HƯỚNG DẪN SỬ DỤNG

BỘ THÍ NGHIỆM VI ĐIỀU KHIỂN EITPS-3192 (PHẦN 2)

MỤC LỤC

Lời nói đầu 1

Giới thiệu 1

Giới thiệu về EITPS-3192 2

Bộ thí nghiệm vi điều khiển MDK-ARM 3

*** Cài đặt chương trình S-ARM V3 Setup.exe 3**

*** Cài đặt phần mềm MDK534.EXE 6**

CHƯƠNG 3 - PHẦN CỨNG VÀ THIẾT BỊ NGOẠI VI 11

Thí nghiệm 3.1 - GPIO (Genaral Purpose Input/Output – Các cổng vào/ra đa năng) và đèn LED 11

3.1.1. Vi điều khiển STM32F100 trong bộ thí nghiệm EITPS-3192 11

3.1.2. Các thuật ngữ và bảng chú giải về GPIO 13

3.1.3. Mô tả chức năng GPIO 14

3.1.4. Thanh ghi GPIO 17

3.1.5. Chương trình với GPIO 22

3.1.6. Đèn LED và kết nối của chúng với cổng đầu ra 24

3.1.7. Mạch đèn Led của bộ thí nghiệm EITPS-3192 26

Thí nghiệm 3.2 - Thao tác bit và Khởi tạo GPIO 30

3.2.1. Thao tác bit với các phép toán logic 30

3.2.2. Thao tác bit với thanh ghi BSRR 32

3.2.3. Đặt lại (Reset) và Kiểm soát Xung (RCC) 33

Thử nghiệm 3.3 - Hiển thị Led 7 thanh 38

3.3.1. Hiển thị Led 7 thanh 38

3.3.2. Hiển thị led 7 thanh trong ghép kênh 39

Thí nghiệm 3.4 - Màn hình LCD 47

3.4.1. Màn hình tinh thể lỏng (LCD) 47

Thử nghiệm 3.5 - GPIO và Công tắc 54

3.5.1. Xác định công tắc đang được nhấn 54

Thí nghiệm 3.6 - Kết nối một ma trận bàn phím 60

3.6.1. Bàn phím 60

Thí nghiệm 3.7 - Còi, Rơ le và Động cơ bước 64

3.7.1. Còi (Buzzer) 64

3.7.2. Rơ le điện 65

3.7.3. Động cơ bước 66

3.7.4. Nút nhấn và công tắc hành trình 68
3.7.5. Cổng đầu ra bên ngoài 68
3.7.6. Khởi tạo GPIOA, GPIOB và GPIOC 70
3.7.7. Điều khiển tải sử dụng các bit GPIO 71
3.7.8. Chương trình điều khiển động cơ bước 72
3.7.9. Đèn LED ma trận 74
Thử nghiệm 3.8 - Vận hành DAC 80
3.8.1. Triển khai một DAC với một bộ khuếch đại & một mạng điện trở 80
3.8.2. DAC nguyên khối 81
3.8.3. DAC điện áp kép đầu vào nối tiếp 8 bit AD7303 83
3.8.4. Đầu ra Analog 84
3.8.5. Giao diện nối tiếp 85
3.8.6. Bộ DAC ARM 85
Thử nghiệm 3.9 - Sử dụng ADC 90
3.9.1. ADC 0808 - kênh ADC nguyên khối được ghép kênh 90
3.9.2. ADC nối tiếp 8 bit ADC0838 91
3.9.3. ADC ARM 91
3.9.4. Chiết áp 92
Thí nghiệm 3.10 – ADC tích hợp, điện trở nhiệt và bộ ghép Quang 95
3.10.1. ADC ARM 95
3.10.2. Điện trở nhiệt 95
3.10.3. Bóng bán dẫn quang (Transistor quang – Phototransistor) 96
3.10.4. Bộ ghép OPTO 97
CHƯƠNG 4 - ARM CORTEX M3 100
4.1. Các thanh ghi 100
4.2. Các thanh ghi đặc biệt 103
4.3. Chế độ bộ xử lý 104
4.4. Chế độ hoạt động 108
4.5. Bộ điều khiển ngắt vectơ lồng nhau được tích hợp sẵn 109
4.5.1. Hỗ trợ ngắt lồng nhau 110
4.5.2. Hỗ trợ ngắt vector 110
4.5.3. Hỗ trợ thay đổi ưu tiên động 110
4.5.4. Giảm độ trễ ngắt 110
4.5.5. Mặt nạ ngắt 110
4.6. Bản đồ bộ nhớ 110

4.7. Giao diện bus **111**
4.8. MPU **112**
4.9. Các ngắt và các ngoại lệ **112**
4.9.1. Công suất thấp và hiệu năng cao 113
4.10. Đặt lại trình tự **113**
4.11. Trình tự ngắt/ngoại lệ **115**
4.11.1. Xếp chồng 115
4.11.2. Tìm nạp vectơ 117
4.11.3. Cập nhật thanh ghi 117
4.12. Các lối ra ngoại lệ **117**
4.13. Ngắt lồng nhau **118**
4.14. Ngắt Tail-chaining **118**
Thí nghiệm 4.1 - Yêu cầu ngắt (IRQ) **118**
4.1.1. Đoạn chương trình ngắt **119**
Thí nghiệm 4.2 - Giao tiếp nối tiếp **123**
4.2.1. Phân loại các phương thức giao tiếp **123**
4.2.2. Giao tiếp không đồng bộ nối tiếp **124**
4.2.3. Mã ASCII **127**
4.2.4. Giao tiếp với PC **128**

Lời nói đầu

Giới thiệu

Tài liệu này hướng dẫn và thực hành ngôn ngữ lập trình C cho các hệ thống vi điều khiển nhúng và vi điều khiển ARM Cortex M3. Vi điều khiển này thuộc họ ARM, là họ vi xử lý và vi điều khiển hàng đầu trên thế giới.

ARM là một bộ xử lý có thể được sử dụng trong bất kỳ loại ứng dụng vi xử lý và vi điều khiển như: sản phẩm tiêu dùng, hệ thống điều khiển, hệ thống thời gian thực và hệ điều hành nhúng.

Mỗi ứng dụng này yêu cầu khả năng xử lý đặc biệt. Tất cả những khả năng này đều có thể được tìm thấy trong bộ xử lý ARM.

Tài liệu này mô tả các ứng dụng bộ điều khiển nhúng với ARM.

Các thí nghiệm trong sách hướng dẫn này được thiết kế để chạy trên Bộ thí nghiệm vi điều khiển EITPS-3192.

Chương 1 - Nguyên lý hoạt động của máy vi tính Mô tả cấu trúc cơ bản của vi điều khiển và nguyên lý hoạt động của nó.

Chương 2 - Ngôn ngữ C bao gồm tập lệnh ngôn ngữ lập trình C và xây dựng kiến thức cho học sinh về cách viết và chạy chương trình cho các hệ thống điều khiển nhúng bằng cách sử dụng chương trình phát triển ARM và các thiết bị công tắc và đèn LED của bộ thí nghiệm.

Chương 3 - Phần cứng và thiết bị ngoại vi mô tả và thực hành cách viết các chương trình ứng dụng vận hành các thiết bị ngoại vi nội bộ ARM như là cổng GPIO, DAC và ADC, cũng như các thiết bị ngoại vi bên ngoài của Bộ thí nghiệm như cổng giao tiếp ngoài, DAC, ADC, bộ hiển thị, động cơ bước, công tắc, cảm biến, v.v.

Chương 4 – Bộ ARM Cortex M3 mô tả chung về bộ vi xử lý và vi điều khiển họ ARM và đặc biệt là vi điều khiển ARM Cortex-M3. Phần mô tả về Cortex-M3 bao gồm cấu trúc bên trong và các tính năng độc đáo của nó. Các bài tập trong chương này là về các ngắt và giao tiếp nối tiếp với USART.

Các ngôn ngữ lập trình cho vi điều khiển là Hợp ngữ (Assembly) và ngôn ngữ C.

Ngôn ngữ C là ngôn ngữ phổ biến nhất để lập trình, đặc biệt là đối với hệ thống ARM - một vi điều khiển phức tạp.

Đây là lý do tại sao các chương trình bài tập lại sử dụng ngôn ngữ C. Một lý do khác là ARM có các thư viện chương trình miễn phí to lớn để sử dụng. Chúng đều được viết bằng ngôn ngữ C.

Giới thiệu về EITPS-3192

EITPS-3192 là một hệ thống đào tạo vi điều khiển ARM độc lập. Nó bao gồm các thành phần và mạch sau:

- Mạch cấp nguồn
- Mạch giao tiếp USB
- Vi điều khiển ARM
- Mạch giải mã
- Bộ nhớ EEPROM nối tiếp
- Cổng đầu vào
- 8 công tắc
- Bàn phím 16 phím
- Led ma trận (16 Led)
- Cổng đầu ra
- Động cơ bước
- Còi
- Rơ le
- 8 đèn Led
- Bộ hiển thị 4 LED 7 thanh
- Màn hình LCD
- Chiết áp biến đổi điện áp
- Cảm biến nhiệt độ
- Cảm biến hồng ngoại
- ADC – Bộ chuyển đổi tương tự sang số
- DAC – Bộ chuyển đổi số sang tương tự
- Nút nhấn RST
- Nút bấm ngắt
- Đầu vào của thiết bị đầu cuối ADC ARM
- Đầu ra của thiết bị đầu cuối DAC ARM

Hình 2. 1

Bộ thí nghiệm vi điều khiển MDK-ARM

Keil MDK là môi trường phát triển phần mềm hoàn chỉnh cho một loạt thiết bị vi điều khiển nền tảng Arm Cortex-M. MDK bao gồm µVision IDE (Môi trường phát triển tích hợp) mà chúng ta sử dụng để biên tập và biên dịch các chương trình bằng ngôn ngữ C cho bộ thí nghiệm EITPS-3192.

S-ARM V3 setup.exe

MDK534.exe

Chương trình này sẽ thực hiện những việc sau:

a) Xây dựng một thư viện với tên: S_ARM.

b) Xây dựng một thư viện với tên: ARM_Project (bao gồm các tệp ứng dụng).

c) Cài đặt trình điều khiển USB (CP210xVCP Installer).

d) Cài đặt chương trình S-ARM để có thể tải xuống và chạy các chương trình đối tượng trong bộ thí nghiệm.

Lưu ý quan trọng:

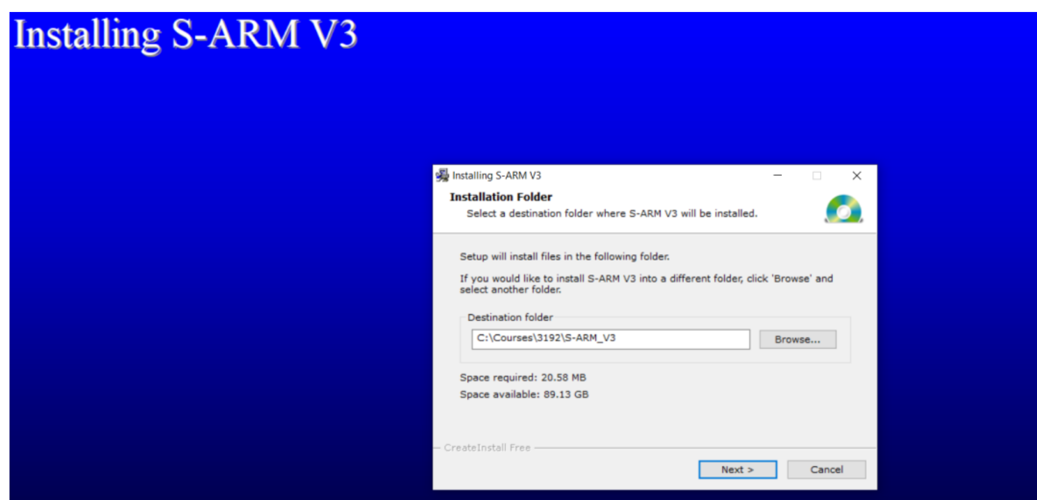
Lưu ý 1:

Cài đặt hai chương trình **S-ARM V3 Setup.exe** và **MDK534.exe** để tải xuống và chạy các chương trình đối tượng trong bộ thí nghiệm.

Các bước cài đặt:

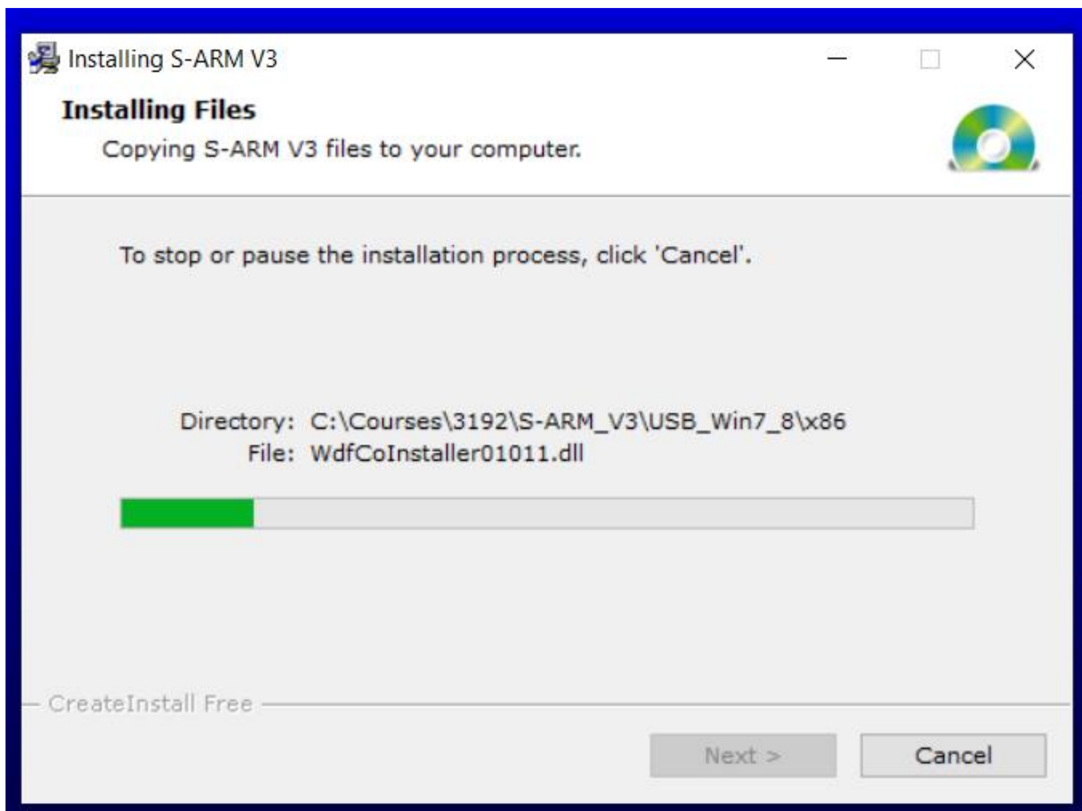
*** Cài đặt chương trình S-ARM V3 Setup.exe**

1. Kích đúp chuột vào biểu tượng



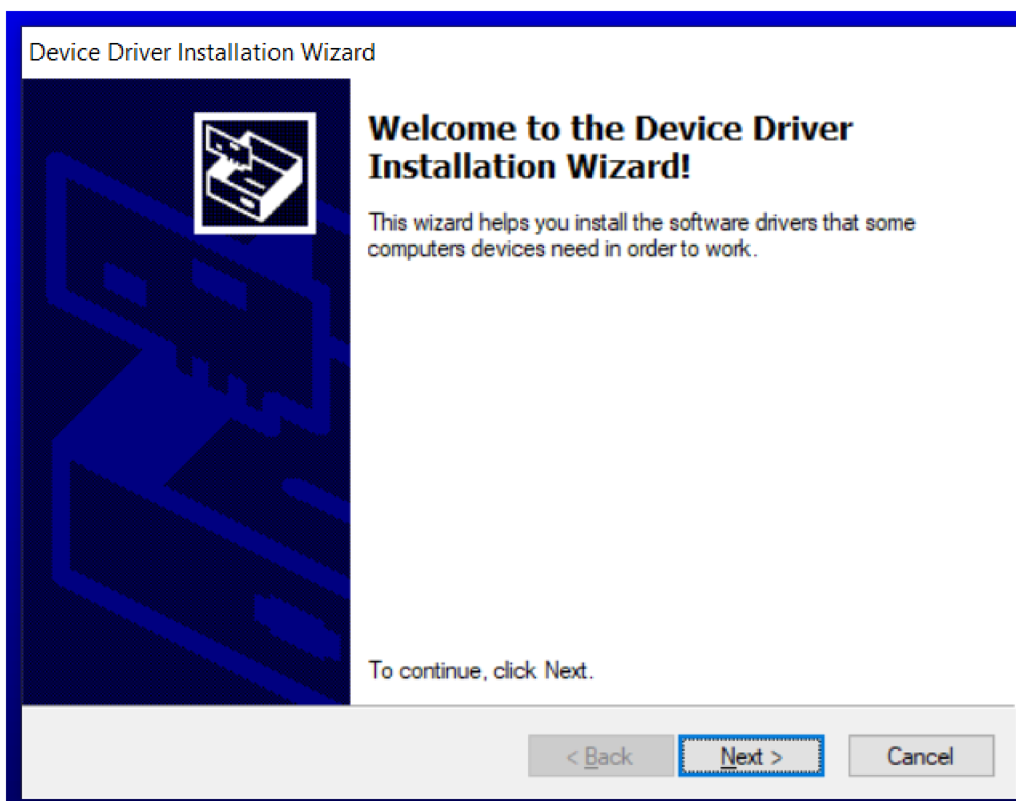
Hình 2. 2

2. Nhấn **Next**



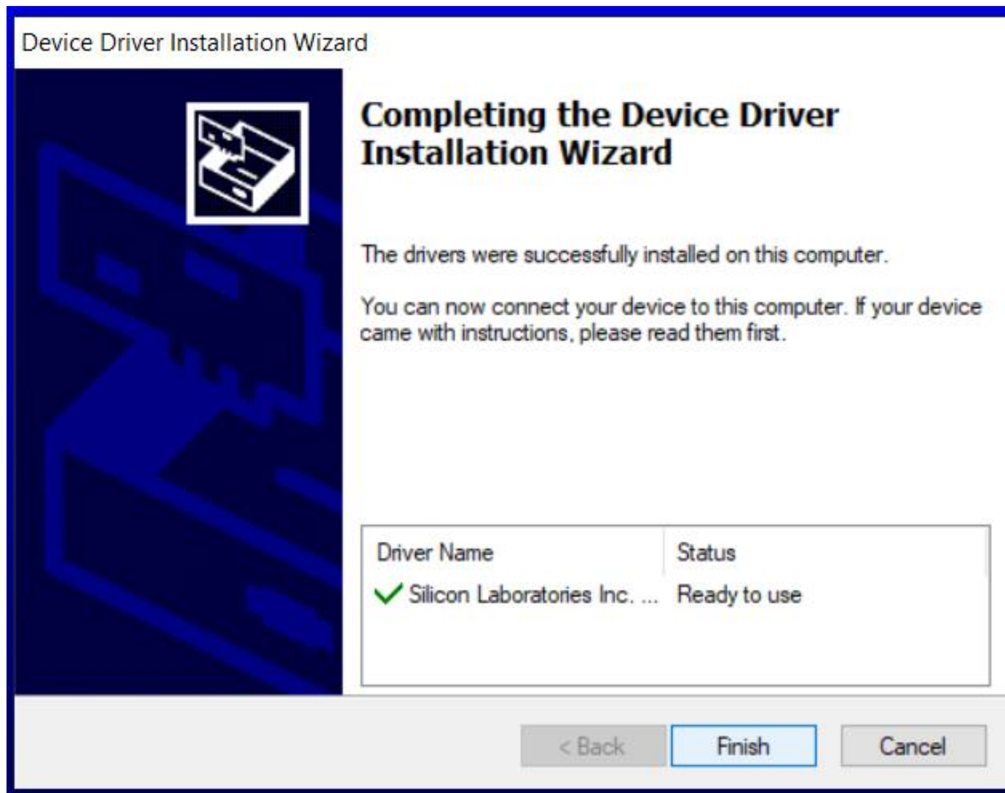
Hình 2. 3

3. Nhấn **Next**



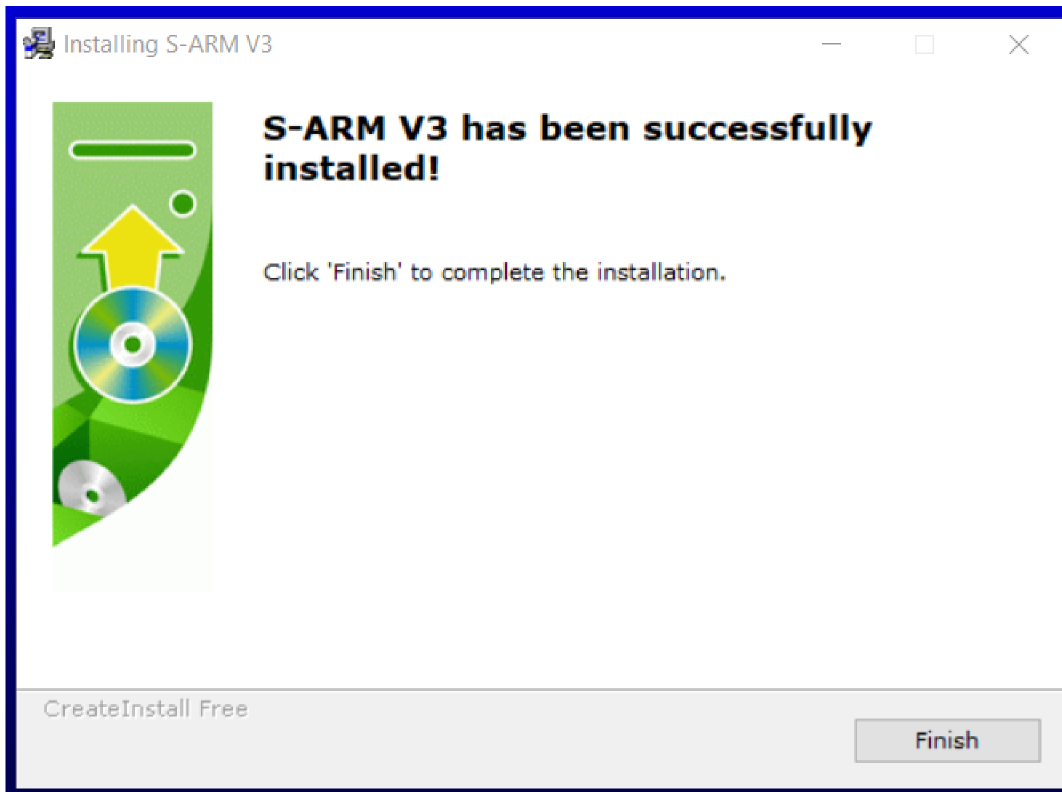
Hình 2. 4

4. Nhấn **Finish**



Hình 2. 5

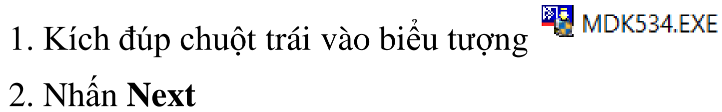
5. Thông báo đã cài đặt thành công S-ARM V3 xuất hiện. Nhấn **Finish**.



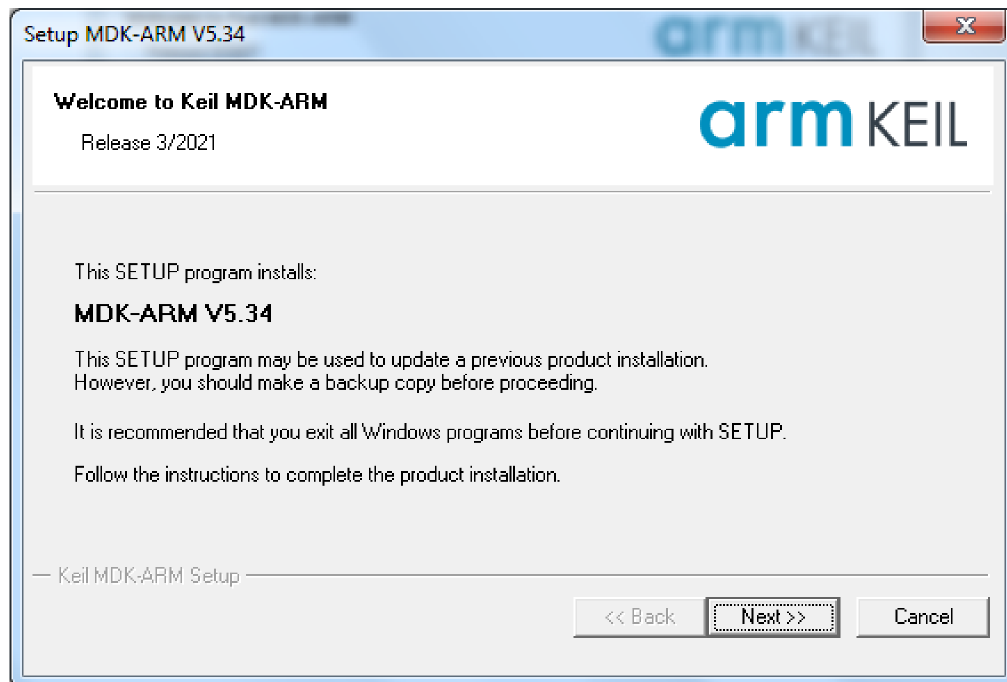
Hình 2. 6

*** Cài đặt phần mềm MDK534.EXE**

1. Kích đúp chuột trái vào biểu tượng



2. Nhấn **Next**

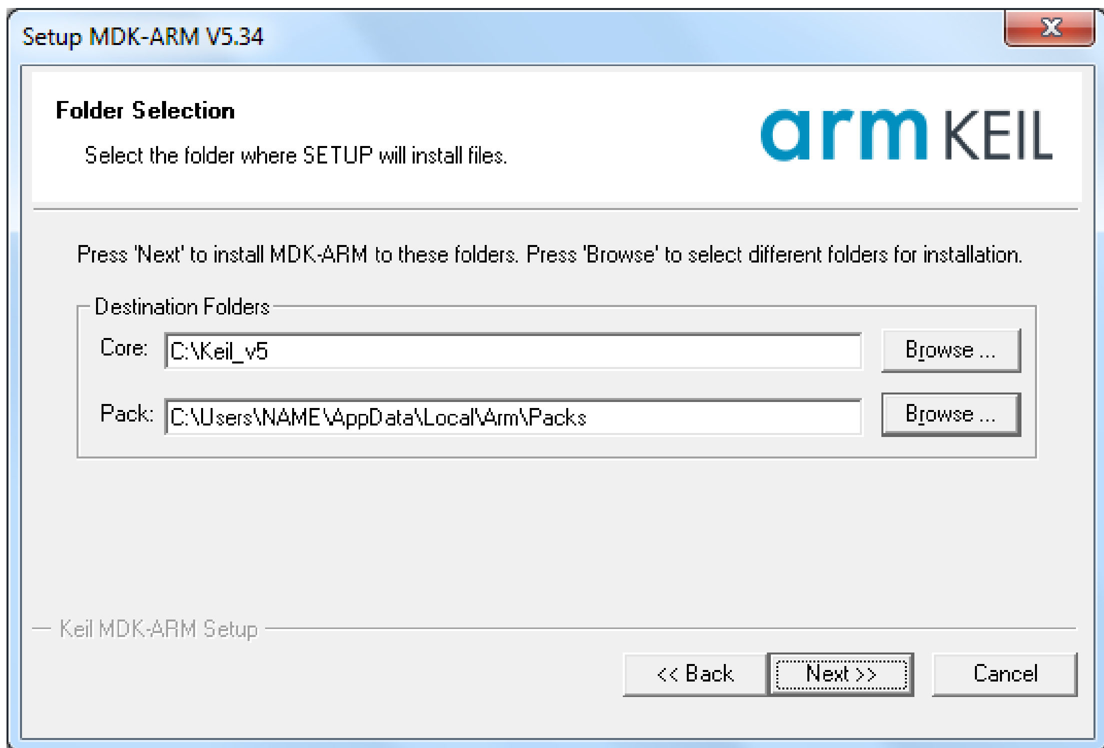


Hình 2. 7

3. Tích chọn “I agree to all the terms of the preceding License Agreement”. Sau đó nhấn **Next**.

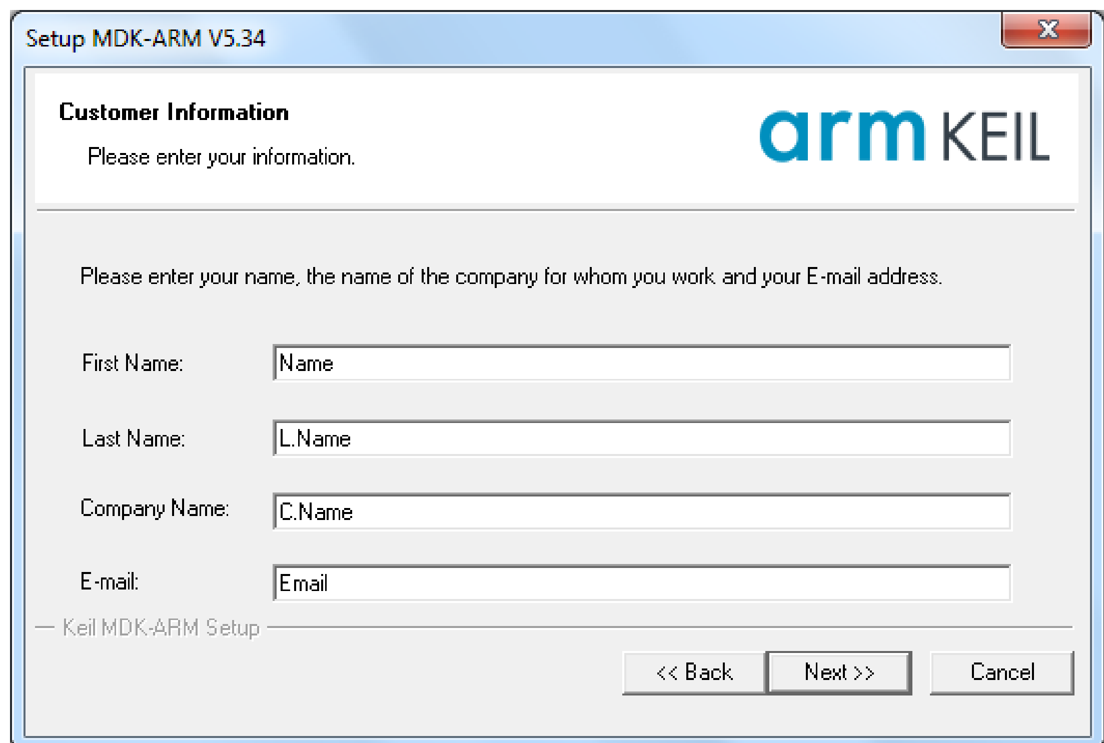
Hình 2. 8

4. Lựa chọn thư mục cài đặt phần mềm. Sau đó nhấn **Next**.



Hình 2. 9

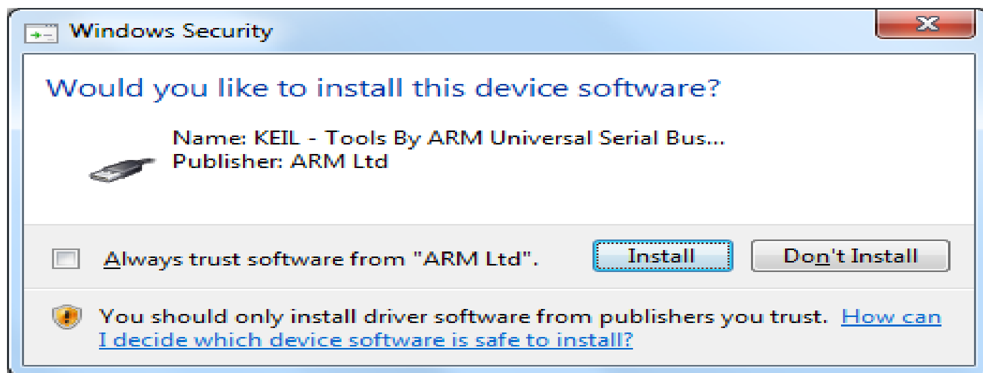
5. Khai báo thông tin cá nhân. Sau đó nhấn **Next**.



Hình 2. 10

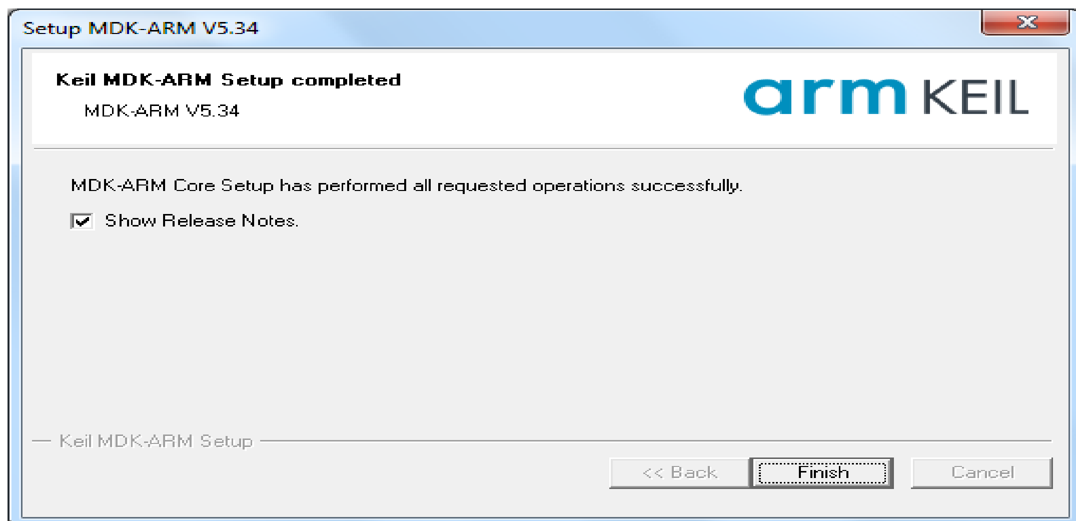
6. Chờ quá trình xử lý cài đặt.

7. Trong khi cài đặt, một màn hình **KEIL-tools** cho bus nối tiếp sẽ xuất hiện. Bấm vào nút **Install**.



Hình 2. 11

8. Khi quá trình cài đặt kết thúc, hãy bấm vào nút **Finish**.

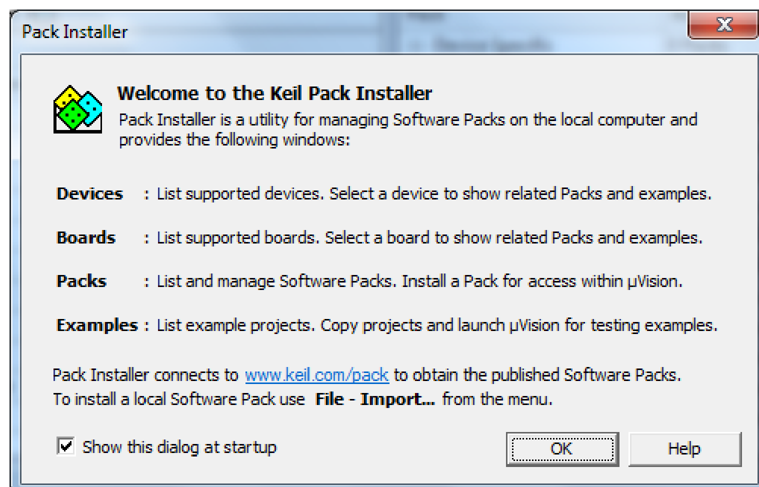


Hình 2. 12

9. Chờ cửa sổ "**Packs Installer**" hiện ra.

Nếu cửa sổ không xuất hiện, bạn có thể mở ứng dụng theo cách thủ công. Đi tới thư mục **µVision** (ứng dụng cuối cùng được cài đặt), sau đó mở và chạy ứng dụng **Packs Installer**.

Cửa sổ tự động sẽ mở ra, nhấp vào nút **OK**.



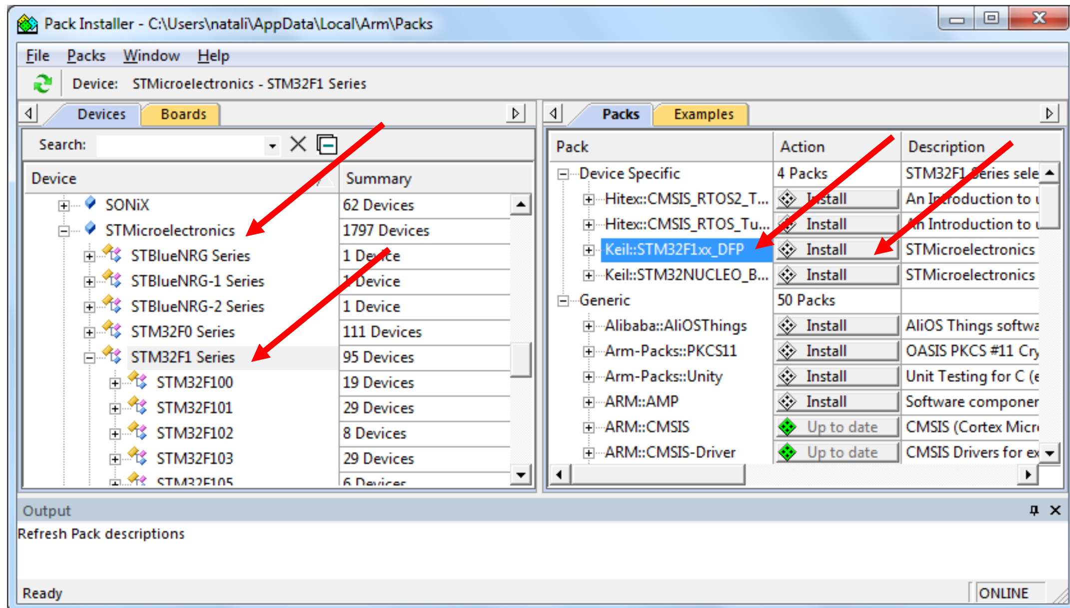
Hình 2. 13

Pack Installer cài đặt các tệp liên quan của bộ điều khiển yêu cầu.

10. Cài đặt thiết bị "**STM32F1xx_DFP**" (chúng ta sử dụng thư viện STM32F1xx của bộ vi điều khiển trong mã lệnh).

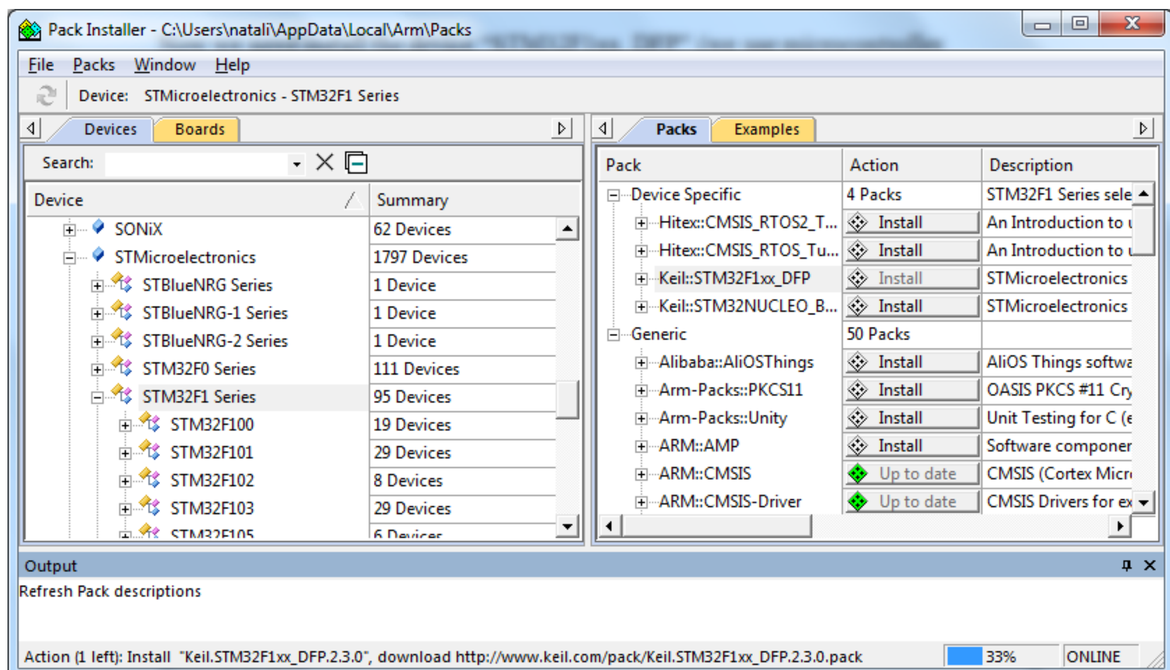
Để cài đặt, hãy làm theo các bước sau:

- Chọn "**STMicroelectronics**" và "**STM32F1Series**" trong cửa sổ **Devices** (màn hình bên trái)
- Chọn tệp "**STM32F1xx_DFP**" trong cửa sổ **Packs** (màn hình bên phải).
- Nhấp vào nút **Install** bên cạnh "**STM32F1xx_DFP**" để cài đặt.



Hình 2. 14

11. **Pack Installer** sẽ được tải xuống và cài đặt chương trình (xem kết quả trong cửa sổ bên dưới). Hãy kiên nhẫn chờ đợi tới khi quá trình xử lý kết thúc.



Hình 2. 15

12. Đóng ứng dụng **Pack Installer**.

Ứng dụng **µVision** đã sẵn sàng cho các chương trình của bộ thí nghiệm EITPS-3192.

Ứng dụng **µVision** sẽ tự động được mở khi bạn nhấp vào chương trình dự án (project) được phát triển bởi **µVision**.

Chú ý 2:

Tạo một Project là một quá trình lâu dài với nhiều công đoạn.

Để tiết kiệm thời gian, chúng ta sẽ sử dụng thư viện và tệp đã chuẩn bị có tên ARM_Project.

1. Vào thư mục chúng ta đã cài đặt S-ARM V3 như trong phần ghi chú 1. Theo tài liệu hướng dẫn này, đường dẫn vào thư mục cài đặt S-ARM V3 là “C:\Courses\3192\S-ARM_V3”

2. Vào thư viện ARM_Project.

3. Xác định tệp ARM_Project .

4. Biểu tượng cho biết tệp này là một **Project µVision**.

CHƯƠNG 3 - PHẦN CỨNG VÀ THIẾT BỊ NGOẠI VI

Thí nghiệm 3.1 - GPIO (Genaral Purpose Input/Output – Các cổng vào/ra đa năng) và đèn LED

Mục tiêu:

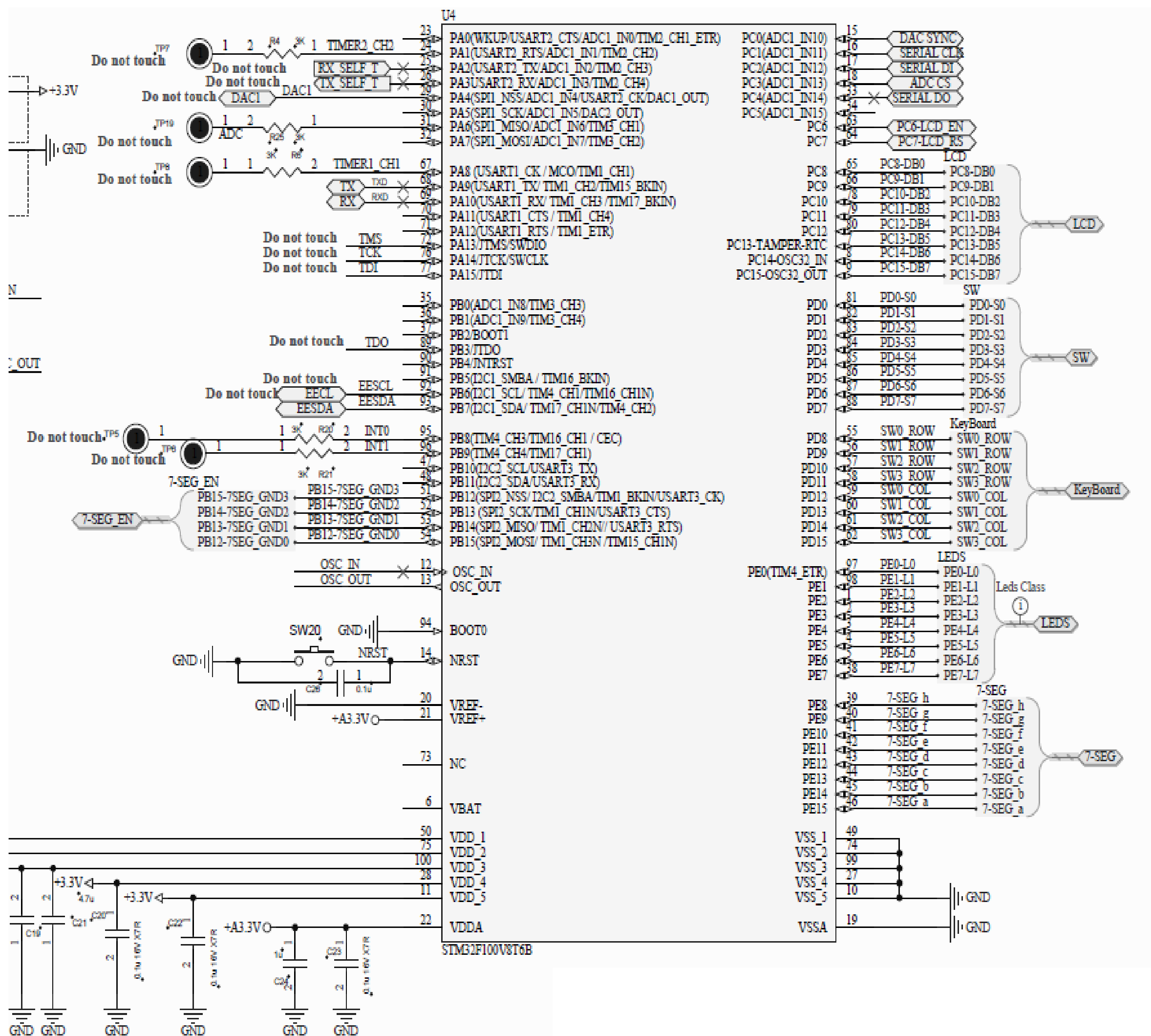
- GPIO như một cổng đầu ra.
- Cách điều khiển đèn LED bằng cổng đầu ra.
- Mạch đèn LED của bộ thí nghiệm EITPS-3192.

Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192

Thảo luận:

3.1.1. Vi điều khiển STM32F100 trong bộ thí nghiệm EITPS-3192



Hình 2. 16

GPIO:

STM32F100 là bộ điều khiển có 100 chân. 80 chân trong số đó thuộc về 5 cổng vào/ra đa năng GPIO với 16 chân mỗi cổng (PA - PE). Các cổng GPIO cho phép triển khai hầu hết các hệ thống điều khiển nhúng mạnh mẽ trong một mạch phần cứng đơn giản.

Mặc dù chúng được tổ chức thành các cổng và thanh ghi với 16 chân, mỗi chân lại có thể có một chức năng khác nhau (một chân là cổng đầu ra, một chân là cổng đầu vào, một chân là cổng ADC, v.v.) và có thể hoạt động riêng biệt.

VDD (thế nguồn dương) và VSS (thế nguồn âm):

Mười trong số các chân điều khiển khác được kết nối với các đường cấp nguồn VDD và VSS.

VDD của STM32F100 là 3,3V. Xu hướng ngày nay là giảm điện áp hoạt động của bộ vi xử lý càng thấp càng tốt. Công suất thấp tạo ra mức tiêu thụ điện năng thấp và cho phép sử dụng các linh kiện nhỏ hơn và ghép được nhiều hơn trong một khối. Kích thước linh kiện không chỉ phụ thuộc vào công nghệ mà còn phụ thuộc vào điện áp hoạt động.

VDD và VSS được kết nối với một số đường của bộ vi điều khiển để chia dòng điện tiêu thụ thành nhiều đường và để tạo điều kiện thuận lợi cho thiết kế bên trong.

Trong các hệ thống kỹ thuật số, dòng điện thay đổi theo xung. Những biến đổi này có thể làm giảm điện áp VDD trong một thời gian rất ngắn nhưng đủ để làm hỏng bộ vi xử lý đang chạy trên một chương trình.

Để ngăn chặn điều đó, một tụ điện được mắc song song với mỗi dòng VDD. Tụ điện cung cấp xung điện và được sạc lại bằng nguồn điện.

VDDA và VSSA:

Bộ vi điều khiển có các mạch số và mạch tương tự. Các mạch tương tự chứa bộ khuếch đại và các xung điện có thể ảnh hưởng đến các mạch tương tự như hiện tượng nhiễu điện tử. Đây là lý do tại sao bộ vi điều khiển có các dòng VDDA và VSSA để tách nguồn điện tương tự khỏi nguồn kỹ thuật số.

VREF:

Đầu vào điện áp khác là các đường điện áp tham chiếu/ điện áp chuẩn (VREF) được sử dụng bởi các Bộ chuyển đổi tương tự sang số (ADC) và Bộ chuyển đổi số sang tương tự (DAC). ADC và DAC sẽ được giải thích ở phần sau.

VBAT (Battery Voltage – Nguồn pin)

Bộ vi điều khiển có thể được kết nối với pin thông qua đường VBAT. Pin cho phép bộ vi điều khiển hoạt động với mức tiêu thụ điện năng thấp cho các tác vụ đặc biệt, ngay cả khi tắt nguồn. Những tác vụ đặc biệt này có thể là đồng hồ thời gian thực (RTC) và kiểm tra ngắt (là tác vụ mà có thể khởi động lại nó).

OSC IN và OSC OUT:

Xung (clock) của bộ vi điều khiển dựa trên một đồng hồ nội bộ kết nối với một tinh thể bên ngoài (external crystal). Tinh thể tạo ra các dao động với tốc độ chính xác.

Bộ vi điều khiển có thể được kết nối với một xung dao động ngoài thông qua đường OSC IN .

Đồng hồ nội bộ có thể hoạt động mà không cần tinh thể bên ngoài và các xung của nó xuất hiện trên đường OSC OUT.

RST:

Bộ vi điều khiển có một đường RST (RESET). Nối đường này xuống GND (GROUND thế điện áp) sẽ đặt lại CPU và nó thực hiện trình tự đặt lại (reset).

Đường này được kéo lên bởi nội điện trở tới VDD. Một tụ điện bên ngoài được kết nối giữa đường RST và GND. Tụ điện giữ CPU ở trạng thái RST trong một thời gian ngắn khi nguồn bật. Điều này buộc CPU luôn khởi động với trình tự đặt lại (reset) trong khi bật nguồn.

3.1.2. Các thuật ngữ và bảng chú giải về GPIO

Mỗi chân GPIO có thể được cấu hình bằng phần mềm thành ngõ ra (push-pull – đầu ra mức logic luôn nằm trong hai lựa chọn 0 hoặc 1; hoặc open-drain – đầu ra mức tín hiệu phụ thuộc vào nguồn bên ngoài), và ngõ vào (có hoặc không có pull-up - điện trở kéo lên hoặc pull-down – điện trở kéo xuống) hoặc thành chức năng thay thế ngoại vi.

Hầu hết các chân GPIO được chia sẻ với các chức năng thay thế số (digital) hoặc tương tự (analog). Tất cả các GPIO đều có dòng điện cao ngoại trừ đầu vào tương tự (analog). Ta có thể quan sát ở hình 2.16 phía trên.

Cấu hình chức năng thay thế I/O có thể bị khóa nếu cần thiết bằng cách tuân theo một trình tự cụ thể để tránh lỗi ghi sai vào các thanh ghi I/O.

Phần này giải thích cách sử dụng các cổng GPIO làm cổng đầu ra số bằng cách sử dụng các phần được lấy từ sách hướng dẫn tham khảo của bộ vi điều khiển ARM STM32F100.

Chúng ta không thể và không cần phải mô tả tất cả các chức năng và các tùy chọn của bộ vi điều khiển đa chức năng này. Ý tưởng là hiểu cách đọc sách hướng dẫn để có thể tìm thấy thông tin cần thiết từ đó.

Tài liệu sử dụng một số từ viết tắt được mô tả trong bảng chú giải sau.

Các thiết bị vi điều khiển STM32F100xx được chia thành **Low-density**, **Medium-density** và **High-density** theo kích thước của bộ nhớ Flash bên trong.

Các thiết bị ngoại vi có sẵn trong bộ vi điều khiển cũng phụ thuộc vào mật độ của nó. Điều này nên được kiểm tra trong tài liệu tham khảo có liên quan.

Bảng chú thích:

- **Word**: dữ liệu có độ dài 32 bit.
- **Half-word**: dữ liệu có độ dài 16 bit.
- **Byte**: dữ liệu có độ dài 8 bit.

read/write (rw) Phần mềm có thể đọc và ghi vào các bit này.

read-only (r) Phần mềm chỉ có thể đọc các bit này.

write-only (w) Phần mềm chỉ có thể ghi vào bit này. Đọc bit sẽ trả về giá trị đặt lại.

read/clear (rc_w1) Phần mềm có thể đọc cũng như xóa bit này bằng cách viết '1'. Viết '0' không ảnh hưởng đến giá trị bit.

read/clear by read (rc_r) Phần mềm có thể đọc bit này. Việc đọc bit này sẽ tự động xóa nó thành ‘0’. Viết '0' không ảnh hưởng đến giá trị bit.

read/set (rs) Phần mềm có thể đọc cũng như thiết lập bit này. Viết '0' không ảnh hưởng đến giá trị bit.

read-only write trigger (rt_w) Phần mềm có thể đọc bit này. Việc viết '0' hoặc '1' sẽ kích hoạt một sự kiện nhưng không ảnh hưởng đến giá trị bit.

toggle (t) Phần mềm chỉ có thể chuyển đổi bit này bằng cách viết '1'. Viết '0' không có hiệu lực.

3.1.3. Mô tả chức năng GPIO

Mỗi cổng GPIO đa năng có (x chỉ ra tên cổng A-E):

1. Thanh ghi cấu hình 32-bit cho phần GPIO Thấp (GPIOx_CRL).
2. Thanh ghi cấu hình 32-bit cho phần GPIO Cao (GPIOx_CRH).
3. Thanh ghi dữ liệu đầu vào 32-bit (GPIOx_IDR).
4. Thanh ghi dữ liệu đầu ra 32-bit (GPIOx_ODR).
5. Thanh ghi thiết lập/đặt lại Bit 32-bit (GPIOx_BSRR).
6. Thanh ghi đặt lại Bit 16 bit (GPIOx_BRR).
7. Thanh ghi khóa 32-bit (GPIOx_LCKR).

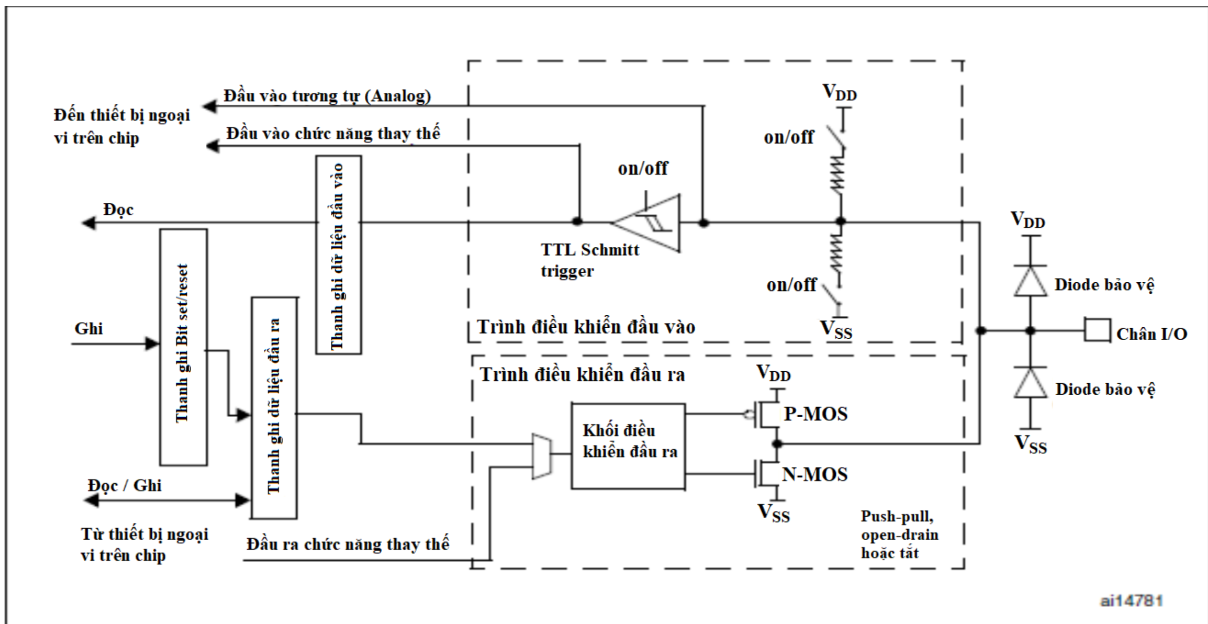
Mỗi một bit cổng I/O có thể lập trình tự do; tuy nhiên, các thanh ghi cổng I/O phải được truy cập dưới dạng các word 32 bit (không cho phép truy cập half-word hoặc byte).

ARM không có lệnh thao tác bit. Các lệnh này cho phép định địa chỉ và thao tác một bit nhất định mà không ảnh hưởng đến bit khác của cổng.

Thao tác bit trên cổng đầu ra có thể đạt được bằng cách đọc dữ liệu cổng đầu ra và thực hiện phép toán AND (để đặt lại) và OR (để thiết lập) trên dữ liệu và ghi dữ liệu trở lại cổng đầu ra.

Một vấn đề có thể xảy ra khi việc đọc và ghi các bit khác trong cổng đầu ra bị thay đổi bởi một số đoạn chương trình ngắt. Chúng ta phải nhớ rằng mỗi bit cổng đầu ra có thể thuộc về một mạch phần cứng khác.

Mục đích của thanh ghi GPIOx_BSRR và GPIOx_BRR là cho phép thao tác nguyên tử (chu kỳ CPU đơn) đọc/sửa đổi quyền truy cập vào bất kỳ bit đầu ra nào của GPIO mà không ảnh hưởng đến các bit đầu ra khác. Bằng cách này, không có rủi ro rằng IRQ xảy ra giữa quyền truy cập đọc và truy cập sửa đổi.



Hình 2. 17

Tùy thuộc vào đặc điểm phần cứng cụ thể của từng cổng I/O được liệt kê trong biểu dữ liệu, mỗi bit của các cổng GPIO có thể được định cấu hình riêng bằng phần mềm ở một số chế độ:

- Đầu vào floating
- Đầu vào pull-up
- Đầu vào pull-down
- Tương tự (analog)
- Đầu ra open-drain
- Đầu ra push-pull
- Chức năng thay thế push-pull
- Chức năng thay thế open-drain

Ví dụ, đường đầu vào có thể được đọc trực tiếp dưới dạng đầu vào tương tự (Analog Input) bởi một ADC.

Nó có thể được sử dụng làm đầu vào số (digital input) (sau bộ đệm TTL Schmitt Trigger) bởi một đầu vào chức năng thay thế (Alternate Function Input) như là giao tiếp nối tiếp, bộ đếm xung, yêu cầu ngắt (interrupt request – IRQ), v.v.

Nó có thể được đọc dưới dạng một bit word 32 bit trong thanh ghi dữ liệu đầu vào (Input Data Register). Các bit phần cao (bit 16-31) của IDR là '0'.

Khi cổng I/O được lập trình làm đầu vào:

- Bộ đệm đầu ra bị vô hiệu hóa.
- Đầu vào Schmitt Trigger được kích hoạt.
- Các điện trở yếu pull-up và pull-down được kích hoạt hay không tùy thuộc vào cấu hình đầu vào (pull-up, pull-down hoặc floating).

- Dữ liệu hiện tại trên chân I/O được lấy mẫu vào thanh ghi dữ liệu đầu vào mỗi chu kỳ xung APB2 được xác định bằng phần mềm sử dụng thanh ghi cho phép đồng hồ ngoại vi APB2 (RCC_APB2ENR).
- Quyền truy cập đọc vào thanh ghi dữ liệu đầu vào đạt được trạng thái I/O.

Đầu vào floating:

Như đã thấy trong hình 2.17 bên trên, đường đầu vào có hai công tắc điện tử có thể được thiết lập bởi phần mềm.

Ở chế độ này, hai công tắc đang mở.

Đầu vào pull-up:

Ở chế độ này, công tắc dưới đang mở.

Đầu vào pull-down:

Ở chế độ này, công tắc trên đang mở.

Analog:

Khi cổng I/O được lập trình dưới dạng cấu hình analog:

- Bộ đệm đầu ra bị vô hiệu hóa.
- Đầu vào Schmitt Trigger được hủy kích hoạt cung cấp mức “0” cho mọi giá trị tương tự (analog) của chân I/O. Đầu ra của Schmitt Trigger buộc phải có giá trị không đổi ('0').
- Các điện trở yếu pull-up và pull-down yếu bị vô hiệu hóa.
- Quyền truy cập đọc vào thanh ghi dữ liệu đầu vào nhận giá trị '0'.

Khi được cấu hình như một đầu ra, giá trị được ghi vào thanh ghi dữ liệu đầu ra (GPIOx_ODR) sẽ được xuất trên chân I/O. Có thể sử dụng trình điều khiển (driver) đầu ra ở chế độ push-pull hoặc chế độ open-drain (chỉ có N-MOS được kích hoạt khi xuất '0').

Đầu ra open-drain:

Trong chế độ này, bóng bán dẫn (transistor) P-MOS không được kích hoạt và bóng bán dẫn N-MOS hoạt động như open-drain.

Đầu ra push-pull:

Trong chế độ này, các bóng bán dẫn P-MOS và N-MOS được kích hoạt.

Đối với các đầu ra chức năng thay thế, cổng phải được định cấu hình ở chế độ đầu ra chức năng thay thế (push-pull hoặc open-drain).

Đối với các chức năng thay thế hai chiều, bit cổng phải được định cấu hình ở chế độ đầu ra chức năng thay thế (push-pull hoặc open-drain). Trong trường hợp này, trình điều khiển đầu vào được cấu hình ở chế độ đầu vào floating.

Khi một bit cổng được cấu hình làm đầu ra chức năng thay thế, nó sẽ ngắt kết nối thanh ghi đầu ra và kết nối chân với tín hiệu đầu ra của thiết bị ngoại vi bên trong chip.

Nếu phần mềm cấu hình một chân GPIO làm đầu ra chức năng thay thế, nhưng thiết bị ngoại vi bên trong không được kích hoạt, đầu ra của nó sẽ không được xác định.

Các thiết bị ngoại vi đầu ra chức năng thay thế có thể là DAC, giao tiếp nối tiếp, tín hiệu PWM, v.v.

<u>Chức năng thay thế push-pull:</u>

Trong chế độ này, các bóng bán dẫn P-MOS và N-MOS được kích hoạt.

<u>Chức năng thay thế open-drain:</u>

Trong chế độ này, bóng bán dẫn P-MOS không được kích hoạt và bóng bán dẫn N-MOS hoạt động như open-drain.

3.1.4. Thanh ghi GPIO

ARM ST32F100 có 5 cổng GPIO (GPIOA, GPIOB, GPIOC, GPIOD và GPIOE) được kết nối với 80 chân I/O.

Mỗi GPIO có 7 thanh ghi riêng (CRL, CRH, IDR, ODR, BSRR, BRR và LCRR).

Mỗi GPIO có địa chỉ cơ sở tuyệt đối, là địa chỉ CRL của chính nó (offset = 0x00).

Mỗi thanh ghi khác đều có địa chỉ offset của 4 byte liên tiếp từ địa chỉ cơ sở.

Mỗi thanh ghi có giá trị reset (đặt lại) riêng (giá trị mặc định của nó sau khi nghỉ).

Địa chỉ cơ sở của các cổng GPIO là:

GPIOA: 0x40010800

GPIOB: 0x40010c00

GPIOC: 0x40011000

GPIOD: 0x40011400

GPIOE: 0x40011800

Thanh ghi cấu hình cổng thấp (GPIOx_CRL) (x = A-E):

Địa chỉ bù: 0x00

Giá trị đặt lại: 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

Nội dung của thanh ghi này định cấu hình cho GPIO 8 bit thấp GPIO (bit 0 đến bit 7). Bốn bit của CRL định cấu hình cho một bit của cổng GPIO. Hai bit xác định chế độ của cổng và hai bit xác định cấu hình của chế độ này.

CNF1	CNF0	MODE1	MODE0	Chức năng	Cấu hình	ODR bit	MOS
0	0	0	0	Đầu vào	Analog	-	
0	1	0	0	Đầu vào	Float	-	
1	0	0	0	Đầu vào	Pull-Down	0	
1	0	0	0	Đầu vào	Pull-Up	1	
0	0	0	1	Đầu ra GP	Push-Pull	0 hoặc 1	10MHz
0	1	0	1	Đầu ra GP	Open-Drain	0 hoặc 1	10MHz
1	0	0	1	Đầu ra AF	Push-Pull	-	10MHz
1	1	0	1	Đầu ra AF	Open-Drain	-	10MHz
0	0	1	0	Đầu ra GP	Push-Pull	0 hoặc 1	2MHz
0	1	1	0	Đầu ra GP	Open-Drain	0 hoặc 1	2MHz
1	0	1	0	Đầu ra AF	Push-Pull	-	2MHz
1	1	1	0	Đầu ra AF	Open-Drain	-	2MHz
0	0	1	1	Đầu ra GP	Push-Pull	0 hoặc 1	50MHz
0	1	1	1	Đầu ra GP	Open-Drain	0 hoặc 1	50MHz
1	0	1	1	Đầu ra AF	Push-Pull	-	50MHz
1	1	1	1	Đầu ra AF	Open-Drain	-	50MHz

MOS – Tốc độ đầu ra lớn nhất

GP – Đa chức năng

AF – Chức năng thay thế

Các bit 0-3 định cấu hình cho bit GPIO0

Các bit 4-7 định cấu hình cho bit GPIO1

Các bit 8-11 định cấu hình cho bit GPIO2

Các bit 12-15 định cấu hình cho bit GPIO3

Các bit 16-19 định cấu hình cho bit GPIO4

Các bit 20-23 định cấu hình cho bit GPIO5

Các bit 24-27 định cấu hình cho bit GPIO6

Các bit 28-31 định cấu hình cho bit GPIO7

Thanh ghi cấu hình cổng cao (GPIOx_CRH) (x=A-E):

Địa chỉ bù: 0x04

Địa chỉ đặt lại: 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

Nội dung của thanh ghi này định cấu hình cho GPIO 8 bit cao GPIO (bit 8 đến bit 15). Bốn bit của CRH định cấu hình cho một bit của cổng GPIO. Hai bit xác định chế độ của cổng và hai bit xác định cấu hình của chế độ này.

CNF1	CNF0	MODE1	MODE0	Chức năng	Cấu hình	ODR bit	MOS
0	0	0	0	Đầu vào	Analog	-	
0	1	0	0	Đầu vào	Float	-	
1	0	0	0	Đầu vào	Pull-Down	0	
1	0	0	0	Đầu vào	Pull-Up	1	
0	0	0	1	Đầu ra GP	Push-Pull	0 hoặc 1	10MHz
0	1	0	1	Đầu ra GP	Open-Drain	0 hoặc 1	10MHz
1	0	0	1	Đầu ra AF	Push-Pull	-	10MHz
1	1	0	1	Đầu ra AF	Open-Drain	-	10MHz
0	0	1	0	Đầu ra GP	Push-Pull	0 hoặc 1	2MHz
0	1	1	0	Đầu ra GP	Open-Drain	0 hoặc 1	2MHz
1	0	1	0	Đầu ra AF	Push-Pull	-	2MHz
1	1	1	0	Đầu ra AF	Open-Drain	-	2MHz
0	0	1	1	Đầu ra GP	Push-Pull	0 hoặc 1	50MHz
0	1	1	1	Đầu ra GP	Open-Drain	0 hoặc 1	50MHz
1	0	1	1	Đầu ra AF	Push-Pull	-	50MHz
1	1	1	1	Đầu ra AF	Open-Drain	-	50MHz

MOS – Tốc độ đầu ra lớn nhất

GP – Đa chức năng

AF – Chức năng thay thế

Các bit 0-3 định cấu hình cho bit GPIO8

Các bit 4-7 định cấu hình cho bit GPIO9

Các bit 8-11 định cấu hình cho bit GPIO10

Các bit 12-15 định cấu hình cho bit GPIO11

Các bit 16-19 định cấu hình cho bit GPIO12

Các bit 20-23 định cấu hình cho bit GPIO13

Các bit 24-27 định cấu hình cho bit GPIO14

Các bit 28-31 định cấu hình cho bit GPIO15

Thanh ghi cấu hình cổng cao (GPIOx_IDR) (x=A-E):

Địa chỉ bù: 0x08

Giá trị đặt lại: 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

Các bit 16-31 được bảo lưu, giữ ở giá trị đặt lại.

Các bit 0-15 IDRy[15:0]: dữ liệu cổng đầu vào (y = 0-15)

Các bit này chỉ được đọc và chỉ có thể được truy cập trong chế độ Word. Chúng chứa giá trị đầu vào của cổng I/O tương ứng.

Thanh ghi dữ liệu đầu ra cổng (GPIOx_ODR) (x=A-E):

Địa chỉ bù: 0x0C

Giá trị đặt lại: 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
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Các bit 16-31 được bảo lưu, giữ ở giá trị đặt lại.

Các bit 0-15 ODRy[15:0]: dữ liệu cổng đầu ra (y = 0-15)

Các bit này có thể được đọc và ghi bằng phần mềm và chỉ có thể được truy cập ở chế độ Word.

<u>Ghi chú:</u>

Đối với thiết lập/đặt lại (set/reset) bit nguyên tử, các bit ODR có thể được thiết lập và xóa một cách riêng lẻ bằng cách ghi vào thanh ghi GPIOx_BSRR (x = A-E).

Thanh ghi set/reset bit cổng (GPIOx_BSRR) (x=A-E):

Địa chỉ bù: 0x10

Giá trị đặt lại: 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**
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

Các bit 0-15 BSy: cổng x thiết lập bit y (y = 0-15)

Các bit này là chỉ ghi và chỉ có thể được truy cập trong chế độ Word.

0: Không có hành động nào trên bit ODRx tương ứng

1: Thiết lập bit ODRx tương ứng

Các bit 16-31 BRy: cổng x đặt lại bit y (y = 0-15)

Các bit này là chỉ ghi và chỉ có thể được truy cập trong chế độ Word.

0: Không có hành động nào trên bit ODRx tương ứng

1: Đặt lại bit ODRx tương ứng

Chú ý:

Nếu cả BSx và BRx đều được thiết lập (set), thì BSx có quyền ưu tiên.

Thanh ghi reset bit cổng (GPIOx_BRR) (x=A-E):

Địa chỉ bù: 0x14

Giá trị đặt lại: 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

Các bit 16-31 được bảo lưu

Các bit 0-15 Bry : cổng x đặt lại bit y (y = 0-15)

Các bit này là chỉ ghi và chỉ có thể được truy cập trong chế độ Word.

0: Không có hành động nào trên bit ODRx tương ứng

1: Đặt lại bit ODRx tương ứng

Thanh ghi khóa cấu hình cổng (GPIOx_LCKR) (x=A-E):

Thanh ghi này được sử dụng để khóa cấu hình của các bit cổng khi một trình tự ghi đúng được áp dụng cho bit 16 (LCKK).

Giá trị của các bit [15: 0] được sử dụng để khóa cấu hình của GPIO. Trong trình tự ghi, giá trị của LCKR [15: 0] không được thay đổi.

Khi chuỗi LOCK đã được áp dụng trên một bit cổng, thì ta không còn có thể sửa đổi giá trị của bit cổng cho đến lần reset tiếp theo.

Mỗi bit khóa đóng băng 4 bit tương ứng của thanh ghi điều khiển (CRL, CRH).

Địa chỉ bù: 0x18

Giá trị đặt lại: 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

Các bit 17-31 được bảo lưu

Bit 16 LCKK[16]: phím khóa

Bit này có thể được đọc bất cứ lúc nào. Nó chỉ có thể được sửa đổi bằng cách sử dụng trình tự ghi phím khóa (Lock Key Writing Sequence).

0: Phím khóa cấu hình cổng không hoạt động

1: Phím khóa cấu hình cổng hoạt động. Thanh ghi GPIOx_LCKR bị khóa cho đến khi reset MCU.

Trình tự ghi phím khóa:

Write 1

Write 0

Write 1

Read 0

Read 1 (Read (đọc) này là tùy chọn nhưng nó xác nhận rằng khóa đang hoạt động)

Ghi chú:

Trong trình tự LOCK Key Writing, giá trị của LCK [15:0] không được thay đổi. Bất kỳ lỗi nào trong trình tự khóa sẽ hủy khóa.

Các bit 0-15 LCKy : cổng x khóa bit y (y = 0-15)

Các bit này là đọc ghi nhưng chỉ có thể được ghi khi bit LCKK bằng 0.

0: Cấu hình cổng không bị khóa.

1: Cấu hình cổng bị khóa.

3.1.5. Chương trình với GPIO

Sau đây là chương trình Led nhấp nháy.

```
int main (void)
{
int i;
Leds_Init();
while(1)
{
LED_PORT->ODR = 0x55; //01010101
```

```
for (i = 1 ; i != 0X100000 ; i++);
LED_PORT->ODR = 0xaa; //10101010
for (i = 1 ; i != 0X100000 ; i++);
}
}
```

Chương trình này ghi vào thanh ghi dữ liệu ngõ ra - ODR (Output Data Register) của LED_PORT các số 0x55 và 0xaa với độ trễ giữa mỗi lần ghi.

LED_PORT trong EITPS-3192 là **GPIOE**.

Lệnh đầu ra có thể là:

GPIOE->ODR = 0x55;

Trong trường hợp này khi chúng ta đọc chương trình, chúng ta nên nhớ rằng các đèn LED được kết nối với GPIOE. Nói cách khác, chúng ta định nghĩa GPIOE như là LED_PORT với khai báo như sau:

#define LED_PORT GPIOE

ODR là thanh ghi Đọc/Ghi. Chúng ta có thể đọc dữ liệu của nó. Trong chương trình sau, ta có thể sử dụng tính năng này.

Hàm **Leds_Init()** sẽ được giải thích trong thí nghiệm tiếp theo.

Chương trình sau đây ghi vào ODR phần bù dữ liệu của nó. Dấu '~' có nghĩa là bù.

```
int main (void)
{
int i;
Leds_Init();
LED_PORT->ODR = 0x00;
while(1)
{
LED_PORT->ODR = ~LED_PORT->ODR;
for (i = 1 ; i != 0X100000 ; i++);
}
}
```

Trong tất cả các chương trình trước, chúng ta sử dụng các khai báo đươc chuẩn bị sẵn bởi Keil (nhà phát triển trình biên dịch và các bộ phát triển phần mềm).

Chương trình sau đây mô tả cách xây dựng các khai báo như vậy.

```
typedef struct
{
volatile unsigned int Crl;
```

```
volatile unsigned int Crh;
volatile unsigned int Idr;
volatile unsigned int Odr;
volatile unsigned int Bsrr;
volatile unsigned int Brr;
volatile unsigned int Lckr;
}GPIO_struct;
#define GPIOE_leds ((GPIO_struct *) (unsigned int) 0x40011800)
int main (void)
{
int i;
Leds_Init();
GPIOE_leds ->Odr = 0x00;
while(1)
{
GPIOE_leds ->Odr = ~ GPIOE_leds ->Odr;
for (i = 1 ; i != 0X100000 ; i++);
}
}
```

Dòng **'#define GPIOE_leds ((GPIO_struct *) (unsigned int) 0x40011800)'** thay thế word GPIOE_leds trong chương trình bằng một con trỏ tới một cấu trúc (được định nghĩa bởi GPIO_struct) tại địa chỉ 0x40011800, là địa chỉ của các thanh ghi cổng GPIOE.

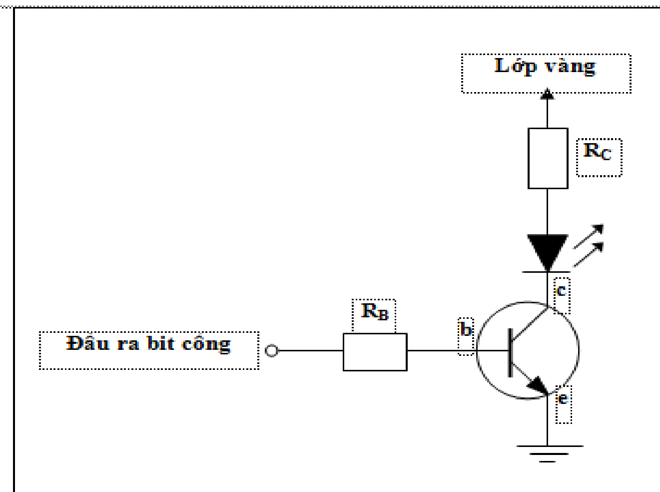
Thông thường, khai báo trên được thay thế bằng hai khai báo như sau:

```
#define GPIOE_PORT (unsigned int) 0x40011800
#define GPIOE_leds ((GPIO_struct *) GPIOE_PORT)
```

3.1.6. Đèn LED và kết nối của chúng với cổng đầu ra

Đèn LED (Light Emitting Diode), như tên gọi của nó, là một đi-ốt (diode) phát ra ánh sáng khi dòng điện chạy qua nó. Cường độ của ánh sáng là một hàm của mật độ dòng điện qua đèn LED. Một dòng điện từ 1mA đến 10mA là đủ để tạo ra ánh sáng có cường độ nhẹ.

Đèn LED có thể được kết nối trực tiếp với cổng đầu ra, nhưng tốt hơn nên kết nối nó với "mạch điều khiển" được vận hành bởi cổng (port). Mạch điều khiển này có thể được dựng bởi một mạch đơn giản (một bóng bán dẫn đơn), như trong hình sau.



Hình 2. 18

Giá trị của R_C và R_B được xác định theo mật độ dòng điện mong muốn (I_C) được thiết kế để chạy qua đèn LED. Giả sử chúng ta muốn có dòng điện với I_C = 10mA. Điện áp giảm trên diode và trên V_{CE} (ở trạng thái bão hòa) là khoảng 1,5V. Do đó, điện áp giảm trên R_C là 3,5V.

$$R_C = \frac{VR_C}{I_C} = 350\Omega$$

Kết nối một điện trở R= 330Ω khi chúng ta muốn có một dòng điện xấp xỉ 10mA và một điện trở R= 680Ω khi muốn có dòng điện 5mA là một thực tế được công nhận.

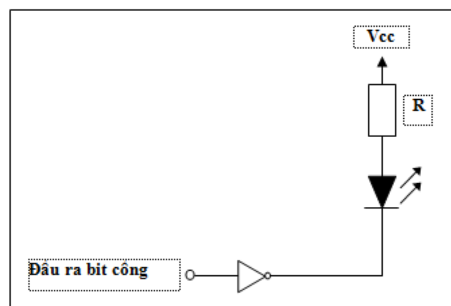
Bóng bán dẫn sẽ được chuyển mạch (kích hoạt) khi byte đầu ra của cổng, được kết nối với bóng bán dẫn, là V_{OH} ($V_{Out,\ High}$), điện áp đầu ra cao, luôn là 3,3V cho các thiết bị trong mạch logic. Dòng I_b cần thiết để kích hoạt bóng bán dẫn được tính toán thông qua β_{min}:

$$I_B = \frac{I_C}{\beta min} = \frac{10m}{50} = 0.2mA$$

$$R_B = \frac{V_{OH} - V_{BE}(Sat)}{I_B} = \frac{3.3 - 0.8}{0.2m} = 12.5K$$

Chúng ta thấy R_B là khoảng 10KΩ. Nếu R_B được lấy nhỏ hơn giá trị này, bóng bán dẫn sẽ rơi vào trạng thái bão hòa sâu. Lưu ý rằng các lần chuyển mạch không quan trọng đối với các thành phần hiển thị. Do đó, R_B thấp hơn có thể được dung nạp.

Đèn LED cũng có thể được kích hoạt thông qua một mạch điều khiển tích hợp, chẳng hạn như 7406 hoặc ULN2001, như trong hình 2.19.



Hình 2. 19

Đèn LED sáng khi bit của cổng đầu ra ở mức cao, giống như đối với trường hợp trước.

3.1.7. Mạch đèn Led của bộ thí nghiệm EITPS-3192

<u>Các đường E của cổng GPIO:</u>

PE0 (chân 97) được kết nối với PE0-L0 (LED0)

PE1 (chân 98) được kết nối với PE0-L1 (LED1)

PE2 (chân 1) được kết nối với PE0-L2 (LED2)

PE3 (chân 2) được kết nối với PE0-L3 (LED3)

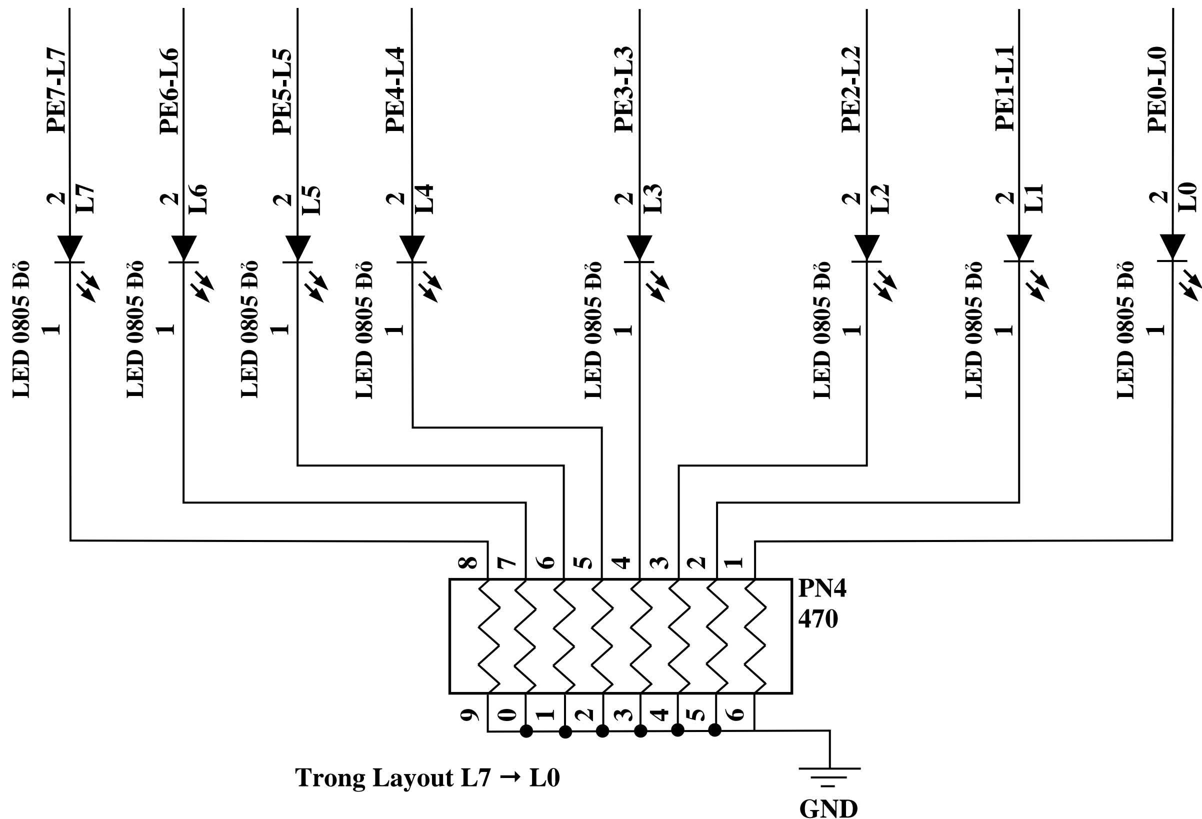
PE4 (chân 3) được kết nối với PE0-L4 (LED4)

PE5 (chân 4) được kết nối với PE0-L5 (LED5)

PE6 (chân 5) được kết nối với PE0-L6 (LED6)

PE7 (chân 38) được kết nối với PE0-L7 (LED7)

<u>Mạch đèn LED:</u>



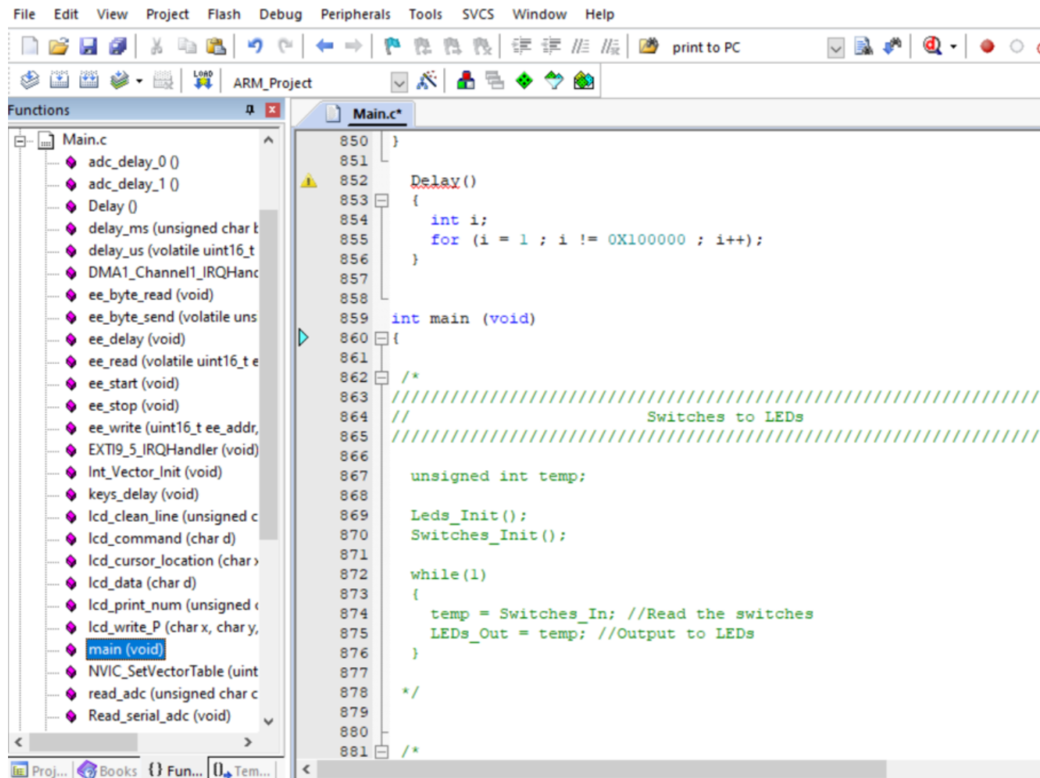
Hình 2. 20

Như được thấy trong hình, các đèn LED có thể được kết nối trực tiếp thông qua các điện trở hạn dòng tới các chân cổng ARM. ARM có khả năng điều khiển đèn LED mà không cần bất kỳ trình điều khiển nào.

Trình tự thực hiện:

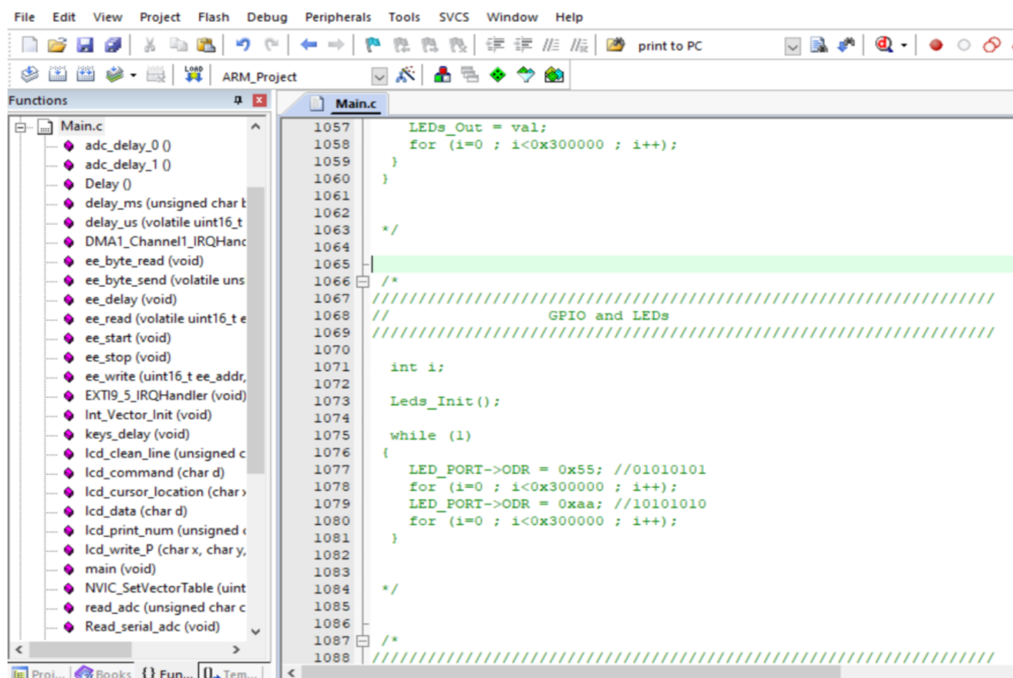
1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project**. Nhập vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này, đường dẫn đến tệp ARM_Project.uvprojx là“C:\Courses\3192\S-ARM_V3\ARM_Project”

2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:



Hình 2. 21

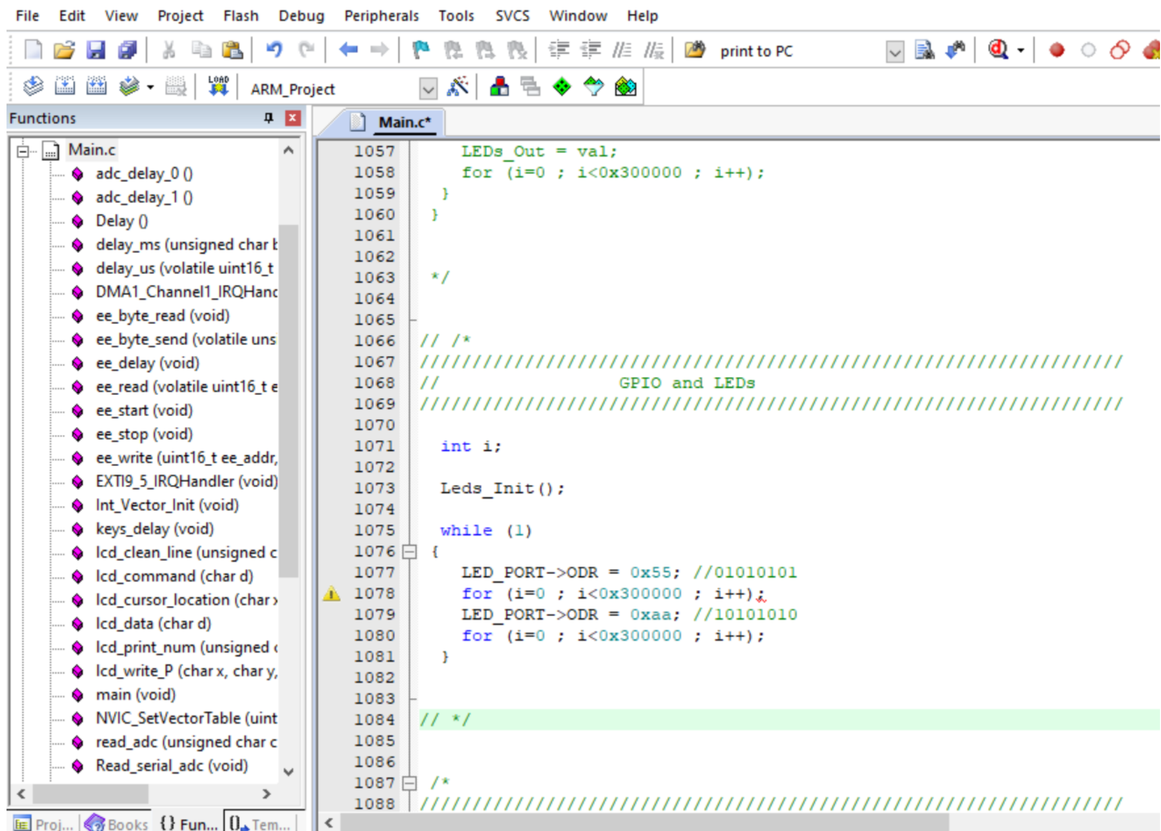
3. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:



Hình 2. 22

Phần này chứa chương trình **GPIO and LEDs**.

4. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:



Hình 2. 23

5. Quan sát chương trình và so sánh nó với chương trình bài tập:

```
int main (void)
{
   int i;
   Leds_Init();
   while(1)
   {
LED_PORT->ODR = 0x55; //01010101
for (i = 1 ; i != 0X100000 ; i++);
LED_PORT->ODR = 0xaa; //10101010
for (i = 1 ; i != 0X100000 ; i++);
   }
}
```

6. Kích hoạt EITPS-3192.

7. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

8. Quan sát các đèn LED nhấp nháy.
9. Nhấn RST để dừng chương trình đang chạy.
10. Thay đổi chương trình thành chương trình sau:

```
int main (void)
{
   int i;
   Leds_Init();
   LED_PORT->ODR = 0x00;
   while(1)
   {
LED_PORT->ODR = ~LED_PORT->ODR;
for (i = 1 ; i != 0X100000 ; i++);
   }
}
```

11. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

12. Kiểm tra xem tốc độ nháy có thay đổi không.
13. Nhấn RST để dừng chương trình.
14. Thay đổi chương trình thành chương trình sau:

```
typedef struct
{
   volatile unsigned int Crl;
   volatile unsigned int Crh;
   volatile unsigned int Idr;
   volatile unsigned int Odr;
   volatile unsigned int Bsrr;
   volatile unsigned int Brr;
volatile unsigned int Lckr;
}GPIO_struct;
#define GPIOE_leds ((GPIO_struct *) (unsigned int) 0x40011800)
int main (void)
{
int i;
   Leds_Init();
```

```
    GPIOE_leds ->Odr = 0x00;
    while(1)
    {
GPIOE_leds ->Odr = ~ GPIOE_leds ->Odr;
for (i = 1 ; i != 0X100000 ; i++);
    }
    }
```

15. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

16. Kiểm tra xem tốc độ nháy có thay đổi không.

17. Nhấn RST để dừng chương trình.

18. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

Thí nghiệm 3.2 - Thao tác bit và Khởi tạo GPIO

Mục tiêu:

- Thao tác bit với cổng ARM.
- Nghiên cứu thanh ghi BSRR.
- Khởi tạo GPIO như một cổng ngõ ra.

Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192

Thảo luận:

3.2.1. Thao tác bit với các phép toán logic

Thao tác bit có thể đạt được bằng phép toán logic được mô tả trong thí nghiệm 3.5.

<u>Phép toán AND với số nhị phân:</u>

Phép toán AND được sử dụng để đặt lại (reset) các bit trong một số, bất kể chúng có gì trước đó.

Ví dụ:

A = 11011011

B = 01101010

Sau phép toán logic AND:

Y = A & B;

Chúng ta nhận được:

Y = 01001010

Bit 2 và 5 là 0, bất kể chúng là gì trước đó. Phần còn lại của các bit không thay đổi.

Phép toán OR với số nhị phân:

Phép toán OR được sử dụng để thiết lập (set) các bit trong một số, bất kể chúng có gì trước đó.

Ví dụ:

A = 00100100

B = 01101010

Sau phép toán logic OR:

Y = A | B;

Chúng ta nhận được:

Y = 01101110

Bit 2 và 5 là 1, bất kể chúng là gì trước đó. Phần còn lại của các bit không thay đổi.

Phép toán NOT trên số nhị phân:

Phép toán NOT được sử dụng để bù các bit trong một số (từ 0 thành 1 và từ 1 thành 0);

Ví dụ:

A = 01011001

Sau phép toán logic NOT:

Y = ~A;

Chúng ta nhận được:

Y = 10100110

Chương trình sau làm LED2 nhấp nháy mà không ảnh hưởng đến các đèn LED khác của cổng ngõ ra.

```
int main (void)
{
int i;
Leds_Init();
LED_PORT->ODR = 0xff;
while(1)
{
LED_PORT->ODR = LED_PORT->ODR & 0xfffb;
for (i = 1 ; i != 0X100000 ; i++);
LED_PORT->ODR = LED_PORT->ODR | 0x0004;
for (i = 1 ; i != 0X100000 ; i++);
}
}
```

Số 0xfffb là 1111,1111,1111,1011. Phép toán AND với số này sẽ đặt lại (reset) bit 2 và không ảnh hưởng đến các bit khác.

Lệnh đầu tiên trong vòng lặp while, đọc cổng ODR, đặt lại (reset) bit 2 và ghi số vào ODR.

Số 0x0004 là 000,0000,0000,0100. Phép toán OR với số này thiết lập (set) bit 2 và không ảnh hưởng đến các bit khác.

Lệnh thứ ba trong vòng lặp while, đọc cổng ODR, thiết lập (set) bit 2 và ghi số vào ODR.

3.2.2. Thao tác bit với thanh ghi BSRR

Đoạn chương trình trên là tốt nếu các bit cổng ngõ ra chỉ được thao tác bởi chương trình chính.

Thông thường, thiết bị ngoại vi được xử lý bởi các đoạn chương trình ngắt và trình xử lý ngắt.

Vấn đề với đoạn chương trình trên là lệnh thao tác không phải là lệnh nguyên tử (chu kỳ CPU đơn). Nó đọc ODR, thực hiện thao tác và ghi vào ODR.

Nếu giữa quá trình đọc và quá trình ghi mà đoạn chương trình ngắt sẽ thay đổi một số bit khác, thì việc ghi sẽ làm hỏng thao tác đoạn chương trình ngắt.

Thanh ghi BSRR sẽ giải quyết vấn đề đó. Nó cho phép thiết lập hoặc đặt lại các bit nhất định mà không cần đọc ODR và không ảnh hưởng đến các bit khác trong một chu kỳ CPU đơn.

BSRR là thanh ghi 32 bit.

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

Các bit 0-15 Bsy: Cổng x thiết lập bit y (y = 0-15)

Các bit này là chỉ ghi và chỉ có thể được truy cập trong chế độ Word.

0: Không có hành động nào trên bit ODRx tương ứng

1: Thiết lập (set) bit ODRx tương ứng

Các bit 16-31 Bry: Cổng x đặt lại bit y (y = 0-15)

Các bit này là chỉ ghi và chỉ có thể được truy cập trong chế độ Word.

0: Không có hành động nào trên bit ODRx tương ứng

1: Đặt lại (reset) bit ODRx tương ứng

<u>Chú ý:</u>

Nếu cả BSx và BRx đều được thiết lập (set), thì BSx có quyền ưu tiên.

Chương trình sau làm LED2 nhấp nháy mà không ảnh hưởng đến các đèn LED khác của cổng ngõ ra.

```
int main (void)
{
int i;
Leds_Init();
LED_PORT->ODR = 0x00;
while(1)
{
LED_PORT->BSRR = (unsigned int) 0x00000004;
for (i = 1 ; i != 0X100000 ; i++);
LED_PORT->BSRR = (unsigned int) 0x00040000;
for (i = 1 ; i != 0X100000 ; i++);
}
}
```

Khi chúng ta gán một số cho một thanh ghi, chúng ta phải cho trình biên dịch biết kích thước của số đó. Trình biên dịch bỏ qua các số “0” ở đầu của số. Nó coi số 0x00000004 là 0x4. Để thuận tiện, ta viết số đó là 0x00000004.

3.2.3. Đặt lại (Reset) và Kiểm soát Xung (RCC)

ARM Cortex M3 có một hệ thống Đặt lại và Kiểm soát Xung (RCC - Reset and Clock Control) rất phong phú. Hệ thống này được định cấu hình và xác định bởi 11 thanh ghi:

Thanh ghi kiểm soát xung - RCC_CR (Clock Control Register)

Thanh ghi cấu hình xung - RCC_CFGR (Clock Configuration Register)

Thanh ghi ngắt xung - RCC_CIR (Clock Interrupt Register)

Thanh ghi đặt lại ngoại vi APB2 - RCC_APB2RSTR (APB2 Peripheral Reset Register)

Thanh ghi đặt lại ngoại vi APB1 - RCC_APB1RSTR (APB1 Peripheral Reset Register)

Thanh ghi kích hoạt xung ngoại vi AHB - RCC_AHBENR (AHB Peripheral Clock Enable Register)

Thanh ghi kích hoạt xung ngoại vi APB2 - RCC_APB2ENR (APB2 Peripheral Clock Enable Register)

Thanh ghi kích hoạt xung ngoại vi APB1 - RCC_APB1ENR (APB1 Peripheral Clock Enable Register)

Thanh ghi kiểm soát miền dự phòng - RCC_BDCR (Backup Domain Control Register)

Thanh ghi Điều khiển/Trạng thái - RCC_CSR (Control/Status Register)

Thanh ghi cấu hình xung2 - RCC_CFGR2 (Clock Configuration Register 2)

Đặt lại hệ thống:

Việc đặt lại hệ thống sẽ thiết lập tất cả các thanh ghi thành giá trị đặt lại (reset) của chúng ngoại trừ các cờ reset (reset flag) trong thanh ghi CSR của bộ điều khiển xung và các thanh ghi trong Backup Domain. Việc đặt lại hệ thống được khởi tạo khi một trong các sự kiện sau xảy ra:

- Đặt một mức thấp trên chân NRST (reset bên ngoài)
- Window watchdog kết thúc điều kiện đếm (reset WWDG reset)
- Independent watchdog kết thúc điều kiện đếm (IWDG reset)
- Đặt lại một phần mềm (SW reset)
- Đặt lại quản lý năng lượng thấp (xem phần Đặt lại quản lý năng lượng thấp)
- Đặt lại nguồn được khởi tạo khi đặt lại power-on/power-down (POR/PDR reset) hoặc khi thoát khỏi chế độ Chờ (Standby mode).
- Đặt lại Backup domain được khởi tạo tại Software reset, được kích hoạt bằng cách thiết lập bit BDRST trong thanh ghi điều khiển Backup domain (RCC_BDCR) hoặc khi VDD hay VBAT bật nguồn.

Tất cả các tùy chọn này được giải thích trong tài liệu tham khảo STM32F100.

Xung:

Ba nguồn xung khác nhau có thể được sử dụng để điều khiển xung hệ thống (SYSCLK):

- Xung dao động HSI (High Speed Internal) dựa trên bộ dao động RC bên trong nhưng có thể được hiệu chỉnh để có tần số chính xác.
- Xung dao động HSE (High Speed External)
- Xung PLL (Phase Locked Loop)

Các thiết bị có hai nguồn xung nhịp phụ sau:

- Xung nhịp tốc độ thấp 40KHz của RC nội bộ (LSI RC) điều khiển independent watchdog và tùy chọn RTC được sử dụng cho Auto-wakeup từ chế độ Stop/Standby.
- Xung nhịp tốc độ thấp 32,768KHz của tinh thể ngoài (tinh thể LSE) điều khiển một cách tùy chọn đồng hồ thời gian thực (RTCCLK).

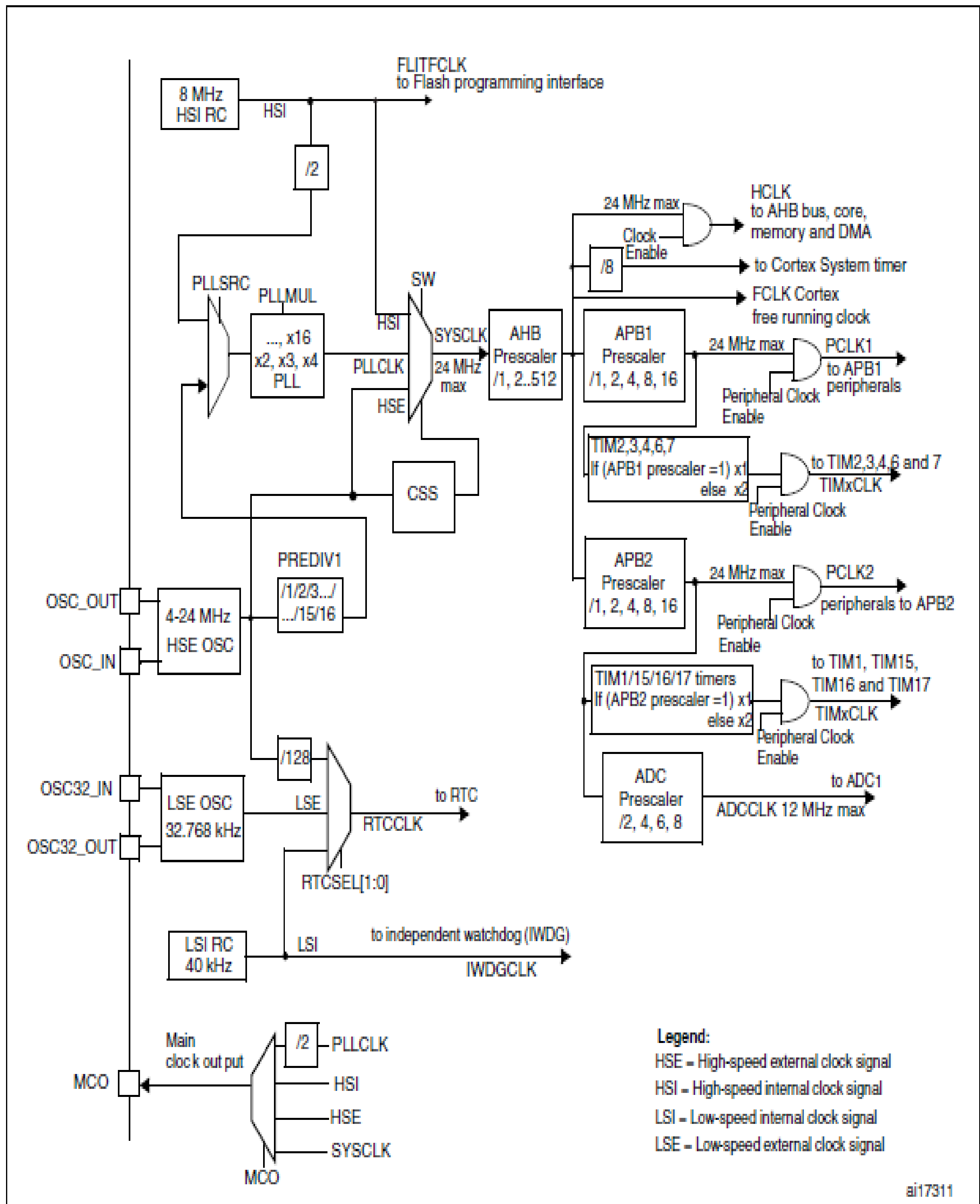
Mỗi nguồn xung nhịp có thể được bật hoặc tắt độc lập khi nó không được sử dụng, để tối ưu hóa mức tiêu thụ điện năng.

Các xung phục vụ các thiết bị ngoại vi ARM như: GPIO, ADC, DAC, Timers, DMA (Direct Memory Access), USARTS (Universal Synchronous Asynchronous Receiver Transmitter), Watchdogs.

Tất cả các bộ phận này đều là hệ thống đồng bộ và được đồng bộ với xung. Dòng điện là cao trong quá trình chuyển đổi xung nhịp giúp cho các xung nhịp tiêu thụ dòng điện năng.

ARM cho phép khoá hoặc mở xung nhịp cho bất kỳ thiết bị ngoại vi nào để tiết kiệm dòng điện ở những nơi không cần xung nhịp.

Mạch sau đây mô tả mạch xung của STM32F100.



Hình 2. 24

Các thanh ghi RCC được khai báo là cấu trúc RCC với các khai báo sau.

```
typedef struct
{
Volatile unsigned int CR;
Volatile unsigned int CFGR;
Volatile unsigned int CIR;
Volatile unsigned int APB2RSTR;
Volatile unsigned int APB1RSTR;
Volatile unsigned int AHBENR;
Volatile unsigned int APB2ENR;
Volatile unsigned int APB1ENR;
Volatile unsigned int BDCR;
Volatile unsigned int CSR;
Volatile unsigned int AHBRSTR;
Volatile unsigned int CFGR2;
}RCC_Typedef
#define RCC((RCC_Typedef *) ((unsigned int) 0x400210000)
```

Đoạn chương trình **Leds_Init** mà chúng ta sử dụng trong mọi chương trình được viết như sau:

```
Void Leds_Init (void)
{
#define RCC_APB2ENR_IOPEN ((unsigned int) 0x00000040)
#define MODE_OUT_2MHZ_PP_0_7 0x22222222
RCC->APB2ENR |= RCC_APB2ENR_IOPEN;//Enable GPIOE clock
GPIOE->CRL = ((unsigned int) 0x00000000;//clear the CRL register
GPIOE->CRL |= MODE_OUT_2MHZ_PP_0_7;//Set PE0-PE7 as outputs
}
```

Đoạn chương trình này định cấu hình thanh ghi GPIOE CRL với số 0x22222222 để khai báo PE0-PE7 là cổng ngõ ra ở 2MHZ.

Nó cũng kích hoạt xung của GPIOE.

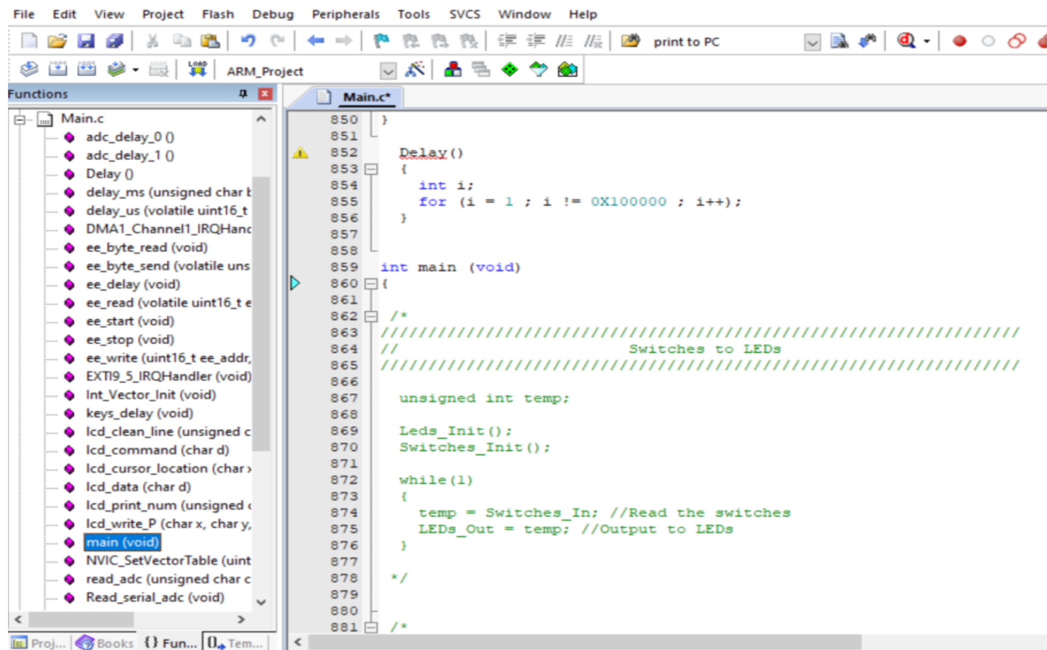
Như đã giải thích trước đó, tốt hơn là tất cả đoạn chương trình khởi tạo và khai báo phần cứng sẽ là một phần của **main.h** và không được đưa vào chương trình.

Trình tự thực hiện:

1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project**. Nhập vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này,

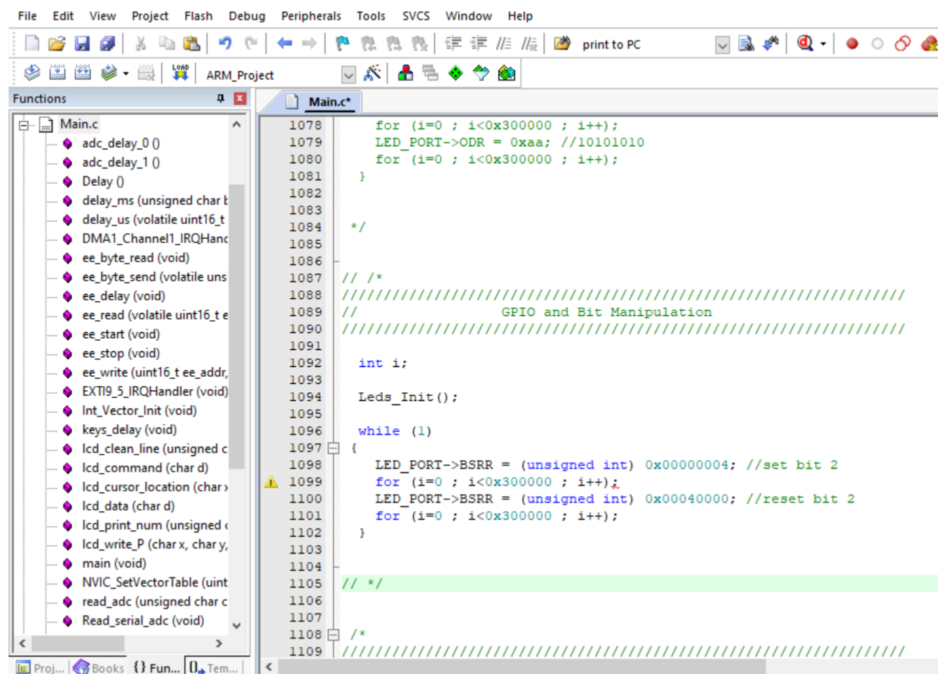
đường dẫn đến tệp ARM_Project.uvprojx là“C:\Courses\3192\S-ARM_V3\ARM_Project”

2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:



Hình 2. 25

3. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:



Hình 2. 26

4. Quan sát chương trình và so sánh với chương trình bài tập:

int main (void)

{

int i;

Leds_Init();

```
    LED_PORT->ODR = 0x00;
    while(1)
    {
LED_PORT->BSRR = (unsigned int) 0x00000004; //set bit 2
for (i = 1 ; i != 0X100000 ; i++);
LED_PORT->BSRR = (unsigned int) 0x00040000; //reset bit 2
for (i = 1 ; i != 0X100000 ; i++);
  }
  }
```

5. Kích hoạt EITPS-3192.

6. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

6. Quan sát đèn LED nhấp nháy.

Chương trình sau đây làm cho LED2 nhấp nháy mà không ảnh hưởng đến các đèn LED ở cổng ngõ ra khác.

7. Nhấn RST để dừng chương trình đang chạy.

8. Thay đổi chương trình để LED4 sẽ nhấp nháy thay vì LED2 và thời gian BẬT sẽ gấp đôi thời gian TẮT.

9. Lặp lại các bước 3-7 và kiểm tra xem tốc độ nhấp nháy có thay đổi không.

10. Nhấn RST để dừng chương trình.

11. Thay đổi chương trình để LED2 và LED4 sẽ nhấp nháy. Khi LED2 BẬT, LED4 TẮT và ngược lại.

12. Lặp lại các bước 3-7 và kiểm tra xem tốc độ nháy có thay đổi không.

13. Nhấn RST để dừng chương trình.

14. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

Thử nghiệm 3.3 - Hiển thị Led 7 thanh

Mục tiêu:

- Cách vận hành bộ hiển thị Led 7 thanh 4 số thông qua cổng ngõ ra và bộ giải mã.
- Cách viết một chương trình điều khiển Led 7 thanh ở chế độ ghép kênh.

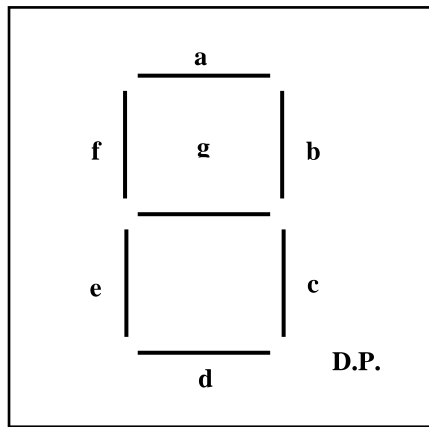
Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192

Thảo luận:

3.3.1. Hiển thị Led 7 thanh

Như đã biết, hiển thị Led 7 thanh được tạo thành từ bảy thanh LED của chữ số 8 và LED thứ tám dùng cho dấu chấm.



Hình 2. 27

Tám đèn LED chung một cực âm. Cực âm này được nối đất. Trên thị trường cũng có các Led 7 thanh với cực dương chung. Tám cực dương có thể được kết nối qua mạch điều khiển với bit của cổng ngõ ra. Đặt một bit ngõ ra lên mức cao sẽ làm sáng một đèn LED.

Khi cần vận hành màn hình hiển thị Led 7 thanh, ta có thể kết nối mỗi Led với một cổng ngõ ra khác nhau. Phương pháp này có hai nhược điểm:

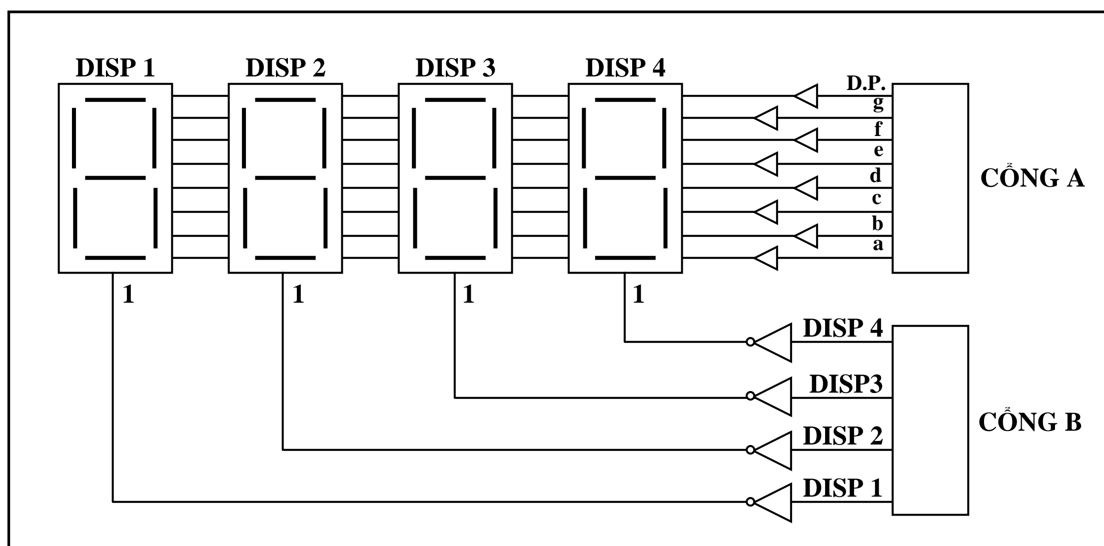
1. Sử dụng một số lượng lớn các cổng ngõ ra.

2. Mức tiêu thụ dòng điện cao. Theo phương pháp này, mọi linh kiện hoạt động mọi lúc. Dòng điện yêu cầu là tổng của tất cả các dòng điện phục vụ cho tất cả các linh kiện.

3.3.2. Hiển thị led 7 thanh trong ghép kênh

Phương pháp được chấp nhận để vận hành nhiều đơn vị hiển thị song song là sử dụng hoạt động động. Theo phương pháp này, tám đèn LED của bộ hiển thị được kết nối song song, thông qua một mạch điều khiển, với các ngõ ra của một cổng đầu ra.

Hãy ký hiệu cổng này là cổng A. Một số nhị phân xuất hiện ở các ngõ ra của cổng này, thực tế sẽ xuất hiện ở tất cả các đầu vào của 7 thanh. Các đầu vào điện áp của các Led hiển thị được kết nối thông qua mạch điều khiển với một cổng bổ sung - chẳng hạn như cổng B (xem hình 3-8).



Hình 2. 28

Các bước vận hành màn hình hiển thị theo phương pháp này như sau:

1. Xuất ra số dành cho thiết bị hiển thị đầu tiên thông qua cổng A.

2. Đóng mạch của thiết bị này bằng cách sử dụng cổng B.

3. Một độ trễ ngắn (có thể bỏ qua bước này).

4. Ngắt mạch khỏi bộ hiển thị (00 đến cổng B).

5. Xuất ra số dành cho thiết bị hiển thị thứ hai thông qua cổng A.

6. Đóng mạch của thiết bị này bằng cách sử dụng cổng B.

7. Một độ trễ ngắn.

8. Ngắt kết nối mạch khỏi bộ hiển thị, và tiếp tục như vậy. Sau khi kích hoạt thiết bị hiển thị cuối cùng, chúng ta quay lại đơn vị hiển thị đầu tiên và cứ thế lặp đi lặp lại.

Khi phương pháp này được thực hiện, CPU luôn hoạt động trong việc chạy màn hình hiển thị. Chương trình này cũng có thể được xây dựng theo cách sử dụng một chương trình con thứ cấp và gọi chương trình con này bất kỳ lúc nào cần hiển thị một số trên thiết bị hiển thị.

Trong EITPS-3192, các đường cổng A (PORTA) được kết nối với PE8-PE15 của GPIOE.

Các đường cổng B (PORTB) được kết nối với PB12-PB15 của GPIOB.

Các phân đoạn của các LED 7 thanh được kết nối với GPIOE như sau:

D15 - a

D14 - b

D13 - c

D12 - d

D11 - dp. (dấu thập phân)

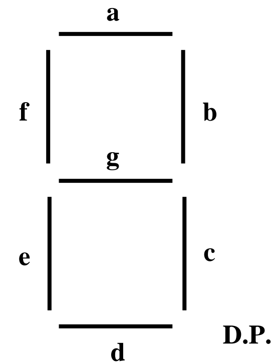
D10 - g

D9 - f

D8 - e

Để tính toán các số cần thiết cho mỗi chữ số, chúng ta ưu tiên sử dụng các bit thấp hơn D0-D7, theo như bảng sau và để chuyển số đó sang trái 8 lần.

Chữ số	a D7 128	b D6 64	c D5 32	d D4 16	dp. D3 8	g D2 4	f D1 2	e D0 1	Giá trị HEX	Giá trị thập phân
0	1	1	1	1	0	0	1	1	F3	243
1	0	1	1	0	0	0	0	0	60	96
2	1	1	0	1	0	1	0	1	D5	213
3	1	1	1	1	0	1	0	0	F4	244
4	0	1	1	0	0	1	1	0	66	102
5	1	0	1	1	0	1	1	0	B6	182
6	1	0	1	1	0	1	1	1	B7	183
7	1	1	1	0	0	0	0	0	E0	224
8	1	1	1	1	0	1	1	1	F7	247
9	1	1	1	1	0	1	1	0	F6	246



Chương trình sau đây xuất ra các số 0-9 trong một vòng lặp có độ trễ đến một trong các Led 7 thanh.

Chương trình sử dụng một bảng tra để chuyển số thập phân thành 7 thanh theo bảng sau:

```
int main (void)
{
int i,j;
char seg[10]={243, 96, 213, 244, 102, 182, 183, 224, 247, 246};
Seg_7_Init();
GPIOB->ODR = (unsigned int) 0x8000; //display at unit 1
while (1)
{
for (i=0 ; i<10 ; i++)
{
LED_PORT->ODR = seg[i] << 8; //to the high part of ODR
for (j=0 ; j<0x300000 ; j++);
}
}
}
```

Sau đây là đoạn chương trình khởi tạo Led 7 thanh (**Seg_Init ()**). Nó được viết đơn giản hơn một chút so với ARM_Project để làm cho nó rõ ràng hơn.

//I/O port B clock enable

```
#define RCC_APB2ENR_IOPBEN ((unsigned int)0x00000008)
void Seg_7_Init(void)
{
//PB12-PB15 are the Enable of the 7_seg (ground)
RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;//Enable GPIOB clock
//Clear the relevant bits in CRH register
GPIOB->CRH &= ( 0x0000ffff);

//Define PB12-PB15 as outputs
GPIOB->CRH |= (0x20000000 >>12)+(0x20000000>>8)+(0x20000000>>4) +
(0x20000000);
//PE8-PE15 are the DATA to 7_seg
#define RCC_APB2ENR_IOPEEN ((unsigned int) 0x00000040)
RCC->APB2ENR |= RCC_APB2ENR_IOPEEN;//Enable GPIOE clock
GPIOE->CRH &= ((unsigned int)0x00000000);//Clear the CRH register
GPIOE->CRH  = 0x22222222;//Define PE8-PE15 as outputs
}
```

Chương trình sau đây hiển thị các số từ 0 đến 9 trong bốn Led 7 thanh. Chương trình sử dụng một hàm lấy số để hiển thị và số đơn vị để hiển thị.

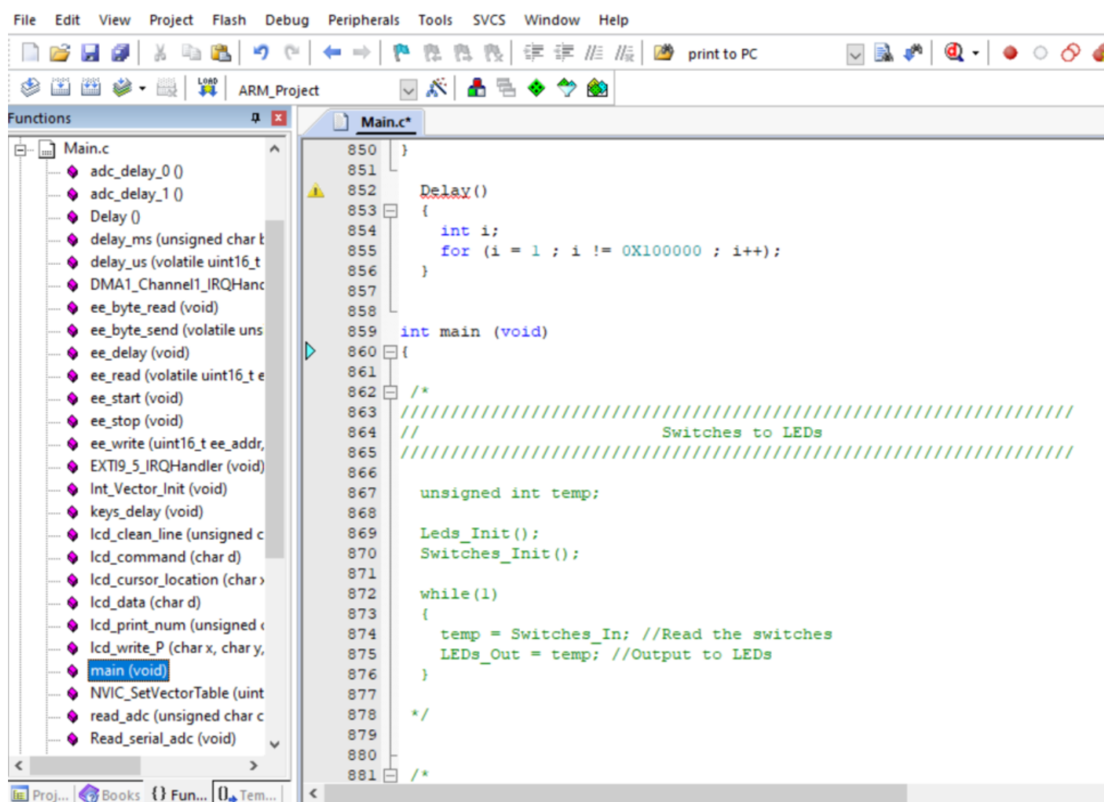
```
void Disp_7_Seg (int unit, char numb)
{
GPIOB->ODR = (unsigned int) 0x8000;
LED_PORT->ODR = numb << 8; // to the high part of ODR
}
int main (void)
{
int i,j,k;
char seg[10]={ 243, 96, 213, 244, 102, 182, 183, 224, 247, 246};
Seg_7_Init();
while (1)
{
for (j=0; j < 4; j++)
{
for (i=0 ; i<10 ; i++)
{
```

```
Disp_7_Seg (j , seg[i]);
for (k=0 ; k<0x300000 ; k++);
}
}
}
}
```

Trình tự thực hiện:

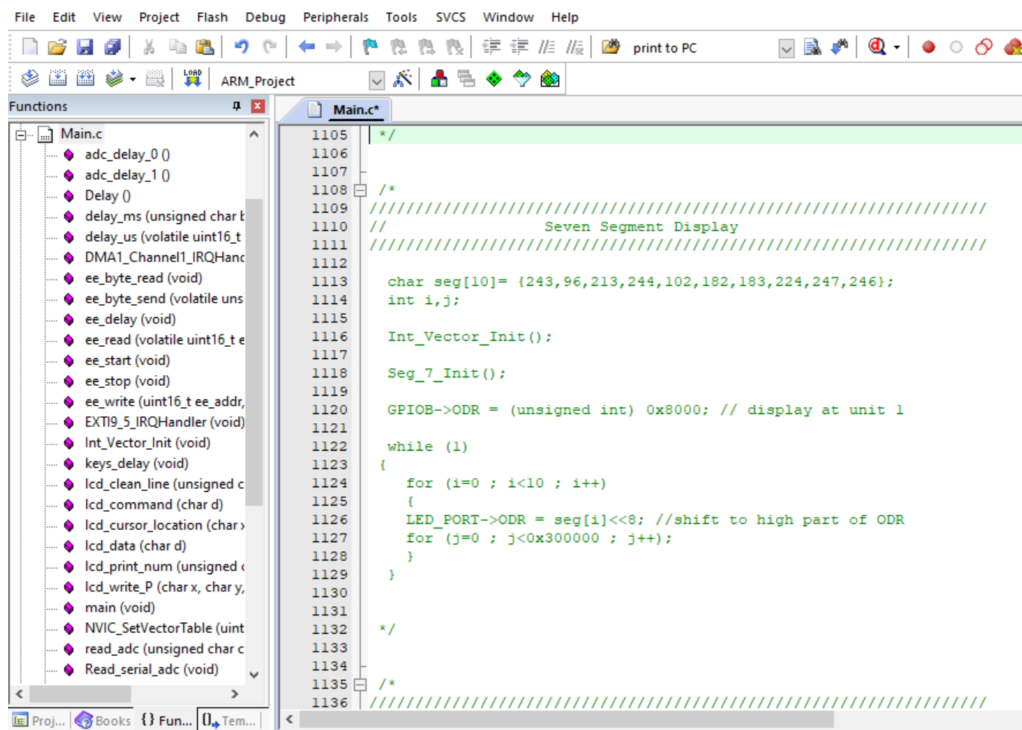
1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project**. Nhập vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này, đường dẫn đến tệp ARM_Project.uvprojx là“C:\Courses\3192\S-ARM_V3\ARM_Project”

2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:



Hình 2. 29

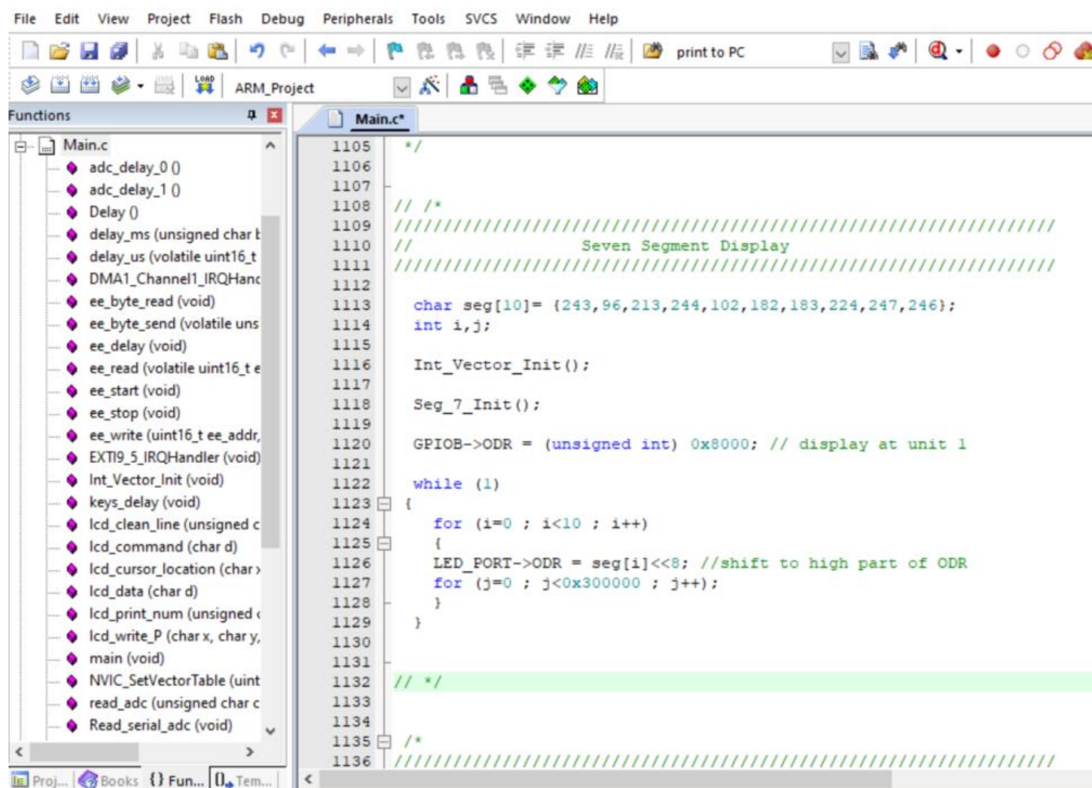
3. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:



Hình 2. 30

Phần này chứa chương trình **Seven Segment Display**

4. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:



Hình 2. 31

5. Quan sát chương trình và so sánh nó với chương trình bài tập

```
main ()
{
char seg[10]={ 243, 96, 213, 244, 102, 182, 183, 224, 247, 246};
   int i,j;
   Seg_7_Init();
   GPIOB->ODR = (unsigned int) 0x8000; //display at unit 1
while (1)
{
for (i=0 ; i<10 ; i++)
{
LED_PORT->ODR = seg[i] << 8; //shift to high part of ODR
for (j=0 ; j<0x300000 ; j++);
}
}
}
```

6. Kích hoạt EITPS-3192.
7. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

8. Quan sát các số đang chạy trong thiết bị led 7 thanh.
9. Nhấn RST để dừng chương trình đang chạy.
10. Thay đổi chương trình thành chương trình sau:

```
void Disp_7_Seg (int unit, char numb)
{
GPIOB->ODR = (unsigned int) 0x8000;
LED_PORT->ODR = numb << 8; // to the high part of ODR
}
main ()
{
char seg[10]={ 243, 96, 213, 244, 102, 182, 183, 224, 247, 246};
   int i,j,k;
   Seg_7_Init();
while (1)
{
for (j=0; j < 4; j++)
{
```

```
for (i=0 ; i<10 ; i++)
{
Disp_7_Seg (j , seg[i]);
for (k=0 ; k<0x300000 ; k++);
}
}
}
}
```

11. Nhấn RST để dừng chương trình.

12. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng. Chương trình chuyển sang màu xanh lục.

Thí nghiệm 3.4 - Màn hình LCD

Mục tiêu:

- Màn hình tinh thể lỏng (Liquid Crystal Display - LCD).
- Cách vận hành và ghi lên LCD.
- Cách viết một chương trình ghi lên LCD.

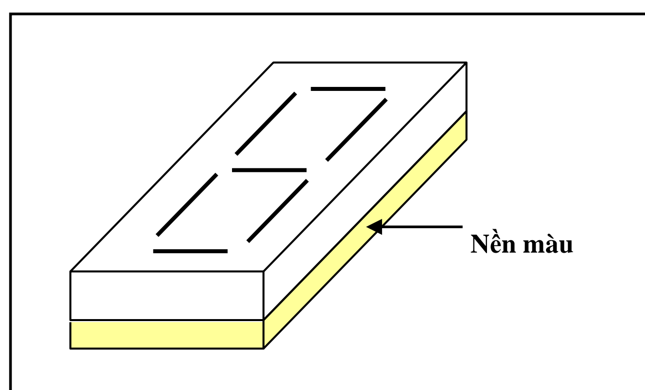
Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192

Thảo luận:

3.4.1. Màn hình tinh thể lỏng (LCD)

Màn hình tinh thể lỏng dựa trên một dung dịch cụ thể chứa trong khoảng trống giữa hai tấm thủy tinh. Một tấm được phủ bởi một lớp vàng rất mỏng, và tấm còn lại được phủ bởi một lớp rất mỏng lá vàng. Lớp vàng mỏng đến mức chúng trong suốt. Một thiết bị tinh thể lỏng 7 thanh được thể hiện trên hình 3-9.



Hình 2. 32

Dây dẫn kết nối giữa các chân của thiết bị và các lớp vàng mỏng. Tính chất độc đáo của dung dịch được thể hiện là khi có một điện trường xuyên qua nó, các phân tử sẽ được sắp xếp theo dạng tinh thể. Ở trạng thái này, dung dịch trở nên trong suốt.

Khi cung cấp điện áp cho một trong những lá trên (so với tấm vàng bên dưới), dung dịch nằm dưới nó sẽ trở nên trong suốt. Nếu bất kỳ loại nền màu nào được đặt dưới nó, ta đều có thể nhìn thấy.

Lớp này trở nên mờ đục, che mất nền, khi điện áp bị tắt, các dung dịch trở về trạng thái đục (bình thường) của chúng (do đó mới có tên "màn hình tinh thể lỏng").

Các lá vàng có thể được đặt trong bất kỳ thiết kế mong muốn nào và do đó cung cấp nhiều loại màn hình. Một lợi thế lớn của hệ thống này bắt nguồn từ thực tế rằng nó không yêu cầu dòng điện - nó được điều khiển bằng điện áp.

Hạn chế lớn nhất của nó là nó yêu cầu chiếu sáng bên ngoài và người dùng phải nhìn trực diện để có thể xem thông tin rõ ràng. Màn hình LCD hiện đại bao gồm nguồn chiếu sáng bên trong và các cấu trúc đặc biệt, cho phép nhận biết một thông báo (hiển thị) ở chất lượng cao, ngay cả khi được xem ở một góc nghiêng.

Các màn hình chữ và số chuyên dụng, cũng như các mẫu đồ họa tùy chỉnh, đều có sẵn. Phạm vi của LCD từ màn hình một dòng ngắn 8 ký tự đến màn hình lớn có kích thước bằng màn hình máy tính với 640*400 pixel (yếu tố hình ảnh).

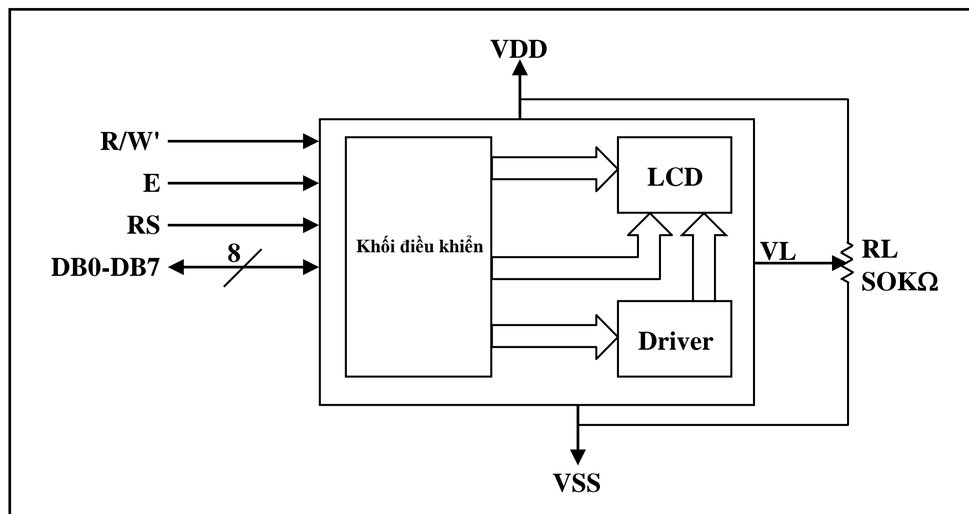
Để có thể dễ dàng xử lý khi kết nối màn hình LCD với hệ thống µP, các nhà sản xuất màn hình LCD cung cấp cho cho LCD một mạch vi điều khiển tương thích, giúp điều khiển các lá màn hình. Mạch vi điều khiển được điều chỉnh để kết nối như một thiết bị hỗ trợ của CPU. Nó chứa một RAM bên trong, nơi dữ liệu được ghi vào đó.

Mỗi byte RAM đại diện cho một mục hiển thị chữ và số. Thiết bị này cũng bao gồm một mạch giải mã chuyên dụng, được gọi là trình tạo ký tự (Character Generator - CG). CG chuyển số nhị phân do byte cung cấp thành các số nhị phân phù hợp cho các cột khác nhau của bộ hiển thị thuộc byte này.

Hãy xem xét kết nối của màn hình LCD rất phổ biến sau đây, được sản xuất bởi nhiều hãng và có nhiều cấu hình đa dạng. Phương thức kết nối là giống nhau trong mọi trường hợp và không phụ thuộc vào số lượng thiết bị hiển thị mà nó chứa.

Thiết bị chiếm hai địa chỉ I/O. Các lệnh điều khiển khác nhau được ghi vào địa chỉ thấp hơn. Dữ liệu được ghi vào hoặc đọc từ địa chỉ cao hơn. Ví dụ, giả sử rằng chúng ta muốn nhập (ghi) một biến đã cho vào một ô cụ thể của RAM bên trong. Địa chỉ của ô được chỉ định trong lệnh đến địa chỉ thấp hơn, và sau đó biến được ghi vào địa chỉ cao hơn.

Hình sau thể hiện sơ đồ khối của thiết bị.



Hình 2. 33

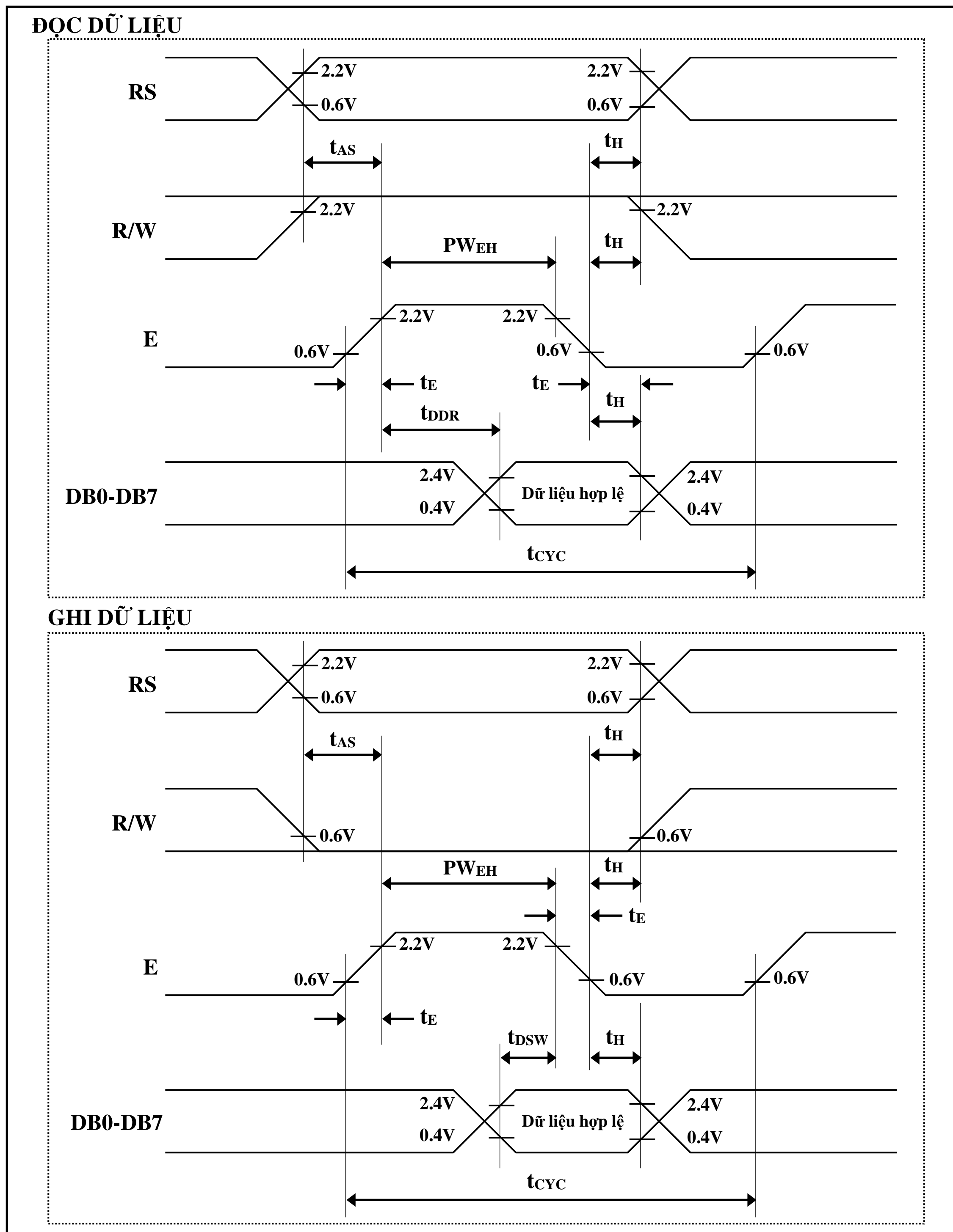
Các đường DB0-DB7 là các đường dữ liệu LCD. Như đã lưu ý, thành phần này chiếm hai địa chỉ trong bộ nhớ. RS cho biết chúng ta biết là đang truy cập địa chỉ cao hay địa chỉ thấp. Dòng R/W' cho ta biết nếu chúng ta đang ghi vào thiết bị hoặc đọc từ nó.

Khi RS ở mức thấp, việc ghi vào thanh ghi COMMAND được thực hiện; còn khi RS ở mức thấp cao thì việc ghi vào thanh ghi DATA được thực hiện.

Chúng ta có thể bỏ các chức năng đọc từ màn hình LCD. Chúng ta có thể sử dụng một hình ảnh RAM của dữ liệu cần thiết trong bộ vi điều khiển RAM và đọc nó từ đó.

Màn hình LCD cần thời gian để thực hiện các lệnh và hiển thị. Chúng ta có thể đọc trạng thái của nó và đợi trong khi bộ vi điều khiển LCD đang hoạt động. Thay vào đó, chúng ta cũng có thể thay thế trạng thái đọc này và chờ đợi trong một khoảng thời gian trễ ngắn.

Sơ đồ sau là sơ đồ thời gian của thiết bị.



Hình 2. 34

Màn hình LCD trên bộ thí nghiệm EITPS-3192 được điều khiển bởi các cổng ngõ ra. Đường R/W' được kết nối liên tục với GND. Chúng ta chỉ ghi vào màn hình LCD mà không đọc.

Các đường dữ liệu LCD DB0-DB7 được kết nối với PC8-PC15 của GPIOC.

RS được điều khiển bởi PC7 của GPIOC.

E được điều khiển bởi PC6 của GPIOC.

Màn hình LCD được khởi tạo bởi chuỗi số liệu sau vào thanh ghi lệnh của nó:

38H, 38H, 38H, 38H, 08H, 06H, 0CH

Chuỗi này cũng xóa màn hình và bật con trỏ.

Ghi 80H vào thanh ghi lệnh sẽ đưa con trỏ về đầu dòng trên của màn hình LCD.

Ghi C0H vào thanh ghi lệnh sẽ đưa con trỏ về đầu dòng dưới của màn hình LCD.

Các đoạn chương trình sau là các đoạn chương trình LCD của **ARM_Project**:

void LCD_Init()\\Initializes the LCD GPIO registers and sends commands to the to the LCD Command register.

void SYSTICK_Init()\\Initializes the core Tick counter for the delay routines

void lcd_command (char d)\\sends d to the LCD command register and makes delay of 18 ms.

void lcd_data (char d)\\sends d to the LCD data register and makes delay of 200 µs.

void lcd_cursor_location (char x, char y)\\moves LCD cursor to x,y position

void lcd_write_P(char x, char y, char *str_pointer\\moves cursor to x,y position and writes the pointed string at

void lcd_clean_line(unsigned char line)\\clean line (1 or 2)

void lcd_print_num(unsigned char x, unsigned char y, unsigned short v)\\converts a number v into a string at write it in x,y position

Chương trình sau khởi chạy màn hình LCD và hiển thị "Hello World" trên hai dòng.

```
main ()
{
LCD_Init();
SYSTICK_Init(); for delay function
//Print "Hello" at line 1 character 6
lcd_write_P ( 6, 1, ("Hello"));
lcd_clean_line(2); //clean line number 2
lcd_write_P ( 6, 2, ("World"));
while(1)
{
}
}
```

Chương trình sau đây khởi chạy màn hình LCD và hiển thị các số.

```
main ()
{
unsigned short num_for_lcd = 1000;
LCD_Init();
```

SYSTICK_Init(); for delay function

lcd_clean_line(1);//clean line number 1

lcd_write_P (4, 1, ("Numbers"));

while(1)

{

lcd_clean_line(2);//clean line number 2

num_for_lcd++;

lcd_print_num (5, 2, num_for_lcd);

delay_ms(255);

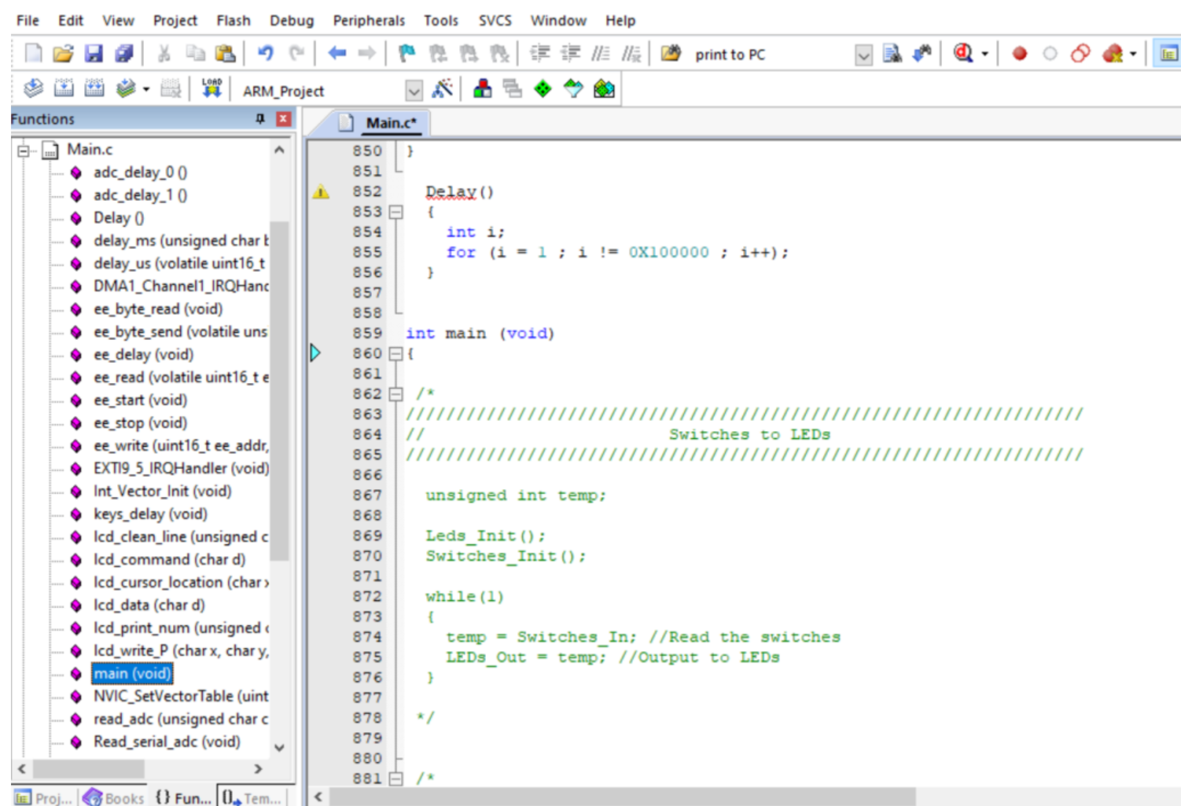
}

}

Trình tự thực hiện:

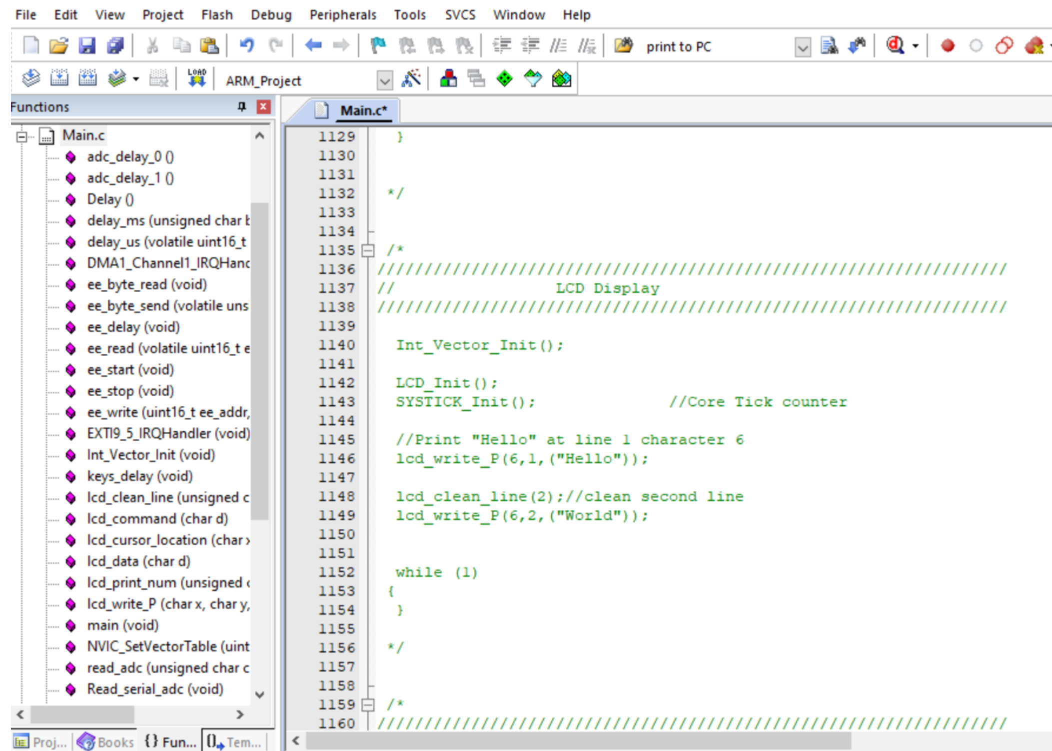
1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project**. Nhập vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này, đường dẫn đến tệp ARM_Project.uvprojx là“C:\Courses\3192\S-ARM_V3\ARM_Project”

2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:



Hình 2. 35

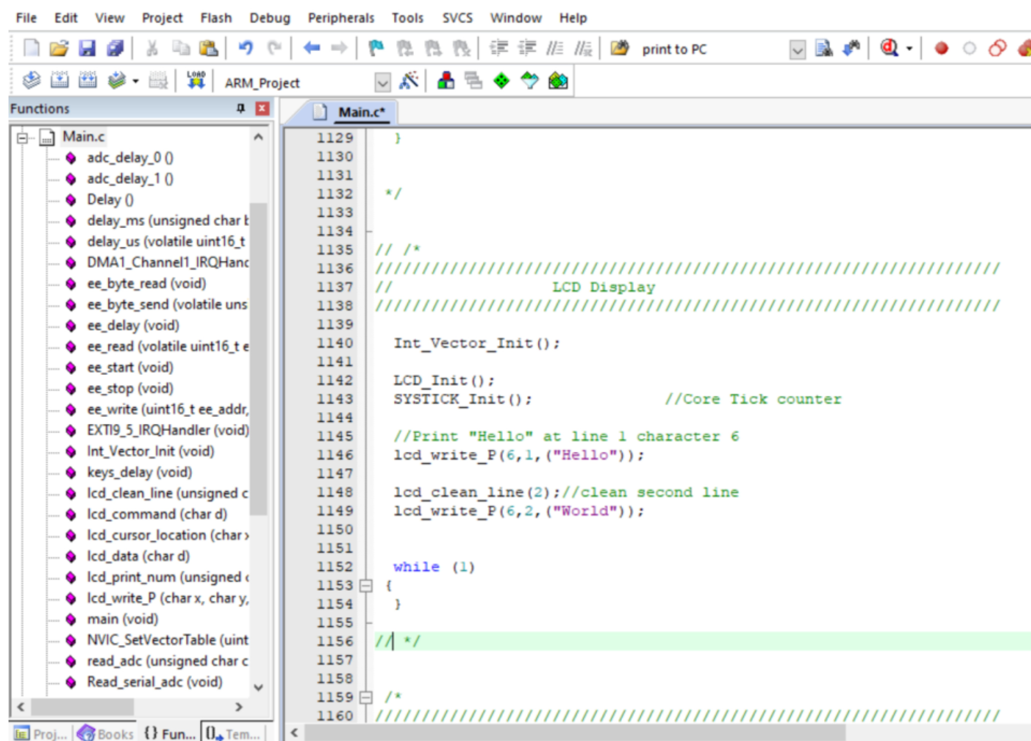
3. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:



Hình 2. 36

Phần này chứa chương trình **LCD Display**.

4. Kích hoạt chương trình bằng cách chuyển hai ký hiệu giới hạn nhận xét thành các ghi chú bằng cách thêm hai dấu gạch chéo ở đầu mỗi nhận xét và bạn sẽ nhận được màn hình sau:



Hình 2. 37

5. Quan sát chương trình và so sánh nó với chương trình bài tập:

```
main ()
{
LCD_Init();
SYSTICK_Init(); for delay function
\\Print "Hello" at line 1 character 6
lcd_write_P ( 6, 1, ("Hello") );
lcd_clean_line(2); //clean second line
lcd_write_P ( 6, 2, ("World")  );
while(1)
{
}
}
```

6. Kích hoạt EITPS-3192.

7. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

8. Quan sát thông báo trên màn hình LCD.

9. Nhấn RST để dừng chương trình đang chạy.

10. Thay đổi chương trình để in các tin nhắn khác nhau ở các vị trí khác nhau trên màn hình LCD.

11. Biên dịch, tải xuống và chạy các chương trình của bạn.

12. Thay đổi chương trình thành chương trình sau:

```
int main (void)
{
unsigned short num_for_lcd = 1000;
LCD_Init();
SYSTICK_Init(); //for delay function
lcd_clean_line(1);//clean line number 1
lcd_write_P ( 4, 1, ("Numbers")  );
while(1)
{
lcd_clean_line(2);//clean line number 2
num_for_lcd++;
lcd_print_num ( 5, 2, num_for_lcd );
delay_ms(255);
```

}

}

13. Lặp lại các bước 3-7 và xem các số đang chạy trên màn hình LCD.

14. Nhấn RST để dừng chương trình.

15. Kích hoạt các dấu '/* */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

Thử nghiệm 3.5 - GPIO và Công tắc

Mục tiêu:

- Cách viết chương trình đọc công tắc và xác định công tắc nào đang hoạt động.

Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192

Thảo luận:

3.5.1. Xác định công tắc đang được nhấn

Kết nối giữa các bộ chuyển mạch với một cổng đầu vào đã được thực hành trong chương 3, nơi chúng ta cũng đã học cách đọc trạng thái của các bộ chuyển mạch và chuyển dữ liệu đến một cổng đầu ra, cổng này sẽ xuất ra đèn LED. Để phản hồi khi kích hoạt bất kỳ công tắc nào, cần phải xác định trong số nhị phân nhận được từ các công tắc, khi nào bit tương ứng lên cao ('1'). Phép kiểm tra được thực hiện thông qua phép toán logic "AND".

Hãy xem xét ví dụ: bit D1 (bit thứ hai) phải được kiểm tra và số nằm trong bộ tích lũy. Phép toán AND được thực hiện trên bộ tích lũy bằng cách sử dụng chế độ truy cập tức thì, [AND] với số nhị phân 00000010. Các số 0 sẽ buộc các bit tương ứng của bộ tích lũy xuống thấp ('0'), bất kể chúng có giá trị nào trước đó. Bit D1 là bit duy nhất vẫn ở trạng thái (giá trị) trước đó của nó.

Nếu bộ tích lũy chứa số 0 sau phép toán này, thì D1 là '0'. Nếu một số khác 0 tồn tại trong bộ tích lũy, thì D1 là '1'.

Hãy viết một chương trình xuất ra số của công tắc được nhấn vào đèn LED đầu ra. Ví dụ: nhấn công tắc thứ tư, ta được kết quả là hiển thị số 04H ở đèn đầu ra (không phải số 08H sẽ xuất hiện ở đó nếu trạng thái của công tắc được gửi trực tiếp đến đầu ra).

Phép toán AND thay đổi nội dung của bộ tích lũy. Do đó, cần phải bảo quản nội dung của bộ tích lũy trong một thanh ghi và tải lại làm mới bộ tích lũy sau mỗi phép toán.

Lưu ý rằng chương trình này kiểm tra lần lượt tám công tắc, trong một vòng lặp. Hãy xem xét điều gì sẽ xảy ra khi hai công tắc hoạt động đồng thời.

Mục đích của thí nghiệm này là đọc trạng thái của các công tắc trên bộ hướng dẫn để xác định công tắc đang được nhấn, bằng cách chuyển số nối tiếp của nó sang các đèn LED đầu ra.

Tám công tắc của EITPS-3192 được kết nối với PD0-PD7 của PIOD.

Chúng ta phải khai báo các bit này làm các ngõ vào.

Chương trình **Swiches_Init** sau đây thực hiện điều đó.

```
void Switches_Init(void)
{
RCC->APB2ENR |= RCC_APB2ENR_IOPDEN;//Enable GPIOD clock
//PD0-PD7: Switches inputs
GPIOD->CRL &= ((unsigned int)0x00000000);//Clear the CRL register
GPIOD->CRL |= MODE_IN_FLOATING_0_7;//Define PD0-PD7 as inputs
}
```

RCC_APB2ENR_IOPDEN định nghĩa là **((unsigned int)0x00000020)**

MODE_IN_FLOATING_0_7 định nghĩa là **0x44444444**

Các khai báo này có thể được tìm thấy trong EX1_Project.

Chương trình sau đây là một chương trình chuyển các công tắc sang LED_PORT.

```
int main (void)
{
unsigned int temp;

Leds_Init();
Switches_Init();
while(1)
{
temp = Switches_In;//read switches
LED_PORT->ODR = temp;//output to LEDs
}
}
```

Chương trình sau chuyển số công tắc đang BẬT tới các đèn LED. Nó sử dụng một đoạn chương trình **quét** trả về bit nào được bật.

```
unsigned int scan (unsigned int val)
{
unsigned int shr, i;
shr = 1;
for (i = 1; i < 9; i++)
{
if (val & shr) break;
   shr <<=1;
}
return (i);
```

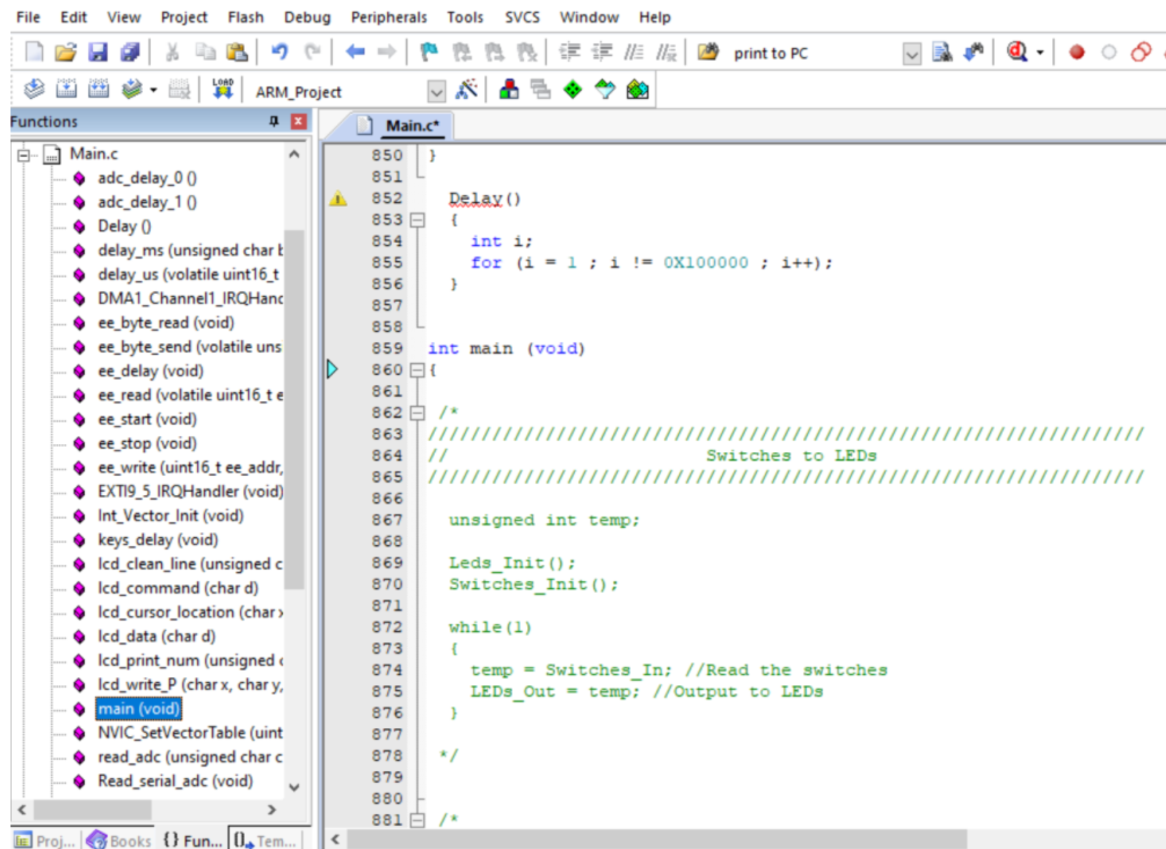
```
}
void main (void)
{
unsigned int temp;
Leds_Init();
Switches_Init();
while(1)
{
temp = Switches_In;//read switches
if (temp > 0) LED_PORT->ODR = scan(temp);
}
}
```

Số (kết quả) là bao nhiêu khi hai công tắc BẬT?

Trình tự thực hiện:

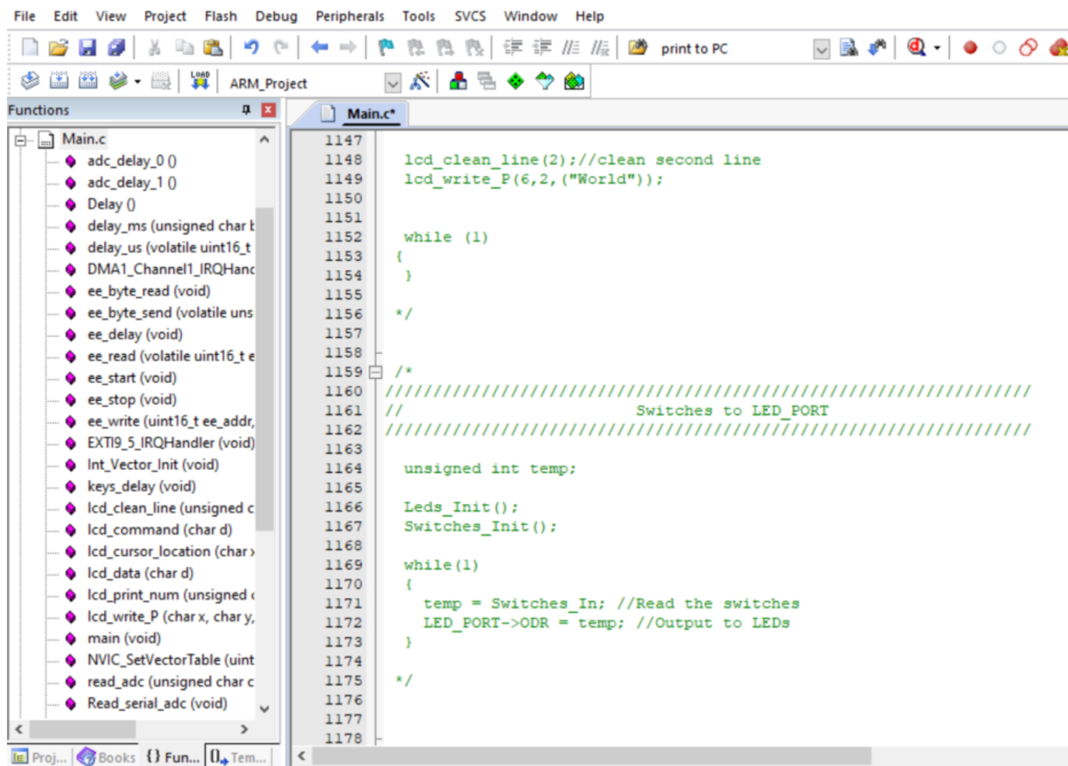
1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này, đường dẫn đến tệp ARM_Project.uvprojx là“C:\Courses\3192\S-ARM_V3\ARM_Project”

2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:



Hình 2. 38

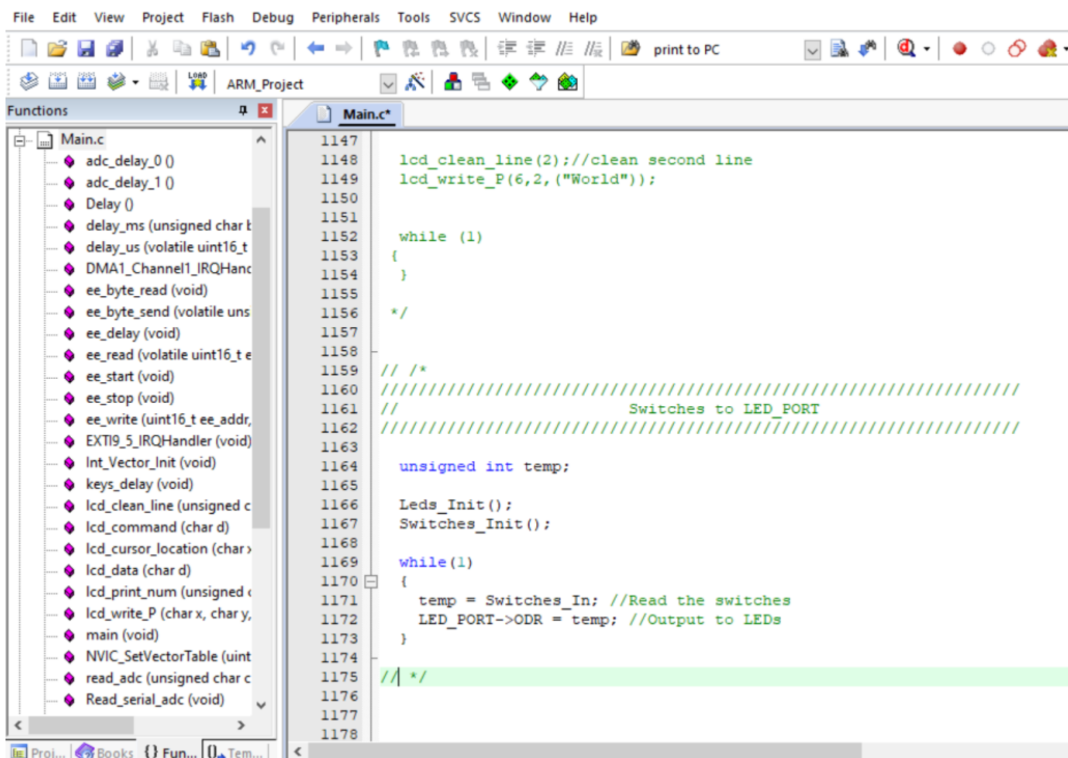
3. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:



Hình 2. 39

Phần này chứa chương trình **Switches to LED_PORT**.

4. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:



Hình 2. 40

5. Quan sát chương trình và so sánh nó với chương trình bài tập:

```
int main (void)
{
unsigned int temp;
Leds_Init();
Switches_Init();
while(1)
{
temp = Switches_In; //read switches
LED_PORT->ODR = temp; //output to LEDs
}
}
```

6. Kích hoạt EITPS-3192.

7. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

8. Thay đổi các công tắc và quan sát các đèn LED.

9. Nhấn RST để dừng chương trình đang chạy.

10. Thay đổi chương trình thành chương trình sau:

```
unsigned int scan (unsigned int val)
{
unsigned int shr, i;
shr = 1;
for (i = 1; i < 9; i++)
{
if (val & shr) break;
shr <<=1;
}
return (i);
}
int main (void)
{
unsigned int temp;
Leds_Init();
Switches_Init();
while(1)
```

{

temp = Switches_In;   //read switches

if (temp > 0) LED_PORT->ODR = scan(temp);

}

}

11. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

12. Gạt từng công tắc lên, thả xuống và quan sát các con số trên đèn LED.

13. Kiểm tra điều gì xảy ra khi hai công tắc BẬT.

14. Nhấn RST để dừng chương trình.

15. Thay hai câu sau của chương trình:

temp = Switchs_In; //read switches

if (temp > 0) LED_PORT->ODR = scan(temp);

Thành một câu sau:

if (Switchs_In) LED_PORT->ODR = scan(temp);

16. Phân tích câu này.

17. Biên dịch, tải xuống và chạy chương trình của bạn. Kiểm tra xem nó có hoạt động giống như chương trình trước không.

18. Nhấn RST để dừng chương trình.

19. Mỗi khi các công tắc khác 0, chương trình trên sẽ ghi vào các đèn LED, ngay cả khi các công tắc không được thay đổi.

Bằng cách thêm một dòng vào chương trình, chương trình sẽ ghi vào các đèn LED một lần và sẽ đợi các công tắc tới khi tất cả TẮT.

20. Thay đổi chương trình thành chương trình sau:

```
unsigned int scan (unsigned int val)
{
unsigned int shr, i;
shr = 1;
for (i = 1; i < 9; i++)
{
if (val & shr) break;
shr <<=1;
}
return (i);
}
```

```
int main (void)
{
unsigned int temp;
Leds_Init();
Switches_Init();
while(1)
{
do
{
temp = Switchs_In;
if (temp) LED_PORT->ODR = scan(temp);
} while(!temp);
//Waiting for all switches to be OFF
while(Switchs_In);
}
}
```

21. Biên dịch, tải xuống và chạy chương trình của bạn. Kiểm tra xem nó có hoạt động giống như chương trình trước không.

22. Nhấn RST để dừng chương trình.

23. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

Thí nghiệm 3.6 - Kết nối một ma trận bàn phím

Mục tiêu:

- Cách kết nối ma trận bàn phím với các cổng I/O.
- Cách viết chương trình quét các phím và xác định phím đang được nhấn.

Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192

Thảo luận:

3.6.1. Bàn phím

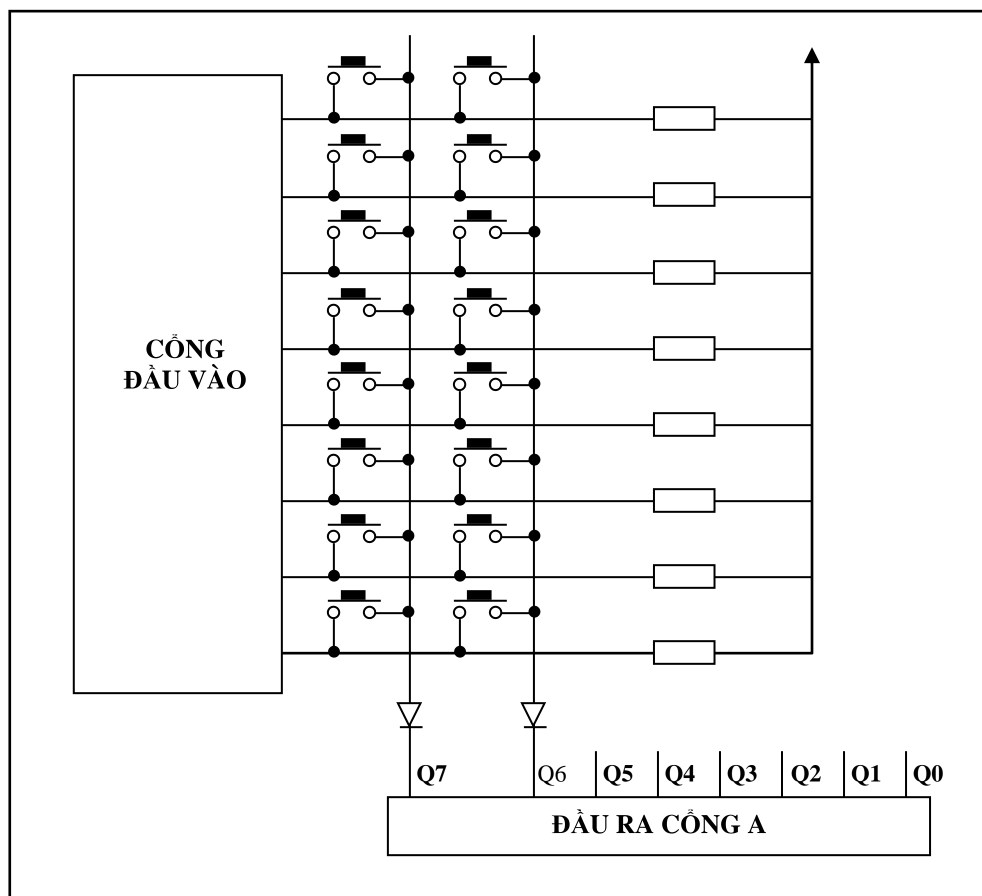
Các phương pháp quét phím được mô tả trong thí nghiệm trước chỉ định kết nối của mỗi công tắc với một bit của một cổng. Điều này yêu cầu sử dụng N/8 cổng ngõ vào N/8, trong đó N là số phím.

Một bàn phím chứa nhiều phím, vì thế phương pháp quét như vậy là một phương pháp lãng phí. Ví dụ, một bàn phím (đầy đủ) có 64 phím sẽ cần phải sử dụng tám cổng và một số lượng lớn dây.

Để khắc phục sự cần thiết của việc kết nối một cổng đầu vào cho mỗi 8 phím, bàn phím được sắp xếp trong một ma trận gồm các cột và hàng; một cổng đầu vào và một cổng đầu ra được sử dụng. Ví dụ sau đây mô tả cấu hình của một mảng 16 phím.

Để vận hành mảng này, chúng ta cần hai bit cổng ngõ ra và tám bit cổng ngõ vào. Cũng có thể sử dụng bốn bit ngõ ra và bốn bit cổng ngõ vào. Hãy xem xét cấu hình đầu tiên trong số hai cấu hình này.

Trong thí nghiệm này, chúng ta sẽ sử dụng ma trận 16 phím được kết nối (song song với 8 công tắc) với cổng đầu vào và viết chương trình xác định phím đang được nhấn.



Hình 2. 41

Nguyên tắc quét phím dựa trên thuật toán sau:

a) Xuất '0' đến đường Q7 của cổng đầu ra.

b) Quét các đường ngõ vào của cổng đầu vào. Nếu không có phím nào trong dòng này được nhấn, tất cả các đầu vào sẽ là '1'.

c) Nếu một trong các phím ở dòng này được nhấn, thì tại đường của nó sẽ nhận được số '0'. Chuyển đến đoạn (f).

d) Xuất '0' đến dòng Q6 của cổng đầu ra.

e) Lặp lại đoạn (b) và (c).

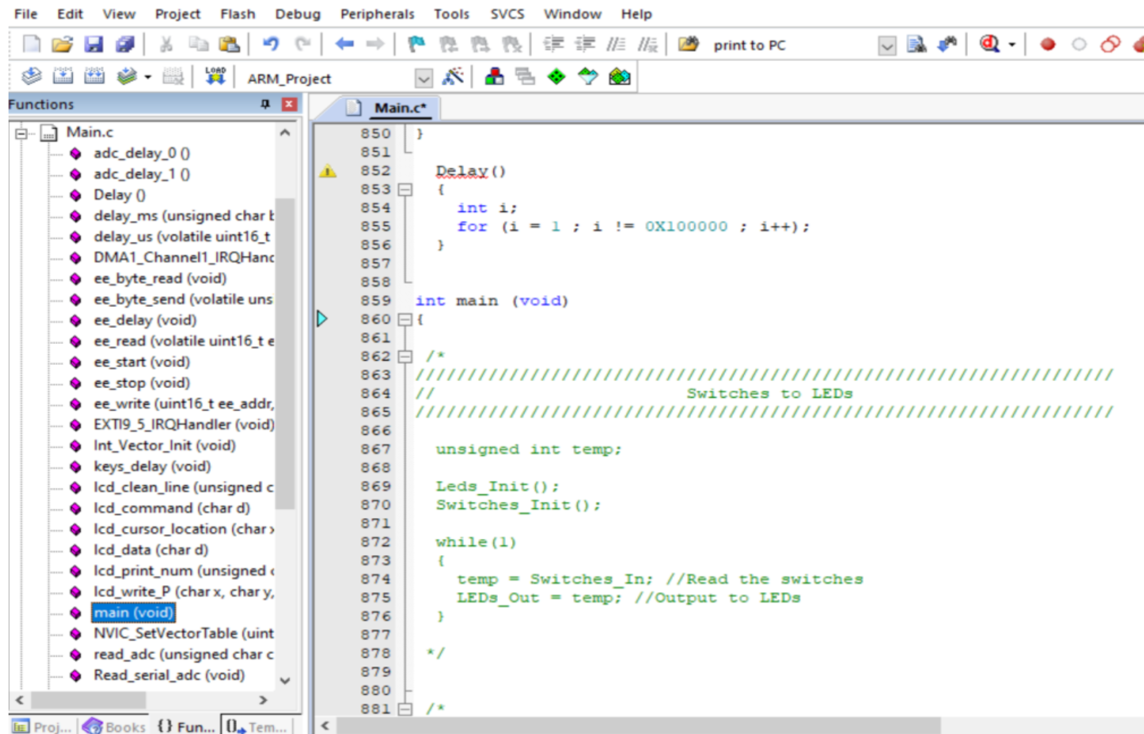
f) Số của phím được nhấn là số của đường trong cổng đầu ra kết nối với phím được nhấn tám lần (8), cộng với số của đường trong cổng đầu vào mà tại đó '0' xuất hiện.

Trong hình 2.41, ta có thể dễ dàng nhận ra phương pháp mở rộng bàn phím lên 64 phím bằng cách thêm các đường phím bổ sung.

Trình tự thực hiện:

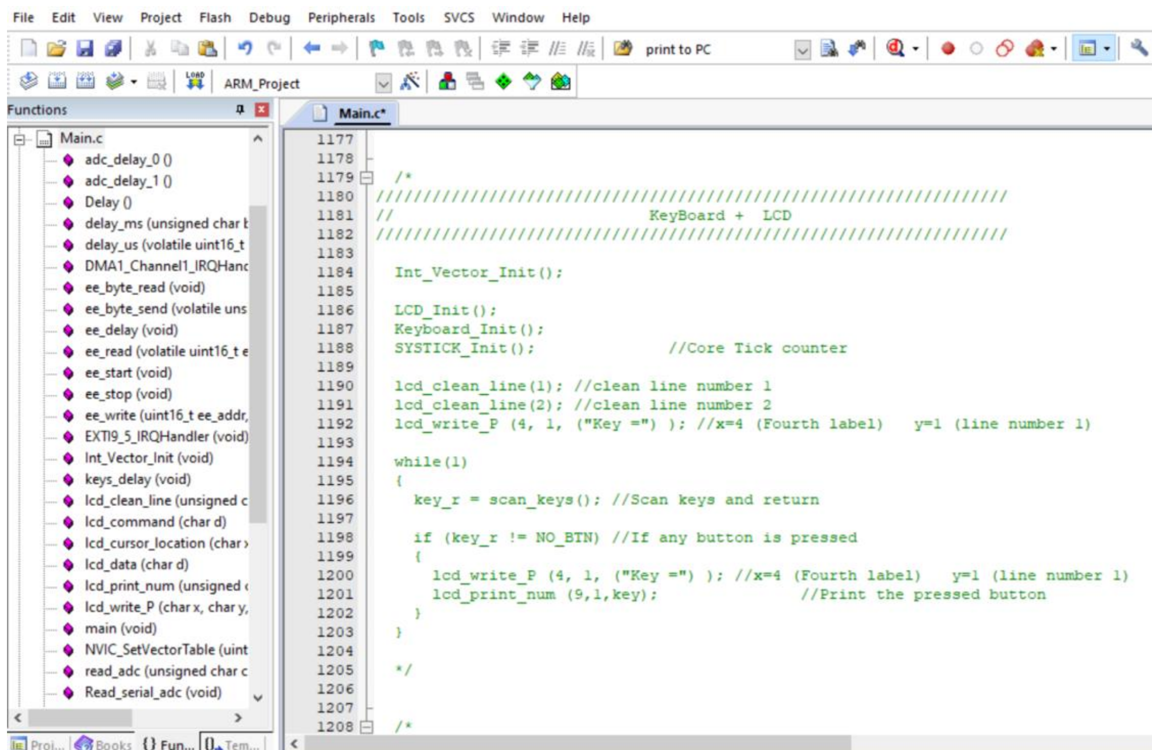
1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này, đường dẫn đến tệp ARM_Project.uvprojx là"C:\Courses\3192\S-ARM_V3\ARM_Project".

2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:



Hình 2. 42

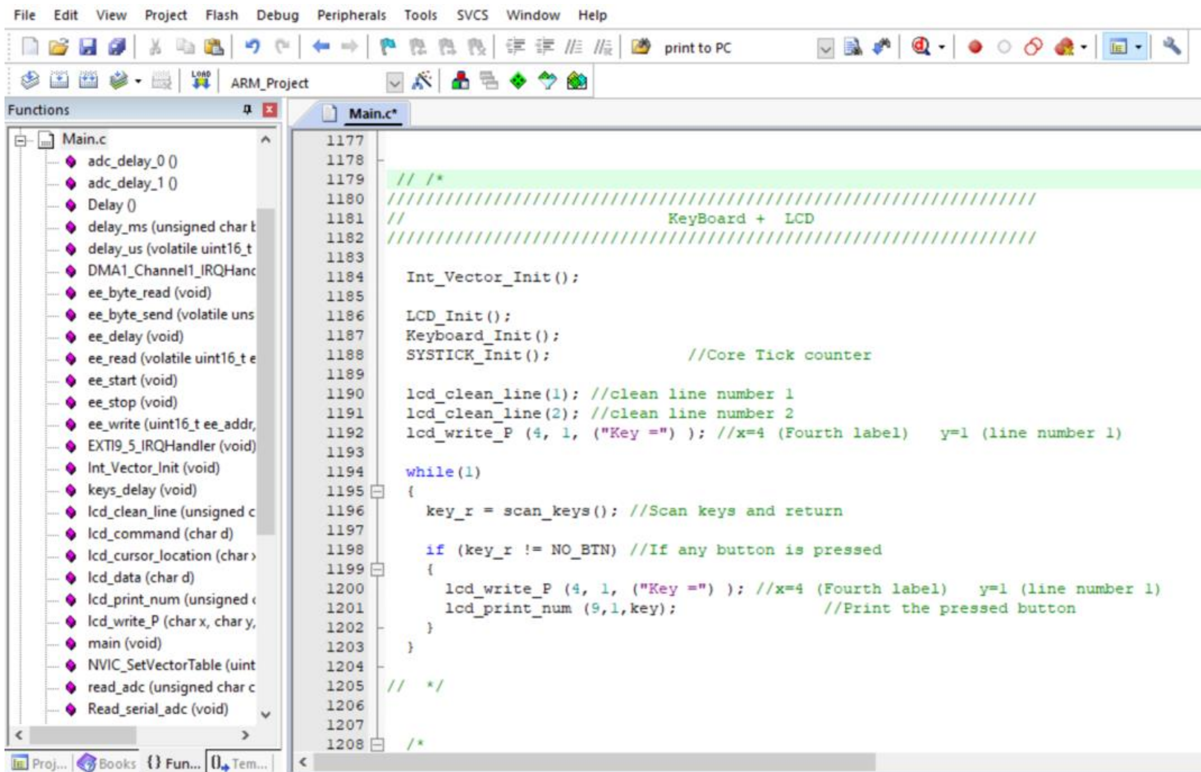
3. Cuộn xuống chương trình **main**() cho đến khi bạn nhận được màn hình sau:



Hình 2. 43

Phần này chứa chương trình **Keyboard** + **LCD**.

4. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:



Hình 2. 44

5. Quan sát và phân tích chương trình.

6. Kích hoạt EITPS-3192.

7. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

8. Nhấn các phím của bàn phím và quan sát màn hình.

9. Nhấn RST để dừng chương trình đang chạy.

10. Nhập các hàm của màn hình LCD và bàn phím thông qua các hàm của lệnh và cố gắng hiểu chúng.

Điều này là rất quan trọng.

Các chương trình và hàm mà chúng ta sử dụng hầu hết thời gian đều do người khác chuẩn bị và cải tiến.

11. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

Thí nghiệm 3.7 - Còi, Rơ le và Động cơ bước

Mục tiêu:

- Còi và cách điều khiển nó.
- Rơ le và cách điều khiển nó.
- Động cơ bước và cách điều khiển nó.
- Cách đọc nút nhấn và công tắc hành trình.

Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192
- Dây cắm thí nghiệm

Thảo luận:

Bộ thí nghiệm EITPS-3192 bao gồm một cổng bên ngoài với bốn đầu ra (TP25, TP26, TP27, TP28), được điều khiển bởi 4 bit GPIOC (PC8 đến PC11) cho phép điều khiển các thiết bị ngoại vi như: còi, rơ le và động cơ bước trên bộ thiết bị EITPS-3192.

Bộ thí nghiệm EITPS-3192 cũng bao gồm 2 ngõ vào GPIOA (PA1 và PA8) được kết nối với TP7 và TP8, và 2 đầu vào GPIOB (PB8 và PB9) được kết nối với TP5 và TP6. Các đầu vào này cho phép lập trình chúng theo các tùy chọn GPIO khác nhau và cũng có thể đọc các thiết bị như: nút nhấn hoặc công tắc hành trình trên bộ thí nghiệm EITPS-3192.

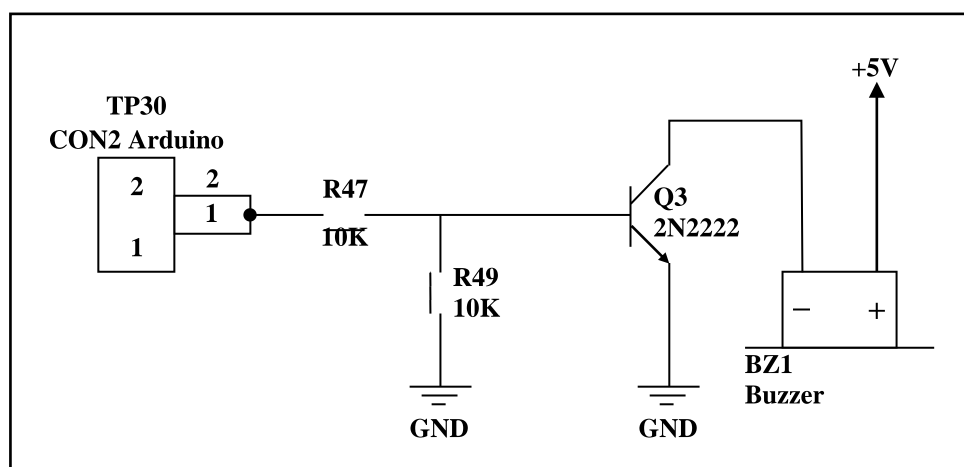
Trong thí nghiệm này, chúng ta sẽ thực hành việc điều khiển các thiết bị ngoại vi đó.

3.7.1. Còi (Buzzer)

Buzzer được cấu tạo bởi một tấm kim loại nhỏ có gắn một tinh thể gốm (tương tự như thủy tinh). Tinh thể có một đặc điểm rất đặc biệt, khi nhận một hiệu điện thế biến thiên, nó sẽ co lại và giãn nở tương ứng.

Buzzer có một mạch điện tử, mạch này tạo ra loại điện áp thay đổi dạng này, làm cho tinh thể co lại và giãn nở, do đó làm tấm kim loại dao động. Dao động của tấm kim loại tạo ra âm thanh mà chúng ta nghe thấy.

Sau đây là mạch điện còi trên bộ thí nghiệm EITPS-3192:



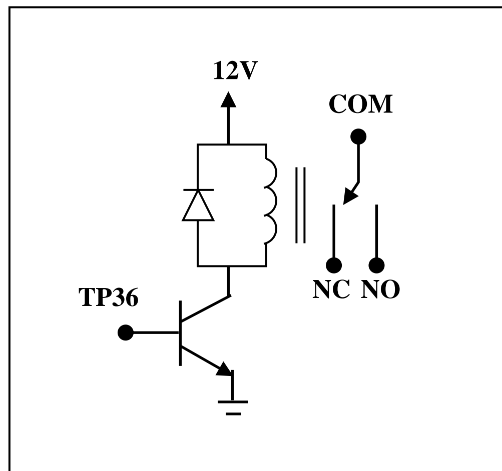
Hình 2. 45

3.7.2. Rơ le điện

Rơ le là một thiết bị đổi mạch điện, bao gồm một hoặc nhiều tiếp điểm, nó đóng hoặc mở mạch điện. Đôi khi cần phải kích hoạt một mạch điện hoạt động với dòng điện cao bằng một mạch điều khiển có dòng điện thấp. Để không làm hỏng mạch điều khiển với dòng điện thấp, chúng ta sử dụng một rơ le làm trung gian giữa hai mạch.

Rơ le được khởi động bởi một nam châm điện vận hành bởi một mạch dòng điện thấp. Kích hoạt nam châm điện sẽ kéo phần ứng và đóng các tiếp điểm rơ le, do đó điều khiển dòng điện chạy sang mạch khác có công suất dòng điện cao hơn.

Rơle chủ yếu được sử dụng kết hợp với các tải tiêu thụ dòng điện lớn. Bản thân rơ le được kích hoạt với dòng điện thấp và có thể chuyển dòng điện cao qua các tiếp điểm của nó sang mạch khác.



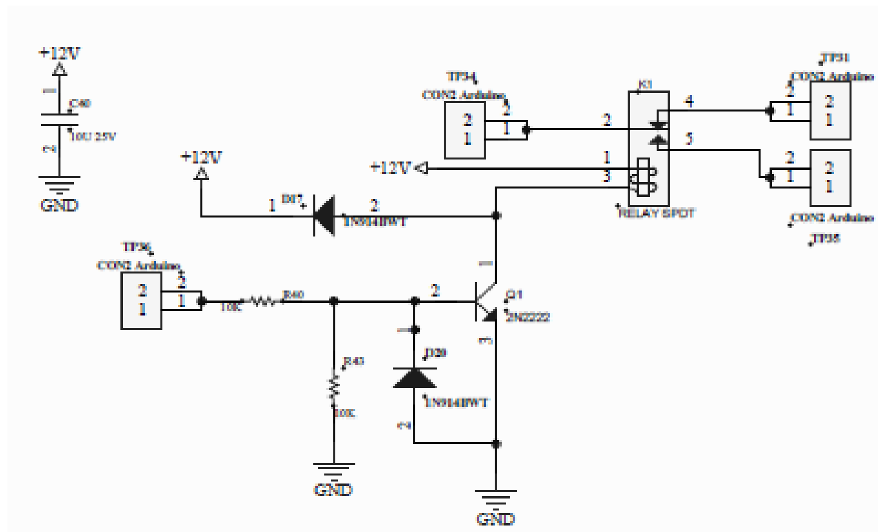
Hình 2. 46

Rơ le có 3 tiếp điểm: COM (Common - Chung), NC (Normally Close - Thường đóng), NO (Normally Open - Thường mở).

Chân COM được nối với chân NC khi không có dòng điện qua cuộn dây của nó.

Chân COM được nối với chân NO khi không có dòng điện nào chạy qua cuộn dây của nó.

Sau đây là mạch rơ le trên bộ thiết bị EITPS-3192:

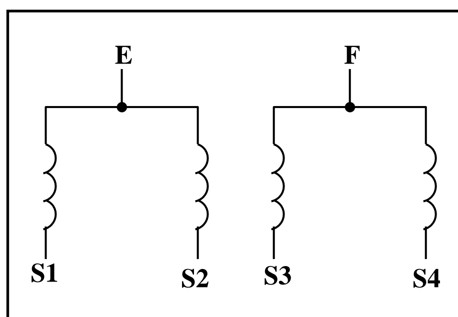


Hình 2. 47

3.7.3. Động cơ bước

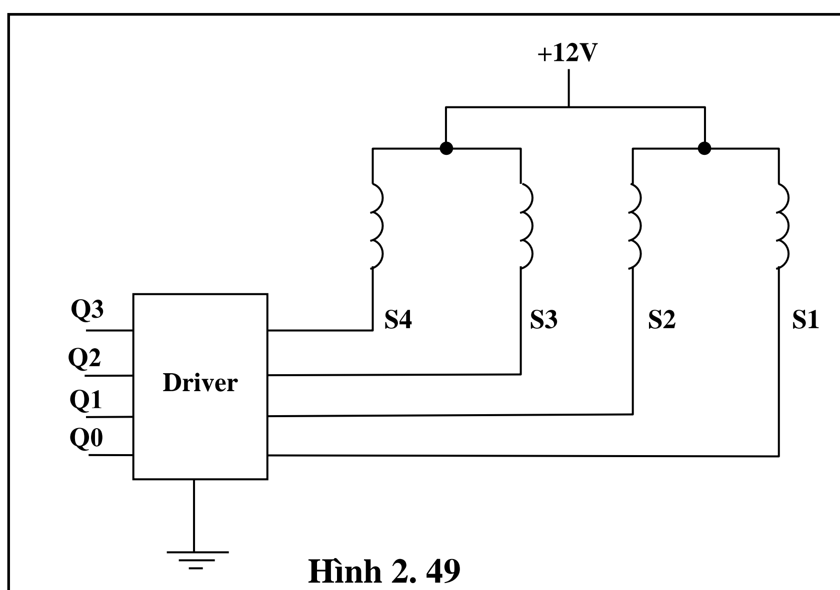
Động cơ bước (stepper) là một động cơ chuyển động theo các bước định trước, sau mỗi bước, động cơ vẫn bị khóa ở vị trí đó. Có thể kiểm soát số bước mà động cơ thực hiện, và điều này cho phép chúng ta di chuyển động cơ ở những góc rất chính xác.

Động cơ bước có 4 cuộn dây được kết nối theo cách sau:



Hình 2. 48

Các chân ST4, ST3, ST2 và ST1 của động cơ có thể được kết nối liên tiếp với các cổng ngõ ra Q0 đến Q3. Các đầu nối E và F phải được nối với cực dương của nguồn điện áp. Trong bộ thí nghiệm EITPS-3192, chúng đã được kết nối thông qua một điện trở tới điểm +12V trên bảng điều khiển.



Hình 2. 49

Xuất '0' tới một trong các đầu ra làm cho cuộn dây kết nối với nó ngắt kết nối, và dòng điện không chạy trong cuộn dây này.

Xuất '1' tới một trong các đầu ra làm cho dòng điện chạy trong cuộn dây được kết nối, từ điểm +12V đến GND.

Dòng điện trong mỗi cuộn dây chạy theo một chiều (từ +12V đến ngõ ra Q). Đây là lý do tại sao động cơ này được gọi là đơn cực.

Như mô tả trong hình 3-17, bốn đầu ra động cơ được chia thành hai nhóm: S4, S3 và S2, S1.

Khi động cơ hoạt động, mỗi cặp phải có một bit BẬT và một bit TẮT.

Để vận hành động cơ, chúng ta cần tạo ra một sự thay đổi trong bit hoạt động, luân phiên cho mỗi cặp cuộn dây. Ví dụ, động cơ quay theo một chiều sẽ được thực hiện bằng cách xuất chuỗi số sau tới cổng đầu ra.

	S1	**S2**	**S3**	**S4**	
Bước số	**D3**	**D2**	**D1**	**D0**	**Giá trị thập lục phân (Hexa)**
0	0	1	1	0	6
1	0	1	0	1	5
2	1	0	0	1	9
3	1	0	1	0	A
4	0	1	1	0	6

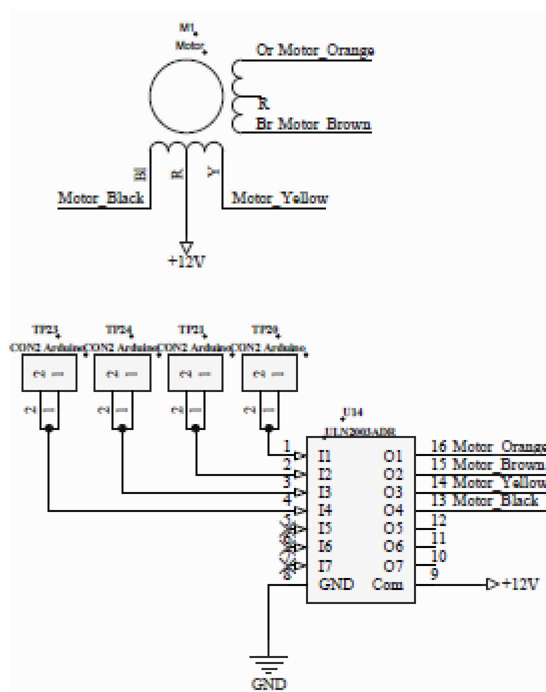
Lưu ý rằng bước số 4 đã đưa ta trở lại trạng thái bắt đầu. Nói cách khác, để quay động cơ, chúng ta phải xuất lần lượt dãy bốn số trên liên tiếp ở phía trên. Tất nhiên, một trình tự như trên không thực hiện một chuyển động quay hoàn chỉnh của động cơ mà chỉ là một phần của chuyển động quay tùy thuộc vào kích thước của mỗi bước.

Độ trễ giữa các bước xác định tốc độ quay. Mỗi động cơ có một tốc độ quay cực đại.

Để quay động cơ theo hướng ngược lại, ta phải đảo thứ tự các số trong dãy: 6, A, 9, 5, 6...

Chuỗi không nhất thiết phải bắt đầu bằng 6.

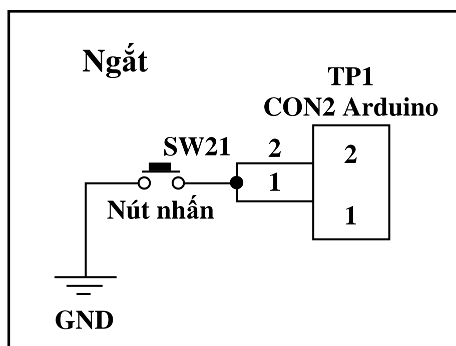
Sau đây là mạch động cơ bước trên bộ thiết bị EITPS-3192:



Hình 2. 50

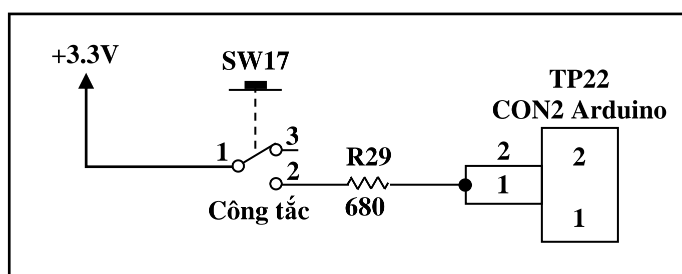
3.7.4. Nút nhấn và công tắc hành trình

Bộ thí nghiệm EITPS-3192 có một nút nhấn SW21, được kết nối như mô tả trong mạch sau.



Hình 2. 51

Bộ thí nghiệm EITPS-3192 cũng bao gồm một công tắc hành trình cho động cơ bước, được kết nối như mô tả trong mạch sau.



Hình 2. 52

Khi chúng ta kết nối một công tắc với một cổng đầu vào, chúng ta sử dụng điện trở kéo lên (pull-up) hoặc điện trở kéo xuống (pull-down), để không làm cho cổng ngõ vào bị thả nổi mức tín hiệu mà không có trạng thái rõ ràng (0 hoặc 1).

Nút nhấn cần một điện trở kéo lên (pull-up) và công tắc hành trình cần một điện trở kéo xuống (pull-down).

3.7.5. Cổng đầu ra bên ngoài

VCC của bộ vi điều khiển STM32F100 (giống với hầu hết các bộ vi điều khiển hiện đại) là 3,3V. Nó cũng bị hạn chế về khả năng điều khiển dòng điện.

Chúng ta có thể kết nối các cổng GPIO với các tải dòng điện thấp.

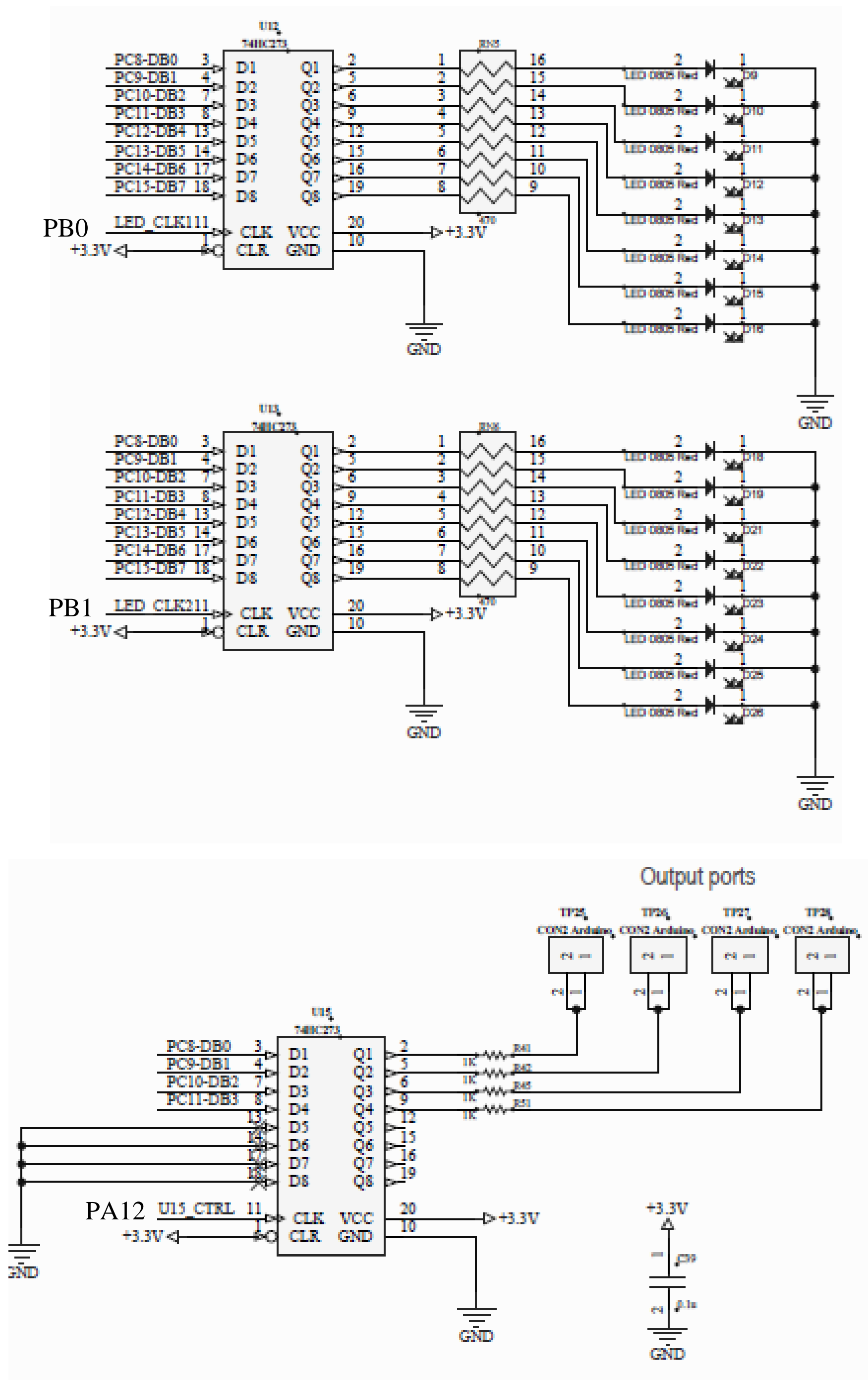
Đối với các tải dòng điện cao hơn hoặc cho các đầu ra đa năng khác, chúng ta cần một cổng bên ngoài làm bộ đệm, để điều khiển dòng điện cần thiết và để bảo vệ vi điều khiển.

8 đầu vào dữ liệu của Flip-Flop – FF) loại D 74HC273 được kết nối một đầu ra dữ liệu GPIO và đầu vào đồng hồ của nó với một bit đầu ra GPIO khác. Khi đường xung nhịp được nâng từ mức thấp lên mức cao, dữ liệu trên các đầu vào D-FF được truyền và chốt trên các đầu ra D-FF cho đến xung nhịp tiếp theo.

Tất cả ba đầu vào dữ liệu 74HC273 được kết nối song song với PC8-PC15 của GPIOC, ngoại trừ cổng ra đa năng được kết nối với PC8-PC11.

Các cổng LED đầu vào CLK được điều khiển bởi PB0 và PB1 của GPIOB.

Trong bộ thí nghiệm EITPS-3192, chúng ta sử dụng ba đơn vị Flip-Flop loại D 74HC273 với các đầu vào reset và xung nhịp chung làm bộ đệm cho ma trận LED và các đầu ra đa năng.



Hình 2. 53

Cổng đầu vào đa năng CLK được điều khiển bởi GPIOA PA12.

Các đường CLR được kết nối với VCC.

Tất cả các bit này (PC8-PC15, PB0, PB1 và PA12) phải được khai báo là các bit cổng đầu ra.

3.7.6. Khởi tạo GPIOA, GPIOB và GPIOC

Như được mô tả trong phần 3.1.2, 3.1.3 và 3.1.4, mỗi bit trong số 16 bit của GPIO có thể được xác định là đầu vào tương tự (analog), đầu vào số (floating, pull down, pull up), đầu ra đa năng (General Purpose), đầu ra chức năng thay thế (Alternate Function) trong các tùy chọn tốc độ khác nhau.

Các tùy chọn cấu hình này được xác định bởi hai thanh ghi cấu hình 32 bit CRL và CRH.

Nội dung của thanh ghi CRL định cấu hình 8 bit thấp (bit 0 đến bit 7) cho GPIO. Bốn bit của CRL định cấu hình cho một bit của cổng GPIO. Hai bit xác định chế độ (Mode) của cổng và hai bit xác định cấu hình của chế độ này.

CNF1	**CNF0**	**MODE1**	**MODE0**	**Chức năng**	**Cấu hình**	**ODR bit**	**MOS**
0	0	0	0	Đầu vào	Analog	-	
0	1	0	0	Đầu vào	Float	-	
1	0	0	0	Đầu vào	Pull-Down	0	
1	0	0	0	Đầu vào	Pull-Up	1	
0	0	0	1	Đầu ra GP	Push-Pull	0 hoặc 1	10MHz
0	1	0	1	Đầu ra GP	Open-Drain	0 hoặc 1	10MHz
1	0	0	1	Đầu ra AF	Push-Pull	-	10MHz
1	1	0	1	Đầu ra AF	Open-Drain	-	10MHz
0	0	1	0	Đầu ra GP	Push-Pull	0 hoặc 1	2MHz
0	1	1	0	Đầu ra GP	Open-Drain	0 hoặc 1	2MHz
1	0	1	0	Đầu ra AF	Push-Pull	-	2MHz
1	1	1	0	Đầu ra AF	Open-Drain	-	2MHz
0	0	1	1	Đầu ra GP	Push-Pull	0 hoặc 1	50MHz
0	1	1	1	Đầu ra GP	Open-Drain	0 hoặc 1	50MHz
1	0	1	1	Đầu ra AF	Push-Pull	-	50MHz
1	1	1	1	Đầu ra AF	Open-Drain	-	50MHz

Nội dung của thanh ghi CRH định cấu hình 8 bit cao (bit 8 đến bit 15) cho GPIO. Bốn bit của CRH định cấu hình cho một bit của cổng GPIO. Hai bit xác định chế độ (Mode) của cổng và hai bit xác định cấu hình của chế độ này.

MOS – Tốc độ đầu ra tối đa (MOS)

GP – Đa năng (GP)

AF – Chức năng thay thế (AF)

Bit 0-3 trong CRL/CRH định cấu hình bit GPIO0/GPIO8.

Bit 4-7 trong CRL/CRH định cấu hình bit GPIO1/GPIO9.

Bit 8-11 trong CRL/CRH định cấu hình bit GPIO2/GPIO10.

Bit 12-15 trong CRL/CRH định cấu hình bit GPIO3/GPIO11.

Bit 16-19 trong CRL/CRH định cấu hình bit GPIO4/GPIO12.

Bit 20-23 trong CRL/CRH định cấu hình bit GPIO5/GPIO13.

Bit 24-27 trong CRL/CRH định cấu hình bit GPIO6/GPIO14.

Bit 28-31 trong CRL/CRH định cấu hình bit GPIO7/GPIO15.

Khi chúng ta xác định một bit, chúng ta phải cẩn thận để không thay đổi cấu hình các bit khác.

Giá trị reset (0x44444444) của CRL và CRH xác định tất cả các đường IO dưới dạng đầu vào thả nổi (float). Điều này được thực hiện để bảo vệ GPIO. Là một đầu vào, những gì được kết nối tới đường IO (không có, VCC, GND hoặc một số điện áp tương tự) là không quan trọng.

3.7.7. Điều khiển tải sử dụng các bit GPIO

Chương trình dưới đây thực hiện những việc sau:

- Xác định GPIOA PA1 là đầu vào với điện trở kéo lên (cho nút nhấn) và chuyển nó đến cổng bên ngoài TP26 (bit 1).
- Xác định GPIOA PA8 là đầu vào với điện trở kéo xuống (cho công tắc hành trình) và chuyển nó đến cổng bên ngoài TP25 (bit 0).
- Xác định GPIOA PA12 là đầu ra để điều khiển cổng bên ngoài.
- Xác định GPIOC PC8 đến PC15 làm đầu ra.

```
#defineLATCH0 GPIOA->BSRR= (GPIO_BSRR_BR12);//0 to PA12
#defineLATCH1 GPIOA->BSRR= (GPIO_BSRR_BS12);//1 to PA12 for latching
unsigned char temp;
Leds_Init();
//GPIOA Settings
RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;//Enable GPIOA clock
GPIOA->CRH &= PBIT12_CLR;//0xFFF0FFFF Clear PA12
GPIOA->CRH |= (MODE_OUT_10MHZ_PP >> PBIT12);//Shift right 12 bits and OR for PA12
//GPIOA->CRH |= (0x10000000 >> 12); //Shift right 12 bits and OR for PA12
GPIOA->CRH &= 0xFFFFFFF0;//Clear PA8
GPIOA->CRH |= 0x00000008;//PA8 input with pull-up or down according to ODR
GPIOA->CRL &= 0xFFFFFF0F;//Clear PA1
GPIOA->CRL |= 0x00000080;//PA1 input with pull-up or down according to ODR
GPIOA->ODR &= 0xfeff;//0 for PA8 pull-down and leave 1 for PA1 pull up
```

```
//GPIOC Settings
RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;//Enable GPIOC clock
GPIOC->CRH &= PBIT8_15_CLR;//0x0
GPIOC->CRH = MODE_OUT_2MHZ_PP_0_7;//0x22222222 to declare C8-C15 as outputs
while(1)
{
temp = (GPIOA->IDR);
temp &= 0x0102;//Clear all bits besides bit1 and bit8
if ((temp & 0x100) == 0x100)//Check bit8
{
temp &= 2;//Clear bit8 and leave Bit1 as is
temp |= 1;//Set bit0
}
LED_PORT->ODR = temp;//Output to LEDs
GPIOC->BSRR = (temp << 8) | ((char)(~temp) << 24 );//move D0-D3 to PC8-PC11
LATCH0;//Lower PA12 to 0
LATCH1;//Raise PA12 to 1
}
```

Chúng ta sẽ sử dụng chương trình này để điều khiển đèn LED bên ngoài, còi và rơ le.

3.7.8. Chương trình điều khiển động cơ bước

Chương trình sau sẽ khiến động cơ bước quay 20 bước theo chiều kim đồng hồ và 20 bước ngược chiều kim đồng hồ trong vòng lặp.

```
#defineLATCH0 GPIOA->BSRR= (GPIO_BSRR_BR12);// 0 to PA12
#defineLATCH1 GPIOA->BSRR = (GPIO_BSRR_BS12);// 1 to PA12 for latching
unsigned int st[4]={6,5,9,10};
int dir,i,j,k;
int steps = 20;
unsigned int temp;
Leds_Init();
//GPIOA Settings
RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;//Enable GPIOA clock
GPIOA->CRH &= PBIT12_CLR;//0xFFF0FFFF Clear PA12
```

```
GPIOA->CRH |= (MODE_OUT_10MHZ_PP >> PBIT12);//Shift right 12 bits and OR for PA12
GPIOA->CRH |= (0x10000000 >> 12);//Shift right 12 bits and OR for PA12
//GPIOC Settings
RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;//Enable GPIOC clock
GPIOC->CRH &= PBIT8_15_CLR;//0x0
GPIOC->CRH = MODE_OUT_2MHZ_PP_0_7;//0x22222222 to declare C8-C15 as outputs
i=0;
dir=1;
k=1;
while(1)
{
temp = st[i];
LED_PORT->ODR = temp;//Output to LEDs
GPIOC->BSRR = ( temp << 8) | ( (char)(~temp) << 24 );//move D0-D3 to PC8-PC11
LATCH0;//Lower PA12 to 0
LATCH1;//Raise PA12 to 1
for (j=1; j<2000000; j++);
k += 1;
if (k == steps)
{
dir = -dir;
k = 1;
}
if (dir > 0)
{
i += 1;
if (i == 4) i = 0;
}
else
{
i -= 1;
if (i < 0) i = 3;
```

}

}

Chương trình xuất ra cổng bên ngoài các số 6, 5, 9 và A theo hướng quay với độ trễ giữa mỗi bước. Độ trễ quyết định tốc độ quay.

3.7.9. Đèn LED ma trận

Như được mô tả trong phần 3.7.5, bộ thí nghiệm EITPS-3192 bao gồm 16 đèn LED được kết nối với hai cổng bên ngoài trong một mảng 4 X 4.

PC8-PC15 được kết nối với các đường đầu vào dữ liệu của các cổng bên ngoài và xung đồng hồ dừng được điều khiển bởi PB0 và PB1.

Trong thí nghiệm này, chúng ta sẽ vận hành một ma trận 16 đèn LED được kết nối với hai cổng đầu ra.

```
#defineLATCHB0L GPIOB->BSRR = (GPIO_BSRR_BR0);//0 to PB0
#defineLATCHB0H GPIOB->BSRR = (GPIO_BSRR_BS0);//1 to PB0 for latching
#defineLATCHB1L GPIOB->BSRR = (GPIO_BSRR_BR1);//0 to PB1
#defineLATCHB1H GPIOB->BSRR = (GPIO_BSRR_BS1);//1 to PB1 for latching
unsigned int LED[9]={1,2,4,8,16,32,64,128,0};
int i,j,k;
unsigned int temp;
//GPIOB Settings
RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;//Enable GPIOB clock
GPIOB->CRL &= 0xFFFFFFF0;//Clear PB0
GPIOB->CRL |= 0x00000001;//GP output 10MHz for bit0
GPIOB->CRL &= 0xFFFFFF0F;//Clear PB1
GPIOB->CRL |= 0x00000010;//GP output 10MHz for bit1
//GPIOC Settings
RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;//Enable GPIOC clock
GPIOC->CRH &= PBIT8_15_CLR;//0x0
GPIOC->CRH = MODE_OUT_2MHZ_PP_0_7;//0x22222222 to declare C8-C15 as outputs
while(1)
{
for (i=1; i<3; i++)
{
for (j=0; j<9; j++)
{
```

```
temp = LED[j];
GPIOC->BSRR = ( temp << 8) | ( (char)(~temp) << 24 );//move D0-D3 to PC8-PC11
if (i == 1)
{
LATCHB1L;//Lower PB1 to 0
LATCHB1H;//Raise PB1 to 1
}
else
{
LATCHB0L;//Lower PB0 to 0
LATCHB0H;//Raise PB0 to 1
}
f ( j != 8) for (k=1; k<2000000; k++);
}
}
}
```

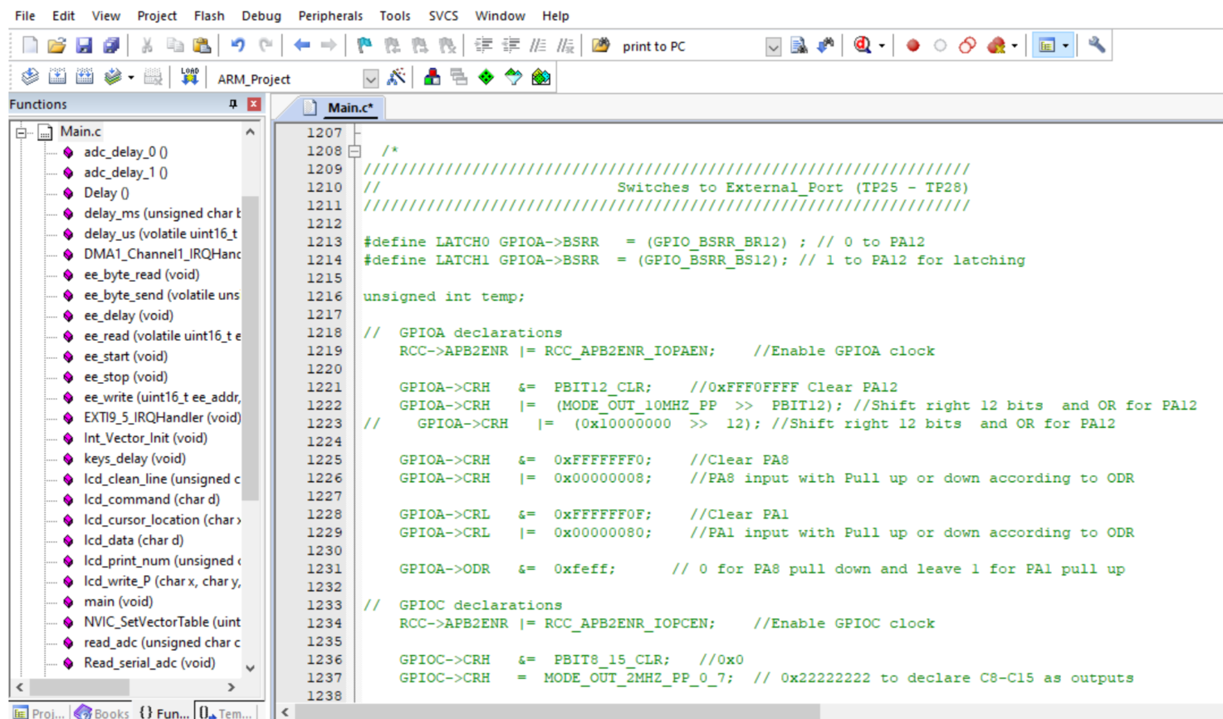
Trình tự thực hiện:

1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này, đường dẫn đến tệp ARM_Project.uvprojx là“C:\Courses\3192\S-ARM_V3\ARM_Project”

2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:

Hình 2. 54

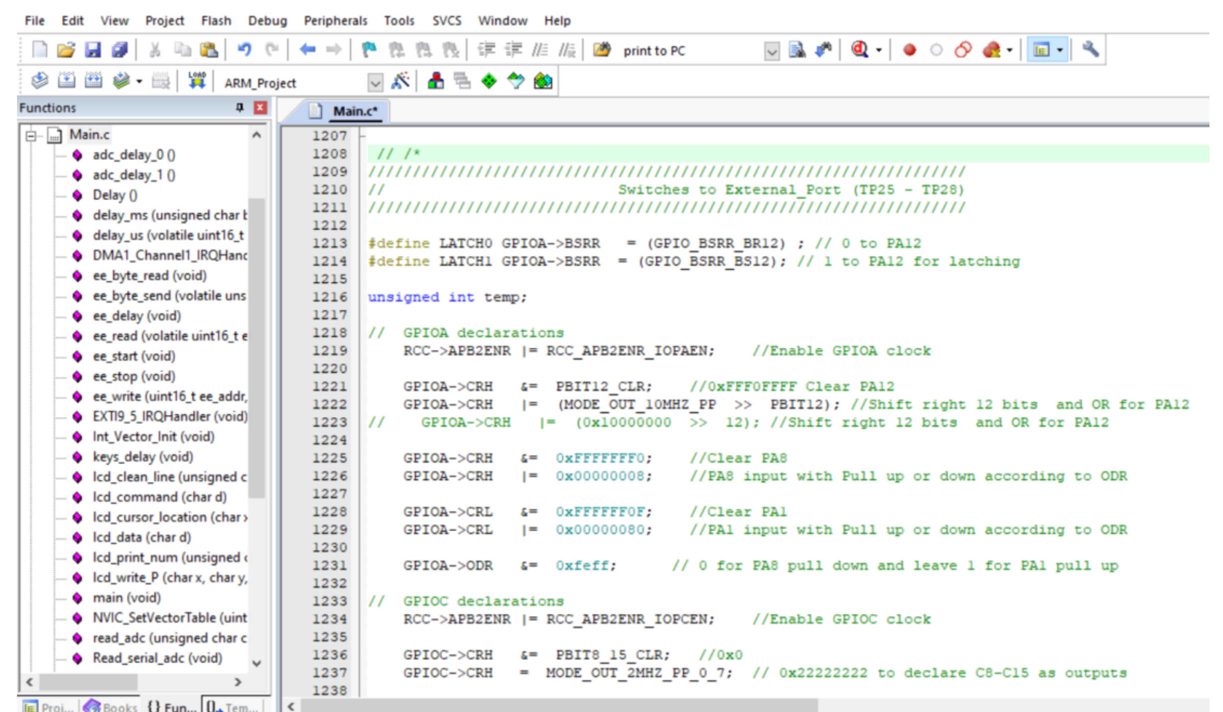
3. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:



Hình 2. 55

Phần này chứa chương trình **Switches to external port.**

4. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:



Hình 2. 56

5. Quan sát và phân tích chương trình.

6. Kích hoạt EITPS-3192.

7. Dùng một dây cắm kết nối TP1 của nút nhấn SW21 với TP7 của PA1.

8. Dùng một dây cắm kết nối TP22 của công tắc hành trình với TP8 của PA8.

9. Dùng một dây cắm kết nối TP25 của cổng bên ngoài với TP30 của còi.

10. Dùng một dây cắm và dây nối dài kết nối TP26 của cổng bên ngoài với TP32 của đèn LED màu đỏ ở bên trái.

11. Lưu và biên dịch (sửa lỗi nếu có và biên dịch lại).

Nhấn RST trước khi tải xuống, sau đó tải xuống và chạy chương trình.

Đèn LED màu đỏ bên trái sẽ BẬT. PA1 được kéo lên với một điện trở bên trong.

12. Nhấn nút nhấn SW21 và đèn LED sẽ TẮT.

13. Nhấn công tắc hành trình và còi sẽ phát ra âm thanh.

14. Ngắt kết nối TP25 khỏi TP30 của còi và kết nối nó với TP36 của rơ le.

15. Nhấn công tắc hành trình và rơ le sẽ phát ra âm thanh khi thay đổi các tiếp điểm.

16. Nhấn RST để dừng chương trình đang chạy.

17. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

18. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:

```
1258
1259
1260   /*
1261 ///////////////////////////////////////////////////////////////////////
1262 //                          Stepper motor control
1263 ///////////////////////////////////////////////////////////////////////
1264
1265 #define LATCH0 GPIOA->BSRR   = (GPIO_BSRR_BR12) ; // 0 to PA12
1266 #define LATCH1 GPIOA->BSRR  = (GPIO_BSRR_BS12); // 1 to PA12 for latching
1267
1268 unsigned int st[4]={6,5,9,10};
1269 int dir,i,j,k;
1270 int steps = 20;
1271 unsigned int temp;
1272
1273   Leds_Init();
1274
1275 //  GPIOA Settings
1276     RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;    //Enable GPIOA clock
1277
1278     GPIOA->CRH   &=  PBIT12_CLR;    //0xFFF0FFFF Clear PA12
1279     GPIOA->CRH   |=  (MODE_OUT_10MHZ_PP  >>  PBIT12); //Shift right 12 bits  and OR for PA12
1280     GPIOA->CRH   |=  (0x10000000  >>  12); //Shift right 12 bits  and OR for PA12
1281
1282 //  GPIOC Settings
1283     RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;    //Enable GPIOC clock
1284
1285     GPIOC->CRH   &=  PBIT8_15_CLR;   //0x0
1286     GPIOC->CRH   =  MODE_OUT_2MHZ_PP_0_7;  // 0x22222222 to declare C8-C15 as outputs
1287
1288   i=0;
1289   dir=1;
```

Hình 2. 57

Phần này chứa chương trình **Stepper motor control**.

19. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:

```
1258
1259
1260  // /*
1261 ///////////////////////////////////////////////////////////////////////
1262 //                            Stepper motor control
1263 ///////////////////////////////////////////////////////////////////////
1264
1265 #define LATCH0 GPIOA->BSRR   = (GPIO_BSRR_BR12) ; // 0 to PA12
1266 #define LATCH1 GPIOA->BSRR  = (GPIO_BSRR_BS12); // 1 to PA12 for latching
1267
1268 unsigned int st[4]={6,5,9,10};
1269 int dir,i,j,k;
1270 int steps = 20;
1271 unsigned int temp;
1272
1273   Leds_Init();
1274
1275 //  GPIOA Settings
1276     RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;    //Enable GPIOA clock
1277
1278     GPIOA->CRH   &=  PBIT12_CLR;    //0xFFF0FFFF Clear PA12
1279     GPIOA->CRH   |=  (MODE_OUT_10MHZ_PP  >>  PBIT12); //Shift right 12 bits  and OR for PA12
1280     GPIOA->CRH   |=  (0x10000000  >>  12); //Shift right 12 bits  and OR for PA12
1281
1282 //  GPIOC Settings
1283     RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;    //Enable GPIOC clock
1284
1285     GPIOC->CRH   &=  PBIT8_15_CLR;   //0x0
1286     GPIOC->CRH   =  MODE_OUT_2MHZ_PP_0_7;  // 0x22222222 to declare C8-C15 as outputs
1287
1288   i=0;
1289   dir=1;
```

Hình 2. 58

20. Quan sát và phân tích chương trình.

21. Dùng dây cắm kết nối TP25 của cổng bên ngoài với ST4 của động cơ bước.

22. Dùng dây cắm kết nối TP26 của cổng bên ngoài với ST3 của động cơ bước bước.

23. Dùng dây cắm kết nối TP27 của cổng bên ngoài với ST2 của động cơ bước.

24. Dùng dây cắm kết nối TP28 của cổng bên ngoài với ST1 của động cơ bước.

25. Lưu và biên dịch (sửa lỗi nếu có và biên dịch lại).

Nhấn RST trước khi tải xuống, sau đó tải xuống và chạy chương trình.

Động cơ quay 20 bước theo chiều kim đồng hồ và sau đó quay 20 bước ngược chiều kim đồng hồ trong vòng lặp.

26. Đánh dấu các điểm mà động cơ thay đổi hướng.

27. Dừng động cơ bằng tay để làm nó mất bước.

Điều này có ảnh hưởng đến các điểm thay đổi hướng không?

28. Nhấn RST để dừng chương trình đang chạy.

29. Thay đổi số bước cho đến khi động cơ hoàn thành một vòng lặp trước khi đổi hướng.

30. Thay đổi chương trình để động cơ chỉ quay theo một hướng; thay đổi độ trễ giữa mỗi bước và quan sát điều đó ảnh hưởng đến tốc độ quay như thế nào.

Tìm tốc độ tối đa của động cơ.

31. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

32. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:

```
1321   */
1322
1323   /*
1324 ////////////////////////////////////////////////////////////////////
1325 //                          LEDs in matrix
1326 ////////////////////////////////////////////////////////////////////
1327
1328 #define LATCHB0L GPIOB->BSRR   = (GPIO_BSRR_BR0) ; // 0 to PB0
1329 #define LATCHB0H GPIOB->BSRR  = (GPIO_BSRR_BS0);  // 1 to PB0 for latching
1330 #define LATCHB1L GPIOB->BSRR   = (GPIO_BSRR_BR1) ; // 0 to PB1
1331 #define LATCHB1H GPIOB->BSRR  = (GPIO_BSRR_BS1);  // 1 to PB1 for latching
1332
1333 unsigned int LED[9]={1,2,4,8,16,32,64,128,0};
1334 int i,j,k;
1335 unsigned int temp;
1336
1337 //  GPIOB Settings
1338     RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;    //Enable GPIOB clock
1339
1340     GPIOB->CRL   &=  0xFFFFFFF0;    // Clear PB0
1341     GPIOB->CRL   |=  0x00000001; // GP output 10MHz for bit0
1342     GPIOB->CRL   &=  0xFFFFFF0F;    // Clear PB1
1343     GPIOB->CRL   |=  0x00000010; // GP output 10MHz for bit1
1344
1345 //  GPIOC Settings
1346     RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;    //Enable GPIOC clock
1347
1348     GPIOC->CRH   &=  PBIT8_15_CLR;   //0x0
1349     GPIOC->CRH   =  MODE_OUT_2MHZ_PP_0_7;  // 0x22222222 to declare C8-C15 as outputs
1350
1351   while(1)
1352   {
```

Hình 2. 59

Phần này chứa chương trình **LEDs in matrix**.

35. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:

```
1321   */
1322
1323 //  /*
1324 ////////////////////////////////////////////////////////////////////
1325 //                          LEDs in matrix
1326 ////////////////////////////////////////////////////////////////////
1327
1328 #define LATCHB0L GPIOB->BSRR   = (GPIO_BSRR_BR0) ; // 0 to PB0
1329 #define LATCHB0H GPIOB->BSRR  = (GPIO_BSRR_BS0);  // 1 to PB0 for latching
1330 #define LATCHB1L GPIOB->BSRR   = (GPIO_BSRR_BR1) ; // 0 to PB1
1331 #define LATCHB1H GPIOB->BSRR  = (GPIO_BSRR_BS1);  // 1 to PB1 for latching
1332
1333 unsigned int LED[9]={1,2,4,8,16,32,64,128,0};
1334 int i,j,k;
1335 unsigned int temp;
1336
1337 //  GPIOB Settings
1338     RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;    //Enable GPIOB clock
1339
1340     GPIOB->CRL   &=  0xFFFFFFF0;    // Clear PB0
1341     GPIOB->CRL   |=  0x00000001; // GP output 10MHz for bit0
1342     GPIOB->CRL   &=  0xFFFFFF0F;    // Clear PB1
1343     GPIOB->CRL   |=  0x00000010; // GP output 10MHz for bit1
1344
1345 //  GPIOC Settings
1346     RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;    //Enable GPIOC clock
1347
1348     GPIOC->CRH   &=  PBIT8_15_CLR;   //0x0
1349     GPIOC->CRH   =  MODE_OUT_2MHZ_PP_0_7;  // 0x22222222 to declare C8-C15 as outputs
1350
1351   while(1)
1352   {
```

Hình 2. 60

36. Quan sát và phân tích chương trình.

37. Lưu và biên dịch (sửa lỗi nếu có và biên dịch lại).

Nhấn RST trước khi tải xuống, sau đó tải xuống và chạy chương trình.

16 đèn LED sẽ BẬT lần lượt trong vòng lặp không đổi.

38. Nhấn RST để dừng chương trình đang chạy.

40. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

Thử nghiệm 3.8 - Vận hành DAC

Mục tiêu:

- Các DAC song song và nối tiếp.
- Cách chuyển một số đến DAC và quan sát nó được chuyển đổi thành một tín hiệu tương tự (điện áp).
- Cách viết một chương trình đọc một số nhị phân từ bộ chuyển mạch và xuất nó tới DAC.
- Cách viết một chương trình tạo ra tín hiệu tương tự (chẳng hạn như sóng tam giác) ở đầu ra của DAC nối tiếp.

Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192

Thảo luận:

3.8.1. Triển khai một DAC với một bộ khuếch đại & một mạng điện trở

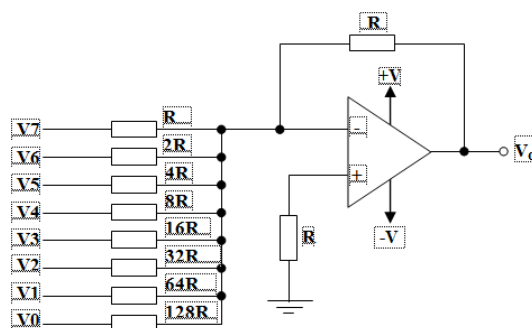
Bộ chuyển đổi số sang tương tự (DAC) là một mạch nhận bất kỳ số nhị phân nào ở các đầu vào của nó và xuất ra một điện áp (tín hiệu) ở các đầu ra của nó. Số nhị phân được tạo từ các chữ số nhị phân (bit) có thể gán giá trị của logic '0' hoặc '1', trong đó '0' bằng 0V và '1' bằng điện áp tham chiếu nhất định - V_R.

Xuyên suốt tài liệu này, chúng ta sử dụng ba thuật ngữ sau (cho mỗi công cụ chuyển đổi) như những từ đồng nghĩa hoàn toàn tương đương:

DAC, D/A, D to A - cho Bộ chuyển đổi số sang tương tự.

ADC, A/D, A to D - cho Bộ chuyển đổi tương tự sang số.

Một mạch DAC thường được cấu tạo bởi một bộ khuếch đại và một mạch điện trở, như trong hình 2.61.



Hình 2. 61

Tính điện áp đầu ra V_o dưới dạng hàm của V0-V7:

$$V_o = -\left(\frac{R}{R}V7 + \frac{R}{2R}V6 + \frac{R}{4R}V5 + \frac{R}{8R}V4 + \frac{R}{16R}V3 + \frac{R}{32R}V2 + \frac{R}{64R}V1 + \frac{R}{128R}V0\right)$$

Chia đều (rút gọn các phân số) cho R:

$$V_o = -\left(V7 + \frac{1}{2}V6 + \frac{1}{4}V5 + \frac{1}{8}V4 + \frac{1}{16}V3 + \frac{1}{32}V2 + \frac{1}{64}V1 + \frac{1}{128}V0\right)$$

Chia và nhân với 128:

$$V_o = -\frac{1}{128}(128V7 + 64V6 + 32V5 + 16V4 + 8V3 + 4V2 + 2V1 + 1V0)$$

Nếu mỗi đầu vào điện áp bằng điện áp tham chiếu nhân với giá trị của bit được kết nối với nó ('0' hoặc '1'):

$$VI = VR \cdot DI$$

Chúng ta nhận được:

$$V_o = -\frac{1}{128}VR(128D7 + 64D6 + 32D5 + 16D4 + 8D3 + 4D2 + 2D1 + 1D0)$$

Do đó, chúng ta nhận được điện áp đầu ra có giá trị tuyệt đối là một hàm của số nhị phân được cấp cho các đầu vào của mạch. Hệ số của đoạn mạch này bằng $\frac{1}{128}$ V_R.

Điện áp ở đầu ra của mạch bằng giá trị thập phân của số nhị phân, nhân với V_R và nhân với một hệ số (hệ số là một hàm của mạch). Giả sử, ví dụ rằng, số nhị phân là 01001110, bằng 78_{10}, điện áp tham chiếu là 5V và hệ số được xác định bởi mạch là 0,01. Điện áp ở đầu ra của mạch sẽ là:

$$V_o = 0.01 \cdot 5 \cdot 78 = 0.05 \cdot 78 = 3.9V$$

Điện áp đầu ra có giá trị âm. Để nhận được giá trị điện áp dương, một bộ đảo (bộ khuếch đại) có thể được kết nối với đầu ra của mạch. Cũng có thể cấp cho mạch một độ lợi (độ khuếch) cho trước, để thay đổi hệ số chung của mạch.

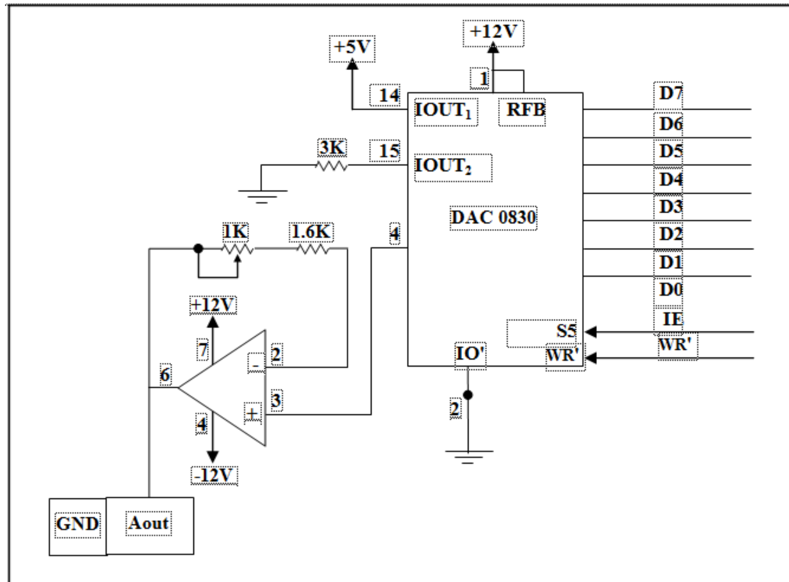
Độ chính xác của các điện trở là một yếu tố quan trọng trong các mạch DAC. Vấn đề trong các mạch như vậy là kết quả của việc phải sử dụng các điện trở có giá trị khác nhau mà vẫn phải duy trì độ chính xác cao giữa các tỷ lệ giữa chúng, như được mô tả trong hình. Do đó, lắp ráp một mạch chính xác loại này là một việc khó khăn và tốn kém. Tuy nhiên, có thể mua các mạng điện trở chính xác trong một gói duy nhất (do đó chúng có hiệu suất tương tự trong các điều kiện môi trường).

3.8.2. DAC nguyên khối

Một DAC nguyên khối là một bộ chuyển đổi số (digital) sang tương tự (analog) được lắp ráp trên một chip duy nhất. Điều này được thực hiện bằng cách sản xuất một mạch tích hợp chứa mạng điện trở, bộ khuếch đại và các mạch bù khác nhau. Có rất nhiều loại DAC trên thị trường. Sự khác biệt chính giữa chúng là ở độ phân giải (số bit của số nhị phân), tốc độ phản hồi và độ chính xác. Hầu hết, thiết bị tạo ra một dòng điện, dòng điện này chính là một hàm của số nhị phân tại các đầu vào của nó.

Bộ khuếch đại chuyển đổi các dòng điện này thành điện áp trong phạm vi được chỉ định. Các đầu vào của DAC được kết nối với các ngõ ra của cổng đầu ra.

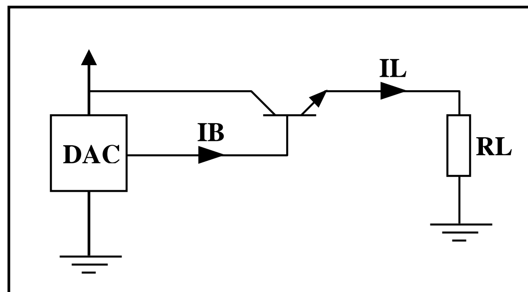
Chế độ kết nối DAC 0830 được mô tả trong hình 2.62.



Hình 2. 62

Chiết áp đặt ở đầu vào của bộ khuếch đại được sử dụng để hiệu chỉnh độ lợi của nó nhằm đạt được phạm vi yêu cầu. Thông thường, chân 15 được nối với thế điện áp và một nguồn điện áp được (đã được lọc và điều chỉnh) sẽ được kết nối với chân 14.

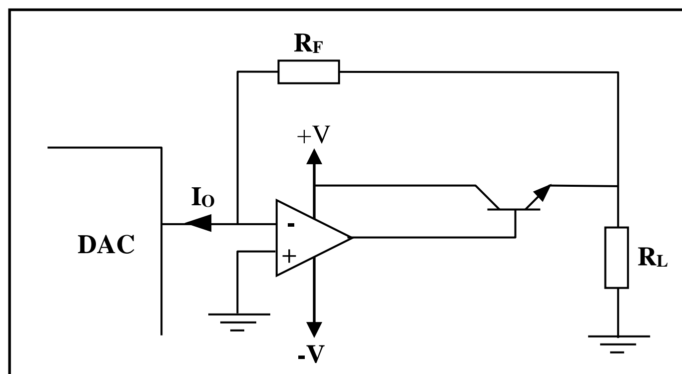
Khi cần vận hành một thiết bị nhất định (đã chọn), dòng điện đạt yêu cầu thường cao hơn dòng điện có thể được cung cấp bởi bộ khuếch đại. Một bộ khuếch đại trong một mạch như hình 2.63 sẽ được thêm vào mạch trong những trường hợp như vậy.



Hình 2. 63

DAC cung cấp dòng điện IB {bằng IL/(B+1)} cho bóng bán dẫn. Nếu yêu cầu cần có mức tăng dòng điện cao hơn, một mạch Darlington có thể được thêm nối tiếp. Một nguồn điện được kiểm soát có thể được tạo ra bởi một cấu hình tương tự.

Chúng ta sẽ nhận được một mạch chính xác hơn nếu kết nối điện trở phản hồi với tải chứ không phải với bóng bán dẫn như sau:



Hình 2. 64

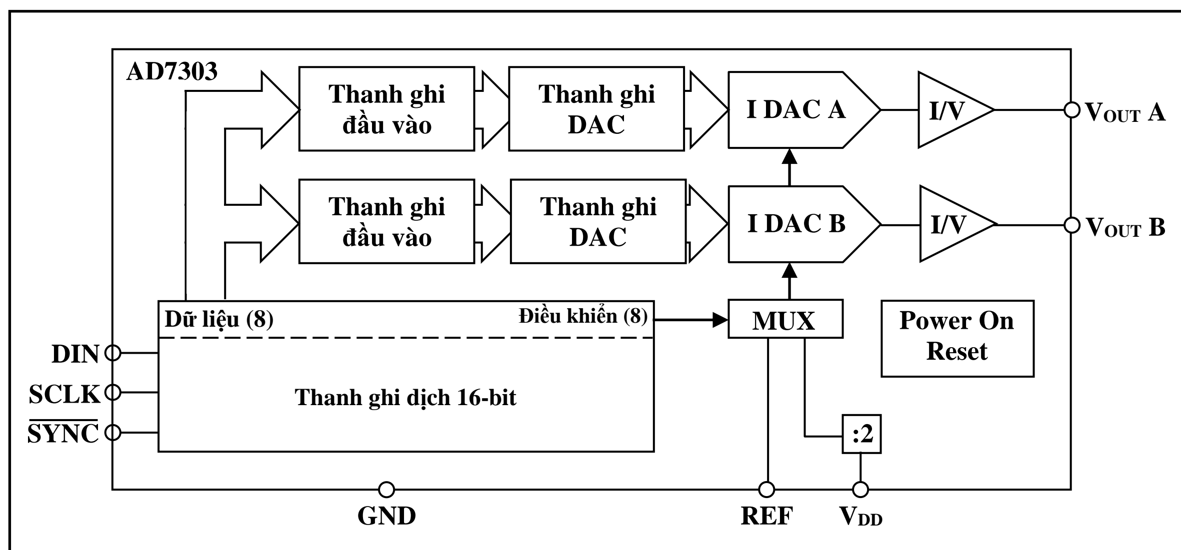
Một ứng dụng thú vị của DAC là sự tích hợp của nó thành một bộ khuếch đại điều khiển µP cho nguồn AC. Một điện áp xoay chiều được khuếch đại đến giá trị lớn nhất (tất nhiên là không bị méo) được kết nối với đầu vào V_{REF}. Đầu ra DAC sau đó sẽ là nguồn điện áp xoay chiều có giá trị là một hàm của số nhị phân tại đầu vào của nó.

Mạch DAC bao gồm một chiết áp trimmer, dùng để hiệu chỉnh bộ khuếch đại một lần. Khi DAC nhận 00 tại các đầu vào của nó, bộ khuếch động cung cấp 0V. Khi DAC nhận FF ở các đầu vào của nó, bộ khuếch đại sẽ cung cấp 5V.

3.8.3. DAC điện áp kép đầu vào nối tiếp 8 bit AD7303

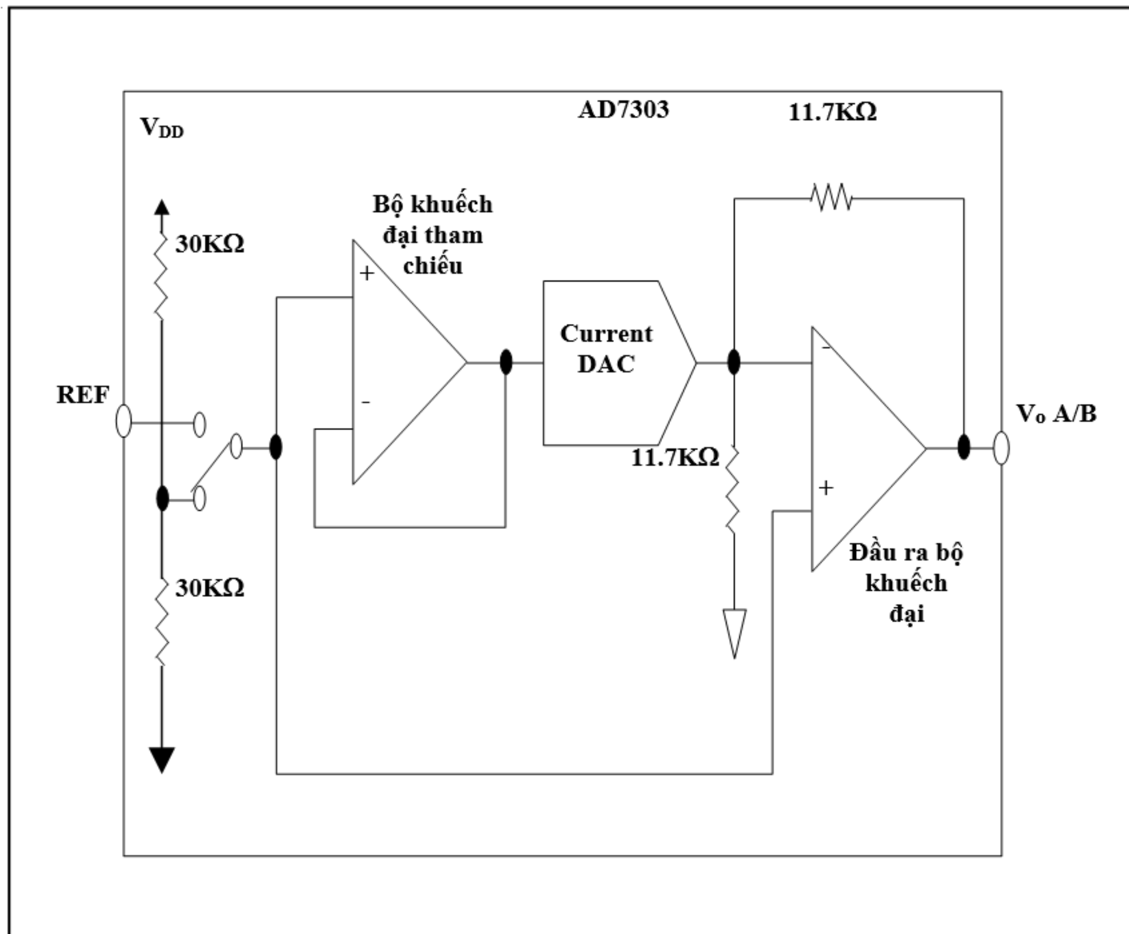
AD7303 là DAC đầu ra điện áp kép 8 bit hoạt động từ một nguồn cung cấp +2,7V đến +5,5V duy nhất. Bộ đệm đầu ra chính xác trên chip của nó cho phép các đầu ra DAC chuyển hướng. Thiết bị này sử dụng giao diện nối tiếp 3 dây linh hoạt hoạt động ở tốc độ xung nhịp lên đến 30MHz và tương thích với các chuẩn giao tiếp QSPI, SPI, microwire và bộ xử lý tín hiệu kỹ thuật số. Thanh ghi đầu vào nối tiếp rộng 16 bit; 8 bit hoạt động như các bit dữ liệu cho DAC, và 8 bit còn lại tạo thành một thanh ghi điều khiển.

Thanh ghi điều khiển trên chip được sử dụng để định địa chỉ DAC liên quan, nhằm tắt nguồn thiết bị hoàn chỉnh hoặc một DAC riêng lẻ, để chọn tham chiếu bên trong hoặc bên ngoài và cung cấp cơ sở nạp đồng bộ cho cập nhật đồng thời của các đầu ra DAC bằng phần mềm chức năng LDAC.



Hình 2. 65

AD7303 là bộ chuyển đổi số sang tương tự đầu ra điện áp kép 8 bit. Cấu trúc của nó bao gồm một bộ khuếch đại tham chiếu và một nguồn dòng điện DAC, tiếp theo là một bộ chuyển đổi dòng điện sang điện áp có khả năng tạo ra các điện áp trên đầu ra của DAC. Hình 3-26 cho thấy một sơ đồ khối của cấu trúc DAC cơ bản.



Hình 2. 66

Cả hai đầu ra DAC A và DAC B đều được đệm bên trong và các bộ khuếch đại đệm đầu ra này có đặc tính đầu ra rail-to-rail. Bộ khuếch đại đầu ra có khả năng điều khiển một tải 10KΩ tới cả V_{DD}, GND và 100pF . Lựa chọn tham chiếu cho DAC có thể được tạo bên trong từ VDD hoặc được đặt bên ngoài thông qua chân REF. Lựa chọn tham chiếu thông qua một bit trong thanh ghi điều khiển. Phạm vi trên đầu vào của tham chiếu bên ngoài là từ 1.0V đến $V_{DD}/2$.

Điện áp đầu ra từ một trong hai DAC được cho bởi:

$$V_o A/B = 2 \cdot V_{REF} \cdot (N/256) V$$

Trong đó:

V_{REF} là điện áp đặt vào chân REF bên ngoài hoặc $V_{DD}/2$ khi tham chiếu bên trong được chọn.

N là số tương đương thập phân của mã được nạp vào thanh ghi DAC và nằm trong khoảng từ 0 đến 255.

3.8.4. Đầu ra Analog

AD7303 có hai DAC đầu ra điện áp độc lập với độ phân giải 8-bit và chế độ hoạt động theo rail-to-rail. Bộ đệm đầu ra cung cấp độ lợi là hai ở đầu ra. Tốc độ quay của bộ khuếch đại đầu ra thường là 8V/µs và có sự ổn định toàn thang (full-scale) tới 8 bit với một tải điện dung 100pF trong 1.2µs.

Mã hóa đầu vào DAC là mã nhị phân tiêu chuẩn. Bảng sau đây cho thấy chức năng truyền nhị phân cho AD7303. Bất kỳ điện áp đầu ra DAC nào đều có thể được biểu thị một cách lý tưởng bằng:

$$V_{out}=(N/256)\cdot 2\cdot V_{REF}$$

Trong đó:

N là số tương đương thập phân của mã đầu vào nhị phân. N nằm trong khoảng từ 0 đến 255.

V_{REF} là điện áp đặt vào chân REF bên ngoài khi tham chiếu bên ngoài được chọn và là $V_{DD}/2$ nếu tham chiếu bên trong được sử dụng.

Đầu vào kỹ thuật số		**Đầu ra tương tự [V]**
MSB	**LSB**	
11111111		255/256 x2 xV_{REF}
11111110		254/256 x2 xV_{REF}
10000001		129/256 x2 x V_{REF}
10000000		128/256 x2 x V_{REF}
01111111		127/256 x2 xV_{REF}
00000001		1/256 x2 x V_{REF}
00000000		0

3.8.5. Giao diện nối tiếp

AD7303 có giao tiếp nối tiếp 3 dây linh hoạt tương thích với các tiêu chuẩn giao tiếp SPI, QSPI và Microwire, cũng như một loạt các bộ xử lý tín hiệu kỹ thuật số.

$\overline{(SYNC)}$ mức thấp cho phép thanh ghi dịch nhận dữ liệu từ đầu vào dữ liệu nối tiếp DIN. Dữ liệu được khóa vào thanh ghi dịch trên cạnh lên của xung nhịp nối tiếp. Tần số xung nhịp nối tiếp có thể cao tới 30MHz. Thanh ghi dịch này rộng 16 bit.

Tám bit đầu tiên là bit điều khiển và tám bit thứ hai là bit dữ liệu cho DAC.

Mỗi lần chuyển phải bao gồm một lần ghi 16 bit hoặc hai lần ghi 8 bit. Giao tiếp SPI và Microwire xuất dữ liệu trong các byte 8 bit và do đó yêu cầu hai lần truyền 8 bit.

Trong trường hợp này, đầu vào $\overline{(SYNC)}$ của DAC phải ở mức thấp cho đến khi tất cả 16 bit được chuyển đến thanh ghi dịch. Các giao tiếp QSPI có thể được lập trình để truyền dữ liệu bằng các word 16 bit.

Sau khi đưa tất cả 16 bit vào thanh ghi dịch, cạnh lên của $\overline{(SYNC)}$ thực thi lệnh đã được lập trình. Các DAC được đệm kép cho phép cho phép các đầu ra của chúng được cập nhật đồng thời.

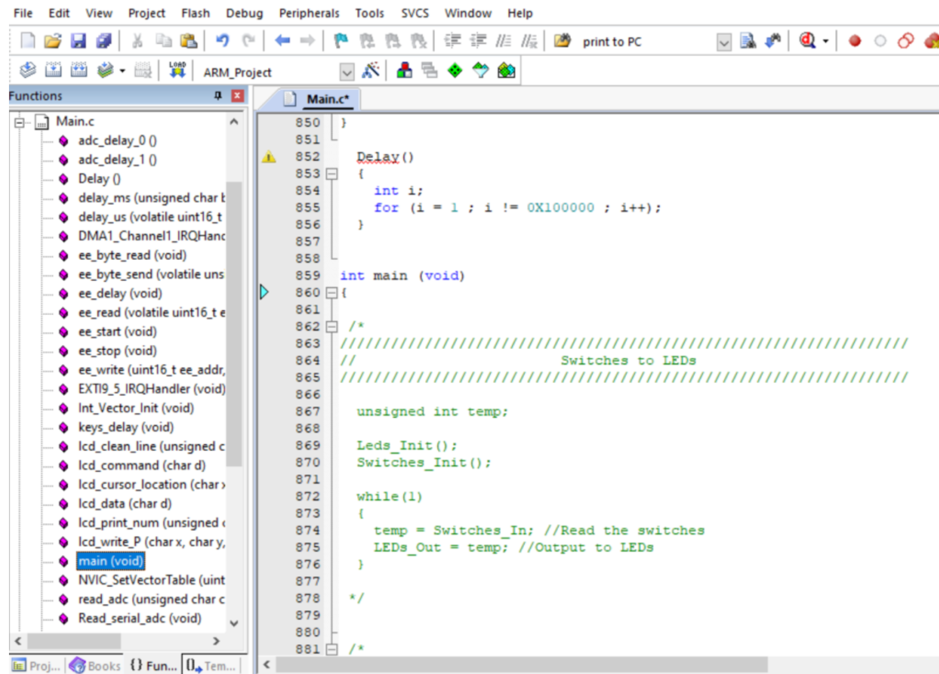
3.8.6. Bộ DAC ARM

Hầu hết các vi điều khiển ngày nay đều có các đầu ra DAC. ARM mà chúng ta sử dụng cũng có các đầu ra như vậy. Chúng ta sẽ sử dụng một kênh DAC tích hợp sẵn. DAC này được đánh dấu là DAC1 và đầu ra của nó là TP18.

Trình tự thực hiện:

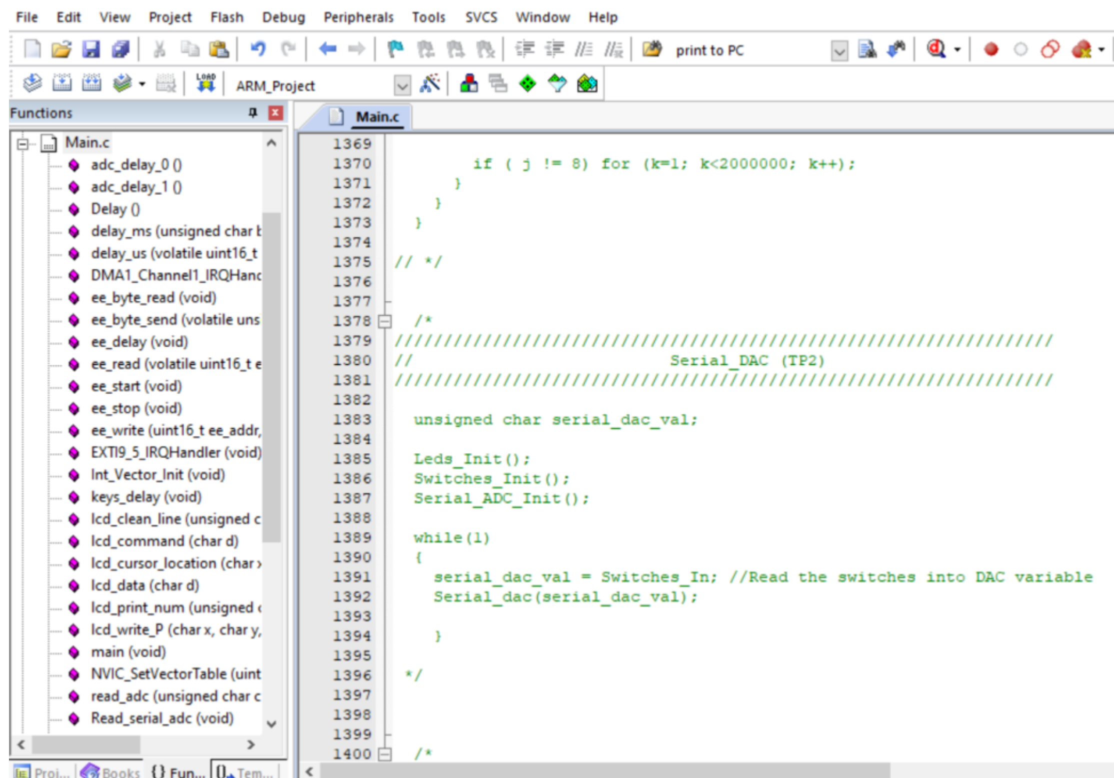
1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này, đường dẫn đến tệp ARM_Project.uvprojx là“C:\Courses\3192\S-ARM_V3\ARM_Project”

2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:



Hình 2. 67

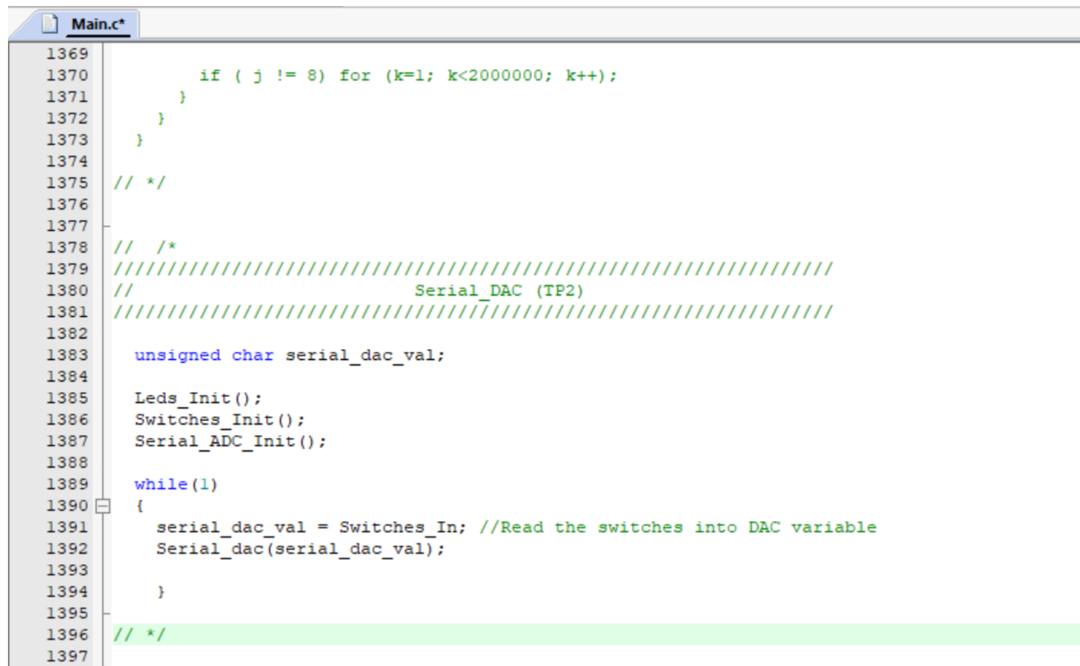
3. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:



Hình 2. 68

Phần này chứa chương trình **Serial DAC (TP2).**

4. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:



Hình 2. 69

5. Quan sát chương trình.

6. Kích hoạt EITPS-3192.

7. Lưu và biên dịch (sửa lỗi nếu có và biên dịch lại).

Nhấn RST trước khi tải xuống, sau đó tải xuống và chạy chương trình.

8. Chương trình đọc các công tắc và xuất nó đến DAC trong một vòng lặp.

9. Thiết lập tất cả các công tắc lên 1.

10. Kết nối dây màu đen với điểm kiểm tra TP33. Đây là điểm GND của mạch.

11. Kết nối dây màu đỏ với điểm kiểm tra TP2 (đầu ra DAC nối tiếp).

12. Dùng vôn kế để đo hiệu điện áp giữa hai đầu dây màu đỏ và đen.

Nó phải có giá trị khoảng 3,3V

13. Thay đổi công tắc và các điện áp giữa hai đầu.

Nó phải được thay đổi trong khoảng giữa 0V đến 3,3V.

14. Viết điện áp của các số sau:

0000 0000 =

0000 0001 =

0000 0010 =

0000 0100 =

0000 1000 =

0001 0000 =

0010 0000 =

0100 0000 =

1000 0000 =

1111 1111 =

15. Nhấn RST để dừng chương trình.

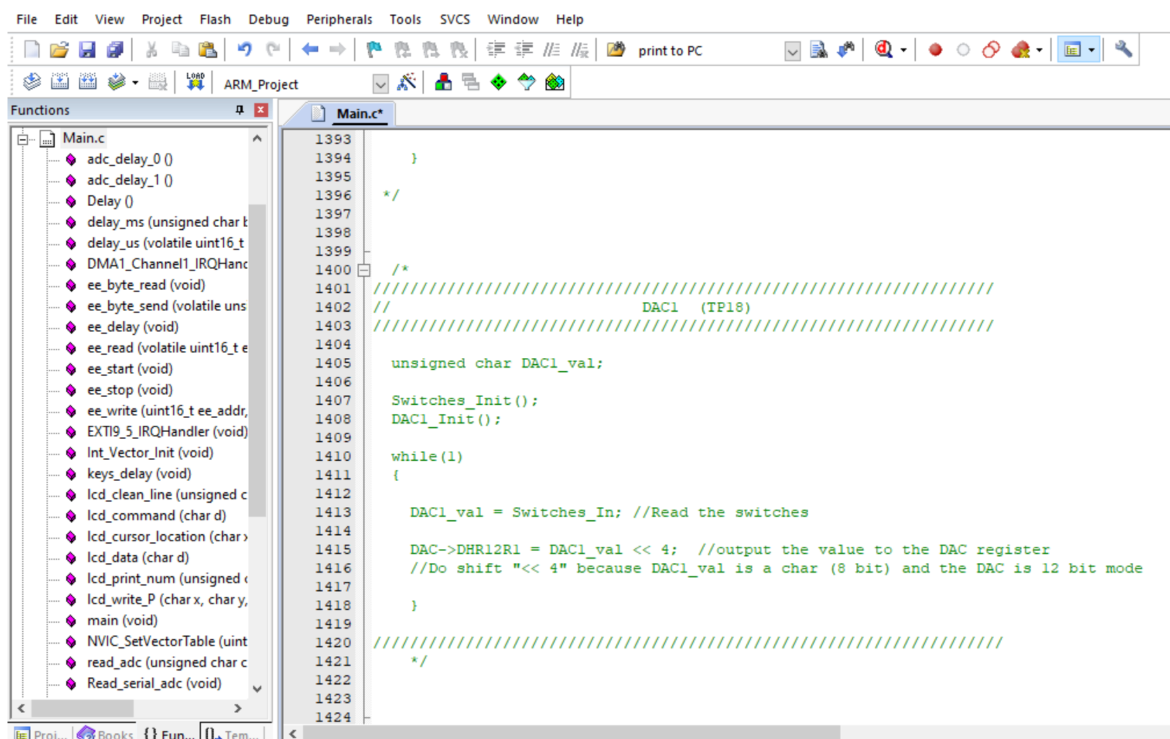
16. Sử dụng "Go to definition of" và nhập các hàm DAC thông qua các hàm lệnh của nó và cố gắng hiểu chúng.

17. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

18. Trong **ARM_Project** chỉ sau chương trình **Serial DAC (TP2)**, có một chương trình khác tên là **DAC1 (TP18)**. Chương trình này dành cho ARM được tích hợp trong DAC.

19. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:



Hình 2. 70

Phần này chứa chương trình **DAC1 (TP18).**

20. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:

```
1393
1394        }
1395
1396    */
1397
1398
1399
1400  //  /*
1401  ///////////////////////////////////////////////////////////////////////////
1402  //                              DAC1  (TP18)
1403  ///////////////////////////////////////////////////////////////////////////
1404
1405    unsigned char DAC1_val;
1406
1407    Switches_Init();
1408    DAC1_Init();
1409
1410    while(1)
1411    {
1412
1413      DAC1_val = Switches_In; //Read the switches
1414
1415      DAC->DHR12R1 = DAC1_val << 4;  //output the value to the DAC register
1416      //Do shift "<< 4" because DAC1_val is a char (8 bit) and the DAC is 12 bit mode
1417
1418      }
1419
1420  ///////////////////////////////////////////////////////////////////////////
1421  //    */
1422
```

Hình 2. 71

21. Quan sát chương trình.

Lưu và biên dịch (sửa lỗi nếu có và biên dịch lại).

Nhấn RST trước khi tải xuống, sau đó tải xuống và chạy chương trình.

22. Chương trình đọc các công tắc và xuất nó tới DAC1 trong một vòng lặp.

23. Đặt tất cả các công tắc lên 1.

24. Nối dây màu đen với điểm kiểm tra TP33. Đây là điểm GND của mạch.

25. Kết nối dây màu đỏ với điểm kiểm tra TP18 (đầu ra ARM DAC1).

26. Dùng vôn kế để đo điện áp giữa hai đầu.

Nó phải có giá trị khoảng 3,3V

27. Thay đổi công tắc và các điện áp giữa hai đầu.

Nó phải được thay đổi trong khoảng giữa 0V đến 3,3V.

28. Viết điện áp của các số sau:

0000 0000 =

0000 0001 =

0000 0010 =

0000 0100 =

0000 1000 =

0001 0000 =

0010 0000 =

0100 0000 =

1000 0000 =

1111 1111 =

29. Nhấn RST để dừng chương trình.

30. Sử dụng “Go to definition of” và nhập các hàm DAC thông qua các hàm lệnh của nó và cố gắng hiểu chúng.

31. Nhấn RST để dừng chương trình đang chạy.

32. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

Thử nghiệm 3.9 - Sử dụng ADC

Mục tiêu:

- ADC-0808 được kết nối với bus CPU như thế nào.
- Cách viết chương trình chuyển tín hiệu tương tự (điện áp) thành số nhị phân và xuất ra các đèn LED.

Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192
- Vôn kế

Thảo luận:

3.9.1. ADC 0808 - kênh ADC nguyên khối được ghép kênh

ADC 0808 là một trong những ADC được kết nối song song với bus dữ liệu vi điều khiển. Các thiết bị này chứa bên trong tất cả mạch chuyển đổi được mô tả trong đoạn trước với các phương thức chuyển đổi khác nhau, bao gồm cả bộ đệm đầu ra. Tất cả những gì chúng cần để hoạt động là các xung đồng hồ ở tần số thích hợp và một xung START để bắt đầu chuyển đổi. Khi kết thúc quá trình chuyển đổi, chúng xuất ra tín hiệu EOC. Việc truy cập kênh đầu ra dẫn đến việc xuất dữ liệu ra các đường đầu ra.

Ngày nay, người ta thường sử dụng các thiết bị phức tạp hơn ADC để xử lý một điện áp đầu vào đơn. Một thiết bị ghép nhiều đầu vào (một bộ ghép kênh) được sử dụng. Chúng ta sẽ mô tả một thiết bị ghép kênh tám ngõ vào tương tự. Ngoài các đường ADC thông thường, thiết bị này có ba đầu vào địa chỉ. Tám đầu vào tương tự khác nhau có thể được kết nối với thiết bị. Truy cập thành phần có địa chỉ kênh trong ba đường địa chỉ này, chuyển đổi kênh từ tương tự sang số.

ADC0808, được mô tả trong hình 3-28, là một thiết bị như vậy.

Hình 2. 72

3.9.2. ADC nối tiếp 8 bit ADC0838

Dòng ADC0831 là bộ chuyển đổi A/D ước lượng liên tiếp 8 bit với I/O nối tiếp và bộ ghép kênh đầu vào có thể được định cấu hình với tối đa 8 kênh. I/O nối tiếp được định cấu hình tuân theo tiêu chuẩn trao đổi dữ liệu nối tiếp NSC Microwire để dễ dàng giao tiếp với các vi xử lý họ COPS và có thể giao tiếp với các thanh ghi dịch tiêu chuẩn hoặc μPs. Bộ ghép kênh 2, 4 hoặc 8 kênh được định cấu hình cho các đầu vào một đầu hoặc đầu vào vi sai cũng như gán kênh.

Đầu vào điện áp tương tự vi sai cho phép loại bỏ chế độ chung và bù trừ giá trị điện áp đầu vào tương tự zero. Ngoài ra, đầu vào tham chiếu điện áp có thể được điều chỉnh để cho phép mã hóa bất kỳ khoảng điện áp tương tự nhỏ hơn nào thành độ phân giải 8 bit đầy đủ.

Thiết kế của các bộ chuyển đổi này sử dụng một cấu trúc so sánh dữ liệu mẫu cung cấp đầu vào tương tự vi sai được chuyển đổi theo đoạn chương trình ước lượng liên tiếp. Điện áp thực tế được chuyển đổi luôn là sự khác biệt giữa cổng đầu vào được gán "+" và cổng đầu vào "-". Cực của mỗi cổng đầu vào của cặp này được chuyển đổi cho biết đường nào mà bộ chuyển đổi ghi nhận là tích cực nhất. Nếu đầu vào "+" được gán nhỏ hơn đầu vào "-", bộ chuyển đổi sẽ phản hồi bằng một mã đầu ra tất cả logic “0”.

Một sơ đồ ghép kênh đầu vào duy nhất đã được sử dụng để cung cấp nhiều kênh tương tự với một đầu cuối vi sai có thể được định cấu hình phần mềm hoặc tùy chọn giả vi sai mới sẽ chuyển đổi sự khác nhau giữa điện áp tại bất kỳ đầu vào tương tự nào và đầu cuối chung nào. Việc điều chỉnh tín hiệu tương tự cần thiết trong hệ thống thu thập dữ liệu dựa trên bộ chuyển đổi được đơn giản hóa đáng kể với loại đầu vào linh hoạt này. Một gói bộ chuyển đổi hiện nay có thể xử lý các đầu vào tham chiếu thế điện áp và các đầu vào vi sai thực sự cũng như các tín hiệu với một số điện áp tham chiếu tùy ý.

Trước khi bắt đầu chuyển đổi, một cấu hình đầu vào cụ thể được gán trong chuỗi địa chỉ MUX. Địa chỉ MUX chọn đầu vào tương tự nào sẽ được hoạt động và quyết định đầu vào này là đầu cuối đơn hay đầu vào vi sai. Trong trường hợp vi sai, nó cũng gán cực cho các kênh. Các đầu vào vi sai bị hạn chế đối với các cặp kênh liền kề. Ví dụ: kênh 0 và kênh 1 có thể được chọn là một cặp vi sai nhưng chúng không thể hoạt động vi sai với bất kỳ kênh nào khác. Ngoài việc chọn chế độ vi sai, dấu cũng có thể được chọn. Kênh 0 có thể được chọn làm đầu vào dương và kênh 1 làm đầu vào âm hoặc ngược lại.

Địa chỉ MUX được chuyển vào bộ chuyển đổi qua đường DI. Bởi vì ADC0831 chỉ chứa một kênh đầu vào vi sai với phân cực cố định, nó không yêu cầu định địa chỉ.

Dòng đầu vào chung trên ADC0838 có thể được sử dụng làm đầu vào giả vi sai. Trong chế độ này, điện áp trên chân này được coi là đầu vào "-" cho bất kỳ kênh đầu vào nào khác. Điện áp này không nhất thiết phải là thế điện áp tương tự; nó có thể là bất kỳ điện thế tham chiếu nào chung cho tất cả các đầu vào. Tính năng này hữu ích nhất trong ứng dụng nguồn cung cấp đơn lẻ trong đó mạch tương tự có thể bị sai lệch về điện thế khác với thế điện áp và các tín hiệu đầu ra đều được quy về điện thế này.

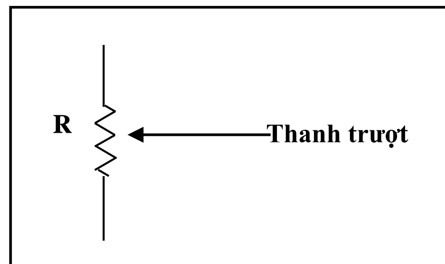
3.9.3. ADC ARM

Hầu hết các bộ vi điều khiển ngày nay đều có đầu vào ADC và ADC tích hợp sẵn. ARM mà chúng ta sử dụng cũng có các đầu vào như vậy. Chúng ta sẽ thực hành một

kênh ADC tích hợp. ADC này được đánh dấu là ADC1_IN6 và đầu vào của nó là TP19. Chúng ta sẽ thực hành với ADC này trong thí nghiệm tiếp theo.

3.9.4. Chiết áp

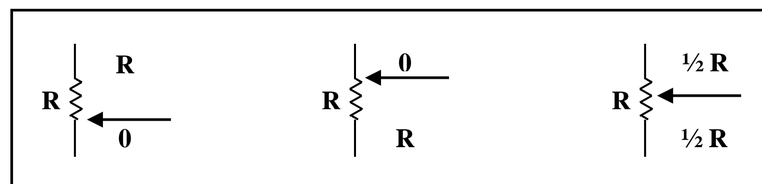
Đôi khi chúng ta cần một điện trở mà giá trị của nó thay đổi được trong số các thành phần tiêu chuẩn. Trong trường hợp này, chúng ta sử dụng biến trở, hoặc **chiết áp**. Thông thường, chiết áp có ba dây dẫn và một núm điều chỉnh điện trở của nó. Hình sau cho thấy kết nối bên trong của chiết áp:



Hình 2. 73

Dây dẫn ở giữa được gọi là thanh trượt. Điện trở thường được đo giữa dây dẫn thanh trượt và một trong các dây dẫn khác. Rõ ràng là điện trở giữa các dây dẫn bên ngoài là không đổi và không phụ thuộc vào vị trí của dây dẫn thanh trượt.

Mặt khác, điện trở giữa dây dẫn thanh trượt và một trong các dây dẫn bên ngoài có thể được điều chỉnh bằng cách xoay núm (hoặc dịch chuyển tay cầm trong một số chiết áp):



Hình 2. 74

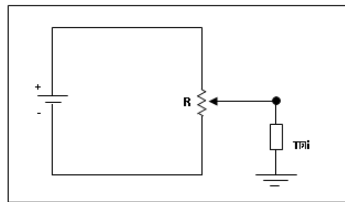
Dễ dàng nhận thấy rằng hai điện trở đo giữa dây dẫn thanh trượt và hai dây dẫn ngoài cùng là bù nhau. Điều này có nghĩa là tổng của chúng luôn bằng R. Khi các dây dẫn bên ngoài được kết nối với một nguồn điện áp, điện áp trên dây dẫn thanh trượt tỷ lệ với vị trí tiếp xúc của nó bên trong chiết áp, do đó ta sử dụng bộ chia điện áp thay đổi.

Có thể thu được biến trở bằng cách sử dụng một chiết áp có một trong các dây dẫn bên ngoài bị ngắt kết nối. Giá trị của chiết áp phải không nhỏ điện trở tối đa được yêu cầu.

Một số chiết áp được chế tạo để truyền công suất biến đổi tới tải. Chúng thường có trở kháng thấp và được gọi là **biến trở** (rheostat).

Chiết áp thường có điện trở lớn hơn và được làm bằng than; các biến trở có điện trở nhỏ và được làm bằng thép.

Mạch sau đây đưa ra ví dụ về việc sử dụng một biến trở để đặt điện áp thay đổi cho tải:



Hình 2. 75

Trình tự thực hiện:

1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này, đường dẫn đến tệp ARM_Project.uvprojx là“C:\Courses\3192\S-ARM_V3\ARM_Project”

2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:

```
850   }
851
852   Delay()
853   {
854     int i;
855     for (i = 1 ; i != 0X100000 ; i++);
856   }
857
858
859   int main (void)
860   {
861
862    /*
863   ///////////////////////////////////////////////////////////////////////
864   //                       Switches to LEDs
865   ///////////////////////////////////////////////////////////////////////
866
867     unsigned int temp;
868
869     Leds_Init();
870     Switches_Init();
871
872     while(1)
873     {
874       temp = Switches_In; //Read the switches
875       LEDs_Out = temp; //Output to LEDs
876     }
877
878    */
879
880
881    /*
```

Hình 2. 76

3. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:

```
1423
1424
1425   /*
1426  //////////////////////////////////////////////////////////////////////
1427  //               Serial_ADC (TP3)  +  LCD
1428  //////////////////////////////////////////////////////////////////////
1429
1430
1431    LCD_Init();
1432    SYSTICK_Init();             //Core Tick counter
1433    Serial_ADC_Init();
1434
1435    while(1)
1436    {
1437
1438      lcd_clean_line(1); //clean line number 1
1439      lcd_clean_line(2); //clean line number 2
1440
1441      lcd_write_P ( 4, 2, ("ADC = ")  );  //x=4 (Fourth label)   y=2 (line number 2)
1442
1443
1444      num_to_lcd = Read_serial_adc();  //Read TP3
1445      lcd_print_num ( 10, 2, num_to_lcd ); //x=10 (Tenth label)   y=2 (line number 2)
1446
1447      delay_ms(255);
1448    }
1449
1450  //////////////////////////////////////////////////////////////////////
1451   */
1452
1453
1454   /*
```

Hình 2. 77

Phần này chứa chương trình **Serial ADC (TP3) + LCD**.

4. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:

```
Main.c*
1423
1424
1425  //  /*
1426  ////////////////////////////////////////////////////////////////////////
1427  //                  Serial_ADC (TP3)   +  LCD
1428  ////////////////////////////////////////////////////////////////////////
1429
1430
1431    LCD_Init();
1432    SYSTICK_Init();              //Core Tick counter
1433    Serial_ADC_Init();
1434
1435    while(1)
1436    {
1437
1438      lcd_clean_line(1); //clean line number 1
1439      lcd_clean_line(2); //clean line number 2
1440
1441      lcd_write_P ( 4, 2, ("ADC = ")  );  //x=4 (Fourth label)   y=2 (line number 2)
1442
1443
1444      num_to_lcd = Read_serial_adc();  //Read TP3
1445      lcd_print_num ( 10, 2, num_to_lcd ); //x=10 (Tenth label)   y=2 (line number 2)
1446
1447      delay_ms(255);
1448    }
1449
1450  ////////////////////////////////////////////////////////////////////////
1451  //  */
1452
```

Hình 2. 78

5. Quan sát chương trình và phân tích nó.

6. Kích hoạt EITPS-3192.

7. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

8. Một chiết áp được kết nối với +3.3V và GND.

Kết nối chân TP4 của chiết áp với đầu vào TP3 của ADC.

9. Đồng thời kết nối hai dây với GND (TP33) và với chiết áp (TP4).

10. Chương trình đọc ADC và in số trên màn hình LCD trong một vòng lặp.

11. Thay đổi chiết áp, quan sát số trên màn hình LCD và đo điện áp giữa hai đầu dây.

12. Viết các số cho các điện áp sau:

0.2 =

0.4 =

0.8 =

1.6 =

3.0 =

13. Nhấn RST để dừng chương trình đang chạy.

14. Viết một chương trình thực thi như sau:

a) Chuyển đổi điện áp từ đầu vào của ADC (TP3) thành số nhị phân.

b) Xuất số nhị phân tới các đèn LED được kết nối với cổng đầu ra.

c) Chuyển đến đoạn (a).

15. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

16. Thay đổi chiết áp.

Đo điện áp đầu vào ADC và so sánh nó với số nhị phân mà bạn đã thu được.

17. Nhấn RST để dừng chương trình đang chạy.

18. Cải thiện chương trình (cũng gửi số đến DAC).

19. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

20. Thay đổi chiết áp và so sánh điện áp ở đầu vào ADC với điện áp ở đầu ra DAC.

21. Nhấn RST để dừng chương trình đang chạy.

22. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

Thí nghiệm 3.10 – ADC tích hợp, điện trở nhiệt và bộ ghép Quang

Mục tiêu:

- ADC tích hợp sẵn trong ARM.
- Điện trở nhiệt (thermistor) làm một cảm biến nhiệt độ.
- Một đèn LED và một transistor quang (phototransistor) làm bộ ghép quang.

Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192
- Dây cắm thí nghiệm

Thảo luận:

3.10.1. ADC ARM

Hầu hết các bộ vi điều khiển ngày nay đều có đầu vào ADC và ADC tích hợp sẵn. ARM mà chúng ta sử dụng cũng có các đầu vào như vậy. Chúng ta sẽ thực hành một kênh ADC tích hợp. ADC này được đánh dấu là ADC1_IN6 và đầu vào của nó là TP19. Chúng ta sẽ thực hành với ADC này trong thí nghiệm tiếp theo.

3.10.2. Điện trở nhiệt

Điện trở nhiệt là một điện trở, giá trị điện trở của nó thay đổi theo nhiệt độ. Mọi vật chất trong tự nhiên đều có các electron chuyển động xung quanh hạt nhân của nguyên tử. Trong chất dẫn điện, một số electron là **electron tự do**, có thể di chuyển từ nguyên tử này sang nguyên tử khác và thay đổi vị trí với các electron tự do khác. Càng nhiều electron tự do hơn trong vật chất sẽ làm cho nó dẫn điện tốt hơn và ít điện trở hơn.

Làm nóng một điện trở thực tế là truyền năng lượng cho vật dẫn. Electron chính là phần tử nhận năng lượng này và khuếch đại vận tốc của nó.

Một điện trở nhiệt là một điện trở nhạy cảm với nhiệt. Điện trở nhiệt hoạt động trong phạm vi nhiệt độ từ -50 đến 300°C, điều này khiến cho chúng hữu ích trong nhiều ứng dụng khác nhau.

Các điện trở nhiệt có thể được sản xuất với đường kính nhỏ chỉ khoảng 0,5mm (0,02 inch) với công suất nhiệt rất nhỏ và thời gian phản hồi nhanh. Để đo sự thay đổi điện trở của Thermistor, một dòng điện phải chạy qua nó để tạo ra giá trị điện áp tỷ lệ với nhiệt độ. Điện trở nhiệt trong bộ thí nghiệm EITPS-3192 là loại NTC (Negative Temperature Coefficient - Hệ số nhiệt độ âm). Đó là một Thermistor hệ số nhiệt độ âm. Khi nhiệt độ tăng, điện trở của nó giảm.

Trong một số điện trở, nhiều electron được giải phóng khỏi nguyên tử khi chúng nóng lên và điện trở của chúng giảm. Các điện trở này được gọi là điện trở Hệ số nhiệt độ âm hoặc NTC.

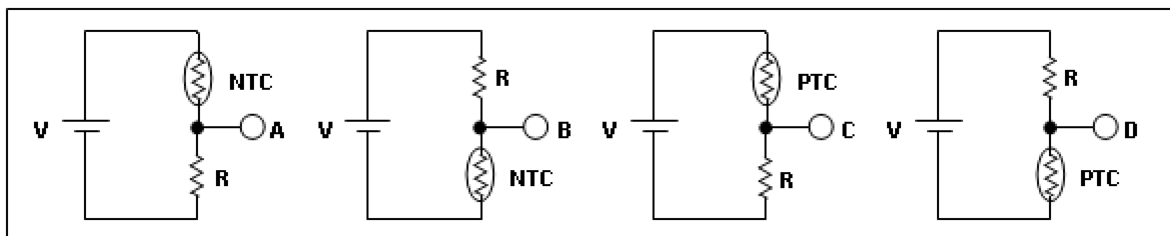
Có một số điện trở mà nhiệt độ ảnh hưởng nghịch lên chúng. Khi chúng bị đốt nóng, điện trở lại tăng lên. Loại điện trở này được gọi là PTC (Positive Temperature Coefficient - Hệ số nhiệt độ dương).

Cả hai cái tên trên đều xuất phát từ một công thức mô tả giá trị của điện trở phụ thuộc vào nhiệt độ:

$$\mathbf{R = R_o + C(T - T_o)}$$

Trong công thức này R là giá trị điện trở ở nhiệt độ cho trước T, R_o là điện trở ở nhiệt độ riêng T_o, thường là nhiệt độ phòng: +20°C. C là hệ số nhiệt độ và nó có giá trị dương trong PTC và âm trong NTC.

Để mắc một công tắc theo nhiệt độ, ta mắc nối tiếp điện trở nhiệt với một điện trở khác theo một trong các cách sau:



Hình 2. 79

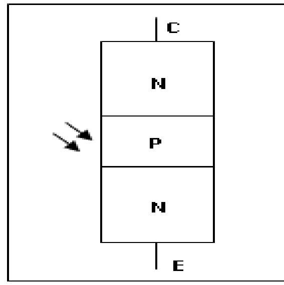
Điện trở R thích nghi với giá trị điện trở nhiệt, tùy thuộc vào điện áp mà chúng ta muốn nhận được tương ứng với nhiệt độ của phòng.

Khi nhiệt độ tăng, điện áp V_A và V_D cũng sẽ tăng và khi nhiệt độ giảm, điện áp V_B và V_C cũng sẽ giảm.

Thay vì sử dụng điện trở R không đổi, chúng ta có thể sử dụng một chiết áp để xác định điện áp mong muốn tương ứng với nhiệt độ của phòng.

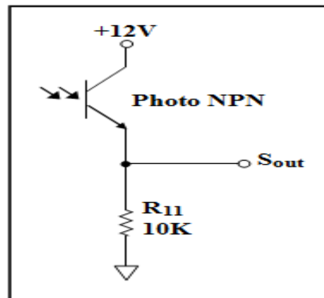
3.10.3. Bóng bán dẫn quang (Transistor quang – Phototransistor)

Bộ thu quang học dựa trên một transistor quang. Transistor quang là một transistor NPN không có chân đế. Thay vào đó, linh kiện này nằm trong một bọc nhựa trong suốt, cho phép ánh sáng đi đến chân đế của nó.



Hình 2. 80

Mạch chuyển đổi rất đơn giản. Tất cả những gì bạn phải làm là kết nối một điện trở (mắc nối tiếp) với linh kiện. Điện áp trên điện trở chuyển đổi, chính là hàm của ánh sáng trên cảm biến.

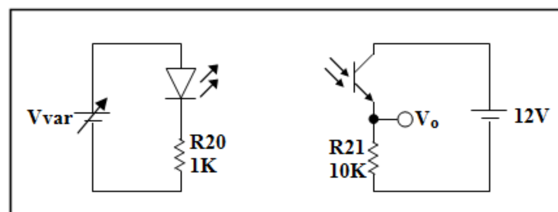


Hình 2. 81

Có những transistor quang phản ứng với một dải ánh sáng đặc biệt, như ánh sáng có thể nhìn thấy hoặc ánh sáng IR (Hồng ngoại).

3.10.4. Bộ ghép OPTO

Một bộ ghép OPTO bao gồm một đèn LED và một transistor quang được đặt đối diện với nhau như thể hiện trong mạch sau.



Hình 2. 82

Thay đổi điện áp trong mạch LED ảnh hưởng đến ánh sáng của đèn LED và điện áp đầu ra trong mạch transistor quang.

Bằng cách này, chúng ta có thể chuyển các tín hiệu điện trong các mạch không được kết nối điện.

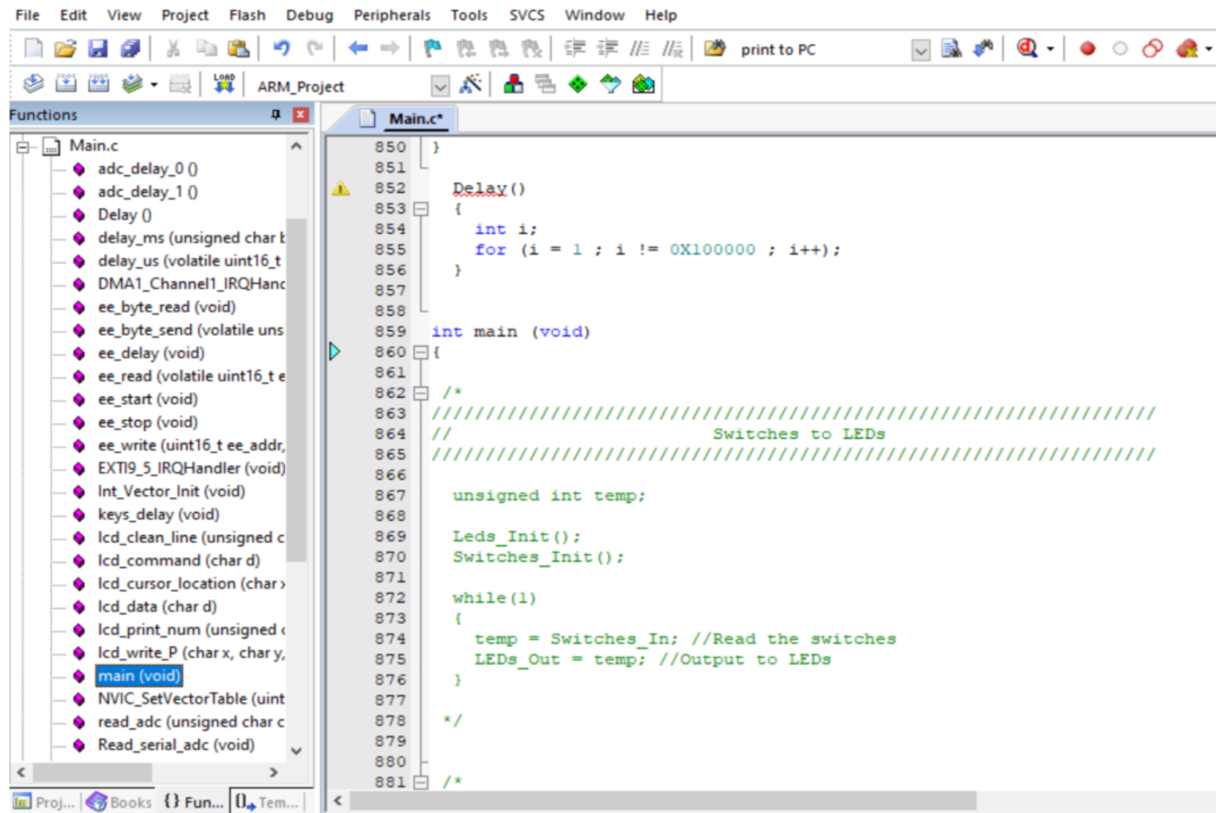
Bộ ghép OPTO có nhiều ứng dụng và đi kèm với nhiều biến thể. Trong hầu hết các trường hợp, ánh sáng hồng ngoại được sử dụng để loại bỏ ảnh hưởng của ánh sáng môi trường.

Bộ ghép OPTO trong bộ thí nghiệm EITPS-3192 dựa trên một đèn LED màu đỏ và một transistor quang phản ứng với ánh sáng IR và ánh sáng đỏ. LED và transistor quang được đặt đối diện nhau. Bộ ghép OPTO này được sử dụng để xác định một vật thể di chuyển qua môi trường giữa chúng. Nó cũng được sử dụng để truyền tín hiệu bằng ánh sáng chứ không phải bằng kết nối dây.

Trình tự thực hiện:

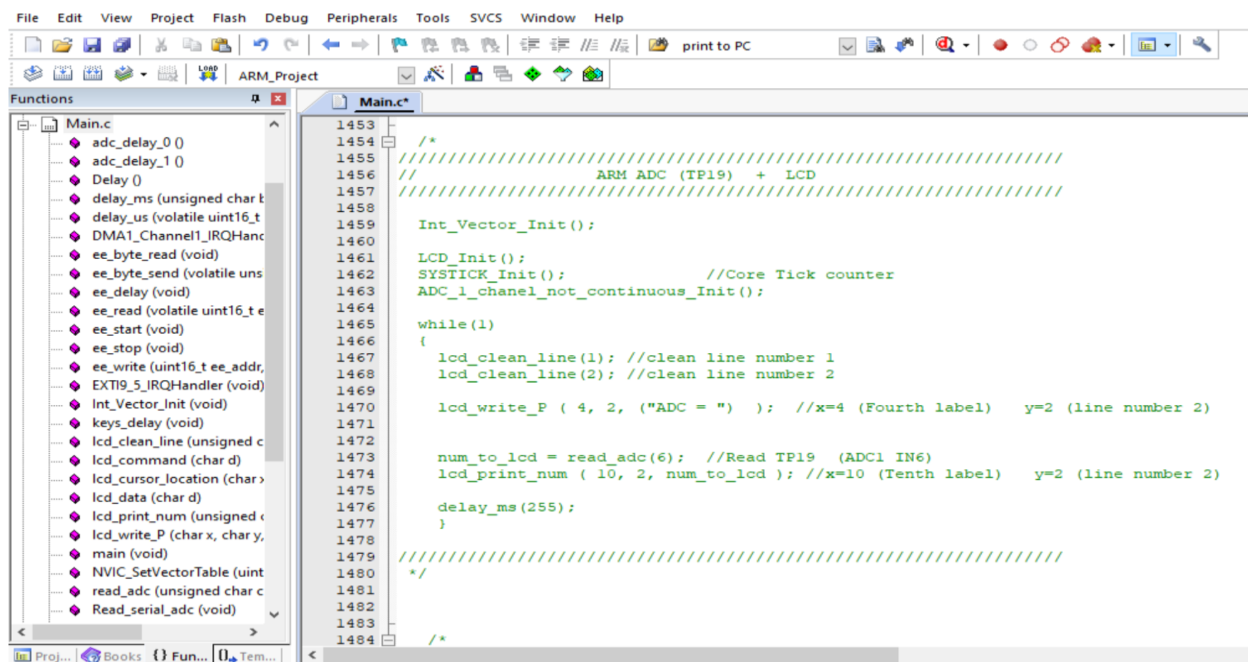
1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này, đường dẫn đến tệp ARM_Project.uvprojx là"C:\Courses\3192\S-ARM_V3\ARM_Project"

2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:



Hình 2. 83

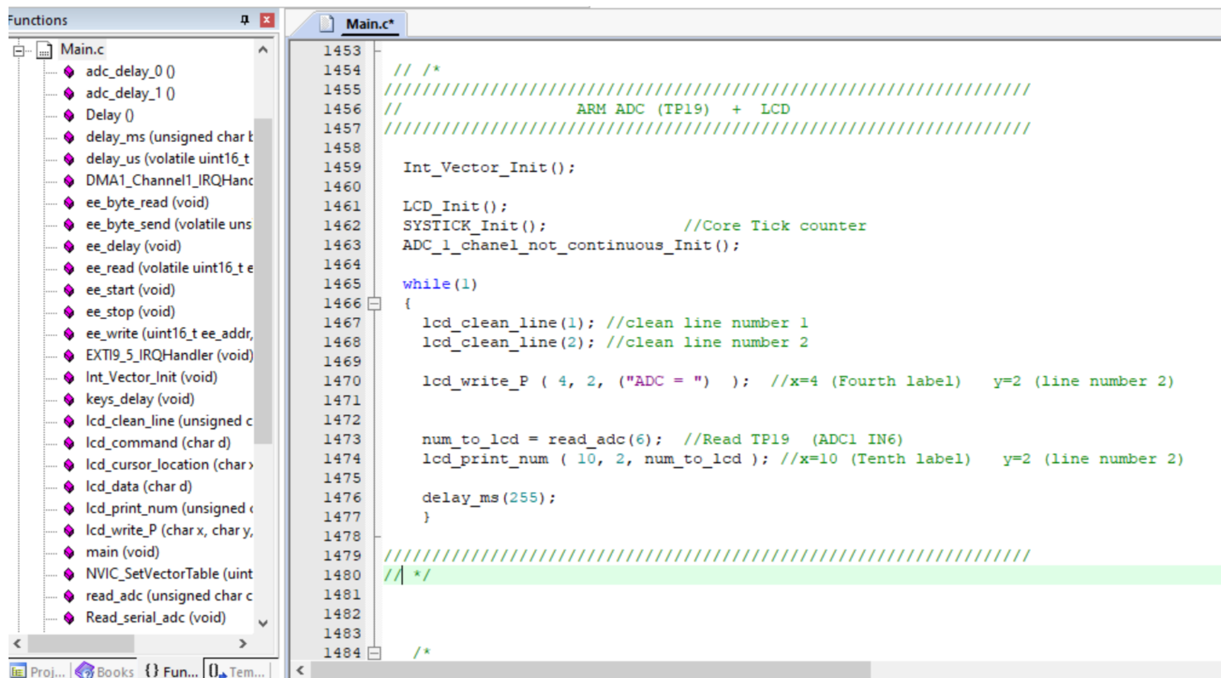
3. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:



Hình 2. 84

Phần này chứa chương trình **ARM ADC (TP19) + LCD**.

4. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:



Hình 2. 85

5. Quan sát chương trình và phân tích nó.

6. Kích hoạt EITPS-3192.

7. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

8. Một chiết áp được kết nối với +3.3V và GND.

Kết nối chân chiết áp (TP4) với đầu vào ADC (TP19).

9. Đồng thời kết nối hai dây cắm với GND (TP33) và chiết áp (TP4).

10. Chương trình đọc ADC và in số trên màn hình LCD trong một vòng lặp.

11. Thay đổi chiết áp, quan sát số trên màn hình LCD và đo điện áp giữa hai đầu dây cắm.

12. Viết các số cho các điện áp sau:

0.2 =

0.4 =

0.8 =

1.6 =

3.0 =

13. Ngắt kết nối dây khỏi TP19 và TP4 và kết nối dây tới TP19 và TP29 (đầu ra mạch NTC).

14. Chạm vào NTC bằng ngón tay của bạn và quan sát nó ảnh hưởng như thế nào đến các số trên màn hình LCD. Phản ứng là khá nhanh.

15. Ngắt kết nối dây cắm khỏi TP19 và TP29, và kết nối dây tới TP19 và TP38 (đầu ra mạch transistor quang).

16. Dùng ngón tay che transistor quang và quan sát xem nó ảnh hưởng như thế nào đến các con số trên màn hình LCD.

17. Nối công tắc hành trình (TP22) với đèn LED màu đỏ (TP32) ở bên trái bằng dây cắm và dây nối dài với.

18. Nhấn công tắc hành trình và đèn LED màu đỏ sẽ bật. Quan sát nó ảnh hưởng như thế nào đến các số trên màn hình LCD.

19. Nhấn RST để dừng chương trình đang chạy.

21. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

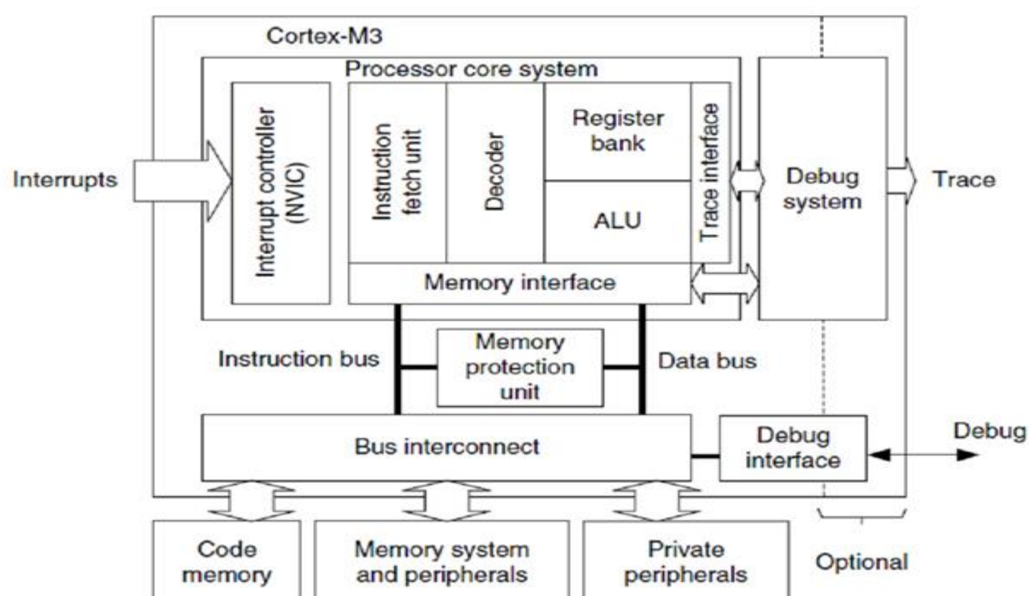
CHƯƠNG 4 - ARM CORTEX M3

4.1. Các thanh ghi

Cortex-M3 là bộ vi xử lý 32 bit. Nó có một đường dẫn dữ liệu 32 bit, một thanh ghi 32 bit và các giao diện bộ nhớ 32 bit (xem hình sau).

Bộ xử lý có cấu trúc Harvard, nghĩa là nó có một bus lệnh và một bus dữ liệu riêng biệt. Điều này cho phép các lệnh và các truy cập dữ liệu diễn ra cùng một lúc, và do đó, hiệu suất của bộ xử lý tăng lên bởi vì các truy cập dữ liệu không ảnh hưởng đến cấu trúc lệnh.

Tính năng này dẫn đến nhiều giao diện bus trên Cortex-M3, mỗi giao diện đều có cách sử dụng được tối ưu và khả năng được sử dụng đồng thời. Tuy nhiên, các bus lệnh và dữ liệu chia sẻ cùng một không gian bộ nhớ (một hệ thống bộ nhớ thống nhất). Nói cách khác, bạn không thể nhận được 8GB dung lượng bộ nhớ chỉ vì bạn có các giao diện bus riêng biệt.



Hình 2. 86

Đối với các ứng dụng phức tạp đòi hỏi nhiều tính năng hệ thống bộ nhớ hơn, bộ xử lý Cortex-M3 có Bộ bảo vệ bộ nhớ (Memory Protection Unit - MPU) tùy chọn và ta có thể sử dụng bộ nhớ đệm ngoài nếu cần. Cả hai hệ thống bộ nhớ little-endian và big-endian đều được hỗ trợ.

Bộ xử lý Cortex-M3 bao gồm một số thành phần gỡ lỗi nội bộ cố định. Các thành phần này cung cấp các tính năng và hỗ trợ thao tác gỡ lỗi, chẳng hạn như các điểm ngắt (breakpoint) và điểm theo dõi (watch point).

ARM là một cấu trúc 32-bit. Các thuật ngữ sau đây khi được sử dụng liên quan đến ARM là:

- **Byte** có nghĩa là 8 bit.
- **Half word** có nghĩa là 16 bit (2 byte).
- **Word** có nghĩa là 32 bit (4 byte)

Hầu hết các ARM đều triển khai ba tập lệnh:

- Tập lệnh ARM 32-bit.
- Tập lệnh Thumb 16-bit.
- Tập lệnh Jazelle 8-bit của ngôn ngữ Java.

CPU phải được thông báo theo tập lệnh mà chương trình sử dụng.

Tập lệnh Thumb có thể được chọn trực tiếp khi chương trình được viết bằng hợp ngữ. Trong ngôn ngữ C, việc chọn được thực hiện trong các tùy chọn trình biên dịch.

Như đã giải thích trong chương 1, chương trình bộ xử lý là một dãy số nhị phân (các mã lệnh) nằm trong bộ nhớ. CPU tìm nạp hết mã này đến mã khác, phân tích và thực thi nó. Sau đó tiếp tục đến số tiếp theo. Các mã ARM có thể có kích thước 2 byte (trong Tập lệnh Thumb) hoặc 4 byte.

CPU dừng chương trình đang chạy khi xảy ra ngắt. Ngắt được tạo ra khi một dòng điều khiển nhất định của CPU được nâng lên hoặc một thành phần bên trong (bộ định thời, I / O, thiết bị ngoại vi) đưa ra một yêu cầu.

Trong trường hợp này, CPU phải lưu trạng thái của chương trình đang chạy và bắt đầu chạy một chương trình thay thế theo yêu cầu ngắt và mức độ ưu tiên của nó.

Có nhiều kiểu ngắt chương trình khác nhau, được gọi là exception. Chúng ta sẽ trở lại với chúng trong phần tiếp theo.

CPU có thể truy cập các thanh ghi khi đang chạy trên một chương trình. Thanh ghi là một ô bộ nhớ trong của CPU có độ dài 32 bit. Các lệnh chương trình sử dụng các thanh ghi này làm nội dung tạm thời. Bộ xử lý Cortex-M3 có các thanh ghi R0 đến R15. Thanh ghi R13 (con trỏ ngăn xếp - stack pointer) được phân chia, mà chỉ có một bản sao của R13 được hiển thị tại một thời điểm.

Name		Functions (and banked registers)	
R0		General-purpose register	Low registers
R1		General-purpose register	
R2		General-purpose register	
R3		General-purpose register	
R4		General-purpose register	
R5		General-purpose register	
R6		General-purpose register	
R7		General-purpose register	
R8		General-purpose register	High registers
R9		General-purpose register	
R10		General-purpose register	
R11		General-purpose register	
R12		General-purpose register	
R13 (MSP)	R13 (PSP)	Main Stack Pointer (MSP), Process Stack Pointer (PSP)	
R14		Link Register (LR)	
R15		Program Counter (PC)	

Hình 2. 87

R0-R12 - Thanh ghi đa năng (General-purpose):

R0 – R12 là các thanh ghi đa năng 32 bit cho các phép toán dữ liệu. Một số lệnh Thumb 16-bit chỉ có thể truy cập một tập hợp con của các thanh ghi này (các thanh ghi thấp, R0 – R7).

R13 - Con trỏ ngăn xếp:

Thanh ghi R13 là con trỏ ngăn xếp, còn được gọi là SP. Nội dung của nó trỏ đến một địa chỉ nhất định trong bộ nhớ nơi nội dung của thanh ghi khác có thể được lưu tạm thời. Vùng bộ nhớ này được gọi là ngăn xếp (STACK) và dữ liệu được "đẩy" và "xuất" từ nó.

Khi nội dung thanh ghi được đẩy vào ngăn xếp, nó sẽ được đẩy theo từng byte đến địa chỉ mà SP trỏ đến và thanh ghi SP giảm đi một tại mỗi thời điểm, để trỏ đến địa chỉ tiếp theo cho nội dung mới được đẩy.

Khi nội dung được xuất từ ngăn xếp, thanh ghi SP tăng lên một bốn lần và nội dung ô nhớ được xuất tại mỗi thời điểm.

Cortex-M3 chứa hai con trỏ ngăn xếp (R13). Chúng được phân chia sao cho chỉ có một con trỏ được hiển thị tại một thời điểm.

Hai con trỏ ngăn xếp được hiểu như sau:

- **Main Stack Pointer (MSP)** - con trỏ ngăn xếp mặc định, được sử dụng bởi nhân hệ điều hành (OS) và các trình xử lý ngoại lệ (exception handler).
- **Process Stack Pointer (PSP)** - Được sử dụng bởi mã ứng dụng của người dùng.

2 bit thấp nhất của con trỏ ngăn xếp luôn là '0', điều đó có nghĩa là chúng luôn được căn chỉnh thẳng hàng word 32 bit.

Xử lý ngắt và ngoại lệ có thể được thực hiện trong hệ thống mà không liên quan đến chương trình của người dùng. Chúng có chương trình hỗ trợ riêng. Hệ thống sử dụng ngăn xếp để lưu các giá trị của thanh ghi mà nó sử dụng khi chuyển sang đoạn chương

trình ngắt để không gây hại cho chương trình của người dùng. Việc sử dụng hai con trỏ ngăn xếp giúp ngăn chặn việc trộn lẫn giữa dữ liệu ngăn xếp được lưu bởi chương trình của người dùng và dữ liệu ngăn xếp được hệ thống lưu.

R14 – Thanh ghi liên kết:

Thanh ghi R14 là thanh ghi nơi CPU lưu nội dung của bộ đếm chương trình (R15) trong các lệnh BL (Branch with Link). Thanh ghi này còn được gọi là thanh ghi liên kết (LR).

Khi một hàm được gọi, địa chỉ trả về được lưu trữ trong thanh ghi liên kết.

R15 – Bộ đếm chương trình:

Thanh ghi R15 là bộ đếm chương trình, còn được gọi là PC. PC luôn trỏ đến địa chỉ của lệnh tiếp theo sẽ được tìm nạp và thực thi. Khi một lệnh được tìm nạp, PC sẽ tăng thêm bốn hoặc hai tùy theo tập lệnh được sử dụng. Trong tập lệnh Jazelle (các lệnh byte đơn), bốn lệnh được đọc cùng một lúc.

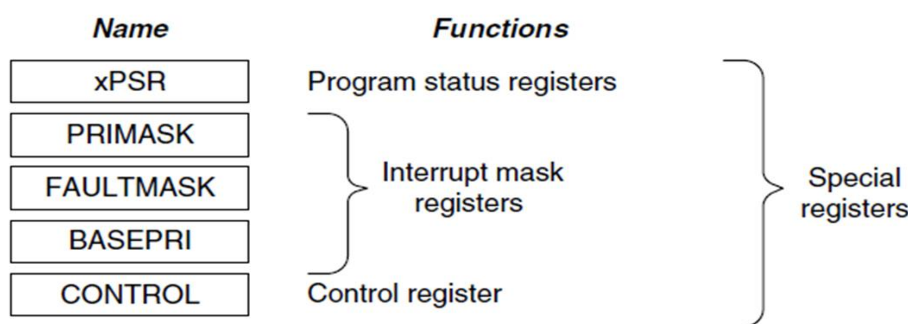
CPU có hai loại lệnh rẽ nhánh. Một là rẽ nhánh mà không lưu địa chỉ trả về. Hai là lưu địa chỉ trả về (giống như các lệnh "call" trong các bộ xử lý khác). Lệnh rẽ nhánh này được gọi là "Rẽ nhánh có Liên kết" - BL (Branch with Link).

Bộ đếm chương trình là địa chỉ chương trình hiện tại. Thanh ghi này có thể được ghi khi phải thay đổi luồng chương trình.

4.2. Các thanh ghi đặc biệt

Bộ xử lý Cortex-M3 cũng có một số thanh ghi đặc biệt. Chúng là các thanh ghi như sau:

- Các thanh ghi trạng thái chương trình (PSR)
- Các thanh ghi mặt nạ ngắt - Interrupt Mask register (gồm PRIMASK, FAULTMASK và BASEPRI)
- Thanh ghi điều khiển (CONTROL)



Hình 2. 88

Các thanh ghi này có các chức năng đặc biệt và chỉ có thể được truy cập bằng các lệnh đặc biệt. Chúng không thể được sử dụng để xử lý dữ liệu thông thường.

CPU sử dụng một số bit PSR làm các lệnh điều kiện và các bit trạng thái. Các bit điều kiện giúp CPU thực hiện các lệnh điều kiện. Các bit trạng thái cho CPU biết tập lệnh nào (ARM, Thumb hoặc Jazelle) được sử dụng, chế độ bộ xử lý hiện tại là gì và khi có một yêu cầu ngắt được kích hoạt hoặc vô hiệu hóa.

Các thanh ghi mặt nạ ngắt được sử dụng để kích hoạt hoặc vô hiệu hóa CPU để phục vụ các yêu cầu ngắt.

Thanh ghi CONTROL xác định trạng thái ưu tiên và lựa chọn con trỏ ngăn xếp, nó sẽ được giải thích trong phần sau.

4.3. Chế độ bộ xử lý

Như được mô tả trong phần 1.7, một chương trình bộ xử lý được cấu tạo từ một tập hợp các số nhị phân mà bộ xử lý đọc lần lượt, sau đó giải mã chúng dưới dạng các lệnh thực thi và thực thi chúng.

Một số lệnh của bộ xử lý là lệnh rẽ nhánh thay đổi thứ tự tiến trình chương trình. Trong số các lệnh rẽ nhánh có các lệnh rẽ nhánh có các lệnh rẽ nhánh có điều kiện, trong đó việc phân nhánh được thực hiện theo sự tồn tại hoặc không tồn tại của một điều kiện được chỉ ra trong lệnh.

Bộ xử lý được yêu cầu thực thi một số phân đoạn trong chương trình nhiều lần. Thay vì viết lại chúng ở các vị trí khác nhau trong chương trình, phân đoạn được viết dưới dạng một hàm. Tại những nơi cần đến phân đoạn chương trình này, một lệnh rẽ nhánh trả về (gọi một hàm) sẽ được viết.

Loại chương trình này được gọi là một chuỗi. Một hệ thống với các bộ xử lý tiên tiến như ARM, một số chuỗi có thể được lưu trữ trong bộ nhớ của máy tính và khiến bộ xử lý chạy một chương trình khác nhau cho mỗi lần, do đó có vẻ như bộ xử lý đang thực hiện nhiều chương trình cùng một lúc.

Sự khác biệt giữa việc thực thi nhiều chuỗi so với việc gọi các hàm khác nhau là mỗi chuỗi là một đơn vị thực thi độc lập bao gồm ngăn xếp và các thanh ghi của riêng nó.

Trong mọi chương trình đều có một địa chỉ đến các đơn vị I/O. Ví dụ: lấy mẫu tín hiệu từ bên ngoài đến máy tính, định địa chỉ bàn phím, gửi dữ liệu đến máy in, giao tiếp, định địa chỉ bộ nhớ ngoài, hệ điều hành, v.v. Loại chương trình này chạy trong một vòng lặp.

Ngay cả việc gửi dữ liệu đến đơn vị đầu ra hoặc đơn vị giao tiếp chỉ được thực hiện sau khi kiểm tra rằng đơn vị đầu ra hoặc đơn vị giao tiếp đã sẵn sàng để nhận dữ liệu. Có những tình huống trong đó chương trình đợi một dữ liệu hoặc đợi một đơn vị có sẵn để nhận dữ liệu.

Phương pháp đơn giản nhất để định địa chỉ các đơn vị I/O là thông qua kiểm soát vòng (polling). Việc định địa chỉ các lệnh của các đơn vị I/O được kết hợp bên trong chuỗi. Các lệnh định địa chỉ này được thực thi ngay cả khi không có dữ liệu nào trong đầu vào để chuyển đi, và khi đơn vị đầu ra không được chuẩn bị để nhận nó.

Một phương pháp khác là cho phép các đơn vị I/O báo hiệu cho bộ xử lý mỗi khi chúng sẵn sàng chuyển hoặc nhận dữ liệu. Chỉ báo này được thực hiện bằng cách sử dụng các đường đặc biệt được gọi là đường ngắt bên ngoài. Khi đường ngắt này được kích hoạt, bộ xử lý ngừng chạy chương trình, lưu vị trí của nó trong chương trình và thực thi một chương trình dịch vụ xử lý dữ liệu.

Trong cả hai phương pháp (kiểm soát vòng và ngắt), dữ liệu đều đi qua bộ xử lý. Có một tùy chọn thứ ba yêu cầu cần có các đơn vị I/O phức tạp. Sau khi vận hành các đơn vị này, chúng trực tiếp truyền dữ liệu vào bộ nhớ hoặc lấy dữ liệu từ bộ nhớ mà không cần qua trung gian của máy tính. Chúng sử dụng các khoảng thời gian khi bộ xử lý

không định địa chỉ bộ nhớ hoặc chỉ báo cho bộ xử lý biết rằng chúng cần truy cập trực tiếp. Cách tiếp cận truy cập này được gọi là DMA - Truy cập bộ nhớ trực tiếp.

Thông thường, các hệ thống máy tính cũng bao gồm một bộ đếm thời gian tạo yêu cầu ngắt trong mỗi khoảng thời gian xác định (mỗi vài hoặc chục phần nghìn giây). Chương trình ngắt có thể thực hiện đếm bộ đếm thời gian, các giá trị của bộ đếm sẽ được lưu trong các ô nhớ nhất định. Bằng cách này, hệ thống có một bộ đếm thời gian thực hoạt động ở chế độ nền.

Một chương trình hẹn giờ có thể thực hiện định địa chỉ I/O, và do đó để cải thiện phương pháp kiểm soát vòng thành hoạt động ở chế độ nền chứ không phải trong chương trình chính.

Trong trường hợp có nhiều chương trình được dự định chạy song song, một chương trình hẹn giờ có thể chuyển bộ xử lý đang chạy chương trình này sang chương trình khác và ngược lại. Bằng cách này, kết quả cuối cùng là một số chương trình độc lập chạy cùng một lúc mặc dù chúng đều sử dụng cùng một bộ xử lý.

Bất cứ khi nào bộ xử lý di chuyển từ chương trình này sang chương trình khác hoặc đến một chương trình ngắt, nó phải duy trì các biến của chương trình và chủ yếu là nội dung của các thanh ghi; nếu không, sẽ xảy ra những điều không mong muốn trong quá trình bộ xử lý quay trở lại chương trình mà nó đã để lại.

Trong hầu hết các bộ xử lý khác nhau, các chương trình ngắt phải bao gồm một lệnh **Push** cho nội dung của các thanh ghi mà chương trình sử dụng cho ngăn xếp. Trước khi quay trở lại từ chương trình ngắt, lệnh **Pop** kéo các nội dung đã lưu trữ trở lại thanh ghi phải được thực hiện.

Quá trình đẩy và kéo nội dung này là một quá trình tương đối dài, được thực hiện nhiều lần và làm chậm tốc độ hoạt động của bộ xử lý.

Để cải thiện tốc độ xử lý, ARM bao gồm một bộ phận tự động đẩy và kéo nội dung của một vài thanh ghi trong khi chuyển đến chương trình ngắt và quay trở lại từ đó, như chúng ta sẽ thấy ở phần sau. Nó giúp tiết kiệm nhu cầu về các lệnh push-pull và được thực hiện bởi phần cứng của bộ xử lý.

Trong khi một chương trình chạy, có thể do dữ liệu không thể đoán trước, nhiễu trong hệ thống hoặc do lỗi trong chương trình, nên chương trình sẽ thực hiện các hành động có thể làm hỏng hệ thống và đánh sập nó. Chương trình loại này có thể ghi vào các nơi không cho phép và thậm chí gây hỗn loạn chính nó.

Để tăng khả năng miễn nhiễm của hệ thống đối với các tình huống trên, ARM cho phép vận hành các chương trình ở mức chế độ không ưu tiên và vận hành phần khác của các chương trình ở mức chế độ ưu tiên. Nội dung của thanh ghi CONTROL xác định mức đặc quyền của chương trình.

ARM có 7 chế độ xử lý:

- **User** - chế độ không ưu tiên, theo đó hầu hết các tác vụ đều chạy.
- **FIQ** - Yêu cầu ngắt nhanh được nhập khi ngắt ở mức ưu tiên cao (nhanh) được gửi đi.
- **IRQ** - Yêu cầu ngắt được nhập khi ngắt ở mức ưu tiên thấp (bình thường) được gửi đi.

▪ **Supervisor** - được nhập khi đặt lại (reset) và khi ngắt mềm được (software interrupt) thực thi.

▪ **Abort** - được sử dụng để xử lý các vi phạm truy cập bộ nhớ.

▪ **Undef** - được sử dụng để xử lý các lệnh không xác định.

▪ **System** - chế độ ưu tiên sử dụng các thanh ghi giống như chế độ người dùng.

Khi vận hành hệ thống, CPU chạy chương trình chính ở trạng thái **Supervisor**, nghĩa là với tất cả các ưu tiên.

Chương trình chính có thể vận hành chương trình khác và chuyển nó sang trạng thái **User** với các quyền hạn chế.

Hệ thống có hai loại ngắt:

▪ FIQ - Yêu cầu ngắt nhanh

▪ IRQ - Yêu cầu ngắt

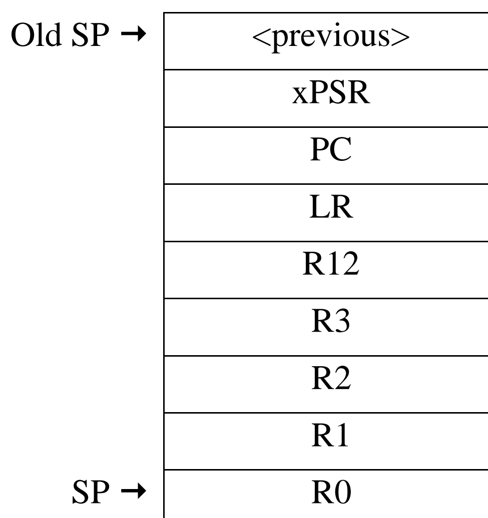
Ngoài ra, một lệnh định địa chỉ đến một bộ nhớ bị chặn hoặc bộ nhớ không tồn tại có thể sẽ được thực hiện. Trong trường hợp này, một ngắt **Abort** sẽ được thực thi.

Nếu một điện áp không mong muốn xuất hiện hoặc việc đọc dữ liệu sai xảy ra trong hệ thống, bộ xử lý sẽ đọc một ký hiệu không thể xác định được và nó sẽ thực hiện ngắt **Undef**.

Các ngắt **Abort** và **Undef** cho phép cải thiện độ tin cậy của hệ thống và khắc phục các sự kiện có vấn đề, đôi khi là ngay cả trong khi thực hiện quá trình tự động khởi động lại hệ thống.

Bất cứ khi nào xảy ra ngắt, bộ xử lý sẽ tự động lưu các thanh ghi PSR, PC, LR, R12, R3, R2, R1 và R0.

Chương trình ngắt có thể sử dụng các thanh ghi này một cách tự do mà không làm hại đến chương trình bị ngắt mà bộ xử lý sẽ quay trở lại khi kết thúc chương trình ngắt.



Nội dung gốc của các thanh ghi này được tự động trả lại cho chúng khi trở về từ chương trình ngắt.

Khi kết thúc quá trình đẩy, SP chỉ báo thanh ghi R0 và nội dung của nó sẽ là thứ đầu tiên được trả về. Hoạt động đẩy tự động được gọi là **pre-emption**. Thao tác này tiết

kiệm rất nhiều thời gian và tăng đáng kể tốc độ của bộ xử lý, vì hoạt động thường xuyên của bộ xử lý bao gồm một số lượng lớn các chương trình ngắt đang chạy, nên việc này giúp tiết kiệm yêu cầu lưu và khôi phục dữ liệu như một phần của chương trình ngắt.

Thời gian cần thiết để chương trình ngắt kể từ thời điểm được yêu cầu cho đến khi nó bắt đầu thực hiện nhiệm vụ của mình, và thêm nữa là thời gian cần thiết kể từ thời điểm hoàn thành nhiệm vụ cho đến khi chương trình (chương trình ngắt được phân nhánh từ đó) trở lại nhiệm vụ ban đầu được gọi là **Interrupt Latency**. Mục đích là để thời gian này càng ngắn càng tốt.

Nếu chương trình ngắt cần các thanh ghi, nó sẽ đẩy chúng vào ngăn xếp và kéo chúng trở lại ngăn xếp trước khi quay trở lại chương trình mà chúng phân nhánh.

Mỗi chương trình ngắt nhận được hai thanh ghi bổ sung (SP và LR) để sử dụng riêng. Bằng cách này, chương trình ngắt có thể sử dụng một vùng nhớ khác làm ngăn xếp và dành địa chỉ trả về cho chính nó, nếu nó sử dụng các hàm. Phương pháp này làm tăng khả năng miễn dịch của hệ thống lên rất nhiều.

Một thanh ghi khác có thể được tùy ý sử dụng bởi mọi chương trình ngắt là SPSR. Thanh ghi này lưu trạng thái cờ của chương trình mà bộ xử lý rẽ nhánh từ đó. Các cờ là một cơ chế cho phép bộ xử lý hoạt động theo lệnh được điều chỉnh. Các cờ sẽ được giải thích ở phần sau.

Ngắt FIQ nhận các thanh ghi bổ sung R8-R12 (tổng cộng là 8) để sử dụng riêng. Bằng cách này, nó không phải đẩy nội dung của các thanh ghi này vào ngăn xếp và tăng tốc độ phản ứng của nó.

Bảng các thanh ghi ARM thể hiện như sau:

User supervisor	FIQ	IRQ	Abort	Undef
R0				
R1				
R2				
R3				
R4				
R5				
R6				
R7				
R8	R8			
R9	R9			
R10	R10			
R11	R11			
R12	R12			
R13(SP)	R13(SP)	R13(SP)	R13(SP)	R13(SP)

R14(LR)	R14(LR)	R14(LR)	R14(LR)	R14(LR)
R15(PC)				
	SPSR	SPSR	SPSR	SPSR

Ngoại trừ các ngắt trên còn có các ngắt khác. Một số là bởi các thành phần của ARM như ngắt giao tiếp, bộ định thời, các thành phần ngoại vi và hơn thế nữa.

Một kiểu ngắt khác là ngắt phần mềm (software interrupt). Ở đây chúng ta giải quyết các đoạn chương trình dịch vụ nhất định giống như khi xử lý một ngắt.

Tất cả các loại ngắt được gọi là các **Ngoại lệ (Exceptions)**.

Cơ chế xử lý ngắt được gọi là **Trình xử lý ngoại lệ (Exception handler)**.

4.4. Chế độ hoạt động

Bộ vi xử lý ARM được thiết kế dành cho các hệ điều hành **(OS)**. Trong các hệ thống này, chương trình chính tải, chạy và hỗ trợ nhiều chương trình khác nhau theo yêu cầu của người dùng hoặc nhu cầu của hệ thống.

Hệ điều hành cấu tạo bởi một **nhân hệ điều hành (kernel)**, nó bao gồm tất cả các chương trình hỗ trợ và dịch vụ, định địa chỉ các đoạn chương trình đối với phần cứng của hệ thống, các chương trình giao tiếp để kết nối nhiều chương trình ứng dụng khác nhau, yêu cầu của người dùng đối với các chương trình ứng dụng, dịch vụ và hỗ trợ.

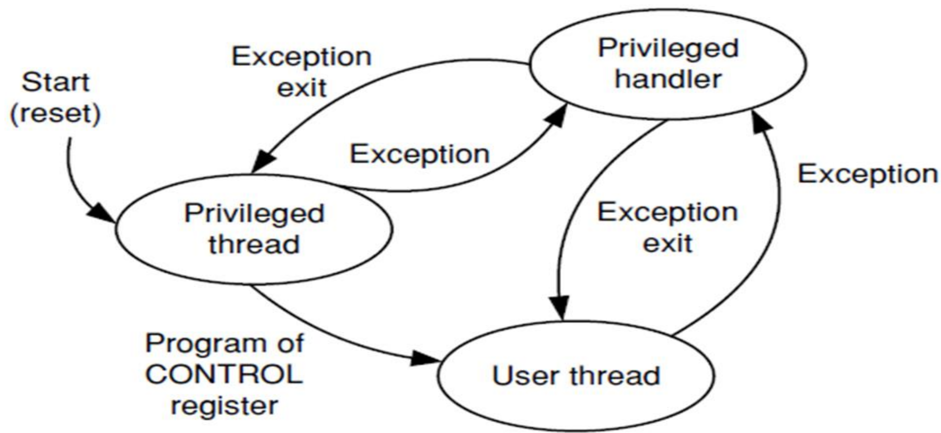
Bộ vi xử lý Cortex-M3 có hai chế độ và hai mức ưu tiên. Các chế độ hoạt động (chế độ chuỗi và chế độ xử lý) xác định xem bộ vi xử lý đang chạy một chương trình bình thường hay đang chạy một trình xử lý ngoại lệ như trình xử lý ngắt hoặc trình xử lý ngoại lệ hệ thống (xem bên dưới). Các mức ưu tiên (mức được ưu tiên và mức người dùng) cung cấp một cơ chế để bảo vệ các truy cập bộ nhớ tới các vùng quan trọng cũng như cung cấp một mô hình bảo mật cơ bản.

	Privileged	*User*
When running an exception handler	Handler mode	
When not running an exception handler (e.g., main program)	Thread mode	Thread mode

Hình 2. 89

Bộ vi xử lý có thể ở trạng thái ưu tiên hoặc ở trạng thái người dùng khi nó chạy trên một chương trình chính (**thread mode**). Các trình xử lý ngoại lệ chỉ có thể ở trạng thái ưu tiên. Khi bộ vi xử lý thoát khỏi thiết đặt lại (reset), nó thread mode, với các quyền truy cập ưu tiên. Ở trạng thái ưu tiên, một chương trình có quyền truy cập vào tất cả các phạm vi bộ nhớ, ngoại trừ trường hợp bị cấm bởi cài đặt MPU (Bộ bảo vệ bộ nhớ) và có thể sử dụng tất cả các lệnh được hỗ trợ.

Phần mềm ở cấp độ truy cập ưu tiên có thể chuyển chương trình sang cấp độ truy cập của người dùng bằng cách sử dụng thanh ghi điều khiển. Khi một ngoại lệ (exception) xảy ra, bộ vi xử lý sẽ luôn chuyển về trạng thái ưu tiên và trở về trạng thái trước đó khi thoát khỏi trình xử lý ngoại lệ. Một chương trình người dùng không thể thay đổi trở lại trạng thái ưu tiên bằng cách ghi vào thanh ghi điều khiển (xem hình sau).



Hình 2. 90

Nó phải thông qua một trình xử lý ngoại lệ để lập trình thanh ghi điều khiển giúp chuyển bộ vi xử lý trở lại mức truy cập ưu tiên khi quay trở lại thread mode.

Việc tách biệt các cấp ưu tiên và người dùng giúp cải thiện độ tin cậy của hệ thống bằng cách ngăn các thanh ghi cấu hình hệ thống bị truy cập hoặc thay đổi bởi một số chương trình không đáng tin cậy. Nếu có sẵn một MPU, nó có thể được sử dụng cùng với các mức ưu tiên để bảo vệ các vị trí bộ nhớ quan trọng, chẳng hạn như chương trình và dữ liệu cho hệ điều hành.

Ví dụ, với các quyền truy cập ưu tiên, thường được sử dụng bởi nhân hệ điều hành, tất cả các vị trí bộ nhớ đều có thể được truy cập (trừ khi bị cấm bởi thiết lập MPU). Khi HĐH khởi chạy một ứng dụng người dùng, nó có khả năng được thực thi ở cấp độ truy cập của người dùng để bảo vệ hệ thống không bị lỗi do sự cố của các chương trình người dùng không đáng tin cậy.

4.5. Bộ điều khiển ngắt vectơ lồng nhau được tích hợp sẵn

Mỗi đoạn chương trình dịch vụ xử lý ngắt đều nằm ở một vị trí khác nhau trong bộ nhớ. Đoạn chương trình dịch vụ này được gọi là đoạn chương trình ngắt.

Địa chỉ (vector) của đoạn chương trình ngắt được đặt tại một vị trí xác định trong bộ nhớ.

Khi nhận được một yêu cầu ngắt, bộ điều khiển ngắt định địa chỉ ô có chứa vectơ ngắt (địa chỉ), trích xuất địa chỉ của đoạn chương trình ngắt được yêu cầu và giúp bộ vi xử lý để xử lý đoạn chương trình ngắt.

Các ô chứa các vectơ đoạn chương trình ngắt được đặt lần lượt. Vùng nhớ này thường là vùng cố định.

Trong các ngắt lồng nhau (nested interrupt), vùng bộ nhớ có chứa các vectơ ngắt có thể được thay đổi. Có thể được mô tả như là việc di chuyển bộ xếp lồng vào nhau của các ngắt, đó chính là nguồn gốc của cái tên ‘nested interrupt’.

Bộ vi xử lý Cortex-M3 bao gồm một bộ điều khiển ngắt được gọi là Khối quản lý ngắt trên các dòng vi điều khiển (NVIC). Nó được kết hợp chặt chẽ với lõi bộ vi xử lý và cung cấp một số tính năng như sau:

- Hỗ trợ ngắt lồng nhau
- Hỗ trợ ngắt vector

- Hỗ trợ thay đổi ưu tiên động
- Giảm độ trễ ngắt
- Mặt nạ ngắt

4.5.1. Hỗ trợ ngắt lồng nhau

NVIC cung cấp hỗ trợ ngắt lồng nhau. Tất cả các ngắt bên ngoài và hầu hết các ngoại lệ của hệ thống đều có thể được lập trình theo các mức ưu tiên khác nhau. Khi một ngắt xảy ra, NVIC sẽ so sánh mức ưu tiên của ngắt này với mức ưu tiên đang chạy hiện tại. Nếu mức độ ưu tiên của ngắt mới cao hơn mức hiện tại, trình xử lý ngắt của ngắt mới sẽ ghi đè lên tác vụ đang chạy hiện tại.

4.5.2. Hỗ trợ ngắt vector

Bộ vi xử lý Cortex-M3 có hỗ trợ ngắt vector. Khi một ngắt được chấp nhận, địa chỉ bắt đầu của đoạn chương trình phục vụ ngắt (ISR) được đặt từ một bảng vectơ trong bộ nhớ. Không cần sử dụng phần mềm để xác định và phân nhánh đến địa chỉ bắt đầu của ISR. Do đó, việc này cần ít thời gian hơn để xử lý yêu cầu ngắt.

4.5.3. Hỗ trợ thay đổi ưu tiên động

Mức độ ưu tiên của các ngắt có thể được thay đổi bằng phần mềm trong suốt thời gian chạy. Các ngắt đang chờ sẽ bị chặn kích hoạt lâu hơn cho đến khi ISR được hoàn thành, vì vậy mức độ ưu tiên của chúng có thể được thay đổi mà không gây ra nguy cơ vô tình thử lại.

4.5.4. Giảm độ trễ ngắt

Bộ vi xử lý Cortex-M3 cũng bao gồm một số tính năng nâng cao để giảm độ trễ ngắt.

Chúng bao gồm tự động lưu và khôi phục một số nội dung của thanh ghi, giảm độ trễ khi chuyển từ ISR này sang ISR khác và xử lý các ngắt đến sau.

4.5.5. Mặt nạ ngắt

Xử lý ngắt và các ngoại lệ hệ thống có thể được che lại dựa trên mức ưu tiên của chúng hoặc được che hoàn toàn bằng cách sử dụng các thanh ghi mặt nạ ngắt BASEPRI, PRIMASK và FAULTMASK. Chúng có thể được sử dụng để đảm bảo rằng các tác vụ quan trọng về thời gian có thể được hoàn thành đúng hạn mà không bị gián đoạn.

4.6. Bản đồ bộ nhớ

Cortex-M3 có một bản đồ bộ nhớ được xác định trước. Điều này cho phép các thiết bị ngoại vi tích hợp sẵn, chẳng hạn như bộ điều khiển ngắt và các thành phần gỡ lỗi, được truy cập bằng các lệnh truy cập bộ nhớ đơn giản. Do đó, hầu hết các tính năng của hệ thống đều có thể truy cập được trong mã chương trình C. Bản đồ bộ nhớ được xác định trước cũng cho phép bộ vi xử lý Cortex-M3 được tối ưu hóa cao về tốc độ và dễ dàng tích hợp trong các thiết kế hệ thống trên chip (SoC).

Nhìn chung, không gian bộ nhớ 4GB có thể được chia thành các dải như trong hình sau.

Address	Region	Description
0xFFFFFFFF – 0xE0000000	System level	Private peripherals including build-in interrupt controller (NVIC), MPU control registers, and debug components
0xDFFFFFFF – 0xA0000000	External device	Mainly used as external peripherals
0x9FFFFFFF – 0x60000000	External RAM	Mainly used as external memory
0x5FFFFFFF – 0x40000000	Peripherals	Mainly used as peripherals
0x3FFFFFFF – 0x20000000	SRAM	Mainly used as static RAM
0x1FFFFFFF – 0x00000000	CODE	Mainly used for program code. Also provides exception vector table after power up

Hình 2. 91

Thiết kế của Cortex-M3 có cơ sở hạ tầng bus nội bộ được tối ưu hóa cho việc sử dụng bộ nhớ này. Ngoài ra, thiết kế này cũng cho phép các vùng này được sử dụng khác nhau. Ví dụ, bộ nhớ dữ liệu vẫn có thể được đưa vào vùng CODE, và mã chương trình có thể được thực thi từ vùng bộ nhớ truy cập tạm thời (RAM) bên ngoài.

Vùng bộ nhớ cấp hệ thống chứa bộ điều khiển ngắt và các thành phần gỡ lỗi. Các thiết bị này có địa chỉ cố định, được trình bày chi tiết trong Chương 5. Với việc có được địa chỉ cố định cho các thiết bị ngoại vi này, bạn có thể chuyển các ứng dụng giữa các sản phẩm Cortex-M3 khác nhau một cách dễ dàng hơn nhiều.

4.7. Giao diện bus

Có một số giao diện bus trên bộ vi xử lý Cortex-M3. Chúng cho phép Cortex-M3 thực hiện tìm nạp lệnh và truy cập dữ liệu cùng một lúc. Có các giao diện bus chính như sau:

- Bus bộ nhớ mã
- Bus hệ thống
- Bus thiết bị ngoại vi riêng

Việc truy cập vùng bộ nhớ mã được thực hiện trên các bus bộ nhớ mã, về mặt vật lý bao gồm hai bus, một được gọi là I-Code và một được gọi là D-Code. Chúng được tối ưu hóa cho việc tìm nạp lệnh để có tốc độ thực thi lệnh tốt nhất.

Bus hệ thống được sử dụng để truy cập bộ nhớ và các thiết bị ngoại vi. Điều này cung cấp quyền truy cập vào bộ nhớ truy cập ngẫu nhiên tĩnh (SRAM), các thiết bị ngoại vi, RAM ngoài,các thiết bị bên ngoài và một phần của vùng bộ nhớ cấp hệ thống.

Bus ngoại vi riêng (PPB) cung cấp quyền truy cập vào một phần của bộ nhớ cấp hệ thống dành riêng cho các thiết bị ngoại vi riêng, chẳng hạn như các thành phần gỡ lỗi.

4.8. MPU

Cortex-M3 có một MPU tùy chọn. Thiết bị này cho phép thiết lập các quy tắc truy cập dành cho truy cập ưu tiên và truy cập chương trình người dùng. Khi một quy tắc truy cập bị vi phạm, một ngoại lệ lỗi sẽ được khởi tạo và trình xử lý ngoại lệ lỗi sẽ có thể phân tích lỗi và sửa chữa nó, nếu có thể.

MPU có thể được sử dụng theo nhiều cách khác nhau. Trong các tình huống thông thường, hệ điều hành có thể thiết lập MPU để bảo vệ dữ liệu được sử dụng bởi nhân hệ điều hành và các chương trình ưu tiên khác để được bảo vệ khỏi các chương trình người dùng không đáng tin cậy.

MPU cũng có thể được sử dụng để biến các vùng bộ nhớ thành chế độ chỉ đọc, để ngăn chặn việc vô tình xóa dữ liệu hoặc để cô lập các vùng bộ nhớ giữa các tác vụ khác nhau trong một hệ thống đa nhiệm. Nhìn chung, nó có thể giúp làm cho các hệ thống nhúng mạnh mẽ và đáng tin cậy hơn.

Tính năng MPU là tùy chọn và được xác định trong giai đoạn thiết kế vi điều khiển hoặc thiết kế on-chip (SoC).

4.9. Các ngắt và các ngoại lệ

Bộ vi xử lý Cortex-M3 thực hiện một kiểu ngoại lệ mới, được giới thiệu trong cấu trúc ARMv7-M.

Kiểu ngoại lệ này khác với kiểu ngoại lệ ARM truyền thống, cho phép xử lý ngoại lệ rất hiệu quả. Nó có nhiều ngoại lệ hệ thống cộng thêm với một số yêu cầu ngắt ngoài (IRQ) (các ngõ vào ngắt ngoài). Không có ngắt nhanh (FIQ) trong Cortex-M3; tuy nhiên, xử lý ưu tiên ngắt và hỗ trợ ngắt lồng nhau hiện được đưa vào cấu trúc ngắt. Do đó, rất dễ dàng để thiết lập một hệ thống hỗ trợ các ngắt lồng nhau (một ngắt có mức ưu tiên cao hơn có thể ghi đè hoặc chặn một trình xử lý ngắt có mức ưu tiên thấp hơn) và nó hoạt động giống như FIQ trong các bộ vi xử lý ARM truyền thống.

Các tính năng ngắt trong Cortex-M3 được thực hiện trong NVIC. Ngoài việc hỗ trợ các ngắt ngoài, Cortex-M3 còn hỗ trợ một số nguồn ngoại lệ bên trong, chẳng hạn như xử lý lỗi hệ thống.

Do đó, Cortex-M3 có một số kiểu ngoại lệ được xác định trước, như được hiển thị trong bảng 2.2.

Bảng 2.2 Các loại ngoại lệ Cortex-M3

Exception Number	Exception Type	Priority (Default to 0 if Programmable)	Description
0	NA	NA	No exception running
1	Reset	–3 (Highest)	Reset
2	NMI	–2	NMI (external NMI input)
3	Hard fault	–1	All fault conditions, if the corresponding
4	MemManage fault	Programmable	fault handler is not enabled
			Memory management fault; MPU
5		Programmable	Violation or access to illegal

6	Bus fault	Programmable	locations
7-10	Usage fault	NA	Bus error (prefetch abort or data abort)
11	Reserved	Programmable	Program error
12	SVCall	Programmable	Reserved
	Debug monitor		Supervisor call
13		NA	Debug monitor (break points),
14	Reserved	Programmable	Watchpoints, or external debug request)
15	PendSV	Programmable	Reserved
16	SYSTICK	Programmable	Pendable request for system service
17	IRQ #0	Programmable	
…	IRG #1	…	
255	…	Programmable	System tick timer
	IRQ #239		External interrupt #0
			External interrupt #1
			…
			External interrupt #239
Số lượng ngõ vào ngắt ngoài được xác định bởi các nhà sản xuất chip. Có thể hỗ trợ tối đa 240 ngõ vào ngắt ngoài. Ngoài ra, Cortex-M3 còn có ngõ vào ngắt NMI. Khi được gán, NMI-ISR được thực thi vô điều kiện.			

4.9.1. Công suất thấp và hiệu năng cao

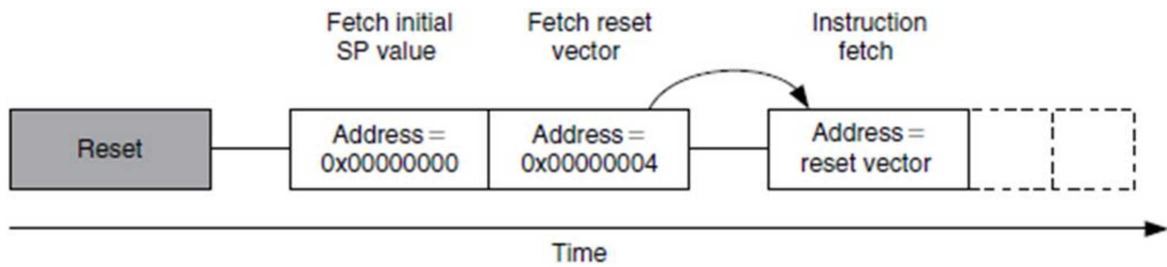
Bộ vi xử lý Cortex-M3 được thiết kế với nhiều tính năng khác nhau cho phép các nhà thiết kế phát triển các sản phẩm tiết kiệm năng lượng và hiệu năng cao. Đầu tiên, nó hỗ trợ chế độ ngủ và chế độ ngủ sâu, vì thế có thể hoạt động với nhiều phương pháp thiết kế hệ thống khác nhau để giảm tiêu thụ điện năng khi máy tính nghỉ.

Thứ hai, số lượng cổng thấp và kỹ thuật thiết kế của nó làm giảm các hoạt động của mạch điện trong bộ vi xử lý để cho phép giảm công suất hoạt động. Ngoài ra, vì Cortex-M3 có mật độ mã cao, nó đã làm giảm yêu cầu về kích thước chương trình. Đồng thời, nó cho phép hoàn thành các tác vụ xử lý trong thời gian ngắn, vì thế bộ vi xử lý có thể quay lại chế độ ngủ sớm nhất có thể nhằm cắt giảm năng lượng sử dụng. Do đó, hiệu năng của Cortex-M3 tốt hơn nhiều so với các vi điều khiển 8-bit hoặc 16-bit.

Bắt đầu từ phiên bản 2 của Cortex-M3, một tính năng mới được gọi là Wakeup Interrupt Controller (WIC) đã có sẵn. Tính năng này cho phép toàn bộ lõi bộ vi xử lý được tắt nguồn, trong khi các trạng thái của bộ vi xử lý được giữ lại và bộ vi xử lý có thể trở về trạng thái hoạt động gần như ngay lập tức khi xảy ra ngắt. Điều này làm cho Cortex-M3 thậm chí còn phù hợp hơn với nhiều ứng dụng tiêu thụ điện năng cực thấp mà trước đây chỉ có thể được thực hiện với các vi điều khiển 8 bit hoặc 16 bit.

4.10. Đặt lại trình tự

Sau khi bộ vi xử lý thoát đặt lại (reset) nó sẽ đọc hai word từ bộ nhớ (xem hình sau):



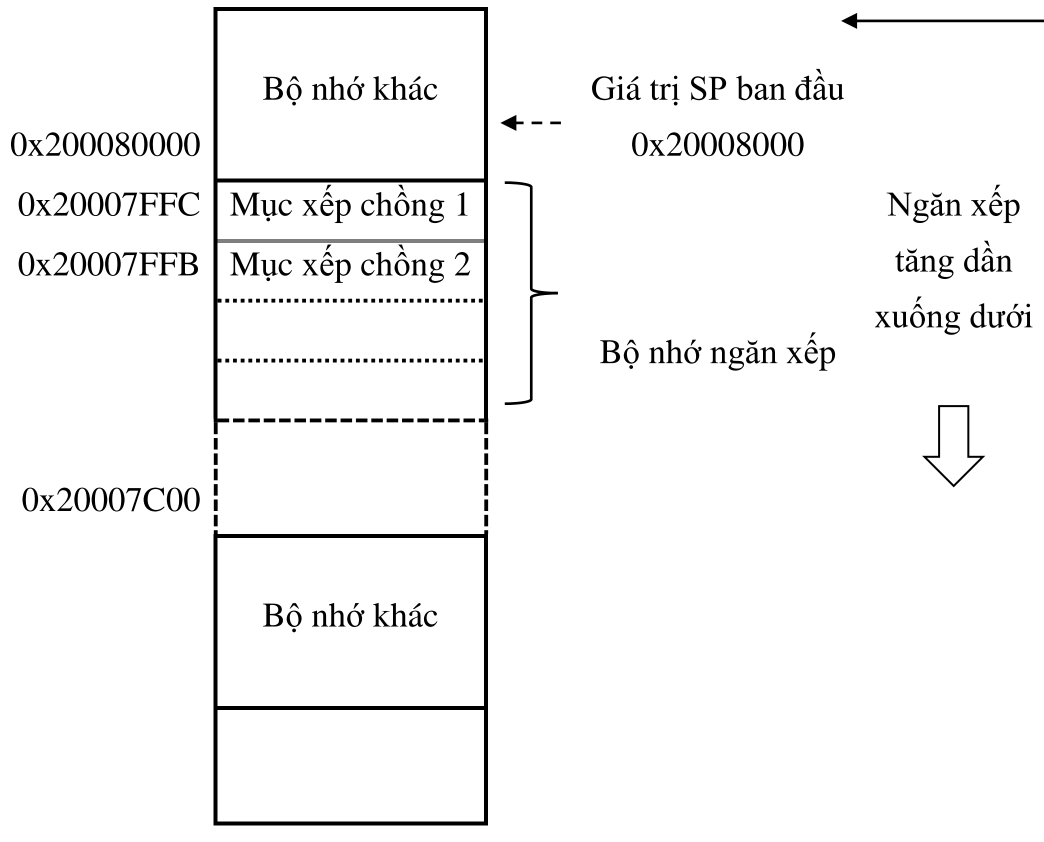
Hình 2. 92

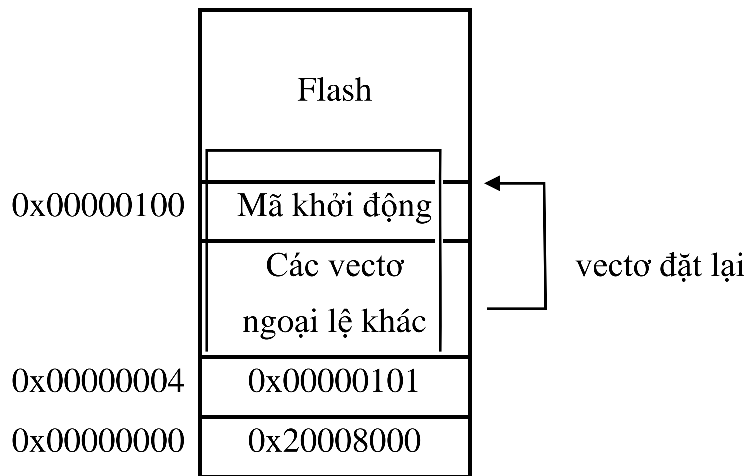
- Địa chỉ 0x00000000 - Giá trị bắt đầu của R13 (SP)
- Địa chỉ 0x00000004 - Vectơ đặt lại (địa chỉ bắt đầu thực thi chương trình; LSB phải được đặt thành '1' để báo trạng thái Thumb)

Trong Cortex-M3, giá trị ban đầu cho MSP được đặt ở đầu bản đồ bộ nhớ, tiếp theo là bảng vectơ chứa các giá trị địa chỉ vectơ (bảng vectơ có thể được chuyển đến vị trí khác sau này trong suốt quá trình thực thi chương trình). Ngoài ra, nội dung của bảng vectơ là các giá trị địa chỉ chứ không phải là các lệnh rẽ nhánh. Vectơ đầu tiên trong bảng vectơ (ngoại lệ loại 1) là vectơ đặt lại, là phần dữ liệu thứ hai được bộ xử lý tìm nạp sau khi đặt lại.

Bởi vì hoạt động ngăn xếp trong Cortex-M3 là một ngăn xếp đầy đủ giảm dần (giảm SP trước khi lưu trữ), giá trị SP ban đầu phải được đặt thành bộ nhớ đầu tiên sau đỉnh của vùng ngăn xếp. Ví dụ: nếu bạn có dải bộ nhớ ngăn xếp từ 0x20007C00 đến 0x20007FFF (1KB), giá trị ngăn xếp ban đầu phải được đặt thành 0x20008000.

Bảng vectơ bắt đầu sau giá trị SP ban đầu. Vectơ đầu tiên là vectơ đặt lại. Lưu ý rằng trong Cortex-M3, LSB của các địa chỉ vectơ trong bảng vectơ phải được đặt thành “1” để báo rằng chúng là mã Thumb. Vì lý do đó, ví dụ trước có giá trị 0x101 trong vectơ đặt lại, trong khi mã khởi động bắt đầu ở địa chỉ 0x100 (xem hình sau).





Hình 2. 93

Sau khi vectơ đặt lại được tìm nạp, Cortex-M3 có thể bắt đầu thực hiện chương trình từ địa chỉ vectơ đặt lại và bắt đầu hoạt động bình thường. Cần phải khởi tạo SP vì một số ngoại lệ (chẳng hạn như NMI) có thể xảy ra ngay sau khi đặt lại và bộ nhớ ngăn xếp có thể được yêu cầu cho trình xử lý các ngoại lệ đó.

Các công cụ phát triển phần mềm khác nhau có thể có các cách khác nhau để chỉ định giá trị SP bắt đầu và vectơ đặt lại. Nếu bạn cần thêm thông tin về chủ đề này, tốt nhất bạn nên xem các ví dụ Project được cung cấp cùng với các công cụ phát triển.

4.11. Trình tự ngắt/ngoại lệ

Một ngoại lệ xảy ra có thể kéo theo một số sự kiện khác, chẳng hạn như:

- Xếp chồng (đẩy nội dung của tám thanh ghi vào ngăn xếp)
- Tìm nạp vectơ (đọc địa chỉ bắt đầu của trình xử lý ngoại lệ từ bảng vectơ)
- Cập nhật con trỏ ngăn xếp, thanh ghi liên kết (LR) và bộ đếm chương trình (PC)

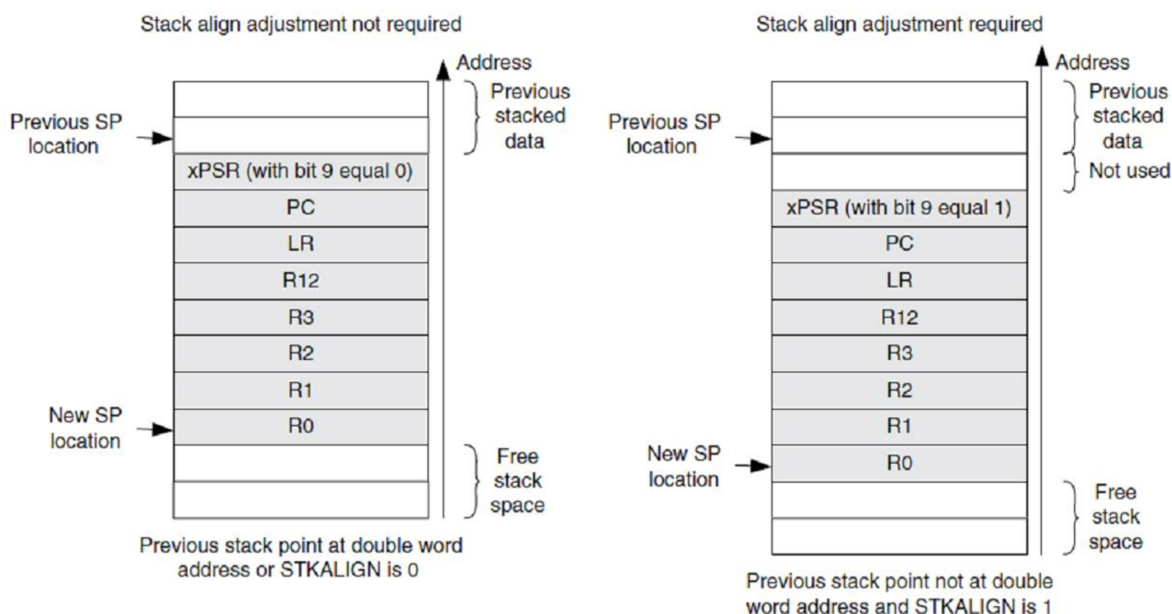
4.11.1. Xếp chồng

Khi một ngoại lệ xảy ra, các thanh ghi R0-R3, R12, LR, PC và thanh ghi trạng thái chương trình (PSR) được đẩy vào ngăn xếp. Nếu mã đang chạy sử dụng con trỏ ngăn xếp quy trình (PSP), thì ngăn xếp quy trình sẽ được sử dụng; nếu mã đang chạy sử dụng con trỏ ngăn xếp chính (MSP), thì ngăn xếp chính sẽ được sử dụng.

Sau đó, ngăn xếp chính sẽ luôn được sử dụng trong trình xử lý, vì vậy tất cả các ngắt lồng nhau sẽ sử dụng ngăn xếp chính.

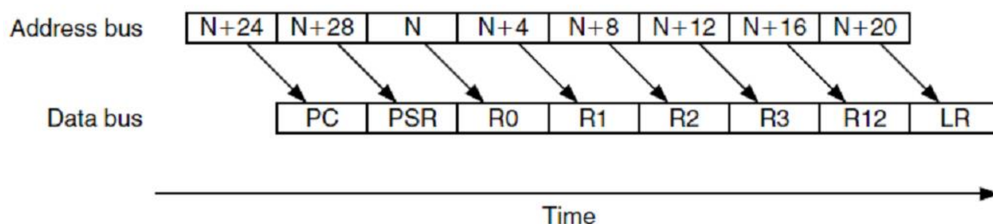
Khối 8 word dữ liệu được đẩy vào ngăn xếp thường được gọi là khung ngăn xếp. Trong Cortex-M3, khung ngăn xếp được sắp xếp thành địa chỉ double word theo mặc định, mặc dù tính năng mảng có thể được tắt bằng cách lập trình bit STKALIGN trong NVIC về “0”.

Sự sắp xếp dữ liệu bên trong một khung ngăn xếp ngoại lệ được thể hiện trong hình 2.94



Hình 2. 94

Thứ tự xếp chồng được thể hiện trong hình 2.95 (giả sử rằng giá trị [SP] của con trỏ ngăn xếp là N sau ngoại lệ).



Hình 2. 95

Do bản chất đường ống của giao tiếp Bus hiệu suất cao (AHB), địa chỉ và dữ liệu được bù bởi một trạng thái đường ống.

Các giá trị của PC và PSR được xếp chồng lên nhau trước để có thể bắt đầu sớm việc tìm nạp lệnh (điều này yêu cầu thay đổi PC) và có thể sớm cập nhật thanh ghi trạng thái chương trình ngắt (IPSR).

Sau khi xếp chồng, SP sẽ được cập nhật và cách sắp xếp dữ liệu xếp chồng trong bộ nhớ ngăn xếp sẽ giống như hình 2.95.

Lý do các thanh ghi R0-R3, R12, LR, PC và PSR được xếp chồng lên nhau vì đây là các thanh ghi dễ thay đổi (caller-saved), căn cứ theo các tiêu chuẩn C. Sự sắp xếp này cho phép trình xử lý ngắt là một hàm C bình thường bởi vì các thanh ghi có thể được thay đổi bởi trình xử lý ngoại lệ được lưu trong ngăn xếp.

Các thanh ghi chung (R0-R3 và R12) được đặt ở cuối khung ngăn xếp để chúng có thể dễ dàng truy cập bằng cách định địa chỉ liên quan đến SP. Kết quả là, thật dễ dàng để chuyển các tham số tới các ngắt phần mềm bằng cách sử dụng các thanh ghi xếp chồng lên nhau.

4.11.2. Tìm nạp vectơ

Mặc dù bus dữ liệu đang bận xếp chồng các thanh ghi thì bus lệnh thực hiện một tác vụ quan trọng khác của chuỗi ngắt: Nó tìm nạp véc tơ ngoại lệ (địa chỉ bắt đầu của trình xử lý ngoại lệ) từ bảng vectơ. Vì việc xếp chồng và tìm nạp vectơ được thực hiện trên các giao tiếp bus riêng biệt, nên chúng có thể được thực hiện cùng một lúc.

4.11.3. Cập nhật thanh ghi

Sau khi hoàn thành việc xếp chồng và tìm nạp vectơ, vectơ ngoại lệ sẽ bắt đầu thực thi. Khi nhập trình xử lý ngoại lệ, nhiều thanh ghi sẽ được cập nhật như sau:

- SP - SP (là MSP hoặc PSP) sẽ được cập nhật vào vị trí mới trong quá trình xếp chồng. Trong quá trình thực hiện chương trình phục vụ ngắt, MSP sẽ được sử dụng nếu ngăn xếp được truy cập.
- PSR - IPSR (phần thấp nhất của PSR) sẽ được cập nhật thành số ngoại lệ mới.
- PC - PC sẽ thay đổi thành trình xử lý vectơ khi quá trình tìm nạp vectơ hoàn tất và bắt đầu tìm nạp các lệnh từ vectơ ngoại lệ.
- LR - LR sẽ được cập nhật thành một giá trị đặc biệt gọi là EXC_RETURN.1. Giá trị đặc biệt này điều khiển hoạt động trả về ngắt. 4 bit cuối cùng của LR được sử dụng để cung cấp thông tin trả về ngoại lệ.

Một số các thanh ghi NVIC khác cũng sẽ được cập nhật. Ví dụ, trạng thái đang chờ xử lý của ngoại lệ sẽ bị xóa và bit kích hoạt của ngoại lệ sẽ được thiết lập.

4.12. Các lối ra ngoại lệ

Ở cuối trình xử lý ngoại lệ, cần có một lối ra ngoại lệ (được biết đến như là trả về ngắt trong một số bộ vi xử lý) để khôi phục trạng thái hệ thống, do đó chương trình bị ngắt có thể tiếp tục thực thi bình thường.

Có ba cách để kích hoạt trình tự trả về ngắt; tất cả đều sử dụng giá trị đặc biệt được lưu trong LR ở phần đầu của trình xử lý (xem bảng 9.1).

Bảng 9.1 Các lệnh có thể được sử dụng để kích hoạt trả về ngoại lệ

Lệnh trả về	Mô tả
BX reg	Nếu giá trị EXC_RETURN vẫn nằm trong LR, chúng ta có thể sử dụng lệnh BX LR để thực hiện trả về ngắt. Giá trị của LR được đẩy vào ngăn xếp sau khi nhập trình xử lý ngoại lệ.
POP {PC}, or OP {…, PC}	Chúng ta có thể sử dụng lệnh POP, một POP đơn lẻ hoặc nhiều POP, để đặt giá trị EXC-RETURN vào bộ đếm chương trình. Điều này sẽ khiến bộ xử lý thực hiện trả lại ngắt. Có thể tạo ra một trả về ngắt bằng cách sử dụng Lệnh LDR hoặc LDM với PC làm thanh ghi đích.

Load (LDR) or Load multiple (LDM)	

Một số cấu trúc bộ vi xử lý sử dụng các lệnh đặc biệt để trả về ngắt (ví dụ: reti trong 8051). Trong Cortex-M3, một lệnh trả về bình thường được sử dụng để toàn bộ trình xử lý ngắt có thể được thực hiện như một hàm C.

Khi lệnh trả về ngắt được thực thi, các quá trình xếp chồng và cập nhật các thanh ghi NVIC được liệt kê trong bảng 9.1 sẽ được thực hiện.

1. **Bỏ xếp chồng** - Các thanh ghi được đẩy vào ngăn xếp sẽ được khôi phục. Thứ tự của POP sẽ giống như khi xếp chồng. Con trỏ ngăn xếp cũng sẽ được thay đổi trở lại.

2. **Cập nhật thanh ghi NVIC** - Bit kích hoạt của ngoại lệ sẽ bị xóa. Đối với các ngắt ngoài, nếu đầu vào ngắt vẫn được gán, bit đang chờ xử lý sẽ được thiết lập lần nữa, khiến nó nhập lại vào trình xử lý ngắt.

4.13. Ngắt lồng nhau

Hỗ trợ ngắt lồng nhau được tích hợp trong lõi bộ vi xử lý Cortex-M3 và NVIC. Không cần sử dụng hợp ngữ mã trình bao bọc (wrapper) để kích hoạt các ngắt lồng nhau. Trên thực tế, bạn không phải làm gì ngoài việc thiết lập mức ưu tiên thích hợp cho từng nguồn ngắt. NVIC trong bộ xử lý Cortex-M3 sẽ sắp xếp giải mã ưu tiên. Vì vậy, khi bộ vi xử lý đang xử lý một ngoại lệ, tất cả các ngoại lệ khác có cùng mức độ ưu tiên hoặc thấp hơn sẽ bị chặn. Việc xếp chồng và bỏ xếp chồng phần cứng tự động sẽ cho phép trình xử lý ngắt lồng nhau thực thi mà không có nguy cơ mất dữ liệu trong các thanh ghi.

Tuy nhiên, cần chú ý một điều – hãy đảm bảo rằng có đủ không gian trong ngăn xếp chính nếu một vài ngắt lồng nhau được cho phép. Vì mỗi mức ngoại lệ sẽ sử dụng tám word không gian ngăn xếp và mã trình xử lý ngoại lệ có thể yêu cầu thêm không gian ngăn xếp, nên nó có thể kết thúc bằng việc sử dụng nhiều bộ nhớ ngăn xếp hơn mong đợi.

Không cho phép các ngoại lệ Reentrant trong Cortex-M3. Vì mỗi ngoại lệ đều có một mức ưu tiên được gán và trong quá trình xử lý ngoại lệ, các ngoại lệ có cùng mức độ ưu tiên hoặc thấp hơn sẽ bị chặn, ngoại lệ tương tự không thể được thực hiện cho đến khi trình xử lý kết thúc. Vì lý do này, không thể sử dụng các lệnh Supervisor Call (SVC) bên trong trình xử lý SVC, bởi làm như vậy sẽ gây ra ngoại lệ lỗi.

4.14. Ngắt Tail-chaining

Cortex-M3 sử dụng một số phương pháp để cải thiện độ trễ ngắt. Điều đầu tiên chúng ta sẽ xem xét là Tail-chaining.

Khi một ngoại lệ xảy ra nhưng bộ vi xử lý đang xử lý một ngoại lệ khác có cùng mức ưu tiên hoặc cao hơn, ngoại lệ này sẽ chuyển sang trạng thái chờ xử lý cho đến khi bộ vi xử lý hoàn thành việc tác vụ đang thực thi.

Thí nghiệm 4.1 - Yêu cầu ngắt (IRQ)

Mục tiêu:

- Phản ứng với yêu cầu ngắt.

Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192
- Dây cắm thí nghiệm

Thảo luận:

4.1.1. Đoạn chương trình ngắt

Đoạn chương trình ngắt được xây dựng giống như bất kỳ hàm nào. Tên của đoạn chương trình yêu cầu ngắt IRQ (Interrupt Request) cho trình biên dịch biết rằng đây là một ngắt nhất định và trình biên dịch và trình liên kết xác định địa chỉ của đoạn chương trình IRQ ở đúng vị trí của bảng vectơ ngắt. Đoạn chương trình IRQ được gọi là Bộ xử lý IRQ (IRQ Handler).

Bộ vi điều khiển STM32F100 có đến 60 kênh ngắt có thể che lại tùy theo loại thiết bị. Mỗi kênh có thể xử lý bằng một đoạn chương trình xử lý IRQ đặc biệt. Bộ điều khiển ngắt vectơ lồng nhau NVIC (Nested vectored Interrupt Controller) phản ứng với các kênh yêu cầu ngắt theo mức độ ưu tiên và bảng vectơ của chúng.

Bảng vectơ bắt đầu tại địa chỉ 0x0000_0010. Mỗi 4 byte chứa địa chỉ của một đoạn chương trình xử lý IRQ khác. Một project trình biên dịch ARM C phải bao gồm một tệp khởi động với tên của tất cả các đoạn chương trình xử lý IRQ.

Sau đây là đoạn chương trình xử lý IRQ9:5 (một trong những thiết bị ngắt ngoài của STM32F100).

```
void EXTI9_5_IRQHandler (void)
{
EXTI->PR |= NVIC_ISPR_SETPEND_8;//Clear bit 8 in pending register
//must clear every interrupt!
Ext_flag = 1;//Don't blink L0 in main()
all_leds_on;
for (dly = 1 ; dly != 0X400000 ; dly++);
all_leds_off;
Ext_flag = 0;
}
```

Đoạn chương trình này được xây dựng như một hàm tiêu chuẩn. NVIC thực hiện quá trình di chuyển bộ vi xử lý từ chương trình chính sang đoạn chương trình xử lý IRQ, bao gồm lưu bộ đếm chương trình và các thanh ghi, để có thể quay lại và tiếp tục với chương trình chính chính xác tại nơi nó đã dừng.

Project trình biên dịch bao gồm một tệp đặc biệt **startup_stm32f10x_ld_vl.s** với bảng chứa tất cả các địa chỉ vectơ ngoại lệ (đặt lại, ngắt và ngắt mềm). Tên của đoạn chương trình xử lý ngắt này cũng có ở đó.

Chúng ta khai báo các lệnh dễ nhớ như sau:

```
#define all_leds_off GPIOE->BSRR = ((uint32_t)0x00ff0000)//Port E Reset bits 0-
7
```

```
#define all_leds_on GPIOE->BSRR = ((uint32_t)0x000000ff)//Port E Set bits 0-7
```

Thông thường, việc thực thi một IRQ là rất nhanh và có vẻ giống như đang chạy song song với chương trình chính. Trong ví dụ này, đoạn chương trình IRQ có độ trễ để có thể biết khi nào thì đoạn chương trình IRQ này được thực thi.

Sau đây là chương trình chính. Chương trình nhấp nháy đèn LED L0.

```
Int main(void
{
NVIC_SetVectorTable(NVIC_VectTab_FLASH,
INTERUPT_VECTOR_START);
External_Interrupt_Init();
__enable_irq();  //(Enable Interrupts)
Leds_Init();
while(1)
{
led_L0_on;
for(dly=0; dly < 3000000; dly++);  //delay
led_L0_off;
for(dly=0; dly < 3000000; dly++);  //delay
}
}
```

Chương trình sử dụng các khai báo sau.

```
#define GPIO_BSRR_BR0 ((uint32_t)0x00010000)//!< Port x Reset bit 0
#defineled_L0_off GPIOE->BSRR = GPIO_BSRR_BR0//Port E Reset bit 0
#defineled_L0_on GPIOE->BSRR = GPIO_BSRR_BS0//Port E Set bit 0
```

Chương trình gọi bảng vectơ bộ NVIC.

Chương trình cũng gọi hàm ngắt ngoài. Hàm này xác định PB8 của GPIOB là một đường yêu cầu ngắt IRQ (Interrupt Request).

```
void External_Interrupt_Init (void)//Define as external interrupt for pin PB8
{
RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;//Enable GPIOB clock
  //PB8: TP5
GPIOB->CRH &= PBIT8_CLR;//Clear the relevant bits in CRH register
GPIOB->CRH |= (MODE_IN_PU_PD >> PBIT8);//Define PB8 Input with pull-
up / pull-down
GPIOB->BSRR = GPIO_BSRR_BS8;//Port B Set bit 8 - Input with pull-up
```

RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;//Enable clock for Alternate Function

AFIO->EXTICR[2] |= AFIO_EXTICR3_EXTI8_PB;//Set PB[8] pin to use as Alternate Function

EXTI->IMR |= EXTI_IMR_MR8;//Interrupt request from Line 8 is enabled

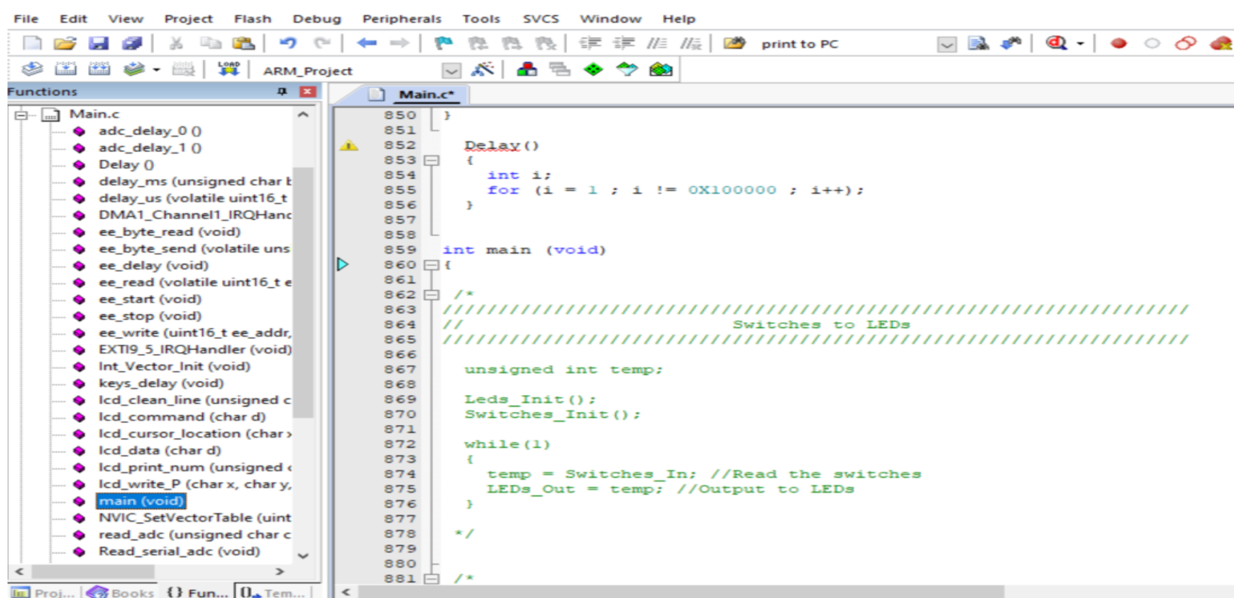
EXTI->FTSR |= EXTI_FTSR_TR8;//Set falling edge at line 8

NVIC->ISER[0] |= NVIC_ISER_SETENA_23;//Enable interrupt EXTI 9..5

}

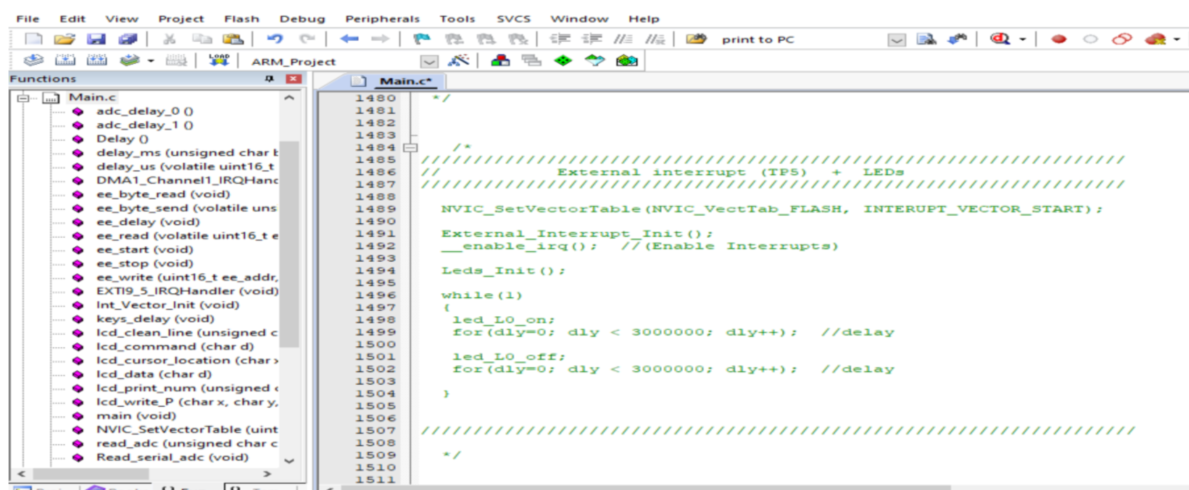
Trình tự thực hiện:

1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này, đường dẫn đến tệp ARM_Project.uvprojx là“C:\Courses\3192\S-ARM_V3\ARM_Project”
2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:



Hình 2. 96

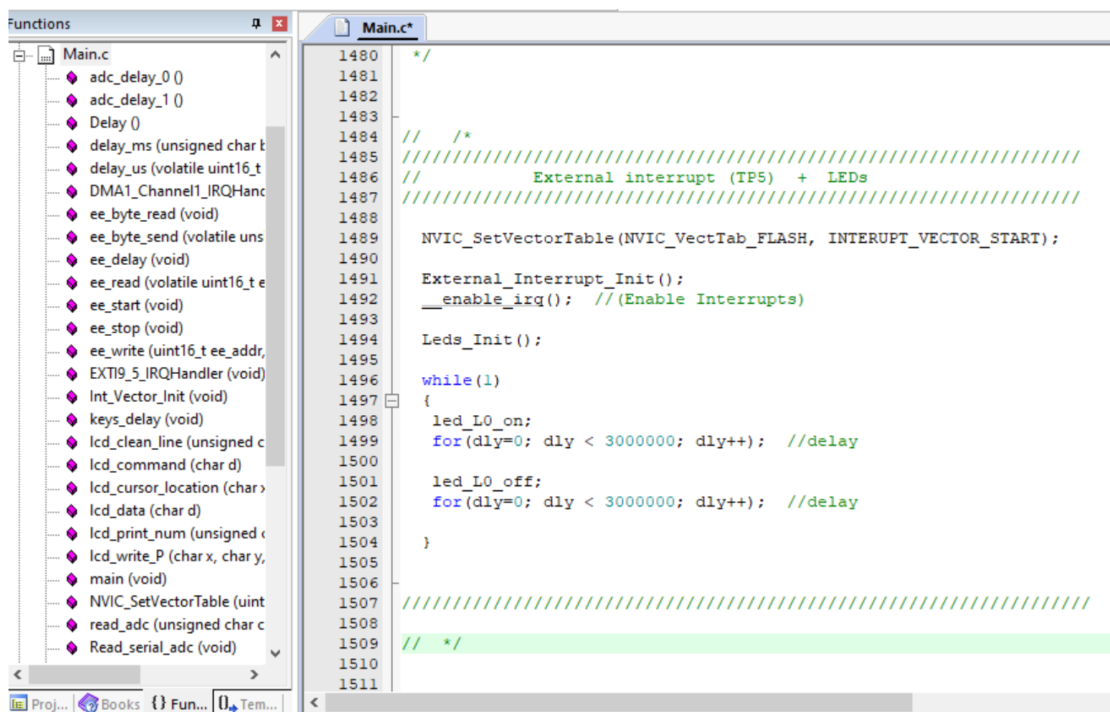
3. Cuộn xuống chương trình **main**() cho đến khi bạn nhận được màn hình sau:



Hình 2. 97

Phần này chứa chương trình **External interrupt (TP%) + LEDs.**

4. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:



Hình 2. 98

5. Quan sát và phân tích chương trình.

6. Kích hoạt EITPS-3192.

7. Dùng dây cắm kết nối TP1 của nút nhấn với TP5 của PB8.

8. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

9. Quan sát đèn LED L0 nhấp nháy.

10. Ấn nút nhấn.

Tất cả các đèn LED sẽ bật và tắt sau 1 giây.

Đèn LED L0 sẽ trở lại nhấp nháy.

11. Nhấn RST để dừng chương trình đang chạy.

12. Cuộn lên chương trình **main.c** và tìm đoạn chương trình **void EXTI9_5_IRQHandler (void).**

13. Thay đổi đoạn chương trình để 4 đèn LED bên trái sẽ nhấp nháy hai lần.

14. Lặp lại các bước 8-11 và kiểm tra phản ứng của hệ thống.

15. Kích hoạt các dấu '/* */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

Thí nghiệm 4.2 - Giao tiếp nối tiếp

Mục tiêu:

- Phân loại các phương thức giao tiếp
- Giao tiếp không đồng bộ nối tiếp
- UART và USART
- Mã ASCII
- Giao tiếp với PC

Thiết bị cần thiết:

- Bộ thí nghiệm vi điều khiển EITPS-3192

Thảo luận:

4.2.1. Phân loại các phương thức giao tiếp

Trong giao tiếp giữa các máy tính, chúng được kết nối với nhau bằng các đường giao tiếp. Ở mỗi giai đoạn của giao tiếp, có một máy tính truyền và một máy tính nhận. Máy phát truyền thông tin thông qua một cổng đầu ra và máy nhận nhận thông tin đó thông qua một cổng đầu vào.

Máy tính truyền có thể truyền và sau đó chuyển sang điều kiện nhận và ngược lại. Không máy tính nào có thể "nhìn thấy" những gì đang xảy ra trong máy tính kia. Các máy tính chỉ có thể đọc thông tin được được đặt trên các cổng đầu vào của chúng. Đó là lý do tại sao một phần của thông tin được truyền đi bao gồm các tín hiệu liên quan đến trạng thái của máy tính truyền và máy tính nhận, các tín hiệu như: "sẵn sàng nhận", "nhận một tin nhắn", "kết thúc tin nhắn" v.v.

Các phương pháp giao tiếp khác nhau được phân thành ba nhóm cơ bản:

a) Đồng bộ và không đồng bộ:

Trong giao tiếp đồng bộ, các máy tính được kết nối với một đường dây chung, đường dây này cung cấp tín hiệu đồng bộ cho cả hai. Tín hiệu đồng bộ cho phép các máy tính biết khi nào cần truyền và khi nào chờ nhận một thông điệp qua các đường giao tiếp. Mỗi máy tính, trước khi truyền một thông điệp, sẽ đợi sự xuất hiện của tín hiệu đồng bộ và sau đó mới bắt đầu truyền. Một máy tính dự kiến nhận một thông báo, phải chờ sự xuất hiện của tín hiệu đồng bộ và sau đó mới thu thập thông tin từ các cổng ngõ vào của nó.

Trong giao tiếp không đồng bộ, chúng ta tránh sử dụng đường xung đồng bộ và máy phát xung. Trên các đường thông tin, chúng ta truyền tín hiệu bắt đầu ở đầu mỗi tin nhắn. Máy tính nhận chờ nhận một tín hiệu như vậy. Sau khi xác định vị trí của nó, máy tính nhận sẽ thu thập tin nhắn, tin nhắn này theo sau tín hiệu đó. Phương thức giao tiếp này được sử dụng phổ biến nhất.

b) Song song và nối tiếp:

Trong giao tiếp song song, chúng ta truyền thông tin ở dạng song song. Một byte gồm 8 bit được truyền qua cáp 8 dây. Mỗi bit được truyền đồng thời trên một dây riêng biệt. Phương pháp này yêu cầu cần có cáp có số lượng dây lớn.

Trong giao tiếp nối tiếp, chúng ta sử dụng một số lượng ít dây. Byte được truyền qua một đường từng bit một. Cả máy phát và máy thu đều phải được đồng bộ với cùng một tần số liên lạc.

c) Polling hoặc Interrupt:

Vấn đề trong giao tiếp là nhận biết khi nào cuộc đối thoại bắt đầu. Một trong những phương pháp để vượt qua trở ngại này là xác định một trong các máy tính là "MASTER" (‘chủ’) và các máy khác là "SLAVES" (‘tớ’). Trạm chủ luôn luôn bắt đầu cuộc giao tiếp. Nó quay sang trạm tớ và hỏi liệu có bất kỳ thông tin nào để truyền hay không. Trạm chủ đợi trong một thời gian nhất định để nhận được tin nhắn từ trạm tớ. Nếu trạm tớ không trả lời trong thời gian đó, thì trạm chủ sẽ trở lại chương trình chính của nó.

Quy trình trên được thực hiện theo các khoảng thời gian định kỳ được xác định trước. Khi trạm tớ có một thông điệp cần truyền, nó sẽ đợi trạm chủ chuyển đến nó và khi điều này xảy ra, trạm tớ sẽ trả lời bằng cách truyền một thông điệp mở đầu. Trạm chủ phản hồi và cuộc đối thoại diễn ra. Phương pháp này được gọi là "giao tiếp bằng kiểm soát vòng (polling)".

Một phương pháp khác để bắt đầu cuộc trò chuyện là ngắt. Chúng ta sử dụng các cổng ngõ vào có đường nhấp nháy (STB). Khi một máy tính muốn ‘nói chuyện’ với một máy khác, nó gửi một thông báo trên cổng đầu ra của chính nó cùng với một xung nhấp nháy. Một cổng đầu vào thu thập thông báo và thực hiện yêu cầu ngắt trong máy tính nhận. Máy nhận thực hiện chương trình ngắt, chương trình này sẽ xử lý thông báo nhận được.

Phương pháp này nhanh chóng và thuận tiện mặc dù nó yêu cầu sử dụng các cổng và các chương trình ngắt thích hợp.

Một cách diễn đạt khác trong giao tiếp là "bắt tay". Điều này có nghĩa là máy truyền tin nhắn đang chờ một xác nhận của máy nhận khi nhận được tin nhắn đó. Nếu không có xác nhận như vậy, máy truyền sẽ không tiếp tục với chương trình.

4.2.2. Giao tiếp không đồng bộ nối tiếp

Đây là phương thức giao tiếp phổ biến nhất trong các hệ thống máy vi tính. Trong phương pháp này, đường giao tiếp là tối thiểu và có thể chỉ bao gồm hai hoặc ba dây. Có thể truyền và nhận thông qua đường dây điện thoại (với sự trợ giúp của một giao diện được gọi là modem) và thậm chí thông qua kết nối không dây.

Giao tiếp nối tiếp là một phương pháp trong đó một byte gồm 8 bit được biên dịch thành một chuỗi các xung nối tiếp, các số “0” và “1”, được truyền qua đường giao tiếp. Máy nhận biết khoảng thời gian của mỗi xung do máy truyền truyền đi. Trong giao tiếp không đồng bộ nối tiếp, có một vấn đề trong việc xác định điểm bắt đầu của mỗi byte. Vì thế trình tự sau đây đã được xác định.

Bit bắt đầu (Start bit):

Trạng thái bình thường của đường truyền là "high". Trước khi mỗi byte được truyền dưới dạng nối tiếp, một bit '0' phải được truyền trong cùng một khoảng thời gian cần thiết cho việc truyền từng bit khác. Đây được gọi là "bit bắt đầu". Máy nhận xác định điểm bắt đầu truyền của một ký tự truyền bằng cách xác định sự chuyển đổi từ '1' thành '0'.

Các bit dữ liệu:

Khi kết thúc quá trình truyền bit bắt đầu, các bit dữ liệu được truyền lần lượt. Thời gian truyền của mỗi bit bằng thời gian truyền của các bit khác. Vì máy nhận biết khi nào quá trình truyền bắt đầu, nên nó có thể định thời gian lấy mẫu các bit dữ liệu để khắc phục vấn đề quá trình chuyển tiếp.

Bit chẵn lẻ:

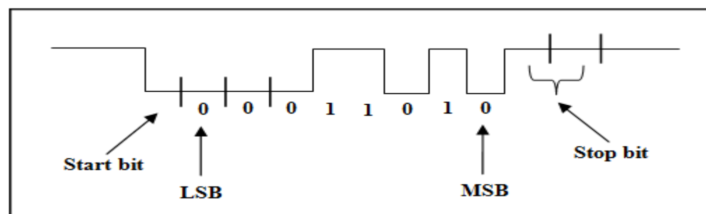
Đôi khi chúng ta sử dụng bit thứ tám của các bit dữ liệu làm bit chẵn lẻ, nó được sử dụng bởi máy nhận để kiểm tra độ chính xác của dữ liệu nhận được bởi chính nó. Giá trị của bit ('0' hoặc '1') được xác định theo số lượng bit có giá trị 1 trong byte dữ liệu. Có hai cách để xác định điều này: "even parity" (chẵn) và "odd parity" (lẻ). Trong "even parity", số lượng bit có giá trị 1 bao gồm cả bit chẵn lẻ phải là số chẵn. Ví dụ, nếu có 3 bit có giá trị 1 trong một byte, thì máy truyền xác định bit chẵn lẻ là '1'. Nếu có 4 bit có giá trị 1, thì bit chẵn lẻ sẽ là '0'.

Trong "odd parity", số bit có giá trị 1 bao gồm cả bit chẵn lẻ phải là số lẻ.

Bit dừng (Stop bit):

Vào cuối mỗi byte, các bit logic 1 được truyền đi (thường là 2). Các bit này được sử dụng để chuyển đường truyền về trạng thái bình thường trong một khoảng thời gian, điều này cho phép máy nhận thực hiện xử lý sơ cấp thông tin do nó thu thập và đồng bộ lại khi bắt đầu truyền ký tự tiếp theo.

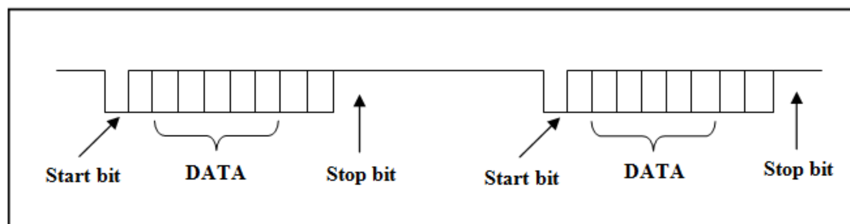
Việc truyền một ký tự đơn (58H) sẽ như sau:



Hình 2. 99

Tốc độ truyền được đo bằng đơn vị baud, là các bit được truyền trong một giây. Một cách truyền khác là "bit trên giây", nghĩa là tổng các bit dữ liệu được truyền trong một giây. Ví dụ, nếu chúng ta truyền với tốc độ 10 ký tự mỗi giây, tốc độ truyền sẽ là 110. Mỗi ký tự yêu cầu 11 bit truyền cho quá trình truyền của nó (bao gồm cả bit bắt đầu và bit dừng). Tốc độ này cũng bằng 80 bit/giây (bit dữ liệu).

Để kết luận lại, giao tiếp nối tiếp không đồng bộ được mô tả như trong hình sau:



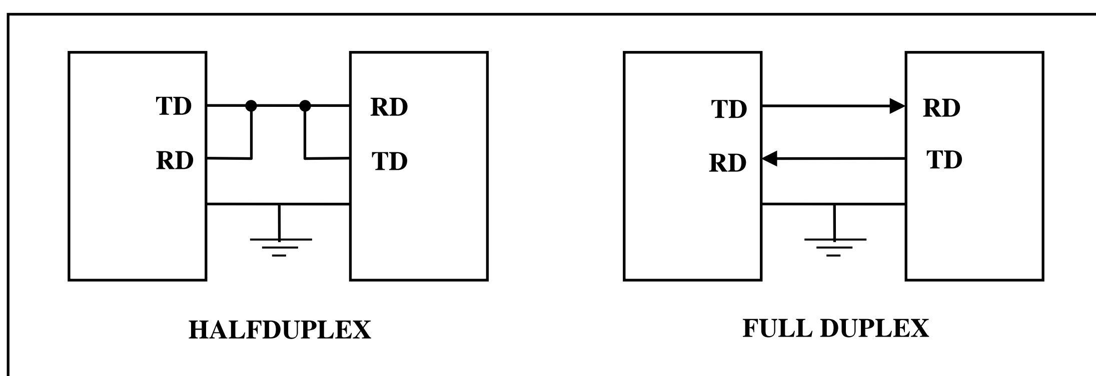
Hình 2. 100

Máy tính truyền chuyển đổi một ký tự từ dạng song song của nó (dưới dạng số nhị phân) thành dạng nối tiếp, rồi truyền nó. Máy tính nhận sẽ biên dịch ngược lại từ dạng nối tiếp sang dạng song song.

Máy nhận cần biết tốc độ truyền và số lượng bit dữ liệu trong byte được truyền (không phải lúc nào cũng là 7). Nó cũng cần phải biết liệu bit thứ tám là chẵn hay lẻ, hay là bit có giá trị thấp, cũng như cần phải biết số của các bit dừng.

Trong giao tiếp giữa các máy tính - một máy tính truyền và máy kia nhận. Thông thường, cả hai máy tính đều có khả năng truyền và nhận. Mỗi máy tính có đầu ra truyền dữ liệu TD (Transmit Data) và đầu vào nhận dữ liệu RD (Receive Data).

Trong giao tiếp, có hai hình thức kết nối chính. Một được gọi là: "Full duplex Communication" – Giao tiếp song công toàn phần (Full duplex).



Hình 2. 101

Hình thức kết nối thứ hai được gọi là: "Half duplex Communication" – Giao tiếp bán song công (Half duplex), và nó chỉ sử dụng hai dây kết nối. Trong giao tiếp bán song công, một máy tính chuyển từ trạng thái nhận sang trạng thái truyền phải đảm bảo rằng máy tính kia đã hoàn tất quá trình truyền và nó đã xóa đường truyền.

Thông thường, ở đầu vào và đầu ra của đường truyền giao tiếp, có các thành phần dẫn động, cho phép truyền tín hiệu trên một khoảng cách xa. Có nhiều phương pháp kết nối khác nhau giữa các máy tính. Phổ biến nhất là RS232, RS422 và vòng lặp dòng 20mA.

Quá trình nhận thông tin nối tiếp và chuyển đổi của nó thành dạng song song hoạt động theo phương thức sau: Máy tính nhận lấy mẫu đường ngõ vào RD và chờ bit bắt đầu, tức là kéo đường này về '0'. Ngay khi nhận thấy sự chuyển đổi này, nó sẽ đợi một khoảng thời gian bằng nửa thời gian của một bit, và sau đó lấy mẫu lại đường ngõ vào. Nếu đường ngõ vào vẫn là '0', điều đó có nghĩa là bit bắt đầu đã được nhận. Bây giờ nó lấy mẫu đường ngõ vào của một bit theo khoảng thời gian của một bit, ứng theo số lượng bit dữ liệu.

Trong khi lấy mẫu, các bit được đẩy lần lượt (LSB đang được nhận trước) vào một thanh ghi dịch. Vào cuối quá trình, thanh ghi dịch chứa byte được truyền, byte này có thể đọc được ở dạng song song.

Quá trình này được thực hiện với sự trợ giúp của một thiết bị phần cứng được gọi là UART (Universal Asynchronous Receiver Transmitter) - Máy phát thu không đồng bộ đa năng.

Khi UART xác định bit START, nó sẽ thu thập tất cả các bit DATA vào một thanh ghi đệm nhất định và sau đó tạo ra một yêu cầu ngắt để báo cho CPU biết rằng một byte đang đợi trong thanh ghi đệm.

Một số UART cũng có thể làm việc ở chế độ đồng bộ, có nghĩa là chúng chỉ bắt đầu thu thập dữ liệu sau khi nhận được một byte hoặc word nhất định. Chúng được gọi là USART (Universal Synchronous and Asynchronous Receiver Transmitter – Máy phát thu đồng bộ/ không đồng bộ đa năng).

4.2.3. Mã ASCII

Mã ASCII là một mã tiêu chuẩn quốc tế được sử dụng để trao đổi thông tin giữa các thiết bị đầu vào/đầu ra (như các loại máy in, bàn phím, bộ nhớ ngoài) và máy tính, cũng như giữa các máy tính với nhau. Tên ASCII là viết tắt của: American Standard Code for Information Interchange - Chuẩn Mã Trao đổi Thông tin Hoa Kỳ).

Mỗi ký tự (chữ cái, chữ số hoặc ký hiệu khác) được gán một số nhị phân đã được thống nhất trước, theo đó nó được thể hiện trong bảng ASCII. Ví dụ, nếu chúng ta muốn một máy in in ra chữ A, nó phải được nạp bằng số nhị phân 01000001 hoặc 41_{16}.

Sau đây là một bảng với các ký tự khác nhau và mã ASCII tương ứng của chúng, trong đó mã ASCII được thể hiện ở dạng nhị phân, thập lục phân và thập phân.

32 số đầu tiên (0-31) được sử dụng làm các mã đặc biệt cho hộp thoại giữa các máy tính như: Start Of Message (SOM), End Of Text (EOT), Carriage Return (CR), Line Feed (LF), v.v.

Dec.	Hex.	Binary	Char.					Dec.	Hex.	Binary	Char.
32	20	00100000	SPACE					64	40	01000000	@
33	21	00100001	!					65	41	01000001	A
34	22	00100010	"					66	42	01000010	B
35	23	00100011	#					67	43	01000011	C
36	24	00100100	$					68	44	01000100	D
37	25	00100101	%					69	45	01000101	E
38	26	00100110	&					70	46	01000110	F
39	27	00100111	'					71	47	01000111	G
40	28	00101000	(					72	48	01001000	H
41	29	00101001	)					73	49	01001001	I
42	2A	00101010	*					74	4A	01001010	J
43	2B	00101011	+					75	4B	01001011	K
44	2C	00101100	,					76	4C	01001100	L
45	2D	00101101	-					77	4D	01001101	M

Dec.	Hex.	Binary	Char.					Dec.	Hex.	Binary	Char.
46	2E	00101110	.					78	4E	01001110	N
47	2F	00101111	/					79	4F	01001111	O
48	30	00110000	0					80	50	01010000	P
49	31	00110001	1					81	51	01010001	Q
50	32	00110010	2					82	52	01010010	R
51	33	00110011	3					83	53	01010011	S
52	34	00110100	4					84	54	01010100	T
53	35	00110101	5					85	55	01010101	U
54	36	00110110	6					86	56	01010110	V
55	37	00110111	7					87	57	01010111	W
56	38	00111000	8					88	58	01011000	X
57	39	00111001	9					89	59	01011001	Y
58	3A	00111010	:					90	5A	01011010	Z
59	3B	00111011	;					91	5B	01011011	[
60	3C	00111100	<					92	5C	01011100	↓
61	3D	00111101	=					93	5D	01011101	←
62	3E	00111110	>					94	5E	01011110	→
63	3F	00111111	?					95	5F	01011111	_

4.2.4. Giao tiếp với PC

Chương trình sau sẽ làm nhấp nháy đèn LED L0 bằng cách để chương trình phản ứng khi thông báo 'hear?' được nhận.

```
Int main(void
{
NVIC_SetVectorTable(NVIC_VectTab_FLASH,
INTERUPT_VECTOR_START);
USART1_Interupt_Init();
__enable_irq(); //(Enable Interrupts)
//USART1_IRQHandler() send echo to PC and also
//Wait until PC sendthe message 'h', 'e' ,'l', 'l', 'o'
//Get the string bytes in USART1_IRQHandler()
//If the message was received, USART1_IRQHandler() set rx_ix to RX_OK
Leds_Init();
while(1)
{
```

```
//blink L0 until Ext_flag will be 1 by the External_Interrupt
dly++;
if (dly == 800000)
{
led_L0_on;
}
if (dly >= 1600000)
{
led_L0_off;
dly=0;
}

//Check if message was received
if (rx_ix==RX_OK) //If message was received
{
delay_us(50);
rx_ix=0;//Allow to get new string in USART1_IRQHandler()
//Send to PC
sc_send( ' ' );
sc_send( ' ' );
sc_send( ' ' );
sc_send( 'H' );
sc_send( 'i' );
sc_send( ' ' );
sc_send( ' ' );
sc_send( ' ' );
while((USART1->SR & USART_SR_TC) == 0);//Wait until the
Transmission_Complete flag is set
USART1->SR &= ~USART_SR_TC;//Reset the Transmission Complete flag
}
}
}
NVIC_SetVectorTable(NVIC_VectTab_FLASH,
INTERUPT_VECTOR_START);
```

Khai báo cho CPU nơi đặt các đoạn chương trình ngắt trong bộ nhớ. Bảng này được gọi là bảng điều khiển ngắt vectơ lồng nhau NVIC (Nested Vector Interrupt Control).

USART1_Interupt_Init(); khởi tạo đoạn chương trình ngắt USART.

Đoạn chương trình này khai báo như sau:

- GPIOA PA9 là TX và PA10 là RX của USART.
- Kích hoạt USART
- Tốc độ truyền 115000 baud

USART nhận một yêu cầu ngắt khi nhận được một ký tự. Lệnh sau sẽ kích hoạt các yêu cầu ngắt.

__enable_irq(); //(Enable Interrupts)

Leds_Init (); //initializes the LEDs output routine.

Yêu cầu ngắt USART được xử lý bởi đoạn chương trình USART1 IRQ Handler bên dưới.

Đoạn chương trình này gửi một lệnh echo của ký tự đã nhận đến thiết bị đầu cuối và phân tích các ký tự đã nhận.

Chỉ khi nhận được chuỗi 'hear?', nó mới thay đổi biến **rx_ix** thành giá trị **RX_OK**.

Trong khi làm nhấp nháy đèn LED L0, chương trình chính kiểm tra biến **rx_ix**. Khi **RX_OK** được tìm thấy, chương trình sẽ gửi word '**Hi**' đến PC.

Chương trình xóa biến **rx_ix**, đợi quá trình truyền kết thúc và đặt lại cờ hoàn tất quá trình truyền để cho phép nhận một chuỗi khác.

Sau đây là đoạn chương trình USART IRQ Handler.

```
void USART1_IRQHandler (void)//Get from PC
{
unsigned char sc_byte;
sc_byte = USART1->DR;//Take the byte from DATA_REGISTER
if (sc_byte == '\r')
{
USART1->DR = '\r';//Send carriage return
while((USART1->SR & USART_SR_TXE) == 0);//Wait until Transmit Data Register Empty
USART1->DR = '\n';//Send new line
}
else
{
USART1->DR = sc_byte;//Send echo to bord
}
switch(rx_ix)
```

```
{
case 0:
if (sc_byte=='h') //Increaserx_ix only if first byte is 'h'
{
rx_ix++;
}
break;
case 1:
if (sc_byte=='e') //Increaserx_ix only if second byte is 'e'
{//Else, start to get from case 0
rx_ix++;
}
else rx_ix=0;
break;
case 2:
if (sc_byte=='l') //Increaserx_ix only if next byte is 'l'
{//Else, start to get from case 0
rx_ix++;
}
else rx_ix=0;
break;
case 3:
if (sc_byte=='l') //Increaserx_ix only if next byte is 'l'
{//Else, start to get from case 0
rx_ix++;
}
else rx_ix=0;
break;
case 4:
if (sc_byte=='o')      //Only if next byte is 'o'
{
rx_ix=RX_OK;//string is ready, the main() is waiting...(RX_OK==99)
}
else rx_ix=0;//Else, start to get from case 0
break;
```

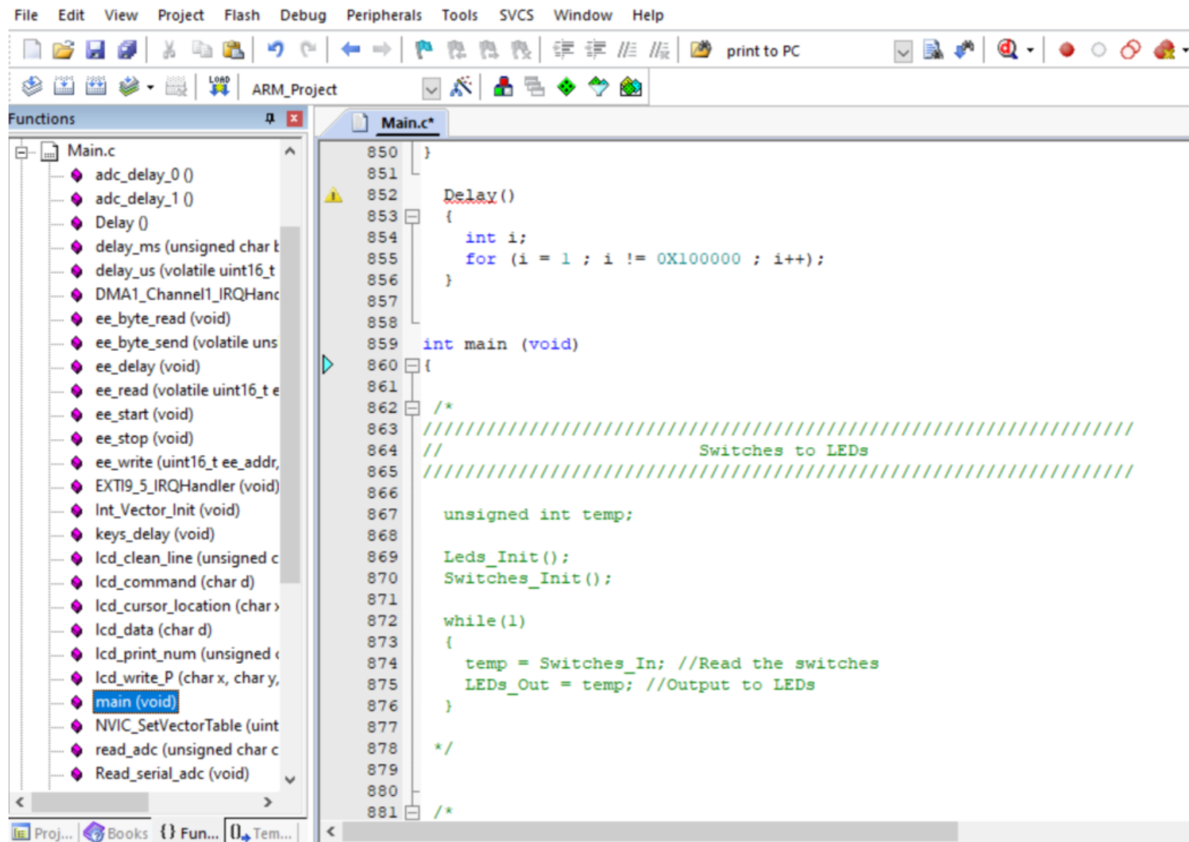
}

}

Trình tự thực hiện:

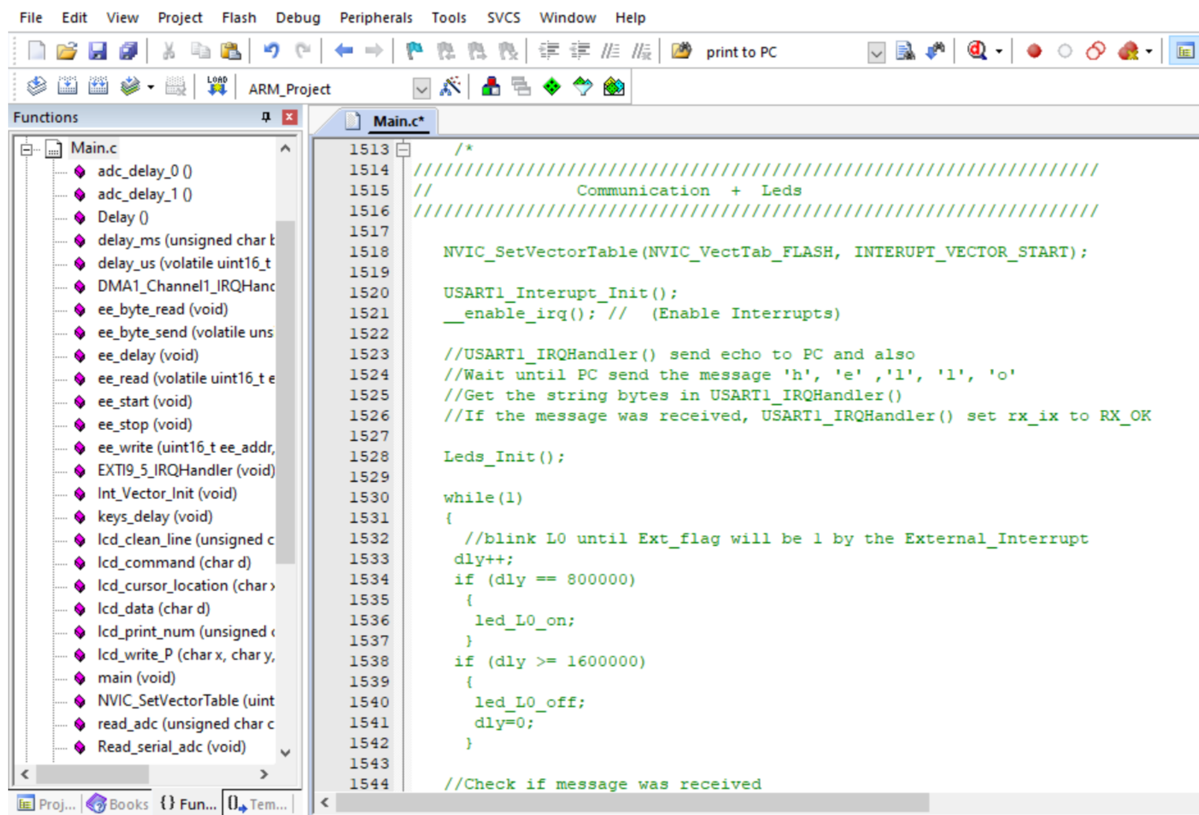
1. Vào thư viện **ARM_Project** và nhấp đúp vào tệp **ARM_Project.uvprojx**. Theo tài liệu này, đường dẫn đến tệp ARM_Project.uvprojx là“C:\Courses\3192\S-ARM_V3\ARM_Project”

2. Kiểm tra xem **main.c** có đang mở như trong màn hình sau đây không:



Hình 2. 102

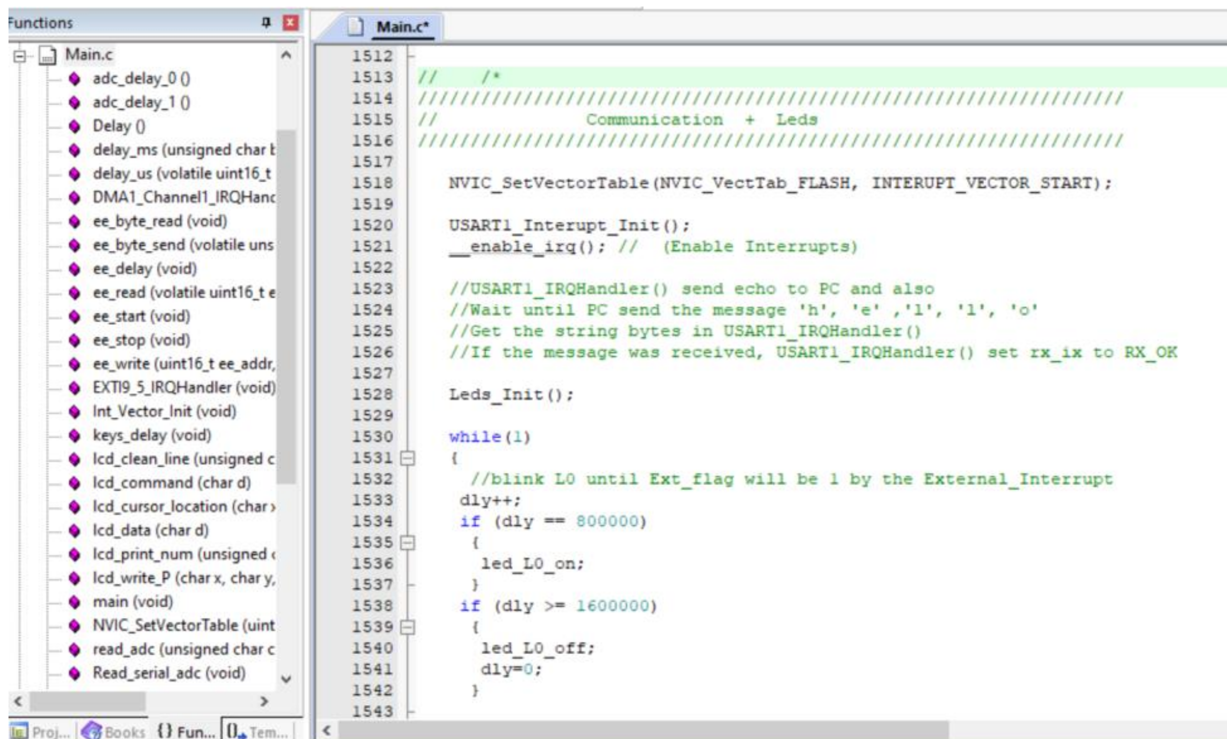
3. Cuộn xuống chương trình **main()** cho đến khi bạn nhận được màn hình sau:



Hình 2. 103

Phần này chứa chương trình **Communication** + **LEDs**.

4. Kích hoạt chương trình bằng cách sửa hai ký hiệu giới hạn đoạn chương trình thành đoạn ghi chú bằng cách thêm hai dấu gạch chéo vào đầu mỗi ký hiệu giới hạn và bạn sẽ nhận được màn hình sau:



Hình 2. 104

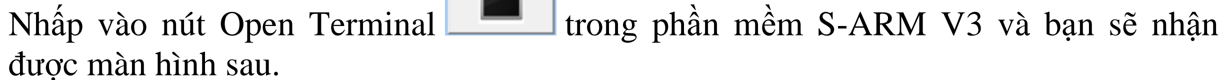
5. Quan sát và phân tích chương trình.

6. Kích hoạt EITPS-3192.

7. Lưu và biên dịch (sửa lỗi nếu có và biên dịch lại).

Nhấn RST trước khi tải xuống, sau đó tải xuống và chạy chương trình.

8. Để giao tiếp với PC, chúng ta phải chuyển PC thành một thiết bị đầu cuối. Thiết bị đầu cuối truyền mã ASCII của bất kỳ phím nào được nhấn và hiển thị các ký tự nhận được.



Nhấp vào nút Open Terminal trong phần mềm S-ARM V3 và bạn sẽ nhận được màn hình sau.

Hình 2. 105

9. Nhấp vào màn hình thiết bị đầu cuối để xem con trỏ nhấp nháy.

10. Nhấp vào phím 'a'.

Hai chữ cái 'a' sẽ xuất hiện trên màn hình. Một là từ bàn phím đầu cuối và một là echo được gửi bởi đoạn chương trình IRQ Handler.

11. Bấm vào các chữ cái **hello**.

Ngay sau dấu '?', thông báo **'Hi'** sẽ xuất hiện trên màn hình.

12. Thử gõ các chữ cái khác nhau.

13. Nhấn RST để dừng chương trình đang chạy.

14. Thay đổi đoạn chương trình IRQ Handler để tránh sending và echo.

15. Lưu và biên dịch (nếu có sai sót thì sửa lỗi và biên dịch lại).

Trước khi tải xuống, hãy nhấn RST, sau đó tải xuống và chạy chương trình.

16. Kiểm tra chương trình đang chạy.

17. Nhấn RST để dừng chương trình đang chạy.

18. Kích hoạt các dấu '/*   */' bằng cách xóa hai dấu gạch chéo ở đầu mỗi dòng.

Chương trình chuyển sang màu xanh lục.

Liên hệ hỗ trợ kỹ thuật:
CTCP ĐIỆN TỬ CHUYÊN DỤNG HANEL
Địa chỉ: Tầng 11 tòa nhà Diamond Flower, số 48 Lê Văn Lương, Thanh Xuân, Hà Nội
Hotline: 0942195862