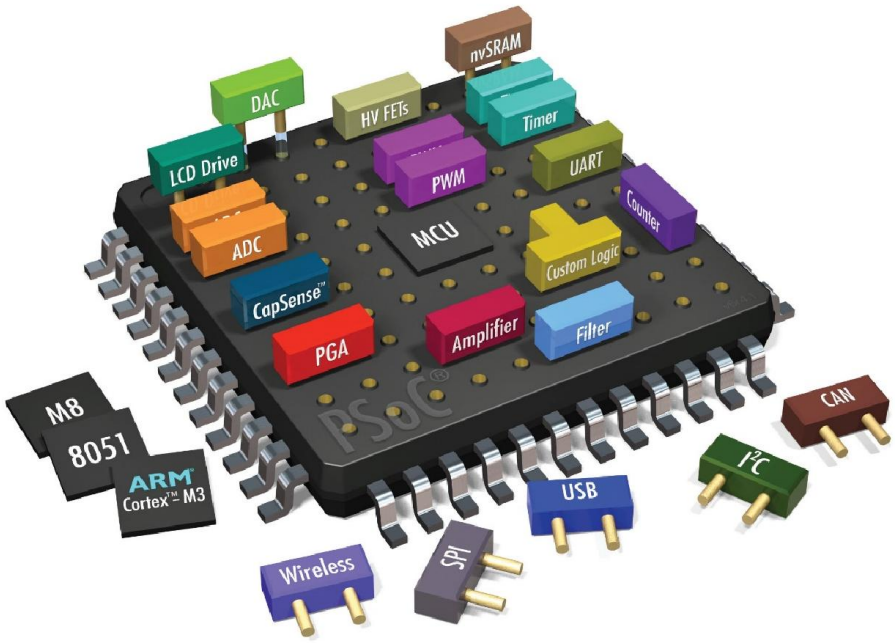
TRƯỜNG ĐẠI HỌC VĂN HIẾN TPHCM
KHOA KỸ THUẬT – CÔNG NGHỆ
BỘ MÔN ĐIỆN TỬ - VIỄN THÔNG

LẬP TRÌNH NHÚNG



TÀI LIỆU LƯU HÀNH NỘI

MỤC LỤC

Chương 1: TỔNG QUAN

1.1 Mở đầu

Trong sự phát triển mạnh mẽ của khoa học kĩ thuật với nền kinh tế trí thức và xu hướng hội nhập toàn cầu như hiện nay, thế giới và Việt Nam đang thực hiện việc kết hợp giữa các ngành thuộc lĩnh vực công nghệ cao trong một Khoa hoặc cơ sở đào tạo. Đó là lĩnh vực khoa học dưới 3 ngọn cờ: Máy tính, Điện tử- Viễn thông và Điều khiển tự động mà ta thường gọi là "3 C" (Computer – Communication - Control). Có thể nói, các quá trình sản xuất và quản lí hiện nay như: các hệ thống đo lường điều khiển tự động trong sản xuất công nghiệp; các hệ thống di động và không dây tiên tiến, các hệ thống thông tin vệ tinh, các hệ thống thông tin dựa trên Web, chính phủ điện tử, thương mại điện tử, các cơ sở dữ liệu của nhiều ngành kinh tế và của Quốc gia, các hệ thống thiết bị Y tế hiện đại, các thiết bị điện tử dân dụng, ... đều là sản phẩm của sự kết hợp giữa các lĩnh vực khoa học trên.

Hiện nay chúng ta đang ở thời đại hậu PC sau giai đoạn phát triển của máy tính lớn (Mainframe) 1960-1980, và sự phát triển của PC-Internet giai đoạn 1980-2000. Giai đoạn hậu PC-Internet này được dự đoán từ năm 2000 đến 2020 là giai đoạn của môi trường thông minh mà hệ thống nhúng là cốt lõi và đang làm nên làn sóng đổi mới trong công nghệ thông tin nói riêng và lĩnh vực công nghệ cao "3C", nói chung. Một thực tế khách quan là thị trường của các hệ thống nhúng lớn gấp khoảng 100 lần thị trường PC, trong khi đó chúng ta mới nhìn thấy bề nổi của công nghệ thông tin là PC và Internet còn phần chìm của công nghệ thông tin chiếm 99% số processor trên toàn cầu này nằm trong các hệ nhúng thì còn ít được biết đến.

Sức đẩy của công nghệ đưa công nghệ vi điện tử, công nghệ vi cơ điện, công nghệ sinh học hội tụ tạo nên các chip của công nghệ nano, là nền tảng cho những thay đổi cơ bản trong lĩnh vực công nghệ cao "3C, sức kéo của thị trường đòi hỏi các thiết bị phải có nhiều chức năng thân thiện với người dùng, có mức độ thông minh ngày càng cải thiện đưa đến vai trò và tầm quan trọng của các hệ thống nhúng ngày càng cao trong nền kinh tế quốc dân.

Phát triển các hệ nhúng và phần mềm nhúng là quốc sách của nhiều quốc gia trên thế giới, nhất là giai đoạn hậu PC hiện nay. Ở nước ta đáng tiếc lĩnh vực này lâu nay đã bị lãng quên, do vậy cần có những điều chỉnh phù hợp trong chiến lược phát triển để có thể theo kịp, rút ngắn khoảng cách tụt hậu đối với các nước trong khu vực và trên thế giới trong quá trình hội nhập nền kinh tế toàn cầu không thể tránh khỏi hiện nay.

1.2 Khái niệm về hệ nhúng

Hệ thống nhúng (tiếng Anh: Embedded system) là một thuật ngữ để chỉ một thống có khả năng tự trị được nhúng vào trong một môi trường hay một hệ thống mẹ. Hệ thống nhúng có vai trò đảm nhận một phần công việc cụ thể của hệ thống mẹ. Hệ thống nhúng có thể là một hệ thống phần cứng và cũng có thể là một hệ thống phần mềm.
(Wikipedia, 2010)

Ví dụ quanh ta có rất nhiều sản phẩm nhúng như lò vi sóng, nồi cơm điện, điều hòa, điện thoại di động, ô tô, máy bay, tàu thủy, các đầu đo, cơ cấu chấp hành thông minh v.v... ta có thể thấy hiện nay hệ thống nhúng có mặt ở mọi lúc mọi nơi trong cuộc sống của chúng ta.

Hình 1.1 Một số ví dụ về các thống nhúng thông dụng

Các nhà thống kê trên thế giới đã thống kê được rằng số chip vi xử lý ở trong các máy PC và các server, các mạng LAN, WAN, Internet chỉ chiếm khoảng 1% tổng số chip vi xử lý có trên thế giới, 99% số vi xử lý còn lại nằm trong các hệ thống nhúng.

Như vậy công nghệ thông tin không chỉ đơn thuần là PC, mạng LAN, WAN, Internet phần mềm quản lý ... như nhiều người thường nghĩ. Đó chỉ là bề nổi của một tảng băng chìm. Phần chìm của công nghệ thông tin chính là các ứng dụng của các hệ nhúng có mặt trong mọi ngành nghề của đời sống xã hội hiện nay. Các hệ nhúng được tích hợp trong các thiết bị đo lường điều khiển tạo nên đầu não và linh hồn của sản phẩm. Trong các hệ nhúng, hệ thống điều khiển nhúng đóng một vai trò hết sức quan trọng.

Hệ điều khiển nhúng là hệ thống mà máy tính được nhúng vào vòng điều khiển của sản phẩm nhằm điều khiển một đối tượng, điều khiển một quá trình công nghệ đáp ứng các yêu cầu đặt ra. Hệ thống điều khiển nhúng lấy thông tin từ các cảm biến, xử lý tính toán các thuật điều khiển và phát tín hiệu điều khiển cho các cơ cấu chấp hành.

Khác với các hệ thống điều khiển cổ điển theo nguyên lý thủy lực, khí nén, rơle, mạch tương tự, hệ điều khiển nhúng là hệ thống điều khiển số được hình thành từ những năm 1960 đến nay. Trước đây các hệ điều khiển số thường do các máy tính lớn đảm nhiệm, ngày nay chức năng điều khiển số này do các chíp vi xử lý, các hệ nhúng đã thay thế. Phần mềm điều khiển ngày càng tinh sảo tạo nên độ thông minh của thiết bị và ngày càng chiếm tỉ trọng lớn trong giá thành của thiết bị.

Điểm qua về chức năng xử lý tin ở PC và ở các thiết bị nhúng có những nét khác biệt. Đối với PC và mạng internet chức năng xử lý đang được phát triển mạnh ở các lĩnh vực quản lý và dịch vụ như thương mại điện tử, ngân hàng điện tử, chính phủ điện tử, thư viện điện tử, đào tạo từ xa, báo điện tử ... các ứng dụng này thường sử dụng máy PC để bàn, mạng WAN, LAN hoạt động trong thế giới ảo. Còn đối với các hệ nhúng thì chức năng xử lý tính toán được ứng dụng cụ thể cho các thiết bị vật lý (thế giới thật) như mobile phone, quần áo thông minh, các thiết bị điện tử cầm tay, thiết bị y tế, xe ô tô, tầu tốc hành, phương tiện vận tải thông minh, máy đo, đầu đo, cơ cấu chấp hành thông minh, các hệ thống điều khiển, nhà thông minh, thiết bị gia dụng thông minh v.v...

1.3 Vai trò của hệ thống nhúng trong sự phát triển của lĩnh vực công nghệ cao "3C".

Các hệ thống tự động đã được chế tạo trên nhiều công nghệ khác nhau như các thiết bị máy móc tự động bằng các cam chốt cơ khí, các hệ thống tự động hoạt động bằng nguyên lý khí nén, thủy lực, rơle cơ điện, mạch điện tử số ... các thiết bị, hệ thống này có chức năng xử lý và mức độ tự động thấp so với các hệ thống tự động hiện đại được xây dựng trên nền tảng của các hệ thống nhúng.

Trong khi các hệ thống tin học sử dụng máy tính để hỗ trợ và tự động hóa quá trình quản lý, thì các hệ thống điều khiển tự động dùng máy tính để điều khiển và tự động hóa quá trình công nghệ. Chính vì vậy các thành tựu của công nghệ phần cứng và công nghệ phần mềm của máy tính điện tử được áp dụng và phát triển một cách có chọn lọc và hiệu quả cho các hệ thống điều khiển tự động. Và sự phát triển như vũ bão của công nghệ thông tin kéo theo sự phát triển không ngừng của lĩnh vực tự động hóa.

Ta có thể thấy quá trình các hệ nhúng thâm nhập vào từng phần tử, thiết bị thuộc lĩnh vực tự động hóa như đầu đo, cơ cấu chấp hành, thiết bị giao diện với người vận hành thậm chí vào các rơle, contactor, nút bấm mà trước kia hoàn toàn làm bằng cơ khí.

Trước khi đầu đo gồm phần tử biến đổi từ tham số đo sang tín hiệu điện, mạch khuyếch đại, mạch lọc và mạch biến đổi sang chuẩn 4-20mA để truyền tín hiệu đo về trung tâm xử lý. Hiện nay đầu đo đã tích hợp cả chip vi xử lý, biến đổi ADC, bộ truyền dữ liệu số với phần mềm đo đạc, lọc số, tính toán và truyền kết quả trên mạng số về thẳng máy tính trung tâm. Như vậy đầu đo đã được số hóa và ngày càng thông minh do các chức năng xử lý từ máy tính trung tâm trước kia nay đã được chuyển xuống xử lý tại chỗ bằng chương trình nhúng trong đầu đo.

Tương tự như vậy cơ cấu chấp hành như môtơ đã được chế tạo gắn kết hữu cơ với cả bộ servo với các thuật toán điều chỉnh PID tại chỗ và khả năng nối mạng số tới máy tính chủ.

Các tủ rơle điều khiển chiếm diện tích lớn trong các phòng điều khiển nay được co gọn trong các PLC(programble Logic Controller).

Các bàn điều khiển với hàng loạt các đồng hồ chỉ báo, các phím, núm điều khiển, các bộ tự ghi trên giấy cồng kềnh nay được thay thế bằng một vài PC.

Hệ thống cáp truyền tín hiệu analog 4-20mA, ± 10V từ các đầu đo, cơ cấu chấp hành về trung tâm điều khiển nhằng nhịt trước đây đã được thay thế bằng vài cáp đồng trục hoặc cáp quang truyền dữ liệu số

Ta có thể nói các hệ nhúng đã "thay thế và chiếm phần ngày càng nhiều" trong các phần tử, hệ thống thuộc lĩnh vực công nghệ cao "3C".

Vào những năm 30 các hệ thống tự động bằng cam chốt cơ khí thường hoạt động đơn lẻ với một chức năng xử lý. Các hệ thống tự động dùng rơle điện từ xuất hiện vào những năm 40 có mức xử lý khoảng 10 chức năng. Các hệ thống tự động dùng bán dẫn hoạt động theo nguyên lý tương tự (Analog) của thập kỷ 60 có mức xử lý khoảng 30 chức năng. Vào những năm 70 các thiết bị điều khiển khả trình PLC ra đời với mức độ xử lý lên hàng trăm và vào những năm 80 với sự tham gia của các máy tính điện tử main frame mini đã hình thành các hệ thống điều khiển phân cấp với số chức năng xử lý lên tới hàng chục vạn (105). Sang thập kỷ 90 với sự phát triển của công nghệ phần cứng cũng như phần mềm, các hệ thống điều khiển phân tán ra đời(DCS) cho mức xử lý lên tới hàng trục triệu (107). Và sang thế kỷ 21, những hệ thống tự động có tính tự tổ chức, có tư duy hợp tác sẽ có mức xử lý lên tới hàng tỷ(109). Tuy nhiên để đạt được độ thông minh như những sinh vật sống còn cần nhiều thời gian hơn và các hệ thống tự động còn cần tích hợp trong nó nhiều công nghệ cao khác như công nghệ cảm biến, công nghệ vật liệu mới, công nghệ quang và laser v.v... Đây cũng là xu thế phát triển của các hệ thống tự động là ngày càng sử dụng nhiều công nghệ mới hơn trong cấu trúc và hoạt động của mình.

Trong điều khiển quá trình công nghệ, việc áp dụng các hệ nhúng đã tạo ra khả năng tự động hóa toàn bộ dây chuyền sản xuất. Kiến trúc hệ thống điều khiển trước kia tập trung về xử lý tại một máy tính thì nay các đầu đo, cơ cấu chấp hành, giao diện với người vận hành đều được thông minh hóa có nhiều chức năng xử lý tại chỗ và khả năng nối mạng nhanh tạo thành hệ thống mạng máy điều khiển hoạt động theo chế độ thời gian thực. Ngoài các chức năng điều khiển và giám sát dây chuyền sản xuất hệ thống còn có nhiều cơ sở dữ liệu, khả năng tự xác định và khắc phục hỏng hóc, khả năng thống kê, báo cáo và kết hợp hệ thống mạng máy tính quản lý, lập kế hoạch, thiết kế và kinh doanh tạo thành hệ thống tự động hóa sản xuất toàn cục.

Trong lĩnh vực rôbôt, với sự áp dụng các thành tựu của các hệ nhúng, rôbôt đã có thị giác và xúc giác. Việc áp dụng trí khôn nhân tạo vào rôbôt đã đưa rôbôt từ ứng dụng chủ yếu trong công nghiệp sang các lĩnh vực dịch vụ và y tế. Kết hợp với các thành tựu của cơ điện tử, rôbôt ngày càng uyển chuyển và thông minh hơn. Trong tương lai rôbôt không chỉ thay thế hoạt động cơ bắp của con người mà còn có thể thay thể các công việc đòi hỏi hoạt động trí não của con người. Lúc này hệ thống điều khiển của rôbôt không chỉ là các vi xử lý mạnh mà còn có sự hỗ trợ của các máy tính mạng nơron nhân tạo, xử lý song song nhúng trong rôbôt. Các nghiên cứu phát triển này hiện nay còn ở giai đoạn ban đầu.

1.4 Đặc tính, phương pháp thiết kế và xu thế phát triển của các hệ nhúng

Các hệ nhúng là những hệ kết hợp phần cứng và phần mềm một cách tối ưu. Các

hệ nhúng là những hệ chuyên dụng, thường hoạt động trong chế độ thời gian thực, bị hạn chế về bộ nhớ, giá thành phải rẻ nhưng lại phải hoạt động tin cậy và tiêu tốn ít năng lượng. Các hệ nhúng rất đa dạng và có nhiều kích cỡ, khả năng tính toán khác nhau. Ngoài ra các hệ nhúng thường phải hoạt động trong môi trường khắc nghiệt có độ nóng ẩm, rung xóc cao. Ví dụ như các điều khiển các máy diesel cho tàu biển, các thiết bị cảnh báo cháy nổ trong hầm lò. Các hệ thống nhúng lớn thường là các hệ nối mạng. Ở máy bay, tàu vũ trụ thường có nhiều mạng nhúng kết nối để kiểm soát hoạt động và điều khiển.Trong ô tô hiện đại có đến trên 80 nút mạng kết nối các đầu đo cơ cấu chấp hành để bảo đảm ô tô hoạt động an toàn và thoải mái cho người sử dụng.

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

Phần mềm trong các hệ nhúng ngày càng chiếm tỉ trọng cao và đã trở thành một thành phần cấu tạo nên thiết bị bình đẳng như các phần cơ khí, linh kiện điện tử, linh kiện quang học ... các hệ nhúng ngày càng phức tạp hơn đáp ứng các yêu cầu khắt khe về thời gian thực, tiêu ít năng lượng, hoạt động tin cậy ổn định hơn, có khả năng hội thoại cao, có khả năng kết nối mạng, có thích nghi, tự tổ chức cao có khả năng tái cấu hình như một thực thể, một tác nhân.

Và có khả năng tiếp nhận năng lượng từ nhiều nguồn khác nhau (ánh sáng, rung động, điện từ trường, sinh học ...) để tạo nên các hệ thống tự tiếp nhận năng lượng trong quá trình hoạt động.

Tuy nhiên hệ thống nhúng hiện nay còn phải đối mặt với nhiều thách. Độ phức tạp của hệ thống tăng cao do nó kết hợp nhiều lĩnh vực đa ngành, kết hợp phần cứng - mềm, trong khi các phương pháp thiết kế và kiểm tra chưa chín muồi. Khoảng cách giữa lý thuyết và thực hành lớn và còn thiếu các phương pháp và lý thuyết hoàn chỉnh cho khảo sát phân tích toàn cục của hệ nhúng bao gồm lý thuyết điều khiển tự động, thiết kế máy, công nghệ phần mềm, điện tử, vi xử lý, các công nghệ hỗ trợ khác. Mặt khác các hệ nhúng còn nhiều vấn đề cần giải quyết với độ tin cậy và tính mở của hệ thống. Do hệ thống nhúng thường phải hội thoại với môi trường xung quanh nên nhiều khi gặp những tình huống không lường trước dễ dẫn đến hệ thống bị loạn. Trong quá trình hoạt động một số phần mềm thường phải chỉnh lại và thay đổi nên hệ thống phần mềm có thể không kiểm soát được. Đối với hệ thống mở, các hãng thứ 3 đưa các module mới, thành phần mới vào cũng có thể gây nên sự hoạt động thiếu tin cậy.

1.5 Môi trường thông minh

Công nghệ bán dẫn phát triển mạnh theo xu thế ngày càng rẻ, tích hợp cao, có khả năng tính toán lớn, khả năng kết nối toàn cầu, khả năng phối hợp với các cảm biến và cơ cấu chấp hành vi cơ điện và sinh học, khả năng giao diện không qua bàn phím đang tạo tiền đề và cơ sở cho sự bùng nổ của các thiết bị vật dụng thông minh xung quanh ta. Đây là sự khởi đầu của thời đại hậu PC - Môi trường thông minh. Các phần mềm nhúng trong các chip vi hệ htống rất phong phú và có độ mềm dẻo, tái sử dụng cao.

Sức đẩy của công nghệ sẽ đưa công nghệ vi điện tử tiếp cận và cộng năng với công nghệ sinh học tạo nên công nghệ nano với độ phức tạp giga vào thập niên 2010-2020. Các chip vi hệ thống xử lý hỗn hợp tương tự và số MS-SoC (Mixed Signal System on chip) vào giai đoạn này sẽ có trên 2 tỷ transistor, 1000 lõi CPU, 100MB bộ nhớ và hoạt động ở tần số 200GHz.

Với những vi hệ thống có khả năng tính toán siêu hạng này việc thiết kế các hệ nhúng sẽ gặp không ít thách thức như xử lý song song, độ phức tạp của phần mềm nhúng và khả năng cung cấp năng lượng cho các thiết bị cầm tay. Trong tương lai năng lượng cho truyền dữ liệu sẽ lớn gấp từ 5 đến 30 lần năng lượng hoạt động của các CPU.

Trước đây các hệ thống thường được thiết kế trên nền phần cứng là PC và phần mềm là Windown hoặc Linux, thì ngày nay số lượng các hệ nền (platform) cho thiết kế các hệ nhúng có khoảng 25. Trong tương lai các hệ nhúng sẽ được thiết kế trên nền các chip MS-SoC tạo nên các platform thiết kế chuyên dụng với số lượng sẽ lên đến hơn 500 loại. Ta có thể liệt kê một số ví dụ điển hình như platform raptor II cho thiết kế camera số, PXA240 cho thiết kế các thiết bị PDA, TL850 cho TV số, BLUECORE cho công nghệ không dây Bluetooth, CDMA cho mobile phone 3G ... các hệ MS-SoC sẽ có khả năng tái cấu hình và sẽ là công cụ chủ chốt cho các sản phẩm của công nghệ cao "3C".

1.6 Các hệ điều hành nhúng và phần mềm nhúng

1.6.1 Hệ điều hành nhúng

Khác với PC thường chạy trên nền hệ điều hành windows hoặc unix, các hệ thống nhúng có các hệ điều hành nhúng riêng của mình. Các hệ điều hành dùng trong các hệ nhúng nổi trội hiện nay bao gồm Embedded linux, VxWorks, WinCE, Lynyos, BSD, Green Hills, QNX và DOS, Embeddde linux hiện đang phát triển mạnh. Năm 2001 hệ điều hành này chiếm 12% thị phần các hệ điều hành nhúng thì năm 2002 chiếm 27% và chiếm vị trí số 1. Hiện nay 40% các nhà thiết kế các hệ nhúng cân nhắc đầu tiên sử dụng Embedded linux cho các ứng dụng mới của mình và sau đó mới đến các hệ điều hành nhúng truyền thống như VxWorks, WinCE. Các đối thủ cạnh tranh của Embedded linux hiện nay là các hệ điều hành nhúng tự tạo và windows CE. Sở dĩ Embedded linux có sự phát triển vượt bậc là do có sức hấp dẫn đối với các ứng dụng giá thành thấp và đòi hỏi thời gian đưa sản phẩm ra thị trường nhanh. Mặt khác Linux là phần mềm mã nguồn mở nên bất kỳ ai cũng có thể hiểu và thay đổi theo ý mình. Linux cũng là một hệ điều hành có cấu trúc module và chiếm ít bộ nhớ trong khi windows không có các đặc tính ưu

việt này. Do thị trường của các sản phẩm nhúng tăng mạnh lên các nhà sản xuất ngày càng sử dụng các hệ điều hành nhúng để đảm bảo sản phẩm có sức cạnh tranh và Embedded linux đang là sản phẩm hệ điều hành nhúng có uy tín chiếm vị trí số 1 trong những năm tới.

1.6.2 Phần mềm nhúng

Phần mềm nhúng là phần mềm tạo nên phần hồn, phần trí tuệ của các sản phẩm nhúng. Phần mềm nhúng ngày càng có tỉ lệ cao trong giá trị của các sản phẩm nhúng. Hiện nay phần lớn các phần mềm nhúng nằm trong các sản phẩm truyền thông và các sản phẩm điện tử gia dụng (consumer electronics) tiếp đến là trong các sản phẩm ô tô, phương tiện vận chuyển, máy móc thiết bị y tế, các thiết bị năng lượng các thiết bị cảnh báo bảo vệ và các sản phẩm đo và điều khiển. Để có thể tồn tại và phát triển, các sản phẩm công nghiệp và tiêu dùng cần phải thường xuyên đổi mới và ngày càng có nhiều chức năng tiện dụng và thông minh hơn. Các chức năng này phần lớn do các chương trình nhúng tạo nên. Phần mềm nhúng là một lĩnh vực công nghệ then chốt cho sự phát triển kinh tế của nhiều quốc gia trên thế giới như Nhật Bản, Hàn Quốc, Phần Lan và Trung Quốc. Tại Mỹ có nhiều chương trình hỗ trợ của nhà nước để phát triển các hệ thống và phần mềm nhúng. Hàn Quốc có những dự án lớn nhằm phát triển công nghệ phần mềm nhúng như các thiết bị gia dụng nối mạng Internet, hệ thống phần mềm nhúng cho phát triển thành phố thông minh, dự án phát triển ngành công nghiệp phần mềm nhúng, trung tâm hỗ trợ các ngành công nghiệp hậu PC v.v... Hàn Quốc cũng chấp nhận Embedded linux như một hệ điều hành chủ chốt trong việc phát triển các sản phẩm nhúng của mình. Thụy Điển coi phát triển các hệ nhúng có tầm quan trọng chiến lược cho sự phát triển của đất nước. Phần Lan có những chính sách quốc gia tích cực cho nghiên cứu phát triển các hệ nhúng đặc biệt là các phần mềm nhúng. Những quốc gia này còn thành lập nhiều viện nghiên cứu và trung tâm phát triển các hệ nhúng.

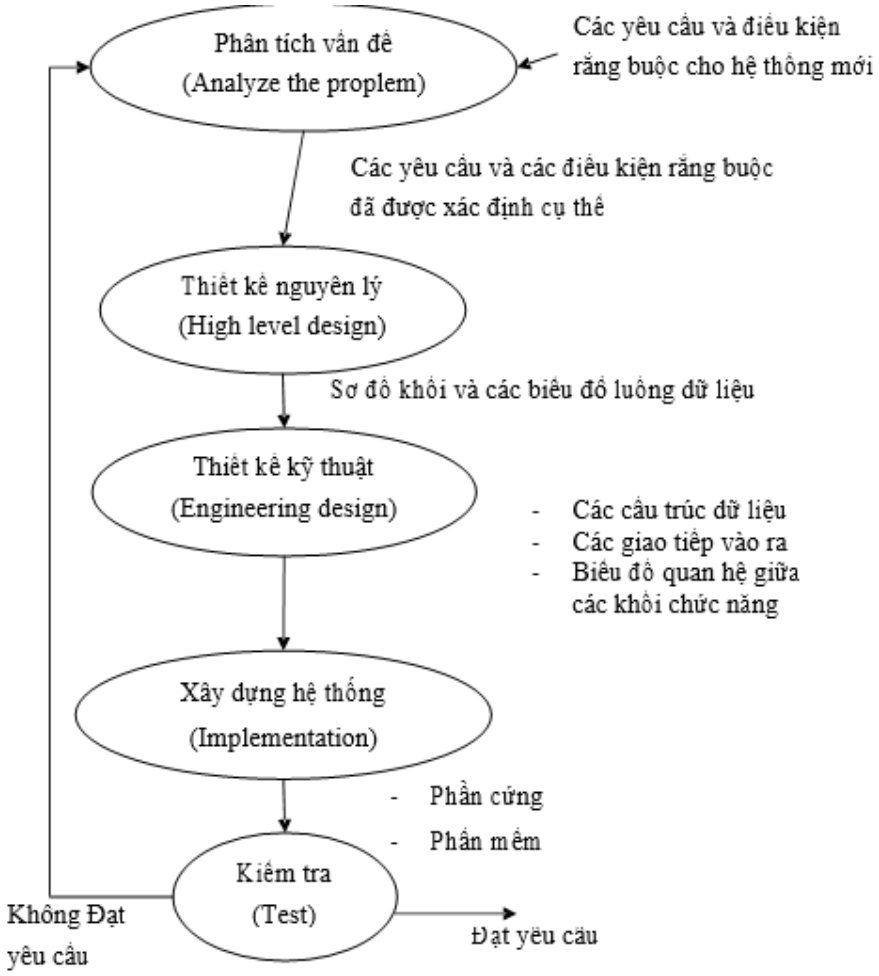
Chương 2: LÝ THUYẾT THIẾT KẾ HỆ THỐNG NHÚNG

2.1 Quy trình thiết kế Top-Down

Quy trình Top-Down thường được áp dụng cho các bài toán đã có giải pháp công nghệ cả về phần mềm cũng như phần cứng. Các giải pháp này đã được phát triển trước đó ở các ứng dụng khác, và đã được kiểm định.

Trong thực tế chúng ta sẽ thấy, bản chất hay mấu chốt của quy trình là vấn đề tìm hiểu và xác định bài toán, làm sao để xác định được chính xác và đầy đủ nhất các yêu cầu cũng các rằng buộc mà hệ thống phải đạt được.

Sơ đồ khối quy trình kế top-down ở hình 2.1



Hình 2.1 Sơ đồ khối quy trình Top - Down

2.1.1 Pha phân tích

Pha này là pha quan trọng nhất quyết định hệ thống có đạt yêu cầu hay không. Một hệ thống nhúng cụ thể phải được đặt vào (nhúng vào) một hệ thống lớn cụ thể. Vì thế ta

cần phải biết có những yêu cầu nào cho nó, môi trường hay điều kiện làm việc của nó. Thông tin này gọi là các yêu cầu và các điều kiện rằng buộc cho hệ thống, nó giúp cụ thể hoá được việc chọn giải pháp công nghệ và thiết bị cho các kỹ sư thiết kế ở pha sau.

- Các yêu cầu: Các thông tin chi tiết về nhiệm vụ mà hệ thống phải giải quyết được, các tham số đầu vào đầu ra, các giới hạn trong hệ thống cụ thể,...

- Các rằng buộc: Điều kiện làm việc và các hạn chế như thời tiết, độ ẩm, độ nhiễu, độ chính xác, tính thời gian thực, loại tín hiệu giao tiếp với hệ thống mẹ,...

Ví dụ: Xét bài toán thiết kế hệ thống điều khiển cho 1 cửa tự động

Giả sử qua quá trình khảo sát thực tế và yêu cầu của bên khác hàng ta phải xác định được các thông số tối thiểu sau:

a) Yêu cầu:

1) Hệ thống áp dụng cho 1 cửa hai chiều (vào/ra)
2) Cửa cao 2,5m rộng 3m
3) Có người trong phạm vi 2m trước và sau cửa là cửa phải mở
4) Thời gian mở và đóng cửa 3s
5) Cửa đang đóng gặp vật cản phải mở ra ngay
6) Làm việc điện áp 220v/50Hz
7) Chi phí cho bộ điều khiển không quá 10 triệu VNĐ
8) Hệ thống thống có 2 chế độ làm việc tự động và bằng tay
9) Sensor và công nghệ tuỳ chọn

b) Điều kiện rằng buộc

1) Sử dụng động cơ động lực có sẵn loại AC một pha 220V/50Hz 3kW.
2) Nơi đặt cửa có nhiều người qua lại, nên hệ thống sẽ phải làm việc với tần suất cao.
3) Điều kiện môi trường: Trong nhà, nhiệt độ từ 18^0C đến 36^0C
4) Bộ điều khiển bằng tay đặt cạnh cửa phía bên trong nhà
5) Hệ thống điện cấp mới từ đầu
6) Chịu được quá tải khi gặp chướng ngại vật trong khoảng thời gian dài.

Trong các bài toán cụ thể, với các hệ thống mẹ cụ thể, mỗi hệ thống nhúng sẽ có các yêu cầu khác nhau. Tuy nhiên vẫn có những tiêu chuẩn và yêu cầu chung mà hầu hết các hệ thống phải tính đến như:

- Tính đến chi phí bảo trì định kỳ
- Kích thước và trọng lượng
- Khả năng thực thi: Thời gian đáp ứng, độ chính xác, độ phân giải, ...

- Nguồn nuôi
- Tính mềm dẻo, khả năng nâng cấp, khả năng tương thích, khả năng phục hồi sau khi mất nguồn,...
- Thời gian thử
- Thời gian để thương mại sản phẩm
- Độ an toàn
- Khả năng chống lại sự phá hoại hay xâm nhập.

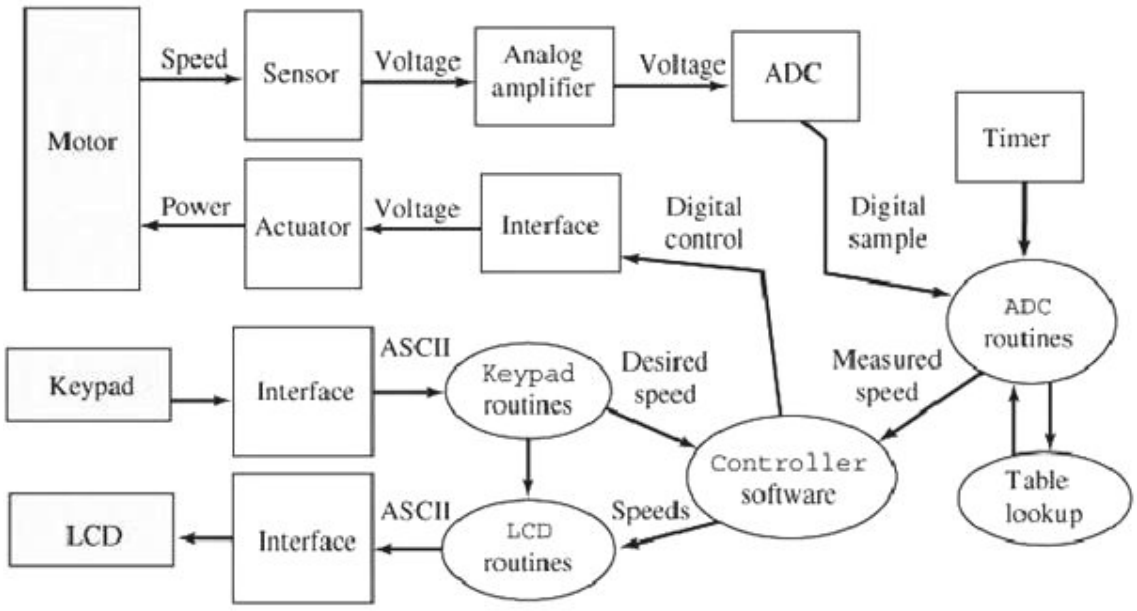
2.1.2 Pha thiết kế nguyên lý

Mục tiêu của pha này là xác định các giải pháp công nghệ từ các yêu cầu đặt ra ở pha Phân tích, từ đó đi thiết kế mô hình, sơ đồ nguyên lý cho toàn bộ hệ thống bao gồm cả phần cứng và phần mềm.

Để thực hiện bước này thông thường trải qua các bước sau:

- Trước tiên ta phải xây dựng một sơ đồ mô hình tổng quát của toàn hệ thống.
- Sau đó phân tách thành các module hay các hệ thống con.
- Định giá cho hệ thống, lập kế hoạch phát triển và ước lượng thời gian phát triển hệ thống.
- Xây dựng các sơ đồ luồng dữu liệu giữa các module hay các hệ thống con trong hệ thống

Ví dụ 2: Xây dựng và phân tách các module trong mô hình tổng quát của bài toán điều khiển động cơ.



Hình 2.2 Sơ đồ tổng quát của một hệ thống điều khiển động cơ

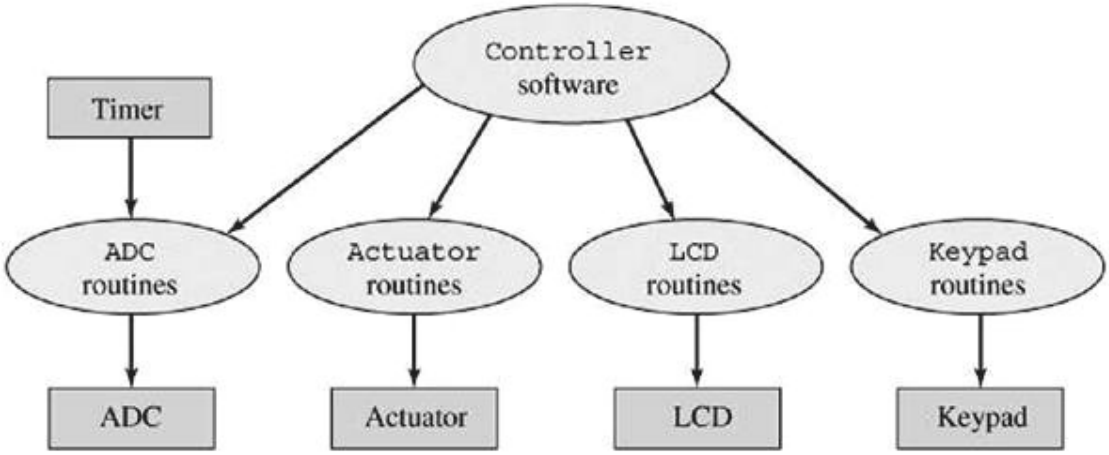
2.1.3 Pha thiết kế kỹ thuật

- Xây dựng các thiết kế chi tiết cho cả phần cứng phần mềm, các bản thiết kế này sẽ được chuyển sang pha thực thi để xây dựng hệ thống. Vì thế ở pha này người thiết kế phải đưa ra các bản thiết kế như:

1. Sơ đồ khối, sơ đồ thuật toán.
2. Cấu trúc dữu liệu, dữ liệu chia sẻ.
3. Sơ đồ nguyên lý mạch, chi tiết về các đầu vào/ra, loại tín hiệu hay giao thức giao tiếp.
4. Thông số linh kiện được chọn hoặc có thể thay thế.
5. Các tham số vào/ra cho hệ thống.
6. Lựa chọn thiết bị, công cụ phát triển hệ thống, các tài nguyên sẽ sử dụng.
7. ...

- Xây dựng sơ đồ quan hệ giữa các module và các hàm trong hệ thống (call graph), sơ đồ này mô tả cách thức tương tác giữa phần cứng và phần mềm trong hệ thống.

Ví dụ: Sơ đồ Call graph của hệ thống điều khiển động cơ



Hình 2.3 Sơ đồ quan hệ (call graph) giữa các module phần cứng và phân mềm trong hệ thống điều khiển động cơ

Để phát hiện và hạn chế tối đa các lỗi mà hệ thống sẽ gặp phải sau khi được xây dựng, ta có thể mô hình hóa các thành phần hoặc toàn bộ hệ thống nếu có thể, nhằm thử nghiệm hoạt động của hệ thống với các đầu vào và tình huống giả lập, đồng thời thử nghiệm tính thân thiện của giao diện người dùng.

2.1.4 Pha xây dựng hệ thống

Từ các bản thiết kế, bước này tiến hành xây dựng hoàn thiện hệ thống trên cả phân mềm và phần cứng. Trong suốt quá trình xây dựng phải tuân thủ theo các bước và sơ đồ công nghệ từ các bản thiết kế kỹ thuật. Đặc biệt là các tham số vào ra giữa các

hàm, điều này ảnh hưởng đến việc tích hợp các module giữa các nhóm làm việc khác nhau hay sự kế thừa từ các module khác. Các linh kiện và thiết bị sử dụng phải tuân thủ theo bản thiết kế, nhằm giúp hệ thống thỏa mãn đầy đủ các thông số rằng buộc đã được đặt ra ở pha phân tích.

Việc phát triển hệ thống có thể được phân tách thành nhiều nhóm, nhiều phần không cần tuân theo tuần tự, hoặc có thể trên nhiều môi trường khác nhau miễn sao đảm bảo việc trao đổi tham số giữa các module là tương thích và đầy đủ. Nếu chúng ta phân tách và thiết kế tốt, việc phát triển hệ thống có thể được tiến hành song song, hoặc kế thừa cái có sẵn, sẽ làm giảm thời gian phát triển hệ thống đáng kể mà vẫn đảm bảo các yêu cầu đã đặt ra.

Một số các kỹ thuật nhằm phát hiện và hạn chế lỗi mà người phát triển có thể áp dụng trong pha này là Debug hay mô phỏng Simulation. Tuy nhiên trong một bài toán cụ thể việc Debug là rất khó, thông thường người phát triển luôn sử dụng một trình mô phỏng (Simulation) với các phép thử trên các tín hiệu giả lập.

2.1.5 Pha kiểm tra

Mục tiêu của pha này là đánh giá khả năng thực thi của hệ thống sau khi đã hoàn thiện, thông thường ta thực hiện các bước sau:

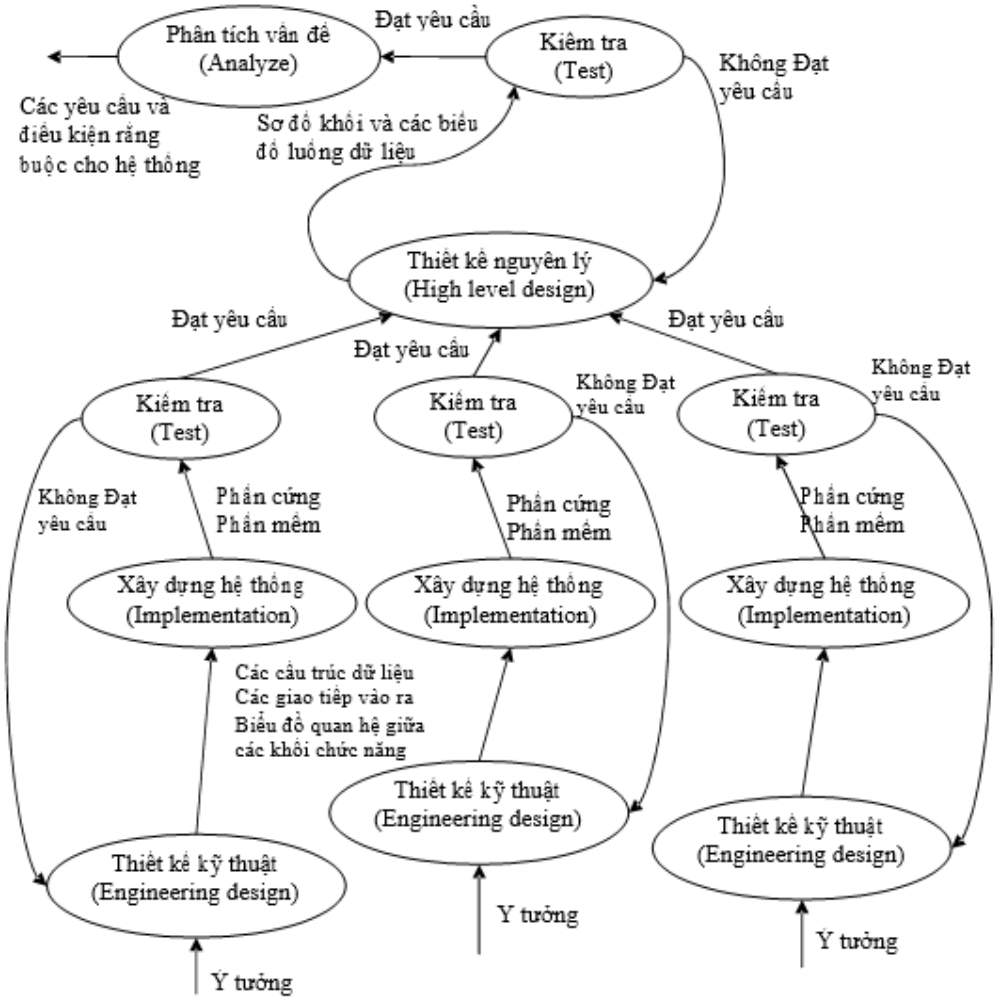
- Đầu tiên ta tiến hành gỡ lỗi (debug) và kiểm định cho từng chức năng cơ của hệ thống.
- Tiếp theo đánh giá khả năng thực thi của hệ thống dựa trên nhiều tiêu chí khác nhau như: Tốc độ, độ chính xác, tính ổn định,…

Ngày nay rất nhiều công ty trên nhiều lĩnh vực tuyển dụng nhân sự riêng cho pha này gọi là testing hay các tester. Nhiệm của của các Tester không chỉ là kiểm các tính năng của sản phẩm có phù hợp với các yêu cầu đã đề ra hay không, mà còn phải nghĩ ra các tình huống, các mẫu,… để tìm ra các lỗi mà người thiết kế hay người phát triển chưa phát hiện ra được.

Các thông tin kết quả của bước này quyết định một sản phẩm có thể được thương mại hóa hay không, hay phải bắt đầu một chu kỳ mới với pha đầu tiên là pha phân tích bao gồm các thông tin mới thu thập được từ bước này.

2.2. Quy trình Bottom-Up

Quy trình Bottom-Up trong thực tế thường áp dụng trong các bài toán chưa lựa chọn hay chưa tìm ra được giải pháp công nghệ. Mấu chốt của quy trình tập trung chủ yêu và quá trình thử nghiệm với hệ thống và tín hiệu thực, từ đó chọn ra giải pháp công nghệ và linh kiện phù hợp nhất cho bài toán. Sơ đồ tổng quát của quy trình như hình 2.4



Hình 2.4 Sơ đồ khối quy trình Bottom - Up

Quy trình Bottom-Up bắt đầu từ các ý tưởng đơn lẻ, sau đó xây dựng luôn thiết kế kỹ thuật. Như ta thấy quy trình hoàn toàn ngược so với Top-Down. Quy trình này thường áp dụng có các bài toán chưa lắm chắc về lời giải, người thiết kế mới chỉ có ý tưởng về một vấn đề nào đó và muốn tìm một giải pháp hoặc giải pháp tốt nhất để giải quyết vấn đề. Việc giải quyết các ý tưởng có thể 1 hoặc nhiều để có một sản phẩm hoàn chỉnh. Ở quy trình này ta cần chú ý có 2 khâu test nhằm kiểm định chính xác lại các thiết kế kỹ thuật và thiết kế nguyên lý trước khi lựa chọn 1 giải pháp tối ưu nhất.

Chính từ việc thí nghiệm và thiết kế thử hệ thống trước, sau đó mới có thể phân tích nguyên lý để chọn các đặc tính mới, rằng buộc mới cho một hệ thống mới. Với quy trình này khâu thiết kế kỹ thuật và Test sau khi xây dựng hệ thống là quan trọng nhất. Vì với Top-Down việc xây dựng một sản phẩm là theo nhu cầu của người dùng và môi trường đặt hệ thống. Còn với Bottom-Up có thể người ta còn chưa tìm ra cách để thiết kế ra sản phẩm đó, hoặc sản phẩm đó chưa hề có trên thị trường, khi đó cả người dùng và người thiết kế chưa thể có thông tin gì về các yêu cầu cho sản phẩm hay các đặt tính kỹ thuật của sản phẩm, vì vậy khâu thiết kế kỹ thuật và Test sau thực thi các kỹ sư phải tìm ra các đặt tính đó, nhằm xác định được các ưu việt cũng như các hạn chế của sản phẩm mới.

2.3 Đặc tả hệ thống

2.3.1 Khái niệm đặc tả (specification)

Kỹ thuật đặc tả nhằm định nghĩa một hệ thống, module hay sản phẩm cần phải làm gì, nhưng không mô tả nó phải làm thế nào. Nghĩa là đặc tả chỉ tập trung vào việc mô tả các chức năng, đầu vào và đầu ra, còn giải pháp để từ đầu vào có được đầu ra như thế nào thì không mô tả. Đồng thời nêu nên các tính chất của vấn đề đặt ra.

Như vậy bản chất ta sử dụng một công cụ để mô hình hoá một hệ thống thành các module, các luồng dữ liệu vào ra, các yêu cầu của tín hiệu hay dữ liệu, các thông tin trao đổi giữa các module,... Sao cho toàn bộ quy trình làm việc của hệ thống có thể đọc được thông qua đặc tả. Ở đây ta không quan tâm đến công nghệ của hệ thống, mà chỉ quan tâm đến chức năng, nhiệm vụ và các tính chất mà hệ thống phải có được.

Đặc tả có thể được tiến hành ở nhiều giai đoạn, nhiều lần trong tiến trình phát triển hệ thống như:

- Đặc tả yêu cầu (requirement specification): Nhằm mô tả đầy đủ và chi tiết những thống nhất giữa người sử dụng tương lai và người thiết kế về hệ thống. Đây là các thông tin cơ sở để người thiết kế thiết hệ thống của mình, nhiệm vụ người thiết kế pahỉ thiết kế đầy đủ và hoàn thiện theo yêu cầu đã thống nhất, người sử dụng tương lại được phép đỏi hỏi, giám sát việc thực thi các yêu cầu trong phạm vi đã thoả thuận.
- Đặc tả kiến trúc hệ thống (system architect specification): Nhằm mô tả sự thống nhất giữa người thiết kế và người cài đặt. Các đặt tả nhằm chỉ ra được các chức năng, tính chất và nhiệm vụ cho các thành phần hệ thống chuẩn bị được cài đặt. Việc xác định các yêu cầu từ đó chuyển thành bản thiết kế đã được đặc tả ởkhâu đặc tả yêu cầu, vì thế khâu này người cài đặt chỉ quan tầm đến bản thiết kế và những rằng buộc đã thống nhất với người thiết kế.
- Đặc tả Module (module specification): Mô tả sự thống nhất giữa người lập trình cài đặt module và người lập trình sử dụng module. Quá trình cài đặt hệ thống có thể được phân ra thành nhiều module được cài đặt một cách song song, hay kế thừa từ module có sẵn hay module đặt hàng, vì thế người cài đặt và người sử dụng module cần đặc tả chi tiết các chức năng, nhiệm vụ và đặc biệt các giao tiếp của hàm, để sao cho có thể ghép nối tương thích và hiệu quả module đó vào hệ thống.

2.3.2 Tại sao cần đặc tả

Việc đặc tả là rất cần thiết trước khi xây dựng một hệ thống bất kỳ trong lĩnh vực nào bởi vì:

- Đặc tả mang tính hợp đồng: Mô tả đầy đủ và chi tiết sự thống nhất giữa ngườisử dụng và người phát triển, chính vì thế đây là cơ pháp lý và khoa học để người phát triển thiết kế, còn người sử dụng thì giám sát và kiểm định.
- Giúp sản phẩm hợp thức hoá: Sản phẩm được phát triển theo các yêu cầu đã thoả thuận, vì thế sản phẩm đương nhiên sẽ đặt yêu cầu với người sử dụng.

- Đặc tả là công cụ để trao đổi: Ngôn ngữ đặc tả là một ngôn ngữ được quy định chung và thống nhất vì thế các đặc tả có thể được trao đổi giữa các người thiết kế với nhau. Đồng thời thông qua đó người sử dụng cũng có thể có cái nhìn tổng quát về sản phẩm.
- Tái sử dụng: Một đặc tả có thể được lưu giữ để phát triển tiếp hoặc được bổ sung nâng cấp về sau.

2.3.3 Phân loại các kỹ thuật đặc tả

a) Đặc tả phi hình thức: Sử dụng 1 trong các công cụ sau để mô tả hệ thống
   - Ngôn ngữ tự nhiên tự do
   - Ngôn ngữ tự nhiên có cấu trúc
   - Các ký hiệu đồ họa

b) Đặc tả nửa hình thức: Sử dụng hỗn hợp cả ngôn ngữ tự nhiên, các ký hiệu toán học và các ký hiệu đồ họa

c) Đặc tả hình thức: Sử dụng các ký hiệu toán học để mô hình hóa hệ thống, các ký hiệu toán học là các ký hiệu thống nhất chung có thể là:
   - Ngôn ngữ đặc tả
   - Ngôn ngữ lập trình

So sánh đặc tả hình thức và đặc tả phi hình thức:

Đặc tả hình thức	Đặc tả phi hình thức
• Chính xác (vì sử dụng các ký hiệu toán học đã được thống nhất trên thế giới). • Hợp thức hóa hình thức. • Sử dụng là công cụ trao đổi giữa các người thiết kế, tuy nhiên vì sử dụng các ký hiệu toán học chuyên ngành nên khó đọc và khó hiểu. • Chỉ dành cho những người có kiến thức chuyên môn nên khó sử dụng.	• Phương pháp sử dụng ngôn ngữ tự nhiên, tự do và bản địa nên dễ hiểu, dễ sử dụng. • Khả năng mềm dẻo cao, vì công cụ mô tả là ngôn ngữ dễ thay thế. • Thiếu sự chính xác và đầy đủ do hạn chế của ngôn ngữ và mang tính chủ quan. • Nhập nhằng.

2.3.4 Ứng dụng và ưu việt khi sử dụng đặc tả

a) Ứng dụng

- Đặc tả thường sử dụng trong giai đoàn đầu của tiến trình phát triển sản phẩm

- Nhằm hạn chế lỗi trong quá trình phát triển phần mềm, thông qua việc đặc tả chi tiết và đầy đủ các chức năng, nhiệm vụ và tính chất cho tất cả các thành phần của sản phẩm
- Kỹ thuật đặc tả thường dùng khi phát triển các hệ thống
  - Hệ thống điều khiển
  - Hệ thống nhúng
  - Hệ thống thời gian thực

b) Ưu việt khi sử dụng đặc tả

Ưu điểm khi sử dụng đặc tả được thể hiện rõ nhất qua qua chi phí phát triển hệ thống, ta có thể thấy rõ qua biểu đồ sau:

Hình 2.5 Biểu đồ so sánh chi phí phát triển hệ thống sử dụng đặc tả và không sử dụng

Qua biểu đồ ta dễ dành nhận ra tổng chi phí khi sử dụng đặc tả giảm đi rất nhiều, điều nay sẽ là cho giá thành sản phẩm đạt được tính cạnh tranh.

2.3.5 Phương pháp đặc tả sử dụng "Máy trạng thái hữu hạn FSM(Finite state machine)"

Trong thực tế có một số phương pháp đặc tả như:

- Máy trạng thái hữu hạn
- Mạng Petri (thường dùng để mô tả các hệ thống phức tạp không đồng bộ, có nhiều khâu làm việc song song)
- Điều kiện trước sau (thường dùng đặc tả cho các hàm hoặc module)

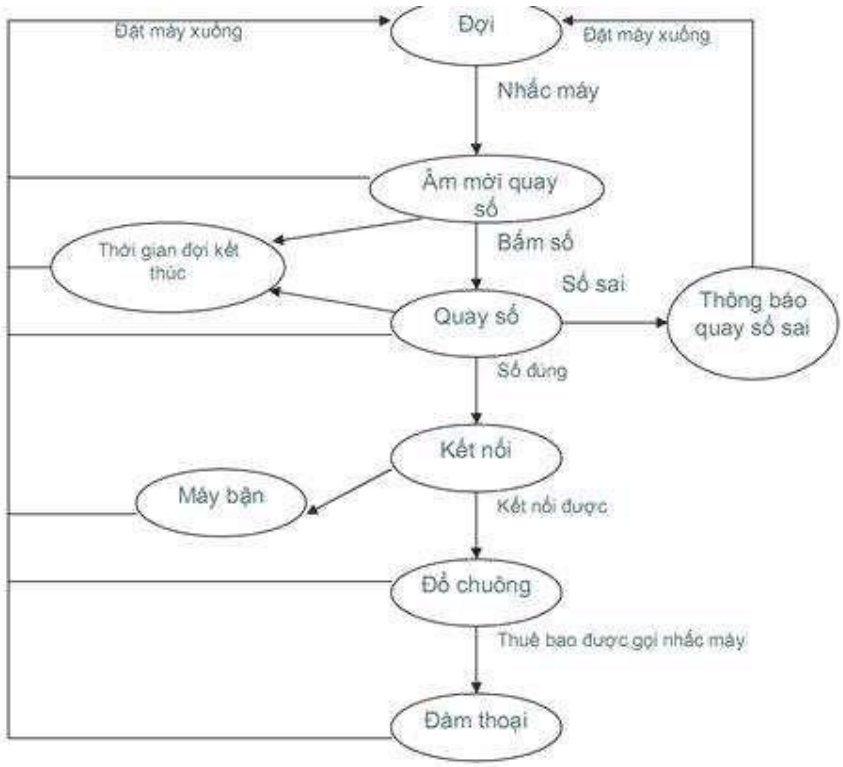
- Kiểu trừu tượng (mô tả dữ liệu và các thao tác trên dữ liệu đó ở một mức trừu tượng độc lập với cách cài đặt dữ liệu bởi ngôn ngữ lập trình)

Trong phạp vi này bài giảng chỉ tập trung trình bày về kỹ thuật đặc tả sử dụng máy trạng thái hữu hạn, do tính đơn giản và phù hời với các hệ thống nhỏ có tính moudle và tuần tự cao.

Phương pháp máy trạng thái hữu hạn:

- Phương pháp mô tả các luông điều khiển trong hệ thống
- Biểu diễn hệ thống dưới dạng đồ thị
- Đồ thị bao gồm:
  - Nút: mỗi nút là một trạng thái S của hệ thống
  - Nhãn: Mô tả dữ liệu đầu vào, là nhãn của một cung
  - Cung: Một chuyển tiếp T : S x I -> S (chuyển tiếp mô tả một sự biến đổi trạng thái khi một trạng có dữ liệu vào)

Ví dụ 1:



Hình 2.6 Đặc tả cách thức làm việc của một máy điện thoại sử dụng máy trạng thái hữu hạn

2.4 Các phương pháp biểu diễn thuật toán

Khi chứng minh hoặc giải một bài toán trong toán học, ta thường dùng những ngôn từ toán học như : "ta có", "điều phải chứng minh", "giả thuyết", ... và sử dụng những phép suy luận toán học như phép suy ra, tương đương, ...Thuật toán là một phương pháp thể hiện lời giải bài toán nên cũng phải tuân theo một số quy tắc nhất định.

Để có thể truyền đạt thuật toán cho người khác hay chuyển thuật toán thành chương trình máy tính, ta phải có phương pháp biểu diễn thuật toán. Có 3 phương pháp biểu diễn thuật toán :

1. Dùng ngôn ngữ tự nhiên.
2. Dùng lưu đồ-sơ đồ khối (flowchart).
3. Dùng mã giả (pseudocode)

2.4.1 Ngôn ngữ tự nhiên

Trong cách biểu diễn thuật toán theo ngôn ngữ tự nhiên, người ta sử dụng ngôn ngữ thường ngày để liệt kê các bước của thuật toán . Phương pháp biểu diễn này không yêu cầu người viết thuật toán cũng như người đọc thuật toán phải nắm các quy tắc. Tuy vậy, cách biểu diễn này thường dài dòng, không thể hiện rõ cấu trúc của thuật toán, đôi lúc gây hiểu lầm hoặc khó hiểu cho người đọc. Gần như không có một quy tắc cố định nào trong việc thể hiện thuật toán bằng ngôn ngữ tự nhiên. Tuy vậy, để dễ đọc, ta nên viết các bước con lùi vào bên phải và đánh số bước theo quy tắc phân cấp như 1, 1.1, 1.1.1

Ví dụ: Biểu diễn thuật toán giải phương trình bậc hai dùng ngôn ngữ tự nhiên (không xét đến nghiệm phức)

Đầu vào: Phương trình $ax^2 + bx + c = 0$

Đầu ra: Nghiệm

Bước 1: Xác định hệ số a,b,c

1.1. Nếu a≠0 chuyển đến bước 2

1.2. Nếu a=0 không thực hiện vì la phương trình bậc nhất

Bước 2: Tính $\Delta = b^2 - 4ac$

Bước 3: Kiểm tra Δ

3.1. Nếu Δ>0 đến bước 4

3.2. Nếu Δ=0 đến bước 5

3.3. Nếu Δ<0 báo phương trình vô nghiệm, kết thúc thuật toán

Bước 4: Báo phương trình có 2 nghiệm $x = \frac{-b \pm \sqrt{\Delta}}{2a}$, kết thúc thuật toán

Bước 5: Phương trình có nghiệm kép $x_1=x_2=-b/2a$, kết thúc thuật toán

2.4.2 Dùng lưu đồ

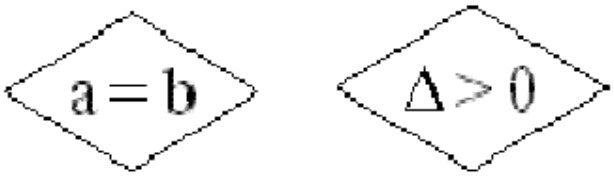
Lưu đồ hay sơ đồ khối là một công cụ trực quan để diễn đạt các thuật toán. Biểu diễn thuật toán bằng lưu đồ sẽ giúp người đọc theo dõi được sự phân cấp các trường hợp và quá trình xử lý của thuật toán. Phương pháp lưu đồ thường được dùng trong những thuật toán có tính rắc rối, khó theo dõi được quá trình xử lý.

Để biểu diễn thuật toán theo sơ đồ khối, ta phải phân biệt hai loại thao tác. Một thao

tác là thao tác chọn lựa dựa theo một điều kiện nào đó. Chẳng hạn : thao tác "nếu a = b thì thực hiện thao tác B2, ngược lại thực hiện B4" là thao tác chọn lựa. Các thao tác còn lại không thuộc loại chọn lựa được xếp vào loại hành động. Chẳng hạn, "Chọn một hộp bất kỳ và để lên đĩa cân còn trống." là một thao tác thuộc loại hành động.

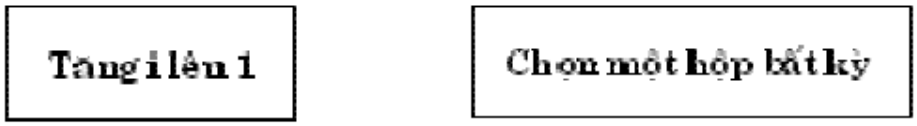
a) Thao tác chọn lựa (hay còn gọi khối điều kiện)

Thao tác chọn lựa được biểu diễn bằng một hình thoi, bên trong chứa biểu thức điều kiện.



b) Thao tác xử lý (process)

Thao tác xử lý được biểu diễn bằng một hình chữ nhật, bên trong chứa nội dung xử lý.

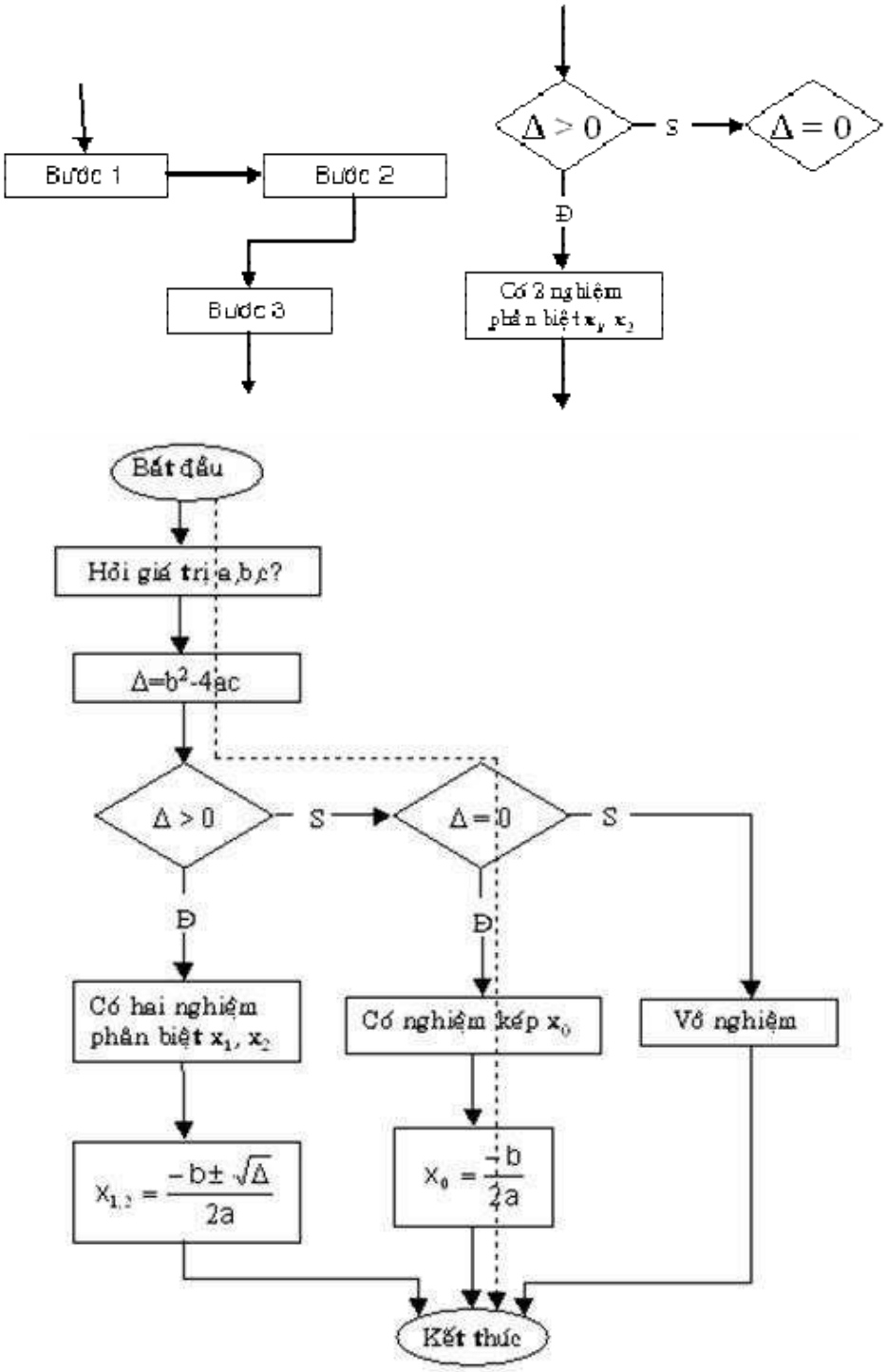


c) Đường đi (route)

Khi dùng ngôn ngữ tự nhiên, ta mặc định hiểu rằng quá trình thực hiện sẽ lần lượt đi từ bước trước đến bước sau (trừ khi có yêu cầu nhảy sang bước khác). Trong ngôn ngữ lưu đồ, do thể hiện các bước bằng hình vẽ và có thể đặt các hình vẽ này ở vị trí bất kỳ nên ta phải có phương pháp để thể hiện trình tự thực hiện các thao tác.

Hai bước kế tiếp nhau được nối bằng một cung, trên cung có mũi tên để chỉ hướng thực hiện. Chẳng hạn trong hình dưới, trình tự thực hiện sẽ là B1, B2, B3.

Từ thao tác chọn lựa có thể có đến hai hướng đi, một hướng ứng với điều kiện thỏa và một hướng ứng với điều kiện không thỏa. Do vậy, ta dùng hai cung xuất phát từ các đỉnh hình thoi, trên mỗi cung có ký hiệu Đ/Đúng/Y/Yes để chỉ hướng đi ứng với điều kiện thỏa và ký hiệu S/Sai/N/No để chỉ hướng đi ứng với điều kiện không thỏa.



Hình 2.7 Lưu đồ thuật toán giải phương trình bậc 2

d) Điểm cuối (terminator)

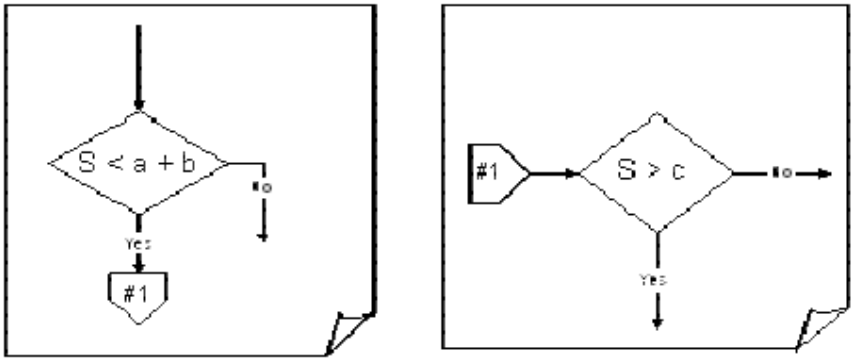
Điểm cuối là điểm khởi đầu và kết thúc của thuật toán, được biểu diễn bằng hình ovan, bên trong có ghi chữ bắt đầu/start/begin hoặc kết thúc/end. Điểm cuối chỉ có cung đi ra (điểm khởi đầu) hoặc cung đi vào (điểm kết thúc). Xem lưu đồ thuật toán giải phương trình bậc hai ở trên để thấy cách sử dụng của điểm cuối.

e) Điểm nối (connector)

Điểm nối được dùng để nối các phần khác nhau của một lưu đồ lại với nhau. Bên trong điểm nối, ta đặt một ký hiệu để biết sự liên hệ giữa các điểm nối.

f) Điểm nối sang trang (off-page connector)

Tương tự như điểm nối, nhưng điểm nối sang trang được dùng khi lưu đồ quá lớn, phải vẽ trên nhiều trang. Bên trong điểm nối sang trang ta cũng đặt một ký hiệu để biết được sự liên hệ giữa điểm nối của các trang.



Ở trên chỉ là các ký hiệu cơ bản và thường được dùng nhất. Trong thực tế, lưu đồ còn có nhiều ký hiệu khác nhưng thường chỉ dùng trong những lưu đồ lớn và phức tạp. Đối với các thuật toán trong cuốn sách này, ta chỉ cần sử dụng các ký hiệu trên là đủ.

2.4.3 Mã giả

Tuy sơ đồ khối thể hiện rõ quá trình xử lý và sự phân cấp các trường hợp của thuật toán nhưng lại cồng kềnh. Để mô tả một thuật toán nhỏ ta phải dùng một không gian rất lớn. Hơn nữa, lưu đồ chỉ phân biệt hai thao tác là rẽ nhánh (chọn lựa có điều kiện) và xử lý mà trong thực tế, các thuật toán còn có thêm các thao tác lặp.

Khi thể hiện thuật toán bằng mã giả, ta sẽ vay mượn các cú pháp của một ngôn ngữ lập trình nào đó để thể hiện thuật toán. Tất nhiên, mọi ngôn ngữ lập trình đều có những thao tác cơ bản là xử lý, rẽ nhánh và lặp. Dùng mã giả vừa tận dụng được các khái niệm trong ngôn ngữ lập trình, vừa giúp người cài đặt dễ dàng nắm bắt nội dung thuật toán. Tất nhiên là trong mã giả ta vẫn dùng một phần ngôn ngữ tự nhiên. Một khi đã vay mượn cú pháp và khái niệm của ngôn ngữ lập trình thì chắc chắn mã giả sẽ bị phụ thuộc vào ngôn ngữ lập trình đó.

Ví dụ: Một đoạn mã giả của thuật toán giải phương trình bậc hai

```
if Delta > 0 then
    begin
        x1=(-b-sqrt(delta))/(2*a)
        x2=(-b+sqrt(delta))/(2*a)
        xuất kết quả : phương trình có hai nghiệm là x1 và x2
        end
```

else

if delta = 0 then

xuất kết quả : phương trình có nghiệm kép là -b/(2*a)

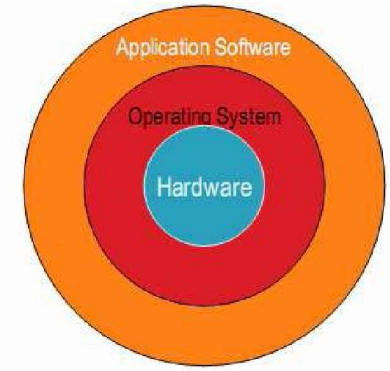
else {trường hợp delta < 0 }xuất kết quả : phương trình vô nghiệm

Chương 3: CẤU TRÚC PHẦN CỨNG

3.1 Cấu trúc tổng quát của hệ thống nhúng

3.1.1 Kiến trúc cơ bản

Kiến trúc cơ bản của hệ thống nhúng được miêu tả ở hình 3.1, như vậy một hệ thống đầy đủ về cơ bản có 3 thành phần:



Hình 3.1 Kiến trúc cơ bản của một hệ thống nhúng

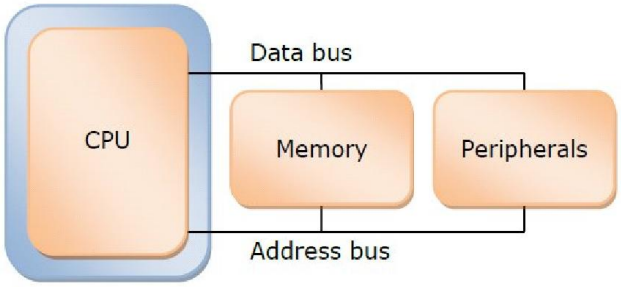
1. Chương trình ứng dụng: là các chương trình chạy trên nền hệ thống nhúng. Người sử dụng tác động trực tiếp trên các phần mềm này.

2. Hệ điều hành: Hệ điều hành giúp cho việc phát triển phần mềm và phần cứng có thể tách biệt trên 2 công nghệ không cần liên quan đến nhau nhưng vẫn có thể kết hợp để có một sản phẩm tốt nhất. Cũng như máy tính hệ điều hành giúp người khác thác hệ thống được hiệu quả và dễ dàng hơn, hệ thống nhúng có hệ điều hành thường linh hoạt và mềm dẻo hơn.

3. Phần cứng: Đây là thành phần cốt lõi không thể thiếu, thành phần phần cứng quan trọng nhất là CPU, các thành phần khác có thể tùy biến theo từng ứng dụng cụ thể.

3.1.2 Cấu trúc phần cứng

Cấu trúc tổng quát thông dụng nhất của một vi xử lý/vi điều khiển nhúng được đưa ra ở hình 3.2.



Hình 3.2 Cấu trúc thông dụng của một VXL/VĐK nhúng

VXL/VĐK nhúng là một chủng loại rất điển hình và đang được sử dụng rất phổ biến hiện này. Chúng được ra đời và sử dụng theo sự phát triển của các Chip xử lý ứng dụng cho máy tính. Vì đối tượng ứng dụng là các thiết bị nhúng nên cấu trúc cũng được thay đổi theo để đáp ứng các ứng dụng. Hiện nay chúng ta có thể thấy các họ vixử lý điều khiển của rất nhiều các nhà chế tạo cung cấp như, Intel, Atmel, Motorola, Infineon. Về cấu trúc, chúng cũng tương tự như các Chíp xử lý phát triển cho PC nhưng ở mức độ đơn giản hơn nhiều về công năng và tài nguyên. Phổ biến vẫn là các Chip có độ rộng bus dữ liệu là 8bit, 16bit, 32bit. Về bản chất cấu trúc, Chip vi điều khiển là chip vi xử lý được tích hợp thêm các ngoại vi. Các ngoại vi thường là các khối chức năng ngoại vi thông dụng như bộ định thời gian, bộ đếm, bộ chuyển đổi A/D, giao diện song song, nối tiếp…Mức độ tích hợp ngoại vi cũng khác nhau tuỳ thuộc vào mục đích ứng dụng sẽ có thể tìm được Chip phù hợp. Thực tế với các ứng dụng yêu cầu độ tích hợp cao thì sẽ sử dụng giải pháp tích hợp trên chip, nếu không thì hầu hết các Chip đều cung cấp giải pháp để mở rộng ngoại vi đáp ứng cho một số lượng ứng dụng rộng và mềm dẻo.

3.2 Một số nền tảng phần cứng thông dụng

Hiện trên thị trường có rất nhiều nhà sản xuất phát triển nhiều loại chip VXL/VĐK nhúng khác nhau, mỗi loại đều có những tính năng và ưu việt riêng. Trong thực tế mỗi dòng chip ra đời được thiết kế chuyên biệt cho một phân khúc ứng dụng nào đấy, vì thế không có một nguyên tắc chung hay một loại VXL/VĐK chung cho mọi bài toán. Tuỳ vào khả năng và nhiệm vụ cụ thể của hệ thống mà ta chọn lựa loại chip phù hợp. Trong một số bài toán và một số dòng chip thì sự phân biệt giữa các chip là không rõ dàng, nghĩa là có thể việc sử dụng các chip khác nhau nhưng mạng lại hiệu qua như nhau.

Ngoài vi xử lý của máy tính, ta có thể phân loại ra hai dòng chip nhúng là vi xử lý nhúng và vi điều khiển nhúng. Vi xử lý nhúng được sử dụng rộng rãi với thị phần lớn nhất hiện nay là ARM, còn vi điều khiển thì có nhiều loại như AVR, PIC, 8051,... Tuy nhiên trong phạm vi bài giảng này chi trình bày tóm lược một số loại chip là PIC, 8051, ARM, …

3.2.1 Vi điều khiển Atmega8

3.2.1.1 Đặc điểm

Vi điều khiển Atmega8 của hãng ATMEL là một loại vi điều khiển AVR mới với kiến trúc rất phức tạp. Atmega 8 là bộ vi điều khiển RISC 8 bit tiêu thụ năng lượng nhưng đạt hiệu suất rất cao, dựa trên kiến trúc RISC AVR. Bằng việc thực hiện các lệnh trong một chu kỳ xung nhịp, Atmega8 đạt được tốc độ xử lý dữ liệu lên đến 1 triệu lệnh/giâyở tần số 1MHz. Atmega8 còn cho phép người thiết kế hệ thống tối ưu hoá mức độ tiêu thụ năng lượng mà vẫn đảm bảo tốc độ xử lý. Atmega 8 đã tích hợp đầy đủ các tính năng như bộ chuyển đổi ADC 10bit, bộ so sánh, bộ truyền nhận nối tiếp, bộ định thời, bộ đếm thời gian thực, bộ điều chế độ rộng xung…Do đó ta phải nghiên cứu và khai thác triệt để các tính năng này để ứng dụng hiệu quả vào những mạch trong thực tế. Atmega8 sử dụng kiến trúc RISC (Reduced Instruction Set Computer) AVR.

 Atmega8 với kiến trúc RISC có chỉ tiêu chất lượng cao và tiêu thụ năng lượng ít:

- 130 lệnh hầu hết được thực hiện trong một chu kỳ xung nhịp.

- 32 thanh ghi làm việc đa năng.

- Tốc độ xử lý lệnh lên đến 16 triệu lệnh/giây ở tần số 16MHz.

 Bộ nhớ dữ liệu và bộ nhớ chương trình không tự mất dữ liệu:

- 8K byte bộ nhớ Flash lập trình được ngay trên hệ thống, có thể nạp xoá 10000 lần.

- 512 byte bộ nhớ EEFROM lập trình được ngay trên hệ thống, có thể ghi xóa 100000 lần.

- 1K byte bộ nhớ SRAM.

- Có thể giao tiếp với 8K byte bộ nhớ ngoài.

- Khóa bảo mật phần mềm lập trình được.

- Giao diện nối tiếp SPI để lập trình ngay trên hệ thống.

 Các tính năng ngoại vi:

- Hai bộ đếm/ bộ định thời 8 bit với chế độ so sánh và chia tần số tách biệt.

- Một bộ định thời 16 bit với chế độ so sánh, chia tần số tách biệt và chế độ bắt mẫu (Capture Mode).

- Bộ đếm thời gian thực (RTC) với bộ dao động tách biệt.

- Bộ điều chế độ rộng xung PWM 8 bit.

- Bộ biến đổi ADC bên trong 8 kênh 10 bit.

- 2 bộ USART nối tiếp lập trình được.

- Bộ định thời Watchdog lập trình được với bộ dao động trên chip.

- Một bộ so sánh Analog.

 Các tính năng vi điều khiển đặc biệt:

- Có mạch power - on reset và có thể reset bằng phần mềm.

- Các nguồn ngắt ngoài và trong.

- Có 5chế độ ngủ: nghỉ (Idle). Tiết kiệm năng lượng (power save) và power down, ADC Noise Reduction, Standby.

- Tần số làm việc có thể thay đổi được bằng phần mềm.

 Vào ra và các cách đóng vỏ

- 23 đường vào ra lập trình được.

- 32 chân dán kiểu vỏ vuông (TQFP)

 Điện thế làm việc:

- VCC = 2,7V đến 5,5V đối với Atmega8L.

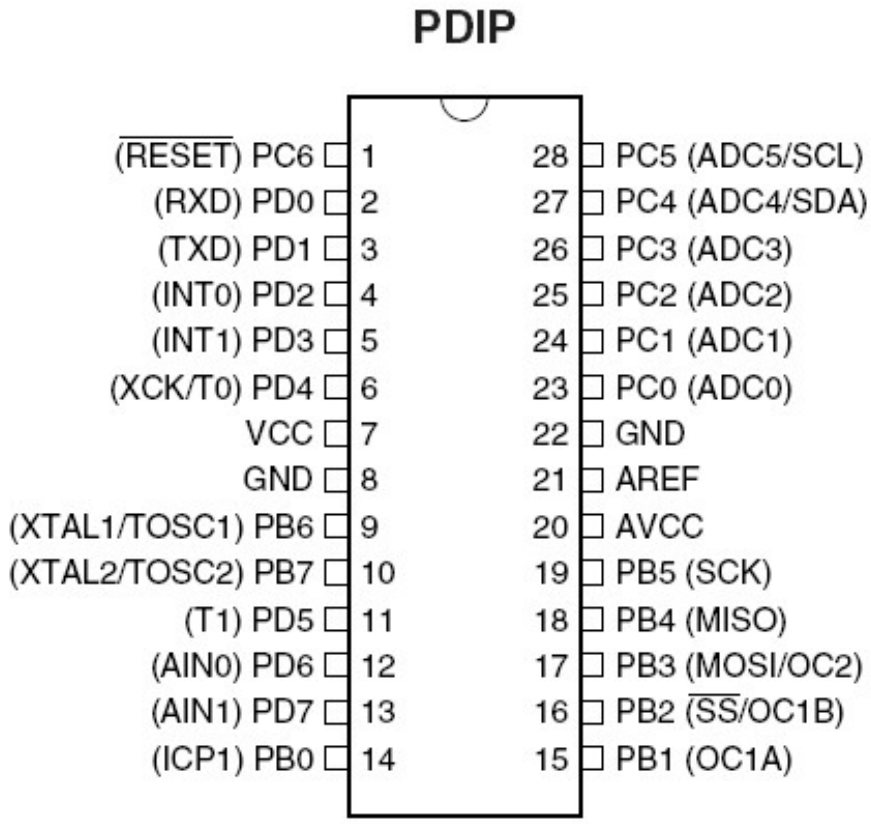
- VCC = 4,5V đến 5,5V đối với Atmega8.

 Vùng tốc độ làm việc:

- 0 đến 8 MHz đối với Atmega8L.

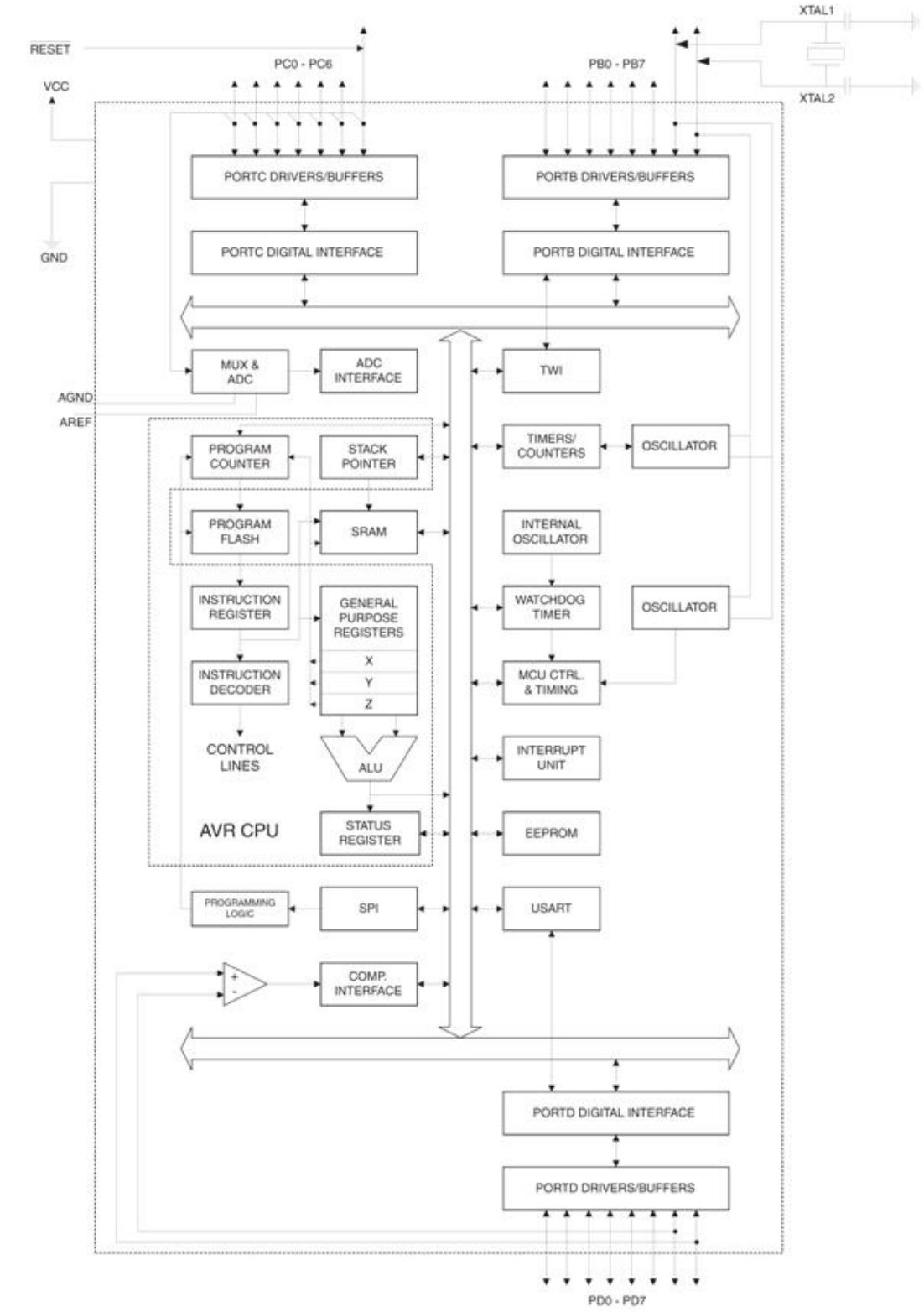
- 0 đến 16 MHz đối với Atmega8.

3.2.1.2 Sơ đồ chân Atmega8



Hình 3.3 Sơ đồ chân Atmega8

3.2.1.3 Sơ đồ khối của Atmega8



Hình 3.4 Sơ đồ khối vi điều khiển AVR Atmega8

3.2.1.4 Mô tả chức năng các chân Atmega8

 VCC: Điện áp nguồn nuôi.

 GND: Đất.

 Port B (PB0…PB7)- 6

Port B là port I/O 8 bit với điện trở kéo lên ở bên trong, cung cấp dòng điện 40mA có thể điều khiển trực tiếp led đơn.

- Khi các chân Port B là các lối vào được đặt xuống mức thấp từ bên ngoài, chúng sẽ là nguồn dòng nếu như các điện trở nối lên nguồn dương được kích hoạt. Các chân này sẽ ở trạng thái tổng trở cao khi tín hiệu Reset ở mức tích cực hoặc ngay cả khi không có dao động.

 Port C (PC0…PC6)

- Port C là port I/O 8 bit với điện trở kéo lên ở bên trong, cung cấp dòng điện 40mA có thể điều khiển trực tiếp led đơn.

- Khi các chân Port C là các lối vào được đặt xuống mức thấp từ bên ngoài, chúng sẽ là nguồn dòng nếu như các điện trở nối lên nguồn dương được kích hoạt. Các chân này sẽ ở trạng thái tổng trở cao khi tín hiệu Reset ở mức tích cực hoặc ngay cả khi không có dao động.

- Port C cũng đóng vai trò như 8 đường địa chỉ cao từ A8 đến A15 khi kết nối bộ nhớ SRAM bên ngoài.

 Port D (PD0…PD7)

- Port D là port I/O 8 bit với điện trở kéo lên ở bên trong, cung cấp dòng điện 40mA có thể điều khiển trực tiếp LED đơn.

- Khi các chân Port D là các lối vào được đặt xuống mức thấp từ bên ngoài, chúng sẽ là nguồn dòng nếu như các điện trở nối lên nguồn dương được kích hoạt. Các chân này sẽ ở trạng thái tổng trở cao khi tín hiệu Reset ở mức tích cực hoặc ngay cả khi không có dao động.

 Reset: Ngõ vào được đặt lại. ATmega8 sẽ được đặt lại khi chân này ở mức thấp trong hơn 50ns hoặc ngay cả khi không có tín hiệu xung clock. Các xung ngắn hơn không tạo ra tín hiệu đặt lại.

 AVCC: Cung cấp nguồn cho Port C và bộ chuyển đổi ADC hoạt động. Ngay khi không sử dụng bộ chuyển đổi ADC thì chân AVCC vẫn phải đươc kết nối tới nguồn VCC.

 AREF: Đây là chân điều chỉnh điện áp tham chiếu cho chuyển đổi A/D.

 XTAL1: Ngõ vào bộ khuếch đại đảo và ngõ vào mạch tạo xung nhịp bên ngoài.

 XTAL2: Ngõ ra bộ khuếch đại đảo.

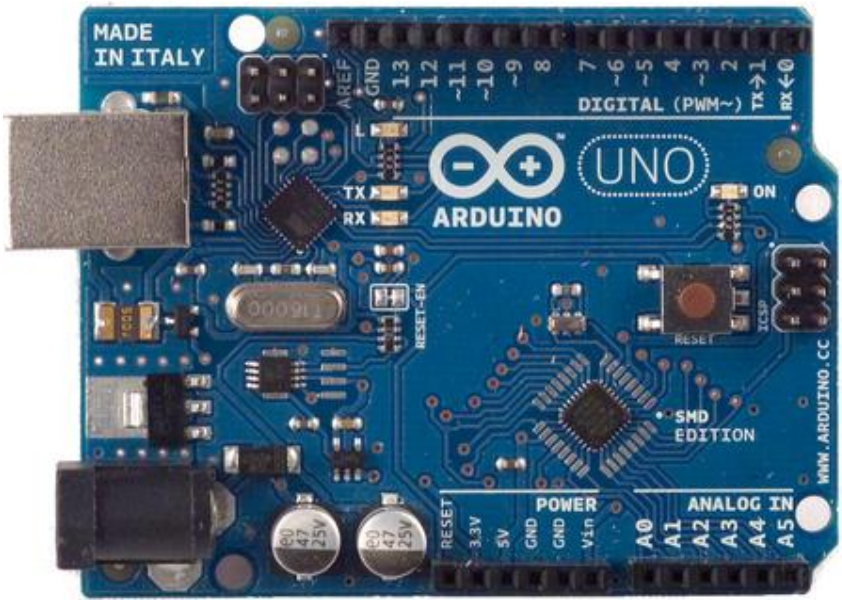
 Bộ tạo dao động thạch anh :

- XTAL1 và XTAL2 lần lượt là lối vào và lối ra của một bộ khuếch đại đảo, bộ khuếch đại này được bố trí để làm bộ tạo dao động trên chip

- Để điều khiển được bộ Vi Điều Khiển từ một nguồn xung nhịp bên ngoài, chân XTAL2 để không, chân XTAL1 được nối với tín hiệu dao động bên ngoài.

3.2.2 Kit Arduino Uno R3

Hiện dòng mạch Arduino UNO đã phát triển tới thế hệ thứ 3 (R3).



Hình 3.5 Kit Arduino Uno R3

Một vài thông số của Arduino UNO R3

Vi điều khiển	ATmega328 (họ 8bit)
Điện áp hoạt động	5V – DC (chỉ được cấp qua cổng USB)
Tần số hoạt động	16 MHz
Dòng tiêu thụ	30mA
Điện áp vào khuyên dùng	7-12V – DC
Điện áp vào giới hạn	6-20V – DC
Số chân Digital I/O	14 (6 chân PWM)
Số chân Analog	6 (độ phân giải 10bit)
Dòng tối đa trên mỗi chân I/O	30 mA
Dòng ra tối đa (5V)	500 mA
Dòng ra tối đa (3.3V)	50 mA

Bộ nhớ flash	32 KB (ATmega328) với 0.5KB dùng bởi bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)

Vi điều khiển

Arduino UNO có thể sử dụng 3 vi điều khiển họ 8bit AVR là ATmega8, ATmega168, ATmega328. Bộ não này có thể xử lí những tác vụ đơn giản như điều khiển đèn LED nhấp nháy, xử lí tín hiệu cho xe điều khiển từ xa, làm một trạm đo nhiệt độ - độ ẩm và hiển thị lên màn hình LCD,… hay những ứng dụng nhúng khác.

Thiết kế tiêu chuẩn của Arduino UNO sử dụng vi điều khiển ATmega328. Tuy nhiên nếu yêu cầu phần cứng không cao có thể sử dụng các loại vi điều khiển khác có chức năng tương đương nhưng rẻ hơn như ATmega8 (bộ nhớ flash 8KB) hoặc ATmega168 (bộ nhớ flash 16KB).

Năng lượng

Arduino UNO có thể được cấp nguồn 5V thông qua cổng USB hoặc cấp nguồn ngoài với điện áp khuyên dùng là 7-12V DC và giới hạn là 6-20V. Thường thì cấp nguồn bằng pin vuông 9V là hợp lí nhất nếu không có sẵn nguồn từ cổng USB. Nếu cấp nguồn vượt quá ngưỡng giới hạn trên, sẽ làm hỏng Arduino UNO.

Các chân năng lượng

- **GND (Ground)**: cực âm của nguồn điện cấp cho Arduino UNO. Khi dùng các thiết bị sử dụng những nguồn điện riêng biệt thì những chân này phải được nối với nhau.
- **5V**: cấp điện áp 5V đầu ra. Dòng tối đa cho phép ở chân này là 500mA.
- **3.3V**: cấp điện áp 3.3V đầu ra. Dòng tối đa cho phép ở chân này là 50mA.
- **Vin (Voltage Input)**: để cấp nguồn ngoài cho Arduino UNO, nối cực dương của nguồn với chân này và cực âm của nguồn với chân GND.
- **IOREF**: điện áp hoạt động của vi điều khiển trên Arduino UNO có thể được đo ở chân này. Và dĩ nhiên nó luôn là 5V. Mặc dù vậy ta không được lấy nguồn 5V từ chân này để sử dụng bởi chức năng của nó không phải là cấp nguồn.
- **RESET**: việc nhấn nút Reset trên board để reset vi điều khiển tương đương với việc chân RESET được nối với GND qua 1 điện trở 10KΩ.

Lưu ý:

- Arduino UNO không có bảo vệ cắm ngược nguồn vào. Do đó phải hết sức cẩn thận, kiểm tra các cực âm – dương của nguồn trước khi cấp cho Arduino UNO.
- Các chân 3.3V và 5V trên Arduino là các chân dùng để cấp nguồn ra cho các thiết bị khác, không phải là các chân cấp nguồn vào. Việc cấp nguồn sai vị trí có thể làm hỏng board.

- Cấp nguồn ngoài không qua cổng USB cho Arduino UNO với điện áp dưới 6V có thể làm hỏng board.
- Cấp điện áp trên 13V vào chân RESET trên board có thể làm hỏng vi điều khiển ATmega328.
- Cường độ dòng điện vào/ra ở tất cả các chân Digital và Analog của Arduino UNO nếu vượt quá 200mA sẽ làm hỏng vi điều khiển.
- Cấp điệp áp trên 5.5V vào các chân Digital hoặc Analog của Arduino UNO sẽ làm hỏng vi điều khiển.
- Cường độ dòng điện qua một chân Digital hoặc Analog bất kì của Arduino UNO vượt quá 40mA sẽ làm hỏng vi điều khiển. Do đó nếu không dùng để truyền nhận dữ liệu, phải mắc một điện trở hạn dòng.

Bộ nhớ

Vi điều khiển Atmega328 tiêu chuẩn cung cấp cho người dùng:

- **32KB bộ nhớ Flash**: những đoạn lệnh lập trình sẽ được lưu trữ trong bộ nhớ Flash của vi điều khiển. Thường thì sẽ có khoảng vài KB trong số này sẽ được dùng cho bootloader nhưng với các ứng dụng nhỏ hiếm khi nào cần quá 20KB bộ nhớ này.
- **2KB cho SRAM** (**S**tatic **R**andom **A**ccess **M**emory): giá trị các biến khai báo khi lập trình sẽ lưu ở đây. Khai báo càng nhiều biến thì càng cần nhiều bộ nhớ RAM. Khi mất điện, dữ liệu trên SRAM sẽ bị mất.
- **1KB cho EEPROM** (**E**lectrically **E**raseble **P**rogrammable **R**ead **O**nly **M**emory): đây giống như một chiếc ổ cứng mini – nơi có thể đọc và ghi dữ liệu của mình vào đây mà không bị mất dữ liệu khi cúp điện giống như dữ liệu trên SRAM.

Các cổng vào/ra

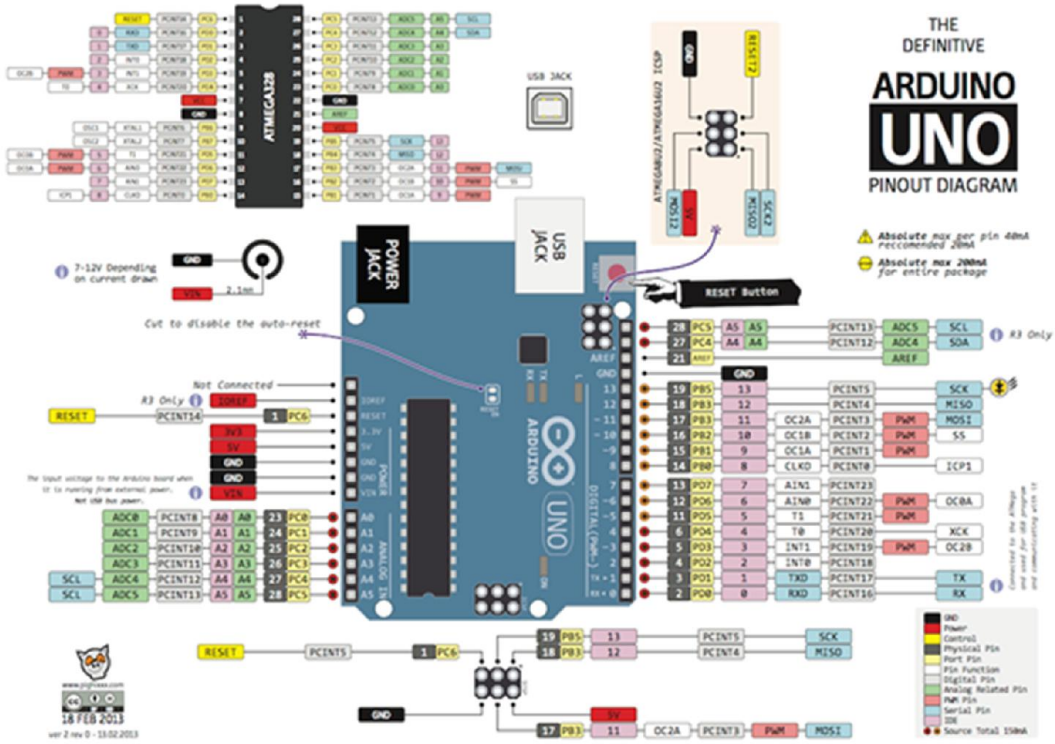
Arduino UNO có 14 chân digital dùng để đọc hoặc xuất tín hiệu. Chúng chỉ có 2 mức điện áp là 0V và 5V với dòng vào/ra tối đa trên mỗi chân là 40mA. Ở mỗi chân đều có các điện trở pull-up từ được cài đặt ngay trong vi điều khiển ATmega328 (mặc định thì các điện trở này không được kết nối).

Một số chân digital có các chức năng đặc biệt như sau:

- **2 chân Serial**: 0 (RX) và 1 (TX): dùng để gửi (transmit – TX) và nhận (receive – RX) dữ liệu TTL Serial. Arduino Uno có thể giao tiếp với thiết bị khác thông qua 2 chân này. Nếu không cần giao tiếp Serial, không nên sử dụng 2 chân này nếu không cần thiết
- **Chân PWM (~): 3, 5, 6, 9, 10, và 11**: cho phép xuất ra xung PWM với độ phân giải 8bit (giá trị từ 0 → 2^8-1 tương ứng với 0V → 5V) bằng hàm analogWrite(). Nói một cách đơn giản, có thể điều chỉnh được điện áp ra ở chân này từ mức 0V đến 5V thay vì chỉ cố định ở mức 0V và 5V như những chân khác.
- **Chân giao tiếp SPI:** 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Ngoài các chức năng thông thường, 4 chân này còn dùng để truyền phát dữ liệu bằng giao thức SPI với các thiết bị khác.

- **LED 13**: trên Arduino UNO có 1 đèn led màu cam (kí hiệu chữ L). Khi bấm nút Reset, sẽ thấy đèn này nhấp nháy để báo hiệu. Nó được nối với chân số 13. Khi chân này được người dùng sử dụng, LED sẽ sáng.
- Arduino UNO có 6 chân analog (A0 → A5) cung cấp độ phân giải tín hiệu 10bit (0 → 2^{10}-1) để đọc giá trị điện áp trong khoảng 0V → 5V. Với chân **AREF** trên board, có thể để đưa vào điện áp tham chiếu khi sử dụng các chân analog. Tức là nếu cấp điện áp 2.5V vào chân này thì có thể dùng các chân analog để đo điện áp trong khoảng từ 0V → 2.5V với độ phân giải vẫn là 10bit.
- Đặc biệt, Arduino UNO có 2 chân A4 (SDA) và A5 (SCL) hỗ trợ giao tiếp I2C/TWI với các thiết bị khác.

Sơ đồ khối các chân vào ra trên Arduino Uno

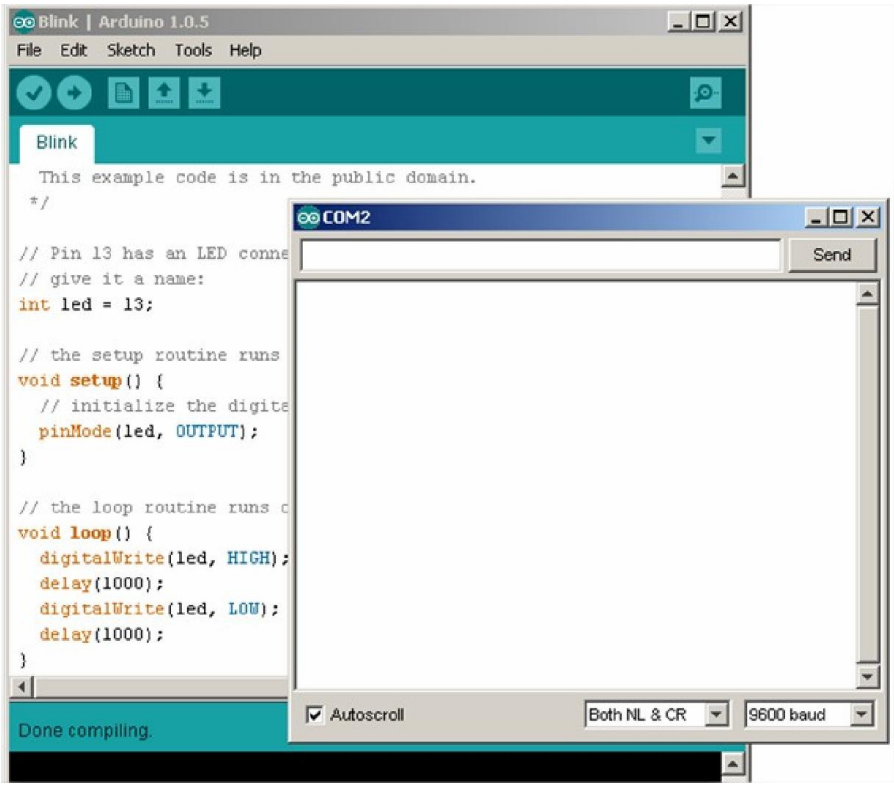


Hình 3.6 Sơ đồ khối các chân vào ra trên Kit Arduino Uno

Lập trình cho Arduino

Các thiết bị dựa trên nền tảng Arduino được lập trình bằng ngôn riêng. Ngôn ngữ này dựa trên ngôn ngữ Wiring được viết cho phần cứng nói chung. Và Wiring lại là một biến thể của C/C++. Một số người gọi nó là Wiring, một số khác thì gọi là C hay C/C++. Riêng đội ngũ phát triển Arduino gọi là "*ngôn ngữ Arduino*". Ngôn ngữ Arduino bắt nguồn từ C/C++ phổ biến hiện nay do đó rất dễ học, dễ hiểu.

Để lập trình cũng như gửi lệnh và nhận tín hiệu từ mạch Arduino, nhóm phát triển dự án này đã cũng cấp đến cho người dùng một môi trường lập trình Arduino được gọi là Arduino IDE (**I**ntergrated **D**evelopment **E**nvironment) như hình dưới đây.



Hình 3.7 Arduino IDE

3.2.3 Vi điều khiển MSP430G2553

Các dòng vi điều khiển msp430 này do hãng TI (Texas Instruments) sản xuất, ngoài ra thì TI còn sản xuất và cung cấp nhiều linh kiện điện tử và các module khác. Vi điều khiển(Micro controller unit – MCU) là đơn vị xử lý nhỏ, nó được tích hợp toàn bộ các bộ nhớ như ROM , RAM , các port truy xuất , giao tiếp ngoại vi trực tiếp trên 1 con chip hết sức nhỏ gọn. Được thiết kế dựa trên cấu trúc VON-NEUMAN , đặc điểm của cấu trúc này là chỉ có duy nhất 1 bus giữa CPU và bộ nhớ (data và chương trình) , do đó mà chúng phải có độ rộng bit tương tự nhau. MSP430 có một số phiênbản như: MSP430x1xx, MSP430x2xx, MSP430x3xx, MSP430x4xx, MSP430x5xx. Dưới đây là những đặc điểm tổng quát của họ vi điều khiển MSP430:

+ Cấu trúc sử dụng nguồn thấp giúp kéo dài tuổi thọ của Pin

-Duy trì 0.1µA dòng nuôi RAM.

-Chỉ 0.8µA real-time clock.

-250 µA/ MIPS.

+ Bộ tương tự hiệu suất cao cho các phép đo chính xác

-12 bit hoặc 10 bit ADC-200 kskp, cảm biến nhiệt độ, Vref ,

-12 bit DAC.

-Bộ giám sát điện áp nguồn.

+ 16 bit RISC CPU cho phép được nhiều ứng dụng, thể hiện một phần ở kích thước Code lập trình.

-Thanh ghi lớn nên loại trừ được trường hợp tắt nghẽn tập tin khi đang làm việc.

-Thiết kế nhỏ gọn làm giảm lượng tiêu thụ điện và giảm giá thành.

-Tối ưu hóa cho những chương trình ngôn ngữ bậc cao như C, C++

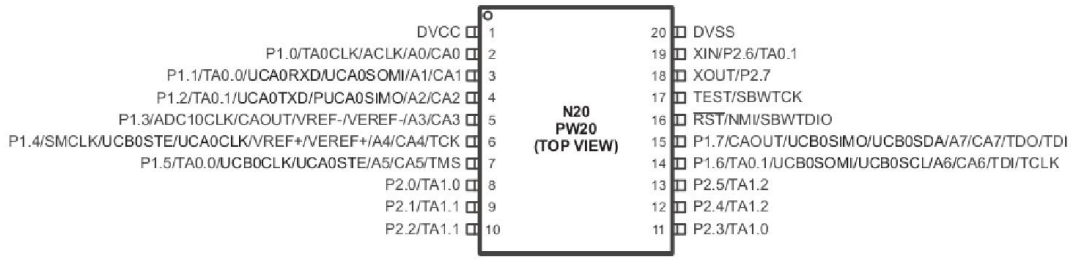
-Có 7 chế độ định địa chỉ.

-Khả năng ngắt theo véc tơ lớn.

+ Trong lập trình cho bộ nhớ Flash cho phép thay đổi Code một cách linh hoạt, phạm vi rộng, bộ nhớ Flash còn có thể lưu lại như nhật ký của dữ liệu.

Sơ đồ chân :

Chip MSP430 có kích thước nhỏ gọn , chỉ với 20 chân đối với kiểu chân DIP. Bao gồm 2 port I/O (hay GPIO general purprose input/ output : cổng nhập xuất chung).



Hình 3.8 Sơ đồ chân vi điều khiển MSP430G2553

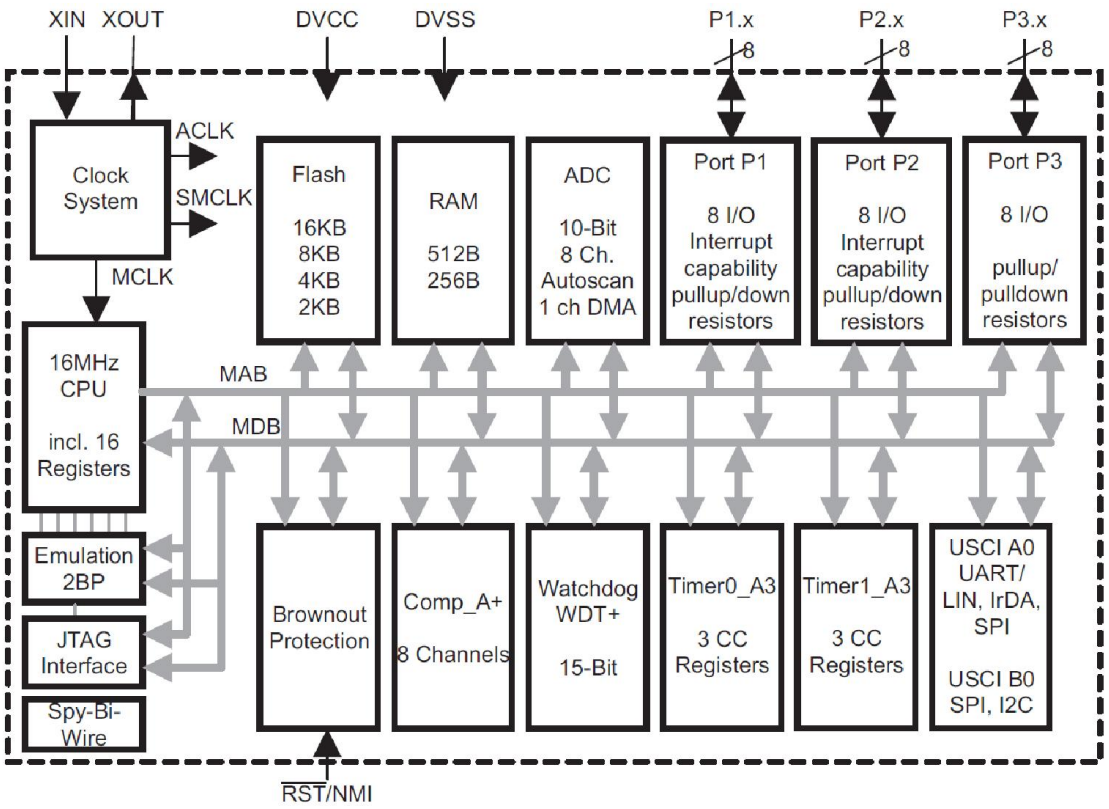
Ta thấy rằng mỗi port đều có 8 chân.

Port 1 : có 8 chân từ P1.0 đến P1.7 tương ứng với các chân từ 2-7 và 14 , 15.

Port 2 : cũng gồm có 8 chân P2.0 – P2.7 ứng với các chân 8 – 13 , 18,19.

Ngoài chức năng I/O thì trên mỗi pin của các port đều là những chân đa chức năng.

Sơ đồ khối của vi điều khiển MSP430G2553



Hình 3.9 Sơ đồ khối vi điều khiển MSP430G2553

- Chân số 1 là chân cấp nguồn Vcc(ký hiệu trên chip là DVcc) , ở đây nguồn cho chip chỉ được cấp ở mức 3,3V, nếu cấp nguồn cao quá mức này thì chip có thể hoạt động sai hay cháy chip.
-Chân 20 là chân nối cực âm (0V).

-Chân reset : Chính là chân số 16 RST , chân này có dấu gạch ngang trên có nghĩa là chân này tích cực ở mức thấp . Mục đích của việc reset là nhằm cho chương trình chạy lại từ đầu .

-Mạch dao động : Cũng giống như những dòng vi điều khiển khác thì Msp430 cũng hỗ trợ người dùng thạch anh ngoài (external crystal), nhưng thạch anh ngoại vi cho phép chỉ có thể lên tới 32,768 kHz mà thôi, và tín hiệu này được mắc trên 2 chân 18 và 19. Nhưng msp430 lại hỗ trợ thạch anh nội có thể lên đến 16Mhz, tùy vào cách khai báo trong lập trình. Và mặc định của chip là thạch anh nội. Như vậy thì chúng ta không cần thiết phải sử dụng mạch dao động ngoại cho chip giống như những dòng khác.

- Port I/O :

Port 1 : có 8 chân từ P1.0 đến P1.7 tương ứng với các chân từ 2-7 và 14 , 15.

Port 2 : cũng gồm có 8 chân P2.0 – P2.7 ứng với các chân 8 – 13 , 18,19.

Trong chế độ nhập (input) thì cả 2 port đều có 1 mạch điều khiển điện trở kéo dương – gọi là PULL UP nhưng giá trị của điện trở này rất lớn khoảng 47K nên gọi là WEAK PULL UP RESISTAN.

3.2.4 Kit MSP430 Launchpad

Msp430 là dòng value line , power low, và low – cost . Chính vì vậy mà TI đã cung cấp cho người dùng 1 mạch nạp code + debug chỉ trên 1 mạch nhỏ gọn. Đó là kit MSP430 Launchpad, trong kit còn có hỗ trợ:

-1 mạch nạp code có cả debug

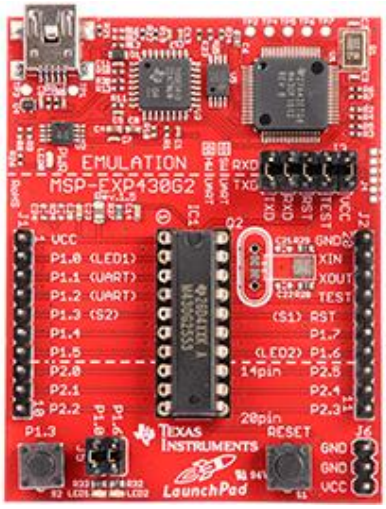
-1 dây cáp USB tốt để kết nối kit với máy tính.

-1 chip thạch anh 32,768kHz

-1 chip Msp430G2553

-1 chip Msp430G2453

-1 header female.



Hình 3.10 Kit MSP430 Launchpad

Một kit LaunchPad gồm hai thành phần, với GND được phủ chung:

- Nửa trên: Là phần mạch nạp theo chuẩn spy-bi-wire Jtag (2 dây), kết hợp với chuyển đổi giao tiếp UART với máy tính. Trên cùng là đầu USBmini để nối với máy tính, phía dưới là hàng Header để nối ra đối tượng cần giao tiếp, bao gồm các chân:

· *TXD*, *RXD*: phục vụ giao tiếp UART với máy tính.

· *RST*, *TEST*: phục vụ nạp và debug (sửa lỗi) theo chuẩn spy-bi-wire Jtag.

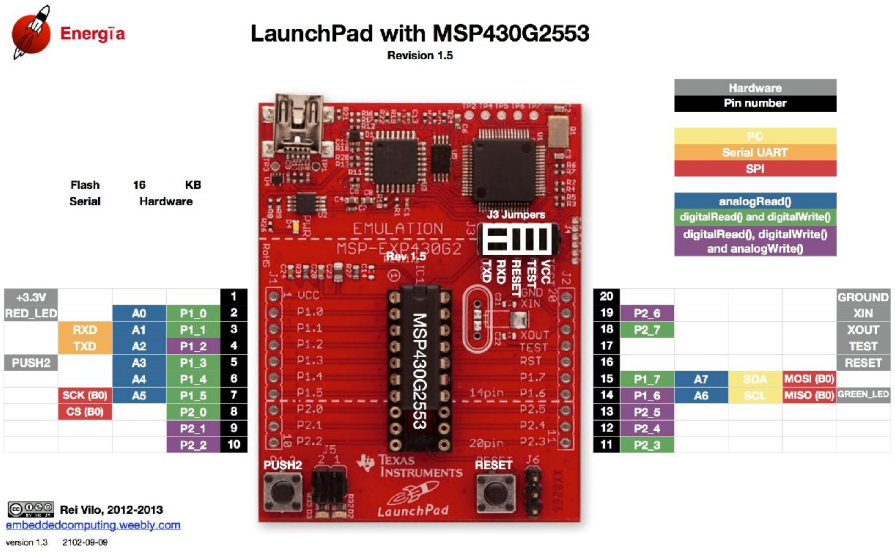
· *VCC*: cấp nguồn 3V3 cho đối tượng (thường là nửa dưới LaunchPad).

- Nửa dưới: là một mạch phát triển MSP430 đơn giản, bao gồm:

· Socket cắm MSP430 (thường gắn sẵn chip MSP430G2553), Pad hàn thạch anh, Nút nhấn Reset chip.

· Nút nhấn gắn vào P1.3, hai Led hiển thị có jumper để gắn vào P1.0 và P1.6. Hai hàng header để kết nối hai hàng chân của chip ra ngoài, một hàng header nguồn*GND-GND-VCC* để lấy nguồn 3V3 trên LaunchPad.

Sơ đồ khối các chân vào ra trên Kit MSP430 Launchpad



Hình 3.11 Sơ đồ các chân vào ra trên Kit MSP430 Launchpad

Lập trình cho MSP430 Launchpad

Kit MSP430 Launchpad được lập trình trên nền tảng ngôn ngữ Arduino. Để lập trình cũng như gửi lệnh và nhận tín hiệu từ kit, nhóm phát triển dự án này đã cũng cấp đến cho người dùng một môi trường lập trình ngôn ngữ Arduino trên kit MSP430 Launchpad được gọi là Energia IDE (**I**ntergrated **D**evelopment **E**nvironment) như hình dưới đây.

Hình 3.12 Energia IDE

3.2.4 Vi điều khiển PIC18F2550

Họ vi điều khiển PIC18F2455/2550/4455/4550 là họ vi điều khiển tiên tiến của MICROCHIP, đặc biệt họ này có tích hợp cổng USB 2.0, ADC 10 bit và tích hợp nhiều công cụ khác. Mạnh, mềm dẻo là từ đánh giá ngắn gọn về họ vi điều khiển này. Các khối tích hợp của họ vi điều khiển PIC18F và các đặc điểm của các khối tích hợp như sau:

Các đặc điểm cổng USB:

 USB V2.0

 Tốc độ thấp (1.5 Mb/s) và tốc độ toàn phần (12Mb/s)

 Hỗ trợ tới 32 điểm cuối

 RAM 1 kByte cho khối USB

 Mạch thu phát USB trên chip cùng với mạch ổn áp 3.3V

 Cổng song song streaming (SPP) cho truyền streaming USB

 Hỗ trợ cả 4 chế độ truyền:

- Truyền điều khiển (Control transfer)

- Truyền ngắt (Interrupt transfer)

- Truyền đồng bộ (Isochronous transfer)

- Truyền khối (Bulk transfer)

Các chế độ quản lý năng lượng : có nhiều chế độ quản lý năng lượng để giảm năng lượng

tiêu thụ tối đa, đặc biệt có ý nghĩa cho các ứng dụng sử dụng pin.

 Hoạt động bình thường (Run): CPU bật, ngoại vi bật

 Nghỉ (Idle): CPU tắt, ngoại vi bật, tiêu thụ dòng tiêu biểu 5.8A

 Ngủ (Sleep): CPU tắt, ngoại vi tắt, tiêu thụ dòng tiêu biểu 0.1A

 Dao động timer1: tiêu thụ dòng tiêu biểu 1.1A , 32kHz, 2V.

 Watchdog timer: tiêu thụ dòng tiêu biểu 2.1A

 Khởi động dao động 2 tốc độ

Cấu trúc dao động mềm dẻo:

 Bốn chế độ tinh thể bao gồm PLL độ chính xác cao cho USB

 Hai chế độ xung clock ngoài lên đến 48 MHz

 Khối dao động nội

- 8 tần số chọn được bởi người sử dụng, từ 31kHz đến 8 Mhz

- Tinh chỉnh bởi người dùng để bổ chính độ trôi tần số

 Dao động thứ cấp dùng timer1 32kHz

 Tùy chọn dao động đôi cho phép CPU và USB hoạt động 2 tần số xung nhịp khác nhau

 Khối theo dõi an toàn xung nhịp

- Cho phép shutdown an toàn khi tắt xung nhịp

- hay vi điều khiển sẽ tiếp tục hoạt động với tần số xung nhịp thấp hơn

Các đặc điểm ngoại vi:

 Dòng vào/ra (sink/source) cao 25mA/25mA

 Ba ngắt ngoài

 Bốn khối timer (timer0 đến timer3)

 Có tới 2 khối CCP (Capture/Compare/PWM)

- Capture 16 bit cực đại, độ phân giải 6.25 ns (T /16 CY)

- So sánh 16 bit cực đại, độ phân giải 100 ns (TCY)

- PWM có lối ra với độ giải từ 1 đến 10 bit

 Khối CCP nâng cao ECCP (Enhanced Capture/Compare/PWM)

- Nhiều chế độ lối ra

- Có thể chọn cực tính

- Thời gian chết lập trình được

- Tự động tắt và tự động khởi động

 Khối USART nâng cao

- Hỗ trợ bus LIN

 Cổng truyền nối tiếp đồng bộ chủ MSSP (Master Synchronous Serial Port)

 ADC 10 bit, 13 lối vào, thời gian thu thập dữ liệu lập trình được

 2 bộ so sánh tương tự với đa hợp lối vào

Các đặc điểm của vi điều khiển

 Cấu trúc tối ưu biên dịch C với tập lệnh mở rộng tùy chọn

 Bộ nhớ chương trình flash nâng cao cho phép 100.000 lần xóa/ghi

 Bộ nhớ dữ liệu EEPROM cho phép 1.000.000 lần xóa/ghi

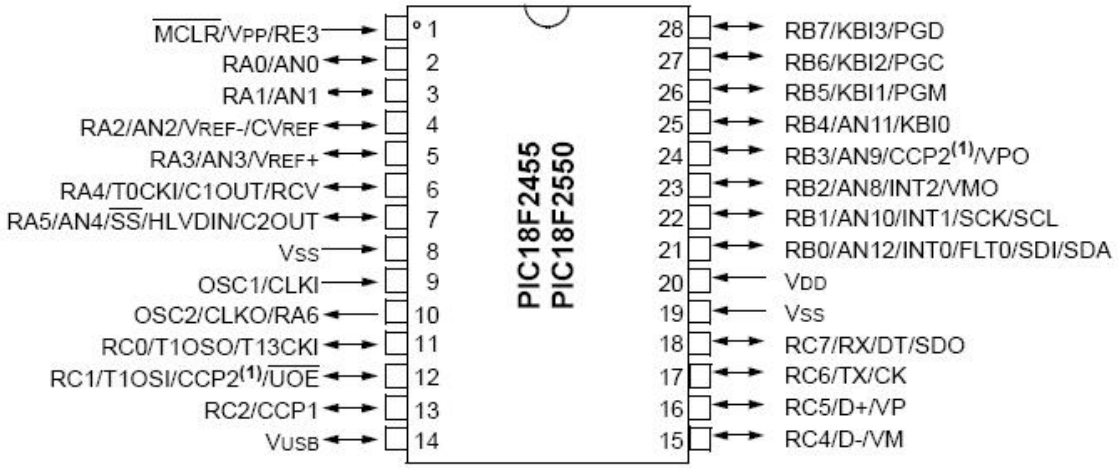
 Lưu trữ dữ liệu trong bộ nhớ flash/EEPROM hơn 40 năm

 Ngắt nhiều mức ưu tiên

 Watchdog timer mở rộng, chu kỳ khả trình từ 41 ms đến 131 s

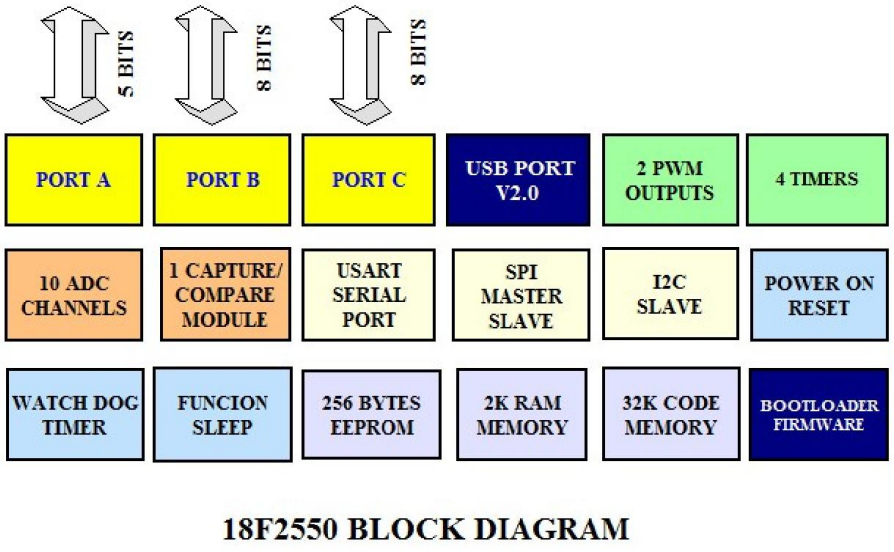
- Bảo vệ mã lập trình
- Nguồn nuôi đơn 5V cho lập trình nối tiếp trên mạch qua 2 chân
- Mạch gỡ lỗi qua 2 chân
- Dải điện áp hoạt động rộng (2.0V đến 5.5V)

Sơ đồ chân vi điều khiển PIC18F2550



Hình 3.13 Sơ đồ chân vi điều khiển PIC18F2550

Sơ đồ khối vi điều khiển PIC18F2550



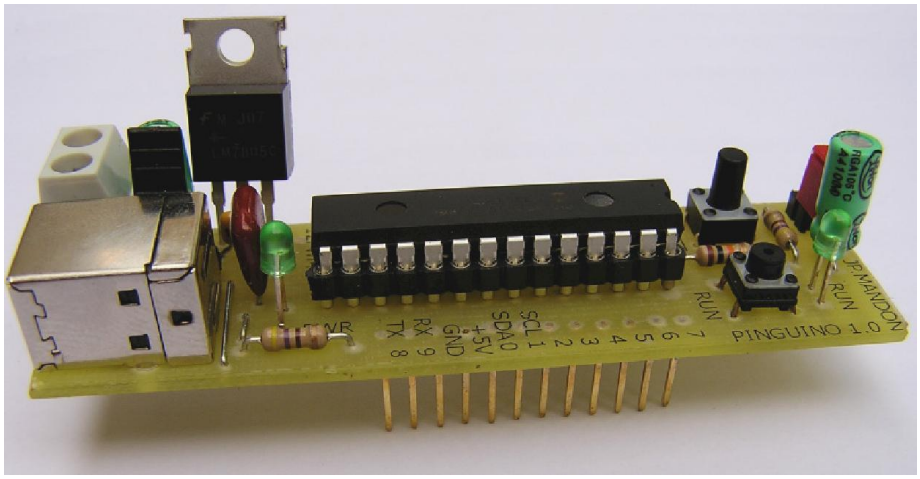
Hình 3.14 Sơ đồ khối vi điều khiển PIC18F2550

3.2.4 Kit PIC18F2550 Pinguino

Kit PIC18F2550 Pinguino được phát triển trên nền tảng vi điều khiển PIC18F2550. Trên Kit có 18 chân Digital I/O chia sẽ với 5 chân ngõ vào Analog. Ngoài ra một số chân còn có những chức năng đặc biệt như: 2 kênh ngõ ra fast PWM (3000Hz), giao tiếp I2C, giao

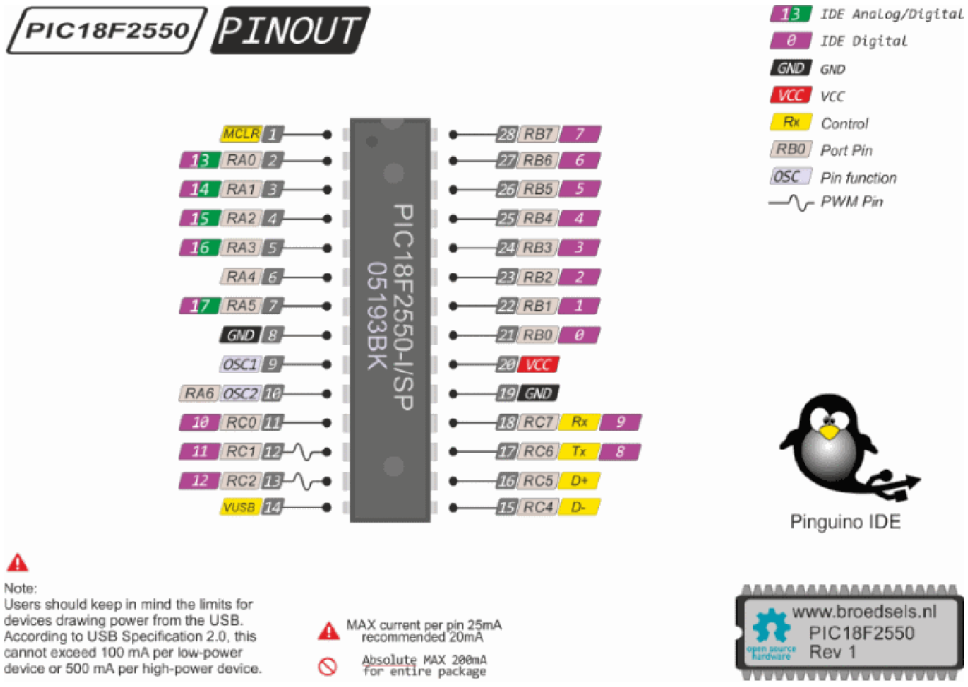
tiếp nối tiếp tốc độ lên đến 57600 bauds, USB native, đọc ghi EEPROM, truy xuất ngắt…

Kit PIC18F2550 Pinguino hỗ trợ 2 nút nhấn (1 dùng cho Reset, 1 dùng cho Run), trong một số phiên bản bootloader không dùng đến nút Run, 2 led (1 led báo nguồn, 1 led báo trạng thái Run). Ngoài ra Kit còn hỗ trợ khối nguồn ổn áp nên có thể sử dụng nguồn qua USB hoặc dùng nguồn ngoài (7-12VDC).



Hình 3.15 Kit PIC18F2550 Pinguino

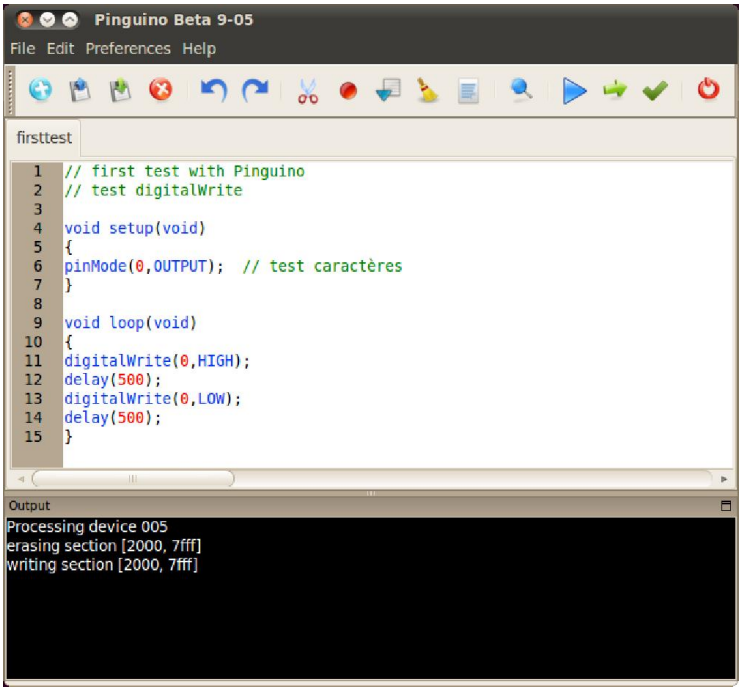
Sơ đồ khối các chân vào ra trên Kit PIC18F2550 Pinguino



Hình 3.16 Sơ đồ khối các chân vào ra trên Kit PIC18F2550 Pinguino

Lập trình cho PIC18F2550 Pinguino

Kit PIC18F2550 Pinguino được lập trình trên nền tảng ngôn ngữ Arduino. Để lập trình cũng như gửi lệnh và nhận tín hiệu từ kit, nhóm phát triển dự án này đã cũng cấp đến cho người dùng một môi trường lập trình ngôn ngữ Arduino trên kit PIC18F2550 Pinguino được gọi là Pinguino IDE (**I**ntergrated **D**evelopment **E**nvironment) như hình dưới đây.



Hình 3.17 Pinguino IDE

Chương 4: PHẦN MỀM NHÚNG

4.1. Đặc điểm phần mềm nhúng

Nói một cách đơn giản, phần mềm nhúng là phần mềm dành cho tất cả những thiết bị không liên quan gì đến máy tính, chẳng hạn như thiết bị điều khiển, định hướng cho ô tô, điện thoại di động, ví tiền điện tử, đồ gia dụng (ti vi, tủ lạnh, máy giặt, điều hòa…). Như ta đã biết, số chip vi xử lý dùng trong các máy tính, mạng nội bộ và Internet chỉ chiếm hơn 1% tổng số chip vi xử lý trên thế giới. Số còn lại thuộc về các hệ thống nhúng.

Theo số liệu của Business Communications Company, tổng thị trường phần mềm nhúng thế giới năm 2004 đạt khoảng 46 tỷ USD. Đến năm 2009, con số này là 88 tỷ USD.

Nhật Bản hiện nay được đánh giá là một trong những thị trường phần mềm nhúng hàng đầu thế giới. Theo thống kê của JISA (Hiệp hội Dịch vụ CNTT Nhật Bản), phần mềm nhúng hiện nay chiếm tới 40% thị phần phần mềm Nhật Bản, với các sản phẩm rất đa dạng: lò vi ba, máy photocopy, máy in laser, máy FAX, các bảng quảng cáo sử dụng hệ thống đèn LED, màn hình tinh thể lỏng… Năm 2004, thị trường phần mềm nhúng của Nhật Bản đạt khoảng 20 tỷ USD với 150.000 nhân viên. Đây được coi là thị trường đầy hứa hẹn với các đối tác chuyên sản xuất phần mềm nhúng như Trung Quốc, Indonesia, Nga, Ireland, Israel và cả Việt Nam.

Đặc điểm phần mềm nhúng:

- Hướng chức năng hoá đặc thù
- Hạn chế về tài nguyên bộ nhớ
- Yêu cầu thời gian thực

4.2 Lập trình nhúng với ngôn ngữ Arduino

Chương trình Arduino có thể được chia làm 3 phần: cấu trúc (structure), giá trị (variables), hàm và thủ tục (function).

4.2.1 Cấu trúc

4.2.1.1 setup()

Những lệnh trong setup() sẽ được chạy khi chương trình khởi động, có thể sử dụng nó để khai báo giá trị của biến, khai báo thư viện, thiết lập các thông số,…Những lệnh trong setup() chỉ chạy một lần để khai báo.

Ví dụ

```
int led = 13;
void setup() {
```

```
  pinMode(led, OUTPUT);
}
```

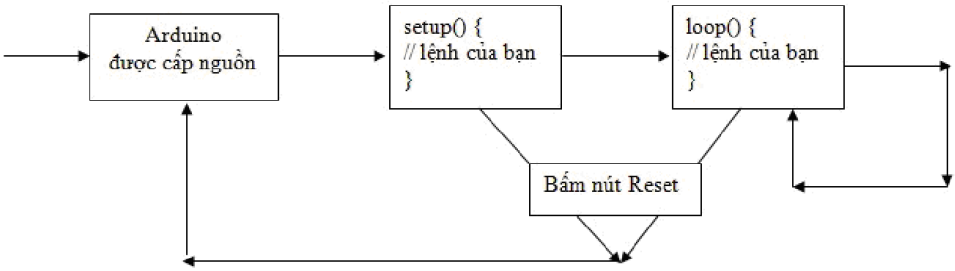
Khi cấp nguồn cho Arduino, lệnh "*pinMode(led, OUTPUT);"* sẽ được chạy 1 lần để khai báo.

4.2.1.2 loop()

Sau khi setup() chạy xong, những lệnh trong loop() được chạy. Chúng sẽ lặp đi lặp lại liên tục cho tới khi nào nguồn của board Arduino được ngắt mới thôi.

Bất cứ khi nào nhất nút Reset, chương trình sẽ trở về lại trạng thái như khi Arduino mới được cấp nguồn.

Quá trình này có thể được miêu tả như sơ đồ dưới đây



Ví dụ

```
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Lệnh ở loop() sẽ được chạy và được lặp đi lặp lại liên tục, tạo thành một chuỗi:

```
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
```

```
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
……
```

4.2.1.3 Cú pháp mở rộng

4.2.1.3.1 Dấu chấm phẩy “;”

Dùng để kết thúc một dòng lệnh.

Ví dụ

```
int ledPin = 13;
```

Khi lập trình nếu quên dấu ";" chương trình dịch của Arduino sẽ tự động dò tìm lỗi này và thông báo chính xác dòng bị lỗi.

4.2.1.3.2 Dấu ngoặc nhọn “{}”

Nhiệm vụ của “{}” là cung cấp một cú pháp để gọi những lệnh cho những cấu trúc đặc biệt như (if, while, for,...)

Ví dụ:

Trong hàm và thủ tục

```
void myfunction(<kiểu dữ liệu> <tham số>){
   <lệnh>;
}
```

Vòng lặp

```
while (<điều kiện>)
{
  <câu lệnh>
}

do
{
  <câu lệnh>
} while (<điều kiện>);

for (<khởi tạo>; <điều kiện>; <bước>)
```

```
{
  <câu lệnh>
}
```

Lệnh rẻ nhánh

```
if (<điều kiện 1>)
{
 <lệnh 1>
}

else if (<điều kiện 2>)
{
 <lệnh 2>
}
else
{
 <lệnh 3>
}
```

4.2.1.3.3 Dòng chú thích "//"

Dòng chú thích sẽ giúp ghi chú cho từng dòng code hoặc trình bày nhiệm vụ của nó để người đọc có thể hiểu được chương trình này làm được những gì. Dòng chú thích sẽ không được Arduino biên dịch nên cho dù dòng chú thích có dài đến đâu thì cũng không ảnh hưởng đến bộ nhớ flash của vi điều khiển.

Ví dụ

```
x = 5;  // Đây là dòng chú thích
    // nó sẽ ghi chú tất cả những chữ (text, câu lệnh,... everything) nằm sau dấu // cho đến khi hết dòng
```

4.2.1.3.4 Đoạn chú thích "/* */"

Cũng giống như dòng chú thích nhưng được ghi trên dòng khác nhau.

Ví dụ

```
/* Đây là đoạn chú thích
Nó sẽ "ghi chú" tất cả những gì nằm trong cặp dấu "/ *" và "* /"
```

```
if (gwb == 0){   // Có thể dùng dòng chú thích trong này
x = 3;          /* nhưng không được dùng một đoạn chú thích khác, chương trình sẽ bị lỗi cú pháp
*/
}
// kết thúc với ký tự đóng "* /"
*/
```

Dòng chú thích thường dùng khi cần ghi chú một đoạn code. Đoạn chú thích thường dùng để debug một đoạn code mới thêm vào. Khi thêm một đoạn code mới vô tình làm cho chương trình hoạt động lỗi thì hãy thử dùng đoạn chú thích để đánh dấu là ghi chú những dòng đó (chương trình dịch sẽ bỏ qua). Sau đó, xem thử chương trình có chạy đúng hay không, nếu có lỗi thì mở rộng dòng chú thích lên những dòng code trước đó nữa, còn nếu không thì thu hẹp đoạn chú thích lại và tiếp tục thực hiện cho đến khi hết lỗi.

4.2.1.3.5 #define

#define là một đối tượng của ngôn ngữ C/C++ cho phép đặt tên cho một hằng số nguyên hay hằng số thực. Trước khi biên dịch, trình biên dịch sẽ thay thế những tên hằng đang sử dụng bằng chính giá trị của chúng. Quá trình thay thế này được gọi là quá trình tiền biên dịch (pre-compile).

Cú pháp

```
#define [tên hằng] [giá trị của hằng]
```

Ví dụ

```
#define pi 3.14
```

Nếu viết code như sau:

```
#define pi 3.14
float a = pi * 2.0;  //    pi = 6.28
```

thì sau khi pre-compile trước khi biên dịch, kết quả là:

```
#define pi 3.14
float a = 3.14 * 2.0;  //   a = 6.28
```

Lưu ý

Nếu một biến có tên trùng với tên hằng số được khai báo bằng *#define* thì khi pre-comile, biến ấy sẽ bị thay thế bằng giá trị của hằng số kia. Hệ quả tất yếu là khi biên dịch, chương trình sẽ bị lỗi cú pháp (không dễ dàng nhận ra điều này). Đôi khi nó cũng dẫn đến lỗi logic - một lỗi rất khó sửa.

Nếu viết code như sau:

```
#define pi 3.14
```

```
float pi = 3.141592654;
```

thì sau khi pre-compile trước khi biên dịch, kết quả là:

```
#define pi 3.14
float 3.14 = 3.141592654;     //   lỗi cú pháp
```

Vì vậy, nên hạn chế tối đa việc sử dụng #define để khai báo hằng số khi không cần thiết. Có thể sử dụng cách khai báo sau để thay thế:

```
const float pi = 3.14;
```

4.2.1.3.6 #include

#include cho phép chương trình tải một thư viện đã được viết sẵn. Tức là có thể truy xuất được những tài nguyên trong thư viện này từ chương trình. Nếu có một đoạn code và cần sử dụng nó trong nhiều chương trình, có thể dùng #include để nạp đoạn code ấy vào chương trình, thay vì phải chép đi chép lại đoạn code ấy.

Cú pháp

```
#include <[đường dẫn đến file chứa thư viện]>
```

Ví dụ

Giả sử có thư mục cài đặt Arduino IDE tên là ArduinoIDE, thư viện có tên là EEPROM (được lưu ở \ArduinoIDE\libraries\EEPROM\)

Một đoạn code lưu ở file code.h nằm trong thư mục function của thư viện EEPROM thì được khai báo như sau:

```
#include <function/code.h> //đường dẫn đầy đủ: \ArduinoIDE\libraries\EEPROM\function\code.h
```

4.2.1.4 Toán tử số học

4.2.1.4.1 Phép gán "="

Dùng để gán một giá trị cho một biến. Giá trị nằm trên phải, tên biến nằm bên trái ngăn cách nhau bởi 1 dấu bằng.

Ví dụ

```
int sensVal;              // Khai báo kiểu dữ liệu cho biến sensVal là int
sensVal = analogRead(0);      // đặt giá trị cho sensVal là giá trị analog tại chân A0
```

4.2.1.4.2 Phép cộng, phép trừ, phép nhân, phép chia

Những phép toán trên có nhiệm vụ thực hiện việc tính toán trong Arduino. Tùy thuộc vào kiểu dữ liệu của các biến hoặc hằng số mà kết quả trả về của nó có kiểu dữ liệu như thế nào. Nếu là kiểu số nguyên thì nó cũng sẽ overflow. Và nếu các giá trị đưa vào là số thực thì được phép sử dụng các dấu chấm "." để ngăn cách phần nguyên và phần thực.

Ví dụ

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

Cú pháp

```
result = value1 + value2;
result = value1 - value2;
result = value1 * value2;
result = value1 / value2;
```

Tham số

value1: là một số ở bất kỳ kiểu dữ liệu nào

value2: là một số ở bất kỳ kiểu dữ liệu nào

4.2.1.4.3 Phép chia lấy dư

Phép chia lấy dư là phép lấy về phần dư của một phép chia các số nguyên.

Cú pháp

```
<phần dư> = <số bị chia> / <số chia>;
```

Ví dụ

```
x = 7 % 5;   // x bây giờ là 2
x = 9 % 5;   // x bây giờ là 4
x = 5 % 5;   // x bây giờ là 0
x = 4 % 5;   // x bây giờ là 4
```

Mã lập trình tham khảo

```
/* cập nhập lại giá trị trong hàm loop */
int values[10];
int i = 0;
void setup() {}


void loop()
{
  values[i] = analogRead(0);
```

```
  i = (i + 1) % 10;   // giá trị của biến i sẽ không bao giờ vượt quá 9.
}
```

Lưu ý

Phép chia lấy dư không khả dụng với kiểu dữ liệu float

4.2.1.5 Toán tử so sánh

Các toán tử so sánh thường dùng để so sánh 2 số có cùng một kiểu dữ liệu.

Toán tử	**Ý nghĩa**	**Ví dụ**
==	So sánh bằng	(a == b) trả về TRUE nếu a bằng b và ngược lại
!=	So sánh không bằng	(a != b) trả về TRUE nếu a khác b và ngược lại
>	So sánh lớn	(a > b) trả về TRUE nếu a lớn hơn b và FALSE nếu a bé hơn hoặc bằng b
<	So sánh bé	(a < b) trả về TRUE nếu a bé hơn b và FALSE nếu ngược lại
<=	Bé hơn hoặc bằng	(a <= b) tương đương với ((a < b) or (a = b))
>=	Lớn hơn hoặc bằng	(a >= b) tương đương với ((a > b) or (a = b))

4.2.1.6 TOÁN TỬ LOGIC

Toán tử	**Ý nghĩa**	**Ví dụ**
and (&&)	Và	(a && b) trả về TRUE nếu a và b đều mang giá trị TRUE. Nếu một trong a hoặc b là FALSE thì (a && b) trả về FALSE
or (||)	Hoặc	(a || b) trả về TRUE nếu có ít nhất 1 trong 2 giá trị a và b là TRUE, trả về FALSE nếu a và b đều FALSE
not (!)	Phủ định	nếu a mang giá trị TRUE thì (!a) là FALSE và ngược lại
xor (^)	Loại trừ	(a ^ b) trả về TRUE nếu a và b mang hai giá trị TRUE/FALSE khác nhau, các trường hợp còn lại trả về FALSE

4.2.1.7 Toán tử hợp nhất

4.2.1.7.1 Cộng một, trừ một

Tăng hoặc trừ một biến đi 1 đơn vị.

Cú pháp

```
x++;  // tăng x lên 1 giá trị và trả về giá trị cũ của x
++x;  // tăng x lên 1 giá trị và trả về giá trị mới của x
x-- ;   // trừ x lên 1 giá trị và trả về giá trị cũ của x
--x ;   // trừ x lên 1 giá trị và trả về giá trị mới của x
```

Tham số

x: bất kỳ kiểu số nguyên nào (int, byte, unsigned int,...)

Trả về

Giá trị cũ của biến hoặc giá trị mới đã được cộng / hoặc bị trừ của biến ấy.

Ví dụ

```
x = 2;
y = ++x;      // x bây giờ có giá trị là 3, và y cũng có giá trị là 3
y = x--;      // x bây giờ đã trở lại với giá trị 2, nhưng y không đổi vẫn là 3
```

4.2.1.7.2 Các phép toán rút gọn

Cung cấp một số phép toán rút gọn để viết code trong đẹp hơn và dễ nhớ hơn.

Cú pháp

```
x += y;   // giống như phép toán x = x + y;
x -= y;   // giống như phép toán x = x - y;
x *= y;   // giống như phép toán x = x * y;
x /= y;   // giống như phép toán x = x / y;
```

Tham số

x: mọi kiểu dữ liệu

y: mọi kiểu dữ liệu hoặc hằng số

Ví dụ

```
x = 2;
x += 4;      // x bây giờ có giá trị là 6
x -= 3;      // x bây giờ có giá trị là 3
x *= 10;     // x bây giờ có giá trị là 30
x /= 2;      // x bây giờ có giá trị là 15
```

4.2.1.8 Cấu trúc điều khiển

4.2.1.8.1 Câu lệnh if, if…else

Cú pháp:

```
if ([biểu thức 1] [toán tử so sánh] [biểu thức 2]) {    //biểu thức điều kiện
   [câu lệnh 1]
} else {
   [câu lệnh 2]
}
```

Nếu biểu thức điều kiện trả về giá trị TRUE, [câu lệnh 1] sẽ được thực hiện, ngược lại, [câu lệnh 2] sẽ được thực hiện.

Ví dụ:

```
int a = 0;
if (a == 0) {
   a = 10;
} else {
   a = 1;
}
// a = 10
```

Lệnh if không bắt buộc phải có nhóm lệnh nằm sau từ khóa else

```
int a = 0;

if (a == 0) {

    a = 10;

}

// a = 10
```

Có thể kết hợp nhiều biểu thức điều kiện khi sử dụng lệnh if. Chú ý rằng mỗi biểu thức con phải được bao bằng một ngoặc tròn và phải luôn có một cặp ngoặc tròn bao toàn bộ biểu thức con.

Cách viết đúng	**Cách viết sai**

int a = 0;
if ((a == 0) && (a < 1)) {
a = 10;
}
	int a = 0;
	if (a == 0) && (a < 1) {
	a = 10;
	}

	int a = 0;
	if (a == 0 && a < 1) {
	a = 10;
	}

Chú ý:

- *a = 10; là một câu lệnh gán, giá trị logic của nó luôn là TRUE (vì lệnh gán này luôn thực hiện được)*
- *(a == 10) là một biểu thức logic có giá trị TRUE hay FALSE tùy thuộc vào giá trị của biến a.*

Nếu:

```
int a = 0;
if (a = 1) {
  a = 10;
}
```

… thì giá trị của a sẽ bằng 10, vì (a = 1) là một câu lệnh gán, trong trường hợp này nó được xem như một biểu thức logic và luôn trả về giá trị TRUE.

4.2.1.8.2 Câu lệnh switch…case

Giống như if, switch / case cũng là một dạng lệnh nếu thì, nhưng nó được thiết kế chuyên biệt để xử lý giá trị trên một biến chuyên biệt.

Ví dụ, có một biến là action sẽ nhận trị từ những module khác qua serial. Nhưng action sẽ nằm trong một các giá trị nào đó thì lúc này hãy sử dụng switch / case.

Ví dụ

```
switch (action) {
  case "callMyMom":
    //gọi điện cho mẹ của tôi
    break;
  case "callMyDad":
```

```
    //gọi điện cho ba của tôi
    break;
  default:
    // mặc định là không làm gì cả
    // có thể có default: hoặc không
 }
```

Cú pháp

```
switch (var) {
 case label:
  //đoạn lệnh
   break;
 case label:
   // Đoạn lệnh
   break;
/*
 case ... more and more
*/
 default:
   // statements
}
```

Tham số

var: biến muốn so sánh

label: sẽ đem giá trị của biến SO SÁNH BẰNG với nhãn này

4.2.1.8.3 Vòng lặp for

Hàm for có chức năng làm một vòng lặp. Hàm for làm đi làm lại một công việc có một tính chất chung nào đó. Chẳng hạn, bật tắt một con LED thì dùng digitalWrite xuất HIGH delay rồi lại LOW rồi lại delay. Nhưng nếu muốn làm nhiều hơn 1 con LED thì đoạn code sẽ dài ra.

Ví dụ:

Hãy xuất 10 chữ số (từ 1 - 10) ra Serial.

Không dùng hàm for

```
void setup() {
  Serial.begin(9600);
```

```
    Serial.println(1);
    Serial.println(2);
    Serial.println(3);
    Serial.println(4);
    Serial.println(5);
    Serial.println(6);
    Serial.println(7);
    Serial.println(8);
    Serial.println(9);
    Serial.println(10);
}
void loop() {
    // không làm gì cả;
}
```

Khi sử dụng hàm for:

```
void setup(){
    Serial.begin(9600);
    int i;
    for (i = 1;i<=10;i=i+1) {
        Serial.println(i);
    }
}
void loop(){
}
```

Cấu trúc

Theo ví dụ trên, đoạn code sử dụng hàm for như sau:

```
for (i = 1;i<=10;i = i + 1) {
    Serial.println(i);
}
```

Hàm for trong ví dụ này sẽ :

1. *Chắc chắn nó sẽ xử lý đoạn code Serial.println(i); 10 lần*

2. *Chạy từ vị trí xuất phát là 1, đến vị trí kết thúc là 10 // i = 1 ; i <= 10*
3. *Cứ mỗi lần bước xong (tính luôn cả vị trí xuất phát tại thời điểm i = 1) thì nó lại chạy lệnh Serial.println(i);. Trong đó biến i, gọi là biến con chạy*
4. *Mỗi lần chạy xong, i lại bước thêm 1 bước nữa. Chừng nào mà thằng i còn <= 10 thì nó còn quay về bước 3*

Hàm for được chia làm 2 loại

- For tiến (xuất phát từ một vị trí nhỏ chạy đến vị trí lớn hơn) <vị trí kết thúc> bé hơn <vị trí kết thúc>

> for (<kiểu dữ liệu nguyên> <tên thằng chạy> = <vị trí xuất phát>; <tên thằng chạy> <= <vị trí kết thúc>; <tên thằng chạy> += <mỗi lần bước mấy bước>) {
>
> <đoạn câu lệnh>;
>
> }

- For lùi (xuất phát từ một vị trí lớn chạy về vị trí nhỏ hơn) <vị trí xuất phát> lớn hơn <vị trí kết thúc>

> for (<kiểu dữ liệu nguyên> <tên thằng chạy> = <vị trí xuất phát>; <tên thằng chạy> <= <vị trí kết thúc>; <tên thằng chạy> -= <mỗi lần lùi mấy bước>) {
>
> <đoạn câu lệnh>;
>
> }

Cú pháp chính của hàm For:

> for (<biến chạy> = <start>;<điều kiện>;<bước>) {
>
> //lệnh
>
> }

4.2.1.8.4 Vòng lặp while

Vòng lặp while là một dạng vòng lặp theo điều kiện, không thể biết trước số lần lặp của nó, nhưng có thể quản lý lúc nào thì nó ngừng lặp!

1. *While là một vòng lặp không biết trước số lần lặp, nó dựa vào điều kiện, điều kiện còn đúng thì còn chạy.*
2. *Chạy một đoạn lệnh (trong đó có những hàm ảnh hưởng đến điều kiện).*

Cú pháp

> while (<điều kiện>) {
>
> //các đoạn lệnh;
>
> }

Ví dụ

```
int day = 1;
int nam = 2014; // Năm 2014
while (day < 365) { //Chừng nào day < 365 thì còn chạy (<=364). Khi day == 365 thì hết 1 năm...
   day += 1; //
   delay(60*60*24);// Một ngày có 24 giờ, mỗi giờ có 60 phút, mỗi phút có 60 giây
}
nam += 1; //... bây giờ đã là một năm mới ! Chúc mừng năm mới :)
```

4.2.1.8.5 Câu lệnh break

break là một lệnh có chức năng dừng ngay lập tức một vòng lặp (do, for, while) chứa nó trong đó. Khi dừng vòng lặp, tất cả những lệnh phía sau break và ở trong vòng lặp chịu ảnh hưởng của nó sẽ bị bỏ qua.

Ví dụ

```
int a = 0;
while (true) {
   if (a == 5) break;
   a = a + 1;
}
//a = 5
```

```
while (true) {
   while (true) {
      a++;
      if (a > 5) break;
   }
   a++;
   if (a > 100) break;
}
//a = 101
```

4.2.1.8.6 Câu lệnh continue

continue là một lệnh có chức năng bỏ qua một chu kì lặp trong một vòng lặp (for, do, while) chứa nó trong đó. Khi gọi lệnh continue, những lệnh sau nó và ở trong cùng vòng lặp với nó sẽ bị bỏ qua để thực hiện những chu kì lặp kế tiếp.

Ví dụ

```
int a = 0;
int i = 0;
while (i < 10) {
  i = i + 1;
  continue;
  a = 1;
}
//a vẫn bằng 0
```

4.2.1.8.7 Câu lệnh return

return có nhiệm vụ trả về một giá trị (cùng kiểu dữ liệu với hàm) mà nó được gọi!

Cú pháp

```
return;
return value; // cả 2 đều đúng
```

Thông số

value: bất kỳ giá trị hoặc một đối tượng.

Ví dụ

```
//Hàm kiểm tra giá trị của cảm biến có hơn một ngưỡng nào đó hay không
int checkSensor(){
  if (analogRead(0) > 400) {
    return 1;
  else{
    return 0;
}
```

4.2.1.8.8 Câu lệnh goto

Goto có nhiệm vụ tạm dừng chương trình rồi chuyển đến một nhãn đã được định trước, sau đó lại chạy tiếp chương trình!

Cú pháp

```
label: //Khai báo một nhãn có tên là label
```

goto label; //Chạy đến nhãn label rồi sau đó thực hiện tiếp những đoạn chương trình sau nhãn đó

Lệnh goto chỉ hữu ích khi đang dùng nhiều vòng lặp quá và muốn thoát khỏi nó một cách nhanh chóng.

Ví dụ

```
for(byte r = 0; r < 255; r++){
   for(byte g = 255; g > -1; g--){
      for(byte b = 0; b < 255; b++){
         if (analogRead(0) > 250){ goto bailout;}
         //thêm nhiều câu lệnh nữa
      }
   }
}
bailout:
```

4.2.2 Giá trị

4.2.2.1 Hằng số

4.2.2.1.1 HIGH

*Trong lập trình trên Arduino, **HIGH** là một hằng số có giá trị nguyên là 1. Trong điện tử, **HIGH** là một mức điện áp lớn hơn 0V. Giá trị của **HIGH** được định nghĩa khác nhau trong các mạch điện khác nhau, nhưng thường được quy ước ở các mức như 1.8V, 2.7V, 3.3V 5V, 12V, ...*

HIGH là một hằng số có giá trị nguyên là 1

Xét 2 đoạn code ví dụ sau:

```
int led = 13;


void setup() {
   pinMode(led, OUTPUT);
   digitalWrite(led, HIGH);
}
void loop() {
}
```

Đoạn code này có chức năng bật sáng đèn led nối với chân số 13 trên mạch Arduino (Arduino Nano, Arduino Uno R3, Arduino Mega 2560, ...).

```
int led = 13;
void setup() {
   pinMode(led, OUTPUT);
   digitalWrite(led, 1);
}
void loop() {
}
```

Sẽ xuất hiện 2 vấn đề:

- Trong đoạn code thứ 2, "***HIGH***" đã được sửa thành "***1***".
- Đèn led trên mạch Arduino vẫn sáng bình thường với 2 chương trình khác nhau.

Điều này khẳng định "***HIGH*** *là một* **hằng số** *có giá trị nguyên là 1"* đã nêu ở trên.

HIGH là một điện áp lớn hơn 0V

Điện áp (điện thế) tại một điểm là trị số hiệu điện thế giữa điểm đó và cực âm của nguồn điện (0V). Giả sử ta có một viên pin vuông 9V thì ta có thể nói điện áp ở cực dương của cục pin là 9V, hoặc hiệu điện thế giữa 2 cực của cục pin là 9V.

Điện áp ở mức ***HIGH*** không có giá trị cụ thể như 3.3V, 5V, 9V, ... mà trong mỗi loại mạch điện, nó có trị số khác nhau và đã được quy ước trước. Trong các mạch Arduino, ***HIGH*** được quy ước là mức 5V mặc dù 4V vẫn có thể được xem là ***HIGH***. Ví dụ như trong mạch Arduino Uno R3, theo nhà sản xuất, điện áp được xem là ở mức HIGH nằm trong khoảng từ 3V đến 5V.

Dù HIGH không có một trị số nào rõ ràng nhưng nhất quyết rằng giá trị của nó luôn lớn hơn 0V.

4.2.2.1.2 LOW

Trong lập trình trên Arduino, ***LOW*** *là một hằng số có giá trị nguyên là 0. Trong điện tử,* ***LOW*** *là mức điện áp 0V hoặc gần bằng 0V, giá trị này được định nghĩa khác nhau trong các mạch điện khác nhau, nhưng thường là 0V hoặc hơn một chút xíu.*

LOW là một hằng số có giá trị nguyên là 0

Xét 2 đoạn code ví dụ sau:

```
int led = 13;
void setup() {
   pinMode(led, OUTPUT);
}
void loop() {
   digitalWrite(led, HIGH);
```

```
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Đoạn code này có chức năng bật làm led chớp tắt nối với chân số 13 trên mạch Arduino với thời gian delay là 1s (Arduino Nano, Arduino Uno R3, Arduino Mega 2560, ...).

```
int led = 13;
void setup() {
  pinMode(led, OUTPUT);
}
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, 0);
  delay(1000);
}
```

Sẽ xuất hiện 2 vấn đề:

- Trong đoạn code thứ 2, "***LOW***" đã được sửa thành "***0***".
- Đèn led trên mạch Arduino vẫn chớp tắt bình thường với 2 chương trình khác nhau.

Điều này khẳng định "***LOW*** *là một* **hằng số** *có giá trị nguyên là 0"* đã nêu ở trên.

LOW là một điện áp lớn hơn 0V

Điện áp (điện thế) tại một điểm là trị số hiệu điện thế giữa điểm đó và cực âm của nguồn điện (0V). Giả sử ta có một viên pin vuông 9V thì ta có thể nói điện áp ở cực dương của cục pin là 9V, ở cực âm là 0V, hoặc hiệu điện thế giữa 2 cực của cục pin là 9V.

Điện áp ở mức ***LOW*** không có giá trị cụ thể như 3.3V, 5V, 9V, ... mà trong mỗi loại mạch điện, nó có một trị số khác nhau nhưng **thường là 0V hoặc gần bằng 0V**. Trong các mạch Arduino, ***LOW*** được quy ước là mức 0V mặc dù 0.5V vẫn có thể được xem là ***LOW***. Ví dụ như trong mạch Arduino Uno R3, theo nhà sản xuất, điện áp được xem là ở mức ***LOW*** nằm trong khoảng từ 0V đến 1.5V ở các chân I/O.

4.2.2.1.3 Thiết đặt Digital Pins như là INPUT, INPUT_PULLUP, và OUTPUT

Chân vi điều khiển có thể được sử dụng như là INPUT, INPUT_PULLUP , hoặc OUTPUT. Để thay đổi cách sử dụng một pin, chúng ta sử dụng hàm pinMode().

Cấu hình một pin là INPUT

Các pin của Arduino (Atmega) được cấu hình là một INPUT với pinMode ( ) có nghĩa là làm cho pin ấy có trở kháng cao (không cho dòng điện đi ra) . Pin được cấu hình là INPUT làm việc tiêu thụ năng lượng điện của mạch rất nhỏ, nó tương đương với một loạt các điện trở 100 Mega-ôm ở phía trước của pin . Điều này làm cho chúng cực kì hữu ích cho việc đọc một cảm biến, nhưng không cung cấp năng lượng một đèn LED. Khi một pin được cấu hình là INPUT thì sẽ dễ dàng đọc được các tín hiệu điện (Có điện <= 5V).

Nếu đã cấu hình pin là INPUT và muốn pin có một tham chiếu đến GND (cực âm), thường được thực hiện với một điện trở kéo xuống.

Cấu hình một pin là INPUT_PULLUP

Chip Atmega trên Arduino có nội kéo lên điện trở (điện trở kết nối với hệ thống điện nội bộ) có thể truy cập. Nếu không thích mắc thêm một điện trở ở mạch ngoài, có thể dùng tham số INPUT_PULLUP trong pinMode(). Mặc định khi không được kết nối với một mạch ngoài hoặc được kết nối với cực dương thì pin sẽ nhận giá trị là HIGH, khi pin được thông tới cực âm xuống GND thì nhận giá trị là LOW.

Cấu hình một pin là đầu ra (OUTPUT)

Để thiết đặt pin là một OUTPUT, chúng ta dùng pinMode ( ), điều này có nghĩa là làm cho pin ấy có một trở kháng thấp (cho dòng điện đi ra). Pin sẽ cung cấp một lượng điện đáng kể cho các mạch khác . Pin của vi điều khiển Atmega có thể cung cấp một nguồn điện liên tục 5V hoặc thả chìm (cho điện thế bên ngoài chạy vào) lên đến 40 mA (milliamps). Lúc bấy giờ, nếu làm ngắn mạch (digitalWrite 1 pin là HIGH rồi nối trực tiếp đến cực âm hoặc digitalWrite 1 pin là LOW rồi mắc trực tiếp đến cực dương) thì mạch sẽ bị hỏng. Ngoài ra, với dòng điện chỉ 40mA thì trong một số trường hợp chúng ta không thể làm cho mô tơ hoặc relay hoạt động được. Để làm chúng hoạt động thì chúng ta cần chuẩn bị cho mình một số mạch sử dụng các IC khuếch đại chuyên dụng (gọi là mạch giao tiếp)

4.2.2.1.4 LED_BUILTIN

Hầu hết các mạch Arduino đều có một pin kết nối với một on-board LED (led nằm trên mạch) nối tiếp với một điện trở, chúng có giá trị là 13. LED_BUILTIN là một hằng số thay thế cho việc tuyên bố một biến có giá trị điều khiển on-board LED.

4.2.2.1.5 true

true là một hằng logic, cũng có thể *HIỂU true* là một hằng số nguyên mang giá trị là 1. Trong các biểu thức logic, một hằng số hay giá trị của một biểu thức khác 0 được xem như là mang giá trị *true*.

Lưu ý

- Không được viết "*true*" thành TRUE hay bất kì một dạng nào khác.

- Các giá trị sau là tương đương nhau: ***true***, ***HIGH***, ***1***

4.2.2.1.6 false

Trái lại với *true, false* là một hằng logic có giá trị là phủ định của *true* (và ngược lại), tức là *(!true) = false,* cũng có thể *HIỂU false* là một hằng số nguyên mang giá trị là 0. Trong các biểu thức logic, một hằng số hay giá trị của một biểu thức bằng 0 được xem như là bằng *false*.

Lưu ý

- Không được viết "*false*" thành FALSEhay bất kì một dạng nào khác.
- Các giá trị sau là tương đương nhau: ***false***, ***LOW***, ***0***

4.2.2.1.7 Hằng số nguyên

Hằng số nguyên là những con số được sử dụng trực tiếp trong chương trình. Theo mặc định, những con số này có kiểu là int (trong pascal thì kiểu int giống như kiểu integer).

Ví dụ

```
if (a >= 10) {
   a = 0;
}
```

Ở đây 0 và 10 được gọi là những hằng nguyên.

Thông thường, các hằng số nguyên được biểu thị dưới dạng thập phân, nhưng có thể biểu thị chúng trong các hệ cơ số khác như sau:

Hệ cơ số	**Ví dụ**	**Định dạng**	**Ghi chú**
10 (thập phân)	159		biểu thị bằng các chữ số từ 0 đến 9
2 (nhị phân)	B1111011	bắt đầu bằng 'B' (hoặc '0B')	biểu thị bằng 2 chữ số 0 và 1
8 (bát phân)	0173	bắt đầu bằng '0'	biểu thị bằng các chữ số từ 0-7
16 (thập lục phân)	0x7B	bắt đầu bằng '0x'	biểu thị bằng các chữ số từ 0-9 và kí tự từ A-F (hoặc a-f)

Hệ thập phân (decimal) là hệ cơ số 10, cũng là hệ cơ số được dùng phổ biến nhất.

Ví dụ:

```
int a = 101;     // có giá trị là 101 ở hệ thập phân  ((1 * 10^2) + (0 * 10^1) + 1)
```

Hệ nhị phân (binary) là hệ cơ số 2, chỉ được biểu thị bởi 2 chữ số "0" và "1", có tiền tố "B" (đôi khi là "0B") đứng đầu

Ví dụ:

```
int a = B101;    // có giá trị là 5 ở hệ thập phân   ((1 * 2^2) + (0 * 2^1) + 1)
```

Các định dạng nhị phân chỉ hoạt động trên byte (8 bit) giữa 0 (B0) và 255 (B11111111). Nếu muốn sử dụng hệ nhị phân trên một số nguyên int (16 bít) thì phải thực hiện quy trình 2 bước như sau:

```
long myInt = (B11100000 * 256) + B11111111; // myInt = B1110000011111111
```

Hệ bát phân (octal) là hệ cơ số 8, chỉ dùng các chữ số từ 0-7 để biểu thị, có tiền tố "0" đứng đầu

```
int a = 0101;    // có giá trị là 65 ở hệ thập phân   ((1 * 8^2) + (0 * 8^1) + 1)
```

Chú ý:

Rất khó để tìm lỗi một cách dễ dàng nếu không cẩn thận hoặc không thành thạo trong việc sử dụng hệ bát phân.

Giả sử muốn khai báo một hằng số nguyên ở hệ thập phân ví dụ 161, nhưng do đặt thêm số 0 phía trước. => chương trình dịch hiểu sai hệ số đang sử dụng => sinh ra lỗi logic khiến chương trình chạy sai.

Hệ thập lục phân (hexadecimal) là hệ cơ số 16, chỉ dùng các chữ số từ 0-9 và A-F (hoặc a-f) để biểu thị; A đại diện cho 10, B là 11, lên đến F là 15. Các chữ số dùng ở hệ này có tiền tố "0x" đứng trước

Ví dụ:

```
int a = 0x101;   // có giá trị là 257 trong hệ thập phân   ((1 * 16^2) + (0 * 16^1) + 1)
```

Hậu tố U và L

Theo mặc định, một hằng số nguyên được coi là một số nguyên kiểu int với những hạn chế trong giao tiếp giữa các giá trị. Để xác định một hằng số nguyên với một kiểu dữ liệu khác phải tuân theo quy tắc sau:

- Dùng 'u' hoặc 'U' để biểu thị kiểu dữ liệu *unsigned* (không âm). Ví dụ: 33u
- Dùng 'l' hoặc 'L' để biểu thị kiểu dữ liệu *long*. Ví dụ: 100000L
- Dùng 'ul' hoặc 'UL' để biểu thị kiểu dữ liệu *unsigned long*. Ví dụ: 32767ul

4.2.2.1.8 Hằng số thực

Tương tự như hằng số nguyên, hằng số thực (floating point constants) cũng có cách làm việc và sử dụng tương tự. Khi viết một biểu thức tính toán, giá trị của biểu thức này sẽ được tính ra và trình biên dịch sẽ thay thế biểu thức này bằng một hằng số thực đã tính ra được. Điều đó gợi ý rằng trong những chương trình lớn, để giảm thời gian biên dịch nên tính trước giá trị của những biểu thức thay vì bắt trình biên dịch tính toán.

Ví dụ

```
float a = .159;
float b = 0.159;    // a = b
```

Để biểu thị những hằng số thực có giá trị cực lớn hoặc cực nhỏ phải sử dụng 2 kí hiệu khoa học là "E" và "e".

Hằng số thực	**Ý nghĩa**	**Giá trị**
10.0	10	
2.34E5	$2.34 * 10^5$	234000
67e-12	$67.0 * 10^{-12}$	0.000000000067

4.2.2.2 Kiểu dữ liệu

4.2.2.2.1 void

"void" là một từ khóa chỉ dùng trong việc khai báo một function. Những function được khai báo với "void" sẽ không trả về bất kì dữ liệu nào khi được gọi.

Ví dụ

```
led = 13;
void setup() {
  pinMode(led, OUTPUT);
}
void loop() {
  blink();
}
void blink() {
  digitalWrite(led, LOW);
  delay(1000);
  digitalWrite(led, HIGH);
  delay(1000);
}
```

Giải thích

- "blink" là một function được định nghĩa với từ khóa "void", do đó nó không trả về một giá trị nào. Nhiệm vụ của "blink" chỉ là làm nhấp nháy đèn LED ở chân số 13 trên mạch Arduino.
- Có thể thấy rằng những function kiểu này không dùng lệnh "return" để trả về giá trị của function.

4.2.2.2.2 boolean

Một biến được khai báo kiểu **boolean** sẽ chỉ nhận một trong hai giá trị: true hoặc false và sẽ mất 1 byte bộ nhớ cho điều đó.

Lưu ý

Những cặp giá trị sau là tương đương nhau. Về bản chất, chúng đều là những hằng số nguyên với 2 giá trị 0 và 1:

- true - false
- HIGH - LOW
- 1 - 0

Ví dụ:

```
int led = 13;
boolean led_status;
void setup() {
  pinMode(led, OUTPUT);
  led_status = true;     // led ở trạng thái bật
}
void loop() {
  digitalWrite(led, led_status);     // bật đèn, led_status = 1
  delay(1000);
  digitalWrite(led, !led_status);    // tắt đèn, !led_status = 0
  delay(1000);
}
```

4.2.2.2.3 char

Kiểu dữ liệu này là kiểu dữ liệu biểu diễn cho 1 KÝ TỰ (nếu cần biểu diễn một chuỗi trong chương trình Arduino cần sử dụng kiểu dữ liệu String). Kiểu dữ liệu này chiếm 1 byte bộ nhớ. Kiểu char chỉ nhận các giá trị trong bảng mã ASCII. Kiểu char được lưu dưới dạng 1 số nguyên byte có số âm (có các giá trị từ -127 - 128), thay vì thiết đặt một biến kiểu char có giá trị là 'A', có thể đặt là 65.

Ví dụ

```
char myChar = 'A';
char myChar = 65;      // cả 2 cách khai báo đều hợp lệ
```

4.2.2.2.4 unsigned char

Giống kiểu char, tuy nhiên kiểu unsigned char lại biểu hiệu một số nguyên byte không âm (giá trị từ 0 - 255).

Ví dụ

```
unsigned char myChar = 240;
```

4.2.2.2.5 byte

Là một kiểu dữ liệu biểu diễn số nguyên nằm trong khoảng từ 0 đến 255, sẽ mất 1 byte bộ nhớ cho mỗi biến mang kiểu *byte*

Ví dụ

```
byte a = 123;    //khai báo biến a mang kiểu byte, có giá trị là 123
```

4.2.2.2.6 int

Kiểu int là kiểu số nguyên chính được dùng trong chương trình Arduino. Kiểu int chiếm 2 byte bộ nhớ.

Trên mạch Arduino Uno, nó có đoạn giá trị từ -32,768 đến 32,767 (-2^{15} đến $2^{15}-1$) (16 bit)

Trên mạch Arduino Due, nó có đoạn giá trị từ -2,147,483,648 đến 2,147,483,647 (-2^{31} đến $2^{31}-1$) (32 bit) (lúc này nó chiếm 4 byte bộ nhớ)

Ví dụ

```
int ledPin = 13;
```

Cú pháp

```
int var = val;
```

var: tên biến

val: giá trị

Một số thủ thuật lập trình

```
int x;
x = -32768;
x = x - 1; // x sẽ nhận giá trị là 32767



int x;
x = 32767;
x ++; // x sẽ nhận giá trị -32768
```

4.2.2.2.7 unsigned int

Kiểu unsigned int là kiểu số nguyên nằm trong khoảng từ 0 đến 65535 (0 đến 2^{16} - 1). Mỗi biến mang kiểu dữ liệu này chiếm 2 byte bộ nhớ.

Lưu ý

Trên Arduino Due, *unsigned int* có khoảng giá trị từ 0 đến 4,294,967,295 (2^{32} - 1) (lúc này nó chiếm 4 byte bộ nhớ)

Có thể dễ dàng nhận ra rằng kiểu dữ liệu này không chứa các giá trị âm so với kiểu *int.*

Cú pháp

```
unsigned int [tên biến] = [giá trị];
```

Ví dụ

```
unsigned int ledPin = 13;
```

***Lưu ý đặc biệt* (nói chung cho các kiểu dữ liệu unsigned)**

Khi một biến kiểu *unsigned int* được gán trị vượt ngoài phạm vi giá trị (bé hơn 0 hoặc lớn hơn 65525), giá trị của biến này sẽ tự động được đẩy lên giới hạn trên hoặc giới hạn dưới trong khoảng giá trị của nó.

Ví dụ

```
unsigned int x = 0;  // x nhận giá trị trong khoảng từ 0 đến 65535
x = x - 1          // x = 0 - 1     = 65535 (giới hạn trên của x)
x = x + 1          // x = 65535 + 1 = 0     (giới hạn dưới của x)
```

4.2.2.2.8 word

Giống như kiểu unsigned int, kiểu dữ liệu này là kiểu số nguyên 16 bit không âm (chứa các giá trị từ 0 đến 65535), và nó chiếm 2 byte bộ nhớ!

Ví dụ

```
word w = 10000;
```

4.2.2.2.9 long

long là một kiểu dữ liệu mở rộng của *int*. Những biến có kiểu *long* có thể mang giá trị 32bit từ -2,147,483,648 đến -2,147,483,647, sẽ mất 4 byte bộ nhớ cho một biến kiểu *long.*

Khi tính toán với số nguyên (biến kiểu *int*) phải thêm hậu tố "L" phía sau các số nguyên kiểu *int* để chuyển chúng sang kiểu *long.* Việc tính toán (cộng, trừ, nhân,...) giữa 2 số thuộc 2 kiểu dữ liệu khác nhau là không được phép.

Ví dụ

```
long a = 10;
long b = a + 10L // b = 20
```

4.2.2.2.10 unsigned long

Kiểu unsigned long là kiểu số nguyên nằm trong khoảng từ 0 đến 4,294,967,295 (0 đến 2^{32} - 1). Mỗi biến mang kiểu dữ liệu này chiếm 4 byte bộ nhớ.

Ví dụ

```
unsigned long time;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.print("Time: ");
  time = millis();
  Serial.println(time);//Xuất thời gian lúc chạy hết đoạn lệnh trên
  delay(1000);//Dừng chương trình trong 1 giây. Lúc này sẽ tạm dừng hệ thống,  không thể chạy bất cứ lệnh gì trong thời gian delay
}
```

4.2.2.2.11 short

Giống kiểu int, tuy nhiên trên mọi mạch Arduino nó đều chiếm 4 byte bộ nhớ và biểu thị giá trị trong khoảng -32,768 đến 32,767 (-2^{15} đến $2^{15}-1$) (16 bit).

Ví dụ

```
short ledPin = 13;
```

Cú pháp

```
short var = val;
```

var: tên biến

val: giá trị

4.2.2.2.12 float

Để định nghĩa 1 kiểu số thực, có thể sử dụng kiểu dữ liệu float. Một biến dùng kiểu dữ liệu này có thể đặt một giá trị nằm trong khoảng -3.4028235E+38 đến 3.4028235E+38. Nó chiếm 4 byte bộ nhớ.

Với kiểu dữ liệu float có từ 6-7 chữ số có nghĩa nằm ở bên mỗi bên dấu ".". Điều đó có nghĩa rằng có thể đặt một số thực dài đến 15 ký tự (bao gồm dấu **.**)

Lưu ý

Để biểu diễn giá trị thực của một phép chia phải là 2 số thực chia lẫn nhau. Ví dụ: xử lý phép tính 5.0 / 2.0 thì kết quả sẽ trả về là 2.5. Nhưng nếu mà xử lý phép tính 5 / 2 thì kết quả sẽ là 2 (vì hai số nguyên chia nhau sẽ ra một số nguyên).

Ví dụ

```
float myfloat;
float sensorCalbrate = 1.117;
```

Cú pháp

```
float var = val;
```

var: tên biến

val: giá trị

Code tham khảo

```
int x;
int y;
float z;
x = 1;
y = x / 2;            // y sẽ trả về kết quả là 0
z = (float)x / 2.0;   //z sẽ có kết quả là 0.5 (nhập 2.0, chứ không phải là 2)
```

4.2.2.2.13 double

Giống như kiểu float, nhưng trên mạch Arduino Due thì kiểu double lại chiếm đến 8 byte bộ nhớ (64 bit). Vì vậy hãy cẩn thận khi sử dụng kiểu dữ liệu này.

4.2.2.2.14 Array

Array là mảng (tập hợp các giá trị có liên quan và được đánh dấu bằng những chỉ số). Array được dùng trên Arduino chính là Array trong ngôn ngữ lập trình C.

Các cách khởi tạo một mảng

```
int myInts[6]; // tạo mảng myInts chứa tối đa 6 phần tử (được đánh dấu từ 0-5), các phần tử này đều có kiểu là int => khai báo này chiếm 2*6 = 12 byte bộ nhớ
int myPins[] = {2, 4, 8, 3, 6}; // tạo mảng myPins chứa 5 phần tử (lần lượt là 2, 4, 8, 3, 6). Mảng này không giới hạn số lượng phần tử vì có khai báo là "[]"
int mySensVals[6] = {2, 4, -8, 3, 2}; // tạo mảng mySensVals chứa tối đa 6 phần tử, trong đó 5 phần tử đầu tiên có giá trị lần lượt là 2, 4, -8, 3, 2
char message[6] = "hello"; // tạo mảng ký tự (dạng chuỗi) có tối đa 6 ký tự!
```

Truy cập các phần tử trong mảng

Chú ý: Phần tử đầu tiên trong mảng luôn được đánh dấu là **số 0**.

```
mySensVals[0] == 2, mySensVals[1] == 4, vâng vâng
```

Điều này có nghĩa rằng, việc khai báo một mảng có tối đa 10 phần tử, thì phần tử cuối cần (thứ 10) được đánh dấu là **số 9**

```
int myArray[10]={9,3,2,4,3,2,7,8,9,11};
  // myArray[9]    có giá trị là 11
  // myArray[10]   sẽ trả về một giá trị "hên xui" nằm trong khoảng giá trị của int
```

Vì vậy, hãy chú ý trong việc truy cập đến giá trị trong mảng, nếu muốn truy cập đến phần tử cuối cùng thì hãy truy đến đến ô giới hạn của mảng - 1.

Trong trình biên dịch ngôn ngữ C, nó không kiểm tra việc có truy cập đến một ô có nằm trong bộ nhớ hay không. Nên nếu không cẩn thận trong việc truy cập mảng, chương trình sẽ mắc lỗi logic và rất khó để tìm lỗi.

Gán một giá trị cho một phần tử

```
mySensVals[0] = 10;
```

Đọc một giá trị của một phần tử và gán cho một biến nào đó cùng kiểu dữ liệu

```
x = mySensVals[0]; //10
```

Dùng mảng trong vòng lặp

Mảng rất thường được dùng trong vòng lặp (chẳng hạn như dùng để lưu các chân digital quản lý đèn led). Trong đó, biến chạy của hàm for sẽ đi hết (hoặc một phần) của mảng.

Ví dụ về việc in 5 phần tử đầu của mảng myPins:

```
int i;
for (i = 0; i < 5; i = i + 1) {
  Serial.println(myPins[i]);
}
```

4.2.2.2.15 string

Trong một chương trình Arduino có 2 cách để định nghĩa chuỗi, cách thứ nhất là sử dụng mảng ký tự để biểu diễn chuỗi. Ở đây tả chi tiết về cách thứ nhất.

Cách khai báo

```
char Str1[15]; // khai bảo chuỗi có độ dài là 15 ký tự.
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'}; //khai báo chuỗi có độ dài tối đa là 8 ký tự và đặt nó giá trị ban đầu là arduino (7 ký tự). Buộc phải khai báo chuỗi nằm giữa hai dấu nháy đơn nhé!
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};//khai báo chuỗi có độ dài tối đa là 8 ký tự và đặt nó giá trị ban đầu là arduino<ký tự null>  (8 ký tự)
char Str4[ ] = "arduino";// Chương trình dịch sẽ tự động điều chỉnh kích thước cho chuỗi Str4 này và ngoài ra phải đặt một chuỗi trong dấu ngoặc kép
char Str5[8] = "arduino";// Một cách khai báo như Str3
char Str6[15] = "arduino"; // Một cách khai báo khác với độ dài tối đa lớn hơn
```

CHÚ Ý: mỗi chuỗi đều cần có 1 ký tự NULL, nếu không khai báo ký tự NULL (\0) ở cuối thì trình biên dịch sẽ tự động thêm vào. Đó là lý do vì sao Str2, Str4 lại có độ dài là 8 nhưng chỉ chứa một chuỗi 7 ký tự. Ký tự NULL này dùng để trình biên dịch biết điểm dừng của một chuỗi. Nếu không nó sẽ đọc tiếp những phần bộ nhớ khác (mà phần ấy không lưu chuỗi).

Có thể khai bảo một chuỗi dài như sau:

```
char myString[] = "This is the first line"
" this is the second line"
" etcetera"
```

Mảng chuỗi

Khi cần phải thao tác với một lượng lớn chuỗi (ví dụ như trong các ứng dụng trả lời người dùng bằng LCD) thì cần sử dụng một mảng chuỗi. Mà bản chất của chuỗi là mảng các ký tự. Vì vậy để khai báo 1 mảng chuỗi cần sử dụng một mảng 2 chiều!

Để khai báo một mảng chuỗi, rất đơn giản:

```
char* myStrings[] = {"I'm number 1", "I'm number 2"};
```

Chỉ cần thêm dấu * sau chữ char và trong dấu ngoặc vuông phía sau myStrings có thể thiết đặt số lượng phần tử tối đa của mảng chuỗi!

Ví dụ

```
char* myStrings[]={"This is string 1", "This is string 2", "This is string 3",
"This is string 4", "This is string 5","This is string 6"};


void setup(){
Serial.begin(9600);
}
void loop(){
for (int i = 0; i < 6; i++){
  Serial.println(myStrings[i]);
  delay(500);
  }
}
```

4.2.2.2.16 object String

Nếu muốn lưu một chuỗi nào đó, có thể sử dụng kiểu string (mảng ký tự) hoặc sử dụng object String. Với object String có thể làm nhiều việc hơn, chẳng hạn như cộng chuỗi, tìm kiếm chuỗi, xóa chuỗi,.... Tuy nhiên, cũng sẽ tốn nhiều bộ nhớ hơn, nhưng ngược lại sẽ có nhiều hàm hỗ trợ cho việc xử lý chuỗi này

Để phân biệt giữa mảng chuỗi (string) và object String rất đơn giản. string là chuỗi với chữ "s" thường còn object String là chữ "S" hoa. Ngoài ra, chuỗi được định nghĩa trong hai dấu nháy kép (ví dụ như "Arduino ") được xem làm mảng chuỗi string.

4.2.2.3 Chuyển đổi kiểu dữ liệu

4.2.2.3.1 char()

Hàm char() có nhiệm vụ chuyển kiểu dữ liệu của một giá trị về kiểu char.

Cú pháp

```
char(x)
```

Tham số

x: là một giá trị thuộc bất kỳ kiểu dữ liệu nào

Trả về

Giá trị thuộc kiểu char

4.2.2.3.2 byte()

Hàm byte() có nhiệm vụ chuyển kiểu dữ liệu của một giá trị về kiểu byte.

Cú pháp

```
byte(x)
```

Tham số

x: là một giá trị thuộc bất kỳ kiểu dữ liệu nào

Trả về

Giá trị thuộc kiểu byte

4.2.2.3.3 int ()

Hàm int() có nhiệm vụ chuyển kiểu dữ liệu của một giá trị về kiểu int.

Cú pháp

```
int(x)
```

Tham số

x: là một giá trị thuộc bất kỳ kiểu dữ liệu nào

Trả về

Giá trị thuộc kiểu int

4.2.2.3.4 word()

Hàm word() có nhiệm vụ chuyển kiểu dữ liệu của một giá trị về kiểu word. Hoặc ghép 2 giá trị thuộc kiểu byte thành 1 giá trị kiểu word

Cú pháp

word(x)

word(8bitDauTien,8bitSauCung)

Tham số

x: là một giá trị thuộc bất kỳ kiểu dữ liệu nào

8bitDauTien, 8bitSauCung: là giá trị kiểu byte

Trả về

Giá trị thuộc kiểu word

4.2.2.3.5 long()

Hàm long() có nhiệm vụ chuyển kiểu dữ liệu của một giá trị về kiểu long.

Cú pháp

long(x)

Tham số

x: là một giá trị thuộc bất kỳ kiểu dữ liệu nào

Trả về

Giá trị thuộc kiểu long

4.2.2.3.6 float()

Hàm float() có nhiệm vụ chuyển kiểu dữ liệu của một giá trị về kiểu float.

Cú pháp

float(x)

Tham số

x: là một giá trị thuộc bất kỳ kiểu dữ liệu nào

Trả về

Giá trị thuộc kiểu float

4.2.2.4 Phạm vi của biến và phân loại biến

4.2.2.4.1 Phạm vi của biến

Ngôn ngữ Arduino được xây dựng trên ngôn ngữ lập trình C. Các biến của Arduino, cũng như C, có một phạm trù được gọi là phạm vi biến. Điều này trái ngược với ngôn ngữ BASIC, ở ngôn ngữ BASIC này, mọi biến đều là biến toàn cục.

Một biến toàn cục có nghĩa là, tất cả mọi nơi trong chương trình có thể đọc được và thay đổi dữ liệu của nó mà không cần sử dụng biện pháp hỗ trợ nào. Còn biến cục bộ thì chỉ có có hàm khai báo nó (hoặc các hàm con của hàm đó) có thể thấy và thay đổi được giá trị. Ví dụ,

mọi biến nằm ngoài các hàm (như setup() hay loop()) là biến toàn cục, còn nằm bên trong các hàm là biến cục bộ của hàm đó.

Khi chương trình dần trở nên lớn hơn (về kích thước file lập trình) hoặc phức tạp hơn thì bạ nên dùng các biến cục để trong các hàm để dễ dàng quản lý (thay cho việc khai báo hết toàn bộ là biến toàn cục). Biến cục bộ rất có ích trong việc này vì chỉ có mỗi hàm khai báo nó (và các hàm con) mới sử dụng được nó. Điều này sẽ ngăn chặn các lỗi về logic sẽ xảy ra nếu một hàm thay đổi giá trị của một hàm khác. Ngoài ra, sau khi đoạn chương trình con kết thúc, các biến cục bộ sẽ được tự động giải phóng khỏi bộ nhớ, chương trình chính sẽ có thêm vùng nhớ cho việc xử lý.

Biến cục độ khá hữu ích cho việc khai báo biến của vòng lặp vì chỉ có vòng lặp mới dùng được nó.

Ví dụ

```
int gPWMval;  // mọi hàm đều có thể thao tác với biến này
void setup()
{
  // ...
}
void loop()
{
  int i;    //  biến "i" chỉ có thể được thao tác bên trong hàm loop()
  float f;  // biến "f" chỉ có thể được thao tác bên trong hàm loop()
  // ...
  for (int j = 0; j <100; j++){
     //biến "j" chỉ có thể được thao tác bên trong vòng lặp này
  }
}
```

4.2.2.4.2 static - biến tĩnh

Biến tĩnh là biến sẽ được tạo ra duy nhất một lần khi gọi hàm lần đầu tiên và nó sẽ không bị xóa đi để tạo lại khi gọi lại hàm ấy. Đây là sự khác biệt giữa biến tĩnh và biến cục bộ.

Biến tĩnh là loại biến lưỡng tính, vừa có tính chất của 1 biến toàn cục, vừa mang tính chất của 1 biến cục bộ:

- Tính chất 1 biến toàn cục: biến không mất đi khi chương trình con kết thúc, nó vẫn nằm trong ô nhớ của chương trình và được tự động cập nhật khi chương trình con được gọi lại. Giống như 1 biến toàn cục vậy.

- Tính chất 1 biến cục bộ: biến chỉ có thể được sử dụng trong chương trình con mà nó được khai báo.

  Để khai báo chỉ cần thêm từ khóa "static" trước khai báo biến.

  Ví dụ

```
void setup(){
  Serial.begin(9600); // Khởi tạo cổng Serial ở baudrate 9600
}
void loop() {
  testStatus();// Chạy hàm testStatus
  delay(500); // dừng 500 giây để thấy được sự thay đổi
}
void testStatus() {
  static int a = 0;// Khi khai báo biến "a" là biến tĩnh
  // thì duy nhất chỉ có 1 lần đầu tiên khi gọi hàm testStatus
  // là biến "a" được tạo và lúc đó ta gán "a" có giá trị là 0
  a++;
  Serial.println(a);
  // Biến a sẽ không bị mất đi khi chạy xog hàm testStatus
  // Đó là sự khác biệt giữa biến tĩnh và biến cục bộ!
}
```

4.2.2.4.3 const - biến hằng

Với một từ khóa "const" nằm trước một khai báo biến, sẽ làm cho biến này thành một biến chỉ có thể đọc "read-only". Nếu thay đổi giá trị của một biến hằng thì chương trình dịch sẽ báo lỗi.

Các biến có từ khóa const vẫn tuân theo phạm vi hiệu lực của biến. Ngoài cách sử dụng const để khai báo một biến hằng, ta còn có thể sử dụng #define để khai báo một **hằng số hoặc hằng chuỗi**. Tuy nhiên sử dụng const được ưa chuộng hơn trong lập trình, vì khả năng tuân theo phạm vi hiệu lực của biến.

Ví dụ

```
const float pi = 3.14;
float x;
// ....
x = pi * 2;    // Có thể dụng hằng số pi trong tính toán - vì đơn giản chỉ đọc nó
```

```
pi = 7;        // lỗi  vì không thể thay đổi giá trị của một hằng số
```

Dùng const hay dùng #define ?

Để khai báo một biến hằng số (nguyên / thực) hoặc hằng chuỗi thì có thể dùng cả 2 cách đều được. Tuy nhiên, để khai báo một biến mảng (array) là một hằng số chỉ có thể sử dụng từ khóa const.

4.2.2.4.4 volatile

Một biến được nên khai báo với từ khóa ***volatile*** nếu giá trị của nó có thể bị tác động bởi các yếu tố nằm ngoài sự kiểm soát của chương trình đang chạy, nói đơn giản hơn là giá trị của biến thay đổi một cách không xác định. Các biến kiểu ***volatile*** thường là:

1. *Memory-mapped peripheral registers* (thanh ghi ngoại vi có ánh xạ đến ô nhớ)
2. *Biến toàn cục được truy xuất từ các tiến trình con xử lý ngắt* (Interrupt Service Routine - ISR)
3. *Biến toàn cục được truy xuất từ nhiều tác vụ trong một ứng dụng thực thi đa luồng (*Concurrently Executing Thread).

Trong lập trình Arduino mà cụ thể hơn là trong việc giao tiếp với các chip Atmega, nơi duy nhất xảy ra sự tác động không dự báo trước đến giá trị các biến là những phần chương trình có sự giao tiếp với các ***ngắt (interrupts).*** Người ta ngăn sự thay đổi bất thường này bằng cách khai báo từ khóa ***volatile*** khi khai báo biến - cách mà sẽ đưa biến lên lưu trữ ở RAM để xử lí tạm thời thay vì lưu trữ ở các thanh ghi (register) bị ảnh hưởng bởi ***interrupts***. Những biến này thuộc kiểu **số 2** kể trên.

Khai báo ví dụ

```
volatile int state;
```

Chương trình mẫu

Chương trình sau sẽ thực hiện bật tắt đèn LED ở chân Digital 13 khi chân Interrupt 0 trên Arduino thay đổi trạng thái

```
int pin = 13;
volatile int state = LOW;
void setup() {
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}
void loop() {
  digitalWrite(pin, state);
}
void blink() {
```

```
  state = !state;
}
```

4.2.2.5 Hàm hỗ trợ sizeof()

Hàm sizeof() có nhiệm vụ trả về số byte bộ nhớ của một biến, hoặc là trả về tổng số byte bộ nhớ của một mảng array.

Cú pháp

```
sizeof(variable)
```

Tham số

variable: mọi kiểu dữ liệu hoặc mọi biến (thuộc bất cứ kiểu dữ liệu nào) hoặc một mảng.

Ví dụ

Hàm sizeof() tỏ ra rất hiệu quả trong việc kiểm tra độ dài chuỗi, nhưng cần lưu ý cho về ký tự "cần câu" của Arduino. Sau đây là một ví dụ về việc đọc từng giá trị của một chuỗi cho trước. Để thấy được hiệu quả chương trình hãy thử thay chuỗi trong ví dụ bằng một chuỗi khác xem.

```
char myStr[] = "this is a test";
int i;
void setup(){
  Serial.begin(9600);
}
void loop() {
  for (i = 0; i < sizeof(myStr) - 1; i++){
    Serial.print(i, DEC);
    Serial.print(" = ");
    Serial.write(myStr[i]);
    Serial.println();
  }
  delay(5000); // làm chậm chương trình để thấy được chương trình này muốn nói lên điều gì
}
```

Lưu ý

Vì hàm sizeof sẽ trả về số byte bộ nhớ của một biến hay một mảng nào đó, vì vậy nếu muốn ĐẾM Số phần tử của một mảng số nguyên có kiểu dữ liệu > 1 byte (như là: int, word, float,...) thì cần chia số bộ nhớ của mảng cho số bộ nhớ của kiểu dữ liệu của mảng đó. Ví dụ một mảng có kiểu int.

```
for (i = 0; i < (sizeof(myInts)/sizeof(int)) - 1; i++) {
  // hàm làm gì đó với biến myInts[i]
}
```

4.2.3 Hàm và thủ tục

4.2.3.1 Nhập xuất Digital (Digital I/O)

4.2.3.1.1 pinMode()

Cấu hình 1 pin quy định hoạt động như là một đầu vào (INPUT) hoặc đầu ra (OUTPUT). Xem mô tả kỹ thuật số (datasheet) để biết chi tiết về các chức năng của các chân.

Như trong phiên bản Arduino 1.0.1, nó có thể kích hoạt các điện trở pullup nội bộ với chế độ INPUT_PULLUP. Ngoài ra, chế độ INPUT vô hiệu hóa một cách rõ ràng điện trở pullups nội bộ.

Cú pháp

```
pinMode(pin, mode)
```

Thông số

pin: Số của chân digital muốn thiết đặt

mode: INPUT, INPUT_PULLUP hoặc OUTPUT

Trả về

không

Ví dụ

```
int ledPin = 13;                // đèn LED được kết nối với chân digital 13
void setup()
{
  pinMode(ledPin, OUTPUT);      // thiết đặt chân ledPin là OUTPUT
}
void loop()
{
  digitalWrite(ledPin, HIGH);   // bật đèn led
  delay(1000);                  // dừng trong 1 giây
  digitalWrite(ledPin, LOW);    // tắt đèn led
  delay(1000);                  // dừng trong 1 giây
}
```

4.2.3.1.2 digitalWrite()

Xuất tín hiệu ra các chân digital, có 2 giá trị là HIGH hoặc là LOW

Nếu một pin được thiết đặt là OUTPUT bởi pinMode(). Và dùng digitalWrite để xuất tín hiệu thì điện thế tại chân này sẽ là 5V (hoặc là 3,3 V trên mạch 3,3 V) nếu được xuất tín hiệu là HIGH, và 0V nếu được xuất tín hiệu là 0V.

Nếu một pin được thiết đặt là INPUT bởi pinMode(). Lúc này digitalWrite sẽ bật (HIGH) hoặc tắt (LOW) hệ thống điện trở pullup nội bộ.

Cú pháp

```
digitalWrite(pin,value)
```

Thông số

pin: Số của chân digital muốn thiết đặt

value: HIGH hoặc LOW

Trả về

không

Ví dụ

```
int ledPin = 13;                // đèn LED được kết nối với chân digital 13
void setup()
{
  pinMode(ledPin, OUTPUT);      // thiết đặt chân ledPin là OUTPUT
}
void loop()
{
  digitalWrite(ledPin, HIGH);   // bật đèn led
  delay(1000);                  // dừng trong 1 giây
  digitalWrite(ledPin, LOW);    // tắt đèn led
  delay(1000);                  // dừng trong 1 giây
}
```

4.2.3.1.3 digitalRead()

Đọc tín hiệu điện từ một chân digital (được thiết đặt là INPUT). Trả về 2 giá trị HIGH hoặc LOW.

Cú pháp

```
digitalRead(pin)
```

Thông số

pin: giá trị của digital muốn đọc

Trả về

HIGH hoặc LOW

Ví dụ

Ví dụ này sẽ làm cho đèn led tại pin 13 nhận giá trị như giá trị tại pin 2

```
int ledPin = 13; // chân led 13
int inPin = 2;   // button tại chân 2
int val = 0;     // biến "val" dùng để lưu tín hiệu từ digitalRead
void setup()
{
  pinMode(ledPin, OUTPUT);      // đặt pin digital 13 là output
  pinMode(inPin, INPUT);      // đặt pin digital 2 là input
}
void loop()
{
  val = digitalRead(inPin);   //  đọc tín hiệu từ digital2
  digitalWrite(ledPin, val);    // thay đổi giá trị của đèn LED là giá trị của digital 2
}
```

4.2.3.2 Nhập xuất Analog (Analog I/O)

4.2.3.2.1 analogReference()

Hàm analogReference() có nhiệm vụ đặt lại mức (điện áp) tối đa khi đọc tín hiệu analogRead. Ứng dụng như sau, giả sử cần đọc một tín hiệu dạng analog có hiệu điện thế từ 0-1,1V. Nhưng mà nếu dùng mức điện áp tối đa mặc định của hệ thống (5V) thì khoảng giá trị sẽ ngắn hơn => độ chính xác kém hơn => hàm này dùng để giải quyết việc đó.

Cú pháp

```
analogReference(type)
```

type: một trong các kiểu giá trị sau: DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56, hoặc EXTERNAL

Kiểu	Nhiệm vụ đảm nhiệm	Ghi chú
DEFAULT	Đặt mức điện áp tối đa là 5V (nếu trên mạch dùng nguồn 5V làm nuôi chính) hoặc là 3,3V (nếu trên mạch dùng nguồn 3,3V làm nguồn nuôi chính)	
INTERNAL	Đặt lại mức điện áp tối đa là 1,1 V (nếu sử dụng vi điều khiển ATmega328 hoặc ATmega168) Đặt lại mức điện áp tối đa là 2,56V (nếu sử dụng vi điều khiển ATmega8)	
INTERNAL1V1	Đặt lại mức điện áp tối đa là 1,1 V	Chỉ có trên Arduino Mega
INTERNAL2V56	Đặt lại mức điện áp tối đa là 2,56 V	Chỉ có trên Arduino Mega
EXTERNAL	**Đặt lại mức điện áp tối đa BẰNG với mức điện áp được cấp vào chân AREF**	**Chỉ được cấp vào chân AREF một điện áp nằm trong khoảng 0-5V**

Trả về

không

Cảnh báo

NẾU sử dụng kiểu EXTERNAL cho hàm analogReference thì **BUỘC** phải cấp nó một nguồn nằm trong khoảng từ 0-5V, và nếu đã cấp một nguồn điện thỏa mãn điều kiện trên vào chân AREF thì **BUỘC** phải gọi dòng lệnh analogReference(EXTERNAL) trước khi sử dụnganalogRead(), NẾU KHÔNG MẠCH CHÁY

Ngoài ra, có thể sử dụng một điện trở 5kΩ đặt trước chân AREF rồi đặt nguồn điện ngoài (điện áp muốn cấp vào chân AREF). Bởi vì chỗ gắn chân AREF có một nội điện trở (điện trở có sẵn trong mạch) khoảng 32kΩ => sẽ tạo ra mạch giảm áp => giảm điện thế gắn vào chân AREF => Mạch không cháy nếu có lỡ gắn nguồn hơn 5V.

4.2.3.2.2 analogRead()

Nhiệm vụ của analogRead() là đọc giá trị điện áp từ một chân Analog (ADC). Trên mạch Arduino UNO có 6 chân Analog In, được kí hiệu từ A0 đến A5. Trên các mạch khác cũng có những chân tương tự như vậy với tiền tố "A" đứng đầu, sau đó là số hiệu của chân.

analogRead() **luôn** trả về 1 số nguyên nằm trong khoảng từ 0 đến 1023 tương ứng với thang điện áp (mặc định) từ 0 đến 5V. Có thể điều chỉnh thang điện áp này bằng hàm analogReference().

Hàm analogRead() cần 100 micro giây để thực hiện.

Khi nói "đọc tín hiệu analog", có thể hiểu đó chính là việc đọc giá trị điện áp.

Cú pháp

```
analogRead([chân đọc điện áp]);
```

Ví dụ

```
int voltage = analogRead(A0);
```

Trong đó A0 là chân dùng để đọc điện áp.

Nếu chưa kết nối chân đọc điện áp, hàm analogRead() sẽ trả về một giá trị ngẫu nhiên trong khoảng từ 0 đến 1023. Để khắc phục điều này, phải mắc thêm một điện trở có trị số lớn (khoảng 10k ohm trở lên) hoặc một tụ điện 104 từ chân đọc điện áp xuống GND.

4.2.3.2.3 analogWrite()

analogWrite() là lệnh xuất ra từ một chân trên mạch Arduino một mức tín hiệu analog (phát xung PWM). Người ta thường điều khiển mức sáng tối của đèn LED hay hướng quay của động cơ servo bằng cách phát xung PWM như thế này.

Không cần gọi hàm *pinMode()* để đặt chế độ OUTPUT cho chân sẽ dùng để phát xung PWM trên mạch Arduino.

Cú pháp

```
analogWrite([chân phát xung PWM], [giá trị xung PWM]);
```

Giá trị mức xung PWM nằm trong khoảng từ 0 đến 255, tương ứng với mức duty cycle từ 0% đến 100%

analogWrite	tỉ lệ	chu kì xung
analogWrite(0)	0/255	0%
analogWrite(64)	64/255	25%
analogWrite(127)	127/255	50%
analogWrite(191)	191/255	75%
analogWrite(255)	255/255	100%

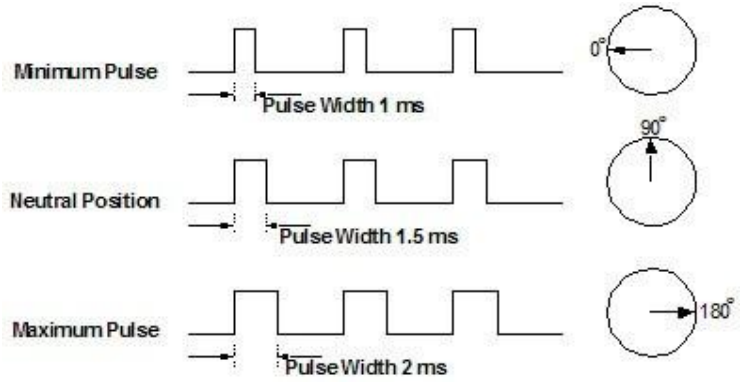
Hàm analogWrite() trong Arduino giúp việc tạo 1 xung dễ dàng hơn. Hàm này truyền vào tham số cho phép thay đổi chu kì xung, có thể tính toán ra được chu kì xung như ở bảng trên. Tần số xung được Arduino thiết lập mặc định.

Đối với board Arduino Uno, xung trên các chân 3,9,10,11 có tần số là 490Hz, xung trên chân 5,6 có tần số 980Hz.

Xung PPM khác với PWM ở chỗ:

1. tần số thông thường có giá trị trong khoảng 50Hz (20 mili giây), không quan trọng
2. thời gian xung ở mức cao chỉ từ 1ms đến 2ms, rất quan trọng.
3. có thể có nhiều hơn 1 sự thay đổi trạng thái điện cao/thấp

Nắm bắt được 2 ý trên ta đã có thể phân biệt được xung PPM và xung PWM giống nhau và khác nhau như thế nào.



Thời gian xung ở mức cao quy định góc quay của RC servo.

Với thời gian 1ms mức cao, góc quay của servo là 0, 1.5ms góc quay 90 và 2ms góc quay là 180. Các góc khác từ 0-180 được xác định trong khoảng thời gian 1-2ms.

Lưu ý: có thể ghép nhiều xung trong cùng 1 thời gian là 20ms để xác định vị trí góc của nhiều servo cùng 1 lúc. Tối đa là 10 servo.

Ví dụ

```
int led = 11;
void setup() {
}

void loop() {
   for (int i = 0; i <= 255; i++) {
      analogWrite(led,i);
      delay(20);
   }
}
```

Đoạn code trên có chức năng làm sáng dần một đèn LED được kết nối vào chân số 11 trên mạch Arduino.

4.2.3.3 Hàm thời gian

4.2.3.3.1 millis()

millis() có nhiệm vụ trả về một số - là thời gian (tính theo mili giây) kể từ lúc mạch Arduino bắt đầu chương trình. Nó sẽ tràn và quay về số 0 (sau đó tiếp tục tăng) 50 ngày.

Tham số

không

Trả về

một số nguyên kiểu unsigned long là thời gian kể từ lúc thương trình Arduino được khởi động

Ví dụ

```
unsigned long time;
void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.print("Time: ");
  time = millis();
  // in ra thời gian kể từ lúc chương trình được bắt đầu
  Serial.println(time);
  // đợi 1 giây trước khi tiếp tục in
  delay(1000);
}
```

Lưu ý quan trọng:

Các hàm về thời gian trong Arduino gồm millis() và micros() sẽ bị tràn số sau 1 thời gian sử dụng. Với hàm millis() là khoảng 50 ngày. Tuy nhiên, do là kiểu số nguyên không âm (unsigned long) nên ta dễ dàng khắc phục điều này bằng cách sử dụng hình thức ép kiểu.

```
unsigned long time;
byte ledPin = 10;
void setup()
{
   // khởi tạo giá trị biến time là giá trị hiện tại
   // của hàm millis();
   time = millis();
   pinMode(ledPin, OUTPUT);
```

```
  digitalWrite(ledPin, LOW);
}
void loop()
{
  // Lưu ý các dấu ngoặc khi ép kiểu
  // đoạn chương trình này có nghĩa là sau mỗi 1000 mili giây
  // đèn Led ở chân số 10 sẽ thay đổi trạng thái
  if ( (unsigned long) (millis() - time) > 1000)
  {
    // Thay đổi trạng thái đèn led
    if (digitalRead(ledPin) == LOW)
    {
      digitalWrite(ledPin, HIGH);
    } else {
      digitalWrite(ledPin, LOW);
    }
    // cập nhật lại biến time
    time = millis();
  }
}
```

Thông thường, nếu ta có 2 số A, B và B lớn hơn A (B > A) thì phép trừ thu được A-B là một số âm. Nhưng khi ép kiểu unsigned long là kiểu số nguyên dương, không có số âm nên giá trị trả về là 1 số nguyên dương lớn.

Ví dụ: kết quả của phép trừ:

```
unsigned long ex = (unsigned long) (0 - 1);
```

là 4294967295, con số này chính là giá trị lớn nhất của kiểu số unsigned long.

4.2.3.3.2 micros()

micros() có nhiệm vụ trả về một số - là thời gian (tính theo micro giây) kể từ lúc mạch Arduino bắt đầu chương trình. Nó sẽ tràn và quay về số 0 (sau đó tiếp tục tăng) sau 70 phút. Tuy nhiên, trên mạch Arduino 16MHz (ví dụ Duemilanove và Nano) thì giá trị của hàm này tương đương 4 đơn vị micro giây. Ví dụ micros() trả về giá trị là 10 thì có nghĩa chương trình đã chạy được 40 microgiây. Tương tự, trên mạch 8Mhz (ví dụ LilyPad), hàm này có giá trị tương đương 8 micro giây.

Lưu ý: 10^6 micro giây = 1 giây

Tham số

không

Trả về

một số nguyên kiểu unsigned long là thời gian kể từ lúc thương trình Arduino được khởi động

Ví dụ

```
unsigned long time;
void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.print("Time: ");
  time = micros();
  // in ra thời gian kể từ lúc chương trình được bắt đầu
  Serial.println(time);
  // đợi 1 giây trước khi tiếp tục in
  delay(1000);
}
```

4.2.3.3.3 delay()

delay có nhiệm vụ dừng chương trình trong thời gian mili giây. Và cữ mỗi 1000 mili giây = 1 giây.

Cú pháp

```
delay(ms)
```

Thông số

ms: thời gian ở mức mili giây. ms có kiểu dữ liệu là unsigned long

Trả về

Không

4.2.3.3.4 delayMicroseconds()

delayMicroseconds có nhiệm vụ dừng chương trình trong thời gian micro giây. Và cữ mỗi 1000000 microgiây giây = 1 giây.

Cú pháp

```
delayMicroseconds(micro)
```

Thông số

micro: thời gian ở mức micro giây. micro có kiểu dữ liệu là unsigned int. micro phải <= 16383. Con số này là mức tối đa của hệ thống Arduino, và có thể sẽ được điều chỉnh tăng trong tương lai. Và nếu muốn dừng chương trình lâu hơn thì cần dùng hàm delay

Trả về

không

Ví dụ

```
int outPin = 8;                 // digital pin 8


void setup()
{
  pinMode(outPin, OUTPUT);      // đặt là output
}
void loop()
{
  digitalWrite(outPin, HIGH);   // xuất 5V
  delayMicroseconds(50);        // đợi 50 micro giây
  digitalWrite(outPin, LOW);    // xuất 0V
  delayMicroseconds(50);        // đợi 50 micro giây
}
```

Ví dụ cho ta một cách để tạo một xung PWM tại chân số 8.

4.2.3.4 Hàm toán học

4.2.3.4.1 min()

Hàm min có nhiệm vụ trả về giá trị nhỏ nhất giữa hai biến.

Cú pháp

```
min(x, y);
```

Tham số

x: số thứ nhất, mọi kiểu dữ liệu đều được chấp nhận.

y: số thứ hai, mọi kiểu dữ liệu đều được chấp nhận.

Trả về

Số nhỏ nhất trong 2 số.

Gợi ý

Hàm min được dùng để lấy chặn trên (không để giá trị vượt quá một mức quy định nào đó).

Cảnh báo cú pháp

```
min(a++, 100);   // nếu nhập như thế này thì sẽ bị lỗi

a++;

min(a, 100);    // hãy nhập như thế này, và hãy ghi nhớ là không được để bất cứ phép tính nào bên trong hàm này
```

4.2.3.4.2 max()

Hàm max có nhiệm vụ trả về giá trị lớn nhất giữa hai biến.

Cú pháp

```
max(x, y);
```

Tham số

x: số thứ nhất, mọi kiểu dữ liệu đều được chấp nhận.

y: số thứ hai, mọi kiểu dữ liệu đều được chấp nhận.

Trả về

Số lớn nhất trong 2 số.

Gợi ý

Hàm max được dùng để lấy chặn dưới (không để giá trị tụt xuống quá một mức quy định nào đó).

Cảnh báo cú pháp

```
max(a--, 0);   // nếu nhập như thế này thì sẽ bị lỗi

a--;

max(a, 0);    // hãy nhập như thế này, và hãy ghi nhớ là không được để bất cứ phép tính nào bên trong hàm này
```

4.2.3.4.3 abs()

Hàm abs có nhiệm vụ trả về giá trị tuyệt đối của một số.

Cú pháp

```
abs(x);
```

Tham số

x: một số bất kỳ

Trả về

Nếu x >= 0, thì trả về x còn ngược lại là trả về -x

Cảnh báo cú pháp

```
abs(a--);   // nếu nhập như thế này thì sẽ bị lỗi
a--;
abs(a);     // hãy nhập như thế này, và hãy ghi nhớ là không được để bất cứ phép tính nào bên trong hàm này
```

4.2.3.4.4 map()

map() là hàm dùng để chuyển một giá trị từ thang đo này sang một giá trị ở thang đo khác. Gía trị trả về của hàm map() luôn là một số nguyên.

Cú pháp

```
map(val,A1,A2,B1,B2);
```

Trong đó:

- val là giá trị cần chuyển đổi
- A1, A2 là giới hạn trên và dưới của thang đo hiện tại
- B1,B2 là giới hạn trên và dưới của thang đo cần chuyển tới

Ví dụ

```
//Chuyển đổi 37 độ C sang độ F
int C_deg = 37;
int F_deg = map(37,0,100,32,212);   //F_deg = 98
```

4.2.3.4.5 pow()

pow() là hàm dùng để tính lũy thừa của một số bất kì (có thể là số nguyên hoặc số thực tùy ý). pow() trả về kết quả tính toán này.

Cú pháp

```
pow([cơ số], [lũy thừa]);
```

Ví dụ

```
int luythua1 = pow(2,3);
float luythua2 = pow(1.2,2.3);
double luythua3 = pow(1.11111,1.11111);
//luythua1 = 8      (=23)
//luythua2 = 1.52   (=1.22.3)
//luythua3 = 1.12   (=1.111111.11111)
```

Chú ý

Cả 2 tham số đưa vào hàm pow() đều được định nghĩa là kiểu số thực float. Kết quả trả về của pow() được định nghĩa là kiểu số thực double

4.2.3.4.6 sqrt()

sqrt() là hàm dùng để tính căn bậc 2 của một số bất kì (có thể là số nguyên hoặc số thực tùy ý) và trả về kết quả này.

Cú pháp

```
sqrt([số cần tính căn bậc 2]);
```

Ví dụ

```
int v1 = sqrt(9);
float v2 = sqrt(6.4);
double v3 = sqrt(6.5256);
int v4 = sqrt(-9);
float v5 = sqrt(-6.4);
//v1 = 3
//v2 = 2.53
//v3 = 2.55
//v4 = 0
//v5 = NaN    (tham khảo hàm isnan())
```

Chú ý

Tham số đưa vào hàm sqrt() có thể là bất kì kiểu dữ liệu biểu diễn số nào. Kết quả trả về của sqrt() được định nghĩa là kiểu số thực double hoặc NaN nếu tham số đưa vào là số thực bé hơn 0.

4.2.3.4.7 sq()

Hàm sq() được dùng để tính bình phường của một số bất kì, số này có thể thuộc bất kì kiểu dữ liệu biển diễn số nào. sq() trả về giá trị mà nó tính được với kiểu dữ liệu giống như kiểu dữ liệu của tham số ta đưa vào.

Cú pháp

```
sq([số cần tính bình phương]);
```

Ví dụ

```
int binhphuong1 = sq(5);
int binhphuong2 = sq(-5);
float binhphuong3 = sq(9.9);
```

```
float binhphuong4 = sq(-9.9);
//binhphuong1 = 25
//binhphuong2 = 25
//binhphuong3 = 98.01
//binhphuong4 = 98.01
```

4.2.3.4.8 isnan()

isnan sẽ trả về là true nếu giá trị cần kiểm tra không phải là một biểu thức toán học đúng đắn. Chữ nan có nghĩa là Not-A-Number.

Cú pháp

```
isnan(double x);
```

Trả về

true hoặc false

Ví dụ

```
isnan(sqrt(-2)); //true
isnan(sqrt(2)); // false
```

4.2.3.4.9 constrain()

Bắt buộc giá trị nằm trong một khoảng cho trước.

Cú pháp

```
constrain(x, a, b)
```

Tham số

x: giá trị cần xét

a: chặn dưới (a là giá trị nhỏ nhất của khoảng)

b: chặn trên (b là giá trị lớn nhất của khoảng)

Trả về

x: nếu a <= x <= b

a: nếu x < a

b: nếu x > b

Ví dụ

```
int sensVal = analogRead(A2);
sensVal = constrain(sensVal, 10, 150);
//Giới hạn giá trị sensVal trong khoảng [10,150]
```

4.2.3.5 Các hàm lượng giác

4.2.3.5.1 cos()

Hàm này có nhiệm vụ tính cos một một góc (đơn vị radian). Giá trị chạy trong đoạn [-1,1].

Cú pháp

```
cos(rad)
```

Tham số

rad: góc ở đơn vị radian (kiểu float)

Trả về

cos của góc rad (kiểu double)

4.2.3.5.2 sin()

Hàm này có nhiệm vụ tính sin một một góc (đơn vị radian). Giá trị chạy trong đoạn [-1,1].

Cú pháp

```
sin(rad)
```

Tham số

rad: góc ở đơn vị radian (kiểu float)

Trả về

sin của góc rad (kiểu double)

4.2.3.5.3 tan()

Hàm này có nhiệm vụ tính tan một một góc (đơn vị radian). Giá trị chạy trong khoảng từ âm vô cùng đến dương vô cùng.

Cú pháp

```
tan(rad)
```

Tham số

rad: góc ở đơn vị radian (kiểu float)

Trả về

tan của góc rad (kiểu double)

4.2.3.6 Sinh số ngẫu nhiên

4.2.3.6.1 randomSeed()

Hàm ***random()*** luôn trả về một số ngẫu nhiên trong phạm vi cho trước. Giả sử gọi hàm này 10 lần, nó sẽ trả về 10 giá trị số nguyên ngẫu nhiên. Nếu gọi nó **n** lần, ***random()*** sẽ trả về **n** số. Tuy nhiên những giá trị mà nó trả về luôn được biết trước (cố định).

Xét ví dụ sau:

```
void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.println(random(100));
  delay(200);
}
```

Chương trình cho 10 giá trị "ngẫu nhiên" đầu tiên nhận được là: 7, 49, 73, 58, 30, 72, 44, 78, 23, 9,.... Điều này có vẻ không được "ngẫu nhiên".

Nhưng hãy xét ví dụ sau:

```
void setup(){
  Serial.begin(9600);
  randomSeed(10);
}

void loop(){
  Serial.println(random(100));
  delay(200);
}
```

Nhận thấy rằng: chuỗi giá trị mà hàm ***random()*** trả về đã có sự thay đổi. Tuy nhiên chuỗi này vẫn là chuỗi cố định. Thử thay đổi tham số của lệnh ***randomSeed()*** từ ***10*** sang một số khác, sẽ thấy chuỗi số trả về cũng thay đổi theo nhưng giá trị xuất ra thì vẫn cố định, cho dù có bấm nút reset trên Arduino thì chuỗi số được in ra những lẫn sau đều y hệt như lần đầu tiên chúng được in ra. Để ý rằng tham số của hàm random() vẫn cố định, dĩ nhiên nếu thay đổi tham số này thì chuỗi ngẫu nhiên trả về sẽ thay đổi theo, nhưng chúng cũng vẫn là một chuỗi số cố định.

*Với cùng khoảng giá trị truyền vào hàm **random()**, hàm **randomSeed()** quyết định trật tự các giá trị mà **random()** trả về. Trật tự này phụ thuộc vào tham số mà ta truyền vào **randomSeed().***

Cú pháp

```
randomSeed(number);
```

Với ***number*** là một số nguyên bất kì.

Lưu ý: *nếu gọi hàm* ***random()*** *mà không chạy lệnh* ***randomSeed()*** *trước đó, chương trình sẽ mặc định chạy sẵn lệnh* ***randomSeed(0)*** *(tham số là 0).*

Ví dụ

Nếu chạy **randomSeed(0)**, hàm **random(100)** sẽ trả về 10 giá trị đầu tiên là: 7, 49, 73, 58, 30, 72, 44, 78, 23, 9, ...

Nếu chạy **randomSeed(10)**, hàm **random(100)** sẽ trả về 10 giá trị đầu tiên là: 70 , 43, 1, 92, 65, 26, 40, 98, 48, 67, ...

Nếu chạy **randomSeed(-46)**, hàm **random(100)** sẽ trả về 10 giá trị đầu tiên là: 15, 50, 82, 36, 36, 37, 25, 59, 93, 74, ...

Nếu chạy **randomSeed(159)**, hàm **random(100)** sẽ trả về 10 giá trị đầu tiên là: 13, 51, 67, 38, 22, 50, 67, 73, 81, 75, ...

Nếu chạy **randomSeed(159)**, hàm **random(99)** sẽ trả về 10 giá trị đầu tiên là: 67, 42, 70, 34, 53, 6, 42, 38, 29, 64, ...

Mẹo nhỏ

Nếu cần một chuỗi số ngẫu nhiên một cách thực sự, tức là giá trị của chuỗi số trả về không thể xác định được, hãy chạy lệnh ***randomSeed()*** với một tham số ngẫu nhiên truyền vào.

Tham khảo chương trình sau

```
void setup() {
  Serial.begin(9600);
  randomSeed(analogRead(A0));
}
void loop() {
  Serial.println(random(100));
  delay(300);
}
```

Ở đây tham số ngẫu nhiên truyền vào ***randomSeed()*** là giá trị trả về của ***analogRead(A0)*** với chân A0 trên Arduino không được sử dụng. Ngoài ra, cũng có thể sử dụng tham số truyền vào ***randomSeed()*** là thời gian trên đồng hồ hệ thống.

4.2.3.6.2 random()

Trả về một giá trị nguyên ngẫu nhiên trong khoảng giá trị cho trước.

Cú pháp

```
random(max+1);
random(min,max+1);
```

Trong đó:

min và **max** là giá trị đầu và cuối của khoảng giá trị mà *random()* trả về.

Trả về

Giá trị nguyên ngẫu nhiên nằm trong khoảng từ **min** đến **max**. Nếu giá trị **min** không được đưa vào thì nó được hiểu ngầm là 0.

Ví dụ

```
int a = random(100);   //giá trị a nằm trong khoảng từ 0 đến 99
int b = random(0,11);  //giá trị b nằm trong khoảng từ 0 đến 10
```

4.2.3.7 Nhập xuất nâng cao (Advanced I/O)

4.2.3.7.1 tone()

Hàm này sẽ tạo ra một sóng vuông ở tần số được định trước (chỉ nửa chu kỳ) tại một pin digital bất kỳ (analog vẫn được). Thời hạn của quá trình tạo ra sóng âm có thể được định trước hoặc nó sẽ phát ra âm thanh liên tục cho đến khi Arduino IDE chạy hàm noTone(). Chân digital đó cần được kết nối tới một buzzer hoặc một loa để có thể phát được âm thanh.

Lưu ý rằng, chỉ có thể sử dụng duy nhất mộ hàm tone() trong cùng một thời điểm. Nếu hàm tone() đang chạy trên một pin nào đó, bây giờ lại tone() thêm một lần nữa thì hàm tone() sau sẽ không có hiệu lực. Nếu tone() lên pin đang được tone() thì hàm tone() sau sẽ thay đổi tần số sóng của pin đó.

Trên mạch Arduino Mega, sử dụng hàm tone() thì sẽ can thiệp đến đầu ra PWM tại các chân digital 3 và digital 11.

Hàm tone() sẽ không thể phát ra âm thanh có tần số < 31 Hz. Để biết têm về kĩ thuật này, hãy xem trang này.

Chú ý: Nếu muốn chơi nhiều cao độ khác nhau trên nhiều pin. Thì trước khi chơi trên một pin khác thì phải noTone() trên pin đang được sử dụng.

Cú pháp

```
tone(pin, frequency)
tone(pin, frequency, duration)
```

Tham số

pin: cổng digial / analog muốn chơi nhạc (nói cách khác là pin được kết nối tới loa)

frequency: tần số của sóng vuông (sóng âm) - *unsigned int*

duration: thời gian phát nhạc, đơn vị là mili giây (tùy chọn) - *unsigned long*

Trả về

không

Ví dụ

Phát nhạc bằng Arduino với một cái loa hoặc buzzer

4.2.3.7.2 noTone()

Hàm này có nhiệm vụ kết thúc một sự kiện tone() trên một pin nào đó (đang chạy lệnh tone()). Nếu không có bất kỳ hàm tone() nào đang hoạt động thì hàm này sẽ không gây bất kỳ ảnh hưởng gì đến chương trình.

Chú ý: Nếu muốn chơi nhiều cao độ khác nhau trên nhiều pin. Thì trước khi chơi trên một pin khác thì b phải noTone() trên pin đang được sử dụng.

Cú pháp

```
noTone(pin)
```

Tham số

pin: cổng digial / analog muốn chơi nhạc (nói cách khác là pin được kết nối tới loa)

Trả về

Không

4.2.3.7.3 shiftOut()

shiftOut() có nhiệm vụ chuyển 1 byte (gồm 8 bit) ra ngoài từng bit một. Bit được chuyển đi có thể được bắt đầu từ bit nằm bên trái nhất (leftmost) hoặc từ bit nằm bên phải nhất (rightmost). Các bit này được xuất ra tại chân dataPin sau khi chân clockPin được pulsed (có mức điện thế là HIGH, sau đó bị đẩy xuống LOW).

Lưu ý: Nếu đang giao tiếp với một thiết bị mà chân clock của nó có giá trị được thay đổi từ mức điện thế LOW lên HIGH (rising edge) khi shiftOut, thì cần chắc chắn rằng chân clockPin cần được chạy lệnh này: digitalWrite(clockPin,LOW);

Cú pháp

```
shiftOut(dataPin, clockPin, bitOrder, value)
```

Tham số

dataPin: pin sẽ được xuất ra tín hiệu (*int*)

clockPin: pin dùng để xác nhận việc gửi từng bit của **dataPin** (*int*)

bitOrder: một trong hai giá trị **MSBFIRST** hoặc **LSBFIRST**. (Bắt đầu từ bit bên phải nhất hoặc Bắt đầu từ bit bên trái nhất)

value: dữ liệu cần được shiftOut. (*byte*)

Chú ý

shiftOut() chỉ xuất được dữ liệu kiểu byte. Nếu muốn xuất một kiểu dữ liệu lớn hơn thì phải shiftOut 2 lần (hoặc nhiều hơn), mỗi lần là 8 bit.

Trả về

không

Ví dụ

Điều khiển 8 đèn LED sáng theo ý muốn.

4.2.3.7.4 pulseIn()

Đọc một xung tín hiệu digital (HIGH/LOW) và trả về chu kì của xung tín hiệu, tức là thời gian tín hiệu chuyển từ mức HIGH xuống LOW hoặc ngược lại (LOW -> HIGH). Một số cảm biến như cảm biến màu sắc như TCS3200D hay cảm biến siêu âm dòng HC-SRxx phải giao tiếp qua xung tín hiệu nên ta phải kết hợp giữa 2 hàm ***digitalWrite()*** để xuất tín hiệu và ***pulseIn()*** để đọc tín hiệu.

Cú pháp

```
pulseIn(pin, value);
pulseIn(pin, value, timeout);
```

Trong đó:

pin là chân được chọn để đọc xung. ***pin*** có kiểu dữ liệu là *int*.

Nếu đặt ***value*** là HIGH, hàm ***pulseIn()*** sẽ đợi đến khi tín hiệu đạt mức HIGH, khởi động bộ đếm thời gian. Khi tín hiệu nhảy xuống LOW, bộ đếm thời gian dừng lại. ***pulseIn()*** sẽ trả về thời gian tín hiệu nhảy từ mức *HIGH* xuống LOW này. Nếu đặt *value* là LOW, hàm ***pulseIn()*** sẽ làm ngược lại, đó là đo thời gian tín hiệu nhảy từ mức LOW lên HIGH. *value* có kiểu dữ liệu là *int*.

Nếu tín hiệu luôn ở một mức HIGH/LOW cố định thì sau khoảng thời gian ***timeout***, hàm ***pulseIn()*** sẽ dừng bộ đếm thời gian và trả về giá trị 0. ***timeout*** được tính bằng đơn vị micro giây. Giá trị mặc định của ***timeout*** là 60.10^6 tương ứng với 1 phút. Giá trị tối đa là 180.10^6 tương ứng với 3 phút. ***timeout*** có kiểu dữ liệu là *unsigned long*.

Trả về

Một số nguyên kiểu *unsigned long*, đơn vị là micro giây. ***pulseIn()*** trả về 0 nếu thời gian nhảy trạng thái HIGH/LOW vượt quá ***timeout***

Ví dụ

```
int pin = 7;
unsigned long duration;
void setup() {
  Serial.begin(9600);
  pinMode(pin, INPUT);
}
```

```
void loop() {
  duration = pulseIn(pin, HIGH);
  //Hãy nối chân 7 của Arduino vào đường tín hiệu
  //muốn đọc xung
  Serial.println(duration);
}
```

4.2.3.8 Bits và Bytes

4.2.3.8.1 *lowByte()*

lowByte() là hàm trả về byte cuối cùng (8 bit cuối cùng) của một chuỗi các bit. Một số nguyên bất kì cũng được xem như là một chuỗi các bit, vì bất kì số nguyên nào cũng có thể biểu diễn ở hệ nhị phân dưới dạng các bit "0" và "1".

Lưu ý:

lowByte() không nhận giá trị thuộc kiểu dữ liệu số thực.

Cú pháp

```
lowByte([giá trị cần lấy ra 8 bit cuối]);
```

Trả về

byte

Ví dụ

```
int A = lowByte(0B11110011001100);  //A = 0B11001100 = 204;
int B = lowByte(511);               //B = lowByte(0B111111111) = 255;
int C = lowByte(5);                 //C = lowByte(0B00000101) = 0B101 = 5;
```

4.2.3.8.2 *highByte()*

highByte() là hàm trả về một chuỗi 8 bit kề với 8 bit cuối cùng của một chuỗi các bit. Như vậy, nếu dữ liệu đưa vào một chuỗi 16bit thì*highByte()* sẽ trả về 8 bit đầu tiên, nếu dữ liệu đưa vào là một chuỗi 8bit hoặc nhỏ hơn, *highByte()* sẽ trả về giá trị 0. Một số nguyên bất kì cũng được xem như là một chuỗi các bit, vì bất kì số nguyên nào cũng có thể biểu diễn ở hệ nhị phân dưới dạng các bit "0" và "1".

Lưu ý:

highByte() không nhận giá trị thuộc kiểu dữ liệu số thực.

Cú pháp

```
highByte([giá trị đưa vào]);
```

Trả về

byte

Ví dụ

```
int A = highByte(0B1111111100000000);   //A = 0B11111111 = 255;
int B = highByte(0B10101010);           //B = 0
int C = highByte(0B1100000000011111111)  //C = 0B00000000 = 0
int D = highByte(1023);                 //D = highByte(0B111111111) = 0B11 = 3
```

4.2.3.8.3 bitRead()

bitRead() sẽ trả về giá trị tại một bit nào đó được xác định bởi người lập trình của một số nguyên.

Cú pháp

```
bitRead(x, n)
```

Tham số

x: một số nguyên thuộc bất cứ kiểu số nguyên nào

n: bit cần đọc. Các bit sẽ được tính từ phải qua trái, và số thứ tự đầu tiên là số 0

Trả về

Giá trị của 1 bit (1 hoặc là 0)

Ví dụ

```
bitRead(B11110010,0); // trả về 0
bitRead(B11110010,1); // trả về 1
bitRead(B11110010,2); // trả về 0
//Hàm bitRead có thể viết như sau
B11110010 >> 0 & 1 // = 0
B11110010 >> 1 & 1 // = 1
B11110010 >> 2 & 1 // = 0
```

4.2.3.8.4 bitWrite()

bitWrite() sẽ ghi đè bit tại một vị trí xác định của số nguyên.

Cú pháp

```
bitWrite(x, n, b)
```

Tham số

x: một số nguyên thuộc bất cứ kiểu số nguyên nào

n: vị trí bit cần ghi. Các bit sẽ được tính từ phải qua trái, và số thứ tự đầu tiên là số 0.

b: 1 hoặc 0

Trả về

không

Ví dụ

```
bitWrite(B11110010,0,1); // B11110011
bitWrite(B11110010,1,0); // B11110000
bitWrite(B11110010,2,1); // B11110110
//Hàm bitWrite có thể viết như sau
B11110010 | (1 << 0)  // = B11110011
B11110010 & ~(1 << 1) // = B11110000
B11110010 | (1 << 2)  // = B11110110
```

4.2.3.8.5 bitSet()

bitSet() sẽ thay giá trị tại một bit xác định của một số nguyên thành 1.

Cú pháp

```
bitSet(x, n)
```

Tham số

x: một số nguyên thuộc bất cứ kiểu số nguyên nào

n: vị trí bit cần ghi. Các bit sẽ được tính từ phải qua trái, và số thứ tự đầu tiên là số 0.

Trả về

không

Ví dụ

```
bitSet(B11110010,0); // B11110011
bitSet(B11110010,2); // B11110110
//Hàm bitSet có thể viết như sau
B11110010 | (1 << 0)  // = B11110011
B11110010 | (1 << 2)  // = B11110110
```

4.2.3.8.6 bitClear()

bitClear() sẽ thay giá trị tại một bit xác định của một số nguyên thành 0.

Cú pháp

```
bitClear(x, n)
```

Tham số

x: một số nguyên thuộc bất cứ kiểu số nguyên nào

n: vị trí bit cần ghi. Các bit sẽ được tính từ phải qua trái, và số thứ tự đầu tiên là số 0.

Trả về

không

Ví dụ

```
bitClear(B11110010,1); // B11110000


//Hàm bitClear có thể viết như sau
B11110010 & ~(1 << 1)  // = B11110000
```

4.2.3.8.7 bit()

Trả về một số nguyên dạng 2^n (2 mũ n).

Cú pháp

```
bit(n)
```

Tham số

n: số nguyên

Trả về

một số nguyên

Ví dụ

```
bit(0); // 2^0 = 1
bit(1); // 2^1 = 2
bit(2); // 2^2 = 4
bit(8); // 2^8 = 256
// cũng có thể viết như sau
1 << 0 // = 1
1 << 1 // = 2
1 << 2 // = 4
1 << 8 // = 256
```

4.2.3.9 Ngắt (interrupt)

4.2.3.9.1 attachInterrupt()

Ngắt (interrupt) là những lời gọi hàm tự động khi hệ thống sinh ra một sự kiện. Những sự kiện này được nhà sản xuất vi điều khiển thiết lập bằng phần cứng và được cấu hình trong phần mềm bằng những tên gọi cố định.

Vì ngắt hoạt động độc lập và tự sinh ra khi được cấu hình nên chương trình chính sẽ đơn giản hơn. Một ví dụ điển hình về ngắt là hàm millis(). Hàm này tự động chạy cùng với chương trình và trả về 1 con số tăng dần theo thời gian mặc dù chúng ta không cài đặt nó. Việc cài đặt hàmmillis() sử dụng đến ngắt và được cấu hình tự động bên trong mã chương trình Arduino.

Vì sao cần phải dùng đến ngắt?

Ngắt giúp chương trình gọn nhẹ và xử lý nhanh hơn. Chẳng hạn, khi kiểm tra 1 nút nhấn có được nhấn hay không, thông thường cần kiểm tra trạng thái nút nhấn bằng hàm digitalRead() trong đoạn chương trình loop(). Với việc sử dụng ngắt, chỉ cần nối nút nhấn đến đúng chân có hỗ trợ ngắt, sau đó cài đặt ngắt sẽ sinh ra khi trạng thái nút chuyển từ HIGH->LOW. Thêm 1 tên hàm sẽ gọi khi ngắt sinh ra. Vậy là xong, biến trong đoạn chương trình ngắt sẽ cho ta biết trạng thái nút nhấn.

Số lượng các ngắt phụ thuộc vào từng dòng vi điều khiển. Với Arduino Uno chỉ có 2 ngắt, Mega 2560 có 6 ngắt và Leonardo có 5 ngắt.

Board	**int.0**	**int.1**	**int.2**	**int.3**	**int.4**	**int.5**
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	

Cú pháp

```
attachInterrupt(interrupt, ISR, mode);
```

Thông số

interrupt: Số thứ tự của ngắt. Trên Arduino Uno, có 2 ngắt với số thứ tự là 0 và 1. Ngắt số 0 nối với chân digital số 2 và ngắt số 1 nối với chân digital số 3. Muốn dùng ngắt phải gắn nút nhấn hoặc cảm biến vào đúng các chân này thì mới sinh ra sự kiện ngắt. Nếu dùng ngắt số 0 mà gắn nút nhấn ở chân digital 4 thì không chạy được rồi.

ISR: tên hàm sẽ gọi khi có sự kiện ngắt được sinh ra.

mode: kiểu kích hoạt ngắt, bao gồm

- **LOW**: kích hoạt liên tục khi trạng thái chân digital có mức thấp
- **HIGH**: kích hoạt liên tục khi trạng thái chân digital có mức cao.
- **RISING**: kích hoạt khi trạng thái của chân digital chuyển từ mức điện áp thấp sang mức điện áp cao.
- **FALLING**: kích hoạt khi trạng thái của chân digital chuyển từ mức điện áp cao sang mức điện áp thấp.

Lưu ý: với mode LOW và HIGH, chương trình ngắt sẽ được gọi liên tục khi chân digital còn giữ mức điện áp tương ứng.

Trả về

không

Ví dụ

Đoạn chương trình dưới đây sẽ làm sáng đèn led khi không nhấn nút và làm đèn led tắt đi khi người dùng nhấn nút, nếu vẫn giữ nút nhấn thì đèn led vẫn còn tắt. Sau khi thả nút nhấn, đèn led sẽ sáng trở lại.

```
int ledPin = 13;
void tatled()
{
   digitalWrite(ledPin, LOW); // tắt đèn led
}
void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(2, INPUT_PULLUP); // sử dụng điện trở kéo lên cho chân số 2, ngắt 0
  attachInterrupt(0, tatled, LOW); // gọi hàm tatled liên tục khi còn nhấn nút
}
void loop()
{
 digitalWrite(ledPin, HIGH);   // bật đèn led
}
```

Một ví dụ khác khi sử dụng ngắt, có thể thoát khỏi các hàm delay để xử lý 1 đoạn chương trình khác

```
int ledPin = 13;
void tatled()
{
   // tắt đèn led khi nhấn nút, nhả ra led nhấp nháy trở lại
   digitalWrite(ledPin, LOW);
}
void setup()
```

```
{
  pinMode(ledPin, OUTPUT);
  pinMode(2, INPUT_PULLUP); // sử dụng điện trở kéo lên cho chân số 2, ngắt 0
  attachInterrupt(0, tatled, LOW);
}
void loop()
{
  // đoạn chương trình này nhấp nháy led sau 500ms
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
}
```

4.2.3.9.2 detachInterrupt()

Hàm detachInterrupt() sẽ tắt các ngắt đã được kích hoạt tương ứng với thông số truyền vào. Giả sử sau khi nhấn nút bấm lần đầu tiên đèn led sẽ tắt nhưng nhấn lần thứ 2 đèn sẽ không tắt nữa. Lúc này cần dùng đến detachInterrupt() để tắt ngắt chúng ta đã tạo ra.

Cú pháp

```
detachInterrupt(interrupt);
```

Thông số

interrupt: số thứ tự ngắt (xem thêm ở bài attachInterrupt())

Trả về

không

Ví dụ

Đoạn chương trình dưới đây sẽ bật sáng đèn led và chỉ tắt nó khi nhấn lần đầu tiên, thả ra đèn sẽ sáng lại. Nếu tiếp tục nhấn nữa thì đèn vẫn sáng mà không bị tắt đi.

```
int ledPin = 13;        // đèn LED được kết nối với chân digital 13
boolean daNhan = false;  // lưu giữ giá trị cho biết đã nhấn nút hay chưa
void tatled()
{
  digitalWrite(ledPin, LOW); // tắt đèn led khi còn nhấn nút
  daNhan = true; // lúc này đã nhấn nút
```

```
}
void setup()
{
 pinMode(ledPin, OUTPUT);      // thiết đặt chân ledPin là OUTPUT
 pinMode(2, INPUT_PULLUP); // sử dụng điện trở kéo lên cho chân số 2, ngắt 0
 attachInterrupt(0, tatled, LOW); // cài đặt ngắt gọi hàm tatled
}
void loop()
{
  digitalWrite(ledPin, HIGH);   // bật đèn led
  if (daNhan == true)
  {
      // Nếu đã nhấn nút thì tắt ngắt đi
      detachInterrupt(0);
  }
}
```

4.2.3.9.3 interrupts()

Mặc định, Arduino luôn bật các ngắt nên trong phần setup(), không cần gọi hàm này để bật các ngắt. Hàm interrupts() sẽ bật toàn bộ các ngắt đã được cài đặt. Nếu vì lý do nào đó ta tắt các ngắt bằng hàm noInterrupts(), sử dụng hàm này để bật lại các ngắt.

Cú pháp

```
interrupts();
```

Thông số

không

Trả về

không

Ví dụ

```
void setup() {}
void loop()
{
 noInterrupts();
 // tắt các ngắt để chạy
```

```
 // đoạn chương trình yêu cầu cao về thời gian
 interrupts();
 // bật lại các ngắt, các ngắt hoạt động
 // bình thường trở lại
}
```

4.2.3.9.4 noInterrupts()

Khi cần chạy các đoạn chương trình yêu cầu chính xác về thời gian, cần tắt các ngắt để Arduino chỉ tập trung vào xử lý các tác vụ cần thiết và chỉ duy nhất các tác vụ này. Các ngắt chạy nền sẽ không được thực thi sau khi gọi hàm noInterrupts().

Cú pháp

```
noInterrupts();
```

Thông số

không

Trả về

không

Ví dụ

```
void setup() {}
void loop()
{
 noInterrupts();
 // tắt các ngắt để chạy
 // đoạn chương trình yêu cầu cao về thời gian
 interrupts();
 // bật lại các ngắt, các ngắt hoạt động
 // bình thường trở lại
}
```