**CÁC KHÁI NIỆM CẦN BIẾT VÀ LINKS TRONG THẾ GIỚI LẬP TRÌNH NHÚNG (written in 2024.07.25)**

**Makefile**

<https://hocarm.org/makefile-la-gi/>

<https://b4050n.wordpress.com/2016/12/24/stm8s-3-gpio-va-bien-dich-voi-make/> ( có giải thích về quy trình biên dịch luôn )

Có trang web chuyên về dạy make file luôn.

1. Về cài đặt, công cụ cần thiết để lập trình cho MCU, cũng như giải thích ý nghĩa.

<https://b4050n.wordpress.com/2017/09/16/debug-1-gdb/>

Video setup để lập trình cho stm32f407 trên Eclipse IDE.

Bộ video của Jan Horcika. Có tên STM32 with Eclipse, STM32Cube, GNU ARM and J-Link

1. Start up code

* <https://www.youtube.com/watch?v=zFAnW7Tzu4U>

1. Các bài viết ổn để đọc:

* <https://kienltb.wordpress.com/2015/04/27/lap-trinh-c-cho-embedded-systems/>
* https://kienltb.wordpress.com/category/embedded-system/

**BOOTLOADER**

Bootloader, khái niệm rất dễ bị hiểu nhầm nên cần phải đưa nó vào một TH cụ thể nào đó để giải thích.

+ Thông thường, đối với hệ thống vi điều khiển bare-metal, thì không cần bootloader, mà nó chạy ngay code của chúng ta nạp vào.

+ Trường hợp có bootloader như Arduino, đó là chương trình nhỏ để người dùng có thể nạp code dễ dàng qua cổng USB, không cần dùng mạch nạp. Chương trình nó sẽ check ngắt USART, nếu có yêu cầu nạp code thì nó sẽ thay đổi nội dung vùng code (Arduino internal flash). Nhưng chương trinhfh bootloader ở đầu Flash vẫn còn nguyên không thay đổi.

+ Bootloader của hãng ST, cũng là một cách để chúng ta nạp code cho chip, ở đây là dòng chip của ST. Ta nhảy vào chương trình này bằng cách cấu hình BOOT pin -> sau đó nó tương tự như bên Arduino.

+ Bootloader của hệ thống có chạy hệ điều hành,

1. ví dụ là laptop Lenovo dùng chip Intel core i7. Từ khi ta bắt đầu nhấn nút nguồn, nó sẽ chạy BIOS/UEFI (là code nằm trên Flash của motherboard), sau đó chạy bootloader, bootloader sau đó sẽ chỉ tới Kernel image để hệ thống chạy hệ điều hành (Windows hoặc là Ubuntu).
2. Beaglebone Black, có bootloader là u-boot, chạy hệ điều hành Linux.

<https://www.quora.com/What-is-the-difference-between-the-BIOS-and-a-boot-loader>

thực ra hệ điều hành Angstrom (mặc định được cài đặt trên Beaglebone Black khi bạn mới mua về). Được gọi là 1 Linux-distro, nó bao gồm u-boot + Linux kernel + some other stuffs được gọi lại thành một file, bạn chỉ cần dùng tools để burn that image vào uSD card, và đưa vào con BBB chạy.

**C programming language and what you don’t know.**

Khi bạn gõ một lệnh trên Terminal, ví dụ cat /proc/cpuinfo, có nghĩa là nó đang chạy một chương trình C để thực hiện lệnh đó, và kết quả output sẽ xuất hiện lên Terminal của bạn luôn.

<https://drive.google.com/file/d/13MQLianiGSD4c0ClzOYydYo3rpYOyLe2/view?fbclid=IwAR1kInRuyDRpjk0wWxd6E-VsYhhDKSg1mbz9r5de0_Ea9i_1xKWK3Lb7H3s>

Group Cùng nhau học Linux Kernel trên FB.

C standard input and output!

Cũng như bạn gõ lệnh để chạy chương trình C bình thường vậy. Nhưng ở trường hợp trên thì là chạy và truyền arguments (file cần đọc) vào cho chương trình,...

---

How to port u-boot onto new board? Ví dụ có công ty A sản xuất board nhúng, như công ty Ampere sản xuất ra chip, sau đó cho chip đó chạy trên board nào đó. Thì để có thể chạy u-boot, ta không phải là lấy nguyên source code về rồi compile, mà để u-boot chạy được trên board nhúng của mình thiết kế, công việc cần làm là porting.

1. Chuẩn bị môi trường để biên dịch cho chip của mình (toolchain, cross-compiler,...)
2. Định nghĩa một số thứ bằng cách thay đổi một số file.
3. Cần thì phải viết lại driver cho các ngoại vi, giao tiếp của board nhúng. (như cổng UART Terminal, rồi display LED)

---

Công việc này cũng khá là giống việc mình muốn port Ardupilot firmware onto STM32F407VG board.

Đầu tiên là chỉnh sửa, build và load bootloader (Ardupilot bootloader), rồi sau đó mới load firmware qua cái giao diện Mission Planner.

**Questions.**

Tại sao u-boot là một chương trình dùng cho nhiều board, nhưng nó có thể sử dụng các ngoại vi trên board cụ thể nào đó? Cơ chế nào để nó biết rằng sẽ xuất và nhận dữ liệu qua một cổng UART cụ thể cho việc giao tiếp với người dùng?

Once again! What is linux driver development is all about?

Giải thích và cắt nghĩa câu ‘tương tác với HĐH’?

Tại sao lại cần phải viết driver, hiểu về linux kernel??

OK trong bài tập tốt nghiệp khóa embedded linux được đăng tải lên facebook, đó là viết driver làm game SNAKE trên màn hình gì đó.

Anh sản xuất ra một thiết bị, ví dụ keyboard, mouse, joystick USB, hoặc là thiết bị điện tử nào đó giao tiếp USB. Anh có một cái máy tính, nó chạy một distro của Linux, anh phải viết driver cho thiết bị của anh để khi cắm vào máy nó sử dụng được, nếu là chuột thì phải hoạt động như con chuột, còn nếu là ,...

---

Nó còn thêm một hướng đi nữa, đó là customized OS. Ví dụ như một cái máy đọc sách điện tử kindle của Amazon, ta muốn khi bật máy lên, nó tự chạy vào apps luôn, chứ không phải đầu tiên là vào hệ điều hành, click vào apps thì nó mới bật.

Người ta nói rằng các cây ATM, cái mà chúng ta thường giao tiếp là cái màn hình, màn hình đó là một máy tính chạy hđh Windows xp, windows 7.

Một thứ có thể được diễn tả bởi nhiều cách, quan trọng ngôn từ và kiến thức của người trình bày, nếu trình bày hay vấn đề có thể được diễn giải một cách

---

Đi ngược lại vs anh Phú và nhằm mục đích phổ biến cho các bạn lập trình bare-metal, hoặc chỉ sử dụng hệ điều hành nhỏ, không cỡ bự như Linux, ta có thể dùng FreeRTOS, NuttX, ChbiOS, vậy những thằng này, hệ điều hành này ai sẽ sử dụng, nó nằm ở phân khúc nào??

bare-metal nó sẽ thiên về phần cứng và phần điện hơn, như bạn muốn làm máy in 3D, máy CNC.

còn OS Linux kernel nó sẽ thiên về IT, tầng ứng dụng.

Bạn nhìn thấy con drone như phantom 3, hay con thuyền tự hành trên mặt sông? Những con chip điều khiển cho những con robot đó sẽ yêu cầu về tính real time mạnh hơn so với ứng dụng của Linux, như việc cân bằng cho con drone ở trên không trung luôn luôn ổn định thôi thì cũng là một việc cần sự xử lý tốc độ hơn với việc bạn bị đơ vài giây trên hđh , con robot bị đơ vài giây là con drone của bạn bị crash.

Hệ thống autopilot open source lớn là Ardupilot và PX4 thì sử dụng C++, hệ điều hành NuttX (sau Ardupilot chuyển sang dùng ChibiOS).

Hoặc là betaflight, sử dụng C, dùng trên các RC drones.

Và tại sao mình lại theo hướng lập trình vi điều khiển hướng theo bare-metal hơn, vì mình học bên điện, và điều khiển tự động, thích ứng dụng thuật toán điều khiển, bên Linux Kernel thì nó hướng IT và xa rời quá, có thể nếu hướng theo cái này là muốn bỏ kiến thức học trên trường luôn.

Thuật toán mình cần ứng dụng là gì? thuật toán lọc nhiễu, đọc tín hiểu cảm biến, fuse cho nó, thuật toán PID,... điều khiển nâng cao.

Tính toán tối ưu cho một ứng dụng nào đó (chip, vật dụng,...).

---

Ardupilot on Linux board.

Ta có thể sử dụng một board nhúng Linux (như Beaglebone) để chạy Ardupilot trên đó, nghĩa là nó sẽ chạy Linux kernel(Debian IoT image), và chạy code Ardupilot như một program bình thường trên hệ điều hành vậy!

<https://github.com/mirkix/ardupilotblue>

<https://discuss.ardupilot.org/t/beaglebone-blue-released-linux-enabled-autopilot-for-80/15772>

---

12.5.2019

Các công đoạn để làm ra 1 smartphone? Chip thì có sẵn rồi nhá (Snapdragon),...?

Arch Linux là gì? Hình như là một bản Linux khác, người ta chia ra rất là nhiều, wait!!

---

Giải thích về bash, shell, terminal.

Các toolchain options trong khi build, phải hiểu được,...

Keywords:

libgcc.a

-specs=nano.specs -specs=nosys.specs

atf: ARM trusted firmware.

26.5.2019

Arch Linux là gì? Hình như là một bản Linux khác, người ta chia ra rất là nhiều, wait!!

Rất là dễ nhầm lẫn, cần phải xem biểu đồ phát triển của Unix lên Linux distributions.

Hiểu và phân biệt được FreeBSD, Debian Unix, Arch Linux, MacOS... ta có thể xem qua cái project open source về ST-Link này để hiểu, cái này quan trọng vì sẽ liên quan đến việc cài đặt phần mềm lên OS sử dụng:

<https://github.com/texane/stlink>

LIBRARIES IN EMBEDDED. Language standard – 25.07.2021

Bàn về ngôn ngữ C++

<http://www.stroustrup.com/bs_faq.html>

Khái niệm library cần được hiểu rõ, và hiểu sâu khi làm việc với ngôn ngữ C/C++.

Cần phải biết được thêm về một số Topics khác, ví dụ như chuẩn ngôn ngữ đang sử dụng, (C99 mode là gì, và khi biên dịch với cái mode đó thì những gì sẽ xảy ra!),

Ngoài những cái lợi thế khi sử dụng một IDE nhất định nào đó mang lại (như tinh năng quản lý source code tích hợp vào IDE luôn, hay là tổ chức project dễ dàng, configure dễ, nhìn code dễ hơn,...), thì sẽ có một số thứ mà IDE nào cũng có,... mình nên quan tâm về cái đó hơn.

I.

Chuẩn ngôn ngữ trong Tiếng Anh là gì? -> <https://isocpp.org/std/the-standard>

Chuẩn C++: C++11, C++17 and C++14 standards.

Vậy thì làm sao biết compiler của mình sử dụng chuẩn nào? Cách chuyển đổi chuẩn ngôn ngữ

II. thư viện chuẩn (standard library).

Các thư viện như <string.h>, <stdlib.h>, nó nằm ở đâu và trình biên dịch nó link vào file compiled như thế nào?

Làm thế nào để thay đổi nó.

<http://code5s.com/tin-cong-nghe/chuan-c11.html>

<https://www.tutorialspoint.com/What-s-the-difference-between-STL-and-Cplusplus-Standard-Library>

Tận dụng sức mạnh của phần cứng máy tính mà code C++ đang chạy, như chip đa nhân,...

**STM32 - LEARNING CURVE**

I. Getting Started by doing something with microcontrollers.

+ STM8s,STM32F407VG discovery kit.

+ Basic knowledge with C programming.

+ Configure environment for learning.

a. Using sample projects and their libraries.

HAL and SPL

b. Understand common modules and communication protocols.

SPI, UART, I2C,….

c. Read how a library is written. where defined macros are. question everything.

Then try to write a project with only registers.

d. Basic debugging knowledge like: watch a variable. Using keil and STstudio.

II.

a. Setting tools and environment for development

GCC

RTOS: primitive RTOS ( mutex, semaphore, …)

+ configure environment for working.

Bit-banding and why do we need it.

Race condition

To be discovered:

<http://www.bravegnu.org/gnu-eprog/index.html>

## *CÁC FILE TRONG PROJECT STM32F4*

(2019.10.26)

Khi làm việc với project thì việc hiểu nguồn gốc của các folder, các file trong project cũng rất quan trọng. Đầu tiên vê thư viện

**CMSIS:**

Nó chứa các file căn bản như:

**1.Folder Drivers\CMSIS\Device\ST\STM32F4xx\Include**

system\_stm32f4xx.h

system\_stm32f4xx.c

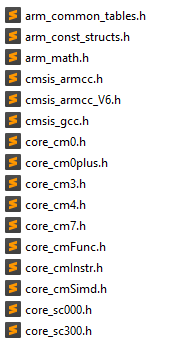
stm32f407xx.h

Drivers\CMSIS\Device\ST\STM32F4xx\Source\Templates\arm

startup\_stm32f407xx

( phụ thuộc vào Device, các file này thường được đội ngũ của ARM lẫn ST ( 1 manufacturer ) kết hợp để viết ra. Hoặc có thể do team của manufacturer viết ra hoàn toàn ).

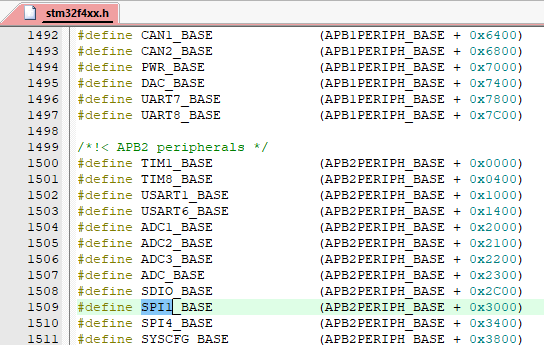
**2. Folder Drivers\CMSIS\Include**



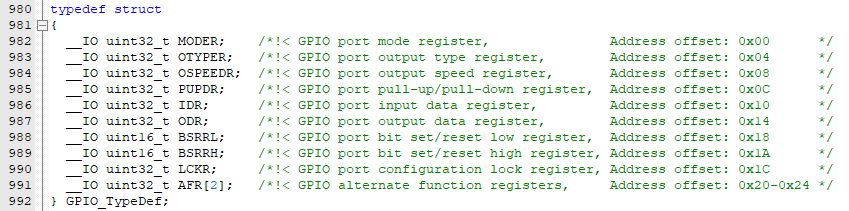
Được cty ARM viết. Chứa định nghĩa các kiểu dữ liệu sử dụng,hàm toán, để hiểu rõ hơn thì đọc.!

Stm32fxx.h:

1. Định nghĩa số của vector table
2. Định nghĩa các thanh ghi như thanh ghi GPIO, SPI …



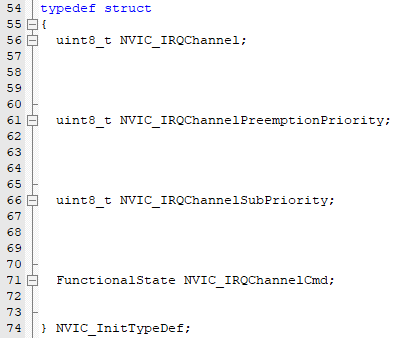
1. Định nghĩa các thanh ghi của 1 ngoại vi bằng struct mục đích để có thể truy cập qua code C một cách dễ dàng



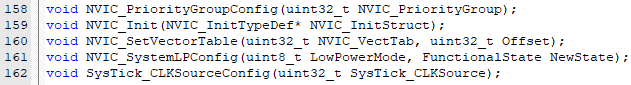
Khai báo biến kiểu struct … -> truyền vào hàm -> DONE

stdint.h : định nghĩa các kiểu dữ liệu được dung như uint8\_t được ARM Ltd viết. Sử dụng cho thư viện CMSIS hoặc là các thư viện khác.

misc.h: định nghĩa kiểu struct cho khai báo ngắt NVIC.

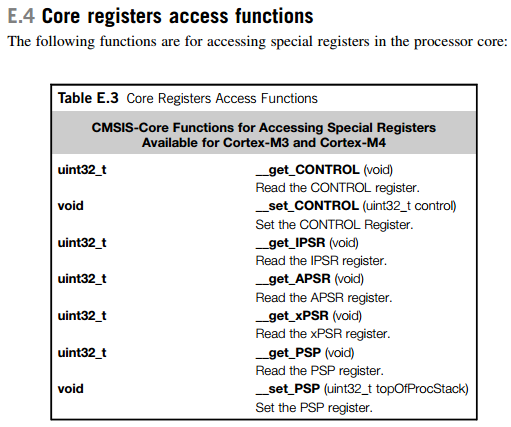


Prototype cho các hàm liên quan đến NVIC.



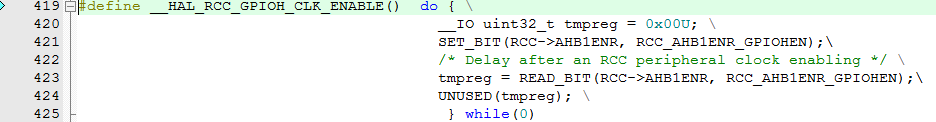
core\_cmFunc.h:

Functions to access core registers.



Enable Clock to Port H:

1. in main.c*: \_\_HAL\_RCC\_GPIOH\_CLK\_ENABLE();*

2. stm32f4xx\_hal\_rcc.h: \_\_HAL\_RCC\_GPIOH\_CLK\_ENABLE() is defined as followed:

2.1. RCC->AHB1ENR is equivalent to (\*RCC).AHB1ENR. RCC is a pointer pointing to a struct which has AHB1ENR member. ( which is defined in 2.1.1 RCC is a pointer pointing to a struct …. )

2.2.

*typedef struct*

*{*

*...*

*\_\_IO uint32\_t AHB1ENR; /\*!< RCC AHB1 peripheral clock register, Address offset: 0x30 \*/*

*...*

} RCC\_TypeDef;

2.1.1. in stm32f407xx.h:

*#define RCC ((RCC\_TypeDef \*) RCC\_BASE)*

*#define RCC\_BASE (AHB1PERIPH\_BASE + 0x3800U)*

*#define AHB1PERIPH\_BASE (PERIPH\_BASE + 0x00020000U)*

*#define PERIPH\_BASE 0x40000000U /\*!< Peripheral base address in the alias region \*/*

2.2 stm32f4xx.h: SET\_BIT and READ\_BIT are defined in here. But we just consider the SET\_BIT ( the other is the same !)

*#define SET\_BIT(REG, BIT) ((REG) |= (BIT))*

misc.h: định nghĩa các loại biến cấu trúc, thường dung để cấu hình một chức năng nào đó. Ví dụ GPIO. Hoặc NVIC dưới dạng typedef struct