CHUẨN PHÁT TRIỂN PHẦN MỀM NHÚNG

IMET & MANDEVICES EMBEDDED SOFTWARE

Phiên bản:	1.0
Sửa đổi lần cuối:	02/03/2021
Người biên soạn:	Nguyễn Bá Đạt

Lưu hành nội bộ

Phạm vi áp dụng

Tài liệu này định nghĩa phương thức chuẩn hóa để tất cả các lập trình viên tạo firmware nhúng. Mọi lập trình viên phải quen thuộc với Tiêu chuẩn, hiểu và chấp nhận các yêu cầu này. Tất cả các đối tác, nhà tư vấn và nhà thầu cũng sẽ tuân thủ Tiêu chuẩn này. Tiêu chuẩn này nhằm bổ sung cho các tiêu chuẩn khác, chẳng hạn như MISRA C.

Lý do của Tiêu chuẩn là để đảm bảo tất cả các phần mềm cơ sở do công ty phát triển đáp ứng các mức độ dễ đọc và khả năng bảo trì tối thiểu. Mã nguồn có hai chức năng quan trọng như nhau: nó phải hoạt động và phải truyền đạt rõ ràng cách thức hoạt động của nó với một lập trình viên mới hoặc với chính bản thân bạn trong tương lai. Cũng giống như một ngữ pháp và chính tả tiếng Anh tiêu chuẩn làm cho văn xuôi có thể đọc được, các quy ước mã nguồn được tiêu chuẩn hóa để dễ dàng đọc được firmware của một người.

Một phần của mỗi lần code-review là để đảm bảo các mô-đun và chức năng đã được xem xét đáp ứng các yêu cầu của Tiêu chuẩn. Mã không đáp ứng được tiêu chuẩn này sẽ bị loại.

Chúng tôi nhận ra rằng không có Tiêu chuẩn nào có thể bao gồm mọi trường hợp. Đôi khi có thể hợp lý khi có ngoại lệ đối với một hoặc nhiều yêu cầu được đưa vào tài liệu này. Mọi ngoại lệ phải đáp ứng các yêu cầu sau:

- *Có lý do rõ ràng*: Trước khi đưa ra một ngoại lệ cho Tiêu chuẩn, (các) lập trình viên sẽ trình bày rõ ràng và hiểu các lý do liên quan, đồng thời sẽ thông báo những lý do này cho người quản lý dự án. Các lý do phải liên quan đến lợi ích rõ ràng của dự án và/hoặc công ty; sở thích và phong cách lập trình riêng của lập trình viên không phải là lý do thích hợp để đưa ra ngoại lệ.
- *Phê duyệt*: Người quản lý dự án sẽ phê duyệt tất cả các trường hợp ngoại lệ được thực hiện.
- *Tài liệu*: Ghi lại ngoại lệ trong các nhận xét, để trong quá trình xem xét mã và bảo trì sau này, nhân viên kỹ thuật hiểu bản chất và lý do của ngoại lệ.

Dự án

Cấu trúc đường dẫn

Để đơn giản hóa việc sử dụng hệ thống kiểm soát phiên bản và để đối phó với sự ra đi đột xuất và ốm đau của lập trình viên, mọi lập trình viên tham gia vào mỗi dự án sẽ duy trì một cấu trúc thư mục giống hệt nhau cho mã nguồn được liên kết với dự án.

Thư mục "gốc" chung cho một dự án có dạng:

`/projects/project-name/rom_name`

Trong đó:

- "`/projects`" là gốc của tất cả firmware do Công ty phát triển. Bằng cách giữ tất cả các dự án dưới một thư mục chung kiểm soát phiên bản và sao lưu được đơn giản hóa; nó cũng làm giảm kích thước của thư mục gốc máy tính.
- "`/project-name`" là tên chính thức của dự án đang được phát triển.
- "`/rom_name`" là tên của ROM mà mã liên quan đến. Một dự án có thể liên quan đến một số bộ vi xử lý, mỗi bộ vi xử lý có ROM và mã riêng. Hoặc, một dự án đơn lẻ có thể có nhiều mã nhị phân, mỗi mã nhị phân trong số đó đi vào bộ ROM riêng của nó.

Các thư mục bắt buộc:

"`/projects/project-name/tools`" - trình biên dịch (compilers), trình liên kết (linkers), trình lắp ráp (assemblers) được sử dụng bởi dự án này. Tất cả các công cụ sẽ được đưa vào VCS để sau 5 đến 10 năm, khi cần thay đổi, các công cụ (hiện đã lỗi thời và không thể mua được) sẽ vẫn tồn tại. Không thể biên dịch lại và kiểm tra lại mã dự án mỗi khi có phiên bản mới của trình biên dịch hoặc trình hợp dịch; giải pháp thay thế duy nhất là duy trì các phiên bản cũ, mãi mãi, trong VCS.

"`/projects/project-name/rom_name/headers`" – tất cả các tệp tiêu đề, chẳng hạn như .h hoặc các tệp tiêu đề assembly, đưa hết vào đây.

“/projects/project-name/rom_name/source” – mã nguồn. Tại đây có thể được chia nhỏ hơn nữa thành các thư mục tiêu đề, C và hợp ngữ. MAKE FILE cũng được lưu trữ ở đây.

“/projects/project-name/rom_name/object” – mã đối tượng, bao gồm các đối tượng trình biên dịch/hợp dịch và các tệp nhị phân được liên kết và định vị.

“/projects/project-name/rom_name/test” – Thư mục này là thư mục duy nhất không được kiểm tra trong VCS và có bố cục thư mục con hoàn toàn phụ thuộc vào từng lập trình viên. Nó chứa công việc đang tiến hành, thường được giới hạn trong một mô-đun duy nhất. Khi mô-đun được phát hành cho VCS hoặc phần còn lại của nhóm phát triển, nhà phát triển phải dọn sạch thư mục và loại bỏ bất kỳ tệp nào bị trùng lặp trong VCS.

Version File

Mỗi dự án sẽ có một mô-đun đặc biệt cung cấp tên phiên bản firmware, ngày của phiên bản và mã linh kiện của ROM. Mô-đun này sẽ liệt kê theo thứ tự (với các thay đổi mới nhất ở đầu tệp, tất cả được thay đổi từ phiên bản này qua phiên bản khác của mã đã được phát hành.

Hãy nhớ rằng bộ phận sản xuất và bảo trì sẽ phải hỗ trợ sản phẩm trong hàng thập kỷ. Tài liệu có thể bị mất và các nhãn ROM có thể bị trôi đi. Để có thể giải quyết các vấn đề tương quan với phiên bản ROM, ngay cả khi nhãn phiên bản ROM đã bị mất từ lâu, tệp phiên bản chỉ nên tạo một bit mã – một xâu ký tự, bằng mã ASCII, chỉ thị phiên bản ROM hiện tại. Bằng cách kết xuất lại nội dung của ROM, ta có thể biết chính xác được phiên bản ROM đang sử dụng.

Example:

```
/*************************************************
* Version Module - Project SAMPLE
*
* Copyright 1997 Company
* All Rights Reserved
*
* The information contained herein is confidential
* property of Company. The use, copying, transfer or
* disclosure of such information is prohibited except
* by express written agreement with Company.
*
* 12/18/97 - Version 1.3 - ROM ID 78-130
*              Modified module AD_TO_D to fix scaling
*              algorithm; instead of y=mx, it now
*              computes y=mx+b.
* 10/29/97 - Version 1.2 - ROM ID 78-120
*              Changed modules DISPLAY_LED and READ_DIP
*              to incorporate marketing's request for a
*              diagnostics mode.
* 09/03/97 - Version 1.1 - ROM ID 78-110
*              Changed module ISR to properly handle
*              non-reentrant math problem.
* 07/12/97 - Version 1.0 - ROM ID 78-100
*              Initial release
*************************************************/
# undef VERSION
# define VERSION "Version 1.30"
```

Make file và Project file

Mọi tệp thực thi sẽ được tạo qua MAKE FILE hoặc tệp tương đương được hỗ trợ bởi chuỗi công cụ (Tool chains) đã chọn. MAKE FILE bao gồm tất cả các thông tin cần thiết để tự động xây dựng ảnh ROM (tệp thực thi). Điều này bao gồm biên dịch và lắp ráp các tệp nguồn, liên kết, định vị và bất kỳ điều gì khác phải được thực hiện để tạo ra tệp thực thi nạp vào ROM cuối cùng.

Một phiên bản thay thế của MAKE FILE có thể được cung cấp để tạo các phiên bản gỡ rối của code. Các phiên bản gỡ rối có thể bao gồm mã chuẩn đoán đặc biệt hoặc có thể có định dạng mã thực thi nhị phân hơi khác để sử dụng với công cụ gỡ rối.

Trong môi trường phát triển tích hợp (chẳng hạn Visual C++) chỉ định PROJECT FILE được lưu cùng với mã nguồn để định cấu hình cho tất cả các phụ thuộc giống MAKE.

Trong mọi trường hợp, không có bất kỳ công cụ nào được sử dụng bằng cách nhập lệnh, vì các đối số dòng lệnh luôn "tích lũy" trong quá trình của một dự án và sẽ nhanh chóng bị lãng quên khi phiên bản 1.0 được xuất xưởng.

Startup code

Hầu hết các mã ROM, đặc biệt khi sử dụng trình biên dịch C, yêu cầu mô-đun khởi động ban đầu thiết lập gói môi trường chạy của trình biên dịch và khởi tạo phần cứng nhất định trên chính bộ vi xử lý, bao gồm lựa chọn chip, trạng thái chờ,…

Startup code thường được cung cấp bởi trình biên dịch hoặc trình định vị, sau đó được các nhóm dự án sửa đổi để đáp ứng các nhu cầu cụ thể của từng dự án. Nó luôn luôn dành riêng cho trình biên dịch và định vị. Do đó, sửa đổi đầu tiên được thực hiện đối với startup code là một bình luận ban đầu mô tả số phiên bản của tất cả các công cụ (trình biên dịch, trình hợp dịch, trình liên kết và định vị) được sử dụng.

Startup code được tạo ra bởi các nhà cung cấp nổi tiếng là ít tài liệu. Để tránh tạo ra các vấn đề khó theo dõi, không bao giờ xóa một dòng mã khỏi mô-đun khởi động. Chỉ cần bình luận ra những dòng không cần thiết, cẩn thận ghi chú rằng bạn chịu trách nhiệm vô hiệu hóa những dòng cụ thể. Điều này sẽ dễ dàng kích hoạt lại mã trong tương lai.

Nhiều thiết bị ngoại vi có thể được khởi tạo trong startup code. Hãy cẩn thận khi sử dụng các công cụ tạo mã tự động do nhà cung cấp bộ vi xử lý cung cấp (ví dụ: các công cụ tự động hóa thiết lập chọn chip). Vì nhiều phần khởi động bộ vi xử lý với các lựa chọn chip RAM bị vô hiệu hóa, hãy luôn bao gồm mã trạng thái chọn chip và trạng thái đợi bằng in-line code (không phải như một chương trình con). Hãy cẩn thận khởi tạo các lựa chọn này ở đầu mô-đun, để cho phép các lệnh gọi chương trình con trong tương lai hoạt động và vì một số công cụ gỡ lỗi sẽ không hoạt động đáng tin cậy cho đến khi chúng được thiết lập.

Các vấn đề về HEAP và STACK

Luôn khởi tạo ngăn xếp trên một địa chỉ chẵn. Hãy chống lại sự cám dỗ để đặt nó thành một giá trị lẻ như 0xffff, vì trên một máy word (16 bit), một ngăn xếp lẻ sẽ làm tê liệt hiệu suất của hệ thống.

Vì rất ít lập trình viên có cách hợp lý để xác định yêu cầu dung lượng ngăn xếp tối đa, nên luôn cho rằng ước tính của bạn không chính xác. Đối với mỗi ngăn xếp trong hệ thống, hãy đảm bảo mã khởi tạo lấp đầy toàn bộ dung lượng bộ nhớ được phân bổ cho ngăn xếp với giá trị 0x55. Sau đó, khi gỡ lỗi, bạn có thể xem ngăn xếp và phát hiện tràn ngăn xếp bằng cách phát hiện không thấy khối 0x55 nào trong vùng đó. Tuy nhiên, hãy chắc chắn rằng mã lấp đầy ngăn xếp bằng 0x55 sẽ tự động phát hiện kích thước của ngăn xếp, do đó, thay đổi kích thước ngăn xếp vào ban đêm sẽ không phá hủy công cụ hữu ích này.

Các hệ thống nhúng thường không chịu được các vấn đề về heap (bộ nhớ động). Việc phân bổ động bộ nhớ giải phóng địa chỉ có thể, theo thời gian, phân mảnh heap đến mức chương trình bị treo do không thể phân bổ thêm RAM. (Các chương trình máy tính để bàn ít bị ảnh hưởng bởi điều này vì chúng thường chạy trong khoảng thời gian ngắn hơn nhiều).

Vì vậy, hãy cảnh giác với việc sử dụng hàm malloc(). Khi sử dụng một chuỗi công cụ mới, hãy kiểm tra hàm malloc, nếu có thể, để xem liệu nó có thực hiện thu gom rác để giải phóng các khối bị phân mảnh hay không (lưu ý rằng điều này có thể dẫn đến một vấn đề khác, vì trong quá trình thu gom rác, hệ thống có thể không đáp ứng với các ngắt). Đừng bao giờ mù quáng cho rằng việc phân bổ và giải phóng bộ nhớ là không tốn kém hoặc không có vấn đề.

Nếu bạn chọn sử dụng malloc(), hãy luôn kiểm tra giá trị trả về và dừng an toàn (với thông tin chuẩn đoán) nếu nó không thành công.

Khi sử dụng C, nếu có thể (tùy thuộc vào vấn đề tài nguyên và giới hạn của bộ xử lý), hãy luôn bao gồm gói MEM của Walter Bright cùng với mã, ít nhất là để gỡ lỗi.

MEM cung cấp:

- ISO/ANSI verification of allocation/reallocation functions
- Logging of all allocations and frees
- Verifications of Frees
- Detection of pointer over- and under-runs.
- Memory leak detection
- Pointer checking
- Out of memory handling

Modules

Tổng quan

Mô-đun là một tệp mã nguồn duy nhất chứa một hoặc nhiều hàm hoặc quy trình, cũng như các biến cần thiết để hỗ trợ các hàm.

Mỗi mô-đun chứa một chức năng liên quan. Ví dụ: một mô-đun chuyển đổi A/D có thể bao gồm tất cả các trình điều khiển A/D trong một tệp duy nhất. Nhóm các hàm theo cách này giúp dễ dàng tìm thấy các phần mã liên quan và cho phép đóng gói hiệu quả hơn.

Đóng gói - ẩn chi tiết hoạt động của một hàm và giữ các biến được sử dụng bởi các hàm cục bộ - là điều hoàn toàn cần thiết. Mặc dù ngôn ngữ C và hợp ngữ không hỗ trợ rõ ràng tính năng đóng gói, nhưng với việc mã hóa cẩn thận, bạn có thể nhận được tất cả những lợi ích của ý tưởng mạnh mẽ này cũng như những người sử dụng ngôn ngữ OOP.

Trong ngôn ngữ C và hợp ngữ, bạn có thể xác định tất cả các biến và RAM bên trong các mô-đun sử dụng các giá trị đó. Đóng gói dữ liệu bằng cách xác định từng biến cho phạm vi của các hàm sử dụng các biến này. Giữ chúng ở chế độ riêng tư trong hàm hoặc trong mô-đun sử dụng chúng.

Các mô-đun có xu hướng phát triển đủ lớn đến mức không thể quản lý được. Giữ kích thước mô-đun dưới 1000 dòng để đảm bảo các công cụ (trình gỡ lỗi nguồn, trình biên dịch, v.v.) không bị quá tải đến mức chúng trở nên chậm hoặc không đáng tin cậy và để tăng cường độ rõ ràng.

Templates

Để khuyến kích giao diện mô-đun đồng nhất, hãy tạo các mẫu mô-đun có tên “module_template.c” và “module_template.asm”, được lưu trữ trong thư mục nguồn, trở thành một phần của cơ sở mã do VCS duy trì. Sử dụng một trong những tệp này làm cơ sở cho tất cả các mô-đun mới. Mẫu mô-đun bao gồm biểu mẫu tiêu chuẩn hóa cho tiêu đề (khối bình luận trước tất cả code), một điểm tiêu chuẩn cho tệp includes và khai báo toàn mô-đun, nguyên mẫu hàm và macro. Các mẫu cũng bao gồm định dạng tiêu chuẩn cho các hàm.

Mẫu mô-đun cho C code:

```
/************************************************
* Module name:
*
* Copyright 1997 Company as an unpublished work.
* All Rights Reserved.
*
*  The information contained herein is confidential
* property of Company. The user, copying, transfer or
* disclosure of such information is prohibited except
* by express written agreement with Company.
*
* First written on xxxxx by xxxx.
*
* Module Description:
* (fill in a detailed description of the module's
* function here).
*
************************************************/
/*  Include section
* Add all #includes here
*
************************************************/
/*  Defines section
* Add all #defines here
*
************************************************/
/*  Function Prototype Section
* Add prototypes for all functions called by this
* module, with the exception of runtime routines.
*
************************************************/
```

Mẫu bố cục chung của các hàm:

```
/************************************************
* Function name: TYPE foo(TYPE arg1, TYPE arg2.)
*    returns           : return value description
*    arg1                     : description
*    arg2                     : description
* Created by           : author's name
* Date created         : date
* Description          : detailed description
* Notes                : restrictions, odd modes
************************************************/
```

Mẫu mô-đun cho assembly code:

```
;************************************************
; Module name:
;
; Copyright 1997 Company as an unpublished work.
; All Rights Reserved.
;
;  The information contained herein is confidential
; property of Company. The user, copying, transfer or
; disclosure of such information is prohibited except
; by express written agreement with Company.
;
; First written on xxxxx by xxxx.
;
; Module Description:
; (fill in a detailed description of the module
;  here).
;
;************************************************
;  Include section
; Add all  "includes" here
;************************************************
```

Mẫu bố cục chung của các hàm:

```
;************************************************
; Routine name          : foobar
;    returns            : return value(s) description
;    arg1                      : description of arguments
;    arg2                      : description
; Created by            : author's name
; Date created          : date
; Description           : detailed description
; Notes                 : restrictions, odd modes
;************************************************
```

Tên mô-đun

Không bao giờ bao gồm tên hoặc từ viết tắt của dự án như một phần của mỗi tên mô-đun. Sẽ tốt hơn nhiều nếu sử dụng các thư mục riêng biệt cho từng dự án.

Các dự án lớn có thể bao gồm hàng chục mô-đun; cuộn qua một danh sách thư mục để tìm kiếm một danh sách có chứa hàm main() có thể dây khó chịu và khó hiểu. Do đó, hãy lưu trữ hàm main() trong một mô-đun có tên là main.c hoặc main.asm.

Tên tệp sẽ được viết thường để tăng cường tính di động giữa các hệ thống Windows và Linux/Unix.

Phần mở rộng của file sẽ là:

C Source Code	filename.c
C header files	filename.h
Assembler files	filename.asm
Assembler include files	filename.inc
Object Code	filename.obj
Libraries	filename.lib
Shell Scripts	filename.bat
Directory Contents	README
Build rules for make	project.mak

Biến

Cách đặt tên

Bất kể ngôn ngữ nào, hãy sử dụng tên dài để xác định rõ ràng ý nghĩa của biến. Nếu công cụ của bạn không hỗ trợ tên dài, hãy mua công cụ mới.

Phân tách các từ trong biến bằng dấu gạch dưới. Không sử dụng các chữ cái viết hoa làm dấu phân cách. Hãy xem xét ***IcantReadThis*** khó đọc hơn nhiều so với ***I_can_read_this***.

Tên biến và tên hàm được xác định với các từ đầu tiên là mô tả các ý tưởng rộng và các từ sau đó thu hẹp lại các chi tiết cụ thể. Ví dụ: ***Universe_Galaxy_System_Planet***. Hãy xem xét các tên sau: ***Timer_0_Data***, ***Timer_0_Overflow*** và ***Timer_0_Capture***. Quy ước này nhanh chóng thu hẹp các biến vào các phân đoạn cụ thể của chương trình. Đừng bao giờ cho rằng một động từ phải đứng trước, như thường thấy khi đặt tên cho các hàm. ***Open_Serial_Port*** và ***Close_Serial_Port*** thực hiện công việc nhóm kém hơn nhiều so với sự thay thế tốt hơn của ***Serial_Port_Open*** và ***Serial_Port_Close***.

Từ viết tắt và chữ viết tắt không được phép sử dụng như một phần của tên biến trừ khi:

1. Được định nghĩa trong Bảng viết tắt đặc biệt được lưu trữ trong tệp Version.
2. Quy ước công nghiệp được chấp nhận như LCD, LED, DSP.

Sự rõ ràng là mục tiêu của chúng ta! Một bảng viết tắt ví dụ là:

```
/*  Abbreviation Table
* Dsply    == Display (the verb)
* Disp     == Display (our LCD display)
* Tot      == Total
* Calc     == Calculation
* Val      == Value
* Pos      == Position
*/
```

Đặc tả ANSI C hạn chế việc sử dụng các tên bắt đầu bằng dấu gạch dưới và chữ hoa hoặc dấu gạch dưới khác (_[A-Z_][0-9A-Za-z_]). Nhiều mã môi trường chạy trình biên dịch cũng bắt đầu với dấu gạch dưới hàng đầu. Để tránh nhầm lẫn, đừng bao giờ đặt tên biến hoặc hàm với dấu gạch dưới ở đầu.

Những tên sau cũng được ANSI dành riêng cho việc mở rộng trong tương lai:

E[0-9A-Z][0-9A-Za-z]*	Errno values
is[a-z][0-9A-Za-z]*	Character classification
to[a-z][0-9A-Za-z]*	Character manipulation
LC_[0-9A-Za-z_]*	Locale
SIG[_A-Z][0-9A-Za-z_]*	Signals
str[a-z][0-9A-Za-z_]*	String manipulation
mem[a-z][0-9A-Za-z_]*	Memory manipulation
wcs[a-z][0-9A-Za-z_]*	Wide character manipulation

Biến toàn cục

Tất cả các chương trình C và đặc biệt là các chương trình hợp ngữ thường có một mô-đun khổng lồ với tất cả các định nghĩa thay đổi. Mặc dù có vẻ tốt khi tổ chức các biến ở một điểm chung, nhưng điều nguy hiểm là tất cả những biến này sau đó đều có phạm vi toàn cục. Các biến toàn cục là nguyên nhân gây ra nhiều code không thể debug, các vấn đề về sự xuất hiện lại, sự nóng lên toàn cầu và chứng hói đầu ở nam giới. Hãy tránh chúng!

Mã thời gian thực đôi khi có thể yêu cầu một vài – và chỉ một vài – biến toàn cục để đảm bảo phản ứng hợp lý các sự kiện bên ngoài. Mọi biến toàn cục phải được phê duyệt bởi người quản lý dự án.

Khi các biến toàn cục được sử dụng, hãy đặt tất cả chúng vào một mô-đun duy nhất. Chúng có vấn đề đến mức tốt nhất bạn nên xác định rõ ràng tội lỗi thông qua cái tên globals.c hoặc globals.asm.

Tính di động

Tránh sử dụng “int” và “long”, vì các khai báo này khác nhau tùy theo máy. Tạo typedefs như sau:

	Signed	Unsigned
8 bit	int8_t	uint8_t
16 bit	int16_t	uint16_t
32 bit	int32_t	uint32_t
64 bit	int64_t	uint64_t

Đừng cho rằng địa chỉ của một đối tượng int cũng là địa chỉ của byte thấp nhất của nó. Điều này không đúng trên các big-endian machines.

Hàm

Bất kể ngôn ngữ nào, hãy giữ các hàm nhỏ! Kích thước lý tưởng của một hàm là nhỏ hơn một trang; trong bất kỳ trường hợp nào một hàm không nên vượt quá hai trang. Chia nhỏ những hàm lớn thành nhiều hàm con.

Ngoại lệ duy nhất của nguyên tắc này là trường hợp rất hiếm gặp khi những ràng buộc về thời gian thực (hoặc thỉnh thoảng là giới hạn về ngăn xếp) yêu cầu những chuỗi lệnh dài của in-line code. Người quản lý dự án phải phê duyệt tất cả các mã đó… nhưng trước hết hãy tìm một giải pháp thay thế có cấu trúc hơn!

Khai báo rõ ràng mọi tham số được truyền vào mỗi hàm. Ghi lại rõ ràng ý nghĩa của tham số trong các chú thích.

Xác định một nguyên mẫu cho mọi hàm được gọi, ngoại trừ những hàm trong thư viện môi trường chạy của trình biên dịch. Các nguyên mẫu cho phép trình biên dịch bắt các lỗi quá phổ biến của các loại đối số không chính xác và số lượng đối số không đúng. Chúng là bảo hiểm giá rẻ.

Nói chung, tên hàm nên tuân theo giao thức đặt tên biến.

Quy trình dịch vụ ngắt

ISRs, mặc dù thường chỉ là một tỷ lệ phần trăm nhỏ của mã, thường là các bit khó nhất của phần firmware để thiết kế và gỡ rối. ISRs cũ nát sẽ phá hủy tiến độ dự án!

Tuy nhiên, các chu trình ngắt tốt yêu cầu phần cứng được thiết kế phù hợp. Đôi khi, bạn có thể tiết kiệm một vài cổng bằng cách để thiết bị bên ngoài chuyển đổi dòng ngắt trong vài micro giây. Điều này là không thể chấp nhận được. Mọi ngắt phải được chốt cho đến khi được xác nhận, theo chu trình xác nhận ngắt của bộ xử lý (đảm bảo phần cứng sử dụng nguồn ngắt thích hợp) hoặc thông qua bắt tay giữa code và phần cứng.

Chỉ sử dụng ngắt không che được (NMI) cho các sự kiện thảm khốc, chẳng hạn như ngày tận thế hoặc mất điện vĩnh viễn. Nhiều công cụ không thể gỡ lỗi mã NMI đúng cách. Tệ hơn nữa, NMI được đảm bảo sẽ phá vỡ cấu trúc code mà không trở lại.

Nếu có thể, hãy thiết kế một vài bit I/O dự phòng trong hệ thống. Chúng rất hữu ích để đo lường hiệu suất ISR.

Giữ ISR ngắn gọn! Dài (quá nhiều dòng code) và chậm là cặp song sinh của thảm họa ISR. Hãy nhớ rằng lâu và chậm có thể gây gián đoạn; ISR năm dòng với một vòng lặp có thể gây ra nhiều vấn đề như một quy trình 500 dòng không có vòng lặp. Khi ISR phát triển quá lớn hoặc quá chậm, hãy sinh ra một tác vụ khác và thoát ra. ISR lớn là một dấu hiệu cho thấy chắc chắn cần phải bao gồm RTOS.

Ngân sách thời gian cho mỗi ISR. Trước khi viết chu trình, hãy hiểu thời lượng thời gian có sẵn để phục vụ ngắt. dựa trên tất cả các mã hóa của bạn, sau đó đo lường hiệu suất ISR thu được xem liệu bạn có đáp ứng được nhu cầu của hệ thống hay không. Vì mọi ngắt cạnh tranh về tài nguyên CPU, ISR hoạt động chậm đó cũng có lỗi cũng giống như lỗi có mã hoàn toàn bị hỏng.

Không bao giờ cấp phát hay giải phóng bộ nhớ trong ngắt ISR trừ khi bạn hiểu rõ về hoạt động của các quy trình cấp phát bộ nhớ. Việc thu gom rác hoặc hành

vi không tốt của nhiều gói môi trường chạy có thể làm cho thời gian ISR không xác định.

Trên bộ xử lý có bảng vector ngắt, hãy điền vào mọi mục nhập của bảng. Trỏ những mục nhập đó không được hệ thống sử dụng tới trình xử lý lỗi, vì vậy bạn đã có kinh nghiệm tìm kiếm các vấn đề do các vector được lập trình sai trong thiết bị ngoại vi.

Mặc dù mã không đăng nhập lại (non-reentrant code) luôn nguy hiểm trong hệ thống thời gian thực, nhưng nó thường không thể tránh khỏi trong ISR. Ví dụ, các giao diện phần cứng thường không đăng nhập lại. Đặt tất cả các mã như vậy càng gần đầu ISR càng tốt, để sau đó bạn có thể bật lại các ngắt. Hãy nhớ rằng miễn là khi tắt hết các ngắt, hệ thống không phản hồi các yêu cầu bên ngoài.

Chú thích (Comments)

Code thực hiện một thuật toán; các chú thích (comments) truyền đạt hoạt động của code cho chính bạn và những người khác. Chú thích đầy đủ cho phép bạn hiểu hoạt động của hệ thống mà không cần phải đọc code.

Viết comments bằng tiếng Anh rõ ràng. Sử dụng cấu trúc câu mà giáo viên đã cố gắng nhét vào đầu bạn trong những tiết học tiếng Anh ở trường. Tránh viết Tiểu thuyết Mỹ vĩ đại; ngắn gọn nhưng rõ ràng…nhưng phải đầy đủ.

Tránh các đoạn văn dài. Sử dụng các câu đơn giản: danh từ, động từ, tân ngữ. Sử dụng thể câu chủ động: "*Motor_Start actuates the induction relay after a 4 second pause*". Hãy đầy đủ. Comments tốt nắm bắt mọi thứ quan trọng của vấn đề trong tầm tay.

Sử dụng kiểu chữ thích hợp. Sử dụng tất cả các chữ hoa hoặc tất cả các chữ thường chỉ làm cho các chú thích khó đọc hơn và làm cho tác giả giống như một kẻ thất học.

Nhập comments trong C ở độ phân giải khối và khi cần thiết để làm rõ một dòng. Đừng cảm thấy bắt buộc phải comment từng dòng. Sẽ tự nhiên hơn nhiều khi comment các nhóm dòng code hoạt động cùng nhau để thực hiện một chức năng lớn. Tuy nhiên, đừng bao giờ cho rằng các tên biến dài tạo ra "mã tự lập tài liệu". Mã tự tạo lập tài liệu là một mâu thuẫn, vì vậy hãy thêm các comments nếu cần để làm cho hoạt động của firmware trở nên rõ ràng. Chỉ cần đọc các comments là có thể hiểu được hoạt động của hệ thống.

Giải thích ý nghĩa và chức năng của mọi loại khai báo biến. Từng biến một, giải thích về giá trị trả về, nếu có. Tên biến dài chỉ là một trợ giúp để hiểu; kèm theo tên miêu tả bằng đoạn văn miêu tả sâu sắc, ý nghĩa.

Giải thích các tham số trong quá trình định nghĩa hàm như sau:

```
type function_name(type parameter1 /* comment */
                   type parameter2 /* comment */)
```

Chú thích các khối hợp ngữ và bất kỳ dòng nào không rõ ràng. Những chú thích tồi tệ nhất là những chú thích nói rằng "move AX to BX" trên một lệnh MOV!

Từ viết tắt và chữ viết tắt không được phép trừ khi được định nghĩa trong bảng viết tắt được lưu trong tệp Version. Mặc dù DSP có thể có nghĩa là Display với bạn, nó còn có nghĩa là Digital Signal Processor đối với tôi. Sự rõ ràng là mục tiêu của chúng tôi!

Mặc dù là hữu ích khi đánh dấu các phần của chú thích bằng chuỗi dấu hoa thị, nhưng không bao giờ có các ký tự ở bên phải của một khối chú thích. Thật quá khó để duy trì khoảng cách thích hợp khi các chú thích thay đổi sau này. Nói cách khác, điều này không được phép:

```
/*********************************************
*  This comment incorrectly uses right-hand     *
* asterisks                                          *
*********************************************/
```

Mẫu đúng là:

```
/*********************************************
*  This comment does not use right-hand
* asterisks
*********************************************/
```

<u>Nghiên cứu thêm về chuẩn comment của doxygen!</u>

Các quy ước coding

Quy ước chung

Mỗi dòng code không nhiều hơn 80 ký tự.

Không sử dụng tên đường dẫn tuyệt đối khi including các header files. Sử dụng mẫu #include<module/name> để lấy các tệp header công khai từ một vị trí chuẩn.

Không bao giờ, sử dụng "những con số kỳ diệu" – magic number. Thay vào đó, trước tiên hãy hiểu số đó đến từ đâu, sau đó xác định nó trong một hằng số, rồi ghi lại sự hiểu biết của bạn về số đó trong khai báo của hằng số.

Giãn cách và căn lề

Đặt dấu cách sau mỗi từ khóa, trừ khi dấu chấm phẩy là ký tự tiếp theo, nhưng không bao giờ có dấu cách nằm giữa tên hàm và danh sách đối số.

Đặt một dấu cách sau mỗi dấu phẩy trong danh sách đối số và sau dấu chấm phẩy ngăn cách biểu thức trong câu lệnh for.

Đặt một dấu cách trước và sau mỗi toán tử nhị phân (như +, -, v.v.). Không bao giờ đặt dấu cách giữa toán tử một bậc và toán hạng của nó (ví dụ: -100)

Đặt một dấu cách trước và sau các biến thể con trỏ (dấu *, dấu &) trong khai báo. Khi sử dụng trong biểu thức, các biến thể con trỏ có dấu cách phía trước nhưng không có dấu cách phía sau.

Thụt lề mã C theo gia số của hai dấu cách. Có nghĩa là, mỗi mức thụt lề là hai, bốn, sáu,…

Luôn đặt # trong chỉ thị tiền xử lý ở cột 1.

Định dạng code C

Không bao giờ lồng các câu lệnh IF sâu hơn ba lớp; rẽ nhánh sâu làm tổ hợp lệnh nhanh chóng trở nên khó hiểu. Tốt hơn là gọi một hàm hoặc thậm chí tốt hơn là thay thế các tổ hợp IF phức tạp bằng một câu lệnh SWITCH.

Đặt dấu ngoặc nhọn sao cho dấu ngoặc nhọn mở là thứ cuối cùng trên dòng và đặt dấu ngoặc nhọn đóng trước, như sau:

```
if (result > a_to_d) {
 do a bunch of stuff
}
```

Lưu ý rằng dấu ngoặc nhọn đóng nằm trên một dòng của riêng nó, ngoại trừ khi nó được theo sau bởi một phần tiếp theo của cùng một câu lệnh, chẳng hạn như:

```
do {
  body of the loop
} while (condition);
```

Khi một câu lệnh if-else được lồng trong một câu lệnh if khác, hãy luôn đặt dấu ngoặc nhọn xung quanh if-else để làm cho phạm vi của if đầu tiên rõ ràng.

Khi tách một dòng code, hãy thụt lề dòng thứ hai như sau:

```
function (float arg1, int arg2, long arg3,
          int arg4)
```

hoặc

```
if (long_variable_name && constant_of_some_sort == 2
    && another_condition)
```

Luôn sử dụng dấu ngoặc đơn. Không bao giờ để trình biên dịch giải quyết quyền ưu tiên; khai báo rõ ràng mức độ ưu tiên thông qua dấu ngoặc đơn.

Không bao giờ sử dụng các lệnh gán bên trong câu lệnh if. Ví dụ: không viết code như sau:

```
if ((foo = (char *) malloc (sizeof *foo)) == 0)
  fatal ("virtual memory exhausted");
```

thay vào đó, viết:

```
foo = (char *) malloc (sizeof *foo);
if (foo == 0)
  fatal ("virtual memory exhausted");
```

Nếu bạn sử dụng #ifdef để chọn trong số một tập hợp các tùy chọn cấu hình, hãy thêm mệnh đề #else cuối cùng chứa chỉ thị #error để trình biên dịch sẽ tạo ra thông báo lỗi nếu không có tùy chọn nào được xác định:

```
#ifdef sun
#define USE_MOTIF
#elif hpux
#define USE_OPENLOOK
#else
#error unknown machine type
#endif
```

Định dạng code assembly

Các điểm dừng tab trong hợp ngữ như sau:

- Tab 1: cột 8
- Tab 2: cột 16
- Tab 3: cột 32

Lưu ý rằng tất cả đều là bội số của 8, đối với các editor không hỗ trợ cài đặt tab rõ ràng. Một khoảng trống lớn – 16 cột – nằm giữa các toán hạng và các chú thích.

Tự đặt nhãn trên các dòng, như sau:

```
label:
      mov       r1, r2          ; r1=pointer to I/O
```

Bắt đầu và theo dõi các khối chú thích bằng dấu chấm phẩy:

```
;
;  Comment block that shows how comments stand
; out from the code when preceded and followed by
; "blank" lines.
;
```

Không nên dàn trải chú thích giữa các dòng code, ví dụ, không viết như sau:

```
mov     r1, r2  ; Now we set r1 to the value
add     r3, [data]      ; we read back in read_ad
```

Thay vào đó, sử dụng một khối chú thích, hoặc viết một dòng không có câu lệnh, như sau:

```
mov     r1, r2  ; Now we set r1 to the value
                  ; we read back in read_ad
add     r3, [data]
```

Hãy cảnh giác với macro (chuỗi lệnh). Mặc dù hữu ích, macro có thể nhanh chóng làm xáo trộn ý nghĩa. Hãy chọn những cái tên rất có ý nghĩa cho macro.

End.