Embedded C

I. Sự giống và khác nhau giữa C thông thường với Embedded C

C là một **ngôn ngữ lập trình cấp cao.** Nó chủ yếu dành cho việc phát triển phần mềm hệ thống. Nhưng nó cũng được sử dụng để phát triển phần mềm ứng dụng rất thường xuyên. Ngôn ngữ lập trình C phổ biến, đã ảnh hưởng đến nhiều ngôn ngữ lập trình máy tính khác như C ++ và Java. Trên thực tế, C ++ được bắt đầu như một phần mở rộng của C, và cùng với Java, nó có cú pháp rất giống với C.

Nhu cầu liên tục để viết các ứng dụng nhúng bằng ngôn ngữ lập trình cấp cao (chẳng hạn như C) chủ yếu là vì hai lý do. **Thứ nhất**, **độ phức tạp của các ứng dụng nhúng ngày càng tăng** và việc **quản lý các ứng dụng sử dụng các ngôn ngữ cấp thấp** như hợp ngữ trở nên **rất khó khăn**. **Thứ hai**, bởi vì các mẫu bộ vi xử lý mới được **phát hành rất thường xuyên**, cần phải **liên tục cập nhật/điều chỉnh** các chương trình nhúng của bạn với các tập lệnh mới hơn. Tính năng tái sử dụng có trong các ngôn ngữ như C có thể cung cấp giải pháp cho cả hai vấn đề này.

Embedded C là một bước hướng tới việc **điều chỉnh ngôn ngữ lập trình C** để viết các ứng dụng nhúng hiệu quả. Embedded C là một **ngôn ngữ lập trình mở rộng** từ C cho phép các lập trình viên có tất cả các tính năng hữu ích của một ngôn ngữ lập trình cấp cao, đồng thời có khả năng giao tiếp trực tiếp với các bộ xử lý nhúng để cải thiện hiệu suất. *Trong những năm qua, nhiều lập trình viên C độc lập đã thêm các phần mở rộng để hỗ trợ truy cập phần cứng I / O cơ bản. Embedded C là một nỗ lực để kết hợp các thực hành đó và cung cấp một cú pháp thống nhất duy nhất.*

Tóm lại, C là một ngôn ngữ lập trình cấp cao có mục đích chung được sử dụng rộng rãi chủ yếu dành cho lập trình hệ thống. Embedded C là một phần mở rộng của ngôn ngữ lập trình C cung cấp hỗ trợ phát triển các chương trình hiệu quả cho các thiết bị nhúng. C nhúng không phải là một phần của ngôn ngữ C. C thường dành cho lập trình máy tính để bàn, trong khi Embedded C thích hợp hơn cho lập trình nhúng. Không giống như C, Embedded C cho phép các lập trình viên nói chuyện trực tiếp với bộ xử lý mục tiêu và do đó cung cấp hiệu suất được cải thiện so với C. C tạo ra các tệp thực thi phụ thuộc vào hệ điều hành, trong khi Embedded C ngừng các tệp thường được tải trực tiếp vào bộ vi điều khiển. Không giống như C, Embedded C có các kiểu điểm cố định, nhiều vùng nhớ và ánh xạ thanh ghi I / O.

II. Lưu ý khi lập trình Embedded C

Đặc điểm đối với hệ thống nhúng:

- ROM và RAM hạn chế.
- Lập trình phụ thuộc phần cứng.
- Cần đáp ứng chính xác về thời gian (hàm xử lý ngắt, tác vụ...)
- Nhiều kiểu pointer (far/rom/ui/paged/...)
- Một số keywords và token đặc biệt (@, interrupt, tiny,..)

Để phát triển tốt phần mềm nhúng bằng ngôn ngữ C cần nắm vững:

- Thiết kế kiến trúc phần mềm hợp lý.
- Thành thạo sử dụng các tool và debugging
- Data types native support
- Các thư viện chuẩn.
- Phân biệt rõ về simple code với efficient code.

Một số điểm có thể tạo ra "sự khác biệt":

- Inline assembly
- Hàm xử lý ngắt.
- Assembly language generation
- Thư viện chuẩn
- Startup code
- Sử dụng các từ khóa near và far để tăng hiệu suất của biến khi biến nằm ở các vùng nhớ gần hoặc xa vùng đang sử dụng

III. C memory management

https://www.geeksforgeeks.org/memory-layout-of-c-program/

IV. Compilation process, toolchain, compiler, linker.

1. Compiler (trình biên dịch):

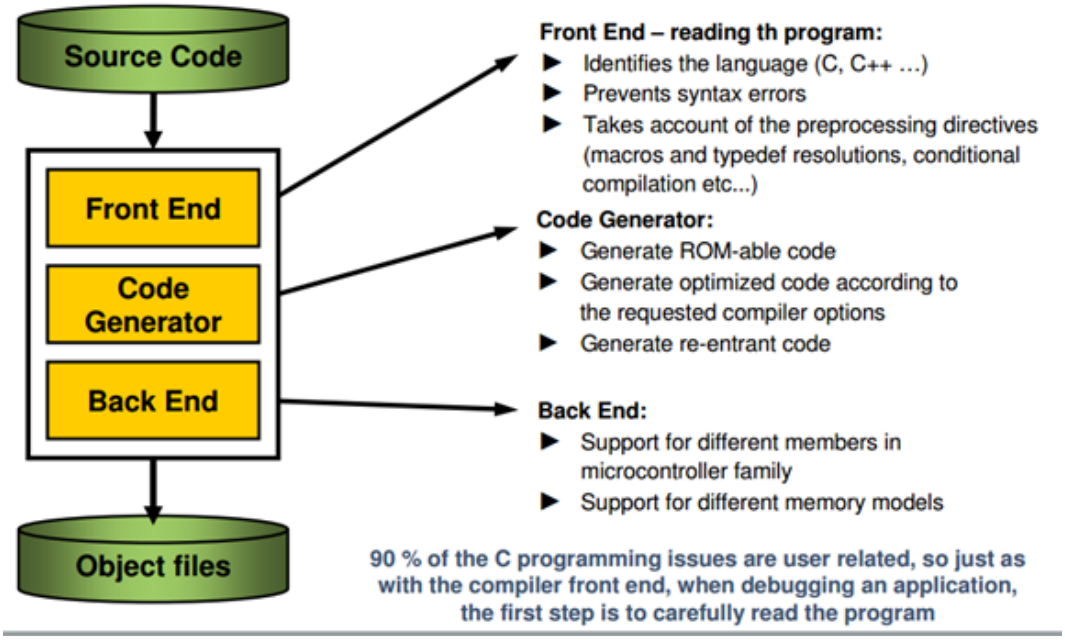
Là một chương trình máy tính làm công việc **dịch các chuỗi câu lệnh viết bằng ngôn ngữ lập trình thành chương trình tương đương** nhưng viết dưới dạng **ngôn ngữ máy tính**. Chương trình mới này được gọi là mã đối tượng (object code). Nói cách khác, compiler là thứ dùng để **chuyển ngôn ngữ bậc cao** (C) **về object code** (định

dạng mã máy có thể đọc được), để hiểu đơn giản thì compiler này nó cũng giống như phần mềm dịch tiếng anh (C) về tiếng việt (mã máy).

Ví dụ: C code (z = x+y;) → Assembly code (ADD R2,R1,R0) → Machine code (0xEB010200)

- Generate ROM-able code
- Generate Optimized code
- Generate Re-entrant code
- Support for Different Members in Microcontroller Family
- Support for Different Memory Models

2. Quá trình biên dịch (Compilation Process)



Quá trình biên dịch thông qua nhiều bước:

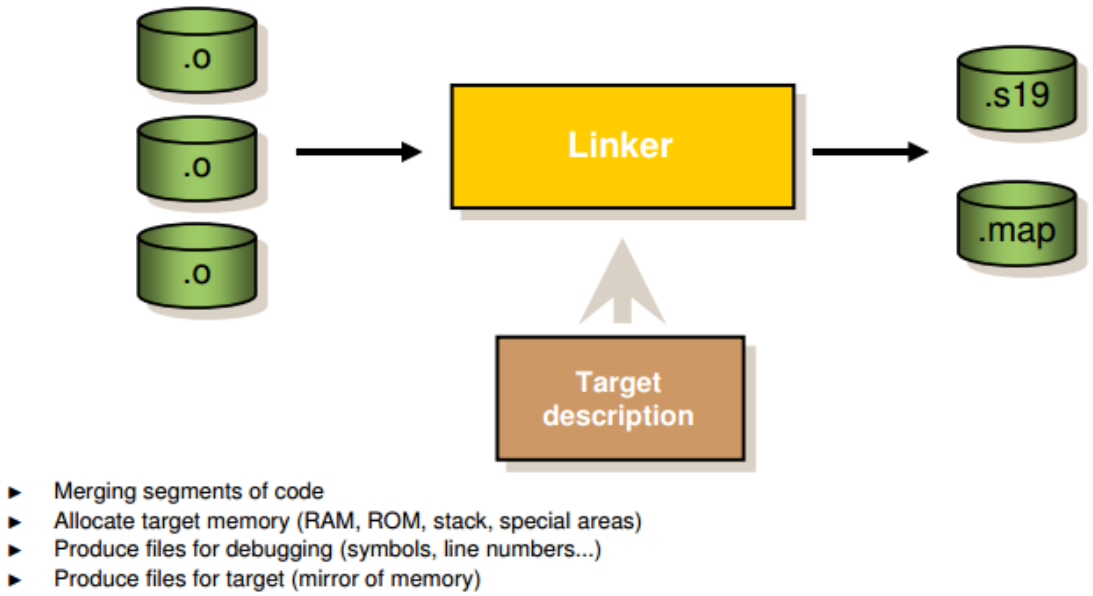
a. Front End

Trình biên dịch sẽ thực hiện phân tích từ vựng, chia các dòng mã nguồn thành từng phần nhỏ gọi là các thẻ khóa. Sau đó nó sẽ thực hiện việc phân tích cú pháp theo quy tắc để phát hiện lỗi. Tiếp theo là phân tích ý nghĩa nhằm biết được ý nghĩa của mã nguồn và chuẩn bị cho ra kết quả. Giai đoạn này sẽ thông báo tất cả các lỗi có thể gây ra trong mã nguồn.

b. Code Generator và Backend

Trong giai đoạn này, trình biên dịch sẽ thực hiện tạo ra các mã máy và thực hiện optimized code theo các tùy chọn người lập trình cài đặt cho trình biên dịch.

Linker: là công cụ kết hợp các tập tin object và tập tin nén, sắp xếp lại dữ liệu của các tập tin đó và liên kết chúng lại với nhau thành tập tin thực thi, là phần mềm để build hệ thống bằng cách kết nối(linking) các thành phần của software lại với nhau

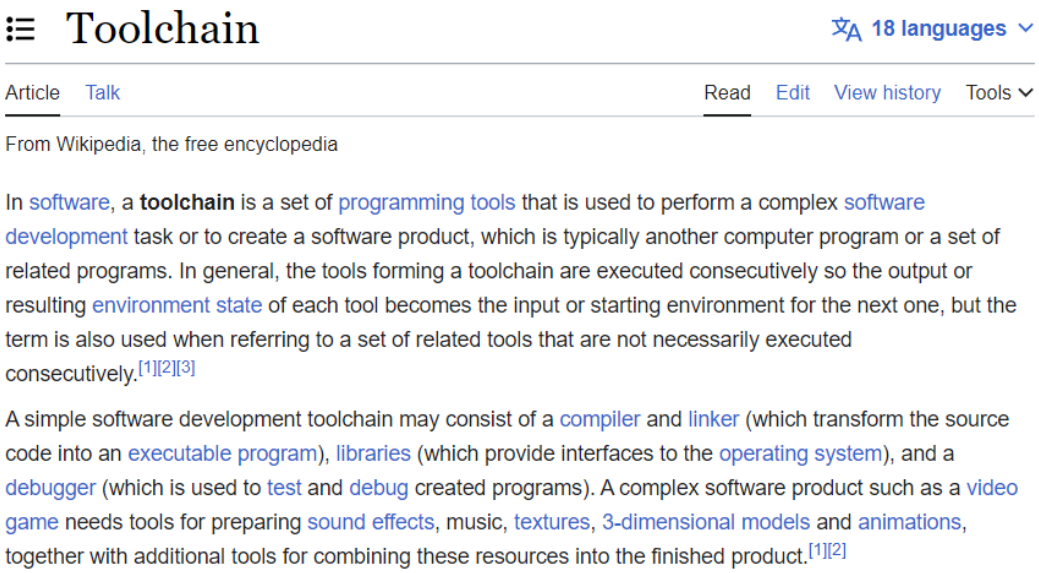


Linker còn có tạo ra bản đồ liên kết (link map) vào tập tin output chuẩn. Bản đồ này sẽ cung cấp các thông tin về sự ánh xạ của tập tin object được ánh xạ vào bộ nhớ như thế nào, giá trị cấp pháp ra sao...

Nói một cách "nôm na", Compiler sẽ tạo ra các file mã máy mà không cần biết nó sẽ được bố trí vào bộ nhớ như thế nào. Việc bố trí các file mã máy này sẽ là công việc của Linker.

3. Toolchain

Là tập hợp của các tool dùng để **biên dịch source code thành các file executable** có thể chạy được trên target device (thiết bị thật). Toolchain bao gồm compiler, linker, các thư viện runtime và một số tool khác.



Toolchain

18 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

In software, a **toolchain** is a set of programming tools that is used to perform a complex software development task or to create a software product, which is typically another computer program or a set of related programs. In general, the tools forming a toolchain are executed consecutively so the output or resulting environment state of each tool becomes the input or starting environment for the next one, but the term is also used when referring to a set of related tools that are not necessarily executed consecutively.[1][2][3]

A simple software development toolchain may consist of a compiler and linker (which transform the source code into an executable program), libraries (which provide interfaces to the operating system), and a debugger (which is used to test and debug created programs). A complex software product such as a video game needs tools for preparing sound effects, music, textures, 3-dimensional models and animations, together with additional tools for combining these resources into the finished product.[1][2]

Một toolchain về cơ bản sẽ luôn có ít nhất 3 thành phần sau

- Binutils: là tập hợp các binary tool như as (assembler), ld (linker), ar, objcopy, ...
- Compiler: C compiler và C++ compiler
- C library: là một bộ các API chuẩn dựa trên POSIX, các ứng dụng sử dụng C library để giao tiếp với kernel

Có thể chia toolchain ra làm 2 loại tùy vào mục đích: Native toolchain và Cross toolchain.

- Native toolchain: Loại toolchain chạy trên hệ thống giống với target device (device dùng để run chương trình build ra bởi toolchain). Thường là toolchain cho desktop dùng để build ra các chương trình chạy trên destop luôn.
- Cross toolchain: Loại toolchain này chạy trên hệ thống khác với target device, ví dụ dùng desktop để build ra chương trình chạy trên một thiết bị nhúng khác. Toolchain này giúp cho quá trình phát triển phần mềm trở nên nhanh hơn vì chúng ta có thể develop phần mềm trên desktop PC và load chúng lên chạy trên target device.

V. Make, build system.

Tham khảo:

https://www.quora.com/What-is-CMake-and-make-and-what-are-they-used-for-What-is-a-build-system

Giống Toolchain, Build System là tất cả **các công cụ hoặc phần mềm cần thiết để biên dịch phần mềm từ source code tới sản phẩm cuối cùng**. Tên gọi toolchain bắt nguồn từ UNIX, khi chúng ta cần một tool đơn giản cho mỗi công việc, sau đó phải mắt xích tất cả lại với nhau để hoàn thành chương trình phức tạp.

Ví dụ:

C source code file → C Compiler → C object file → Linker → Finished EXE file

Nếu biên dịch một chương trình đơn giản như "hello world", thì ta không cần Build System. Tuy nhiên, nếu biên dịch một chương trình phức tạp bao gồm nhiều tệp và thư viện, thì điều này trở nên khó khăn hơn, vì ta không chỉ cần biên dịch tất cả các tệp liên quan đến chương trình, mà thường phải làm điều đó theo một thứ tự cụ thể, bởi vì một số tệp phụ thuộc vào các tệp khác đã được biên dịch trước đó. Đây là lúc mà **Build System cho phép bạn chỉ định những tệp cần phải biên dịch, những tệp phụ thuộc vào các tệp khác và một số điều khác.**

Build files: This is a single or an entire set of automated scripts or macros. They are either in your native shell script, Autoconf, M4, or some other tool (personal choice). It contains the set of steps that the toolchain needs to complete specific tasks. These tasks can be anything the programmer needs, from building the entire software package from source code to removing unused files.

CMake và make đều là Build Systems, đây là các công cụ giúp quản lý quá trình biên dịch và liên kết các chương trình phần mềm. CMake là một **hệ thống xây dựng mã nguồn mở và đa nền tảng được sử dụng để tạo ra các tệp make hoặc các native build files** khác cho các nền tảng khác nhau, chỉ cần một trình biên dịch C. Nó không xây dựng phần mềm, mà chỉ tạo ra các tệp xây dựng cần thiết cho native toolchain

Make cũng là một build system - bạn viết một tập lệnh make script, chạy tiện ích make trên tập lệnh đó và nó sẽ tuân thủ theo tập lệnh đó để xây dựng chương trình của bạn. Make là một command-line tool được sử dụng để xây dựng các chương trình

từ source code bằng cách đọc một tệp makefile và thực hiện các lệnh được chỉ định trong đó.

Makefiles là một tập hợp các build files, được viết đặc biệt cho công cụ "make" theo kiểu UNIX, để xây dựng phần mềm từ mã nguồn. Make đã tồn tại từ rất lâu. Các phiên bản sử dụng ngày nay, thậm chí trên Windows, đến từ truyền thống UNIX.

Tóm lại, CMake là một công cụ tạo Generator Tool; nó sẽ tạo ra tệp Make và tệp Make được sử dụng bởi công cụ "make". CMake là một tầng cao hơn của Build Script. Nếu bạn nhìn vào tệp CMakeLists.txt và tệp Make, bạn sẽ thấy sự khác biệt. Thay vì viết một tập lệnh phức tạp trong tệp Make bằng cách thủ công, bạn có thể làm cho nó đơn giản hơn bằng cách sử dụng CMake.

VI. 8bit/16bit/32bit machine

Vi điều khiển giống như các máy tính nhỏ có thể thực hiện các chương trình nhỏ và thường được sử dụng cho tự động hóa và người máy. Phổ biến nhất cho những người bắt đầu là 8 bit và 16 bit microcontrollers. Sự khác biệt chính giữa 8 bit và 16 bit microcontrollers là chiều rộng của ống dữ liệu. Một vi điều khiển 8 bit có một đường dẫn dữ liệu 8 bit trong khi một vi điều khiển 16 bit có một đường dữ liệu 16 bit.

Sự khác biệt cơ bản giữa các vi điều khiển 8 bit và 16 bit được cảm nhận trong các **hoạt động toán học.** Số 16 bit cung cấp cho bạn độ chính xác hơn nhiều so với số 8 bit. Mặc dù tương đối hiếm, sử dụng một vi điều khiển 8 bit có thể không đủ độ chính xác yêu cầu của ứng dụng. Các bộ vi điều khiển 16 bit cũng hiệu quả hơn trong việc **xử lý toán học với các con số dài hơn** 8 bit. Một vi điều khiển 16 bit có thể tự động hoạt động trên hai số 16 bit, giống như định nghĩa chung của một số nguyên. Nhưng khi bạn đang sử dụng một vi điều khiển 8 bit, quá trình này không phải là đơn giản. Các chức năng được thực hiện để hoạt động trên con số như vậy sẽ **mất thêm chu kỳ**. Tùy thuộc vào mức độ xử lý ứng dụng của bạn và tính toán bao nhiêu bạn làm, điều này có thể ảnh hưởng đến hiệu suất của mạch.

Một sự khác biệt quan trọng khác giữa các bộ điều khiển 8 bit và 16 bit là trong **bộ tính giờ** của chúng. Bộ vi điều khiển 8 bit chỉ có thể sử dụng 8 bit, dẫn đến khoảng cuối cùng là 0x00 - 0xFF (0-255) mỗi chu kỳ. Ngược lại, các bộ điều khiển 16 bit, với chiều rộng dữ liệu 16 bit của nó, có một khoảng 0x0000 - 0xFFFF (0-65535) cho mỗi chu kỳ. Giá trị tối đa của bộ đếm thời gian dài hơn chắc chắn sẽ có ích trong các ứng dụng và mạch nhất định.

Ban đầu, giá của bộ vi điều khiển 16 bit là cao hơn so với các vi điều khiển 8 bit. Nhưng khi thời gian tiến triển và thiết kế cải tiến, giá của bộ vi điều khiển 8 bit và 16 bit đã giảm khá nhiều. 8 bit vi điều khiển có thể được mua bằng giá rẻ. Trong khi vi điều khiển 16 bit chi phí nhiều hơn, giá có xu hướng thay đổi rất nhiều tùy thuộc vào các tính năng được bao gồm trong vi điều khiển.

Tóm lại:

- **Bộ vi điều khiển 16 bit có hai lần dữ liệu dài hơn vi điều khiển 8 bit**
- **16 bit vi điều khiển chính xác hơn trong toán học hơn**
- **Bộ điều khiển 16 bit hiệu quả hơn vi điều khiển 8 bit trong toán học lớn hơn 8 bit**
- **16 bit vi điều khiển có timers dài hơn so với vi điều khiển 8 bit**
- **16 bit vi điều khiển được hơi đắt hơn 8 bit, microcontrollers**

8-bit Microcontroller	16-bit Microcontroller
An 8-bit microcontroller is capable of handling 8-bit data and program memory.	A 16-bit microcontroller is capable of handling 16-bit data and program memory.
An 8-bit reading bus is present on 8-bit microcontrollers.	A 16-bit reading bus is present on 16-bit microcontrollers.
8-bit microcontrollers have lower clock speeds but are more reliable.	16-bit microcontrollers offer double the clock speed but are less reliable.
8-bit microcontrollers are less efficient than 16-bit microcontrollers.	Compared to 8-bit microcontrollers, 16-bit microcontrollers are more efficient.
8-bit microcontrollers require more ROM.	16-bit microcontrollers require less ROM.
Microcontrollers with 8 bits take up less space than microcontrollers with 16 bits.	Microcontrollers with 16 bits take up more space than microcontrollers with 8 bits.
The 8-bit microcontroller is cheaper.	A 16 bit microcontroller is costly compared to an 8-bit microcontroller.
The 8-bit range for each instruction cycle is 0 to 255.	The 16- bit range for each instruction cycle is 0 to 65535.
An 8-bit microcontroller takes 20 mA of electricity to operate, which is twice as much as a 16-bit microcontroller's current consumption.	16 bit microcontroller takes 10 mA of current.
8-bit input and output peripherals are less advanced than 16 bit microcontroller input and output peripherals.	16 bit input and output peripherals are more advanced than 8-bit microcontroller input and output peripherals.
At 48 MHz, an 8-bit microcontroller shows a speed of 12 MIPS.	At 32 MHz, a 16-bit microcontroller shows a speed of 16 MIPS.
Microcontroller with 8 bits have 8-bit reading bus.	Microcontroller with 16 bits have 16-bit reading bus.

Tương tự với vi điều khiển 32bit

VII. Tổ chức firmware, kiến trúc phần mềm.

1. Embedded Firmware

Firmware nhúng (Embedded firmware) là chip bộ nhớ flash lưu trữ phần mềm chuyên biệt chạy trên một chip trong một thiết bị nhúng để điều khiển các chức năng của nó.

Phần mềm Firmware trong các hệ thống nhúng có mục tiêu tương tự như một ROM nhưng có thể được cập nhật dễ dàng hơn để thích nghi tốt hơn với các điều kiện hoặc liên kết với thiết bị bổ sung.

Các nhà sản xuất phần cứng sử dụng phần mềm Firmware nhúng để điều khiển các chức năng của các thiết bị và hệ thống phần cứng khác nhau, tương tự như hệ điều hành của máy tính điều khiển chức năng của các ứng dụng phần mềm. Khi độ phức tạp của một thiết bị tăng lên, thường có lý khi sử dụng Firmware nhúng trong trường hợp có lỗi thiết kế mà cập nhật có thể sửa chữa.

Firmware nhúng được sử dụng để điều khiển các chức năng giới hạn và cố định của các thiết bị phần cứng và hệ thống có độ phức tạp lớn hơn, nhưng vẫn cung cấp tính năng giống như các thiết bị gia dụng thay vì một loạt các lệnh dòng lệnh. Các chức năng của Firmware nhúng được kích hoạt bằng các điều khiển bên ngoài hoặc các hành động bên ngoài của phần cứng. Firmware nhúng và phần mềm nhúng dựa trên ROM thường có liên kết giao tiếp với các thiết bị khác để cung cấp tính năng hoặc để đáp ứng nhu cầu của việc điều chỉnh, hiệu chuẩn hoặc chẩn đoán thiết bị hoặc để xuất các tệp nhật ký. Cũng thông qua các kết nối này mà người khác có thể thử làm đột nhập vào thiết bị nhúng.

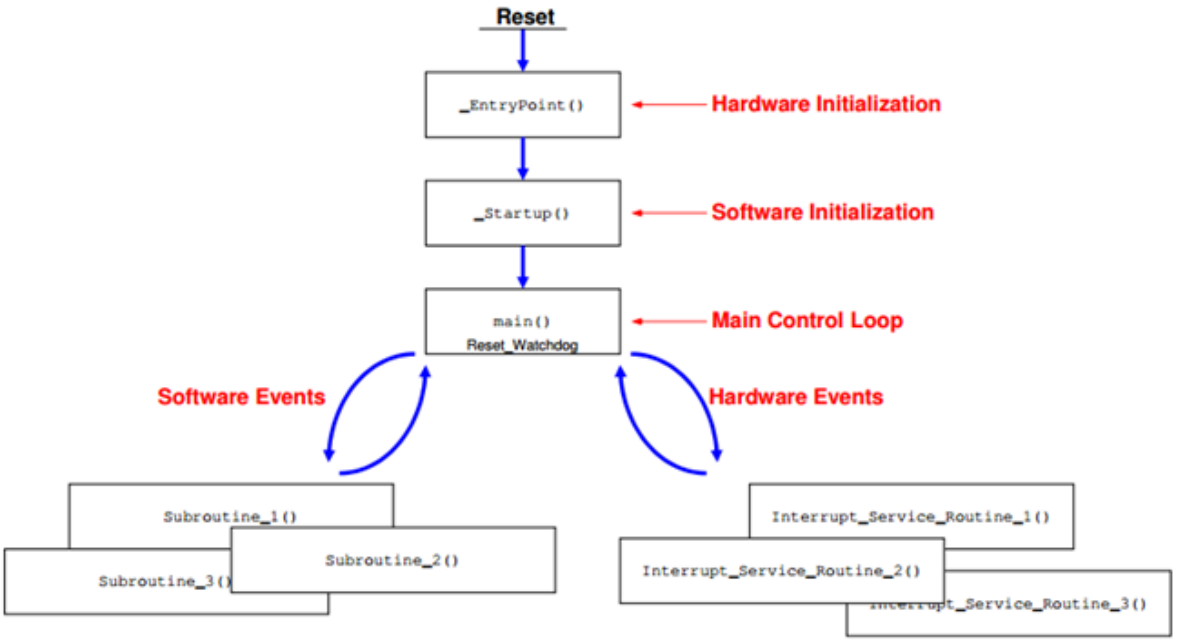
Phần mềm nhúng có sự biến đổi về độ phức tạp cũng như các thiết bị mà nó được sử dụng để điều khiển. Mặc dù phần mềm nhúng và Firmware nhúng đôi khi được sử dụng đồng nghĩa, chúng không hoàn toàn giống nhau. Ví dụ, phần mềm nhúng có thể chạy trên các chip ROM. Ngoài ra, phần mềm nhúng thường là mã máy tính duy nhất chạy trên một thiết bị phần cứng trong khi Firmware nhúng cũng có thể đề cập đến chip chứa EFI hoặc BIOS của máy tính, nó trao quyền kiểm soát cho một hệ điều hành, sau đó khởi chạy và điều khiển các chương trình.

Tổ chức firmware trong lập trình C nhúng đòi hỏi sự cẩn thận và sắp xếp logic của mã nguồn để quản lý các thành phần và chức năng của thiết bị nhúng một cách hiệu quả.

- Chia thành các lớp hoặc modules logic
- Sử dụng hệ thống tệp hợp lý
- Xây dựng một giao diện (API) tốt
- Quản lý tài nguyên
- Xử lý lỗi và ghi nhật ký
- Tích hợp tính năng cập nhật firmware
- Kiểm tra và đánh giá mã nguồn
- Tài liệu hóa mã nguồn giúp các nhà phát triển tương lai hiểu cách thiết kế và sử dụng firmware
- Thực hiện kiểm tra và gỡ lỗi

2. Kiến trúc phần mềm trong Embedded C

Các chương trình sẽ có cấu trúc cơ bản sau



Ngay sau khi reset, chương trình sẽ thực hiện khởi tạo thiết lập phần cứng (xung nhịp clock, cấu hình các chân GPIO, khai báo các ngoại vi sẽ sử dụng trong chương trình ...).

Bước tiếp theo là là cấu hình phần mềm, cài đặt chế độ cho các ngoại vi, đăng ký các hàm xử lý ngắt...

Sau khi đã xong các bước thiết lập và khởi tạo, chương trình sẽ đi vào vòng lặp vô hạn. Trong vòng lặp này sẽ có các hàm xử lý các sự kiện xảy ra đối với hệ thống. Trong vòng lặp vô hạn thường có thêm Watchdog timer để phát hiện và reset chương trình khi bị treo (chương trình quá thời gian quy định ở watchdog).

Đối với các dự án lớn, việc thiết kế kiến trúc phần mềm là rất quan trọng.

- Thứ nhất, nó phân chia chương trình thành các tầng có chức năng riêng biệt để dễ dàng quản lý bảo trì và kiểm soát lỗi.
- Thứ hai, nó tối ưu tính sửa dụng lại của chương trình khi thực hiện thay đổi nền tảng phần cứng.

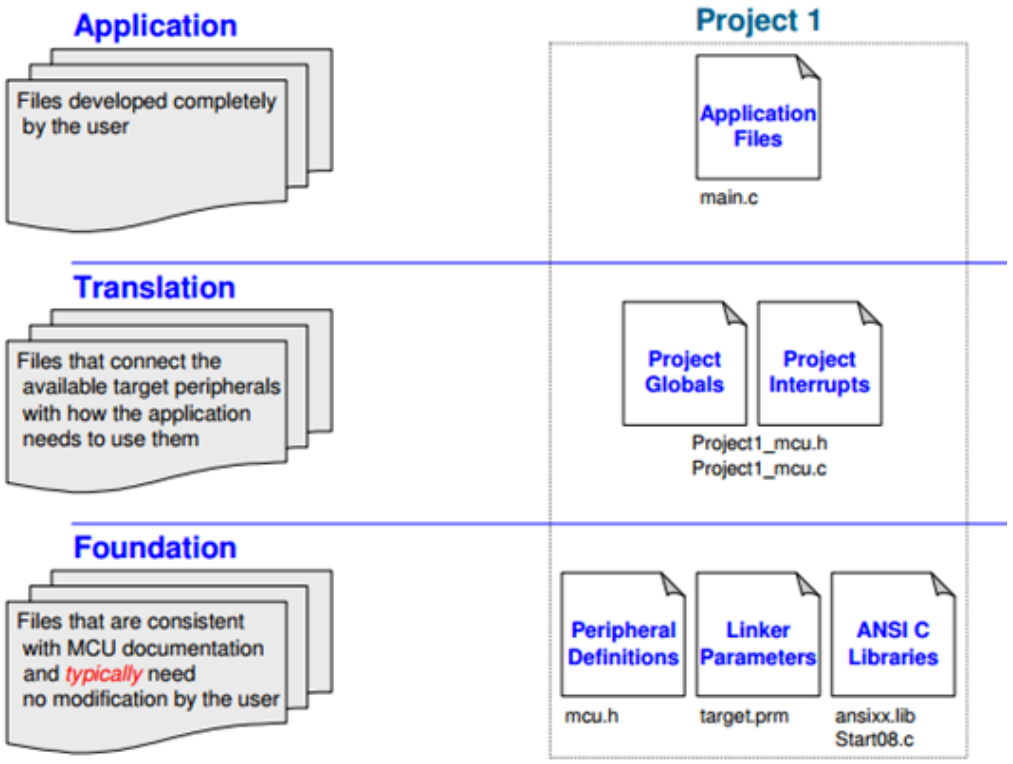
Ta lần lượt phân tích từng ý nghĩa.

Đối với việc phân chia thành các tầng, thông thường sử dụng 3 tầng bao gồm:

- Foundation: Chứa các phần source code riêng biệt cho từng nền tảng phần cứng và không thể thay đổi theo ý người phát triển. Các source code này thường được cung cấp bởi hãng cung cấp giải pháp dưới dạng các driver cho chức năng cơ bản GPIO, UART, Timer, I2C..., tóm lại nó dùng để thiết lập các chế độ hoạt động của từng nền tảng phần cứng.
- Translation: Chứa các phần source code được trừu tượng hóa, cài đặt các thuật toán nâng cao và không phụ thộc phần cứng. Đây là phần chuyển tiếp giữa tầng dưới (Foundatation) và tầng ứng dụng. Nó sử dụng các hàm từ tầng Foundatation và đưa ra các API cho tầng ứng dụng.
- Application: Chứa phần cài đặt cho từng ứng dụng cụ thể, tầng này sử dụng các API từ tầng Translation và hạn chế truy cập trực tiếp phần cứng.

Nếu cài đặt đúng theo kiến trúc như trên, khi chuyển đổi nền tảng phần cứng ta sẽ tối ưu sử dụng lại source code mà không phải viết lại từ đầu.

- Tầng Foundation do phụ thuộc hoàn toàn đối với từng nền tảng nên sẽ bị thay thế bằng nền tảng mới.



- Tầng Translation sẽ được chỉnh sửa lại cho phù hợp với nền tảng mới (thêm bớt modue so với nền tảng cũ).
- Tâng Application có thể giữ nguyên.