

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN ĐIỆN TỬ - VIỄN THÔNG

TÀI LIỆU HƯỚNG DẪN

**XÂY DỰNG MỘT SỐ ỨNG DỤNG CƠ BẢN
VỚI VI ĐIỀU KHIỂN AVR**

(Dùng cho học phần Đồ án II)

MỤC LỤC

1. THÔNG TIN CHUNG	1
<i>1.1 Giới thiệu tổng quan.....</i>	<i>1</i>
<i>1.2 Mạch Kit cho VĐK họ AVR.....</i>	<i>1</i>
<i>1.3 Ngôn ngữ lập trình và các công cụ phần mềm.....</i>	<i>2</i>
2. CẤU HÌNH MẠCH KIT.....	3
<i>2.1 Cấu trúc mạch.....</i>	<i>3</i>
<i>2.2 Các thông số chính</i>	<i>5</i>
<i>2.3 Mạch nạp mã nguồn.....</i>	<i>5</i>
<i>2.4 Màn hình LCD và mô-đun UART-USB.....</i>	<i>6</i>
3. THỰC HÀNH LẬP TRÌNH CƠ BẢN CHO VĐK	7
<i>3.1 Tạo Project và nạp thử mã máy cho VĐK</i>	<i>7</i>
3.1.1 Mục tiêu	7
3.1.2 Công việc cần làm.....	7
<i>3.2 Điều khiển cổng ra số.....</i>	<i>10</i>
3.2.1 Mục tiêu	10
3.2.2 Công việc cần làm.....	11
3.2.3 Hướng dẫn thực hiện.....	12
<i>3.3 Đọc trạng thái logic đầu vào số.....</i>	<i>15</i>
3.3.1 Mục tiêu	15
3.3.2 Công việc cần làm.....	16
3.3.3 Hướng dẫn thực hiện.....	16
<i>3.4 Đo điện áp tương tự và hiển thị kết quả bằng màn hình LCD</i>	<i>19</i>
3.4.1 Mục tiêu	19
3.4.2 Công việc cần làm.....	19
3.4.3 Hướng dẫn thực hiện.....	19
<i>3.5 Giao tiếp với máy tính qua chuẩn UART-USB.....</i>	<i>22</i>
3.5.1 Mục tiêu	22

3.5.2 Công việc cần làm.....	22
3.5.3 Hướng dẫn thực hiện.....	22
4. VẬN DỤNG KIẾN THỨC VÀO ỨNG DỤNG THỰC TẾ	23
<i>4.1 Mục tiêu.....</i>	<i>23</i>
<i>4.2 Một số đề tài và gợi ý cách thực hiện.....</i>	<i>24</i>
<i>4.3 Yêu cầu về kết quả và báo cáo.....</i>	<i>25</i>

1. THÔNG TIN CHUNG

1.1 Giới thiệu tổng quan

Học phần Đồ án II được thiết kế để củng cố và nâng cao năng lực chuyên môn cho sinh viên; giúp liên kết các khối kiến thức về điện tử tương tự, điện tử số, kỹ thuật vi xử lý, xử lý số tín hiệu, thông tin số, v.v. nhằm hoàn thiện khả năng vận dụng kiến thức vào thực tế.

Trong học phần này, nội dung công việc cần thực hiện gồm: làm quen công cụ thiết kế mạch điện, thực hành lập trình phần cứng, và xây dựng một ứng dụng cơ bản với các vi mạch khả trình. Để đảm bảo tiến độ công việc, sinh viên được hỗ trợ một số phương tiện cơ bản để triển khai công việc trên nền tảng xây dựng sẵn, có thể kế thừa.

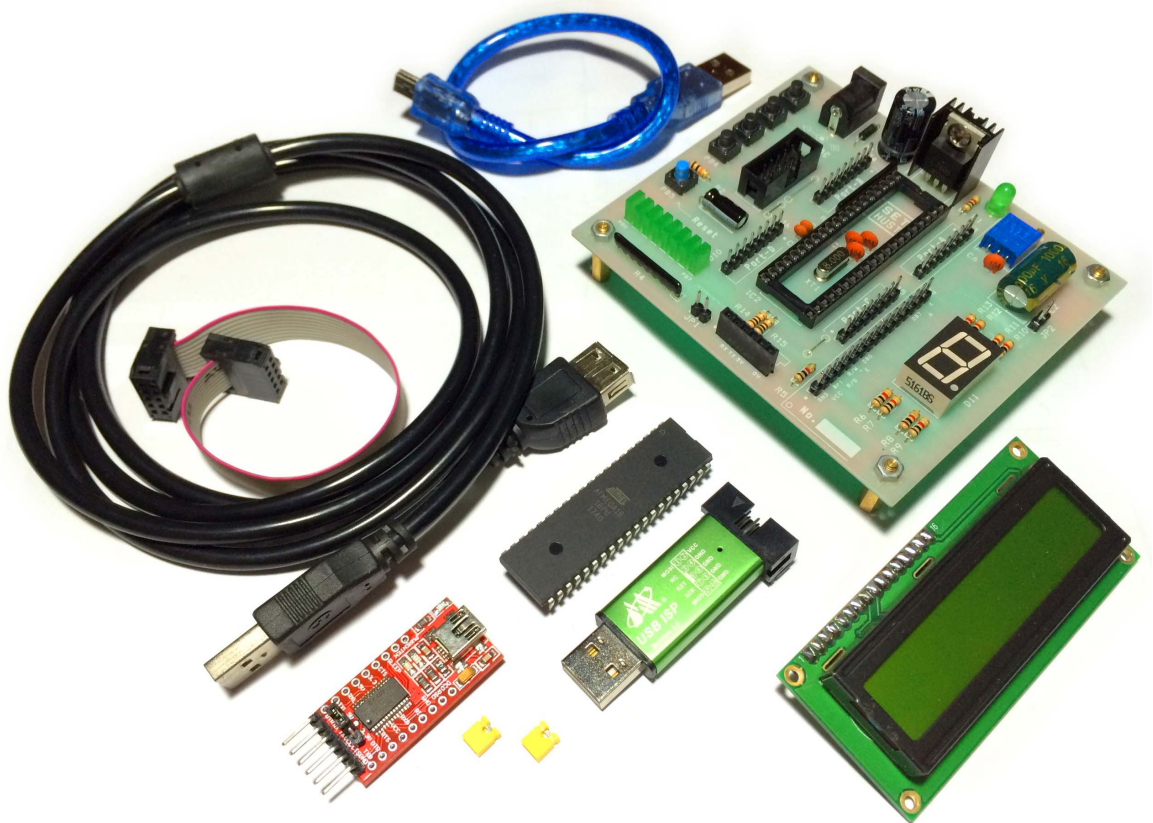
Tài liệu này giúp sinh viên tiếp cận nhanh với vi điều khiển (VĐK) thông qua việc xây dựng một số ứng dụng với họ AVR trên nền tảng mạch Kit cơ bản. Nội dung của tài liệu được chia thành bốn phần chính và một phụ lục để người đọc tiện theo dõi.

1.2 Mạch Kit cho VĐK họ AVR

AVR là một dòng VĐK 8 bit khá mạnh và thông dụng tại thị trường Việt Nam. Với tốc độ xung nhịp tới 16 MHz, bộ nhớ chương trình tối đa tới 256 kB, và rất nhiều chức năng ngoại vi tích hợp sẵn, VĐK họ AVR có thể đáp ứng tốt cho nhiều ứng dụng trong thực tế, từ đơn giản đến phức tạp.

Kit phát triển sử dụng trong học phần Đồ án II (xem Hình 1) được Viện Điện tử - Viễn thông thiết kế riêng để đảm bảo tính hiệu quả trong quá trình đào tạo. Với bộ kit này, sinh viên có thể thử nghiệm các ứng dụng cơ bản như:

- Điều khiển cổng ra số, với LED đơn và LED 7 thanh
- Đọc trạng thái logic đầu vào số, từ bàn phím và giấc cảm mở rộng
- Đo điện áp tương tự, với biến trở vi chỉnh và bộ ADC 10-bit
- Điều khiển màn hình tinh thể lỏng, với màn hình LCD dạng text
- Giao tiếp với máy tính qua chuẩn UART ↔ USB
- Thử nghiệm các ngắt ngoài, thử khả năng điều chế độ rộng xung
- Nhiều ứng dụng điều khiển các chức năng tích hợp sẵn trong VĐK như: vận hành các bộ định thời (Timer) và bộ đếm (Counter), đọc ghi EEPROM, lập trình các ngắt chương trình, thiết lập Watchdog, v.v.



Hình 1 - Mạch Kit phát triển và các phụ kiện

Mặt khác, bằng việc kết nối với các mô-đun mở rộng, sinh viên có thể thử nghiệm các ứng dụng phức tạp hơn như:

- Đo tham số môi trường cơ bản: nhiệt độ, độ ẩm, ánh sáng, v.v.
- Điều khiển tải cơ bản: đèn báo, van điện từ, động cơ DC, động cơ bước, v.v.
- Điều khiển hiển thị cơ bản: LED ma trận, LCD ma trận, màn hình cảm ứng, v.v.
- Giao tiếp I2C và SPI: IC thời gian thực, IC EEPROM, cảm biến gia tốc, v.v.
- Ứng dụng tổng hợp: đo và duy trì sự ổn định các tham số môi trường; số hóa và xử lý tín hiệu âm thanh; điều khiển robot hoặc xe tự hành; v.v.

1.3 Ngôn ngữ lập trình và các công cụ phần mềm

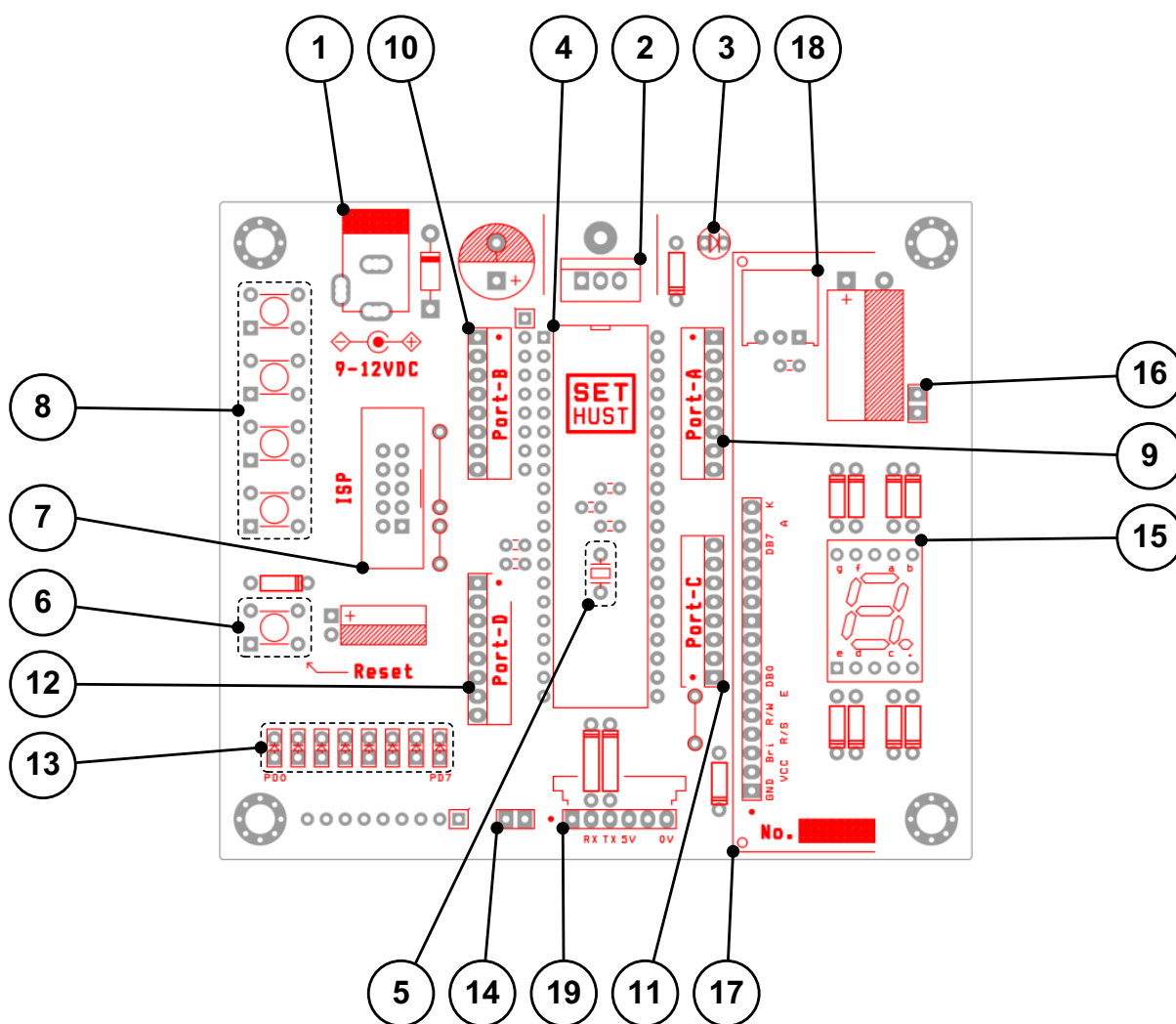
Để học tập và làm việc với VDK AVR, sinh viên có thể sử dụng ngôn ngữ C hoặc Assembly, viết trên một số môi trường phát triển khác nhau. Các bài thực hành trong tài liệu này được xây dựng với ngôn ngữ C; trên môi trường soạn thảo và biên dịch là sự kết hợp của AVR Studio 4 (phiên bản 4.19 build 730, dung lượng 124 MB) với WinAVR (phiên bản 20100110, dung lượng 27.5 MB). Phần mềm nạp mã máy là

PROGISP (phiên bản 1.72, dung lượng khoảng 3-4 MB). Phần mềm thu nhận dữ liệu từ cổng USB hay COM ảo là Terminal (phiên bản 1.9b, dung lượng 243 KB). Tất cả các công cụ này được đặt sẵn trong thư mục *Software* của gói học liệu do giảng viên cung cấp. Sinh viên có thể sử dụng các phiên bản mới hơn hoặc các phần mềm có chức năng tương tự nhưng cần chú ý khả năng tương thích với phần cứng.

2. CẤU HÌNH MẠCH KIT

2.1 Cấu trúc mạch

Để thử nghiệm được nhiều chức năng ngoại vi với phần cứng tối thiểu, mạch Kit được thiết kế như với cấu trúc như trên Hình 2 và chức năng của các linh kiện quan trọng được nêu rõ trong Bảng 1. Sơ đồ nguyên lý và layout chi tiết của mạch Kit được thể hiện trong phụ lục kèm theo.



Hình 2 - Cấu trúc mạch Kit

Bảng 1 - Linh kiện quan trọng của mạch Kit và chức năng tương ứng

STT	Tên linh kiện	Chức năng
1	Giắc cắm nguồn	Nhận nguồn điện 9-12 VDC cấp cho mạch Kit
2	IC ổn áp 7805	Hạ 9-12 VDC xuống 5 VDC và giữ ổn định mức điện áp này để cấp cho toàn mạch
3	LED báo nguồn	Báo nguồn (sáng: có nguồn 5 VDC, tắt: mất nguồn)
4	VĐK họ AVR	Điều khiển hoạt động của toàn mạch theo mã nguồn do người dùng lập trình và nạp xuống
5	Thạch anh	Quyết định tần số xung nhịp cấp cho VĐK
6	Nút ấn Reset	Khởi động lại VĐK
7	Giắc ISP	Kết nối mạch nạp (có bán sẵn) để nạp mã nguồn cho VĐK
8	Nhóm 4 phím ấn	Nhận lệnh điều khiển từ người sử dụng
9	Giắc cắm 8 chân	Nối tới 8 chân vào/ra đa năng (ứng với Port-A) của VĐK
10	Giắc cắm 8 chân	Nối tới 8 chân vào/ra đa năng (ứng với Port-B) của VĐK
11	Giắc cắm 8 chân	Nối tới 8 chân vào/ra đa năng (ứng với Port-C) của VĐK
12	Giắc cắm 8 chân	Nối tới 8 chân vào/ra đa năng (ứng với Port-D) của VĐK
13	Dây LED đơn	Báo trạng thái logic của 8 chân ở Port-D (sáng: mức logic 0, tắt: mức logic 1)
14	Jumper dây LED đơn	Cho phép hoặc vô hiệu hóa dây LED đơn
15	LED 7 thanh	Hiển thị số 0-9 và một vài ký tự do người dùng định nghĩa
16	Jumper LED 7 thanh	Cho phép hoặc vô hiệu hóa LED 7 thanh
17	Giắc cắm LCD	Kết nối màn hình LCD dạng text có bán sẵn. Loại phù hợp nhất là 1602 (16 ký tự \times 2 dòng)
18	Biến trở vi chỉnh	Điều chỉnh trơn và liên tục, từ 0 đến 5 VDC, mức điện áp tại đầu vào ADC0 của bộ ADC (chân PA0)
19	Giắc UART-USB	Kết nối mô-đun chuyển đổi UART-USB (còn gọi là COM-USB) có bán sẵn

2.2 Các thông số chính

Các thông số kỹ thuật của mạch Kit:

- Điện áp nguồn:
 - Tiêu chuẩn: 9-12 VDC
 - Giới hạn: 7-18 VDC
- Dòng điện tiêu thụ:
 - Khi không có mô-đun mở rộng, toàn bộ LED chỉ thị I/O tắt: 18 mA
 - Khi có LCD và mô-đun USB, các LED chỉ thị I/O bị vô hiệu hóa: 22 mA
 - Khi có LCD và mô-đun USB, toàn bộ LED chỉ thị I/O sáng: 80 mA
- Mạch có khả năng tự bảo vệ khi bị lắp ngược cực tính nguồn
- Mức logic các cổng I/O: TTL (5 V)
- Điện áp tương tự vào các chân ADC: từ 0 đến +5 V
- Loại VĐK được hỗ trợ: ATmega16, ATmega32, và tương đương
- Cổng I/O mở rộng: 4 giắc cắm (loại 8 chân) ứng với 4 Port (8 bit mỗi Port)
- Hỗ trợ màn hình LCD: dạng text, giao tiếp 8 bit hoặc 4 bit
- Hỗ trợ mô-đun USB: UART-USB hay COM-USB (mức 5 VDC)
- Xung nhịp tích hợp sẵn: thạch anh 8 MHz

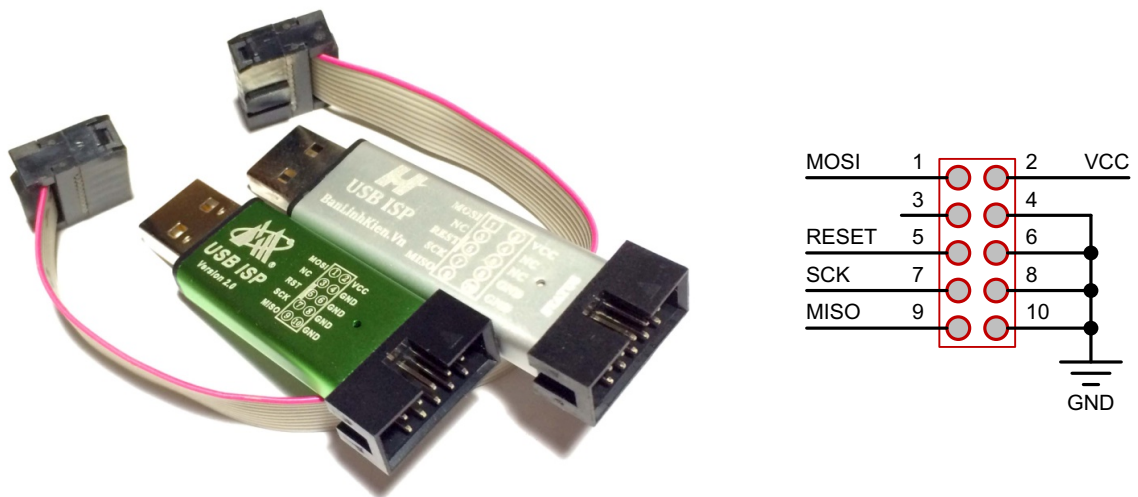


Các giá trị dòng điện trên được đo với KIT mạch mẫu. Các con số này có thể tăng giảm tùy theo: chế độ hoạt động của VĐK; loại LED sử dụng và giá trị các điện trở hạn dòng cho LED; số lượng và chủng loại mô-đun mở rộng kết nối tới cổng I/O của VĐK. Khi điện áp vào cao và dòng điện tiêu thụ lớn, IC ổn áp nguồn có thể bị quá nhiệt. Để đảm bảo an toàn, công thức sau nên được thỏa mãn:

$$[(\text{điện áp nguồn} - 5) \times \text{dòng điện tiêu thụ}] \leq 1 \text{ W}$$

2.3 Mạch nạp mã nguồn

Mạch nạp mã nguồn cho VĐK trong Kit là loại mạch nạp ISP thông dụng. Sinh viên có thể tìm thấy mạch nạp này tại hầu hết cửa hàng bán lẻ hay đại lý phân phối sản phẩm liên quan đến VĐK AVR. Hình 3 minh họa một vài loại mạch nạp ISP thông dụng tại Việt Nam và chuẩn kết nối ISP 10 chân tương ứng.



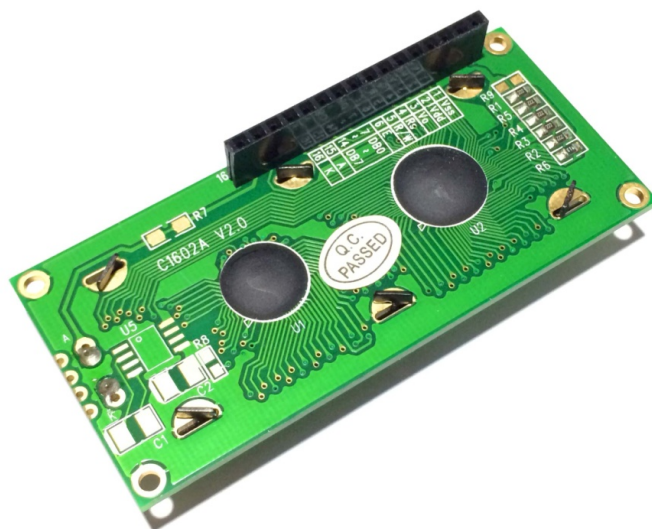
Hình 3 - Mạch nạp ISP chuẩn 10 chân



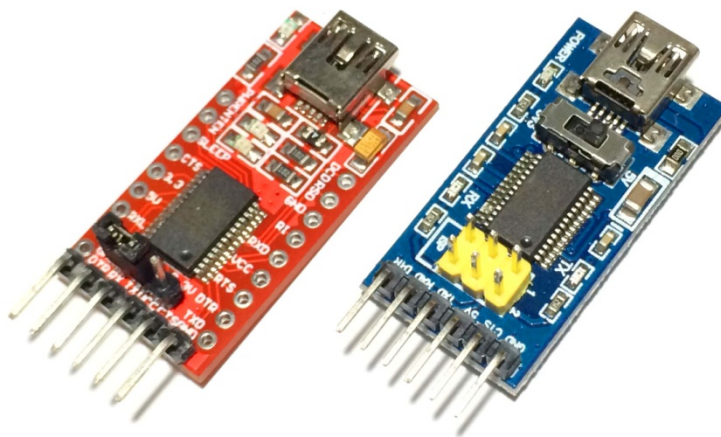
Nếu kết nối giữa mạch nạp ISP và mạch Kit quá dài, tín hiệu truyền tải có thể bị can nhiễu trong quá trình nạp mã nguồn. Sinh viên nên sử dụng cáp dẹt ngắn (khoảng 10-20 cm) kết hợp với một cáp kéo dài USB để đảm bảo tính linh hoạt, cơ động, nhưng vẫn tin cậy.

2.4 Màn hình LCD và mô-đun UART-USB

Mạch Kit được thiết kế để tương thích với màn hình LCD text và mô-đun chuyển đổi UART-USB (còn gọi là COM-USB) sử dụng chip FT232RL, như trên Hình 4 và Hình 5. Đây đều là các mô-đun ngoại vi quen thuộc và điển hình.



Hình 4 - Màn hình LCD 1602 và chuẩn kết nối 16 chân



Hình 5 - Mô-đun UART-USB sử dụng chip FT232RL

3. THỰC HÀNH LẬP TRÌNH CƠ BẢN CHO VDK

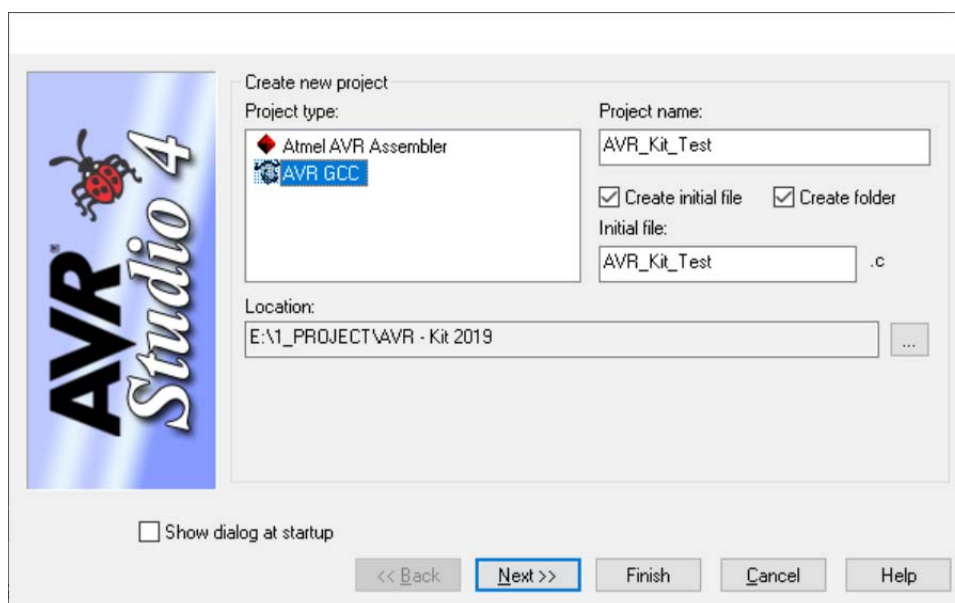
3.1 Tạo Project và nạp thử mã máy cho VDK

3.1.1 Mục tiêu

Giúp sinh viên làm quen với các công việc cơ bản sau:

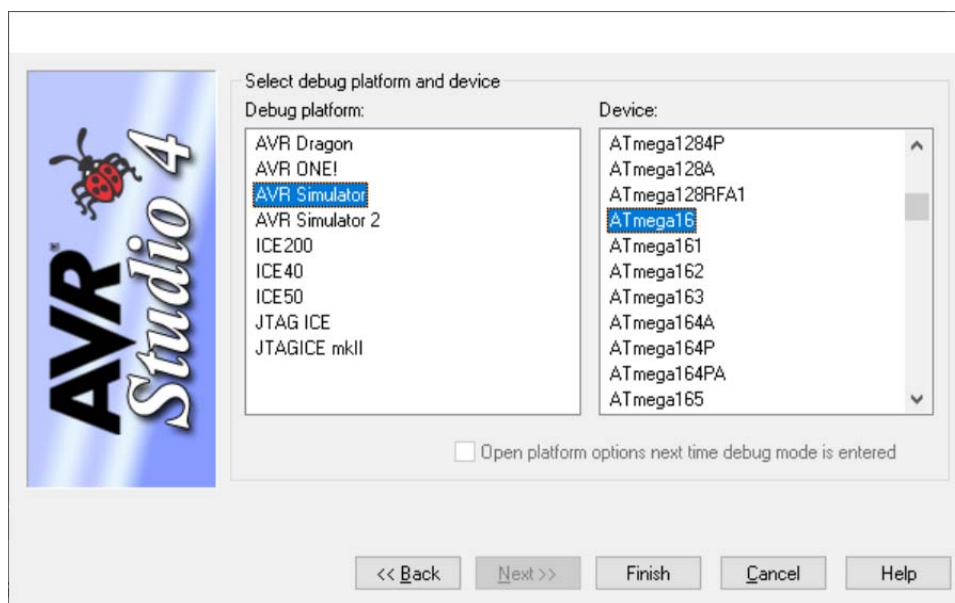
- Tạo ra và thiết lập một Project mới để lập trình.
- Dịch mã nguồn sang mã máy (mã hex).
- Chỉnh cấu hình phù hợp cho VDK, nạp mã, và kiểm tra kết quả.

3.1.2 Công việc cần làm



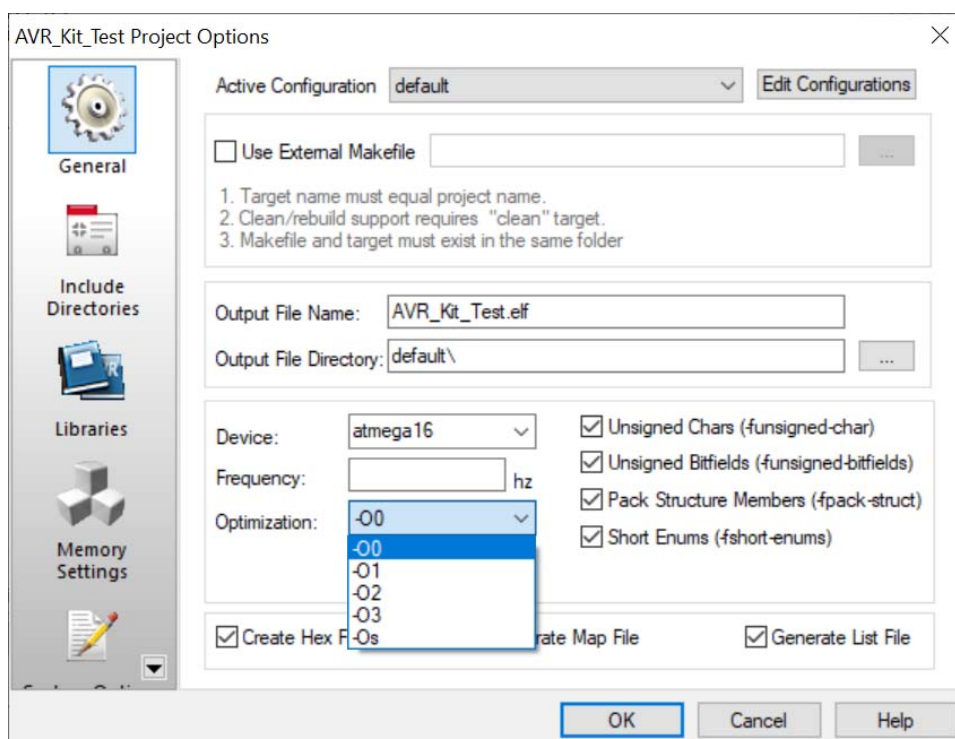
Hình 6 – Bước đầu tiên khi tạo một Project mới

- Trong AVR Studio 4, vào *Project > New Project* để tạo một Project chương trình có tên là *AVR_Kit_Test* với các tùy chọn như trên Hình 6 và Hình 7.

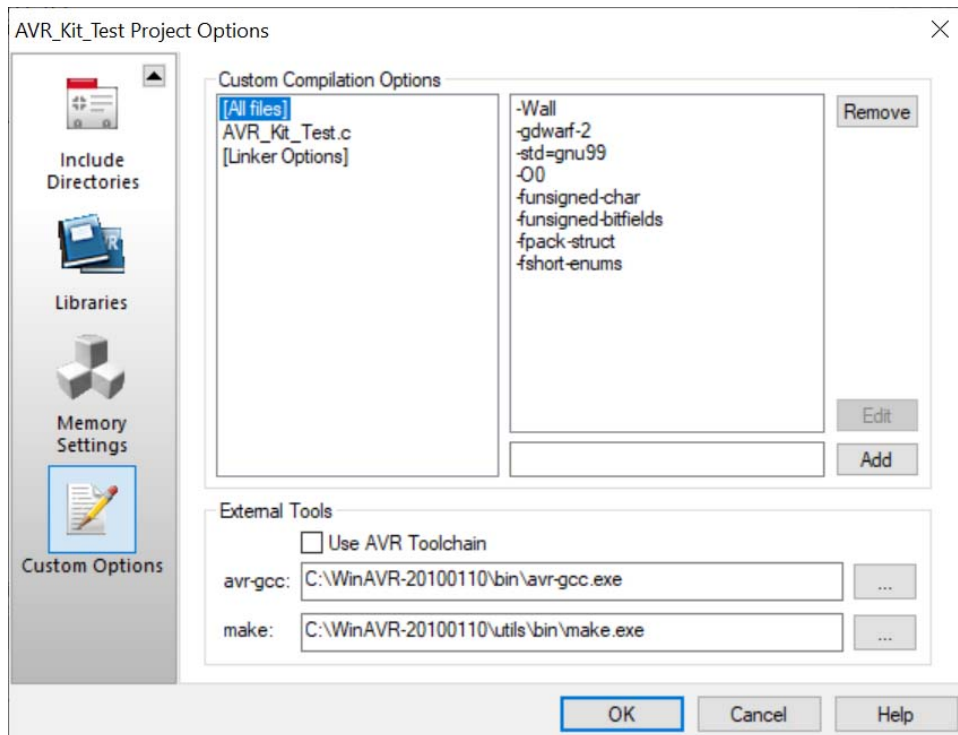


Hình 7 – Bước chọn Debug platform và chọn chip khi tạo một Project mới

- Trong Project vừa tạo, vào *Project > Configuration Options* để thiết lập một số tùy chọn quan trọng gồm: mức độ Optimization (xem Hình 8) và AVR Toolchain (xem Hình 9).



Hình 8 – Chỉnh mức độ Optimization cho Project mới



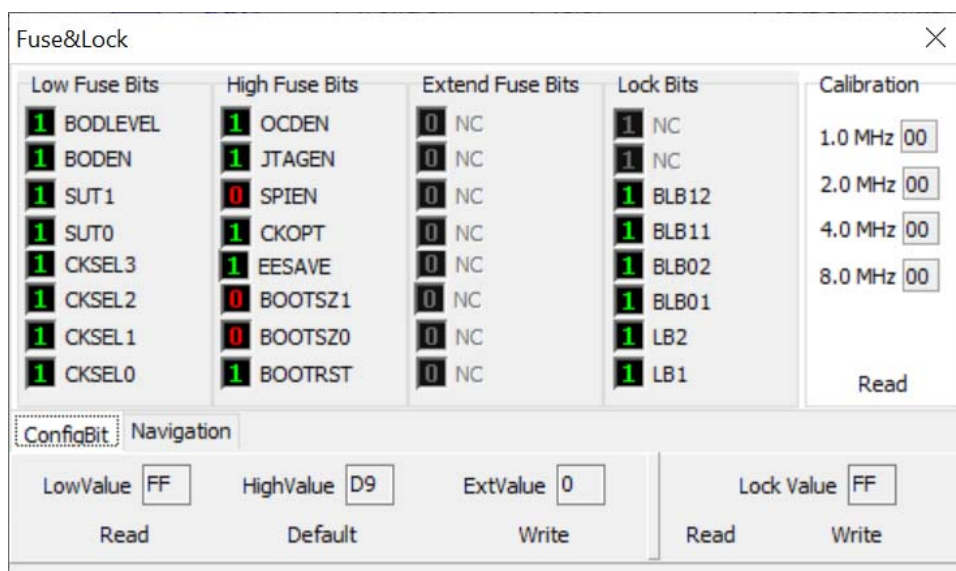
Hình 9 – Thiết lập Toolchain cho Project mới

- Copy đoạn mã nguồn mẫu từ file *Test.txt* (xem thư mục *Sample*) vào file *AVR_Kit_Test.c* trong Project vừa tạo. File *Test.txt* có nội dung như sau:

```
#include <avr\io.h>

int main()
{
    DDRD |= 0xFF;
    PORTD |= 0xAA;
    DDRC |= 0xFF;
    PORTC |= 0x00;
    return 0;
}
```

- Dịch đoạn mã nguồn trên sang mã máy bằng cách vào *Build > Build* hoặc click vào biểu tượng có hình tương ứng trên menu. Nếu không có lỗi gì xảy ra, file mã máy *AVR_Kit_Test.hex* sẽ được tạo ra trong thư mục *default* của Project.
- Sử dụng phần mềm PROGISP để chỉnh cấu hình Fuse bit cho VĐK như trên Hình 10 rồi nạp file *AVR_Kit_Test.hex* xuống VĐK. Nếu không có lỗi gì xảy ra, dãy 8 LED đơn trên mạch Kit sẽ sáng tắt xen kẽ nhau và LED 7 thanh sẽ hiện số 0. Sinh viên cũng có thể nạp một số mã máy mẫu trong thư mục *Sample* xuống VĐK để kiểm tra tính năng và các kết nối khi cần thiết.



Hình 10 – Thiết lập Fuse bit và Lock bit cho VĐK



Fuse bit là các bit rất quan trọng, quyết định chế độ hoạt động và giao diện nạp IC VĐK. Việc thiết lập sai các bit này có thể đưa VĐK vào chế độ đặc biệt, không thể nạp xóa và cũng không thể khôi phục lại như cũ bằng các mạch nạp thông thường. Vì vậy, sinh viên cần thao tác rất cẩn thận khi làm việc với Fuse bit.

Để tránh rủi ro, nên hạn chế việc thay đổi Fuse bit. Nếu việc điều chỉnh Fuse bit là bắt buộc, thực thi lệnh theo thứ tự sau để giảm thiểu nhầm lẫn: đọc (Read) giá trị Fuse bit hiện tại > chỉ sửa những bit cần thiết > rồi nạp (Write) giá trị mới xuống.

3.2 Điều khiển cổng ra số

3.2.1 Mục tiêu

Mục tiêu chung:

- Giúp sinh viên làm quen với việc viết mã nguồn, biên dịch/sửa lỗi mã nguồn, nạp mã máy và kiểm tra hoạt động của VĐK.
- Điều khiển trạng thái logic (0 và 1) cho các chân IC VĐK.

Cụ thể, sinh viên điều khiển LED sáng/tắt theo quy tắc sau:

- Với LED đơn: khi bật nguồn, toàn bộ LED tắt. Sau mỗi 0.5 s, có thêm một LED sáng (từ trái sang phải) để tạo thành dải sáng có độ dài tăng dần. Sau khi dải sáng đạt độ dài cực đại, toàn bộ LED tắt và quy trình được lặp lại từ đầu.

- Với LED 7 thanh: khi bật nguồn, LED 7 thanh hiện số 0. Sau mỗi 0.5 s, số đếm trên LED 7 thanh tăng thêm 1 đơn vị và dấu chấm trên LED đảo trạng thái (nhấp nháy). Sau khi tăng đến 9, số đếm quay lại giá trị 0 và quy trình được lặp lại từ đầu.

3.2.2 Công việc cần làm

Sinh viên cần thực hiện các công việc sau:

- Tạo mới Project *AVR_Kit_Test* hoặc chỉnh sửa Project đã có sẵn trong phần 3.1. Viết chuỗi lệnh tuần tự vào file *AVR_Kit_Test.c* để:
 - Khai báo các thư viện sẽ sử dụng (các file *.h), định nghĩa các hằng số, khai báo các biến toàn cục, và gọi hàm *main()* – là hàm bắt buộc phải xuất hiện.
 - Lập trình hàm *main()*: chứa các lệnh đơn hoặc các lệnh gọi chương trình con để điều khiển hoạt động của IC VĐK. Trong Project này, sinh viên cần gọi hai chương trình con *INIT()* và *PORT()* từ file *thu_vien_rieng.h* sẽ tạo ra bên cạnh file *AVR_Kit_Test.c* đã có.
- Tạo file mới và lưu lại với tên *thu_vien_rieng.h* trong cùng thư mục với file *AVR_Kit_Test.c*. File thư viện này chứa các chương trình con do người lập trình tự phát triển. Trong bài này, sinh viên tạo ra 4 chương trình con:
 - *INIT()*: là chương trình con dùng để khởi tạo trạng thái cho các chân I/O của VĐK.
 - *PORT()*: là chương trình con dùng để bật/tắt các LED thông qua việc điều khiển các PORT của VĐK.
 - *LED7_OUT(num)*: là chương trình con dùng để điều khiển LED 7 thanh hiển thị số theo biến *num* ($0 \leq num \leq 9$).
 - *DELAY_MS(mili_count)*: là chương trình con dùng để tạo ra các khoảng thời gian trễ, tính bằng mili giây, giữa các lần bật/tắt LED để dễ dàng quan sát hiệu ứng.
- Tiến hành biên dịch sang mã máy và nạp mã máy xuống IC VĐK.
- Sửa lỗi (nếu có) và quan sát các hiệu ứng LED khi hoàn thành.

3.2.3 Hướng dẫn thực hiện

Sinh viên thực hành viết chương trình bằng cách gõ lại các đoạn mã nguồn mẫu sau đây vào các file tương ứng. Các đoạn text nằm trên cùng một dòng sau `"/"` hoặc nằm giữa `"/*"` và `*/` là các lời giải thích cho mã nguồn. Các lời giải thích này không bắt buộc phải xuất hiện trong code nhưng sinh viên được khuyến khích sử dụng để rèn luyện thói quen chú thích rõ ràng khi lập trình.

❖ File *AVR_Kit_Test.c*

```
// Khai báo thư viện chuẩn
#include <avr\io.h>

// Định nghĩa hằng số FREQ = 8, là xung nhịp của hệ thống (tính bằng MHz)
#define FREQ 8

// Khai báo file thư viện riêng
#include "thu_vien_rheng.h"

// Lập trình hàm main()
int main()
{
    // Gọi hàm INIT() trong file thu_vien_rheng.h để khởi tạo
    INIT();

    // Gọi hàm PORT() trong file thu_vien_rheng.h để điều khiển LED
    PORT();

    // Lệnh return luôn xuất hiện ở cuối hàm main()
    return 0;
}
```

❖ File *thu_vien_rheng.h*

```
// Khai báo các nguyên mẫu hàm
void INIT();
void PORT();
void LED7_OUT(unsigned char num);
void DELAY_MS(unsigned int mili_count);

/*****
Hàm INIT() là hàm không có tham số và không trả lại giá trị, do người lập
trình tự xây dựng để khởi tạo trạng thái các PORT của vi điều khiển. Như
bất kỳ hàm tự xây dựng nào khác, người lập trình có thể đổi tên, chia nhỏ,
gộp to, hoặc thay đổi các lệnh trong hàm này theo dụng ý riêng khi lập
trình.
*****/
void INIT()
```



```

{
    // Khởi tạo trạng thái Output cho các chân nối tới các LED đơn
    DDRD |= 0xFF;

    // Khởi tạo trạng thái logic 1 cho các chân nối tới các LED đơn
    PORTD |= 0xFF;

    // Khởi tạo trạng thái Output cho các chân nối tới LED 7 thanh
    DDRC |= 0xFF;

    // Khởi tạo trạng thái logic 1 cho các chân nối tới LED 7 thanh
    PORTC |= 0xFF;
}

/*****
Hàm PORT() là hàm không có tham số và không trả lại giá trị, do người lập
trình tự xây dựng để điều khiển trạng thái logic 0/1 của các chân trong các
PORT của vi điều khiển. Trong mạch Kit này, trạng thái logic làm các đèn
LED sáng/tắt theo quy tắc: 0 - LED sáng, 1 - LED tắt.
*****/
void PORT()
{
    /* Khai báo các biến sẽ dùng tới trong hàm này */

    // Biến led_shift để điều khiển các LED đơn
    // Giá trị đầu là 255 = 0xFF = 0b11111111 -> tắt cả các LED đều tắt
    unsigned char led_shift = 255;

    // Biến đếm cho LED 7 thanh, giá trị đầu là 0
    unsigned char led_7_count = 0;

    // Vòng for giúp các LED sáng/tắt theo quy luật lặp đi lặp lại
    for(;;)
    {
        /* Đoạn mã điều khiển các LED đơn */

        // Các LED sáng/tắt theo 8 bit của biến led_shift
        PORTD = led_shift;

        // Thay đổi biến led_shift
        if(led_shift != 0)                // Nếu led_shift khác 0
            led_shift = led_shift << 1;  // Dịch trái 1 bit
        else
            led_shift = 255;              // Trở lại giá trị 255

        /* Đoạn mã điều khiển LED 7 thanh */

        // Xuất giá trị đếm ra LED 7 thanh
        LED7_OUT(led_7_count);

        // Đảo trạng thái PC3 để nhấp nháy dấu chấm trên LED 7 thanh
    }
}

```



```

        PORTC ^= (1<<PC3);

        // Tăng dần giá trị đếm
        led_7_count = led_7_count + 1;

        // Khi vượt quá 9, giá trị đếm được reset về 0
        if(led_7_count > 9)
            led_7_count = 0;

        // Hàm trễ khoảng 0.5 s = 500 ms
        DELAY_MS(500);
    }
}

/*****
Hàm LED7_OUT() là hàm có tham số num nhưng không trả lại giá trị, do người
lập trình tự xây dựng để điều khiển LED 7 thanh chỉ thị giá trị của num (0-
9) bằng cách sáng/tắt các đoạn LED một cách phù hợp. Hàm LED7_OUT() không
làm thay đổi trạng thái sáng/tắt của dấu chấm trên LED 7 thanh. Trong mạch
Kit này, trạng thái logic làm các thanh LED sáng/tắt như sau: 0 - thanh LED
sáng, 1 - thanh LED tắt.
*****/
void LED7_OUT(unsigned char num)
{
    // Khai báo biến temp lưu trạng thái của PORTC
    unsigned char temp = PORTC;

    // Các chân vi điều khiển ứng với các thanh LED
    // a - PC5                PC5
    // b - PC4                PC6        PC4
    // c - PC2                PC6        PC4
    // d - PC1                PC7
    // e - PC0                PC0        PC2
    // f - PC6                PC0        PC2
    // g - PC7                PC1        PC3
    // dot - PC3

    // Tắt các đoạn LED hiện đang sáng trước khi sáng các đoạn LED mới
    temp &= 0B00001000;

    // Gán mức logic cho 8 bit của biến temp ứng với giá trị của biến num
    switch(num)
    {
        case 0: temp |= 0B10000000; break;
        case 1: temp |= 0B11100011; break;
        case 2: temp |= 0B01000100; break;
        case 3: temp |= 0B01000001; break;
        case 4: temp |= 0B00100011; break;
        case 5: temp |= 0B00010001; break;
        case 6: temp |= 0B00010000; break;
        case 7: temp |= 0B11000011; break;
    }
}

```

```

        case 8: temp |= 0B00000000; break;
        case 9: temp |= 0B00000001; break;
    }

    // Xuất giá trị logic mới ra PORTC để làm sáng LED 7 thanh
    PORTC = temp;
}

/*****
Hàm DELAY_MS() là hàm có tham số mili_count nhưng không trả lại giá trị, do
người lập trình tự xây dựng để tạo ra khoảng thời gian trễ (thời gian chờ)
tính bằng mili giây. Việc trễ được thực hiện bằng các vòng lặp rỗng. Vòng
lặp rỗng (cụ thể là vòng for) tuy không thực hiện công việc gì nhưng vẫn
làm CPU tiêu tốn một khoảng thời gian nhất định cho việc khởi tạo và kết
thúc. Nhiều vòng for liên tiếp sẽ tạo một khoảng trễ đáng kể.
*****/
void DELAY_MS(unsigned int mili_count)
{
    // Khai báo hai biến chạy cho hai vòng for
    unsigned int i,j;

    // Xung nhịp của hệ thống càng cao, số vòng lặp càng tăng
    mili_count = mili_count * FRE;

    // Các vòng for gây trễ
    for(i = 0; i < mili_count; i++)
        for(j = 0; j < 53; j++);
}

```

3.3 Đọc trạng thái logic đầu vào số

3.3.1 Mục tiêu

Mục tiêu chung:

- Giúp sinh viên làm quen với việc bổ sung lệnh vào chương trình cũ và viết thêm hàm vào thư viện đã có.
- Nhận biết phím được ấn và hiển thị số thứ tự phím ra LED.

Cụ thể, sinh viên điều khiển LED sáng/tắt theo quy tắc sau:

- Với LED đơn: khi bật nguồn, toàn bộ LED tắt. Nếu phím PB1 được ấn, chỉ hai LED ngoài cùng bên trái sáng. Nếu phím PB2 được ấn, chỉ hai LED tiếp theo sáng,... Nếu phím PB4 được ấn, chỉ hai LED ngoài cùng bên phải sáng.
- Với LED 7 thanh: khi bật nguồn, LED 7 thanh hiện số 0. Nếu phím PBx được ấn, LED 7 thanh hiện giá trị của x.

3.3.2 Công việc cần làm

Sinh viên cần thực hiện các công việc sau:

- Tạo một bản sao lưu của toàn bộ Project *AVR_Kit_Test* rồi mới chỉnh sửa.
- Tạo thêm hai chương trình con trong thư viện *thu_vien_rieng.h* là:
 - *PB_2_LED()*: chương trình con dùng để điều khiển LED theo phím ấn.
 - *PB_CHECK()*: chương trình con dùng để nhận diện phím đang được ấn.
- Chỉnh sửa và bổ sung các lệnh cần thiết vào file *AVR_Kit_Test.c* đã có.
- Tiến hành biên dịch sang mã máy và nạp mã máy xuống IC VDK.
- Sửa lỗi (nếu có) và chạy thử khi hoàn thành.
- Thay đổi lệnh trong hàm *PB_CHECK()* rồi chạy thử để quan sát hiệu ứng.

3.3.3 Hướng dẫn thực hiện

Sinh viên thực hành viết chương trình bằng cách chỉnh sửa hoặc bổ sung các đoạn mã nguồn mẫu sau đây vào các file tương ứng.

❖ File *thu_vien_rieng.h*

Bổ sung hai nguyên mẫu hàm:

```
// Khai báo các nguyên mẫu hàm
void INIT();
void PORT();
void LED7_OUT(unsigned char num);
void DELAY_MS(unsigned int mili_count);
void PB_2_LED();
unsigned char PB_CHECK();
```

Bổ sung hai hàm:

```
/******
Hàm PB_2_LED() là hàm không có tham số và không trả lại giá trị, do người
lập trình tự xây dựng để điều khiển các LED theo phím ấn với quy tắc mô tả
tại Mục 3.3.1.
******/
void PB_2_LED()
{
    // Vòng for giúp việc quét phím ấn được lặp đi lặp lại
    for(;;)
    {
        // Gọi hàm quét phím, lưu kết quả phím ấn vào biến push_button
        push_button = PB_CHECK();
    }
}
```

```

// Hiện số thứ tự phím ấn ra LED 7 thanh
LED7_OUT(push_button);

// Điều khiển hàng LED đơn
switch (push_button)
{
    // Nếu push_button = 1, sáng 2 LED ngoài cùng bên trái
    case 1: PORTD = 0b11111100; break;

    // Nếu push_button = 2, ...
    case 2: PORTD = 0b11110011; break;
    case 3: PORTD = 0b11001111; break;
    case 4: PORTD = 0b00111111; break;

    // push_button = 0, tắt tất cả các LED
    default: PORTD = 0xFF;
}
}

/*****
Hàm PB_CHECK() là hàm không có tham số và có trả lại giá trị, do người lập
trình tự xây dựng để nhận diện phím đang được ấn. Giá trị trả lại của hàm
chính là thứ tự của phím. Khi được ấn, phím sẽ kết nối chân tương ứng của
VĐK tới GND (mức logic 0). Khi nhả phím, chân tương ứng của VĐK được treo
lên mức logic 1 nhờ các trở kéo có sẵn. Lưu ý: phím 1 được nối tới PB0,
phím 2 được nối tới PB1,... phím 4 được nối tới PB3.
*****/
unsigned char PB_CHECK()
{
    // Kiểm tra trạng thái logic của 4 chân PB0-3. Nếu khác 1111
    if((PINB & 0x0F) != 0x0F)
    {
        // Kiểm tra PB0. Nếu là mức logic 0, hàm kết thúc và = 1
        if(!(PINB & (1<<PB0)))
            return 1;

        // Kiểm tra PB1. Nếu là mức logic 0, hàm kết thúc và = 2
        if(!(PINB & (1<<PB1)))
            return 2;

        // Kiểm tra PB2. Nếu là mức logic 0, hàm kết thúc và = 3
        if(!(PINB & (1<<PB2)))
            return 3;

        // Kiểm tra PB3. Nếu là mức logic 0, hàm kết thúc và = 4
        if(!(PINB & (1<<PB3)))
            return 4;
    }

    // Nếu không có phím nào được ấn, hàm kết thúc và = 0

```

```

    return 0;
}

```

❖ File *AVR_Kit_Test.c*

Chỉnh sửa mã nguồn và bổ sung lệnh mới:

```

// Khai báo thư viện chuẩn
#include <avr\io.h>

// Định nghĩa hằng số FRE = 8, là xung nhịp của hệ thống (tính bằng MHz)
#define FRE 8

// Khai báo push_button là biến toàn cục, lưu số thứ tự phím được ấn
unsigned char push_button = 0;

// Khai báo file thư viện riêng
#include "thu_vien_rieng.h"

// Lập trình hàm main()
int main()
{
    // Gọi hàm INIT() trong file thu_vien_rieng.h để khởi tạo
    INIT();

    // Tạm vô hiệu hóa hàm PORT()
    // PORT();

    // Gọi hàm PB_2_LED() trong file thu_vien_rieng.h
    PB_2_LED();

    // Lệnh return luôn xuất hiện ở cuối hàm main()
    return 0;
}

```

Sau khi hoàn thành việc chỉnh sửa, bổ sung, và chạy thử chương trình điều khiển LED theo phím ấn, sinh viên thử thay đổi lệnh cuối cùng trong hàm PB_CHECK() từ *return 0* thành *return push_button* rồi chạy thử và tìm cách giải thích sự khác biệt.



Khi biên dịch toàn bộ mã nguồn, các lệnh bổ sung vào file *.h sẽ không được cập nhật một cách tự động. Do đó, sau khi chỉnh sửa một thư viện, sinh viên cần thực hiện lệnh save file thư viện đó để trình biên dịch làm việc với mã nguồn mới nhất.

Trong các file *.h, thứ tự trước/sau khi viết hàm là không quan trọng. Tuy nhiên, trong file *.c, thứ tự khai báo các thư viện *.h (nếu có nhiều hơn 1) phải được cân nhắc kỹ. Thư viện liệt kê sau có thể sử dụng hàm trong thư viện trước nhưng ngược lại sẽ không được chấp nhận khi biên dịch mã nguồn.

3.4 Đo điện áp tương tự và hiển thị kết quả bằng màn hình LCD

3.4.1 Mục tiêu

Mục tiêu chung:

- Giúp sinh viên làm quen với việc đọc hiểu và kế thừa mã nguồn của người khác thông qua việc sử dụng hàm có sẵn trong thư viện ADC và thư viện LCD.
- Điều khiển bộ ADC số hóa mức điện áp tương tự tại đầu vào ADC0 (thay đổi bằng biến trở vi chỉnh) rồi hiển thị kết quả ra màn hình LCD.

Cụ thể, sinh viên điều khiển bộ ADC và màn hình LCD như sau:

- Bộ ADC: liên tục số hóa mức điện áp tại đầu vào ADC0, 10 bit kết quả được lưu trong thanh ghi ADC.
- Màn hình LCD: hiển thị dòng text và liên tục cập nhật giá trị ADC theo mẫu dưới đây:



The image shows a green LCD screen with black text. The first line reads "Test ADC & LCD" and the second line reads "ADC0 : xxxx / 1023".

3.4.2 Công việc cần làm

Sinh viên cần thực hiện các công việc sau:

- Tạo một bản sao lưu của toàn bộ Project *AVR_Kit_Test* rồi mới chỉnh sửa.
- Copy 2 file chứa thư viện ADC và LCD từ thư mục *Library* đã được cung cấp sẵn vào thư mục chứa file *thu_vien_rieng.h* trong Project.
- Chỉnh sửa và bổ sung các lệnh cần thiết vào thư viện *thu_vien_rieng.h* để khởi tạo ADC, LCD, rồi điều khiển hoạt động.
- Chỉnh sửa và bổ sung các lệnh cần thiết vào file *AVR_Kit_Test.c* đã có.
- Tiến hành biên dịch sang mã máy và nạp mã máy xuống IC VĐK.
- Sửa lỗi (nếu có) và chạy thử khi hoàn thành.

3.4.3 Hướng dẫn thực hiện

Sinh viên thực hành viết chương trình bằng cách chỉnh sửa hoặc bổ sung các đoạn mã nguồn mẫu sau đây vào các file tương ứng.

❖ File *thu_vien_rieng.h*

Bổ sung nguyên mẫu hàm:

```
// Khai báo các nguyên mẫu hàm
void INIT();
void PORT();
void LED7_OUT(unsigned char num);
void DELAY_MS(unsigned int mili_count);
void PB_2_LED();
unsigned char PB_CHECK();
void ADC_2_LCD();
```

Chỉnh sửa và bổ sung hàm:

```
void INIT()
{
    // Khởi tạo trạng thái Output cho các chân nối tới các LED đơn
    DDRD |= 0xFF;

    // Khởi tạo trạng thái logic 1 cho các chân nối tới các LED đơn
    PORTD |= 0xFF;

    // Khởi tạo trạng thái Output cho các chân nối tới LED 7 thanh
    DDRC |= 0xFF;

    // Khởi tạo trạng thái logic 1 cho các chân nối tới LED 7 thanh
    PORTC |= 0xFF;

    // Khởi tạo trạng thái input thả nổi cho 8 đầu vào ADC
    DDRA = 0x00;
    PORTA = 0x00;

    // Gọi các hàm khởi tạo tham số cho bộ ADC
    ADC_PRES(128);
    ADC_AVCC();
    ADC_IN(0);
}
```

```
void ADC_2_LCD()
{
    // Khởi tạo màn hình LCD
    DDRD |= (1<<PD5);
    PORTD &= ~(1<<PD5);
    PORTC |= 0x0F;

    LCD4_INIT(0,0);

    // Hiển thị các dòng text tĩnh
    LCD4_CUR_GOTO(1,0);
    LCD4_OUT_STR("Test ADC & LCD");
```

```

LCD4_CUR_GOTO(2,0);
LCD4_OUT_STR("ADC0: 0000/1024");

// Vòng lặp đo và cập nhật giá trị ADC
for(;;)
{
    ADC_STA_CONVERT();
    LCD4_CUR_GOTO(2,6);
    LCD4_OUT_DEC(ADC, 4);
    DELAY_MS(200);
}
}

```

❖ File *AVR_Kit_Test.c*

Chỉnh sửa và bổ sung lệnh mới:

```

// Khai báo thư viện chuẩn
#include <avr\io.h>

// Định nghĩa hằng số FRE = 8, là xung nhịp của hệ thống (tính bằng MHz)
#define FRE 8

// Khai báo push_button là biến toàn cục, lưu số thứ tự phím được ấn
unsigned char push_button = 0;

// Khai báo thêm hai thư viện trước thư viện riêng
#include "hunget_adc.h"
#include "hunget_lcd.h"
#include "thu_vien_rieng.h"

// Lập trình hàm main()
int main()
{
    // Gọi hàm INIT() trong file thu_vien_rieng.h để khởi tạo
    INIT();

    // Tạm vô hiệu hóa hàm PORT() và PB_2_LED();
    // PORT();
    // PB_2_LED();

    // Gọi hàm ADC_2_LCD()
    ADC_2_LCD();

    // Lệnh return luôn xuất hiện ở cuối hàm main()
    return 0;
}

```


3.5 Giao tiếp với máy tính qua chuẩn UART-USB

3.5.1 Mục tiêu

Mục tiêu chung:

- Giúp sinh viên củng cố kỹ năng đọc hiểu, kế thừa, và tạo mới mã nguồn.
- Điều khiển bộ UART để giao tiếp với máy tính qua cổng USB (COM ảo).

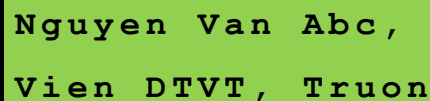
Cụ thể, sinh viên điều khiển bộ UART và màn hình LCD như sau:

- Bộ UART: hoạt động ở chế độ truyền 8 bit, tốc độ 9600 bps, liên tục gửi đoạn text sau lên máy tính:

[Họ và tên], [MSSV], [Lớp - Khóa]

Vien Dien tu - Vien Thong, Truong DHBK HN

- Màn hình LCD: hiển thị và dịch dần (sau mỗi 1 s) hai dòng chữ trên sang trái để có thể quan sát toàn bộ đoạn text.



Nguyen Van Abc,
Vien DTVT, Truon

3.5.2 Công việc cần làm

Sinh viên cần thực hiện các công việc sau:

- Tạo một bản sao lưu của toàn bộ Project *AVR_Kit_Test* rồi mới chỉnh sửa.
- Copy thêm file chứa thư viện UART từ thư mục *Library* đã được cung cấp sẵn vào thư mục chứa file *thu_vien_rieng.h* trong Project.
- Chỉnh sửa và bổ sung các lệnh cần thiết vào thư viện *thu_vien_rieng.h* để khởi tạo UART rồi điều khiển hoạt động.
- Chỉnh sửa và bổ sung các lệnh cần thiết vào file *AVR_Kit_Test.c* đã có.
- Tiến hành biên dịch sang mã máy và nạp mã máy xuống IC VĐK.
- Sửa lỗi (nếu có) và chạy thử khi hoàn thành.

3.5.3 Hướng dẫn thực hiện

Gợi ý mã nguồn hàm điều khiển bộ UART và hàm dịch cửa sổ của màn hình LCD:

```
void UART()  
{  
    UART_INIT(51, 8, 0, 1);  
}
```

```

    DDRD |= (1<<PD5);
    PORTD &= ~(1<<PD5);
    PORTC |= 0x0F;

    LCD4_INIT(0,0);

    LCD4_CUR_GOTO(1,0);
    LCD4_OUT_STR("Nguyen Van Abc, 20192019, DT01 - K64");
    LCD4_CUR_GOTO(2,0);
    LCD4_OUT_STR("Vien DTVT, Truong DHBK HN");

    DELAY_MS(1000);

    for(;;)
    {
        UART_TRAN_STR("Nguyen Van Abc, 20192019, DT01 - K64");
        UART_TRAN_BYTE(13);
        UART_TRAN_BYTE(10);

        UART_TRAN_STR("Vien DTVT, Truong DHBK HN");
        UART_TRAN_BYTE(13);
        UART_TRAN_BYTE(10);

        UART_TRAN_BYTE(13);
        UART_TRAN_BYTE(10);

        DELAY_MS(1000);

        LCD4_DIS_SHIFT(1, 1);
    }
}

void LCD4_DIS_SHIFT(unsigned char lcd4_direct, unsigned char lcd4_step)
{
    unsigned char i;
    if(lcd4_direct == 0)
        for(i=0;i<lcd4_step;i++)
            LCD4_OUT_CMD(0x1C);
    else
        for(i=0;i<lcd4_step;i++)
            LCD4_OUT_CMD(0x18);
}

```

4. VẬN DỤNG KIẾN THỨC VÀO ỨNG DỤNG THỰC TẾ

4.1 Mục tiêu

Sau quá trình làm quen công cụ thiết kế mạch điện và thực hành lập trình phần cứng với mã nguồn mẫu, sinh viên vận dụng các kiến thức đã học để thiết kế và hoàn

thiện một yêu cầu đặt ra trong thực tiễn. Thông qua nội dung công việc này, sinh viên không chỉ được củng cố các kiến thức đã học mà còn hoàn thiện các kỹ năng cơ bản về thiết kế và lập trình phần cứng.

4.2 Một số đề tài và gợi ý cách thực hiện

Mỗi sinh viên lựa chọn và thực hiện một trong các đề tài sau:

- **Đề tài số 1:** thiết kế và chế tạo mạch đo nhiệt độ sử dụng cảm biến tương tự.
 - *Yêu cầu công việc:* chế tạo một mạch điện có khả năng đo nhiệt độ trong dải 0-100 °C sử dụng các cảm biến tương tự. Kết quả đo được hiển thị bằng màn hình LCD với độ phân giải tối thiểu là 0.25 °C.
 - *Gợi ý cách thực hiện:* thiết kế một mô-đun đo sử dụng cảm biến LM35 kết nối tới mạch Kit để số hóa, xử lý, và hiển thị kết quả. Do đầu ra cảm biến LM35 thay đổi 10 mV/°C và độ phân giải của bộ ADC trong VDK AVR là 10-bit, cần thiết kế một mạch khuếch đại trung gian giữa cảm biến và bộ ADC để đáp ứng yêu cầu về độ phân giải.
 - Trong một hướng tiếp cận khác, sinh viên có thể thiết kế một mạch điện hoàn toàn mới, sử dụng bộ ADC có độ phân giải cao hoặc IC VDK khác để thực hiện yêu cầu của đề tài.
- **Đề tài số 2:** thiết kế và chế tạo mạch điều khiển động cơ DC công suất nhỏ.
 - *Yêu cầu công việc:* chế tạo một mạch công suất có khả năng điều khiển chiều quay và tốc độ của động cơ DC. Điện áp hoạt động của mạch là 20-28 VDC, dòng điện trung bình cấp cho động cơ tối đa là 2 A.
 - *Gợi ý cách thực hiện:* thiết kế mô-đun mạch công suất sử dụng E-MOSFET IRF540 để đóng cắt và một rơ-le loại DPDT để đảo chiều dòng điện DC cấp vào động cơ. Mô-đun mạch công suất được kết nối tới và nhận lệnh điều khiển từ mạch Kit. Bốn lệnh điều khiển chạy/dừng, đảo chiều quay, tăng tốc, giảm tốc ứng với 4 phím ấn trên mạch Kit. Việc điều chỉnh tốc độ động cơ được thực hiện bằng cơ chế điều chế độ rộng xung (PWM).
 - Trong một hướng tiếp cận khác, sinh viên có thể thiết kế một mạch điện hoàn toàn mới, sử dụng mạch cầu H và IC VDK khác để thực hiện yêu cầu của đề tài.

4.3 Yêu cầu về kết quả và báo cáo

Sinh viên lựa chọn, thực hiện, và hoàn thiện đề tài theo kế hoạch do giảng viên hướng dẫn quyết định. Sản phẩm cuối cùng có thể là một mạch điện hoàn chỉnh hoặc một mô-đun kết nối tới mạch Kit. VDK sử dụng có thể là AVR hoặc một loại khác, nhưng các yêu cầu về chức năng của mạch (nêu trong Mục 4.2) cần được đảm bảo và sinh viên phải hiểu rõ các nội dung công việc đã làm.

Kết thúc quá trình học, ngoài sản phẩm mạch điện đã hoạt động tốt, sinh viên phải nộp báo cáo thiết kế. Nội dung của báo cáo cần phản ánh trung thực quá trình thực hiện và hình thức của báo cáo cần tuân theo template được nêu rõ trong tài liệu hướng dẫn trình bày đồ án tốt nghiệp, dành cho sinh viên Viện Điện tử - Viễn thông, Trường Đại học Bách Khoa Hà Nội, áp dụng từ học kỳ 20182.