thoát ra khỏi chương trình tránh bị quẩn khi thiết bị không làm việc do sự cố … Thiết bị loại này được sử dụng phổ biến trong HTN, ví dụ các cảm biến qua ADC/DAC gởi dữ liệu vào RAM qua các cổng. Đôi khi cũng sử dụng cổng truyền nối tiếp để nhận hay gởi dữ liệu tới các thiết bị để ở xa bo mạch, cổng lập trình và debug lỗi SPI.

- Thiết bị mạng (Network device drivers): Thiết bị mạng nhận từ đường truyền từng bit, chuyển thành byte để ở buffer vỉ mạng sau đó chuyển vào RAM bằng ngắt và DMA. TĐKTB mạng nằm ở lớp 2 của mô hình mạng OSI.

3.1.3 3.1.3 Hoạt động của TĐKTB

Khi TĐKTB được kích hoạt, một loạt các thao tác sẽ được thực hiện. Nhìn ở góc độ khi TĐKTB thuốc quản lí của hệ điều hành, các thao tác đó như sau:

- Tự cấu hình (Autoconfiguration): Vào thời điểm nhân HĐH gọi TĐHTB để xác định thiết bị kết nối với cổng (probe interface), lập các thông số phù hợp và khởi động thiết bị.
- Thao tác vào/ra (I/O operations): TĐKTB sẽ kiểm soát quá trình vao/ra dữ liệu giữa RAM và thiết bị, Các thao tác bao gồm: mở cổng với thiết bị, thực hiện ghi/đọc dũ liệu và đóng thiết bị.
- Xử lý ngắt (Interrupt handling): Khi có ngắt được chấp nhận, CPU xác định số hiệu của vector ngắt, gọi ISR tương ứng. Các bước chi tiết đã trình bày ở chương 2.
- Những yêu cầu đặc biệt (Special requests): Có nhiều trường hợp cần tới các xử lý đặt biệt, thì các TĐKTB cũng là dạng đặc biệt, ví dụ các ngắt không trong tổ chức vector mà là các ngắt không thể che với những mức ưu tiên cao, xử lý các sự kiện đặc biệt xảy ra trong hệ thống.
- Tái khởi động (Reinitialization): Có những thành phần theo chu kì phải tái khởi động đưa thiết bị về trạng thái ban đầu. Ví dụ "watchdog" timer, hay nếu thiết bị có vấn đề sau thời gian thử truy nhập (timeout) cần tái khởi động thiết bị đẻ đưa thiết bị về trạng thái ban đầu. Thực hiện thao tác này có thể tự động bởi phần mềm hệ thống, cũng có thể cho phép thực hiện bằng tay.

3.1.4 Phát triển TĐKTB

Các công việc cần làm để thiết kế TĐKTB:

- Thu thập thông tin:
  - ✓ Các thông tin về máy tính (host) sẽ cài TĐKTB lên đó: loại CPU, loại HĐH, kiến trúc BUS;
  - ✓ Các qui định được dùng để viết TĐKTB: Cách đặt tên cho thiết bị, mô tả thiiest bị, tài liệu.
  - ✓ Xác định các thuộc tính của thiết bị: thiết bị là kiểu khối (block device) hay kiểu kí tự (character device), thiết bị có hỗ trợ cấu trúc hệ thống tệp (đĩa cứng, USB, …), hổ trọ chuỗi (byte stream), có khả năng phản ứng với ngắt (cấm ngắt đệ qui khi đang trao đổi khối lượng lớn dữ liệu, khi kéo dài chu kì ngắt không cấm ngắt

tiếp theo, thao tác hàng đợi dữ liệu có thể bị ngắt hay ngắt) , khởi động lại (reset) như thế nào.

- ✓ Mô tả về thiết bị: kiểu thiết bị, có bao nhiêu thiết bị cùng kiểu có thể cài đặt trên hệ thống, mục đích của thiết bị.

- Thiết kế TĐKTB
  - ✓ Kiểu thiết bị: kí tự (Character), Khối (Block), Cả hai: khối và kí tự, Mạng (Network).
  - ✓ Đầu vào tiếp cận thiết bị: là giao diện ghép nối thiết bị: địa chỉ cổng sử dụng, cách nhận biết thiết bị có trên hệ thống (thăm dò-probe, quét-scan) đẻ ghi nhận.
- Hiểu cấu trúc dữ liệu hệ thống và kiểu địa chỉ gán cho thiết bị
  - ✓ Kiểu dữ liệu thao tác: Phụ thuộc vào kiểu thiết bị, cách ghép vào BUS hệ thống, dữ liệu song song, song song máy bit, nói tiếp, biểu diễn kiểu nhị phân, hexa ,…
  - ✓ Địa chỉ ảo không ánh xạ: là địa chỉ ảo trong không gian HĐH đa nhiệm. Khi dùng với HĐH đa nhiệm sẽ có loại địa chỉ này nhân HĐH dung cơ chế này để dăng kí ánh xạ địa chỉ với đí chỉ ảo.
  - ✓ Địa chỉ vật lí của nhân HĐH: là đi chỉ thực trong không gian địa chỉ của nhần HĐH, dùng khi HĐH là đa nhiệmKernel physical address .
  - ✓ Địa chỉ vật lí của BUS: Có một số Bus đối với thiết bị có thể địa chỉ hóa (Xem chương 2 về BUS), BUS này thiết bị sử dụng để chuyển dữ liệu tới thiết bị khác. ví dụ SCSI BUS và ID của thiết bị nối vào BUS, max=8 hay 16 thiết bị, hay IDE BUS có IDE#1 cho 2 thiết bị, ID#2 cho 2 thiết bị.
  - ✓ Địa chỉ RAM nới sẽ thực hiện trao đổi vào/ra dữ liệu.
  - ✓ Địa chỉ thiết bị hay địa chỉ I/O: Lá địa chỉ cổng ghép nối tới thiết bị.
  - ✓ Xác định phương pháp đăng kí ngắt của thiết bị với module quản lí ngắt (interrupt handler): thực hiện qua một gọi hệ thống tới trình quản lí ngắt hay nhảy trực tiếp vào bảng vector ngắt. Trình quản lí ngắt có hai đầu vào với cấu trúc: *ihandler_t* và *handler_intr_info.*
- Cách sắp đặt cấu trúc dữ liệu của TĐKTB: Khi khởi động hệ thống các cấu trú dữ liệu về thiết bị sẽ được cài vào vị trí nào, con trỏ trỏ tới ISR để ở đâu (qua interrupt handler hay bảng vector ngắt).
- Tạo môi trường phát triển TĐKTB: Khi đi vào phát triển thường sử dụng các bộ kit cho môi trường HĐH triển khai trên HTN. Công cụ sẽ yêu cầu cài đặt với tổ chức thư mục để biên dịch, tạo mã thực thi và sau đó là hợp nhất vào HĐH, và nạp xuống HTN đang phát triển. Ví dụ của hãng KIEL Software có công cụ chip Chip 8051/80251. Tất cả các hãng sản xuất Chip CPU cho HTN đều có DDK (Device Driver Development Kit) đi cùng.

3.1.5 Một số ví dụ về TĐKTB

- TĐKTB điều khiển ngắt:

  Ngắt là cơ chế phổ biến được sử dụng. Chương 2 đã có mô tả về cách thức ngắt xuất hiện và các bước xử lý ngắt. Dưới đây sẽ nêu ra một số nét chính khi thiết kế TDDKTB cho ngắt. Đầu chương ta đã liệt kê tám chức năng của TĐKTB, trong đó có ít nhất 4 chức năng kết hợp với mô tả khi xây dựng TĐKTB ngắt:

  1) Khởi động (start) phần cứng quản lí ngắt : gởi các lệnh điều khiển tới điều khiển ngắt (ví dụ vi mạch 8259), cài đặt bảng vector ngắt, kích hoạt (cho phép nhận ngắt) … sau khi bật máy (POST).
  2) Tắt (shutdown) điều khiển ngắt: chuyển điều khiển ngắt về trạng thái không làm việc (idle).
  3) Cho phép phần mềm khác tạm thời vô hiệu các ngắt: đưa điều khiển ngắt vào chế độ vô hiệu (disable).
  4) Cho phép phần mềm khác khôi phục lại hoạt động của điều khiển ngắt (enable).
  5) Kích hoạt ISR: CPU xác định số vector ngắt, tìm tới địa chỉ của ISR của ngắt và kích hoạt xử lý ngắt đó. Khi viết mã thực thi của ISR rất thận trọng, dựa vào phân tích cách quản lí các sự kiện mà hệ thống hoài vọng: sự xuất hiện, mức ưu tiên, cho/cấm ngắt đệ qui. Mã này thuộc loại tới hạn, phải rất tối ưu, chạy hiệu quả.

  Tất cả các chức năng trên khi chuyển thành mã chạy dự vào qui tắc sau:

  1) Kiểu, số hiệu, mức ưu tiên của ngắt: xác định trên CPU, vi mạch quản lí ngắt trên bo mạch.
  2) Ngắt cứng: Tín hiệu ngắt xuất hiện như thế nào: bằng sườn của tín hiệu (độ nhạy cao) hay bằng mức của tín hiệu (độ nhạy thấp hơn).
  3) Chiến lược tổ chức ngắt: danh sách các sự kiện mà hệ thống sẽ xử lý và được phân loại để hệ thống có đáp ứng phù hợp.

- TĐKTB điều khiển BUS

  Mối BUS hệ thống trên các hệ phức tạp đều có một qui ước hoạt động, goi là giao thức (BUS protocol), như đã đề cập trong chương2. Protocol cho biết: thiết bị sẽ có khả năng truy nhập vào BUS như thế nào (qua trọng tài BUS), các qui định thực hiện "bắt tay" khi các thiết bị trao đổi dữ liệu, các tín hiệu điều khiển sử dụng trên mỗi BUS sẽ được sử dụng thế nào cho qui trình " bắt tay" và trao đổi dữ liệu. Các chức năng sau đây phải được cài vào TĐKTB khi thiết bị sử dụng BUS:

  1) Khởi động BUS khi bật nguồn.
  2) Tắt BUS khi tắt nguồn.
  3) Treo BUS (disable): cho phép phần mềm khác treo BUS.

4) Kích hoạt BUS trở lại (enable): cho phép phần mềm khác dành lại BUS và đưa BUS vào hoạt động.
5) Khóa BUS: cho phép phần mềm khác tạm thời độc quyền BUS bằng chế độ khóa (lock) BUS.
6) Giải khóa BUS: cho phép phần mềm khác giải khóa (unlock) BUS.
7) Đọc/ghi: cho phép phần mềm khác đọc dữ liệu trêm BUS, hay đưa dữ liệu ra BUS.
8) Cài đặt hay tháo cài đặt BUS: cho phép phần mềm khác cài/tháo một thiết bị mới nhằm mở rộng BUS.

- TĐKTB lại bo mạch cắm vào khe I/O hệ thống
  Khi mở rộng chức năng hệ thống thường cần tới bo mạch đã thiết kế cho chức năng mới. Thông thường cắm bo mạch vào BUS hệ thống sẽ hình thành một cơ chế kiểu chủ (master) và thứ cấp (slave). Giao tiếp bo cắm thêm sẽ được kiểm soát bởi vi mạch ghép nối-điều khiển trên bo (slave). Trong vi mạch sẽ có các thanh ghi điều khiển, thanh ghi trạng thái sao cho master có thể truy nhập và kiểm tra hoạt động của bo mạch cắm thêm. TĐKTB cho bo mạch sé có các chức năng sau :

  1) Khởi động I/O: Khởi động bo cắm thêm sau khi lấy nguồn từ hệ thống.
  2) Tắt I/O: tắt hoạt động I/O trên bo mạch cắm thêm khi tắt nguồn.
  3) Vô hiệu I/O bo mạch cắm thêm: cho phép phần mềm khác vô hiệu I/O.
  4) Kích hoạt bo mạch : cho phép phần mềm khác kích hoạt I/O trở lại.
  5) Khóa/giải khóa: cho phép một phần mềm khác khóa bo mạch, hay cho phép phần mềm khác kích hoạt bo mạch trở lại.
  6) Đọc/ghi bo: cho phép phần mềm khác thực hiện đọc/ghi dữ liệu với bo mạch.

Trong chương 2 có ví dụ về TĐKTB với lưu đồ điều khiển ghép nối có điều kiện, đó cũng là một lõi để viết TĐKTB.

Có thể tham khảo với MS OS tại :
How to Write a Windows Driver.
http://msdn.microsoft.com/en-us/library/ms809956.aspx
1.Cài DDK
2. http://www.adp-gmbh.ch/win/misc/writing_devicedriver.html

3.2 HỆ THỐNG NHÚNG THỜI GIAN THỰC

Trước khi đi vào mô tả về HTN thời gian thực, cần trở lại một số khái niệm liên quan tới hệ điều hành cho máy tính. Một hệ thống tính toán hay một thiết bị kĩ thuật số luôn cần có phần mềm hệ thống. Phần mềm hệ thống đảm nhiệm chức năng quản lí tài nguyên như phần cứng máy tính (ROM, RAM, thiết bị lưu trữ thông tin, các thiết bị ngoại vi …), tệp … thực hiện giao tiếp người máy, khởi động, cung cấp tài nguyên và điều phối các chương trình ứng dụng, v.v. Mức độ phức tạp của phần mềm hệ thống có thể khác nhau từ đơn giản đến tinh xảo. Các HTN cũng không phải là ngoại lệ, có HTN đơn giản có HTN phức tạp, đối với các HTN phức tạp khi phải thực hiện nhiều ứng dụng đồng thời, thì cần có hệ điiều hành đa nhiệm hỗ trợ.

3.2.1 Hệ điều hành đa nhiệm (multitasking)

Phần lớn HTN được sử dụng trong môi trường điều khiển công nghệ, tại đó tùy vào vị trí trong toàn bộ dây chuyền, HTN có thể là các loại với qui mô kiến trúc khác nhau. Tuy nhiên phần mềm hệ thống đòi hỏi tương đối khắt khe và loại HĐH sử dụng phổ biến là *hệ điều hành thời gian thực (Real Time Operating System – RTOS), đa tác vụ*. Khi thảo luận về RTOS là một đề tài lớn và phức tạp, cho nên chương chỉ sẽ trích những điểm cần thiết cần quan tâm khi phát triển RTOS trên HTN.

Khi đề cập tới HĐH kiểu đa tác vụ, cần nhắc lại một số khái niệm sau đây: **, Nhân HĐH, Tiến trình, chế độ hoạt động, chức năng, ngắt, dịch vụ, cờ hiệu (semaphorre), ngắt, ngoại lê …**

- **Nhân HĐH (kernel):** Nhân HĐH thực hiện chức năng chính của HĐH, gồm:
  - ✓ Quản lí các tiến trình: tạo lập các tiến trình, cung cấp nguồn tài nguyên cần thiết để tiến trình hoạt động, giám sát thực thi tiến trình trong môi trường đa nhiệm.
  - ✓ Quản lí tài nguyên: Các tài nguyên máy tính bao gồm: CPU, RAM, thiết bị I/O, hệ thống tệp (File system) …. Cung cấp các tài nguyên khi tạo tiến trình và khi tiến trình đòi hỏi, giải quyết các tranh chấp tài nguyên, thu hồi tài nguyên khi tiến trình kết thúc thực thi. Cung cấp thư viện các hàm chức năng của nhân cho ứng dụng qua API gọi hệ thống (GHT).
  - ✓ Để tự bảo vệ, nhân chạy trong một chế độ riêng (*supervisor/protected*). Chế độ này được hỗ trợ thư kiến trúc của CPU. Nhìn từ HĐH thì gọi là **chế độ nhân** (*kernel mode*).
- **Tiến trình (Process)**:

Trên một hệ đa nhiệm, không chỉ một chương trình chạy trên máy và phương thức phổ biến để nhiều chương trình có thể chạy đồng thời là kĩ thuật phân chia thời gian sử dụng CPU cho mỗi chương trình (*time sharing*). Như vậy mỗi chương trình không chạy liên tục mà có những thời điểm chương trình "dừng". Chương trình " dừng" vì khung thời gian sử dụng CPU cho nó đã hết, hay khi tài nguyên nó yêu cầu chưa được đáp ứng, hay bị một xử lí khác có xu hướng chen ngang sử dụng CPU, … Để biểu diễn sự vận động như vậy, cần có một khái niệm, gọi là *tiến*

trình. Về ngữ nghĩa là nói đến *quá trình tiến triển (thực hiện) chương trình theo thời gian* trong khi chương trình rơi vào những hoàn cảnh khác nhau, trong mỗi hoàn cảnh như vậy gọi là trạng thái của tiến trình.

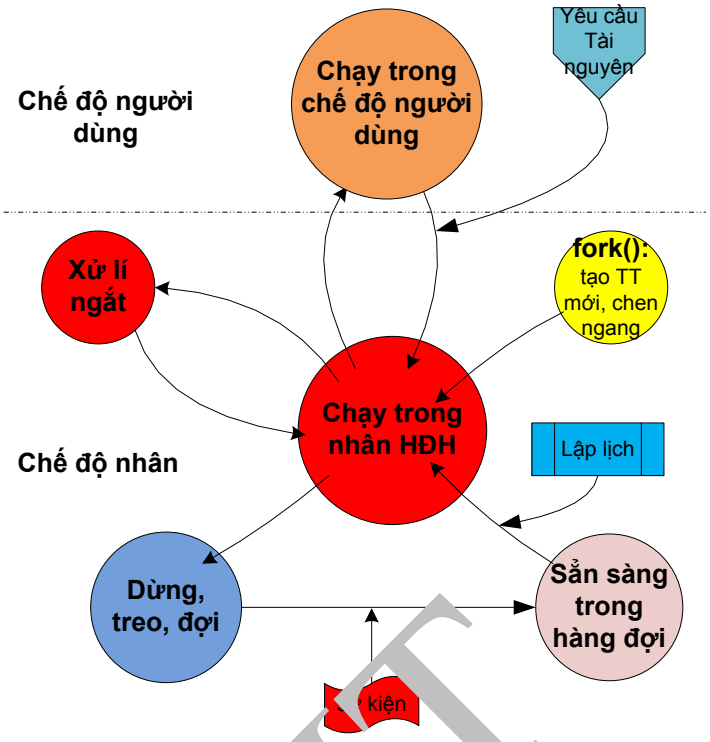
Vậy *tiến trình* (TT) là thời điểm thực hiện của một chương trình (*instance of execution*) và việc thực hiện đó chỉ xảy ra trong một khoản thời gian nhất định (*slice time*). Nhìn ở khía cạnh quản lí của HĐH, thì tiến trình do HĐH tạo ra để thâu gói tất cả các thông tin liên qua tới việc thực hiện chương trình. Như vậy để thực hiện được chương trình, TT sẽ sử dụng CPU để chạy các lệnh của nó, và bộ nhớ nơi có mã lệnh (*code* hay *text*), dữ liệu (*data*), và ngăn xếp (*stack)* của chương trình. Một TT khi thực hiện phải làm theo một trình tự các lệnh trong vùng *code* của TT. TT chỉ có thể đọc/ghi truy nhập *data* và *stack* của nó, nhưng không thể trên *data* và *stack* của TT khác. TT liên lạc với các TT khác và phần còn lại của hệ thống bằng các Goị Hệ Thống (GHT, *system call*) và cơ chế liên lạc giữa các tiến trình (IPC).

Mô hình về quá trình tiến triển chạy chương trình gọi là mô hình trạng thái.

Về cơ bản có thể đưa ra ba trạng thái chính:

1. TT trình *chạy* (*run*): đó là khi các lệnh của chương trình được thực hiện
2. TT *dừng* (*suspended, wait*): do hết thời lượng được phân phối, do tài nguyên chưa đáp ứng, do bị chen ngang bởi các xử lí khác, chờ sự kiện.
3. TT *sẵn sàng* (*ready*): tài nguyên đã có, đã đến lượt chạy lại TT chuyên vào hàng đợi chờ kích hoạt bởi hệ lập lịch.

Khi phát triển hệ điều hành số trạng thái có thể nhiều hơn và chi tiết hơn.



Hình 3.3 Trạng thái của tiến trình

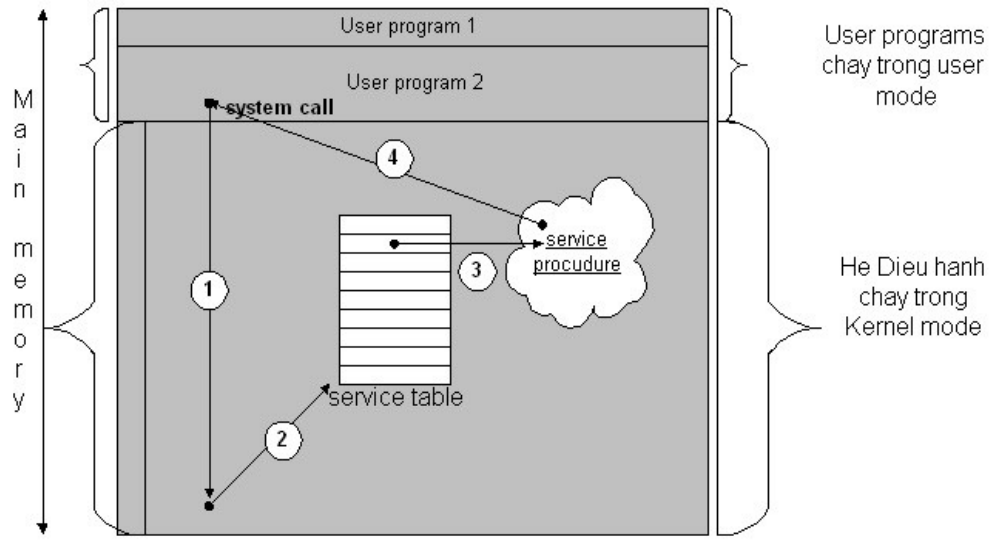
- ✓ Tiến trình ***chạy trong chế độ người dùng***: người dùng kích hoạt chương trình ứng dụng.
- ✓ Tiến trình ***chạy trong nhân HĐH***, tác nhân: khi chương trình đòi hỏi tài nguyên, nhân HĐH "thay mặt" người dùng truy nhập tài nguyên (RAM, I/O) bằng các chức năng của nhân, các chức năng này được kích hoạt từ ứng dụng qua GHT hay ngắt mềm.
- ✓ Tiến trình ***dừng, đợi***, tác nhân: khung thời gian cho tiến trình đã hết, hay tiến trình bị gián đoạn vì các lí do khác (dừng để xử lí ngắt, *đợi* có tài nguyên…).
- ✓ Tiến trình chuyển vào hàng đợi (wake up) ***sẵn sàng*** chạy lại, tác nhân: khi nhu cầu tài nguyên đã thỏa mãn, đến lượt chạy lại. Hệ lập lịch sẽ thực hiện quá trình này: Dừng → Hành đợi → Chạy trong nhân →Chạy trong chế đọ người dùng …→kết thúc.
- ✓ ***fork():*** tạo ra tiến trình mới. Tạo tiến trình mới có mức ưu tiên cao, do đó có khả năng chen ngang thực thi của một tiến trình khác đang chạy.
- ✓ Các trạng thái sẽ biến chuyển theo mô hình cho tới khi chương trình kết thúc.
  (Xem thêm lí thuyết HĐH đa trình, đa nhiệm).

- **Chế độ người dùng và chế độ nhân HĐH**

Trong các HĐH đa nhiệm có một ranh giới giữa hai "không gian"gọi là không gian (địa chỉ) người dùng và không gian (địa chỉ) nhân HĐH dành riêng cho HĐH và được bảo vệ bởi CPU và mã nhân HĐH. Nói nhân HĐH "thay mặt" người dùng truy nhập tài nguyên (RAM, I/O), có

nghĩa là tài nguyên do nhân HĐH quản lí, cung cấp cho chương trình ứng dụng ("người dùng") tài nguyên khi có nhu cầu, có như vậy mới tránh được tranh chấp tài nguyên hệ thống. Để hình dung quá trình chuyển trạng thái từ không gian người dùng và không gian nhân như sau:

- *Mã HĐH* chạy trong *không gian địa chỉ nhân HĐH*, gọi là chế độ nhân (***kernel mode*** hay ***supervisor mode*** hay ***privilege mode***). Chế độ này được hỗ trợ bởi kiến trúc của CPU (bởi các lệnh máy đặc biệt hay bằng phần cứng) và nó ngăn người dùng truy nhập vào phần cứng (quản lí phần cứng chuẩn xác cho nhiều người dùng đồng thời, còn gọi là chế độ được bảo vệ (***protected mode***)).
- Mã các trình ứng dụng chạy trong *không gian địa chỉ người dùng*, gọi là chế độ người dùng (***user mode***). Việc kết nối giữa hai chế độ chạy trình được thực hiện bởi ***gọi hệ thống*** (***system call***). Gọi hệ thống (hay *gọi các dịch vụ của hệ thống*, GHT), là một giao diện lập trình giữa HĐH và ứng dụng. Nó được thực hiện bằng cách đặt các thông số vào những chỗ được định nghĩa rỏ ràng (vào các thanh ghi của CPU hay đặt vào stack) và sau đó thực hiện một lệnh bẫy đặt biệt (**trap instruction**) của CPU. Lệnh này chuyển chế độ chạy máy từ *user mode* vào *kernel mode* và từ đó điều khiển chuyển cho HĐH (1). Tiếp theo HĐH kiểm tra *số hiệu* và các *thông số của GHT* để xác định GHT nào sẽ thực hiện (2). Từ trong bảng với chỉ số là số hiệu của GHT), HĐH lấy ra con trỏ trỏ đến qui trình (*procedure*) thực hiện GHT đó (3). Khi thực hiện xong GHT, điều khiển chuyển trả lại cho chương trình của người dùng.
- Thuật ngữ ***nhân*** (*kernel)* để cập đến phần mã cốt yếu nhất của các chương trình hệ thống, nó kiểm soát tài nguyên máy tính như các tệp, các thiết bị điện tử, khởi động và cho chạy các chương trình ứng dụng đồng thời, cấp bộ nhớ cũng như các tài nguyên khác cho các chương trình của người dùng. Bản thân kernel không làm gì nhiều nhưng cung cấp các công cụ nguyên thuỷ (*primitive functions*) mà các tiện ích khác, các dịch vụ khác của HĐH. Do đó các chương trình hệ thống, các trình ứng dụng sử dụng các dịch vụ của HĐH, và chạy trong ***user mode***. Tuy nhiên có sự khác biệt là các trình ứng dụng thì tận dụng những tiện ích hệ thống cho, còn các trình hệ thống là sự cần thiết để máy tính chạy được.

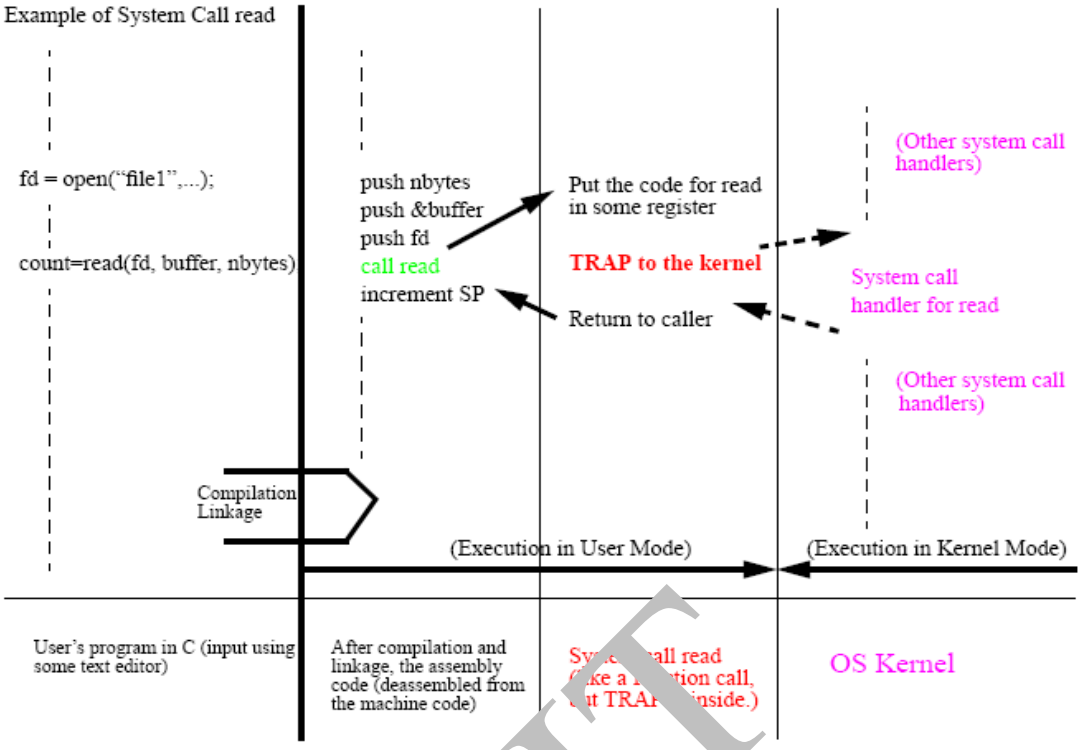


Hình 3.4 Triển khai API qua GHT

Từ đây có thể thấy cấu trúc cơ bản của GHT như sau:

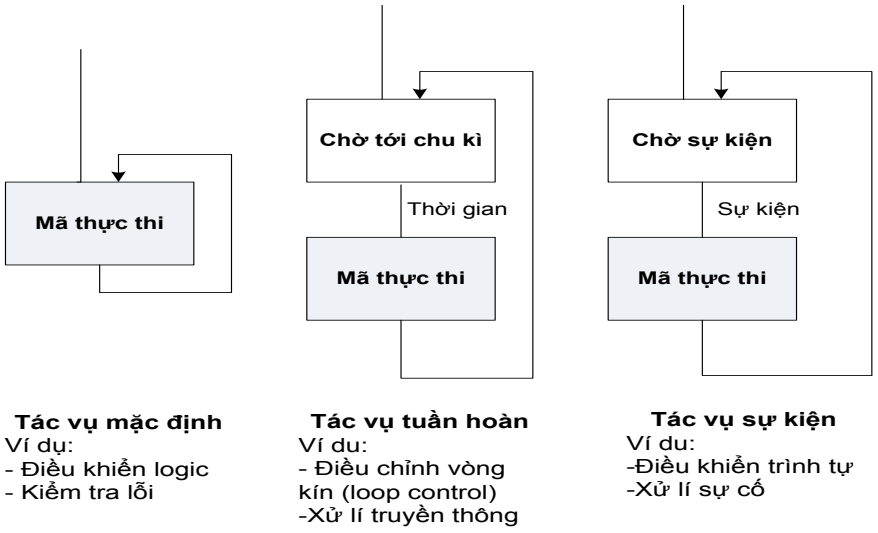
1. Một chương trình chính kích hoạt dịch vụ hệ thống bằng một GHT.
2. Bẩy (TRAP) chuyển GHT vào nhân HĐH, nhân xác định số hiệu của dịch vụ.
3. Thực hiện dịch vụ.
4. Kết thúc dịch vụ và trở về nơi phát sinh GHT.

Hình sau cho các bước theo trình tự từ lập trình đến thực thi GHT *read()*:



Hình 3.4 Nguyên lí đa trình và quan hệ giữa chế độ người dùng và chế độ nhân HĐH

- **Tác vụ (Task)** : là một ứng dụng chạy trong thời gian thực và bị giám sát bằng hệ thống lập lịch của HĐH. Trong các hệ thống điều khiển, khái niệm tác vụ cũng hay được sử dụng bên cạnh quá trình tính toán. Tác vụ có thể thực hiện theo cơ chế tuần hoàn (*periodic task*) hoặc theo sự kiện (*event task*). Các dạng tác vụ qui định trong chuẩn IEC 61131-3 (Programmable Controllers – Part3: Programming Languages) được minh họa trên hình 3.5. Ví dụ, một tác vụ thực hiện nhiệm vụ điều khiển cho một hoặc nhiều mạch vòng kín có chu kỳ trích mẫu giống nhau. Hoặc, một tác vụ có thể thực hiện nhiệm vụ điều khiển logic, điều khiển trình tự theo các sự kiện xảy ra. Tác vụ có thể thực hiện dưới dạng một quá trình tính toán duy nhất, hoặc một dãy các quá trình tính toán khác nhau.



Hình 3.5 Các kiểu tác vụ

So sánh giữa chức năng (function), dịch vụ ngắt và tác vụ

Chức năng (function)	dịch vụ ngắt (ISR)	tác vụ (Task)
- Tập hợp các lệnh để thực hiện một hành động. - Được kích hoạt bởi thủ tục Proc/Task/ISR. - Mỗi chức năng có bối cảnh thực thi riêng.	- ISR độc lập. - Kiscg hoạt bởi ngắt cứng hay mềm. - ISR được gán mức ưu tiên. - Mỗi ISR có bối cảnh riêng.	- Tác vụ là độc lập. - Được đồng bộ thực thi bởi RTOS. - Lập lịch chọn một tác vụ để chạy trong một khung thời gian.

- **Luồng (Threads)**: Là những phần chia nhỏ của một tiến trình, chạy trên các hệ có đa CPU, và được kiểm soát bởi hệ thống kiểm soát tiến trình.
- **Ngắt và ngoại lệ (*exception)*:** là bất kì sự kiện nào làm gián đoạn thực thi của CPU và "đẩy" CPU vào thực thi những lệnh đặc biệt ở trạng thái đặc quyền của CPU. Các ngoại lệ phân loại vào 2 hạng: ngoại lệ đồng bộ vad ngoại lệ không đồng bộ.

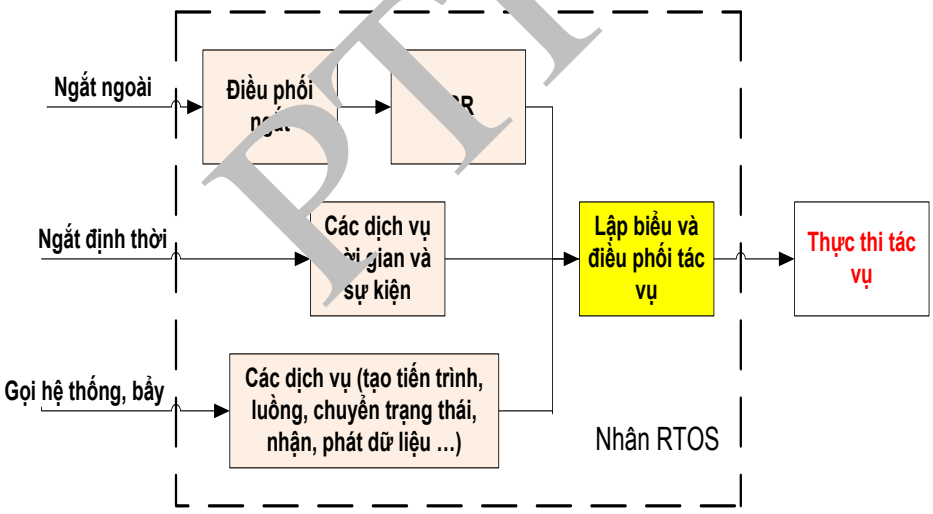
- ✓ Ngoại lệ đồng bộ: phát sinh bởi các sự kiện bên trong, ví dụ khi thực hiện các lệnh của CPU : Có một số CPU truy nhập bộ nhớ bắt đầu ở địa chỉ chẵn, nếu ngược lạ sẽ báo lỗi, hay khi thực hiện phép chia mà mẫu số bằng 0.
- ✓ Ngoại lệ không đồng bộ: là các sự kiện phát sinh từ bên ngoài không liên quan tới thực hiện lệnh của CPU. Các sự kiện này gắn liề với các tín hiệu ngắt, từ các thiết bị. Ví du: ấn nút RESET trên HTN (system reset exeption), các thiết bị truyền thông của HTN tạo ra các ngắt khi nhận được dữ liệu. *Ngắt ngoài* cũng được qui chiếu vào loại này.

Sự khác biệt của hai loại ngoại lệ chỉ là nguồn phát sinh của sự kiện.

- **Điều phối chạy chương trình bằng lập lịch (*schedule)***

Lập lịch là một phép thực hiện phân bổ và gán quy trình thực thi các tác vụ cho bộ xử lý sao cho mỗi tác vụ được thực hiện hoàn toàn. Như vậy công việc của lập lịch là tìm kiếm một giản đồ phân bố thời gian thực thi theo kiểu đa nhiệm hợp lý với các điều kiện ràng buộc cho trước. Hay nói cách khác là lập lịch phải xử lý thông tin điều kiện để quyết định và điều phối thực hiện tiến trình/tác vụ.

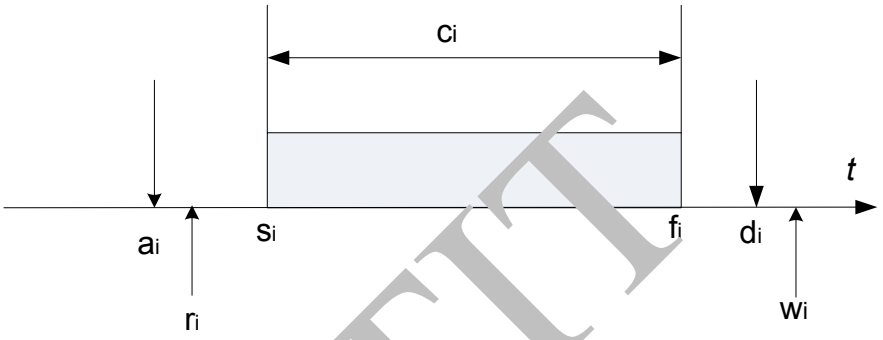
Quản lí tác vụ thông qua lập biểu:



Hình 3.6 Hầu hết các loại tác vụ đề xuyên qua lập biểu.

- **Hạn chót (*deadline*)**: thời điiểm cuối cùng phải cho ra một đáp ứng trong các hệ ràng buộc về thời gian. Trong hình dưới đây *di* là hạn chót.
  - ✓ Thời gian xuất hiện *ai* (*arrival time*): Khi sự kiện xảy ra và tác vụ tương ứng được kích hoạt.
  - ✓ Thời điểm bắt đầu thực thi *ri* (*release time*): Thời điểm sớm nhất khi việc xử lý đã sẵn sàng và có thể bắt đầu.

- ✓ Thời điểm bắt đầu thực hiện *si* (*starting time*): Là thời điểm mà tại đó tác vụ bắt đầu việc thực hiện của mình.
- ✓ Thời gian tính toán/thực thi *ci* (*Computation time*): Là khoảng thời gian cần thiết để bộ xử lý thực hiện xong nhiệm vụ của mình mà không bị ngắt.
- ✓ Thời điểm hoàn thành *fi* (*finishing time*): Là thời điểm mà tại đó tác vụ hoàn thành việc thực hiện của mình.
- ✓ Thời gian rủi ro/ xấu nhất *wi* (*worst case time*): khoảng thời gian thực hiện lâu nhất có thể xảy ra, dung sai mềm = wi-di.
- ✓ Thời điểm kết thúc d*i* (*due time-deadline*): Thời điểm mà tác vụ phải hoàn thành.



Hình 3.7 Biểu đồ thực hiện một tác vụ

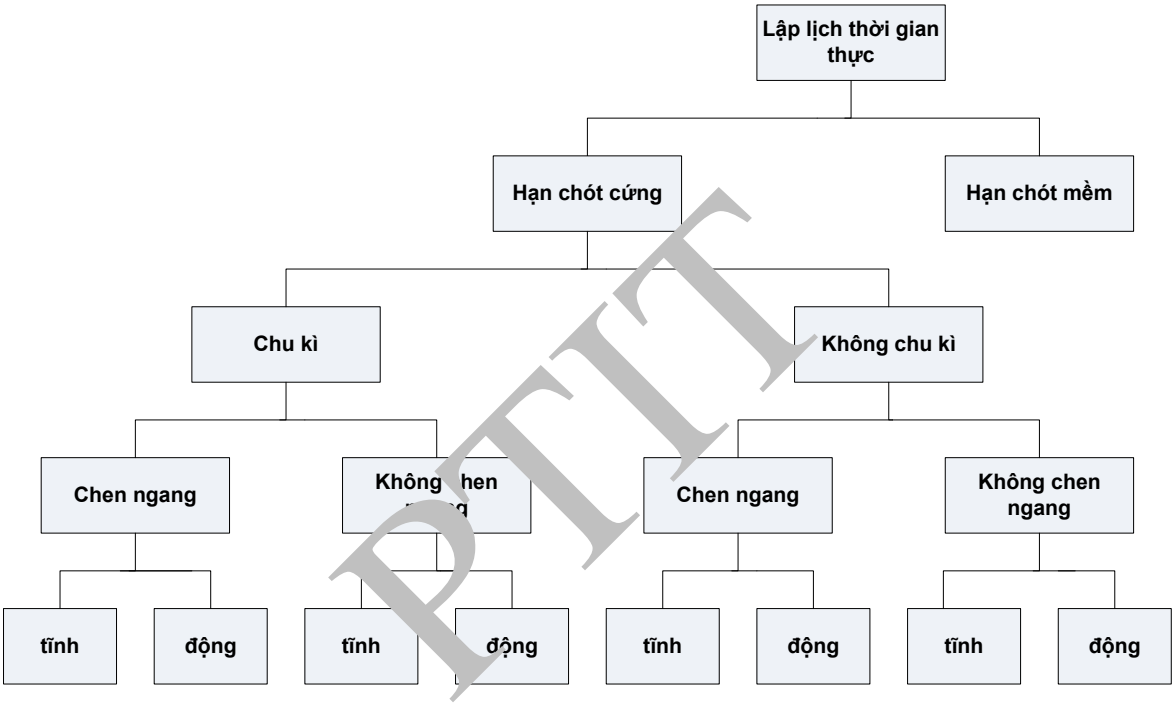
- **Lập lịch (scheduling) :** kế hoạch hóa việc thực hiện các tác vụ theo thời gian.
  a) *Phân loại lập lịch*, các TT được phân ra làm hai lớp theo nhu cầu thời gian, là :
     1. *"hướng I/O"* (**I/O bound**), ý nghĩa TT sử dụng nhiều tới I/O và sử dụng nhiều thời gian để đợi kết quả I/O;
     2. " hướng CPU" (**CPU-bound**), là lớp TT yêu cầu nhiều thời gian CPU.

Ở các HĐH đa năng, cũng có cách phân biệt khác theo lớp ứng dụng như:

- Lớp TT tương tác (**interactive**), tương tác thường xuyên với user, do đó chi phí nhiều thời gian cho nhận bàn phím, hay chuột. Khi có đầu vào TT phảI thức dậy thật nhanh, thời gian trễ trung bình là 50 đến 150 ms, nếu quá chận, user co hệ có vấn đề; Các ứng dụng như soạn văn bản, shell, đồ hoạ (táI tạo hình ảnh) thuộc lớp này;
- Lớp TT xử lý lô (**batch**), không cần có sự can thiệp của user, và thường chạy nền (background). Do không cần phải có đáp ứng nhanh, nên thường không được xếp ưu tiên cao, Ví dụ loại này là các trình dịch (compiler) hay cơ cấu tìm dữ liệu (database search engine), các tính toán khoa học;

- TT thời gian thực (**real time**), có yêu cầu rất khắc khe, không bao giờ bị cản trở bởi các TT mức ưu tiên thấp, cần đáp ứng thời gian thật nhanh, và quan trọng hơn cả là thời gian đáp ứng chỉ được thay đổi ở mức tối thiểu. Các ứng dụng như video, âm thanh, điều khiển robot, thu nhặt số liệu vật lí thuộc lớp này.

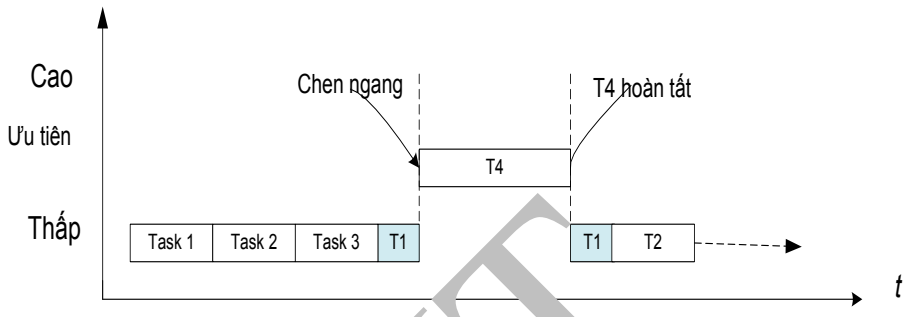
b) *Các giải thuật lập lịch* có nhiều và theo nhiều tiêu chuẩn. Mô hình dưới đây trình bày mmojt số giải thuật lập lịch ở hệ thời gian thực:



Hình 3.8 Phân loại các giải thuật lập lịch thực hiện tác vụ

- ✓ Lập lịch *hạn chót mềm* thường được thực hiện thông qua mở rộng một số chức năng của hệ điều hành. Ví dụ như cung cấp các mức ưu tiên cho việc kích hoạt các tác vụ hay gọi hệ thống. Hạn chó mềm tự thân nó không chứa đựng nhiều yếu tố có tính quyết định, thời gian xác định có thể xê dịch với dung sai (có thể đáng kể) hay tính nhạy cảm thời gian không quan trojnh lắm (less time-sensitive).
- ✓ Lập lịch *hạn chót cứng* lại rất nhạy cảm với thời gian (more time-sensitive). Đúng hạn phải thực hiện/hay kết thúc không thể kéo dài, sai hạn định sẽ dẫn tới hậu quả (nghiêm trọng).
- ✓ *Chen ngang* (p*reemptive)*: giải thuật được sử dụng nếu có tác vụ nào đó có thời gian thực thi quá lâu, nó có thể bị tác vụ khác ngắt hay nếu một sự kiện bên ngoài cần thời gian đáp ứng ngắn, sụ kiện đó sẽ ngắt sự kiện khác.

- ✓ *Không chen ngang* (n*on-preemptive)*: giải thuật giả định rằng các tác vụ sẽ thực hiện cho tới khi hoàn tất. Như vậy nếu có một tác vụ thực thi quá lâu, thì đáp ứng cho các sự kiện ngoài sẽ lâu.
- ✓ *Quay vòng kết hợp mức ưu tiên (Co-operative Round robin Scheduling):* Sử dụng qui tắc quay vòng theo mức ưu tiên giảm dần. Trong mối mức ưu tiên có thể có vài tác vụ đồng mức. Vì quay vòng ít hiệu quả trong RTOS, nên khi dùng cần kết hợp với cơ chế chen ngang thì mới đạt hiệu quả: T1 quay vòng và bị T4 mới chen ngang:



Hình 3.9 Quay vòng kết hợp ưu tiên với chen ngang

Việc tính mức ưu tiên của mỗi tiến trình được thực hiện theo một trong số các thuật toán sau:

- *Rate monotonic*: tác vụ nào chương trình diễn ra thường xuyên càng được ưu tiên.

- *Deadline monotonic*: tác vụ nào càng gấp, có thời hạn cuối càng sớm càng được ưu tiên.
- *Least laxity*: tác vụ nào có tỷ lệ thời gian tính toán/thời hạn chót (deadline) càng lớn càng được ưu tiên.

- ✓ *Lập lịch tuần hoàn / không tuần hoàn (periodic and aperiodic):* Tác vụ sẽ thực hiện cứ sau một đơn vị thời gian, gọi là tác vụ tuần hoàn. Ngược lạ là tác vụ không tuần hoàn và có đặc điểm là sẽ thực hiện vào các thời điểm không dự đoán trước được.
- ✓ *Lập lịch tĩnh (offline)*: Việc lập lịch được thực hiện dựa trên các hiểu biết hoặc dự báo về các sự kiện tác vụ thực hiện trong hệ thống (thời điểm xuất hiện, thời gian thực hiện, hạn chót ước tính (*deadline)* và được quyết định tại thời điểm thiết kế và được áp dụng cố định trong suốt quá trình hoạt động của hệ thống. Các tác vụ được khởi động ở các thời điểm đã lập trong một bảng trước đó, module phân phối (dispacher) khởi động tác vụ theo chỉ dẫn trong bảng. Module phân phối được kiểm soát bởi các bộ định thời (timer), đưa module vào thực hiện phân tích bảng lịch để kích hoạt tác vụ đúng thời điểm. Một hệ thống bị giám sát bởi timer gọi là **time trigged** (*TT system).*
Việc lập lịch trước có một số các ưu điểm sau:

- Tác vụ tiếp theo có thể được lựa chọn thực thi trong khoảng thời gian là hằng số
- Khả năng đáp ứng yêu cầu thời gian thực có thể được biết trước và được đảm bảo

Nhược điểm:

- Không thể thay đổi lịch trình thực hiện của hệ thống trong quá trình thực hiện
- Đòi hỏi phải có thông tin thời gian chính xác về các tác vụ để tính toán lập lịch.

Một thuật toán lập lịch tĩnh được gọi là tối ưu nếu nó luôn luôn có thể tìm được một lịch điều phối thoả mãn các ràng buộc đã cho trong khi một thuật toán tĩnh khác cũng tìm được một lời giải.

- ✓ *Lập lịch động (online)*: Bộ xử lý thực hiện việc lập lịch trong quá trình thực thi (*run time*) dựa trên cơ sở các thông tin hoạt động hiện hành của hệ thống. Sơ đồ lập lịch là không xác định trước và thay đổi động theo quá trình thực hiện. Lập lịch động linh hoạt nhưng tốn thời gian tính toán để ra quyết định và cũng không có "nhận thức" tới bối cảnh tổng thể như các yêu cầu tài nguyên, sự phụ thuộc giữa các tác vụ. Ở các HTN các bối cảnh tổng thể thời được định hình lúc thiết kế. Ví dụ là lập lịch kiểu Earliest deadline first scheduling and Least slack time scheduling.
- ✓ *Lập lịch tập trung hoặc phân tán* (*centralized / distributed*): là các giải thuật lập lịch chạy trên một hay vài CPU của một máy.
- ✓ *Lập lịch Mono* hay *Multi-processor*: Giải thuật lập lịch trên các CPU đơn lẽ hay trên nhiều CPU. Loại giải thuật này triển khai trên các hệ thống liên kết, các hệ thống có thể là đồng nhất (cùng loại CPU) hay không đồng nhất (nhiều máy, các máy có loại CPU khác nhau) ứng dụng để thao tác những đích xác định (target specific) có thời gian ràng buộc. Sử dụng áp dụng trên các hệ thống hỗn hợp hardware/software, trên đó một số tác vụ chuyển cho phần cứng thực hiện.

Các nhân HĐH được điều khiển theo cơ chế ngắt thường thực thi cơ chế lập lịch không chen ngang (*dymnamic nonpreemtive)* động , trong khi loại hạt nhân HĐH vận hành theo quá trình lại thực thi theo cơ chế chen ngang động (*dynamic preemptive)*.

Trên cơ sở đó bộ lập lịch sẽ phải thực hiện bài toán tối ưu về:

- ✓ Thời gian đáp ứng (*response time*)
- ✓ Hiệu suất thực hiện (số lượng công việc thực hiện xong trong một đơn vị thời gian
- ✓ Sự công bằng và thời gian chờ đợi (các tác vụ không phải chờ đợi quá lâu)

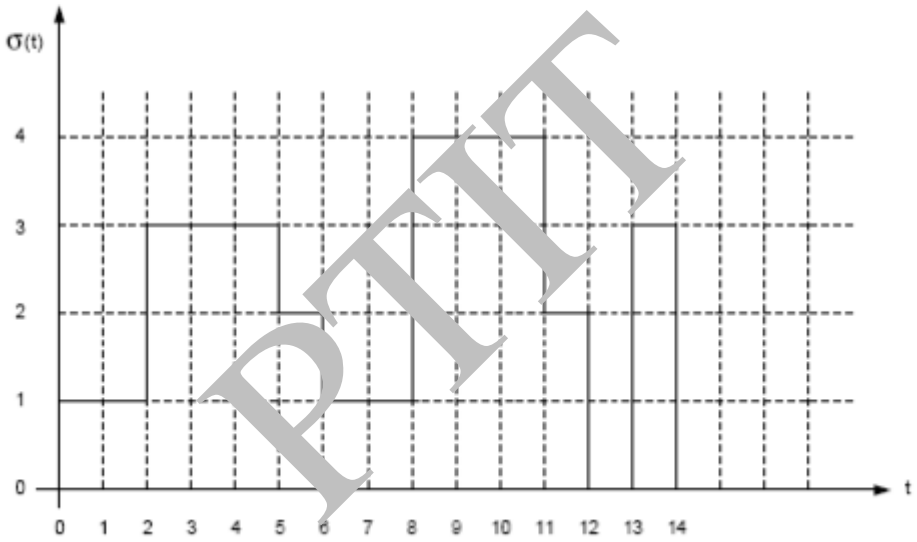
Ví dụ bài toán lập lịch:

Scheduling Function

A real-time system had to execute four tasks J1, J2, J3, J4 with arrival times and deadlines shown in the

following table. The scheduling function _(t) observed is shown in Figure 6.

	J_1	J_2	J_3	J_4
arrival time	0	4	2	4
deadline	9	10	17	13



Determine

a) the maximum lateness,

b) the tasks' laxities, and

c) the processor utilization for this schedule.

Is the schedule feasible? If not, try to modify the scheduling function so that the schedule becomes

feasible.

Solution - Task 4

The *lateness* of a task is the delay of the task completion with respect to its deadline (note that if the task

finishes before its deadline, its lateness is negative).

The maximum lateness is of the task J2 = fi − di = 12 − 10 = 2 (task J2 is the only task that violates

the given constraints).

The *laxity* (or *slack time*) is the maximum time that a task can be delayed on its activation to complete

within its deadline.

The laxities of the tasks J1, . . . , J4 are as follows (in the table below are the individual tasks' computation

times):

1. X1 = d1 − a1 − C1 = 9 − 0 − 4 = 5
2. X2 = 10 − 4 − 2 = 4
3. X3 = 17 − 2 − 4 = 11
4. X4 = 13 − 4 − 3 = 6

	J_1	J_2	J_3	J_4
C_i	4	2	4	3

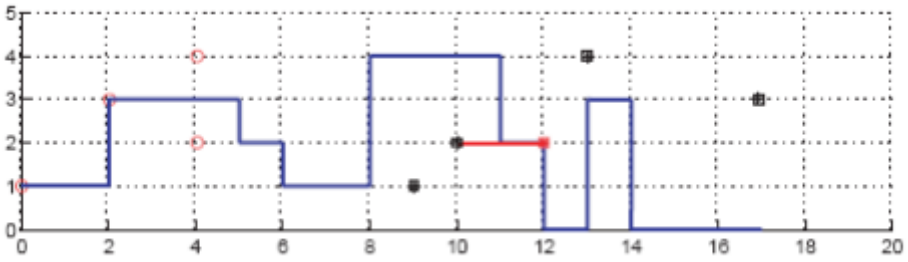
The CPU_Utilization is 13/14 = 0.93.

A schedule is said to be *feasible (khả thi)* if all tasks can be completed according to a set of specified constraints.

The scheduling function suggested in the exercise, _(t) is thus **not** feasible (the task J2 does not meet its

deadline).

Figure 7: Violation in the old scheduling function



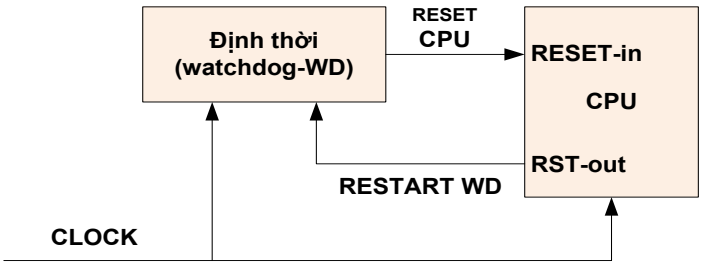
The schedule can be modified in order to complete the execution of all tasks before their respective

deadline. One (of several) feasible scheduling functions is depicted in the figure. Figure 8: New scheduling function

- **Bộ giám sát định thời (watchdog timer):** là đồng *hồ thời gian cứng* (dùng các bộ đếm điện tử)
  với các ứng dụng sau đây*:*

  - Làm đồng hồ thời gian thực (real time clock-RTC) cho hệ thống;
  - Khởi động/ khởi động lại một sự kiện sau một thời gian đặt trước;
  - Tạo khung cửa sổ thời gian cho một sự kiện;
  - Phân giải khoản thời gian giữa 2 sự kiện;
  - Chó canh chừng.

  hay *hồ thời gian mềm* (lập các giá trị đếm cho một biến chương trình), các thao tác tăng/giảm giá trị đếm thực hiện bằng lệnh máy, do đó phụ thuộc vào CPU clock (mỗi loại CPU có clock khác nhau).



Hình 3.10 Mô hình nguyên lí cho WD

Chó canh chừng sẽ thực hiện một tái khởi động (reset) hệ thống hay một hành động (xử lý), hay kích hoạt một xử lý hiệu chỉnh nếu chương trình chính bị lỗi, không tiến triển được do một điều kiện nào đó không thể đạt được trong khung thời gian dự tính (chương trình bị treo). Trong các hệ thời gian thực, trong đó có HTN, chó canh chừng rất quan trọng, được sử dụng để tự động khởi động lại một ứng dụng nhúng hay thậm chí cả hệ thống về trạng thái ban đầu mà không có sự can thiệp của con người. Trong các CPU nhúng ta thấy có vài bộ đếm thời gian (*specialized timers*) cứng, đặt các giá trị khác nhau cho các ứng dụng quan trọng, mà trong khung thời gian đó ứng dụng phải kết thúc hay phải đưa ra được đáp ứng, nếu không (khả năng có sự bất thường) ứng dụng sẽ được chó canh chừng khởi động lại từ đầu. Các hệ có cài trình gởi rối sẽ ghi lại vào bộ lưu trữ đặc biệt để hỗ trợ khác phục sự cố. Có thể timer mềm cho các ứng dụng tới hạn (*critical*) hơn.

Ví dụ *watchdog timer* là một bộ đếm (counter) cứng/mềm với *x* bit, với đầu vào có tần số đếm *f* , thời gian đặt vào bộ đếm là T, sau T đơn vị thời gian, bộ đếm đạt giá trị đặt (hay từ giá trị đặt kuif về 0). Nếu sau thời gian đó mà bộ đếm không được tái khởi động, hệ thống sẽ bị khởi động lại.

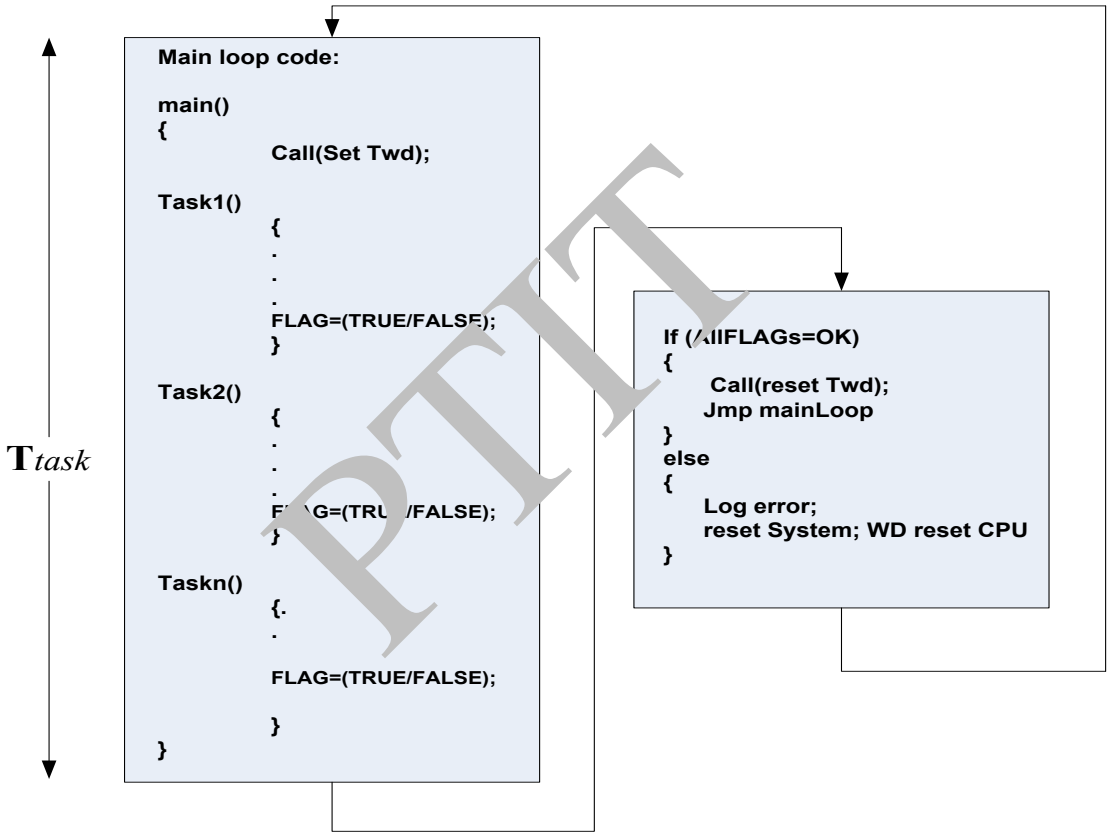
Ví dụ : giả định một phần mềm, có có vòng lặp chạy trong 25 micro giây, hay với dung sai tối đa là 35 micro giây để thực hiện xử lý. Dùng một một *watchdog timer* có đầu ra nối vào một ngắt không che (NMI), hay vào chân RESET của CPU. *watchdog timer* được nạp một giá trị 50 micro giây, nếu sau thời gian đó phần mềm vòng lặp không kết thúc, đầu ra của *watchdog timer* sẽ kích hoạt RESET, khởi động lại hệ thống. Nếu vòng lặp kết thúc với trạng hái bình thường, *watchdog timer* sẽ được nạp lại giá trị cho một chu kì mới.

HTN là loại hệ thống con người không thể giám sát hoạt động thường xuyên, do vậy chó canh chừng là giải pháp nếu hệ thống bị treo (lí do ….). Có những trường hợp nếu phần mềm có lỗi nghiêm trọng, chó canh chừng có thể sẽ loại (*disable*) phần mềm đó không cho chạy nữa.

Giải thuật với *watchdog timer:*

Mỗi tác vụ khác biệt thực thi bởi một "*main loop*", nếu kết thúc hoàn hảo, đặt cờ trạng thái lên (Flagi set = TRUE). Tất cả các *main loop* thực hiện tối đa trong 35 micro giây. Sau vòng cuối cùng là đoạn mã kiểm tra :

Nếu tất cả các Flags đều là TRUE, khởi động chu kì watchdog mới (50 micro giây), nếu Flags = FALSE, ghi nhận sự cố và đặt tất cả Flags = FALSE, watchdog không được khởi động lại (kick the dog) trong thời gian 50 micro giây, đầu ra của bộ đếm watchdog sẽ kích hoạt RESET hệ thống.



Hình 3.11 Giải thuật với giám sát định thời (chó canh chừng)

Đoạn mã

```
if  (all flag are OK)
        {
            Call(Reset Twd);        //Hệ hoạt động bình thường,
                                    //đặt mới giá trị 50 micro giây co watchdog
```

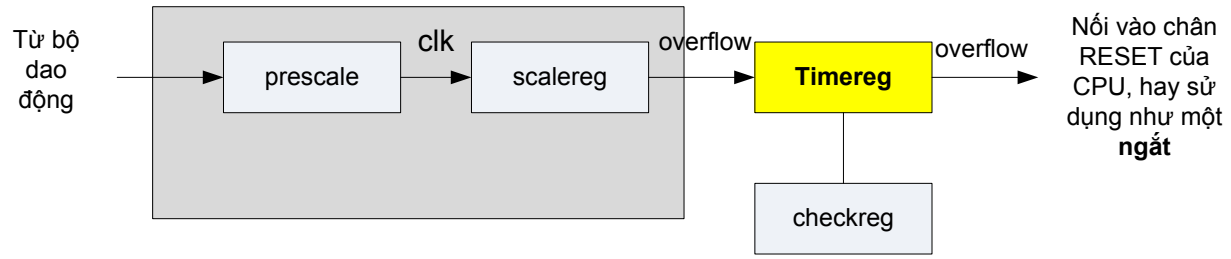
```
            Jmp mainloop;        //Trở về chu kì mới;
        }
else                    //nếu thực hiện đoạn code này có nghĩa
                        //counter sẽ vượt 50 giây ấn định →xung đầu ra sẽ RESET CPU.
        {
            Log eror;           //Record failure
            Reset System;       //WD reset CPU
        }
```

Ví dụ tạo một watchdog timer:

Bài toán: Phải reset timer cứ sau x đơn vị thời gian, nếu không timer sẽ phát sinh tín hiệu overflow (dùng để khởi động lại hệ thống cứng RESET hay qua ngắt).

Sử dụng: Phát hiện lỗi hệ thống thời gian thực sau x đơn vị thời gian; hay tự RESET hệ thống cứ sau x đơn vị thời gian.

- Ví dụ với máy ATM rút tiền tự động timer 16 bit, thời gian rút tiền tối đa là 2 phút,
- Với 16 bit, đếm nhị phân từ 0 → 65.535 trong 2 phút ta có:
  T_{clk} =2*60*1000(ms)/65.535 = 0,54 ms →f_{clk} =1/0,54 ms= 1,85 KHz



Giải thuật:

/*main.c*/

Main()

```
{
        Wait until card inserted;
        Call watchdog_reset_routine;
        while(transaction in progress)
        {
                If(button pressed) {
                                        Perform correspondding actions;
                                        Call watchdog_reset_routine;
                                    }
        }
        /* Nếu sau 2 phút Call watchdog_reset_routine không kích hoạt sau  t < 2 phút,*/
        /* thì interrupt_service_routine sẽ kích hoạt.*/
 }

watchdog_reset_routine()
{
        /* Kiểm tra nếu checkreg = 1, nạp giá trị vào timereg: Nạp 0 vào scalereg, và giá trị*/
        /*65.535 vào timereg*/
        checkreg =1;
        scalereg = 0;
        timereg = 65.535;
}

Void interrupt_service_routine()
```

```
{
        eject card;
        reset screen;
}
```

Có rất nhiều kĩ xảo sử dụng với *watchdog timer* khi thực hiện một hệ thống cụ thể.

- **Truyền thông giữa các tiến trình (Inter Process communication – IPC)**
  Để đồng bộ chạy các chương trình hệ điều hành thực hiện một số cơ chế để đồng bộ thực hiện các tiến trình, bao gồm:

  1) *pipe* (đường ống): không tên, *FIFO:* có tên;
  2) *thông điệp (message)*: cho phép các TT gửi các khuôn dữ liệu có khuôn dạng tới bất kì TT nào;
  3) *vùng nhớ chia sẽ (shared memory)*: các TT chia sẻ một phần không gian địa chỉ ảo của mình;
  4) *đánh tín hiệu (semaphore)*: các TT dùng để đồng bộ việc thực hiện.

Ghi chú: Phần này liên quan tới Hệ điều hành, nên khi học nên xem lại Lí thuyết Hệ điều hành đa nhiệm, đặc biệt RTOS

3.2.2 Hệ thống thời gian thực

Khái niệm *hệ thời gian thực* không đồng nghĩa với khái niệm hệ xử lý tốc độ cao, xử lý nhanh. Nếu ta cho rằng, phải là các ứng dụng điều khiển có yêu cầu thời gian tính toán rất nhanh mới gọi là điều khiển thời gian thực, thì một câu hỏi sẽ được đặt ra là: như thế nào mới được gọi là nhanh? Ta có thể thống nhất là, cỡ một vài micro-giây là rất nhanh, tuy nhiên nếu một vài chục micro-giây thì sao, một trăm micro-giây thì sao? Nếu một trăm micro-giây mới gọi là nhanh, thì 101, 102, ... có nhanh không? Các hệ điều khiển với chu kỳ trích mẫu 5ms, 6 ms, 7ms có được gọi là hệ thời gian thực hay không? Tốc độ không phản ánh thời gian thực nhưng để có đặc tính thời gian thực thì phụ thuộc rất nhiều vào tốc độ. Tốc độ càng cao thì sai số càng nhỏ và càng dễ thực hiện các tác vụ thời gian thực.

Có thể nói đơn giản hơn, *tính thời gian thực là khả năng đáp kịp thời* và *chính xác*.

- **Đặc điểm của hệ thời gian thực**
  - ✓ *Tính bị động*: Hệ thống phải phản ứng với các sự kiện xuất hiện vào các thời điểm thường không biết trước. Ví dụ, sự vượt ngưỡng của một giá trị đo, sự thay đổi trạng thái của một thiết bị quá trình phải dẫn đến các phản ứng trong bộ điều khiển.
  - ✓ *Tính chuẩn xác chức năng và chính xác về thời gian:* Các chức năng phải được thực hiện chuẩn xác. Các tính toán, xử lý phải cho ra kết quả trong một chu kì thời gian đã xác định trước. Chính xác về thời gian sẽ cho phép hệ đưa ra đáp ứng một cách kịp thời. Tuy tính chính xác thời gian là một đặc điểm tiêu lịch, nhưng một hệ thống có tính năng thời gian thực không nhất thiết phải có đáp ứng thật nhanh mà quan trọng hơn là phải có phản ứng kịp thời đối với các yêu cầu, tác động bên ngoài.

    Hãy khảo sát các ví dụ sau đây để làm rõ yếu tố thời gian trong hệ thống thời gian thực:

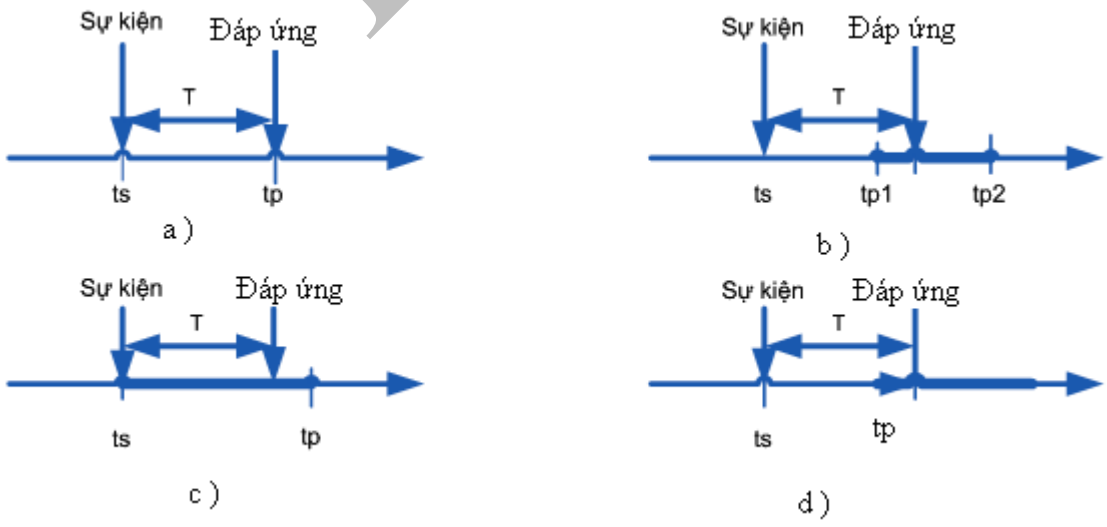
    ts là thời điểm một sự kiện xảy ra từ thiết bị và tác động tới hệ thống;

    T là khoản thời gian thực hiện các tính toán, xử lý từ ts tới tp để có đáp ứng đầu ra.

    tp là thời điểm ra đáp ứng.

    Ta sẽ có các trường hợp sau đây:

    a) Đáp ứng đúng theo yêu cầu tại tp: đáp ứng chính xác.
    b) Đáp ứng xảy ra trong khoản <tp1, tp2>.
    c) Đáp ứng xảy ra trong khoản <ts, tp>.
    d) Đáp ứng xảy ra từ tp cho tới …



Hình 3.12 Sự kiện và đáp ứng

Có thể còn có các phân tích khác đề cập tới thời điểm phải đưa ra đáp ứng cho tác động của sự kiện, qua đây ta thấy rõ hơn yêu cầu về tính thời gian thực của hệ thống thời gian thực: chính xác (a), chính xác tương đối (b, c) và tương đối lỗng lẻo. Như vậy khi áp dụng, ta có các thiết bị ghép nối mà đặc thù kĩ thuật chắc chắn sẽ rơi vào các trường hợp trên, giúp hình thành chiến thuật xử lý. Ở mô hình trên khung thời gian T là yếu tố quan trọng, vì đó là khung thời gian không thể nhỏ hơn được khi xem xét trên một HTN thời gian thực, tuy nhiên T khác nhau cho các hệ thống khác nhau. Giải pháp cho T phụ thuộc vào lựa chọn thiết kế phần cứng (đặc biệt là tốc độ của CPU) và kĩ năng viết phần mềm hệ thống và phần mềm ứng dụng.

- ✓ *Tính đồng thời*: Hệ thống phải có khả năng phản ứng và xử lý đồng thời nhiều sự kiện diễn ra. Ví dụ, cùng một lúc một hệ thống trong vai trò là một bộ điều khiển PID, được yêu cầu thực hiện nhiều vòng điều chỉnh, giám sát ngưỡng giá trị nhiều đầu vào, cảnh giới trạng thái làm việc của một số thiết bị khác
- ✓ *Tính tiền định*: Dự đoán trước được thời gian phản ứng tiêu lịch, thời gian phản ứng chậm nhất cũng như trình tự đưa ra các đáp ứng. Ví dụ, nếu một bộ điều khiển phải xử lý đồng thời nhiều nhiệm vụ, ta phải tham gia quyết định được về trình tự thực hiện các công việc và đánh giá được thời gian xử lý mỗi công việc. Như vậy người sử dụng mới có cơ sở để đánh giá về khả năng đáp ứng tính thời gian thực của hệ thống.

- **Xử lý thời gian thực**

Xử lý thời gian thực là hình thức xử lý thông tin trong một hệ thống để đảm bảo tính năng thời gian thực của nó. Như vậy, xử lý thời gian thực cũng có các đặc điểm tiêu lịch nêu trên như tính bị động, tính chuẩn xác, thời gian chính xác, tính đồng thời và tính tiền định. Để có thể phản ứng với nhiều sự kiện diễn ra cùng một lúc, một hệ thống xử lý thời gian thực sử dụng các quá trình tính toán đồng thời.

Quá trình tính toán là một tiến trình thực hiện một hoặc một phần chương trình theo tuần tự do hệ điều hành quản lý trên một máy tính, có thể tồn tại đồng thời với các quá trình khác kể cả trong thời gian thực hiện lệnh và thời gian xếp hàng chờ đợi thực hiện.

Các hình thức tổ chức các quá trình tính toán đồng thời:

- ✓ *Xử lý cạnh tranh*: Nhiều quá trình tính toán chia sẻ thời gian sử dụng của một bộ xử lý.
- ✓ *Xử lý song song*: Các quá trình tính toán được phân chia thực hiện song song trên nhiều bộ xử lý của một máy tính.
- ✓ *Xử lý phân tán*: Mỗi quá trình tính toán được thực hiện riêng trên một máy tính.

Trong các hình thức trên đây thì hình thức xử lý cạnh tranh có vai trò chủ chốt. Mặc dù hệ thống điều khiển có thể có nhiều trạm, và mỗi trạm có thể là một hệ đa vi xử lý, số lượng các quá trình tính toán cần thực hiện thường bao giờ cũng lớn hơn số lượng vi xử lý. Trong khi một vi xử lý không thể thực hiện song song nhiều lệnh, nó phải phân chia thời gian để thực hiện xen kẽ nhiều nhiệm vụ khác nhau theo thứ tự tùy theo mức ưu tiên và phương pháp lập lịch.

- **Phân loại hệ thời gian thực**

Như trình bày trên thì hệ thời gian thực có tính ràng buộc thời gian và mọi xử lý đều có thời hạn chót (*deadline*). Vậy thời gian ở đây qui chiếu theo cách nào ? Giống như trên máy tính nói chung ta có hệ thời gian thực mềm (*soft real time system*) và hệ thời gian thực cứng (*hard real time system*). Hai loại hệ thời gian thực này khác nhau ở chỗ nào, đó cách đánh giá sự dung sai của thời hạn chót, tính hữu ích của các kết quả tính toán sau thời hạn chót, và sự đánh giá ngặt nghèo khi rơi vào thời hạn chót. Với hệ thời gian thực cứng, mức độ dung sai của thời hạn chót là rất nhỏ có thể bằng không, các kết quả tính toán sau hạn chót không có giá trị, do đó những phát sinh sau hạn chót được coi là thảm họa. Trong khi đó dung sai ở hệ thời gian thực mềm là con số khác không, các kết quả tính toán sau hạn chót lại có tính chất khấu hao, cho nên giá trị của kết quả vào lúc đi qua hạn chót vẫn có ích.

✓ *hệ thời gian thực cứng*: là hệ mà dung sai tới hạn chót xấp xỉ bằng không. Nói cách khác phải đúng thời điểm nếu không sẽ là thảm họa. Các tiêu chí hệ thống như sau:

- Phải đảm bảo không để bất kì một sự kiện tới hạn (*critical event*) nào bị sự cố trong bất kì hoàn cảnh nào của hệ thống;
- Độ trễ đáp ứng cho sự kiện rất nhỏ (xét theo từng ứng dụng);
- Các sự kiện có tính chu kì phải được đảm bảo thực hiện đúng chu kì.

Khi thiết kế hệ này cần tính để kết quả tính toán có được trước hạn chót trước khi hệ phát ra đáp ứng.

✓ *hệ thời gian thực mềm:* là hệ phải hợp thời gian nhưng hạn chót có tính mềm dẻo. Như vậy hạn chót có thể có nhiều mức, hạn chót với thời gian T ước tính với trị trung bình, xác xuất đáp ứng đưa ra nằm trong các mức độ khác nhau với độ trễ trung bình và chấp nhận được. Tuy không gây ra thảm họa hệ thống nhưng phải trả giá khi độ trễ hệ thống tăng tỷ lệ thuận tùy thuộc vào ứng dụng. Cần có cơ chế bù trừ để loại trừ độ trễ này.

(

✓ ***real time computing:***

- *the objective is to meet the individual timing requirement of each task*
- *correct behavior depends on both (1) correct computation and (2) time at which results are produced*
- *system that must react within precise timing constraints to events in the environment*
- *characterized by a deadline*
- *should be predictable (when the timing constraints cannot be met, this must be notified in advance, so that an alternative (scheduling) plan may be planned, and possibly avoid the catastrophe)*
- *real time applications*
- *the most important features timeliness, design for peak load, predictability, fault tolerance, maintainability.*

✓ *Difference between Hard and Soft Real-Time Systems*

Hard Real-time systems

A real-time task is said to be hard, if missing its deadline may cause catastrophic consequences on the environment under control. Examples are sensory data acquisition, detection of critical conditions, actuator servoing.

Soft Real-time systems

A real-time task is called soft, if meeting its deadline is desirable for performance reasons, but missing its deadline does not cause serious damage to the environment and does not jeopardize correct system behavior. Examples are command interpreter of the user interface, displaying messages on the screen

A hard real-time system guarantees that critical tasks complete on time. This goal requires that all delays in the system be bounded from the retrieval of the stored data to the time that it takes the operating system to finish any request made of it. A soft real time system where a critical real-time task gets priority over other tasks and retains that priority until it completes. As in hard real time systems kernel delays need to be bounded

)

Hệ thời gian thực gắn liền với tương lai của các hệ thống nhúng. Các HTN trước đây đơn giản, như các thiết bị hoạt động độc lập (automous device), có chu kì sống dài. Tuy nhiên ngày nay, công nghiệp nhúng đã trải nghiệm một sự chuyển đổi mạnh mẽ với các số liệu như sau (báo cáo của Gartner Group):

- ✓ Chu kì phát triển trên thị trường nhanh chóng, chỉ trong vòng 6 đến chín tháng;
- ✓ Toàn cầu hóa đã dẫn đến xác định lại thi trường và không gian ứng dụng;
- ✓ Kiểu kết nối các hệ thống có dây và không dây, trở thành không thể thiếu ở các thiết bị nhúng;
- ✓ Các thiết bị điện tử ngày càng tinh vi;
- ✓ Kết nối các HTN là lĩnh vực ứng dụng mới và phụ thuộc vào hạ tầng mạng (LAN, Internet, Extranet, …);
- ✓ Tốc độ xử lí ngày càng nhanh theo tỉ lệ transitor trên chip theo định luật More: tổng số transistors/vi mạch tưng gấp đôi cứ sau 18 tháng.

Tất cả những điều đó với hệ thời gian thực sẽ làm cho HTN có đặc tính "xử sự chính xác theo thời gian thực".

3.2.3 Hệ điều hành thời gian thực (RTOS)

Hệ điều hành thời gian thực (Real-Time Operating Systems RTOS), là loại hệ điều hành hỗ trợ để xây dựng các hệ thống thời gian thực. Đặc biệt RTOS được dùng trong những ứng dụng điện toán nhúng có tài nguyên bộ nhớ hạn chế và yêu cầu ngặt nghèo về thời gian đáp ứng tức thời, tính sẵn sàng cao và khả năng tự kiểm soát một cách chính xác. RTOS xuất hiện ở hai dạng: cứng và mềm. Nếu tính năng xử lý ứng với một sự kiện nào đó không xảy ra hay xảy ra không đủ nhanh, RTOS cứng sẽ chấm dứt hoạt động này và giữ không gây ảnh hưởng đến độ tin cậy và tính sẵn sàng của phần còn lại của hệ thống.

Vì RTOS và HTN trở nên phổ biến trong các ứng dụng quan trọng, các nhà phát triển thương mại đang tạo nên những RTOS mới với tính sẵn sàng cao. Những sản phẩm này có một thành phần phần mềm chuyên dụng làm chức năng cảnh báo, chạy các chương trình chẩn đoán hệ thống để giúp xác định chính xác vấn đề trục trặc hay tự động chuyển đổi sang hệ thống dự phòng. Hiện thời RTOS sẵn sàng cao hỗ trợ bus Compact PCI của tổ chức PCI Industrial Computer Manufacturers Group, bus này dùng cho phần cứng có thể trao đổi nóng.

RTOS có rất nhiều dạng. Sản phẩm thương mại như VxWorks và VxWorks AE, đều của Wind River Systems Inc; VxWorks AE được thiết kế với tính sẵn sàng cao, hỗ trợ khả năng gửi thông điệp phân tán và có thể chịu lỗi. RTOS cho phép lập trình viên tách biệt thư viện dùng chung, dữ liệu và phần mềm hệ thống cũng như ứng dụng.

LynxOS là loại RTOS cứng, làm việc với Unix và Java. QNX chạy trên bộ xử lý Intel x86 với nhân chỉ có 10 KB.

RTOS của giới nghiên cứu gồm có Chimera của Đại học Carnegie Mellon. Đây là hệ thống đa nhiệm, đa bộ xử lý thời gian thực, được thiết kế để tạo sự dễ dàng cho các nhà lập trình trong việc tái cấu hình và tái sử dụng mã. Chimera nhắm vào các hệ thống rô bô và tự động. RTOS của Đại học Maryland, có tên là Maruti, hỗ trợ cho cả ứng dụng thời gian thực cứng và mềm.

Trong nhiều năm, ứng dụng dựa trên RTOS chủ yếu là trong các hệ thống nhúng công nghiệp và mới gần đây thì chúng đã có mặt khắp nơi, từ thiết bị y tế được điều khiển bằng camera ảnh cho đến máy pha cà phê, những ứng dụng tính toán phân tán, đặc biệt trong các thiết bị truyền thông, đang thúc đẩy các nhà phát triển hệ điều hành thực hiện nghiên cứu và phát triển các chuẩn.

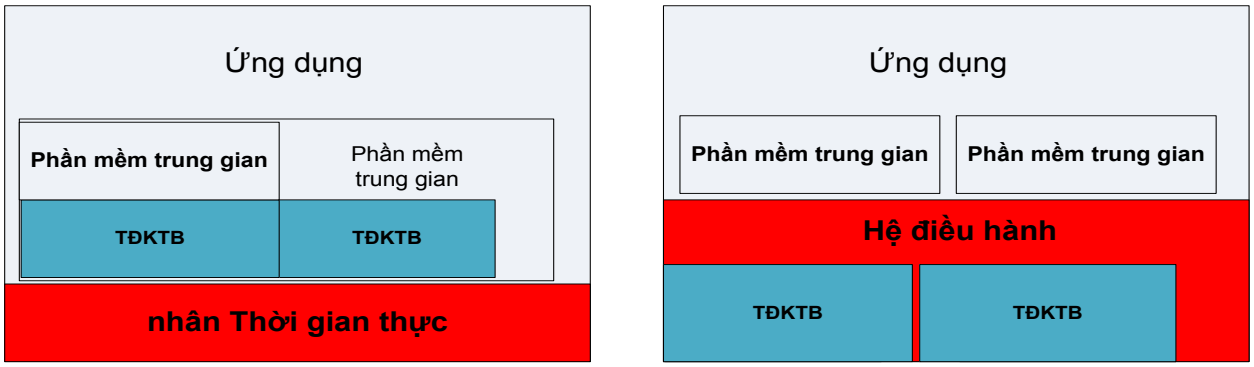
Như đã nói, RTOS tạo điều kiện để xây dựng các hệ thời gian thực, tuy nhiên nó không đảm bảo chắc chắn rằng kết quả cuối cùng sẽ là thời gian thực; điều đó phải được cải tiến trong phần mềm ứng dụng. RTOS không nhất thiết cần phải có tính xuyên suốt cao, mà RTOS cho các tiện ích, nếu sử dụng chính xác, thể có thể đảm bảo thỏa mãn được yếu tố hạn chót (deadline) bởi đặc tính thời gian thực mềm hay thời gian thực cứng. RTOS sử dụng giải thuật lập lịch đặc biệt, cung cấp công cụ cho người phát triển hệ thống kiểm nghiệm các tập tính của hệ thống đích. RTOS được

đánh giá về khả năng cho đáp ứng nhanh mức độ nào, khả năng tiên liệu để phản ứng với các sự kiện riêng biệt, chứ không phải là số lượng các tác vụ xử lý được trong một khung thời gian. Yếu tố then chốt trong RTOS là thời gian chờ xử lý ngắt tối thiểu, và thời gian chuyển đổi thực thi (*context switching*) giữa các tác vụ là tối thiểu. Yếu tố thứ hai ngày nay đạt được với hỗ trợ của các CPU đời mới tốc độ nhanh, kiến trúc hiện đại, cần ít CPU Clock hơn để chuyển đổi giữa các tác vụ, ví dụ như kiểu đường ống (pipeline), hay đa luồng.

Các yêu cầu cơ bản trên một RTOS

- **Các thao tác ràng buộc với thời gian phải dự tính được**. Mỗi dịch vụ của HĐH ngưỡng trên của thời gian thực hiện phải được đảm bảo. Trong thực tế có nhiều ngưỡng dự tính, tùy vào loại sự kiện hệ thooang chấp nhận.
- **RTOS phải quản lí chi phí thời gian chạy tác vụ và lập lịch để chạy các tác vụ**. RTOS chịu trách nhiệm về các hạn chót của các tác vụ, để trên cơ sở đó sử dụng giải thuật, kĩ thuật lập lịch phù hợp, ví dụ có thể lập lịch kiểu ofline, khởi động tác vụ theo thời điểm hay theo mức ưu tiên, hoặc theo các giải thuật online. Cung cấp các dịch vụ thời gian phải có độ chính xác cao, ví dụ các hệ thống điện đồng bộ thời gian giữa các nhà máy điện ở các vùng cách xa nhau tính theo múi giờ, sai số lớn là rất nguy hiểm.
- **RTOS có tốc độ xử lý nhanh.** Bên cạnh khả năng dự đoán, RTOS phải nhanh, có khả năng hỗ trợ các ứng dụng có thời gian hạn chế chỉ là vài phần của giây đồng hồ. Nhân của RTOS còn gọi là nhân thời gian thực, quản lí tài nguyên hệ thống (bộ nhớ, CPU, các bộ định thời). Các cơ chế bảo vệ nhân không cần có. Tuy nhiên ở mức độ hệ thống ứng dụng vò lí do như an ninh, tính độc lập có thể có cơ chế bảo vệ. Ví dụ, các thiết bị mạng là các HTN, có RTOS phải xử lý một khối lược rất lớn thông lượng, được bảo vệ bởi các phần mềm mạng riêng biệt như lad ứng dụng lớp trên HĐH.

Có hai kiểu RTOS:



Hình 3.13 RTOS nhân thời gian thực và RTOS đa năng

- ✓ HĐH RTOS đa năng: các TĐKTB có thể là nằm trong nhân, Các phần mềm trung gian, ứng dụng ở lớp trên và kết nối vào nhân qua giao diện lập trình API -GHT, ngắt mềm.

- ✓ RTOS với nhân thời gian thực: các TĐKTB không phải là thành phần nhúng trong nhân, được đặt trên nhân và chỉ có các TĐKTB cần thì đưa vào hệ thống. Các phần mềm trung gian và ứng dụng đặt ngay trên các TĐKTB, chứ không thông qua API như HĐH đa năng. Điều này là rõ ràng và hợp lí, vì các thiết bị nối trực tiếp vào hệ thống để nhân điều khiển trực tiếp thiết bị, đẩm bảo chi phí thời gian là ít nhất.

- ✓ Ở đây có sự khác biệt khi cài đặt TĐKTB, các tác vụ sẽ thao tác các thiết bị, chứ không phải các thiết bị hợp nhất trong nhân hệ đuều hành, do vậy sẽ:
  - Cải thiện tính tiền định, dự đoán (predictability), vì mội xử lí đều đi qua bộ lập lịch;
  - Các TĐKTB được thiết kế cho từng hệ thời gian thực, hhieju chỉnh cho lớp ứng dụng hệ thực hiện.

    ***(Device drivers handled by tasks** instead of integrated drivers:*

    - *Improved predictability; everything goes through scheduler;*
    - *Effectively no device that needs to be supported by all versions of the OS, except maybe the system timer.)*

    (Def.: A real-time operating system is an operating system that supports the construction of real-time systems.)

- ✓ ***Three key requirements:***

  1. The timing behavior of the OS must be predictable.

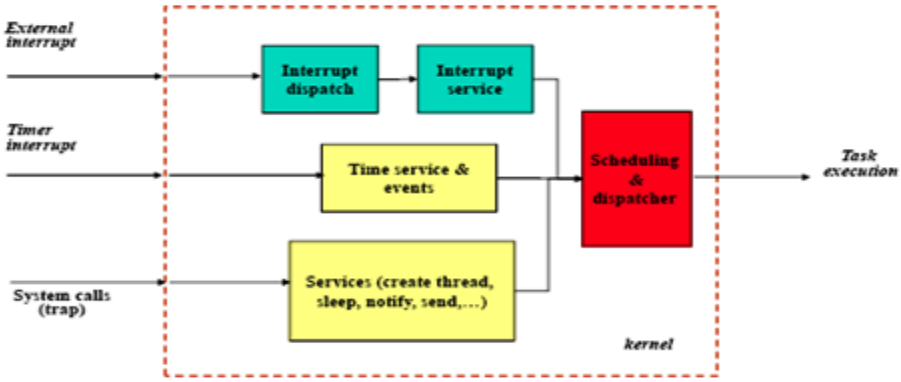
  The services of the OS: Upper bound on the execution time!

  RTOSs must be deterministic:

  • unlike standard Java,

  • upper bound on times during which interrupts are disabled,

  • almost all activities are controlled by scheduler:



Hình 3.14 Module lập biểu của nhân HĐH

2. OS must manage the timing and scheduling

• OS possibly has to be aware of task deadlines;

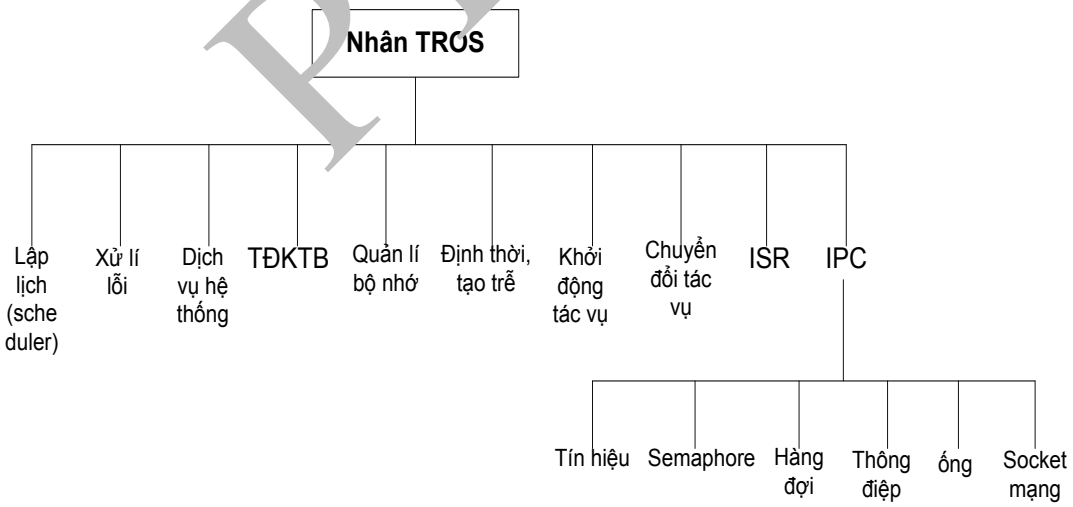
(unless scheduling is done off line).

• OS must provide precise time services with high resolution.

3. The OS must be fast

• Practically important.

- **Chức năng chính của nhân RTOS là** quản lí tài nguyên CPU, bộ nhớ, quản lí tác vụ, I/O, thực thi liên lạc giữa các tiến trình, lập lịch, mức ưu tiên, thời gian, dự đoán tình huống sự kiện.

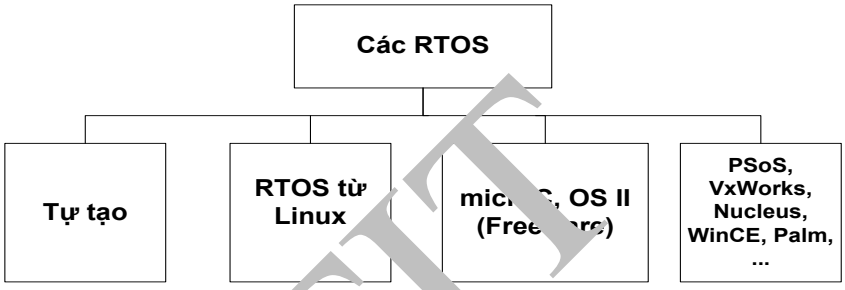


Hình 3.15 Các chức năng nhân RTOS

- Trong qúa trình phát triển HTN với RTOS cần tiến hành gỡ rối mã, và thử nghiệm :
  - ✓ Các chức năng thực thi đa tác vụ (C hay C++);
  - ✓ Đồng hồ định thời mềm (software timers);

- ✓ Đồng hồ định thời cứng;
- ✓ Chó canh cửa (watchdog);
- ✓ Module lập lịch;
- ✓ Điều khiển chen ngang;
- ✓ Ngắt;
- ✓ Ghép nối thiết bị ngoài và các module TĐKTB;
- ✓ Các chức năng IPC;
- ✓ Các chức năng xử lý lỗi;
- ✓ Thử nghiệm phần mềm trung gian dùng để gở rối nhúng trong hệ thống.

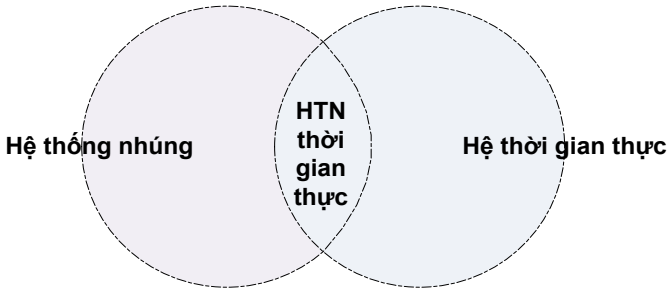
- Các RTOS



Hình 3.16 Các hệ điều hành RTOS

- **HTN thời gian thực**

HTN có thể được tạo thành khi có phần cứng nhúng và phần mềm hệ thống là RTOS. Mối quan hệ giữa hệ thời gian thực và HTN như hình dưới đây:



Hình 3.17 Hệ thống nhúng thời gian thực

Hình cho thấy rằng không phải tất cả HTN đều là HTN thời gian thực và ngược lại không phải tất cả hệ thời gian thực là HTN. Và phần chung chính là biểu diễn của *HTN thời gian thực*.

Còn có các HTN không thời gian thực, và HTN thời gian thực được xây dựng trên hệ thống không thời gian thực. Các HTN kiểu này tương đối phổ biến cho các ứng dụng nhúng không đòi hỏi quá khắt khe về đặc tính, hay cần đáp ứng nhanh. Để xây dựng các hệ như vậy cần có những lựa chọn phần cứng có tốc độ xử lý đủ nhanh, RAM đủ lớn, đáp ứng cho ứng dụng. Phần mềm hệ thống không phức tạp nhưng hiệu quả chạy trình cao. Ví dụ các dòng PC 104 nhúng trên thị trường phổ biến chạy với HĐH DOS 6.4 cũng thỏa mãn cho nhiều lớp ứng dụng nhúng, hay chạy với Linux nhân 2.2 không thời gian thực cũng là lựa chọn hợp lí.

- Một số Hệ điều hành thương mại và mã nguồn mở:
  pSOS+ 248
  pSOS+ kernel 248
  pSOS+m multiprocessor kernel
  pREPC+ runtime support
  pHILE+ file system
  pNA+ network manager
  pROBE+ system level debugger
  XRAY+ source level debugger
  OS-9
  VXWorks
  VRTX-32
  IFX
  TNX
  RTL
  RTscope
  MPV
  LynxOS-Posix conformance
  QNX
  Windows NT
  QNX 4 RTOS
  Windows CE and embedded Linux.
  Palm OS
  Windows CE
  MS-DOS or DOS Clones
  Linux, including RTLinux and MontaVista Linux and Unison OS
  Linux Easysbox

3.2.4 Hệ thời gian thực không có hệ điều hành thời gian thực

Phần này bàn tới việc thiết kế và phát triển một cách xử lí thời gian thực việc thu thập, sắp xếp các dữ liệu khác nhau trong bối cảnh tranh chấp nguồn tài nguyên hệ thống. Tức là bài toán với các hoạt động dữ liệu thời gian thực *mà không có sử dụng phần mềm tinh xảo như RTOS* để giải quyết. Ví dụ như điều khiển ôto ngày nay: lấy mẫu dữ liệu từ nhiều nguồn khác nhau khi xe chạy, lưu, xử lí, hiển thi, … Có nhiều phần mềm máy tính xử lí cài trên ô tô, như điều khiển động cơ (engine management unit (EMU)), hệ thống điều khiển sức kéo trong sự liên quan tới chuyển động của 4 bánh xe… EMU tương tác với các thành phần qua cổng với tốc độ 19,2 kbaub… hãy tính xem loại máy tính nào có thể sử dụng trên ô tô, đủ mạnh để thực các tác vụ như vậy. Đó là phạm trù phần cứng, một HTN đủ mạnh vậy.

- **Chọn môi trường phần mềm (software environment)**

Vậy bắt đầu từ đâu với bài toán như trên ? Thông thường ta thiết kế phần cứng, phát triển phần mềm, cài đặt, thử nghiệm, sửa đổi …cài đặt thử chạy …. Tuy nhiên có một thực tế là các thành phần phần mềm như hệ điều hành, chương trình dịch thì không thể thay đổi và có thể phù hợp với phần cứng. Trong thực tế một quyết định thông minh là phải có sự hài hòa tính tới cả phần cứng và phần mềm, sau đó phần mềm có thứ tự ưu tiên cao nhất. Tại sao ? Có gì quan trọng hơn nếu ta dùng máy tính để điều và kiểm soát một hệ thống thực ? và câu hỏi tiếp theo: HĐH nào sẽ là phần mềm hệ thống như là ứng viên ? RTOS nào đó, MSDOS hay Windows CE … ? Cần 1 phần mềm thời gian thực thỏa mãn các thông số hạn chót (deadline) để duy trì quá trình lấy mẫu dữ liệu, thông qua truy nhập trực tiếp cổng ghép nối (song song) và phải đơn giản, ổn định. Các kĩ sư ngay lập tức có vẽ sẽ chọn RTOS, tuy nhiên RTOS đắt và sẽ sử dụng thế nào để phát triển ở HTN đích. MS DOS có sẵn và cơ bản là cho phép truy nhập phần cứng dễ dàng, trong khi MS WINDOWS kềnh càng, không thể truy nhập trực tiếp phần cứng và tương đối phức tạp và cản trở việc truy cập làm quá tải nguồn tài nguyên, do vậy không hợp, trừ khi được xây dựng mới (WINDOWS CE). Vậy sự lựa chọn ở đây là MS DOS cùng với công cụ Borland C không phí, là thích hợp, mặc dù cả hai có vẽ củ kỉ, nhưng khả năng còn rất tiềm ẩn. Bộ biên dịch Borland C có hệ thư viện hỗ trơ truy nhập cấp thấp và phần cứng và khai thác các trình chạy ở BIOS. Vậy MS DOS có thể thay thế cho một RTOS ? Cái này tùy thuộc vào sự chuyên nghiệp cho hệ đích phát triển. Nói vậy để nhắc tới đặc điểm là MS DOS chỉ thực thi một luồng tác vụ, cho dù có thể gọi các thủ tục và phần mềm có cấu trúc modular, thì vẫn chỉ có 1 thực thi mà thôi. Nếu phải cần tới đa luồng thực thi thì đây là vấn đề khó ! Nhưng có một việc là: hãy 'gói' các đa tác vụ đó vào một tác vụ đơn ! (WINDOWS là đa tác vụ, nhưng không phải là thời gian thực, nhưng nếu khéo tổ chức các phần mềm chạy phối hợp (co-operate) và chia sẻ thời gian xử lí với nhau, thì có thể xử lí thời gian thực được. Tuy nhiên nếu không duy trì được kĩ thuật này, hệ thống sẽ bị rối loạn vì các tranh chấp không thể điều hòa được, ví dụ như một tác vụ độc quyền CPU trong thời gian quá dài, không đáp ứng được yêu cầu hạn chót (*deadline*), các tác vụ chỉ có thể thoát bằng

hết hạn (*time-out)* để tránh bế tắc hệ thống). Dưới đât thử bàn về một số giải pháp để thực hiện thời gian thực hóa một hệ không thời gian thực.

- **Chuyển hóa hiệu năng thời gian thực sang hệ không thời gian thực.** Trước hết hãy phân loại đặc trưng thời gian thực cần thực hiện và sau đó xác định mức độ năng lực xử lí cần thiết. Nhắc lại là thời gian thực có nghĩa tác vụ phải hoàn thành trong khung thời gian đã định (hàng giây, hay phút). Hầu như vấn đề có tính tới hạn (*critical*) ***thường là ở các tác vụ thu thập dữ liệu***, khung thời gian xử lí dữ liệu giữa hai lần lấy mẫu và điều này lại phụ thuộc vào tốc độ lấy mẫu. Vậy thời gian tối thiểu cho tác vụ này là bao nhiêu. Một tính toán khác là tốc độ của CPU: CPU sẽ thực hiện bao nhiêu lệnh giữa các lần lấy mẫu dữ liệu, mỗi lệnh chi phí bao nhiêu CPU_{CLK} và qui ra thời gian toàn bộ. Ví dụ với CPU 80386 chạy ở CPU_{CLK} = 30MHz (T_{CLK}= 33 ns), tốc độ lấy mẫu là 30Hz (t_s= 33ms), thì giữa 2 lần lấy mẫu sẽ trôi qua 10^6 T_{CLK} . Nếu 1 lệnh máy mất 10 T_{CLK}, CPU có thể chạy 100.000 lệnh. Cho một tác vụ cụ thể nào đó, có thích nợp ? Nếu OK, CPU không nằm trong vấn đề, và ngược lại tốc độ CPU chưa thỏa mãn.
- **Chọn phần cứng.** Chọn một phần cứng không có khó khăn. Những bộ vi điều khiển (microcontroller) không phải là lựa chọn do có định hướng thiết kế, vì các chức năng giới hạn, do bộ nhớ nhỏ, tốc độ chậm. Bo mạch máy tính (based micro-computer board) sẽ là lựa chọn phù hợp vì có bộ nhớ mở rộng, có thiết bị lưu trữ ngoài đa dạng (đĩa mềm, đĩa cứng, USB) để lưu dữ liệu cho các nhu cầu thống kê, phân tích. Các bo mạch này lại rất nhỏ, cấu trúc modular, dễ bổ sung, ví dụ loại PC 104 chuẩn công nghiệp (như đã giới thiệu ở chương 1). Hơn nữa là khả năng cài đặt các phần mềm.
- **Lập lịch lấy mẫu dữ liệu.** Việc thu thập dữ liệu thường kì trong những khoản thời gian cố định cho các chu kì lấy mẫu là thông thường ở các hệ thống có định thời. Có nghĩa là các chu kì lấy mẫu phải kiên định sao cho thời gian lấy dữ liệu không bị sai lệch, như vậy dữ liệu sẽ đúng như hoài vọng (tức dữ liệu đúng các thời điểm cần lấy). Có thể có nhiều kỉ thuật đảm bảo bài toán này, ví dụ: chương trình lấy dữ liệu cho chạy tự do, còn số liệu thu được sẽ được so sánh và hiệu chỉnh với đồng hồ định thời lấy mẫu, hoặc phần mềm hiệu chỉnh thời điểm lấy mẫu chính xác hơn sau khi làm bước trước đó, tức có dự đoán khi nào sẽ có số liệu, thì chuẩn bị lấy mẫu. Từ đó chu kì lấy mẫu sẽ được tính sau bao lâu thì lấy số liệu và dự đoán khi nào nguồn sẽ có dữ liệu. Tuy nhiên cách này cũng có một vài vấn đề:
    - ✓ Chu kì lấy mẫu sẽ không giống nhau từ một hệ này sang hệ khác và phụ thuộc vào tốc độ CPU, điều khiển cổng ghép nối với nguồn dữ liệu.
    - ✓ Thời gian lấy mẫu phụ thuộc vào xử lí mẫu và chu kì lấy mẫu sẽ biến động và có ảnh hưởng tới độ chính xác mẫu lấy được.
    - ✓ Khi đề cập tới hiện thị kết quả, đặc biệt là giá trị tức thời sẽ có sự chênh lệch thời điểm (bị trễ).
    - ✓ Để khắc phục, việc sử dụng đồng hồ thời gian thực (*real-time clock- RTC*) trong hệ thống làm điểm qui chiếu là cần thiết, đặc biệt khi lập biểu. Các PC, PC 104 …

đều có đồng hồ này và dùng cách gọi hàm sẽ nhận được các giá trị biểu diễn ở cấp ms, là rất chính xác cho các sự kiện thực. Đọc được thời điểm hiện tại, so sánh với thời điểm trước đó, xác định thời gian đã trôi qua. Nếu thời gian này bé hơn chu kì lấy mẫu, đọc lại, so sánh… Nếu thời gian đó bằng với độ dài chu kì, lưu thời điểm này lại, chuyển điều khiển tới code lấy mẫu. Chu trình này sẽ được lặp lại trong suốt quá trình lấy dữ liệu. Tuy nhiên giải pháp này có thể sẽ có chênh lệch thời gian nếu chu kì lấy mẫu tính bằng ms, RTC trả lại cũng là ms ! Đây là câu hỏi dùng cái gì đo cái gì ! Nói cách khác nếu RTC trả lại ns, chắc sẽ tốt hơn. Hay nếu chu kì lẫy mẫu tính là giây, vài giây, thì sẽ tốt hơn. Nếu đồng hồ RTC của máy tính không đủ độ phân giải do đó không phải là lựa chọn, thì có giải pháp khác sẽ được sử dụng: Cơ chế ngắt (DOS INT: *INT 15 - AH = 86h SYSTEM - WAIT (μs) máy AT,XT2,XT286,CONV,PS*) đưa tác vụ vào trạng thái "ngủ" (sleeep) hay "đợi" (wait) một thời gian, sau đó đánh thức khi thời gian cần đã đạt giá trị. Thời gian tính sẽ qui vào lệnh, tính theo μs, cho nên có độ chính xác cao. Chu kì thời gian sẽ tính cho từng hệ cụ thể. Giải pháp khác là dùng đồng hồ phần cứng độc lập (timer) phát sinh ngắt theo chu kì để lấy mẫu dữ liệu, ở đây cần viết code xử lí ngắt cẩn thận và tối ưu để có thể kết thúc sớm trước khi sang ngắt mới, không làm quá tải CPU cho các tác vụ khác ! (Ngoài ra DOS còn có một ngắt báo thời gian hệ thống, xẩy ra 18.2 lần/giây (18.2 Hz), cũng là giải pháp cho cập nhật dữ liệu.

- **Lấy mẫu dữ liệu.** Để lấy dữ liệu, cần lựa chọn kỉ thuật ghép nối từ các cảm biến vào hệ thống. Có vài kỉ thuật cơ bản, đó là dữ liệu được chuyển vào khi các bộ biến đổi đã chuyển từ tương tự sang số. Việc chọn truyền số liệu vào hệ có ảnh hương rất lớn tới thời gian lấy dữ liệu toàn phần. Truyền thông có thể là nối tiếp (*serial line*), song song (*parallel-bus*), sau đó là cách kích hoạt chu trình phần mềm: chu kì (*polling*) theo đồng hồ như trình bày ở trên hay chủ động qua ngắt mỗi khi dữ liệu đã sẵn sàng ở đầu ra ADC. Đối với nhận dữ liệu nối tiếp, thường dữ liệu đã có định dạng kiểu gói với số byte nhất định, việc nhận các byte liên tiếp phụ thuộc nhiều vào tốc độ thu/phát, có hay không có đối thoại và khi hệ lấy từ nhiều nguồn (cảm biến) dữ liệu theo kiểu polling, chi phí thời gian sẽ là vấn đề lớn. Đối với các CPU tốc độ cao, việc này có thể giải quyết được. Trong khi dùng ngắt, cần code ở mức thấp để nhận biết số hiệu vector ngắt, kích hoạt chương trinh xử lí ngắt (ISR). Dùng ngắt có thể nhận cả khung dữ liệu, nhanh hay chậm là do bên phát dữ liệu. Các UART hiện tại thể chạy tới 115000 kbaud và để thích ứng cần một FIFO nhận dữ liệu đủ lớn. Việc quản lí FIFO sao cho không bị tràn và hoạt động trôi chảy, đặc biệt với polling khi cần lưu nhiều mẫu dữ liệu từ nhiều nguồn. Khi viết code cho polling, cần debug để tránh các sự cố khi khai thác FIFO.
- **Một số lưu ý.**
  - ✓ Sai số tính toán thời gian do phần mềm thực hiện thông qua các phép biến đỗi hệ cơ số (ví dụ đưa vào số hexa, tính toán là số nguyên …), sai số tích lũy khi làm

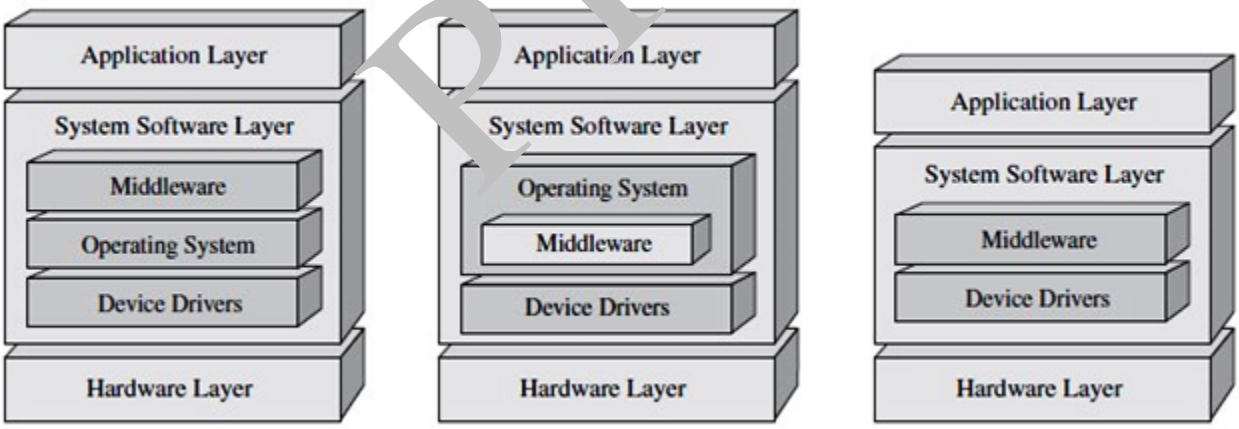
tròn ở các phép tính số học. cascv chương trình dịch thường dùng các phương pháp tiệm cận gần đúng nên sai số rất đáng kể. Để tránh sai số, cần thực nghiệm các giá trị cần có để nạp các thông số chuẩn cho các hàm tính toán thời gian.

- ✓ Dữ liệu bị xáo trộn cũng có thể xảy ra. Vì vậy trước một chu trình lấy mẫu dữ liệu mới, cần làm sạch bộ đệm (FIFO), kích thước bộ đệm đủ lớn để tránh bị tràn dữ liệu, quản lí con trỏ vào bộ đệm phải thật chính xác, tránh bị đảo lộn thứ tự của dữ liệu. Các thao tác xóa, tái khởi động phải thực hiện đúng vị trí. Khi sử dụng bộ đệm FIFO như các hàng đợi gán cho các tác vụ riêng biệt, việc truy cập đọc/ghi phải chính xác ứng với các nguồn cung cấp (sensor+ADC+UART) dữ liệu vào.

3.3 PHẦN MỀM TRUNG GIAN (middleware)

Trong giới hạn nhất định, phần mềm trung gian (gọi tắt PMTG) là bất kì phần mềm nào không phải là phải của nhân HĐH, TĐKTB hay phần mềm ứng dụng. Nhưng cũng có một số HĐH hợp nhất PMTG vào HĐH (nhưng không vào nhân HĐN !), như hình 3.2 đã đề cập. Ví dụ với các HĐH máy để bàn, rất nhiều PMTG có sẵn khi bán HĐH (các dịch vụ, mạng, Sun embedded Java, Microsoft's .NET Compact Framework, CORBA của Cty Object Management Group (OMG) … và đã hợp nhất với HĐH, mang lại cho người dùng có ngay các tiện ích khi sử dụng. Phần mềm trung gian không hợp nhất vào phần mềm ứng dụng, là để chia sẻ sử dụng cho nhiều ứng dụng đồng thời, hay tái sử dụng ở các môi trường HĐH khác nhau.

Trong HTN, PMTG được coi như phần mềm hệ thống, mà vị trí có thể theo các mô hình sau:



Hình 3.18 Vị trí của PMTG ở HTN

Trên hình ta nhận thấy PMTG giống như cầu nối giữa các phần nềm khác của phần mềm hệ thống, cung cấp các dịch vụ cho các phần mềm ứng dụng, như: an ninh hệ thống, kết nối mạng, tryền thông cục bộ giữa các ứng dụng trong hệ thống, mang lại sự linh hoạt khi triển khai các ứng dụng. Với vị trí "trung gian", các PMTG làm giảm đáng kể tính phức tạp của các ứng, vì các tiện ích đã có sẵn và chia sẻ ngay trong PMTG. Tuy nhiên khi đưa PMTG vào hệ thống cũng là tăng thêm một lớp xếp chồng, có tác động đáng kể vào tính mở rộng, hiệu năng của các HTN, vì PMTG tác động vào tất cả các lớp phần mềm khác.

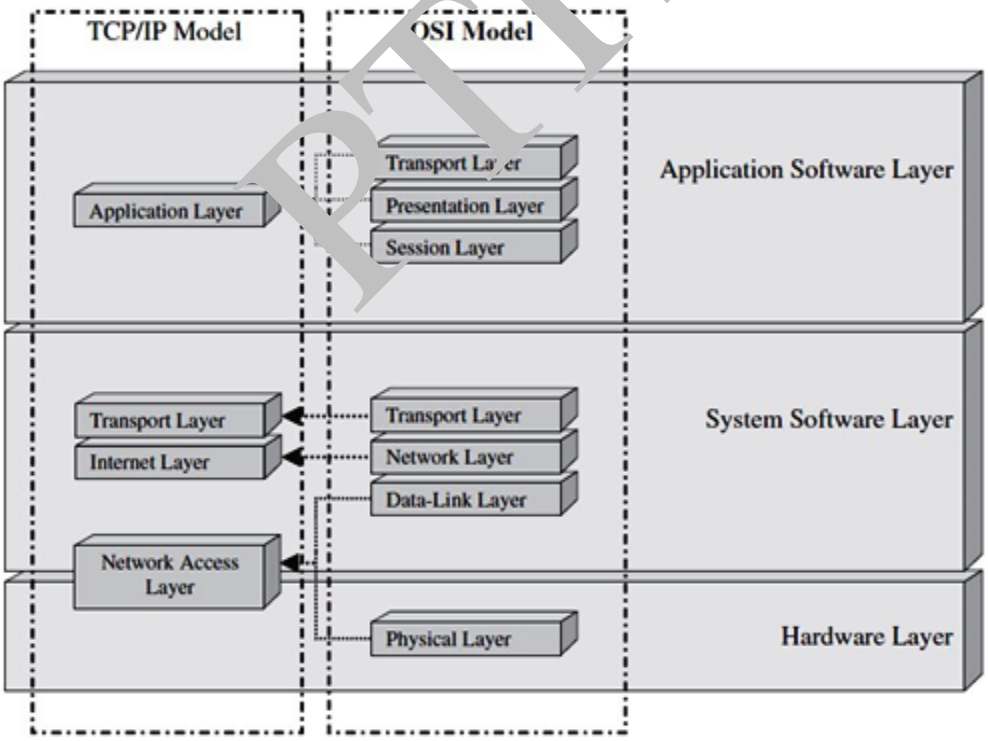
Có nhiều kiểu PMTG, như PMTG hướng thông điệp (message oriented middleware (MOM)), môi giới đối tượng (object request brokers (ORBs)), thủ tục gọi từ xa (remote procedure calls (RPCs)) , Truy nhập cơ sở dữ liệu (database/database access) và các lớp giao thức mạng (networking protocols). Tuy nhiệ khi phân loại có hai tập hợp cơ bản:

- ✓ Đa năng (*general-purpose)*: được cài đặt trên nhiều HTN khác nhau, ví dụ giao thức mạng, máy Java ảo (JVM). Hai thành phần này đều có trên các thiết bị mạng hiện đại. như router, Concentrator, Switch, các thiết bị dạng SOHO,…
- ✓ Xác định (*market-specific)*: dành cho các yêu cầu riêng biệt cho các HTN riêng biệt. Ví dụ PMTG trên các máy thu hình kĩ thuật số hỗ trợ kết nối mạng TCP/IP, được xếp trên HĐH và JVM.

Các PMTG thường là sản phẩm của riêng các công ty, cho các HTN của công ty chế tạo và có bản quyền sở hữu trí tuệ. Cũng có một số là phần mềm mã nguồn mở được hỗ trợ bởi các tổ chức công nghiệp sử dụng theo giấy phép GPL, ví dụ Linux và các PMTG chạy với Linux.

Ví dụ các PMTG trong các HTN

Tùy vào phạm trù ứng dụng và qui mô xây dựng, HTN có thể có PMTG cũng có thể không. Dưới đây là các ví dụ các HTN cần có PMTG, như ở các thiết bị truyền thông nối mạng, thì các module và thiết bị mạng là cần thiết:



Hình 3.19 Mô hình các lớp mạng theo TCP/IP, OSI và ánh xạ vào HTN

Lớp 4, 3, 2 của mô hình OSI được đặt trong lớp phần mềm hệ thống, nhưng không thuộc nhân HĐH nhúng. Các lớp 5, 5, 7 thuộc phần mềm ứng dụng, lớp 1 thuộc thành phần phần cứng.

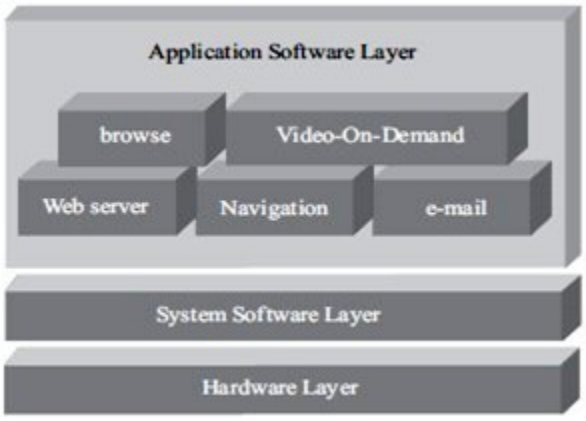
Ngày nay hầu hết các ứng dụng WEB đề có sự hỗ trợ của Java, cho nên ở các HTN đều có vài Java Virtual Machine.

3.4 PHẦN MỀM ỨNG DỤNG

Phần mềm ứng dụng chính là mục đích để chạy trên HTN. HTN được ứng dụng ở nhiều lĩnh vực khác nhau, do đó các ứng dụng là rất cụ thể, và được phát triển bởi nhà chế tạo ra HTN. Ví dụ trong công nghiệp chế tạo các thiết bị cho tự động hóa, robot,… các HTN có những bài toán riêng để giải quyết, tức là phải phát triển phần mềm cho bài toán đó.
Ví dụ HTN kín: để một bộ chuyển mạch mạng (switch) hoạt động được, ngoài một phần mềm nhân điều khiển, PMTG mạng, cần phát triển các module như:

- Chuyển gói dữ liệu và/ra ở một cổng;
- Quản lí buffer vào/ra của mỗi cổng;
- Truy vấn địa chỉ MAC trong gói để tìm cổng [illegible];
- Chạy phần mềm chọn đường đi từ cổng vào tới cổng [illegible];
- Điều khiển đóng các công tắc trong ma trận công tắc điện tử tạo ra đường nối từ cổng vào đến cổng ra;
- Cập nhật MAC vào bảng cơ sở dữ liệu theo cấu trúc các MAC thuộc một cổng;
- Quản trị thông lượng;
- Kết nối với hệ thống quản trị qua giao thức SNMP với tư cách là Agent;
- …

Ví dụ khác HTN mở: trong "ngôi nhà thông minh", các HTN kiểm soát các khu vực với các tác vụ cụ thể, sau đó sẽ nối vào máy chủ gia đình và máy chủ gia đình nối Interrnet. Login vào máy chủ qua môi trường ứng dụng WEB, chủ nhà có thể kiểm soát nhà mình (qua video), hay ra lệnh thực hiện một công việc nào đó trên một HTN nào đó. Và ta có mô hình sau đây:



Hình 3.20 Các ứng dụng WEB trong HTN, đặt ở lớp phần mềm ứng dụng.

Phần mềm ứng dụng ở đây là các phần mềm liên quan tới Internet: Web server, Navigation, e-mail, Video-On-Demand …

Phát triển PMTG và phần mềm ứng dụng nhúng là đề tài nằm ở phần khác, ở đây không đề cập chi tiết.

3.5 KẾT CHƯƠNG

Chương 3 đề cập những vấn đề cơ sở của phần mềm hệ thống bao gồm:

- *Trình điều khiển thiết bị (TĐKTB)*. Trong khi chương 2 đề cập tới các giải pháp ghép nối thiết bị ngoài vào hệ thống, thì phần này đề cập tới cách nhìn nhận một số nét đặc tả và các bước khi viết chương trình điều khiển thiết bị. TĐKTB là phần phát triển độc lập trên các phần cứng cụ thể, hợp nhất với phần mềm hệ thống và phần mềm ứng dụng để truy nhập và trao đổi dữ liệu giữa hệ thống với các thiết bị ngoài.
- *Phần mềm hệ thống***.** Phần mềm hệ thống được mô tả thông qua các bước: giới thiệu các khái niệm cơ bản khi sử dụng mô tả về HĐH. Đặc biệt là HĐH đa trình, môi trường phần mềm hệ thống được cài đặt phổ biến trong các HTN. Các khái niệm này rất cần thiết khi viết chương trình ứng dụng cho HTN, các sử dụng các hàm chức năng của HĐH thông qua API kiểu lời gọi hàm (GHT). Những vấn đề liên quan tới hệ thời gian thực, những đặc thù cơ bản khi đề cập tới hệ thời gian thực cứng cần được mô tả. Triển khai một hệ thời gian thực trên một hệ thống nhúng yêu cầu những giới hạn và việc chuyển hóa phần mềm hệ thời gian thực lên hệ thống nhúng tạo ra một hệ thống nhúng thời gian thực.
- *Phần mềm trung gian và phần mềm ứng dụng*. Sau khi có TĐKTB, phần mềm hệ thống và viết các phần mềm ứng dụng, HTN đã có thể hoạt động. Tuy nhiên khi HTN nằm trong một tổng thể hệ thống lớn hơn, thì một lớp phần mềm khác, độc lập, dùng chung là cần thiết, đó là các phần mềm trung gian. Các phần mềm trung gian có xu hướng tiêu chuẩn hóa, mở, nhằm tạo cơ sở để liên kết, kết nối các HTN lại với nhau, ví dụ như mạng máy tính. Có nhiều phần mềm trung gian có thể tích hợp vào HTN, phụ thuộc vào nhu cầu và thực tế triển khai. Các phần mềm ứng dụng là thể hiện "mềm" các ý tưởng khi thiết kế một HTN và đó chính là lí do phát triển HTN. HTN ứng dụng cho mục đích gì, cần tạo ra phần mềm ứng dụng cho HTN đó, chính vì thế số lượng, chủng loại, qui mô, mức độ "thông minh" ngày nay rất lớn, và phát triển rất nhanh.

3.6 CÂU HỎI CUỐI CHƯƠNG

1.Nêu định nghĩa TĐKTB.

2.Đưa các mô vào mô hình cho thấy vị trí của TĐKTB nằm ở đâu trong kiến trúc hệ thống (HTN không có HĐH, HTN có HĐH).

3. Nêu ra các chức năng của TĐKTB.

4. Trong các CPU hỗ trợ nhiều chế độ hoạt động (*user mode, supervisor mode*), thì TĐKTB chạy trong chế độ nào ? Tại sao ?

5. Tổng kết các đặc tả về hệ thời gian thực

6. ổng kết các đặc tả về hệ điều hành thời gian thực (RTOS).

7. Thế nào là chen (preemptive) ngang trong hệ đa trình ? Cho ví dụ của hiện tượng chen ngang.

8. Giải thích hoạt động của tiến trình theo mo hinh trạng thái của tiến trình.

9. Thế nào là hạn chót (*deadline*), kĩ thuật giải quyết hạn chót như thế nào ?

10. Lập lịch là gì trong các hệ đa trình? Ứng dụng lập lịch trong hệ thời gian thực như thế nào ?

11. Nêu các quan điểm để thực hiện lập lịch.

12. "Chó canh chừng" là gì ? Đóng vai trò gì trong hệ thời gian thực ?

13. Thiết kế một giải thuật cho chó canh chừng với các điều kiện giả định tùy ý.

14. HTN có cần chó canh chừng, cho dù HTN có hay không có hệ điều hành ?

15. HTN chạy độc lập, không có sự giám sát của con người. Nếu hệ bị "quẩn", làm sao thoát ra tình trạng đó ?

Chương 4 THIẾT KẾ VÀ CÀI ĐẶT CÁC HỆ THỐNG NHÚNG

Những vấn đề kỹ thuật liên quan tới HTN đã được trình bày ở các chương trước, đặc biệt là phần cứng và những khái niệm về phần mềm, TĐKTB, phần mềm hệ thống, các phần mềm hỗ trợ như PMTG, và phần mềm ứng dụng. Chương tiếp theo đưa ra một số ý tưởng thiết kế các HTN, và phần cuối là cách thiết kế phần cứng và cài đặt phần mềm cho một HTN.

Những điểm quan tâm khi thiết kế HTN bao gồm:

- Một hệ với các thành phần hợp thành:
  - Processor
  - Memory
  - Peripherals
- Hệ tích hợp:
  - Microcontroller
  - Expanded microcontroller
  - Microprocessor based
  - Board based
- Phần mềm:
  - Phần mềm hệ thống
  - Phần mềm ứng dụng nhúng, các giải thuật ứng dụng nhúng.

4.1 THIẾT KẾ HỆ THỐNG

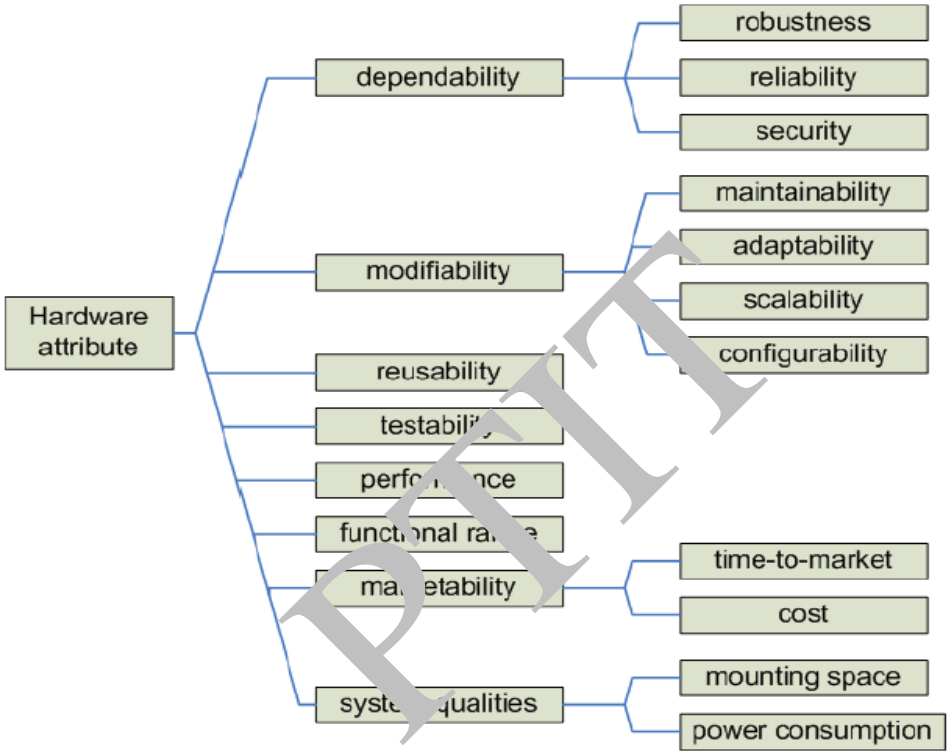
HTN hợp nhất các thành phần phần cứng và phần mềm và người phát triển HTN phải có kỷ năng ở cả hai chủ đề này. Các kỷ năng gồm khả năng thích ứng và cách tiếp cận có tính hệ thống, để đưa ra một quyết định thiết kế trong khi phải lựa chọn các phần cứng khác nhau cho ứng dụng nhúng xác định. Các phần cứng phổ biến hiện nay bao gồm hệ thống xây dựng trên CPU ví dụ kiểu vi điều khiển (MCU-microcontroller) hay CPU xử lí tín hiệu (DSP-digital signal processing) và các hệ thống phần cứng khả lập trình (PLD-Programmable Logic Device, CPLD-Complex Programmable Logic Device), mảng các phần tử cổng logic khả trình (FPGA-Field Programmable Gate Array).

Phân tích qui trình lựa chọn phần cứng: Việc quyết định chọn phần cứng cho HTN đi ra từ những kinh nghiệm có được từ các dự án và các hiểu biết có tính chuyên gia của lĩnh vực HTN và còn chịu ảnh hưởng của nhiều yếu tố khác. Trước tiên, phần cứng phải có thể thực hiện được các chức năng mong muốn đặt ra bởi các yêu cầu thiết kế. Các thuộc tính phần cứng khi xác định các yêu cầu gồm phạm trù các chức năng - số lượng các chức năng, kích thước của các cấu trúc dữ liệu, có thể thực hiện trên phần cứng và nếu có tính thời gian thực thì đó là hiệu năng - thực hiện các chức năng nhanh đến mức nào. Còn một số yêu cầu khác

không mang tính chức năng như: độ tin cậy, bảo trì, tiêu dùng năng lượng, khả kiểm tra hệ thống và cạnh tranh trên thị trường.

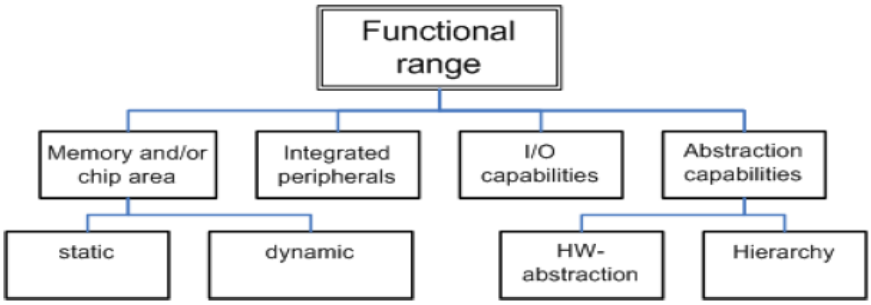
Qui trình chọn phần cứng: Kết nối giữa chất lượng của hệ thống với thuộc tính phần cứng. Các thuộc tính phần cứng được trình bày như hình dưới đây:

Hệ thống thuộc tính phần cứng: Phần cứng nhúng thông thường ít có tính chuẩn, do các lớp ứng dụng đa dạng. Tuy nhiên có một số yêu cầu chung được áp dụng:

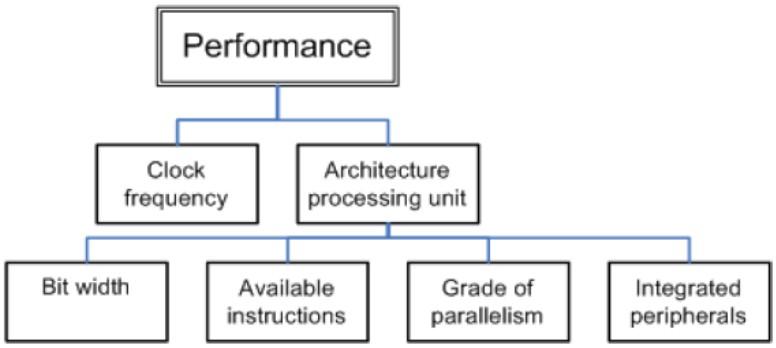


Hình 4.1 Các thuộc tính chung của phần cứng của một HTN

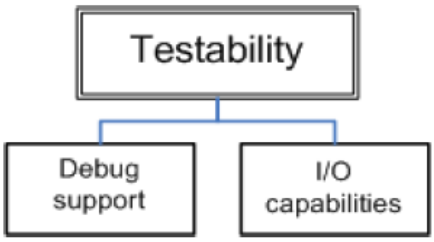
Thuộc tính các chức năng phần cứng:



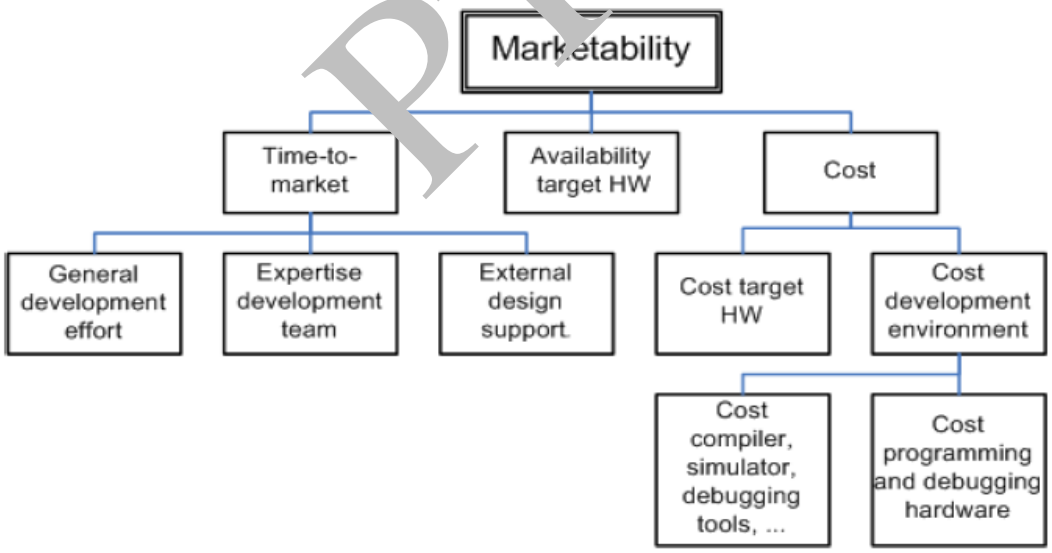
Thuộc tính hiệu năng:



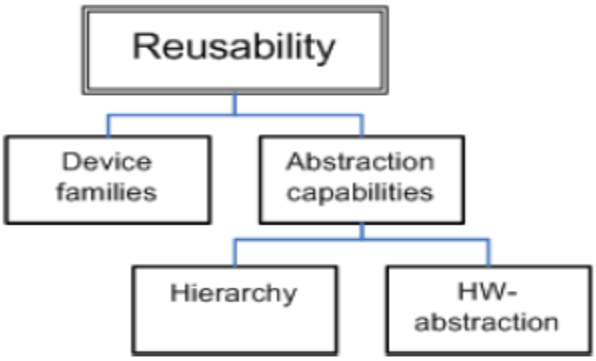
Thuộc tính kiểm tra được phần cứng:



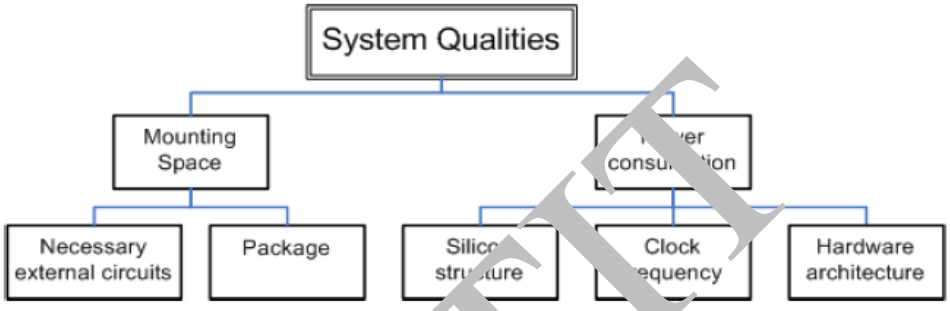
Thuộc tính thị trường của phần cứng:



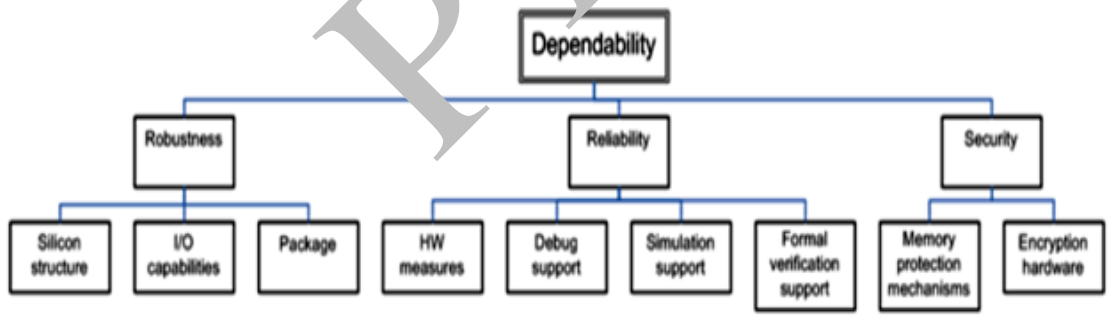
Thuộc tính tái sử dụng của phần cứng:



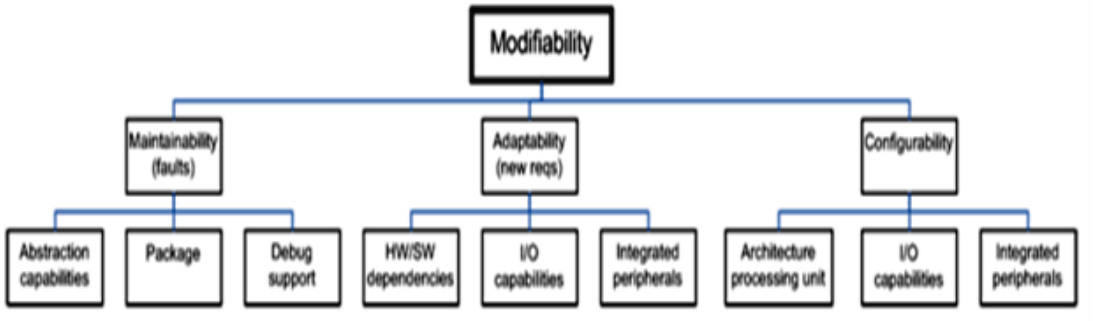
Thuộc tính chất lượng của phần cứng hệ thống:



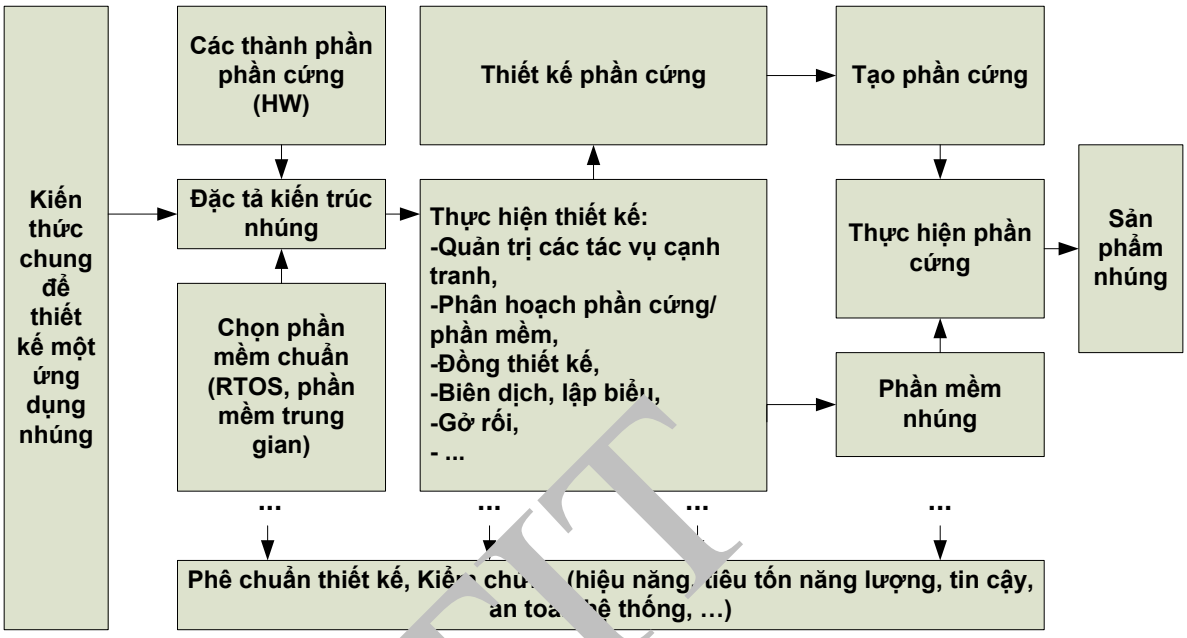
Thuộc tính độc lập của phần cứ



Thuộc tính sử đổi được của phần cứng:



Khi thiết kế một sản phẩm điện tử thông minh nói chung hay HNT nói riêng, có thể có những điểm xuất phát khác nhau và phương pháp tiếp cận khác nhau, dưới đây đưa ra một trong các cách tiếp cận đó. Có thể tóm tắt phương pháp luận thiết kế như sau:



Hình 4.2 Một kiểu đặc tả tiền thiết kế HTN

4.1.1 Các nền tảng cơ bản khi xây dựng kiến trúc HTN

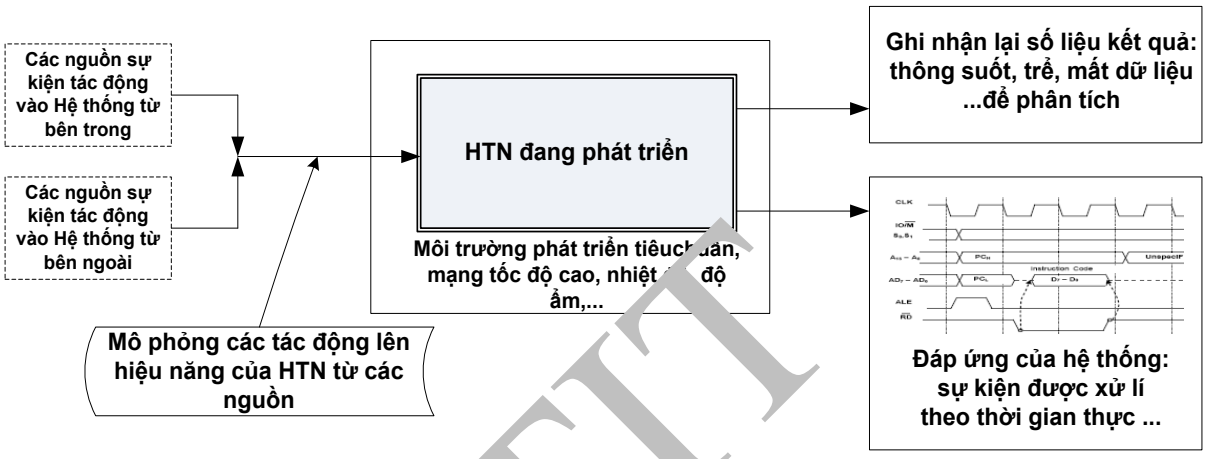
1) Cần kiến thức tốt về phần cứng (Thiết kế logic, kiến trúc máy tính, kiến trúc CPU, ngoại vi, hệ điều hành …). Hiểu biết tốt và các thành phần hợp thành phần cứng của một hệ thống nhúng, có khả năng hiểu và kiểm soát các thiết bị nối vào HTN (hình 3.17).

2) Sự tương tác với thị trường vào quá trình xây dựng HTN:

- ✓ Nhu cầu của thị trường ảnh hưởng tới kiến trúc của HTN và không chỉ giới hạn ở kĩ thuật, công nghệ;
- ✓ Cần nhận ra các yêu cầu của thị trường có tác động vào qui trình thiết kế, bao gồm: kĩ thuật, xu hướng thương mại, ảnh hưởng của chính trị, xã hội. Điểm này gọi là chu kì kiến trúc thương mại của HTN (Architecture Business Cycle).
- ✓ Từ nhận thức các yêu cầu, đưa ra giải pháp về phần cứng/phần mềm thông qua các bước sau:
  - Định nghĩa tập các kịch bản mà tập đó phác thảo mỗi một trong những yêu cầu,

- Đưa ra chiến thuật có thể áp dụng cho mỗi kịch bản, có thể dùng để tạo ra hệ thống như mong muốn,
- Sử dụng kịch bản để phác thảo các chức năng hệ thống cần có, sau đó lên danh mục các thành phần phần cứng, phần mềm thực hiện các chức năng đó.

Ví dụ xây dựng một kịch bản về hiệu năng hệ thống nhúng:



Hình 4.3 Kịch bản mô phỏng hiệu năng khi thiết kế HTN

✓ Định ra các phần cứng, phần mềm có thể đáp ứng với yêu cầu của thị trường.

3) Định nghĩa mẫu kiến trúc (architecrute pattern) và mô hình qui chiếu:

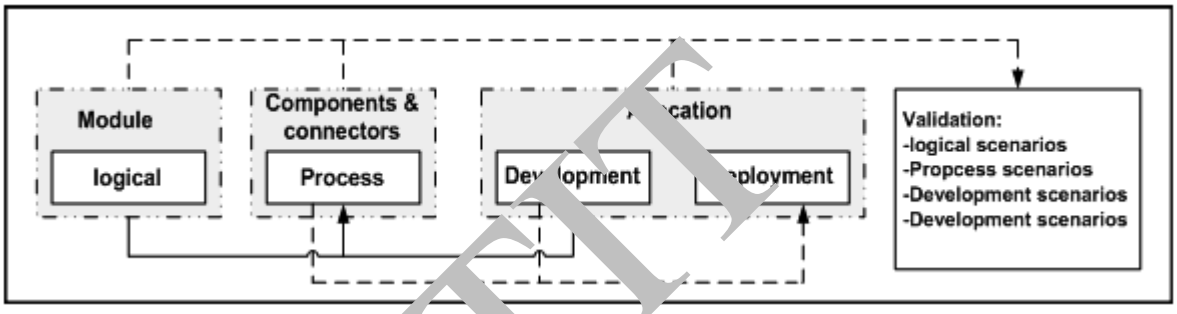
Mẫu kiến trúc hệ thống hay còn gọi là phong cách kiến trúc hệ thống thực chất là một mẫu mô tả (profile) của hệ thống, chứa đựng các đặc tả khác nhau về các thành phần phần cứng và phần mềm, chức năng của các thành phần đó bên trong hệ thống, một sơ đồ bố cục (topo layout) hay còn gọi là mô hình qui chiếu, các liên kết giữa và giao diện ghép nối giữa các các thành phần đó. Các mẫu thiết kế được tạo dựa trên phần cứng và các thành phần dẫn xuất từ các yêu cầu chức năng hay không chức năng qua các thiết kế ban đầu (protopype), các kịch bản hay các chiến thuật nói trên.

Profile sau đó hợp nhất cùng với các mô hình phần cứng, mô hình phần mềm để có được một thiết kế cụ thể.

4) Định nghĩa các cấu trúc có tính kiến trúc

Tiếp theo bước 3) là tạo ra kiến trúc của HTN. Kiến trúc HTN sẽ được hình thành bằng cách phân định toàn bộ HTN thành các thành phần phần cứng, phần mềm, sau đó các thành phần đó lại được phân định đến chi tiết. Sự phân định được biểu diễn bởi tổ hợp của các cấu trúc khác nhau và các mẫu tạo ra ở điểm 3) nói trên được sử dụng cho việc xây dựng một cấu trúc có tính kiến trúc của HTN. Một số kĩ thuật được sử dụng phổ biến trong công nghiệp để thiwjc hiện quiu trình trên là Rational Unified Process, hay Attribute Driven Design. Ví dụ mô hình "4 + 1":

Mô hình "4+1" nói rằng người kiến trúc hệ thống có thể tạo ra ít nhất 5 cấu trúc đồng thời, mỗi cấu trúc có cái nhìn khác nhau về hệ thống. Trong đó 4 cấu trúc thu nhận các yêu cầu khác nhau của hệ thống, cấu trúc thứ 5 là để phê chuẩn tính hợp thức của thiết kế.



Hình 4.4 Các cấu trúc kiểu "4+1"

- ✓ Cấu trúc *logical* là cấu trúc kiểu modul đặc tả các thành chức năng phần cứng, phần mềm, mối liên hệ tương tác chức năng giữa chúng, mà hệ thống yêu cầu. Các thông tin ở đây sẽ làm nền để xây dựng một hệ thống thực tế;
- ✓ Cấu trúc *process* bao gồm các cấu trúc thành phần và cấu trúc kết nối, phản ánh tính tương tranh và đồng bộ của các tiến trình trong hệ thống có hệ điều hành. Cấu trúc này mô tả các yêu cầu phi chức năng (như hiệu năng, tính hợp nhất hệ thống, nguồn tài nguyên sẵn có, v.v) thích ứng cho hệ điều hành. Bằng cách nhìn từ tiến trình, cấu trúc này phác thảo ra các tiến trình trong hệ thống, cơ chế tạo cách lập lịch, cơ chế quản trị tài nguyên.
- ✓ Cấu trúc *allocation* mô tả cách ánh xạ phần cứng/phần mềm vào môi trường phát triển hợp nhất (*integrated development environment-IDE*) với các công cụ: gởi rối, biên dịch, liên kết, sử dụng cho ngôn ngữ lập trình hợp ngữ hay bậc cao (C, C+).
- ✓ Cấu trúc *deployment* là mô tả làm thế nào để triển khai phần mềm vào phần cứng. Phần này cho biết các phần cứng cần có khi phần mềm được xây dựng đòi hỏi:

thực hiện code, xủ lí dữ liệu, tốc độ của CPU phải đạt ở mức tối thiểu, tốc đọ BUS, tốc độ trao đổi dữ liệu …

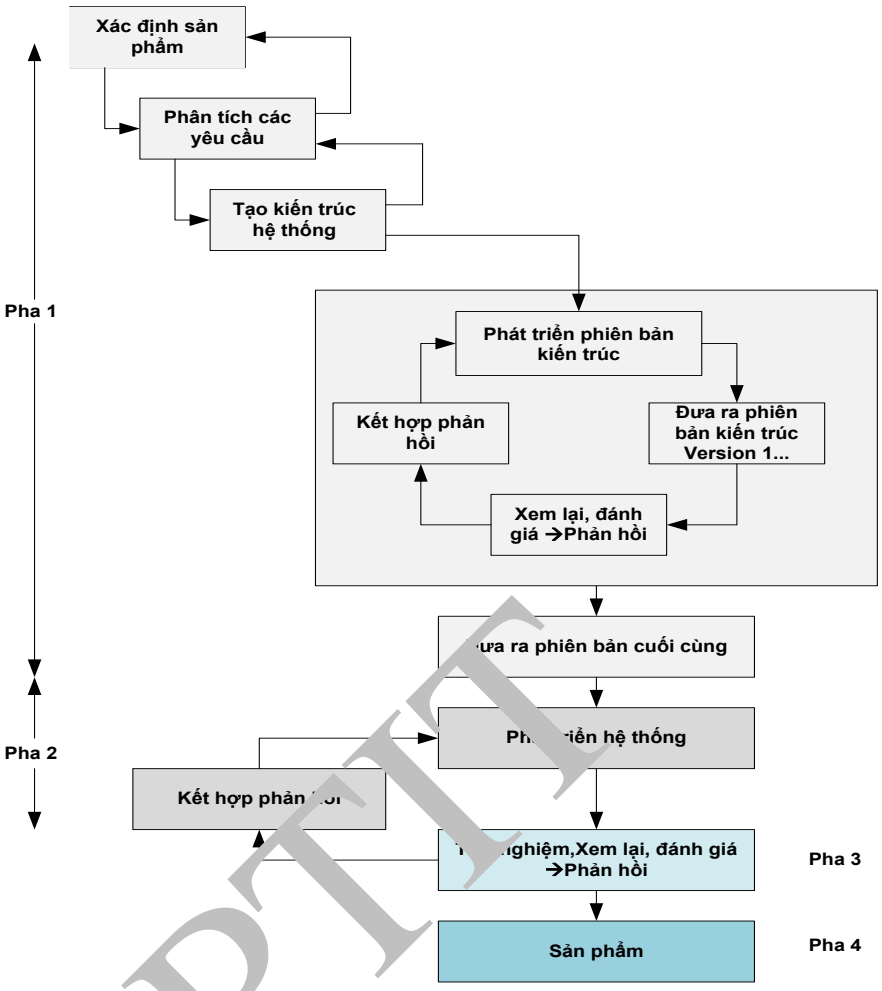
Với mô hình được định nghĩa như trên, thì mô hình có phần mềm và hệ thống có hệ điều hành. Kiến trúc module cho phép áp dụng mà không cần phải biết các phần mềm nào có trong hệ thống hay ngay cả khi hệ không có phần mềm hệ thống (như một số HTN đơn giản).

- ✓ Thành phần "+1" là việc ánh xạ tập các kịch bản quan trọng nhất và các chiến thuật từ 4 cấu trúc. Điều đó đảm bảo các thành phần khác nhau của 4 cấu trúc không có sự xung đột lẫn nhau, dẫn đến việc phê chuẩn hệ thống.

5) Phân tích và đánh giá kiến trúc và các pha thiết kế

- ✓ Theo hướng tiếp cận kiến trúc: kiến trúc có đạt các yêu cầu;
- ✓ Theo hướng chất lượng thiết kế (chất lượng và số lượng): Các kiến trúc khác nhau với yêu cầu có cùng chất lượng như nhau. Đánh giá xu hướng rủi ro hệ thống, hỏng hóc có thể;
- ✓ Hiệu chỉnh, tinh chỉnh kiến trúc và áp dụng vào phần cứng, phần mềm.

Các bước này được thực hiện như điều phản hồi và làm cho đến lúc đạt các tiêu chí trước khi đưa ra sản xuất.



Hình 4.5 Các pha thiết kế HTN

6) Viết tài liệu

- ✓ Tài liệu về toàn bộ hệ thống theo các chuẩn tài liệu.
- ✓ Tài liệu về từng cấu trúc
- ✓ Tài liệu tổng thể về kiến trúc hệ thống.

4.1.2 Phân hoạch thiết kế phần cứng, phần mềm

Tổng quan: Các HTN phản ứng thời gian thực (reactive real-time), hoạt động với cường độ cao, là các hệ thống hỗn hợp phần cứng và phần mềm, sử dụng các bộ vi xử lí, vi điều khiển, xử lí tín hiệu (DSP), trong đó phần mềm hỗ trợ tính mềm dẻo linh hoạt, phần cứng phải đảm bảo tính hiệu năng cao. Từ đó cho thấy thiết kế các HTN là đối mặt với các thách thức như: đáp ứng thời gian (cứng hay mềm) của hệ thống với các sự kiện, kích thước, trọng lượng, tiêu thụ năng lượng, độ tin cậy, giá thành.

Các phương pháp hiện dùng bao gồm việc đặc tả và thiết kế phần mềm và phần cứng tách rời. Một đặc tả thiết kế thường là chưa hoàn hảo, được viết bởi ngôn ngữ nào đó và chuyển cho cácc kỉ sư phần cứng và phần mềm. Việc phân hoạch phần cứng-phần mềm cần được lưu tâm trước tiên và được chấp nhận, tôn trọng, vì bất kì sự thay đổi nào ở quá trình này sẽ dẫn tới các thay đổi khác và làm cho quá trình thiết kế sẽ kéo dài. Các nhà thiết kế phải nổ lực rất nhiều để mọi thứ có thể chuyển vào phần mềm, chỉ chuyển một phần của thiết kế cho phần cứng nhằm đảm bảo thỏa mãn các tiêu chí về thời gian phản ứng của hệ thống. Vấn đề ở đây có thể là:

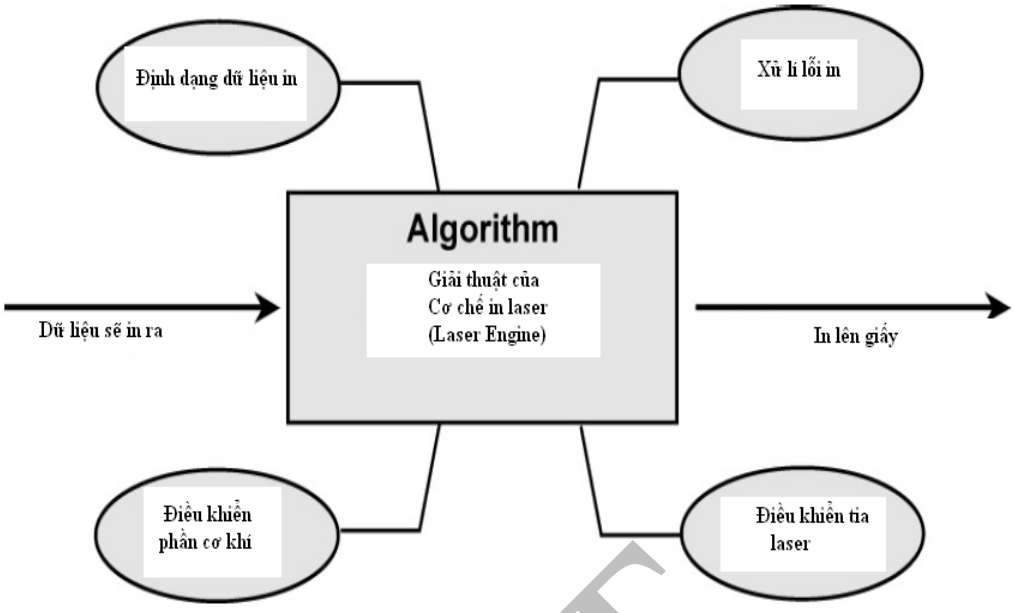
- Chưa đầy đủ các đặc tả phần cứng và phần mềm, nên khó kiểm chứng tổng thể hệ thống, do đó dẫn đến sự không tương thích khi tiếp cận ranh giới giữa phần cứng và phần mềm.
- Các phát thảo ban đầu sẽ kéo theo các phát thảo (bổ sung/tuỳ chọn) khác bên dưới.
- Sự chận trể chưa có một lưu trình thiết kế hoàn hảo sẽ gây khó khăn cho việc sử đổi và tác động tới thời điểm đưa sản phẩm ra thị trường.

Có nhiều cách tiếp cận để giải quyết vấn đề thiết kế HTN, tuy nhiên không có cách nào có thể thỏa mãn và đạt hiệu quả như mong muốn. Dưới đây sẽ trình bày một số tiếp cận khi thiết kế một HTN.

Thiết kế HTN là kết hợp giải quyết một hệ thống đầu cuối bao gồm cả phần cứng và phần mềm. Do đó cần có những quyết định phần nào của thiết kế sẽ được giải quyết ở phần cứng và phần nào ở phần mềm để có được một hệ thống với những đặc điểm chuyên biệt (thiên về hiệu năng), khác với các máy tính thông thường. Có những xử lý phần mềm lại phải cứng hóa (gọi là được silicon hóa) mà kết quả là hiệu năng tính toán nâng cao. Ví dụ trước đây, CPU 286/386 khi không có vi chip FPU 387, mọi phép tính dấu phẩy động phải chạy bằng tập lệnh mềm, do đó nhiều phần mềm cao cấp không hoạt động được (AutoCAD chẳng hạn). Cách lựa chọn thiết kế như vậy gọi là sự phân hoạch thiết kế.

Nếu tổng quát hóa khái niệm giải thuật thành các bước để thực thi một thiết kế, thì giải thuật như một kết hợp của các thành phần mần cứng và thành phần phần mềm, và mỗi phần của phân hoạch cứng/mềm đó sẽ thực thi một giải thuật. Tất nhiên có thể áp dụng giải thuật thuần mềm (CPU không có FPU), hay thuần cứng hoặc kết hợp cả 2 (ví dụ như vĩ đồ họa máy tính).

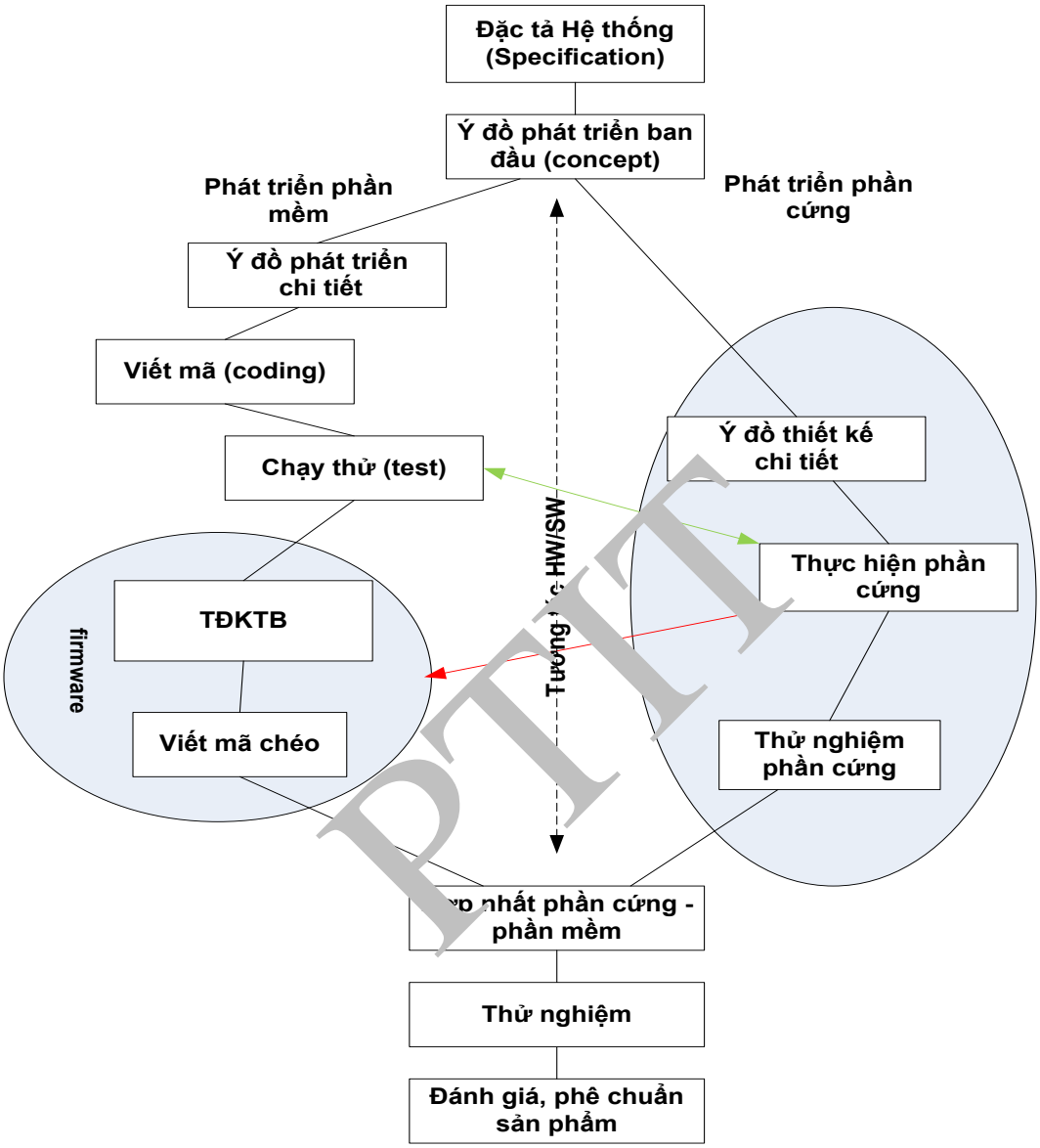
Ví dụ: Giải thuật thiết kế máy in laser:



Hình 4.6 Giải thuật thiết kế m[illegible] in laser: phân hoạch cứng/mềm

Dữ liệu nhận vào từ cổng LPT, CPU phải chuyển đổi thành xâu nối tiếp để làm đầu vào điều chế tia laser, quay các gương chiếu tia, quay trống in, phát sinh ảnh in lên giấy, đốt mực sấy khô bản in, … Kết quả mất công suất in 5-7 trang /phút. Tăng tốc in ? Tăng cường thêm CPU … Giải pháp như vậy là tối ưu ? Tuy nhiên khi phân tích giải pháp thiết kế một cách sâu hơn, có thể biểu diễn thiết kế như một giải thuật ở đó có những tác vụ có thể cứng hóa, ví dụ cứng hóa khối chức năng ghi laser lên các phần tử nhạy sáng trên bề mặt trống in, giải phóng CPU khỏi tác vụ này như trước đó đã làm. Tất nhiên phần khó sẽ là cần phần cứng rất tin cậy và bền vững, một vi mạch kiểu ***ASIC*** (**Application-Specific Integrated Circuit-** kiểu vi mạch được thiết kế dành cho một ứng dụng cụ thể) sẽ xuất hiện. Tuy nhiên ASIC sẽ làm tăng sự phức hợp hệ thống thiết kế. Do vậy nhóm phần mềm sẽ phải nỗ lực và hiệu chỉnh phần mềm rất tỉ mỉ, sao cho thêm phần cứng nhưng không quá phức tạp và đắt đỏ, làm giá thành tăng thêm.

Sơ đồ phân hoạch thiết kế cứng và mềm:

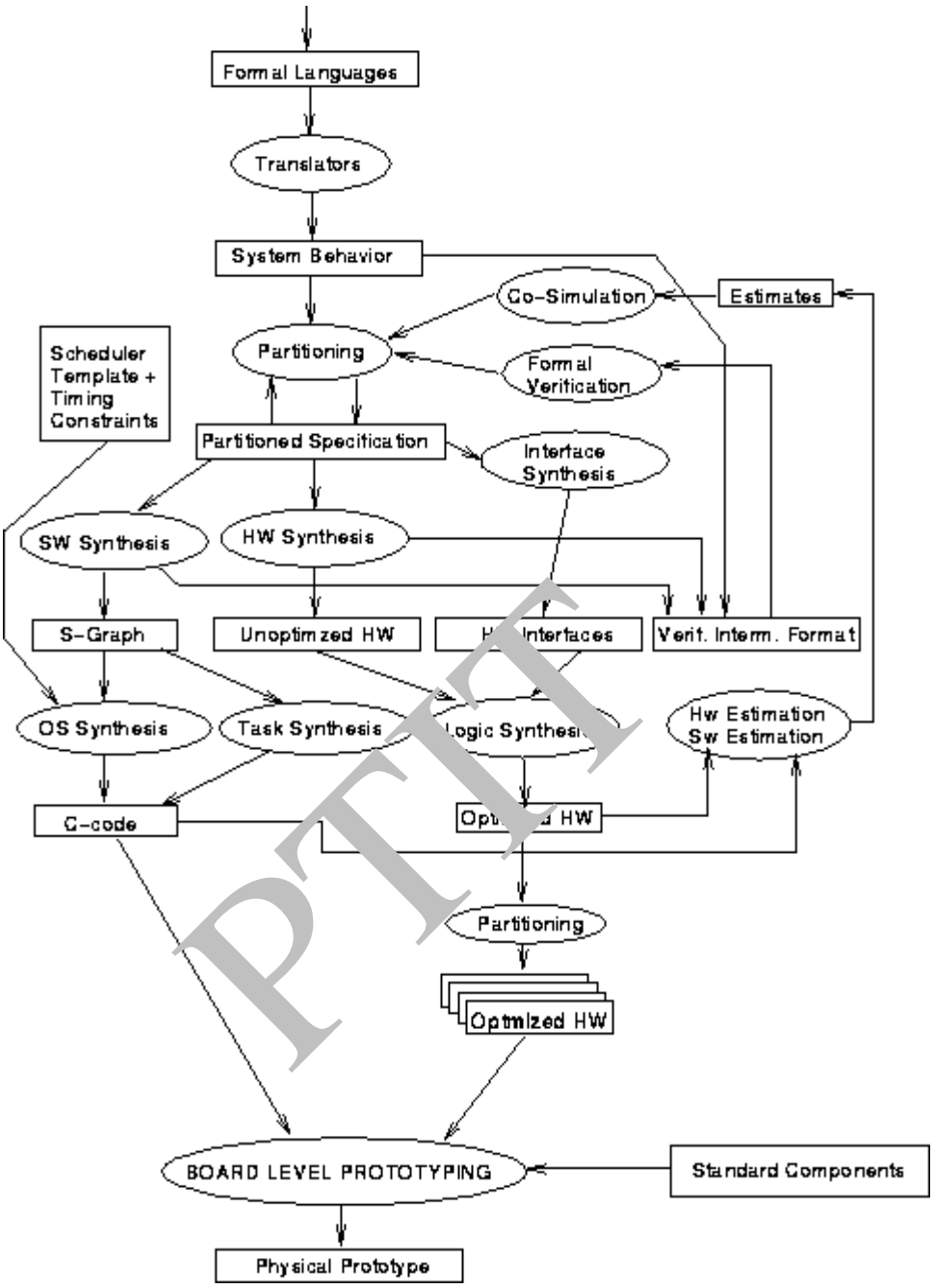


Hình 4.7 Phân hoạch thiết kế phần cứng và phần mềm

- Cần có sự tương tác ở pha phân hoạch cứng/mềm ở hai nhóm thiết kế: Tuy đã hình thành các khối chức năng cứng/mềm rõ ràng, nhưng khi ranh giới vẫn chưa thể khẳng định khi các thách thức chưa được đưa vào thiết kế. Các công cụ thiết kế phần cứng (ICE, Simulation, …) sẽ cho thấy có thể cải thiện phần cứng khi kết hợp với phần mềm, trong khi phần mềm chạy thử bằng mô phỏng trên phần cứng để đánh giá tốc độ xử lý. Nói

cách khác hai nhóm thiết kế cứng/mềm cần tương tác với nhau để bổ sung cho thiết kế cuối cùng. Bo mạch sản phẩm là bo mạch đích tuy nhiên vẫn chưa là bo mạch thực chế tạo. Quá trình tương tác sẽ lặp lại cho tới khi các chỉ tiêu sản phẩm chấp nhận được, trước khi tiến hành làm sản phẩm mẫu thực. Các công nghệ thiết kế ngày nay tạo điều kiện để hai nhóm thiết kế làm việc với nhau chặt chẽ và tiết kiệm nhiều thời gian, hạn chế tối đa các rủi ro cho tới khi chế tạo sản phẩm.

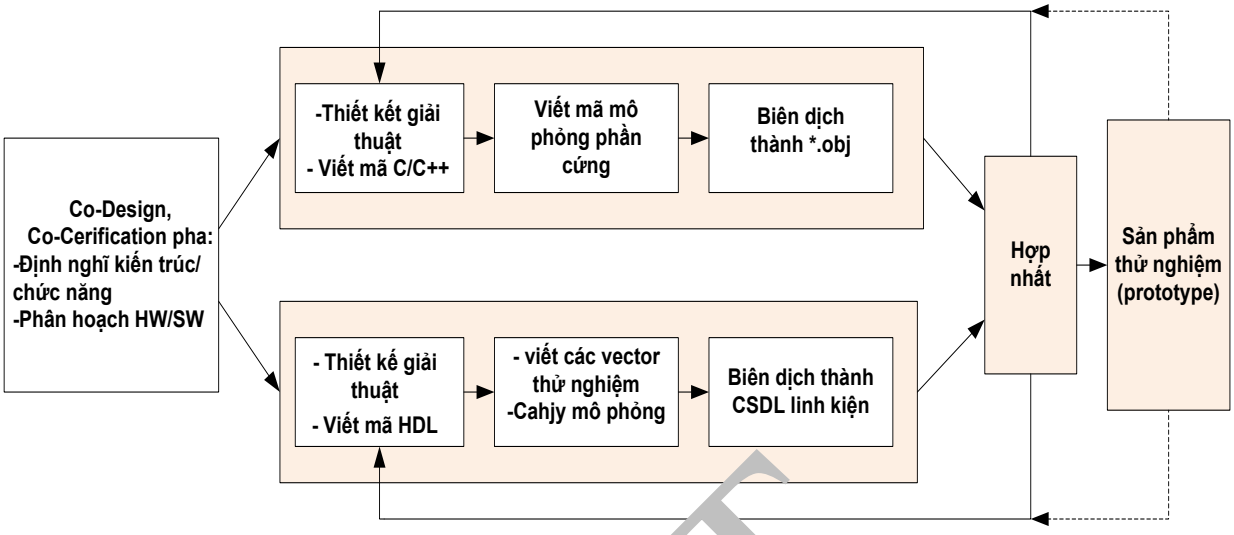
- Khi nào thì quyết định phân hoạch cứng/mềm ? Khi đã phân tích hầu hết (hay tất cả) các giải pháp trình bày giải thuật thiết kế. Từ đó sẽ thấy phần nào của giải thuật sẽ nằm ở phần cứng, phần nào ở phần mềm, là tối ưu. Một số qui tắc có thể dùng để cân nhắc: Tăng tốc độ xử lý, có thể đáp ứng bằng phần cứng, nhưng tăng giá thành, và đôi khi phải thiết kế lại chip xử lý ! Phần mềm có thể chủ động bằng mô phỏng, không chờ đợi có phần cứng hoành chỉnh mới thử nghiệm, viết code thật hợp lí cũng sẽ giảm chi phí phần cứng. Dù sao thì phần cứng cũng có những lối thoát, đó là sự lựa chọn các bộ xử lý để đưa vào thiết kế (hay dùng ASIC, rồi đi đến hệ thống trên chip (System on Chip-SoC). Phần còn lại là thiết kế bo mạch tổng thể, "code" hóa một số xử lý phần cứng thành phần mềm sụn (*firmware*).
- Kết hợp phần cứng/phần mềm. Hiện nay các sản phẩm thương mại được phát triển theo kĩ thuật "*đồng thiết kế phần cứng và phần mềm-đồng kiểm nghiệm*" (*Hardware/software co-design and co-verification)*. Nếu theo thiết kế truyền thống sau khi làm phân hoạch các phần thiết kế thực hiện tương đối độc lập, thời gian phản hồi-hiệu chỉnh ở cả phần cứng và phần mềm sẽ lâu, do vậy chi phí nhiều thời gian hơn cho sản phẩm cuối cùng. Ngày nay một qui trình thiết kế mới đã phát triển, trong đó thiết kế phần cứng và phần mềm tiến hành song song các phản hồi-hiệu chỉnh thực hiện liên tục, cho tới khi có kết quả tốt nhất qua đồng kiểm nghiệm. Trong công nghệ này phần cứng được trình diễn bởi ngôn ngữ mô tả phần cứng (***VHDL****-very-high speed integrated circuit hardware description language*) và đó là phần cứng ảo để phần mềm hoạt động. Như vật quá trình thiết kế có thể "mềm hóa" và sự kết hợp hài hòa là rất rõ ràng.



Hình 4.8 Đồng thiết kế phần cứng và phần mềm-đồng kiểm nghiệm, tối ưuthiết kế

Ví dụ kĩ thuật thiết kế ASIC: là cuộc cách mạng trong thiết kế các HTN. Sản phẩm ASIC có thể tìm thấy, như các Chipset trong PC, chip xử lý âm thanh, tăng tốc đồ họa, chip cho MODEM, … Quá trình thiết kế có tên là biên dịch bán dẫn (*silicon compilation)*, trong đó phần cứng và phần mềm được biểu diễn bởi các tệp dữ liệu của ngôn ngữ thiết kế cao cấp. Như vậy HTN sẽ được đặc tả như một cơ sở dữ liệu mềm (*software database*): một phần mô tả kiến trúc phần cứng, một phần mô tả cách điều khiển hoạt động của phần

cứng. Ở đây sự phân biệt giữa phần cứng và phần mềm có vẽ mập mờ: thiết kế phần cứng lại như là thiết kế phần mềm.



Hình 4.9 Qui trình thiết kế kiểu ASIC

4.1.3 Xây dựng bo mạch khi phát triển hệ thống

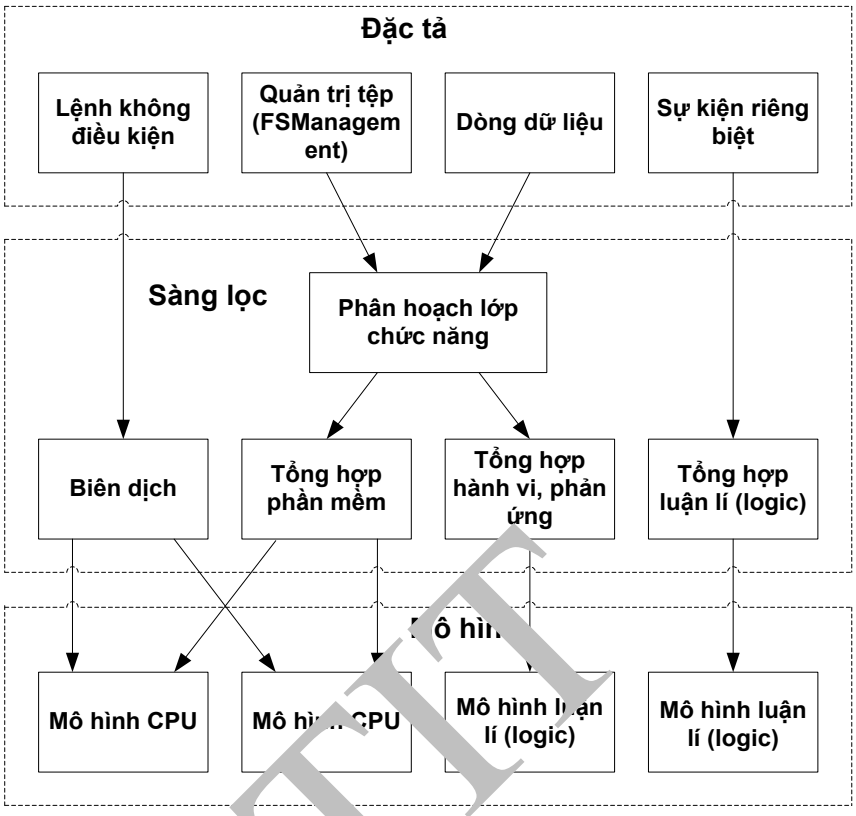
Triển khai các yêu cầu mô tả ở phần 4.1.1 khi thiết kế bo mạch sẽ cụ thể như sau:

1) Đặt vấn đề: Mục đích thiết kế HTN, mục tiêu cần đạt được, tính thương mại …;
2) Xác định HTN:

✓ Xây dựng một mô hình chính tắc (*formal model*) của hệ thống với các yêu cầu sau đây:

- Xác định chức năng: xây dựng tập các mối quan hệ tường minh hay không tường minh liên quan tới đầu vào/đầu ra, và thông tin trạng thái bên trong của hệ thống.
- Xây dựng tập các thuộc tính mà thiết kế sẽ phải thỏa mãn. Kết hợp các thuộc tính và tập các quan hệ vào/ra, trạng thái hệ thống, xác định lại các chức năng hệ thống. Các thuộc tính bao gồm:
  - Các thuộc tính có tính kế thừa các hệ thống tính toán (máy tính số);
  - Các thuộc tính có thể kiểm chứng trên một chức năng;
  - Các thuộc tính phải được kiểm chứng trên các đặc tả phải có khi các tiêu chí ở đầu vào xuất hiện.
- Xây dựng tập các chỉ số hiệu năng để đánh giá thiết kế theo các tiêu chí: giá thành, năng lượng. độ tin cậy, tốc độ xử lý, kích thước …
- Xây dựng tập các khác biệt coi đó như những thách thức của thiết kế, đề ra giải pháp giải quyết.

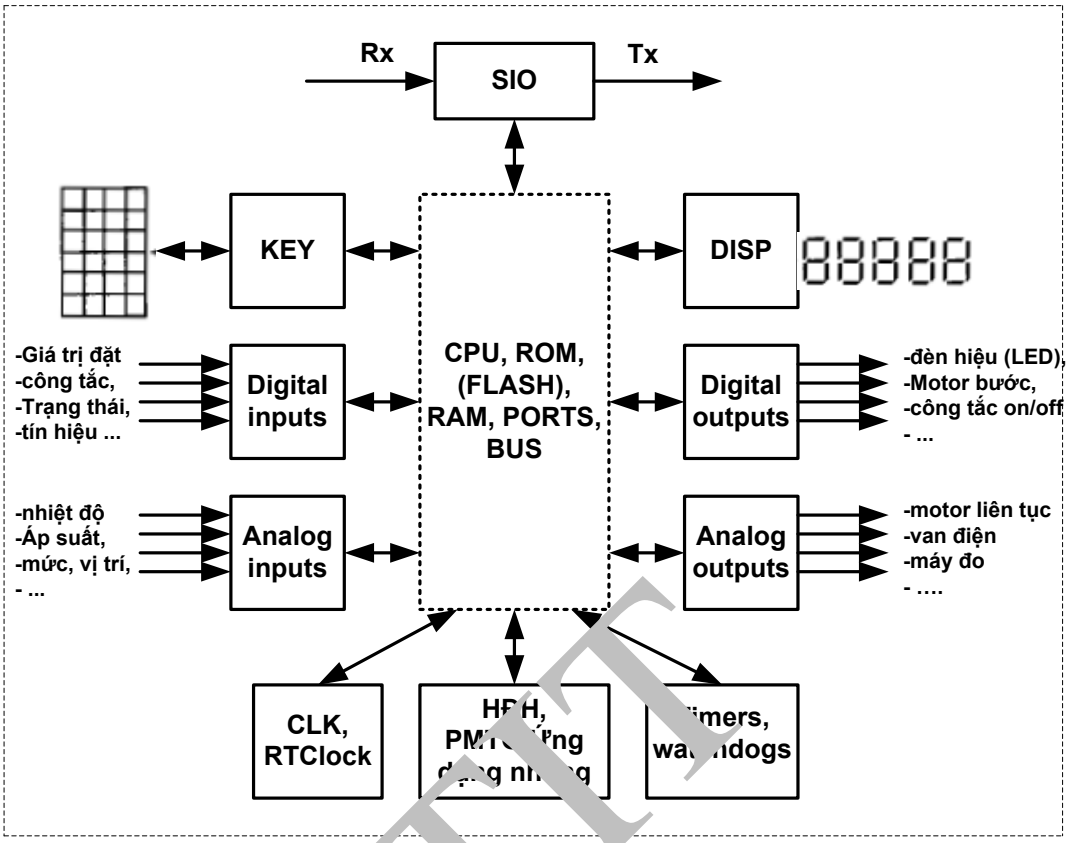
- Thực hiện tinh lọc thiết kế để có thiết kế từ ý tưởng đến mô hình:



Hình 4.10. Xây dựng mô hình hình thức: Bước sàng lọc sử dụng cách tổng hợp phần cứng và phần mềm để chuyển hóa xác định chức năng vào mô hình phần cứng của thiết kế.

- ✓ Mô tả hệ thống phản ứng như thế nào với các đầu vào,
- ✓ Cộng cụ mô hình hóa sự vận động của hệ thống, ví dụ phương pháp lưới Petri (***Petri nets)*** hay biểu đồ trạng thái (***StateCharts)***.
- ✓ Yêu cầu của sản phẩm sẽ có: đưa ra mô tả sản phẩm;

3) Thiết kế phần cứng: lên mô hình kiến trúc của bo mạch chính, lựa chọn các thành phần phần cứng;



Hình 4.1. Sơ mạch HTN

4) Mô hình tổ chức ghép nối các thiết bị ngoài, số lượng, chủng loại các thiết bị ngoài;
5) Thiết kế phần mềm: phác họa các phần mềm:
   - Mô hình các lớp phần mềm;
   - Có hay không có các phần mềm trung gian;
   - Có hay không có hệ điều hành, loại hệ điều hành sẽ phát triển cho phần cứng đích;
   - Phần mềm điều khiển I/O (TĐKTB);
   - Phương thúc liên kết các phần mềm theo nguyên lí lớp xếp chồng;
   - Các phần mềm ứng dụng.
6) Tương tác giữa phần mềm và phần cứng, tinh lọc phần cứng, phần mềm
7) Viết phần mềm trên hệ phát triển;
8) Viết ứng dụng trên hệ phát triển;
9) Thử nhiệm chương các phần mềm và hiệu chỉnh phần mềm trên hệ phát triển;
10) Hợp nhất phần cứng và phần mềm vào sản phẩm chế thử (prototype):

Nạp phần mềm vào hệ thống đích:

- Nạp các TĐKTB, thử nghiệm kết hợp TĐKTB với các cổng ghép nối trên bo mạch, hiệu chỉnh mã;
- Nạp phần mềm hệ thống (hệ điều hành, hay monitor, …) và thử nhiệm các chức năng của hệ điều hành có và không có TĐKTB, hiệu chỉnh mã;
- Nạp phần mềm trung gian nếu có, thử nhiệm kết hợp với hệ điều hành, hiệu chỉnh mã.
- Nạp thử một ứng dụng, thủ nghiệm, chỉnh sửa mã.

11) Hợp nhất hệ thống, tăng cường thử nhiệm các chức năng với các điều kiện phức tạp giả định, thời gian thực, đáp ứng … Đánh giá sức chịu đựng của môi trường vận hành trong thực tế …
12) Làm hồ sơ, tài liệu.

Tóm tắt qui trình thiết kế HTN

- ***Các qui tắc thiết kế***

1) Nền tảng kiến thức để thiết kế HTN : Khoa học về máy tính và kĩ thuật điện tử;
2) Thiết kế vi mạch với ngôn ngữ phần cứng Verilog hay VHDL (FPGA, ASIC);
3) Khả năng công nghệ và những giới hạn công nghệ phần cứng.
4) Lĩnh vực ứng dụng HTN rất rộng, thiết kế phải hướng vào đối tượng ứng dụng cụ thể;
   Với các xu hướng:
   - Gia tăng kích thước mã chương trình: 16 – 64 KB lên đến 64kB đến 512 KB,
   - Tái sử dụng các thành phần cứng (CPUs, micro-controller, DSPs) và mềm (device drivers),
   - Có sự hợp nhất cao trong 1 hệ thống (DSP, mạng, RF, CPU 32 bit, IO processors kiểu Intelligent Input/Output-I2O).
5) Sử dụng phần mềm có sẵn, phần mềm tái sử dụng, mã nguồn mở.
6) Công nghệ lập trình (ngôn ngữ lập trình, hệ phát triển phần mềm);
7) Thiết kế vi mạch (dạng VLSI, ASIC), thiết kế hệ thống điện tử (số, analog);
8) Hệ thống xử lý kiểu thời gian thực (thời gian thực cứng, thời gian thực mềm).

- ***Các bước thiết kế***

1) Xây dựng đặc tả HTN, mô hình hóa HTN sẽ được thiết kế, thực nghiệm với các giải thuật liên quan;
2) Tập hợp và mô tả phần cứng cơ bản : ghép nối cơ sở, truyền thông, công nghệ tính toán ứng dụng vi điện tử, công nghệ bộ nhớ, các thiết bị ghép nối vào hệ thống.

3) Hệ thống phần mềm sẽ có: điều khiển thiết bị, phần mềm trung gian, hệ điều hành, phần mềm ứng dụng …
4) Phân hoạch, chọn lọc các phần của thiết kế: phần cứng, phần mềm. Ở mỗi phần, phân rã thiết kế thành các phần nhỏ hơn, xây dựng các mối tương tác giữa các phần đó;
5) Sử dụng các công cụ mô phỏng thiết kế chạy mô phỏng phần cứng, phần mềm và kết hợp cả hai trên mô hình ảo;
6) Các phần mềm và phần cứng tới hạn (ràng buộc thời gian) cần thử nghiệm và điều chỉnh theo kiểu "sụn hóa": phần mềm kết hợp giữa phần cứng và phần mềm, trong đó xử lý khi vài module chia sẽ khai thác chung một phần cứng. Phát triển giải thuật lập lịch tối ưu.
7) Thử nghiệm trên bo mạch phần cứng (prototype) với CPU đã chọn.
8) Gỡ rối và tinh chỉnh phần cứng, phần mềm;
9) Hoàn thiện sản phẩm.

4.2 CÀI ĐẶT VÀ THỬ NGHIỆM HTN

Phần tiếp theo sẽ đề cập đến việc thực hiện xây dựng một HTN. Giả định rằng ta có một dự án phát triển một ứng dụng nhúng nào đó, và hãy bắt đầu với công việc thiết kế HTN đó.

4.2.1 Chọn CPU cho thiết kế

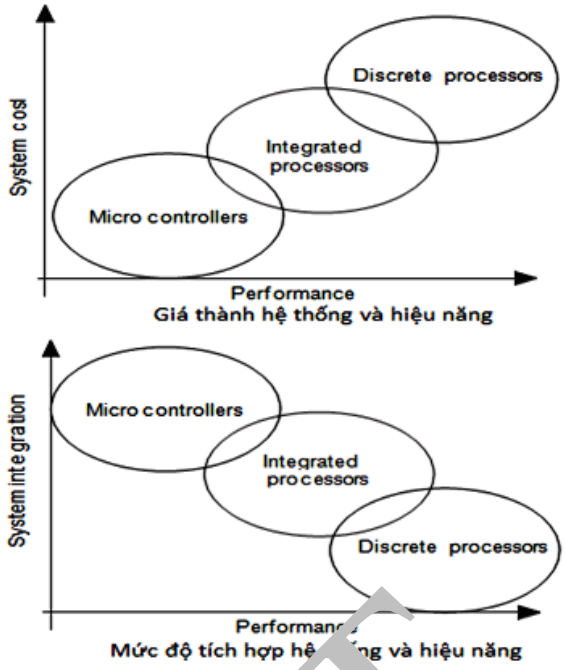
Sau khi phân tích yêu cầu của HTN đặt ra, trên cơ sở các dịch vụ mà HTN sẽ thực hiện, sẽ dẫn đến việc xác định tính năng của bộ xử lý trung tâm (CPU). Như đã giới thiệu có nhiều loại HTN và mỗi loại được xây dựng trên một kiến trúc với CPU khác nhau, và trên thị trường hiện tại sẽ cho nhiều lựa chọn phù hợp.

Như đã phân loại các dòng CPU dùng cho HTN, một số câu hỏi cần đặt ra khi thiết kế như sau:

1) Chế tạo CPU riêng biệt kiểu tùy biến (customize) với các công nghệ như ASIC ? Vấn đề sẽ là chi phí cho lựa chọn này ? Rõ ràng hướng phát triển này mang lại nhiều ưu điểm như: học thuật chế tạo chip, chủ động sáng tạo v.v… Tuy nhiên cần có đủ kiến thức về chế tạo mạch, đặc biệt là mạch có mật độ tích hợp lớn và rất lớn, kiến thức và kinh nghiệm thiết kế logic theo khối chức năng, và ý tưởng kiến trúc và chức năng của một CPU phải rất rõ ràng. Hơn nữa trang thiết bị sẽ rất đắt. Nói chung lựa chọn này đòi hỏi một đội ngũ rất mạnh, chuyên nghiệp và khối lượng kiến thức chuyên ngành rất sâu.
2) Tìm hiểu thị trường CPU cho ứng dụng nhúng: bo mạch controller với microprocessor truyền thống, bo MCU (micro-controller) với công nghệ mới PIC (*Programmable Interface Controller* của Microchip Technology), PSoC (*Programmable System-on-Chip* của Cypress Semiconductor.), kết hợp với PFGA (Field Programmable Gate Aray của Xilink) hay các sản phẩm khác.
3) Một số ý tưởng lựa chọn CPU cho HTN:

Lựa chọn CPU có thể trên cơ sở xác định đặc thù sản phẩm hay nhu cầu mà thị trường chờ đợi. Tuy nhiên tốt hơn cả là người thiết kế HTN phải cân nhắc để có CPU thỏa mãn ứng dụng nhúng mình đang xây dựng. Dưới đây là một số tiêu chí hỗ trợ khi thực hiện chọn CPU cho thiết kế:

- HTN sẽ kết nối với các loại ngoại vi (sensors) nào ? Đặc tính kĩ thuật của các ngoại vi ? Danh sách này sẽ cho biết số cổng vào/ra (I/O) cần có mà CPU hỗ trợ.
- Có bao nhiêu chương trình, cần bao nhiêu không gian dữ liệu (bao gồm cả các ngăn xếp, không gian bộ nhớ cấp phát động cho chương trình (heap)) cần cho hệ thống ?
- Có bao nhiêu ngắt hệ thống cần ? Và chắc chắn là như vậy ?
- Số lượng các cổng vào/ra mà thiết kế sẽ sử dụng ?
- Loại xử lý nào ràng buộc nhất về thời gian tới hạn (*critcal-time*) mà CPU phải thực hiện ?
- Loại công cụ phát triển nào (cho cả phần cứng và phần mềm) có sẵn cho CPU chọn ? (mã nguồn mở, phần mềm trung gian ngôn ngữ bậc cao không phụ thuộc CPU như Java … ngôn ngữ lập trình, công cụ biên dịch hướng đối tích …) .
- Trong trường hợp CPU trên bo mạch (CPU on board), cùng các thành phần hợp nhất, nguồn nuôi, công cụ thử, thì giá thành thực sự là bao nhiêu ?
- Có các loại thiết bị sẵn có nào ghép được vào hệ thống ở hiện tại và hỗ trợ thiết bị mới ?
- Phần mềm sẽ làm việc với phần cứng như thế nào và ngược lại ? (Phát tiển ứng dụng sẽ tương thích …?).
- Loại công cụ phát triển nào sẵn có, ví dụ hợp ngữ, biên dịch C, C+, mô phỏng .. ?
- Không gian địa chỉ mà CPU hỗ trợ là bao nhiêu ? KB, MB, vài MB đến GB ? (Lưu ý là HTN sử dụng hai loại bộ nhớ chính: bộ nhớ chương trình ứng dụng, hệ điều hành là flash, ROM, UVPROM, EEPROM. Loại thứ hai cho dữ liệu là RAM (Ram tĩnh, hay RAM động).
- CPU Clock, giá/hiệu năng của CPU ? Có một câu hỏi đơn giản, nhưng không dễ trả lời là khái niệm hiệu năng. Ở đây sẽ thiên về lựa chọn liên quan tới các phép tính số nguyên hay số dấu phẩy động, cũng như các lệnh chuyển dữ liệu giữa các thành phần trong hệ thống. Một đánh giá khác là đáp ứng thời gian và hiệu quả xử lí ngắt (nhận và xử lí dữ liệu). Việc đánh giá hiệu năng có thể đạt được cho tới khi có được một phiên bản thiết kế, chạy các phần mềm được phát triển bằng các ngôn ngữ bậc cao. Thường khi dùng ngôn ngữ bậc cao, ta hay bị mất hiệu năng đặt ra khi thiết kế, so với khi chạy bằng hợp ngữ. Một số tài liệu có nêu rằng sẽ mất đi chừng 20%-30% hiệu năng khi so sánh giữa hai ngôn ngữ nói trên khi tính qui ra MIPS (triệu lệnh CPU/s).



Hình 4.12 Mối tương quan giữa giá thành hệ thống/hiệu năng và mức độ tích hợp hệ thống/hiệu năng

Đó là một số tiêu chí khi chọn CPU cho thiết kế một hệ thống nhúng. Nhưng CPU không phải là tất cả, còn có các thiết bị, ghép nối cũng cần xem xét về tính tương thích, có sẵn trên thị trường để có giải pháp tích hợp hệ thống sao cho hợp lí.

4.2.2 Bộ nhớ cho HTN

Chọn bộ nhớ cho HTN là phần đặc biệt quan trọng vì phải đối mặt với nhiều chủng loại, kiểu, cho một lớp thiết bị nhúng. Chọn bộ nhớ bao gồm cả hai vấn đề: loại bộ nhớ và kỉ thuật điều khiển hoạt động, cơ chế bảo vệ nội dung (tách biệt giữa phân đoạn bộ nhớ hệ thống cho Hệ điều hành và phân đoạn cho ứng dụng). Thêm vào là bộ nhớ ẩn (cache) để chứa bản sao của mã lệnh (thường ở dạng nén trong ROM) đã được giản nén để chạy và dữ liệu, sao cho có thể truy nhập nhanh nhất có thể, như ậy sẽ đảm bảo cho CPU chạy được tốc độ tối đa. Ở điểm này việc thiết kế bộ nhớ sẽ trở nên rất phức tạp Dưới đây là một số loại bộ nhớ tiêu biểu, lựa chọn và ảnh hưởng tới thiết kế HTN.

a) *Công nghệ bộ nhớ:* Thông thường trong HTN ta sử dụng bộ nhớ *EPROM* để chứa phần mềm hệ thống (HĐH, dịch vụ hệ thống) và DRAM cho dữ liệu và chương trình ứng dụng. Cho các HTN tốc độ cao thì *SRAM* là lựa chọn thích hợp. Về cấu trúc bộ nhớ đã đề cập ở chương 2. *DRAM* cho phép tạo chip nhớ dung lượng rất lớn, mật độ cao và giá thành rẻ, nhưng hoạt động chậm, cần cơ chế làm tươi dữ liệu. Ngoài ra còn có một vài

loại RAM hiện đại khác, như *Pseudo-static RAM* sử dụng nhân DRAM có mật độ cao, nhưng có hệ thống làm tươi ngay trong chip, mà không cần cả một module làm tươi truyền thống vì vậy hoạt động giống như *DRAM* (nhưng vẫn không nhanh như DRAM). Việc sử dụng là hiệu quả về giá thành hệ thống, cũng như đáp ứng thỏa mãn cho ứng dụng nhúng. Loại tiếp theo là *Battery backed-up SRAM: c*ho tới khi có thiết kế công nghệ năng lượng thấp, cho phép tạo ra bộ nhớ không mất nội dung (*non-volatile memories*), tức là dữ liệu không bị mất đi ngay cả khi nguồn nuôi được thay thế bởi 1 pin nhỏ cấp nguồn cho bộ nhớ trong thời gian dài. Đặc điểm công nghệ nền tảng là dòng điện dò rỉ vô cùng nhỏ trong mỗi bóng bán dẫn- *transistor*. Một vài hãng còn chế tạo các SRAM với nguồn pin để lưu cấu hình hện thống, ví dụ *RAM CMOS* trong máy tính PC. Và đó là tên gọi *Battery backed-up SRAM.* Bộ nhớ *Flash* là loại xóa bằng điện và có mật độ transistor và thời gian truy nhập (WR=100ns, RD=70-100ns) gần bằng DRAM và cũng là loại bộ nhớ không bị mất nội dung được dùng thay cho EPROM. Trong HTN, Flash dùng để lưu phần mềm có nhu cầu nâng cấp từ xa (tải về sau đó nâng cấp. Ví dụ các Router của CISCO hay các ADSL hiện đại). EPROM cũng là loại bộ nhớ không bị mất nội dung và xóa bằng điện cũng được dùng cho HTN, lưu phần mềm hệ thống và ứng dụng nhúng. Về bộ nhớ còn nhiều vấn đề có kiên quan khác như cách thức tổ chức Chip nhớ, module nhớ, kỉ thuật quản lí bộ nhớ của hệ thống dưới sự tổ chức và khai thác của hệ điều hành. Những vấn đề này nên tham khảo ở tài liệu *Kiến trúc máy tính*. Bộ nhớ ẩn (*Cache*) là loại bộ nhớ SRAM rất nhanh, làm việc phối hợp với phần cứng đặc biệt và sự dụng để chứa lệnh/dữ liệu cung cấp cho CPU lấy từ bộ nhớ hệ thống, đảm bảo CPU luôn có thể chạy mà không phải chờ đợi. Hơn nữa phần lớn code có cấu trúc vòng lặp, ở đó một đoạn code thực hiện lặp đi lặp lại. Cache chứa các code này sau khi lấy từ bộ nhớ hệ thống, sẵn sàng cho các thực hiện liên tiếp sau đó, mà không tìm ở bộ nhớ hệ thống. Truy nhập cache rất nhanh, nên hệ thống chạy nhanh hơn rất nhiều. Cache được chế tạo ngay trong CPU với các mức độ khác nhau (L1, L2, L3) với dung lượng khác nhau. Quản lí cache là kỉ thuật phức tạp. Tuy vậy các hệ thống hiện đại không thể thiếu cache khi CPU có tốc độ cao. Một thông số khi lựa chọn bộ nhớ đó là thời gian truy nhập (*Access time*). Thông số này rất quan trọng và liên quan chặt chẻ với CPU, tức là khi tần số CPU càng cao, thì thời gian truy nhập càng ngắn, tức là bộ nhớ càng phải nhanh.

b) Khi xây dựng bộ nhớ cần lưu ý tới CPU lựa chọn để thích ứng cho loại bộ nhớ sử dụng, trong đó cần chú ý: BUS địa chỉ, truy nhập theo thứ tự byte chẳng, byte lẻ, loại hệ kiểu như *Big and Little endian,* truy nhập theo kiểu chế độ theo trang (***Page mode***), trang xden kẻ (***Page interleaving***), hay CPU đọc trước dữ liệu nhiều hơn là cần thiết và lưu vào cache (***burst mode)*** …

4.2.3 Ghép nối với thiết bị

Ghép nối với thế giới thực, nơi HTN hoạt động, bao gồm: các thông tin tương tự (analog), các thông tin số hóa giải số hóa bằng các vi mạch chức năng (ADC/DAC). Sau đó là truyền thông dữ liệu nối tiếp tốc độ chậm, tốc độ cao, đồng bộ hay dị bộ có hay không có giao thức điều khiển (UART, chip truyền thông có cấy giao thức), khoản cách xa hay gần cần có lựa chọn thích hợp. Khi ghép vào hệ thống để trao đổi dữ liệu (IN/OUT data) sử dụng các kỉ thuật ghép nối như ghép song song, ghép nối tiếp, qua cổng hay DMA, cần cân nhắc cụ thể cho mỗi loại thiết bị. cách thức quản lí thiết bị bao gồm các lựa chọn như : CPU chủ động (điều khiển có điều kiện-đối thoại, không điều kiện). Thiết bị chủ động (ngắt, DMA). Sự lựa chọ phương pháp sẽ cho các hiệu quả khác nhau, ảnh hưởng lớn tới hiệu năng hệ thống, đặc biệt với xử lí thời gian thực.

4.2.4 Phát triển phần mềm cho HTN

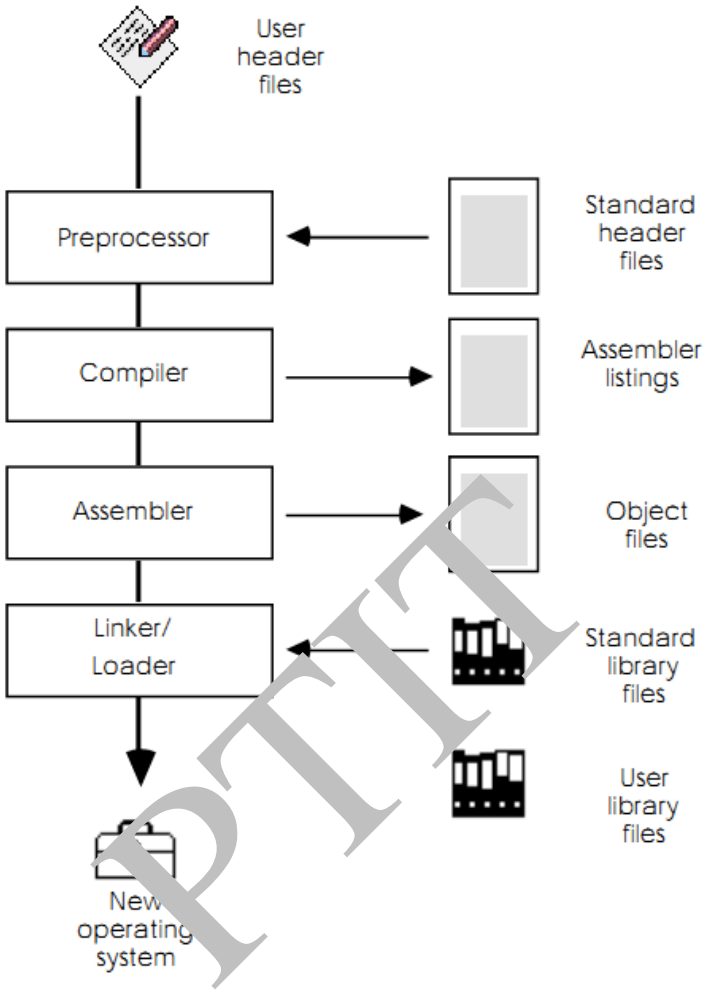
Có nhiều cách để viết mã (*coding)* cho HTN, mà sự lựa chọn phụ thuộc và tính phức tạp của phần mềm nhúng, và chi phí. Ví dụ như coding cho phần cứng đã được xây dựng (mua bo mạch có sẵn) thì dễ dàng hơn là cho một thiết kế độc lập tự làm. Ngày nay cũng có thể là theo cách sử dụng nhiều phần mềm mở (bao gồm *mã nguồn* bằng ngôn ngữ C, *thư viện* đi cùng và *công cụ phát triển*), tải xuống, gỡ rối (*debugging code*), sửa đổi cho phù hợp, biên dịch … và nạp vào HTN. Còn phát triển kiểu truyền thống lại tiếp cận theo hướng CPU đã chọn và phần mềm mô phỏng (*emulator*). Bộ mô phỏng có thể dùng để gỡ rối phần cứng trên bo của sản phẩm thử nghiệm (*prototype hardware*) trước khi giao cho giúp kỉ sư phần mềm triển khai phần mềm nhúng đích. Bộ mô phỏng được cài trên một hệ khác, gọi là hệ phát triển (*development system*), ví dụ trên PC. Một số bo mạch thương mại thường có kèm theo mã gỡ rối, là tiện ích giúp kỉ sư phần mềm chạy thử phần mềm viết ra. Sau đây đề cập tới một số vấn đề có thể phải đối mặt khi muốn sử dụng các phần mềm có sẵn cho HTN đích được thiết kế.

- ✓ Compiling code

Khi sử dụng các ngôn ngữ bậc cao để lập trình, ta không quan tâm tới các bước quá trình dịch để tạo ra mã thực thi. Việc biên dịch cần chương trình dịch thích hợp, cần tới cả thư viện chạy liên kết (*run-time libraries*) và cả trình liên kết để hợp nhất các module thành tệp thực thi cuối cùng. Vấn đề ở đây là sự hợp nhất với hệ thống dùng để phát triển (hệ phát triển-*development system*) phần mềm cho HTN, mà hệ này có thể mô phỏng được HTN đích tương lai. Phương pháp phát triển như vậy, gọi là biên dịch chéo (*cross-compilation+emulation*), tại đó phần mềm cho một hệ đích khác được phát triển toàn bộ trên một hệ hoàn toàn khác. Ví dụ dùng PC trên đó có cài phần mềm phát triển hướng cho loại HTN đích, mà thông thường các hãng bán CPU có thể cung cấp.

Hầu hết các trình dịch, cho C, hay PASCAL, …, đều chỉ phát sinh ra một tập các tiện ích, các lệnh từ các thành phần xây dựng bên trong (built-in routines) liên kết với thư viện của ngôn

ngữ để cuối cùng tạo ra các hàm chức năng. Quá trình dẫn tới kết quả trải qua vài bước nhất định, được mô tả như sau:



Hình 4.14 Quá trình biên dịch thành mã máy tạo ra HĐH

Pre-processor: tiền xử lí, thường gọi là biên dịch, xử lí mã nguồn, với chức năng kiểm tra cú pháp, khai báo biến, hằng, bắt lỗi cú pháp, phối hợp các tệp *include, define* vào code nguồn.

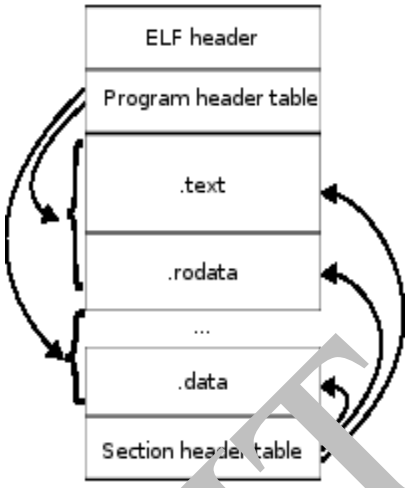
Compiler*:* dịch từ ngôn ngữ lập trình ra ngôn ngữ hợp ngữ - *assembler source listing* (***. list** tệp kết quả từ mã nguồn ***.asm**). Quá trình này đồng thời thực hiện tối ưu code để sau này chương trình chạy nhanh hơn, sử dụng bộ nhớ hợp lí hơn.

Assembler: dịch từ hợp ngữ ra mã đích (của CPU đích) nhị phân (tệp kết quả: ***.obj hay *.o**)

Linker/Loader*:* các tệp *.obj chưa thể chạy được, mà cần có linker/loader để kết hợp các mã hàm gọi, từ thư viện, liên kết nhiều module, liên kết các hàm có sẵn trong thư viện, tạo ra tệp thực thi phù hợp cho từng loại Hệ điều hành đích, gọi là tệp có định dạng kiểu **ELF** (*Executable*

and Linkable Format) ví dụ HĐH Microsoft: **.com* hay **.exe*, HĐH Linux có kiểu *FatELF* khả thi với thuộc tính *r-x r-x r-x.*

Mô hình của tệp *elf* như sau:



Hình 4.15 Định dạng một tệp thực thi ELF

Trong đó khối ELF header cho biết không gian địa chỉ sử dụng của CPU là 32 hay 64 bit. Khi kết hợp với Program header table tạo thành khối **PSP** (*Program Segment Prefix*) trong môi trường HĐH Microsoft, cho đặc tả về trạng thái hoạt động của một chương trình khi chạy.

Các khối khác **.text** (hay **.code**) chứa code thực thi, **.data** chứa dữ liệu (gồm các biến, các hằng, bảng …), **.rodata** (hay **.stack**) là vùng ngăn xếp.

- ✓ Thư viện các hàm chức năng chạy khi kích hoạt ***(Run-time libraries)***

  Vấn đề đầu tiên cho việc thiết kế HTN là có *thư viện các chức năng* mà chương trình dịch sử dụng để kích hoạt một số các thực thi khi hệ chạy ứng dụng. Việc thực hiện lời gọi thư viện (*system call*) bằng API (*Application Programming Interface*), tác động tới sự liên lạc giữa ứng dụng và phần mềm hệ thống đang chạy (***runtime system***), trong đó liên quan nhiều tới cơ chế vào ra (*input/output*) và quản lí bộ nhớ của máy tính. Nói đơn giản *run-time library* chính là một phần của cách xử lí hay sự phản ứng của máy tính, tức là một phần code của phần mềm hệ thống. Như vậy thư viện cho ta hầu như tất cả các chức năng mà qua ngôn ngữ lập trình ta có để sử dụng. Tất nhiên thư viện bao gồm nhiều kiểu chức năng, phụ thuộc vào phần cứng của HTN. Điều này thì không có sẵn khi thiết kế HTN (cả phần cứng và phần mềm), cho nên phải biến đổi để làm cho các chức năng đó chạy được trên HTN đích. Các trình điều khiển thiết bị là một ví dụ.

- ✓ **Sự phụ thuộc vào bộ xử lí (*Processor dependent*)**

Sử dụng các thư viện từ ngôn ngữ lập trình bậc cao thông thường sẽ có đòi hỏi chạy các hàm toán học, các xử lí xâu kí tự … mà các xử lí đó hoàn toàn sử dụng CPU, không có liên lạc với thiết bị, vì vậy thư viện các chức năng này không cần có sửa đổi. Tuy nhiên với các phép tính dấu phẩy động thì khác, bởi cần vi mạch chuyên về các phép tính này. Các vi mạch như vậy thường là tùy chọn với loại CPU, ví như CPU Intel 80386 trước đây có 80387 là bộ số học dấu phẩy động. Nói vậy để thấy khi chọn CPU cho HTN cũng phải lưu ý cần hay không cần bộ số học loại này. Điều này sẽ phụ thuộc vào ứng dụng trên HTN.

- ✓ **Phụ thuộc vào các thao tác vào/ra dữ liệu *(I/O dependent)***

Nếu chương trình không có các thao tác vào/ra dữ liệu, thì phần mềm có thể dùng trên các hệ đích mà hầu như không có các trục trặc. Tuy nhiên nếu có vào/ra, thì phải có định nghĩa phần cứng để phần mềm truy nhập. Nói vaajty tức phải viết lại trình điều khiển thiết bị cho hệ đích.

- ✓ **Gọi hệ thống** (System calls)

Các thủ tục gọi chức năng cũng cần sửa đổi để phù hợp với hệ đích, sao cho phù hợp với cơ chế xử lí các semaphore, cách phân phát bộ nhớ, cách xử lí tài nguyên, các lệnh điều khiển.

- ✓ **Thoát ra từ chu trình, thủ tục (Exit routines)**:viết đoạn code thoát

Các phần mềm tải về sử dụng, sau khi dịch thành tệp thực thi không dễ dàng nạp và chạy trong bộ nhớ. Đó là vì còn có các module khác, hay các tệp kèm theo (*modulr header*) mà hệ điều hành trên HTN sẽ sử dụng để nạp tệp thực thi vào đúng địa chỉ, khởi động chính xác các thanh ghi của CPU, ngăn xếp chạy tĩnh …Tất cả các việc này cần sửa đổi và chạy mô phỏng trước khi chạy trên HTN đích. Các thủ tục (*start-up routine, end routine*) thì thường viết mã (nằm) trong các module. Kết thúc thực thi các chu trình thường dùng lệnh *exit()*. Code của lệnh này sẽ thực hiện thu hồi bộ nhớ trả lại cho HĐH, cho nên cũng cần phải viết lại, có điều mã này lại thường nằm trong thư viện động.

- ✓ **Tạo thư viện**

Làm thế nào tạo ra các tệp thư viện phù hợp với CPU đích trên HTN thiết kế thay thế các tệp thư viện từ mã nguồn tải về ? Có hai giải pháp: thứ nhất là thay thiết kế trong phần cứng sao cho phần cứng phù hợp với code trong thư viện. Có thể đơn giản là thay đổi ánh xạ địa chỉ cho các hàm thư viện hay cổng (port) sao cho các địa chỉ hướng vào là trùng với các địa chỉ trong code của thư viện. nếu có dùng cơ chế I/O, thì địa chỉ mà I/O sử dụng phải giống như đã định nghĩa trong thư viện sao cho phần mềm truy cập đúng. Cách thứ hai là phải sửa code thư viện sao cho hợp với phần cứng của thiết kế. Điều này có nghĩa sẽ thay đổi cách phân bố sử dụng bộ nhớ, cổng I/O và cụ thể là sẽ sử cả TĐKTB (device driver) nếu có thiết bị mới khác biệt mà HTN sử dụng, đôi khi cả cho CPU khác biệt. cách này phụ thuộc nhiều vào thư viện đi kèm, có thể là trong dạng hợp ngữ, hay C và dễ dàng để biến đổi và liên kết thành thư viện mới. Thường các

code này bao gồm: đầu vào, cho phép truyền dữ liệu vào thủ tục xử lí và đầu ra chuyển sang thủ tục khác. Trường hợp phức tạp là khi thư viện ở dạng đã dịch thành nhị phân (*.obj) .

Tạo thư viện (Creating a library): Cần có kinh nhiệm lập trình khi viết các mã dạng như TĐKTB (thường gọi là lập trình hệ thống). Viết code cho TĐKTB tạo thư viện chạy thường không có hạn chế nào cho HĐH, cũng như để thay thế các chức năng I/O. Kỉ thuật này sử dụng để viết code cho một thiết bị ảo (theo cách phân loại ở chương TĐKTB) hỗ trợ các ngôn ngữ cấp cao truy nhập vào phần cứng cấp thấp mà không cần chèn đoạn code hợp ngữ đặc biệt. Khi chuyển sang hệ đích khác, thiết bị ảo thay đổi, chỉ cần biện dịch lại và liên kết (link) vào mã cho hệ đích mới.

Nhiều nhà cung cấp bo mạch thường cung cấp thông tin về cấu trúc phần cứng, và chương trình gỡ rối (debuger) với các lệnh I/O đi kèm, giúp tham khảo phần cứng bằng cách chạy debug. Việc này rất bổ ích để viết lại thư viện cho hệ đích.

Khi đã viết xong thư viện, công việc tiếp theo là đưa vào chương trình ứng dụng. Việc này thông qua tiện ích liên kết (linker): Chương trình và các thủ tục dạng hợp ngữ được dịch, hợp vào thành các module OBJ và sau đó được liên kết lại bởi linker để tạo ra mã thực thi cuối cùng. Thư viện mới được tích hợp bởi các kỉ thuật: liên kết thêm thư viện (gồm thư viện chuẩn và thư viện mới), hay liên kết bằng thay thư viện mới, bỏ thư viện trùng nhưng không còn sử dụng ở hệ đích.

- ✓ **Vấn đề với hệ điều hành**

Trong nhiều trường hợp, HĐH là không có sẵn cho bo mạch HTN đã thiết kế, mềm cần phải "tạo" HĐH theo yêu cầu. Có một số giải pháp sau đây:

- *Nếu mua bo mạch HTN*, có thể mua luôn phần mềm hệ thống đã được cấu hình cho bo đó. Trên thị trường có nhiều bo mạch HTN phổ biến và đi kèm là danh sách các HĐH, phần mềm đã được chuyển tải vào bo. Trong trường hợp này chi phí cho dự án HTN sẽ giảm đi vì rằng các phần cứng và phần mềm đã được thử nghiệm.
- *Tạo ra nhân HĐH mới cho cấu hình (HW) mới*: Khi không thể dùng các HĐH chuẩn trên thị trường và cần nhiều thay đổi cho hợp với cấu hình thiết kế, thì cần tạo mới HĐH (*rebuilding the operating system or kernel*). Thông thường từ mã nguồn, chọn các module (đã có thay đổi code), và biệ dịch, liên kết lại để tạo ra nhân với phiên bản HĐH mới. Một trong các HĐH dễ thực hiện sẵn có là HĐH Linux, ví dụ với RED HAT 9.0 cài trên máy đầy đủ với mã HĐH, sau đó dùng tiện ích make xconfig đi kèm chọn các module phù hợp, biện dịch lại nhân, tạo và chạy với phiên bản mới, trước khi nạp cho HTN thiết kế. (Xem **http://redhat.activeventure.com/9/customizationguide/s1-custom-kernel-modularized.html**).
  (Building a Custom Kernel from Linux RED HAT source code)

  > *The instructions in this section apply to building a custom modularized kernel. To build a monolithic kernel instead, see Section A.3 Building a*

Monolithic Kernel for an explanation of the different aspects of building and installing a monolithic kernel.

Note

This example uses 2.4.20-2.47.1 as the kernel version (the kernel version might differ). To determine the kernel version, type the command uname -r and replace 2.4.20-2.47.1 with the kernel version that is returned.

To build a custom kernel for the x86 architecture (perform all these steps as root):

1. Open a shell prompt and change to the directory /usr/src/linux-2.4/. All commands from this point forward must be executed from this directory.
2. It is important that kernel build starts with the source tree in a known condition. Therefore, it is recommended that the command make mrproper is issued first to remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. If an existing configuration file already exists as the file /usr/src/linux-2.4/.config, back it up to a different directory before running this command and copy it back afterward.
3. It is recommended that the configuration of the default Red Hat Linux kernel be used as a starting point. To do this, copy the configuration file for the system's architecture from the /usr/src/linux-2.4/configs/ directory to /usr/src/linux-2.4/.config. If the system has more than four gigabytes of memory, copy the file that contains the keyword bigmem.
4. Next, customize the settings. If the X Window System is available, the recommended method is to use the command make xconfig to run the ***Linux Kernel Configuration****.*

Note

To use the graphical tool started with the

 # make xconfig

command, the tk package, which provides the wish command, must be installed. For more information on installing RPM packages, refer to Part V Package Management.

5. As shown in windows, select a category to configure by clicking on it. Within each category are components. Select ***y*** *(yes),* ***m*** *(module), or* ***n*** *(no) beside the component to compile it into the kernel, compile it as a kernel module, or not compile it. To learn more about the component, click the* ***Help*** *button beside it.*
6. Click ***Main Menu*** *to return to the categories list.*
7. After finishing the configuration, click the ***Save and Exit button*** *in the main menu window to create the configuration file /usr/src/linux-2.4/.config and exit the* ***Linux Kernel Configuration*** *program.*
8. Even if no changes were made to any of the settings, running the make xconfig command (or one of the other methods for kernel configuration) is required before continuing.

9. Other available methods for kernel configuration include:
 # make config

An interactive text program. Components are presented in a linear format and answered one at a time. This method does not require the X Window System and does not allow answers to be changed for previous questions.

 # make menuconfig

A text mode, menu driven program. Components are presented in a menu of categories; select the desired components in the same manner used in the text mode Red Hat Linux installation program. Toggle the tag corresponding to the item to be included: ***[*]*** *(built-in),* ***[ ]*** *(exclude),* ***<M>*** *(module), or* ***< >*** *(module capable). This method does not require the X Window System.*

 # make oldconfig

This is a non-interactive script that sets up the configuration file to contain the default settings. If the system is using the default Red Hat Linux kernel, it creates a configuration file for the kernel that shipped with Red Hat Linux for the architecture. This is useful for setting up the kernel to known working defaults and then turning off features not wanted.

Note

*To use kmod and kernel modules answer **Yes** to kmod support and module version (CONFIG_MODVERSIONS) support during the configuration.*

After creating a /usr/src/linux-2.4/.config file, use the command make dep to set up the dependencies correctly.

10. Use the command make clean to prepare the source tree for the build.
11. It is recommended that the custom kernel have a modified version number so that the existing kernel is not overwritten. The method described here is the easiest to recover from in the event of a mishap. For other possibilities, details can be found at http://www.redhat.com/mirrors/LDP/HOWTO/Kernel-HOWTO.html or in the Makefile in /usr/src/linux-2.4.

By default, /usr/src/linux-2.4/Makefile includes the word custom at the end of the line beginning with EXTRAVERSION. Appending the string allows the system to have the old working kernel and the new kernel (version 2.4.20-2.47.1custom) on the system at the same time.

If the system contains more than one custom kernel, a good method is to append the date at the end (or another identifier).

12. Build the kernel with

 # make bzImage

13. Build any modules configured with

 # make modules

14. Use the command

 # make modules_install

to install the kernel modules (even if nothing was actually built). Notice the underscore (_) in the command. This installs the kernel modules into the directory path /lib/modules/<KERNELVERSION>/kernel/drivers (where KERNELVERSION is the version specified in the Makefile). In this example it would be /lib/modules/2.4.20-2.47.1custom/kernel/drivers/.

15. Use

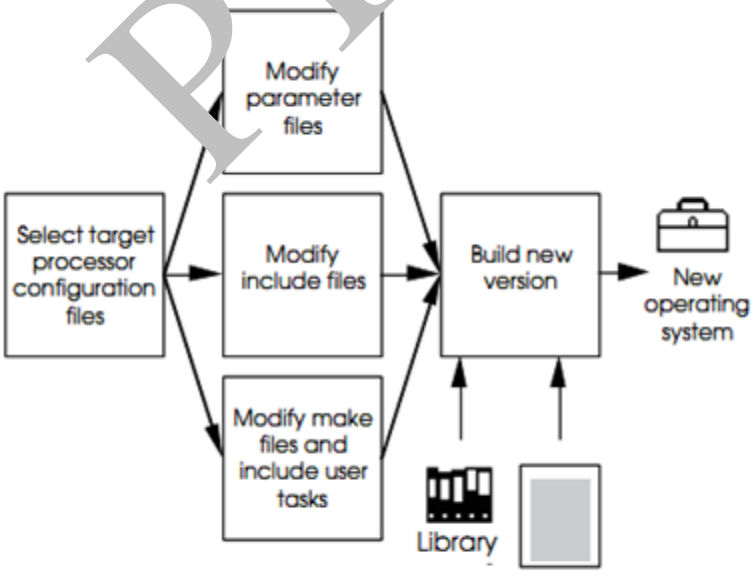
 # make install

to copy the new kernel and its associated files to the proper directories.

In addition to installing the kernel files in the /boot directory, this command also executes the /sbin/new-kernel-pkg script that builds a new initrd image and adds new entries to the boot loader configuration file.

If the system has a SCSI adapter and the SCSI driver was compiled as a module or if the kernel was built with ext3 support as a module (the default in Red Hat Linux), the initrd image is required.

16. Even though the initrd image and boot loader modifications are made, verify that they were done correctly and be sure to use the custom kernel version instead of 2.4.20-2.47. Refer to Section 30.5 Verifying the Initial RAM Disk Image and Section 30.6 Verifying the Boot Loader for instructions on verifying these modifications.)



1. Chọn CPU đích. Cấu hình HW Module Software

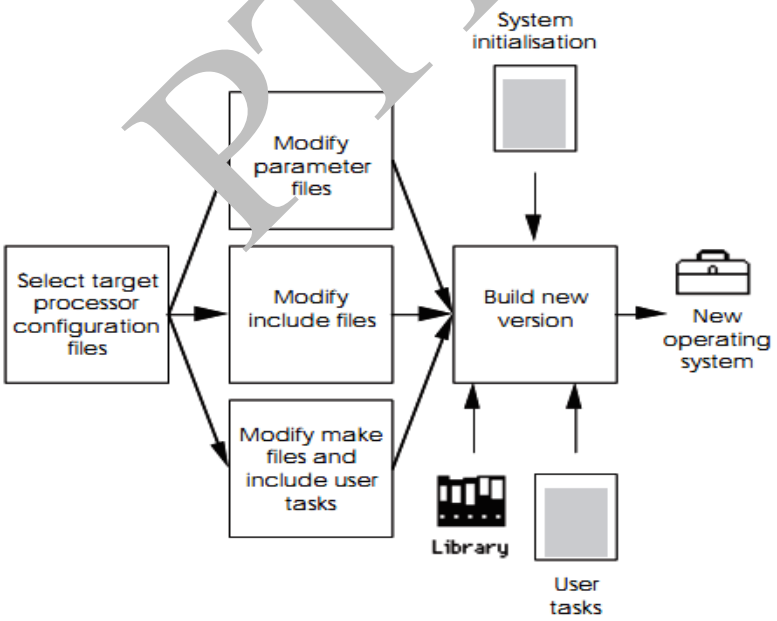
2. Tùy chọn các thống số, tích hợp tiện ích, tệp ứng dụng

3. Xây dựng nhân mới, thử chạy nhân + sử đổi -> 1, đến khi thỏ mãn với HĐH mới.

Hình 4.16 Tổng quát các bước tạo nhân HĐH mới từ mã nguồn

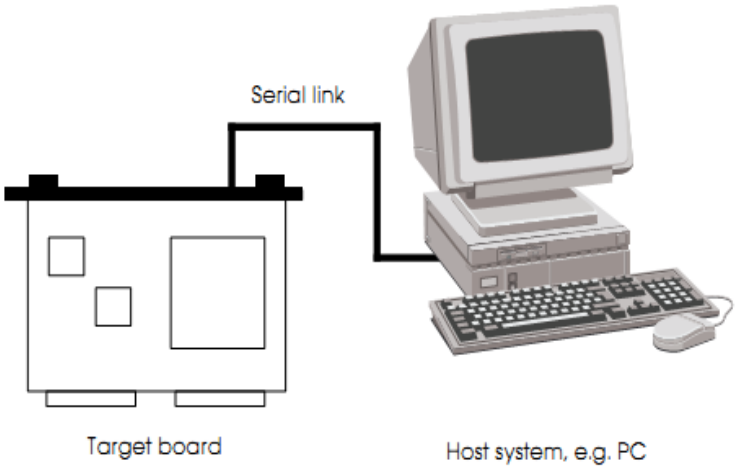
Các bước mô tả gồm:

- Chọn hệ phần cứng đích (chủ yếu là CPU, HW)
- Từ mã nguồn HĐH: chọn các module, thay đổi các tham số của module. Sửa đổi các tệp *header* (*.h). Sửa đổi tệp kịch bản make, thêm vào các tệp *.h của người thiết kế. Kết quả sẽ là một phiên bản mới gọi là *customized OS.*
- Biện dịch (*build*) với thư viện đi kèm mã nguồn, và thư viện của người phát triển phần mềm.
- Tạo nhân HĐH cho HTN
- Trong mô hình này sự khác biệt là nhúng ứng dụng, hay tác vụ (task) của người phát triển phần mềm nhúng vào phiên bản mới của HĐH. Các tác vụ này được đua vào khi biên dịch qua các khai báo #include () vào thời điểm dịch qua tiện ích #make và #link bằng việc khai báo các tệp ứng dụng đã ở dạng *.OBJ. Kết quả sẽ là một phần mềm tích hợp HĐH chuyên dụng và các ứng dụng nhúng. Mặc dù ứng dụng nhúng đã được tích hợp, nhưng việc ứng dụng chạy sau khi boot HĐH thì cần tạo ra các kịch bản (shell script) dạng run command (rc) với thuộc tính được kích hoạt bởi HĐH. Các tệp kịch bản sau đó được tự động kích hoạt sau khi HĐH đã hoàn tất quá trình khởi động. Cách viết các tệp kịch bản cần tham khảo từ HĐH gốc.



Hình 4.17 Tổng quát các bước tạo nhân HĐH mới từ mã nguồn và kiểu khởi động

- ***Tải phần mềm vào phần cứng***



Hình 4.18 Tải (nạp) HĐH vào phần cứng mới

Trước tiên cần nhắc lại về qui hoạch bộ nhớ trên cơ chế hoạt động của CPU đã chọn. Các CPU đều có cách để phân bố sử dụng bộ nhớ, qui định vùng cho ROM-boot, vùng cho HĐH, vùng cho ứng dụng. Các tiêu chí này phải tôn trọng khi biên dịch HĐH, sao cho có thể "đặt" HĐH đúng chỗ trong bộ nhớ. HĐH có thể ở các dạng khác nhau và lưu ở các bộ nhớ khác nhau (ROM, FLASH) và sẽ copy vào RAM để chạy cho nhanh. Phần này đề cập ở các phần sau.

4.2.5 Gỡ rối và mô phỏng

Mô phỏng và gỡ rối là hai công việc thường được dùng để thử nghiệm các đoạn phần mềm quan trọng và khi kết hợp để thử nghiệm với hai: phần cứng và phần mềm, ví dụ TĐKTB. Nói chung các đoạn mã phần mềm như vậy có ảnh hưởng lớn tới hoạt động hệ thống, đặc biệt khi liên quan tới thời gian thực.

- ***Với phần mềm từ ngôn ngữ cấp cao***, có thể chạy kiểm tra cả một phần lớn code mà không cần có phần cứng, thậm chí các code không liên quan tới I/O hay các tiện ích hệ thống (của HĐH), có thể chạy thử trên một máy khác (PC chẳng hạn). Cách làm này cho phép phát triển song song đồng thời cả phần cứng và phần mềm, mà khi hợp nhất đảm bảo hệ sẽ làm việc. Đối với các dữ liệu cho mô phỏng có thể nhận từ bàn phím hay tạo ra các bảng dữ liệu để chạy thử các xử lí, mà không cần có các hoạt động I/O thật sự. Tuy nhiên hạn chế của phương pháp này là ở chỗ mô phỏng là sẽ chạy và dùng thư viện của HĐH trên máy thử chứ không phải trên môi trường của hệ đích. Còn nếu thư viện sử dụng là một phần của code sẽ chạy trên máy đích, thì việc sử code thư viện là cần thiết. Lí tưởng nhất là HĐH máy chạy mô phỏng giống như HĐH HTN đích, ví dụ dùng PC cài Linux chạy mô phỏng cho HTN sẽ chạy Linux. Hiện nay rất nhiều HTN sử dụng HĐH

đích có nguồn gốc từ UNIX/Linux (Android (cho rất nhiều thiết bị nhúng), Ubuntu, bada,,Firefox OS (project name: Boot to Gecko), Openmoko Linux, OPhone, MeeGo (from merger of Maemo & Moblin), Mobilinux, MotoMagx, Qt Extended, Sailfish OS, Tizen (earlier called LiMo Platform), webOS, ipodlinux, Apple IOS, BusyBox (tập hợp các tiện ích của UNIX thu nhỏ), hay Microsoft Windows (Windows Embedded CE, Windows Phone OS), cho các HTN.

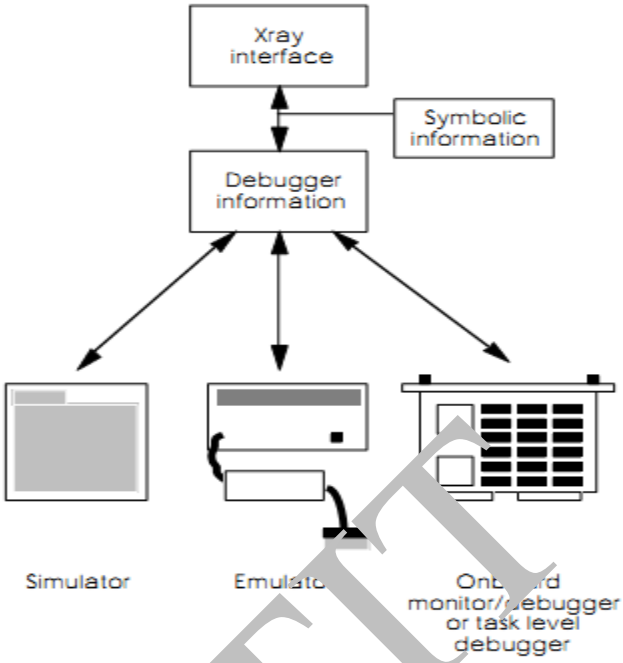
- ***Mô phỏng mức thấp*** có các công cụ để chạy mô phỏng hoạt động của CPU, bộ nhớ và một số thiết bị ngoại vi đích, chạy băng ngôn ngữ máy (hợp ngữ). các mô phỏng này liên quan tới mô hình chương trình di với mô hình bộ nhớ với khả năng gỡ rối. cách mô phỏng cấp thấp này thường không cho các thực thi liên quan tới định thời, tức các code có rang buộc thời gian (cho hệ thời gian thực). Tuy nhiên đây là phương pháp thiết kế ít tốn kém khi có các dự ám HTN lớn và phức tạp.
- ***Gỡ rối trên bo mạch***, là cách gỡ rối trực tiếp trên phần cứng THN đích. Các nhà sản xuất luôn cài sẵn trong EPROM chương trình gõa rối (Debuger), hay một tập các khối thực thi (routine) có thể hợp nhất vào các ứng dụng (tương lai). Các công cụ như vậy tạo điều kiện thử nghiệm hệ thống rất nhanh và chuẩn xác. Các đoạn coe này thường có:
- Khởi động CPU và các Chip khả trình trên bo mạch vào trạng thái ban đầu, hay chạy ở chế độ chờ với dấu nhắc (prompt) trên màn hình;
- Người phát triển chạy Debuger, truy nhập vào các phần cứng qua lập trình đơn giản với các chức năng có sẵn, hay đoạn code nối với cổng truyền thông (COM port) với hệ phát triển đẻ tải xuống các phần mềm (HĐH, ứng dụng nhúng). Lần sau khi khởi động nguội bo mạch, CPU sẽ tìm trong bảng vector ngắt vector RESET có trong EPROM, khởi động bo, và chuyển điều khiển từ EPROM sang code trong RAM. Lưu ý là EPROM phải nằm ở vung địa chỉ phù hợp với địa chỉ đầu tiên sau khi RESET. Debuger cho phép truy nhập, theo dõi, thay đổi giá trị các thanh ghi của CPU, rất tiện cho gỡ rối code. Các giá trị thường biểu diễn ở dạng số hệ 16 (hexadecimal). Để tiện cho gỡ rối, khi lập trình cần tạo các bảng kí hiệu (symbol table, hay bảng các biến) khi dịch hay khi link phần mềm, tạo khả năng tra chéo giữa các nhãn và tên kí hiệu qui chiếu vào địa chỉ vật lí, từ đó theo dõi dễ dàng các giá trị cần quan sát ứng với các kí hiệu đó. Có thể theo dõi khi debug khi dùng với ***tệp hợp dịch*** (*listing file,* là các tệp có phần mở rộng ***.lst** tạo ra sau khi dịch tệp hợp ngữ ***.asm**) thì rất dễ theo dõi. Debug mức thấp mang lại kết quả tin cậy về hoạt động của hệ đích, đặc biệt với các HTN khó chạy theo cách mô phỏng, hay mô phỏng không thực tế sát với hệ đích.
- ***Gỡ rối ở mức tác vụ (task debug)***. Một số trường hợp gỡ rối mức thấp không nhiều hiệu quả tuy có thể debug cả một khối lệnh (hay routine) khi chọn điểm dừng (break point) code rồi quan sát kết quả. Thực tế cần debug cho cả các task và cách này cần debug ở mức độ HĐH, bởi các task khi chạy phụ thuộc nhiều vào bối cảnh thực thi code, nên điểm dừng chỉ có thể đặt qua (ở) các thông điệp, các sự kiện xuất hiện/kết thúc, các xử lí

ngắt (bắt đầu/kết thúc). Xử lí sự kiện có thể dùng cách lọc (bỏ/chấp nhận), ghi lại (trace) kết quả thực thi các thông tin về thời gian, suwrb dụng bộ nhớ, trạng thái tác vụ, giá trị các thanh ghi CPU để phán đoán, đánh giá các kết quả.

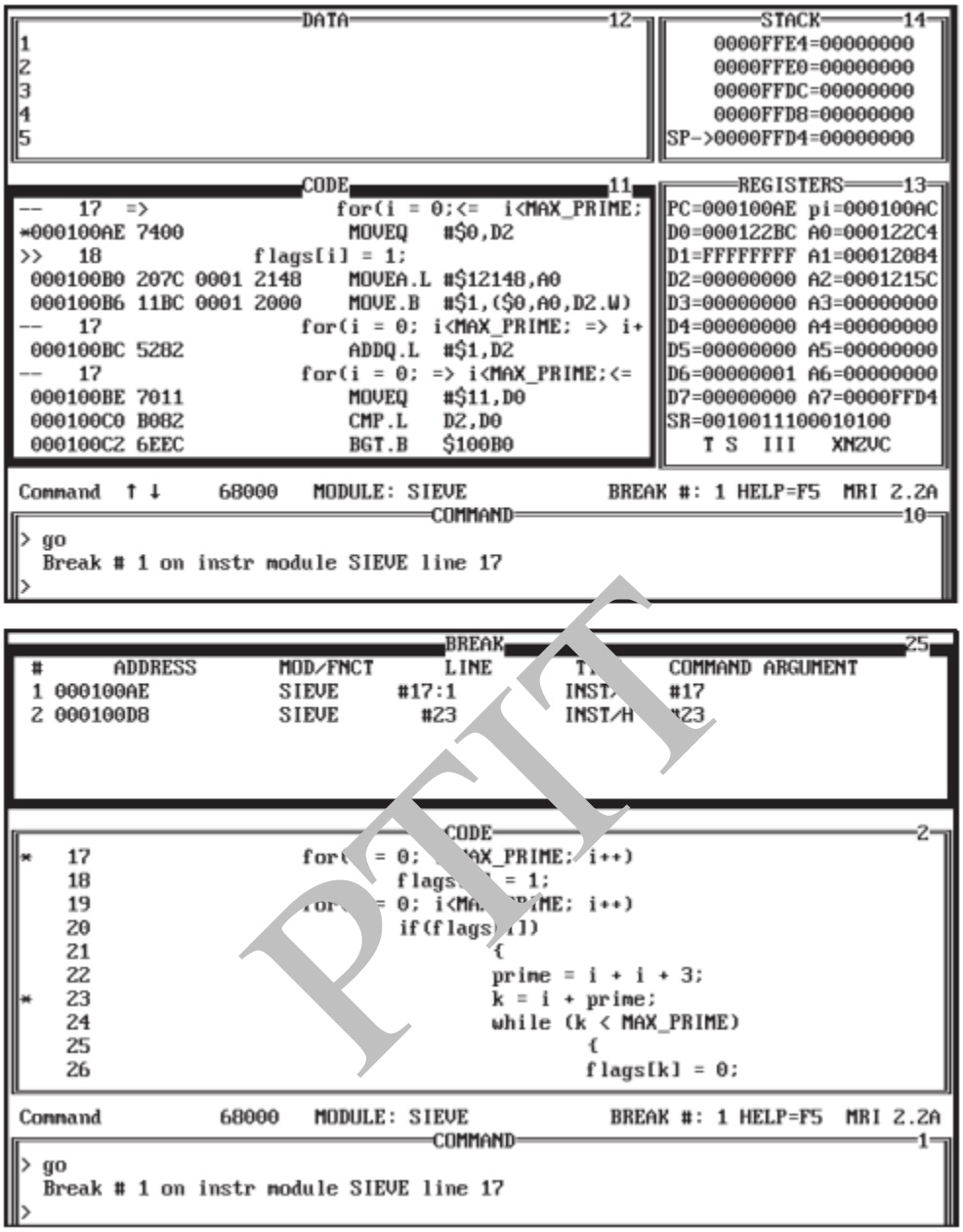
- ***Gỡ rối kí hiệu (symbolic debug)*:** Khi gỡ rối bằng ngôn ngữ bậc cao, như C, thay vì dùng tệp hợp dịch để xác định địa chi điểm dừng, gỡ rối tượng trưng cho phép dừng ở dòng lệnh hay tại tên của hàm. Cách tương tác như vậy hiệu quả hơn so với khi dùng hợp ngữ. Lí do sử dụng phương pháp này đó là cách mà gỡ rối tượng trung làm việc: có thể coi đây là kiểu đầu cuối của gỡ rối hợp ngữ khi phần mềm gỡ rối tra cứu (look-up) và chuyển đổi giữa cấu trúc của ngôn ngữ bậc cao và cách tính địa chỉ, nội dung của hợp ngữ. Chìa khóa của cách này là tạo ra bảng các kí hiệu (symbol) có khả năng cung cấp các thông tin tra chéo (look-up) mà gỡ rối cần đến. Một tệp dạng nhị phân liên quan tới object và tệp chứa địa chỉ tuyệt đối được ra, hoặc 2 tệp tách biệt. Khi gỡ rối kích hoạt tệp này. Nó sẽ tìm các thông tin kí tự để hiển thi các thông tin có nhiều ý nghĩa, và điểm dừng chương trình có thể đặt ở các câu lệnh cũng như địa chỉ cụ thể. Nói cách khác có thể gỡ rối theo giỏi code (trace) qua từng dòng lệnh, hay từng lệnh CPU. Có chút vấn đề xảy ra là: bảng kí hiệu sẽ lớn, cần nhiều bộ nhớ, kích thước tệp cũng lớn theo. Điều này sẽ rắc rối khi xây dựng ừn dụng cho một HĐH mới. Lỗi sẽ xuất hiện khi link là 'bảng kí hiệu bị tràn (*symbol table overflow error*). Có thể xử lí bằng cách chặn (dừng) gỡ rối kí hiệu hay cấp thêm bộ nhóe cho trình liên kết (linker). Có thể còn có nhiều mẹo hơn khi gỡ rối !
- ***Tối ưu code:*** tối ưu code có thể thực hiện vào lúc biên dịch mã nguồn với tùy chọn đặt với thông số tối ưu ở dòng lệnh. Khi đó trình chương trình dịch sẽ kiểm tra code, thay đổi (code, trình tự gọi code …) để tăng hiệu quả thực thi mà không làm thay đổi tư duy logic của chương trình. Có nhiều cách, tuy nhiên có hai cách phổ biến sau đây: gỡ bỏ hay thêm hay thay đổi code. Chương trình dịch có thể boỏ ài biến, hay các đoạn code thực thi (routine) không bao giờ sử dụng hay trả lại bất kì hàm chức năng nào. Các vòng lặp cũng cần tối ưu để bỏ đi các trễ rẻ nhánh (ở các chương trình lớn). Đoạn code thực thi dấu phẩy động có thể thay vào bằng các dòng lệnh dấu phẩy động trực tiếp. kết quả cuối cùng có thể khác so với tệp *.lst trước đó, tương tự bảng các kí hiệu (symbol table) cũng có thay đổi không như hoài vọng từ mã nguồn. Tuy nhiên cách tối ưu này cũng không hẳn là đạt yêu cầu, bởi như trình bày, có thể một vài biến bị loại trừ, vòng lặp hay hàm delay có thể sẽ bị thay bằng NOP đơn giản. Hiệu quả thời gian thực thi có thể nhanh, nhưng kết quả tính toán thì cần thẩm định ! Hãy thận trọng với các đoạn code liên quan tới I/O và liên quan tới ánh xạ I/O vào bộ nhớ, tối ưu code có thể xảy ra lỗi ! *Để chắc chắn và thẩm định lại code, hãy so sánh hai kết quả thể hiện ở các tệp *.lst sau khi dịch có tối ưu và không tối ưu !*
- ***Xray (****Soi tìm lỗi kiểu chụp X quang)*: Gỡ rối phần mềm như mô tả có vẽ nhiều bước, tuy nhiên cũng tồn tại phần mềm mang đặc điểm hợp nhất cho gỡ rối. Đó là phần mềm của công ty Microtec có tên Xray. Xray là tập các công cụ hỗ trợ gỡ rối hợp nhất của

compiler, công cụ *debuger* liên kết với *emulator*, hổ trợ gỡ rối tới mức tác vụ trên bo mạch.

Xray structure:

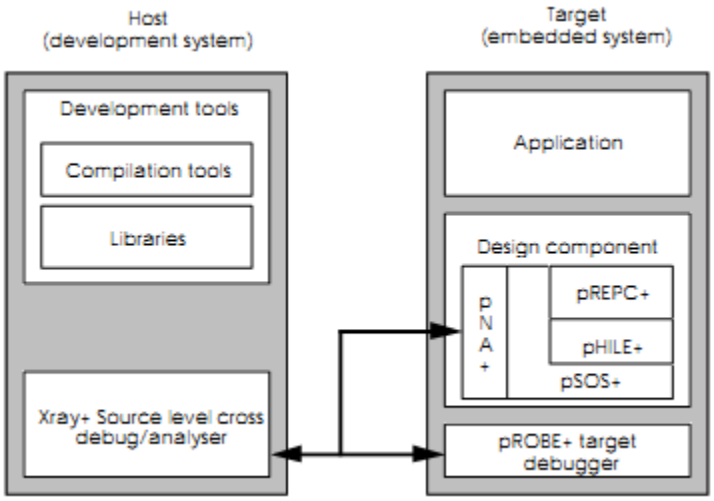


Hình 4.19 Các loại công cụ hổ trợ gở rối



Xray screen shots

Hình 4.20 Kết quả hiển thị của gở rối



Hình 4.21 Liên kết giữ hệ phát triển và hệ đích đang được gở rối

- ***Hệ phát triển-HPT (development system)***: Một giải pháp khác để phát triển phần mềm cho HTN là sử dụng HĐH đích là công cụ phát triển ngay trên bo mạch hay trên một máy tính khác tương tự. Cách này là cách truyền thống trước khi có các PC giá rẻ kết hợp phần mềm dịch chéo (*cross-compilation*) hợp nhất trên đó. Thuận lợi khi dùng HPT là code tạo ra chính là code cho hệ đích, kể cả các thư viện động, không có thay đổi. Phần mềm đích có thể thử nghiệm trên HPT trước khi tải xuống hệ đích. HPT có HĐH đầy đủ và có sẵn công cụ gỡ rối mà công cụ này không có trên hệ đích. Tuy nhiên cũng có vài điểm cần lưu ý:
- *Chức năng dấu phẩy động và quản lí bộ nhớ (Floating point and memory management)*. Hai thành phần này có trên HPT, nhưng hệ đích không có. Có nghĩa là code tạo ra trên HPT sẽ không chạy trên hệ đích, do đó cần viết lại các code này và liên kết vào code đích. Ngược lại code mới này có thể không chạy trên HPT, lí do là hệ mô phỏng.
- ***Kĩ thuật mô phỏng (Emulation)***: Một trong các kỉ thuật mô phỏng là *In-circuit emulation (ICE)* dùng để chạy mô phỏng một CPU trong quá trinh thiết kế HTN. ICE cho ta một 'cửa sổ" để tương tác với hệ đích. Thông thường phần mềm này sẽ nối vào đế cắm của CPU hệ đích (bo không có CPU!) qua một thiết bị hỗ trợ (cổng *JTAG-Joint Test Action Group* hay BDM- Background Debug Mode), dùng phần mềm mô phỏng CPU, lập trình và chạy chương trình. Hệ chủ có thể là một máy *Analyzer logic* với nhiều đầu tín hiệu để ghi nhớ các giá trịn trên BUS của CPU. Đây là công cụ mạnh và khá đắt, nhưng rất hiệu quả để phát triển HTN. Ngoài ra còn có một thiết bị khác kết hợp với *Analyzer logic* với các đầu cặp (*probe*) vào bất kì tín hiệu nào trên bo mạch để kiểm tra, gỡ rối, đó là OnCE (on-chip emulation).

  (Đọc thêm các thiết bị liên quan !)

4.2.6 Phát triển HTN

Có nhiều giải pháp lựa chọn để phát triển HTN, có thể nêu ra như sau:

1) Dùng bo mạch thương mại, máy tính nhúng chuẩn, kiểu **PC** gọi là PC/104 (*embedded computer standard* kiểm soát bởi PC/104 Consortium) với một số **CPU đa năng** (dòng **low-cost x86 processor).**
   Loại này thuộc kiểu controller, có hệ điều hành, có IO như máy tính thông thường, lắp theo kiểu xếp chồng lên nhau, hình khối: motherboard, analog-to-digital converter, digital I/O (data acquisition) module, ghép nối với các ngoại vi khác có trên thị trường, BUS ISA, PCI, kể cả thiết bị thu tín hiệu GPS, mạng không dây IEEE 802.11, và USB (đã đề cập ở chương 1). Hệ điều hành: DOS 6.4, Linux, RTOS.

   Với giải pháp này, công việc tiếp theo phần lớn là phát triển các phần mềm ứng dụng, các trình điều khiển thiết bị, thư nghiệm và đưa vào ứng dụng. Nếu lớp ứng dụng nhúng nhỏ, dùng giải pháp này sẽ lãng phí.

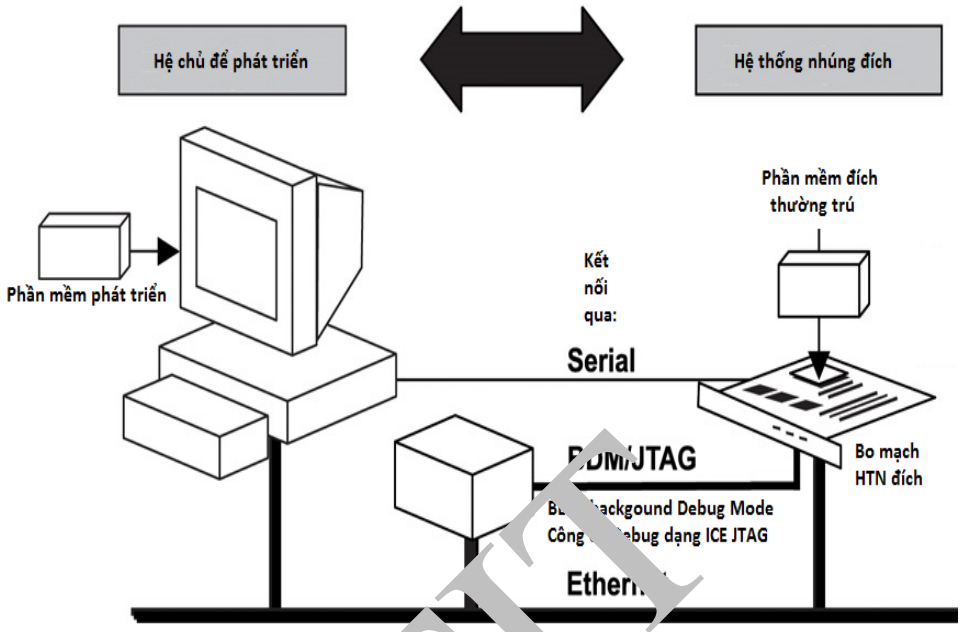
   Tham khảo thêm: http://www.controlled.com/pc104faq/

   http://www.pc104.org/pdf/PC104_trade_show.pdf

2) Thiết kế theo yêu cầu (*customize)*
   Giải pháp này có thể sử dụng để xây dựng các HTN với các qui mô rộng, từ đơn giản đến phức tạp. Qui trình và kĩ thuật thiết kế như đã trình bày ở các phần trên. Chủng loại CPU cho giải pháp có phổ rất rộng. Hiện ở nước ta phổ biến các công nghệ sau đây: dùng chip PIC ((*Programmable Interface Controller* của Microchip Technology**),** hay PSoC (*Programmable System-on-Chip* của Cypress Semiconductor) xây dựng các micro-controller. Các sản phẩm loại này dạng Kit hay linh kiện rời có công cụ phát triển bán kèm.

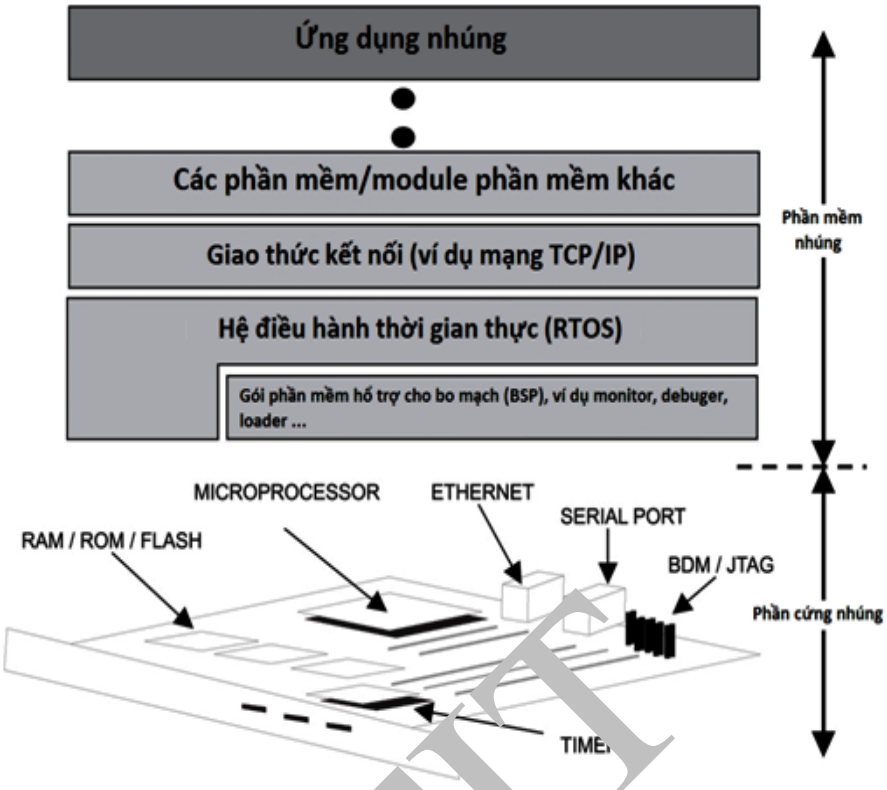
   Ví dụ thiết kế bo mạch từ linh kiện: Xem ví dụ *Hình 2.18- Cấu hình tối thiểu bo mạch CPU 8085, RAM/ROM/Ports* với CPU 8085, ROM 8755, port 6165

- ***Hệ phát triển, công cụ xây dựng phần mềm và nạp vào HTN đích***
  - ✓ Hệ phát triển:



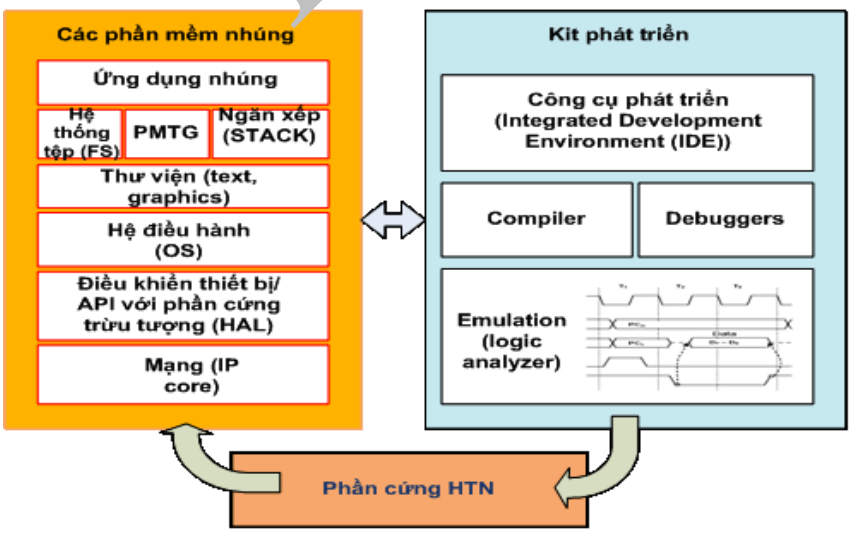
Hình 4.22 Môi trường phát triển nhúng theo: hệ phát triển – công cụ - HTN đích

  - ✓ Phần mềm đích trong một hệ thống nhúng, sữ gọi tắt là ***code image:***

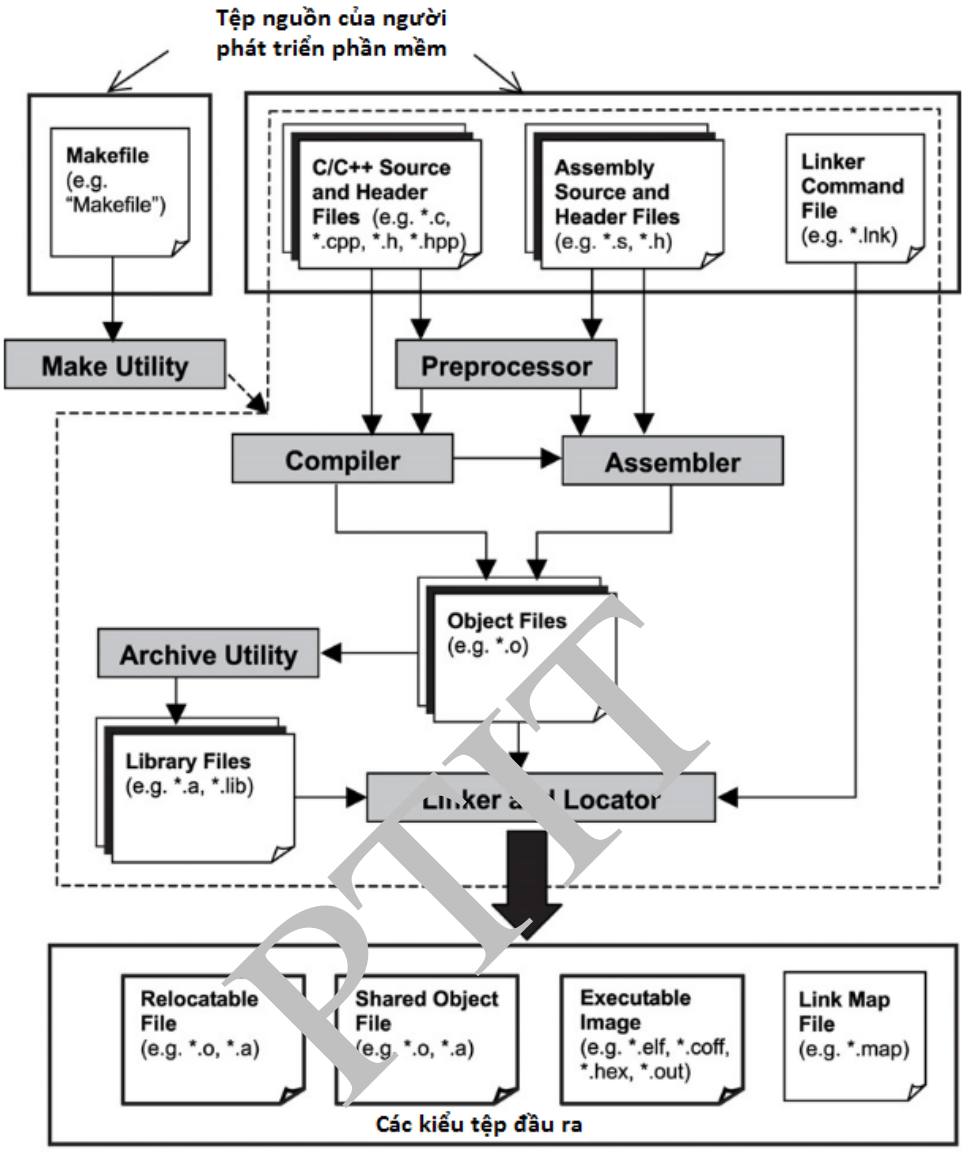


Hình 4.23 Hệ thống nhúng – phần mềm nhúng và phần cứngnhúng

✓ Xây dựng, phát triển phần mềm nhúng gồm:
Lập trình, gỡ rối, mô phỏng (cứng/mềm), hiệu chỉnh, …



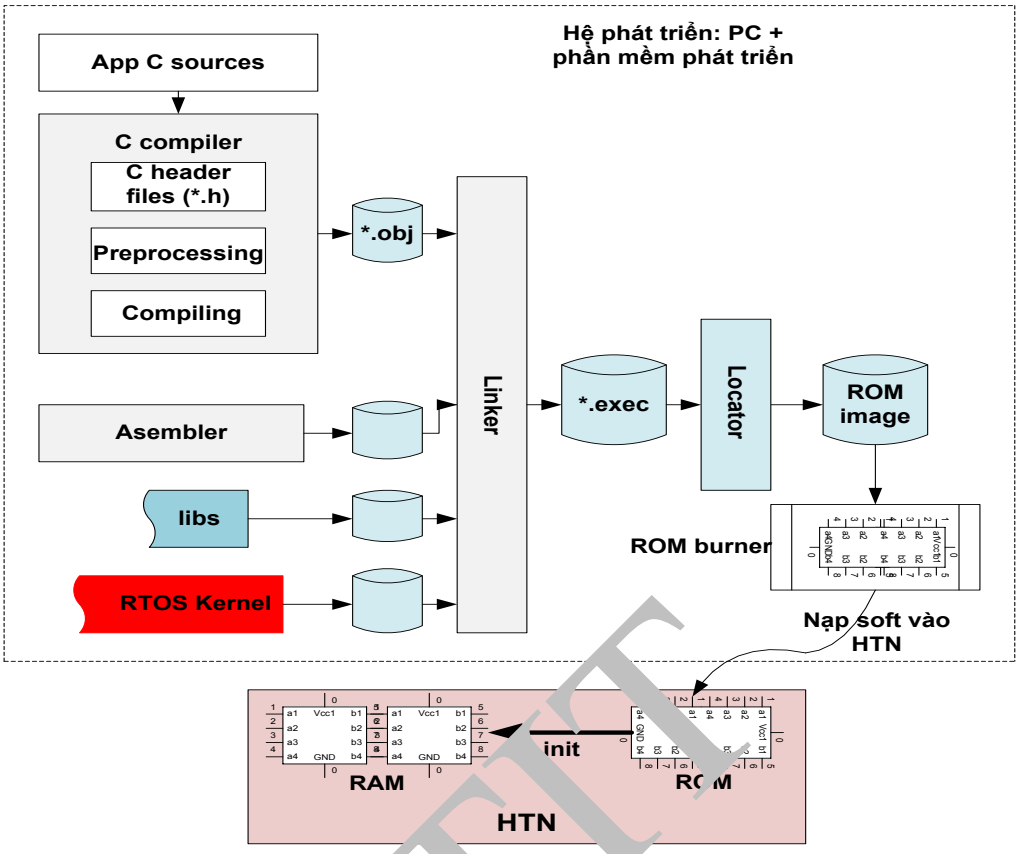
Hình 4.24 Hệ phát triển HTN



Hình 4.25 Qui trình phát triển phần mềm đích để nạp vào HTN đích.

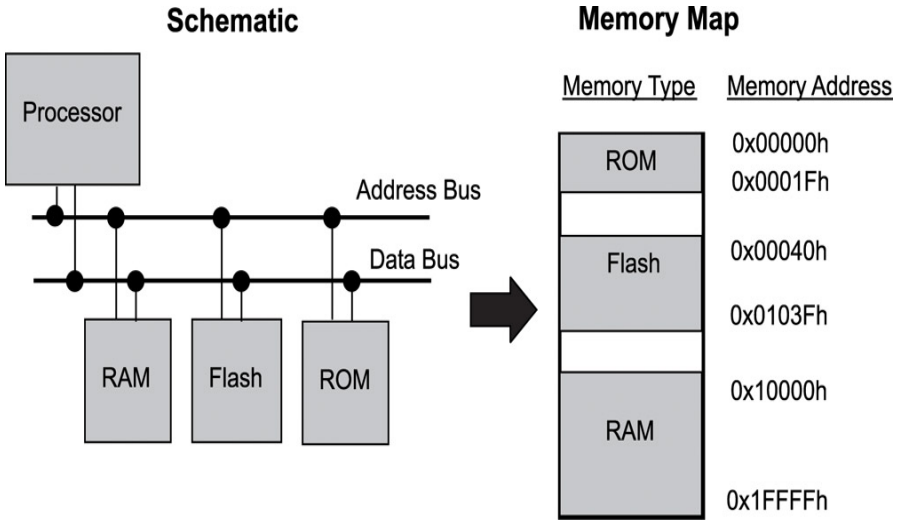
- ✓ Nạp vào bo mạch đích
  Thử nghiệm, đánh giá

  Hoàn chỉnh.



Hình 4.26 Qui trình phát triển phần mềm cho HTN

Công cụ liên kết (*linker)* và định vị (*locator)* tạo ra tệp thực thi kiểu ELF (executable and linking format) dạng nhị phân (kiểu image hay *code image*) có thể ánh xạ hay nạp vào ROM.



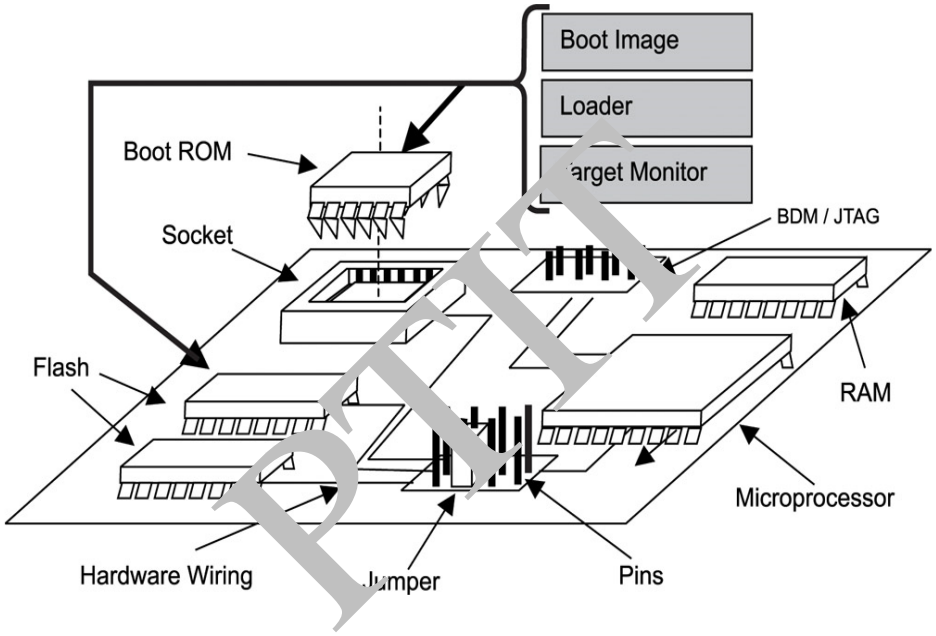
Hình 4.27 Sở đồ đơn giản hệ thống và ánh xạ bộ nhớ vào EEPROM hay FLASH của HTN đích.

- **Khởi động hệ thống**

Sau khi phần mềm phát triển đã đạt được các chức năng cơ bản, đã đến lúc "nạp" vào HTN đang xây dựng để chạy thử. Trong mọi trường hợp, dù HTN có hay không có hệ điều hành, đều cần có một qui trình, gọi là " khởi động hệ thống". Khởi động hệ thống (hay *boot*) nằm trong quá trình thiết kế, kết hợp một vài yếu tố như sau:

- ✓ **Tạo công cụ**:

Sử dụng chương trình nạp (*loader*) với chức năng sẽ "nạp" tệp ảnh (code image) từ hệ phát triển (máy PC) vào HTN:



Hình 4.28 Vai trò của trình loader

Loader được viết riêng và sẽ được ghi vào boot ROM. Một phần địa chỉ của boot ROM sẽ chứa *boot image* (là phần code do kỉ sư phần cứng viết) sẽ thực hiện các mã cần thiết theo qui trình bật máy nguội (Power *ON System Test-Cold Boot*): khởi động phần cứng, đưa các vi mạch, phân vùng bộ nhớ, không gian địa chỉ … vào trạng thái ban đầu. Sau hết boot loader "nhảy" tới địa chỉ RESTART hay START của phần mềm hệ thống để chuyển điều khiển cho nó. Từ lúc đó hệ thống bắt đầu hoạt động.

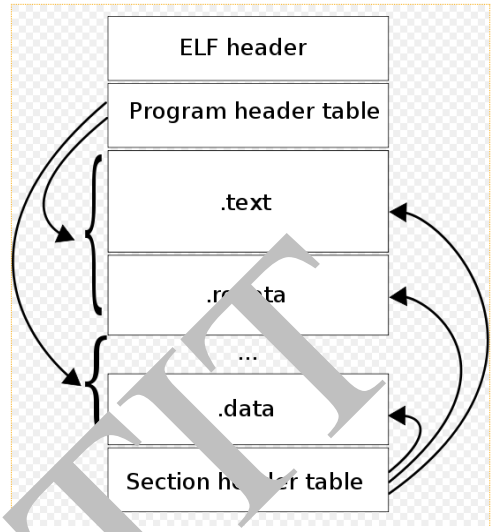
Trong quá trình phát triển, các phần mềm cần có bao gồm:

- Boot hệ thống nạp vào EEPROM/ROM; Boot ROM cần có tiện ích có khả năng kết nối với hệ phát triển, ví dụ như giao thức TFTP, dung để tải phần mềm vào FLASH;

- Phần mềm hệ thống (Monitor hay RTOS);
- Phần mềm ứng dụng nhúng;
- Tải phần mềm hệ thống và các ứng dụng nhúng vào FLASH. Nếu nạp vào EEPROM rồi thì không cần bước này).

- **Nguyên lí khởi động hệ thống:**

Nhắc lại định dạng tệp thực thi ELF (Executable and Linkable Format)

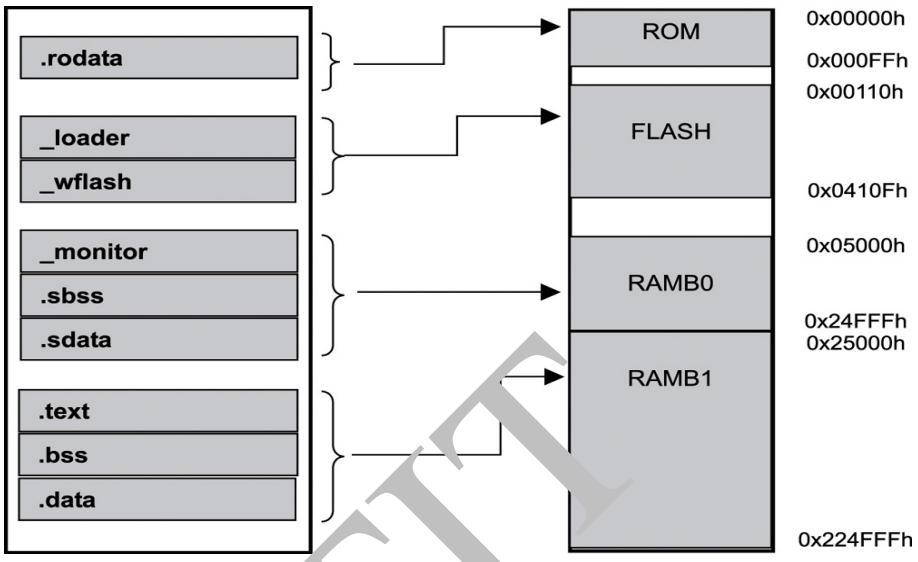


*Hình 4.29 Phần program header table chỉ ra các phân đoạn được sử dụng lúc chạy chương trình (runtime) và phần header liệt kê tập các phần nhị phân : **.text**: mã chương trình, **.rodata**: dữ liệu chỉ đọc, **.data**: dữ liệu đọc/ghi được.*

- ✓ Phần cứng liên quan tới RESET của loại CPU sử dụng.
- ✓ Khởi động có hai cách: "khởi động nguội" là khi bật nguồn máy, và khởi động nóng, tức RESET nóng khi máy đã bật nguồn, hay chính xác là khởi động lại nóng.
  Khi thiết kế phần cứng phải có mạch điện tử tạo ra xung RESET và nối vào chân RESET của CPU. Độ rộng của xung này bằng mấy CPU-Clock phụ thuộc vào từng loại CPU sử dụng và cần thực hiện chính xác. Đầu vào này thực tế là tổ hợp của một số tín hiệu quan trọng, có tác động khởi động lại máy. Ví dụ tín hiệu từ "watchdogs", các sự kiện sự cố hệ thống, cần thoát khỏi vòng lặp quẩn và có thể có một vài tín hiệu khác, phụ thuộc vào thiết kế.

- ✓ Phần mềm thực hiện có tên phổ biến là: boot, bootstrap, hay bootloader (với hệ có hệ điều hành). Phần mã thực thi gọi là ***boot code***. Thông thường mã này nằm trong ROM như là một phần của BIOS. Một số CPU có kiến trúc với một bộ đếm chương trình

(Program Counter – PC) tự động cấu hình chứa địa chỉ của ROM mà tại địa chỉ đó là lệnh đầu tiên sẽ được thực hiện, hay phức tạp hơn là lệnh "nhảy" tới một bảng để chọn "chế độ máy" sẽ khởi động (ví dụ khi khởi động hệ thống có hệ điều hành, với Intel X86, ban đầu chạy chế độ thực (*real mode*), sau đó nạp nhân hệ điều hành và chuyển sang chế độ chạy bảo vệ (*protected mode*) với hệ điều hành).



Hình 4.30 Ánh xạ thực thi chuyển vào bộ nhớ của hệ thống

.rodata: Các thống số khởi động hệ thống, thường không đổi, do dó để ở ROM;

_loader hay *_wflash* : sử dụng code hay ***loader***, code này để ở RAM hay FLASH;

_monitor: mã của chương trình monitor.

.sbss (block *started by symbol)* hay nơi chứa các biến tĩnh có hay chưa khởi động với giá trị bằng 0 trong ngôn ngữ C), và,

.sdata: HTN dùng monitor hay OS được tái định vị vào RAM;

.text: mã chương trình hệ thống *và* mã ứng dụng nhúng.

Khi hiểu được phác thảo bố trí bộ nhớ của CPU sử dụng, sẽ dễ dàng hơn khi nạp mã thực thi vào hệ thống. Hãy theo dõi như trên hình 4. 11:

- Sau khi kích hoạt RESET CPU thực hiện *khởi động cứng*, đẩy CPU bắt đầu thực hiện một chương trình, hay một mã "nhảy" tới một đoạn mã khác tiếp tục quá trình khởi động, gọi là *khởi động mềm*.

- Khởi động mềm (*start code*, hay *loader OS*) thực hiện một loạt các thao tác tạo ra sự phân chia bộ nhớ thành các vùng chức năng (STACK, DATA cho hệ điều hành, nhân hệ điều hành, …), khởi động các vi mạch với các thông số có trong BIOS DATA ở ROM, … và nạp hệ điều hành và chuyển tới lệnh đầu tiên của nhân hệ điều hành. Nhân tiếp tục khởi động các cơ sở dữ liệu của nhân, khởi động các dịch vụ nhân … và chuyển sang chế độ bảo vệ.
  Với các bo mạch thương mại, *start code* cho HTN cần khả năng tùy biến để áp dụng cho các bo CPU khác nhau. Để có chức năng này, bộ dịch C tự động sinh code có chứa tệp hợp ngữ tách riêng chưa mã của *start code*. Các tệp này có tên crt0 hay crt1 (crt=C Run Time). Tùy biến các tệp này để có start code phù hợp với bo mạch.

✓ Đối với các sản phẩm bo mạch trên thị trường, các bo mạch đều có gói phần mềm hỗ trợ đi cùng (*board support package* (BSP)) với mã chương trình cho hệ điều hành lựa chọn chạy trên bo mạch. Thông thường là một loại *bootloader* kết với thiết bị chứa hệ điều hành và các TĐKTB (device drivers) để điều khiển tất cả các thiết bị trên bo.

- **Ví dụ về một phần mềm hỗ trợ : Monitor**

Một giải pháp tích hợp *loader* và *boot image* là sử dụng một phần mềm *monitor nhúng*. Đó là một loại ứng dụng nhúng mà các nhà sản xuất các HTN cung cấp để sử dụng trong quá trình người dùng phát triển bo mạch nhúng của họ. moni tỏ giuap người phát triển phần mềm cho HTN kiểm tra gỡ rối hệ thống đích ngay khi cho hệ hoạt động (*run time*). Giống như boot image, monitor được thực hiện khi bật nguồn và nó sẽ khởi động hệ thống:

  - ✓ Khởi dộng các thiết bị, như các cổng các kết nối (nối tiếp, song song, mạng). khởi động các bộ định thời, …
  - ✓ Khởi động địa chỉ của bộ nhớ để tải phần mềm hệ thống,
  - ✓ Khởi động vi mạch điều khiển ngắt, nạp các vector ngắt, các ISP …

Monitor định nghĩa giao diện người dùng qua thiết bị đầu cuối (là PC) qua đương kết nối, thường là kết nối nối tiếp, với chế độ dòng lệnh (*Command Line Interface-CLI*), qua đó người phát triển có thể thực hiện:

  - ✓ Tải image code xuống hệ đích;
  - ✓ Đọc/ghi vào bộ nhớ của hệ thống đích;
  - ✓ Đọc/ghi các thanh ghi của CPU của hệ thống đích;
  - ✓ Lập và xóa các điểm dừng khi chạy gỡ rối;
  - ✓ Thực hiện chạy từng lệnh;
  - ✓ Reset hệ thống qua lệnh.

Monitor thường do kỉ sư phần cứng viết ra dùng cho cả hai chức năng: chuẩn đoán phần cứng (*diagnostics*) và mã gở rối cấp thấp. phần mềm monitor có thể có ở dạng mã nguồn, nên việc tùy biến cho hệ cụ thể là rất khả dĩ.

- **Kịch bản boot hệ đích**

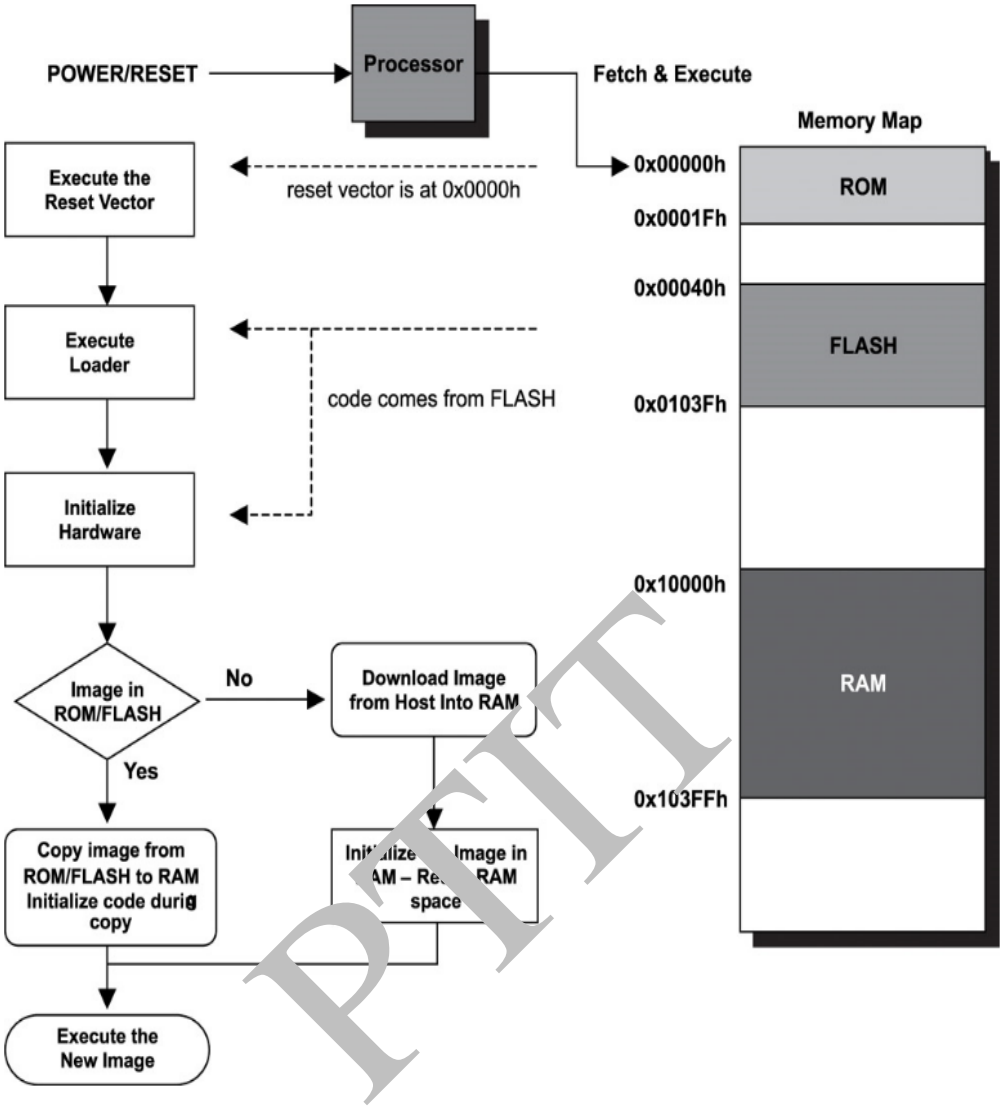
  Phần này giới thiệu quá trình khởi động hệ đích, tự đưa mình vào hoạt động.

Các CPU của HTN sau khi bật nguồn, hay RESET (mềm do định thời, hay cứng ấn công tắc) sẽ tìm và thực hiện các lệnh đầu tiên mà địa chỉ do CPU được thiết kế. mã tại địa chỉ này còn gọi là *reset vector.* Thông thường reset vector đơn giản là một lệnh nhảy (JMP addr) tới một địa chỉ khác, mà ở đó mã khởi động chính sẽ được thực hiện (xem ví dụ ở PC chuẩn phía sau). Sở dĩ như vậy là vì các CPU có một không gian điọa chỉ rất nhỏ dành riên cho các mục đích đặc biệt, do vậy nếu đoạn mã khởi động quá dài sẽ không đủ chổ cho nó. Hơn nữa reset vector cung như mã boot khởi động phải nằm ở vị trí nhớ cố định (trên ROM hệ thống, hay FLASH trên bo mạch, hay trên RAM không bị mất nội dung NVRAM). Khi mô tả loader ở đây sẽ qui chiếu tới mã thực thi bẩy boot hệ thống (*system-bootstrap*), tải phần mềm hệ thống (boot image code) và các bước khởi động.

Tất cả sẽ được giới thiệu thông qua ví dụ, là đơn giản nhất.

Giả định HTN được phát triển và lập trình với bộ nhớ FLASH trên bo mạch đích. Phần mềm đích (boot image) là tập các phân đoạn chương trình khác nhau. Mỗi phân đoạn có vị trí xác định trong bộ nhớ. Reset vector chứa trong một ROM nhỏ được ánh xạ vào địa chỉ 0x0h. ROM chứa một số giá trị (hằng) thường thường mà CPU cần có khi reset. Các giá trị đó là reset vector, con trỏ thanh ghi ngăn xếp (stack pionter), một số địa chỉ của RAM …

Ví dụ được minh họa như hình sau:



*Hình 4.31 Ví dụ tổng quan về **boostrap** hệ thống*

Reset vector định vị tại 0x0000h của ROM, từ đó có lệnh JMP 0x00040h: nhảy tới địa chỉ 0x00040h ở FLASH. Điều khiển (hay ***startup initialization***) bắt đầu thực hiện ở địa chỉ này. Mã ở đây gồm chương trình tải boot image code, các ngắt ngoại lệ của hệ thống (trỏ tới các ISP tại FLASH) và các chương trình khác. Trên hình là đoạn "code comes from FLASH".

Phần đầu tiên của quá trình bootstrap là đưa hệ vào trạng thái đầu tiên (*known state*): các thanh ghi của CPU được khởi động với các giá trị mặc định phù hợp, stack pionter nạp giá trị đã xác định (hằng) trong ROM. Loader sẽ cấm ngắt ở giao đoạn này vì chưa có các ISP trong bộ nhớ; khởi động địa chỉ của RAM, địa chỉ của cache (nếu có). Loader thực hiện kiểm thử các thiết bị với vài chế độ cơ bản.

Phần mềm chạy ở RAM thường nhanh hơn ở FLASH, do đó loader sẽ copy một phần mã từi FLASH vào RAM. Tại đây phần loader sẽ có hai địa chỉ: địa chỉ để nạp vào RAM và địa chỉ chạy của chương trình đã copy và RAM. Ở đây sử dụng debuger sẽ giúp giám sát tình huống *out-of-RAM* khi copy và chạy.

Mã thực thi có phần dữ liệu (DATA segment) đã được khởi động và phần chưa được khởi động. Phần dữ liệu này cần copy vào RAM và do đó đọc / ghi được. Các mã lập trình như *.data* hay *.sdata* chứa các giá trị khởi động cho các biến tổng thể và các biến cục bộ. Phần mã xác định ở *.bss* và *.sbss* không khởi động lúc này.

Tiếp theo là khởi động các thiết bị. Chỉ các thiết bị mà loader cần là được khởi động vào thời điểm này. Các thiết bị này chỉ là một phần của I/O và sẽ được khởi động tất cả sau khi imagr code đã được tải hoàn taastvaf thực hiện ở khâu startup.

Đến đây loader đã đủ các điều kiện để tải các phần mềm hệ thống, RTOS, ứng dụng … các phần mềm này xuất phát từ hai nguồn:

1. Từ hệ phát triển;
2. Từ thiết bị nhớ chỉ đọc nào đó bê ngoài trên h phát triển.

- **Các kịch bản chạy phần mềm hệ ống có thể như sau:**

1. Chạy từ ROM, còn RAM dành ch dữ liệu

Một số HTN có giới hạn bộ nhớ do đó hệ sẽ khởi động trực tiếp từ ROM. Trong trường hợp này sẽ không có quá trình copy mã lệnh vào RAM để chạy. Tuy nhiên không gian cho dữ liệu vẫn phải xác điịnh RAM (nhớ khi lập trình). Có 2 thanh ghi cơ bản là IP (Instruction register)-thanh ghi lệnh, trỏ vào lệnh tiếp theo sẽ thực hiện (.text) và SP (Stack Pointer), trỏ vào địa chỉ tiếp theo trong ngăn xếp. ngôn ngữ C sử dụng ngăn xếp để truyền các thông số khi kích hoạt một hàm. Vùng ngăn xếp phải ở RAM và SP phải trỏ vào đó khi khởi động CPU.

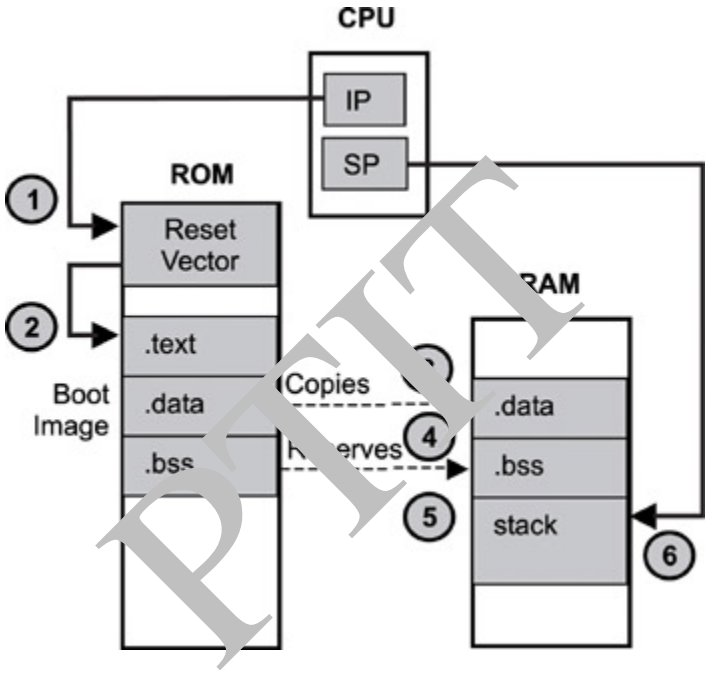
Qui trình khởi động như sau:

1) Thanh ghi IP được thiết kế cứng để thực hiện lệnh đầu tiên trong bộ nhớ, đó là *reset vector.*
2) Reset vector nhảy tới lệnh đầu tiên của phần .text của mã boot (tức boot boot image), *.text* thường trú trong ROM; CPU dùng IP để thực hiện *.text*, khởi động bộ nhớ, kể cả RAM.
3) Phần *.data* của boot image được copy vào RAM để có thể đọc/ghi.
4) Xác lập *.bss* trong RAM.

5) Xác lập ngăn xếp .stack trong RAM, khởi động SP trỏ vào địa chỉ đầu của stack.

6) Hoàn tất khâu khởi động, CPU tiếp tục thực hiện các lệnh trong .text cho tới khi hệ thống hoặc shutdown hay RESET.

Lưu ý ở đây là các lệnh đầu ở bước 1) không ở định dạng chuẩn ELF mà đơn giản là mã máy nhị phân sẵn sàng chạy. Nhưng boot image ở định dạng ELF (viết ra từ công cụ phát triển) nhưng không có program header và header table, do đó lưu ý viết mã thực thi sao cho có thể khởi tạo các phần .datta, .bss, .stack trong RAM. (ví dụ nếu dùng hợp ngữ thì gán nhãn, …).



Hình 4.32 Trình tự boot boot image chạy từ ROM

2. Chạy ở RAM sau khi mã đã copy từ ROM vào RAM

Ở kịch bản này, boot loader sẽ chuyển một chương trình nhỏ từ ROM vào RAM và kích hoạt nó chạy. Thường mã chương trình hệ thống trong ROM rất lớn mà được ghi kiểu nén để nạp vừa ROM, nên boot loader phải giải nén trước khi khởi động phần mã này và nó cần không gian nhớ ở RAM để thực hiện.

1) --- *đến 6)* giống như ở trên. Môi trường làm việc cho loader được khởi động ở RAM (3, 4, 5).

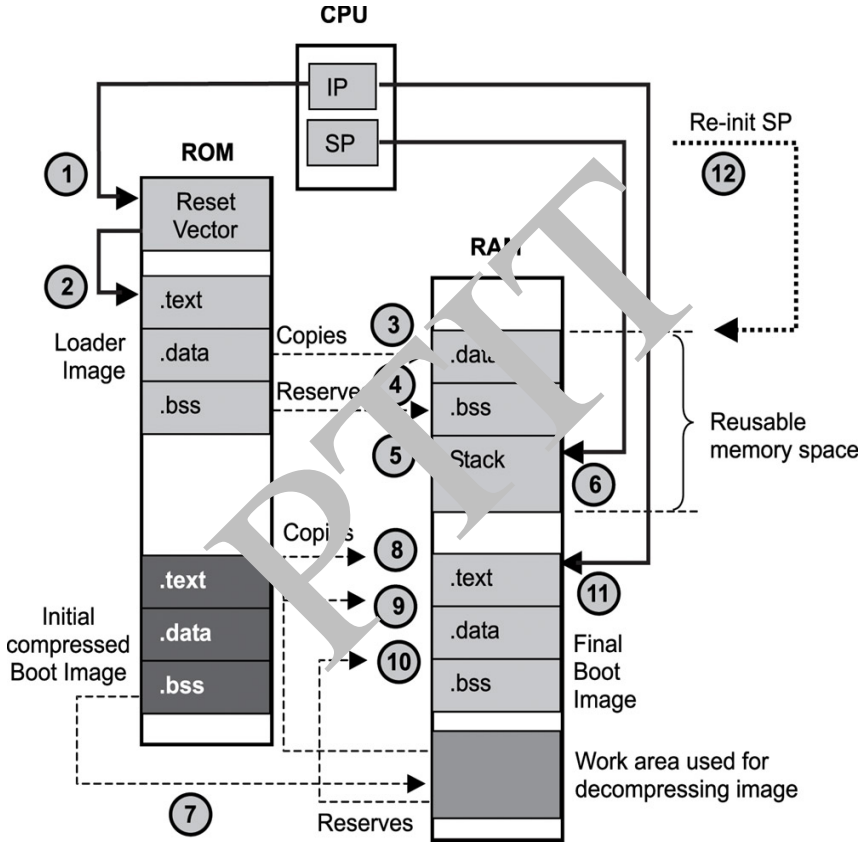
7) Loader copy phần mã nén của image vào RAM.

8) *–> 10)* Copy các phần mã đã giải nén vào các vùng làm việc tạm trong RAM (8, 9, 10) . Hoàn tất giải nén image. Image trong RAM ở hình là đoạn *Final Boot image.*

10) ...

11) Loader chuyển điều khiển cho image bằng một lệnh JMP vào .text (nạp cho IP địa chỉ này trước khi JMP tới đó).

12) Vùng RAM mà loader chiếm khi được copy từ ROM là tái sử dụng, SP được tái khởi động để trỏ vào đó và được dùng như ngăn xếp cho một chương trình mới nào đó. Vung RAM để giải nén giải phóng để sử dụng khác. Hệ thống đi vào hoạt động.



Hình 4.33 Trình tự boot thực hiện ở RAM sau khi image đã được copy từ ROM vào RAM

3. Chạy từ RAM sau khi tải xuống từ hệ phát triển (đang phát triển hệ thống)

Là kịch bản trong quá trình phát triển. Môi trường phát triển có một PC hỗ trợ. Sử dụng để phát triển các phần mềm ứng dụng cho hệ thống nhúng. Phaaffn mềm phát triển nằm trên PC và sẽ tải xuống hệ đích để chạy thử hay nạp vào hệ đích ở pha cuối cùng. Trong RON có một chương trình gọi là Debug Agent đóng vai trò kép như một loader như ở các kịch bản trên.

1) *…6)* giống như trước

7) Tải ứng dụng từ PC vào hệ đích.

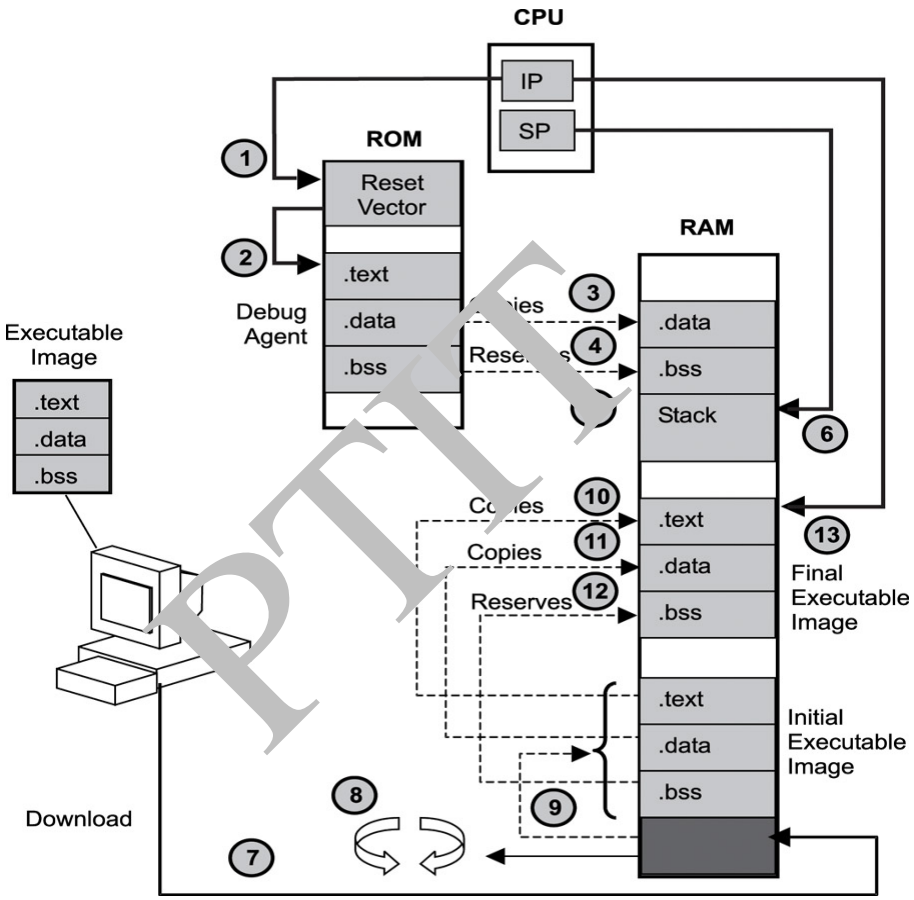
8) Kiểm tra sự hợp nhất của phần mềm tải xuống.

9) Giải nén ứng dụng nếu cần.

10) … 12) chương trình debug tái định vị ứng dụng vào vị trí trong RAM (10, 11, 12).

…

13) Debug chuyển đuều khirn cho image đã tải xuống. Hệ chạy.



Hình 4.34 Chạy image sau khi đã tải xuống hệ đích từ hệ phát triển (PC)

- **Trình tự khởi động phần mềm của hệ đích**

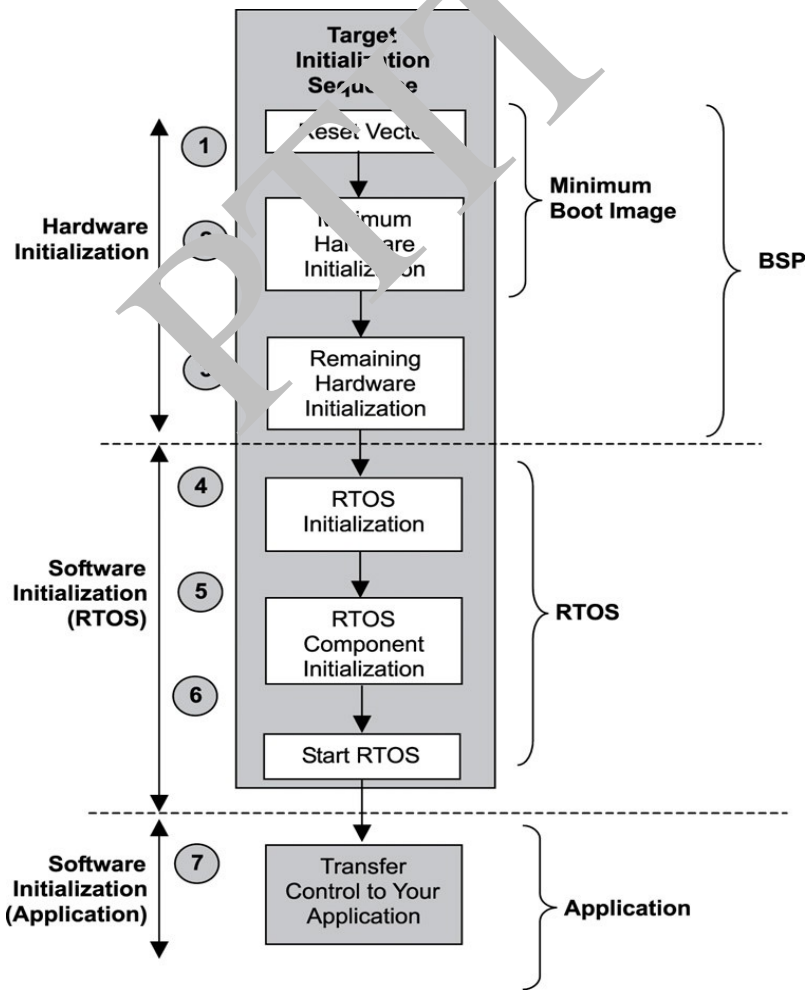
Cho dù hệ sẽ chạy theo kiểu nào thì sau khi loader trao quyền điều khiển cho code image, thì phần mềm này sẽ thực hiện các bước khởi động hệ thống. Các bước khởi động hệ thống bao gồm:

- Khởi động phần cứng,
- Khởi động RTOS,
- Kích hoạt ứng dụng nhúng.

✓ ***Khởi động phần cứng***

Sau khi thực hiện reset vector, các thao tác cơ bản là cần phải khởi động các phần cứng tối thiểu, bao gồm:

- Bắt đầu thực hiện reset vector (JMP tới địa chỉ của boot loader).
- Đặt CPU vào trạng thái các đing ban đầu, các thanh ghi được khởi động với các giá trị thích hợp, lấy số hiệu của CPU, lập tần số CPU_{Clock}.
- Cấm các ngắt và cấm cache nếu hệ có cache.
- Khởi động các Chip điều khiển bộ nhớ (MMU), điều khiển cache (MCU).
- Lập địa chỉ đầu cho RAM (thông số có ở ROM), lấy toàn bộ kích thước của RAM trên hệ, thực hiện kiểm tra RAM (ghi/đọc, ví dụ với giá trị AAhex/55hex).



Hình 4.35 Tiến trình khởi động phần mềm HTN

- Sau khi khởi động CPU và RAM, boot loader sẽ copy và giải nén code imafe vào RAM, như đã nói trên.

Hầu hết phần mã của boot loader và các bước khởi động viết bằng hợp ngữ và hướng cụ thể tới loại CPU sử dụng trên bo mạch đích. Phần code còn lại viết bằng C/C++.

- Tiếp theo là mã khởi động các phần cứng khác của code image, bao gồm: khởi động các vi mạch điều khển, nạp các chu trình thao tác, như ISR, khởi động các vector ngắt (cứng/mềm), khởi động giao thức BUS.
- Khởi động các thiết bị ngoại vi (cổng truyền thông nối tiếp/song song, mạng, …).

Các kỉ sư phát triển HTN coi bước 1) và 2) ở hình trên là bước khởi đầu, còn từ 1) đến 3) gọi là các bước cơ sở BSP hay còn gọi là khởi động phần cứng. Để viest mã cho BSP, cần nắm chắc phần cứng của hệ đích. Có thể dùng BSP của phần mềm phát triển, đánh giá bán kèm bo mạch.

Kết thúc thành công BSP phần cứng đã sẵn sàng và người phát triển có các hàm chức năng để kích hoạt phần mềm hệ thống như RTOS chẳng hạn.

✓ ***Khởi động RTOS***

Bước 4) đến bước 6) hình trên bắt đầu cho khởi động phần mềm hệ thống, RTOS. Bao gồm:

- Khởi động RTOS.
- Khởi động các dịch vụ của RTOS (các đối tượng của tác vụ, các đối tượng cờ tranh chấp-*semaphore*), các hàng đợi thông điệp, các dịch vụ định thời, các dịch vụ ngắt ISR, các dịch vụ quản trị bộ nhớ.
- Tạo các stack cho RTOS.
- Khởi động mạng (TCP/IP stack).
- Khởi động hệ thống tệp FS (file system).
- Kích hoạt RTOS với các tệp khởi động hệ thống của RTOS (/initd).

✓ ***Khởi động phần mềm ứng dụng nhúng***

Sau khi RTOS đã chạy, phần tiếp là kích hoạt các ứng dụng. Cách kịch hoạt phụ thuộc vào người phát triển ứng dụng, thường là RTOS sẽ gọi một chức năng xác định trước trong chuỗi các lệnh init hay cron viết theo shell hệ thống.

✓ ***Gở rối bằng công cụ bán kèm (On Chip Debugging – OCD)***

Các nhà cung cấp phần cứng thương mại thường có OCD đi cùng, trong đó phần mềm BDM (Background Debug mode) và JTAG (*In the 1980s, the **Joint Test Action Group** (JTAG) developed a specification for JTAG testing that was standardized in 1990 as the IEEE Std. 1149.1-1990. In 1993 a new revision to the IEEE Std. 1149.1 standard was introduced (titled 1149.1a) and it contained many clarifications, corrections, and enhancements. In 1994, a supplement containing a description of the Boundary-Scan Description Language (BSDL) was added to the standard. Since that time, this standard has been adopted by major electronics companies all over the world. Applications are found in high volume, high-end consumer products, telecommunication products, defense systems, computers, peripherals, and avionics. In fact, due to its economic advantages, some smaller companies that cannot afford expensive in-circuit testers are using JTAG.)*

http://www.corelis.com/education/JTAG_Tutorial.htm

- ***Các ví dụ***

Ví dụ Intel CPU 8085: RESET => IP=0x0000 là địa chỉ của EPROM. Khởi động chế độ nối với Console (Keyboard), sau đó nhảy về chương trình khởi động hệ. **JMP CLDST** (Khởi động nguội) ở địa chỉ 0x01F1

```
                    ;  ***** "RESET" KEY ENTRY POINT - COLD START
                    ;  ***** RST 0 ENTRY POINT
                    ;
0000 3E00           MVI     A,KMODE ; GET CONTROL CHARACTER
0002 320019         STA     CNTRL   ; SET KEYBOARD/DISPLAY MODE
0005 C3F101         JMP     CLDST   ; GO FINISH COLD START
                    CLDBK:          ; THEN JUMP BACK HERE
                    ;
                    ;  ***** RST 1 ENTRY POINT - WARM START
                    ;
```

Đoạn CLDST:

```
                    ;
                    CLDST:
01F1 3ECC           MVI     A,KBNIT ; GET CONTROL CHARACTER
01F3 320019         STA     CNTRL   ; INITIALIZE KEYBOARD/DISPLAY BLANKING
01F6 3E00           MVI     A,CSNIT ; INITIAL VALUE OF COMMAND STATUS REGISTER
01F8 D320           OUT     CSR     ; INITIALIZE CSR
01FA 32FF20         STA     USCSR   ; INITIALIZE USER CSR VALUE
01FD C30800         JMP     CLDBK   ; BACK TO MAIN PROCEDURE
                    ;
                    ;*******************************************************************
                    ;
```

Trong khi đó Intel CPU x86 trên PC hoạt động như sau: Sau khi thực hiện bật nguồn ("khởi động nguội"), hay ấn tổ hợp phím CTRL+DEL (khởi động nóng = RESET), hai

thanh ghi phân đoạn mã (code segment – CS) và con trỏ lệnh (Instruction pointer – IP), viết chung là CS:IP được nạp giá trị 0xFFFF:0000 (địa chỉ vật lí 0xFFFF0). Lệnh đầu tiên phải thực hiện để ở địa chỉ này, tại đây thực hiện một lệnh nhảy tới đoạn lệnh khởi động gọi là Kiểm tra hệ thống sau bật nguồn (POST Power On System Test) tại nhãn START. Đoạn mã này thực hiện các công việc như sau: cấm ngắt, khởi động các cờ của CPU, đọc/ghi thử các thanh ghi, kiểm tra lỗi (CRC) của EPROM, khởi động các vi mạch điều khiển của bo mạch chính v.v. Tiếp theo khởi động lại và cho phép các ngắt không che hoạt động, thực hiện INT 19 để chạy chương trình mồi (*bootstrap loader*) nạp mã khởi động hệ điều hành (*boot-record*) từ đĩa cứng xuống bộ nhớ, sau đó nhảy tới địa chỉ của boot-record. Chương trình boot-record tiếp tục nạp hệ điều hành xuống phần bộ nhớ hệ thống. Lúc này chế độ hoạt động là chế độ thực (*real mode*) của CPU. Sau khi HĐH đã nạp hoàn tất, chế độ thực chuyển sang chế độ ảo hay còn gọi là chế độ có bảo vệ (protected mode), do HĐH kiểm soát. Lúc này HĐH cho phép người dùng sử dụng máy tính.

(Ví dụ của Intel 80286: A_{23} - $A_{0:\ Cho}$ cho 16 MB địa chỉ vật lí, 1 GB địa chỉ ảo.

Initialization and Processor Reset Processor initialization or start up is accomplished by driving the RESET input pin HIGH. RESET forces the M80C286 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active. After RESET becomes inactive and an internal processing interval elapses, the M80C286 begins execution in real address mode with the instruction at physical location FFFFF0(H). RESET also sets some registers to predefined values as shown in Table 6.

HOLD must not be active during the time from the leading edge of RESET to 34 CLKs after the trailing edge of RESET.

Table 6. M80C286 Initial Register State after RESET

Flag word	0002(H)
Machine Status Word	FFF0(H)
Instruction pointer	FFF0(H)
Code segment	F000(H)
Data segment	0000(H)
Extra segment	0000(H)
Stack segment	0000(H)

M8086 REAL ADDRESS MODE

The M80C286 executes a fully upward-compatible superset of the M8086 instruction set in real address mode. In real address mode the M80C286 is object code compatible with M8086 and M8088 software. The real address mode architecture (registers and

addressing modes) is exactly as described in the M80C286 Base Architecture section of this Functional Description.

Memory Size

Physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A_0 through A_{19} and BHE. A_{20} through A_{23} should be ignored.

Memory Addressing

In real address mode physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A0 through A19 and BHE. Address bits A20±A23 may not always be zero in real mode. A20±A23 should not be used by the system while the M80C286 is operating in Real Mode.

The selector portion of a pointer is interpreted as the upper 16 bits of a 20-bit segment address. The lower four bits of the 20-bit segment address are always zero. Segment addresses, therefore, begin on multiples of 16 bytes. See Figure 7 for a graphic representation of address information.

All segments in real address mode are 64K bytes in size and may be read, written, or executed. An exception or interrupt can occur if data operands or instructions attempt to wrap around the end of a segment (e.g. a word with its low order byte at offset FFFF(H) and its high order byte at offset 0000(H)). If, in real address mode, the information contained in a segment does not use the full 64K bytes, the unused end of the segment may be overlayed by another segment to reduce physical memory requirements.

Các segment: *1st:* *00000-0FFFF => 0000:0000-0000:0FFF*

2nd *10000-1FFFF => 1000:0000-1000:1FFF*

3th *20000-2FFFF => 2000:0000-2000:2FFF*

4th *30000-3FFFF*

.

.

.

16th *F0000-FFFF =>F000:0000-F000:FFFF*

Hiện tượng chồng chéo trong mô hình này:

Ví dụ: Cho địa chỉ 1256A, tìm các địa chỉ trùng với nó ở các segment sau đây: 1256 và 1240:

*Địa chỉ vật lí = (segment * 16) + offset*

Từ đó:

*Segnent * 16 = Địa chỉ vật lí – offset*

1256A=12560 + X, và 1256A=1240 + Y

Vậy:

X=1256A-12560=A, và Y=1256A-1240=16A

Cho nên ta có:

1256A=1256:000A = 1240:016A !!! Một địa chỉ vật lí tồn tài ở 2 segnent !!!

Khi có địa chỉ vật lí và có offset, có thể tính địa số của segment.

Ví dụ: cho địa chỉ vật lí là 80FD2, với offset=BFD2. Tìm segment ?

*Segment*16=80FD2-BFD2=7500 =>đi chỉ tại 7500:BFD2*

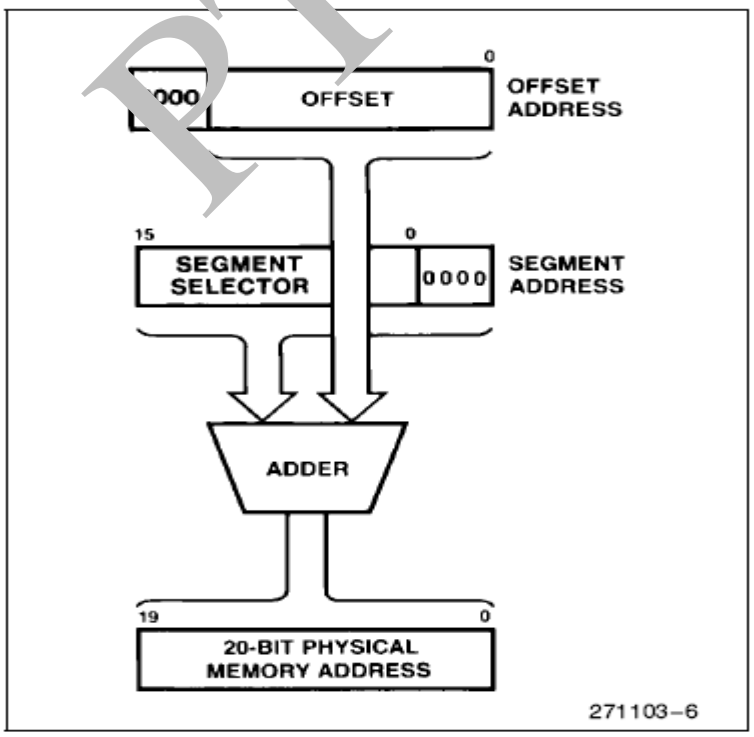
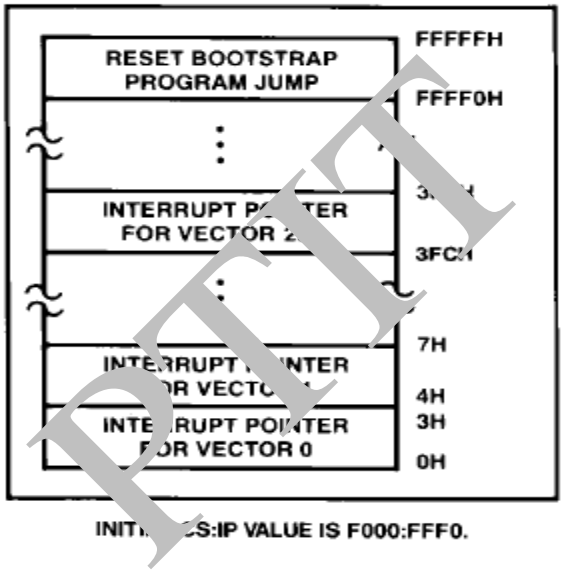


Figure 7. M8086 Real Address Mode Address Calculation

Reserved Memory Locations

The M80C286 reserves two fixed areas of memory in real address mode (see Figure 8); system initialization initialization area and interrupt table area. Locations from addresses FFFF0(H) through FFFFF(H) are reserved for system initialization. Initial execution begins at location FFFF0(H). Locations 00000(H) through 003FF(H) are reserved for interrupt vectors.



Real address Mode Initially Reserved Locations

Sau RESET ***CS:IP=F000:FFF0***

Do đó địa chỉ thật sẽ là ***(CS*16) + IP = F0000 + FFF0 = FFFF0***

```
-
-DF000:FFF0
F000:FFF0  EA 5B E0 00 F0 30 34 2F-32 39 2F 30 35 00 FC 70   .[...04/29/05..p
-
-
-
-
-
```

)

Đoạn mã hợp ngữ sau mô tả qui trình trên:

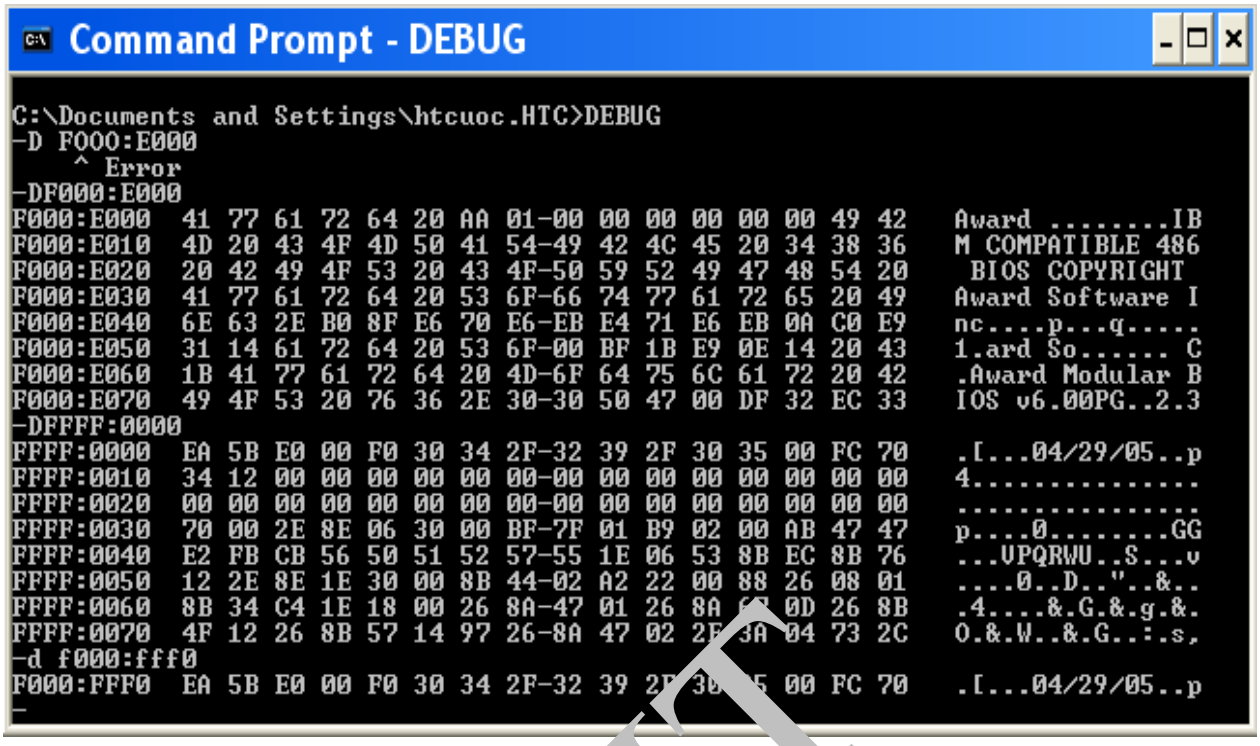
```
**********************************
*IBM PC AT System ROM BIOS Resident *
**********************************
        CODE SEGMENT AT 0F000H
                DB      57344                   ; fill lonest       56K

                DB      'Award ……..IBM  COMPATIBLE 486 BIOS COPYRIGHT Award
                        Software Inc…'
                .
                .
                .
                .
F000:E010       STGTST          PR  NEAR
F000:E010       MOV CX,4000H                    ;SETUP CNT TO TESTT 16K BLK
                .
                .
                .
                STGTST ENDP

  ************************************************
  * NOW CPU TEST: FLAGS, REGS, CONDITIONALJMPS*
  ************************************************

        ASSUME CS:CODE,DS: NOTHING ,ES: NOTHING ,SS: NOTHING
        ORG 0E05BH
; System start:
```

```
F000:E05B       RESET       LABEL       FAR
F000:E05B       START:
F000:E05B       FA          CLI                     ;DISABLE INTERRUPTS
F000:E05C       B4D5        MOV AH, 0D5H            ;SET flags SF, CF, ZF,F on
……..
……..
(Tiếp tục mã khởi động)
CODE ENDS
;
```

Bật nguồn khởi động bắt đầu ở đây:

```
;----------------------------------------
; POWER ON RESET VECTOR
;----------------------------------------
VECTOR SEGMENT AT 0FFFFH
FFFF:0000       EA5BE000F0          JMP     RESET ; Nhảy tới nhãn RESET

FFFF:0005       30342F32392F30      DB      '04/29/05'      ;RELEASE MARKER
VECTOR ENDS
```

Có thể quan sát các địa chỉ bằng lệnh debug:



Thông thường *start code* thực hiện một số thao tác như kiểm tra CRC của ROM, test RAM, tái định vị mã ở ROM vào FLASH hay RAM (*shadown moad*) để thực thi lệnh nhanh hơn, thực hiện khởi động các thanh ghi, khởi động các vi mạch điều khiển tích hợp trên bo với các giá trị đã lập trình cho vi mạch ở ROM …. Sau đó thực hiện "nạp" hệ điều hành (trên đĩa cứng) hay chuyển điều khiển tới hệ điều hành ở FLASH. Hệ thống đi vào hoạt động.

Ví dụ *star code:*

main()

{

//Check hardware (ROM, RAM);

……

//Khởi động các device drivers: các Chip on board;

…

//Khởi động các biến môi trường;

….

//khởi động các CSDL hệ thống;

.....

//Relocate RAM address, define memory map, load OS. ...;

...

//JMP to 1st instruction of OS

JMP ……..

//system run ...

}

4.2.7 Ví dụ phát triển HTN

Mục này đưa ra một số đầu bài thực hành phát triển HTN, sẽ có tài liệu riêng cho mỗi bài tập. Các HTN bao gồm:

1) Thiết kế bo mạch với các lựa chọn:
   - CPU Intel 8085/8086,
   - ROM, RAM
   - Các vi mạch ngoại vi : 74257, 74244, 74245, 7474, 8253, 8255, 8237, UART 8250/16450/16550
2) http://www.beyondlogic.org/serial/serial.htm
3) Phát triển HTN với micro controller Intel 8051 với phần mềm KEIL Soft.
4) HTN với PIC:
   http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2123&param=en022497
   Hiện nay trên thị trường có bán mạch với PIC 16F877 với phần mềm Hệ phát triển với giá 900.000,00 VNĐ, phù hợp cho thực hành viết chương trình ứng dụng. Sau khi thành thạo, có thể thiết kế phần cứng với các vi mạch rời và phát triển ứng dụng cụ thể từ đơn giản đến phức tạp.

5) HTN với Linux, bo Tri-M MZ104 (Họ PC 104):
   http://www.tri-m.com/products/engineering/mz104.html

6) Một số hệ điều hành trên HTN:
   QNX 4 RTOS,

   Windows CE and embedded Linux.

   Palm OS
   Windows CE
   MS-DOS or DOS Clones
   Linux, including RTLinux and MontaVista Linux and Unison OS
   QNX .

4.3 KẾT CHƯƠNG

Chương 4 nêu ra các bước khi thiết kế một HTN nói chung, về sự liên kết với thị trường ứng dụng, về các qui tắc và nền trí thức cần có. Thiết kế HTN là một bài tập rất tổng hợp với bất kì ai khi nói đến thiết kế máy tính, bởi khối lượng kiến thức rất rộng, từ kĩ thuật điện tử, khoa học máy tính, công nghệ bán dẫn, ứng dụng máy tính trong công nghiệp, trong gia đình … Thiết kế HTN không chỉ là *thiết kế phần cứng* mà còn sáng tạo các *giải pháp phần mềm phù hợp để giải quyết cho loại ứng dụng nhúng*. Do vậy rất quan trọng là cần *phân hoạch chi tiết và phối hợp khéo léo* khi thiết kế. Sau khi phần mềm đã phát triển và chạy đúng như thiết kế, công đoạn tiếp theo là "nạp" phần mềm vào bo mạch. Qui trình này đòi hỏi phải hiểu biết cách một CPU khởi động và chiến lược qui hoạch sử dụng bộ nhớ của HTN thiết kế. Các công cụ thiết kế và phát triển phần mềm là không thể thiếu được, trong đó bao gồm hệ phát triển, công cụ mô phỏng (IDE, ICE), phần mềm hỗ trợ cho bo mạch (*board support package* - BSP). Thông thường các công cụ này có phần tổng quát chung, nhưng có phần cụ thể cho CPU ứng dụng cụ thể và do nhà phân phối sản phẩm bán kèm.

Để thực hành, có thể chọn bất kì loại HTN nào từ bo mạch controller trên thị trường hay bắt đầu từ A đến Z với CPU rời, với PIC hay PSoC.

4.4 CÂU HỎI CUỐI CHƯƠNG

1) Phát thảo và giải trình các pha thiết kế khi thiết kế HTN, pha nào được cho là khó và quan trọng nhất, vì sao ?
2) Nêu các bước cơ bản khi thiết kế kiến trúc một HTN ?
3) Mô hình "4+1" là gì ? Tại sao mô hình này lại có ích cho thiết kế HTN ? Các cấu trúc nào có trong mô hình "4+1" ?
4) Nêu kỹ thuật "nạp" phần mềm vào bo mạch HTN đang thiết kế ?
5) Ý tưởng về "thử để cho qua" và "thử để gây sự cố " khi phát triển và thử nghiệm HTN là thế nào ?
6) *Boot code* là gì ? và làm chức năng gì ?
7) Khi HTN thiết kế đã đưa vào dây chuyền sản xuất, công việc của đội thiết kế và phát triển đã xong ?

TÀI LIỆU THAM KHẢO

[1] Arnold S. Berger: *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques, ISBN: 1578200733*

[2] PETER MARWEDEL : *Embedded System Design*

[3] Stephen Edwards, Luciano Lavagno, Edward A. Lee, and Alberto Sangiovanni- Vincentelli: *Design of Embedded Systems: Formal Models, Validation, and Synthesis*

[4] Steave Heath: *Embedded Systems Design*

[5] Tammy Noergaard: *Embedded System Architecture – A Comprehenvive Guide for Engneers and Programmers, 2005*

[6] G. R. Wilson: *Embedded Systems and Computer Architecture*

[7] Jean J. Labrosse: *Embedded_Systems_Building_Blocks (Soft blocks in C)*

[8] Miles J. Murdocca: *PRINCIPLES OF COMPUTER ARCHITECTURE*

[10] Qing Li with Caroline Yao: *Real-Time Concepts for Embedded Systems,2003*

[11] PHILIP KOOPMAN, HOWIE CHOSET, RAJEEV GANDHI, BRUCE KROGH,

DIANA MARCULESCU, PRIYA NARASIMHAN, JOANN M. PAUL,

RAGUNATHAN RAJKUMAR, DANIEL SIEWIOREK, ASIM SMAILAGIC,

PETER STEENKISTE, DONALD E. THOMAS, and CHENXI WANG

Carnegie Mellon University: *Undergraduate Embedded SystemEducation at Carnegie Mellon*

[12] Wayne H. Wolf: *Hardware-Software Co-Design of Embedded Systems*

[13] Rajesh K. Gupta Information and Computer Science University of California, Irvine: *Introduction to Embedded Systems*

[14] Lưu Hồng Việt: *Hệ thống điều khiển nhúng (Embedded Control Systems)*

[15] Huỳnh Thúc Cước: *Bên trong Hệ điều hành Unix/Linux, ĐHDL Thăng Long*

[16] Trần Quang Vinh: *Nguyên lí phần cứng và Kĩ thuật ghép nối máy vi tính*

[17] Huỳnh Thúc Cước: *Hợp ngữ cho máy vi tính, Phòng Kĩ thuật số, Viện Tin học, VKHVN, 1983*

[18] Huỳnh Thúc Cước , Nguyễn văn Tam: *Ghép nối thông minh đĩa cứng 29 MB vào máy vi tính VT 83, Phòng Kĩ thuật số, Viện Tin học, VKHVN, 1983*

[19] Nguyễn Trung Đồng: *Kĩ thuật vi xử lý*

[20] IBM Coporation: *The IBM Personal Computer Technical Reference manual, 1983*

[21] Falk Salewski, Stefan Kowalewski, Embedded Software Laboratory RWTH Aachen University, Germany: *Hardware Platform Design Decisions in Embedded*

[22] Đặng Thành Phu, VCNTT: NGÔN NGỮ ASSEMBLY VÀ CÁCH LẬP TRÌNH

[23] Karim Yaghmour: Building Embedded Linux Systems, O'Reilly, April 2003

[24] Steve Heath:Embedded Systems Design, Second edition 2003

[25] Wiley IEEE Press, Embedded Signal Processing with the Micro Signal Architecture Feb 2007 eBook-LinG

PHỤ LỤC Các ví dụ

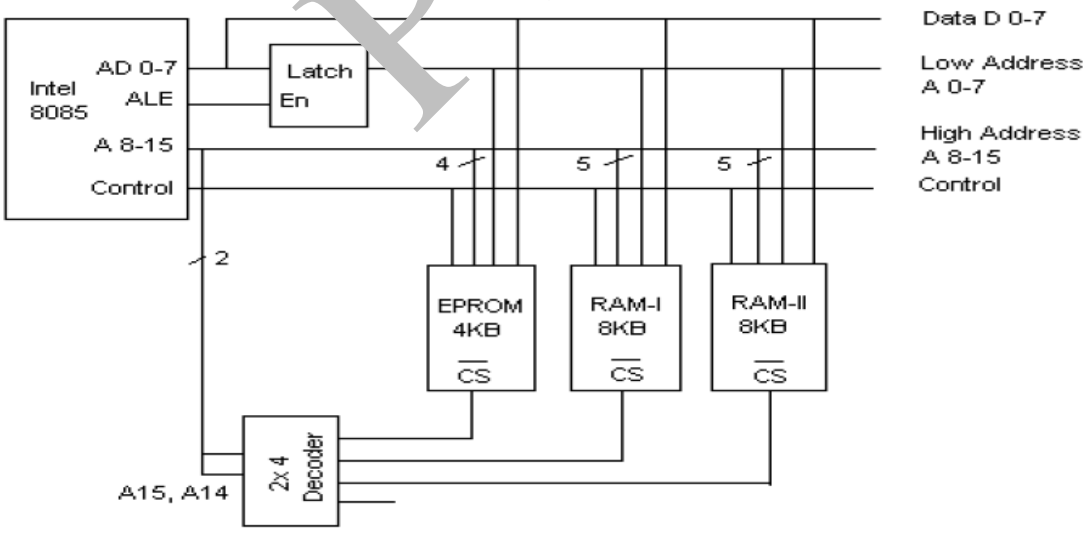
Đáp án bài tập thiết kế:

Chương 2:

Bài 1:

5. EPROM - 4 KB (Address lines required is 12 – A0 to A11)
6. RAM-I - 8 KB (Address lines required is 13 – A0 to A12)
7. RAM-II - 8 KB (Address lines required is 13 – A0 to A12)
8. Ánh xạ địa chỉ vào các Chip nhớ như sau:

ICs	*Binary Address*																*Hex Address*
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
EPROM 4 KB	0	0	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0000
	0	0	x	x	0	0	0	0	0	0	0	0	0	0	0	1	0001
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	x	x	1	1	1	1	1	1	1	1	1	1	1	1	0FFF
RAM-I 8 KB	0	1	x	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
	0	1	x	0	0	0	0	0	0	0	0	0	0	0	0	1	4001
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	x	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
RAM-II 8 KB	1	0	x	0	0	0	0	0	0	0	0	0	0	0	0	0	8000
	1	0	x	0	0	0	0	0	0	0	0	0	0	0	0	1	8001
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	0	x	1	1	1	1	1	1	1	1	1	1	1	1	1	9FFF

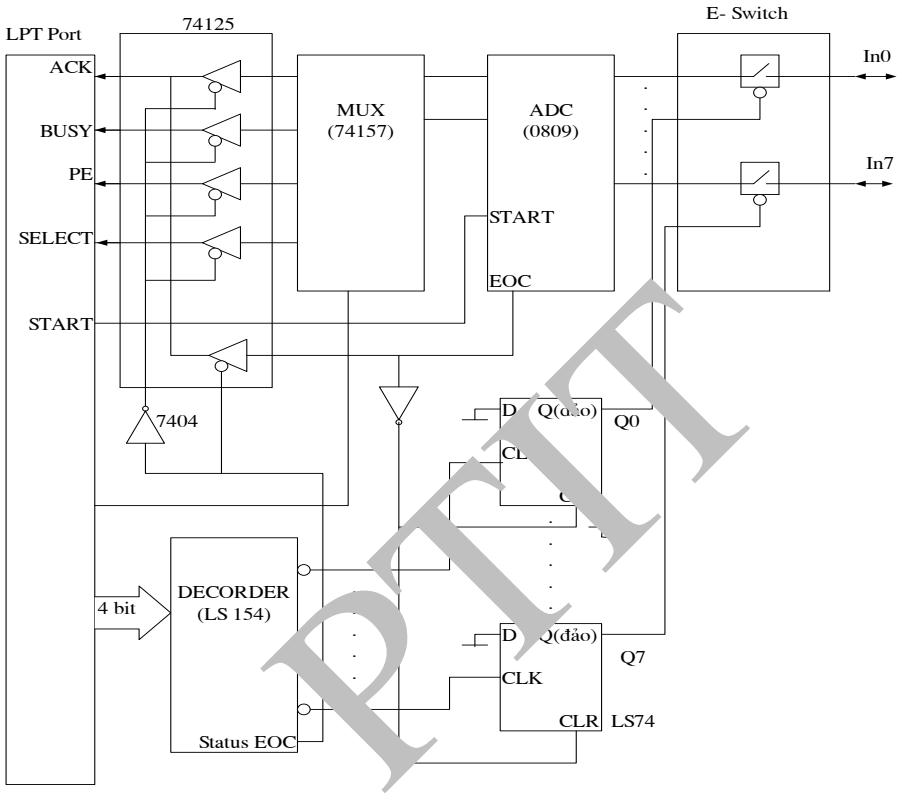


Hình 2.77 Bài tập thiết kế ROM

Bài 2:

Bài giải:

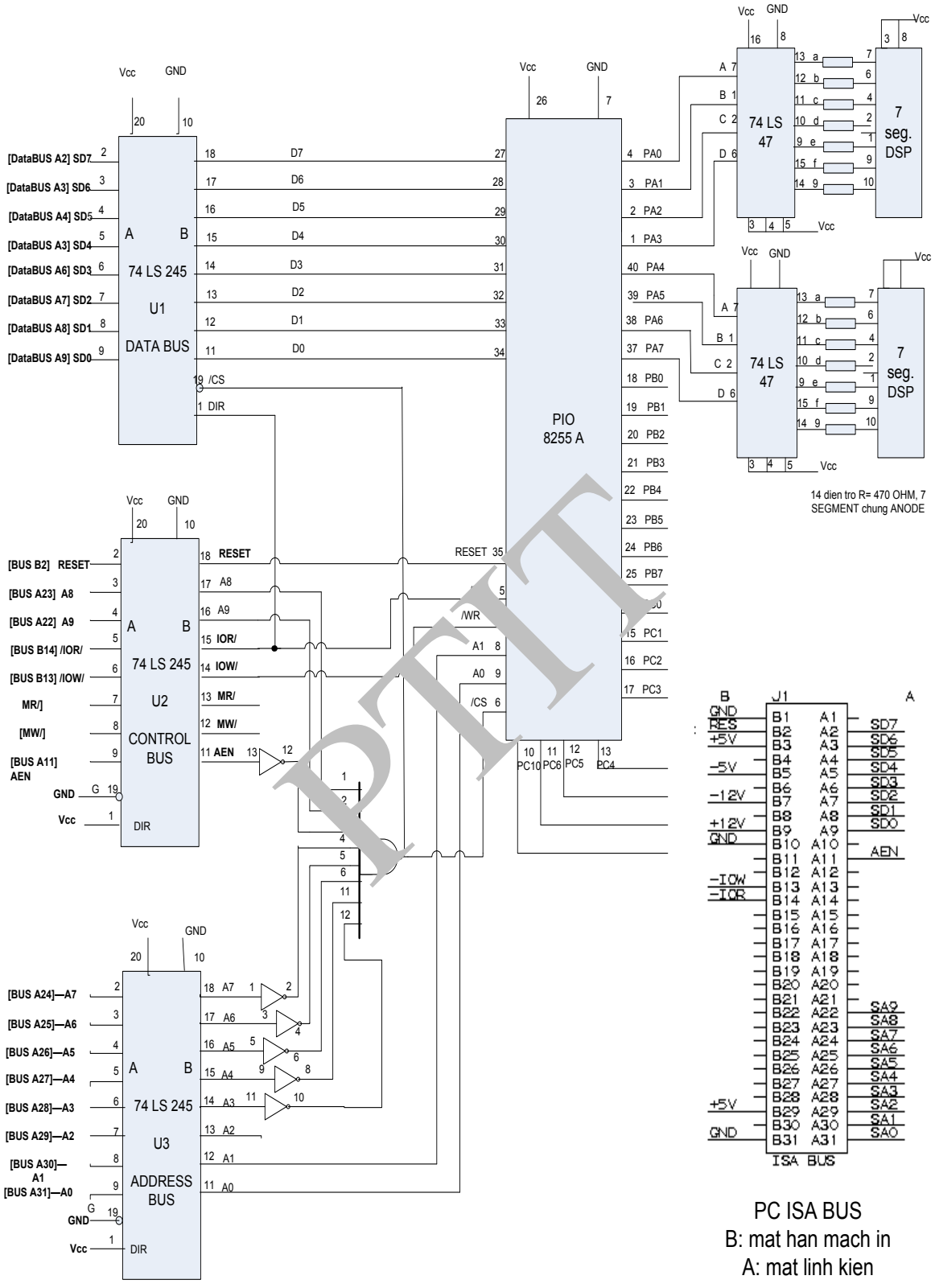
- Tìm hiểu nguyên tắc hoạt động của cổng máy in của PC, cách sử dụng cổng máy in để dọc dữ liệu vào PC: dùng các chân đọc thông tin giao tiếp để đọc dữ liệu vào, mỗi lần 4 bit.
- Nguyên lí làm việc của ADC 0809, cách ghép nối.
- Từ đó lên thiết kế nguyên lí.



Hình 2.79 Bài tập thiết kế ghép nối AD với BUS hệ thống máy tính PC

- Phát thảo lưu đồ phần mềm diều khiển gép nối:
    2. Chọn kênh đo
    3. khỏi động chu kì đo của ADC
    4. Đọc trạng thái sau mỗi biến đổi (STATUS do EOC phát ra)
    5. Đọc và lưu dữ liệu.

Bài 3:



Bài giải :

-Mô tả nguyên lí làm việc

-Chương trình điều khiển ghép nối:

```
/*
** 8255.c
**
** Minh hoa PIO 8255 voi mode 0 (tat ca out_ports on 8255 Ports):
** A, B and C.
**
**   8255 is first setup with control word 0x80
**   Mode set flag active - bit 7 = 1
**   Mode selection 0 - bits 6 5 = 0 0
**   Port A output - bit 4 = 0
**   Port C (upper) output - bit 3 = 0
**   Mode selection 0 - bit 2 = 0
**   Port B output - bit 1 = 0
**   Port C (lower) output - bit 0 = 0
**
** Dữ liệu ra sử dụng gọi hàm out_data (port, data) trong đó:
**   Port A - 0
**   Port B - 1
**   Port C - 2
**
*/

#include <stdio.h>
#include <dos.h>
/* Định nghĩa các biến*/
#define DATA 0x0300  /* Cho ghép nối lập biến DATA=base addr*/
#define STATUS DATA+1
#define CONTROL DATA+3 /*CONTROL=DATA+3*/
void reset(void);
void write_clock(int a1a0);
void set_control_word(void);
void out_data(int port, int data);


void main(void)
{
  int port;
```

```
int data;
/*bat dau chuong trinh*/
reset();
set_control_word();
/* Bay gio da co the cho 8255 chay */
/* Thu dua ra cac so Hex tren 3 cong A,B,C */
        printf("An <ENTER> DE KIEM TRA CAC GIA TRI TU a den f \n");
       /* for(port=0;port<3;port++)*/
        /*{*/
        getchar();
        port=0; /*Tam thoi voi port A*/
        printf("Dua ra aa\n");
        out_data(port, 0xaa);
        getchar();
        printf("Dua ra bb\n");
        out_data(port, 0xbb);
        getchar();
        printf("Dua ra cc\n");
        out_data(port, 0xcc);
        getchar();
        printf("Dua ra dd\n");
        out_data(port, 0xdd);
        getchar();
        printf("Dua ra ee\n");
        out_data(port, 0xee);
        getchar();
        printf("Dua ra ff\n");
        out_data(port, 0xff);
       /* }*/
       /*Nhan bat ki so nao tu 00-99*/
       printf("\n Go vao bat ki so nao tu 00 den 99 va ENTER \n");
       while(1)
       {
        scanf("%x",&data);   /*nhan luon so kieu hexa de dua ra cong*/
        for (port=0;port<3 ;port++ )
                {
                        out_data(port, data);
                }
```

```
        }/*while*/
}
/*Cac ham ho tro*/
void set_control_word(void)
{
  /*int a1a0 = 0x03;*/
  outportb(CONTROL, 0x80); /* out control word = 0x80 */

  /* bring a1 a0 to 1 1, WR to 1 */
  /*outportb(CONTROL,       */
  /*    ( ((0x00) | (a1a0 << 1) | 0x01) ^ 0x0b)  & 0x0f); */

  /* now wink WR low and high */
  /*write_clock(a1a0);       */
}

void out_data(int port, int data)
{
  /*int a1a0 = port;*/
  outportb(DATA, data); /* put data on data leads */

  /* set a1 and a0 to 0 0, 0 1 or 1 0, WR to 1 */
  /*outportb(CONTROL,( ((0x00) | (a1a0 << 1) | 0x01) ^ 0x0b) & 0x0f);*/

  /* now wink WR low and high */
  /*   write_clock(a1a0); */
}

void write_clock(int a1a0)
{
  /* bring WR low */
  outportb(CONTROL,
        (((0x00) | (a1a0<<1) & (~0x01) ) ^ 0x0b) & 0x0f);

  /* bring WR back high */
  outportb(CONTROL,
        (( (0x00) | (a1a0<<1) | (0x01)) ^ 0x0b) & 0x0f);
}
```

```
void reset(void)
{
  /* bring reset high */
  outportb(CONTROL, 0x08^0x0b);

}
;
```

Making an LM35 temperature recorder with PIC 12F675.

http://www.best-microcontroller-projects.com/12F675-tutorial-mikroc-1-7-source.html

This page shows you how you can make an LM35 an temperature recorder by using the 12F675 PIC microcontroller as the controller and data store.

It generates serial output so that you can view the results on a PC and it also calculates the temperature reading in Fahrenheit sending both to the serial port at half second intervals.

The project uses the code from the previous tutorials to report the temperature to the PC using the serial port so the serial RS232 data format is generated in software.

Jump to circuit.
Jump to Solderless breadboard.
Jump to Circuit Diagram.
Jump to Software.

LM35DZ

The LM35 is a precision temperature sensor. It is guaranteed accurate to ±¼°C at 25°C (At different temperatures it is less accurate! but it is never more than 2°C inaccurate and it probably is not this inaccurate anyway it's just the manufacturers maximum limits that may apply).

Typically is stays accurate to within ±¾°C over its temperature range so this is a good general purpose sensor and it's easy to use.

It generates a linear output voltage using a centigrade scale - generating 10mV of output voltage for every degree centigrade change and there are several versions for operation over different temperature ranges:

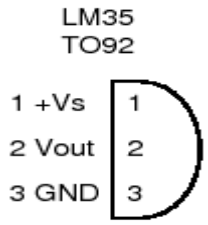
LM35	-55°C to 150°C
LM35C	-40°C to 110°C

LM35D	0°C to 100°C

Note: The project code calculates the temperature in Fahrenheit and generates both Centigrade and Fahrenheit outputs to the serial port.

Temperature recorder : LM35 pinout

Temperature recorder : pinout for the LM35DZ (from the top).



Temperature recorder Circuit

The LM35 is connected to analogue input AN0 which is also the data input line for programming the 12F675 using ICSP so you need a way of connecting the sensor and the programming input at the same time with the programming input overriding the sensor output (and not damaging the sensor!).

This is done here by using 1k resistor that reduces the current flowing back into the sensor and at the same time is not too large (so that the ADC can easily convert the sensor output value - the impedance must be equal to or smaller than 10k Ohm from the sensor)

The voltage reference for the circuit is taken from pin 6 using a resistor divider giving a 2.5V reference. This is simply done to increase the resolution of the ADC as for the LM35 only 0-1V is generated so you loose ADC range when using a 5V reference. You could use a higher reference value but this value gives reasonable results.

Alternatively you could use an amplifier to scale the LM35 output up which would make the ADC less sensitive to noise but for this project it is simpler not to do so.

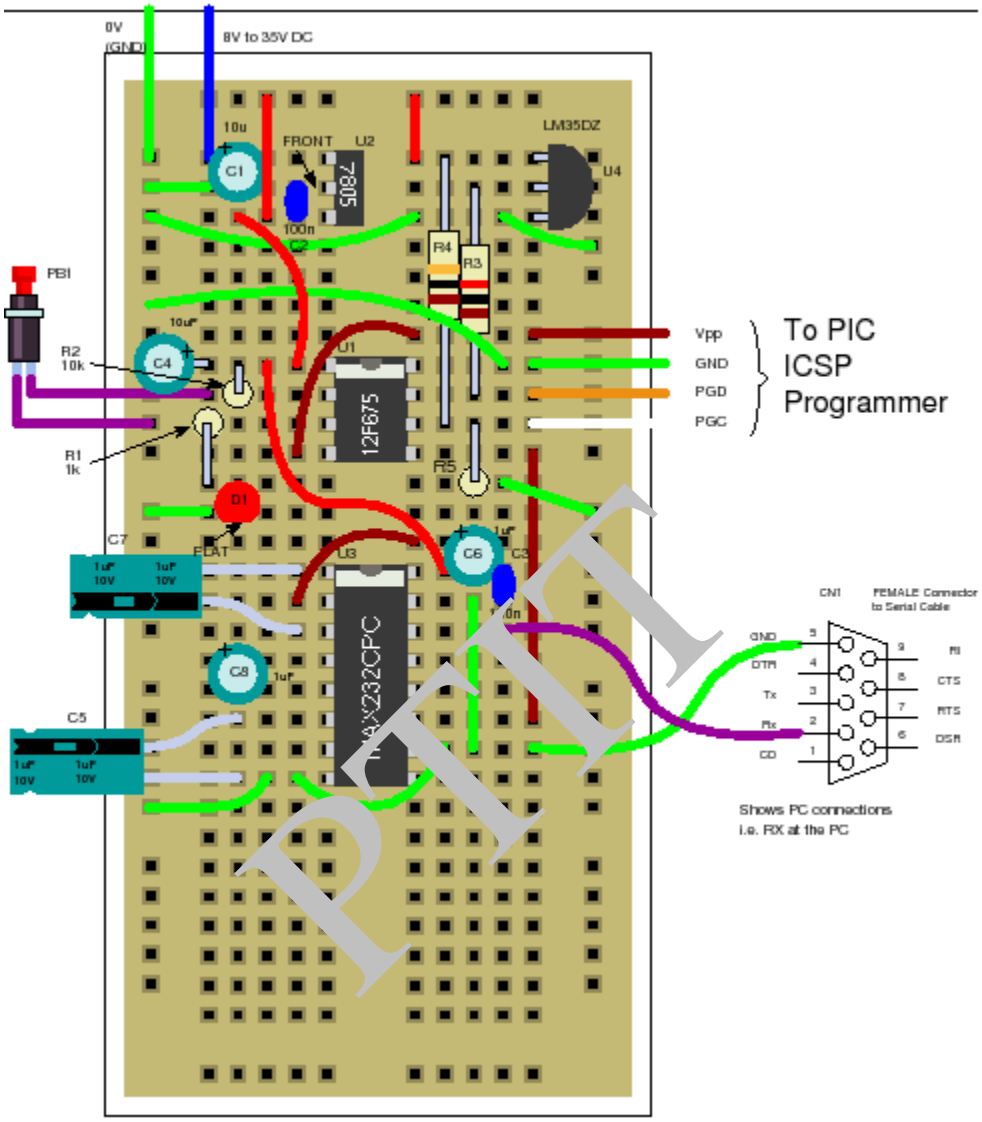
Note: The large decoupling capacitor on the supply input of the 12F675. This reduces noise overall and gives a more consistent reading. However using a plug block and ADC is not a very good idea as there is no ground plane and no control over current paths which you would be able control in a pcb.

In a commercial system the internal ADC is often not used at all as it is essential to separate the noise introduced to the ADC using separate grounds and shielding - some designs encase the ADC in a custom metal shield and along with a ground plane connecting to the shield gives the best possible result.

To overcome noise problems on the ADC the software averages the input readings so you get a better result.

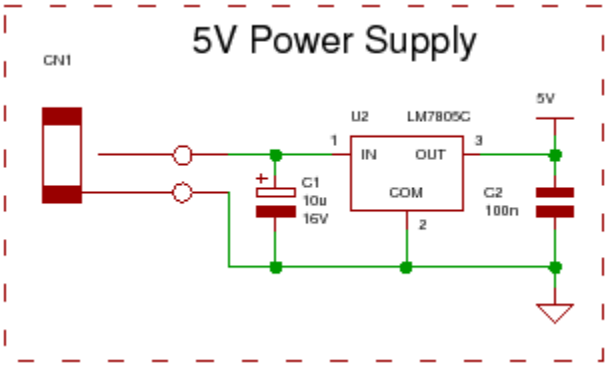
Solderless breadboard

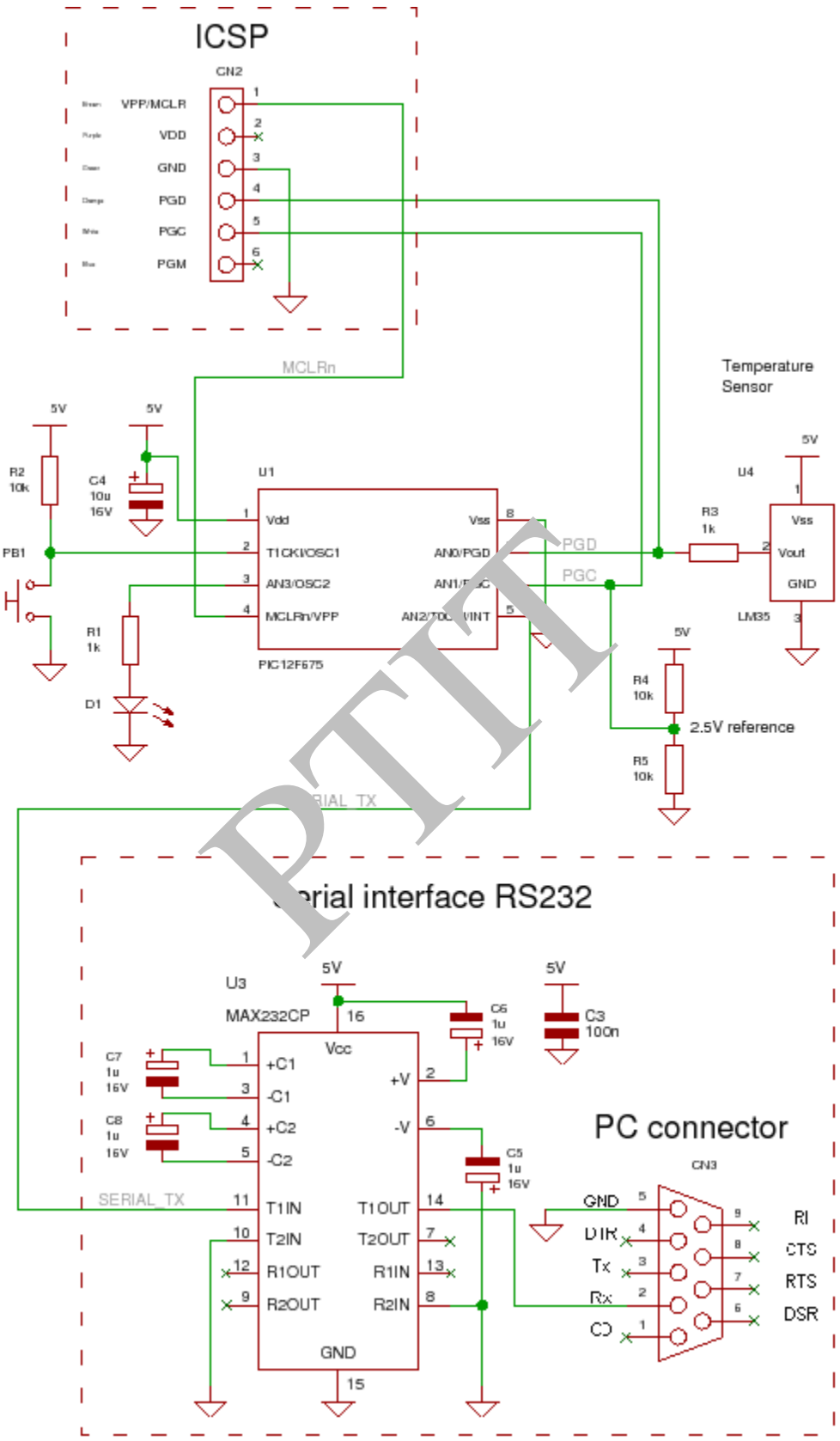
Add the components (at top right to) the temperature recorder - wires and R3,R4,R5 and the LM35 temperature sensor (U4) and the decoupling capacitor C4.



Learn about the tool used for creating this diagram.

LM35 Temperature Recorder Circuit diagram


5V Power Supply
CN1
U2
LM7805C
IN
OUT
COM
1
2
3
C1
10u
16V
C2
100n
5V


ICSP
CN2
VPP/MCLR
VDD
GND
PGD
PGC
PGM
MCLRn
Temperature Sensor
5V
R2 10k
C4 10u 16V
U1
Vdd
T1CKI/OSC1
AN3/OSC2
MCLRn/VPP
Vss
AN0/PGD
AN1/PGC
AN2/T0CKI/INT
PIC12F675
PB1
R1 1k
D1
PGD
PGC
U4
R3 1k
Vss
Vout
GND
LM35
R4 10k
2.5V reference
R5 10k
SERIAL_TX
erial interface RS232
U3
MAX232CP
Vcc
+C1
-C1
+C2
-C2
+V
-V
T1IN
T2IN
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
GND
C6 1u 16V
C3 100n
C7 1u 16V
C8 1u 16V
C5 1u 16V
PC connector
CN3
GND
DTR
Tx
Rx
CD
RI
CTS
RTS
DSR

Learn about the tool used for creating this diagram.

Temperature recorder measurement accuracy

The analogue reference for the ADC is taken from the power supply via a resistive divider to the 12F675 input pin 6 and for the 7805 its accuracy is specified as ±5% so the accuracy of the ADC is only 5% due to the reference -the divider also introduces a 1% error giving a 6% error overall.

Note: Since the 7805 is only accurate to ±5% the accuracy of the temperature reading will be accurate to ±5% (plus errors in the ADC and temperature sensor itself and any noise introduced the the analogue input and the reference). However the reference source gives you the biggest error - the overriding accuracy - if you used a more accurate voltage supply then the ADC accuracy would become more important as well as the temperature sensor accuracy etc.

Temperature recorder Software

The software uses the Soft USART (transmit only) described in the previous tutorial and uses the built in MikroC routines to get the data from analogue input pin AN0.

```
// Temperature recorder analogue input

val = ADC_Read(0);

// more code adds up 10 readings of ADC

val = ((val/MAX_AVG)*122)/50;

val = ((val*18)/10)+320;
```

Software operation

The most interesting parts of the software are shown above. The variable val is an unsigned int so the maximum value it can store is 65535

The reference in use is 2.5V so for the 10bit ADC each ADC bit is worth
2.5/1023 = 2.44mV

If you work out values generated for a maximum temperature of 100°C using the scale factor 2.44mV (or 244/100)

100 * 10mV = 1.0V

1.0V/2.44mV = 410

410 * 244 = 100,040 **which will not fit into an unsigned int.**

So this scale factor does not work for all input value

By using a little maths it can be made to fit -you need to reduce the top number to fit. e.g.

410 * 122 = 50,020 **which does fit.**

Dividing by 50 gets back to the correct scale factor of 244.

So the scale 122/50 works for all input values.

This is an example of avoiding the use of floating point variables which take up too much resources. You can still make the system work but you have to be careful when using fixed types and you have to check all input values and outputs to make sure they fit.

Averaging

Averaging would be better done in the PC as it has more resources - the same goes for calculating and displaying the temperature in Fahrenheit but this gives a demonstration of what you can do.

Note: The RAM is used up since a bug in MikroC 5.0.0.3 puts strings int RAM - in future versions this will be corrected.

Typical output from the temperature recorder

96 RAW 234 C 741 F

The left most value is the RAW ADC value, the next is the temperature sensor output in degrees centigrade and the next is the temperature sensor output in degrees Fahrenheit.

Note: You have to put in the decimal point so the above readings are:

234 C	23.4°C
741 F	74.1°F

Bài đọc thêm:

E380: Design of Embedded Systems Exercises

Krzysztof Kuchcinski
Department of Computer Science
Lund Institute of Technology

The questions and exercise included in this material represent typical questions which can be asked to the contents of the lectures given through the course *Design of Embedded Systems* (E380). The structure of chapters follows directly the structure of the lectures.

Most of the questions are typical for the course and can be asked during the examinations, however, it is not the intention to supply the examination questions. The purpose is to give the guidance for students for preparation to examinations.

Introduction

1. Give a short definition of embedded systems and discuss main features of such systems. Illustrate your discussion with examples of embedded systems.
2. Discuss differences and similarities between embedded systems and general purpose workstations, desktop computers and portable computers.
3. What are basic characteristics of embedded systems? Discuss both inherent features of such systems as well as specific design process challenges.
4. Discuss and give a motivation why implementation of embedded systems using a single processor running a software implementation is not usually possible. What are advantages and disadvantages of such a solution?
5. Explain the term ``design space exploration''. What does it mean for embedded system design? What are typical design parameters which are included in a design space.

6. Design process is often controlled by the time-to-market requirement. Explain this requirement and possible consequences on the design methodology.

Design Methodology

1. Input to a system design is usually defined as system specifications and a set of functional and non-functional requirements. Discuss system specification methods as well as different types of requirements.
2. What are basic features of good requirements? Discuss them briefly.
3. Discuss how design requirements can be created.
4. What is design flow? Give an example of design flow paradigm used for embedded system design.
5. Discuss different design flow approaches. Compare them and point out similarities and differences between them.
6. Explain the ideas behind the following design flow models:
    - waterfall model,
    - spiral model,
    - stepwise refinement model,
    - top-down model,
    - bottom-up model.
7. Hardware/Software co-design methodology becomes popular for embedded systems. Discuss basic ideas behind this methodology and compare it to a traditional design methodologies.
8. What are basic design steps in hardware/software co-design? Why this methodology can improve design process and design quality?
9. Discuss a typical design methodology for embedded systems. Where different design activities, such as design specification, design partitioning, component allocation, and communication synthesis are performed?
10. What are main design activities in communication synthesis?
11. What are IP-components. Discuss briefly hard and soft IP-components and their role in a design process.
12. Discuss different design verification methods and their advantages and disadvantages.
13. Design automation tools are used in many design methodologies to help designer in solving tedious design activities. Discuss theoretical limitations of design automation tools.
14. Many design automation problems belong to the class of NP-complete or NP-hard problems. Discuss briefly how this inherent complexity problem is solved in design automation tools. What kind of algorithm are used?

Specification Languages

1. Discuss briefly what languages can be used for specification and implementation of embedded systems. What are their related advantages and disadvantages.
2. VHDL is a hardware description language which is often used to specify, simulate and synthesis hardware for embedded systems. Discuss basic VHDL constructs.
3. Hardware is inherently parallel and this need to be modeled in specification and design languages. How VHDL supports parallelism for hardware specification.

4. VHDL simulator is implemented as an event-driven simulator. Describe briefly the main idea of event-driven simulation. How time is handled in this simulation paradigm.
5. Discuss how different modeling styles, such as structural, behavioral, and data-flow can be mixed in a single VHDL model.
6. Present briefly VHDL simulation mechanism. Point out when VHDL code is executed and when time and signals are updated?
7. What is a signal driver in VHDL simulator. How is it used during simulation? How signals are updated by the simulator?
8. What is delta delay in VHDL? How is it used during simulation?
9. Give the values assigned to signals and variables by the part of the process code included below.

```
P1: process
   variable a, b : integer;
begin
      :
   -- initial values of s1 = 0, s2 = 0, a = 0, b = 0
   s1 <= 10;
   a := s1;
   b := 3;
   s2 <= b + s1;
   -- give values of s1, s2, a, b here (1)
   wait for 10 ns;
   -- give values of s1, s2, a, b here (2)
      :
end process;
```

24. VHDL is used both for hardware simulation and synthesis. Discuss briefly the language features which create problems for synthesis. What are possible solutions to this problem.
25. Discuss briefly commonly accepted restrictions for VHDL used for high-level synthesis.

Data-flow computational models

1. Discuss basic characteristics of data-flow and control-flow models. What are main application areas for both models of computation.
2. A filter is represented by a data-flow graph depicted in Figure 1. Additions have their usual meaning and multiplications are always by a constant. Using this data-flow graph determine filter equations, i.e., a function which defines y(n).

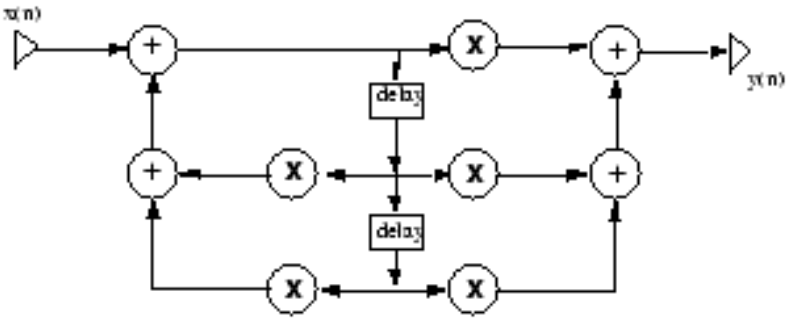


Figure 1: An example of data-flow graph

3. The theory of data-flow models is based on Khan process networks. In this theory it is proved that a network process is monotonous if $X \sqsubseteq X' \sqsubseteq F(X) \sqsubseteq F(X')$. What does this property means in practice? Which computations can be implemented using data-flow models?
4. Describe informally what are execution rules for data-flow network built of actors. Explain a notion of actors, tokens and firing rules.
5. What is a sufficient condition for a data-flow network to be deterministic.
6. Discuss briefly subclasses of data-flow networks called synchronous data-flow and boolean data-flow. Give examples of actors for these networks and their firing rules.
7. What is a firing rule for a data-flow network? Give examples of firing rule.
8. What does it mean that a set of firing rules for an actor of data-flow network is sequential? Give an algorithm for checking this property.

Control-flow computational models

1. Explain the main idea of StateCharts. How they handle concurrency and hierarchy?
2. Explain the main idea of AND and OR states in StateChart model. Why a notion of AND and OR states is defined?
3. Describe a general idea of Petri nets? How they can be used to model parallel activities.
4. Draw simple Petri nets which model sequential execution, parallel execution and non-deterministic choice.
5. How firing of a transition in a Petri net is defined? What does it mean that a transition is enabled and it may fire?
6. Define the following properties of Petri nets:
   - reachability,
   - liveness,
   - boundness,
   - safeness,
   - conservation.

   Try to relate this formal properties to practical properties of embedded system, i.e., system properties which can be analyzed using Petri nets notions.
7. Explain the reachability tree for Petri nets. What kind of problems can be analyzed using this technique? What is a reason for limitation of this method?

8. Describe Petri nets analysis method based on matrix equations. What are limitations of this method? Why the method has these limitations?

Partitioning

1. What is embedded system partitioning? What is decided during partitioning?
2. What is difference between *structural* and *functional* partitioning? Which one is more suitable for system partitioning?
3. What does it mean *partitioning granularity*? Give examples of possible granularity.
4. How are defined *objective* or *closeness* functions for partitioning? Give examples.
5. How design constraints, such as number of pins in a package or maximal permitted are, are handled during partitioning? Give two alternatives.
6. Describe an algorithm for hierarchical partitioning.
7. What is a basic method for transformation based (iterative) partitioning? What is a main problem with this approach?
8. Describe briefly Kernighan-Lin algorithm for bi-partitioning.
9. What is a basic approach to neighborhood search based partitioning. Give an example.
10. Describe briefly simulated annealing partitioning approach.
11. What is hardware/software partitioning? What is a goal of this partitioning?
12. Assume that the weighted graph depicted in Figure 2 represents tasks and their intercommunications. The weight of a node represents a size of the task while the weight of an edge represents the communication ``cost''. Using hierarchical clustering and appropriate closeness function find a partitioning which minimizes communication cost.

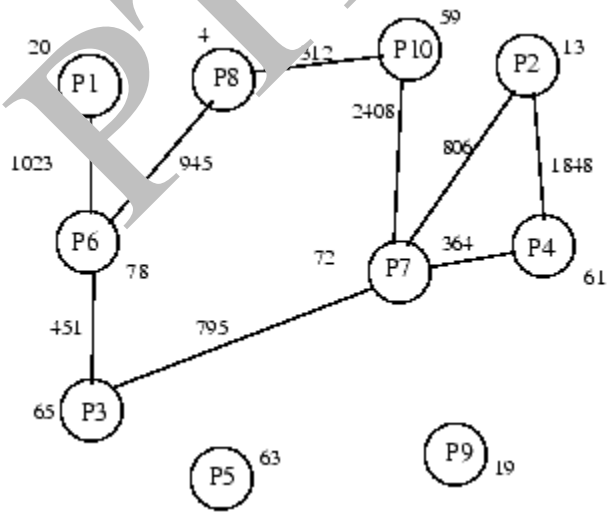


Figure 2: An example system represented as a graph.

Scheduling

1. Discuss static and dynamic scheduling approaches. Point out their application areas as well as advantages and disadvantages.
2. Explain difference between time-constrained and resource-constrained scheduling methods.

3. Explain the following scheduling approaches ASAP, ALAP and list scheduling. What is an advantage of list scheduling?
4. Explain forced-directed scheduling.
5. Explain a role of chaining and pipelining during scheduling.
6. What does it mean resource sharing and conditional resource sharing during scheduling?
7. What is speculative execution during scheduling?
8. How static scheduling influences register allocation?
9. How static scheduling influences power consumption?
10. Make static scheduling of a filter depicted in Figure 1. Assume that a delay defines new input/output pair.
11. Discuss briefly different real-time scheduling policies. Divide them into off-line and on-line. Furthermore, discuss different priority assignment strategies.
12. Explain Rate Monotonic Scheduling (RMS). What is a limitation of this method.
13. What is priority inversion problem for Rate Monotonic Scheduling (RMS)? What are methods for solving this problem?
14. Describe briefly priority ceiling protocol for solving priority inversion problem.
15. Describe briefly Earliest Deadline First (EDF) dynamic priority scheduling.
16. Use the RMS schedulability analysis to check if each of the following task sets is schedulable with RMS. If not, identify which tasks will miss their deadlines.

Task set 1		
Task	Period	Worst execution time
1	5	1
2	3	2
3	8	4
Task set 2		
Task	Period	Worst execution time
1	2	1
2	3	1
Task set 3		
Task	Period	Worst execution time

1	5	1
2	7	2
3	9	4

Table 1: Three task sets

Interface Synthesis

1. Describe briefly basic steps in communication synthesis.
2. What are main objectives of channel binding, communication refinement, interface generation?
3. Describe briefly strobe and handshake protocols for processor communication with other devices.
4. Why interrupts are used for processor communication with external devices?
5. What is a role of an interrupt service routine?
6. Describe briefly functionality of a typical interrupt service routine.
7. What is DMA controller and what is its role in communication between different devices?
8. Describe briefly functionality of a DMA controller.
9. Why there is a need for bus arbitration? What are bus arbitration methods?
10. Present briefly daisy-chain arbitration method for a bus.
11. The handshaking protocol discussed in the course is represented in Figure 3. Formalize this specifications using, for example, Petri nets and prove that the master after sending a request will get data. Extend the example with two slaves and ensure that the protocol is fair, i.e., non of two slaves will wait for the possibility to send data forever. How can you include timing analysis in the model?

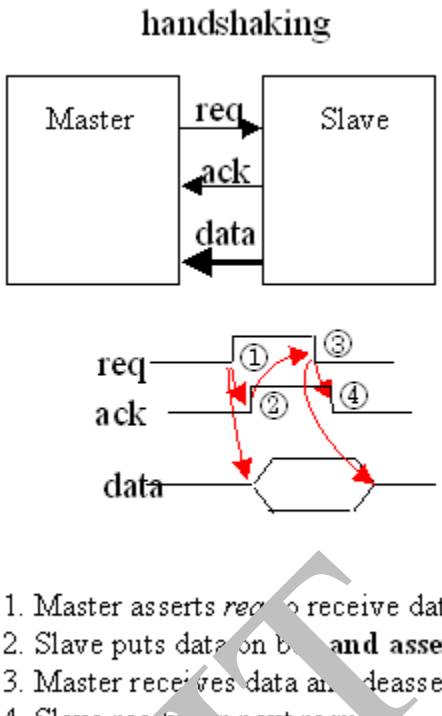


Figure 3: An example handshaking protocol.

Low Power Design

1. What are main sources of power consumption in CMOS technology? How power consumption can be minimized?
2. What is a relation between power consumption and energy consumption? Explain importance of power and energy consumption for embedded systems?
3. What are basic ways of reducing power consumption?

Testability

1. Explain the concept of production faults and their models.
2. Explain stuck-at fault model.
3. Describe briefly test generation method based on fault sensitization and fault propagation.
4. What are reconvergent fan-out points? Why do they create problems for test generation?
5. Discuss basic testability improvement methods.
6. What is the main goal of all testability improvement methods?
7. Discuss briefly main idea of SCAN path testability improvement method.
8. Discuss briefly main idea of BIST testability improvement method.

9. Explain briefly testability improvement by circuit partitioning and enhancement with specific testability improvement methods.

%%%

Hà nội 19 tháng 12 năm 2013.
Kỉ niệm chiến thắng cuộc tập kích 12 ngày đêm vào *Hà Nội 1972 của không lực Hoa Kì.*