HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

NGUYỄN NGỌC MINH
NGUYỄN TRUNG HIẾU

BÀI GIẢNG
HỆ THỐNG NHÚNG

HÀ NỘI – 12.2014

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

4.1 Thiết kế hệ thống

4.1.1 Xác định yêu cầu

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

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

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

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

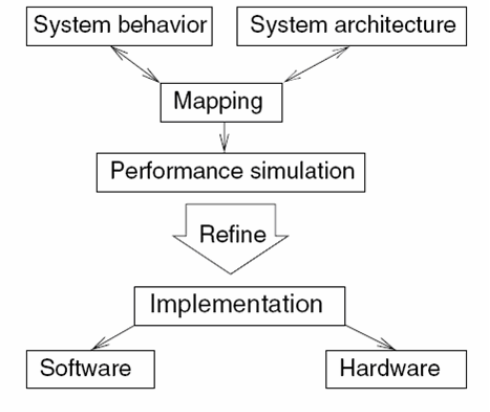
Quản lý các nhiệm vụ cùng mức: Hoạt động này có liên quan với việc xác định các nhiệm vụ phải được hiện diện trong hệ thống nhúng cuối cùng. Những nhiệm vụ này có thể khác với những nhiệm vụ đã bao gồm trong các đặc điểm kỹ thuật, từ đó có những lý do tốt để cho việc sát nhập và chia tách các nhiệm vụ.

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

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

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

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



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

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

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

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

4.1.2 Đặc tả

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

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

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

Có hai loại phân cấp:

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

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

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

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

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

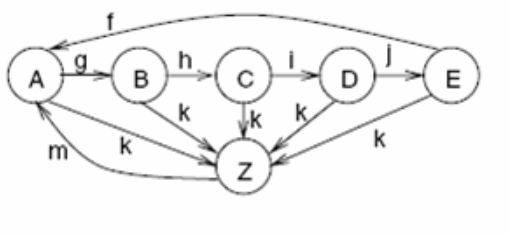
Xử lý-sự kiện : Do tính chất phản ứng tự nhiên của các hệ thống nhúng, các cơ chế cho việc mô tả các sự kiện phải tồn tại. Các sự kiện như vậy có thể là các sự kiện bên ngoài (gây ra bởi môi trường) hoặc các sự kiện nội bộ (gây ra bởi các thành phần của hệ thống).

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

Hỗ trợ các thiết bị I/0 không tiêu chuẩn: Nhiều hệ thống nhúng sử dụng các thiết bị I/O khác với các thiết bị thông thường được tìm thấy trên máy PC. Cần phải có khả năng mô tả những đầu vào và những đầu ra cho những thiết bị đó thuận tiện.

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

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

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

a. Mô hình tính toán

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

b. Biểu đồ trạng thái (StateCharts)

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

c. Những đặc trưng ngôn ngữ khái quát

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

4.1.3 Phân hoạch phần cứng - phần mềm

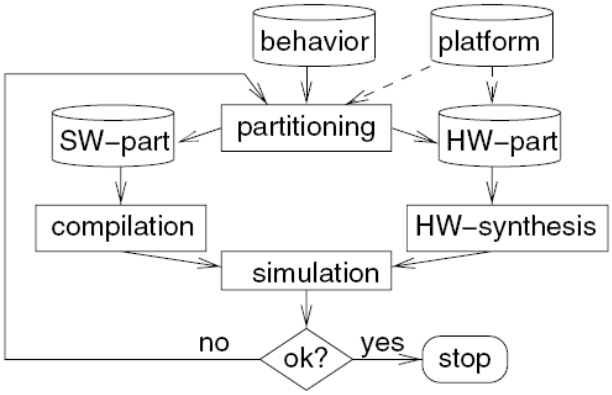
Trong quá trình thiết kế, chúng ta phải giải quyết vấn đề thực hiện các đặc tả kỹ thuật hoặc trong phần cứng hoặc ở dạng của các chương trình chạy trên bộ vi xử lý. Phần này mô tả một số kỹ thuật để lập bản đồ này. Áp dụng các kỹ thuật này, chúng ta sẽ có thể quyết định những phần phải được thực hiện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

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

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

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

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



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

a. COOL

Đối với COOL, đầu vào bao gồm ba phần:

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

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

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

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

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

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

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

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

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

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

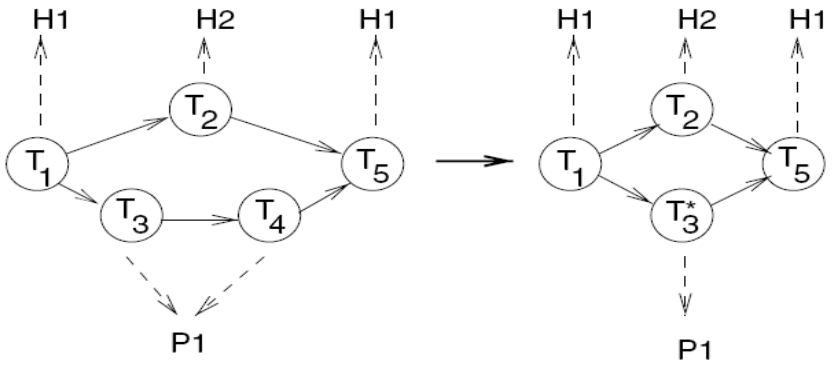
5 Trải phẳng sự phân cấp: Bước tiếp theo sẽ khai triển một đồ thị nhiệm vụ phẳng từ lưu đồ có phân cấp. Khi không có việc sát nhập hoặc chia tách các nút được thực hiện, độ chi tiết được sử dụng bởi nhà thiết kế được duy trì. Chi phí và thông tin thực hiện thu được từ quá trình biên tập và tổng hợp phần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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Chúng ta rõ ràng sẽ giảm thiểu tổng chi phí nếu không có các bộ xử lý, bộ nhớ và phần cứng ứng dụng chuyên biệt đã được bao gồm trong "thiết kế". Do những ràng buộc, đây không phải là một giải pháp pháp lý. Bây giờ chúng ta có thể trình bày một mô tả ngắn gọn của một số các ràng buộc của mô hình IP:
* **Những ràng buộc ấn định hoạt động:** Những ràng buộc này bảo đảm rằng mỗi hoạt động được thực hiện hoặc trong phần cứng hoặc trong phần mềm. Những ràng buộc tương ứng có thể được công thức hóa như sau:

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

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

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

$$X_{i,k} \in IN_0, \quad (4.8)$$

$$Y_{i,k} \in IN_0 \quad (4.9)$$

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

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

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

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

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

Trong văn bản gốc, điều này có nghĩa là: với mọi kiểu *l* thuộc các nút đồ thị nhiệm vụ và với mọi nút i thuộc kiểu này, ràng buộc sau đây phải được duy trì: nếu i được ánh xạ tới bộ xử lý k nào đó (biểu thị bởi $Y_{i,k}$ là 1), thì phần mềm tương ứng với chức năng *l* phải được cung cấp bởi bộ xử lý k, và phần mềm tương ứng phải tồn tại trên bộ xử lý đó (biểu thị bởi $NY_{l,k}$ là 1).
Những sự ràng buộc bổ sung bảo đảm rằng những biến quyết định $NY_{l,k}$ cũng là những biến giá trị 0/1:

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

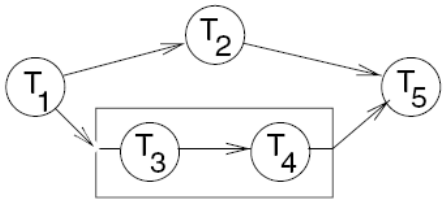
Những ràng buộc tài nguyên: Tập tiếp theo của các ràng buộc đảm bảo rằng "không quá nhiều" các nút được ánh xạ tới cùng một thành phần phần cứng tại cùng một thời điểm. Chúng ta giả định rằng, cứ mỗi chu kỳ đồng hồ, nhiều nhất là một hoạt động có thể được thực hiện trên mỗi thành phần phần cứng. Thật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

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

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

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

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



Hình 4-5. Đồ thị nhiệm vụ

Ví dụ: Trong phần sau, chúng ta sẽ trình bày về việc các ràng buộc này có thể được tạo ra như thế nào cho đồ thị nhiệm vụ trong Hình 4-5.
Giả sử rằng chúng ta có một thư viện thành phần phần cứng có chứa ba kiểu thành phần là H1, H2 và H3 với chi phí tương ứng là 20, 25 và 30 đơn vị chi phí. Hơn nữa, giả sử rằng chúng ta cũng có thể sử dụng một bộ xử lý P với phí là 5. Ngoài ra, chúng ta giả định rằng Bảng 4-2 mô tả thời gian thực hiện của các nhiệm vụ của chúng ta trên các thành phần này.

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

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

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

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

$\boldsymbol{X_{1,1} + Y_{1,1} = 1}$ (Nhiệm vụ 1 được ánh xạ hoặc tới H1 hoặc tới P1)
$\boldsymbol{X_{2,2} + Y_{2,1} = 1}$ (Nhiệm vụ 2 được ánh xạ hoặc tới H2 hoặc tới P1)
$\boldsymbol{X_{3,3} + Y_{3,1} = 1}$ (Nhiệm vụ 3 được ánh xạ hoặc tới H3 hoặc tới P1)
$\boldsymbol{X_{4,3} + Y_{4,1} = 1}$ (Nhiệm vụ 4 được ánh xạ hoặc tới H3 hoặc tới P1)
$\boldsymbol{X_{5,1} + Y_{5,1} = 1}$ (Nhiệm vụ 1 được ánh xạ hoặc tới H1 hoặc tới P1)

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

$$\boldsymbol{NY_{1,1} \geq Y_{1,1}} \quad (4.10)$$
$$\boldsymbol{NY_{2,1} \geq Y_{2,1}}$$
$$\boldsymbol{NY_{3,1} \geq Y_{3,1}}$$
$$\boldsymbol{NY_{3,1} \geq Y_{4,1}}$$
$$\boldsymbol{NY_{1,1} \geq Y_{5,1}} \quad (4.11)$$

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

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

Hàm chi phí là:

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

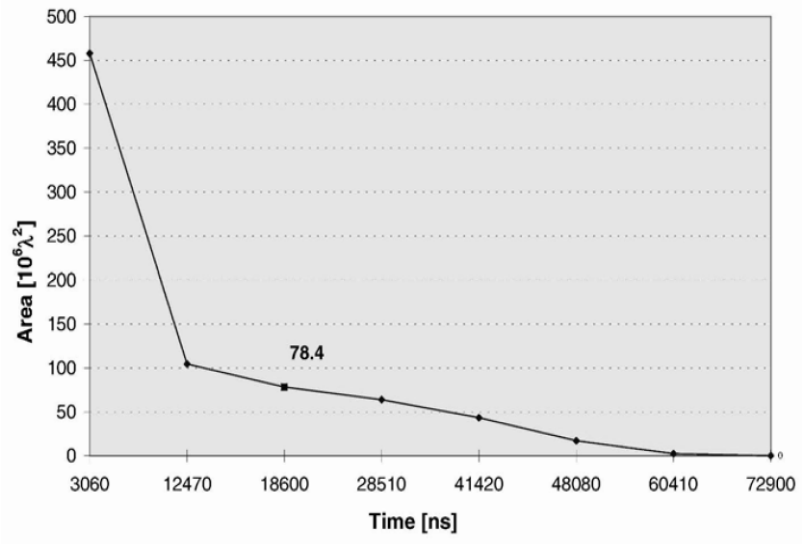
Trong đó #() biểu thị số lượng các trường hợp sử dụng của các thành phần phần cứng. Số này có thể được tính toán từ những biến được giới thiệu cho đến lúc này nếu lịch trình cũng được đưa vào bản kê khai. **V**ới một ràng buộc thời gian của 100 đơn vị thời gian, thiết kế chi phí tối thiểu bao gồm các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

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

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

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

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



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

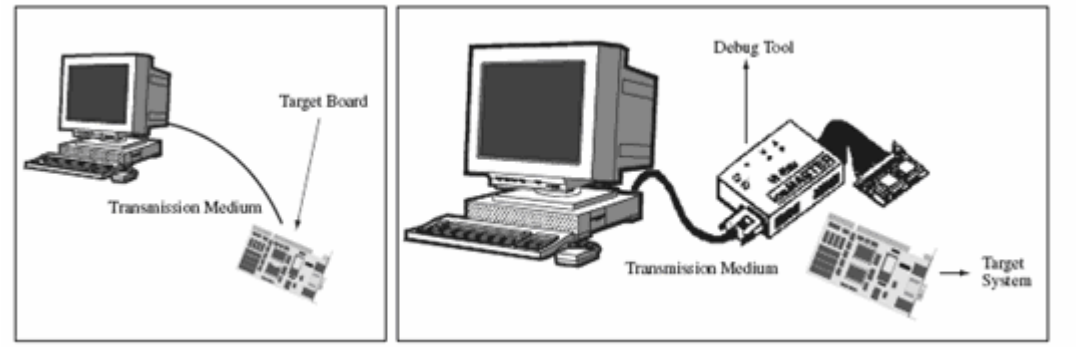
4.1.4 Thiết kế hệ thống

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

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

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

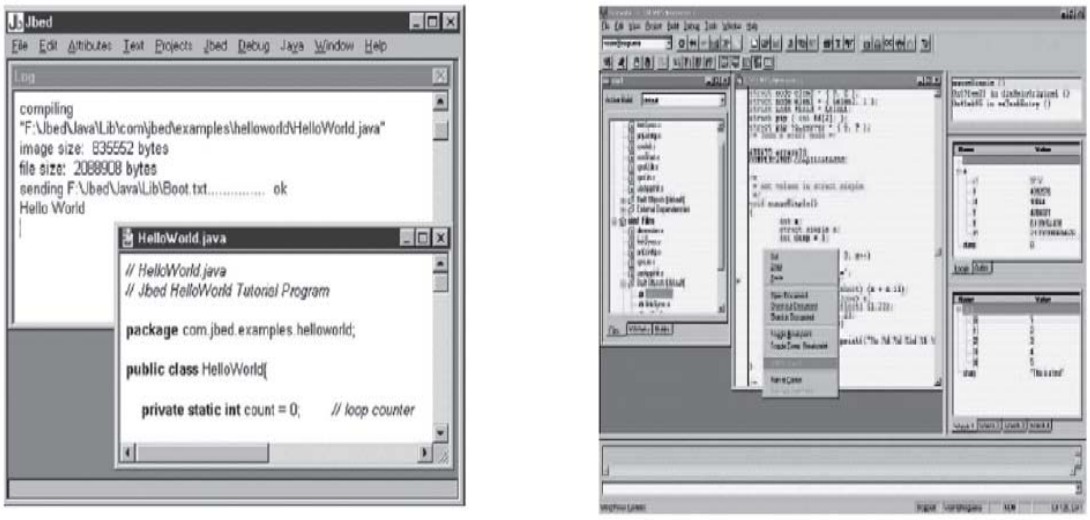
không có các công cụ đúng, việc thực hiện và gỡ lỗi hệ thống sẽ rất khó khăn, nếu không phải là không thể.



Hình 4-7. Môi trường phát triển

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

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

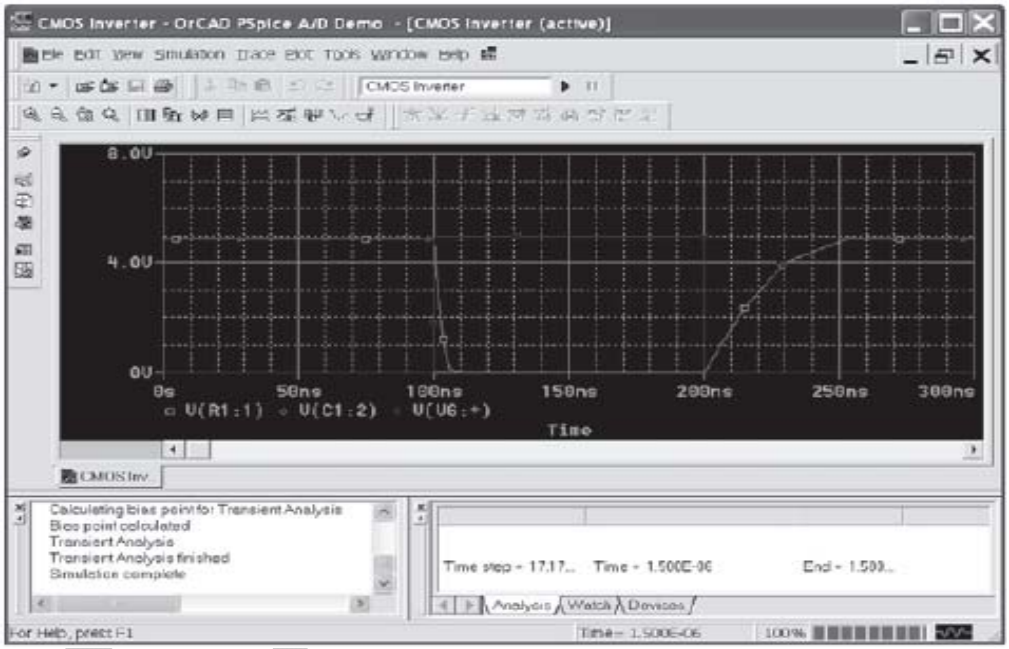


Hình 4-8. IDE

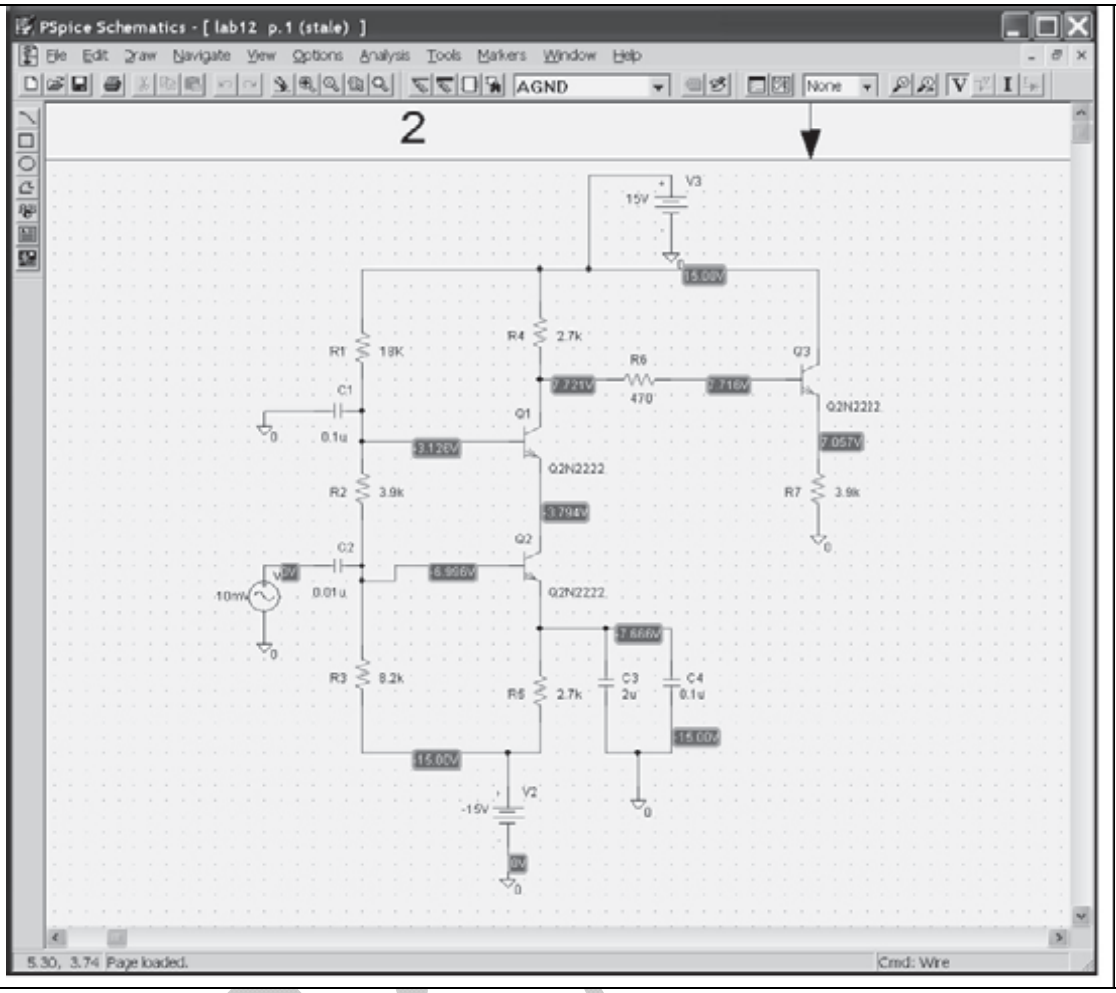
Thiết kế trợ giúp máy tính (CAD) và phần cứng

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

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



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



Hình 4-9b. Ví dụ mạch PSpice CAD

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

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

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

- phân chia các mạch phức tạp hơn thành các mạch nhỏ hơn, và sau đó kết hợp các kết quả.

- sử dụng các đặc tính đặc biệt của một số loại mạch nhất định.

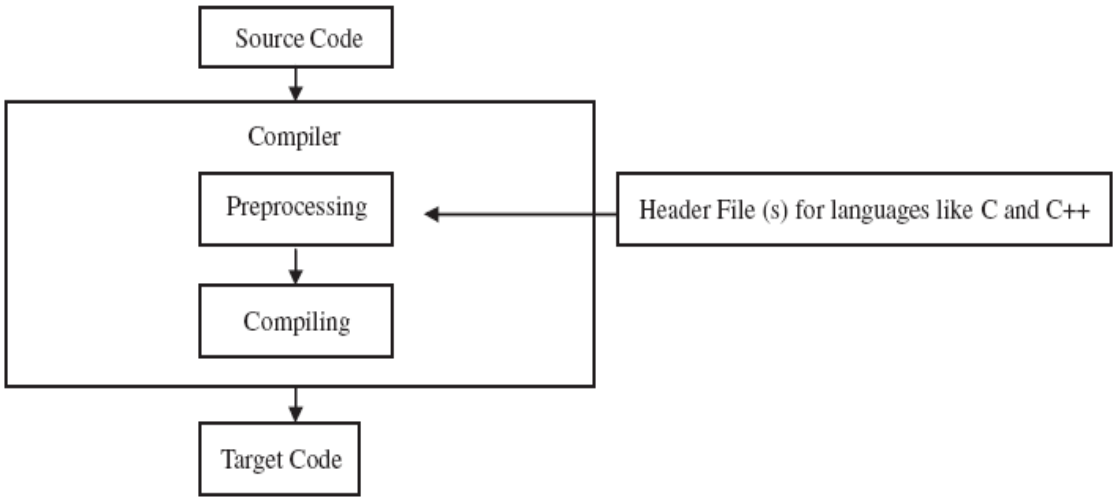
- sử dụng các máy tính vector và/hoặc máy tính song song tốc độ cao.

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

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

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

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

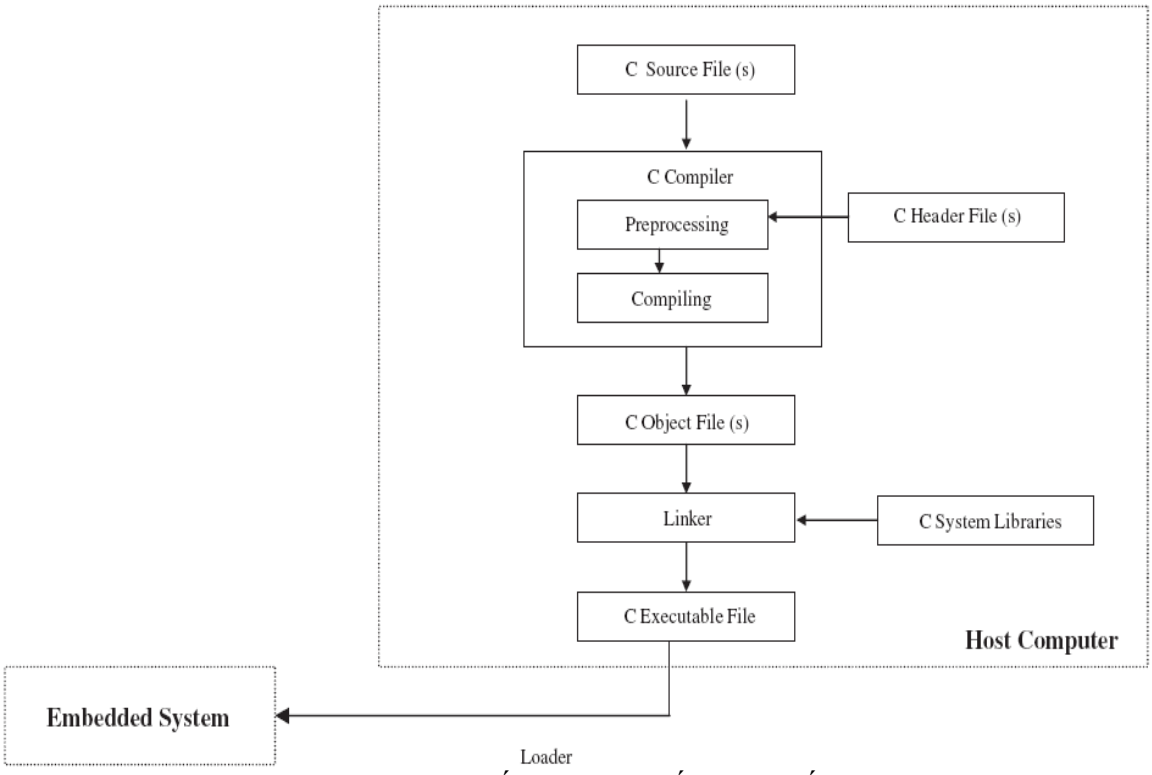


Hình 4-10. Sơ đồ biên dịch

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

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

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



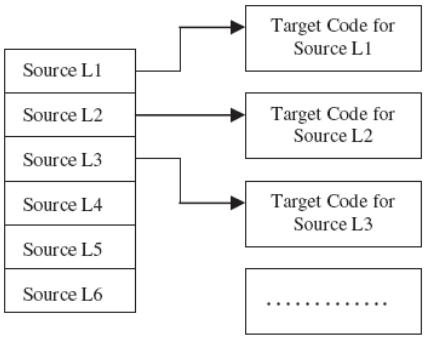
Hình 4-11: các bước biên dịch/liên kết và tập tin đối tượng kết quả, thực hiên trong C

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

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

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

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



Hình 4-12: Sơ đồ sự phiên dịch

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

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

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

Trình biên dịch nhúng lý tưởng

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

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

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

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

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

Công cụ gỡ lỗi

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

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

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

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

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

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

Loại công cụ	Công cụ gỡ lỗi	Mô tả	Ví dụ về sử dụng và những hạn chế
Phần cứng	Bộ mô phỏng trên mạch(In-Circuit Emulator: ICE)	Thiết bị hoạt động thay thế bộ vi xử lý trong hệ thống	* giải pháp gỡ lỗi đắt nhất điển hình, nhưng có nhiều khả năng gỡ lỗi * có thể hoạt động ở tốc độ đầy đủ của bộ xử lý (phụ thuộc vào ICE) và tới phần còn lại của hệ thống đó là bộ vi xử lý * cho phép có thể xem và có thể thay đổi được nội dung bộ nhớ trong , các thanh ghi, các biến, v.v..trong thời gian thực * tương tự như những trình gỡ rối, cho phép đặt những điểm dừng, thực hiện từng bước,... * thường có lớp che bộ nhớ để mô phỏng bộ nhớ ROM * bộ xử lý phụ thuộc .....
	Bộ mô phỏng ROM	Công cụ hoạt động thay thế ROM với các cáp kết nối với RAM cổng kép bên trong bộ mô	* cho phép sửa đổi nội dung trong ROM (không giống như một trình gỡ lỗi) * có thể đặt breakpoint trong mã ROM, và xem mã ROM thời gian thực * thường không hỗ trợ ROM on-chip,

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

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

			* bộ xử lý độc lập và cho phép xem hệ thống thực hiện trong thời gian thực với rất ít sự xâm nhập ....
	Voltmeter	Đo điện áp chênh lệch giữa 2 điểm trên mạch	* để đo các giá trị điện áp đặc biệt * để xác định sự hiện diện của nguồn tại tất cả các điểm trên mạch * rẻ hơn những công cụ phần cứng khác .....
	Ohmmeter	Đo điện trở giữa 2 điểm trên mạch	* rẻ hơn so với các công cụ phần cứng khác * để đo những thay đổi về dòng điện/điện áp trong mối quan hệ với điện trở (dùng định luật Ôm V=IR) *....
	Multimeter	Đo cả điện áp lẫn điện trở	* giống như volt và ohm meter *.....
Phần mềm	Debugger (trình gỡ rối)	Công cụ gỡ lỗi chức năng	Phụ thuộc vào trình gỡ rối - nói chung: * nạp/chạy từng bước/giám sát mã trên đích * thực hiện những điểm breakpoint để dừng sự thực hiện phần mềm * thực hiện những điểm breakpoint có điều kiện để dừng nếu điều kiện đặc biệt xuất hiện trong thời gian thực hiện * có thể sửa đổi nội dung của RAM, thường không thể sửa đổi nội dung của ROM .....
	Profiler	Thu thập giá trị theo thời gian của các biến, các thanh ghi... được lựa chọn	* thời gian bắt giữ phụ thuộc (khi) hành vi của phần mềm đang thực hiện * để bắt mẫu thực hiện (ở đâu) của phần mềm đang thực hiện ....
	Monitor	Giao diện gỡ lỗi tương tự như ICE, với phần mềm gỡ lỗi chạy trên đích và máy chủ. Một phần của monitor nằm trong ROM của bảng mạch đích (thường được gọi là tác nhân gỡ lỗi hoặc tác nhân	* tương tự như in phát biểu nhưng nhanh hơn, ít xâm nhập, làm việc tốt hơn cho giới hạn thời gian thực mềm, nhưng không tốt cho thời gian thực cứng * chức năng tương tự tới trình gỡ rối * Hệ điều hành nhúng có thể bao gồm monitor cho những kiến trúc đặc biệt .....

		đích), và một hạt nhân gỡ lỗi trên máy chủ. Phần mềm trên máy chủ và đích thường giao tiếp nối tiếp hoặc thông qua Ethernet (phụ thuộc vào những gì có sẵn trên đích).	
	Trình mô phỏng tập lệnh	Chạy trên máy chủ và mô phỏng bộ xử lý chủ và bộ nhớ (chương trình nhị phân có thể thực thi được nạp vào trình mô phỏng như là nó sẽ được nạp vào đích) và bắt chước phần cứng	* thường không chạy ở cùng một tốc độ chính xác như đích thực tế, nhưng có thể đánh giá phản ứng và thông qua thời gian bằng cách xem xét sự khác biệt giữa tốc độ máy chủ và đích * thường không mô phỏng phần cứng khác mà có thể tồn tại trên đích, nhưng có thể cho phép thử nghiệm các thành phần được xây dựng trong bộ xử lý * có thể mô phỏng hành vi ngắt * bắt giữ giá trị của biến, bộ nhớ và các thanh ghi * sẽ không mô phỏng chính xác hành vi của các phần cứng thực tế trong thời gian thực * thường phù hợp hơn để thử nghiệm các thuật toán chứ không phải là phản ứng đối với các sự kiện bên ngoài của một kiến trúc hay bảng mạch (dạng sóng và như vậy cần được mô phỏng thông qua phần mềm) * thường rẻ hơn so với đầu tư vào phần cứng và các công cụ thực .....
Manual (thủ công)	Luôn sẵn có, miễn phí hoặc rẻ hơn so với các giải pháp khác, hiệu quả, đơn giản để sử dụng nhưng thường xâm nhập cao hơn các loại công cụ khác, không đủ kiểm soát đối với sự lựa chọn sự kiện, cách ly, hoặc lặp lại. Khó khăn để gỡ lỗi hệ thống thời gian thực nếu phương pháp thủ công phải mất quá lâu để thực thi.		
	In các báo cáo	Công cụ gỡ lỗi chức năng, việc in các báo cáo được đưa vào bên trong mã để in thông tin thay đổi, vị trí trong mã thông	* để xem đầu ra của các biến, giá trị các thanh ghi, v.v.. trong khi các mã đang chạy * để kiểm tra đoạn mã đang được thực hiện * có thể làm chậm đáng kể thời

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

Bảng 4-1: Những công cụ gỡ lỗi

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

Cách rẻ nhất để gỡ lỗi

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

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

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

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

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

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

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

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

---Dựa trên bài viết "Firmware Basics for the Boss" của Jack Ganssle,

Embedded Systems Programming, tháng 2 năm 2004

Khởi động (Boot-Up) hệ thống

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

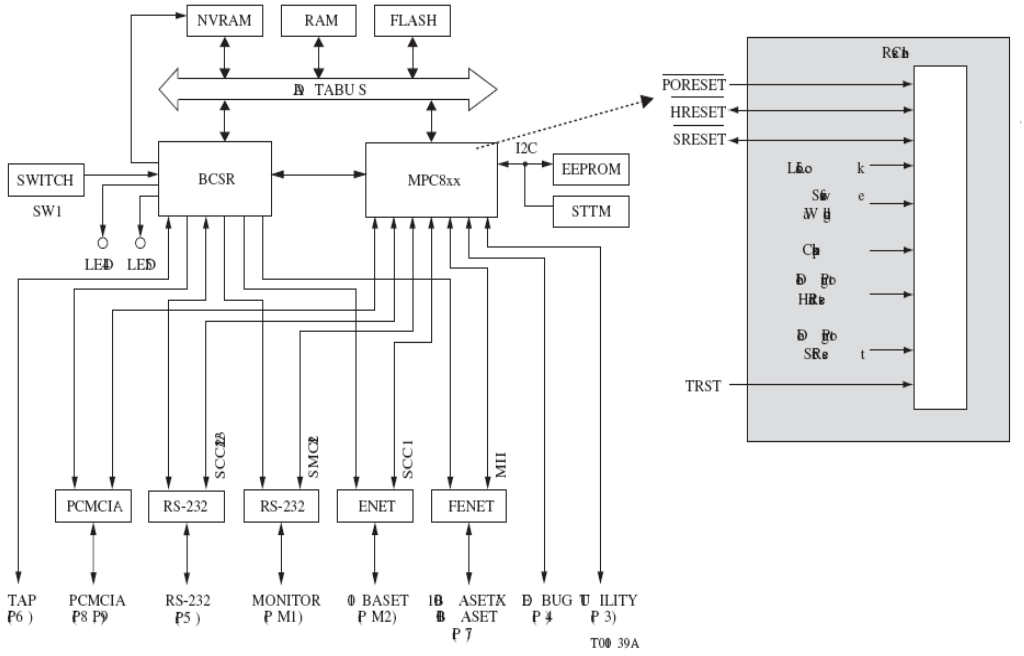
Mã Boot khác ở chiều dài và chức năng tùy thuộc vào thời điểm trong chu kỳ phát triển bảng mạch, cũng như các thành phần của nền tảng thực tế cần sự khởi tạo. Các chức năng chung (tối thiểu) được thực hiện bởi mã khởi động trên các nền tảng khác nhau, những cái khởi tạo chức năng cơ bản phần cứng, bao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

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

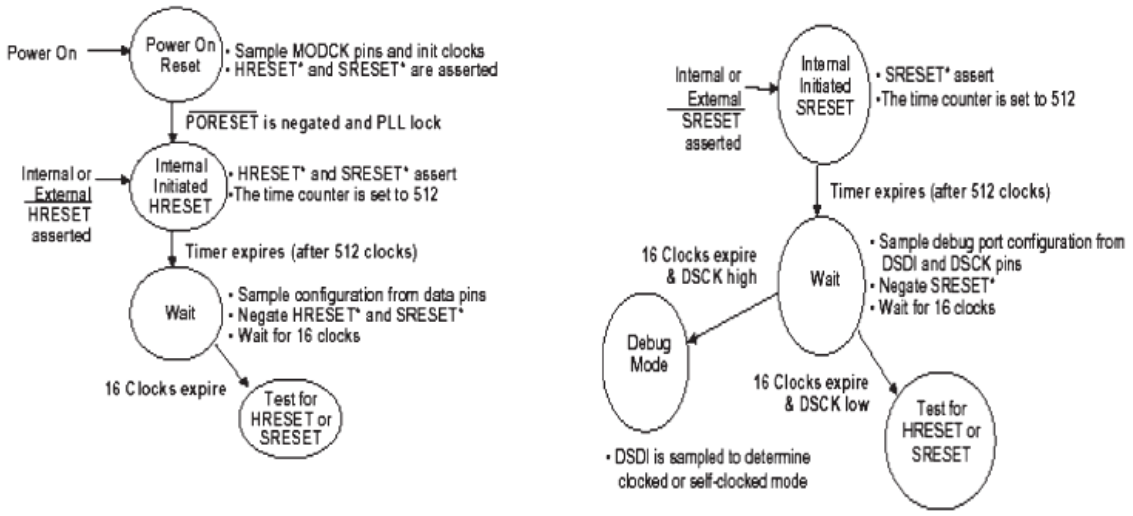
```
// through ethernet
// start executing rest of code(i.e. define memory map,
load OS, etc.)
…
}
```

Ví dụ khởi động bảng mạch dùng MPC823

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



Hình 4-13a: Sơ đồ giải thích



Hình 4-13b: Sơ đồ giải thích

4.2 Cài đặt và thử nghiệm hệ thống nhúng

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

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

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

	cao, các trường hợp sơ suất, bỏ sót (ví dụ, giả vờ là khách hàng, nghiên cứu các hướng dẫn/tiêu chuẩn hiện hành, xem xét và thử nghiệm phần mềm tương tự, vv.) 2. thử nghiệm đặc tả kỹ thuật cấp thấp bởi đảm bảo đầy đủ, chính xác, sự tinh tế, nhất quán, phù hợp, khả thi, v.v.	
Thử nghiệm động	Yêu cầu định nghĩa những gì phần mềm và phần cứng thực hiện, bao gồm: * *thử nghiệm dữ liệu*, đó là việc kiểm tra thông tin các đầu vào và đầu ra người sử dụng * *thử nghiệm điều kiện biên*, đó là thử nghiệm các trạng thái tại cạnh của các giới hạn hoạt động dự kiến của phần mềm * *thử nghiệm đầu vào*, đó là thử nghiệm với dữ liệu vô giá trị, không hợp lệ. * *thử nghiệm trạng thái*, đó là thử nghiệm các phương thức và quá trình chuyển đổi giữa các chế độ phần mềm với các biến trạng thái tức là, các điều kiện chạy đua, thử nghiệm sự lặp lại (lý do chính là để phát hiện rò rỉ bộ nhớ), ứng suất (phần mềm đói = bộ nhớ thấp, cpu chậm = mạng chậm), tải (nguồn cấp dữ liệu phần mềm = kết nối nhiều thiết bị ngoại vi, xử lý một lượng lớn dữ liệu, web server có nhiều khách hàng truy cập vào nó, v.v.), ...	Thử nghiệm hệ thống đang chạy trong khi theo dõi mã, các sơ đồ nguyên lý, v.v. Trực tiếp thử nghiệm ở mức độ thấp và mức độ cao dựa trên sự hiểu biết hoạt động chi tiết, truy cập vào các biến và các kết xuất bộ nhớ. Tìm kiếm các lỗi tham chiếu dữ liệu, các lỗi khai báo dữ liệu, lỗi tính toán, các lỗi so sánh, lỗi lưu đồ điều khiển, các lỗi tham số cho chương trình con, các lỗi I/O, v.v.

Bảng 4-2: Ma trận mô hình thử nghiệm

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

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

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

Câu hỏi ôn tập

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

CHƯƠNG 5 PHÁT TRIỂN HỆ THỐNG NHÚNG DỰA TRÊN VXL ARM

5.1 Giới thiệu chung

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

5.2 Kiến trúc của hệ vi xử lý nhúng ARM

5.2.1 Lõi ARM

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

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

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

Ngoài ra, kiến trúc ARM có thể cung cấp:

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

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

5.2.2 Thanh ghi và các chế độ hoạt động

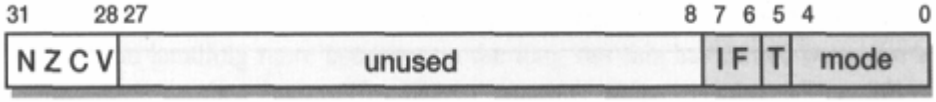
Lõi ARM có 37 thanh ghi trong đó có 31 thanh ghi đa dụng. Tuy nhiên tại một thời điểm chỉ có 16 thanh ghi đa dụng và 2 thanh ghi trạng thái hiển thị. Các thanh ghi khác ở dạng ẩn, chỉ hiển thị ở một số chế độ hoạt động riêng.

Các thanh ghi đa dụng có thể dùng để lưu dữ liệu hoặc địa chỉ. Các thanh ghi này được đánh dấu bằng ký hiệu r. Tất cả các thanh ghi đều là 32 bit.
Trong các thanh ghi đa dụng trên, có 3 thanh ghi còn được dùng để các chức năng hoặc nhiệm vụ đặc biệt riêng: r13, r14, r15.

- Thanh ghi r13 được dùng làm stack pointer (sp).
- Thanh ghi r14 được gọi là thanh ghi kết nối (lr) chứa địa chỉ quay lại của chương trình khi chương trình chạy một hàm con.
- Thanh ghi r15 là bộ đếm chương trình (pc) và chưa địa chỉ của lệnh tiếp theo.

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

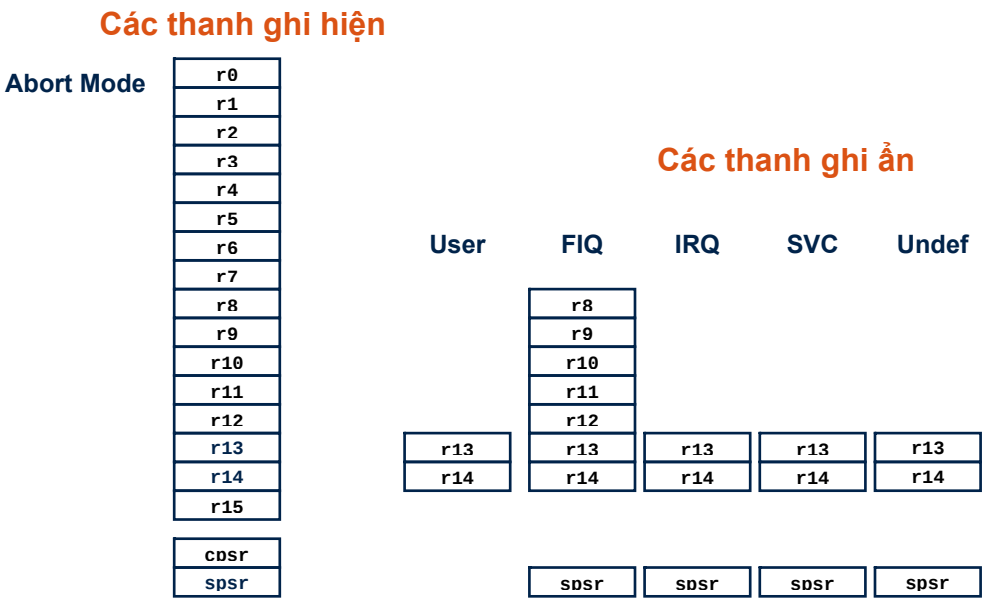
Thanh ghi trạng thái chương trình hiện tại(cpsr) : cpsr có 4 trường, mỗi trường có 8 bit: cờ, trạng thái, mở rộng và điều khiển. Hiện tại phần trạng thái và mở rộng được dự trữ cho các thiết kế tương lai.



Hình 5-1 Cấu trúc của thanh ghi trạng thái chương trình hiện tại

Các cờ của cpsr như sau:

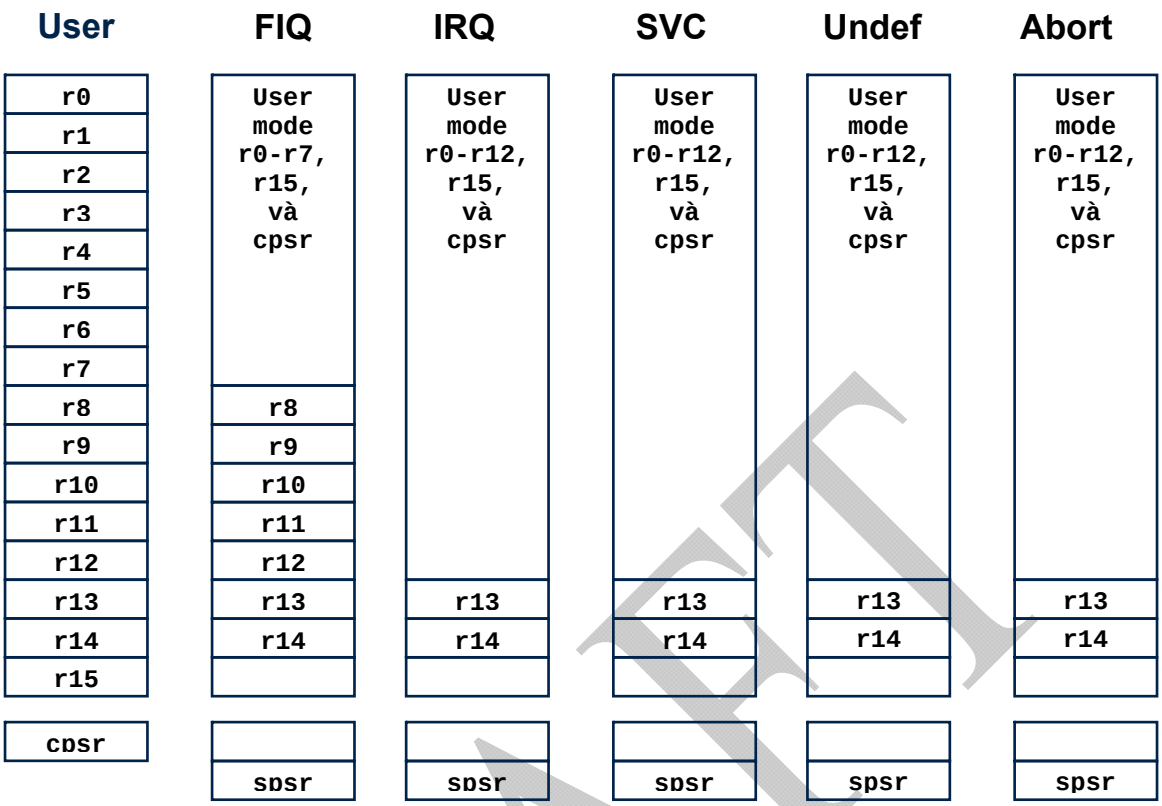
- N: Negative- cờ này được bật khi bit cao nhất của kết quả xử lý ALU bằng 1.
- Z: Zero- cờ này được bật khi kết quả cuối cùng trong ALU bằng 0.
- C: Carry- cờ này được bật khi kết quả cuối cùng trong ALU lớn hơn giá trị 32bit và tràn.
- V: Overflow-cờ báo tràn sang bit dấu.



Hình 5-2 Các thanh ghi của lõi ARM

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

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



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

Hoạt động của các chế độ như sau:

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

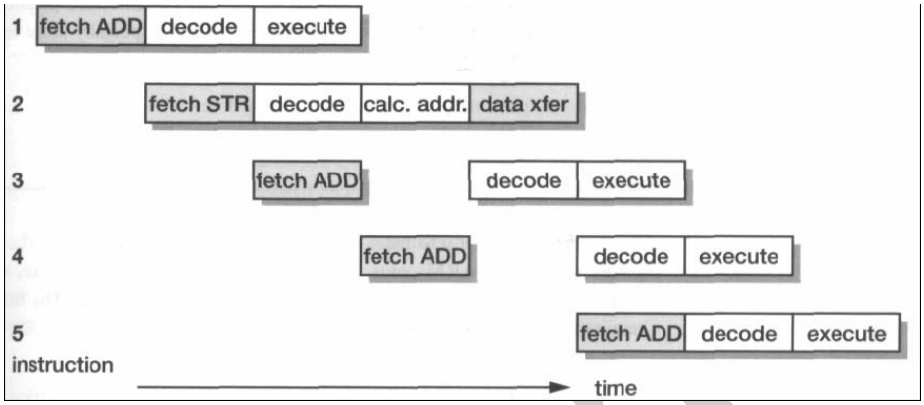
Đối với từng chế độ, có thể có các thanh ghi riêng cho từng chế độ đó.

5.2.3 Pipeline

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

Dòng chảy lệnh 3 tác vụ:

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



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

Dòng chảy lệnh 5 tác vụ:

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

Dòng chảy lệnh 6 tác vụ:

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

Dòng chảy lệnh 9 tác vụ:

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

5.2.4 Cấu trúc bus

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

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

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

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

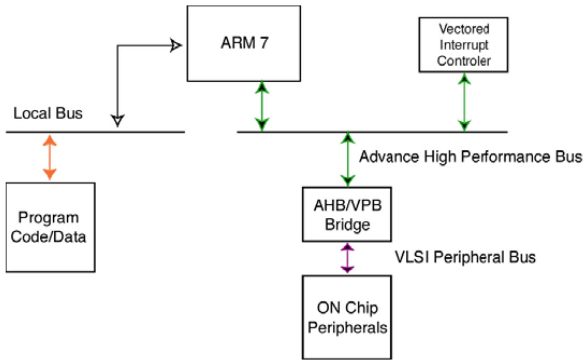
AHB cung cấp băng thông dữ liệu cao hơn so với ASB bởi nó được thiết kế dựa trên lược đồ bus ghép kênh tập trong hơn là thiết kế bus hai chiều ASB. Sự thay đổi này cho phép bus AHB chạy ở tốc độ clock cao hơn và là bus ARM đầu tiên hỗ trợ độ rộng bus lên tới 64 và 128bit. ARM cũng đã giới thiệu 2 biến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

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

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

Chú ý, sau khi reset, bộ chia tần VPB được thiết lập hệ số chia là 4 nên các thiết bị ngoại vi on-chip sẽ chạy ở tốc độ bằng ¼ tần số của CPU.

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

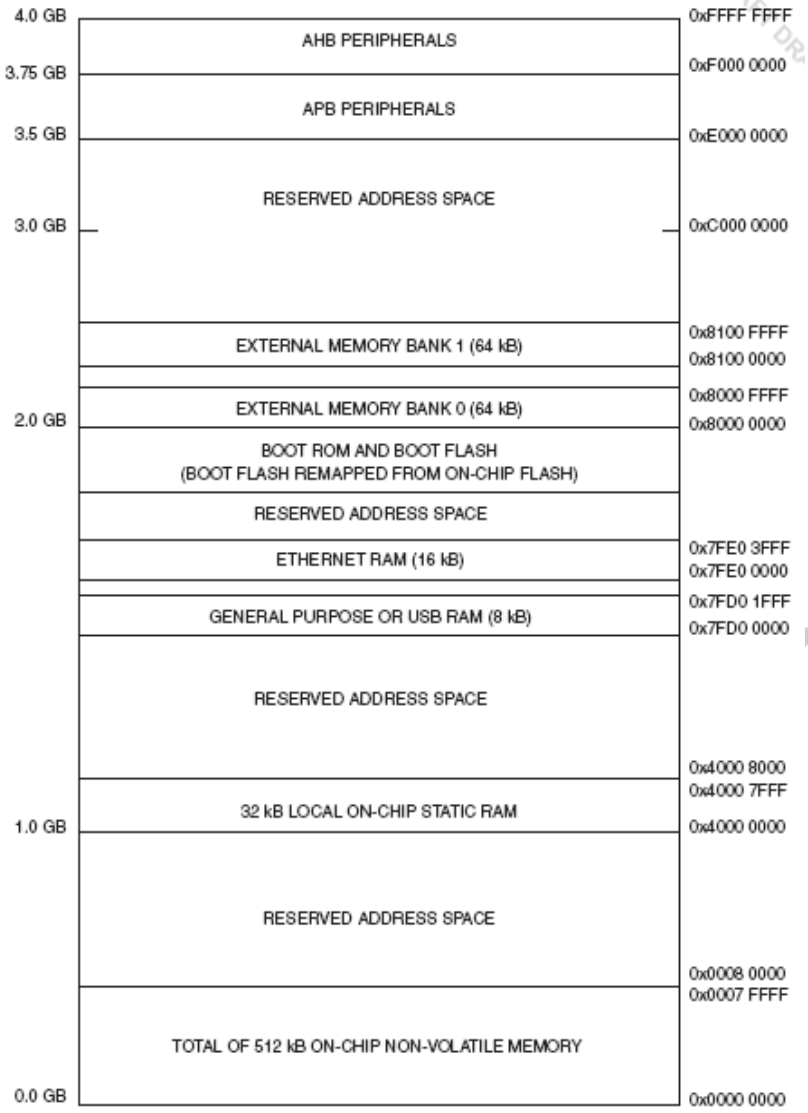


Hình 5-5 . Cấu trúc bus LPC2378

5.2.5 Bản đồ bộ nhớ:

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

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



Hình 5-6 . Bản đồ bộ nhớ của LPC2378

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

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

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

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

và một chương trình giám sát thời gian thực của ARM, chúng được đặt dưới địa chỉ 0x80000000. Khoảng địa chỉ từ 0x80000000 đến 0xE0000000 dành cho bộ nhớ ngoài. Các ngoại vi người sử dụng kết nối với VPB được ánh xạ vào vùng 0xE0000000 và 0xF00000000 như trong bảng 1.

5.2.6 Tập lệnh ARM

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

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

Danh sách các lệnh của ARM

Các lệnh xử lý dữ liệu

Các lệnh xử lý dữ liệu thực thi các phép tính đối với dữ liệu trên các thanh ghi. Các lệnh đó bao gồm chuyển dữ liệu, các phép tính số học, logic, các phép so sánh và phép nhân.
Nếu các lệnh này có thêm S ở cuối, nó sẽ cập nhật cờ trên thanh ghi CPRS. Các lệnh dịch chuyển và phép toán logic cập nhật cờ Carry C, cờ Negative N, và cờ Zero Z.

Các lệnh dịch chuyển

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

Cú pháp : <instruction>{<cond>}{S} Rd, N
Có 2 lệnh dịch chuyển
MOV: Chuyển một giá trị 32 bit đến thanh ghi (Rd=N)
MVN: Chuyển giá trị đảo 32 bit đến thanh ghi (Rd= ~N)

Ví dụ:

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

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

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

Các lệnh số học

Các lệnh số học thực hiện cộng và trừ các giá trị 32 bít có dấu và không có dấu. Kết quả là 32 bit và được đặt trong một thanh ghi. Cấu trúc của lệnh có 3 địa chỉ.
Cú pháp : <instruction>{<cond>}{S} Rd, Rn, N

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

Ví dụ:

Trước:

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

Sau:

```
r0 = 0x00000001
```

Trước:

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

Sau:

```
r0 = -r1 = 0xffffff8
```

Các lệnh logic

Các lệnh logic thực hiện các phép toán logic theo bit trên các thanh ghi
Cú pháp: <instruction>{<cond>}{S} Rd, Rn, N

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

Ví dụ:

Trước;

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

Sau:

```
r0 = 0x12345678
```

Trước:

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

Sau:
r0 = 0b1010

Trước:
r1=0x11111111
r2=0x01100101
BIC r0, r1, r2
Sau:
R0=0x10011010

Các lệnh so sánh

Các lệnh so sánh dùng để so sánh hoặc kiểm tra một thanh ghi với một giá trị 32 bit. Các lệnh này không ảnh hưởng đến các thanh ghi, chỉ thay đổi cập nhật các bit cờ trên thanh ghi CPRS. Các lệnh này không cần thêm S
Cấu trúc: <instruction>{<cond>} Rn, N

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

Các lệnh nhân

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

Cấu trúc:
MLA{<cond>}{S} Rd, Rm, Rs, Rn
MUL{<cond>}{S} Rd, Rm, Rs

```
MUL  R0, R1, R2         ; R0 = (R1xR2)[31:0] (32 bits)
MLA  R4, R3, R2, R1  ; R4 = R3xR2+R1
```

Các lệnh rẽ nhánh

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

Cấu trúc:
B{<cond>} label
BL{<cond>} label
BX{<cond>} Rm
BLX{<cond>} label | Rm

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

BL label: rẽ nhánh có liên kết, pc=label, địa chỉ quay lại được lưu vào thanh ghi lr (R14)

BX Rm; rẽ nhánh trao đổi, pc = Rm & 0xfffffffe, T = Rm &1 (T là Thumb bit trên CPSR)

BLX

Ví dụ:

```
        B  label
                …

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

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

Ví dụ:

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

Các lệnh chuyển dữ liệu Load- Store

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

Load-store dùng một thanh ghi

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

Cấu trúc:

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

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

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

Ví dụ:

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

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

Các chế độ đánh địa chỉ

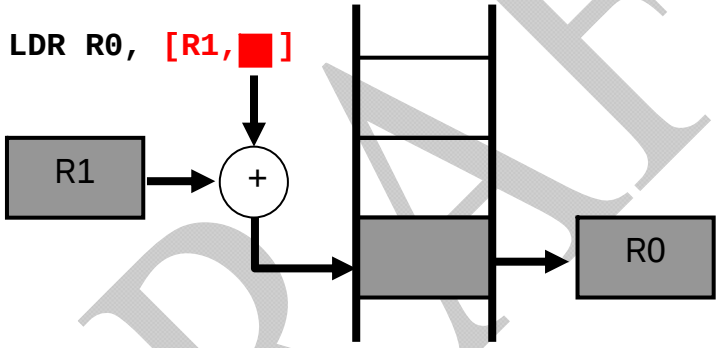
VXL ARM hỗ trợ ba chế độ đánh địa chỉ:

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

  Ví dụ:

  Trước

  ```
  r0 = 0x00000000
  r1 = 0x00090000
  mem32[0x00009000] = 0x01010101
  mem32[0x00009004] = 0x02020202

  LDR R0, [R1, #4]  ; R0=mem[R1+4]
                    ; R1 unchanged
  r0 = 0x02020202
  r1 = 0x00009000
  ```

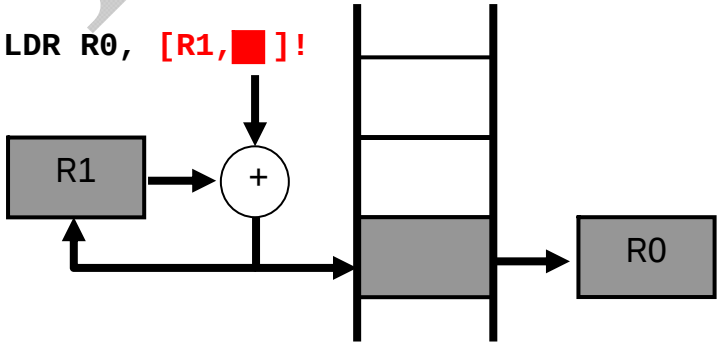


Hình 5-7. Sơ đồ hoạt động lệnh LDR (Pre-indexed addressing)

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

  Ví dụ:

  ```
  LDR R0, [R1, #4]!  ; R0=mem[R1+4]
                     ; R1=R1+4
  r0 = 0x02020202
  r1 = 0x00009004
  ```

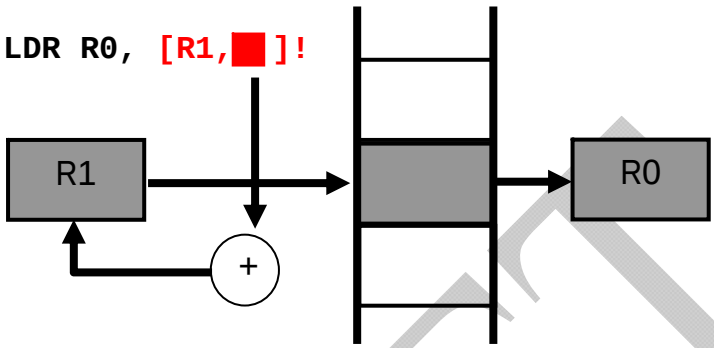


Hình 5-8. Sơ đồ hoạt động lệnh LDR (Auto-indexing addressing)

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

Ví dụ :

```
LDR  R0, R1, #4    ; R0=mem[R1]
                   ; R1=R1+4
r0 = 0x01010101
r1 = 0x00009004
```



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

Load-store dùng nhiều thanh ghi

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

Cấu trúc:

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

LDM Nạp nhiều thanh ghi
STM Lưu nhiều thanh ghi

Chế độ đánh địa chỉ:

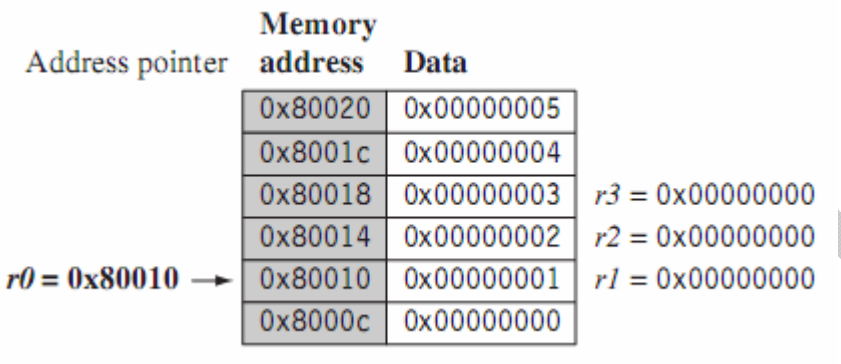
Đuôi	ý nghĩa
IA	tăng địa chỉ sau khi thực hiện
IB	tăng địa chỉ trước khi thực hiện
DA	giảm địa chỉ sau khi thực hiện
DB	giảm địa chỉ trước khi thực hiện

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

Ví dụ :

Trước:

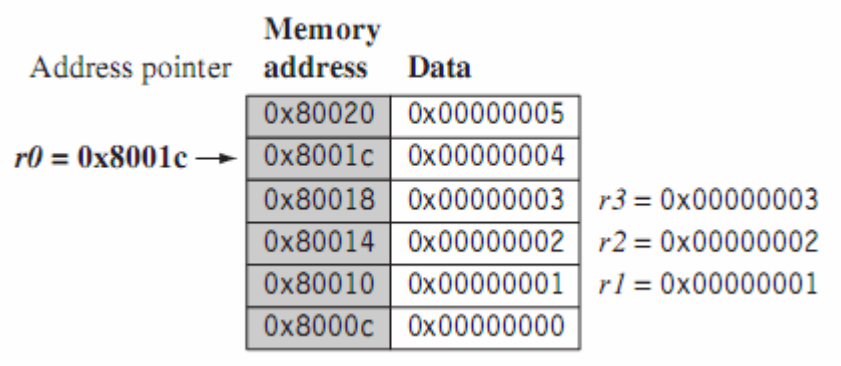
mem32[0x80018] = 0x03
mem32[0x80014] = 0x02
mem32[0x80010] = 0x01
r0 = 0x00080010
r1 = 0x00000000
r2 = 0x00000000
r3 = 0x00000000



LDMIA r0!, {r1-r3}

Sau:

r0 = 0x0008001c
r1 = 0x00000001
r2 = 0x00000002
r3 = 0x00000003

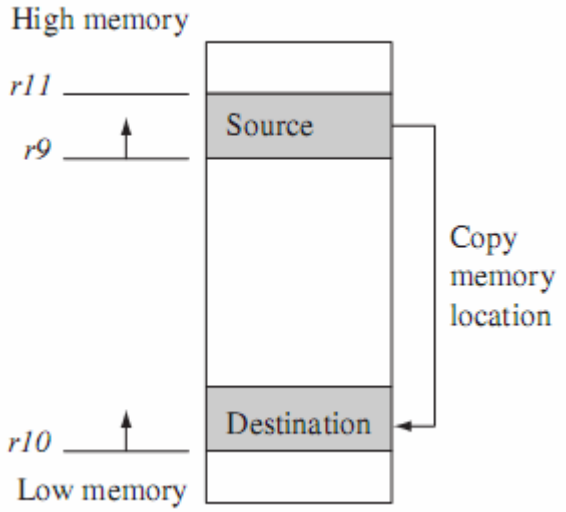


Nếu dùng lệnh LDMIB

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

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

R9: địa chỉ nguồn
R10: địa chỉ đích
R11: địa chỉ cuối của nguồn




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

Các lệnh này có thể ứng dụng trong stack

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

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

Cấu trúc:

SWP{B}{<cond>} Rd,Rm,[Rn]
SWP đổi nội dung 32 bits giữa thanh ghi và bộ nhớ
    tmp =mem32[Rn]
    mem32[Rn] = Rm
    Rd = tmp
SWPB đổi nội dung 8 bits giữa thanh ghi và bộ nhớ

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

Ví dụ

Trước:

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

Sau:

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

5.2.7 Tập lệnh Thumb

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

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

5.3.1 Lập trình các thanh ghi

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

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

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

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

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

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

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

5.3.2 Thiết kế điều khiển I/O

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

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

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

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

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

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

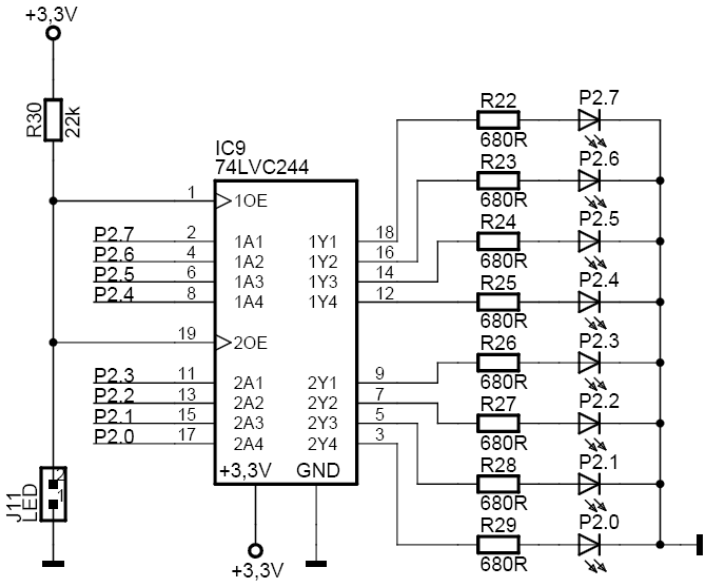
  Ví dụ:

  IOSET= 1>>10 ; ***// Đưa pin 10 lên mức cao***
- IOCLR : tương tự như thanh ghi set, tuy nhiên khi viết 1 bít nào đó tương ứng với đưa pin đó về mức thấp.

  Ví dụ : IOCLR= (1<<1)|(1<<2); ***//Đưa pin 1 và pin 2 về mức thấp***
- IODIR : thanh ghi chọn hướng của pin là input hay output.Nếu bit tương ứng với pin trong thanh ghi bằng 1 thì pin đó là pin output còn bit đó bằng 0 thì pin tương ứng là pin input.

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

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



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

Ví dụ 1 : Điều khiển nháy led

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

Chương trình assembly:

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

```
        ENTRY

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

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

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

          ldr r0,FIO2MASK          ;cho phep dieu khien cac pin
          mov r3,#0x00
          str r3,[r0]

          mov r3,#0xff
  ; vong lap chinh,nhay cac led

led_blinky
          ldr r0,FIO2SET   ;bat led
          str r3,[r0]
          bl delay         ;delay trong khoang 1/2 sec

          ldr r0,FIO2CLR           ;tat led
          str r3,[r0]
          bl delay         ;delay trong khoang 1/2 sec
          b led_blinky

delay           ;ham delay
          ldr r0,DELAY_VAL ;khoi tao gia tri dem
loop_delay
          subs r0,r0,#0x01
          bne     loop_delay

          bx lr            ;tro ve chuong trinh chinh


; khai bao cac thanh ghi dac biet cho viec dieu khien cac io
FIO_BASE_ADDR       EQU     0x3FFFC000
PINSEL10        DCD     0xe002c028
FIO2DIR         DCD     (FIO_BASE_ADDR + 0x40)
FIO2MASK        DCD     (FIO_BASE_ADDR + 0x50)
FIO2PIN          DCD    (FIO_BASE_ADDR + 0x54)
FIO2SET         DCD     (FIO_BASE_ADDR + 0x58)
FIO2CLR         DCD     (FIO_BASE_ADDR + 0x5C)
DELAY_VAL   DCD     50000
```

END

Chương trình viết bằng C:

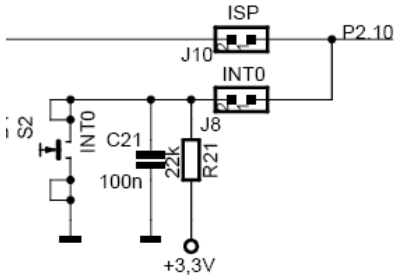
```
#include <LPC23xx.H>
//ham tao tre
void delay(unsigned long n)
 {
     while(n--) {}
 }

int main()
 {

  //khởi tạo các pin io từ P2.0→P2.7 là output
    PINSEL10 = 0x00;
    FIO2DIR  = 0xff;
    FIO2MASK = 0x00;
    FIO2CLR  = 0xff;

    while(1)
     {
         //Set các pin lên mức cao
         FIO2SET=0xff;
         delay(500000);

         //Clear các pin về mức thấp
         FIO2CLR =0xff;
         delay(500000);
     }
    return 0;
 }
```

Ví dụ 2: Đọc phím bấm:

Trên board MCB2300, phím S2 được kết nối tới P2.10.

Về hoạt động:

- Ở trạng thái bình thường: P2.10 ở mức cao qua điện trở kéo lên 3.3V
- Khi phím S2 được nhấn: P2.10 ở trạng thái thấp



Hình 5-11. Sơ đồ kết nối của button S2

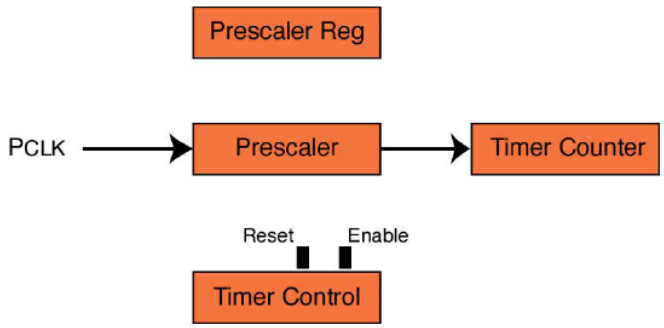
Chương trình của chúng ta sẽ thực hiện đọc phím, khi phím được nhấn nó sẽ thể hiện thông qua việc sáng Led trong 1 khoảng thời gian

Chương trình:

```
#include <LPC23xx.H>

void delay()
 {
      int i;
      for (i=0;i<=4000000;i++) {}
 }

int main()
  {
      unsigned int status=0;
      //inits Port 2
      PINSEL10 =  0;
      FIO2MASK = 0;
      FIO2DIR  = 0xff; //pin 0..7 are output pins
                  //pin 10 is input pin

      //clear pins 0..7 to turn off all leds
      FIO2CLR = 0xff;

      while(1)
       {
            //checks button status
            if((FIO2PIN&(1<<10))==0)
             {
               FIO2CLR = 0xff;
               FIO2SET = (1<<status);
               status++;
               status %= 8;
               delay();
             }
       }
  }
```

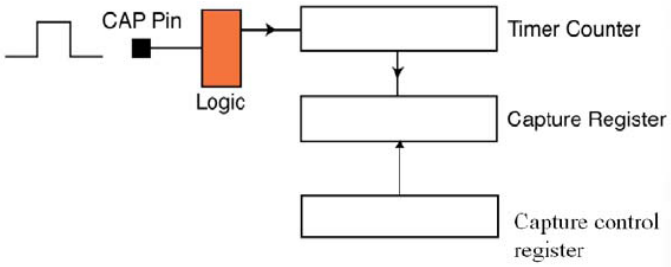
5.3.3 Lập trình điều khiển bộ định thời (Timer) và bộ đếm (Counter)

Timer/Counter là các thiết bị ngoại vi được thiết kế để đếm số chu kì của 1 clock bên trong hệ thống hoặc từ một nguồn clock ngoài. Trong LPC2378 có bốn bộ Timer/Counter. Tất cả đều giống nhau về cấu trúc và cách sử dụng. Các timer đều hoạt động dựa trên các bộ đếm timer 32 bit. Nguồn clock bên trong cho tất cả bộ định thời là clock ngoại vi PCLK được lấy từ clock dành cho lõi CCLK



Hình 5-12. Mô hình bộ timer

Tốc độ tick của bộ timer được điều khiển bởi giá trị được lưu trong thanh ghi Prescaler Reg. Giá trị bộ đếm Prescaler sẽ tăng theo mỗi xung của PCLK cho đến khi nó đạt đến giá trị trong Prescaler Reg. Khi nó đạt tới giá trị này thì thanh ghi Timer Counter được tăng lên 1 và thanh ghi Prescaler reset về 0 và bắt đầu đếm tiếp. Thanh ghi điều khiển Timer Control chỉ gồm 2 bít mà được sử dụng để cho phép/cấm bộ timer và reset bộ đếm của nó.



Hình 5-13 Sơ đồ hoạt động các thanh ghi Timer

Ngoài chức năng cơ bản của một bộ timer, mỗi timer còn có thêm 4 kênh capture. Các kênh capture cho phép bạn chộp lại giá trị của bộ timer counter khi có một sự chuyển mức của tín hiệu ở 1 kênh đầu vào.

Mỗi một kênh capture có một pin tương ứng được cho phép thông qua khối kết nối pin. Thanh ghi **capture control register** có thể cấu hình sườn nào đó kích hoạt 1 sự kiện capture hoặc cả 2 sườn. Khi sự kiện capture xảy ra, giá trị trong timer counter sẽ được ghi vào thanh ghi capture tương ứng và nếu cần thiết, một ngắt có thể được tạo ra.

Mô tả các pin dành cho bộ Timer/Counter:

Pin	Type	Mô tả
Cap0[1:0] Cap1[1:0] Cap2[1:0] Cap3[1:0]	Input	Capture Signals – 1 sự chuyển mức trên 1 pin capture có thể được cấu hình để copy vào thanh ghi Capture tương ứng gí trị hiện thời của bộ đếm thời gian TC (Timer Counter) và tùy chọn có thể tạo ra các ngắt
MAT0[1:0] MAT1[1:0] MAT2[1:0] MAT3[1:0]	Output	External Match Output – Khi 1 thanh ghi match(MR3:0) bằng với giá trị trong bộ đếm thời gian TC (Timer Counter) đầu ra này có thể được cấu hình để thay đổi mức, thiết lập mức cao, thiết lập mức thấp.

Mô tả các thanh ghi liên quan đến bộ Timer/Counter:

Thanh ghi	Mô tả	Kiểu truy cập
IR	Interrupt Register : IR có thể được ghi vào để xóa các ngắt hoặc có thể được đọc để nhận diện một trong tám nguồn ngắt có thể đang chờ được xử lý	Read/Write(R/W)
TCR	Timer Control Register : được sử dụng để điều khiển các chức năng của	R/W

	bộ Timer/Counter. Nó có thể dừng hoặc reset bộ Timer/Counter	
TC	Timer Counter : là thanh ghi 32 bit tăng theo mỗi chu kì PR+1 của PCLK. Được điều khiển bởi thanh ghi TCR	R/W
PR	Prescale Register : khi thanh ghi Prescale Counter bằng giá trị trong thanh ghi này thì ở chu kì clock tiếp theo nó tăng TC và clear PC	R/W
PC	Prescaler Counter : là thanh ghi 32 bit, thực hiện chức năng đếm giá trị clock và so sánh với PR	R/W
MCR	Match Control Register : được sử dụng để điều khiển nếu một ngắt được tạo ra và nếu TC reset khi có một sự kiện Match xảy ra	R/W
MR0/1/2/3	Match Register 0/1/2/3 : MRn có thể được cho phép qua MCR, khi giá trị trong thanh ghi MRn bằng với giá trị trong TC nó có thể tạo ra các đáp ứng như reset TC, dừng cả TC và PC hoặc tạo ra 1 ngắt hoặc không làm gì	R/W
CCR	Capture Control Register: CCR điều khiển sườn nào của các đầu vào Capture để copy giá trị các Capture Register và có hoặc không tạo ra một ngắt khi có một sự kiện capture xảy ra	R/W
CR0/1/2/3	Capture Register 0/1/2/3 : CR0/1/2/3 sẽ được load vào giá trị của TC khi có một sự kiện capture xảy ra ở pin capture tương ứng	ReadOnly(RO)
EMR	External Match Register : điều khiển các chân ngoài MATn.0-3	R/W
CTCR	Count Control Register: chọn mode hoạt động là Timer hay Counter và từ đó chọn tín hiệu clock nguồn	R/W

1.1. Chương trình điều khiển:

Ví dụ bên dưới sẽ sử dụng bộ TIMER0 để tạo ra trễ chính xác 0.5s. Cứ sau mỗi 0.5s, các led sẽ được cập nhật.
Chú ý trong ví dụ chúng ta thiết lập tốc độ PCLK cho TIMER0 là 15MHz.

Chương trình:

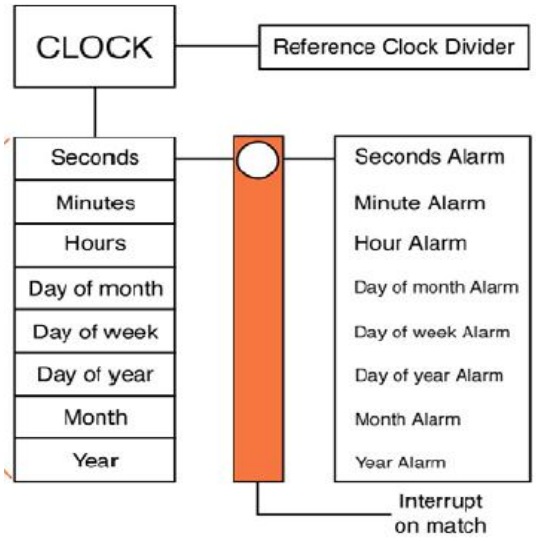
```
#include <LPC23xx.H>

// PCLK for TIMER0
```

```
#define TIMER_CLOCK 15000000

int main()
 {
      int status=0;
      /* inits timer0 */

      /* reset timer */
      T0TCR = 0x02;
      /* Set value for PR to TC increments every 0.5s */
      T0PR  = (TIMER_CLOCK/2)-1;

      /* init leds */
      PINSEL10 = 0;
      FIO2MASK = 0;
      FIO2DIR  = 0xFF;


      while(1)
       {
            /* update leds */
            FIO2CLR=0xFF;
            FIO2SET=(1<<status);
            status++;
            status%=8;

            /* delay 0.5ms */
            T0TCR=0x02;          //Timer0 reset
            T0TCR=0x01;          //TIMER0 run
            while(T0TC!=1)       {}
            T0TCR=0;             //Timer0 stop
       }

      return 0;
 }
```

5.3.4 Đồng hồ thời gian thực

Đồng hồ thời gian thực là một bộ lịch biểu cho phép thực hiện tính toán quản lý thời gian trong lịch biểu như ngày , tháng, năm, tuần lễ,.. và đặc biệt là tạo ra giờ chính xác theo giờ phút giây.

Bộ RTC trong LPC2378 tạo ra lịch biểu chính xác cho đến năm 2099 và tùy chọn có thể tạo ra các ngắt khi thời gian hiện thời phù hợp với những thông số thời gian được chọn lựa trong các thanh ghi Alarm.Bộ RTC có thể chọn hoạt động trên thạch anh 32KHz ngoài hoặc từ PCLK bên trong. Nó cũng có bộ nhớ SRAM công suất thấp 2K bên trong được gọi là battery SRAM. RTC và battery SRAM được cung cấp bởi nguồn riêng từ pin Vbat. Chúng được thiết kế để tiêu tốn công suất thấp và có thể chạy từ nguồn pin.



Hình 5-14. Sơ đồ RTC

Phía trên là hình vẽ mô tả hoạt động chung của bộ RTC, tín hiệu clock đi vào bộ RTC qua khối **Reference Clock Divider** sẽ được chia ra để tạo ra nguồn clock phù hợp. Tín hiệu clock này vào bộ **Clock** tạo ra tín hiệu điều khiển cho các bộ đếm Second, Minute, Hours,… Khi được tùy chọn, nếu thời gian hiện tại phù hợp với các giá trị thời gian trong các thanh ghi Alarm thì nó sẽ tạo ra các ngắt theo ý muốn.

Mô tả các pin dành cho chức năng RTC:

Pin	Type	Mô tả
RTCX1	I	Đầu vào mạch dao động RTC
RTCX2	O	Đầu ra từ mạch dao động RTC
Vbat	I	Nguồn cung cấp cho bộ RTC

Mô tả các thanh ghi dành cho chức năng RTC:

Register	Size	Mô tả	Kiểu truy cập
ILR	2	Interrupt Location Register : đọc thanh ghi này để nhận diện nguồn ngắt và ghi 1 và bit tương ứng sẽ clear ngắt tương ứng	R/W
CTC	15	Clock Tick Counter : được reset thông qua thanh ghi CCR. Nó chứa các bít điều khiển bộ chia clock	RO
CCR	4	Clock Control Register : điều khiển hoạt động của bộ chia cloack như chọn nguồn clock, reset,..	R/W
CIIR	4	Counter Increment Interrupt Register : chọn lựa bộ đếm nào trong các bộ đếm giây, phút, giờ ,.. sẽ tạo ra ngắt mỗi khi các bộ đếm nay tăng	R/W
AMR	8	Alarm Mask Register : được dùng để	R/W

		chọn lựa tổ hợp các thanh ghi nào trong các thanh ghi Alarm được so sánh với thời gian hiện tại để tạo ra ngắt	
CTIME0/1/2	32	Consolidated Time Register 0/1/2 : chứa tổ hợp các giá trị thời gian bậc thấp : giây, phút, giờ, và ngày trong tuần.	R/W
SEC MIN HOUR DOM DOW DOY MONTH YEAR		Là các thanh ghi chứa giá trị thời gian hiện thời	R/W
ALSEC ALMIN ALHOUR ALDOM ALDOW ALDOY ALMON ALYEAR		Các thanh ghi Alarm chứa giá trị để so sánh với giá trị thời gian hiện thời, thực hiện các tác vụ tương ứng	
PREINT	13	Prescale Value, integer portion - cùng với thanh ghi PREFRAC nó được sử dụng để tạo ra giá trị chia phù hợp khi sử dụng nguồn clock là PCLK	R/W
PREFRAC	15	Prescale Value, fractional portion - cùng với thanh ghi PREINT nó được sử dụng để tạo ra giá trị chia phù hợp khi sử dụng nguồn clock là PCLK	R/W

Thiết lập giá trị PREINT và PREFRAC khi sử dụng PCLK:

Như đã giới thiệu, bộ RTC có thể hoạt động hoặc từ bộ dao động RTC 32.768 kHz bên ngoài hoặc từ PCLK cung cấp bởi clock hệ thống sau khi đã được thay đổi bởi bộ **Reference Clock Divider**

Bộ **Reference Clock Divider** cho phép tạo ra tín hiệu clock 32.768 từ bất cứ nguồn clock nào lớn hơn 65.536 kHz(= 2x32.768 kHz). Điều này cho phép bộ RTC luôn chạy ở tốc độ chính xác bất chấp tốc độ của tín hiệu cung cấp. Bộ chia trong trong **Reference Clock Divider** cho phép chia tín hiệu vào từ PCLK với 1 giá trị chưa cả phần nguyên và phần phân số. Nên tốc độ đầu ra có thể biến thiên theo thời gian xong tổng hợp thì nó luôn đếm được 32,768 lần trên 1 giây.

Phần nguyên của bộ chia (chứa trong thanh ghi PREINT) là giá trị 13 bit và phần phân số(chứa trong PREFRAC) là giá trị 15 bit. Cụ thể tính toán PREINT và PREFRAC:

PREINT = phần nguyên(PCLK/32768) -1
PREFRAC = PCLK – ((PREINT + 1) x 32768)

5.3.5 Bộ điều chế độ rộng xung

Bộ điều chế độ rộng xung PWM có chức năng tạo ra các tín hiệu chu kì với độ rộng xung có thể điều khiển thông qua các thanh ghi điều khiển với giá trị chu kì, độ rộng xung được tính toán trước theo yêu cầu cụ thể của ứng dụng. Các bộ PWM hoạt động dựa trên các khối Timer và kế thừa các chức năng của nó.

Như ta biết các bộ Timer được thiết kế để đếm các chu kì của đồng hồ ngoại vi PCLK và tùy chọn có thể tạo ra các ngắt hoặc thực hiện các yêu cầu nào đó khi 1 giá trị thời gian xác định thiết lập trên các thanh ghi Match bằng với giá trị trong thanh ghi TC . Chức năng PWM dựa trên các chức năng này của bộ Timer.

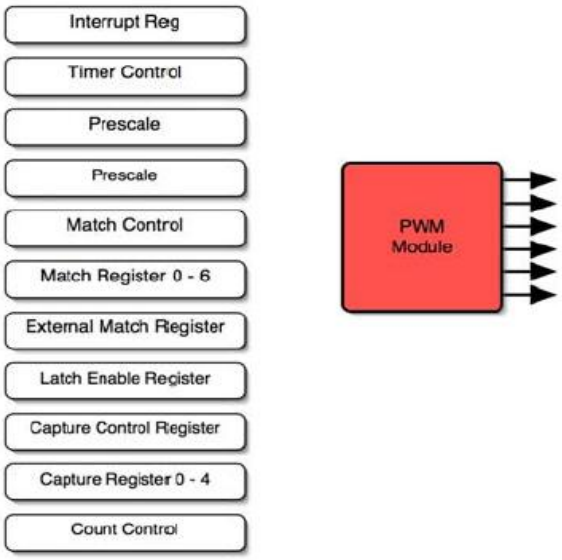
Với khả năng điều khiển riêng các thời điểm sườn dương và sườn âm của tín hiệu PWM nên các bộ PWM được sử dụng với nhiều ứng dụng hơn. Ví dụ điển hình là các bộ điều khiển motor nhiều phase.

Với mỗi tín hiệu đầu ra của bộ điều khiển độ rộng xung được điều khiển thông qua 2 hoặc 3 thanh ghi Match:

- Trường hợp sử dụng 2 thanh ghi Match: được sử dụng để tạo tín hiệu ra được điều khiển theo 1 sườn :
  - Một thanh ghi (PWMMR0) điều khiển chu kì tín hiệu
  - Một thanh ghi khác dùng để điều khiển vị trí của sườn
- Trường hợp sử dụng 3 thanh ghi Match: được sử dụng để tạo ra tín hiệu ra được điều khiển theo cả 2 sườn:
  - Một thanh ghi (PWMMR0) điều khiển chu kì tín hiệu
  - Hai thanh ghi còn lại điều khiển các vị trí của 2 sườn.

Do sử dụng cùng chung PWMMR0 nên chu kì của các tín hiệu đầu ra giống nhau

Hình bên dưới mô tả các thành phần của bộ PWM dưới quan điểm lập trình gồm tập các thanh ghi điều khiển và thanh ghi trạng thái, cùng các tín hiệu đầu ra.



Hình 5-15. Sơ đồ bộ PWM

Mô tả các pin sử dụng cho bộ PWM:

Pin	Type	Mô tả
PWM1[1]	Output	Đầu ra từ bộ PWM kênh 1
PWM1[2]	Output	Đầu ra từ bộ PWM kênh 2
PWM1[3]	Output	Đầu ra từ bộ PWM kênh 3
PWM1[4]	Output	Đầu ra từ bộ PWM kênh 4
PWM1[5]	Output	Đầu ra từ bộ PWM kênh 5
PWM1[6]	Output	Đầu ra từ bộ PWM kênh 6
PCAP1[1:0]	Input	Các đầu vào Capture :sự chuyển mức trên pin capture có thể đượ cấu hình để copy giá trị của TC vào thanh ghi Capture tương ứng và tùy chọn có thể tạo ra 1 ngắt

Mô tả các thanh ghi sử dụng cho bộ PWM:

Tập các thanh ghi của PWM giống với bộ Timer và thêm một tập các thanh ghi riêng

Thanh ghi	Mô tả	Kiểu truy cập
IR	Interrupt Register : IR có thể được ghi vào để xóa các ngắt hoặc có thể được đọc để nhận diện một trong tám nguồn ngắt có thể đang chờ được xử lý	R/W
TCR	Timer Control Register : được sử dụng để điều khiển các chức năng của bộ Timer/Counter. Nó có thể cắm hoặc reset bộ Timer/Counter	R/W
TC	Timer Counter : là thanh ghi 32 bit tăng theo mỗi chu kì PR+1 của PCLK. Được điều khiển bởi thanh ghi TCR	R/W
PR	Prescale Register : khi thanh ghi Prescale Counter bằng giá trị trong thanh ghi này thì ở chu kì clock tiếp theo nó tăng TC và clear PC	R/W
PC	Prescaler Counter : là thanh ghi 32 bit, thực hiện chức năng đếm giá trị clock và so sánh với PR	R/W
MCR	Match Control Register : được sử dụng để điều khiển nếu một ngắt được tạo ra và nếu TC reset khi có một sự kiện Match xảy ra	R/W
MR0/1/2/3/4/5/6	Match Register 0/1/2/3/4/5/6 : MRn có thể được cho phép qua MCR, khi giá trị trong thanh ghi MRn bằng với giá trị trong TC nó có thể tạo ra các đáp ứng như reset TC, dừng cả TC và PC hoặc tạo ra một ngắt hoặc không làm gì. Thêm nữa nó được sử dụng để điều khiển các đầu ra bộ PWM	R/W
CCR	Capture Control Register: CCR điều khiển sườn nào của các đầu vào Capture để copy giá trị các Capture Register và có hoặc	R/W

	không tạo ra một ngắt khi có một sự kiện capture xảy ra	
CR0/1/2/3	Capture Register 0/1/2/3 : CR0/1/2/3 sẽ được load vào giá trị của TC khi có 1 sự kiện capture xảy ra ở pin capture tương ứng	RO
PCR	PWM Control Register : cho phép các đầu ra PWM và chọn loại của kênh là điều khiển đơn sườn hay cả 2 sườn	R/W
CTCR	Count Control Register: chọn mode hoạt động là Timer hay Counter và từ đó chọn tín hiệu clock nguồn	R/W
LER	Load Enable Register : cho phép sử dụng các giá trị Match mới trong bộ PWM. Chỉ khi bít này được cho phép thì đầu ra mới đáp ứng với các giá trị trong thanh ghi Match	R/W

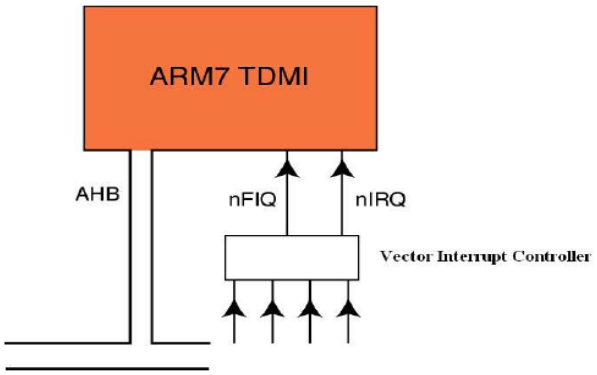
5.3.6 Điều khiển ngắt

Có hai loại bộ điều khiển ngắt được sử dụng trong bộ vi xử lý ARM là bộ điều khiển ngắt chuẩn và bộ điều khiển ngắt vector VIC – *vecter interrupt controller.*

Bộ điều khiển chuẩn gửi một tín hiệu ngắt đến lõi vi xử lý khi một thiết bị ngoài yêu cầu phục vụ. Nó có thể được lập trình để bỏ qua yêu cầu ngắt từ một hoặc nhiều thiết bị. Trình xử lý ngắt xác định thiết bị yêu cầu phục vụ bằng việc đọc một thanh ghi của thiết bị trong bộ điều khiển ngắt.

VIC mạnh hơn bộ điều khiển ngắt chuẩn bởi nó cho phép thiết lập các mức ưu tiên cho các ngắt và đơn giản quá trình xác định nguồn ngắt. Sau khi gắn một mức ưu tiên và địa chỉ trình xử lý ngắt với mỗi ngắt, VIC chỉ cần gửi một tín hiệu ngắt tới lõi nếu mức ưu tiên của ngắt mới cao hơn trình xử lý ngắt đang thực thi. Dựa vào tín hiệu ngắt này, VIC có thể gọi trình xử lý ngoại lệ ngắt chuẩn tại địa chỉ của trình phục vụ ngắt cho thiết bị từ VIC hoặc tạo một lệnh nhảy cho lõi đến trình xử lý ngắt trực tiếp. Cụ thể, mỗi khi có một ngắt xảy ra, VIC sẽ OR các yêu cầu ngắt, và tạo ra một tín hiệu ngắt đến một trong hai ngắt FIQ và IRQ tùy theo các thông số của các thanh ghi cấu hình cho VIC, và nó sẽ gửi địa chỉ của thường trình phục vụ ngắt vào thanh ghi VICVectAddr. Trình xử lý ngắt sẽ bắt đầu bằng việc đọc thanh ghi này và nhảy tới địa chỉ được cung cấp bởi thanh ghi này. Và khi kết thúc chương trình phục vụ ngắt, ngoài việc clear cờ ngắt của nguồn ngắt tương ứng, VICVectAddr phải được ghi vào một giá trị bất kì để update phần cứng mức ưu tiên của VIC.

CPU ARM có 2 đường phục vụ cho ngắt ngoài là FIQ cho ngắt nhanh và IRQ cho ngắt thông thường. Một hệ thống ARM7 chỉ nên có một nguồn ngắt tạo ra FIQ, các ngắt khác còn lại sẽ được kết nối tới IRQ.



Hình 5-16. Cấu trúc hệ thống xử lý ngắt trong LPC2378

5.3.7 Giao tiếp UART

Máy tính truyền dữ liệu theo hai phương pháp : song song hoặc nối tiếp. Truyền dữ liệu song song thường sử dụng nhiều đường dây dẫn để truyền dữ liệu đến một khoảng cách xa vài mét ví dụ như trong giao tiếp với máy in và ổ đĩa cứng. Phương pháp này cho phép truyền với tốc độ cao nhưng khoảng cách truyền bị hạn chế. Để truyền dữ liệu đi xa thì cần phương pháp truyền nối tiếp. Theo phương pháp này, dữ liệu được truyền đi theo từng bít.

Để tổ chức truyền tin nối tiếp, trước hết byte dữ liệu được chuyển thành các bít nối tiếp nhờ thanh ghi dịch vào song song –ra nối tiếp. Tiếp đó dữ liệu được truyền qua một đường dữ liệu đơn, ở đầu thu, nhờ một thanh ghi dịch vào nối tiếp-ra song song, dữ liệu được nhận nối tiếp và được gói thành từng byte một. Khi tín hiệu được truyền đi xa, yêu cầu có một modem để điều chế tín hiệu và sau đó giải điều chế ở bộ thu.

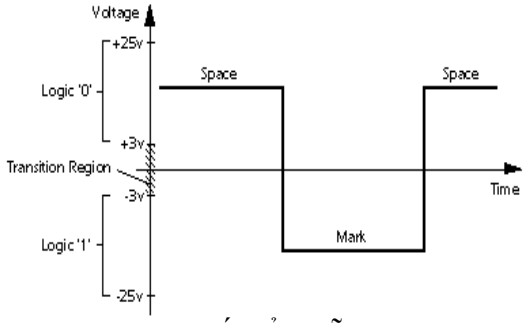
Truyền tin nối tiếp có hai phương pháp: đồng bộ và bất đồng bộ

- Đồng bộ: truyền mỗi lần một khối dữ liệu
- Bất đồng bộ: chỉ truyền từng byte một.

Chuẩn RS-232

Trong truyền tin nối tiếp, RS-232 là một chuẩn cho kết nối tín hiệu dữ liệu nhị phân nối tiếp giữa một DTE (data terminal equipment) và một DCE (data circuit-terminating equiqment) được xây dựng năm 1960 nhằm tương thích dữ các thiết bị truyền dữ liệu nối tiếp giữa các hãng khác nhau. Ngày nay nó là một chuẩn được sử dụng khá rộng dãi. Tuy nhiên do chuẩn này ra đời khá lâu trước khi có họ mạch điện tử TTL vì vậy mức điện áp của nó không tương thích với TTL.

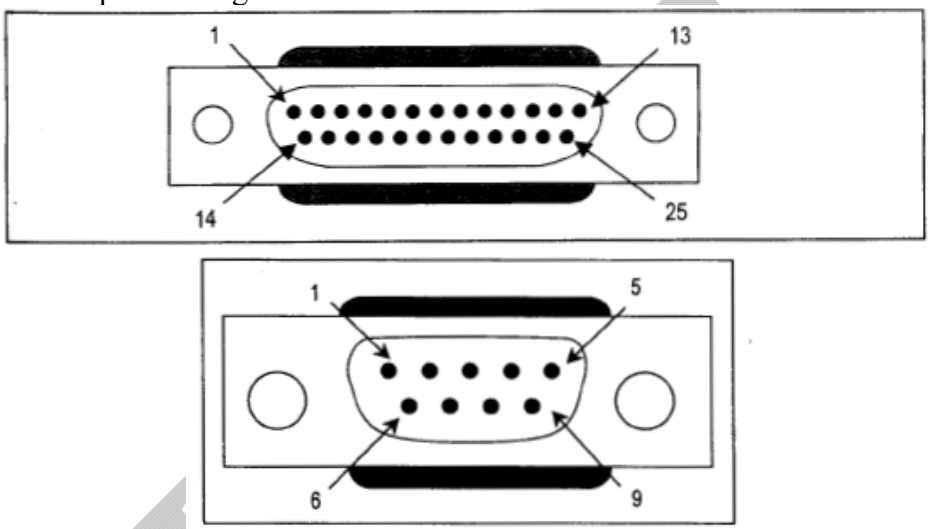
RS-232 định nghĩa mức điện thế mà tương ứng với các mức logic 0 và 1. Tín hiệu có mức điện thế như sau : mức logic 0: từ +3V đến +25V, mức logic 1 : từ -25V đến -3V.



Hình 5-17. Mức điện thế biểu diễn tín hiệu nhị phân

Về bố trí chân của RS232

RS232 có 2 phiên bản giắc đầu nối là DB-25 và DB-9

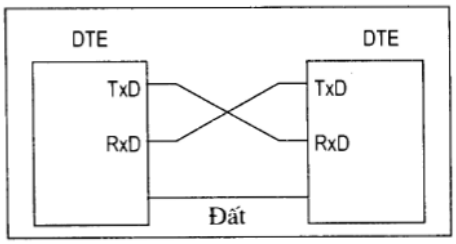


Hình 5-18. DB-25 và DB-9

Hiện tại DB-25 ít được sử dụng nên ta chỉ xem xét cách bố trí chân đấu nối vủa DB-9 thông quan bảng dưới:

Chân	Mô tả	Ý nghĩa
1	Data carrier detect(DCD)	Tách tín hiệu mang dữ liệu
2	Received Data(RxD)	Dữ liệu nhận được
3	Transmitted Data(TxD)	Dữ liệu được gửi
4	Data terminal ready(DTR)	Đầu cuối dữ liệu sẵn sàng
5	Signal Ground(GND)	Đất của tín hiệu
6	Data set ready(DSR)	Dữ liệu sẵn sàng
7	Request to send(RTS)	Yêu cầu gửi
8	Clear to send(CTS)	Xóa để gửi
9	Ring indicator(RL)	Báo chuông

Một kết nối đơn giản nhất giữa PC và bộ vi điều khiển yêu cầu tối thiểu 3 chân TxD, RxD, GND như trong hình vẽ dưới đây :



Hình 5-19. Kết nối tối thiểu giữa PC với vi điều khiển

Định dạng bản tin:

Trong truyền bất đồng bộ, đường liên kết không bao gồm một đường clock, vì trong mỗi đầu cuối thiết bị đều có một clock. Vì vậy hai clock phải có chung tần số,cho phép sai khác nhỏ. Mỗi một byte được truyền gồm một bit Start để đồng bộ các clock, và một hoặc nhiều bit Stop để báo hiệu kết thúc một từ truyền. Một đường truyền bất đồng bộ sử dụng nhiều định dạng, nhưng định dạng phổ biến nhất là 8-N-1 ở đó bộ truyền sẽ gửi mỗi byte dữ liệu với 1 bit Start, theo sau 8 bít dữ liêu, bắt đầu với bít 0(LSB) và kết thúc với 1 bít Stop như mô tả hình dưới

Hình 5-20. Định dạng 8-N-1

Mô tả các chân UART:

Pin	Type	Mô tả
RXD0/2/3	Input	Serial Input : dữ liệu nhận nối tiếp
TXD0/2/3	Output	Serial Output: đữ liệu truyền nối tiếp

Mô tả các thanh ghi:

Thanh ghi	Mô tả		Kiểu truy cập
RBR	Receiver Buffer Register : chứa byte đầu trong bộ đệm nhận FIFO của cổng UART cũng là byte được nhận trước nhất. Bit DLAB trong thanh ghi LCR phải là 0 để có thể truy cập RBR.		RO
THR	Transmit Holding Register : là byte đầu của bộ đệm truyền FIFO của cổng UART cũng là kí tự mới nhất trong bộ đệm này. Bít DLAB trong thanh ghi LCR phải là 0 để truy cập THR. Byte này sẽ được truyền khi nó tiến đến cuối của bộ đệm		WO(WriteOnly)
DLL	Divisor Latch LSB	Kết hợp với nhau chúng chứa giá trị hệ số chia để chia tín hiệu PCLK nhằm tạo ra tốc độ	R/W

DLM	Divisor Latch MSB	baude phù hợp. Giá trị chia là giá trị 16 bit với DLL là byte cao và DLM là byte thấp. Chúng chỉ được truy cập khi bít DLAB bằng 1	
IER	Interrupt Enable Register : được sử dụng để cho phép hoặc cấm các chức năng ngắt trong UART . Chỉ được truy cập khi bit DLAB bằng 0		R/W
IIR	Interrupt Identification Register : thanh ghi nhận diện nguồn ngắt chứa trạng thái các ngắt đang yêu cầu xử lý liên quan đến chức năng UART		RO
FCR	FIFO Control Register : điều khiển hoạt động của các bộ đệm FIFO nhận và truyền như : cho phép bộ đệm hoạt động, reset bộ đệm..		R/W
LCR	Line Control Register : dùng để xác định định dạng của kí tự sẽ được truyền hoặc nhận như : độ dài từ, số lượng bit stop, tùy chọn kiểm tra chẵn lẻ..		R/W
LSR	Line Status Register : chứa thông tin về trạng thái trên các khối TX và RX của cổng UART như đã nhận dữ liệu chưa, lỗi,…		RO

Và còn 1 số thanh ghi nữa tuy nhiên sẽ không đề cập chi tiết trong tài liệu này, tham khảo trong datasheet nhà sản xuất.

Tính toán tốc độ baud:

$$Baudrate = \frac{PCLK}{16 \times (256 \times DLM + DLL)}$$

Chương trình ví dụ:

```
#include <LPC23xx.H>    /* LPC2300 definitions          */

 void init_serial (void)  {
      /* Initialize Serial Interface       */

  U0FDR    = 0;         /* Fractional divider not used       */
  U0LCR    = 0x83;      /* 8 bits, no Parity, 1 Stop bit     */
  U0DLL    = 94;        /* 9600 Baud Rate @ 14.4 MHZ PCLK    */
  U0DLM    = 0;         /* High divisor latch = 0            */
  U0LCR    = 0x03;      /* DLAB = 0                          */
}

/* Implementation of putchar     */
int sendchar (int ch)  {       /* Write character to Serial Port    */
  if(ch=='\n') ch = 0x0d;      /*new-line character*/
  while (!(U0LSR & 0x20));
  return (U0THR = ch);
}


int getkey (void)  {          /* Read character from Serial Port   */
```

```
  while (!(U0LSR & 0x01));
  return (U0RBR);

}

int sendstring(char * s) {    /* write a string */
      while(*s!='\0')
      {
             sendchar(*s);
             s++;
      }
 }

int main()
 {
      init_serial();
      while(1)
      {
            sendstring("hello \n");
      }
 }
```

5.3.8 Giao tiếp I2C

I2C, viết tắt của Inter-Intergrated Circuit , là một loại bus nối tiếp được phát triển bởi hãng sản xuất linh kiện điện tử Philips. Ban đầu, loại bus này chỉ được dùng trong các linh kiện điện tử của Philips. Sau đó, do tính ưu việt và đơn giản của nó, I2C đã được chuẩn hóa và được dùng rộng rãi trong các module truyền thông nối tiếp trên các vi mạch tích hợp ngày nay.

I2C sử dụng chỉ 2 đường 2 chiều là đường dữ liệu – SDA (Serial Data) và đường cung cấp xung nhịp – SCL(Serial Clock) , nên không yêu cầu khối logic chọn và phân quyền bus, việc thiết kế phần cứng trở nên đơn giản hơn nhiều. Mọi thiết bị được gắn lên bus đều có địa chỉ duy nhất. Mỗi thiết bị này hoạt động như 1 bộ thu hoặc 1 bộ nhận hoặc cả thu và nhận. Ví dụ như 1 LCD thì chỉ nhận, trong khi 1 bộ nhớ thì thực hiện cả 2 chức năng.

Sử dụng chỉ 2 đường SDA và SCL trong I2C cho phép truyền dữ liệu nối tiếp dưới dạng các byte địa chỉ 8-bit với không gian địa chỉ 7-bit cho phép hỗ trợ lên tới xấp xỉ 128 thiết bị sử dụng trên cùng 1 bus.

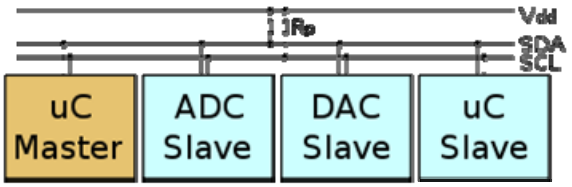
Trong một hệ thống sử dụng bus, thiết bị khởi tạo 1 quá trình trao đổi dữ liệu được gọi là master, còn thiết bị được địa chỉ hóa bởi master hay điều khiển bởi master được gọi là slave. Thông thường master điều khiển tín hiệu clock. I2C là một bus multi-master, có nghĩa là có thể có hơn 1 IC có khả năng khởi tạo 1 quá trình trao đổi dữ liệu.

Tốc độ:

Tốc độ giao tiếp của I2C có thể lên đến 3.4Mbs với nhiều tùy chọn tốc độ:

- Standard : 100 kbps
- Fast : 400 kbps
- Fast mode plus : 1 Mbps

- High speed mode : 3.4 Mbps



Hình 5-21. Ví dụ bus I2C

Hoạt động:

Khi bắt đầu 1 quá trình trao đổi dữ liệu, trước hết master khởi tạo quá trình truyền bằng cách gửi 1 start bit theo sau là địa chỉ 7-bit của slave mà nó muốn giao tiếp, tiếp đó là 1 bit đơn mô tả quá trình là ghi tới (bit 0) hay đọc (bit 1) từ slave.

Nếu slave tồn tại trên bus thì nó sẽ đáp ứng lại 1 bít ACK với địa chỉ trên. Master sau đó sẽ tiếp tục hoặc truyền hoặc gửi.

Nếu master muốn tiếp tục gửi dữ liệu đến slave thì nó tiếp tục gửi 1 byte và slave gửi lại 1 bít ACK

Nếu master muốn tiếp tục đọc từ slave thì nó tiếp tục nhận 1 byte từ slave và gửi 1 bit ACK sau mỗi byte nhận được.

Master muốn kết thúc trao đổi thì nó sẽ gửi 1 stop bit lên bus.

Hình 5-22. Lược đồ thời gian

Lược đồ trên mô tả timing của quá trình truyền. Mỗi quá trình truyền bắt đầu với 1 bít Start(S) khi SDA bị kéo xuống thấp trong khi SCL vẫn ở mức cao. Sau đó SDA bắt đầu thiết lập bít được truyền trong khi SCL mức thấp và dữ liệu được lấy mẫu (nhận) khi SCL chuyển mức. Khi quá trình truyền kết thúc, 1 bít Stop(P) được truyền bằng cách giải phóng đường SDA để nó được kéo lên mức cao trong khi SCL vẫn ở mức cao.

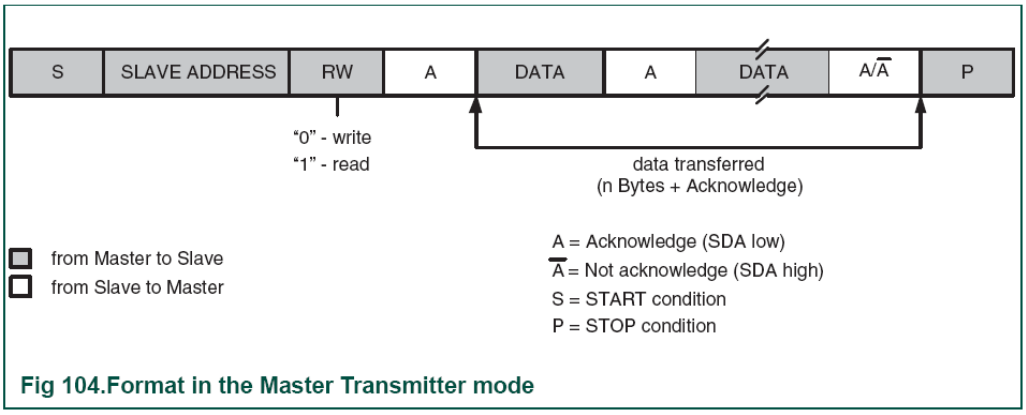
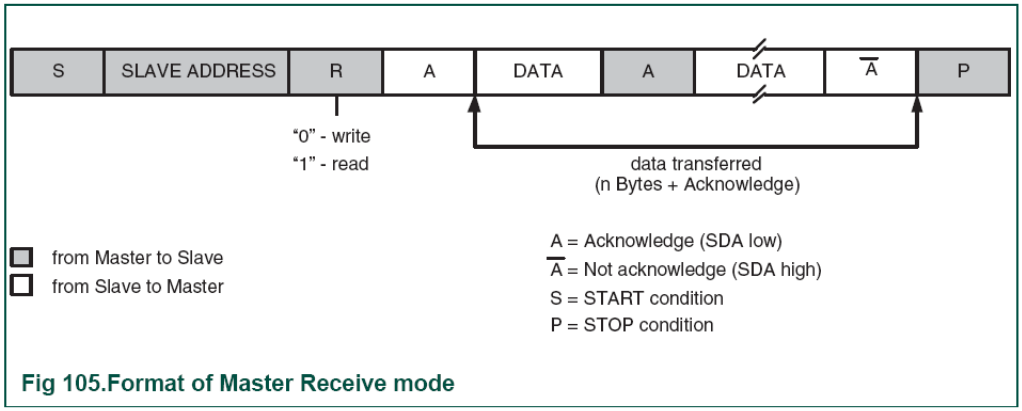


Fig 104.Format in the Master Transmitter mode



Hình 5-23. Ví dụ qua trình truyền tin

I2C trong LPC2378:

LPC2378 hỗ trợ 3 cổng giao tiếp I2C.
Mô tả các pin I2C:

Pin	Loại	Mô tả
SDA0/1/2	I/O	Đường dữ liệu SDA
SCL0/1/2	I/O	Đường tín hiệu đồng hồ SCL

Mô tả các thanh ghi I2C:

Register	Mô tả	Kiểu truy cập
I2CONSET	I2C Control Set Register : Thanh ghi thiết lập điều khiển điều khiển I2C. Khi 1 được ghi vào 1 bit của thanh ghi này thì bit tương ứng trong thanh ghi điều khiển I2C sẽ được set. Ghi 0 vào sẽ không có hiệu ứng gì	R/W
I2STAT	I2C Status Register : thanh ghi trạng thái I2C. Trong hoạt động của I2C, thanh ghi này cung cấp các mã trạng thái chi tiết cho phép phần mềm xác định đáp ứng cần thiết tiếp theo	RO
I2DAT	I2C Data Register : thanh ghi dữ liệu I2C. Trong quá trình truyền dữ liệu, dữ liệu để được truyền sẽ được ghi vào đây, còn trong quá trình đọc dữ liệu, dữ liệu đã nhận được sẽ đọc từ thanh ghi này	R/W
I2ADR	I2C Slave Address Register: thanh ghi dữ liệu slave, chứa địa chỉ 7-bit của slave khi thiết bị hoạt động ở mode slave. Biết trọng số nhỏ nhất xác định rằng slave có đáp ứng tới địa chỉ gọi chung hay không	R/W
I2SCLH	SCH Duty Cycle Register High Half Word : nửa từ cao thanh ghi chu kỳ hoạt động SCH : xác định thời gian mức cao của clock I2C	R/W

I2SCLL	SCH Duty Cycle Register Low Half Word : nửa từ thấp thanh ghi chu kỳ hoạt động SCH cùng với I2SCLH xác định tần số clock được tạo bởi 1 I2C master và thời gian phù hợp được sử dụng trong chế độ slave. Nó xác định thời gian mức thấp của clock I2C	R/W
I2CONCLR	I2C Control Clear Register : thanh ghi xóa điều khiển. Khi 1 được ghi vào một bit của thanh ghi này, bit tương ứng trong thanh ghi điều khiển sẽ được clear, ghi 0 vào sẽ không có hiệu ứng gì	R/W

Xác định tần số hoạt động của bus I2C:

Phần mềm sẽ phải thiết lập các thông số phù hợp cho I2SCLH và I2SCLL để thiết lập tần số hoạt động của bus I2C. Trong đó I2SCLH là số chu kỳ của clock ngoại vi cung cấp cho module I2C PCLK mà SCL ở mức cao còn I2SCLL là số chu kỳ của PCLK mà SCL ở mức thấp. Nên tần số clock của SCL hay của I2C là:

$$I^2C_{bitfrequency} = \frac{f_{PCLK}}{I2CSCLH + I2CSCLL}$$

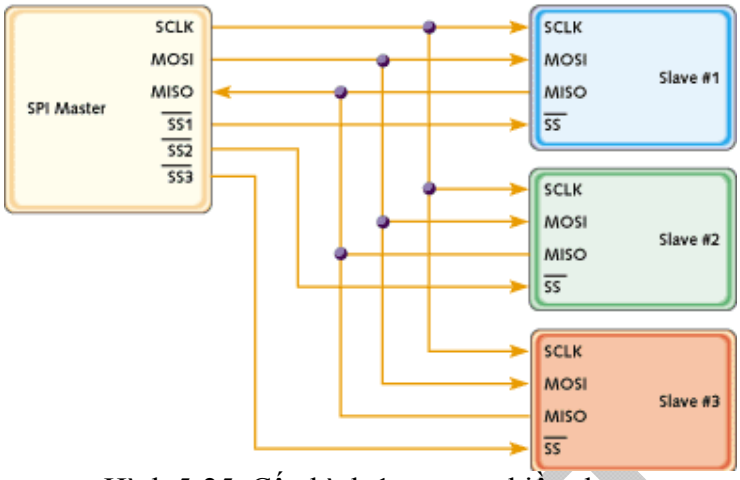
5.3.9 Giao tiếp SPI

Giao diện ngoại vi nối tiếp SPI (Serial Peripheral Interface) là một giao tiếp nối tiếp song công. Nó có thể có nhiều master và slave cùng được kết nối trên một bus xác định. Chỉ một master và một slave có thể truyền dữ liệu tại mỗi thời điểm. Trong suốt quá trình truyền dữ liệu master luôn luôn gửi từ 8 đến 16 bít dữ liệu tới slave và slave luôn gửi một byte dữ liệu tới master.

Cả SPI và I2C đều hỗ trợ tốt cho các ngoại vi tốc độ chậm được truy cập không liên tục như EEPROM và Real-Time Clock nhưng SPI có tốc độ dữ liệu cao hơn I2C, thường lên tới hàng chục Mbps, phù hợp cho các ứng dụng truyền dữ liệu theo luồng. Nó còn hiệu quả hơn nhờ giao diện song công đầy đủ , sử dụng trong các ứng dụng giữa 1 bộ codec và 1 bộ xử lý tín hiệu số.

Cũng như trong I2C, master khởi tạo quá trình trao đổi dữ liệu. Khi master tạo ra clock và chọn thiết bị slave, dữ liệu có thể được truyền theo 2 hướng đồng thời. Thực tế thì dữ liệu được truyền theo cả hai hướng, nên master và slave phải biết khi nào byte nhận có ý nghĩa hay không. Nên thiết bị phải vứt bỏ byte nhận được khi nó đang truyền và tạo ra 1 byte giả để truyền khi nó đang nhận.

Hình 5-24. Cấu hình 1 master, 1 slave



Hình 5-25. Cấu hình 1 master nhiều slave

Giao tiếp SPI gồm 4 đường tín hiệu : clock (SCLK), dữ liệu ra từ master vào slave MOSI (Master Out Slave In), dữ liệu vào master từ slave MISO (Master In Slave Out), và tín hiệu chọn slave SS (Slave Select).

Khi giao tiếp sử dụng SPI cần để ý cặp thông số cực clock CPOL (Clock Polarity) và phase clock CPHA (Clock Phase), cặp thông số này được dùng để xác định sườn của tín hiệu clock mà dữ liệu được lái và lấy mẫu. Mỗi thông số có 2 trạng thái mức cao – mức thấp nên tổng hợp lại có 4 tổ hợp có thể mà mỗi tổ hợp đều không tương thích với các tổ hợp còn lại. Nên cặp giao tiếp master/slave phải sử dụng đúng cặp thông số để giao tiếp.

Các tổ hợp CPOL và CPHA xác định:

- Khi nào bít dữ liệu đầu tiên được thiết lập
- Khi nào các bít dữ liệu khác được thiết lập
- Khi nào dữ liệu được lấy mẫu

Mối quan hệ này được biểu diễn ở bảng dưới

Tổ hợp CPOL và CPHA	Bit đầu tiên được thiết lập	Các bít khác được thiết lập	Dữ liệu được lấy mẫu
CPOL = 0, CPHA =0	Trước sườn dương SCK đầu tiên	Sườn âm SCK	Sườn dương SCK
CPOL = 0, CPHA =1	Sườn dương SCK đầu tiên	Sườn dương SCK	Sườn âm SCK
CPOL = 1, CPHA =0	Trước sườn âm SCK đầu tiên	Sườn dương SCK	Sườn âm SCK
CPOL = 1, CPHA =1	Sườn âm SCK đầu tiên	Sườn âm SCK	Sườn dương SCK

Mô tả các thanh ghi SPI:

Register	Mô tả	Kiểu truy cập
S0SPCR	SPI Control Register : thanh ghi điều khiển hoạt động SPI	R/W
S0SPSR	SPI Status Register : thanh ghi trạng thái SPI,	RO

	thông báo các trạng thái của SPI	
S0SPDR	SPI Data Register : đây là thanh ghi 2 chiều cung cấp dữ liệu truyền và nhận cho SPI. Dữ liệu truyền được cung cấp tới SPI0 được ghi vào thanh ghi này còn dữ liệu nhận được bởi SPI0 có thể đọc từ thanh ghi này	R/W
S0SPCCR	SPI Clock Counter Register : thanh ghi điều khiển tần số của master SCK0	R/W
S0SPINT	SPI Interrupt Flag : chứa cờ ngắt cho giao diện SPI	R/W

Ví dụ sau gồm các hàm cơ bản cho phép thực hiện truyền và nhận dữ liệu trên SPI. Các hàm này có thể sử dụng như là thư viện cho các ứng dụng lớn hơn.

```
#include "LPC23xx.h"

void SPIInit( void )
{

  PCONP |= (1 << 8);
  S0SPCR = 0x00;
  /* enable ports */
  PINSEL0 |= 0xC0000000;
  PINSEL1 |= 0x0000003C;
  IODIR0 =  1 << 16;
  IOSET0 =  1 << 16;

  /* Setting SPI0 clock*/
  S0SPCCR = 0x8;
  S0SPCR  = 0x00000020;
}

/*    Send a block of data to the SPI port, the first */
void SPISend( unsigned char *buf, int Length )
{
  int i;
  unsigned char Dummy;

  if ( Length == 0 )
      return;
  for ( i = 0; i < Length; i++ )
  {
      S0SPDR = *buf;

      while ( !(S0SPSR & SPIF) );
      Dummy = S0SPDR;         /* Flush the RxFIFO */
      buf++;
  }
  return;
}

/*the module will receive a block of data from SPI*/
void SPIReceive( BYTE *buf, int Length )
{
  int i;
```

```
  for ( i = 0; i < Length; i++ )
  {
      *buf = SPIReceiveByte();
      buf++;
  }
  return;
}
/*Receive one byte of data from the SPI port*/
unsigned char SPIReceiveByte( void )
{
  unsigned char data;

  /* wrtie dummy unsigned char out to generate
     clock, then read data from MISO */
  S0SPDR = 0xFF;
  /* Wait for transfer complete, SPIF bit set */
  while ( !(S0SPSR & SPIF) );
  data = S0SPDR;
  return ( data );
}
```

5.3.10 Thiết kế điều kiển giao tiếp với thiết bị tương tự: ADC, DAC

Giao tiếp với ADC

Các bộ ADC như tên gọi đã chỉ rõ là các thiết bị thực hiện chuyển đổi 1 tín hiệu liên tục thành các xung rời rạc. Cụ thể trong điện tử chúng chuyển đổi 1 đầu vào tương tự thành 1 giá trị số hóa tương đương với mức của biên độ của tín hiệu đầu vào hoặc ở dạng dòng diện hoặc ở dạng điện thế trên một dải giá trị biên độ của tín hiệu đầu vào.

Khi xét đến bộ ADC cần quan tâm đến 2 tham số là độ chính xác và tốc độ chuyển đổi.

Độ chính xác mô tả số các mức lượng tử rời rạc mà bộ ADC có thể tạo ra trên dải các giá trị tín hiệu đầu vào tương tự. Các giá trị thông thường được lưu trong các thiết bị số ở dạng nhị phân nên độ chính xác cũng được biểu diễn dưới dạng số bít. Ví dụ một ADC với độ chính xác là 8 bít có khả năng mã hóa một đầu vào tương tự sang 1 trong 256 mức khác nhau. Giá trị có thể là từ 0 đến 256 (số nguyên không dấu)hoặc từ là -128 đến 127 (số nguyên có dấu)tùy thuộc vào ứng dụng.

Tốc độ chuyển đổi là thời gian đáp ứng từ khi bộ ADC bắt đầu chuyển đổi cho đến khi bộ ADC đưa ra được mức lượng tử của đầu vào. Độ chính xác càng cao và thời gian chuyển đổi càng nhanh thì bộ ADC có chất lượng càng tốt.

Bộ ADC trong LPC2378:

Gồm có 6 kênh đầu vào được ghép kênh với độ chính xác là 10 bít, dải điện áp tính toán cho phép là từ 0 đến 3 V với tốc độ chuyển đổi >= 2.44 uS. Ngoài ra nó còn được thiết kế để cho phép điều khiển chuyển đổi thông qua xung khi có một sự dịch chuyển mức từ các pin đầu vào hoặc khi có 1 sự kiện Match trong bộ Timer.Tín hiệu clock được cung cấp bởi PCLK dành cho bộ ADC, đồng thời bên trong bộ ADC còn có bộ chia clock khả trình cho phép tạo ra tín hiệu clock phù hợp

Mô tả các pin:

Pin	Type	Mô tả
AD0[5:0]	Input	Đầu vào tương tự
VREF	Tham chiếu	Đầu vào điện áp tham chiếu : nó cung cấp các mức điện áp tham chiếu cho quá trình chuyển đổi

Mô tả các thanh ghi ADC

Tên	Mô tả	Kiểu truy cập
AD0CR	A/D Control Register. AD0CR phải được viết vào để lựa chọn chế độ hoạt động trước khi việc chuyển đổi AD có thể diễn ra	R/W
AD0GDR	A/D Global Data Register. AD0GDR chứa các kết quả của quá trình chuyển đổi gần nhất	R/W
AD0STAT	AD Status Register. AD0STAT chứa cờ DONE và cờ OVERUN cho tất cả các kênh AD, cũng như cờ ngắt của AD	RO
AD0INTEN	A/D Interrup Enable Register. Thanh ghi này chứa các bit kích hoạt mà cho phép cờ DONE của mỗi kênh A/D được bao gồm hoặc loại trừ ra từ việc đóng góp hoặc tạo ra của một ngắt A/D	R/W
AD0DR0-7	A/D Channel 0-7 Data Register. Các thanh ghi này chứa kết quả của quá trình chuyển đổi gần nhất được hoàn thành trên kênh 0-7	R/W

Ví dụ sử dụng ADC. Trong ví dụ này, chương trình sẽ liên tục đọc giá trị đầu vào ADC và lưu trong biến value. Có thể mở rộng chương trình dựa trên xử lý giá trị này để tạo các tính năng mở rộng hơn cho từng ứng dụng cụ thể.

```
#include "LPC23xx.h"

void ADC_init( void )
{
  /* Enable CLOCK into ADC controller */
    PCONP |= (1 << 12);
  /* select channel AD0.0 on P0.23 */
      PINSEL1 = (1<<14);
  /*set operation mode*/
    AD0CR = ( 0x01 << 0 ) |   /* SEL=1 */
            ( 4 << 8 ) |            /* CLKDIV = 4 */
            ( 0 << 16 ) |           /* BURST = 0 */
            ( 0 << 17 ) |           /* CLKS = 0 */
            ( 1 << 21 ) |           /* PDN = 1 */
            ( 0 << 22 ) |           /* TEST1:0 = 00 */
            ( 0 << 24 ) |           /* START = 0 */
            ( 0 << 27 );            /* EDGE = 0 */
  return;
}
```

```
void start_convert() {
      AD0CR |=(1<<24);
}

void stop_convert() {
      AD0CR &= ~(1<<24);
}

int check_done() {
      if (AD0STAT & 1 ==0) return 0;
      return 1;
}

int get_value() {
      return (AD0DR0>>6)&0x3ff;
}
/*************************
**   Main Function  main()
*************************/

int main (void)
{
      int value;
      ADC_init();
      while(1) {
            start_convert();
            while (!check_done()) {}
            value=get_value();
            stop_convert();
      }
}
```

Giao tiếp DAC

Bộ chuyển đổi số sang tương tự DAC là thiết bị thực hiện chuyển đổi tín hiệu rời rạc dạng số sang tín hiệu liên tục tương tự. Như vậy nó thực hiện nhiệm vụ ngược với bộ ADC.

Bộ DAC trong LPC2378:

- Bộ chuyển đổi số sang tương tự 10 bit
- Kiến trúc chuỗi điện trở
- Đầu ra được đệm
- Chế độ power down để tiết kiệm công suất
- Tốc độ và công suất có thể lựa chọn được

Mô tả chân:

Tên	Kiểu chân	Mô tả
AOUT	Output	Analog Output. Sau khi thời gian quá độ được lựa chọn sau khi DACR được viết một giá trị mới, điện áp trên chân này là VALUE/1024.VREF

VRFE	Reference	Voltage Reference. Chân này cung cấp điện áp tham khảo cho bộ chuyển đổi D/A
V_{DDA}, V_{SSA}	Power	Analog power and Ground. Những chân này trên danh nghĩa có cùng điện áp là V_{DD}(3V3) và V_{SS}, nhưng chúng phải được cách ly để tối thiểu nhiễu và lỗi.

Lưu ý: khi DAC không được sử dụng. V_{DD} và V_{REF} phải được kết nối tới nguồn cung cấp và chân V_{SSA} phải được kết nối với đất. Những chân này không được để thả nổi.

Mô tả thanh ghi

Thanh ghi R/W chứa giá trị số để được chuyển đổi thành analog và một bit mà trao đổi giữa hiệu năng và công suất. Các bit 5:0 được dành trước cho tương lai, cho các bộ D/A có độ phân giải cao hơn.

Bit	Ký hiệu	Giá trị	Mô tả	Giá trị Reset
5:0	-		Dành trước. Các phần mềm phải không được viết các bit 1 vào các bit này. Giá trị đọc từ các bit dành trước không có ý nghĩa	NA
15:6	VALUE		Sau khi thời gian quá độ sau khi trường này được viết một giá trị mới, điện áp trên chân Aout là VALUE/1024xVREF	0
16	BIAS	0	Thời gian quá độ của DAC tối đa là 1µs, và dòng tối đa là 700 µA	0
		1	Thời gian quá độ của DAC tối đa là 2,5µs, và dòng tối đa là 350 µA	
31:17	-		Dành trước. Các phần mềm phải không được viết các bit 1 vào các bit này. Giá trị đọc từ các bit dành trước không có ý nghĩa	NA

Chương trình ví dụ sau thực hiện tạo 1 xung tam giác ở đầu ra bộ DAC. Tín hiệu thể hiển trên speaker của bộ kit MCB2300 là một âm đơn kéo dài.

```
#include "LPC23xx.h"

void DACInit( void )
{
  /* setup the related pin to DAC output */
  /* set p0.26 to DAC output */
  PINSEL1 = 0x00200000;
  return;
}

/*************************
**   Main Function  main()
*************************/
```

```
int main (void)
{
  unsigned long i = 0;

  /* Initialize DAC  */
  DACInit();
  while ( 1 )
  {
      DACR = (i << 6) | 0x00010000;
      i++;
      if ( i == 1024 )
      {
        i = 0;
      }
  }
  return 0;
}
```

5.3.11 Vòng khóa pha

PLL thực chất là một bộ nhân tần, với đầu vào PLL là tín hiệu dao động ngoài có tần số từ 32kHz đến 50MHz từ thạch anh, tín hiệu này được nhân lên và tạo tín hiệu đầu ra có tần số từ 275MHz cho đến 550 MHz, kết hợp cùng khối chia tần trong LPC2378 sẽ cung cấp tín hiệu xung clock cho lõi ARM7 và các ngoại vi. Sử dụng PLL cho phép LPC2378 hoạt động ở tần số tối đa ngay cả với bộ dao động ngoài với tần số thấp, giảm ảnh hưởng khi có tín hiệu cao tần trong mạch.

Tần số đầu ra của PLL có thể thay đổi được trong quá trình CPU đang hoạt động, cho phép tiết kiệm công suất trong trạng thái idle.

Trong PLL có 2 tham số xác định tần số đầu ra là N và M.

Giá trị tần số đầu ra được tính như sau:

$F_{CCO} = (2 \times M \times F_{IN}) / N$

Trong đó

- F_{IN} – tần số tín hiệu đầu vào bộ PLL
- F_{CCO} – tần số tín hiệu đầu ra PLL
- N – hệ số chia của PLL, lấy từ các bit NSEL trong thanh ghi PLLCFG của PLL(có giá trị bằng trường NSEL+1, N=1÷32).
- M – hệ số nhân của PLL, lấy từ các bit MSEL trong thanh ghi PLLCFG của PLL(có giá trị bằng trường MSEL+1, chỉ có 1 số giá trị trong dải trường MSEL được phù hợp).
- *Mô tả các thanh ghi I2C:*

Register	**Mô tả**	**Kiểu truy cập**
PLLCON	PLL Control Register. Thanh ghi chứa các bit điều khiển việc cập nhật PLL. Các giá trị được ghi vào	R/W

	thanh ghi này sẽ không được áp ứng ngay tại đầu ra PLL mà phải chờ cho đến khi chuỗi giá trị trợ giúp PLL được sử dụng	
PLLSTAT	PLL Status Register. Thanhi ghi này chứa thông tin về cấu hình và điều khiển PLL. Nếu PLLCON và PLLCFG được cập nhật giá trị mới nhưng chuỗi giá trị trợ giúp PLL chưa được sử dụng thì giá trị mới này sẽ không phản ảnh trạng thái hiện tại của PLL. Đọc thanh ghi này sẽ cung cấp các giá trị thực đang điều khiển PLL cũng như trạng thái PLL	RO
PLLFEED	PLL Feed Register: thanh ghi này cho phép cập nhật các giá trị thông tin cấu hình và điều khiển PLL từ PLLCOn và PLLCFG vào các thanh ghi ẩn mà thực sự đang điều khiển hoạt động của PLL	WO

Chuỗi trợ giúp PLL phải được ghi vào thanh ghi PLLFEED để thay cho các thay đổi trong PLLCON và PLLCFG được cập nhật thực sự vào các thanh ghi đang điều khiển PLL. Chuỗi này gồm:

(1) Ghi giá trị 0xAA vào PLLFEED

(2) Ghi giá trị 0x55 và PLLFEED

Ví dụ dưới giải thích cách cấu hình PLL cho PLL

```
void init_PLL(void)
{
      // Set multiplier and divider of PLL
      PLLCFG = 0x00000024;

      // Enable the PLL
      PLLCON = 0x00000001;

      // Update PLL registers with feed sequence
      PLLFEED = 0x000000AA;
      PLLFEED = 0x00000055;

      // test Lock bit
      while (!(PLLSTAT & 0x00000400));

      // Connect the PLL
      PLLCON = 0x00000003;

      // Update PLL registers
      PLLFEED = 0x000000AA;
      PLLFEED = 0x00000055;
}
```

5.4 Thiết lập hệ điều hành nhúng trên nền ARM

5.4.1 Firmware và Bootloader

Firmware thường là phần mã nguồn được chạy đầu tiên trong một hệ thống nhúng khi khởi động. Do đó, nó là một trong những thành phần quan trọng nhất. Firmware rất đa dạng, có thể chỉ là một đoạn code khởi động hoặc cả một phần mềm nhúng.

Có nhiều định nghĩa về firmware, chúng ta dùng định nghĩa sau:

Firmware là phần mềm nhúng cấp thấp cung cấp giao diện giữa phần cứng và các phần mềm ứng dụng hoặc hệ điều hành. Firmware thường được lưu trên ROM và chạy khi hệ thống nhúng được khởi động.

Đối với một hệ thống có hệ điều hành, khi khởi động lên, hệ thống cần chạy một chương trình từ ROM để nạp hệ điều hành và dữ liệu để sau đó có thể chạy trên RAM. Do đó, người ta thường dùng định nghĩa sau:

Bootloader là một ứng dụng để nạp hệ điều hành hoặc ứng dụng của một hệ thống. Bootloader thường chỉ tồn tại tới thời điểm hệ điều hành hoặc ứng dụng chạy. Bootloader thường được tích hợp cùng với firmware.

Chu trình chạy thường gặp của một hệ thống nhúng như sau

- Bước đầu tiên là thiết lập nền tảng của hệ thống (hay chuẩn bị môi trường để khởi động một hệ điều hành).
    - Bước này bao gồm đảm bảo hệ thống được khởi tạo đúng (VD như thay đổi bản đồ bộ nhớ hoặc thiết lập các giá trị thanh ghi..).
    - Sau đó, firmware sẽ xác định chính xác nhân chíp và hệ thống. Thường thì thanh ghi số 0 của bộ đồng xử lý lưu loại VXL và tên nhà sx.
    - Tiếp theo, phần mềm chuẩn đoán hệ thống sẽ xác định xem có vấn đề gì với các phần cứng cơ bản không.
    - Thiết lập giao diện sửa lỗi: khả năng sửa lỗi hỗ trợ cho việc sửa lỗi phần mềm chạy trên phần cứng. Các sự hỗ trợ này bao gồm
        - Thiết lập breakpoint trong RAM.
        - Liệt kê và sửa đổi bộ nhớ.
        - Hiển thị nội dung của các thanh ghi.
        - Disassemble nội dung bộ nhớ thành các lệnh ARM và Thumb.
    - Thông dịch dòng lệnh (Command Line Interpreter hoặc CLI) : Tính năng thông dịch dòng lệnh cho phép người sử dụng thay đổi hệ điều hành được khởi động bằng cách thay đổi cấu hình mặc định thông qua các lệnh tại command prompt. Đối với hệ điều thành nhúng, CLI được điều khiển từ một ứng dụng chạy trên máy host. Việc liên lạc giữa host và target thường qua cáp nối tiếp hoặc kết nối mạng
- Bước thứ hai là trừu tượng hóa phần cứng. Lớp trừu tượng hóa phần cứng (HAL) là một lớp phần mềm giấu chi tiết về phần cứng phía dưới bằng cách cung cấp một tập hợp giao diện lập trình đã được định sẵn trước. Khi chuyển sang một nền tảng khác, các giao diện lập trình này được giữ nguyên nhưng phần cứng phía dưới có thể thay đổi. Đối với mỗi một phần cứng cụ thể, phần mềm HAL để giao tiếp với phần cứng đó gọi là trình điều khiển thiết bị (device driver). Trình điều khiển thiết bị cung cấp giao diện lập trình ứng dụng (API) chuẩn để đọc và ghi tới một thiết bị riêng.
- Bước thứ ba là nạp một ảnh khởi động được (bootable image). File này có thể lưu ở trong ROM, network hoặc thiết bị lưu trữ khác. Đối với chip ARM, format phổ biến nhất cho image file là Executable and Linking Format (ELF).

- Bước thứ tư là từ bỏ quyền điều khiển. Đến đây khi hệ điều hành đã bắt đầu được nạp lên và chạy, firmware sẽ chuyển quyền điều khiển hệ thống cho hệ điều hành. Trong một số trường hợp đặc biệt, firmware vẫn có thể giữ quyền điều khiển hệ thống.
    - Trong hệ thống sử dụng chip ARM, trao quyền điều khiển có nghĩa là cập nhật bảng vector và thay đổi thanh ghi PC. Thay đổi bảng vector sẽ làm chúng chỉ đến các hàm điều khiển các ngắt và các ngoại lệ. Thanh ghi PC được cập nhật để chỉ đến địa chỉ ban đầu của hệ điều hành.

5.4.2 Hệ thống file (Filesystem)

Hệ thống file là cách thức để lưu trữ và tổ chức các file và dữ liệu trên máy tính.

Khác với các hệ thống lưu trữ trên máy tính hay máy chủ, các hệ thống nhúng thường sử dụng các thiết bị lưu trữ thế rắn như flash memory, flash disk. Các thiết bị này phải được thiết lập cấu hình hệ thống file hoàn chỉnh cho hệ thống.

Có nhiều loại hệ thống file khác nhau, sau đây là một số hệ thống file phổ biến cho hệ thống nhúng:

ROMFS (ROM file system)

ROMFS là hệ thống file đơn giản nhất, lưu các dữ liệu có kích thước nhỏ, chỉ đọc được. Thường các dữ liệu này phục vụ quá trình khởi tạo RAM disk.

RAMdisk

Ramdisk (còn gọi là ổ nhớ RAM ảo hoặc ổ nhớ RAM mềm) là một ổ đĩa ảo được thiết lập từ một khối RAM. Hệ thống máy tính sẽ làm việc với khối RAM này như một ổ đĩa.

Hiệu năng của RAMdisk thường cao hơn các dạng lưu trữ khác nhiều, tuy nhiên các dữ liệu lưu trên RAM disk sẽ bị mất khi mất nguồn.

CRAMFS (Compressed RAM file system)

CRAMFS là một hệ thống file tiện dụng cho các hệ thống lưu trữ thể rắn. Đây là một hệ thống file chỉ đọc ra, và có khả năng lưu dữ liệu lưu trên hệ thống có dạng nén. CRAMFS dùng thư viện zlib để nén dữ liệu.

Công cụ làm việc với hệ thống file này là `mkcramfs`.

Journaling Flash File System (JFFS và JFFS2)

JFFS là hệ thống file cổ điển cho hệ thống nhúng. JFFS hỗ trợ bộ nhớ flash NOR. Phiên bản cập nhật của JFFS là JFFS2 có thêm nhiều tính năng cải tiến, hỗ trợ bộ nhớ flash NAND. Đồng thời, JFFS2 cũng hỗ trợ nén với một trong ba thuật toán : zlib, rubin và rtime.

5.4.3 Thiết lập nhân (kernel)

Nhân của hệ điều hành là thành phần phần mềm trung tâm của hệ thống nhúng. Khả năng của nhân cũng là chỉ số của khả năng của cả hệ thống.

Kiến trúc, cấu trúc của nhân hệ điều hành nhúng tùy theo từng hệ điều hành có thể từ đơn giản đến phức tạp. Các tài liệu về kiến trúc, cấu trúc, lập trình, sử dụng nhân rất phong phú và phổ biến. Do phạm vi vượt quá nội dung liên quan, ở đây chúng ta chỉ để cập đến vấn đề chuẩn bị một nhân cho hệ điều hành nhúng. Các ví dụ được lấy là nhân Linux.

Cụ thể, chúng ta sẽ đề cập đến việc lựa chọn nhân, cấu hình, dịch và cài đặt.

Lựa chọn nhân

Hiện nay có nhiều hệ điều hành nhúng khác nhau, mỗi hệ điều hành nhúng có nhiều phiên bản khác nhau. Trên thực tế, nhiều phiên bản chỉ dành riêng cho một số bo mạch nhúng, một số phiên bản có thể hoạt động trên nhiều bo mạch nhúng, nhưng cần được sửa đổi phù hợp lại với cấu hình.

Đối với hệ điều hành Linux, các phiên bản nhân cho chip ARM được lưu ở trang web http://www.arm.linux.org.uk/developer/. Ngoài ra, các phiên bản có thể được lưu ở các trang mã nguồn khác, hoặc đi kèm theo kit phát triển.

Sau khi download hoặc copy phiên bản nhân dự định sử dụng. Ta có thể cần phải dùng các bản vá cho nhân (patch). Các bản vá này có thể sửa chữa các lỗi hoặc thay đổi một số tính năng trong nhân.

Cấu hình nhân

Thiết lập cấu hình nhân là bước đầu tiên để xây dựng nhân cho bo mạch. Có nhiều cách để thiết lập cấu hình nhân, trong quá trình thiết lập cấu hình cũng có nhiều lựa chọn khác nhau. Tuy nhiên, không phụ thuộc vào cách dùng hoặc cách lựa chọn các options, nhân sẽ tạo ra một file *.config* và tạo ra các liên kết symbolic và các file headers sẽ được dùng trong quá trình còn lại của việc xây dựng nhân.

Các lựa chọn: Có rất nhiều lựa chọn khác nhau, các lựa chọn này sẽ được dùng để thiết lập nên nhân. Tùy theo yêu cầu của bo mạch và chip mà ta lựa chọn cấu hình thích hợp. Ở đây ta chỉ liệt kê một số lựa chọn chính:

Code maturity level options

Loadable module support

General setup

Memory technology devices

Block devices

Networking options

ATA/IDE/MFM/RLL support

SCSI support

Network device support

Input core support

Character devices

Filesystems

Console drivers

Sound

Kernel hacking

Các cách thiết lập cấu hình: có 4 cách chính

make config

Sử dụng giao diện dòng lệnh (command-line interface) cho lần lượt từng lựa chọn

make oldconfig

Sử dụng cấu hình trong mộ file *.config* sẵn có và chỉ hỏi những lựa chọn chưa được thiết lập.

make menuconfig

Thiết lập cấu hình dùng giao diện thực đơn trên terminal.

make xconfig

Thiết lập cấu hình dùng giao diện X Window

Để xem thực đơn cấu hình nhân, có thể dùng dòng lệnh. VD đối với dòng chip ARM:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- menuconfig
```

Sau đó, ta có thể chọn các tùy chọn cấu hình tương ứng cho bo mạch. Nhiều tính năng và trình điều khiển thường có sẵn theo dạng các module và có thể được chọn để tích hợp trong nhân hoặc kèm theo dạng module.

Sau khi hoàn thành việc cấu hình nhân, thoát và lưu cấu hình nhân. File cấu hình sẽ được lưu lại trong một file *.config*.

Biên dịch nhân

Biên dịch nhân bao gồm ba bước chính sau: Xây dựng các file phụ thuộc, xây dựng ảnh nhân (kernel image), xây dựng các module nhân. Mỗi bước đều dùng các lệnh *make* và được giải thích dưới đây. Tuy nhiên, các bước này có thể được làm gộp bằng cách dùng một lệnh duy nhất.

Xây dựng các file phụ thuộc:

Phần lớn các files trong mã nguồn của nhân phụ thuộc vào nhiều file header. Để xây dựng nhân, file Makefiles cần biết tất cả các phụ thuộc này. Đối với mỗi thư mục trong cây thư mục nhân, một file *.depend* được tạo trong quá trình xây dựng các phụ thuộc. File này chứa danh mục các header file mà các file trong thư mục phụ thuộc vào.
Từ thư mục gốc (root directory) của mã nguồn nhân, để xây dựng các phụ thuộc của nhân dùng lệnh sau:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- clean dep
```

Ở đây ARCH (kiến trúc) là nhân ARM. CROSS_COMPILE là biên dịch chéo, dùng trong việc biên dịch trên kiến trúc khác với ARM (VD kiến trúc x86). CROSS_COMPILE sẽ được dùng để lựa chọn công cụ xây dựng kernel. Trong trường hợp chạy trên Linux, trình biên dịch là *gcc*, do đó nếu bo mạch đích sử dụng chip ARM, trình biên dịch sẽ là *arm-linux-gcc*.

Xây dựng file ảnh nhân (kernel image):

File ảnh của nhân được tạo ra bằng lệnh sau:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- zImage
```

Ở đây, file đích là zImage, có nghĩa là ảnh của nhân được nén dùng thuật toán gzip. Ngoài thuật toán này ra có thể dùng *vmlinux* để tạo một ảnh không nén.

Xây dựng các module nhân:

Sau khi ảnh của nhân đã được xây dựng xong, các module nhân có thể được xây dựng bằng dòng lệnh sau:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- modules
```

Tùy theo các tùy chọn nhân đã được lựa chọn xây dựng theo dạng module trong quá trình thiết lập cấu hình nhân (thay vì tích hợp trong nhân), thời gian cho quá trình này có thể dài ngắn khác nhau.

Sau khi ảnh của nhân và các module của nhân đã được xây dựng, ta có thể cài vào trong bo mạch đích. Trước khi cài, chung ta cần backup file cấu hình nhân và dọn sạch mã nguồn của nhân bằng câu lệnh sau:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- distclean
```

Cài đặt nhân

Bước cuối cùng là cài đặt nhân vào bo mạch đích. Đối với mỗi cấu hình nhân, ta cần copy bốn file: file ảnh nhân đã nén, file ảnh nhân không nén, bản đồ symbil nhân và file cấu hình. File ảnh nhân đã nén ở trong thư mục *arch/arm/boot/zImage*, ba file còn lại lần lượt là *vmlinux*, *System.map*, và *.config*.

Để dễ dàng nhận dạng các file, chung ta cần đặt tên file theo phiên bản mã nhân. Ví dụ nếu nhân được tạo từ mã nguồn 2.4.18-rm5, chúng ta copy các file như sau:

```
$ cp arch/arm/boot/zImage ${PRJROOT}/images/zImage-2.4.18-rmk5
$ cp System.map ${PRJROOT}/images/System.map-2.4.18-rmk5
$ cp vmlinux ${PRJROOT}/images/vmlinux-2.4.18-rmk5
$ cp .config ${PRJROOT}/images/2.4.18-rmk5.config
```

File Makefile của nhân kèm theo môt file *modules_install* cho việc cài đặt các module.Theo mặc định, các module được cài trong thư mục */lib/modules*. Bởi vì các module nhân được dùng với ảnh của nhân tương ứng, các module này nên được cài trong thư mục với tên tương tự như ảnh của nhân. Trong ví dụ trên nhân sử dụng là 2.4.18-rmk5 được cài trong thư mục *${PRJROOT}/images/modules-2.4.18-rmk5*. Toàn bộ nội dung của thư mục này sẽ được copy vào hệ thống file gốc của bo mạch đích để dùng với nhân tương ứng, do đó để cài các module ta dùng lệnh:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- \
> INSTALL_MOD_PATH=${PRJROOT}/images/modules-2.4.18-rmk5 \
> modules_install
```

Sau khi đã hoàn thành việc copy các modules, nhân sẽ thực hiện việc xây dựng sự phụ thuộc của các module cần cho các tiện ích trong quá trình chạy thực. Để làm việc này, chúng ta cần công cụ xây dựng đi kèm với gói phần mềm BusyBox. Download BusyBox về, copy vào thư mục *${PRJROOT}/sysapps* và giải nén tại đây. Từ thư mục BusyBox, copy file Perl script *scripts/depmod.pl* sang thư mục *${PREFIX}/bin.*

Ta dùng lệnh sau để xây dựng các phụ thuộc module cho bảng mạch đích:

```
$ depmod.pl \
> -k ./vmlinux -F ./System.map \
> -b ${PRJROOT}/images/modules-2.4.18-rmk5/lib/modules > \
>       ${PRJROOT}/images/modules-2.4.18-rmk5/lib/modules/2.4.18-
rmk5/modules.dep
```

Ở trên đây tùy chọn *–k* để xác định ảnh của nhân là không nén, *-F* dùng để xác định bản đồ hệ thống, *-b* dùng để xác định thư mục căn bản.

Tài liệu tham khảo

[1]. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*, Tammy Noergaard, Newnes, 2005.

[2]. *Embedded Systems Design*, Steve Heath,,Second Edition, Newnes, 2002 .

[3]. *Embedded Systems- Architecture, Programming and Design,* Raj Kamal, McGraw Hill, 2003

[4]. *Embedded Microprocessor system,2nd ed.,* Stuart R. Ball, Newnes, 2001.

[5]. *Embedded Microcomputer Systems: Real Time Interfacing,* 2nd Edition, ISBN 0534551629, Thomson 2006, by J. W. Valvano.

[6]. *Embedded System Design: A unified Hardware/Software Introduction*, Vahid/Givargis, John Wiley & Sons INC, 2002.

[7].*Co-Synthesis of Hardware and Software for Digital Embedded Systems*, G.D. Micheli, Illinois University at Urbana Champaign, 2000.

[8].*Co-Synthesis of Hardware and Software for Digital Embedded Systems*, G.D. Micheli, Illinois University at Urbana Champaign, 2000.

[9].*Embedded Linux System design and development,* TP. Raghavan , Amol Lad ,Sriram Neelakandan, Auerbach Publications, 2006.

[10]. *The Insider's Guide To The Philips ARM®7 Based Microcontrollers,* Hitek, 2008.

[11]. *Embedded Linux Primer: A Practical, Real-World Approach,* Christopher Hallinan, Prentice Hall, 2006.

[12]. *Building Embedded Linux Systems*, Karim Yaghmour, O'Reilly, 2003