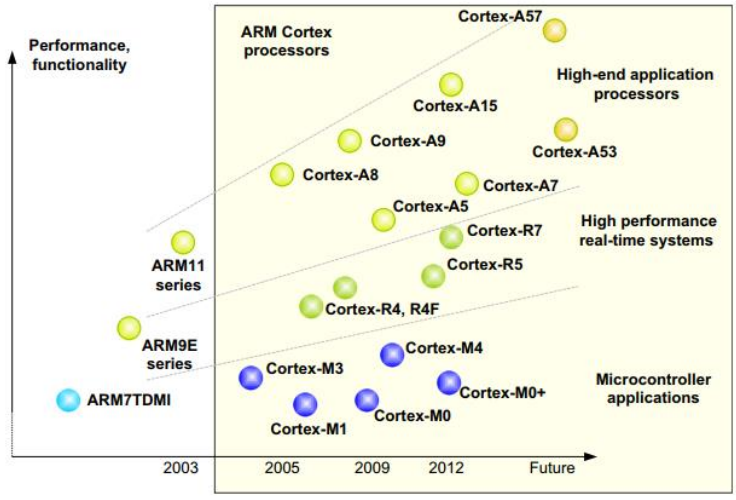
5.3 Giới thiệu về dòng vi xử lý ARM Cortex và ARM Cortex M3

Các dòng ARM Cortex

ARM Cortex là một dòng vi xử lý hoạt động với cấu trúc lệnh 32/64 bit và là một trong những lõi phổ biến trong thế giới hệ thống nhúng. Vi điều khiển ARM Cortex có thể được chia làm ba dòng chính như sau:

- ARM Cortex-A (A- **A**pplication): Là thế hệ các dòng vi xử lý được ứng dụng trong các sản phẩm yêu cầu thực hiện các tính toán phức tạp như xử lý các hệ điều hành (Linux, Android, Tizen, IOS,..). Lõi ARM Cortex-A được tìm thấy trong hầu hết các thiết bị di động hiện nay như: Smartphone, Tablet,...
- ARM Cortex-M (M- e**M**bedded): Là một lõi vi xử lý có thể mở rộng, tương thích nhiều ứng dụng, tiết kiệm năng lượng và dễ sử dụng cho các ứng dụng về thiết kế hệ thống nhúng với chi phí thấp. Lõi ARM Cortex-M đang được thiết kế tối ưu về giá, năng lượng nhằm mục đích phù hợp với các ứng dụng như IoT (Internet of Things), kết nối, điều khiển động cơ, giám sát, các thiết bị tương tác người dùng, điều khiển tự động, thiết bị y tế,... Ở phần sau của giáo trình sẽ tập trung vào hướng dẫn lập trình một dòng vi điều khiển dựa trên lõi ARM Cortex-M đó là STM32F1 (Lõi ARM Cortex-M3).
- ARM Cortex-R (**R**eal Time): Là lõi vi xử lý được theiets kế cho các ứng dụng đòi hỏi hiệu năng tính toán cao đáp ứng trong các ứng dụng hệ thống nhúng yêu cầu về thời gian thực như: Độ tin cậy cao, khả năng đáp ứng tốt, khả năng chịu lỗi, bảo trì và đáp ứng thời gian thực

ARM Cortex-M3



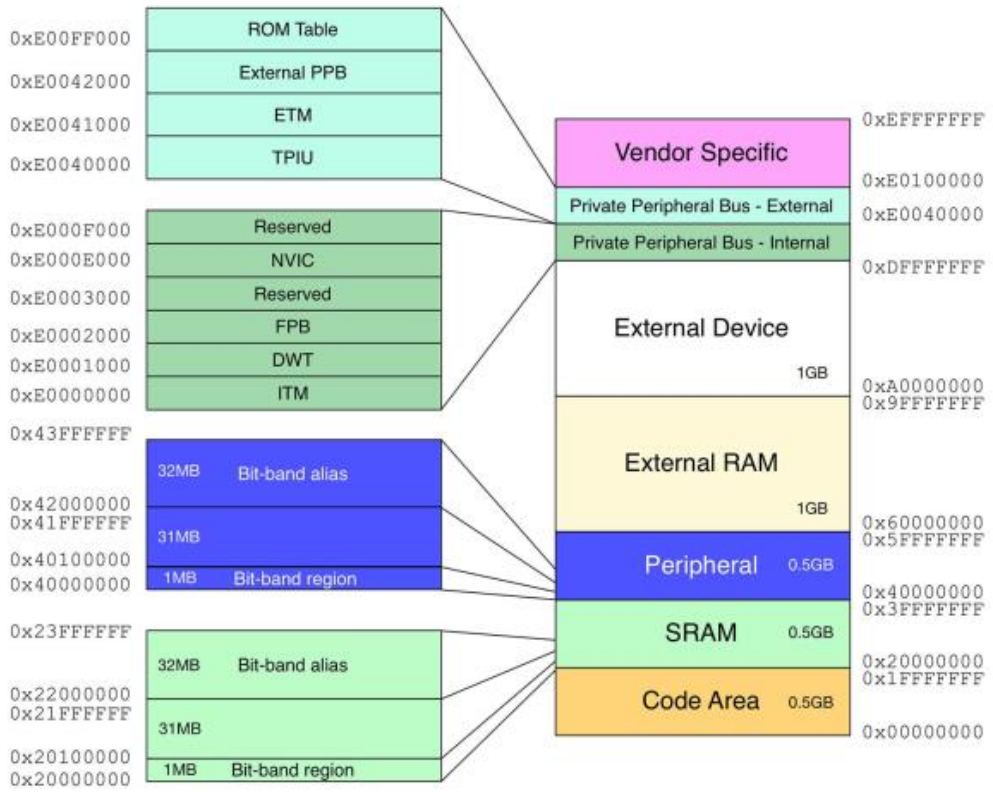
ARM Cortex M3 là một dòng vi xử lý được thiết kế bởi ARM. ARM Cortex M3 được thiết kế dựa trên nền tảng ARMv7-M. Thế hệ ARM Cortex-M3 đầu tiên được công bố

vào năm 2005 và dòng IC đầu tiên dựa trên lõi ARM Cortex-M3 được đưa ra thị trường năm 2006. Lõi ARM-M3 hoạt động trên kiến trúc 32 bit và hỗ trợ chế độ hoạt động của tập lệnh Thumb và Thumb-2.

ARM Cortex M3 có một số đặc tính sau:

- Cơ chế đường lệnh (pipeline) 3 giai đoạn.
- Kiến trúc Havard: Bộ nhớ và dữ liệu sử dụng chung không gian địa chỉ nhớ.
- 32-bit đánh địa chỉ: Hỗ trợ tối đa 4GB bộ nhớ.
- Bus nội được thiết kế dựa trên ARM AMBA (Advanced Microcontroller Bus Architecture) Technology.
- Hỗ trợ hoạt động của nhiều hệ điều hành: System tick timer, Ngăn xếp ẩn.
- Hỗ trợ hoạt động ở chế độ ngủ để tiết kiệm năng lượng.
- Hỗ trợ chế độ hoạt động bảo vệ bộ nhớ bởi MPU (Memory Protection Unit).
- Hỗ trợ truy vấn đến từng bit theo dạng bit-band.

ARM Cotex-M3 và một số dòng vi điều khiển khác như M0, M0+, M4 đang được sử dụng rộng rãi trên thị trường các vi điều khiển thông dụng.



Hình 5. 5: Sơ đồ bộ nhớ của ARM M3

Vi điều khiển STM32F1

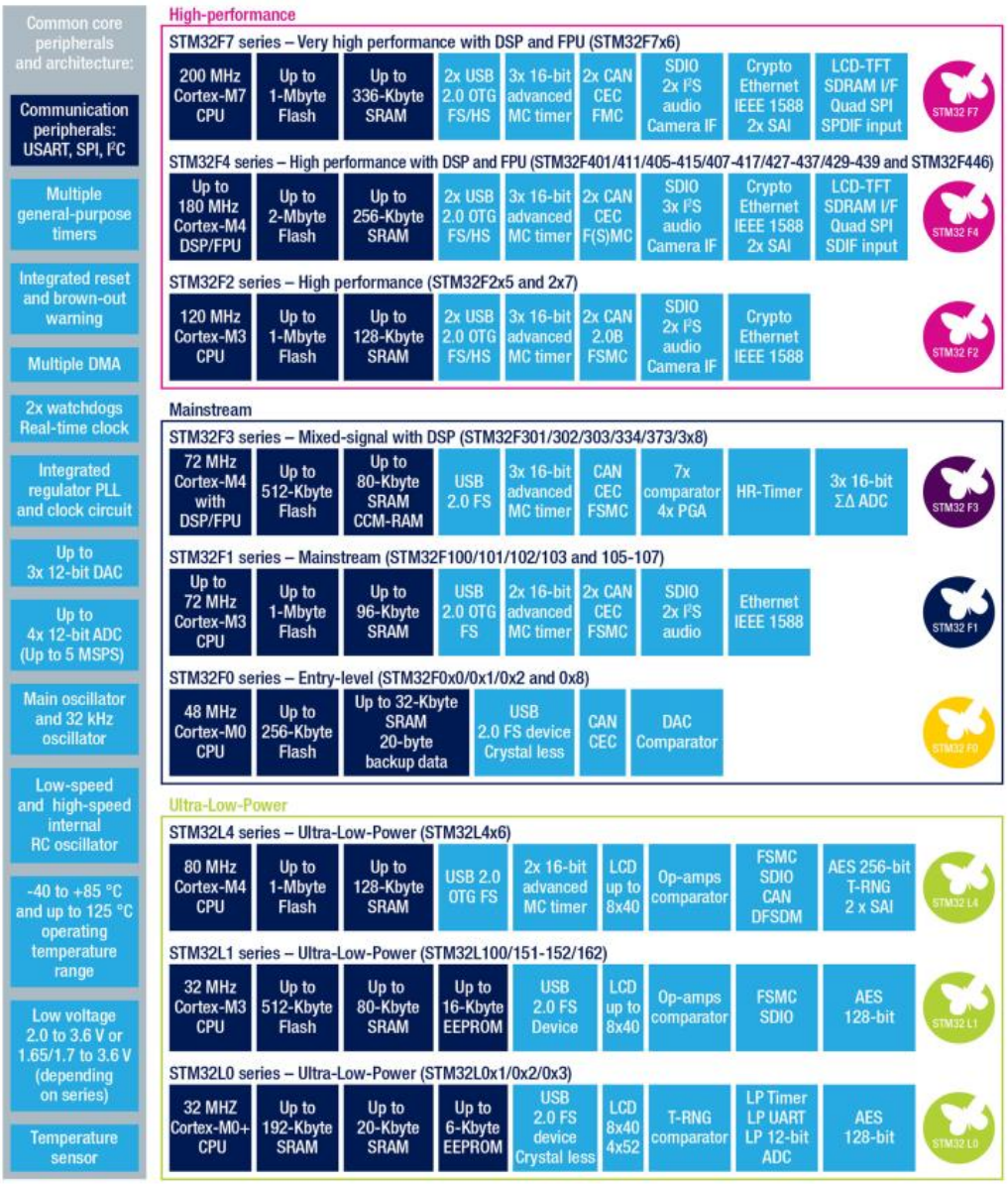
STM32 là một trong những dòng chip phổ biến của ST với nhiều họ thông dụng như F0,F1,F2,F3,F4….. STM32 có đang được thiết kế dựa trên 3 dòng sản phẩm chính như sau: Hiệu năng cao (*High-performance*), Phổ thông (*Mainstream*) và Tiết kiệm năng lượng (*Ultra Low-Power*).

Hiệu năng cao (*High-performance*): là các dòng vi điều khiển có hiệu năng tính toán cao, tốc độ hoạt động lớn và thường được sử dụng trong các ứng dụng đa phương tiện. Dòng này bao gồm các dòng vi điều khiển Cortex-M3/4F/7 với tần số hoạt động từ khoảng 120MHz đến 268MHz. Các dòng này điều hỗ trợ công nghệ ART Accelerator.

Phổ thông (*Mainstream*): Là dòng vi điều khiển được thiết kế với mục tiêu tối ưu về giá. Các sản phẩm này thường có giảm thấp hơn 1$/pcs với tần số hoạt động trong khoảng từ 48MHz đến 72MHz.

Tiết kiệm năng lượng (*Ultra Low-Power*): Là nhóm các vi điều khiển hoạt động với tiêu chí tiết kiệm năng lượng tiêu thụ. Các nhóm vi điều khiển này đều có khả năng hoạt động với pin trong thời gian dài và hỗ trợ nhiều chế độ ngủ khác nhau. Với các sản phẩm của hãng STMicrochips, điển hình của nhóm này là dòng ARM Cortex M0+.

Dưới đây là bảng mô tả hiệu năng, chức năng, hoạt động của các dòng vi điều khiển trên:



Hình 5. 6: hiệu năng, chức năng, hoạt động của các dòng vi điều khiển STM32 Cortex M

STM32F103 thuộc họ F1 với lõi là ARM Cortex M3. STM32F103 là vi điều khiển 32 bit, tốc độ tối đa là 72Mhz. Giá thành cũng khá rẻ so với các loại vi điều khiển có chức năng tương tự. Mạch nạp cũng như công cụ lập trình khá đa dạng và dễ sử dụng.

Một số ứng dụng chính: dùng cho driver để điều khiển ứng dụng, điều khiển ứng dụng thông thường, thiết bị cầm tay và thuốc, máy tính và thiết bị ngoại vi chơi game, GPS cơ bản, các ứng dụng trong công nghiệp, thiết bị lập trình PLC, biến tần, máy in, máy quét, hệ thống cảnh báo, thiết bị liên lạc nội bộ…

Phần mềm lập trình: có khá nhiều trình biên dịch cho STM32 như IAR Embedded Workbench, Keil C…

Thư viện lập trình: có nhiều loại thư viện lập trình cho STM32 như: STM32snippets, STM32Cube LL, STM32Cube HAL, Standard Peripheral Libraries, Mbed core. Mỗi thư viện đều có ưu và khuyết điểm riêng, ở đây mình xin phép sử dụng Standard Peripheral Libraries vì nó ra đời khá lâu và khá thông dụng, hỗ trợ nhiều ngoại vi và cũng dễ hiểu rõ bản chất của lập trình.

Cấu hình chi tiết của STM32F103 như sau:

ARM 32-bit Cortex M3 với clock max là 72Mhz.

Bộ nhớ:

- 64 kbytes bộ nhớ Flash(bộ nhớ lập trình).
- 20kbytes SRAM.

Clock, reset và quản lý nguồn.

- Điện áp hoạt động 2.0V -> 3.6V.
- Power on reset(POR), Power down reset(PDR) và programmable voltage detector (PVD).
- Sử dụng thạch anh ngoài từ 4Mhz -> 20Mhz.
- Thạch anh nội dùng dao động RC ở mode 8Mhz hoặc 40khz.
- Sử dụng thạch anh ngoài 32.768khz được sử dụng cho RTC.

Trong trường hợp điện áp thấp:

- Có các mode :ngủ, ngừng hoạt động hoặc hoạt động ở chế độ chờ.
- Cấp nguồn ở chân Vbat bằng pin để hoạt động bộ RTC và sử dụng lưu trữ data khi mất nguồn cấp chính.

2 bộ ADC 12 bit với 9 kênh cho mỗi bộ.

- Khoảng giá trị chuyển đổi từ 0 – 3.6V.
- Lấy mẫu nhiều kênh hoặc 1 kênh.
- Có cảm biến nhiệt độ nội.

DMA: bộ chuyển đổi này giúp tăng tốc độ xử lý do không có sự can thiệp quá sâu của CPU.

- 7 kênh DMA.
- Hỗ trợ DMA cho ADC, I2C, SPI, UART.

7 timer.

- 3 timer 16 bit hỗ trợ các mode IC/OC/PWM.
- 1 timer 16 bit hỗ trợ để điều khiển động cơ với các mode bảo vệ như ngắt input, dead-time..

- 2 watdog timer dùng để bảo vệ và kiểm tra lỗi.
- 1 sysTick timer 24 bit đếm xuống dùng cho các ứng dụng như hàm Delay….

Hỗ trợ 9 kênh giao tiếp bao gồm:

- 2 bộ I2C(SMBus/PMBus).
- 3 bộ USART(ISO 7816 interface, LIN, IrDA capability, modem control).
- 2 SPIs (18 Mbit/s).
- 1 bộ CAN interface (2.0B Active)
- USB 2.0 full-speed interface

Kiểm tra lỗi CRC và 96-bit ID.

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.

Chuẩn CMSIS

Các vi điều khiển dựa trên nền tảng ARM Cortex-M3 đang trở nên rất phổ biến trong các thiết bị công nghiệp. Gần đây là sự giới thiệu của dòng Cortex-M0 với các đặc tính kỹ thuật nổi trội cùng với chi phí thấp hơn. Cùng với đó là sự ra đời của chuẩn Cortex

Microcontroller Software Interface Standard (**CMSIS**) làm cho việc tương thích phần mềm cũng như sử dụng lại mã nguồn trở nên đơn giản và dễ dàng.

Một trong các hạn chế lớn nhất của kiến trúc 8 bit và 16 bit hiện nay là khả năng sửa lỗi chương trình. Trong nhiều trường hợp chúng ta chỉ có thể sử dụng điểm dừng (breakpoint) để sửa lỗi chương trình còn theo dõi dữ liệu động khi chương trình đang thực thi thì không thể. Với các chương trình đơn giản, các hạn chế trên có thể không ảnh hưởng nhiều đến việc sửa lỗi, nhưng với các ứng dụng phức tạp, lượng mã nguồn nhiều và có sự tích hợp với nhiều thành phần cứng và mềm thì hạn chế trên làm phức tạp quá trình sửa lỗi ảnh hưởng đến tiến độ, thậm chí là sự thành công của dự án. Một ví dụ thực tế trên nền tảng PC, khi chuẩn phần cứng dành cho PC ra đời (Ethernet, USB, Firewire…), cộng thêm sự hỗ trợ đắc lực từ phần cứng đối với việc theo dõi lỗi, rất nhiều phần mềm hữu ích đã được phát triển và triển khai thành công.

Bộ xử lý Cortex-M3 tích hợp nhân CPU 32-bit, bộ điều khiển đánh thức khi có ngắt giúp cho các hoạt động tiêu tốn ít năng lượng hơn, và hệ thống điều khiển ngắt lồng nhau (nested vector interrupt controller-NVIC) cho phép rút ngắn thời gian trì hoãn đáp ứng ngắt (tức hệ thống đáp ứng ngắt nhanh hơn) với nhiều mức ưu tiên khác nhau. ETM là viết tắt của cụm từ “embedded trace macrocell”, đây là hệ thống chuẩn cung cấp đầy đủ chức năng sửa lỗi và theo dõi biến của chương trình. Nó cho phép lấy thông tin từ CPU trước và sau khi thực thi lệnh. ETM có thể được cấu hình bởi phần mềm và thông qua cổng giao tiếp đặc biệt(serial wire viewer) từ đó chương trình ở ngoài có thể giao tiếp, cấu hình và lấy các thông tin cần thiết cho việc sửa lỗi mà không ảnh hưởng gì đến chương trình đang thực thi.

Để chuẩn bị cho việc mở rộng và phổ biến nhân xử lý Cortex trong tương lai, chuẩn CMSIS đã ra đời nhằm giải quyết các vấn đề tương thích giữa phần mềm và phần cứng. CMSIS là chuẩn đặt ra bởi các nhà sản xuất phần cứng và phần mềm nhằm tạo nên một chuẩn phần mềm được chấp thuận rộng rãi trong công nghiệp.

Dễ dàng sử dụng và dễ dàng học, cung cấp các chuẩn giao tiếp cho các thiết bị ngoại vi, hệ điều hành thời gian thực và phần mềm hỗ trợ là mục tiêu chính của CMSIS. Ngoài ra CMSIS cũng tương thích với các trình biên dịch phổ biến hiện nay như GCC, IAR, Keil…

CMSIS gồm 2 lớp:

- Peripheral Access Layer (CMSIS-PAL): lớp này xác định tên, địa chỉ, và các hàm cần thiết để truy cập đến thanh ghi của lõi và thiết bị ngoại vi chuẩn. Ngoài ra CMSIS-PAL còn giới thiệu cách truy cập phù hợp với nhiều loại thiết bị ngoại vi bên trong lõi, và các vector xử lý ngoại lệ (exception) và ngắt. Cùng với đó là các hàm khởi động các thiết lập ban đầu cho hệ thống, hệ thống hàm giao diện chuẩn để giao tiếp với nhân của hệ điều hành và hệ thống hỗ trợ sửa lỗi.

- Middleware Access Layer (CMSIS-MAL): cung cấp các phương thức để truy cập vào thiết bị ngoại vi từ các lớp ứng dụng phía trên. Lớp này được quy định bởi hiệp hội các nhà sản xuất chip, do đó giúp tái sử dụng lại các thư viện cấp cao và phức tạp. Hiện nay, CMSIS-MAL vẫn đang trong quá trình phát triển và được hỗ trợ bởi IAR, Keil, Micrium, Segger và nhiều nhà sản xuất phần mềm khác.

Sự xuất hiện của CMSIS hoàn toàn không ảnh hưởng đến các thiết bị phần cứng hiện thời cũng như các hệ thống sẵn có. Chúng ta hoàn toàn có thể dùng các cách thông thường để truy cập trực tiếp đến thanh ghi để điều khiển thiết bị mà không cần dùng CMSIS. Tài nguyên hệ thống dành cho CMSIS-PAL cũng không đáng kể. CMSIS-MAL chỉ đòi hỏi giao diện mềm và chuẩn cho các lớp bên trên, do vậy cũng không làm tiêu tốn tài nguyên. Xây dựng hệ thống dựa trên chuẩn CMSIS sẽ tạo nên một bộ khung lập trình rõ ràng, dễ dàng theo dõi, mở rộng và tái sử dụng. Thêm vào đó mã nguồn chuẩn CMSIS được hỗ trợ bởi nhiều trình biên dịch. Nhằm tạo sự dễ dàng cho các nhà phát triển, ARM đã biên soạn hệ thống tài liệu tham khảo dành cho Cortex-M3 cũng như các thiết bị ngoại vi đi kèm tương thích với CMSIS.

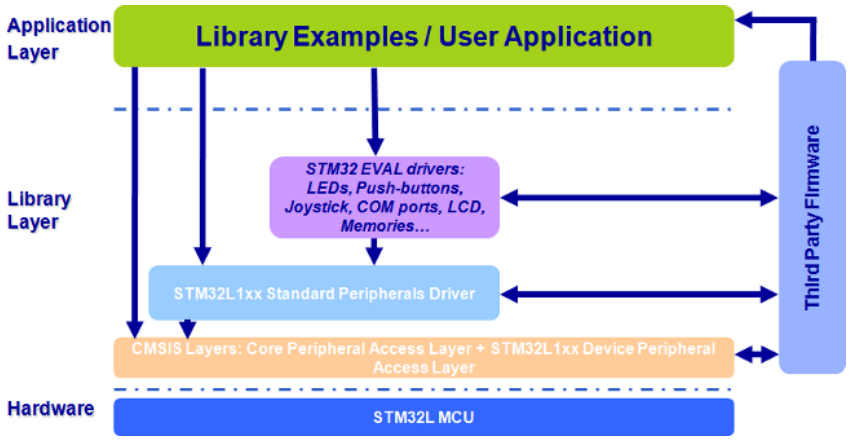
CMSIS đươc thiết kế độc lập với trình biên dịch. Hệ thống hàm chuẩn được hỗ trợ bởi phần cứng và phần mềm cung cấp khung phần mềm chuẩn do đó giảm thiểu rủi ro trong quá trình phát triển. Một khi chuẩn được sử dụng rộng rãi, mã nguồn sẽ trở nên dễ hiểu, dễ sử dụng lại, cũng như dễ dàng chỉnh sửa lỗi.

Trong công nghiệp, hệ thống tiêu chuẩn kỹ thuật rất quan trọng trong việc cải tiến chất lượng sản phẩm, phát triển các thành phần phụ cũng như giảm chi phí phát triển. Tuy nhiên sự phát triển của bộ vi xử lý lại không tuân theo một chuẩn nhất định, mỗi nhà sản xuất lại sử dụng những hệ thống riêng biệt không tương thích lẫn nhau. Do đó với sự ra đời của vi xử lý chuẩn Cortex-M3 không chỉ cho phép sử dụng lại bộ công cụ phát triển mà cùng với CMSIS giúp làm giảm chi phí, thời gian triển khai cùng với rủi ro kỹ thuật. Các nhà phát triển phần cứng có thể tập trung nguồn lực phát triển tính năng nổi trội của thiết bị và hỗ trợ ngoại vi.

Thư viện Standard Peripheral Library (SPL)

Thư viện Standard Peripheral Library (SPL) là một trong các bộ thư viện phổ biến được sử dụng cho các dòng ARM M3 của STM32 như: STM32Snippets, STM32Cube, HAL API, LL API,...

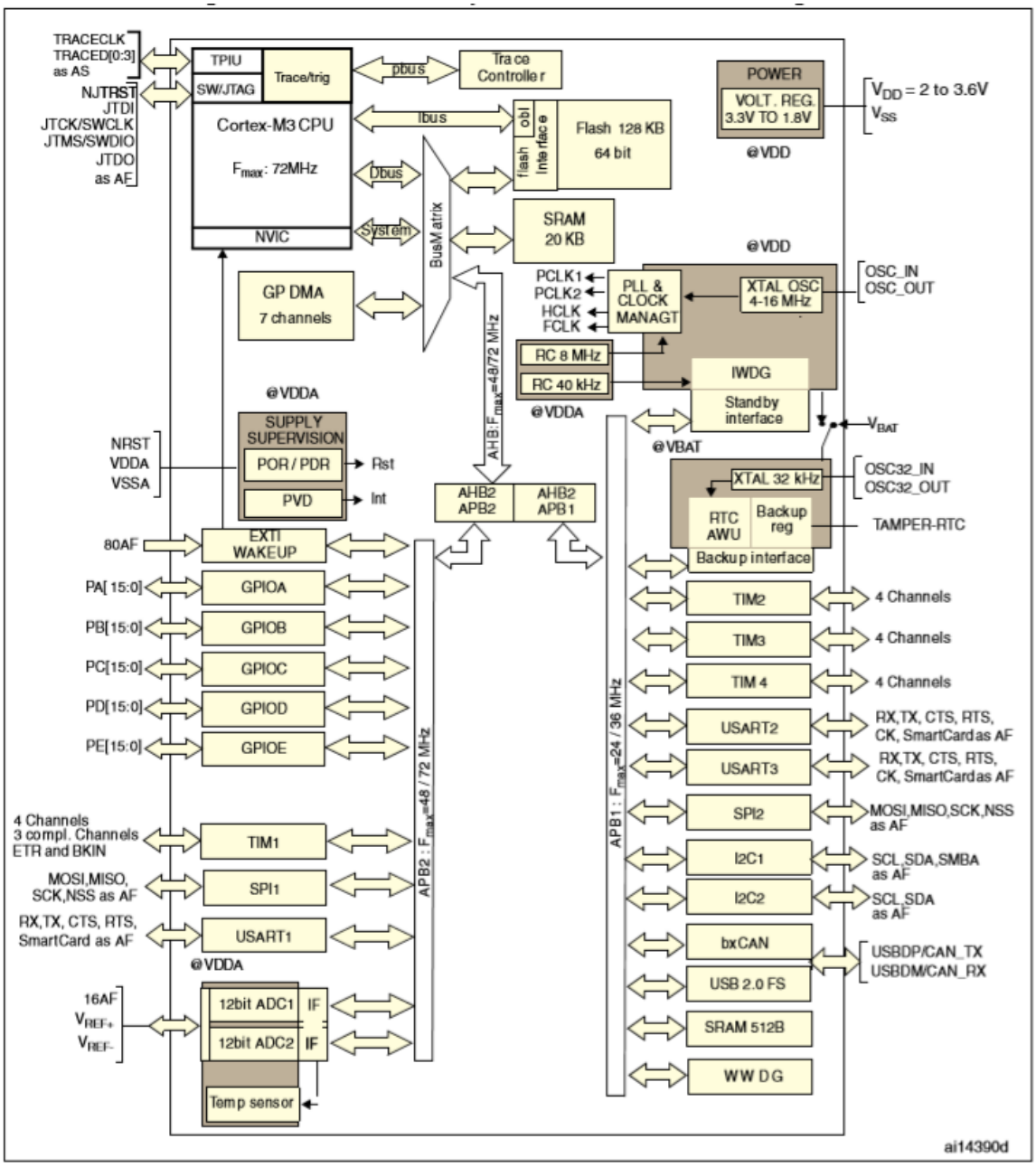
Ở giáo trình này dừng lại ở phân tích và hướng dẫn sử dụng với bộ thư viện SPL, với các bộ thư viện khác người dùng hoàn toàn có thể sử dụng với cách sử dụng tương đương.



Hình 5. 7: Vai trò của CMSIS và SPL trong phát triển phần mềm cho vi điều khiển ARM

Với sự phức tạp của kiến trúc STM32F1, việc sử dụng các thanh ghi trở nên khó khăn do vậy thư viện SPL được thiết kế nhằm mục đích cho người lập trình nhanh chóng tiếp cận và lập trình điều khiển được các ngoại vi của các dòng IC này. Việc sử dụng thư viện SPL giúp cho người lập trình có thể nhanh chóng thiết kế ứng dụng mà không cần tập trung quá nhiều vào cấu trúc cũng như các thanh ghi cấu hình của lõi vi điều khiển đang sử dụng.

Quy trình lập trình chương trình STM32F1



Hình 5. 8: Kiến trúc các ngoại vi của STM32

Theo kiến trúc các ngoại vi của dòng vi điều khiển STM32, để lập trình một chương trình điều khiển cho dòng vi điều khiển STM32F1 cần tuân thủ các bước sau:

- Cấu hình Clock chung của vi điều khiển.
- Cấp phát Clock cho ngoại vi cần sử dụng.
- Thiết lập cấu hình cho ngoại vi cần sử dụng.

Việc cấp clock chung cho hệ thống của vi điều khiển STM32F1 có thể được thực hiện nhanh chóng qua các tập lệnh đã được hỗ trợ bởi CMSIS như sau:

```
SystemInit();
SystemCoreClockUpdate();
```

Với dòng vi điều khiển Cortex M3 của ST chúng ta hoàn toàn có thể cấu hình để thay đổi tần số hoạt động của vi điều khiển. Người đọc muốn tìm hiểu sâu hơn có thể tìm hiểu trong User Manual của hãng. Trong giáo trình này dừng lại ở thiết kế hệ thống với vi điều khiển được cấu hình hoạt động của tần số hoạt động mặc định.

Các việc cấu hình clock cho ngoại vi và cấu hình cho ngoại vi được trình bày chi tiết ở từng phần ngoại vi cần lập trình tiếp theo.

Bộ thanh ghi RCC (Register Clock Control)

Bộ thanh ghi RCC là bộ thanh ghi dùng để điều khiển Clock của IC và tất cả các ngoại vi, tài nguyên của IC. Để một bộ ngoại vi như: GPIO, Timer, UART,... hoạt động ngoài Clock của hệ thống được kích hoạt chương trình phần mềm cần thực hiện các lệnh cấu hình để cấp clock cho bộ ngoại vi tương ứng.

Bộ RCC có số lượng thanh ghi rất lơn. Ở giáo trình này tập trung vào 2 thanh ghi điều khiển cấp Clock cho các bộ ngoại vi của vi điều khiển.

Thanh ghi RCC_AHBENR

Thanh ghi này có nhiệm vụ kích hoạt clock cho các ngoại vi được nối với bus điều khiển ngoại vi AHB như: SDIO, CRC, DMA1, DMA2,...

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					SDIO EN	Res.	FSMC EN	Res.	CRCE N	Res.	FLITF EN	Res.	SRAM EN	DMA2 EN	DMA1 EN
					rw		rw		rw		rw		rw	rw	rw

Hình 5. 9: Thanh ghi RCC_AHBENR

Thanh ghi RCC_APB2ENR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Hình 5. 10: RCC_APB2ENR

Thanh ghi này có nhiệm vụ kích hoạt clock cho các ngoại vi được nối với bus điều khiển ngoại vi APB2 như: Timer, ADC, USART, SPI, IO, AFIO,...

Các bộ GPIO của vi điều khiển đều được kích hoạt thông qua thanh ghi này.

Thanh ghi RCC_APB1ENR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	Res.	CAN EN	Res.	USB EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	Res.
		rw	rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWD GEN	Reserved		TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw			rw	rw	rw	rw	rw	rw	rw	rw	rw

Hình 5. 11: Thanh ghi RCC_APB1ENR

Thanh ghi này có nhiệm vụ kích hoạt clock cho các ngoại vi được nối với bus điều khiển ngoại vi APB1 như: DAC, CAN, USB, Timer,...

Ví dụ để kích hoạt clock cho GPIOA của vi điều khiển chương trình phần mềm cần thực hiện lệnh sau.

```
RCC->APB2ENR |= 1<<2;
```

Hoặc thực hiện dưới dạng macro theo thư viện CMSIS như sau:

```
RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
```

Có thể thực hiện được cách viết như trên là do thư viện CMSIS đã định nghĩa giá trị của RCC_APB2ENR_IOPAEN tương ứng với 0x00000004 (Có thể tham khảo file stm32f10x.h của bộ thư viện CMSIS).

Lưu ý: Việc cấp clock cho các ngoại vi nên được thực hiện trước khi thiết lập cấu hình và điều khiển các ngoại vi đó.

Lập trình điều khiển IO với STM32F1 sử dụng thanh ghi

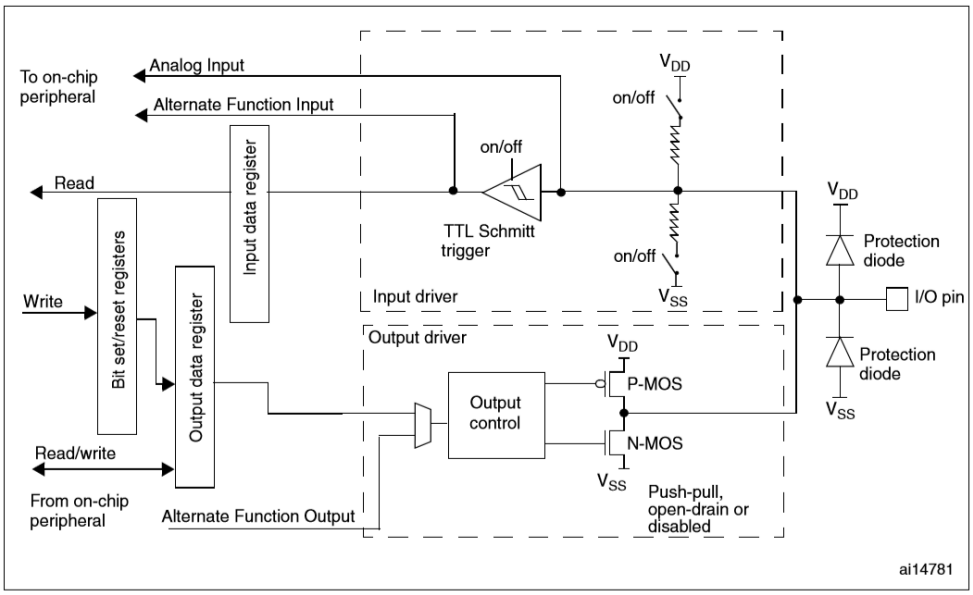
Dòng STM32F1 (Cortex M3) của STM32 cung cấp tối đa 7 Port IO (PortA, PortB, PortC, PortD, PortE, PortF, PortG) với các chân vào ra được cấu hình có thể có tối đa đến 4 chức năng. Các Port này có tên là GPIO (General-Purpose I/O).

Mỗi bộ GPIO của STM32F1 có 2 thanh ghi cấu hình (GPIOx_CRL, GPIOx_CRH), 2 thanh ghi dữ liệu (GPIOx_IDR, GPIOx_ODR), một thanh ghi Lập/Xóa ((GPIOx_BSRR), một thanh ghi (GPIOx_BRR) và một thanh ghi khóa (GPIOx_LCKR).

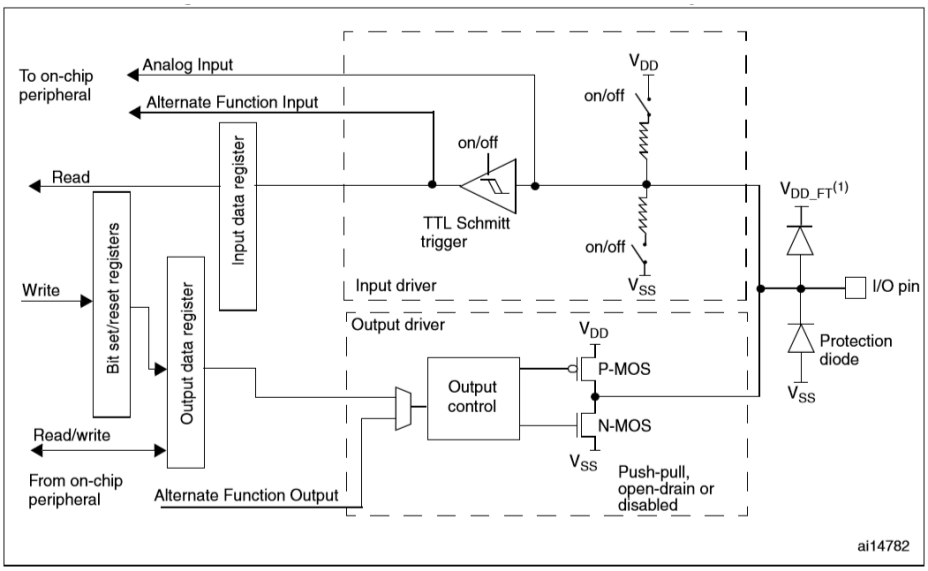
Mỗi bộ GPIO hỗ trợ điều khiển tối đa 16 chân GPIO. Số lượng chân của mỗi GPIO trên mỗi IC phụ thuộc vào số lượng cổng mà IC đó hỗ trợ. Các chân này có thể cấu hình độc lập với các chức năng hoạt động khác nhau. Danh sách các chế độ hoạt động của mỗi cổng như sau:

- Đầu vào thả nổi số
- Đầu vào kéo lên số
- Đầy vào kéo xuống số
- Tương tự
- Đầu ra thả nổi
- Đầu ra kéo lên
- Chức năng luân phiên kéo lên
- Chức năng luân phiên thả nổi

Mỗi GPIO có thể cấu hình độc lập nhưng mỗi thanh ghi cấu hình phải được truy cập cấu hình thông qua giao diện 32 bit.



Hình 5. 12: Cấu trúc của các chân điều khiển thông dụng



Hình 5. 13: Cấu trúc của các chân IO hỗ trợ giao tiếp 5V

Bảng cấu hình của các chân IO được giải thích ở bảng sau:

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR register
General purpose output	Push-pull	0	0	01 10 11 see *Table 21*		0 or 1
	Open-drain		1			0 or 1
Alternate Function output	Push-pull	1	0			Don't care
	Open-drain		1			Don't care
Input	Analog	0	0	00		Don't care
	Input floating		1			Don't care
	Input pull-down	1	0			0
	Input pull-up					1

Hình 5. 14: Thông tin cấu hình IO

Với chế độ hoạt động dạng đầu ra số còn có thêm cấu hình về tốc độ phụ thuộc vào bảng sau:

MODE[1:0]	Meaning
00	Reserved
01	Maximum output speed 10 MHz
10	Maximum output speed 2 MHz
11	Maximum output speed 50 MHz

Hình 5. 15: Các chế độ tốc độ đầu ra số

Ở phần này, giáo trình sẽ phân tích chi tiết về các thanh ghi của bộ GPIO, các phần sau chỉ dừng lại ở thông tin gợi mở, người đọc muốn tìm hiểu chi tiết có thể tham khảo datasheet và hướng dẫn sử dụng IC của hãng.

Mỗi GPIO của STM32F1 hỗ trợ điều khiển 16 cổng (0-15) thông qua giao diện thanh ghi 32 bit. Việc cấu hình hoạt động cho mỗi cổng của IC cần thông qua 4 bit điều khiển CNF[1:0] và MODE [1:0]. Do vậy thanh ghi cấu hình hoạt động cho cả bộ GPIO cần chia làm 2 thanh ghi: Một thanh ghi điều khiển chế độ của các cổng từ 0 đến 7 và một thanh ghi điều khiển chế độ hoạt động của các cổng từ 8 đến 15.

Thanh ghi GPIOx_CRL

Address offset: 0x00

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Hình 5. 16: Thanh ghi GPIOx_CRL

Thanh ghi GPIOx_CRH

Address offset: 0x04

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Hình 5. 17: Thanh ghi GPIOx_CRH

Trong đó chi tiết về các cách cấu hình cho mỗi cổng như sau:

Chế độ Input (MODE[1:0] = 00):

CNF[1:0]: 00: Vào tương tự
01: Vào số thả nổi
10: Vào số kéo lên hoặc kéo xuống (Phụ thuộc vào thanh ghi ODR)
11: Dự trữ

Chế độ Output (MODE[1:0] > 00):

CNF[1:0]: 00: Đầu ra đa dụng kéo lên
01: Đầu ra đa dụng thả nổi
10: Đầu ra luân phiên kéo lên
11: Đầu ra luân phiên thả nổi

Với MODE[1:0] > 00 tức là cổng được cấu hình đầu ra, giá trị của MODE[1:0] quyết định tốc độ hoạt động tối đa của cổng như sau:

MODE[1:0]: 00: Đầu vào
01: Đầu ra, tần số tối đa 10MHz
10: Đầu ra, tần số tối đa 2MHz
11: Đầu ra, tần số tối đa 50MHz

Thanh ghi GPIOx_IDR

Address offset: 0x08h

Reset value: 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Hình 5. 18: Thanh ghi GPIOx_IDR

Thanh ghi này là thanh ghi chỉ đọc, dùng để đọc dữ liệu đầu vào từ các GPIO tương ứng. Mỗi bit IDR tương ứng đọc giá trị đầu vào của chân tương ứng. Thanh ghi này thường được sử dụng khi cần đọc dữ liệu đầu vào số của một cổng nào đó.

Thanh ghi GPIOx_ODR

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Hình 5. 19: Thanh ghi GPIOx_ODR

Thanh ghi này có thể truy cập đọc và ghi có chức năng dùng để điều khiển đầu ra số tại các chân đầu ra tương ứng. Khi tác động đến thanh ghi này các chân được cài đặt ở chế độ đầu ra tương ứng sẽ bị ảnh hưởng.

Thanh ghi Lập/Xóa GPIOx_BSRR

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Hình 5. 20: Thanh ghi Lập/Xóa GPIOx_BSRR

Thanh ghi này là thanh ghi chỉ ghi và không thể đọc về. Trong đó thanh ghi được chia làm 2 nửa từ BS0-BS15 và BR0-BR15.

BS0-BS15: Nếu ghi giá trị 0 vào các bit tương ứng này thì đầu ra của chân tương ứng không thay đổi, nếu ghi giá trị 1 vào các bit tương ứng thì bit tương ứng của thanh ghi ODR được lập lên 1 (Tức là điều khiển cổng tương ứng lên mức đầu ra là 1 – 3,3V).

BR0-BR15: Nếu ghi giá trị 0 vào các bit tương ứng này thì đầu ra của chân tương ứng không thay đổi, nếu ghi giá trị 1 vào các bit tương ứng thì bit tương ứng của thanh ghi ODR được xóa về 0 (Tức là điều khiển cổng tương ứng với mức đầu ra là 0 – 0V).

Thanh ghi Xóa GPIOx_BRR

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Hình 5. 21: Thanh ghi Xóa GPIOx_BRR

Thanh ghi này chứa 16 bit có thể truy cập dưới dạng chỉ ghi có chức năng tương đương như các bit từ BR0-BR15 của thanh ghi BSRR.

Thanh ghi khóa cấu hình GPIOx_LCKR

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															LCKK
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Hình 5. 22: Thanh ghi khóa cấu hình GPIOx_LCKR

Thanh ghi này dùng để khóa cấu hình của chân GPIO tương ứng cho mỗi cổng. Ngoài ra, thanh ghi còn có thêm 1 bit LCKK là bit Lock Key.

Chức năng của các bit cụ thể như sau:

LCKK: 0: Không cho phép chế độ khóa cấu hình cổng của Port tương ứng hoạt động.

1: Kích hoạt chế độ khóa khóa cấu hình cổng của Port tương ứng.

LCKy: 0: Cấu hình của cổng tương ứng được mở.

1: Cấu hình của cổng tương ứng bị khóa.

Khi cấu hình của cổng tương ứng bị khóa chương trình phần mềm không được phép thay đổi chức năng của cổng đó cho đến khi khóa cấu hình được mở.

Điều khiển đầu ra: Với tiến trình và các thanh ghi đã tìm hiểu chương trình điều khiển thực hiện nhấp nháy đèn LED sử dụng STM32F1 có thể được viết trực tiếp bằng các thanh ghi theo một số cách như sau:

Cách 1: Sử dụng thanh ghi ODR

```
#include "stm32f10x.h"


void delay(int _Time){// Hàm delay
	unsigned int Count;
	while(_Time--){
		for (Count = 0; Count < 1000; Count++);
	}
}


int main (void){
	SystemInit();
	SystemCoreClockUpdate();
	RCC->APB2ENR |= RCC_APB2ENR_IOPCEN; // Cap Clock cho GPIOC
	// Khoi tao cau hinh nguyen thuy cho Port C
	GPIOA->CRH = 0x00000000;
```

```
	GPIOA->CRL  = 0x00000000;
	GPIOA->CRH |= 0x00300000; // Cau hinh chan C13 o che do Output Push-pull CNF[1:0] = 00, MODE[1:0] = 11
	while(1){
		GPIOC->ODR |= (1<<13);  // Bat LED – Xuat 3.3V ra C13
		delay(1000);            // Tam dung
		GPIOC->ODR &= ~(1<<13); // Tat LED – Xuat 0V ra C13
		delay(1000);
	}
}
```

Cách 2: Sử dụng thanh ghi BSRR và BRR

```
#include "stm32f10x.h"

void delay(int _Time){// Hàm delay
	unsigned int Count;
	while(_Time--){
		for (Count = 0; Count < 1000; Count++);
	}
}

int main (void){
	SystemInit();
	SystemCoreClockUpdate();
	RCC->APB2ENR |= RCC_APB2ENR_IOPCEN; // Cap Clock cho GPIOC
	// Khoi tao cau hinh nguyen thuy cho Port C
	GPIOA->CRH = 0x00000000;
	GPIOA->CRL  = 0x00000000;
	GPIOA->CRH |= 0x00300000; // Cau hinh chan C13 o che do Output Push-pull CNF[1:0] = 00, MODE[1:0] = 11
	while(1){
		GPIOC->BSRR |= (1<<13); // Bat LED – Xuat 3.3V ra C13
```

```
            delay(1000);            // Tam dung
            GPIOC->BRR |= (1<<13);  // Tat LED – Xuat 0V ra C13
            delay(1000);
      }
}
```

Như vậy tiến trình để điều khiển đầu ra của một cổng như sau:

- Điều khiển clock hệ thống
- Cấp clock cho GPIO tương ứng chứa cổng đó
- Cấu hình cho cổng tương ứng ở chế độ Output
- Điều khiển dữ liệu của cổng tương ứng thông qua thanh ghi ODR hoặc BSRR hoặc BRR.

Đọc dữ liệu đầu vào: Xây dựng chương trình đọc dữ liệu từ 8 bit thấp của cổng A và xuất ra 8 bit cao của cổng A.

Để làm được điều này cần thực hiện cấu hình GPIOA hoạt động với 8 cổng thấp ở chế độ vào số và 8 cổng cao ở chế độ đầu ra số. Sau đó, chương trình thực hiện đọc dữ liệu từ 8 bit thấp của thanh ghi IDR và xuất ra 8 bit cao của thanh ghi IDR. Chương trình cụ thể như sau:

```
#include "stm32f10x.h"

void delay(int _Time){// Hàm delay
      unsigned int Count;
      while(_Time--){
            for (Count = 0; Count < 1000; Count++);
      }
}

int main (void){
      unsigned int TempRead;

      SystemInit();
      SystemCoreClockUpdate();
```

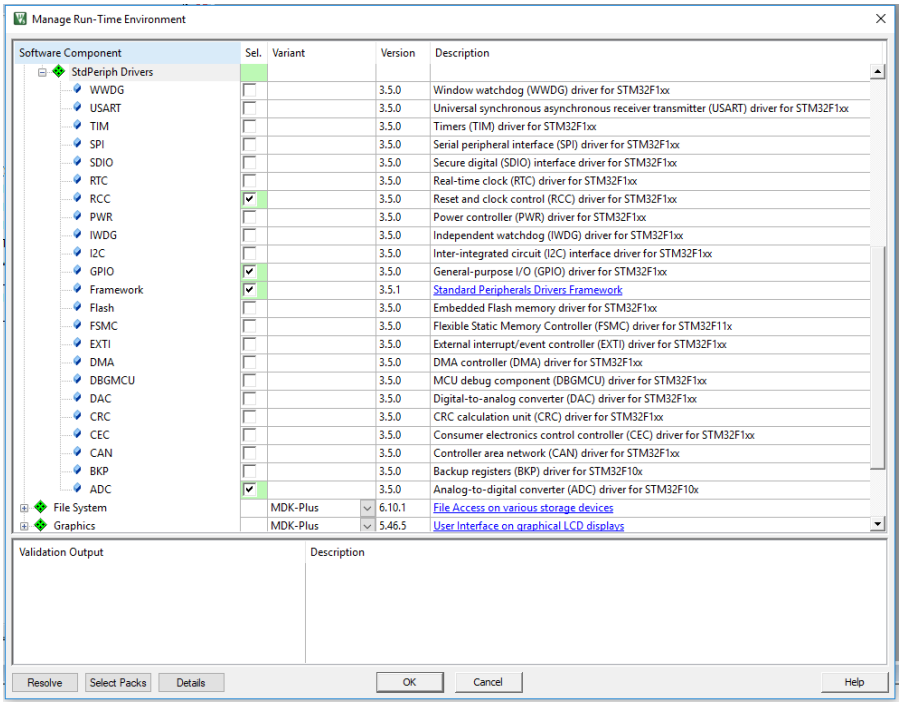
```
	RCC->APB2ENR |= RCC_APB2ENR_IOPAEN; // Cap Clock cho GPIOC


	GPIOA->CRH = 0x33333333; // Cau hinh 8 chan cao o che do Output Push-pull CNF[1:0] = 00, MODE[1:0] = 11


	// Cau hinh 8 chan thap o che do Input Pull-up/Pull-down CNF[1:0] = 10, MODE[1:0] = 00
	GPIOA->CRL = 0x88888888;
	GPIOA->ODR = 0x00FF; // Chuyen 8 bit thap ve che do Pull-Up
	while(1){
		//Doc du lieu dau vao
		TempRead = GPIOA->IDR & 0x00FF;// Chi doc du lieu 8 bit thap tu thanh ghi IDR
		GPIOA->ODR = TempRead<<8; // Dich 8 bit thap thanh 8 bit cao va xuat ra thanh ghi ODR
		delay(1);
	}
}
```

Lập trình điều khiển IO với STM32F1 sử dụng SPL

Như đã trình bày ở trên thư viện SPL là một trong những thư viện phổ biến sử dụng để lập trình cho STM32F1. Thư viện này có vai trò giúp người lập trình nhanh chóng tiếp cận được lập trình các dòng vi điều khiển của ARM mà không cần nghiên cứu sâu về kiến trúc các thanh ghi của vi điều khiển. Sau khi nêu ra một ví dụ sử dụng thanh ghi giáo trình chuyển sang hướng dẫn trên nền tảng sử dụng thư viện SPL để sinh viên, người đọc có thể dễ dàng tiếp cận các dòng vi điều khiển. Tuy nhiên, cũng khuyến nghị các sinh viên, người đọc sau khi có thể điều khiển bằng SPL có thể đọc lại các thanh ghi để nắm được hoạt động cụ thể của các vi điều khiển này.

Sau đây, để thực hiện lập trình điều khiển IO sử dụng SPL, giáo trình xin giới thiệu về cấu trúc thư viện SPL cho STM32F1.

Với KeilC 5 người dùng có thể dễ dàng tùy chọn gói thư viện SPL bằng cách chọn: Manage Run → Device -> StdPeriph Drivers sau đó chọn các gói thư viện cần sử dụng. Sau đó, KeilC sẽ tự động thêm các gói này vào dự án của người lập trình.



Hình 5. 23: Cấu hình các package sử dụng cho Project với KeilC 5

Với chương trình thực hiện điều khiển IO cần thực hiện chọn các gói sau:

- Framework: Cho phép sử dụng nền tảng SPL trong Project
- RCC: Điều khiển clock cho các ngoại vi, ở đây sử dụng để điều khiển clock cho các bộ GPIO.
- GPIO: Cấu hình và điều khiển các bộ GPIO tương ứng

Gói RCC có các hàm chính như sau:

```
void RCC_LSEConfig(uint8_t RCC_LSE);
void RCC_LSICmd(FunctionalState NewState);
void RCC_RTCCLKConfig(uint32_t RCC_RTCCLKSource);
void RCC_RTCCLKCmd(FunctionalState NewState);
void RCC_GetClocksFreq(RCC_ClocksTypeDef* RCC_Clocks);
void RCC_AHBPeriphClockCmd(uint32_t RCC_AHBPeriph, FunctionalState
NewState);
void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState
NewState);
void RCC_APB1PeriphClockCmd(uint32_t RCC_APB1Periph, FunctionalState
NewState);
void RCC_APB2PeriphResetCmd(uint32_t RCC_APB2Periph, FunctionalState
```

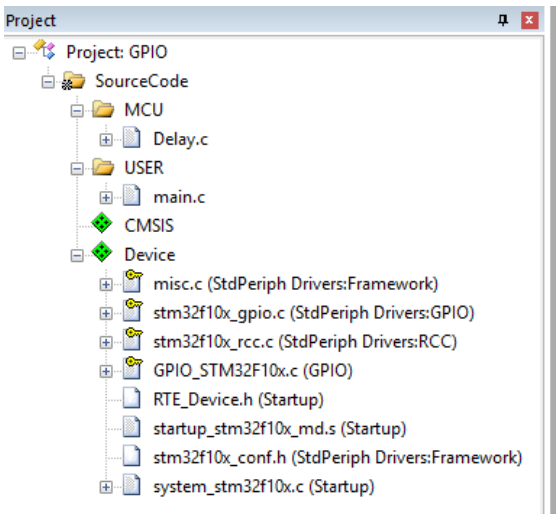
```
NewState);
void RCC_APB1PeriphResetCmd(uint32_t RCC_APB1Periph, FunctionalState NewState);
void RCC_BackupResetCmd(FunctionalState NewState);
void RCC_ClockSecuritySystemCmd(FunctionalState NewState);
void RCC_MCOConfig(uint8_t RCC_MCO);
FlagStatus RCC_GetFlagStatus(uint8_t RCC_FLAG);
void RCC_ClearFlag(void);
ITStatus RCC_GetITStatus(uint8_t RCC_IT);
void RCC_ClearITPendingBit(uint8_t RCC_IT);
```

Gói GPIO có các hàm như sau:

```
void GPIO_DeInit(GPIO_TypeDef* GPIOx);
void GPIO_AFIODeInit(void);
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);
void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct);
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);
uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);
void GPIO_PinLockConfig(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_EventOutputConfig(uint8_t GPIO_PortSource, uint8_t GPIO_PinSource);
void GPIO_EventOutputCmd(FunctionalState NewState);
void GPIO_PinRemapConfig(uint32_t GPIO_Remap, FunctionalState NewState);
void GPIO_EXTILineConfig(uint8_t GPIO_PortSource, uint8_t GPIO_PinSource);
```

```
void GPIO_ETH_MediaInterfaceConfig(uint32_t GPIO_ETH_MediaInterface);
```

Sau khi lựa chọn mục Device của KeilC sẽ có giao diện như sau:



Hình 5. 24: Thông tin các gói sau khi cấu hình

Cá file ở mục Device là các file đã được phần mềm tự động thêm vào theo lựa chọn của người lập trình. Sau khi đã cấu hình có thể lập trình chương trình main cho project thực hiện nhấp nháy LED tại chân A0 như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO

void delay(int _Time){// Hàm delay
	unsigned int Count;
	while(_Time--){
		for (Count = 0; Count < 1000; Count++);
	}
}
void Fn_GPIO_Init (void);
int main (void){
	SystemInit();
	SystemCoreClockUpdate();

	Fn_GPIO_Init(); // Goi ham khoi tao GPIO
	while(1){
```

```
            GPIO_WriteBit(GPIOA, GPIO_Pin_0, Bit_RESET); // Xuat du
lieu 0V (0) ra chan A0
            delay(1000);
            GPIO_WriteBit(GPIOA, GPIO_Pin_0, Bit_SET);     // Xuat du
lieu 3.3V(1) ra chan A0
            delay(1000);
      }
}
void Fn_GPIO_Init (void){
      GPIO_InitTypeDef GPIO_InitStructure; // Bien chua thong tin cau
hinh GPIO


      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,   ENABLE);   //   Cap
Clock cho GPIOA
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0; // Chan cau hinh la
chan 0
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // Che do hoat
dong cua chan cau hinh la Ouput
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;    //  Tan  so
hoat dong toi da cua chan cau hinh la 50MHz
      GPIO_Init(GPIOA, &GPIO_InitStructure); // Nap cau hinh tren vao
GPIOA
}
```

Chương trình điều khiển dữ liệu cho cả GPIOA có thể thực hiện như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO


void delay(int _Time){// Hàm delay
      unsigned int Count;
      while(_Time--){
            for (Count = 0; Count < 1000; Count++);
      }
```

```
}

void Fn_GPIO_Init (void);

int main (void){
	unsigned int Count = 0;
	SystemInit();
	SystemCoreClockUpdate();

	Fn_GPIO_Init(); // Goi ham khoi tao GPIO
	while(1){
		GPIO_Write(GPIOA, Count); // Xuat du lieu tu bien Count ra 16 cong cua GPIOA
		Count++;
		delay(1000);
	}
}

void Fn_GPIO_Init (void){
	GPIO_InitTypeDef GPIO_InitStructure; // Bien chua thong tin cau hinh GPIO

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA);
	// Cau hinh toan bo GPIOA la Output
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

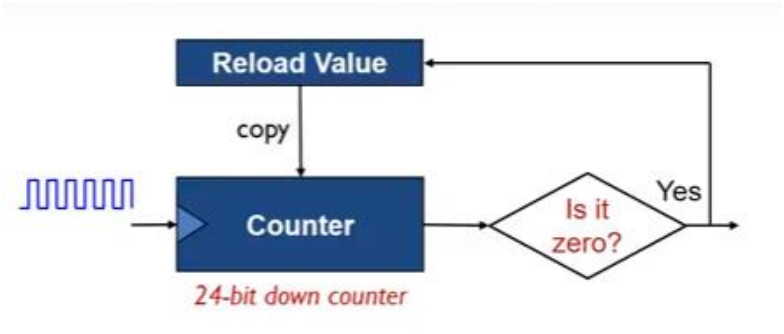
Chương trình đọc dữ liệu tại GPIOB và xuất ra GPIOA được thực hiện như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO

void delay(int _Time){// Hàm delay
	unsigned int Count;
	while(_Time--){
		for (Count = 0; Count < 1000; Count++);
	}
}

void Fn_GPIO_Init (void);

int main (void){
	unsigned int ReadValue = 0;
	SystemInit();
	SystemCoreClockUpdate();

	Fn_GPIO_Init(); // Goi ham khoi tao GPIO
	while(1){
		ReadValue = GPIO_ReadInputData(GPIOB); // Doc du lieu GPIOB vao bien ReadValue
		GPIO_Write(GPIOA, ReadValue); // Xuat gia tri tu bien ReadValue ra GPIOA
		Fn_DELAY_ms(10);
	}
}

void Fn_GPIO_Init (void){
	GPIO_InitTypeDef GPIO_InitStructure; // Bien chua thong tin cau hinh GPIO
```

```
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
RCC_APB2Periph_GPIOB, ENABLE);
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_Init(GPIOA, &GPIO_InitStructure);



      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
      GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

Lập trình Systick

Systick là một bộ đếm xuống 24 bit và có khả năng tự động nạp lại giá trị (auto reload).



Hình 5. 25: Hoạt động của Systick

Systick được ví như một cái đồng hồ đếm ngược, nó được tạo ra để cung cấp một bộ thời gian chuẩn cho hệ thống. Đồng hồ Systick được sử dụng để cung cấp một nhịp đập hệ thống cho hệ điều hành thời gian thực RTOS hoặc để tạo một ngắt có tính chu kì hay đơn giản để tạo một khoảng delay có độ chính xác cao.

Ví dụ xây dựng chương trình sử dụng Systick để tạo một ngắt định kỳ với chu kỳ là 1ms. Để định kỳ hệ thống ngắt sau 1ms cần cấu hình Systick hoạt động với thời gian 1ms theo lệnh như sau:

```
SysTick_Config(SystemCoreClock/1000);
```

SystemCoreClock là tần số hoạt động thực của vi điều khiển tương ứng với thời gian là 1 giây. Vậy để định kỳ 1ms chương trình thực hiện ngắt một lần cần cấu hình Systick với thời gian là thời gian hệ thống đã chia cho 1000. Chương trình cụ thể như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO

volatile char     vruc_SYSTICK_Flag = 0;

GPIO_InitTypeDef GPIO_InitStructure;
void Fn_GPIO_Init (void);

int main (void){
      SystemInit();
      SystemCoreClockUpdate();

      SysTick_Config(SystemCoreClock/1000);
      Fn_GPIO_Init();
      while(1){
            if(vruc_SYSTICK_Flag == 1){
                  vruc_SYSTICK_Flag = 0;
                  if    (GPIO_ReadOutputDataBit(GPIOA,GPIO_Pin_0)    ==
Bit_RESET){
                        GPIO_WriteBit(GPIOA,GPIO_Pin_0, Bit_SET);
                  }
                  else{
                        GPIO_WriteBit(GPIOA,GPIO_Pin_0, Bit_RESET);
                  }
            }
      }
}

void Fn_GPIO_Init (void){
```

```
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOA, &GPIO_InitStructure);
}

void SysTick_Handler(void){
	vruc_SYSTICK_Flag = 1;
}
```

Hàm ngắt của Systick có tên là SysTick_Handler đã được quy định sẵn trong file stm32f10x.h và gắn với bảng vector ngắt của hệ thống. Mỗi khi đủ 1ms hàm ngắt được gọi 1 lần và thực hiện gán giá trị của biến vruc_SYSTICK_Flag bằng 1. Ở hàm main nếu kiểm tra biến vruc_SYSTICK_Flag bằng 1 thì chương trình thực hiện đảo trạng thái của đèn LED tại cổng A0. Do vậy chương trình thực hiện liên tục đổi trạng thái của chân A0 sau định kỳ 1ms.

Ở ví dụ này giáo trình sử dụng Systick để tạo một thư viện Delay với độ chính xác cao và sử dụng xuyên suốt giáo trình. Cụ thể nội dung của thư viện như sau:

```
#include "stm32f10x.h"
#include "Delay.h"

static u8 fac_us=0;
static u16 fac_ms=0;

void Fn_DELAY_Init (unsigned char _CLK)
{
	SysTick->CTRL&=0xfffffffb;
	fac_us=_CLK/8;
	fac_ms=(u16)fac_us*1000;
}
```

```
void Fn_DELAY_ms (unsigned int _vrui_Time)
{
	u32 temp;
	SysTick->LOAD=(u32)_vrui_Time*fac_ms;
	SysTick->VAL =0x00;
	SysTick->CTRL=0x01;
	do
	{
		temp=SysTick->CTRL;
	}
	while((temp&0x01)&&(!(temp&(1<<16))));
	SysTick->CTRL=0x00;
	SysTick->VAL =0X00;
}

void Fn_DELAY_us (unsigned long _vrui_Time)
{
	u32 temp;
	SysTick->LOAD=_vrui_Time*fac_us;
	SysTick->VAL=0x00;
	SysTick->CTRL=0x01 ;
	do
	{
		temp=SysTick->CTRL;
	}
	while((temp&0x01)&&(!(temp&(1<<16))));
	SysTick->CTRL=0x00;
	SysTick->VAL =0X00;
}
```

Trong đó hàm `Fn_DELAY_Init` khởi tạo sử dụng delay sử dụng bộ Systick với thông số đầu vào là tần số hoạt động của vi điều khiển.

Với thư viện này chúng ta có thể tạo được chương trình nhấp nháy đèn LED định kỳ 1 giây như sau:

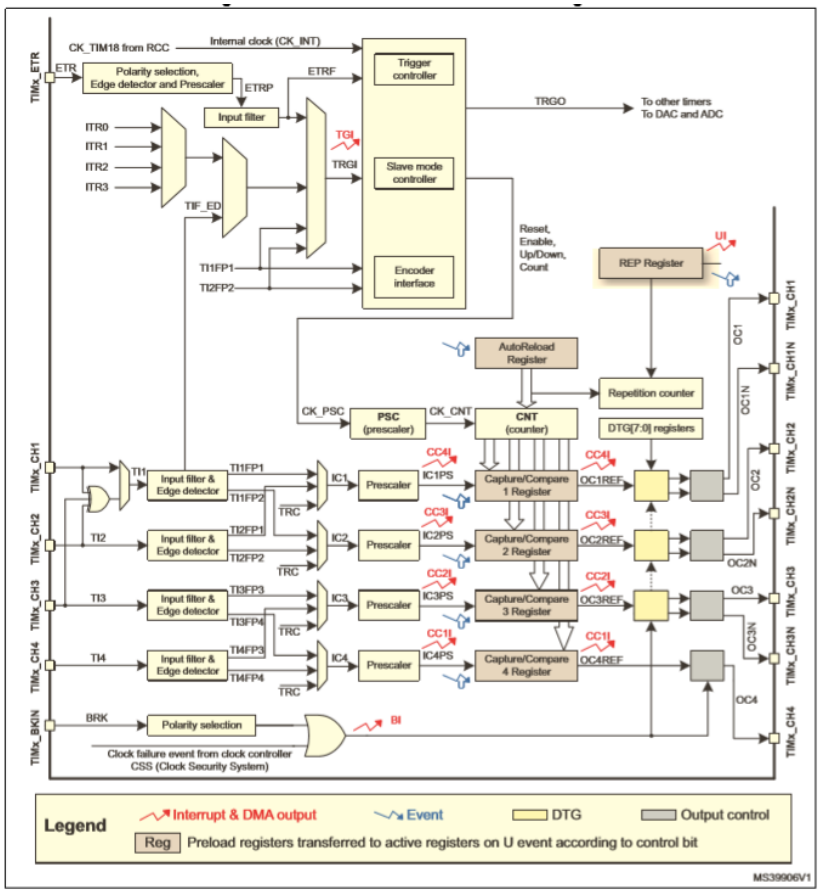
```
#include "stm32f10x.h" // Device header
#include     "stm32f10x_gpio.h"     //     Keil::Device:StdPeriph
Drivers:GPIO
#include "Delay.h"

GPIO_InitTypeDef GPIO_InitStructure;
void Fn_GPIO_Init (void);

int main (void){
     SystemInit();
     SystemCoreClockUpdate();
     Fn_DELAY_Init(72);
     Fn_GPIO_Init();
     while(1){
          if(   GPIO_ReadOutputDataBit(GPIOA,   GPIO_Pin_0)   ==
Bit_SET){
               GPIO_WriteBit(GPIOA, GPIO_Pin_0, Bit_RESET);
          }
          else{
               GPIO_WriteBit(GPIOA, GPIO_Pin_0, Bit_SET);
          }
          Fn_DELAY_ms(1000);
     }
}

void Fn_GPIO_Init (void){
     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
```

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

Sau khi cấu hình xong thời gian các hàm Delay sẽ thực hiện đúng với thời gian yêu cầu do Systick là một hệ thống thời gian chính xác của vi điều khiển luôn chạy song song với chương trình phần mềm.

Lập trình điều khiển Timer

Cấu trúc các bộ Timer

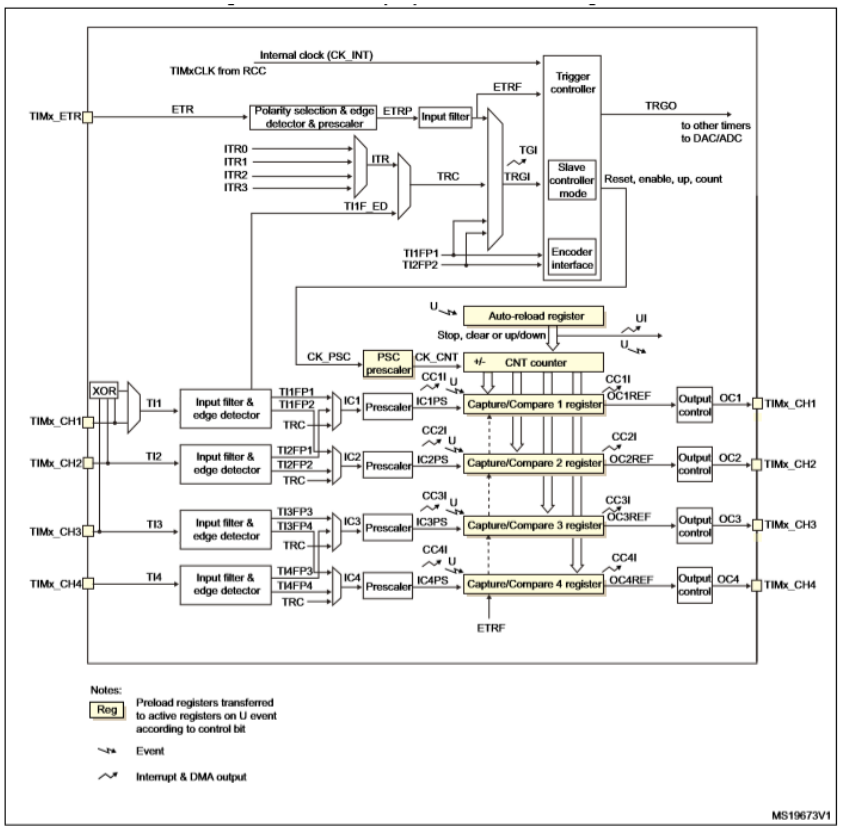


Hình 5. 26: Cấu trúc bộ Timer Advanced-Control

Bộ Timer Advanced-Control bao gồm TIM1 và TIM8 đối với dòng vi điều khiển STM32F1. Các bộ timer này có một số đặc điểm như sau:

- Bộ đếm 16 bit có thể đếm tiến, lùi, tiến/lùi tự động nạp lại

- Hỗ trợ bộ chia tần 16 bit có thể lập trình được.
- Hỗ trợ 4 kênh hoạt động độc lập cho các chức năng:
    - Caputre đầu vào
    - So sánh đầu ra
    - Điều chế độ rộng xung (PWM)
    - Chế độ đầu ra một xung
- Hỗ trợ kích hoạt các sự kiện DMA
- Hỗ trợ kích hoạt ngắt
- Hỗ trợ trigger cho các sự kiện



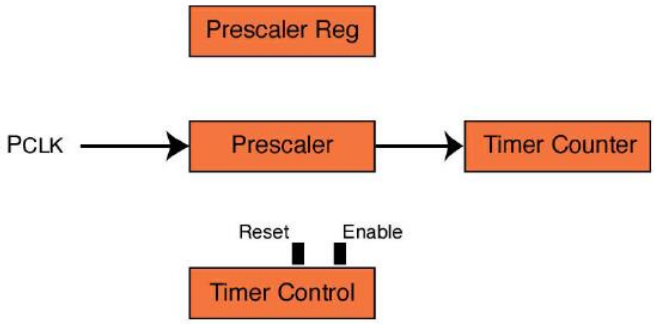
Hình 5. 27: Cấu trúc bộ Timer General-purpose

Bộ Timer Advanced-Control bao gồm TIM2 đến TIM5 đối với dòng vi điều khiển STM32F1. Các bộ timer này có một số đặc điểm như sau:

- Bộ đếm 16 bit có thể đếm tiến, lùi, tiến/lùi tự động nạp lại
- Hỗ trợ bộ chia tần 16 bit có thể lập trình được.
- Hỗ trợ 4 kênh hoạt động độc lập cho các chức năng:

- Caputre đầu vào
- So sánh đầu ra
- Điều chế độ rộng xung (PWM)
- Chế độ đầu ra một xung

- Hỗ trợ kích hoạt các sự kiến DMA
- Hỗ trợ kích hoạt ngắt
- Hỗ trợ trigger cho các sự kiện
- Hỗ trợ đọc cảm biến hall 4 đầu vào



Hình 5. 28: Hoạt động của Timer

Một Timer cở bản gồm các thành phần như sau:

TIM_CLK : clock cung cấp cho timer.

PSC (prescaler): là thanh ghi 16bits làm bộ chia cho timer, có thể chia từ 1 tới 65535

ARR (auto-reload register): là giá trị đếm của timer (16bits hoặc 32bits).

RCR (repetition counter register): giá trị đếm lặp lại 16bits

Timer của STM32 là timer 16 bits có thể tạo ra cácsự kiện trong khoảng thời gian từ nano giây tới vài phút gọi là UEV(update event).

Giá trị của UEV được tính theo công thức sau:

$$UEV = TIM\_CLK/((PSC + 1)*(ARR + 1)*(RCR + 1))$$

Ở giáo trình này dừng lại ở đưa ra một ví dụ sử dụng Timer ở tính năng chính là đếm thời gian sử dụng ngắt và không sử dụng ngắt. Chương trình được xây dựng như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "stm32f10x_tim.h" // Keil::Device:StdPeriph Drivers:TIM
```

```
#include "Delay.h"

GPIO_InitTypeDef GPIO_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

void Fn_GPIO_Init (void);
void Fn_NVIC_Init (void);
void Fn_TIM4_Init (void);
void Fn_TIM2_Init (void);

int main (void){
	SystemInit();
	SystemCoreClockUpdate();
	Fn_DELAY_Init(72);

	Fn_GPIO_Init();
	Fn_TIM4_Init();
	Fn_TIM2_Init();
	Fn_NVIC_Init();
	while(1){
		if(TIM_GetITStatus(TIM2,TIM_IT_Update) != RESET){
			if(   GPIO_ReadOutputDataBit(GPIOB,   GPIO_Pin_1)   ==
Bit_SET){
				GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
			}
			else{
				GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_SET);
			}
			TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
		}
```

```
	}
}

void Fn_GPIO_Init (void){
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOB, &GPIO_InitStructure);
}

void Fn_NVIC_Init (void){
	NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStructure);
}

void Fn_TIM2_Init (void){
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
	TIM_TimeBaseStructure.TIM_Prescaler = 20-1;
	TIM_TimeBaseStructure.TIM_Period = 50000;             //
	TIM_TimeBaseStructure.TIM_ClockDivision = 0;
	TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
	TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
	TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);
	TIM_Cmd(TIM2, ENABLE);
}

void Fn_TIM4_Init (void){
```

```
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
	TIM_TimeBaseStructure.TIM_Prescaler                                    =
((SystemCoreClock/2)/1000000)-1; // TIM2,3,4 Chia tan 2, 1 chia tan 1
	TIM_TimeBaseStructure.TIM_Period = 1000 - 1;
	TIM_TimeBaseStructure.TIM_ClockDivision = 0;
	TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
	TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
	TIM_ITConfig(TIM4,TIM_IT_Update,ENABLE);
	TIM_Cmd(TIM4, ENABLE);
}

void TIM4_IRQHandler(void){
	if (TIM_GetITStatus(TIM4, TIM_IT_Update) != RESET){
		if( GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_0) == Bit_SET){
			GPIO_WriteBit(GPIOB, GPIO_Pin_0, Bit_RESET);
		}
		else{
			GPIO_WriteBit(GPIOB, GPIO_Pin_0, Bit_SET);
		}
		TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
	}
}
```

Chương trình sử dụng 2 Timer là TIM2 và TIM4 trong đó TIM4 hoạt động với ngắt và TIM2 hoạt động không sử dụng ngắt.

Với TIM4 mỗi khi giá trị đếm tràn so với giá trị mong muốn chương trình thực hiện gọi một ngắt thực hiện đảo trạng thái của đèn LED tại chân B0.

Với TIM2 do không sử dụng ngắt nên chương trình cần liên tục kiểm tra trạng thái của cờ tràn của bộ đếm. Khi cờ tràn xảy ra chương trình phần mềm sẽ thực hiện đảo trạng thái tại chân B1.

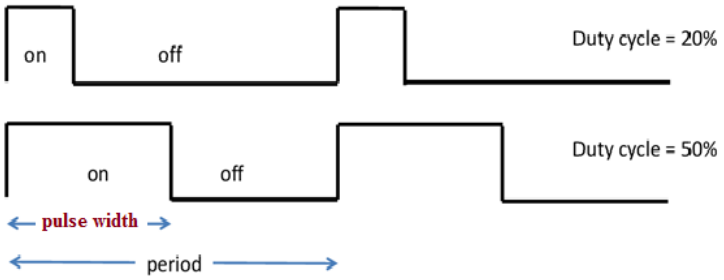
Để khởi tạo và sử dụng Timer sử dụng thư viện SPL cần khai báo thư viện stm32f10x_tim.h trong đó có một struct quan trọng để khởi tạo cấu hình Timer có tên là: TIM_TimeBaseInitTypeDef.

Để khởi tạo cho một Timer cần nạp giá trị đầy đủ cho một biến dữ liệu có kiểu là TIM_TimeBaseInitTypeDef sau đó nạp giá trị đó vào cho bộ Timer cần sử dụng. Cụ thể các giá trị cần quan tâm được liệt kê ở bảng dưới đây. Tùy thuộc vào mục đích sử dụng, thời gian đếm cho mỗi chu kỳ mà cần có những tính toán phù hợp để cấu hình cho Timer hoạt động.

TIM_Prescaler	Hệ số chia tần của Timer. Ví dụ muốn chia tần 1000 thì cần nạp vào giá trị là 199
TIM_Period	Chu kỳ đếm
TIM_ClockDivision	Giá trị chia tần nguồn clock cung cấp cho Timer.
TIM_CounterMode	Chế độ đếm: Up hoặc Down

Lập trình điều chế độ rộng xung - PWM (Pulse-width modulation)

Bộ điều chế xung, hay còn gọi là bộ "*băm xung*" là bộ xử lý và điều khiển tạo ra dạng xung vuông chu kỳ thay đổi theo cấu hình. Đây là phương pháp điều chỉnh điện áp ra tải dựa vào trung bình tín hiệu điều chế. Khi độ rộng xung tăng, trung bình điện áp ra tăng. Các module PWM thường sử dụng tần số điều chế không đổi, và điều chỉnh dựa trên sự thay đổi của chu kỳ nhiệm vụ (Duty Cycle).



Hình 5. 29: Điều chế độ rộng xung PWM

– Duty cycle là tỷ lệ phần trăm mức cao.

– Period là chu kỳ xung.

– Pulse width là giá trị mức cao so với period.

PWM được ứng dụng nhiều trong điều khiển như điều khiển động cơ và các bộ xung áp, điều áp… Sử dụng PWM điều khiển độ nhanh chậm của động cơ và sựổn định tốc độ động cơ. Ngoài lĩnh vực điều khiển hay ổn định tải thì PWM còn tham gia và điều chế các mạch nguồn như : boot, buck, nghịch lưu 1 pha và 3 pha… PWM chuyên dùng để điều khiển các phần tử điện tử công suất có đường đặc tính là tuyến tính khi có sẵn 1 nguồn 1 chiều cố định.

Như hình vẽ bên trên và theo công thức Duty cycle = pulse width x 100/period. Ta thấy pwm có 2 thành phần chính để tạo ra duty cycle của pwm là : độ rộng của 1 chu kỳ xung (period) và giá trị của xung cao là bao nhiêu so với period (pulse width). Do đó để cấu hình sử dụng pwm ta có thể chia làm 2 phần như sau :

- Cấu hình timer cho pwm: quyết định độ rộng của 1 chu kỳ xung pwm là bao nhiêu (period)
- Cấu hình pwm: quyết định phần trăm của xung mức cao là bao nhiêu phần trăm (pulse width)

PWM là một chức năng tích hợp của bộ Timer vì vậy để sử dụng PWM cần khai báo thư viện timer. Cụ thể là thư viện: stm32f10x_tim.h.

Để sử dụng PWM trong Timer cần tìm hiểu 2 kiểu dữ liệu dạng struct sau:

TIM_TimeBaseInitTypeDef: Khai báo thông tin hoạt động cơ bản của Timer

TIM_OCInitTypeDef: Khai báo thông tin hoạt động của bộ PWM

Trong đó đoạn chương trình nạp thông tin cấu hình cho bộ PWM bao gồm 2 đoạn: Cấu hình cho Timer điều khiển PWM và cấu hình cho hoạt động của bộ PWM. Mỗi Timer hỗ trợ 4 bộ PWM vì vậy tùy thuộc vào nhu cầu sử dụng mà người lập trình thiết lập và cấu hình số chân cần điều khiển phù hợp. Chương trình sau viết cho Timer 1 và thực hiện cấu hình cho cả 4 kênh PWM.

Chương trình cụ thể như sau:

```
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_OCInitTypeDef TIM_OCInitStructure;


RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_Period = PWM_DUTY;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);


TIM_OCStructInit(&TIM_OCInitStructure);
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
```

```
TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0;

TIM_OC1Init(TIM1, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);

TIM_OC2Init(TIM1, &TIM_OCInitStructure);
TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Enable);

TIM_OC3Init(TIM1, &TIM_OCInitStructure);
TIM_OC3PreloadConfig(TIM1, TIM_OCPreload_Enable);

TIM_OC4Init(TIM1, &TIM_OCInitStructure);
TIM_OC4PreloadConfig(TIM1, TIM_OCPreload_Enable);

TIM_ARRPreloadConfig(TIM1, ENABLE);
TIM_Cmd(TIM1, ENABLE);
TIM_CtrlPWMOutputs(TIM1, ENABLE);
```

Trong quá trình sử dụng chương trình có thể thay đổi độ rộng xung của từng kênh một cách độc lập sử dụng thanh ghi CCR1, CCR2, CCR3, CCR4 tương ứng của bộ Timer. Khi nạp lại giá trị vào thanh ghi này giá trị độ rộng xung sẽ tự động thay đổi.

Lưu ý: Giá trị của các thanh ghi CCRx không được lớn hơn giá trị Period đã nạp trước đó.

Dưới đây là một đoạn chương trình dùng để cập nhật độ rộng xung cho cả 4 kênh PWM của một bộ Timer:

```
#define PWM_DUTY  (50000)

TIM1->CCR1 = 10 * (PWM_DUTY / 100); //10% Duty cycle
TIM1->CCR2 = 30 * (PWM_DUTY / 100); //30% Duty cycle
TIM1->CCR3 = 60 * (PWM_DUTY / 100); //60% Duty cycle
TIM1->CCR4 = 90 * (PWM_DUTY / 100); //90% Duty cycle
```

Hơn nữa, việc điều chế độ rộng xung cần thông qua các cổng của vi điều khiển và vi điều khiển STM32 có những yêu cầu đặc biệt về chế độ của cổng khi sử dụng với các bộ ngoại vi bên trong. Cụ thể với Timer vi điều khiển yêu cầu thông tin cấu hình của các cổng tương ứng như sau:

Table 22. Advanced timers TIM1 and TIM8

TIM1/8 pinout	Configuration	GPIO configuration
TIM1/8_CHx	Input capture channel x	Input floating
	Output compare channel x	Alternate function push-pull
TIM1/8_CHxN	Complementary output channel x	Alternate function push-pull
TIM1/8_BKIN	Break input	Input floating
TIM1/8_ETR	External trigger timer input	Input floating

Table 23. General-purpose timers TIM2/3/4/5

TIM2/3/4/5 pinout	Configuration	GPIO configuration
TIM2/3/4/5_CHx	Input capture channel x	Input floating
	Output compare channel x	Alternate function push-pull
TIM2/3/4/5_ETR	External trigger timer input	Input floating

Hình 5. 30: Thông tin cấu hình PWM cho các Timer

PWM hoạt động dựa trên chế độ Output Compare Channel do vậy để các chân tương ứng hoạt động ở chế độ PWM cần cấu hình các chân đó ở chế độ hoạt động Alternate function push-pull. Có thể tham khảo phần lập trình điều khiển IO để nắm lại phần kiến thức này.

Chương trình đầy đủ điều khiển 4 bộ PWM cụ thể như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "stm32f10x_tim.h" // Keil::Device:StdPeriph Drivers:TIM

GPIO_InitTypeDef GPIO_InitStructure;
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_OCInitTypeDef TIM_OCInitStructure;

#define PWM_DUTY  (50000)

void TIM_PWM_Configuration(void);
```

```
int main (void){
	SystemInit();
	SystemCoreClockUpdate();

	TIM_PWM_Configuration();
	TIM1->CCR1 = 10 * (PWM_DUTY / 100); //10% Duty cycle
	TIM1->CCR2 = 30 * (PWM_DUTY / 100); //30% Duty cycle
	TIM1->CCR3 = 60 * (PWM_DUTY / 100); //60% Duty cycle
	TIM1->CCR4 = 90 * (PWM_DUTY / 100); //90% Duty cycle
	while(1);
}

void TIM_PWM_Configuration(void){
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

	GPIO_InitStructure.GPIO_Pin   =   GPIO_Pin_8   |   GPIO_Pin_9   |
GPIO_Pin_10 | GPIO_Pin_11;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
	GPIO_Init(GPIOA, &GPIO_InitStructure);

	TIM_TimeBaseStructure.TIM_Prescaler = 0;
	TIM_TimeBaseStructure.TIM_Period = PWM_DUTY;
	TIM_TimeBaseStructure.TIM_ClockDivision = 0;
	TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
	TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);

	TIM_OCStructInit(&TIM_OCInitStructure);
	TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
	TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
```

```
	TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
	TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
	TIM_OCInitStructure.TIM_Pulse = 0;

	TIM_OC1Init(TIM1, &TIM_OCInitStructure);
	TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);

	TIM_OC2Init(TIM1, &TIM_OCInitStructure);
	TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Enable);

	TIM_OC3Init(TIM1, &TIM_OCInitStructure);
	TIM_OC3PreloadConfig(TIM1, TIM_OCPreload_Enable);

	TIM_OC4Init(TIM1, &TIM_OCInitStructure);
	TIM_OC4PreloadConfig(TIM1, TIM_OCPreload_Enable);

	TIM_ARRPreloadConfig(TIM1, ENABLE);
	TIM_Cmd(TIM1, ENABLE);
	TIM_CtrlPWMOutputs(TIM1, ENABLE);
}
```

Lập trình điều khiển bộ 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.

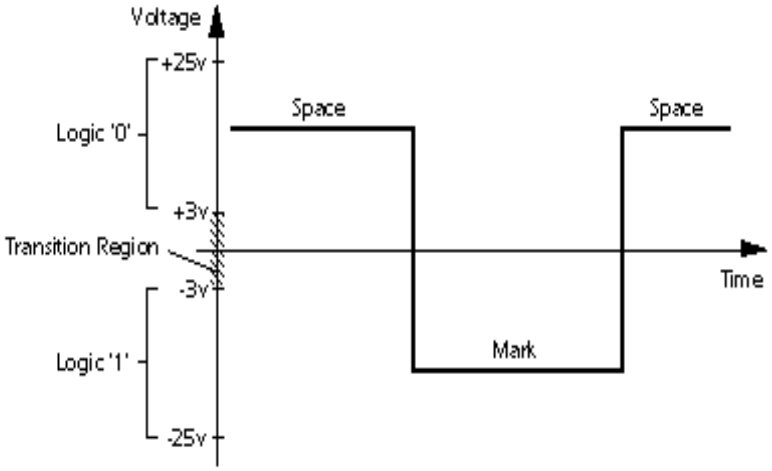
Trong 8051 có bộ thu phát bất đồng bộ tổng hợp UART(Universal Asynchronous Receriver Transmister).

Trong quá trình truyền, dữ liệu vừa có thể phát vừa có thể thu được gọi là truyền song công, trái lại là truyền đơn công (ví dụ máy in: máy tính chỉ phát dữ liệu).

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.

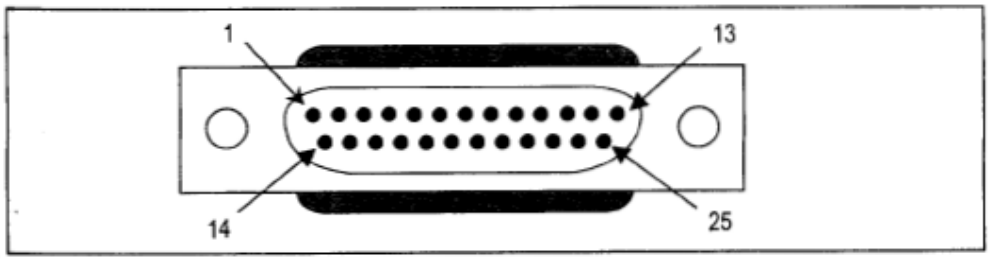
RS-232 định nghĩa mức điện thế mà tương ứng với các mức logic 0 và 1. Tín hiệu có mức điện thế như sau : mức logic 0: từ +3V đến +25V, mức logic 1 : từ -25V đến -3V.



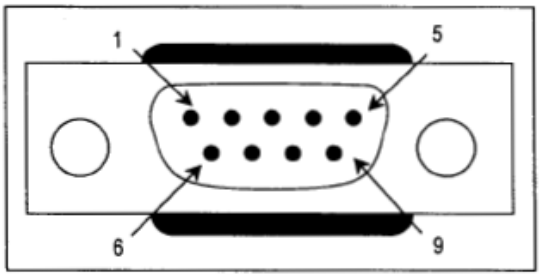
Hình 5. 31: Mức điện thế biểu diễn tín hiệu nhị phân

Về bố trí chân của RS232

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



Hình 5. 32: Cổng DB25



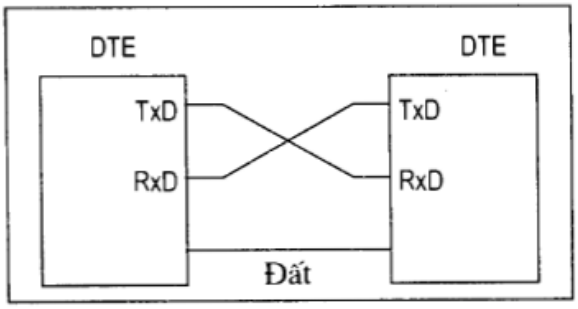
Hình 5. 33: Cổng DB-9

Hiện tại DB-25 ít được sử dụng nên ta chỉ xem xét cách bố trí chân đấu nối của 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

Bảng 5. 1: Bảng các chân của cổng DB-9

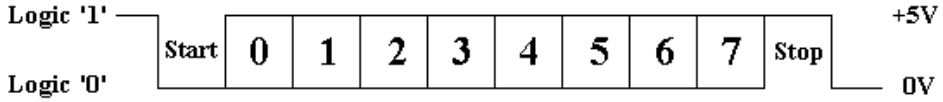
Một kết nối đơn giản nhất giữa PC và bộ vi điều khiển yêu cầu tối thiểu 3 chân TxD, RxD, GND như trong hình vẽ dưới đây :



Hình 5. 34: Kết nối tối thiểu giữa PC với vi điều khiển trên giao tiếp UART

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. 35: 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

Hình 5. 36: Thông tin các chân giao tiếp UART

STM32F1 có 3 bộ UART với nhiều mode hoạt động, với nhiều bộ UART ta có thể sử dụng được nhiều ứng dụng. Một số tính năng nổi bật như sau:

- Đầy đủ các tính năng của bộ giao tiếp không đồng bộ.
- Điều chỉnh baud rate bằng lập trình và tốc độ tối đa lên đến 4.5Mb/s.
- Độ dài được lập trình là 8 hoặc 9 bit.
- Cấu hình bit stop hỗ trợ là 1 hoặc 2.
- Có chân clock nếu muốn chuyển giao tiếp thành đồng bộ.

- Cấu hình sử dụng 1 dây hoặc 2 dây.
- Có bộ DMA nếu muốn đẩy cao thời gian truyền nhận.
- Bit cho phép truyền nhận riêng biệt.

Để lập trình UART với thư viện SPL chúng ta cần tiến hành thêm thư viện stm32f10x_usart.h vào dự án bằng cách sử dụng Manage Run như hướng dẫn ở trên. Thư viện này cung cấp một kiểu dữ liệu nhằm đơn giản hóa việc khai báo, cấu hình và sử dụng bộ USART của STM32F1. Cụ thể:

```
typedef struct
{
 uint32_t USART_BaudRate; /*!< This member configures the USART communication baud rate.
 The baud rate is computed using the following formula:
 - IntegerDivider = ((PCLKx) / (16 * (USART_InitStruct->USART_BaudRate)))
 - FractionalDivider = ((IntegerDivider - ((u32) IntegerDivider)) * 16) + 0.5 */

 uint16_t USART_WordLength; /*!< Specifies the number of data bits transmitted or received in a frame.
 This parameter can be a value of @ref USART_Word_Length */

 uint16_t USART_StopBits; /*!< Specifies the number of stop bits transmitted.
 This parameter can be a value of @ref USART_Stop_Bits */

 uint16_t USART_Parity; /*!< Specifies the parity mode.
 This parameter can be a value of @ref USART_Parity
 @note When parity is enabled, the computed parity is inserted
 at the MSB position of the transmitted data (9th bit when
 the word length is set to 9 data bits; 8th bit when the
 word length is set to 8 data bits). */

 uint16_t USART_Mode; /*!< Specifies wether the Receive or Transmit
```

```
mode is enabled or disabled.
 This parameter can be a value of @ref USART_Mode */


 uint16_t USART_HardwareFlowControl; /*!< Specifies wether the hardware
flow control mode is enabled
 or disabled.
 This parameter can be a value of @ref USART_Hardware_Flow_Control */
} USART_InitTypeDef;
```

Để khởi động bộ USART của STM32F1 cần thực hiện nạp đầy đủ các thông số của biến có kiểu dữ liệu USART_InitTypeDef sau đó nạp cấu hình đó vào bộ USART cần sử dụng. Cụ thể các giá trị của kiểu dữ liệu này được giải thích như sau:

USART_BaudRate	Tốc độ Baudrate cần sử dụng
USART_WordLength	Độ dài dữ liệu mỗi khung truyền
USART_StopBits	Chế độ stop bit cần sử dụng
USART_Parity	Sử dụng bộ chẵn lẻ để phát hiện sai
USART_HardwareFlowControl	Chế độ sử dụng phần cứng theo dõi hay không?
USART_Mode	Chế độ sử dụng với USART: Truyền, nhận

Tương tự như bộ PWM, bộ USART cũng yêu cầu các cổng giao tiếp cần hoạt động ở các chế độ đặc biệt. Thông tin yêu cầu của bộ USART về cổng kết nối cụ thể như sau:

Table 24. USARTs

USART pinout	Configuration	GPIO configuration
USARTx_TX(1)	Full duplex	Alternate function push-pull
	Half duplex synchronous mode	Alternate function push-pull
USARTx_RX	Full duplex	Input floating / Input pull-up
	Half duplex synchronous mode	Not used. Can be used as a general IO
USARTx_CK	Synchronous mode	Alternate function push-pull
USARTx_RTS	Hardware flow control	Alternate function push-pull
USARTx_CTS	Hardware flow control	Input floating/ Input pull-up

Hình 5. 37: Cấu hình thông tin các chân của giao tiếp UART

Với 2 chân sử dụng chính trong USART là RX và TX vi điều khiển yêu cầu được cấu hình ở 2 chế độ sau: Alternate function push-pull và Input floating/Input pull-up.

Chương trình sau được viết để sử dụng bộ USART1 hoạt động ở cả 2 chế độ RX và TX. Trong đó vi điều khiển thực hiện định kỳ truyền sau mỗi 100ms và thực hiện dữ liệu và lưu trữ vào biến vrsc_UART_ReceiveTemp.

Cụ thể chương trình như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "stm32f10x_usart.h" // Keil::Device:StdPeriph Drivers:USART
#include "Delay.h"

USART_InitTypeDef USART_InitStructure;
GPIO_InitTypeDef GPIO_InitStructure;

volatile char vrsc_UART_ReceiveTemp;
volatile char vr_UART_Buffer[1024];
volatile char vr_UART_Count;
volatile char vr_UART_RXDone;

void Fn_UART_Init (unsigned int _vrui_BaudRate);
void Fn_UART_SendChar (char _vrsc_Char);
void Fn_UART_PutStr (char *_vrsc_String);
char Fn_UART_Recieve (void);

int main (void){
      SystemInit();
      SystemCoreClockUpdate();
      Fn_DELAY_Init(72);
      Fn_UART_Init(115200);
      while(1){
```

```
            Fn_UART_PutStr("Hello!\n");
            Fn_DELAY_ms(100);
      }
}

void Fn_UART_Init (unsigned int _vrui_BaudRate){
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA    |    RCC_APB2ENR_AFIOEN,
ENABLE);

      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
 GPIO_Init(GPIOA, &GPIO_InitStructure);

 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
 GPIO_Init(GPIOA, &GPIO_InitStructure);

 USART_InitStructure.USART_BaudRate = _vrui_BaudRate;
 USART_InitStructure.USART_WordLength = USART_WordLength_8b;
 USART_InitStructure.USART_StopBits = USART_StopBits_1;
 USART_InitStructure.USART_Parity = USART_Parity_No;
 USART_InitStructure.USART_HardwareFlowControl                          =
USART_HardwareFlowControl_None;
 USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
 USART_Init(USART1, &USART_InitStructure);
      USART_ITConfig(USART1, USART_IT_RXNE ,ENABLE);// Cho phep ngat
      USART_Cmd(USART1, ENABLE);
}
```

```
void Fn_UART_SendChar (char _vrsc_Char){
	USART_SendData(USART1, _vrsc_Char);
 while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET); //Cho
den khi gui di xong
}

void Fn_UART_PutStr (char *_vrsc_String){
	while(*_vrsc_String){
		Fn_UART_SendChar(*_vrsc_String);
		_vrsc_String++;
	}
}

char Fn_UART_Recieve (void){
	while (USART_GetFlagStatus(USART1, USART_IT_RXNE) == RESET); //
Cho co nhan Reset
 return (USART_ReceiveData(USART1));
}

void USART1_IRQHandler (void){
	if(USART_GetFlagStatus(USART1, USART_IT_RXNE) != RESET){
		vrsc_UART_ReceiveTemp = USART_ReceiveData(USART1);

	}
	USART_ClearITPendingBit(USART1,USART_IT_RXNE);
}
```

Lập trình điều khiển SPI

SPI (Serial Peripheral Bus) là một chuẩn truyền thông nối tiếp tốc độ cao do hang Motorola đề xuất. Đây là kiểu truyền thông Master-Slave, trong đó có 1 chip Master điều phối quá trình tuyền thông và các chip Slaves được điều khiển bởi Master vì thế truyền thông chỉ xảy ra giữa Master và Slave. SPI là một cách truyền song công (full duplex)

nghĩa là tại cùng một thời điểm quá trình truyền và nhận có thể xảy ra đồng thời. SPI đôi khi được gọi là chuẩn truyền thông "4 dây" vì có 4 đường giao tiếp trong chuẩn này đó là SCK (Serial Clock), MISO (Master Input Slave Output), MOSI (Master Ouput Slave Input) và SS (Slave Select). Hình 1 thể hiện một kết SPI giữa một chip Master và 3 chip Slave thông qua 4 đường.

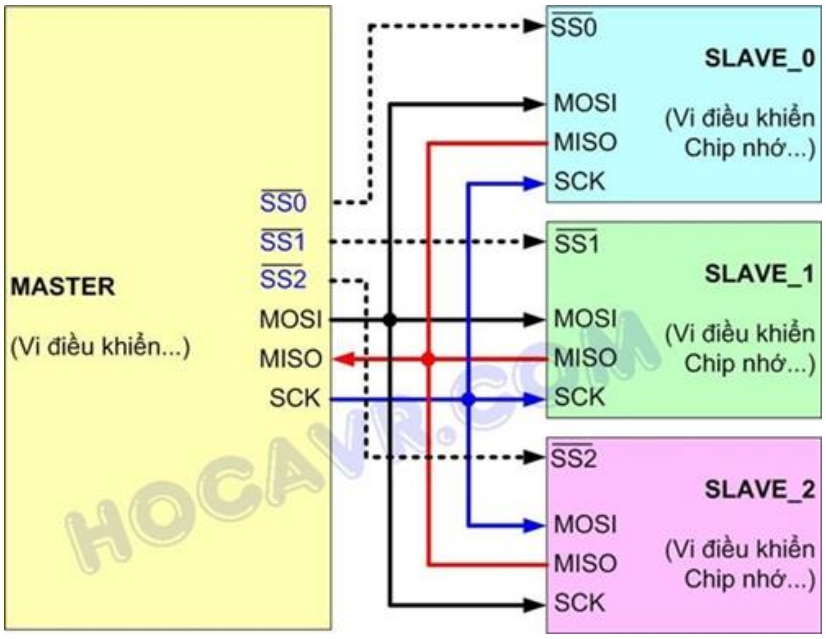
SCK: Xung giữ nhịp cho giao tiếp SPI, vì SPI là chuẩn truyền đồng bộ nên cần 1 đường giữ nhịp, mỗi nhịp trên chân SCK báo 1 bit dữ liệu đến hoặc đi. Đây là điểm khác biệt với truyền thông không đồng bộ mà chúng ta đã biết trong chuẩn UART. Sự tồn tại của chân SCK giúp quá trình tuyền ít bị lỗi và vì thế tốc độ truyền của SPI có thể đạt rất cao. Xung nhịp chỉ được tạo ra bởi chip Master.

MISO – Master Input / Slave Output: nếu là chip Master thì đây là đường Input còn nếu là chip Slave thì MISO lại là Output. MISO của Master và các Slaves được nối trực tiếp với nhau.

MOSI – Master Output / Slave Input: nếu là chip Master thì đây là đường Output còn nếu là chip Slave thì MOSI là Input. MOSI của Master và các Slaves được nối trực tiếp với nhau.

SS – Slave Select: SS là đường chọn Slave cần giap tiếp, trên các chip Slave đường SS sẽ ở mức cao khi không làm việc. Nếu chip Master kéo đường SS của một Slave nào đó xuống mức thấp thì việc giao tiếp sẽ xảy ra giữa Master và Slave đó. Chỉ có 1 đường SS trên mỗi Slave nhưng có thể có nhiều đường điều khiển SS trên Master, tùy thuộc vào thiết kế của người dùng.

Hoạt động: mỗi chip Master hay Slave có một thanh ghi dữ liệu 8 bits. Cứ mỗi xung nhịp do Master tạo ra trên đường giữ nhịp SCK, một bit trong thanh ghi dữ liệu của Master được truyền qua Slave trên đường MOSI, đồng thời một bit trong thanh ghi dữ liệu của chip Slave cũng được truyền qua Master trên đường MISO. Do 2 gói dữ liệu trên 2 chip được gửi qua lại đồng thời nên quá trình truyền dữ liệu này được gọi là "song công".



Hình 5. 38: Kết nối trong giao tiếp SPI

Cực của xung giữ nhịp, phase và các chế độ hoạt động: cực của xung giữ nhịp (Clock Polarity) được gọi tắt là CPOL là khái niệm dùng chỉ trạng thái của chân SCK ở trạng thái nghỉ. Ở trạng thái nghỉ (Idle), chân SCK có thể được giữ ở mức cao (CPOL=1) hoặc thấp (CPOL=0). Phase (CPHA) dùng để chỉ cách mà dữ liệu được lấy mẫu (sample) theo xung giữ nhịp. Dữ liệu có thể được lấy mẫu ở cạnh lên của SCK (CPHA=0) hoặc cạnh xuống (CPHA=1). Sự kết hợp của SPOL và CPHA làm nên 4 chế độ hoạt động của SPI. Nhìn chung việc chọn 1 trong 4 chế độ này không ảnh hưởng đến chất lượng truyền thông mà chỉ cốt sao cho có sự tương thích giữa Master và Slave.

Kiểu dữ liệu cấu trúc dùng để cấu hình SPI của STM32F1 có dạng như sau:

```
typedef struct
{
 uint16_t SPI_Direction; /*!< Specifies the SPI unidirectional or
bidirectional data mode.
 This parameter can be a value of @ref SPI_data_direction */

 uint16_t SPI_Mode; /*!< Specifies the SPI operating mode.
 This parameter can be a value of @ref SPI_mode */

 uint16_t SPI_DataSize; /*!< Specifies the SPI data size.
 This parameter can be a value of @ref SPI_data_size */
```

```
 uint16_t SPI_CPOL; /*!< Specifies the serial clock steady state.
 This parameter can be a value of @ref SPI_Clock_Polarity */

 uint16_t SPI_CPHA; /*!< Specifies the clock active edge for the bit capture.
 This parameter can be a value of @ref SPI_Clock_Phase */

 uint16_t SPI_NSS; /*!< Specifies whether the NSS signal is managed by
 hardware (NSS pin) or by software using the SSI bit.
 This parameter can be a value of @ref SPI_Slave_Select_management */

 uint16_t SPI_BaudRatePrescaler; /*!< Specifies the Baud Rate prescaler value which will be
 used to configure the transmit and receive SCK clock.
 This parameter can be a value of @ref SPI_BaudRate_Prescaler.
 @note The communication clock is derived from the master
 clock. The slave clock does not need to be set. */

 uint16_t SPI_FirstBit; /*!< Specifies whether data transfers start from MSB or LSB bit.
 This parameter can be a value of @ref SPI_MSB_LSB_transmission */

 uint16_t SPI_CRCPolynomial; /*!< Specifies the polynomial used for the CRC calculation. */
}SPI_InitTypeDef;
```

Trong đó mỗi trường quy định thông tin cụ thể theo bảng sau:

SPI_Mode	Chế độ hoạt động Master hoặc Slave
SPI_DataSize	Kích thước dữ liệu 8 bit hoặc 16 bit
SPI_CPOL	Cấu hình CPOL
SPI_CPHA	Cấu hình CPHA

SPI_NSS	Chế độ chip Select (Cứng hoặc mềm)
SPI_BaudRatePrescaler	Tốc độ truyền dữ liệu
SPI_FirstBit	Bit đầu tiên truyền (MSB/LSB)
SPI_CRCPolynomial	Chế độ kiểm tra lỗi CRC

Bảng 5. 2: Các thông tin cấu hình giao tiếp SPI

Với các thông tin trên chúng ta thiết lập một đoạn chương trình cấu hình SPI như sau:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
SPI_InitStruct.SPI_Mode = SPI_Mode_Master;
SPI_InitStruct.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStruct.SPI_CPOL = SPI_CPOL_High;
SPI_InitStruct.SPI_CPHA = SPI_CPHA_2Edge;
SPI_InitStruct.SPI_NSS = SPI_NSS_Soft;
SPI_InitStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_64;
SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStruct.SPI_CRCPolynomial = 7;
SPI_Init(SPI1, &SPI_InitStruct);
SPI_Cmd(SPI1, ENABLE);
SPI_EnableSlave();
```

Tương tự như các bộ ngoại vi khác SPI yêu cầu các cổng kết nối với ngoại vi SPI cần có cấu hình cổng như sau:

Table 25. SPI

SPI pinout	Configuration	GPIO configuration
SPIx_SCK	Master	Alternate function push-pull
	Slave	Input floating
SPIx_MOSI	Full duplex / master	Alternate function push-pull
	Full duplex / slave	Input floating / Input pull-up
	Simplex bidirectional data wire / master	Alternate function push-pull
	Simplex bidirectional data wire/ slave	Not used. Can be used as a GPIO
SPIx_MISO	Full duplex / master	Input floating / Input pull-up
	Full duplex / slave (point to point)	Alternate function push-pull
	Full duplex / slave (multi-slave)	Alternate function open drain
	Simplex bidirectional data wire / master	Not used. Can be used as a GPIO
	Simplex bidirectional data wire/ slave (point to point)	Alternate function push-pull
	Simplex bidirectional data wire/ slave (multi-slave)	Alternate function open drain
SPIx_NSS	Hardware master /slave	Input floating/ Input pull-up / Input pull-down
	Hardware master/ NSS output enabled	Alternate function push-pull
	Software	Not used. Can be used as a GPIO

Bảng 5. 3: Thông tin cấu hình các chân giao tiếp SPI

Chương trình khởi tạo SPI đầy đủ được viết như sau:

<SPI.h>

```
#ifndef _SPI_
#define _SPI_

#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
#include "stm32f10x_spi.h" // Keil::Device:StdPeriph Drivers:SPI

#define SPI_PORT        GPIOA
#define SPI_PIN_MOSI    GPIO_Pin_7
#define SPI_PIN_MISO    GPIO_Pin_6
#define SPI_PIN_SCK     GPIO_Pin_5
#define SPI_PIN_SS      GPIO_Pin_4
#define SPIx            SPI1

#ifdef __cplusplus
```

```
extern "C"{
#endif

void Fn_SPI_Init (void);
uint8_t Fn_SPIx_Transfer(uint8_t data);

#ifdef __cplusplus
}
#endif

#endif
```

<SPI.c>

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
#include "stm32f10x_spi.h"
#include "SPI.h"

void SPI_EnableSlave (void){
 // Set slave SS pin low
 //SPI_PORT->BRR = SPI_PIN_SS;
      GPIO_ResetBits(SPI_PORT, SPI_PIN_SS);
}

static void SPI_DisableSlave (void){
 // Set slave SS pin high
 //SPI_PORT->BSRR = SPI_PIN_SS;
      GPIO_SetBits(SPI_PORT, SPI_PIN_SS);
}
```

```
void Fn_SPI_Init (void){
      SPI_InitTypeDef SPI_InitStruct;
 GPIO_InitTypeDef GPIO_InitStruct;

      RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO                          |
RCC_APB2Periph_GPIOA, ENABLE);
 // GPIO pins for MOSI, MISO, and SCK
 GPIO_InitStruct.GPIO_Pin = SPI_PIN_MOSI | SPI_PIN_MISO | SPI_PIN_SCK;
 GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
 GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
 GPIO_Init(GPIOA, &GPIO_InitStruct);
 // GPIO pin for SS
 GPIO_InitStruct.GPIO_Pin = SPI_PIN_SS;
 GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
 GPIO_InitStruct.GPIO_Speed = GPIO_Speed_10MHz;
 GPIO_Init(GPIOA, &GPIO_InitStruct);

      RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
      SPI_InitStruct.SPI_Mode = SPI_Mode_Master;
      SPI_InitStruct.SPI_DataSize = SPI_DataSize_8b;
      SPI_InitStruct.SPI_CPOL = SPI_CPOL_High;
      SPI_InitStruct.SPI_CPHA = SPI_CPHA_2Edge;
      SPI_InitStruct.SPI_NSS = SPI_NSS_Soft;
      SPI_InitStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_64;
      SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB;
      SPI_InitStruct.SPI_CRCPolynomial = 7;
 SPI_Init(SPI1, &SPI_InitStruct);
 SPI_Cmd(SPI1, ENABLE);
      SPI_EnableSlave();
}
```

```
uint8_t Fn_SPIx_Transfer(uint8_t data)
{
      // Wait until transmit complete
      while ((SPIx->SR & (SPI_I2S_FLAG_TXE)) == RESET);
      // Write data to be transmitted to the SPI data register
      SPI_I2S_SendData(SPIx,data);
      // Wait until transmit complete
      while ((SPIx->SR & (SPI_I2S_FLAG_TXE))== RESET);
      // Wait until receive complete
      while (!(SPIx->SR & (SPI_I2S_FLAG_RXNE)));
      // Wait until SPI is not busy anymore
      while (SPIx->SR & (SPI_I2S_FLAG_BSY));
      // Return received data from SPI data register
      return SPIx->DR;
}
```

Như vậy với thư viện SPI bao gồm SPI.h và SPI.c chương trình có thể nhanh chóng thiết lập cấu hình SPI thông qua hàm Fn_SPI_Init và gửi dữ liệu, nhận dữ liệu thông qua hàm Fn_SPIx_Transfer.

Phần này của giáo trình ứng dụng SPI để điều khiển màn hình Graphic LCD có tên là PCD8544. Màn hình này được sử dụng cho điện thoại Nokia 5110. (Chương trình đầy đủ được đính kèm ở phụ lục của giáo trình).

Hình 5. 39: Màn hình PCD8544 sử dụng giao tiếp SPI

Thư viện điều khiển màn hình PCD8544 có các hàm chính như sau:

```
#ifndef LCD5110_Graph_h
#define LCD5110_Graph_h

#include <stdio.h>
#include <string.h>
#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
#include "stm32f10x_spi.h" // Keil::Device:StdPeriph Drivers:SPI
#include "Delay.h"
#include "SPI.h"

#define LEFT 0
#define RIGHT 9999
#define CENTER 9998

#define LCD_COMMAND 0
#define LCD_DATA 1
// PCD8544 Commandset
// ------------------
// General commands
#define PCD8544_POWERDOWN                0x04
#define PCD8544_ENTRYMODE                0x02
#define PCD8544_EXTENDEDINSTRUCTION 0x01
#define PCD8544_DISPLAYBLANK        0x00
#define PCD8544_DISPLAYNORMAL       0x04
#define PCD8544_DISPLAYALLON        0x01
#define PCD8544_DISPLAYINVERTED          0x05
// Normal instruction set
#define PCD8544_FUNCTIONSET              0x20
```

```
#define PCD8544_DISPLAYCONTROL          0x08
#define PCD8544_SETYADDR                0x40
#define PCD8544_SETXADDR                0x80
// Extended instruction set
#define PCD8544_SETTEMP                 0x04
#define PCD8544_SETBIAS                 0x10
#define PCD8544_SETVOP                  0x80
// Display presets
#define LCD_BIAS                        0x03
#define LCD_TEMP                        0x02
#define LCD_CONTRAST                    0x46
#define fontbyte(x) cfont.font[x]
#define bitmapbyte(x) bitmap[x]

#define byte      char

struct _current_font
{
      uint8_t* font;
      uint8_t x_size;
      uint8_t y_size;
      uint8_t offset;
      uint8_t numchars;
      uint8_t inverted;
};

class LCD5110
{
      public:
            LCD5110   (GPIO_TypeDef   *_SPI_Port,int   SCK,   int   MOSI,
GPIO_TypeDef *_LCD_Port, int DC, int RST, int CS);
```

```
            void InitLCD(int contrast=LCD_CONTRAST);
            void setContrast(int contrast);
            void enableSleep();
            void disableSleep();
            void update();
            void clrScr();
            void fillScr();
            void invert(bool mode);
            void setPixel(uint16_t x, uint16_t y);
            void clrPixel(uint16_t x, uint16_t y);
            void invPixel(uint16_t x, uint16_t y);
            void invertText(bool mode);
            void print(char *st, int x, int y);
            void print(const char *st, int x, int y);
            void printNumI(long num, int x, int y, int length=0, char
filler=' ');
            void printNumF(double num, byte dec, int x, int y, char
divider='.', int length=0, char filler=' ');
            void setFont(uint8_t* font);
            void drawBitmap(int x, int y, uint8_t* bitmap, int sx, int
sy);
            void drawLine(int x1, int y1, int x2, int y2);
            void clrLine(int x1, int y1, int x2, int y2);
            void drawRect(int x1, int y1, int x2, int y2);
            void clrRect(int x1, int y1, int x2, int y2);
            void drawRoundRect(int x1, int y1, int x2, int y2);
            void clrRoundRect(int x1, int y1, int x2, int y2);
            void drawCircle(int x, int y, int radius);
            void clrCircle(int x, int y, int radius);


      protected:
```

```
            GPIO_TypeDef      *SPI_Port, *LCD_Port;
            int                     Pin_MOSI, Pin_SCK, Pin_CD, Pin_RST,
Pin_CS;
            _current_font     cfont;
            uint8_t                 scrbuf[504];
            bool              _sleep;
            int                     _contrast;

            int abs (int x);
            void _LCD_Write(unsigned char data, unsigned char mode);
            void _print_char(unsigned char c, int x, int row);
            void _convert_float(char *buf, double num, int width, byte
prec);
            void drawHLine(int x, int y, int l);
            void clrHLine(int x, int y, int l);
            void drawVLine(int x, int y, int l);
            void clrVLine(int x, int y, int l);
};

#endif
```

Lập trình 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.

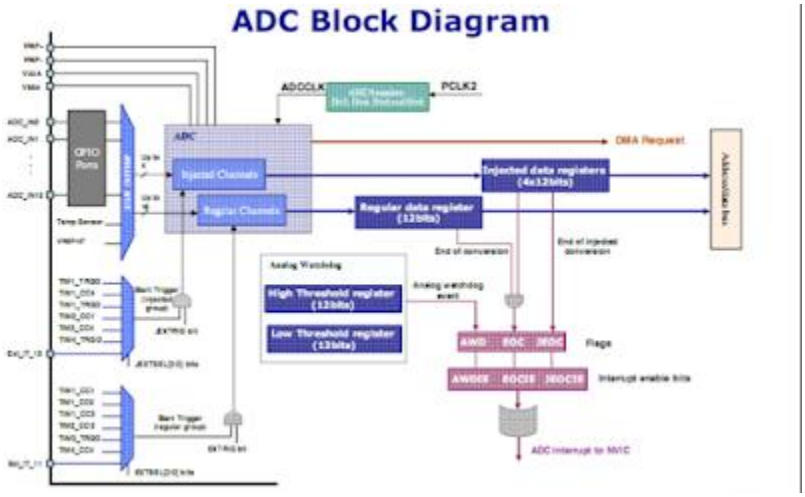
Độ chính xác càng cao và thời gian chuyển đổi càng nhanh thì bộ ADC có chất lượng càng tốt.

ADC trong STM32F1 là bộ ADC có 12 bit tức là giá trị đọc về nằm trong khoảng 0 ->2^12= 4096. Giá trị điện áp đầu vào bộ ADC được cung cấp trên chân VDDA và thường lấy bằng giá trị cấp nguồn cho vi điều khiển VDD(+3V3). STM32F103C8 có 2 kênh ADC đó là ADC1 và ADC2, mỗi kênh có tối đa là 9 channel với nhiều mode hoạt động như: single, continuous,scan hoặc discontinuous. Kết quả chuyển đổi được lưu trữ trong thanh ghi 16 bit.

Một số đặc tính chính được liệt kê như sau:

- Độ phân giải 12 bit tương ứng với giá trị maximum là 4095.
- Có các ngắt hỗ trợ như End conversion, End of Injected Conversion and Analog Watchdog Event.
- Single mode hay Continuous mode.
- Tự động calib và có thể điều khiển hoạt động ADC bằng xung Trigger.
- Thời gian chuyển đổi nhanh : 1us tại tần số 65Mhz.
- Điện áp cung cấp cho bộ ADC là 2.4V -> 3.6V. Nên điện áp Input của thiết bị có ADC 2.4V ≤ VIN ≤ 3.6V.
- Có bộ DMA giúp tăng tốc độ xử lí.

Sơ đồ khối của bộ ADC trong STM32F1 được mô tả trong hình sau:



Hình 5. 40: Sơ đồ khối bộ ADC

Cấu trúc biến khởi tạo ADC cho STM32F1 có dạng như sau:

```
typedef struct
{
 uint32_t ADC_Mode; /*!< Configures the ADC to operate in independent or
 dual mode.
 This parameter can be a value of @ref ADC_mode */

 FunctionalState ADC_ScanConvMode; /*!< Specifies whether the conversion is performed in
 Scan (multichannels) or Single (one channel) mode.
 This parameter can be set to ENABLE or DISABLE */

 FunctionalState ADC_ContinuousConvMode; /*!< Specifies whether the conversion is performed in
 Continuous or Single mode.
 This parameter can be set to ENABLE or DISABLE. */

 uint32_t ADC_ExternalTrigConv; /*!< Defines the external trigger used to start the analog
 to digital conversion of regular channels. This parameter
 can be a value of @ref ADC_external_trigger_sources_for_regular_channels_conversion */

 uint32_t ADC_DataAlign; /*!< Specifies whether the ADC data alignment is left or right.
 This parameter can be a value of @ref ADC_data_align */

 uint8_t ADC_NbrOfChannel; /*!< Specifies the number of ADC channels that will be converted
 using the sequencer for regular channel group.
 This parameter must range from 1 to 16. */
}ADC_InitTypeDef;
```

Trong đó các trường của kiểu dữ liệu cấu trúc này có ý nghĩa cụ thể như sau:

ADC_Mode	Chế độ hoạt động của bộ ADC
ADC_ScanConvMode	Chế độ đọc nhiều kênh
ADC_ContinuousConvMode	Chế độ đọc liên tục, rời rạc
ADC_ExternalTrigConv	Kích hoạt chuyển đổi bởi nguồn kích thích ngoài
ADC_DataAlign	Chế độ căn chỉnh dữ liệu
ADC_NbrOfChannel	Số lượng kênh dữ liệu đọc

Bảng 5. 4: Các thông tin cấu hình ADC

Chương trình dưới đây trình bày một ví dụ đơn giản về đọc 1 kênh dữ liệu ADC ở chế độ không liên tục. Để bộ ADC hoạt động thì cổng ADC cần đọc cần được cấu hình ở trạng thái Analog Input. Chương trình cụ thể như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "stm32f10x_adc.h" // Keil::Device:StdPeriph Drivers:ADC
#include "Delay.h"

ADC_InitTypeDef ADC_InitStructure;
GPIO_InitTypeDef GPIO_InitStructure;
unsigned int vrui_ADC_Value;

void Fn_ADC_Init (void);
unsigned int Fn_ADC_Read (void);

int main (void){
      SystemInit();
      SystemCoreClockUpdate();

      Fn_ADC_Init();
      while(1){
            vrui_ADC_Value = Fn_ADC_Read();
```

```
            Fn_DELAY_ms(100);
      }
}

void Fn_ADC_Init (void){
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 ;
      GPIO_Init(GPIOA, &GPIO_InitStructure);

      RCC_ADCCLKConfig (RCC_PCLK2_Div8);
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

      ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
      ADC_InitStructure.ADC_ScanConvMode = DISABLE;
      ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
      ADC_InitStructure.ADC_ExternalTrigConv                              =
ADC_ExternalTrigConv_None;
      ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
      ADC_InitStructure.ADC_NbrOfChannel = 1;

      ADC_RegularChannelConfig(ADC1,ADC_Channel_1,
1,ADC_SampleTime_71Cycles5); // Cau hinh thoi gian chuyen doi
      ADC_Init( ADC1, &ADC_InitStructure);

      ADC_Cmd (ADC1,ENABLE);

      ADC_ResetCalibration(ADC1);   // Reset hieu Chuan ADC
      while(ADC_GetResetCalibrationStatus(ADC1));
      ADC_StartCalibration(ADC1);   // Hieu chuan lai
```

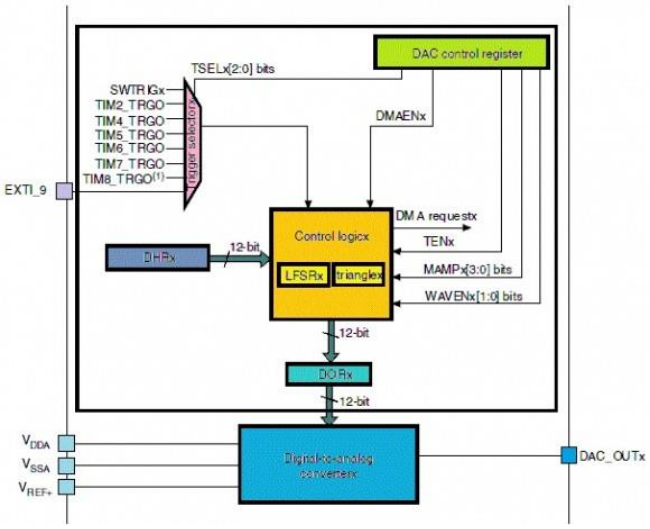
```
	while(ADC_GetCalibrationStatus(ADC1));


	ADC_Cmd (ADC1,ENABLE);
	ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}


unsigned int Fn_ADC_Read (void){
	return(ADC_GetConversionValue(ADC1));
}
```

Chương trình thực hiện đọc dữ liệu ADC ở chân A1 và lưu vào biến vrui_ADC_Value. Người học có thể sử dụng biến này cho các ứng dụng khác hoặc đơn giản gửi dữ liệu đó thông qua kênh UART lên máy tính để theo dõi.

Lập trình 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.

Sơ đồ khối bộ DAC của STM32F1 có dạng như sau:



Hình 5. 41: Sơ đồ khối bộ DAC

DAC của STM32F1 có một số đặc tính như sau:

- 2 bộ DAC
- Độ phân dải 12 bit dữ liệu
- Có thể căn chỉnh dữ liệu trái hoặc phải
- Hỗ trợ tạo sóng có nhiễu
- Hỗ trợ 2 kênh DAC chuyển đổi độc lập
- Hỗ trợ truyền dữ liệu vào từ DMA
- Hỗ trợ kích khởi chuyển đổi từ các nguồn kích ngoài

Cấu trúc biến khởi tạo cấu hình cho bộ DAC cho STM32F1 trên thư viện SPL có dạng như sau:

```
typedef struct
{
 uint32_t DAC_Trigger; /*!< Specifies the external trigger for the selected DAC channel.
 This parameter can be a value of @ref DAC_trigger_selection */

 uint32_t DAC_WaveGeneration; /*!< Specifies whether DAC channel noise waves or triangle waves
 are generated, or whether no wave is generated.
 This parameter can be a value of @ref DAC_wave_generation */

 uint32_t DAC_LFSRUnmask_TriangleAmplitude; /*!< Specifies the LFSR mask for noise wave generation or
 the maximum amplitude triangle generation for the DAC channel.
 This parameter can be a value of @ref DAC_lfsrunmask_triangleamplitude */

 uint32_t DAC_OutputBuffer; /*!< Specifies whether the DAC channel output buffer is enabled or disabled.
 This parameter can be a value of @ref DAC_output_buffer */
}DAC_InitTypeDef;
```

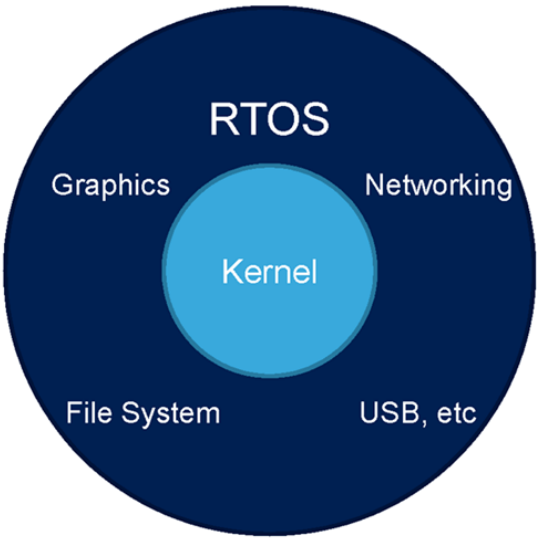
Ví dụ về một chương trình khởi tạo:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "stm32f10x_dac.h" // Keil::Device:StdPeriph Drivers:DAC
#include "Delay.h"

DAC_InitTypeDef DAC_InitStructure;
GPIO_InitTypeDef GPIO_InitStruct;

void Fn_DAC_Init (void);
void Fn_DAC_Write(unsigned int _vrui_Value);

int main (void){
	unsigned int Count;
	SystemInit();
	SystemCoreClockUpdate();
	Fn_DELAY_Init(72);

	Fn_DAC_Init();
	while(1){
		for (Count = 0; Count < 4096; Count++){
			Fn_DAC_Write(Count);
			Fn_DELAY_ms(5);
		}
	}
}

void Fn_DAC_Init (void){
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
```

```
	GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
	GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
	GPIO_Init(GPIOA, &GPIO_InitStruct);

	DAC_InitStructure.DAC_Trigger = DAC_Trigger_Software;
	DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_Noise;
	DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude               =
DAC_LFSRUnmask_Bits8_0;
	DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
	DAC_Init(DAC_Channel_1, &DAC_InitStructure);

	DAC_Cmd(DAC_Channel_1, ENABLE);

	Fn_DAC_Write(0);
}

void Fn_DAC_Write(unsigned int _vrui_Value){
	DAC_SetChannel1Data(DAC_Align_12b_R, _vrui_Value & 0x0FFF);
	DAC_SoftwareTriggerCmd(DAC_Channel_1, ENABLE);
}
```

Chương trình thực hiện lần lượt xuất dữ liệu từ 0 đến 4095 (12 bit) lần lượt ra kênh 1 của bộ DAC số 1 với chế độ căn chỉnh là căn phải. Chúng ta có thể dùng DAC để điều khiển dữ liệu tương tự như xuất âm thanh, tạo dạng sóng,...

Lập trình với FreeRTOS

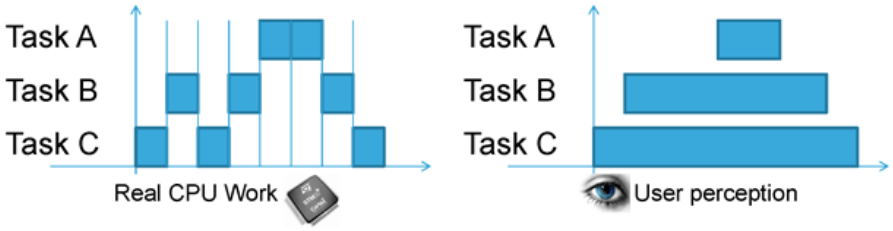


Hình 5. 42: Kiến trúc RTOS

FreeRTOS là một hệ điều hành nhúng thời gian thực (Real Time Operating System) mã nguồn mở được phát triển bởi Real Time Engineers Ltd, sáng lập và sở hữu bởi Richard Barry. FreeRTOS được thiết kế phù hợp cho nhiều hệ nhúng nhỏ gọn vì nó chỉ triển khai rất ít các chức năng như: cơ chế quản lý bộ nhớ và tác vụ cơ bản, các hàm API quan trọng cho cơ chế đồng bộ. Nó không cung cấp sẵn các giao tiếp mạng, drivers, hay hệ thống quản lý tệp (file system) như những hệ điều hành nhúng cao cấp khác. Tuy vậy, FreeRTOS có nhiều ưu điểm, hỗ trợ nhiều kiến trúc vi điều khiển khác nhau, kích thước nhỏ gọn (4.3 Kbytes sau khi biên dịch trên ARM7), được viết bằng ngôn ngữ C và có thể sử dụng, phát triển với nhiều trình biên dịch C khác nhau (GCC, OpenWatcom, Keil, IAR, Eclipse, …), cho phép không giới hạn các tác vụ chạy đồng thời, không hạn chế quyền ưu tiên thực thi, khả năng khai thác phần cứng. Ngoài ra, nó cũng cho phép triển khai các cơ chế điều độ giữa các tiến trình như: queues, counting semaphore, mutexes.

FreeRTOS là một hệ điều hành nhúng rất phù hợp cho nghiên cứu, học tập về các kỹ thuật, công nghệ trong viết hệ điều hành nói chung và hệ điều hành nhúng thời gian thực nói riêng, cũng như việc phát triển mở rộng tiếp các thành phần cho hệ điều hành hành (bổ sung modules, driver, thực hiện porting).

Kernel



Hình 5. 43: Hoạt động của CPU và quan sát của người dùng với Multi Thread

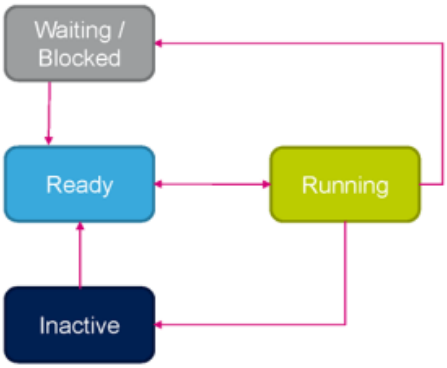
Kernel sẽ có nhiệm vụ quản lý nhiều task cùng chạy 1 lúc, mỗi task thường chạy mất vài ms. Tại lúc kết thúc task cần thực hiện các công việc cơ bản sau:

- Lưu trạng thái task
- Thanh ghi CPU sẽ load trạng thái của task tiếp theo
- Task tiếp theo cần khoảng vài ms để thực hiện

Vì CPU thực hiện tác vụ rất nhanh nên dưới góc nhìn người dùng thì hầu như các task là được thực hiện 1 cách liên tục.

Task state

Một task trong RTOS thường có các trạng thái như sau:



Hình 5. 44: Các trạng thái của Task trong RTOS

RUNNING: đang thực thi

READY: sẵn sàng để thực hiện

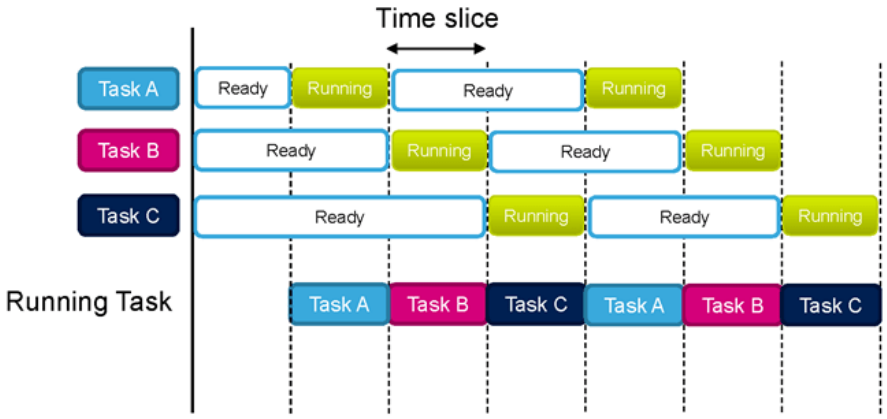
WAITING: chờ sự kiện

INACTIVE: không được kích hoạt

Scheduler

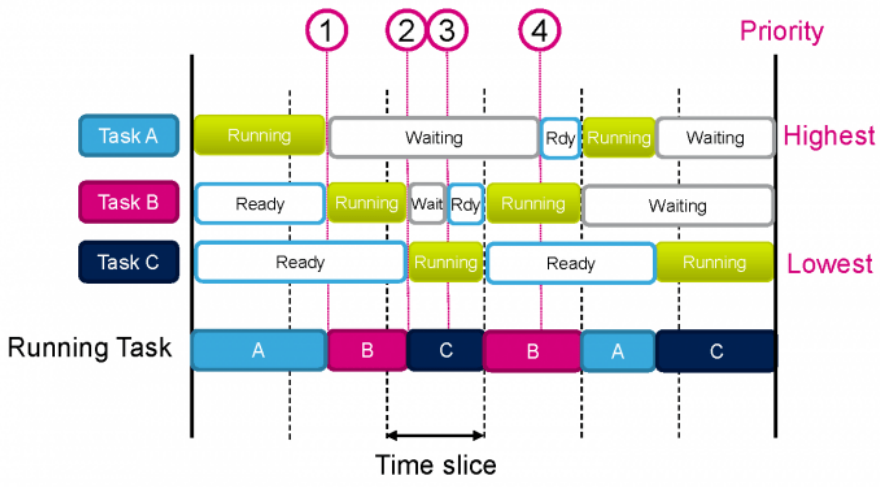
Đây là 1 thành phần của kernel quyết định task nào được thực thi. Có một số luật cho scheduling như:

- Cooperative: Giống với lập trình thông thường, mỗi task chỉ có thể thực thi khi task đang chạy dừng lại, nhược điểm của cơ chế này là task này có thể dùng hết tất cả tài nguyên của CPU
- Round-robin: Mỗi task được thực hiện trong thời gian định trước (time slice) và không có ưu tiên.



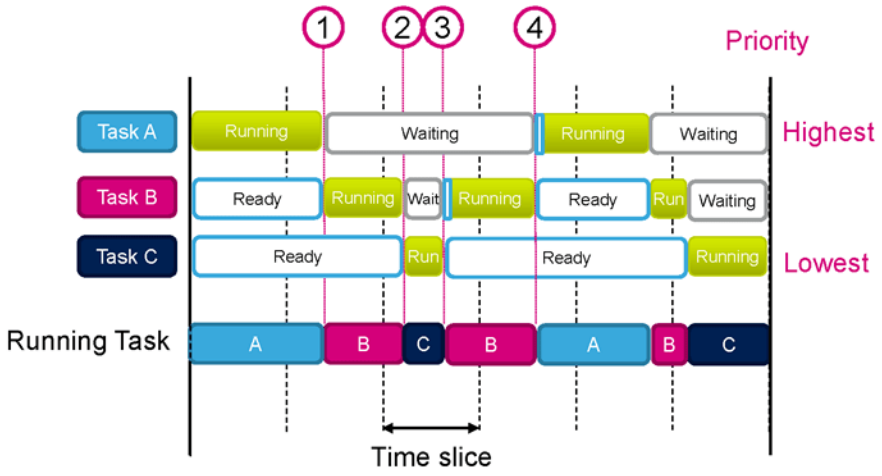
Hình 5. 45: Cơ chế Round-Robin

- Priority base: Task được phân quyền cao nhất sẽ được thực hiện trước, nếu các task có cùng quyền như nhau thì sẽ giống với round-robin, các task có mức ưu tiên thấp hơn sẽ được thực hiện cho đến cuối time slice



Hình 5. 46: Cơ chế Priority Base

- Priority-based pre-emptive: Các task có mức ưu tiên cao nhất luôn nhường các task có mức ưu tiên thấp hơn thực thi trước.



Hình 5. 47: Cơ chế Priority-based pre-emptive

Task

Một task là một chương trình, chương trình này chạy liên tục trong vòng lặp vô tận và không bao giờ dừng lại. Một chương trình thường sẽ có nhiều task khác nhau. Ví dụ như máy bán đồ uống tự động sẽ có các thành task sau:

- Task quản lý việc lựa chọn của người dùng
- Task để kiểm tra đúng số tiền người dùng đã trả
- Task để điều khiển động cơ/ cơ cấu cung cấp nước uống.

Kernel quản lý việc chuyển đổi giữa các task, nó lưu lại ngữ cảnh của task sắp bị hủy và khôi phục lại ngữ cảnh của task tiếp theo bằng cách:

- Kiểm tra thời gian thực thi đã được định nghĩa trước (time slice được tạo ra bởi ngắt systick)
- Khi có các sự kiện unblocking một task có quyền cao hơn xảy ra (signal, queue, semaphore,…)
- Khi task gọi hàm Yield() để ép Kernel chuyển sang các task khác mà không phải chờ cho hết time slice
- Khi khởi động thì kernel sẽ tạo ra một task mặc định gọi là Idle Task.

Để tạo một task thì cần phải khai báo hàm định nghĩa task, sau đó tạo task và cấp phát bộ nhớ.

Kết nối Inter-task & Chia sẻ tài nguyên

Các task cần phải kết nối và trao đổi dữ liệu với nhau để có thể chia sẻ tài nguyên, có một số khái niệm cần lưu ý như sau:

Với Inter-task Communication:

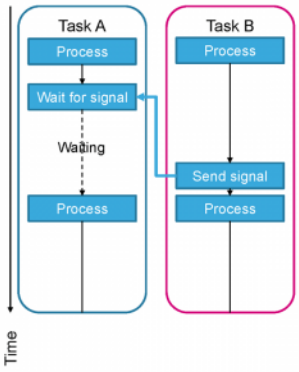
- Signal Events – Đồng bộ các task
- Message queue – Trao đổi tin nhắn giữa các task trong hoạt động giống như FIFO
- Mail queue – Trao đổi dữ liệu giữa các task sử dụng hằng đợi của khối bộ nhớ

Với Resource Sharing

- Semaphores – Truy xuất tài nguyên liên tục từ các task khác nhau
- Mutex – Đồng bộ hóa truy cập tài nguyên sử dụng Mutual Exclusion

Signal event

Signal event được dùng để đồng bộ các task, ví dụ như bắt task phải thực thi tại một sự kiện nào đó được định sẵn.



Hình 5. 48: Sử dụng Signal Event trong đồng bộ thông tin các Task

Ví dụ: Một cái máy giặt có 2 task là Task A điều khiển động cơ, Task B đọc mức nước từ cảm biến nước đầu vào:

- Task A cần phải chờ nước đầy trước khi khởi động động cơ. Việc này có thể thực hiện được bằng cách sử dụng signal event
- Task A phải chờ signal event từ Task B trước khi khởi động động cơ
- Khi phát hiện nước đã đạt tới mức yêu cầu thì Task B sẽ gửi tín hiệu tới Task A
- Với trường hợp này thì task sẽ đợi tín hiệu trước khi thực thi, nó sẽ nằm trong trạng thái là WAITING cho đến khi signal được cài đặt.

Mỗi task có thể được gán tối đa là 32 signal event. Ưu điểm của Event là thực hiện nhanh, sử dụng ít RAM hơn so với semaphore và message queue nhưng có nhược điểm lại chỉ được dùng khi một task nhận được signal.

Message queue

Message queue là cơ chế cho phép các task có thể kết nối với nhau, nó là một FIFO buffer được định nghĩa bởi độ dài (số phần tử mà buffer có thể lưu trữ) và kích thước dữ

liệu (kích thước của các thành phần trong buffer). Một ứng dụng tiêu biểu là buffer cho Serial I/O, buffer cho lệnh được gửi tới task.

Hình 5. 49: Trao đổi dữ liệu giữa các Task sử dụng Queue

Task có thể ghi vào hàng đợi (queue)

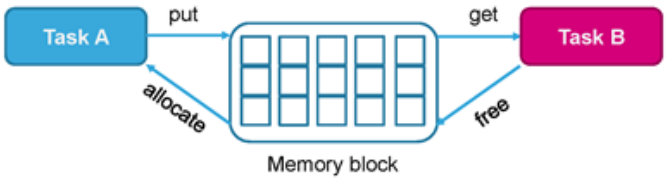
- Task sẽ bị khóa (block) khi gửi dữ liệu tới một message queue đầy đủ
- Task sẽ hết bị khóa (unblock) khi bộ nhớ trong message queue trống
- Trường hợp nhiều task mà bị block thì task với mức ưu tiên cao nhất sẽ được unblock trước

Task có thể đọc từ hàng đợi (queue)

- Task sẽ bị block nếu message queue trống
- Task sẽ được unblock nếu có dữ liệu trong message queue.
- Tương tự ghi thì task được unblock dựa trên mức độ ưu tiên

Mail queue

Giống như message queue nhưng dữ liệu sẽ được truyền dưới dạng khối (memory block) thay vì dạng đơn. Mỗi memory block thì cần phải cấp phát trước khi đưa dữ liệu vào và giải phóng sau khi đưa dữ liệu ra.



Hình 5. 50: Trao đổi dữ liệu giữa các Task sử dụng Mail Queue

Để gửi dữ liệu với mail queue cần thực hiện các bước sau:

- Cấp phát bộ nhớ từ mail queue cho dữ liệu được đặt trong mail queue
- Lưu dữ liệu cần gửi vào bộ nhớ đã được cấp phát
- Đưa dữ liệu vào mail queue

Nhận dữ liệu trong mail queue bởi task khác:

- Lấy dữ liệu từ mail queue, sẽ có một hàm để trả lại cấu trúc/ đối tượng

- Lấy con trỏ chứa dữ liệu
- Giải phóng bộ nhớ sau khi sử dụng dữ liệu

Semaphore

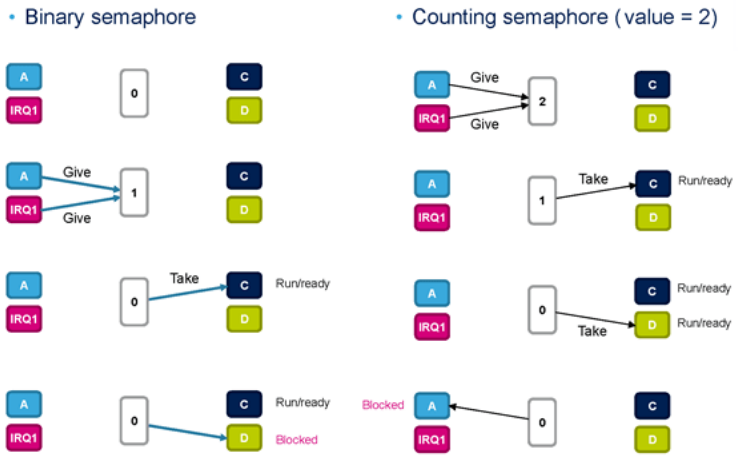
Được sử dụng để đồng bộ task với các sự kiện khác trong hệ thống. Có 2 loại là: Binary và Counting.

Binary semaphore:

- Trường hợp đặc biệt của counting semaphore
- Có duy nhất 1 token
- Chỉ có 1 hoạt động đồng bộ

Counting semaphore

- Có nhiều token
- Có nhiều hoạt động đồng bộ



Hình 5. 51: Binary semaphore và Counting semaphore

Mutex

Sử dụng cho việc loại trừ (mutial exclution), hoạt động như là một token để bảo vệ tài nguyên được chia sẻ. Một task nếu muốn truy cập vào tài nguyên chia sẻ cần thực hiện các việc sau:

- Yêu cầu mutex trước khi truy cập vào tài nguyên chia sẻ.
- Đưa ra token khi kết thúc với tài nguyên.

Tại mỗi một thời điểm thì chỉ có 1 task có được mutex. Những task khác muốn cùng mutex thì phải block cho đến khi task cũ thả mutex ra.

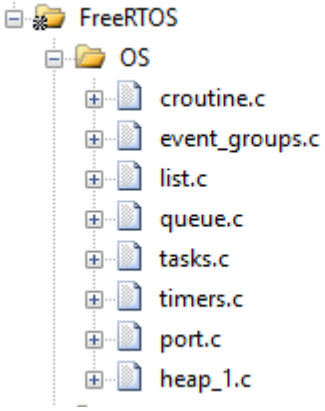
Hình 5. 52: Sử dụng chung tài nguyên hệ thống với Mutex

Về cơ bản thì Mutex giống như binary semaphore nhưng được sử dụng cho việc loại trừ chứ không phải đồng bộ. Ngoài ra thì nó bao gồm cơ chế thừa kế mức độ ưu tiên(Priority inheritance mechanism) để giảm thiểu vấn đề đảo ngược ưu tiên, cơ chế này có thể hiểu đơn giản qua ví dụ sau:

- Task A (low priority) yêu cầu mutex
- Task B (high priority) muốn yêu cầu cùng mutex trên.
- Mức độ ưu tiên của Task A sẽ được đưa tạm về Task B để cho phép Task A được thực thi
- Task A sẽ thả mutex ra, mức độ ưu tiên sẽ được khôi phục lại và cho phép Task B tiếp tục thực thi.

Người dùng có thể tải về bộ chương trình FreeRTOS từ địa chỉ sau: https://www.freertos.org/

Để tích hợp hệ điều hành FreeRTOS người dùng cần thêm vào Project các file sau:



Hình 5. 53: Thêm FreeRTOS vào Project

Trong đó heap có thể được lựa chọn giữa các file heap_1, heap_2, heap_3, heap_4 tùy thuộc vào nhu cầu sử dụng của người dùng. Ở giáo trình này do chỉ dùng ở các ứng dụng đơn giản nên sử dụng heap_1.

Lập trình MultiTask với FreeRTOS

Chương trình sau xây dựng một ví dụ về lập trình 2 task chạy song song với nhau bằng cách sử dụng cơ chế tạo Task của FreeRTOS. Chương trình cụ thể như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "Delay.h"
#include "FreeRTOS.h"
#include "task.h"

GPIO_InitTypeDef GPIO_InitStructure;
void Fn_GPIO_Init (void);
void Fn_RTOS_TaskLed1(void *p);
void Fn_RTOS_TaskLed2(void *p);

int main (void){
	SystemInit();
	SystemCoreClockUpdate();

	Fn_GPIO_Init();
	xTaskCreate(Fn_RTOS_TaskLed1, (const char*) "Red LED Blink", 128,
NULL, 1, NULL);
	xTaskCreate(Fn_RTOS_TaskLed2, (const char*) "Green LED Blink",
128, NULL, 1, NULL);
	vTaskStartScheduler();
	return 0;
}

void Fn_GPIO_Init (void){
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_15;
```

```
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOE, &GPIO_InitStructure);
}

void Fn_RTOS_TaskLed1(void *p){
	while(1){
		GPIOE->ODR ^= GPIO_Pin_15;
		vTaskDelay(100/portTICK_RATE_MS);
	}
}

void Fn_RTOS_TaskLed2(void *p){
	while(1){
		GPIOE->ODR ^= GPIO_Pin_13;
		vTaskDelay(500/portTICK_RATE_MS);
	}
}
```

Chương trình xây dựng 2 task điều khiển 2 đèn LED nhấp nháy với 2 tần số khác nhau. Nếu như không sử dụng hệ điều hành người lập trình cần lập trình xen kẽ phù hợp và linh động nếu không sẽ rất dễ xảy ra xung đột. Với hệ điều hành việc thực hiện các tác vụ định kỳ và song song sẽ trở nên dễ dàng hơn rất nhiều.

Với chương trình trên chúng ta cần xem xét các hàm sau:

xTaskCreate	Tạo một Task vụ gắn với một hàm đã được xây dựng trước.
vTaskDelay	Hàm trễ của hệ điều hành
vTaskStartScheduler	Thực hiện lập lịch cho các task

Tùy thuộc vào số tiến trình cần quản lý người dùng có thể xây dựng một chương trình với nhiều Task hơn nữa. Các Task này tùy thuộc vào độ ưu tiên, thời gian

chạy định kỳ mà sẽ được hệ thống lập lịch để hoạt động vào các thời điểm phù hợp với tài nguyên của hệ thống.

Lập trình FreeRTOS và Mutex

Sử dụng cho việc loại trừ lẫn nhau(mutial exclution), nghĩa là nó được dùng để hạn chế quyền truy cập tới một số resource, mutex hoạt động như là một token để bảo vệ tài nguyên chia sẻ.

Một task nếu muốn truy cập vào tài nguyên chia sẻ thì:

- Cần đợi mutex trước khi truy cập vào tài nguyên chia sẻ.
- Giải phóng token khi kết thúc với tài nguyên.

Tại mỗi một thời điểm thì chỉ có 1 task duy nhất có được mutex. Khi task này có mutex thì những task khác cũng muốn mutex này đều sẽ bị chặn cho tới khi task hiện tại giải phóng mutex ra.

Chương trình sau xây dựng một ví dụ sử dụng chung tài nguyên là màn hình GLCD đã được xây dựng ở trên. Từ đó đồng bộ chia sẻ tài nguyên để các task có thể lần lượt truy cập và điều khiển màn hình khi cần thiết.

Chương trình cụ thể như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "Delay.h"
#include "FreeRTOS.h"
#include "task.h"

#include "N5110.h"
#include "semphr.h"

xSemaphoreHandle xMutex;

char TextTemp[200];
int Number1;
int Number2;
```

```
void Fn_GLCD_Init (void);
void Fn_RTOS_N5110_Text1 (void *p);
void Fn_RTOS_N5110_Text2 (void *p);

int main (void){
	xMutex = xSemaphoreCreateMutex();

	SystemInit();
	SystemCoreClockUpdate();

	Fn_GPIO_Init();
	Fn_N5110_Init();
	sprintf(TextTemp,"Task1:%4d",1234);
	Fn_N5110_GotoXY(0,0);
	Fn_N5110_Puts(TextTemp);
	xTaskCreate(Fn_RTOS_N5110_Text1, (const char*) "Task1 LCD", 128,
NULL, 1, NULL);
	xTaskCreate(Fn_RTOS_N5110_Text2, (const char*) "Task2 LCD", 128,
NULL, 1, NULL);
	vTaskStartScheduler();
	return 0;
}

void Fn_RTOS_N5110_Text1 (void *p){
	while(1){
		xSemaphoreTake(xMutex, portMAX_DELAY);
		Number1++;
		sprintf(TextTemp,"Task1:%4d",Number1);
		Fn_N5110_GotoXY(0,0);
		Fn_N5110_Puts(TextTemp);
		xSemaphoreGive( xMutex );
```

```
            vTaskDelay(100/portTICK_RATE_MS);
      }
}

void Fn_RTOS_N5110_Text2 (void *p){
      while(1){
            xSemaphoreTake(xMutex, portMAX_DELAY);
            Number2++;
            sprintf(TextTemp,"Task2:%4d",Number2);
            Fn_N5110_GotoXY(0,3);
            Fn_N5110_Puts(TextTemp);
            xSemaphoreGive( xMutex );
            vTaskDelay(500/portTICK_RATE_MS);
      }
}
```

Hai task thực hiện điều khiển 2 dòng của màn hình hiển thị 2 số khác nhau. Nếu như không có sự phân chia tài nguyên sử dụng mutex thì có thể sẽ xảy ra hiện tượng khi Task1 đang thực hiện ghi ra màn hình thì Task2 chạy thực hiện ghi tiếp vào vị trí mà Task 1 đang ghi gây ra tình trạng ghi nhầm lẫn dữ liệu.

Mutex sẽ giúp hệ thống quản lý tài nguyên giúp cho khi một Task sử dụng thì đảm bảo Task đó sử dụng xong thì Task sau mới được quyền can thiệp vào tài nguyên chia sẻ.

Chương trình cần chú ý vào các hàm sau:

xSemaphoreCreateMutex	Tạo một Mutex làm cờ cho một tài nguyên nào đó
xSemaphoreTake	Thực hiện lấy cờ tài nguyên về Task đang chạy. Nếu cờ tài nguyên đang được sử dụng thì chương trình tạm dừng ở lệnh này.
xSemaphoreGive	Giải phóng Mutex sau khi sử dụng xong tài nguyên.

Chú ý: Một Task vụ muốn truy cập tài nguyên thì cần thực hiện đúng quy trình lấy cờ tài nguyên trước rồi mới thực hiện điều khiển nếu không sẽ gây ra hiện tượng tài nguyên bị dùng một cách xung đột. Ngay sau khi Task sử dụng xong tài nguyên cần trả lại cờ tài nguyên để các ứng dụng khác có thể truy cập. Nếu không giải phóng thì các ứng dụng khác cũng sử dụng tài nguyên đó sẽ không bao giờ có thể chạy được.

Lập trình FreeRTOS và Queues

Queue có 2 dạng là message queue và mail queue, đây là 2 cơ chế được dùng để các task có thể trao đổi với nhau, sự khác biệt lớn nhất giữa 2 dạng này là message queue thì truyền dữ liệu dưới dạng đơn, còn mail queue sẽ truyền dưới dạng khối.

Sau đây là một ứng dụng sử dụng queues để trao đổi dữ liệu giữa 2 Task. Ví dụ này dừng lại ở sử dụng message queue. Người đọc có thể tìm hiểu thêm về mail queue ở các tài liệu khác.

Ý tưởng chính của chương trình là tạo một Task có trách nhiệm điều khiển màn hình và một Task khác cung cấp dữ liệu cho Task điều khiển màn hình thông qua queues.

Chương trình cụ thể như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "Delay.h"
#include "N5110.h"
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"

xQueueHandle xQueue;

char TextTemp[200];

void Fn_GLCD_Init (void);
void Fn_RTOS_N5110_Send (void *p);
```

```
void Fn_RTOS_N5110_Disp (void *p);


int main (void){
	xQueue = xQueueCreate( 10, sizeof( uint32_t ) );


	SystemInit();
	SystemCoreClockUpdate();


	Fn_GPIO_Init();
	Fn_N5110_Init();
	Fn_N5110_GotoXY(0,0);
	Fn_N5110_Puts(TextTemp);
	xTaskCreate(Fn_RTOS_N5110_Send, (const char*) "Send Data", 128,
NULL, 1, NULL);
	xTaskCreate(Fn_RTOS_N5110_Disp, (const char*) "Display Data",
128, NULL, 1, NULL);
	vTaskStartScheduler();
	return 0;
}


void Fn_GPIO_Init (void){
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);


	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_15;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOE, &GPIO_InitStructure);
}


void Fn_RTOS_N5110_Send (void *p){
	uint32_t DataSend = 0;
```

```
	while(1){
		xQueueSend(xQueue, (void*)&DataSend, (portTickType)0);
		DataSend++;
		vTaskDelay(1000);
		xQueueSend(xQueue, (void*)&DataSend, (portTickType)0);
		DataSend++;
		vTaskDelay(5000);
	}
}

void Fn_RTOS_N5110_Disp (void *p){
	uint32_t DataReceive;
	while(1){
		if(xQueueReceive(xQueue,		(void*)&DataReceive,
(portTickType)0xFFFFFFFF))
		{
			sprintf(TextTemp,"Number:%5d", DataReceive);
			Fn_N5110_GotoXY(0,0);
			Fn_N5110_Puts(TextTemp);

		}
	}
}
```

Task Fn_RTOS_N5110_Disp thực hiện nhiệm vụ nếu có dữ liệu gửi đến thì thực hiện in số đó lên màn hình GLCD. Dữ liệu nhận được thông qua hàm xQueueReceive nhằm lấy dữ liệu từ trong queues có tên là xQueue nếu có bản tin được gửi đến Task đó.

Task Fn_RTOS_N5110_Send thực hiện tạo một dữ liệu số tăng dần sau đó gửi dữ liệu đó đến queues xQueue để các Task khác sử dụng con số này cho mục đích phù hợp.

Các hàm liên quan đến queues cơ bản nằm trong bảng sau:

xQueueCreate	Tạo một queues dữ liệu theo kích thước mong muốn.
xQueueSend	Truyền một dữ liệu vào queues tương ứng.
xQueueReceive	Kiểm tra queues có dữ liệu mới không, nếu có thì lấy dữ liệu đó ra khỏi queues.

Lập trình FreeRTOS và Events

Signal event được dùng để đồng bộ các task, ví dụ như bắt task phải thực thi tại một sự kiện nào đó được định sẵn. Khác với queues, Event dùng để định nghĩa các sự kiến từ đó điều khiển các Task hoạt động đồng bộ với nhau.

Ví dụ như một Task thực hiện hiện việc A và một Task thực hiện việc B với điều kiện việc B được thực hiện sau khi A đã thực hiện xong. Vì vậy trong suốt quá trình A được thực hiện thì Task thực hiện việc B cần hoạt động ở chế độ khóa. Khi Task thực hiện A xong sẽ báo với Task thực hiện B rằng đã sẵn sàng đến Task B hoạt động. Lúc đó B sẽ ở trạng thái sẵn sàng và chờ thời điểm phù hợp để hoạt động.

Chương trình sau nêu một ví dụ về thực hiện 2 Task đồng bộ thông qua Event. Cụ thể TaskMain thực hiện lần lượt điều khiển TaskA và TaskB theo định kỳ. Các TaskA và TaskB là các tác độc lập hoạt động dưới sự điều khiển của TaskMain một cách không trực tiếp thông qua các Event mà hệ thống cài đặt. Cụ thể chương trình như sau:

```
#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "Delay.h"
#include "N5110.h"
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "event_groups.h"

EventGroupHandle_t event_hdl;
```

```
char TextTemp[200];

#define EVENT0          (1<<0)
#define EVENT1          (1<<1)
#define EVENT2          (1<<2)
#define EVENT_ALLS      ((EVENT2<<1) - 1)

void Fn_GPIO_Init (void);

void TaskMain(void *pvParameters);
void TaskA(void *pvParameters);
void TaskB(void *pvParameters);

int main (void){

      SystemInit();
      SystemCoreClockUpdate();

      Fn_GPIO_Init();
      Fn_N5110_Init();
      xTaskCreate(TaskMain,  "TaskMain",  128,  NULL,  tskIDLE_PRIORITY,
NULL);
 xTaskCreate(TaskA, "TaskA", 128, NULL, tskIDLE_PRIORITY, NULL);
 xTaskCreate(TaskB, "TaskB", 128, NULL, tskIDLE_PRIORITY, NULL);
      vTaskStartScheduler();
      return 0;
}

void Fn_GPIO_Init (void){
      GPIO_InitTypeDef GPIO_InitStructure;
```

```
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOE, &GPIO_InitStructure);
}

void TaskMain(void *pvParameters){
	event_hdl = xEventGroupCreate();
	while(1){
		xEventGroupSetBits(event_hdl, EVENT0);
		vTaskDelay(2000);
		xEventGroupSetBits(event_hdl, EVENT1);
		vTaskDelay(2000);
		xEventGroupSetBits(event_hdl, EVENT_ALLS);
		vTaskDelay(2000);
	}
}

void TaskA(void *pvParameters){
	EventBits_t event;
	while(1){
		event = xEventGroupWaitBits(event_hdl, EVENT_ALLS, pdTRUE,
pdFALSE, portMAX_DELAY);
		if(event & EVENT0){
			GPIOE->ODR ^= 0x000FF00;
		}

	}
}
```

```
void TaskB(void *pvParameters){
	int Number = 0;
	EventBits_t event;
	while(1){
		event = xEventGroupWaitBits(event_hdl, EVENT_ALLS, pdTRUE, pdFALSE, portMAX_DELAY);
		if(event & EVENT1){
			Number++;
			sprintf(TextTemp,"Number:%5d", Number);
			Fn_N5110_GotoXY(0,0);
			Fn_N5110_Puts(TextTemp);
		}

	}
}
```

Tiến trình cụ thể như sau:

TaskMain điều khiển TaskA chạy sau đó 2 giây sau thực hiện cho Task B chạy và sau 2 giây thì cho phép TaskA và TaskB cùng chạy. Ở khoảng đầu chỉ TaskA chạy còn TaskB ở trạng thái khóa. Sau 2 giây TaskA ở trạng thái khóa và TaskB ở trạng thái chạy. Tiếp sau 2 giây nữa TaskA và TaskB đều ở trạng thái sẵn sàng và chia sẻ tài nguyên hệ thống để có thể chạy song song.

Ở ví dụ trên TaskA thực hiện điều khiển nhấp nháy đèn LED được nối với các chân từ E8 đến E15 và TaskB thực hiện hiển thị số tăng dần lên màn hình GLCD dưới sự điều khiển định kỳ của TaskMain.

Khi lập trình Events với FreeRTOS cần chú ý các sự kiện sau:

xEventGroupCreate	Tạo một cụm Events
xEventGroupWaitBits	Chờ sự kiện cập nhật events nào đó
event & EVENT0	Kiểm tra một Event nào đó trong cụm Event xem xét có được kích hoạt không

5.4 Thiết lập hệ điều hành nhúng trên nền ARM

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.

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.

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, chung 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.

PHỤ LỤC

Thư viện GLCD PCD8544

LCD5110_Graph.h

```
#ifndef LCD5110_Graph_h
#define LCD5110_Graph_h

#include <stdio.h>
#include <string.h>
#include "stm32f10x.h" // Device header
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
#include "stm32f10x_spi.h" // Keil::Device:StdPeriph Drivers:SPI
#include "Delay.h"
#include "SPI.h"

#define LEFT 0
#define RIGHT 9999
#define CENTER 9998

#define LCD_COMMAND 0
#define LCD_DATA 1

// PCD8544 Commandset
// ------------------
// General commands
#define PCD8544_POWERDOWN                0x04
#define PCD8544_ENTRYMODE                0x02
#define PCD8544_EXTENDEDINSTRUCTION 0x01
#define PCD8544_DISPLAYBLANK        0x00
#define PCD8544_DISPLAYNORMAL       0x04
#define PCD8544_DISPLAYALLON        0x01
#define PCD8544_DISPLAYINVERTED          0x05
// Normal instruction set
#define PCD8544_FUNCTIONSET              0x20
#define PCD8544_DISPLAYCONTROL           0x08
#define PCD8544_SETYADDR                 0x40
#define PCD8544_SETXADDR                 0x80
// Extended instruction set
#define PCD8544_SETTEMP                  0x04
#define PCD8544_SETBIAS                  0x10
#define PCD8544_SETVOP                   0x80
```

```
// Display presets
#define LCD_BIAS                            0x03  // Range: 0-7 (0x00-
0x07)
#define LCD_TEMP                            0x02  // Range: 0-3 (0x00-
0x03)
#define LCD_CONTRAST                        0x46  // Range: 0-127 (0x00-
0x7F)

#define fontbyte(x) cfont.font[x]
#define bitmapbyte(x) bitmap[x]


extern unsigned char TinyFont[];
extern unsigned char SmallFont[];



#define byte      char

struct _current_font
{
      uint8_t* font;
      uint8_t x_size;
      uint8_t y_size;
      uint8_t offset;
      uint8_t numchars;
      uint8_t inverted;
};

class LCD5110
{
      public:
            LCD5110 (GPIO_TypeDef *_SPI_Port,int SCK, int MOSI,
GPIO_TypeDef *_LCD_Port, int DC, int RST, int CS);
            void InitLCD(int contrast=LCD_CONTRAST);
            void setContrast(int contrast);
            void enableSleep();
            void disableSleep();
            void update();
            void clrScr();
            void fillScr();
            void invert(bool mode);
            void setPixel(uint16_t x, uint16_t y);
```

```
            void clrPixel(uint16_t x, uint16_t y);
            void invPixel(uint16_t x, uint16_t y);
            void invertText(bool mode);
            void print(char *st, int x, int y);
            void print(const char *st, int x, int y);
//          void print(String st, int x, int y);
            void printNumI(long num, int x, int y, int length=0, char
filler=' ');
            void printNumF(double num, byte dec, int x, int y, char
divider='.', int length=0, char filler=' ');
            void setFont(uint8_t* font);
            void drawBitmap(int x, int y, uint8_t* bitmap, int sx, int
sy);
            void drawLine(int x1, int y1, int x2, int y2);
            void clrLine(int x1, int y1, int x2, int y2);
            void drawRect(int x1, int y1, int x2, int y2);
            void clrRect(int x1, int y1, int x2, int y2);
            void drawRoundRect(int x1, int y1, int x2, int y2);
            void clrRoundRect(int x1, int y1, int x2, int y2);
            void drawCircle(int x, int y, int radius);
            void clrCircle(int x, int y, int radius);

      protected:
            GPIO_TypeDef      *SPI_Port, *LCD_Port;
            int                     Pin_MOSI, Pin_SCK, Pin_CD, Pin_RST,
Pin_CS;
            _current_font     cfont;
            uint8_t                 scrbuf[504];
            bool              _sleep;
            int                     _contrast;

            int abs (int x);
            void _LCD_Write(unsigned char data, unsigned char mode);
            void _print_char(unsigned char c, int x, int row);
            void _convert_float(char *buf, double num, int width, byte
prec);
            void drawHLine(int x, int y, int l);
            void clrHLine(int x, int y, int l);
            void drawVLine(int x, int y, int l);
            void clrVLine(int x, int y, int l);
};

#endif
```

LCD5110_Graph.cpp

```
#include "LCD5110_Graph.h"


LCD5110::LCD5110(GPIO_TypeDef *_SPI_Port,int SCK, int MOSI,
GPIO_TypeDef *_LCD_Port, int DC, int RST, int CS)
{
	SPI_Port = _SPI_Port;
	LCD_Port = _LCD_Port;
	Pin_CD = DC;
	Pin_CS = CS;
	Pin_MOSI = MOSI;
	Pin_RST = RST;
	Pin_SCK = SCK;

}

void LCD5110::_convert_float(char *buf, double num, int width, byte
prec)
{
	char format[10];

	sprintf(format, "%%%i.%if", width, prec);
	sprintf(buf, format, num);
}

int LCD5110::abs (int x){
	if(x < 0){
		return (-x);
	}
	return (x);
}

void LCD5110::_LCD_Write(unsigned char data, unsigned char mode)
{
	GPIO_ResetBits(LCD_Port, Pin_CS);

	if (mode==LCD_COMMAND)
		GPIO_ResetBits(LCD_Port, Pin_CD);
	else
		GPIO_SetBits(LCD_Port, Pin_CD);
```

```
	Fn_SPIx_Transfer(data);
	GPIO_SetBits(LCD_Port, Pin_CS);
}

void LCD5110::InitLCD(int contrast)
{
	GPIO_InitTypeDef GPIO_InitStruct;
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO | RCC_APB2Periph_GPIOA
| RCC_APB2Periph_GPIOE, ENABLE);
	GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
GPIO_Pin_3;
 GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
 GPIO_InitStruct.GPIO_Speed = GPIO_Speed_10MHz;
 GPIO_Init(GPIOA, &GPIO_InitStruct);

	GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10
| GPIO_Pin_11;
 GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
 GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
 GPIO_Init(GPIOE, &GPIO_InitStruct);

	Fn_SPI_Init();
	Fn_DELAY_ms(10);
	Fn_SPIx_Transfer(0x00);

	if (contrast>0x7F)
		contrast=0x7F;
	if (contrast<0)
		contrast=0;

	//resetLCD;
	GPIO_ResetBits(LCD_Port, Pin_RST);
	Fn_DELAY_ms(100);
	GPIO_SetBits(LCD_Port, Pin_RST);
	Fn_DELAY_ms(100);

	_LCD_Write(PCD8544_FUNCTIONSET | PCD8544_EXTENDEDINSTRUCTION,
LCD_COMMAND);
	_LCD_Write(PCD8544_SETVOP | contrast, LCD_COMMAND);
	_LCD_Write(PCD8544_SETTEMP | LCD_TEMP, LCD_COMMAND);
	_LCD_Write(PCD8544_SETBIAS | LCD_BIAS, LCD_COMMAND);
	_LCD_Write(PCD8544_FUNCTIONSET, LCD_COMMAND);
	_LCD_Write(PCD8544_SETYADDR, LCD_COMMAND);
```

```
	_LCD_Write(PCD8544_SETXADDR, LCD_COMMAND);
	_LCD_Write(PCD8544_DISPLAYCONTROL | PCD8544_DISPLAYNORMAL,
LCD_COMMAND);

	clrScr();
	update();
	cfont.font=0;
	_sleep=false;
	_contrast=contrast;
}

void LCD5110::setContrast(int contrast)
{
	if (contrast>0x7F)
		contrast=0x7F;
	if (contrast<0)
		contrast=0;
	_LCD_Write(PCD8544_FUNCTIONSET | PCD8544_EXTENDEDINSTRUCTION,
LCD_COMMAND);
	_LCD_Write(PCD8544_SETVOP | contrast, LCD_COMMAND);
	_LCD_Write(PCD8544_FUNCTIONSET, LCD_COMMAND);
	_contrast=contrast;
}

void LCD5110::enableSleep()
{
	_sleep = true;
	_LCD_Write(PCD8544_SETYADDR, LCD_COMMAND);
	_LCD_Write(PCD8544_SETXADDR, LCD_COMMAND);
	for (int b=0; b<504; b++)
		_LCD_Write(0, LCD_DATA);
	_LCD_Write(PCD8544_FUNCTIONSET | PCD8544_POWERDOWN, LCD_COMMAND);
}

void LCD5110::disableSleep()
{
	_sleep = false;
	_LCD_Write(PCD8544_FUNCTIONSET | PCD8544_EXTENDEDINSTRUCTION,
LCD_COMMAND);
	_LCD_Write(PCD8544_SETVOP | _contrast, LCD_COMMAND);
	_LCD_Write(PCD8544_SETTEMP | LCD_TEMP, LCD_COMMAND);
	_LCD_Write(PCD8544_SETBIAS | LCD_BIAS, LCD_COMMAND);
	_LCD_Write(PCD8544_FUNCTIONSET, LCD_COMMAND);
```

```
      _LCD_Write(PCD8544_DISPLAYCONTROL | PCD8544_DISPLAYNORMAL,
LCD_COMMAND);
      update();
}

void LCD5110::update()
{
      if (_sleep==false)
      {
            _LCD_Write(PCD8544_SETYADDR, LCD_COMMAND);
            _LCD_Write(PCD8544_SETXADDR, LCD_COMMAND);
            for (int b=0; b<504; b++)
                  _LCD_Write(scrbuf[b], LCD_DATA);
      }
}

void LCD5110::clrScr()
{
      for (int c=0; c<504; c++)
            scrbuf[c]=0;
}

void LCD5110::fillScr()
{
      for (int c=0; c<504; c++)
            scrbuf[c]=255;
}

void LCD5110::invert(bool mode)
{
      if (mode==true)
            _LCD_Write(PCD8544_DISPLAYCONTROL |
PCD8544_DISPLAYINVERTED, LCD_COMMAND);
      else
            _LCD_Write(PCD8544_DISPLAYCONTROL | PCD8544_DISPLAYNORMAL,
LCD_COMMAND);
}

void LCD5110::setPixel(uint16_t x, uint16_t y)
{
      int by, bi;

      if ((x>=0) and (x<84) and (y>=0) and (y<48))
```

```
	{
		by=((y/8)*84)+x;
		bi=y % 8;

		scrbuf[by]=scrbuf[by] | (1<<bi);
	}
}

void LCD5110::clrPixel(uint16_t x, uint16_t y)
{
	int by, bi;

	if ((x>=0) and (x<84) and (y>=0) and (y<48))
	{
		by=((y/8)*84)+x;
		bi=y % 8;

		scrbuf[by]=scrbuf[by] & ~(1<<bi);
	}
}

void LCD5110::invPixel(uint16_t x, uint16_t y)
{
	int by, bi;

	if ((x>=0) and (x<84) and (y>=0) and (y<48))
	{
		by=((y/8)*84)+x;
		bi=y % 8;

		if ((scrbuf[by] & (1<<bi))==0)
			scrbuf[by]=scrbuf[by] | (1<<bi);
		else
			scrbuf[by]=scrbuf[by] & ~(1<<bi);
	}
}

void LCD5110::invertText(bool mode)
{
	if (mode==true)
		cfont.inverted=1;
	else
		cfont.inverted=0;
```

```
}

void LCD5110::print(char *st, int x, int y)
{
	unsigned char ch;
	int stl;

	stl = strlen(st);
	if (x == RIGHT)
		x = 84-(stl*cfont.x_size);
	if (x == CENTER)
		x = (84-(stl*cfont.x_size))/2;

	for (int cnt=0; cnt<stl; cnt++)
			_print_char(*st++, x + (cnt*(cfont.x_size)), y);
}

void LCD5110::print(const char *st, int x, int y)
{
	unsigned char ch;
	int stl;

	stl = strlen(st);
	if (x == RIGHT)
		x = 84-(stl*cfont.x_size);
	if (x == CENTER)
		x = (84-(stl*cfont.x_size))/2;

	for (int cnt=0; cnt<stl; cnt++)
			_print_char(*st++, x + (cnt*(cfont.x_size)), y);
}


void LCD5110::printNumI(long num, int x, int y, int length, char
filler)
{
	char buf[25];
	char st[27];
	bool neg=false;
	int c=0, f=0;

	if (num==0)
	{
```

```
            if (length!=0)
            {
                  for (c=0; c<(length-1); c++)
                        st[c]=filler;
                  st[c]=48;
                  st[c+1]=0;
            }
            else
            {
                  st[0]=48;
                  st[1]=0;
            }
      }
      else
      {
            if (num<0)
            {
                  neg=true;
                  num=-num;
            }

            while (num>0)
            {
                  buf[c]=48+(num % 10);
                  c++;
                  num=(num-(num % 10))/10;
            }
            buf[c]=0;

            if (neg)
            {
                  st[0]=45;
            }

            if (length>(c+neg))
            {
                  for (int i=0; i<(length-c-neg); i++)
                  {
                        st[i+neg]=filler;
                        f++;
                  }
            }
```

```
            for (int i=0; i<c; i++)
            {
                  st[i+neg+f]=buf[c-i-1];
            }
            st[c+neg+f]=0;

      }

      print(st,x,y);
}

void LCD5110::printNumF(double num, byte dec, int x, int y, char
divider, int length, char filler)
{
      char st[27];
      bool neg=false;

      if (num<0)
            neg = true;

      _convert_float(st, num, length, dec);

      if (divider != '.')
      {
            for (int i=0; i<sizeof(st); i++)
                  if (st[i]=='.')
                        st[i]=divider;
      }

      if (filler != ' ')
      {
            if (neg)
            {
                  st[0]='-';
                  for (int i=1; i<sizeof(st); i++)
                        if ((st[i]==' ') || (st[i]=='-'))
                              st[i]=filler;
            }
            else
            {
                  for (int i=0; i<sizeof(st); i++)
                        if (st[i]==' ')
                              st[i]=filler;
```

```
            }
      }

      print(st,x,y);
}

void LCD5110::_print_char(unsigned char c, int x, int y)
{
      if ((cfont.y_size % 8) == 0)
      {
            int font_idx = ((c -
cfont.offset)*(cfont.x_size*(cfont.y_size/8)))+4;
            for (int rowcnt=0; rowcnt<(cfont.y_size/8); rowcnt++)
            {
                  for(int cnt=0; cnt<cfont.x_size; cnt++)
                  {
                        for (int b=0; b<8; b++)
                              if
((fontbyte(font_idx+cnt+(rowcnt*cfont.x_size)) & (1<<b))!=0)
                                    if (cfont.inverted==0)
                                          setPixel(x+cnt,
y+(rowcnt*8)+b);
                                    else
                                          clrPixel(x+cnt,
y+(rowcnt*8)+b);
                              else
                                    if (cfont.inverted==0)
                                          clrPixel(x+cnt,
y+(rowcnt*8)+b);
                                    else
                                          setPixel(x+cnt,
y+(rowcnt*8)+b);
                  }
            }
      }
      else
      {
            int font_idx = ((c -
cfont.offset)*((cfont.x_size*cfont.y_size/8)))+4;
            int cbyte=fontbyte(font_idx);
            int cbit=7;
            for (int cx=0; cx<cfont.x_size; cx++)
            {
```

```
                  for (int cy=0; cy<cfont.y_size; cy++)
                  {
                        if ((cbyte & (1<<cbit)) != 0)
                              if (cfont.inverted==0)
                                    setPixel(x+cx, y+cy);
                              else
                                    clrPixel(x+cx, y+cy);
                        else
                              if (cfont.inverted==0)
                                    clrPixel(x+cx, y+cy);
                              else
                                    setPixel(x+cx, y+cy);
                        cbit--;
                        if (cbit<0)
                        {
                              cbit=7;
                              font_idx++;
                              cbyte=fontbyte(font_idx);
                        }
                  }
            }
      }
}

void LCD5110::setFont(uint8_t* font)
{
      cfont.font=font;
      cfont.x_size=fontbyte(0);
      cfont.y_size=fontbyte(1);
      cfont.offset=fontbyte(2);
      cfont.numchars=fontbyte(3);
      cfont.inverted=0;
}

void LCD5110::drawHLine(int x, int y, int l)
{
      int by, bi;

      if ((x>=0) and (x<84) and (y>=0) and (y<48))
      {
            for (int cx=0; cx<l; cx++)
            {
                  by=((y/8)*84)+x;
```

```
                  bi=y % 8;

                  scrbuf[by+cx] |= (1<<bi);
            }
      }
}

void LCD5110::clrHLine(int x, int y, int l)
{
      int by, bi;

      if ((x>=0) and (x<84) and (y>=0) and (y<48))
      {
            for (int cx=0; cx<l; cx++)
            {
                  by=((y/8)*84)+x;
                  bi=y % 8;

                  scrbuf[by+cx] &= ~(1<<bi);
            }
      }
}

void LCD5110::drawVLine(int x, int y, int l)
{
      int by, bi;

      if ((x>=0) and (x<84) and (y>=0) and (y<48))
      {
            for (int cy=0; cy<l; cy++)
            {
                  setPixel(x, y+cy);
            }
      }
}

void LCD5110::clrVLine(int x, int y, int l)
{
      int by, bi;

      if ((x>=0) and (x<84) and (y>=0) and (y<48))
      {
            for (int cy=0; cy<l; cy++)
```

```
            {
                  clrPixel(x, y+cy);
            }
      }
}

void LCD5110::drawLine(int x1, int y1, int x2, int y2)
{
      int tmp;
      double delta, tx, ty;
      double m, b, dx, dy;

      if (((x2-x1)<0))
      {
            tmp=x1;
            x1=x2;
            x2=tmp;
            tmp=y1;
            y1=y2;
            y2=tmp;
      }
 if (((y2-y1)<0))
      {
            tmp=x1;
            x1=x2;
            x2=tmp;
            tmp=y1;
            y1=y2;
            y2=tmp;
      }

      if (y1==y2)
      {
            if (x1>x2)
            {
                  tmp=x1;
                  x1=x2;
                  x2=tmp;
            }
            drawHLine(x1, y1, x2-x1);
      }
      else if (x1==x2)
      {
```

```
            if (y1>y2)
            {
                  tmp=y1;
                  y1=y2;
                  y2=tmp;
            }
            drawVLine(x1, y1, y2-y1);
      }
      else if (abs(x2-x1)>abs(y2-y1))
      {
            delta=(double(y2-y1)/double(x2-x1));
            ty=double(y1);
            if (x1>x2)
            {
                  for (int i=x1; i>=x2; i--)
                  {
                        setPixel(i, int(ty+0.5));
            ty=ty-delta;
                  }
            }
            else
            {
                  for (int i=x1; i<=x2; i++)
                  {
                        setPixel(i, int(ty+0.5));
            ty=ty+delta;
                  }
            }
      }
      else
      {
            delta=(float(x2-x1)/float(y2-y1));
            tx=float(x1);
if (y1>y2)
{
                  for (int i=y2+1; i>y1; i--)
                  {
                        setPixel(int(tx+0.5), i);
            tx=tx+delta;
                  }
}
else
{
```

```
                  for (int i=y1; i<y2+1; i++)
                  {
                        setPixel(int(tx+0.5), i);
            tx=tx+delta;
                  }
 }
      }

}

void LCD5110::clrLine(int x1, int y1, int x2, int y2)
{
      int tmp;
      double delta, tx, ty;
      double m, b, dx, dy;

      if (((x2-x1)<0))
      {
            tmp=x1;
            x1=x2;
            x2=tmp;
            tmp=y1;
            y1=y2;
            y2=tmp;
      }
 if (((y2-y1)<0))
      {
            tmp=x1;
            x1=x2;
            x2=tmp;
            tmp=y1;
            y1=y2;
            y2=tmp;
      }

      if (y1==y2)
      {
            if (x1>x2)
            {
                  tmp=x1;
                  x1=x2;
                  x2=tmp;
            }
```

```
            clrHLine(x1, y1, x2-x1);
      }
      else if (x1==x2)
      {
            if (y1>y2)
            {
                  tmp=y1;
                  y1=y2;
                  y2=tmp;
            }
            clrVLine(x1, y1, y2-y1);
      }
      else if (abs(x2-x1)>abs(y2-y1))
      {
            delta=(double(y2-y1)/double(x2-x1));
            ty=double(y1);
            if (x1>x2)
            {
                  for (int i=x1; i>=x2; i--)
                  {
                        clrPixel(i, int(ty+0.5));
            ty=ty-delta;
                  }
            }
            else
            {
                  for (int i=x1; i<=x2; i++)
                  {
                        clrPixel(i, int(ty+0.5));
            ty=ty+delta;
                  }
            }
      }
      else
      {
            delta=(float(x2-x1)/float(y2-y1));
            tx=float(x1);
if (y1>y2)
{
                  for (int i=y2+1; i>y1; i--)
                  {
                        clrPixel(int(tx+0.5), i);
            tx=tx+delta;
```

```
                  }
 }
 else
 {
                  for (int i=y1; i<y2+1; i++)
                  {
                        clrPixel(int(tx+0.5), i);
            tx=tx+delta;
                  }
 }
      }

}

void LCD5110::drawRect(int x1, int y1, int x2, int y2)
{
      int tmp;

      if (x1>x2)
      {
            tmp=x1;
            x1=x2;
            x2=tmp;
      }
      if (y1>y2)
      {
            tmp=y1;
            y1=y2;
            y2=tmp;
      }

      drawHLine(x1, y1, x2-x1);
      drawHLine(x1, y2, x2-x1);
      drawVLine(x1, y1, y2-y1);
      drawVLine(x2, y1, y2-y1+1);
}

void LCD5110::clrRect(int x1, int y1, int x2, int y2)
{
      int tmp;

      if (x1>x2)
      {
```

```
            tmp=x1;
            x1=x2;
            x2=tmp;
      }
      if (y1>y2)
      {
            tmp=y1;
            y1=y2;
            y2=tmp;
      }

      clrHLine(x1, y1, x2-x1);
      clrHLine(x1, y2, x2-x1);
      clrVLine(x1, y1, y2-y1);
      clrVLine(x2, y1, y2-y1+1);
}

void LCD5110::drawRoundRect(int x1, int y1, int x2, int y2)
{
      int tmp;

      if (x1>x2)
      {
            tmp=x1;
            x1=x2;
            x2=tmp;
      }
      if (y1>y2)
      {
            tmp=y1;
            y1=y2;
            y2=tmp;
      }
      if ((x2-x1)>4 && (y2-y1)>4)
      {
            setPixel(x1+1,y1+1);
            setPixel(x2-1,y1+1);
            setPixel(x1+1,y2-1);
            setPixel(x2-1,y2-1);
            drawHLine(x1+2, y1, x2-x1-3);
            drawHLine(x1+2, y2, x2-x1-3);
            drawVLine(x1, y1+2, y2-y1-3);
            drawVLine(x2, y1+2, y2-y1-3);
```

```
	}
}

void LCD5110::clrRoundRect(int x1, int y1, int x2, int y2)
{
	int tmp;

	if (x1>x2)
	{
		tmp=x1;
		x1=x2;
		x2=tmp;
	}
	if (y1>y2)
	{
		tmp=y1;
		y1=y2;
		y2=tmp;
	}
	if ((x2-x1)>4 && (y2-y1)>4)
	{
		clrPixel(x1+1,y1+1);
		clrPixel(x2-1,y1+1);
		clrPixel(x1+1,y2-1);
		clrPixel(x2-1,y2-1);
		clrHLine(x1+2, y1, x2-x1-3);
		clrHLine(x1+2, y2, x2-x1-3);
		clrVLine(x1, y1+2, y2-y1-3);
		clrVLine(x2, y1+2, y2-y1-3);
	}
}

void LCD5110::drawCircle(int x, int y, int radius)
{
	int f = 1 - radius;
	int ddF_x = 1;
	int ddF_y = -2 * radius;
	int x1 = 0;
	int y1 = radius;
	char ch, cl;

	setPixel(x, y + radius);
	setPixel(x, y - radius);
```

```
	setPixel(x + radius, y);
	setPixel(x - radius, y);

	while(x1 < y1)
	{
		if(f >= 0)
		{
			y1--;
			ddF_y += 2;
			f += ddF_y;
		}
		x1++;
		ddF_x += 2;
		f += ddF_x;
		setPixel(x + x1, y + y1);
		setPixel(x - x1, y + y1);
		setPixel(x + x1, y - y1);
		setPixel(x - x1, y - y1);
		setPixel(x + y1, y + x1);
		setPixel(x - y1, y + x1);
		setPixel(x + y1, y - x1);
		setPixel(x - y1, y - x1);
	}
}

void LCD5110::clrCircle(int x, int y, int radius)
{
	int f = 1 - radius;
	int ddF_x = 1;
	int ddF_y = -2 * radius;
	int x1 = 0;
	int y1 = radius;
	char ch, cl;

	clrPixel(x, y + radius);
	clrPixel(x, y - radius);
	clrPixel(x + radius, y);
	clrPixel(x - radius, y);

	while(x1 < y1)
	{
		if(f >= 0)
		{
```

```
                  y1--;
                  ddF_y += 2;
                  f += ddF_y;
            }
            x1++;
            ddF_x += 2;
            f += ddF_x;
            clrPixel(x + x1, y + y1);
            clrPixel(x - x1, y + y1);
            clrPixel(x + x1, y - y1);
            clrPixel(x - x1, y - y1);
            clrPixel(x + y1, y + x1);
            clrPixel(x - y1, y + x1);
            clrPixel(x + y1, y - x1);
            clrPixel(x - y1, y - x1);
      }
}

void LCD5110::drawBitmap(int x, int y, uint8_t* bitmap, int sx, int sy)
{
      int bit;
      byte data;

      for (int cy=0; cy<sy; cy++)
      {
            bit= cy % 8;
            for(int cx=0; cx<sx; cx++)
            {
                  data=bitmapbyte(cx+((cy/8)*sx));
                  if ((data & (1<<bit))>0)
                        setPixel(x+cx, y+cy);
                  else
                        clrPixel(x+cx, y+cy);
            }
      }
}
```

DefaultFonts.c

```
unsigned char SmallFont[] =
{
0x06, 0x08, 0x20, 0x5f,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // sp
0x00, 0x00, 0x00, 0x2f, 0x00, 0x00, // !
```

```
0x00, 0x00, 0x07, 0x00, 0x07, 0x00, // "
0x00, 0x14, 0x7f, 0x14, 0x7f, 0x14, // #
0x00, 0x24, 0x2a, 0x7f, 0x2a, 0x12, // $
0x00, 0x23, 0x13, 0x08, 0x64, 0x62, // %
0x00, 0x36, 0x49, 0x55, 0x22, 0x50, // &
0x00, 0x00, 0x05, 0x03, 0x00, 0x00, // '
0x00, 0x00, 0x1c, 0x22, 0x41, 0x00, // (
0x00, 0x00, 0x41, 0x22, 0x1c, 0x00, // )
0x00, 0x14, 0x08, 0x3E, 0x08, 0x14, // *
0x00, 0x08, 0x08, 0x3E, 0x08, 0x08, // +
0x00, 0x00, 0x00, 0xA0, 0x60, 0x00, // ,
0x00, 0x08, 0x08, 0x08, 0x08, 0x08, // -
0x00, 0x00, 0x60, 0x60, 0x00, 0x00, // .
0x00, 0x20, 0x10, 0x08, 0x04, 0x02, // /

0x00, 0x3E, 0x51, 0x49, 0x45, 0x3E, // 0
0x00, 0x00, 0x42, 0x7F, 0x40, 0x00, // 1
0x00, 0x42, 0x61, 0x51, 0x49, 0x46, // 2
0x00, 0x21, 0x41, 0x45, 0x4B, 0x31, // 3
0x00, 0x18, 0x14, 0x12, 0x7F, 0x10, // 4
0x00, 0x27, 0x45, 0x45, 0x45, 0x39, // 5
0x00, 0x3C, 0x4A, 0x49, 0x49, 0x30, // 6
0x00, 0x01, 0x71, 0x09, 0x05, 0x03, // 7
0x00, 0x36, 0x49, 0x49, 0x49, 0x36, // 8
0x00, 0x06, 0x49, 0x49, 0x29, 0x1E, // 9
0x00, 0x00, 0x36, 0x36, 0x00, 0x00, // :
0x00, 0x00, 0x56, 0x36, 0x00, 0x00, // ;
0x00, 0x08, 0x14, 0x22, 0x41, 0x00, // <
0x00, 0x14, 0x14, 0x14, 0x14, 0x14, // =
0x00, 0x00, 0x41, 0x22, 0x14, 0x08, // >
0x00, 0x02, 0x01, 0x51, 0x09, 0x06, // ?

0x00, 0x32, 0x49, 0x59, 0x51, 0x3E, // @
0x00, 0x7C, 0x12, 0x11, 0x12, 0x7C, // A
0x00, 0x7F, 0x49, 0x49, 0x49, 0x36, // B
0x00, 0x3E, 0x41, 0x41, 0x41, 0x22, // C
0x00, 0x7F, 0x41, 0x41, 0x22, 0x1C, // D
0x00, 0x7F, 0x49, 0x49, 0x49, 0x41, // E
0x00, 0x7F, 0x09, 0x09, 0x09, 0x01, // F
0x00, 0x3E, 0x41, 0x49, 0x49, 0x7A, // G
0x00, 0x7F, 0x08, 0x08, 0x08, 0x7F, // H
0x00, 0x00, 0x41, 0x7F, 0x41, 0x00, // I
0x00, 0x20, 0x40, 0x41, 0x3F, 0x01, // J
```

```
0x00, 0x7F, 0x08, 0x14, 0x22, 0x41, // K
0x00, 0x7F, 0x40, 0x40, 0x40, 0x40, // L
0x00, 0x7F, 0x02, 0x0C, 0x02, 0x7F, // M
0x00, 0x7F, 0x04, 0x08, 0x10, 0x7F, // N
0x00, 0x3E, 0x41, 0x41, 0x41, 0x3E, // O

0x00, 0x7F, 0x09, 0x09, 0x09, 0x06, // P
0x00, 0x3E, 0x41, 0x51, 0x21, 0x5E, // Q
0x00, 0x7F, 0x09, 0x19, 0x29, 0x46, // R
0x00, 0x46, 0x49, 0x49, 0x49, 0x31, // S
0x00, 0x01, 0x01, 0x7F, 0x01, 0x01, // T
0x00, 0x3F, 0x40, 0x40, 0x40, 0x3F, // U
0x00, 0x1F, 0x20, 0x40, 0x20, 0x1F, // V
0x00, 0x3F, 0x40, 0x38, 0x40, 0x3F, // W
0x00, 0x63, 0x14, 0x08, 0x14, 0x63, // X
0x00, 0x07, 0x08, 0x70, 0x08, 0x07, // Y
0x00, 0x61, 0x51, 0x49, 0x45, 0x43, // Z
0x00, 0x00, 0x7F, 0x41, 0x41, 0x00, // [
0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, // Backslash (Checker pattern)
0x00, 0x00, 0x41, 0x41, 0x7F, 0x00, // ]
0x00, 0x04, 0x02, 0x01, 0x02, 0x04, // ^
0x00, 0x40, 0x40, 0x40, 0x40, 0x40, // _

0x00, 0x00, 0x03, 0x05, 0x00, 0x00, // `
0x00, 0x20, 0x54, 0x54, 0x54, 0x78, // a
0x00, 0x7F, 0x48, 0x44, 0x44, 0x38, // b
0x00, 0x38, 0x44, 0x44, 0x44, 0x20, // c
0x00, 0x38, 0x44, 0x44, 0x48, 0x7F, // d
0x00, 0x38, 0x54, 0x54, 0x54, 0x18, // e
0x00, 0x08, 0x7E, 0x09, 0x01, 0x02, // f
0x00, 0x18, 0xA4, 0xA4, 0xA4, 0x7C, // g
0x00, 0x7F, 0x08, 0x04, 0x04, 0x78, // h
0x00, 0x00, 0x44, 0x7D, 0x40, 0x00, // i
0x00, 0x40, 0x80, 0x84, 0x7D, 0x00, // j
0x00, 0x7F, 0x10, 0x28, 0x44, 0x00, // k
0x00, 0x00, 0x41, 0x7F, 0x40, 0x00, // l
0x00, 0x7C, 0x04, 0x18, 0x04, 0x78, // m
0x00, 0x7C, 0x08, 0x04, 0x04, 0x78, // n
0x00, 0x38, 0x44, 0x44, 0x44, 0x38, // o

0x00, 0xFC, 0x24, 0x24, 0x24, 0x18, // p
0x00, 0x18, 0x24, 0x24, 0x18, 0xFC, // q
0x00, 0x7C, 0x08, 0x04, 0x04, 0x08, // r
```

```
0x00, 0x48, 0x54, 0x54, 0x54, 0x20, // s
0x00, 0x04, 0x3F, 0x44, 0x40, 0x20, // t
0x00, 0x3C, 0x40, 0x40, 0x20, 0x7C, // u
0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C, // v
0x00, 0x3C, 0x40, 0x30, 0x40, 0x3C, // w
0x00, 0x44, 0x28, 0x10, 0x28, 0x44, // x
0x00, 0x1C, 0xA0, 0xA0, 0xA0, 0x7C, // y
0x00, 0x44, 0x64, 0x54, 0x4C, 0x44, // z
0x00, 0x00, 0x10, 0x7C, 0x82, 0x00, // {
0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, // |
0x00, 0x00, 0x82, 0x7C, 0x10, 0x00, // }
0x00, 0x00, 0x06, 0x09, 0x09, 0x06 // ~ (Degrees)
};

const unsigned char MediumNumbers[] =
{
0x0c, 0x10, 0x2d, 0x0d,
0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00,
0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x01, 0x00, 0x00,
// -
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00,
// .
0x00, 0x00, 0x02, 0x86, 0x86, 0x86, 0x86, 0x86, 0x86, 0x02, 0x00, 0x00,
0x00, 0x00, 0x81, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0x81, 0x00, 0x00,
// /
0x00, 0xfc, 0x7a, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x7a, 0xfc, 0x00,
0x00, 0x7e, 0xbc, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xbc, 0x7e, 0x00,
// 0
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0xfc, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3c, 0x7e, 0x00,
// 1
0x00, 0x00, 0x02, 0x86, 0x86, 0x86, 0x86, 0x86, 0x86, 0x7a, 0xfc, 0x00,
0x00, 0x7e, 0xbd, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0x81, 0x00, 0x00,
// 2
0x00, 0x00, 0x02, 0x86, 0x86, 0x86, 0x86, 0x86, 0x86, 0x7a, 0xfc, 0x00,
0x00, 0x00, 0x81, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xbd, 0x7e, 0x00,
// 3
0x00, 0xfc, 0x78, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x78, 0xfc, 0x00,
0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x3d, 0x7e, 0x00,
// 4
0x00, 0xfc, 0x7a, 0x86, 0x86, 0x86, 0x86, 0x86, 0x86, 0x02, 0x00, 0x00,
0x00, 0x00, 0x81, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xbd, 0x7e, 0x00,
```

```
// 5
0x00, 0xfc, 0x7a, 0x86, 0x86, 0x86, 0x86, 0x86, 0x86, 0x02, 0x00, 0x00,
0x00, 0x7e, 0xbd, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xbd, 0x7e, 0x00,
// 6
0x00, 0x00, 0x02, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x7a, 0xfc, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3c, 0x7e, 0x00,
// 7
0x00, 0xfc, 0x7a, 0x86, 0x86, 0x86, 0x86, 0x86, 0x86, 0x7a, 0xfc, 0x00,
0x00, 0x7e, 0xbd, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xbd, 0x7e, 0x00,
// 8
0x00, 0xfc, 0x7a, 0x86, 0x86, 0x86, 0x86, 0x86, 0x86, 0x7a, 0xfc, 0x00,
0x00, 0x00, 0x81, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xbd, 0x7e, 0x00,
// 9
};

const unsigned char BigNumbers[] =
{
0x0e, 0x18, 0x2d, 0x0d,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x10, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0x38, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // -
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0xe0, 0xe0,
0x40, 0x00, 0x00, 0x00, 0x00, 0x00, // .
0x00, 0x00, 0x02, 0x06, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x06, 0x02,
0x00, 0x00, 0x00, 0x00, 0x10, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0x38, 0x10, 0x00, 0x00, 0x00, 0x00, 0x80, 0xc0, 0xe0, 0xe0, 0xe0, 0xe0,
0xe0, 0xe0, 0xc0, 0x80, 0x00, 0x00, // /
0x00, 0xfc, 0xfa, 0xf6, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0xf6, 0xfa,
0xfc, 0x00, 0x00, 0xef, 0xc7, 0x83, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x83, 0xc7, 0xef, 0x00, 0x00, 0x7f, 0xbf, 0xdf, 0xe0, 0xe0, 0xe0, 0xe0,
0xe0, 0xe0, 0xdf, 0xbf, 0x7f, 0x00, // 0
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf0, 0xf8,
0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x83, 0xc7, 0xef, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x1f, 0x3f, 0x7f, 0x00, // 1
0x00, 0x00, 0x02, 0x06, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0xf6, 0xfa,
0xfc, 0x00, 0x00, 0xe0, 0xd0, 0xb8, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0x3b, 0x17, 0x0f, 0x00, 0x00, 0x7f, 0xbf, 0xdf, 0xe0, 0xe0, 0xe0, 0xe0,
0xe0, 0xe0, 0xc0, 0x80, 0x00, 0x00, // 2
0x00, 0x00, 0x02, 0x06, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0xf6, 0xfa,
```

```
0xfc, 0x00, 0x00, 0x00, 0x10, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0xbb, 0xd7, 0xef, 0x00, 0x00, 0x00, 0x80, 0xc0, 0xe0, 0xe0, 0xe0, 0xe0,
0xe0, 0xe0, 0xdf, 0xbf, 0x7f, 0x00, // 3
0x00, 0xfc, 0xf8, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf0, 0xf8,
0xfc, 0x00, 0x00, 0x0f, 0x17, 0x3b, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0xbb, 0xd7, 0xef, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x1f, 0x3f, 0x7f, 0x00, // 4
0x00, 0xfc, 0xfa, 0xf6, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x06, 0x02,
0x00, 0x00, 0x00, 0x0f, 0x17, 0x3b, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0xb8, 0xd0, 0xe0, 0x00, 0x00, 0x00, 0x80, 0xc0, 0xe0, 0xe0, 0xe0, 0xe0,
0xe0, 0xe0, 0xdf, 0xbf, 0x7f, 0x00, // 5
0x00, 0xfc, 0xfa, 0xf6, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x06, 0x02,
0x00, 0x00, 0x00, 0xef, 0xd7, 0xbb, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0xb8, 0xd0, 0xe0, 0x00, 0x00, 0x7f, 0xbf, 0xdf, 0xe0, 0xe0, 0xe0, 0xe0,
0xe0, 0xe0, 0xdf, 0xbf, 0x7f, 0x00, // 6
0x00, 0x00, 0x02, 0x06, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0xf6, 0xfa,
0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x83, 0xc7, 0xef, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x1f, 0x3f, 0x7f, 0x00, // 7
0x00, 0xfc, 0xfa, 0xf6, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0xf6, 0xfa,
0xfc, 0x00, 0x00, 0xef, 0xd7, 0xbb, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0xbb, 0xd7, 0xef, 0x00, 0x00, 0x7f, 0xbf, 0xdf, 0xe0, 0xe0, 0xe0, 0xe0,
0xe0, 0xe0, 0xdf, 0xbf, 0x7f, 0x00, // 8
0x00, 0xfc, 0xfa, 0xf6, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0x0e, 0xf6, 0xfa,
0xfc, 0x00, 0x00, 0x0f, 0x17, 0x3b, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0xbb, 0xd7, 0xef, 0x00, 0x00, 0x00, 0x80, 0xc0, 0xe0, 0xe0, 0xe0, 0xe0,
0xe0, 0xe0, 0xdf, 0xbf, 0x7f, 0x00, // 9
};

unsigned char TinyFont[] =
{
0x04, 0x06, 0x20, 0x5f,
0x00, 0x00, 0x00, 0x03, 0xa0, 0x00, 0xc0, 0x0c, 0x00, 0xf9, 0x4f, 0x80,
0x6b, 0xeb, 0x00, 0x98, 0x8c, 0x80, 0x52, 0xa5, 0x80, 0x03, 0x00, 0x00,
// Space, !"#$%&'
0x01, 0xc8, 0x80, 0x89, 0xc0, 0x00, 0x50, 0x85, 0x00, 0x21, 0xc2, 0x00,
0x08, 0x40, 0x00, 0x20, 0x82, 0x00, 0x00, 0x20, 0x00, 0x18, 0x8c, 0x00,
// ()*+,-./
0xfa, 0x2f, 0x80, 0x4b, 0xe0, 0x80, 0x5a, 0x66, 0x80, 0x8a, 0xa5, 0x00,
0xe0, 0x8f, 0x80, 0xea, 0xab, 0x00, 0x72, 0xa9, 0x00, 0x9a, 0x8c, 0x00,
// 01234567
0xfa, 0xaf, 0x80, 0x4a, 0xa7, 0x00, 0x01, 0x40, 0x00, 0x09, 0x40, 0x00,
0x21, 0x48, 0x80, 0x51, 0x45, 0x00, 0x89, 0x42, 0x00, 0x42, 0x66, 0x00,
```

```
// 89:;<=>?
0x72, 0xa6, 0x80, 0x7a, 0x87, 0x80, 0xfa, 0xa5, 0x00, 0x72, 0x25, 0x00,
0xfa, 0x27, 0x00, 0xfa, 0xa8, 0x80, 0xfa, 0x88, 0x00, 0x72, 0x2b, 0x00,
// @ABCDEFG
0xf8, 0x8f, 0x80, 0x8b, 0xe8, 0x80, 0x8b, 0xe8, 0x00, 0xf8, 0x8d, 0x80,
0xf8, 0x20, 0x80, 0xf9, 0x0f, 0x80, 0xf9, 0xcf, 0x80, 0x72, 0x27, 0x00,
// HIJKLMNO
0xfa, 0x84, 0x00, 0x72, 0x27, 0x40, 0xfa, 0x85, 0x80, 0x4a, 0xa9, 0x00,
0x83, 0xe8, 0x00, 0xf0, 0x2f, 0x00, 0xe0, 0x6e, 0x00, 0xf0, 0xef, 0x00,
// PQRSTUVW
0xd8, 0x8d, 0x80, 0xc0, 0xec, 0x00, 0x9a, 0xac, 0x80, 0x03, 0xe8, 0x80,
0xc0, 0x81, 0x80, 0x8b, 0xe0, 0x00, 0x42, 0x04, 0x00, 0x08, 0x20, 0x80,
// XYZ[\]^_
0x02, 0x04, 0x00, 0x31, 0x23, 0x80, 0xf9, 0x23, 0x00, 0x31, 0x24, 0x80,
0x31, 0x2f, 0x80, 0x31, 0x62, 0x80, 0x23, 0xea, 0x00, 0x25, 0x53, 0x80,
// `abcdefg
0xf9, 0x03, 0x80, 0x02, 0xe0, 0x00, 0x06, 0xe0, 0x00, 0xf8, 0x42, 0x80,
0x03, 0xe0, 0x00, 0x79, 0x87, 0x80, 0x39, 0x03, 0x80, 0x31, 0x23, 0x00,
// hijklmno
0x7d, 0x23, 0x00, 0x31, 0x27, 0xc0, 0x78, 0x84, 0x00, 0x29, 0x40, 0x00,
0x43, 0xe4, 0x00, 0x70, 0x27, 0x00, 0x60, 0x66, 0x00, 0x70, 0x67, 0x00,
// pqrstuvw
0x48, 0xc4, 0x80, 0x74, 0x57, 0x80, 0x59, 0xe6, 0x80, 0x23, 0xe8, 0x80,
0x03, 0x60, 0x00, 0x8b, 0xe2, 0x00, 0x61, 0x0c, 0x00 // zyx{|}~
};
```

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