CHƯƠNG IV
LẬP TRÌNH GIAO TIẾP NGOẠI VI

Lời đầu chương

Sau khi nghiên cứu những nội dung trong chương III-Lập trình nhúng nâng cao, người học đã có những kiến thức cần thiết về lập trình user application và kernel driver để bắt tay vào viết ứng dụng điều khiển các thiết bị ngoại vi. Để quá trình nghiên cứu đạt hiệu quả cao nhất, trước khi di vào từng bài thực hành trong chương này người học phải có những kiến thức và kỹ năng sau:

- Sử dụng thành thạo những phần mềm hỗ trợ lập trình nhúng như: SSH, Linux ảo, console putty, ... Tất cả đều được trình bày trong chương II-Lập trình nhúng căn bản.
- Biết cách biên dịch chương trình ứng dụng trong user bằng trình biên dịch gcc trong linux ảo và trực tiếp trong hệ thống linux của kit; Biên dịch chương trình driver bằng tập tin Makefile; Cách cài đặt driver và thực thi ứng dụng trong kit.
- Trình bày được các vấn đề có liên quan đến character device driver. Giải thích được các câu lệnh được sử dụng trong quá trình lập trình application cũng như lập trình driver như: các giao diện hàm, gpio, trì hoãn thời gian, ...

Chương này được trình bày bao gồm những bài tập thực hành riêng biệt nhau, được sắp xếp theo thứ tự từ dễ đến khó. Mỗi bài học sẽ nghiên cứu điều khiển một thiết bị ngoại vi hoặc có thể phối hợp với các thiết bị ngoại vi khác tùy theo yêu cầu điều khiển của bài toán. Nhằm mục đích cho người học ứng dụng ngay những kiến thức đã trong trong các chương trước, từ đó sẽ khắc sâu và áp dụng một cách thành thạo vào các trường hợp trong thực tế.

Các ngoại vi được trình bày trong chương là những module được tích hợp trong CHIP vi điều khiển hoặc được lắp đặt trong các bộ thí nghiệm khác. Các ngoại vi đó là: LED đơn, LED 7 đoạn, LCD, ADC on chip, UART, I2C, ... là những module đơn giản phù hợp với trình độ, giúp cho người học có thể hiểu và vận dụng vào những ứng dụng lớn khác nhanh chóng hơn.

Mỗi bài là một dự án thực hành đa số được cấu trúc thành 3 phần: Phác thảo dự án, thực hiện dự án và kết luận-bài tập. Trong đó:

- *Phần phác thảo dự án:* Trình bày sơ lược về yêu cầu thực hiện trong dự án. Sau khi khái quát được yêu cầu, sẽ tiến hành phân công nhiệm vụ thực thi giữa hai

thành phần user application và kernel driver sau cho dự án được hoạt động tối ưu trong hệ thống.

- *Phần thực thi dự án:* Bao gồm sơ đồ nguyên lý kết nối các chân gpio với phần cứng; mã chương trình tham khảo của driver và application, mỗi dòng lệnh đều có những chú thích giúp người học hiểu được quá trình làm việc của chương trình, (mã lệnh chương trình hoàn chỉnh được lưu trong thư mục tham khảo của CD kèm theo đề tài, người học có thể chép vào chương trình soạn thảo và biên dịch thực thi).
- *Phần kết luận-bài tập:* Phần này sẽ tổng hợp lại những kiến thức kinh nghiệm lập trình mà dự án trình bày, nêu lên những nội dung chính trong bài học tiếp theo. Đồng thời chúng tôi cũng đưa ra những bài tập tham khảo giúp cho người học nắm vững kiến thức tạo nền tảng cho dự án mới.

  ***Một số dự án còn thêm phần kiến thức ban đầu nhằm giúp cho người học hiểu hơn những câu lệnh và thuật toán trong chương trình ví dụ.*

  ***Mỗi bài tập thực hành được trình bày mang tính chất kế thừa, do đó người học cần tiến hành theo dúng trình tự được biên soạn để đạt hiệu quả cao nhất.*

BÀI 1

GIAO TIẾP ĐIỀU KHIỂN LED ĐƠN

A- *ĐIỀU KHIỂN SÁNG TẮT 1 LED:*

I. Phác thảo dự án:

Đây là dự án đầu tiên căn bản nhất trong quá trình lập trình điều khiển các thiết bị phần cứng. Người học có thể làm quen với việc điều khiển các chân *gpio* cho các mục đích khác nhau: truy xuất dữ liệu, cài đặt thông số đối với một chân vào ra trong vi điều khiển thông qua *driver* và chương trình ứng dụng. Để hoàn thành được bài này, người học phải có những kiến thức và kỹ năng sau:

- Kiến thức về mối quan hệ giữa *driver* và *application* trong hệ thống nhúng, cũng như việc trao đổi thông tin qua lại dựa vào các giao diện chuẩn;
- Kiến thức về giao diện chuẩn *ioctl* trong giao tiếp giữa *driver* (trong *kernel*) và *application* (trong *user*);
- Kiến thức về *gpio* trong *linux kernel*;
- Lập trình chương trình ứng dụng có sử dụng kỹ thuật hàm *main* có nhiều tham số giao tiếp với người dùng;
- Biên dịch và cài đặt được *driver*, *application* nạp vào hệ thống và thực thi;

***Tất cả những kiến thức yêu cầu nêu trên điều đã được chúng tôi trình bày kỹ trong những phần trước. Nếu cần người học có thể quay lại tìm hiểu để bước vào nội dung này hiệu quả hơn.*

a. Yêu cầu dự án:

Dự án này có yêu cầu là điều khiển thành công 1 led đơn thông qua *driver* và *application*. Người dùng có thể điều khiển led sáng tắt và đọc về trạng thái của một chân gpio theo yêu cầu nhập từ dòng lệnh *shell*.

- Đầu tiên, người dùng xác định chế độ vào ra cho chân *gpio* muốn điều khiển.

- Tiếp theo, nếu là chế độ ngõ vào thì sẽ xuất thông tin ra màn hình hiển thị cho biết trạng thái của chân *gpio* là mức thấp hay mức cao. Nếu là chế độ ngõ ra thì người dùng sẽ nhập thông tin *high* hoặc *low* để điều khiển led sáng tắt theo yêu cầu.

  **Lưu ý, nếu ngõ vào thì người dùng nên kết nối chân *gpio* với một công tắc điều khiển ON-OFF, nếu ngõ ra thì người dùng nên kết nối chân *gpio* với một LED đơn theo kiểu tích cực mức cao.

b. Phân công nhiệm vụ:

- ***Driver***: có tên `single_led_dev.c`

Sử dụng kỹ thuật giao diện *ioctl* để nhận lệnh và tham số từ *user application* thực thi diều khiển chân *gpio* theo yêu cầu. *ioctl* có 5 tham số lệnh tương ứng với 5 khả năng mà *driver* có thể phục vụ cho *application*:

- `GPIO_DIR_IN`: Cài đặt chân *gpio* là ngõ vào;
- `GPIO_DIR_OUT`: Cài đặt chân *gpio* là ngõ ra;
- `GPIO_GET`: Lấy dữ liệu mức logic từ chân gpio ngõ vào trả về một biến của *user application*;
- `GPIO_SET`: Xuất dữ liệu cho chân *gpio* ngõ ra theo thông tin lấy từ một biến trong *user application* tương ứng sẽ là mức thấp hay mức cao;

- ***Application***: có tên `single_led_app.c`

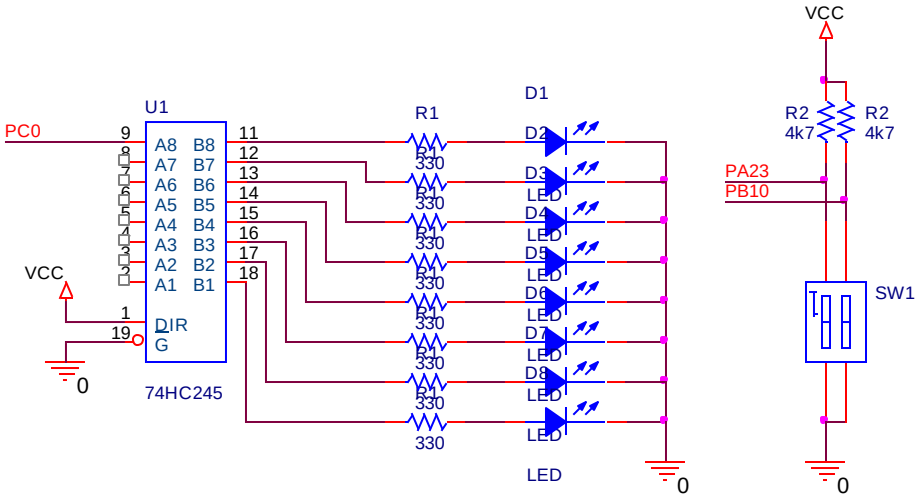
Sử dụng kỹ thuật lập trình hàm *main* có nhiều tham số lựa chọn cho người dùng khả năng điều khiển trên màn hình *shell* trong quá trình thực thi chương trình ứng dụng. Theo đó, chương trình ứng dụng `single_led_app` có những thao tác lệnh sau:

- Đầu tiên người dùng nhập tên chương trình cùng với các tham số mong muốn tương ứng với từng lệnh muốn thực thi.
- Nếu là lệnh `dirin`, người dùng phải cung cấp cho *driver* tham số tiếp theo là số chân *gpio* muốn cài đặt chế độ ngõ vào;
- Nếu là lệnh `dirout`, người dùng phải cung cấp cho *driver* tham số tiếp theo là số chân *gpio* muốn cài đặt chế độ ngõ ra;
- Nếu là lệnh `set`, thông tin tiếp theo phải cung cấp là 1 hoặc 0 và chân *gpio* muốn xuất dữ liệu;
- Nếu là lệnh `get`, thông tin tiếp theo người dùng phải cung cấp là số chân *gpio* muốn lấy dữ liệu. Sau khi lấy dữ liệu, xuất ra màn hình hiển thị thông báo cho người dùng biết.

II. Thực hiện:

a. *Kết nối phần cứng:*

Thực hiện kết nối phần cứng theo sơ đồ sau:



Hình 4-1- Sơ đồ kết nối LED đơn và công tắc điều khiển

b. *Chương trình driver:*

*/*Khai báo thư viện cho các hàm sử dụng trong chương trình*/*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <asm/gpio.h>
#include <asm/atomic.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
```

*/*Định nghĩa tên driver thiết bị*/*

```
#define DRVNAME      "single_led_dev"
#define DEVNAME      "single_led"
```

*/*Định nghĩa số định danh lệnh cho giao diện ioctl*/*

```
#define IOC_SINGLE_LED_MAGIC  'B'
#define GPIO_GET          _IO(IOC_SINGLE_LED_MAGIC, 10)
#define GPIO_SET          _IO(IOC_SINGLE_LED_MAGIC, 11)
#define GPIO_DIR_IN       _IO(IOC_SINGLE_LED_MAGIC, 12)
#define GPIO_DIR_OUT      _IO(IOC_SINGLE_LED_MAGIC, 13)
```

```
/* Counter is 1, if the device is not opened and zero (or less) if opened. */
static atomic_t gpio_open_cnt = ATOMIC_INIT(1);
/*Khai báo và định nghĩa giao diện ioctl*/
static int
gpio_ioctl(struct inode * inode, struct file * file, unsigned int
cmd, unsigned long arg[])
{
   int retval = 0;
   /*Kiểm tra số định danh lệnh thực hiện theo yêu cầu*/
   switch (cmd)
   {
   /*Trong trường hợp là lệnh GPIO_GET*/
   case GPIO_GET:
   /*Lấy thông tin từ chân gpio*/
      retval = gpio_get_value(arg[0]);
      break;
   /*Trong trường hợp là GPIO_SET*/
   case GPIO_SET:
   /*Xuất dữ liệu arg[1] từ user application cho chân gpio arg[0]*/
      gpio_set_value(arg[0], arg[1]);
      break;
   /*Trong trường hợp là lệnh GPIO_DIR_IN*/
   case GPIO_DIR_IN:
  /*Yêu cầu truy xuất chân gpio arg[0]*/
    gpio_request (arg[0], NULL);
  /*Chân gpio arg[0] trong chế độ kéo lên*/
    at91_set_GPIO_periph (arg[0], 1);
  /*Cài đặt chân gpio arg[0] chế độ ngõ vào*/
        gpio_direction_input(arg[0]);
 break;
 /*Trong trường hợp là lệnh GPIO_DIR_OUT*/
   case GPIO_DIR_OUT:
  /*Yêu cầu truy xuất chân gpio arg[0]*/
    gpio_request (arg[0], NULL);
  /*Cài đặt kéo lên cho chân gpio arg[0]*/
```

```
        at91_set_GPIO_periph (arg[0], 1);
```

*/*Cài đặt chế độ ngõ ra cho chân gpio arg[0], giá trị khởi đầu là 0*/*

```
            gpio_direction_output(arg[0], 0);
    break;
```

*/*Trường hợp không có lệnh thực thì, trả về mã lỗi*/*

```
    default:
        retval = -EINVAL;
        break;
    }
```

*/*Trả về mã lỗi cho ioctl*/*

```
    return retval;
}
```

*/*Khai báo và định nghĩa giao diện open*/*

```
static int
gpio_open(struct inode *inode, struct file *file)
{
    int result = 0;
    unsigned int dev_minor = MINOR(inode->i_rdev);
    if (!atomic_dec_and_test(&gpio_open_cnt)) {
        atomic_inc(&gpio_open_cnt);
        printk(KERN_ERR DRVNAME ": Device with minor ID %d already in
use\n", dev_minor);
        result = -EBUSY;
        goto out;
    }
out:
    return result;
}
```

*/*Khai báo và định nghĩa giao diện close*/*

```
static int
gpio_close(struct inode * inode, struct file * file)
{
    smp_mb__before_atomic_inc();
    atomic_inc(&gpio_open_cnt);

    return 0;
```

```
}
/*Gán các giao diện vào file_operations*/
struct file_operations gpio_fops = {
   .ioctl      = gpio_ioctl,
   .open      = gpio_open,
   .release   = gpio_close,
};
/*Cài đặt các thông số trên vào file_node*/
static struct miscdevice gpio_dev = {
        .minor         = MISC_DYNAMIC_MINOR,
        .name         = "single_led",
        .fops         = &gpio_fops,
};
/*Hàm khởi tạo ban đầu*/
static int __init
gpio_mod_init(void)
{
   return misc_register(&gpio_dev);
}
/*Hàm kết thúc khi tháo gỡ driver ra khỏi hệ thống*/
static void __exit
gpio_mod_exit(void)
{
   misc_deregister(&gpio_dev);
}
/*Gán các hàm khởi tạo init và kết thúc exit vào các macro cần thiết*/
module_init (gpio_mod_init);
module_exit (gpio_mod_exit);
/*Cập nhật các thông tin về chương trình*/
MODULE_LICENSE("GPL");
MODULE_AUTHOR("coolwarmboy");
MODULE_DESCRIPTION("Character device for for generic gpio api");
```

c. Chương trình application:

```
/*Khai báo các thư viện sử dụng trong chương trình*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
/*Định nghĩa số định danh lệnh sử dụng trong giao diện ioctl*/
#define IOC_SINGLE_LED_MAGIC  'B'
#define GPIO_GET        _IO(IOC_SINGLE_LED_MAGIC, 10)
#define GPIO_SET        _IO(IOC_SINGLE_LED_MAGIC, 11)
#define GPIO_DIR_IN     _IO(IOC_SINGLE_LED_MAGIC, 12)
#define GPIO_DIR_OUT    _IO(IOC_SINGLE_LED_MAGIC, 13)
/*Chương trình con in ra hướng dẫn cho người dùng, khi có lỗi xảy ra*/
void
print_usage()
{
   printf("single_led_app dirin|dirout|get|set gpio <value>\n");
   exit(0);
}
/*Chương trình chính main() khai báo theo dạng có tham số*/
int
main(int argc, char **argv)
{
/*Số int lưu trữ số chân gpio*/
   int gpio_pin;
 /*Số mô tả tập tin, được trả về khi mở tập tin thiết bị*/
   int fd;
 /*Bộ nhớ đệm trao đổi dữ liệu qua lại giữa kernel và user trong giao diện ioctl*/
   unsigned long ioctl_buff[2];
 /*Biến trả về mã lỗi trong quá trình thực thi chương trình*/
   int result = 0;
 /*Mở tập tin thiết bị trước khi thao tác*/
   if ((fd = open("/dev/single_led", O_RDWR)) < 0)
   {
```

```
/*In ra thông báo lỗi nếu quá trình mở thiết bị không thành công*/
        printf("Error whilst opening /dev/single_led_dev\n");
        return -1;
    }
/*Chuyển tham số nhập từ người dùng thành số gpio lưu vào biến gpio_pin*/
    gpio_pin = atoi(argv[2]);
/*Thông báo cho người dùng đang sử dụng chân gpio*/
    printf("Using gpio pin %d\n", gpio_pin);
/*So sánh tham số nhập từ người dùng để biết phải thực hiện lệnh nào*/
/*Trong trường hợp là lệnh "dirin" cài đặt chân gpio là ngõ vào*/
if (!strcmp(argv[1], "dirin"))
    {
    /*Cập nhật bộ nhớ đệm trước khi chuyển qua kernel*/
    ioctl_buff[0] = gpio_pin;
    /*Sử dụng giao diện ioctl với lệnh GPIO_DIR_IN*/
    ioctl(fd, GPIO_DIR_IN, ioctl_buff);
/*Trong trường hợp là lệnh "dirout" cài đặt chân gpio là ngõ ra*/
    } else if (!strcmp(argv[1], "dirout"))
    {
    /*Cập nhật cùng nhớ đệm trước khi truyền sang kernel*/
       ioctl_buff[0] = gpio_pin;
       /*Dùng giao diện ioctl với lệnh GPIO_DIR_OUT */
    ioctl(fd, GPIO_DIR_OUT, ioctl_buff);
/*Trong trường hợp là lệnh "get" lấy dữ liệu từ chân gpio*/
    } else if (!strcmp(argv[1], "get"))
    {
    /*Cập nhật vùng nhớ đệm trước khi truyền sang kernel*/
       ioctl_buff[0] = gpio_pin;
    /*Sử dụng ioctl cập nhật thông tin trả về cho user*/
       result = ioctl(fd, GPIO_GET, ioctl_buff);
       /*In thông báo cho người sử dũng biết mức cao hay thấp của chân gpio*/
    printf("Pin %d is %s\n", gpio_pin, (result ? "HIGH" : "LOW"));
/*Trong trường hợp là lệnh "set" xuất thông tin ra chân gpio*/
    } else if (!strcmp(argv[1], "set"))
    {
```

```
    /*Kiểm tra lỗi cú pháp*/
    if (argc != 4) print_usage();
    /*Cập nhật thông tin bộ nhớ đệm trước khi truyền cho kernel*/
    /*Cập nhật thông tin về số chân gpio*/
    ioctl_buff[0] = gpio_pin;
    /*Cập nhật thông tin mức muốn xuất ra chân gpio*/
    ioctl_buff[1] = atoi(argv[3]);
    /*Dung giao diện oictl để truyền thông điệp cho dirver*/
    ioctl(fd, GPIO_SET, ioctl_buff);
    } else print_usage();
    return result;
}
```

d. Biên dịch và thực thi dự án:

- *Biên dịch driver:*

Trong thư mục chứa tập tin mã nguồn drive, tạo tập tin `Makefile` có nội dung sau:

```
export ARCH=arm
export CROSS_COMPILE=arm-none-linux-gnueabi-
obj-m += single_led_dev.o
all:
/*Lưu ý phải đúng đường dẫn đến cấu trúc mã nguồn kernel*/
make -C /home/arm/project/kernel/linux-2.6.30 M=$(PWD) modules
clean:
/*Lưu ý phải đúng đường dẫn đến cấu trúc mã nguồn kernel*/
make -C /home/arm/project/kernel/linux-2.6.30 M=$(PWD) clean
```

Biên dịch *driver* bằng lệnh *shell* như sau:

```
make clean all
```

***lúc này tập tin chương trình driver được tạo thành với tên* `single_led_dev.ko`

- Biên dich *application*: Bằng lệnh *shell* sau:

```
./arm-none-linux-gnueabi-gcc single_led_app.c -o single_led_app
```

***Chương trình được biên dịch có tên là single_led_app*

- Thực thi chương trình:

➢ Chép *driver* và *application* vào kit;

➢ Cài đặt *driver* bằng lệnh: `insmod single_led_dev.ko`

➢ Thay đổi quyền thực thi cho chương trình *application* bằng lệnh:

```
chmod 777 single_led_app
```

- Chạy chương trình và quan sát kết quả:
  - Khai báo chân PC0 là ngõ ra:

    ```
    ./single_led_app dirout 96
    Using gpio pin 96
    ```

    (Lúc này led kết nối với PC0 tắt)
  - Xuất dữ liệu mức cao cho PC0:

    ```
    ./single_led_app set 96 1
    Using gpio pin 96
    ```

    (Lúc này ta thấy led nối với chân PC0 sáng lên)
  - Xuất dữ liệu mức thấp cho PC0:

    ```
    ./single_led_app set 96 0
    Using gpio pin 96
    ```

    (Lúc này ta thấy led nối với chân PC0 tắt xuống)
  - Khai báo chân PA23 là ngõ vào:

    ```
    ./single_led_app dirin 55
    Using gpio pin 55
    ```

    ***Khi công tắc nối với PA23 ở vị trí ON, chân PA23 nối xuống mass;*
  - Lấy dữ liệu vào từ chân PA23

    ```
    ./single_led_app get 55
    Using gpio pin 55
    Pin 55 is LOW
    ```

    *** Khi công tắc nối với PA23 ở vị trí OFF, chân PA23 nối lên VCC;*
  - Lấy dữ liệu vào từ chân PA23

    ```
    ./single_led_app get 55
    Using gpio pin 55
    Pin 55 is HIGH
    ```

    ***Tương tự cho các chân gpio khác;*

III. Kết luận và bài tập:

a. Kết luận:

Phần này các bạn đã nghiên cứu thành công các thao tác truy xuất chân *gpio* đơn lẻ. Kiến thức này sẽ làm nền cho các bài lớn hơn, truy xuất theo port 8 bits, hay điều khiển thiết bị ngoại vi bằng nút nhấn... Chúng ta sẽ tìm hiểu các kỹ thuật lập trình giao tiếp *gpio* khác với nhiều chân *gpio* cùng một lúc trong phần sau.

b. Bài tập:

1. Dựa vào các lệnh trong *driver* `single_led_dev.ko` hỗ trợ, hãy viết chương trình *application* cho 1 led sáng tắt với chu kỳ 1s trong 10 lần rồi ngưng.
2. Xây dựng chương trinh *application* dựa vào *driver* `single_led_dev.ko` có sẵn để điều khiển 8 LEDS sáng tắt cùng một lúc với chu kỳ 1 s liên tục.
3. Xây dựng *driver* mới dựa vào *driver* `single_led_dev.ko` với yêu cầu: Thêm chức năng set 1 port 8 bit, và tắt 1 port 8 bit. Viết chương trình *application* sử dụng *driver* mới để thực hiện lại yêu cầu của bài 2.

B. ĐIỀU KHIỂN SÁNG TẮT 8 LED:

I. Phác thảo dự án:

Dự án này chủ yếu truy xuất chân *gpio* theo chế độ ngõ ra, nhưng điểm khác biệt so với dự án trước là không điều khiển riêng lẻ từng bit mà công việc diều khiển này sẽ do *driver* thực hiện. Phần này sẽ cho chúng ta làm quen với cách điều khiển thông tin theo từng *port 8 bits*. Để việc tiếp thu đạt hiệu quả cao nhất, trước khi nghiên cứu người học phải có những kiến thức và kỹ năng sau:

- Kiến thức tổng quát về mối quan hệ giữa *driver* và *application* trong hệ thống nhúng, cũng như việc trao đổi thông tin qua lại dựa vào các giao diện chuẩn;
- Kiến thức về giao diện chuẩn *write* trong giao tiếp giữa *driver* (trong *kernel*) và *application* (trong *user*);
- Kiến thức về *gpio* trong *linux kernel*;
- Lập trình chương trình ứng dụng có sử dụng kỹ thuật hàm *main* có nhiều tham số giao tiếp với người dùng;
- Biên dịch và cài đặt được *driver*, *application* nạp vào hệ thống và thực thi;

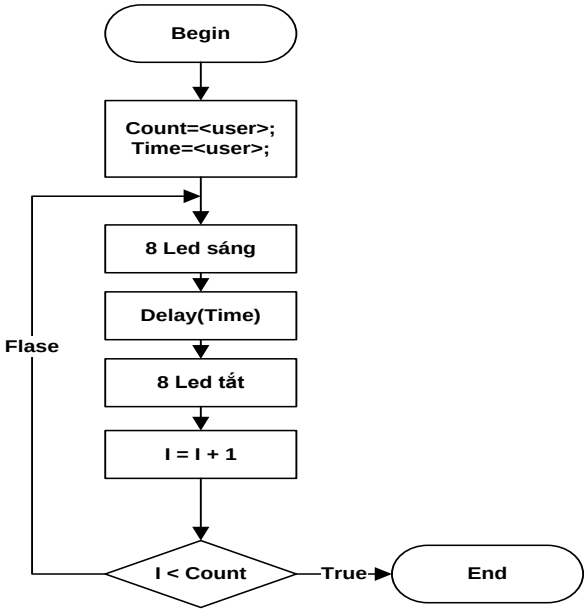
***Tất cả những kiến thức yêu cầu nêu trên điều đã được chúng tôi trình bày kỹ trong những phần trước. Nếu cần người học có thể quay lại tìm hiểu để bước vào nội dung này hiệu quả hơn.*

a. Yêu cầu dự án:

Yêu cầu của dự án là điều khiển thành công *1 port 8 leds* hoạt động chớp tắt cùng lúc theo chù kỳ và số lần được nhập từ người dùng trong lúc gọi chương trình thực thi. Khi hết nhiệm vụ chương trình sẽ được thoát và chờ lần gọi thực thi tiếp theo.

- Đầu tiên người dùng gọi chương trình *driver*, cung cấp thông tin về thời gian của chu kỳ và số lần nhấp nháy mong muốn;
- Chương trình *application* nhận dữ liệu từ người dùng, tiến hành điều khiển *driver* tác động vào ngõ ra gpio làm led sáng tắt theo yều cầu;

- Lưu đồ điều khiển như sau:



Hình 4-2- Lưu đồ điều khiển LED sáng tắt theo số chu kỳ được quy định

b. Phân công nhiệm vụ:

- ***Driver:*** Có tên là `port_led_dev.c`

Driver sử dụng giao diện `write()` nhận dữ liệu từ *user application* xuất ra led tương ứng với dữ liệu nhận được. Dữ liệu nhận từ *user application* là một số char có *8 bits*. Mỗi bit tương ứng với 1 led cần điều khiển. Nhiệm vụ của *driver* là so sánh tương ứng từng bit trong số `char` này để quyết định xuất mức cao hay mức thấp cho led ngoại vi. Công việc của *driver* được thực hiện tuần tự như sau:

- Yêu cầu cài đặt các chân ngoại vi là ngõ ra, kéo lên. Công việc này được thực hiện khi thực hiện lệnh cài đặt *driver* vào hệ thống linux;
- Trong giao diện hàm `write()` (nhận dữ liệu từ *user*) thực hiện xuất ra mức cao hoặc mức thấp cho gpio điều khiển led.
- Giải phóng các chân gpio đã được khai báo khi không cần sử dụng, công việc này được thực hiện ngay trước khi tháo bỏ *driver* ra khỏi hệ thống.

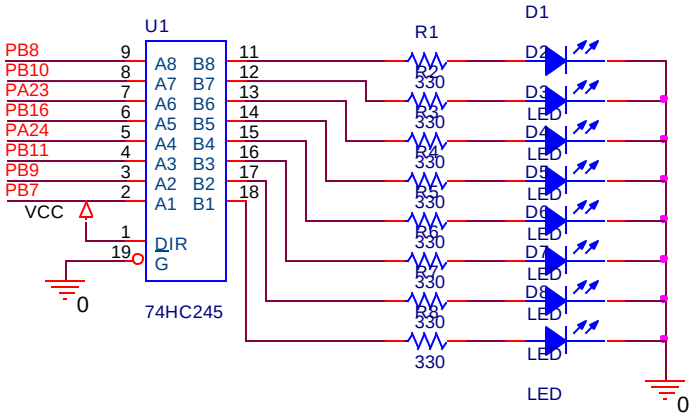
- ***Application:*** Có tên là `port_led_app.c`

Thực hiện khai báo hàm `main` theo cấu trúc tham số để đáp ứng các yêu cầu khác nhau từ người dùng. Chương trình *application* có hai tham số: Tham số thứ nhất là thời gian tính bằng giây của chu kỳ chớp tắt, tham số thứ hai là số chu kỳ muốn chớp tắt.

Bên cạnh đó, phần này còn lập trình thêm một số chương trình tạo hiệu ứng điều khiển led khác như: 8 led sáng dần tắt dần (Trái qua phải, phải qua trái, ...). Các chức năng này được tổng hợp trong một chương trình *application* duy nhất, người sử dụng sẽ lựa chọn hiệu ứng thông qua các tham số người dùng của hàm main.

II. Thực hiện:

a. Kết nối phần cứng: Các bạn thực hiện kết nối phần cứng theo sơ đồ sau:



Hình 4-3- Sơ đồ kết nối 8 LEDs đơn.

***Lưu ý phải đúng số chân đã quy ước.*

b. Chương trình driver: port_led_dev.c

*/*Khai báo thư viện cần thiết cho các hàm sử dụng trong chương trình*/*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <asm/gpio.h>
#include <asm/atomic.h>
#include <asm/atomic.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
```

*/*Đặt tên cho driver thiết bị*/*

```
#define DRVNAME      "port_led_dev"
#define DEVNAME      "port_led"
```

*/*Định nghĩa các chân sử dụng tương ứng với chân trong kit hỗ trợ */*

```
                /*-------------Port Control-----------*/
#define P00             AT91_PIN_PB8
#define P01             AT91_PIN_PB10
#define P02             AT91_PIN_PA23
#define P03             AT91_PIN_PB16
#define P04             AT91_PIN_PA24
#define P05             AT91_PIN_PB11
#define P06             AT91_PIN_PB9
#define P07             AT91_PIN_PB7
```

*/*Định nghĩa port từ các bit đã khai báo*/*

```
#define P0       (P00|P01|P02|P03|P04|P05|P06|P07)
```

*/*Khai báo các lệnh set và clear căn bản cho quá trình điều khiển port*/*

```
                        /*Basic commands*/
#define SET_P00()                       gpio_set_value(P00,1)
#define SET_P01()                       gpio_set_value(P01,1)
#define SET_P02()                       gpio_set_value(P02,1)
#define SET_P03()                       gpio_set_value(P03,1)
#define SET_P04()                       gpio_set_value(P04,1)
#define SET_P05()                       gpio_set_value(P05,1)
#define SET_P06()                       gpio_set_value(P06,1)
#define SET_P07()                       gpio_set_value(P07,1)

#define CLEAR_P00()                 gpio_set_value(P00,0)
#define CLEAR_P01()                 gpio_set_value(P01,0)
#define CLEAR_P02()                 gpio_set_value(P02,0)
#define CLEAR_P03()                 gpio_set_value(P03,0)
#define CLEAR_P04()                 gpio_set_value(P04,0)
#define CLEAR_P05()                 gpio_set_value(P05,0)
#define CLEAR_P06()                 gpio_set_value(P06,0)
#define CLEAR_P07()                 gpio_set_value(P07,0)
```

/ Counter is 1, if the device is not opened and zero (or less) if opened. */*

```
static atomic_t port_led_open_cnt = ATOMIC_INIT(1);
```

*/*Set và clear các bits trong port tương ứng với dữ liệu 8 bit nhận được*/*

```
void port_led_write_data_port(char data)
{
   (data&(1<<0))? SET_P00():CLEAR_P00();
```

```
   (data&(1<<1))? SET_P01():CLEAR_P01();
   (data&(1<<2))? SET_P02():CLEAR_P02();
   (data&(1<<3))? SET_P03():CLEAR_P03();
   (data&(1<<4))? SET_P04():CLEAR_P04();
   (data&(1<<5))? SET_P05():CLEAR_P05();
   (data&(1<<6))? SET_P06():CLEAR_P06();
   (data&(1<<7))? SET_P07():CLEAR_P07();
}
```

*/*Giao diện hàm write, nhận dữ liệu từ user để xuất thông tin ra port led*/*

```
static ssize_t port_led_write (struct file *filp, char __iomem
buf[], size_t bufsize, loff_t *f_pos)
{
```

*/*Sử dụng hàm xuất dữ liệu ra port led đã định nghĩa*/*

```
   port_led_write_data_port(buf[0]);
   return bufsize;
}
static int
port_led_open(struct inode *inode, struct file *file)
{
   int result = 0;
   unsigned int dev_minor = MINOR(inode->i_rdev);
   if (!atomic_dec_and_test(&port_led_open_cnt)) {
      atomic_inc(&port_led_open_cnt);
      printk(KERN_ERR DRVNAME ": Device with minor ID %d already in
      use\n", dev_minor);
      result = -EBUSY;
      goto out;
   }
out:
   return result;
}

static int
port_led_close(struct inode * inode, struct file * file)
{
   smp_mb__before_atomic_inc();
   atomic_inc(&port_led_open_cnt);
```

```
    return 0;
}

struct file_operations port_led_fops = {
    .write      = port_led_write,
    .open      = port_led_open,
    .release   = port_led_close,
};

static struct miscdevice port_led_dev = {
        .minor         = MISC_DYNAMIC_MINOR,
        .name          = "port_led",
        .fops          = &port_led_fops,
};

static int __init
port_led_mod_init(void)
{
    /*Yêu cầu các chân gpio muốn sử dụng*/
    gpio_request (P00, NULL);
    gpio_request (P01, NULL);
    gpio_request (P02, NULL);
    gpio_request (P03, NULL);
    gpio_request (P04, NULL);
    gpio_request (P05, NULL);
    gpio_request (P06, NULL);
    gpio_request (P07, NULL);
    /*Khởi tạo các chân gpio có điện trở kéo lên*/
    at91_set_GPIO_periph (P00, 1);
    at91_set_GPIO_periph (P01, 1);
    at91_set_GPIO_periph (P02, 1);
    at91_set_GPIO_periph (P03, 1);
    at91_set_GPIO_periph (P04, 1);
    at91_set_GPIO_periph (P05, 1);
    at91_set_GPIO_periph (P06, 1);
    at91_set_GPIO_periph (P07, 1);
    /*Khởi tạo các chân gpio có chế độ ngõ ra, giá trị ban đầu là  0*/
```

```
    gpio_direction_output(P00, 0);
    gpio_direction_output(P01, 0);
    gpio_direction_output(P02, 0);
    gpio_direction_output(P03, 0);
    gpio_direction_output(P04, 0);
    gpio_direction_output(P05, 0);
    gpio_direction_output(P06, 0);
    gpio_direction_output(P07, 0);
    return misc_register(&port_led_dev);
}

static void __exit
port_led_mod_exit(void)
{
    misc_deregister(&port_led_dev);
}
module_init (port_led_mod_init);
module_exit (port_led_mod_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("coolwarmboy");
MODULE_DESCRIPTION("Character device for for generic gpio api");
```

c. Chương trình application: có tên là `port_led_app.c`

*/*Khai báo các thư viện cần dùng cho các hàm trong chương trình*/*

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
```

*/*Hàm in ra hướng dẫn thực thi lệnh trong trường hợp người dùng nhập sai cú pháp*/*

```
void
print_usage()
{
    printf("port_led_app <TimePeriod> <NumberPeriod>\n");
    exit(0);
}
```

```
/*Khai báo hàm main có tham số cho người dùng*/
int
main(int argc, char **argv)
{
      /*Khai báo số mô tả tập tin cho driver khi được mở*/
      int port_led_fd;
      /*Khai báo vùng nhớ bộ đệm ghi cho giao diện hàm  write()*/
      char write_buf[1];
      /*Biến điều khiển và lưu trữ thông tin người dùng*/
      int i, time_period, number_period;
      /*Mở driver và kiểm tra lỗi*/
      if ((port_led_fd = open("/dev/port_led", O_RDWR)) < 0)
      {
      /*Nếu có lỗi thì in ra thông báo và kết thúc chương  trình*/
      printf("Error whilst opening /dev/port_led device\n");
      return -1;
   }
   /*Kiểm tra lỗi cú pháp từ người dùng*/
   if (argc != 3) {
   print_usage();
   }
   /*Lấy chu kỳ thời gian nhập từ người dùng*/
   time_period = atoi(argv[1]);
   /*Lấy số chu kỳ mong muốn nhập từ người dùng*/
   number_period = atoi(argv[2]);
   /*Nạp thông tin cho vùng nhớ đệm*/
   write_buf[0] = 0x00;
   /*Thực hiện chớp tắt theo đúng số chu kỳ đã đặt*/
   for (i=0; i < number_period; i++) {
   /*Cập nhật thông tin của vùng nhớ đệm ghi driver*/
         write_buf[0] = ~(write_buf[0]);
   /*Ghi thông tin của bộ đệm sang driver để ra port led*/
         write (port_led_fd, write_buf, 1);
   /*Trì hoãn thời gian theo đúng chu kỳ người dùng nhập  vào*/
         usleep(time_period*500000);
```

```
}
/*Trả về giá trị 0 khi không có lỗi xảy ra*/
return 0;
   }
```

d. Biên dịch và thực thi dự án:

- ***Biên dịch driver:***

Tạo tập tin `Makefile` trong cùng thư mục với *driver*. Có nội dung sau:

```
export ARCH=arm
export CROSS_COMPILE=arm-none-linux-gnueabi-
obj-m += port_led_dev.o
all:
/*Lưu ý phải đúng đường dẫn đến cấu trúc mã nguồn kernel*/
make -C /home/arm/project/kernel/linux-2.6.30 M=$(PWD) modules
clean:
/*Lưu ý phải đúng đường dẫn đến cấu trúc mã nguồn kernel*/
make -C /home/arm/project/kernel/linux-2.6.30 M=$(PWD) clean
```

- ***Biên dịch application:***

Trỏ vào thư mục chứa tập tin chương trình, biên dịch chương trình ứng dụng với lệnh sau:

```
arm-none-linux-gnueabi-gcc port_led_app.c –o port_led_app
**Chương trình biên dịch thành công có tên là: port_led_app
```

- ***Thực thi chương trình:***

Chép *driver* và chương trình vào kit, thực thi và kiểm tra kết quả;

- Cài đặt *driver* vào kit theo lệnh sau:

  ```
  insmod port_led_dev.ko
  ```

- Thay đổi quyền thực thi cho chương trình ứng dụng:

  ```
  chmod 777 port_led_app
  ```

- Thực thi và kiểm tra kết quả:

  ```
  ./port_led_app 1 10
  ```

  ***Chúng ta thấy 8 led nhấp nháy 10 lần với chu kỳ 1s. Các bạn thay đổi chu kỳ và số lần nhấp nháy quan sát kết quả.*

III. Kết luận và bài tập:

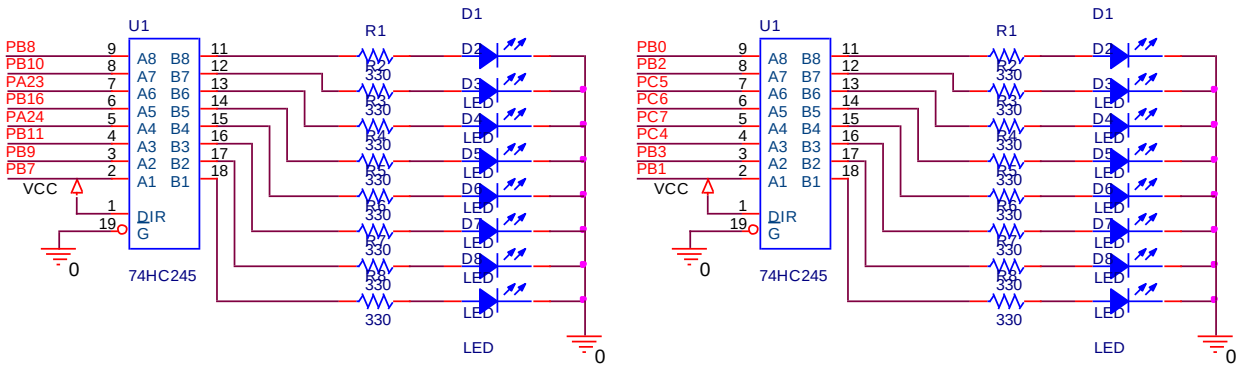
a. Kết luận:

Trong bài này chúng ta đã viết xong *driver* điều khiển 8 led đơn trong cùng một lúc tương ứng với dữ liệu nhận được từ *user application*. Chúng ta cũng đã viết một chương trình điều khiển led chớp tắt theo yêu cầu của người dùng. Trong những bài sau, *driver* này sẽ được áp dụng để lập trình các hiệu ứng điều khiển led khác.

***Do những thao tác biên dịch driver và application đã được chúng tôi trình bày rất kỹ trong phần lập trình hệ thống nhúng căn bản, hơn nữa cũng đã được nhắc lại một cách cụ thể trong những bài đầu tiên của lập trình thực hành điều khiển phần cứng, nên trong những bài tiếp theo sẽ không nhắc lại. Sau khi đã có mã nguồn của driver và application thì công việc còn lại là làm sau cho chúng có thể chay được trên kit, ... thuộc về người học.*

b. Bài tập:

1. Mở rộng *driver* điều khiển 8 LED trên thành *driver* điều khiển 16 LED theo sơ đồ kết nối sau:



Hình 4-4- Sơ đồ kết nối 16 LEDs đơn.

Nhiệm vụ của *driver* là nhận dữ liệu có chiều dài 16 bits từ *user application*. Sau đó xuất ra từng LED tương ứng với 16 bits dữ liệu.

2. Viết chương trình điều khiển 16 LEDs này chớp tắt theo yêu cầu của người sử dụng. (Về thời gian và số lượng chu kỳ muốn điều khiển).

C. SÁNG DẦN TẮT DẦN 8 LED:

I. Phác thảo dự án:

Dự án này dựa vào *driver* đã có từ bài trước để viết chương trình ứng dụng tạo nhiều hiệu ứng chớp tắt 1 port 8 LEDs khác nhau như: Điều khiển sáng, tắt dần; Sáng dồn; Điểm sáng dịch chuyển mất dần; ... Các bài tập này sẽ giúp cho người học làm quen với việc sử dụng *driver* đã xây dựng sẵn vào những chương trình ứng dụng khác nhau để hoàn thành một yêu cầu nào đó trong thực tế.

a. ***Yêu cầu dự án:***

Dự án bao gồm 2 phần, *driver* giống như của bài lập trình sắng tắt 8 LEDs và applicaiton được lập trình sử dụng *driver* này để tạo ra các hiệu ứng hiển thị LEDs khác nhau. Chương trình *user application* cũng sử dụng kỹ thuật lập trình hàm main có tham số. Người sử dụng phải nhập theo dúng cú pháp để lựa chọn cho mình hiệu ứng LEDs. Cú pháp đó là: `<tên chương trình> <kiểu hiệu ứng> <chu kỳ> <số chu kỳ>`

Trong đó:

`<Tên chương trình>` là tên chương trình đã được biên dịch từ mã nguồn;

`<kiểu hiệu ứng>` là hiệu ứng hiển thị LED, ở đây có 4 kiểu*: type1, type2, type3* và *type4;*

`<chu kỳ>` là khoảng thời gian (tính bằng *ms*) giữa 2 lần thay đổi trạng thái;

`<số chu kỳ>` là số chu kỳ lặp lại trạng thái.

b. ***Phân công nhiệm vụ:***

- ***Driver:*** Có tên là `port_led_dev.c` đã được xây dựng trong bài trước.
- ***Application:*** Có tên là `1_3_OtherLedControl.c`. Bao gồm những chức năng sau:

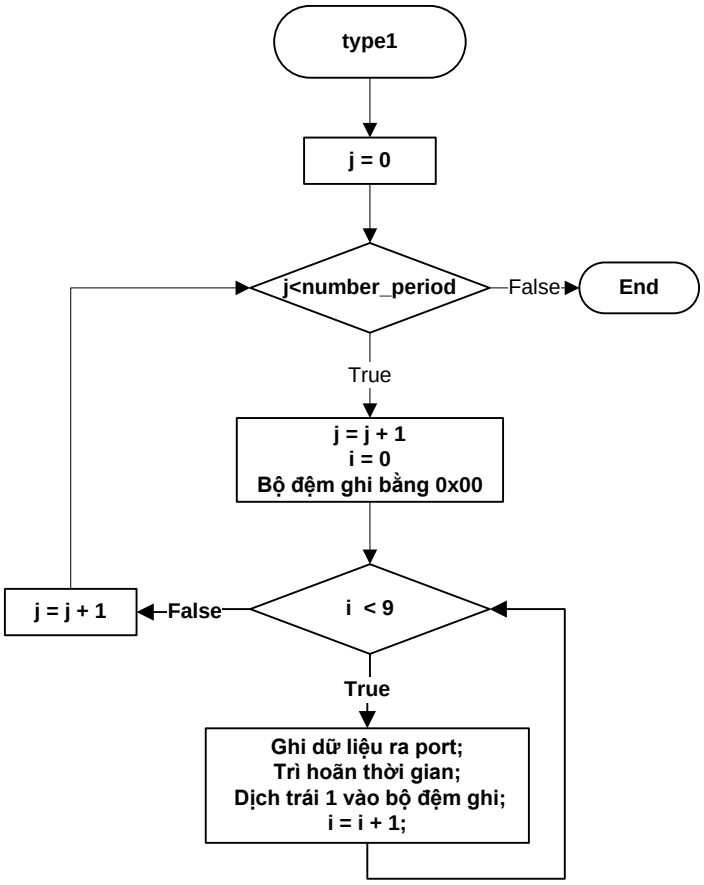
1. 8 LEDs sáng dần tắt hết (Dịch trái);

Cách 1: Tạo một mảng bao gồm có 9 trạng thái khác nhau của 8 bits sao cho có hiệu ứng sáng dần, sau đó cho 8 led tắt hết. Cứ tiếp tục theo đúng số chu kỳ người sử dụng muốn. Các trạng thái tương ứng như sau:

1: 00000000	4: 00000111	7: 00111111
2: 00000001	5: 00001111	8: 01111111
3: 00000011	6: 00011111	9: 11111111

(Quay lại từ đầu sau khi đã hiển thị hết dữ liệu, mỗi trạng thái đều có trì hoãn thời gian để quan sát được kết quả).

Cách 2: Áp dụng lệnh dịch ("<<" và ">>") trong C để viết bài toán này. Chương trình được viết theo lưu đồ thuật toán sau:



Hình 4-5- Lưu đồ điều khiển 8 LEDs sáng dần tắt hết.

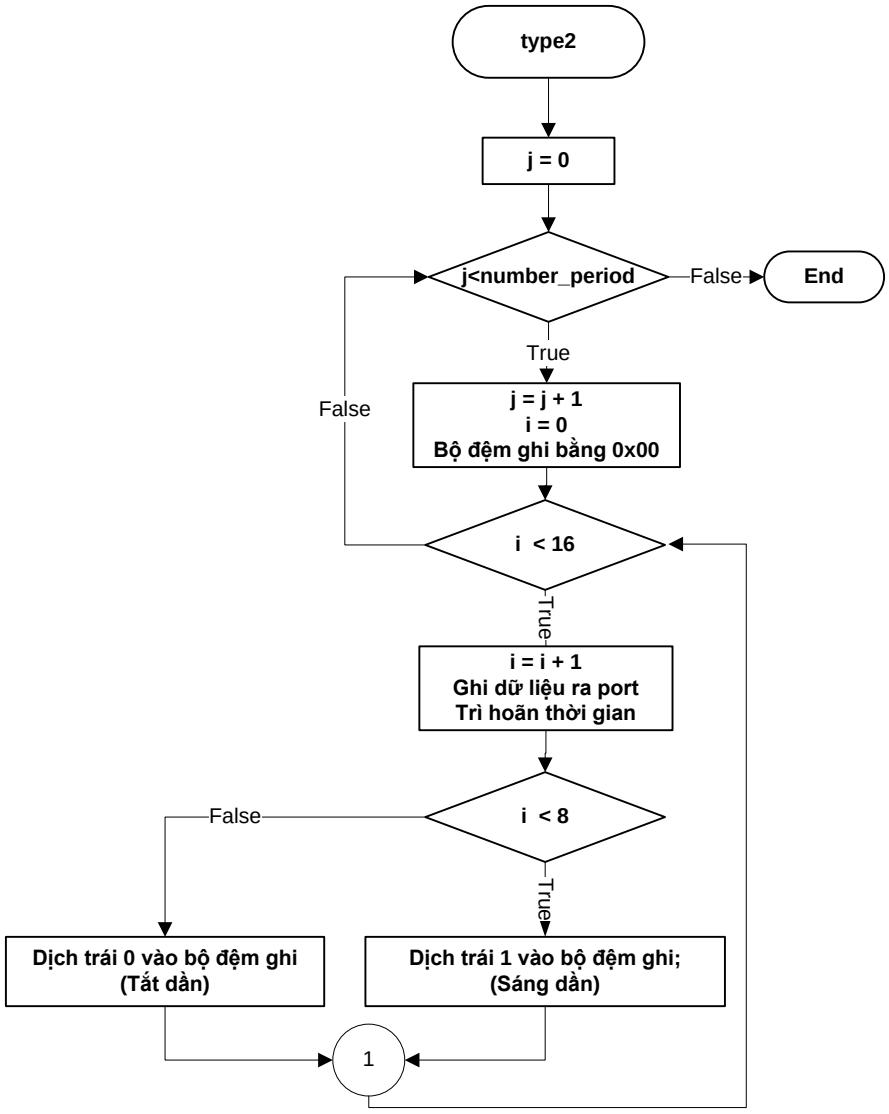
2. 8 LEDs sáng dần tắt dần (Dịch trái);

Cách 1: Tạo một mảng bao gồm có 15 trạng thái khác nhau của 8 bits sao cho có hiệu ứng sáng dần, sau đó cho 8 *led* tắt dần. Cứ tiếp tục theo đúng số chu kỳ người sử dụng muốn. Các trạng thái tương ứng như sau:

1. 00000001
2. 00000011
3. 00000111
4. 00001111
5. 00011111
6. 00111111
7. 01111111
8. 11111111
9. 11111110
10. 11111100
11. 11111000
12. 11110000
13. 11100000
14. 11000000
15. 10000000
16. Trạng thái 1. (Tính là 1 chu kỳ);

(Quay lại từ đầu sau khi đã hiển thị hết dữ liệu, mỗi trạng thái đều có trì hoãn thời gian để quan sát được kết quả).

Cách 2: Áp dụng lệnh dịch ("<<" và ">>") trong C để viết bài toán này. Chương trình được viết theo lưu đồ thuật toán sau:



Hình 4-6- Lưu đồ điều khiển 8 LEDs sáng dần và tắt dần.

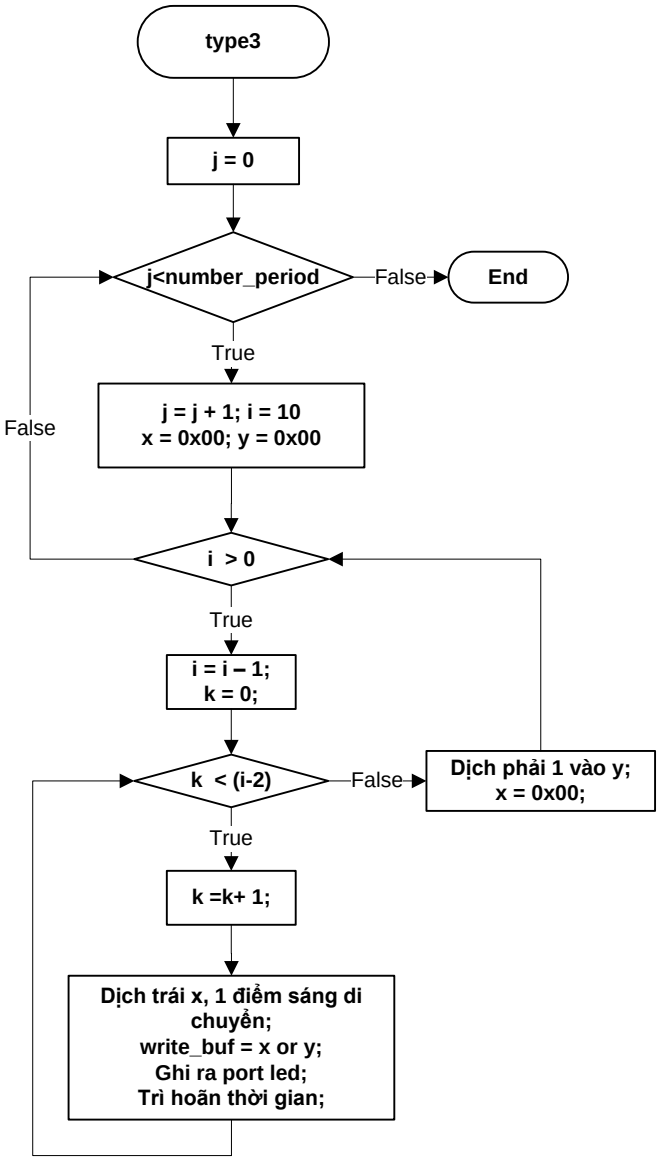
3. 8 LEDs sáng dồn:

Cách 1: Tạo một mảng bao gồm có 36 trạng thái khác nhau của 8 bits sao cho có hiệu ứng sáng dồn. Cứ tiếp tục theo đúng số chu kỳ người sử dụng muốn. Các trạng thái tương ứng như mảng số hex 8 bits sau:

```
char Data_Display_Type_3[36] = {
0x00,
0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
0x81,0x82,0x84,0x90,0xA0,0xC0,
0xC1,0xC2,0xC4,0xC8,0xD0,0xE0,
0xE1,0xE2,0xE4,0xE8,0xF0,
0xF1,0xF2,0xF4,0xF8,
0xF9,0xFA,0xFC,
0xFD,0xFE,
0xFF
}
```

***Mỗi một trạng thái là một giá trị trong mảng, chúng ta chỉ việc xuất các giá trị theo đúng thứ tự 0..35 để đạt được hiệu ứng mong muốn.* (Quay lại từ đầu sau khi đã hiển thị hết dữ liệu, mỗi trạng thái đều có trì hoãn thời gian để quan sát được kết quả).

Cách 2: Áp dụng lệnh dịch ("<<" và ">>") trong C để viết bài toán này. Chương trình được viết theo lưu đồ thuật toán sau:



Hình 4-7- Lưu đồ điều khiển 8 LEDs sáng dồn.

4. 8 LEDs dịch chuyển mất dần;

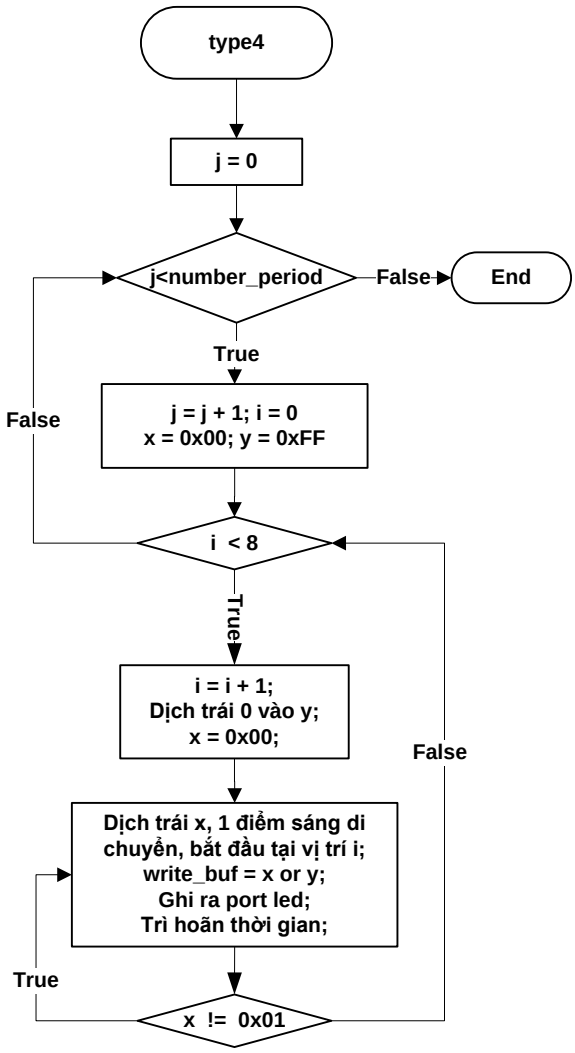
Cách 1: Tạo một mảng bao gồm có 36 trạng thái khác nhau của 8 bits sao cho có hiệu ứng dịch chuyển mất dần. Cứ tiếp tục theo đúng số chu kỳ người sử dụng muốn. Các trạng thái tương ứng như mảng số hex 8 bits sau:

```
char Data_Display_Type_3[36] = {
   0xFF,
   0xFE,0xFD,
   0xFC,0xFA,0xF9,
```

```
    0xF8,0xF4,0xF2,0xF1,
    0xF0,0xE8,0xE4,0xE2,0xE1,
    0xE0,0xD0,0xC8,0xC4,0xC2,0xC1,
    0xC0,0xA0,0x90,0x84,0x82,0x81,
    0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01
}
```

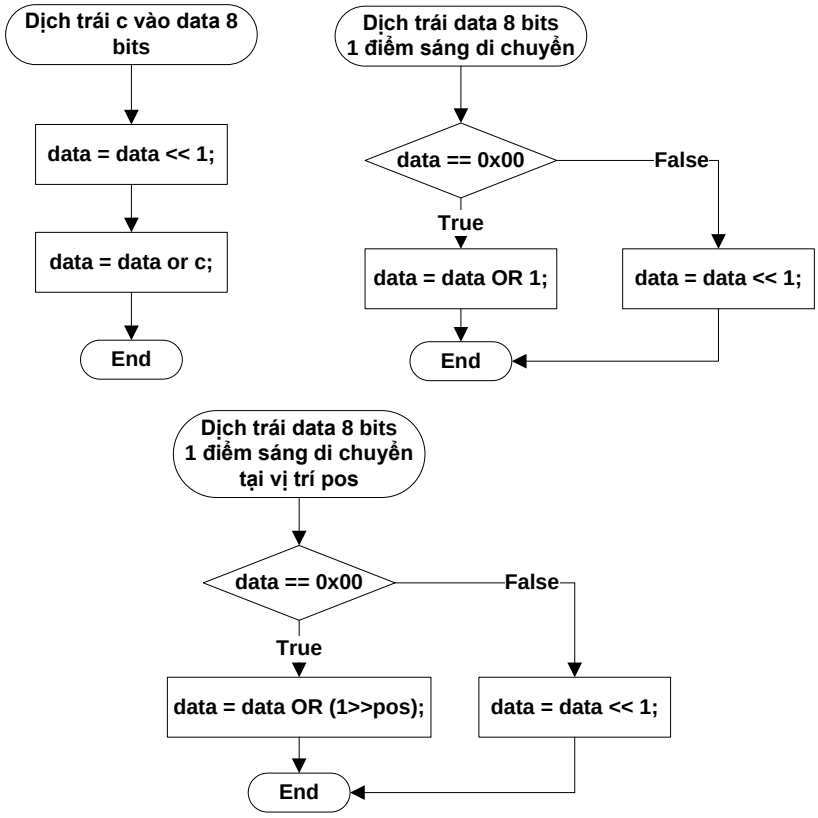
***Mỗi một trạng thái là một giá trị trong mảng, chúng ta chỉ việc xuất các giá trị theo đúng thứ tự 0..35 để đạt được hiệu ứng mong muốn.* (Quay lại từ đầu sau khi đã hiển thị hết dữ liệu, mỗi trạng thái đều có trì hoãn thời gian để quan sát được kết quả).

Cách 2: Áp dụng lệnh dịch ("<<" và ">>") trong C để viết bài toán này. Chương trình được viết theo lưu đồ thuật toán sau:



Hình 4-8- Lưu đồ điều khiển 8 LEDs dịch chuyển mất dần.

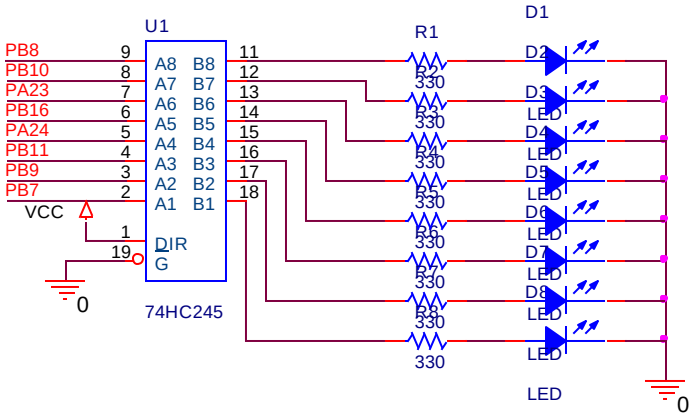
- ***Các hàm dịch bit***: *Một điểm sáng di chuyển, một diểm sáng di chuyển tại vị trí pos, dịch trái bit c vào data* được thực hiện theo những lưu đồ sau:



Hình 4-9- Lưu đồ các chương trình con dịch bits.

II. Thực hiện:

a. Kết nối phần cứng: Các bạn thực hiện kết nối phần cứng theo sơ đồ sau:



Hình 4-10- Sơ đồ kết nối 8 LEDs đơn.

***Lưu ý phải đúng số chân đã quy ước trong sơ đồ kết nối.*

b. ***Chương trình driver:*** Tương tự như bài trước;

c. Chương trỉnh application:

Cách 1: Điều khiển 8 LEDs bằng cách xuất trình tự các dữ liệu trạng thái ra port;

```
/*Chương trình mang tên 1_3_OtherPortControl_Method_1.c*/
/*Khai báo thư viện cần thiết cho các lệnh trong chương trình*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
/*Dữ liệu trạng thái LEDs cho hiệu ứng 1, 8 LEDs sáng dần  và tắt hết*/
char Data_Display_Type_1[9] ={
0x00,0x01,0x03,0x07,0x0F,0x1F,0x3F,0x7F,0xFF };
/*Dữ liệu trạng thái LEDs cho hiệu ứng 2, 8 LEDs sáng dần  và tắt dần*/
char Data_Display_Type_2[15] = {
0x00,0x01,0x03,0x07,0x0F,0x1F,0x3F,0x7F,0xFF,
0xFE,0xFC,0xF8,0xE0,0xC0,0x80 };
/*Dữ liệu trạng thái LEDs cho hiệu ứng 3, 8 LEDs sáng  dồn*/
char Data_Display_Type_3[36] = {
0x00,
0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
0x81,0x82,0x84,0x90,0xA0,0xC0,
0xC1,0xC2,0xC4,0xC8,0xD0,0xE0,
0xE1,0xE2,0xE4,0xE8,0xF0,
0xF1,0xF2,0xF4,0xF8,
0xF9,0xFA,0xFC,
0xFD,0xFE,
0xFF
};
/*Dữ liệu trạng thái LEDs cho hiệu ứng 4, 8 LEDs dịch  chuyển mất dần*/
char Data_Display_Type_4[36] = {
0xFF,
```

```
0xFE,0xFD,
0xFC,0xFA,0xF9,
0xF8,0xF4,0xF2,0xF1,
0xF0,0xE8,0xE4,0xE2,0xE1,
0xE0,0xD0,0xC8,0xC4,0xC2,0xC1,
0xC0,0xA0,0x90,0x84,0x82,0x81,
0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01
};
```

*/*Biến lưu số mô tả tập tin cho driver khi được mở*/*

```
int port_led_fd;
```

*/*Bộ nhớ đệm dự liệu cần ghi vào driver*/*

```
char write_buf[1];
```

*/*Chương trình cho in thông báo hướng dẫn sử dụng khi có lỗi cú pháp từ người dùng*/*

```
void print_usage()
{
printf ( "OtherPortControl <type1|type2|type3|type4>
<TimePeriod> <NumberPeriod>\n");
exit(0);
}
```

*/*Hàm main được khai báo dạng tham số lựa chọn, để lấy thông tin người dùng*/*

```
int
main(int argc, char **argv)
{
   /*Biến time_period lưu thời gian thay đổi trạng thái;
   biến number_period lưu số chu kỳ cần thực hiện*/
   long int time_period, number_period;
   /*Các biến đếm phục vụ cho truy xuất dữ liệu trạng thái*/
   int j;
   char i;
   /*Mở tập tin thiết bị trước khi thao tác*/
   if ((port_led_fd = open("/dev/port_led", O_RDWR)) < 0)
```

```
{
      printf("Error whilst opening /dev/port_led device\n");
      return -1;
}
/*Kiểm tra cú pháp nhập từ người dùng*/
if (argc != 4) {
      print_usage();
      printf("%d\n",argc);
}
/*Lấy thời gian trì hoãn từ người dùng*/
time_period = atoi(argv[2])*500;
/*Lấy số chu kỳ cần lặp lại từ người dùng*/
number_period = atoi(argv[3]);
/*So sánh thực hiện từng kiểu hiệu ứng khác nhau dựa vào tham số nhập từ
người dùng*/
/*Trường hợp 1: Hiệu ứng 8 LEDs sáng dần tắt hết*/
if (!strcmp(argv[1],"type1")) {
      for (j=0; j < number_period; j ++) {
            for (i=0; i < 9; i++) {
                  /*Cập nhật thanh ghi đệm bằng dữ liệu trạng thái*/
                  write_buf[0] = Data_Display_Type_1[i];
                  /*Xuất thanh ghi đệm sang driver*/
                  write(port_led_fd, write_buf, 1);
                  /*Trì hoãn thời gian bằng time_period (ms)*/
                  usleep(time_period);
            };
      };
/*Trường hợp 2: Hiệu ứng 8 LEDs sáng dần tắt dần*/
/*Phương pháp điều khiển cụng tương tự như trường hợp dầu  tiên*/
   } else if (!strcmp(argv[1], "type2")) {
      for (j=0; j < number_period; j ++) {
            for (i=0; i < 15; i++) {
```

```
                        write_buf[0] = Data_Display_Type_2[i];
                        write(port_led_fd, write_buf, 1);
                        usleep(time_period);
                    };
                };
```

*/*Trường hợp 3: Hiệu ứng 8 LEDs sáng dồn*/*

```
    } else if (!strcmp(argv[1], "type3")) {
        for (j=0; j < number_period; j ++) {
            for (i=0; i < 36; i++) {
                write_buf[0] = Data_Display_Type_3[i];
                write(port_led_fd, write_buf, 1);
                usleep(time_period);
            };
        };
```

*/*Trường hợp 4: Hiệu ứng 8 LEDs dịch chuyển mất dần*/*

```
    } else if (!strcmp(argv[1], "type4")) {
        for (j=0; j < number_period; j ++) {
            for (i=0; i < 36; i++) {
                write_buf[0] = Data_Display_Type_4[i];
                write(port_led_fd, write_buf, 1);
                usleep(time_period);
            };
        };
```

*/*In ra thông báo lỗi trong trường hợp không có lệnh nào hỗ trợ*/*

```
    } else {
        print_usage();
    }
    return 0;
}
```

Cách 2: Điều khiển 8 led bằng lệnh dịch (>> và <<);

*/*Chương trình mang tên 1_3_OtherPortControl_Method_2.c*/*

*/*Khai báo thư viện cần thiết cho các lệnh dùng trong chương trình*/*

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
```

*/*Chương trình cho in thông báo hướng dẫn sử dụng khi có lỗi cú pháp từ người dùng*/*

```
void
print_usage()
{
   printf(     "OtherPortControl     <type1|type2|type3|type4>
   <TimePeriod> <NumberPeriod>\n");
   exit(0);
}
```

*/*Hàm dịch trái "cờ c" vào data 8 bits*/*

```
char shift_left_1_8bits_c(char data, char c) {
   return ((data<<1) | c);
}
```

*/*Hàm dịch phải "cờ c" vào data 8 bits*/*

```
char shift_right_1_8bits_c(char data, char c) {
   return ((data>>1) | (c<<7));
}
```

*/*Hàm dịch trái 1 điểm sáng di chuyển trong data 8 bits*/*

```
char shift_left_1_lighting_led_8bits(char data) {
   if (data == 0) {
         return (data | 1);
   } else {
         return (data<<1);
   }
```

```
}
```

*/*Hàm dịch phải một điểm sáng di chuyển trong data 8 bits*/*

```
char shift_right_1_lighting_led_8bits(char data) {
   if (data == 0) {
        return (data | 0x80);
   } else {
        return (data>>1);
   }
}
```

*/*Hàm dịch trái một diểm sáng di chuyển trong data 8 bits tại vị trí pos (bit 0: pos=1, bit 1: pos=2, …*/*

```
char   shift_left_1_lighting_at_position_led_8bits(char   data,
char pos) {
   if (data == 0) {
        return (data | (1>>pos));
   } else {
        return (data<<1);
   }
}
```

*/*Hàm dịch phải một diểm sáng di chuyển trong data 8 bits tại vị trí pos (bit 0: pos=1, bit 1: pos=2, …*/*

```
char  shift_right_1_lighting_at_position_led_8bits(char  data,
char pos) {
   if (data == 0) {
        return (data | (1<<pos));
   } else {
        return (data>>1);
   }
}
```

*/*Chương trình chính main() khai báo dưới dạng tham số nhập từ người dùng*/*

```
int
main(int argc, char **argv)
```

```
{
   /*Biến lưu số mô tả tập tin thiết bị khi nó được mở*/
   int port_led_fd;
   /*Thanh ghi đệm ghi vào driver và các biến phụ thao tác  bit*/
   char write_buf[1], x, y;
   /*Biến lưu trữ thời gian và số chu kỳ người dùng mong  muốn*/
   long int time_period, number_period;
   /*Các biến đếm hỗ trợ thao tác dữ liệu*/
   int j;
   char i,k,l;
   /*Mở tập tin thiết bị trước khi thao tác*/
   if ((port_led_fd = open("/dev/port_led", O_RDWR)) < 0)
   {
        printf("Error whilst opening /dev/port_led device\n");
        return -1;
   }
   /*Kiểm tra lỗi cú pháp từ người dùng*/
   if (argc != 4) {
        print_usage();
        printf("%d\n",argc);
   }
   /*Lấy thời gian chu kỳ từ người dùng*/
   time_period = atoi(argv[2])*500;
   /*Lấy số chu ỳ mong muốn từ người dùng*/
   number_period = atoi(argv[3]);
   /*So sánh các trường hợp lệnh nhập từ người dùng*/
   /*Trường hợp 1: Hiệu ứng LEDs sáng dần tắt hết*/
   if (!strcmp(argv[1],"type1")) {
        for (j=0; j < number_period; j ++) {
             write_buf[0] = 0x00;
             for (i=0; i<8; i++) {
                  write(port_led_fd, write_buf, 1);
                  usleep(time_period);
```

```
write_buf[0]=shift_left_1_8bits_c(write_buf[0],1);
            }
      };
```

*/*Trường hợp 2: Hiệu ứng LEDs di chuyển sáng dần sau đó tắt dần*/*

```
} else if (!strcmp(argv[1], "type2")) {
      for (j=0; j < number_period; j ++) {
            write_buf[0] = 0x00;
            for (i=0; i<16; i++) {
                  write(port_led_fd, write_buf, 1);
                  usleep(time_period);
                  if (i<8) {

write_buf[0]=shift_left_1_8bits_c(write_buf[0],1);
                  } else {

write_buf[0]=shift_left_1_8bits_c(write_buf[0],0);
                  }
            }
      };
```

*/*Trường hợp 3: Hiệu ứng LEDs di chuyển sáng dồn*/*

```
} else if (!strcmp(argv[1], "type3")) {
      for (j=0; j < number_period; j ++) {
            x = 0x00;
            y = 0x00;
            for (i=10;i>0;i--) {
                  for (k=0;k<(i-2);k++) {
                        x=shift_left_1_lighting_led_8bits(x);
                        write_buf[0] = x | y;
                        write(port_led_fd, write_buf, 1);
                        usleep(time_period);
                  }
                  y = shift_right_1_8bits_c (y,1);
                  x = 0x00;
```

```
                }
        }
    /*Trường hợp 4: Hiệu ứng LEDs di chuyển mất dần*/
    } else if (!strcmp(argv[1], "type4")) {
        for (j=0; j < number_period; j ++) {
            x = 0x00;
            y = 0xFF;
            for (i=0;i<8;i++) {
                y = shift_left_1_8bits_c (y,0);
                x = 0x00;
        do {
        x= shift_right_1_lighting_at_position_led_8bits(x,i);
        write_buf[0] = x | y;
        write(port_led_fd, write_buf, 1);
        usleep(time_period);
        } while (x != 0x01);
            }
        }
    } else {
        print_usage();
    }
    return 0;
}
```

d. Biên dịch và thực thi dự án:

- ***Biên dịch driver:*** Mang tên `port_led_dev.ko`

Chép *driver* đã biên dịch thành công trong bài trước vào kít để chuẩn bị cài đặt vào hệ thống.

- ***Biên dịch application:*** Mang tên `1_3_Othercontrol_Method_1.c` và `1_3_Othercontrol_Method_2.c`

Trỏ vào thư mục chứa tập tin chương trình, biên dịch chương trình ứng dụng với lệnh sau:

```
arm-none-linux-gnueabi-gcc 1_3_Othercontrol_Method_1.c -o Othercontrol_Method_1
```

```
arm-none-linux-gnueabi-gcc 1_3_Othercontrol_Method_2.c -o
Othercontrol_Method_2
```

***Chương trình biên dịch thành công có tên là:* `Othercontrol_Method_1` *và* `Othercontrol_Method_2`.

****Lưu ý: câu lệnh trong shell phải viết trên cùng một dòng.*

- ***Thực thi chương trình:***

  Chép *driver* và chương trình vào kit, thực thi và kiểm tra kết quả;

  - Cài đặt *driver* vào kit theo lệnh sau:

    ```
    insmod port_led_dev.ko
    ```

  - Thay đổi quyền thực thi cho chương trình ứng dụng:

    ```
    chmod 777 Othercontrol_Method_1
    chmod 777 Othercontrol_Method_2
    ```

  - Thực thi và kiểm tra kết quả: Thực thi chương trình ứng dụng theo cú pháp được quy định, quan sát và kiểm tra kết quả.

    ***Chúng ta thấy 8 led nhấp nháy 10 lần với chu kỳ 1s. Các bạn thay đổi chu kỳ và số lần nhấp nháy quan sát kết quả.*

Như vậy, để tạo ra nhiều hiệu ứng điều khiển LEDs khác nhau, việc đơn giản nhất là chỉ việc tạo ra các trạng thái điều khiển mong muốn, lưu vào một vùng nhớ nào đó, công việc cuối cùng là định thời xuất các trạng thái đó ra LEDs hiển thị. Nhưng cách này chỉ hiệu quả khi số lượng LEDs điều khiển là nhỏ và trạng thái LEDs xuất ra không mang tính quy luật rõ ràng. Đối với những trường hợp: số lượng LEDs điều khiển lớn, hiệu ứng mang tính quy luật thì cách thứ 2 là phù hợp nhất. Chúng ta phải áp dụng cách nào là tùy trường hợp, sao cho tốn ít dung lượng bộ nhớ và thời gian thực hiện là nhỏ nhất.

III. Kết luận và bài tập:

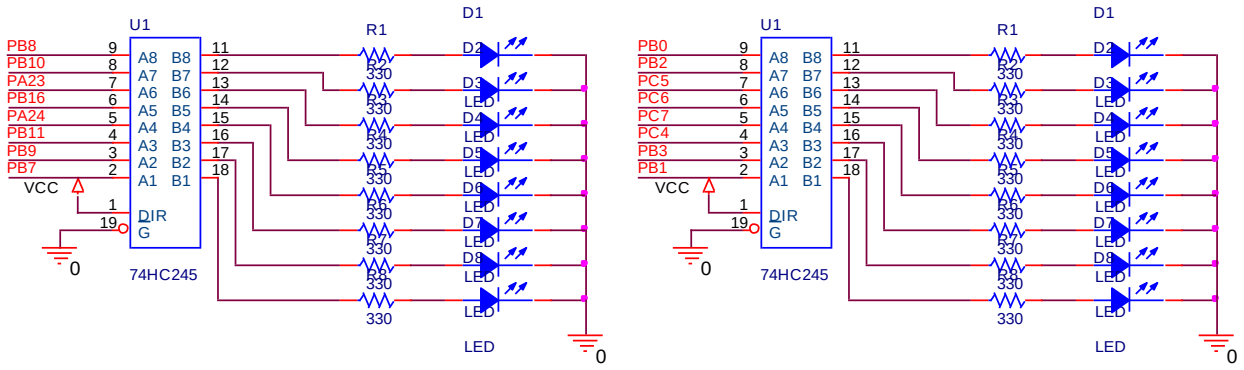
a. Kết luận:

Trong bài này, chúng ta đã viết thành công các chương trình ứng dụng tạo ra nhiều hiệu ứng LEDs khác nhau dựa vào một driver điều khiển LEDs đã có. Đây cũng là mục đích của quyển sách này: Với việc thực hành viết driver và chương trình ứng dụng người học có thể tự mình nghiên cứu một driver có sẵn để viết chương trình ứng dụng phục vụ nhu cầu trong thực tế.

b. Bài tập:

1. Dựa vào driver điều khiển 16 LED đã được hoàn thành trong bài luyện tập trước để viết ứng dụng tạo ra các hiệu ứng điều khiển LEDs tương tự như trong ví dụ trên (nhưng là điều khiển 16 LEDs đơn).

Driver điều khiển theo sơ đồ kết nối sau:



Hình 4-11- Sơ đồ kết nối 16 LEDs đơn.

2. Viết chương trình điều khiển 8 LEDs với cùng yêu cầu như ví dụ trên nhưng có chiều thay đổi, chuyển từ dịch trái sang dịch phải, và ngược lại.

3. Kết hợp các hiệu ứng trên thành một chuỗi hiệu ứng liên tục, kế tiếp nhau không cần phải qua tham số chọn lựa hiệu ứng.

4. Hãy viết chương trình điều khiển 8, 16 LEDs sáng dồn từ ngoài vào trong và từ trong ra ngoài.

5. Hãy viết chương trình điều khiển 8, 16 LEDs dịch chuyển mất dần từ ngoài vào trong và từ trong ra ngoài.

D. CÀI ĐẶT THỜI GIAN DÙNG TIMER:

I. Phác thảo dự án:

Trong những bài trước, để thực hiện trì hoãn thời gian chúng ta thường dùng kỹ thuật dùng các hàm trì hoãn thời gian hỗ trợ sẵn trong *user application*. Thao tác với thời gian còn có một ứng dụng khác rất quan trọng là định thời gian thực hiện tác vụ. Công việc định thời có thể mang tính chu kỳ hoặc không mang tính chu kỳ. Trong bài này chúng ta sẽ áp dụng kỹ thuật định thời gian mang tính chất chu kỳ để cập nhật điều khiển trạng thái LEDs, thay vì dùng kỹ thuật trì hoãn thời gian thông thường.

Định thời gian trong hệ thông Linux có hai cách thực hiện, định thời trong *kernel driver* (không có sự quản lý của hệ điều hành) và định thời trong *user application* (có sự quản lý của hệ điều hành). Cả hai cách điều được sử dụng trong bài này.

a. Yêu cầu dự án:

Nội dung điều khiển LEDs trong bài này không có gì mới, đơn giản vẫn là điều khiển LEDs chớp tắt theo chu kỳ thời gian được quy định bởi người sử dụng khi gọi chương trình ứng dụng thực thi.

Người sử dụng gọi chương trình thực thi theo cú pháp sau:

```
<tên chương trình> <chu kỳ>
```

Trong đó:

`<tên chương trình>` là tên chương trình sau khi được biên dịch;

`<chu kỳ>` là thời gian tính bằng giây do người sử dụng nhập vào;

b. Phân công nhiệm vụ:

1. Timer trong user application:

- *Driver*:

  Làm nhiệm vụ nhận dữ liệu 8 bits từ *user application* để xuất ra LEDs hiển thị, tương tự như *driver* `port_led_dev.ko` trong bài trước.

- *Application*:

  Nhận thông tin khoảng thời gian cập nhật thay đổi trạng thái LEDs. Thông tin này sẽ được chuyển thành thời gian định thời. *User application* sử dụng hàm `alarm()` và hàm `signal()` để khởi tạo định thời. Hàm `alarm()` làm nhiệm vụ cài đặt thời

gian xuất hiện tín hiệu ngắt SIGALRM. Khi xảy ra tín hiệu ngắt SIGALRM, hàm `signal()` sẽ đón bắt và thực thi hàm phục vụ ngắt được người lập trình quy định.

2. Timer trong kernel driver:

- *Driver*:

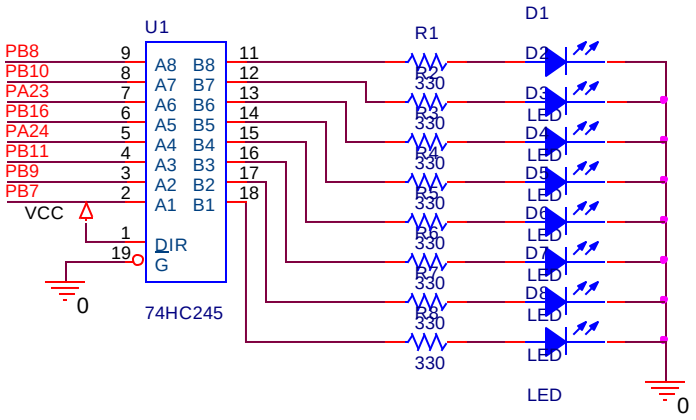
  Nhận thông tin khoảng thời gian (tính bằng s) từ người dùng thông qua *user application*, cài đặt định thời thông qua ngắt thời gian dùng *timer* mềm. Đầu tiên *driver* sẽ tiến hành khởi tạo *timer* dựa vào các hàm thao tác với *timer* đã được tìm hiểu trong những bài trước. Sau mỗi lần đến thời gian định thời, *driver* cập nhật trạng thái LEDs. *Driver* được thay đổi thời gian cập nhật khi giao diện hàm `write()` được gọi bởi *user application*.

- *Application*:

  Lúc này nhiệm vụ của *user application* rất đơn giản, chỉ dùng để cập nhật thời gian định thời cho *driver* thay đổi trạng thái LEDs hiển thị bằng cách gọi giao diện hàm `write()` mỗi khi chương trình thực thi.

II. Thực hiện:

Kết nối phần cứng theo sơ đồ sau:



Hình 4-12- Sơ đồ kết nối 8 LEDs đơn.

1. ***Timer trong user application:***

a. Chương trình driver: Là *driver* `port_led_dev.ko` trong bài trước;

b. Chương trình user application: có tên `1_4_1_TimerLedControl.c`

```
/*Khai báo thư viện cần dùng cho các hàm trong chương trình*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h> //Thư viện cho ham signal()
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
/*Biến lưu số mô tả tập tin thiết bị được mở khi thao tác*/
int port_led_fd;
/*Bộ đệm ghi dữ liệu sang driver xuất ra LEDs*/
char write_buf[1];
/*Biến lưu khoảng thời gian định thời*/
long int time_period;
/*Biến lưu trạng thái đảo LEDs*/
int i=0;
/*Hàm in ra hướng dẫn cho người dùng trong trường hợp nhập  sai cú pháp*/
void
print_usage()
{
   printf("timer_led_control_app <TimePeriod>\n");
   exit(0);
}
/*Hàm xử lý tín hiệu ngắt SIGALRM được cài đặt vào hàm  signal()*/
void catch_alarm(int sig_num) {
     if (i == 0) {
          i = 1;
          /*In thông báo LEDs tắt*/
```

```
            printf ("LEDs are off\n");
            /*Cập nhật trạng thái bộ ghi đệm*/
            write_buf[0] = 0x00;
      } else {
            i  = 0;
            /*In thông báo LEDs sáng*/
            printf ("LEDs are on\n");
            /*Cập nhật trạng thái bộ nhớ đệm */
            write_buf[0] = 0xFF;
      }
      /*Ghi bộ nhớ đệm sang driver, xuất hiển thị LEDs*/
      write(port_led_fd, write_buf, 1);
      /*Cài đặt lại định thời cho tín hiệu SIGALRM*/
      alarm (time_period);
}
/*Chương trình chính, khai báo dưới dạng tham số cho người  dùng lựa chọn nhập tham số thời gian*/
int
main(int argc, char **argv)
{
      /*Trước khi thao tác cần mở tập tin thiết bị driver*/
      if ((port_led_fd = open("/dev/port_led", O_RDWR)) < 0)
      {
         printf("Error whilst opening /dev/port_led device\n");
         return -1;
      }
      /*Kiểm tra lỗi cú pháp nhập từ người dùng*/
      if (argc != 2) {
            print_usage();
      }
      /*Lấy về thông tin thời gian nhập từ người dùng*/
      time_period = atoi(argv[1]);
```

```
        /*Cài đặt bộ định thời tín hiệu SIGALRM gắn với hàm  thực thi ngắt
        catch_alarm()*/
        signal (SIGALRM, catch_alarm);
        /*Quy định khoảng thời gian cho  bộ định thời*/
        alarm (time_period);
        printf ("Go to death loop ...");
        /*Vòng lặp busy, hoặc có thể làm công việc khác trong  khi bộ định thời
        đang hoạt động*/
        while (1);
}
```

c. Biên dịch và thực thi chương trình:

- *Driver*: Chép *driver* `port_led_control.ko` vào kit chuẩn bị cài đặt;
- *Application*: Biên dịch tập tin chương trình bằng lệnh sau:
  `arm-none-linux-gnueabie-gcc 1_3_`*`Timer`*`LedControl_1.c -o `*`Timer`*`LedControl_1`
- Cài đặt *driver* vào hệ thống bằng lệnh sau: insmod port_led_control.ko
- Thực thi chương trình: ./*Timer*LedControl_1.c 1

  Chúng ta thấy LEDs sáng tắt theo chu kỳ 1s, đồng thời in ra câu thông báo trong màn hình hiển thị như sau:

```
./TimerLedControl_1 1
Go to death loop ...
LEDs are off
LEDs are on
LEDs are off
LEDs are on
LEDs are off
...
```

  (Chương trình cứ tiếp tục in ra theo chu kỳ 1s)

  Chúng ta thấy mặc dù chương trình chính đi vào vòng lặp vô tận (Go to death loop ...) nhưng bộ định thời vẫn còn hoạt động và in ra câu thông báo đồng thời

điều khiển LEDs sáng tắt theo chu kỳ 1s cho đến khi người dùng kết thúc chương trình bằng tín hiệu ngắt (bằng cách dùng lệnh ctrl+C).

2. *Timer trong driver:*

a. *Chương trình driver:*

*/*Khai báo thư viện cho các lệnh trong chương trình*/*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <asm/gpio.h>
#include <asm/atomic.h>
#include <asm/atomic.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
#include <linux/time.h> //Thư viện dùng cho timer
#include <linux/jiffies.h> //Thư viện chứa ticks jiffies
```

*/*Tên tập tin thiết bị*/*

```
#define DRVNAME      "timer_led_dev"
#define DEVNAME      "timer_led"
```

/----------Port Control----------*/*

*/*Định nghĩa các chân thao tác tương ứng với chân gpio*/*

```
#define P00          AT91_PIN_PB8
#define P01          AT91_PIN_PB10
#define P02          AT91_PIN_PA23
#define P03          AT91_PIN_PB16
#define P04          AT91_PIN_PA24
#define P05          AT91_PIN_PB11
#define P06          AT91_PIN_PB9
#define P07          AT91_PIN_PB7
#define P0       (P00|P01|P02|P03|P04|P05|P06|P07)
```

```
/*Basic commands*/
/*Định nghĩa các lệnh set và clear PIN căn bản*/
#define SET_P00()                       gpio_set_value(P00,1)
#define SET_P01()                       gpio_set_value(P01,1)
#define SET_P02()                       gpio_set_value(P02,1)
#define SET_P03()                       gpio_set_value(P03,1)
#define SET_P04()                       gpio_set_value(P04,1)
#define SET_P05()                       gpio_set_value(P05,1)
#define SET_P06()                       gpio_set_value(P06,1)
#define SET_P07()                       gpio_set_value(P07,1)

#define CLEAR_P00()                     gpio_set_value(P00,0)
#define CLEAR_P01()                     gpio_set_value(P01,0)
#define CLEAR_P02()                     gpio_set_value(P02,0)
#define CLEAR_P03()                     gpio_set_value(P03,0)
#define CLEAR_P04()                     gpio_set_value(P04,0)
#define CLEAR_P05()                     gpio_set_value(P05,0)
#define CLEAR_P06()                     gpio_set_value(P06,0)
#define CLEAR_P07()                     gpio_set_value(P07,0)
/*Biến sửa lỗi trong quá trình mở thiết bị*/
/* Counter is 1, if the device is not opened and zero (or  less) if opened. */
static atomic_t timer_led_open_cnt = ATOMIC_INIT(1);
/*Khai báo biến cấu trúc lưu timer khởi tạo tên my_timer*/
struct timer_list my_timer;
/*Thời gian khởi tạo ngắt cho mỗi chu kỳ là 1s*/
int time_period=1;
/*Biến cập nhật trạng thái LEDs*/
int i=0;
/*Hàm chuyển dữ liệu 8 bit thành trạng thái tương ứng của  LEDs*/
void timer_led_write_data_port(char data)
{
      (data&(1<<0))? SET_P00():CLEAR_P00();
      (data&(1<<1))? SET_P01():CLEAR_P01();
```

```
        (data&(1<<2))? SET_P02():CLEAR_P02();
        (data&(1<<3))? SET_P03():CLEAR_P03();
        (data&(1<<4))? SET_P04():CLEAR_P04();
        (data&(1<<5))? SET_P05():CLEAR_P05();
        (data&(1<<6))? SET_P06():CLEAR_P06();
        (data&(1<<7))? SET_P07():CLEAR_P07();
}
```

*/*Khai báo và định nghĩa hàm phục vụ ngắt cho timer (my_timer)*/*

```
void my_timer_function (unsigned long data) {
```

*/*Nếu i = 0 thì cho i = 1 cập nhật LEDs tắt và ngược lại để tạo hiệu ứng sáng tắt theo chu kỳ thời gian đã nhập*/*

```
        if (i == 0) {
                i = 1;
                timer_led_write_data_port(0x00);
        } else {
                i = 0;
                timer_led_write_data_port(0xFF);
        }
```

*/*Chu kỳ chớp tắt phải >= 1 */*

```
        if (time_period==0) {
                time_period = 1;
        }
```

*/*Cài đặt lại chu kỳ ngắt sau mỗi lần thực thi hàm phục vụ ngắt, tạo hiệu ứng sáng tắt liên tục, với chu kỳ là time_period giây*/*

```
        mod_timer (&my_timer, jiffies + time_period*HZ);
}
```

*/*Giao diện hàm* write() *làm nhiệm vụ cập nhật thời gian của một chu kỳ ngắt do người sử dụng nhập vào thông qua user application*/*

```
static ssize_t timer_led_write (struct file *filp, char __iomem
buf[], size_t bufsize, loff_t *f_pos)
{
        time_period = buf[0];
```

```
      printk ("Timer period has just been set to %ds\n",
      time_period);
      return bufsize;
}
```

*/*Hàm thực thi khi thiết bị được mở*/*

```
static int
timer_led_open(struct inode *inode, struct file *file)
{
   int result = 0;
   unsigned int dev_minor = MINOR(inode->i_rdev);
   if (!atomic_dec_and_test(&timer_led_open_cnt)) {
      atomic_inc(&timer_led_open_cnt);
      printk(KERN_ERR DRVNAME ": Device with minor ID %d already
      in use\n", dev_minor);
      result = -EBUSY;
      goto out;
   }
out:
   return result;
}
```

*/*Hàm thực thi khi thiết bị được đóng*/*

```
static int
timer_led_close(struct inode * inode, struct file * file)
{
   smp_mb__before_atomic_inc();
   atomic_inc(&timer_led_open_cnt);

   return 0;
}
```

*/*Khai báo và cập nhật tập tin lệnh cho thiết bị*/*

```
struct file_operations timer_led_fops = {
   .write      = timer_led_write,
   .open      = timer_led_open,
   .release   = timer_led_close,
```

```
};
/*Gán tập tin lệnh và tên thiết bị vào tập tin thiết bị sẽ  được tạo*/
static struct miscdevice timer_led_dev = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name          = "timer_led",
        .fops          = &timer_led_fops,
};
/*Hàm được thực thi khi cài đặt driver vào hệ thống*/
static int __init
timer_led_mod_init(void)
{
      /*Cài đặt các thông số cho các PIN muốn thao tác là ngõ  ra*/
      gpio_request (P00, NULL);
      gpio_request (P01, NULL);
      gpio_request (P02, NULL);
      gpio_request (P03, NULL);
      gpio_request (P04, NULL);
      gpio_request (P05, NULL);
      gpio_request (P06, NULL);
      gpio_request (P07, NULL);

      at91_set_GPIO_periph (P00, 1);
      at91_set_GPIO_periph (P01, 1);
      at91_set_GPIO_periph (P02, 1);
      at91_set_GPIO_periph (P03, 1);
      at91_set_GPIO_periph (P04, 1);
      at91_set_GPIO_periph (P05, 1);
      at91_set_GPIO_periph (P06, 1);
      at91_set_GPIO_periph (P07, 1);

      gpio_direction_output(P00, 0);
      gpio_direction_output(P01, 0);
      gpio_direction_output(P02, 0);
      gpio_direction_output(P03, 0);
```

```
        gpio_direction_output(P04, 0);
        gpio_direction_output(P05, 0);
        gpio_direction_output(P06, 0);
        gpio_direction_output(P07, 0);
        /*Các thao tác khởi tạo timer*/
        /*Yêu cầu kernel dành một vùng nhớ lưu timer sắp được  tạo*/
        init_timer (&my_timer);
        /*Cập nhật thời gian sẽ sinh ra ngắt*/
        my_timer.expires = jiffies + time_period*HZ;
        /*Cập nhật dữ liệu cho hàm phục vụ ngắt*/
        my_timer.data = 0;
        /*Gán hàm phục vụ ngắt cho timer*/
        my_timer.function = my_timer_function;
        /*Cuối cùng kích hoạt timer hoạt động trong hệ thống*/
        add_timer (&my_timer);
        /*Đăng ký tập tin thiết bị đã được khai báo trong những  bước đầu tiên*/
        return misc_register(&timer_led_dev);
}
/*Hàm được thực thi khi tháo gỡ thiết bị ra khỏi hệ thống*/
static void __exit
timer_led_mod_exit(void)
{
/*Trước khi tháo gỡ driver ra khỏi hệ thống, chúng ta phải  tháo bỏ timer ra, nếu
không timer vẫn chạy mặc dù driver  không còn tồn tại trong hệ thống nữa. Dẫn
đến phát sinh lỗi  trong những lần cài đặt tiếp theo*/
        del_timer_sync(&my_timer);
/*Thực hiện lệnh tháo bỏ driver ra khỏi hệ thống*/
        misc_deregister(&timer_led_dev);
}
module_init (timer_led_mod_init);
module_exit (timer_led_mod_exit);
MODULE_LICENSE("GPL");
```

```
MODULE_AUTHOR("coolwarmboy");
MODULE_DESCRIPTION("Character device for for generic gpio api");
```

b. *Chương trình application:*

*/*Khai báo các thư viện cần dùng trong chương trình*/*

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

*/*Biến lưu số mô tả tập tin khi thiết bị được mở*/*

```
int port_led_fd;
```

*/*Bộ đệm ghi có kích thước 1 byte*/*

```
char write_buf[1];
```

*/*Biến lưu thời gian nhập vào từ người dùng*/*

```
long int time_period;
```

*/*Hàm in thông báo hương dẫn*/*

```
void
print_usage()
{
   printf("timer_led_control_app <TimePeriod>\n");
   exit(0);
}
```

*/*Hàm main() được sử dụng có tham số*/*

```
int
main(int argc, char **argv)
{
      /*Mở và kiểm tra lỗi trong quá trình mở*/
      if ((port_led_fd = open("/dev/timer_led", O_RDWR)) < 0)
      {
         printf("Error whilst opening /dev/port_led device\n");
         return -1;
      }
```

```
/*Kiểm tra lỗi cú pháp nhập từ người dùng*/
        if (argc != 2) {
              print_usage();
        }
        /*Lưu thông tin thời gian nhập vào từ người dùng*/
        time_period = atoi(argv[1]);
        /*Cập nhật bộ đệm ghi*/
        write_buf[0]  = time_period;
        /*Ghi dữ liệu trong bộ đệm sang driver*/
        write (port_led_fd, write_buf, 1);
        /*Trả về 0 trong trường hợp không xảy ra lỗi trong quá  trình thực thi
        lệnh*/
        return 0;
}
```

c. Biên dịch và thực thi chương trình:

Biên dịch chương trình *driver* và *application* tương tự như trong các bài trước, ở đây chúng ta không nhắc lại.

Chép *driver* và applicaiton vào kit cài đặt và xem kết quả.

Sau khi cài đặt *driver* vào hệ thống, 8 LEDs sẽ sáng tắt với chu kỳ 1s (như đã lập trình). Chúng ta thay đổi tần số chớp tắt bằng cách chay chương trình ứng dụng trên, theo cú pháp đã được quy định trong chương trỉnh:

`<tên chương trình> <thời gian trì hoãn>`

`<tên chương trình>` là tên chương trình ứng dụng sau khi đã biên dịch;

`<thời gian trì hoãn>` là thời gian trì hoãn giữa hai lần thay đổi trạng thái;

III. Kết luận và bài tập:

a. Kết luận:

Trong bài này chúng ta đã thực hành thành công cách sử dụng ngắt thời gian trong cả *driver* và *application*. Mỗi cách khác nhau điều có những ưu và nhược điểm riêng trong quá trình sử dụng. Tuy nhiên hai cách này có một điểm yếu chúng là thời gian phát sinh ngắt tối thiểu là 1s đối với kỹ thuật "alarm" trong *user application*, 10ms đối với *timer* trong *driver (tùy theo định nghĩa của HZ)*. Những ứng dụng đòi hỏi thời gian ngắt ngắn hơn thì không còn phù hợp với kỹ thuật này nữa. Chúng ta sẽ tìm hiểu kỹ thuật khác hiệu quả hơn trong những module điều khiển khác.

b. Bài tập:

1. Viết chương trình sử dụng kỹ thuật "alarm" để lập trình hiệu ứng led sáng dần, tắt dần; sáng dồn; một điểm sáng dịch chuyền mất dần; Với *driver* đã được lập trình trong bài điều khiển 8 LEDs sáng tắt. Chu kỳ ngắt do người sử dụng quy định trong lúc thực thi.
2. Lập trình tương tự với kỹ thuật ngắt dùng *timer* trong *driver*.

***Các bài tập này điều dựa vào sơ đồ kết nối phần cứng trong hai ví dụ trên.*

BÀI 2

GIAO TIẾP ĐIỀU KHIỂN LED 7 ĐOẠN RỜI

I. Phác thảo dự án:

LEDs 7 đoạn là một trong những linh kiện điện tử hiển thị thông tin phổ biến. Các thông tin hiển thị thông thường là các số thập phân, một số trường hợp là số nhị phân, thập lục phân, ... Có nhiều cách hiển thị LEDs 7 đoạn khác nhau, điều khiển trực tiếp thông qua các cổng vào ra, và điều khiển bằng phương pháp quét. Đề đơn giản, trong bài này chúng ta sẽ nghiên cứu cách điều khiển trực tiếp thông qua các cổng xuất mã 7 doạn ra LEDs thông thường. Làm nền tảng cho phương pháp hiển thị bằng phương pháp quét LEDs trong bài sau.

Những kiến thức về LED 7 đoạn đã được nghiên cứu rất kỹ trong những môn học kỹ thuật số căn bản. Nên sẽ không được nhắc lại trong quyển sách này mà chỉ áp dụng vào hướng dẫn thực hành thao tác với hệ thống nhúng.

a. Yêu cầu dự án:

Dự án này sẽ hướng dẫn cách điều khiển 2 LEDs 7 doạn rời hiển thị số nguyên từ 00 đến 99. Người dùng sẽ nhập 3 tham số: giới hạn 1, giới hạn 2 và chu kỳ đếm (tính bằng ms). Hệ thống sẽ tiến hành đếm bắt đầu từ giới hạn 1 đến giới hạn 2 với tốc độ đếm được quy dịnh. Nếu giới hạn 1 lớn hơn giới hạn 2 thì chương trình sẽ thực hiện đếm xuống. Nếu giới hạn 1 nhỏ hơn giới hạn 2 thì chương trình sẽ thực hiện đếm lên. Nếu giới hạn 1 bằng giới hạn 2 thì hiển thị số giới hạn 1 ra LEDs. Cả hai giới hạn đều nằm trong khoảng từ 00 đến 99. Cú pháp lệnh khi thực thi như sau:

```
<tên chương trình> <giới hạn 1> <giới hạn 2> <thời gian trì hoãn>
```

Trong đó:

`<tên chương trình>` là tên chương trình sau khi đã biên dịch thành công;

`<giới hạn 1>` là giới hạn đầu tiên do người dùng nhập vào từ 00 đến 99;

`<giới hạn 2>` là giới hạn sau do người dùng nhập vào có giá trị từ 00 đến 99;

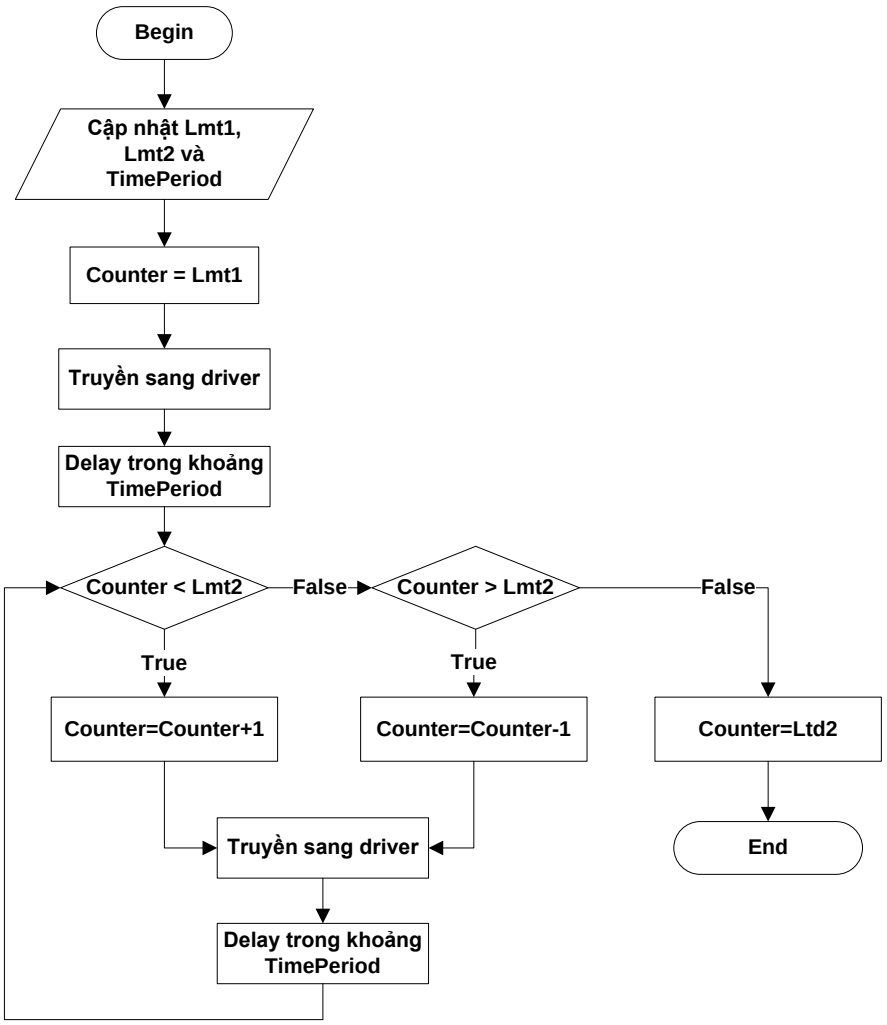
b. Phân công nhiệm vụ:

- *Driver*:

Áp dụng giao diện hàm write() nhận dữ liệu là số nguyên từ 00 đến 99 từ *user application*, giải mã sang số BCD trước khi chuyển thành mã 7 đoạn để hiển thị ra LEDs.

- *Application*:

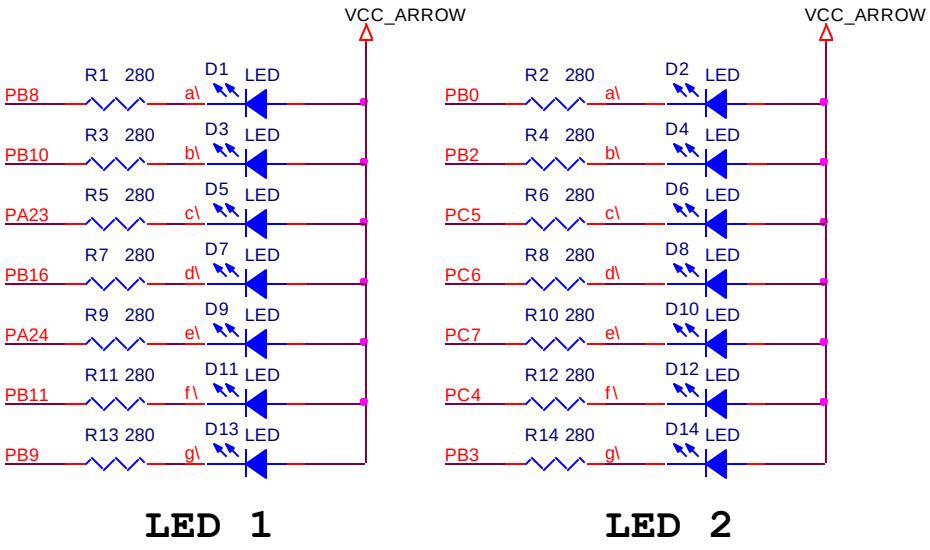
Nhận các tham số từ người dùng nhập khi gọi chương trình thực thi. Thực hiện đếm lên hay đếm xuống tùy thuộc vào giá trị nhận được từ tham số người dùng. Dùng giao diện hàm write() truyền số từ 00 đến 99 cho *driver* hiển thị ra LEDs. Chương trình *application* thực hiện theo lưu đồ sau:



Hình 4-13- Lưu đồ điều khiển đếm hiển thị LED 7 đoạn rời.

II. Thực hiện:

1. Kết nối phần cứng theo sơ đồ sau:



Hình 4-14- Sơ đồ kết nối 2 LEDs 7 đoạn rời.

2. ***Viết chương trình:***

- ***Driver:*** Tên 2_Dis_Seg_led_dev.c

*/*Khai báo thư viện cần dùng cho các hàm trong chương trình*/*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <asm/gpio.h>
#include <asm/atomic.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
```

*/*Tên driver thiết bị*/*

```
#define DRVNAME      "dis_seg_led_dev"
#define DEVNAME      "dis_seg_led"
```

*/*Khai báo các chân LEDs sử dụng tương ứng với chân trong chip*/*

*/*Khai báo các chân của LEDs thứ nhất (LEDs chục)*/*

```
#define A1            AT91_PIN_PB8
#define B1            AT91_PIN_PB10
```

```
#define C1              AT91_PIN_PA23
#define D1              AT91_PIN_PB16
#define E1              AT91_PIN_PA24
#define F1              AT91_PIN_PB11
#define G1              AT91_PIN_PB9
```

*/*Khai báo các chân của LEDs thứ hai (LEDs đơn vị)*/*

```
#define A2              AT91_PIN_PB0
#define B2              AT91_PIN_PB2
#define C2              AT91_PIN_PC5
#define D2              AT91_PIN_PC6
#define E2              AT91_PIN_PC7
#define F2              AT91_PIN_PC4
#define G2              AT91_PIN_PB3
```

*/*Các lệnh set và clear bit cho các chân của LEDs 7 đoạn*/*

*/*Basic commands*/*

*/*Các lệnh cho LEDs 1 */*

```
#define SET_A1()                gpio_set_value(A1,1)
#define SET_B1()                gpio_set_value(B1,1)
#define SET_C1()                gpio_set_value(C1,1)
#define SET_D1()                gpio_set_value(D1,1)
#define SET_E1()                gpio_set_value(E1,1)
#define SET_F1()                gpio_set_value(F1,1)
#define SET_G1()                gpio_set_value(G1,1)

#define CLEAR_A1()              gpio_set_value(A1,0)
#define CLEAR_B1()              gpio_set_value(B1,0)
#define CLEAR_C1()              gpio_set_value(C1,0)
#define CLEAR_D1()              gpio_set_value(D1,0)
#define CLEAR_E1()              gpio_set_value(E1,0)
#define CLEAR_F1()              gpio_set_value(F1,0)
#define CLEAR_G1()              gpio_set_value(G1,0)
```

*/*Các lệnh cho LEDs 2*/*

```
#define SET_A2()                gpio_set_value(A2,1)
```

```
#define SET_B2()                gpio_set_value(B2,1)
#define SET_C2()                gpio_set_value(C2,1)
#define SET_D2()                gpio_set_value(D2,1)
#define SET_E2()                gpio_set_value(E2,1)
#define SET_F2()                gpio_set_value(F2,1)
#define SET_G2()                gpio_set_value(G2,1)

#define CLEAR_A2()              gpio_set_value(A2,0)
#define CLEAR_B2()              gpio_set_value(B2,0)
#define CLEAR_C2()              gpio_set_value(C2,0)
#define CLEAR_D2()              gpio_set_value(D2,0)
#define CLEAR_E2()              gpio_set_value(E2,0)
#define CLEAR_F2()              gpio_set_value(F2,0)
#define CLEAR_G2()              gpio_set_value(G2,0)

static atomic_t dis_seg_led_open_cnt = ATOMIC_INIT(1);
/*Định nghĩa mã LEDs 7 đoạn tích cực mức thấp*/
static int SSC[10] =
{0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
/*Hàm chuyển đổi dữ liệu 8 bit thành các trạng thái LEDs  tương ứng trong LEDs 7 đoạn*/
/*Dành cho LEDs 1*/
void dis_seg_led_write_data_led_1(char data)
{
      (data&(1<<0))? SET_A1():CLEAR_A1();
      (data&(1<<1))? SET_B1():CLEAR_B1();
      (data&(1<<2))? SET_C1():CLEAR_C1();
      (data&(1<<3))? SET_D1():CLEAR_D1();
      (data&(1<<4))? SET_E1():CLEAR_E1();
      (data&(1<<5))? SET_F1():CLEAR_F1();
      (data&(1<<6))? SET_G1():CLEAR_G1();
}
```

```
/*Hàm chuyển đổi dành cho LEDs 2*/
void dis_seg_led_write_data_led_2(char data)
{
      (data&(1<<0))? SET_A2():CLEAR_A2();
      (data&(1<<1))? SET_B2():CLEAR_B2();
      (data&(1<<2))? SET_C2():CLEAR_C2();
      (data&(1<<3))? SET_D2():CLEAR_D2();
      (data&(1<<4))? SET_E2():CLEAR_E2();
      (data&(1<<5))? SET_F2():CLEAR_F2();
      (data&(1<<6))? SET_G2():CLEAR_G2();
}
/*Hàm giải mã số nguyên từ 00 đến 99 sang số BCD và viết  trạng thái ra lEDs vật lý*/
void dis_seg_led_decode_and_write_to_led(char data) {
dis_seg_led_write_data_led_1(SSC[data/10]);
dis_seg_led_write_data_led_2(SSC[data%10]);
}
/*Giao diện hàm write() nhận số nguyên từ user application  hiển thị ra LEDs 7 đoạn*/
static ssize_t dis_seg_led_write (struct file *filp, char __iomem
buf[], size_t bufsize, loff_t *f_pos)
{
      /*Gọi hàm chuyển đổi dữ liệu ghi ra LEDs*/
      dis_seg_led_decode_and_write_to_led(buf[0]);
      return bufsize;
}
static int
dis_seg_led_open(struct inode *inode, struct file *file)
{
   int result = 0;
   unsigned int dev_minor = MINOR(inode->i_rdev);
   if (!atomic_dec_and_test(&dis_seg_led_open_cnt)) {
       atomic_inc(&dis_seg_led_open_cnt);
```

```
        printk(KERN_ERR DRVNAME ": Device with minor ID %d already
        in use\n", dev_minor);
        result = -EBUSY;
        goto out;
    }
out:
    return result;
}
static int
dis_seg_led_close(struct inode * inode, struct file * file)
{
    smp_mb__before_atomic_inc();
    atomic_inc(&dis_seg_led_open_cnt);
    return 0;
}
struct file_operations dis_seg_led_fops = {
    .write      = dis_seg_led_write,
    .open      = dis_seg_led_open,
    .release   = dis_seg_led_close,
};
static struct miscdevice dis_seg_led_dev = {
        .minor         = MISC_DYNAMIC_MINOR,
        .name         = "dis_seg_led",
        .fops         = &dis_seg_led_fops,
};
static int __init
dis_seg_led_mod_init(void)
{
      /*Cài đặt các chân gpio sử dụng thành thành ngõ ra*/
      gpio_request (A1, NULL);
      gpio_request (B1, NULL);
      gpio_request (C1, NULL);
      gpio_request (D1, NULL);
      gpio_request (E1, NULL);
```

```
gpio_request (F1, NULL);
gpio_request (G1, NULL);

at91_set_GPIO_periph (A1, 1);
at91_set_GPIO_periph (B1, 1);
at91_set_GPIO_periph (C1, 1);
at91_set_GPIO_periph (D1, 1);
at91_set_GPIO_periph (E1, 1);
at91_set_GPIO_periph (F1, 1);
at91_set_GPIO_periph (G1, 1);

gpio_direction_output(A1, 0);
gpio_direction_output(B1, 0);
gpio_direction_output(C1, 0);
gpio_direction_output(D1, 0);
gpio_direction_output(E1, 0);
gpio_direction_output(F1, 0);
gpio_direction_output(G1, 0);

gpio_request (A2, NULL);
gpio_request (B2, NULL);
gpio_request (C2, NULL);
gpio_request (D2, NULL);
gpio_request (E2, NULL);
gpio_request (F2, NULL);
gpio_request (G2, NULL);

at91_set_GPIO_periph (A2, 1);
at91_set_GPIO_periph (B2, 1);
at91_set_GPIO_periph (C2, 1);
at91_set_GPIO_periph (D2, 1);
at91_set_GPIO_periph (E2, 1);
at91_set_GPIO_periph (F2, 1);
at91_set_GPIO_periph (G2, 1);
```

```
        gpio_direction_output(A2, 0);
        gpio_direction_output(B2, 0);
        gpio_direction_output(C2, 0);
        gpio_direction_output(D2, 0);
        gpio_direction_output(E2, 0);
        gpio_direction_output(F2, 0);
        gpio_direction_output(G2, 0);
        return misc_register(&dis_seg_led_dev);
}
static void __exit
dis_seg_led_mod_exit(void)
{
   misc_deregister(&dis_seg_led_dev);
}

module_init (dis_seg_led_mod_init);
module_exit (dis_seg_led_mod_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("coolwarmboy");
MODULE_DESCRIPTION("Character device for for generic gpio api");
```

- ***Application:***

```
/*Khai báo thư viện cần thiết cho các lệnh cần dùng trong chương trình*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
/*Hàm in ra hướng dẫn khi người dùng nhập sai cú pháp*/
void
print_usage()
{
   printf("dis_seg_led_app <lmt1> <lmt2> <time_period>\n");
   exit(0);
}

/*Chương trình chính, khai báo dạng tham số cho người dùng*/
int
main(int argc, char **argv)
{
      /*Biến lưu số mô tả tập tin, khi tập tin thiết bị được  mở*/
      int dis_seg_led_fd;
      /*Bộ đệm ghi*/
      char write_buf[1];
      /*Thời gian thay đổi trạng thái do ngnười dùng nhập  vào*/
      long int  time_period;
      /*Các biến phục vụ cho quá trình đếm*/
      char counter, lmt1, lmt2;
      /*Mở tập tin thiết bị trước khi thao tác*/
      if ((dis_seg_led_fd = open("/dev/dis_seg_led", O_RDWR)) <
      0)
      {
         printf("Error whilst opening /dev/dis_seg_led device\n");
```

```
            return -1;
        }
        /*Kiểm tra lỗi cú pháp từ người dùng*/
        if (argc != 4) {
            print_usage();
        }
        /*Cập nhật các tham số cần thiết từ người dùng*/
        lmt1 = atoi(argv[1]);
        lmt2 = atoi(argv[2]);
        time_period = atoi(argv[3])*1000;
        /*Cho counter bằng giời hạn 1 */
        counter = lmt1;
        /*Cập nhật trạng thái LEDs lần đầu tiên*/
        write_buf[0] = counter;
        write(dis_seg_led_fd,write_buf,1);
        usleep(time_period);
        /*Thực hiện đếm theo yêu cầu của thuật toán đã trình  bày*/
        do {
            if (counter < lmt2) {
                counter++;
            } else {
                counter--;
            }
            write_buf[0] = counter;
            write(dis_seg_led_fd,write_buf,1);
            usleep(time_period);
        /*Liên tục đếm cho đến khi counter bằng giới hạn 2 */
        } while (counter != lmt2);
        /*Trả về 0 khi không có lỗi trong quá trình thực thi  chương trình */
        return 0;
}
```

3. Biên dịch và thực thi chương trình:

Biên dịch thực thi và chạy chương trình trên kit, quan sát kết quả.

III. Kết luận và bài tập:

a. Kết luận:

Trong phần này chúng ta đã điều khiển thành công module 2 LEDs 7 đoạn rời để hiển thị số các số nguyên trong khoảng từ 00 đến 99 bằng các chân gpio thông qua giao diện hàm write giao tiếp giữa *application* và *driver*. Trong trường hợp có nhiều LEDs số lượng IO không đủ, thì chúng ta sẽ chuyển sang dùng phương pháp khác tối ưu hơn. Cũng dùng số lượng chân là 16, chúng ta có thể điều khiển 8 LEDs 7 đoạn. Chúng ta sẽ nghiên cứu cách điều khiển LEDs bằng phương pháp quét trong bài sau.

b. Bài tập:

1. Viết chương trình *driver* để điều khiển 2 LEDs 7 đoạn hiển thị thông tin là số hex trong khoảng từ 00-FF. Sau đó viết chương trình *application* với cùng ý tưởng như trong bài ví dụ trên:
   - Người dùng nhập vào hai tham số: Giới hạn 1 và giới hạn 2, là hai số hex trong khoảng từ 00-FF;
   - Chương trình thực hiện đếm lên hay đếm xuống từ giới hạn 1 đến giới hạn 2;
   - Khoảng thời gian giữa hai lần thay đổi giá trị được quy định bởi người dùng.
2. Mở rộng *driver* và *application* trong ví dụ trên để có thể hiển thị số âm. Lúc này giá trị có thể hiển thị trên LEDs có thể nằm trong khoảng từ -9 đến 99;

BÀI 3

GIAO TIẾP ĐIỀU KHIỂN LED 7 ĐOẠN BẰNG PHƯƠNG PHÁP QUÉT

I. Phác thảo dự án:

Bài trước chúng ta đã điều khiển thành công 2 LEDs 7 đoạn dùng phương pháp thông thường là xuất Port điều khiển trực tiếp. Phương pháp này tuy đơn giản nhưng số lương io điều khiển là rất lớn trong trường hợp có nhiều LEDs cần điều khiển đồng thời. Bài này chúng ta sẽ tìm sử dụng phương pháp quét để điều khiển đồng thời 8 LEDs 7 đoạn hiển thị thông tin.

Phương pháp quét LEDs được chúng ta nghiên cứu rất kỹ trong môn học thực tập vi xử lý 89XX51. Giáo trình này sẽ áp dụng thuật toán này vào lập trình điều khiển trong hệ thống nhúng.

Trong bài này chúng ta sẽ sử dụng kỹ thuật ngắt thời gian mới, dùng timer vật lý được tích hợp sẵn trong vi điều khiển. Do timer được sử dụng trong chương trình liên quan đến những chức năng của hệ điều hành nên chúng ta phải tự lập trình chức năng này thay vì dùng những hàm được hỗ trợ sẵn.

a. Yêu cầu dự án:

Dự án này điều khiển 8 LEDs 7 đoạn bằng phương pháp quét. Với phương pháp điều khiển này, chúng ta sẽ lập trình hiển thị nhiều hiệu ứng khác nhau:

- Hiển thị số "07101080" ra 8 LEDs;
- Đếm hiển thị từ XX đến YY với chu kỳ Z do người dùng quy định;
- Đếm giờ phút giây hiển thị 8 LEDs;

b. Phân công nhiệm vụ:

- *Driver*: Trong dự án này chúng ta sử dụng chung một *driver*. *Driver* này có những đặc điểm sau:
  - *Driver* dùng phương pháp ngắt thời gian timer 0, đây là timer vật lý tích hợp trong vi điều khiển. Tạo chu kỳ ngắt 1ms, khi đến thời điểm ngắt, chương trình sẽ cập

nhật thay đổi trạng thái quét LEDs hiển thị theo dữ liệu đã được lưu trữ trong bộ nhớ đệm.

- *Driver* sử dụng hai giao diện hàm: ioctl() và write(). Trong đó:
  - Giao diện hàm write() nhận mảng số nguyên trong khoảng từ 0 đến 9 bao gồm 8 phần tử tương ứng với 8 LEDs 7 đoạn cần hiển thị. Hàm sẽ cập nhật dữ liệu trong bộ nhớ đệm hiển thị ngay sau khi nhận được mảng thông tin từ người dùng. Như vậy nếu *user application* muốn hiển thị bất kỳ số nào ra LEDs thì chỉ cần gán số đó vào bộ đệm ghi, sau đó gọi giao diện hàm write() chuyển thông tin từ bộ đệm ghi trong *user* sang bộ đệm nhận trong *driver* để hiển thị ra LEDs.

    ***Bên cạnh các số từ 0 đến 9, driver còn hỗ trợ hiển thị thêm các ký từ đặt biệt, được quy định bởi những mã số khác nhau: Ký tự "-" được quy định bởi mã số 10, ký tự "_" được quy định bởi mã số 11, ký tự " " (khoảng trắng) được quy định bởi mã số 12, và một số ký tự khác nếu muốn chúng ta có thể thêm vào.*
  - Giao diện hàm ioctl() thực hiện nhiều chức năng khác nhau:
    1. Trì hoãn thời gian với độ phân giải 10ms. Người dùng gọi giao diện ioctl() với tham số lệnh SWEEP_LED_DELAY và khoảng thời gian trì hoãn tương ứng để thực hiện trì hoãn theo yêu cầu.
    2. Nhận số nguyên (00000000 đến 99999999) từ *user application* giải mã hiển thị ra LEDs 7 doạn. Người dùng gọi giao diện ioctl() với tham số lệnh SWEEP_LED_NUMBER_DISPLAY và số nguyên muốn hiển thị. Chương trình *driver* sẽ nhận thông tin này, giải mã và ghi vào bộ nhớ đệm hiển thị trong *driver* để quét ra LEDs.
    3. Nhận 3 số nguyên giờ, phút, giây từ *user application*, giải mã và ghi vào bộ đệm hiển thị ra LEDs đưới dạng "HH-MM-SS". Chức năng này có tham số lệnh là SWEEP_LED_TIME_DISPLAY.

- *Application*:

  Trong dự án này chúng ta sẽ xây dựng một chương trình *application* tổng hợp thực hiện tất cả các hiệu ứng điều khiển LEDs 7 đoạn trên. *Application* sẽ được xây dựng dựa vào cấu trúc hàm main() có nhiều tham số để người dùng nhập tham số thực thi chương trình. Chương trình sẽ nhận tham số này, lựa chọn trường hợp và đáp ứng đúng yêu cầu. Cụ thể từng chức năng như sau:

  - Hiển thị dãy số "01234567" ra 8 LEDs 7 đoạn:

    Người dùng nhập dòng lệnh theo cú pháp sau:

    ```
    ./<Tên chương trình> display_number
    ```

    Trong đó:

    `<Tên chương trình>` là tên chương trình *application* sau khi biên dịch;

    `displaynumber` là tham số yêu cầu xuất dãy số;

    Người lập trình chỉ cần khai báo bộ đệm ghi là một mảng bao gồm 8 thành phần, mỗi phần tử tương đương với 1 LED. Cập nhật bộ đệm ghi là các số từ 0 đến 7. Cuối cùng gọi giao diện hàm write() để ghi bộ đệm ghi sang bộ đệm nhận trong dirver hiển thị ra LEDs.

  - Đếm hiển thị từ XX đến YY với chu kỳ T:

    Người dùng nhập dòng lệnh shell theo cú pháp:

    ```
    ./<Tên chương trình> count_number XX YY T
    ```

    Trong đó:

    `XX`: Là số giới hạn 1;

    `YY`: Là số giới hạn 2;

    `T`: Là chu kỳ (đơn vị là 10ms) thay đổi trạng thái.

    Chương trình sẽ đếm từ XX đến YY với chu kỳ đếm là Tx10ms. Mỗi lần cập nhật trạng thái chương trình sẽ gọi giao diện hàm `ioctl()` để yêu cầu *driver* giải mã hiển thị LEDs.

  - Đếm giờ phút giây hiển thị 8 LEDs 7 đoạn:

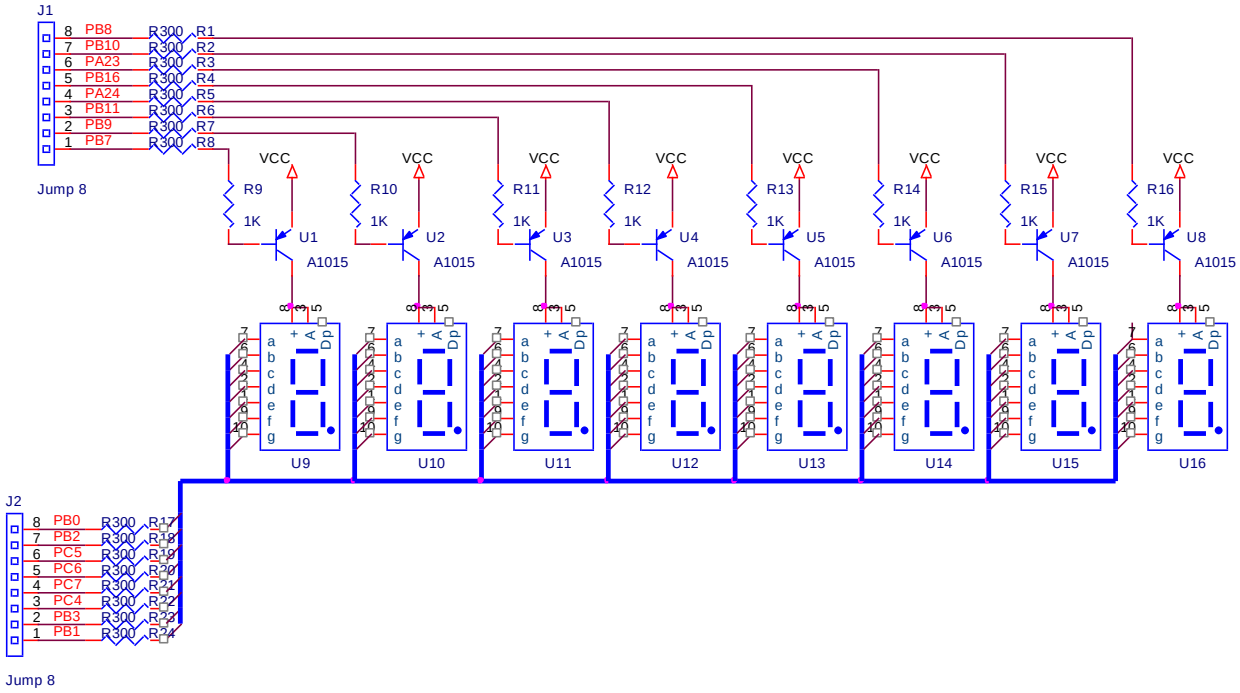
    Người dùng nhập dòng lệnh shell theo cú pháp:

    ```
    ./<Tên chương trình> count_time HH MM SS
    ```

Chương trình thực hiện đếm giờ phút giây bắt đầu , sau mỗi chu kỳ 1s thông tin này sẽ được truyền qua *driver*. Tại đây *driver* sẽ giải mã và hiển thị ra LEDs 7 đoạn theo dạng "HH-MM-SS".

II. Thực hiện:

1. *Kết nối phần cứng theo sơ đồ sau:*



Hình 4-15- Sơ đồ kết nối 8 LEDs 7 đoạn bằng phương pháp quét.

2. *Chương trình driver:*

*/*Khai báo thư viện cần thiết cho các lệnh cần dùng trong chương trình */*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <mach/at91_tc.h>
#include <asm/gpio.h>
#include <asm/atomic.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
#include <asm/uaccess.h>
```

```
/*Các thư viện hỗ trợ cho ngắt và trì hoãn thời gian*/
#include <linux/interrupt.h> //Chứa hàm hỗ trợ khai báo ngắt
#include <linux/clk.h> //Chứa hàm khởi tạo clock
#include <linux/irq.h>
#include <linux/time.h>
#include <linux/jiffies.h>
#include <linux/sched.h>
#include <linux/delay.h>
/*Tên driver thiết bị*/
#define DRVNAME      "sweep_seg_led_dev"
#define DEVNAME      "sweep_seg_led"

            /*-------------Port Control-----------*/
/*Khai báo các chân điều khiển LEDs 7 đoạn*/
/*Các chân điều khiển tích cực cho LEDs*/
#define P00             AT91_PIN_PB8
#define P01             AT91_PIN_PB10
#define P02             AT91_PIN_PA23
#define P03             AT91_PIN_PB16
#define P04             AT91_PIN_PA24
#define P05             AT91_PIN_PB11
#define P06             AT91_PIN_PB9
#define P07             AT91_PIN_PB7
/*Các chân điều khiển truyền dữ liệu cho LEDs tích cực hiển thị*/
#define A               AT91_PIN_PB0
#define B               AT91_PIN_PB2
#define C               AT91_PIN_PC5
#define D               AT91_PIN_PC6
#define E               AT91_PIN_PC7
#define F               AT91_PIN_PC4
#define G               AT91_PIN_PB3
/*Các lệnh cơ bản điều khiển set() và clear() bit cho các chân gpio*/
                    /*Basic commands*/
```

*/*Lệnh set() và clear() bit cho các chân chọn LEDs*/*

```
#define SET_P00()                gpio_set_value(P00,1)
#define SET_P01()                gpio_set_value(P01,1)
#define SET_P02()                gpio_set_value(P02,1)
#define SET_P03()                gpio_set_value(P03,1)
#define SET_P04()                gpio_set_value(P04,1)
#define SET_P05()                gpio_set_value(P05,1)
#define SET_P06()                gpio_set_value(P06,1)
#define SET_P07()                gpio_set_value(P07,1)

#define CLEAR_P00()              gpio_set_value(P00,0)
#define CLEAR_P01()              gpio_set_value(P01,0)
#define CLEAR_P02()              gpio_set_value(P02,0)
#define CLEAR_P03()              gpio_set_value(P03,0)
#define CLEAR_P04()              gpio_set_value(P04,0)
#define CLEAR_P05()              gpio_set_value(P05,0)
#define CLEAR_P06()              gpio_set_value(P06,0)
#define CLEAR_P07()              gpio_set_value(P07,0)
```

*/*Các lệnh set() và clear() bit cho các chân dữ liệu LEDs*/*

```
#define SET_A()                  gpio_set_value(A,1)
#define SET_B()                  gpio_set_value(B,1)
#define SET_C()                  gpio_set_value(C,1)
#define SET_D()                  gpio_set_value(D,1)
#define SET_E()                  gpio_set_value(E,1)
#define SET_F()                  gpio_set_value(F,1)
#define SET_G()                  gpio_set_value(G,1)

#define CLEAR_A()                gpio_set_value(A,0)
#define CLEAR_B()                gpio_set_value(B,0)
#define CLEAR_C()                gpio_set_value(C,0)
#define CLEAR_D()                gpio_set_value(D,0)
#define CLEAR_E()                gpio_set_value(E,0)
#define CLEAR_F()                gpio_set_value(F,0)
#define CLEAR_G()                gpio_set_value(G,0)
```

```
/*Định nghĩa các số định danh lệnh cho giao diện hàm ioctl() */
#define SWEEP_LED_DEV_MAGIC  'B'
#define SWEEP_LED_DELAY _IO(SWEEP_LED_DEV_MAGIC, 0)
#define SWEEP_LED_NUMBER_DISPLAY _IO(SWEEP_LED_DEV_MAGIC, 1)
#define SWEEP_LED_TIME_DISPLAY _IO(SWEEP_LED_DEV_MAGIC, 2)

/* Counter is 1, if the device is not opened and zero (or  less) if opened. */
static atomic_t sweep_seg_led_open_cnt = ATOMIC_INIT(1);
/*Bộ nhớ đệm hiển thị trong driver, chứa những số nguyên được  giải mã sang LEDs 7 đoạn*/
unsigned char DataDisplay[8]={0,1,2,3,4,5,6,7};
/*Bộ giải mã LEDs 7 đoạn, bao gồm các mã 7 đoạn từ 0 đến 9,  các ký tự đặc biệt "-", "_" và ký tự rỗng*/
unsigned char SevSegCode[]= { 0xC0, 0xF9, 0xA4, 0xB0, 0x99,
0x92, 0x82, 0xF8, 0x80, 0x90, 0x3F, 0x77, 0xFF};
/*Biến lưu trạng thái LEDs tích cực*/
int i;
/*Con trỏ nền của thanh ghi điều khiển timer0 tích hợp trong vi  điều khiển*/
void __iomem *at91tc0_base;
/*Con trỏ cấu trúc clock của timer0*/
struct clk *at91tc0_clk;
/*Hàm trì hoãn thời gian dùng jiffies trong kernel, thay thế cho các hàm trì hoãn thời gian trong user; Hàm trì hoãn thời gian có độ phân giải 1ms*/
void sweep_led_delay(unsigned int delay) {
   /*Khai báo biến lưu thời điểm tương lai cần trì hoãn*/
   long int time_delay;
   /*Cập nhật thời điểm tương lai cần trì hoãn*/
   time_delay = jiffies + delay;
   /*So sánh thời điểm hiện tại với thời điểm tương lai*/
   while (time_before(jiffies, time_delay)) {
         /*Thực hiện chia tiến trình trong quá trình trì hoãn thời gian*/
         schedule();
```

```
        }
}
```

*/*Hàm ghi dữ liệu 8 bits cho LEDs tích cực hiển thị*/*

```
void sweep_seg_led_write_data_active_led(char data)
{
   (data&(1<<0))? SET_P00():CLEAR_P00();
   (data&(1<<1))? SET_P01():CLEAR_P01();
   (data&(1<<2))? SET_P02():CLEAR_P02();
   (data&(1<<3))? SET_P03():CLEAR_P03();
   (data&(1<<4))? SET_P04():CLEAR_P04();
   (data&(1<<5))? SET_P05():CLEAR_P05();
   (data&(1<<6))? SET_P06():CLEAR_P06();
   (data&(1<<7))? SET_P07():CLEAR_P07();
}
```

*/*Hàm ghi dữ liệu mã 7 đoạn cho LEDS hiển thị*/*

```
void sweep_seg_led_write_data_led(char data)
{
   (data&(1<<0))? SET_A():CLEAR_A();
   (data&(1<<1))? SET_B():CLEAR_B();
   (data&(1<<2))? SET_C():CLEAR_C();
   (data&(1<<3))? SET_D():CLEAR_D();
   (data&(1<<4))? SET_E():CLEAR_E();
   (data&(1<<5))? SET_F():CLEAR_F();
   (data&(1<<6))? SET_G():CLEAR_G();
}
```

*/*Hàm chọn LEDs tích cực*/*

```
void active_led_choice(char number) {
   sweep_seg_led_write_data_active_led(~(1<<(number)));
}
```

*/*Hàm xuất dữ liệu cho LEDs hiển thị*/*

```
void data_led_stransmitt (char data) {
   sweep_seg_led_write_data_led (SevSegCode[data]);
}
```

*/*Giải mã số 8 chữ số sang từng ký tự số BCD, lưu vào vùng nhớ đệm hiển thị LEDs*/*

```
void sweep_led_number_display (unsigned long int data) {
   DataDisplay[0] = data % 10;
   DataDisplay[1] = (data % 100)/10;
   DataDisplay[2] = (data % 1000)/100;
   DataDisplay[3] = (data % 10000)/1000;
   DataDisplay[4] = (data % 100000)/10000;
   DataDisplay[5] = (data % 1000000)/100000;
   DataDisplay[6] = (data % 10000000)/1000000;
   DataDisplay[7] = data/10000000;
}
```

*/*Giải mã giờ phút giây thành những ký số BCD, và ký tự đặc biệt lưu vào vùng nhớ đệm hiển thị LEDs*/*

```
void sweep_led_time_display(int hh, int mm, int ss) {
   DataDisplay[0] = ss%10;
   DataDisplay[1] = ss/10;
   DataDisplay[2] = 10; //Ký tự "-"
   DataDisplay[3] = mm%10;
   DataDisplay[4] = mm/10;
   DataDisplay[5] = 10; //Ký tự "-"
   DataDisplay[6] = hh%10;
   DataDisplay[7] = hh/10;
}
```

*/*Hàm phục vụ ngắt timer thực hiện quét LEDs 7 đoạn, tại một thời điểm chỉ hiển thị 1 LEDs với mã 7 đoạn của LED đó*/*

```
static irqreturn_t at91tc0_isr(int irq, void *dev_id) {
   int status;
   /*Đọc thanh ghi trạng thái của timer0 reset lại timer sau khi xảy ra ngắt*/
   status = ioread32(at91tc0_base + AT91_TC_SR);
   /*Truyền dữ liệu cho LEDs i*/
   data_led_stransmitt(DataDisplay[i]);
   /*Chọn tích cực cho LEDs i*/
```

```
    active_led_choice(i);
    /*Cập nhật biến trạng thái cho LEDs tiếp theo*/
    i++;
    /*Giới hạn số LEDs hiển thị*/
    if (i==8) i = 0;
    /*Kết thúc quá trình ngắt trở lại chương trỉnh driver chính*/
    return IRQ_HANDLED;
}
/*Khai báo và định nghĩa giao diện hàm write(), nhận dữ liệu từ user application là một mảng 8 bytes cập nhật vào bộ nhớ đệm ghi trong driver*/
static ssize_t dis_seg_led_write (struct file *filp, unsigned char __iomem buf[], size_t bufsize, loff_t *f_pos)
{
    /*Bộ nhớ đệm ghi nhận dữ liệu từ user*/
    unsigned char write_buf[8];
    /*Biến lưu kích thước trả về khi ghi thành công*/
    int write_size = 0;
    int i;
    /*Thực hiện hàm copy_from_user() nhận dữ liệu từ user đến driver */
    if (copy_from_user (write_buf, buf, 8) != 0) {
          return -EFAULT;
    } else {
          write_size = bufsize;
    }
    /*Chuyển dữ liệu từ bộ đệm ghi sang bộ đệm hiển thị LEDs */
    for (i=0; i<8; i++) {
          DataDisplay[i] = write_buf [i];
    }
    return write_size;
}

/*Khai báo và định nghĩa giao diện hàm ioctl()*/
static int
```

```
sweep_seg_led_ioctl(struct inode * inode, struct file * file,
unsigned int cmd,unsigned long int arg[])
{
   int retval=0;
   switch (cmd) {
        /*Trong trường hợp lệnh delay trong kernel, gọi hàm
        sweep_led_delay*/
        case SWEEP_LED_DELAY:
             sweep_led_delay(arg[0]);
        break;
        /*Trong trường hợp lệnh xuất số hiển thị, gọi hàm
        sweep_led_number_display() để giải mã và cập nhật thông tin cho
        bộ đệm hiển thị LEDs*/
        case SWEEP_LED_NUMBER_DISPLAY:
             sweep_led_number_display(arg[0]);
        break;
        /*Trong trường hợp hiển thị thời gian, gọi hàm
        sweep_led_time_display()  để giải mã và xuất ra bộ nhớ đệm hiển
        thị dạng HH-MM-SS*/
        case SWEEP_LED_TIME_DISPLAY:
             sweep_led_time_display(arg[0], arg[1], arg[2]);
        break;
        /*Trong trường hợp không có lệnh hỗ trợ thì in ra  lỗi cho người lập
        trình*/
        default:
             printk("The function you type does not exist\n");
             retval=-1;
        break;
   }
}
```

*/*Khai báo và định nghĩa giao diện hàm open() */*

```
static int
sweep_seg_led_open(struct inode *inode, struct file *file)
{
   int result = 0;
   unsigned int dev_minor = MINOR(inode->i_rdev);
   if (!atomic_dec_and_test(&sweep_seg_led_open_cnt)) {
      atomic_inc(&sweep_seg_led_open_cnt);
      printk(KERN_ERR DRVNAME ": Device with minor ID %d
      already in use\n", dev_minor);
      result = -EBUSY;
      goto out;
   }
out:
   return result;
}
```

*/*Khai báo và định nghĩa giao diện hàm close*/*

```
static int
sweep_seg_led_close(struct inode * inode, struct file * file)
{
   smp_mb__before_atomic_inc();
   atomic_inc(&sweep_seg_led_open_cnt);

   return 0;
}
```

*/*Định nghĩa và cập nhật cấu trúc file operations */*

```
struct file_operations sweep_seg_led_fops = {
   .write     = dis_seg_led_write,
   .ioctl    = sweep_seg_led_ioctl,
   .open     = sweep_seg_led_open,
   .release = sweep_seg_led_close,
};
```

```
/*Định nghĩa và cập nhật cấu trúc i_node */
static struct miscdevice sweep_seg_led_dev = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name           = "sweep_seg_led",
        .fops           = &sweep_seg_led_fops,
};
/*Hàm thực thi khi driver được cài đặt vào hệ thống*/
static int __init
sweep_seg_led_mod_init(void)
{
   int ret=0;
   /*Khai báo các chân trong gpio đã định nghĩa thành chế độ ngõ ra */
   gpio_request (P00, NULL);
   gpio_request (P01, NULL);
   gpio_request (P02, NULL);
   gpio_request (P03, NULL);
   gpio_request (P04, NULL);
   gpio_request (P05, NULL);
   gpio_request (P06, NULL);
   gpio_request (P07, NULL);

   at91_set_GPIO_periph (P00, 1);
   at91_set_GPIO_periph (P01, 1);
   at91_set_GPIO_periph (P02, 1);
   at91_set_GPIO_periph (P03, 1);
   at91_set_GPIO_periph (P04, 1);
   at91_set_GPIO_periph (P05, 1);
   at91_set_GPIO_periph (P06, 1);
   at91_set_GPIO_periph (P07, 1);


   gpio_direction_output(P00, 0);
   gpio_direction_output(P01, 0);
   gpio_direction_output(P02, 0);
```

```
gpio_direction_output(P03, 0);
gpio_direction_output(P04, 0);
gpio_direction_output(P05, 0);
gpio_direction_output(P06, 0);
gpio_direction_output(P07, 0);

gpio_request (A, NULL);
gpio_request (B, NULL);
gpio_request (C, NULL);
gpio_request (D, NULL);
gpio_request (E, NULL);
gpio_request (F, NULL);
gpio_request (G, NULL);

at91_set_GPIO_periph (A, 1);
at91_set_GPIO_periph (B, 1);
at91_set_GPIO_periph (C, 1);
at91_set_GPIO_periph (D, 1);
at91_set_GPIO_periph (E, 1);
at91_set_GPIO_periph (F, 1);
at91_set_GPIO_periph (G, 1);

 gpio_direction_output(A, 0);
gpio_direction_output(B, 0);
gpio_direction_output(C, 0);
gpio_direction_output(D, 0);
gpio_direction_output(E, 0);
gpio_direction_output(F, 0);
gpio_direction_output(G, 0);
```

*/*Khai báo timer0 cho hệ thống*/*

```
at91tc0_clk = clk_get(NULL,"tc0_clk");
```

*/*Cho phép clock hoạt động*/*

```
clk_enable(at91tc0_clk);
```

*/*Di chuyển con trỏ timer0 đến địa chỉ thanh ghi nền cho timer counter 0*/*

```
at91tc0_base = ioremap_nocache(AT91SAM9260_BASE_TC0, 64);
```

*/*Kiểm tra lỗi trong quá trình định vị*/*

```
if (at91tc0_base == NULL)
 {
     printk(KERN_INFO "at91adc: TC0 memory mapping failed\n");
     ret = -EACCES;
     goto exit_5;
 }
```

*/*Cập nhật thông số cho timer, khởi tạo định thời ngắt timer0 với chu kỳ ngắt là 1ms*/*

```
// Configure TC0 in waveform mode, TIMER_CLK1 and to generate interrupt on RC compare.
// Load 50000 to RC so that with TIMER_CLK1 = MCK/2 = 50MHz, the interrupt will be
// generated every 1/50MHz * 50000 = 20nS * 50000 = 1 milli second.
// NOTE: Even though AT91_TC_RC is a 32-bit register, only 16-bits are programmble.

iowrite32(50000, (at91tc0_base + AT91_TC_RC));
iowrite32((AT91_TC_WAVE | AT91_TC_WAVESEL_UP_AUTO), (at91tc0_base + AT91_TC_CMR));
iowrite32(AT91_TC_CPCS, (at91tc0_base + AT91_TC_IER));
iowrite32((AT91_TC_SWTRG | AT91_TC_CLKEN), (at91tc0_base + AT91_TC_CCR));
```

//Khởi tạo ngắt cho timer0

```
ret = request_irq(AT91SAM9260_ID_TC0, //ID ngắt timer0
at91tc0_isr, //Trỏ đến hàm phục vụ ngắt
0, // Định nghĩa cờ ngắt có thể chia sẽ
```

```
    "sweep_seg_led_irq",/*Tên ngắt hiển thị trong /proc/interrupts*/
    NULL); //Dữ liệu riêng trong quá trình chia sẽ ngắt
    /*In ra thông báo lỗi nếu khởi tạo ngắt không thành công*/
    if (ret != 0)
    {
    printk(KERN_INFO   "sweep_seg_led_irq:   Timer   interrupt
    request failed\n");
    ret = -EBUSY;
    goto exit_6;
    }
    /*Đăng ký thiết bị vào hệ thống*/
    ret = misc_register(&sweep_seg_led_dev);
    /*In thông báo cho người dùng khi thiết bị cài đặt thành  công*/
    printk(KERN_INFO "sweep_seg_led: Loaded module\n");
    return ret;
    /*Giải phóng bộ nhớ khi có lỗi xảy ra*/
    exit_6:
    iounmap(at91tc0_base);
    exit_5:
    clk_disable(at91tc0_clk);
    return ret;
}
/*Hàm được thực thi khi tháo gỡ driver ra khỏi hệ thống */
static void __exit
sweep_seg_led_mod_exit(void)
{
    /*Giải phóng vùng nhớ dành cho timer0*/
    iounmap(at91tc0_base);
    /*Giải phóng clock dành cho timer0*/
    clk_disable(at91tc0_clk);
    /*Giải phóng ngắt dành cho timer0*/
    free_irq(  AT91SAM9260_ID_TC0, // Interrupt number
```

```
                    NULL); // Private data for shared interrupts
    /*Tháo driver ra hệ thống*/
    misc_deregister(&sweep_seg_led_dev);
    /*In thông báo cho người dùng driver đã được tháo gỡ */
    printk(KERN_INFO "sweep_seg_led: Unloaded module\n");
}
/*Cài đặt các hàm vào các macro init và exit*/
module_init (sweep_seg_led_mod_init);
module_exit (sweep_seg_led_mod_exit);
/*Các thông tin khái quát cho driver */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("coolwarmboy");
MODULE_DESCRIPTION("Character device for for generic gpio
api");
```

3. *Chương trình application:*

```
/*Khai báo thư viện cần thiết cho các lệnh dùng trong chương trình*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
/*Khai báo số định danh lệnh dùng cho hàm ioctl*/
#define SWEEP_LED_DEV_MAGIC  'B'
#define SWEEP_LED_DELAY _IO(SWEEP_LED_DEV_MAGIC, 0)
#define SWEEP_LED_NUMBER_DISPLAY _IO(SWEEP_LED_DEV_MAGIC, 1)
#define SWEEP_LED_TIME_DISPLAY _IO(SWEEP_LED_DEV_MAGIC, 2)
/*Biến lưu số mô tả tập tin thiết bị khi được mở*/
int sweep_seg_led_fd;
/*Bộ nhớ đệm dữ liệu cho hàm ioctl() chép qua user*/
unsigned long int ioctl_buf[3]={0,0,0};
/*Bộ nhớ đệm cho hàm write chứa dữ liệu cần hiển thị ra LEDs*/
unsigned char write_buf[8]={0,8,0,1,0,1,7,0};
```

```
/*Các biến phục vụ cho chức năng đếm của chương trình*/
unsigned long XX, YY, T, counter;
/*Các biến phục vụ cho chức năng đếm thời gian của chương trình*/
unsigned char HH, MM, SS;
/*Hàm in hướng dẫn cho người dùng trong trường hợp phát sinh lỗi cú pháp*/
int
print_usage (void) {
   printf   ("display_number|count_number|count_time     (XX|HH
   YY|MM T|SS)\n");
   return -1;
}
/*Hàm delay trong user, gọi hàm delay trong driver*/
void user_delay(unsigned long int delay) {
   ioctl_buf[0] = delay;
   ioctl (sweep_seg_led_fd, SWEEP_LED_DELAY, ioctl_buf);
}
/*Hàm truyền các thông số giờ phút giây sang bộ nhớ đệm*/
void user_transmit_time(unsigned long int HH, unsigned long
int MM, unsigned long int SS) {
   ioctl_buf[0] = HH;
   ioctl_buf[1] = MM;
   ioctl_buf[2] = SS;
   ioctl(sweep_seg_led_fd,SWEEP_LED_TIME_DISPLAY,ioctl_buf);
}
/*Hàm main() được khai báo dạng tham số nhập từ người dùng để lựa chọn
chức năng thực thi, và các thông số cần thiết cho từng chức năng */
int
main(int argc, char **argv)
{
   int res;
   /*Trước khi thao tác thì phải mở tập tin thiết bị, đồng thời kiểm tra
   lỗi trong quá trình mở*/
```

```
if ((sweep_seg_led_fd = open("/dev/sweep_seg_led", O_RDWR))
< 0)
{
printf("Error whilst opening /dev/sweep_seg_led device\n");
     return -1;
}
/*Phân biệt các trường hợp lệnh khác nhau*/
switch (argc) {
     case 2:
     /*Thực hiện chức năng hiển thị mã số sinh viên*/
     if (!strcmp(argv[1], "display_number")) {
     /*Gọi giao diện hàm write() ghi vùng nhớ đệm từ user sang driver*/
          write (sweep_seg_led_fd, write_buf, 8);
     } else {
          return print_usage();
          }
     break;
     case 5:
     /*Trong trương hợp đếm số hiển thị LEDs*/
     if (!strcmp(argv[1], "count_number")) {
     /*Cập nhật các tham số từ người dùng*/
          T = atoi(argv[4]);
          XX = atoi(argv[2]);
          YY = atoi(argv[3]);
          counter = XX;
     /*Thực hiện đếm theo quy định của người dùng*/
          while (counter != YY) {
               ioctl_buf[0] = counter;
               ioct(sweep_seg_led_fd,
               SWEEP_LED_NUMBER_DISPLAY, ioctl_buf);
               user_delay(T);
               if (counter < YY) {
                    counter++;
```

```
                    } else {
                        counter--;
                    }
                }
                ioctl_buf[0] = counter;
                ioctl(sweep_seg_led_fd, SWEEP_LED_NUMBER_DISPLAY,
                ioctl_buf);
```

*/*Chức năng đếm thời gian hiển thị LEDs*/*

```
        } else if (!strcmp(argv[1], "count_time")) {
```

*/*Cập nhật giờ phút giây từ người dùng*/*

```
                HH = atoi(argv[2]);
                MM = atoi(argv[3]);
                SS = atoi(argv[4]);
```

*/*Chuyển sang vùng nhớ đệm của ioctl()*/*

```
                user_transmit_time(HH,MM,SS);
                user_delay(100);
```

*/*Thực hiện đếm thời gian giờ phút giây*/*

```
                while (1) {
                    if (SS++ == 59) {
                        SS = 0;
                        if (MM++ == 59) {
                        MM = 0;
                        if (HH++ == 23) HH = 0;
                        }
                    }
                    user_transmit_time(HH,MM,SS);
                    user_delay(100);
                }
            } else {
                return print_usage();
            }
            break;
    default:
            return print_usage();
```

```
            break;
        }
    return 0;
    }
```

4. *Biên dịch và thực thi chương trình:*

Biên dịch *driver* bằng tập tin Makefile có nội dung sau:

```
export ARCH=arm
export CROSS_COMPILE=arm-none-linux-gnueabi-
obj-m += 3_Sweep_Seg_led_dev.o
all:
     make   -C   /home/arm/project/kernel/linux-2.6.30   M=$(PWD)
modules
clean:
     make   -C   /home/arm/project/kernel/linux-2.6.30   M=$(PWD)
clean
```

Biên dịch chương trình ứng dụng bằng dòng lệnh sau:

```
arm-none-linux-gnueabi-gcc        3_Sweep_Seg_led_app.c        -o
Sweep_Seg_led_app
```

Chép tập tin *driver* và *application* sau khi biên dịch vào kit;

Cài đặt *driver* vào hệ thống bằng lệnh:

```
insmod 3_Sweep_Seg_led_dev.ko
```

Chạy chương trình ứng dụng:

- Chức năng hiển thị số:

```
./Sweep_Seg_led_app   display_number
```

- Chức năng đếm sồ hiển thị:

```
./Sweep_Seg_led_app   count_number    1     10    100
```

- Chức năng đếm thời gian:

```
./Sweep_Seg_led_app   count_time 12    12    12
```

Người học quan sát và kiểm tra kết quả thực thi dự án.

III. Kết luận và bài tập:

a. Kết luận:

Trong bài này chúng ta đã điều khiển thành công 8 LEDs 7 đoạn dùng phương pháp quét. Kiến thức quan trọng nhất trong bài, bên cạnh phương pháp quét led, là cách khởi tạo ngắt từ timer vật lý tích hợp trong vi điều khiển. Người học cần phải hiểu rõ các bước khởi tạo ngắt, cách cài đặt các thông số cho timer để đạt được các khoảng thời gian ngắt cần thiết

b. Bài tập:

1. Cải tiến *driver* điều khiển quét LEDs 7 đoạn 3_Sweep_Seg_led_dev.c trên sao cho có thể xóa được số 0 vô nghĩa khi hiển thị số.
2. Cải tiến *driver* điều khiển quét LEDs 7 đoạn 3_Sweep_Seg_led_dev.c trên sao cho có thể hiển thị được số âm trên 8 LEDs 7 đoạn.
3. Viết chương trình ứng dụng *user application* thực hiện các phép toán "+" "-" "x" và ":" kết quả hiển thị trên LEDs 7 đoạn. Các toán hạng được nhập từ người dùng.

BÀI 4

GIAO TIẾP ĐIỀU KHIỂN LCD 16x2

I. Phác thảo dự án:

Bằng phương pháp truy xuất các chân gpio theo một quy luật nào đó chúng ta có thể điều khiển nhiều thiết bị khác nhau. Trong những bài trước, chúng ta đã điều khiển LEDs đơn, LEDs 7 đoạn, trong bài này chúng ta sẽ thực hành điều khiển một thiết bị hiển thị khác là LCD thuộc loại 16x2. Lý thuyết về nguyên lý hoạt động của LCD đã được nghiên cứu trong các tài liệu chuyên ngành khác, hoặc các bạn có thể tham khảo trong datasheet của thiết bị. Ở đây chúng ta không nhắc lại mà chỉ tập vào viết dirver và chương trình điều khiển trong hệ thống nhúng. Các lệnh thao tác sẽ được giải thích kỹ trong quá trình lập trình.

a. Yêu cầu dự án:

Dự án điều khiển LCD bao gồm các chức năng sau:

- Hiển thị thông tin nhập vào từ người dùng: Người dùng nhập vào một chuỗi ký tự trong màn hình console (*terminal display*), chuỗi ký tự này được xuất hiện trong LCD.
- Các thông số về ngày thàng năm của hệ thống được cập nhật và hiển thị trong LCD.
- Đếm hiển thị trên LCD: Người dùng sẽ nhập các thông số về chu kỳ, giới hạn 1, giới hạn 2. Thực hiện đếm tương tự như thuật toán của bài quét LEDs 7 đoạn. Các thông số này đều được hiển thị trên LCDs.

Sau đây chúng ta sẽ tiến hành phân công nhiệm vụ thực hiện của *driver* và *application*.

b. Phân công nhiệm vụ:

- *Driver*:

Sử dụng giao diện `ioctl()` để thực hiện các thao tác điều khiển LCD cơ bản như:

- Nhận mã lệnh điều khiển từ *user application* sau đó xuất ra các chân `gpio` ghi vào thanh ghi lệnh của LCDs;
- Nhận dữ liệu ký tự từ *user application*, xuất ra các chân `gpio` ghi vào thanh ghi dữ liệu của LCDs;
- Và một số lệnh khác như: Trì hoãn thời gian dùng `jiffies`, di chuyển con trỏ về đầu dòng hiển thị, bật tắt hiển thị, ...

- *Application*:

Chương trình trong *application* sử dụng những chức năng của *driver* hỗ trợ, lập trình thành những hàm có khả năng lớn hơn để thực hiện những yêu cầu của dự án.

Chương trình dùng phương pháp lập trình có tham số lựa chọn từ người dùng để thực hiện nhiệm vụ theo yêu cầu. Những nhiệm vụ đó là:

- Hiển thị thông tin từ người dùng:

  Người dùng chạy chương trình theo cú pháp sau:

  `<tên chương trình> display_string <Chuỗi_ký_tự_cần_hiển_thị>`

  Trong đó:

  `<Tên chương trình>` là tên chương trình ứng dụng sau khi biên dịch;

  `< Chuỗi_ký_tự_cần_hiển_thị>` là chuỗi thông tin người dùng muốn hiển thị ra LCD;

  `display_string` là tham số chức năng phải nhập để thực hiện nhiệm vụ;

  Sau khi nhận được yêu cầu, chương trình sẽ tách từng ký tự trong chuỗi vừa nhập lần lượt hiển thị trên LCDs.

- Hiển thị các thông số ngày thàng năm của hệ thống ra LCD:

  Người dùng nhập cú pháp thực thi chương trình như sau:

  `<tên chương trình> display_time`

  Trong đó:

  `display_time` là tham số chức năng phải nhập để thực hiện nhiệm vụ;

Ngày thàng năm được hiển thị trên LCD có dạng:

```
The present time is:
DD/MM/YYYY
```

- Đếm hiển thị trên LCD:

Người dùng nhập lệnh thực thi chương trình theo cú pháp:

```
<tên chương trình> display_counter XX YY T
```

Trong đó:

`display_counter` là tham số chức năng cần phải nhập để thực hiện nhiệm vụ;

`XX` là giới hạn 1;

`YY` là giới hạn 2;

*(**Giới hạn 1 và giới hạn 2 nằm trong khoảng từ 00 đến 99);*

`T` là chu kỳ thay đổi trạng thái đếm, đơn vị là ms;

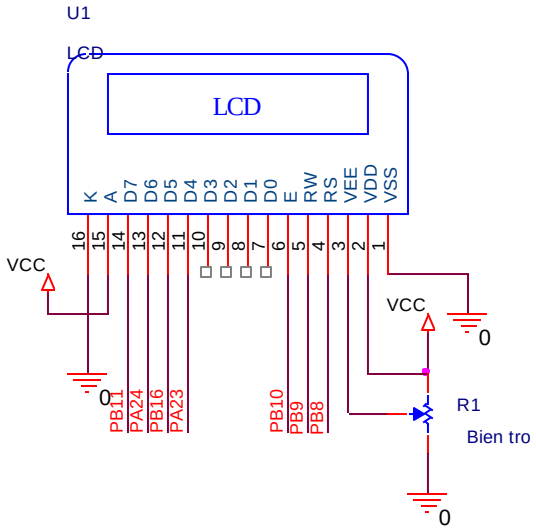
Chương trình sẽ thực hiện đếm từ `XX` đến `YY` với chu kỳ thay đổi là `T`(ms). Hiển thị trên LCD theo dạng:

Hàng 1: `XX YY`

Hàng 2: `<counter display>`

II. Thực hiện:

1. *Kết nối phần cứng theo sơ đồ sau:*



Hình 4-16- Sơ đồ kết nối LCD.

2. *Chương trình driver:*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <asm/gpio.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
#include <asm/atomic.h>
#include <linux/jiffies.h>
#include <linux/sched.h>
#define DRVNAME      "LCDDriver"
#define DEVNAME      "LCDDevice"

     /*-------------LCD COMMAND DEFINED---------------*/
#define IOC_LCDDEVICE_MAGIC      'B'
#define LCD_CONTROL_WRITE        _IO(IOC_LCDDEVICE_MAGIC, 15)
```

```
#define LCD_DATA_WRITE          _IO(IOC_LCDDEVICE_MAGIC, 16)
#define LCD_INIT                _IO(IOC_LCDDEVICE_MAGIC, 17)
#define LCD_HOME                _IO(IOC_LCDDEVICE_MAGIC, 18)
#define LCD_CLEAR               _IO(IOC_LCDDEVICE_MAGIC, 19)
#define LCD_DISP_ON_C           _IO(IOC_LCDDEVICE_MAGIC, 21)
#define LCD_DISP_OFF_C          _IO(IOC_LCDDEVICE_MAGIC, 22)
#define LCD_CUR_MOV_LEFT_C      _IO(IOC_LCDDEVICE_MAGIC, 23)
#define LCD_CUR_MOV_RIGHT_C     _IO(IOC_LCDDEVICE_MAGIC, 24)
#define LCD_DIS_MOV_LEFT_C      _IO(IOC_LCDDEVICE_MAGIC, 25)
#define LCD_DIS_MOV_RIGHT_C     _IO(IOC_LCDDEVICE_MAGIC, 29)
#define LCD_DELAY_MSEC          _IO(IOC_LCDDEVICE_MAGIC, 30)
#define LCD_DELAY_USEC          _IO(IOC_LCDDEVICE_MAGIC, 31)


/*----------LCD PIN CONTROL + DATA----------*/
#define LCD_RW_PIN AT91_PIN_PB9
#define LCD_EN_PIN AT91_PIN_PB10
#define LCD_RS_PIN AT91_PIN_PB8
#define LCD_D4_PIN AT91_PIN_PA23
#define LCD_D5_PIN AT91_PIN_PB16
#define LCD_D6_PIN AT91_PIN_PA24
#define LCD_D7_PIN AT91_PIN_PB11
/*----------LCD command datas----------*/
/* LCD memory map */
/*Vị trí con trỏ bắt đầu dòng đầu tiên*/
#define LCD_LINE0_ADDR 0x00
/*Vị trí con trỏ bắt đầu dòng thứ hai*/
#define LCD_LINE1_ADDR 0x40
/*Vị trí con trỏ bắt đầu dòng thứ ba*/
#define LCD_LINE2_ADDR 0x14
/*Vị trí con trỏ bắt đầu dòng thứ tư*/
#define LCD_LINE3_ADDR 0x54
/* Mã lệnh cơ bản của LCD */
```

```
/*Con trỏ địa chỉ RAM hiển thị*/
#define LCD_DD_RAM_PTR 0x80
/*Con trỏ địa chỉ RAM tạo ký tự*/
#define LCD_CG_RAM_PTR 0x40
/*Lệnh xóa dữ liệu hiển thị, con trỏ hiển thị về 0*/
#define LCD_CLEAR_DISPLAY 0x01
/*Lệnh đưa con trỏ về điểm bắt đầu dòng đầu tiên, dữ liệu bị dịch  sẽ trở về vị trí cũ*/
#define LCD_RETURN_HOME 0x02
/*Lệnh cài dặt chế độ giao tiếp 4 bits và 2 dòng hiển thị*/
#define LCD_DISP_INIT 0x28
/*Con trỏ hiển thị tăng sau khi ghi dữ liệu vào RAM hiển thị*/
#define LCD_INC_MODE 0x06
/*Bật hiển thị và con trỏ nhập nháy*/
#define LCD_DISP_ON 0x0C
/*Tắt hiển thị và con trỏ*/
#define LCD_DISP_OFF 0x08
/*Lệnh bật con trỏ */
#define LCD_CURSOR_ON 0x04
/*Lệnh tắt con trỏ*/
#define LCD_CURSOR_OFF 0x00
/*Di chuyển con trỏ và dữ liệu hiển thị sang bên trái*/
#define LCD_CUR_MOV_LEFT 0x10
/*Di chuyển con trỏ và dữ liệu sang bên phải*/
#define LCD_CUR_MOV_RIGHT 0x14
/*Dữ liệu hiển thị được dịch sang trái*/
#define LCD_DIS_MOV_LEFT 0x18
/*Dữ liệu hiển thị được dịch sang phải*/
#define LCD_DIS_MOV_RIGHT 0x1C
/*Trạng thái LDC đang bận, dùng để kiểm tra trạng thái của LCD*/
#define LCD_BUSY 0x80
```

```
                    /*Các lệnh Set và Clear bit căn bản*/
#define SET_LCD_RS_Line() gpio_set_value(LCD_RS_PIN,1)
#define SET_LCD_BL_Line() gpio_set_value(LCD_BL_PIN,1)
#define SET_LCD_EN_Line() gpio_set_value(LCD_EN_PIN,1)

#define CLR_LCD_RS_Line() gpio_set_value(LCD_RS_PIN,0)
#define CLR_LCD_BL_Line() gpio_set_value(LCD_BL_PIN,0)
#define CLR_LCD_EN_Line() gpio_set_value(LCD_EN_PIN,0)

#define SET_LCD_D4_Line() gpio_set_value(LCD_D4_PIN,1)
#define SET_LCD_D5_Line() gpio_set_value(LCD_D5_PIN,1)
#define SET_LCD_D6_Line() gpio_set_value(LCD_D6_PIN,1)
#define SET_LCD_D7_Line() gpio_set_value(LCD_D7_PIN,1)

#define CLR_LCD_D4_Line() gpio_set_value(LCD_D4_PIN,0)
#define CLR_LCD_D5_Line() gpio_set_value(LCD_D5_PIN,0)
#define CLR_LCD_D6_Line() gpio_set_value(LCD_D6_PIN,0)
#define CLR_LCD_D7_Line() gpio_set_value(LCD_D7_PIN,0)
/*-----------------------------------------------------*/
                    /*Những lệnh trong driver hỗ trợ
void lcd_Control_Write(uint8_t data);
void lcd_Data_Write(uint8_t data);
void lcd_Home(void);
void lcd_Clear(void);
void lcd_Goto_XY(uint8_t xy);*/
/*Hàm trì hoãn thời gian chuyển đơn vị us sang jiffies*/
void lcd_delay_usec(unsigned int u) {
      long int time_delay;
      time_delay = jiffies + usecs_to_jiffies(u);
      while (time_before(jiffies, time_delay)) {
            schedule();
            }
}
```

```
/*Hàm trì hoãn thời gian chuyển đơn vị ms sang jiffies*/
void lcd_Delay_mSec (unsigned long m) {
      long int time_delay;
      time_delay = jiffies + msecs_to_jiffies(m);
      while (time_before(jiffies, time_delay)) {
            schedule();
            }
}
/*Hàm chuyển dữ liệu 4 bit thành các trạng thái trong chân gpio*/
void lcd_write_data_port(uint8_t data)
{
      (data&(1<<0))? SET_LCD_D4_Line():CLR_LCD_D4_Line();
      (data&(1<<1))? SET_LCD_D5_Line():CLR_LCD_D5_Line();
      (data&(1<<2))? SET_LCD_D6_Line():CLR_LCD_D6_Line();
      (data&(1<<3))? SET_LCD_D7_Line():CLR_LCD_D7_Line();
}
/*Hàm ghi dữ liệu 8 bits vào LCD, ghi 2 lần 4 bits vào LCDs để được 8 bits vì
đang ở chế độ giao tiếp 4 bits.*/
void lcd_data_line_write(uint8_t data)
{
      /*Đầu tiên ghi 4 bits cao vào LCDs, sau đó đến 4 bit thấp*/
      /*Chuẩn bị tạo xung cạnh xuống cho EN, cho EN lên mức cao*/
      SET_LCD_EN_Line();
      /*Ghi 4 bit thấp vào các chân dữ liệu*/
      lcd_write_data_port((data>>4)&0x000F);
      /*Tạo xung cạnh xuống ghi 4 bit thấp vào thanh ghi*/
      CLR_LCD_EN_Line();
      /*Trì hoãn một khoảng thời gian chờ thực thi lệnh*/
      lcd_delay_usec(50);
      /*Tiếp theo ghi 4 bits thấp của dữ liệu vào thanh ghi*/
      SET_LCD_EN_Line();
      lcd_write_data_port(data&0x000F);
      CLR_LCD_EN_Line();
```

```
        lcd_delay_usec(50);
}
/*Hàm ghi mã lệnh vào thanh ghi lệnh trong LCD*/
void lcd_Control_Write(uint8_t data)
{
        /*Trì hoãn thời gian chờ thực thi những lệnh trước đó*/
        lcd_delay_usec(50);
        /*Chọn thanh ghi lệnh*/
        CLR_LCD_RS_Line();
        /*Ghi mã lệnh vào thanh ghi lệnh*/
        lcd_data_line_write(data);
}
/*Hàm ghi dữ liệu vào thanh ghi dữ liệu trong LCDs*/
void lcd_Data_Write(uint8_t data)
{
        /*Trì hoãn thời gian chờ thực thi những lệnh trước đó*/
        lcd_delay_usec(50);
        /*Chọn thanh ghi dữ liệu trong LCDs*/
        SET_LCD_RS_Line();
        /*Ghi dữ liệu vào thanh ghi*/
        lcd_data_line_write(data);
}

/*----------Init the LCD----------*/
/*Hàm khởi tạo các thông số cho LCD làm việc theo yêu cầu của người lập trình*/
void lcd_Init(void)
{
        /*Cài đặt LCD hoạt động theo chế độ giao tiếp 4 bits*/
        lcd_Control_Write(0x33);
        /*Trì hoãn thời gian thực thi lệnh*/
        lcd_delay_usec(1000);
```

```
        lcd_Control_Write(0x32);
        lcd_delay_usec(1000);
        /*Cài đặt trạng thái ban đầu của LCDs*/
        lcd_Control_Write(LCD_DISP_INIT);
        /*Gọi lệnh xóa dữ liệu hiển thị trong LCDs*/
        lcd_Control_Write(LCD_CLEAR_DISPLAY);
        /*Trì hoãn thời gian 60ms*/
        lcd_delay_usec(60000);
        /*Cài đặt chế độ con trỏ tăng dần sau khi ghi dữ liệu*/
        lcd_Control_Write(LCD_INC_MODE);
        /*Bật hiển thị cho LCD*/
        lcd_Control_Write(LCD_DISP_ON);
        /*Chuyển con trỏ hiển thị đến vị trí bắt đầu dòng đầu tiên*/
        lcd_Control_Write(LCD_RETURN_HOME);
}
/*Hàm chức năng chuyển con trỏ về vị trí bắt đầu dòng đầu tiên*/
void lcd_Home(void)
{
        lcd_Control_Write(LCD_RETURN_HOME);
}
/*Hàm chức năng xóa dữ liệu hiển thị*/
void lcd_Clear(void)
{
        lcd_Control_Write(LCD_CLEAR_DISPLAY);
}
/*Hàm chức năng bật hiển thị cho LCD*/
void lcd_Display_On(void)
{
        lcd_Control_Write(LCD_DISP_ON);
}
/*Hàm tắt hiển thị cho LCDs*/
void lcd_Display_Off(void)
```

```
{
      lcd_Control_Write(LCD_DISP_OFF);
}
```

*/*Hàm dịch chuyển dữ liệu hiển thị về bên trái*/*

```
void lcd_Dis_Mov_Left(void)
{
      lcd_Control_Write(LCD_DIS_MOV_LEFT);
}
```

*/*Hàm dịch chuyển dữ liệu hiển thị về bên phải*/*

```
void lcd_Dis_Mov_Right(void)
{
      lcd_Control_Write(LCD_DIS_MOV_RIGHT);
}
```

*/*Hàm di chuyển con trỏ về bên trái*/*

```
void lcd_Cursor_Move_Left(void)
{
      lcd_Control_Write(LCD_CUR_MOV_LEFT);
}
```

*/*Hàm dư chuyển con trỏ về bên phải*/*

```
void lcd_Cursor_Move_Right(void)
{
      lcd_Control_Write(LCD_CUR_MOV_RIGHT);
}

static atomic_t lcd_open_cnt = ATOMIC_INIT(1);
```

*/*Giao diện hàm ioctl() thực hiện chức năng do user application yêu cầu*/*

```
static int
lcd_ioctl(struct inode * inode, struct file * file, unsigned int
cmd, unsigned long arg)
{
      int retval = 0;
      switch (cmd)
      {
```

```
/*Chức năng nhận mã lệnh thực thi từ user application*/
case LCD_CONTROL_WRITE:
 lcd_Control_Write(arg);
 break;
/*Chức năng nhận dữ liệu hiển thị từ user application*/
case LCD_DATA_WRITE:
 lcd_Data_Write(arg);
 break;
/*Chức năng trì hoãn thời gian dùng jiffies, chuyển ms  thành jiffies*/
case LCD_DELAY_MSEC:
 lcd_Delay_mSec(arg);
 break;
/*Chức năng trì hoãn thời gian dùng jiffies,chuyển us thành  jiffies*/
case LCD_DELAY_USEC:
 lcd_delay_usec(arg);
 break;
/*Chức năng chuyển con trỏ về hàng đầu tiên của hàng 1*/
case LCD_HOME:
 lcd_Home();
 break;
/*Chức năng xóa hiển thị*/
case LCD_CLEAR:
 lcd_Clear();
 break;
/*Chức năng bậthiển thị*/
case LCD_DISP_ON_C:
 lcd_Display_On();
 break;
/*Chức năng xóa hiển thị*/
case LCD_DISP_OFF_C:
 lcd_Display_Off();
 break;
/*Di chuyển con trỏ và dữ liệu sang trái*/
```

```
        case LCD_DIS_MOV_LEFT_C:
         lcd_Dis_Mov_Left();
         break;
        /*Di chuyển con trỏ và dữ liệu sang phải*/
        case LCD_DIS_MOV_RIGHT_C:
         lcd_Dis_Mov_Right();
         break;
        /*Di chuyển con trỏ sang trái*/
        case LCD_CUR_MOV_LEFT_C:
         lcd_Cursor_Move_Left();
         break;
        /*Di chuyển con trỏ sang phải*/
        case LCD_CUR_MOV_RIGHT_C:
         lcd_Cursor_Move_Right();
         break;
        /*Chức năng khởi tạo LCD hoạt động theo chế độ 4 bits, 2  dòng hiển thị*/
        case LCD_INIT:
         lcd_Init();
         break;
        /*Trong trường hợp không có lệnh nào hỗ trợ*/
        default:
         retval = -EINVAL;
         break;
    }
    return retval;
}
/*Khai báo và định nghĩa giao diện hàm open()*/
static int
lcd_open(struct inode *inode, struct file *file)
{
    int result = 0;
    unsigned int dev_minor = MINOR(inode->i_rdev);
    if (!atomic_dec_and_test(&lcd_open_cnt)) {
```

```
        atomic_inc(&lcd_open_cnt);
        printk(KERN_ERR DRVNAME ": Device with minor ID %d already
        in use\n", dev_minor);
        result = -EBUSY;
        goto out;
    }
out:
    return result;
}
```

*/*Khai báo và định nghĩa giao diện hàm close()*/*

```
static int
lcd_close(struct inode * inode, struct file * file)
{
    smp_mb__before_atomic_inc();
    atomic_inc(&lcd_open_cnt);

    return 0;
}
```

*/*Khai báo và định nghĩa các lệnh mà LCD hỗ trợ file operations*/*

```
struct file_operations lcd_fops = {
    .ioctl      = lcd_ioctl,
    .open      = lcd_open,
    .release   = lcd_close,
};
```

*/*Khai báo và định nghĩa thiết bị*/*

```
static struct miscdevice lcd_dev = {
        .minor         = MISC_DYNAMIC_MINOR,
        .name          = "lcd_dev",
        .fops          = &lcd_fops,
};
```

```
/*Hàm được thực thi khi driver được cài đặt vào hệ thống*/
static int __init
lcd_mod_init(void)
{
      int i;
      /*Khởi tạo các chên gpio là ngõ ra*/
      gpio_request (LCD_RW_PIN, NULL);
      gpio_request (LCD_EN_PIN, NULL);
      gpio_request (LCD_RS_PIN, NULL);
      gpio_request (LCD_D4_PIN, NULL);
      gpio_request (LCD_D5_PIN, NULL);
      gpio_request (LCD_D6_PIN, NULL);
      gpio_request (LCD_D7_PIN, NULL);

      at91_set_GPIO_periph (LCD_RW_PIN, 1);
      at91_set_GPIO_periph (LCD_EN_PIN, 1);
      at91_set_GPIO_periph (LCD_RS_PIN, 1);
      at91_set_GPIO_periph (LCD_D4_PIN, 1);
      at91_set_GPIO_periph (LCD_D5_PIN, 1);
      at91_set_GPIO_periph (LCD_D6_PIN, 1);
      at91_set_GPIO_periph (LCD_D7_PIN, 1);

      gpio_direction_output(LCD_RW_PIN,0);
      gpio_direction_output(LCD_EN_PIN,0);
      gpio_direction_output(LCD_RS_PIN,0);
      gpio_direction_output(LCD_D4_PIN,0);
      gpio_direction_output(LCD_D5_PIN,0);
      gpio_direction_output(LCD_D6_PIN,0);
      gpio_direction_output(LCD_D7_PIN,0);
      /*Lần lượt in thông báo cho người dùng, chuẩn bị cài đặt  driver*/
      for (i=5;i>=0;i--) {
            lcd_delay_usec(1000000);
            printk("Please wait ... %ds\n",i);
      }
```

```
        /*Gọi hàm khởi tạo LCD*/
        lcd_Init();
        /*Gọi hàm xóa LCD*/
        lcd_Clear();
        /*In thông báo cho người dùng cài đặt thành công*/
        printk(KERN_ALERT "Welcome to our LCD world\n");
        /*Cài đặt cài đặt driver vào hệ thống*/
        return misc_register(&lcd_dev);
}
/*Hàm thực thi khi driver bị gỡ bỏ*/
static void __exit
lcd_mod_exit(void)
{
   printk(KERN_ALERT "Goodbye for all best\n");
   misc_deregister(&lcd_dev);
}
/*Gán hàm init và exit vào các macro*/
module_init (lcd_mod_init);
module_exit (lcd_mod_exit);
/*Những thông tin tổng quát về driver*/
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Coolwarmboy / OpenWrt");
MODULE_DESCRIPTION("Character device for for generic lcd
driver");
```

3. *Chương trình application:*

```
/*Khai báo thư viện cho các lệnh cần dùng trong chương trình*/
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <getopt.h>
#include <fcntl.h>
```

```
#include <time.h> //Thư viện hỗ trợ cho các hàm thao tác thời gian

#include <sys/types.h>
#include <sys/stat.h>
#include <linux/ioctl.h>
```

*/*Định nghĩa các số định danh lệnh cho giao diện hàm ioctl() mỗi số tương đương với một chức năng trong driver hỗ trợ*/*

```
     /*-------------LCD COMMAND DEFINED---------------*/
#define IOC_LCDDEVICE_MAGIC      'B'
#define LCD_CONTROL_WRITE        _IO(IOC_LCDDEVICE_MAGIC, 15)
#define LCD_DATA_WRITE           _IO(IOC_LCDDEVICE_MAGIC, 16)
#define LCD_INIT                 _IO(IOC_LCDDEVICE_MAGIC, 17)
#define LCD_HOME                 _IO(IOC_LCDDEVICE_MAGIC, 18)
#define LCD_CLEAR                _IO(IOC_LCDDEVICE_MAGIC, 19)
#define LCD_DISP_ON_C            _IO(IOC_LCDDEVICE_MAGIC, 21)
#define LCD_DISP_OFF_C           _IO(IOC_LCDDEVICE_MAGIC, 22)
#define LCD_CUR_MOV_LEFT_C       _IO(IOC_LCDDEVICE_MAGIC, 23)
#define LCD_CUR_MOV_RIGHT_C      _IO(IOC_LCDDEVICE_MAGIC, 24)
#define LCD_DIS_MOV_LEFT_C       _IO(IOC_LCDDEVICE_MAGIC, 25)
#define LCD_DIS_MOV_RIGHT_C      _IO(IOC_LCDDEVICE_MAGIC, 29)
#define LCD_DELAY_MSEC           _IO(IOC_LCDDEVICE_MAGIC, 30)
#define LCD_DELAY_USEC           _IO(IOC_LCDDEVICE_MAGIC, 31)
```

*/*Định nghĩa địa chỉ đầu tiên của mỗi dòng trong LCD, ở đây điều khiển LCD có hai dòng 16 ký tự*/*

```
#define LCD_LINE0_ADDR 0x00 //Start of line 0 in the DD-Ram
#define LCD_LINE1_ADDR 0x40 //Start of line 1 in the DD-Ram
```

*/*Xác lập bit quy định địa chỉ con trỏ dữ liệu*/*

```
#define LCD_DD_RAM_PTR 0x80 //Address Display Data RAM pointer
```

*/*Định nghĩa những lệnh cơ bản thường sử dụng, làm cho chương trình gọn dễ hiểu trong quá trình lập trình*/*

```
          /*--------------Functions of LCD-----------------*/
#define lcd_Control_Write()ioctl(fd_lcd,LCD_CONTROL_WRITE, data)
#define lcd_Data_Write()   ioctl(fd_lcd, LCD_DATA_WRITE, data)
#define lcd_Init()         ioctl(fd_lcd, LCD_INIT)
#define lcd_Home()         ioctl(fd_lcd, LCD_HOME)
#define lcd_Clear()        ioctl(fd_lcd, LCD_CLEAR)
#define lcd_Display_On()   ioctl(fd_lcd, LCD_DISP_ON_C)
#define lcd_Display_Off()  ioctl(fd_lcd, LCD_DISP_OFF_C)
#define lcd_Cursor_Move_Left()  ioctl(fd_lcd, LCD_CUR_MOV_LEFT_C)
#define lcd_Cursor_Move_Right() ioctl(fd_lcd, LCD_CUR_MOV_RIGHT_C)
#define lcd_Display_Move_Left() ioctl(fd_lcd, LCD_DIS_MOV_LEFT_C)
#define lcd_Display_Move_Right()ioctl(fd_lcd, LCD_DIS_MOV_RIGHT_C)
#define lcd_Delay_mSec() ioctl(fd_lcd, LCD_DELAY_MSEC, data)
#define lcd_Delay_uSec() ioctl(fd_lcd, LCD_DELAY_USEC, data)
```

*/*Phần khai báo các biến toàn cục*/*

```
int fd_lcd; //Biến lưu số mô tả tập tin khi driver LCD được mở
int counter_value; //Biến lưu giá trị đếm hiện tại cho chức năng đếm
```

*/*Delay for unsigned long data mSecond*/*

*/*Hàm dịch trái ký tự hiển thị X vị trí, do người lập trình nhập vào*/*

```
void Display_Shift_Left(int X) {
     int i;
     for (i=0; i<X; i++) {
          /*Gọi hàm dịch trái ký tự hiển thị X lần*/
          lcd_Display_Move_Left();
     }
}
```

*/*Hàm dịch phải ký tự hiển thị X vị trí, do người lập trình nhập vào*/*

```
void Display_Shift_Right(int X) {
     int i;
```

```
	for (i=0; i<X; i++) {
		/*Gọi hàm dịch phải X lần*/
		lcd_Display_Move_Right();
	}
}
```

*/*Di chuyển con trỏ hiển thị đến vị trí x, y. Trong đó x là số thứ tự dòng của LCD bắt đầu từ 0; y là số thứ tự cột của LCD bắt đầu từ 0*/*

```
void lcd_Goto_XY(uint8_t x, uint8_t y)
{
	/*Định nghĩa thanh ghi dữ liệu con trỏ của Data Display RAM*/
	register uint8_t DDRAMAddr;
	/*So sánh số x để chọn số dòng cho phù hợp*/
	switch(x)
	{
	/*Trong trường hợp dòng thứ nhất, dòng 0. Tiến hành cộng giá trị địa chỉ đầu dòng cho số cột y*/
	case 0: DDRAMAddr = LCD_LINE0_ADDR+y; break;
	/*Trong trường hợp dòng thứ hai, dòng 1. Tiến hành cộng giá trị địa chỉ đầu dòng cho số cột y*/
	case 1: DDRAMAddr = LCD_LINE1_ADDR+y; break;
	/*Trong trường hợp dòng 3, 4. Trường hợp này áp dụng cho LCD 16x4. Thuật toán cũng tương tự như 2 trường hợp trên*/
	//case 2: DDRAMAddr = LCD_LINE2_ADDR+y; break; for LCD 16x4 or //20x4 //only
	//case 3: DDRAMAddr = LCD_LINE3_ADDR+y; break;
	/*Trong những trường hợp khác vẫn tính là dòng thứ 1*/
	default: DDRAMAddr = LCD_LINE0_ADDR+y;
	}
	/*Cuối cùng gọi hàm giao diện ioctl() để đặt giá trị địa chỉ đã định nghĩa phía trên*/
	ioctl(fd_lcd, LCD_CONTROL_WRITE, LCD_DD_RAM_PTR | DDRAMAddr);
```

```
}
```
*/*Hàm xuất chuỗi ký tự tại vị trí bắt đầu là dòng x và cột y*/*
```
void Display_Print_Data(char *string, uint8_t x, uint8_t y)
{
```
*/*Đầu tiên di chuyển con trỏ đến vị trí mong muốn x và y*/*
```
      lcd_Goto_XY(x,y);
```
*/*Xuất lần lượt từng ký tự trong chuỗi string ra LCD hiển thi. Lặp lại cho đến khi kết thúc chuỗi */*
```
      while (*string)  {
            ioctl(fd_lcd, LCD_DATA_WRITE, *string++);
      }
}
```
*/*Hàm xuất một ký tự tại vị trí bắt đầu là dòng x và cột y*/*
```
void Display_Print_Char(int data, uint8_t x, uint8_t y)
{
```
*/*Di chuyển con trỏ đến vị trí cần in ký tự ra LCD*/*
```
      lcd_Goto_XY(x,y);
```
*/*Gọi hàm xuất ký tự ra LCD qua giao diện ioctl()*/*
```
      lcd_Data_Write ();
}
```
*/*Chương trình con khởi tạo hiển thị LCD*/*
```
void Display_Init_Display (void) {
```
*/*Biến lưu số lần chớp tắt hiển thị, dùng cho vòng lặp for()*/*
```
      int i;
```
*/*Gọi hàm xóa hiển thị của LCD*/*
```
      lcd_Clear();
```
/ Xuất chuỗi ký tự thông báo khởi tạo LCD thành công*/*
```
      Display_Print_Data ("LCD display",0,2);
      Display_Print_Data ("I am your slave!",1,0);
```
*/*Chớp tắt hiển thị 3 lần*/*
```
      for (i = 0; i < 3; i ++ ) {
```
*/*Gọi hàm tắt hiển thị*/*

```
            lcd_Display_Off();
/*Trì hoản thời gian trong 500ms*/
            ioctl(fd_lcd, LCD_DELAY_MSEC, 500);
/*Gọi hàm bật hiển thị*/
            lcd_Display_On();
/*Trì hoãn thời gian trong 500ms*/
            ioctl(fd_lcd, LCD_DELAY_MSEC, 500);
            }
/*Xóa hiển thị kết thúc lời chào*/
      lcd_Clear();
}
/*Chương trình con in thông báo lỗi ra LCD*/
void Display_print_error_0(void) {
      Display_Print_Data ("LCD information:",1,2);
      Display_Print_Data ("Error while typing!!!",0,0);
}
/*Hàm in ra hướng dẫn cho người dùng trong trường hợp người dùng nhập  sai cú pháp
lệnh*/
int print_usage(void) {
      printf("The command you required is not supported\n");
      printf("./4_lcd_app   display_string|display_time|display_counter
      <string>|<XX> <YY> <Period>\n");
      Display_print_error_0();
      return -1;
}
////////////////////////////////////////////////////////////////
/*Hàm xuất chuỗi ký tự hỗ trợ cho chế độ xuất ký tự của chương trình*/
void display_string(char *string) {
      Display_Print_Data("Your string is:",0,0);
      Display_Print_Data(string,1,0);
}
```

*/*Hàm cập nhật thời gian hiện tại và xuất ra LCD hiển thị. Hỗ trợ cho chế độ hiển thị thời gian của chương trình*/*

```
void display_time(void) {
```

*/*Biến lưu cấu trúc thời gian theo dạng năm thàng ngày giờ … */*

```
     struct tm *tm_ptr;
```

*/*Biến lưu thời gian hiện tại của hệ thống*/*

```
     time_t the_time;
```

*/*Hiển thị các thông tin ban đầu trên LCD*/*

```
     Display_Print_Data ("Date:   /  /  ",0,0);
     Display_Print_Data ("Time:   :  :  ",1,0);
```

*/*Vòng lặp cập nhật thời gian hiện tại và xuất dữ liệu ra LCD*/*

```
     while (1) {
```

*/*Lấy thời gian hiện tại của hệ thống*/*

```
          (void) time(&the_time);
```

*/*Chuyển đổi thời gian hiện tại của hệ thống sang cấu trúc thời gian dạng struc tm*/*

```
          tm_ptr = gmtime(&the_time);
```

*/*Lần lượt chuyển đổi các giá trị thời gian sang ký tự và xuất ra LCD. Thuật toán chuyển đổi:*

- *Chuyển sang số BCD từ số nguyên có 2 chữ số;*
- *Lần lượt cộng các số BCD cho 48 để chuyền sang các ký số trong bảng mã ascii*/*

*/*Chuyển đổi và hiển thị năm ra LCD*/*

```
          Display_Print_Char(((tm_ptr->tm_year)/10)+48,0,6);
          Display_Print_Char(((tm_ptr->tm_year)%10)+48,0,7);
```

*/*Chuyển đổi và hiển thị thàng ra LCD*/*

```
          Display_Print_Char(((tm_ptr->tm_mon+1)/10)+48,0,9);
          Display_Print_Char(((tm_ptr->tm_mon+1)%10)+48,0,10);
```

*/*Chuyển đổi và hiển thị ngày ra LCD*/*

```
          Display_Print_Char(((tm_ptr->tm_mday)/10)+48,0,12);
          Display_Print_Char(((tm_ptr->tm_mday)%10)+48,0,13);
```

*/*Chuyển đổi và hiển thị giờ ra LCD*/*

```
            Display_Print_Char(((tm_ptr->tm_hour)/10)+48,1,6);
            Display_Print_Char(((tm_ptr->tm_hour)%10)+48,1,7);
/*Chuyển đổi và hiển thị giờ ra LCD*/
            Display_Print_Char(((tm_ptr->tm_min)/10)+48,1,9);
            Display_Print_Char(((tm_ptr->tm_min)%10)+48,1,10);
/*Chuyển đổi và hiển thị giờ ra LCD*/
            Display_Print_Char(((tm_ptr->tm_sec)/10)+48,1,12);
            Display_Print_Char(((tm_ptr->tm_sec)%10)+48,1,13);
/*Trì hoãn cập nhật thời gian*/
            usleep(200000);
      }
}
////////////////////////////////////////////////////////////////
/*Các hàm hỗ trợ cho chức năng đếm số của chương trình*/
/*Hàm chuyển đổi số từ 00 đến 99 thành 2 chữ số hiển thị ra LCD tại vị  trí tương ứng
start_v_pos (cột bắt đầu) và h_pos(dòng bắt đầu)*/
void Display_Number(int number, int start_v_pos, int h_pos) {
      Display_Print_Char((number/10)+48,h_pos,start_v_pos);
      Display_Print_Char((number%10)+48,h_pos,start_v_pos+1);
}
/*Chương trình con đếm chính trong chức năng đếm số*/
void display_counter(int lmt1, int lmt2, int period_ms) {
/*Biến lưu chu kỳ thời gian us */
      long int period_us;
/*Biến lưu số lần chớp tắt hiển thị khi quá trình đếm thành công*/
      int i;
/*In thông tin cố định ban đầu lên LCD */
      Display_Print_Data ("Start:   End:   ",0,0);
      Display_Print_Data ("Count Value:",1,0);
/*Hiển thị số giới hạn Start*/
      Display_Number(lmt1,6,0);
/*Hiển thị số giới hạn End*/
```

```
	Display_Number(lmt2,13,0);
/*Cập nhật giá trị ban đầu cho counter_value*/
	counter_value = lmt1;
/*Chuyển thời gian ms sang us dùng cho hàm usleep()*/
	period_us = period_ms*1000;
/*Quá trình đếm bắt đầu, đếm từ giá trị Start cho đến giá trị End*/
	while (counter_value != lmt2) {
/*Hiển thị giá trị đếm hiện tại lên LCD*/
		Display_Number(counter_value,12,1);
/*Trì hoãn thời gian đếm*/
		usleep(period_us);
/*Tăng hoặc giảm giá trị đếm hiện tại cho đến khi bằng với giá trị  End*/
		if (counter_value < lmt2) {
			counter_value++;
		} else {
			counter_value--;
		}
	}
/*Khi quá trình đếm kết thúc in ra thông báo cho người dùng biết*/
	Display_Print_Data("Count commplete!",1,0);
/*Chớp tắt 3 lần trong với tầng số 1Hz*/
	for (i = 0; i < 3; i ++ ) {
		lcd_Display_Off();
		ioctl(fd_lcd, LCD_DELAY_MSEC, 500);
		lcd_Display_On();
		ioctl(fd_lcd, LCD_DELAY_MSEC, 500);
	}
}

/*Chương trình chính khai báo dưới dạng nhận tham số từ người dùng*/
int
main(int argc, char **argv)
{
```

*/*Mở tập tin thiết bị trước thao tác, lấy về số mô tả tập tin lưu trong biến fd_lcd. Kiểm tra lỗi trong quá trình thao tác*/*

```
if ((fd_lcd = open("/dev/lcd_dev", O_RDWR)) < 0)
     {
          printf("Error whilst opening /dev/lcd_dev\n");
          return -1;
     }
```

*/*Gọi các hàm khởi tạo LCD ban đầu*/*

```
printf("LCD initializing ...\n");
lcd_Init();
lcd_Clear();
Display_Init_Display();
```

*/*So sánh các tham số lựa chọn chức năng thực thi chương trình*/*

```
switch (argc) {
     case 2:
```

*/*Trong trường hợp hiển thị thời gian*/*

```
          if (!strcmp(argv[1],"display_time")) {
               display_time();
          } else {
               return print_usage();
          }
     break;
     case 3:
```

*/*Trong trường hợp hiển thị chuỗi ký tự*/*

```
          if (!strcmp(argv[1],"display_string")) {
               display_string(argv[2]);
          } else {
               return print_usage();
          }
     break;
     case 5:
```

*/*Trong trường hợp đếm số hiển thị trong khoảng từ 00 đến 99*/*

```
          if (!strcmp(argv[1],"display_counter")) {
```

```
                display_counter(atoi(argv[2]),atoi(argv[3]),
                atoi(argv[4]));
                } else {
                      return print_usage();
                }
          break;
          default:
```

*/*Trong trường hợp không có lệnh nào hỗ trợ in ra thông báo hướng dẫn cho người dùng */*

```
                return print_usage();
          break;
     }
return 0;
}
```

4. Biên dịch và thực thi chương trình:

Người học tự biên dịch chương trình *driver* và chương trình *application*. Sau khi biên dịch chép vào kít tiến hành kiểm tra kết quả trong từng trường hợp tương ứng với từng chức năng khác nhau của chương trình. Rút ra nhận xét và kinh nghiệm thực hiện.

III. Kết luận và bài tập:

a. Kết luận:

Trong bài này chúng ta đã nghiên cứu thêm một ứng dụng nữa trong việc điều khiển các chân gpio bằng các lệnh set và clear từng chân gpio cơ bản. Chúng ta đã thực hành sử dụng các lệnh thao tác với thời gian trong việc cập nhật và hiển thị trên thiết bị phần cứng LCD.

Chương trình *driver* và *application* trong dự án này mặc dù đã điều khiển thành công LCD nhưng chương trình vẫn còn chưa tối ưu như: Những thao tác điều khiển LCD vẫn còn nằm trong *user application* quá nhiều, khó áp dụng *driver* sang những dự án khác. Các bạn sẽ thực hiện tối ưu hóa hoạt động của *driver* LCD thông qua những bài tập yêu cầu trong phần sau.

b. Bài tập:

1. Thêm chức năng di chuyển con trỏ đến vị trí x và y bất kỳ trên LCD trong *driver* điều khiển LCD trong ví dụ trên. Trong đó: x là vị trí cột, y là vị trí dòng của LCD 16x2. Gợi ý: Chức năng tương tự như hàm void lcd_Goto_XY(uint8_t x, uint8_t y) trong *user application* nhưng chuyển qua *driver* thực hiện thông qua giao diện hàm ioctl().

2. Thêm chức năng hiển thị chuỗi ký tự vào *driver* điều khiển LCD theo yêu cầu sau:

- *Driver* nhận chuỗi ký tự thông qua giao diện hàm write() được gọi từ *user application*;

- Chương trình *driver* thực hiện ghi những ký tự này vào LCD hiển thị.

  ***Chức năng này của driver tương tự như hàm void Display_Print_Data(char *string, uint8_t x, uint8_t y) trong user application.*

3. Biên dịch *driver* và lưu vào thư viện để có thể sử dụng trong những dự án khác.

BÀI 5

GIAO TIẾP ĐIỀU KHIỂN GPIO NGÕ VÀO ĐẾM XUNG

I. Phác thảo dự án:

Trong các bài trước chúng ta chỉ sử dụng các chân của Vi Xử Lý ở chế độ gpio ngõ ra (output). Khi sử dụng các chân của Vi Xử Lý ở chế độ output thì chúng ta có thể điều khiển các thiết bị ngoại vi (led đơn, led 7 đoạn, ic…), nhưng trong thực tế có những trường hợp Vi Xử Lý phải cập nhật các tín hiệu từ bên ngoài (tín hiệu của nút nhấn, ma trận phím, …). Trong những trường hợp này chúng ta phải dùng chân của Vi Xử Lý ở chế độ gpio ngõ vào (input).

Ngoài ra, trong các dự án trước, với một ứng dụng chúng ta viết một chương trình Driver và một chương trình User Application. Với phương pháp lập trình này, chúng ta đã phần nào đó hiểu được điểm mạnh của việc lập trình ứng dụng bằng phương pháp nhúng trong việc chia nhiệm vụ cho Driver và User Application để điều khiển thiết bị. Nhưng trong dự án này chúng ta sẽ làm quen với phương pháp lập trình nhúng mới là sử dụng một User Application điều khiển nhiều Driver khác nhau (có nghĩa là với một ứng dụng chúng ta có thể viết nhiều Driver khác nhau và dùng một chương trình User Applicaton để điều khiển các Driver này). Có thể nói phương pháp này sẽ làm nổi bật lên điểm mạnh của việc lập trình ứng dụng trên hệ thống nhúng một cách rõ rệt hơn so với phương pháp lập trình thông thường cho Vi Xử Lý. Đối với các thiết bị, module khác nhau chúng ta có thể viết các Driver để điều khiển các module này (mỗi module một Driver) và có thể chạy các Driver này cùng lúc với nhau. Như vậy chúng ta có thể điều khiển nhiều thiết bị trong cùng một lúc một cách độc lập.

Chú ý: để tránh sự xung khắc giữa các Driver với nhau chúng ta không được sử dụng một chân của Vi Xử Lý cho nhiều Driver. Một chân của Vi Xử Lý chỉ được dùng cho một Driver duy nhất.

a. Yêu cầu dự án:

Trong dự án này chúng ta sẽ đếm xung từ bên ngoài (nút nhấn) và hiển thị số xung đếm được trên 8 led 7 đoạn theo phương pháp quét.

Cú pháp lệnh của chương trình như sau:

```
./< counter_display_7seg_app > <start>
```

Trong đó:

`< counter_display_7seg_app >` là tên chương trình User Application sau khi đã biên dịch.

`< start >` là lệnh bắt đầu chương trình đếm xung.

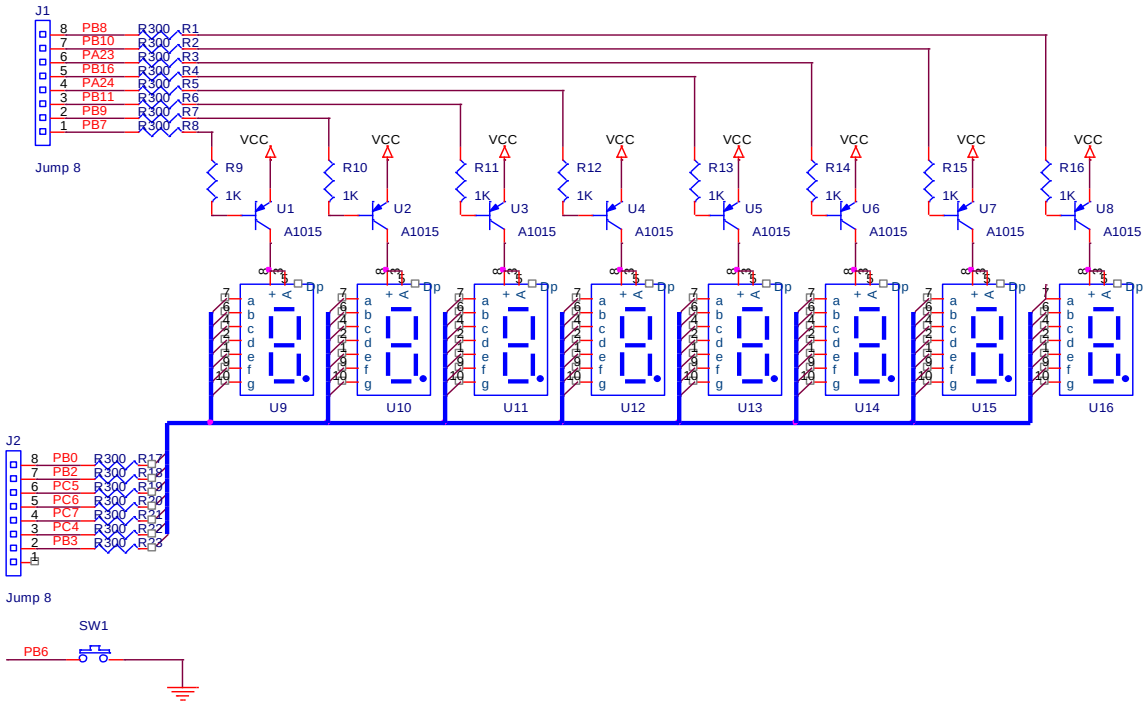
b. Phân công nhiệm vụ:

- ***Driver:*** trong dự án này Driver sẽ có hai nhiệm vụ là:
  - *Nhiệm vụ thứ nhất*: cập nhật xung đếm từ bên ngoài, nếu có xung tác động thì sẽ gửi tín hiệu sang cho User Application.
  - *Nhiệm vụ thứ hai*: nhận dữ liệu từ User Application, điều khiển module Led 7 đoạn theo phương pháp quét thể hiển thị số xung đếm được.

  Như đã nói từ trước, trong dự án này chúng ta sẽ viết hai Driver. Driver thứ nhất làm nhiệm vụ thứ nhất. Driver thứ hai làm nhiệm vụ thứ hai.
- ***User Application:*** có nhiệm vụ cập nhật tín hiệu của Driver thứ nhất, nếu tín hiệu báo có xung thì sẽ tăng biến đếm xung lên một đơn vị, sau đó truyền biến đếm xung này sang cho Driver thứ hai để hiển thị lên module Led 7 đoạn.

II. Thực hiện:

1. Kết nối phần cứng:



Hình 4-17- Sơ đồ kết nối 8 LEDs 7 đoạn dùng phương pháp quét và nút nhấn.

2. Chương trình driver:

- ***Driver thứ nhất:*** Tên update_input_dev.c

/ Khai báo các thư viện cần thiết cho chương trình */*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <mach/at91_tc.h>
#include <asm/gpio.h>
#include <asm/atomic.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
#include <asm/uaccess.h>
#include <linux/clk.h>
#include <linux/irq.h>
#include <linux/time.h>
```

```
#include <linux/jiffies.h>
#include <linux/sched.h>
#include <linux/delay.h>
#include <linux/timer.h>
```

/ Khai báo tên Driver và tên thiết bị */*

```
#define DRVNAME      "update_input_dev"
#define DEVNAME      "update_input"
```

/ Định nghĩa chân của Vi Xử Lý dùng trong chương trình*/*

```
#define COUNTER_PIN AT91_PIN_PB6
```

/ Khai báo biến kiểm tra Driver đã được mở hay chưa */*

```
static atomic_t update_input_open_cnt = ATOMIC_INIT(1);
```

/ Chương trình cập nhật trạng thái của chân Vi Xử Lý (COUNTER_PIN) dùng để đếm xung từ bên ngoài */*

```
static ssize_t update_input_read (struct file *filp, char __iomem
buf[], size_t bufsize, loff_t *f_pos)
{
```

/ Khai báo bộ đệm đọc */*

```
  char driver_read_buf[1];
```

/ Nếu chân COUNTER_PIN được nối đất (giá trị đọc vào từ chân này là 0) thì cho driver_read_buf[0] bằng 0, ngược lại thì thì cho driver_read_buf[0] = 1 */*

```
  if(gpio_get_value(COUNTER_PIN) == 0)
     driver_read_buf[0] = 0;
  else
     driver_read_buf[0] = 1;
```

/ Truyền driver_read_buf sang cho User Application, nếu quá trình truyền thất bại thì báo lỗi cho người dùng biết */*

```
  if(copy_to_user(buf,driver_read_buf,bufsize) != 0)
  {
   printk("Can't read Driver \n");
   return -EFAULT;
  }
}
```

/ Chương trình mở Driver */*

```
static int
update_input_open(struct inode *inode, struct file *file)
{
   int result = 0;
   unsigned int dev_minor = MINOR(inode->i_rdev);
   if (!atomic_dec_and_test(&update_input_open_cnt)) {
      atomic_inc(&update_input_open_cnt);
      printk(KERN_ERR DRVNAME ": Device with minor ID %d already in
      use\n", dev_minor);
      result = -EBUSY;
      goto out;
   }
out:
   return result;
}
```

/ Chương trình đóng Driver */*

```
static int
update_input_close(struct inode * inode, struct file * file)
{
   smp_mb__before_atomic_inc();
   atomic_inc(&update_input_open_cnt);

   return 0;
}
```

/ Cấu trúc file_operations của Driver */*

```
struct file_operations update_input_fops = {
   .read = update_input_read,
   .open    = update_input_open,
   .release = update_input_close,
};
static struct miscdevice update_input_dev = {
        .minor         = MISC_DYNAMIC_MINOR,
        .name         = "update_input",
```

```
        .fops          = &update_input_fops,
};
```

/ Chương trình khởi tạo của Driver, được thực hiện khi người dùng gọi lệnh insmod */*

```
static int __init
update_input_mod_init(void)
{
   int ret=0;
```

/ Khai báo khởi tạo chân COUNTER_PIN ở chế độ gpio ngõ vào có điện trở kéo lên bên trong */*

```
   gpio_free(COUNTER_PIN);
   gpio_request (COUNTER_PIN, NULL);
   at91_set_GPIO_periph (COUNTER_PIN, 1);
   gpio_direction_input(COUNTER_PIN);
   at91_set_deglitch(COUNTER_PIN,1);
   misc_register(&update_input_dev);
   printk(KERN_INFO "sweep_seg_led: Loaded module\n");
   return ret;
}
static void __exit
update_input_mod_exit(void)
{
   misc_deregister(&update_input_dev);
   printk(KERN_INFO "sweep_seg_led: Unloaded module\n");
}
module_init (update_input_mod_init);
module_exit (update_input_mod_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("TranCamNhan");
MODULE_DESCRIPTION("Character device for for generic gpio api");
```

- ***Driver thứ hai:*** Tên sweep_7seg_led_dev.c

/ Khai báo các thư viện cần thiết cho chương trình */*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <mach/at91_tc.h>
#include <asm/gpio.h>
#include <asm/atomic.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
#include <asm/uaccess.h>
#include <linux/clk.h>
#include <linux/irq.h>
#include <linux/time.h>
#include <linux/jiffies.h>
#include <linux/sched.h>
#include <linux/delay.h>
#include <linux/timer.h>
```

/ Khai báo tên Driver và tên thiết bị */*

```
#define DRVNAME      "sweep_7seg_led_dev"
#define DEVNAME      "sweep_7seg_led"
```

/ Định nghĩa các chân của Vi Xử Lý sử dụng trong chương trình */*

```
#define P00             AT91_PIN_PB8
#define P01             AT91_PIN_PB10
#define P02             AT91_PIN_PA23
#define P03             AT91_PIN_PB16
#define P04             AT91_PIN_PA24
#define P05             AT91_PIN_PB11
#define P06             AT91_PIN_PB9
#define P07             AT91_PIN_PB7
#define A               AT91_PIN_PB0
#define B               AT91_PIN_PB2
#define C               AT91_PIN_PC5
```

```
#define D               AT91_PIN_PC6
#define E               AT91_PIN_PC7
#define F               AT91_PIN_PC4
#define G               AT91_PIN_PB3
```

/ Định nghĩa lệnh Set và Clear các chân của Vi Xử Lý */*

```
#define SET_P00()       gpio_set_value(P00,1)
#define SET_P01()       gpio_set_value(P01,1)
#define SET_P02()       gpio_set_value(P02,1)
#define SET_P03()       gpio_set_value(P03,1)
#define SET_P04()       gpio_set_value(P04,1)
#define SET_P05()       gpio_set_value(P05,1)
#define SET_P06()       gpio_set_value(P06,1)
#define SET_P07()       gpio_set_value(P07,1)

#define CLEAR_P00()gpio_set_value(P00,0)
#define CLEAR_P01()     gpio_set_value(P01,0)
#define CLEAR_P02()     gpio_set_value(P02,0)
#define CLEAR_P03()     gpio_set_value(P03,0)
#define CLEAR_P04()     gpio_set_value(P04,0)
#define CLEAR_P05()     gpio_set_value(P05,0)
#define CLEAR_P06()     gpio_set_value(P06,0)
#define CLEAR_P07()     gpio_set_value(P07,0)

#define SET_A()         gpio_set_value(A,1)
#define SET_B()         gpio_set_value(B,1)
#define SET_C()         gpio_set_value(C,1)
#define SET_D()         gpio_set_value(D,1)
#define SET_E()         gpio_set_value(E,1)
#define SET_F()         gpio_set_value(F,1)
#define SET_G()         gpio_set_value(G,1)

#define CLEAR_A()       gpio_set_value(A,0)
#define CLEAR_B()       gpio_set_value(B,0)
#define CLEAR_C()       gpio_set_value(C,0)
```

```
#define CLEAR_D()       gpio_set_value(D,0)
#define CLEAR_E()       gpio_set_value(E,0)
#define CLEAR_F()       gpio_set_value(F,0)
#define CLEAR_G()       gpio_set_value(G,0)
```

*/*Định nghĩa số định danh lệnh và lệnh sử dụng trong ioctl */*

```
#define SWEEP_LED_DEV_MAGIC  'B'
#define UPDATE_DATA_SWEEP_7SEG _IO(SWEEP_LED_DEV_MAGIC,1)
```

/ Khai báo các biến cần thiết sử dụng trong chương trình */*

```
static atomic_t sweep_7seg_led_open_cnt = ATOMIC_INIT(1);
char DataDisplay[8]={0,12,12,12,12,12,12,12};
char SevSegCode[]={ 0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,
                    0x80, 0x90, 0x3F, 0x77, 0xFF};
char ChooseLedActive[]={ 0xfe, 0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0xbf,
                         0x7f};
int i=0;
```

/ Khai báo cấu trúc timer ảo trong kernel với tên my_timer */*

```
struct timer_list my_timer;
```

/ Chương trình chọn Led 7 đoạn tích cực */*

```
void choose_led_active(char data) {
   (data&(1<<0))? SET_P00():CLEAR_P00();
   (data&(1<<1))? SET_P01():CLEAR_P01();
   (data&(1<<2))? SET_P02():CLEAR_P02();
   (data&(1<<3))? SET_P03():CLEAR_P03();
   (data&(1<<4))? SET_P04():CLEAR_P04();
   (data&(1<<5))? SET_P05():CLEAR_P05();
   (data&(1<<6))? SET_P06():CLEAR_P06();
   (data&(1<<7))? SET_P07():CLEAR_P07();
}
```

/ Chương trình viết dữ liệu hiển thị ra các chân Vi Xử Lý */*

```
void write_data_led(char data) {
   (data&(1<<0))? SET_A():CLEAR_A();
   (data&(1<<1))? SET_B():CLEAR_B();
   (data&(1<<2))? SET_C():CLEAR_C();
   (data&(1<<3))? SET_D():CLEAR_D();
```

```
   (data&(1<<4))? SET_E():CLEAR_E();
   (data&(1<<5))? SET_F():CLEAR_F();
   (data&(1<<6))? SET_G():CLEAR_G();
}
```

/ Chương trình chuyển một số có nhiều chữ số ra các số BCD */*

```
void hex_to_bcd (unsigned long int data) {
   DataDisplay[0] = data % 10;
   DataDisplay[1] = (data % 100)/10;
   DataDisplay[2] = (data % 1000)/100;
   DataDisplay[3] = (data % 10000)/1000;
   DataDisplay[4] = (data % 100000)/10000;
   DataDisplay[5] = (data % 1000000)/100000;
   DataDisplay[6] = (data % 10000000)/1000000;
   DataDisplay[7] = data/10000000;
```

/ Chương trình xóa số 0 vô nghĩa */*

```
   if (data/10000000 == 0) {
     DataDisplay[7] = 12;
     if ((data % 10000000)/1000000 == 0) {
          DataDisplay[6] = 12;
          if ( (data % 1000000)/100000 == 0) {
            DataDisplay[5] = 12;
             if ((data % 100000)/10000 == 0) {
               DataDisplay[4] = 12;
               if ( (data % 10000)/1000 == 0) {
                 DataDisplay[3] = 12;
                  if( (data % 1000)/100 == 0) {
                    DataDisplay[2] = 12;
                    if( (data % 100)/10 == 0)
                    DataDisplay[1] = 12;
                  }
               }
             }
          }
        }
```

```
    }
}
```

/ Chương trình phục vụ ngắt: cứ mỗi 1ms chương trình này sẽ được hiện một lần, chương trình này có nhiệm vụ chọn lần lược chọn các Led tích cực và xuất dữ liệu hiển thị ứng với từng Led */*

```
void
sweep_dis_irq(unsigned long data) {
  choose_led_active(ChooseLedActive[i]);
  write_data_led(SevSegCode[DataDisplay[i]]);
  i++;
  if(i==8)
  i=0;
```

/ Khởi tạo lại giá trị của timer ảo với chu kỳ là 1ms*/*

```
  mod_timer (&my_timer, jiffies + 1);
}
```

/ chương trình nhận dữ liệu từ User Application và chuyển dữ liệu vừa nhận sang các số BCD */*

```
static int
sweep_7seg_led_ioctl(struct inode * inode, struct file * file,
unsigned int cmd,unsigned long int arg) {
   int retval=0;
   switch (cmd) {
         case UPDATE_DATA_SWEEP_7SEG:
               hex_to_bcd(arg);
         break;
         default:
               printk("The function you type does not exist\n");
               retval=-1;
         break;
   }
}
static int
sweep_7seg_led_open(struct inode *inode, struct file *file) {
```

```
    int result = 0;
    unsigned int dev_minor = MINOR(inode->i_rdev);
    if (!atomic_dec_and_test(&sweep_7seg_led_open_cnt)) {
        atomic_inc(&sweep_7seg_led_open_cnt);
        printk(KERN_ERR DRVNAME ": Device with minor ID %d already in
        use\n", dev_minor);
        result = -EBUSY;
        goto out;
    }
out:
    return result;
}
static int
sweep_7seg_led_close(struct inode * inode, struct file * file){
    smp_mb__before_atomic_inc();
    atomic_inc(&sweep_7seg_led_open_cnt);
    return 0;
}
struct file_operations sweep_7seg_led_fops = {
    .ioctl   = sweep_7seg_led_ioctl,
    .open    = sweep_7seg_led_open,
    .release = sweep_7seg_led_close,
};
static struct miscdevice sweep_7seg_led_dev = {
        .minor         = MISC_DYNAMIC_MINOR,
        .name         = "sweep_7seg_led",
        .fops         = &sweep_7seg_led_fops,
};
static int __init
sweep_7seg_led_mod_init(void) {
    int ret=0;
/* Khởi tạo các chân của Vi Xử Lý ở chế độ ngõ ra, có điện trở  kéo lên */
    gpio_request (P00, NULL);
    gpio_request (P01, NULL);
```

```
gpio_request (P02, NULL);
gpio_request (P03, NULL);
gpio_request (P04, NULL);
gpio_request (P05, NULL);
gpio_request (P06, NULL);
gpio_request (P07, NULL);
at91_set_GPIO_periph (P00, 1);
at91_set_GPIO_periph (P01, 1);
at91_set_GPIO_periph (P02, 1);
at91_set_GPIO_periph (P03, 1);
at91_set_GPIO_periph (P04, 1);
at91_set_GPIO_periph (P05, 1);
at91_set_GPIO_periph (P06, 1);
at91_set_GPIO_periph (P07, 1);
gpio_direction_output(P00, 0);
gpio_direction_output(P01, 0);
gpio_direction_output(P02, 0);
gpio_direction_output(P03, 0);
gpio_direction_output(P04, 0);
gpio_direction_output(P05, 0);
gpio_direction_output(P06, 0);
gpio_direction_output(P07, 0);
gpio_request (A, NULL);
gpio_request (B, NULL);
gpio_request (C, NULL);
gpio_request (D, NULL);
gpio_request (E, NULL);
gpio_request (F, NULL);
gpio_request (G, NULL);
at91_set_GPIO_periph (A, 1);
at91_set_GPIO_periph (B, 1);
at91_set_GPIO_periph (C, 1);
at91_set_GPIO_periph (D, 1);
at91_set_GPIO_periph (E, 1);
at91_set_GPIO_periph (F, 1);
```

```
    at91_set_GPIO_periph (G, 1);
    gpio_direction_output(A, 0);
    gpio_direction_output(B, 0);
    gpio_direction_output(C, 0);
    gpio_direction_output(D, 0);
    gpio_direction_output(E, 0);
    gpio_direction_output(F, 0);
    gpio_direction_output(G, 0);
    /* Khởi tạo timer ảo */
    init_timer (&my_timer);
    my_timer.expires = jiffies+1; //chu kỳ đầu tiên là 1ms
    my_timer.data = 0;
    my_timer.function = sweep_dis_irq;
    add_timer (&my_timer);

    misc_register(&sweep_7seg_led_dev);
    printk(KERN_INFO "sweep_seg_led: Loaded module\n");
    return ret;
}
static void __exit
sweep_7seg_led_mod_exit(void){
    del_timer_sync(&my_timer);
    misc_deregister(&sweep_7seg_led_dev);
    printk(KERN_INFO "sweep_seg_led: Unloaded module\n");
}
module_init (sweep_7seg_led_mod_init);
module_exit (sweep_7seg_led_mod_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("TranCamNhan");
MODULE_DESCRIPTION("Character device for for generic gpio api");
```

3. Chương trình application:

- ***User Application:*** counter_display_7seg_app.c

```
/* Khai báo các thư viện cần thiết dùng trong chương trình*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
#include <pthread.h>
#include <unistd.h>
/* Định nghĩa số định danh lệnh và lệnh dùng trong ioctl*/
#define SWEEP_LED_DEV_MAGIC  'B'
#define UPDATE_DATA_SWEEP_7SEG _IO(SWEEP_LED_DEV_MAGIC,1)
/* Khai báo các biến cần thiết trong chương trình */
int update_input_fd;
int sweep_7seg_led_fd;
/* Chương trình in ra trên màn hình cú pháp lệnh sử dụng  chương trình */
int print_usage(void) {
  printf("./counter_display_7seg_app <number>\n");
  return -1;
}
/* Chương trình chính */
int
main(int argc, char **argv) {
   char user_read_buf[1];
   long int counter = 0;
/* Mở Driver update_input_dev.ko */
   if ((update_input_fd = open("/dev/update_input", O_RDWR)) < 0) {
         printf("Error whilst opening /dev/update_input device\n");
         return -1;
   }
```

/ Mở Driver sweep_7seg_led_dev.ko */*

```
if ((sweep_7seg_led_fd = open("/dev/sweep_7seg_led", O_RDWR)) <
0) {
      printf("Error     whilst     opening     /dev/sweep_7seg_led
      device\n");
      return -1;
}
if (argc != 2)
  print_usage();
else if (!strcmp (argv[1],"start"))
while(1) {
```

/ Chương trình đọc trạng thái chân đếm xung từ Driver update_input_dev. */*

```
          if(read(update_input_fd,user_read_buf,1)<0)
             printf("User: read from driver fail\n");
          else {
```

/ Nếu chân đếm xung ở mức thấp thì sẽ tăng biến counter lên một đơn vị và truyền biến counter này sang cho Driver sweep_7seg_led_dev.ko */*

```
                if (user_read_buf[0] ==0) {
                      usleep(50000); //Delay để chống dội
                      counter++;
                      ioctl(sweep_7seg_led_fd,
                      UPDATE_DATA_SWEEP_7SEG, counter);
                      printf("value counter: %d\n",counter);
```

/ *Nếu chân đếm xung vẫn ở mức thấp thì tiếp tục cập nhật trạng thái của chân này, khi nào trạng thái của chân này lên cao thì mới ngưng cập nhật trạng thái chân này* */

```
                        while(user_read_buf[0] ==0)
                         {
                          read(update_input_fd,user_read_buf,1);
                         }
                        usleep(50000);//Delay để chống dội
                      }
                }
   }
   else
        print_usage();
   return 0;
}
```

III. Kết luận và bài tập:

1. *Kết Luận:*

Như vậy trong dự án này chúng ta đã đếm xung cạnh xuống chân PB6 của Vi Xử Lý và hiển thị số xung đếm được ra 8 Led 7 đoạn theo phương pháp quét.

Chúng ta đã viết hai Driver khác nhau và một User Application để thực hiện yêu cầu của dự án này. Thực ra đối với dự án này chúng ta có thể chỉ cần viết một Driver và một User Application là có thể thực hiện được yêu cầu. Nên với cách viết chương trình như bài này là nhằm giới thiệu đến cho người học phương pháp viết nhiều Driver cho một ứng dụng, để chúng ta có thể áp dụng phương pháp này giải quyết các ứng dụng có yêu cầu phức tạp hơn.

Trong bài này chúng ta cũng đã tập làm quen với việc sử dụng ngắt timer ảo của kernel với chu kỳ 1ms.

Chú ý: Trong bài này chúng ta đã thay đổi giá trị của số HZ là 1000. Nghĩa là chúng ta sẽ có 1000 tick trong 1 giây trong kernel (mỗi tick cách nhau 1ms).

2. *Bài tập:*

1. Viết chương trình đếm lên từ 5 đến 55 và về lại 5 hiển thị trên module 8 Led 7 đoạn, có xóa số 0 vô nghĩa. Viết 2 Driver và một chương trình User Application.
2. Viết chương trình đếm lên xuống nằm trong khoảng 0 đến 255 hiển thị trên module 8 Led 7 đoạn, xóa số 0 vô nghĩa. Sử dụng hai nút nhấn, một nút nhấn dùng để đếm lên, một nút nhấn dùng để đếm xuống. Viết 2 Driver và một chương trình User Application.

BÀI 6

GIAO TIẾP ĐIỀU KHIỂN LED MA TRẬN 8x8

I. Phác thảo dự án:

Hiện nay, ma trận Led là linh kiện điện tử được sử dụng phổ biến trong công nghiệp quảng cáo. Không giống như Led 7 đoạn chỉ có thể hiển thị những con số, ma trận Led có thể hiển thị được tất cả các ký tự, hình ảnh mà người dùng mong muốn. Ngoài ra ma trận Led còn hiển thị được nhiều hiệu ứng đa dạng khác nhau như: dịch trái, dịch phải, chuyển động của sự vật….

Ngoài thực tế, ma trận Led có nhiều loại khác nhau với kích thước khác nhau, số lượng Led khác nhau, số lượng màu Led khác nhau. Thông thường một ma trận Led có ít nhất 8x8 Led đơn trên nó, vì số lượng Led trên một ma trận Led rất lớn nên chúng ta không thể dùng phương pháp truy xuất trực tiếp từng Led được. Nên để hiển thị được các ký tự hình ảnh trên ma trận Led chúng ta phải dùng phương pháp quét.

Kỹ thuật quét ma trận Led chúng ta đã được học trong môn Thực Tập Vi Xử Lý nên trong quyển sách này chúng tôi sẽ không trình bày lại. Chúng ta chỉ tập trung vào việc làm thế nào để quét và điều khiển ma trận Led bằng hệ thống nhúng.

a. Yêu cầu dự án:

Trong dự án này chúng ta sẽ điều khiển ma trận Led 8X8 hai màu xanh đỏ hiển thị tất cả các ký tự có trên bàn phím máy vi tính. Vì ma trận Led chỉ có 8 hàng và 8 cột nên mỗi lần chúng ta chỉ hiển thị được một ký tự. Cú pháp lệnh khi thực thi như sau:

```
<tên chương trình>  <ký tự muốn hiển thị>
```

Trong đó:

`<tên chương trình>` là tên chương trình sau khi đã biên dịch xong.

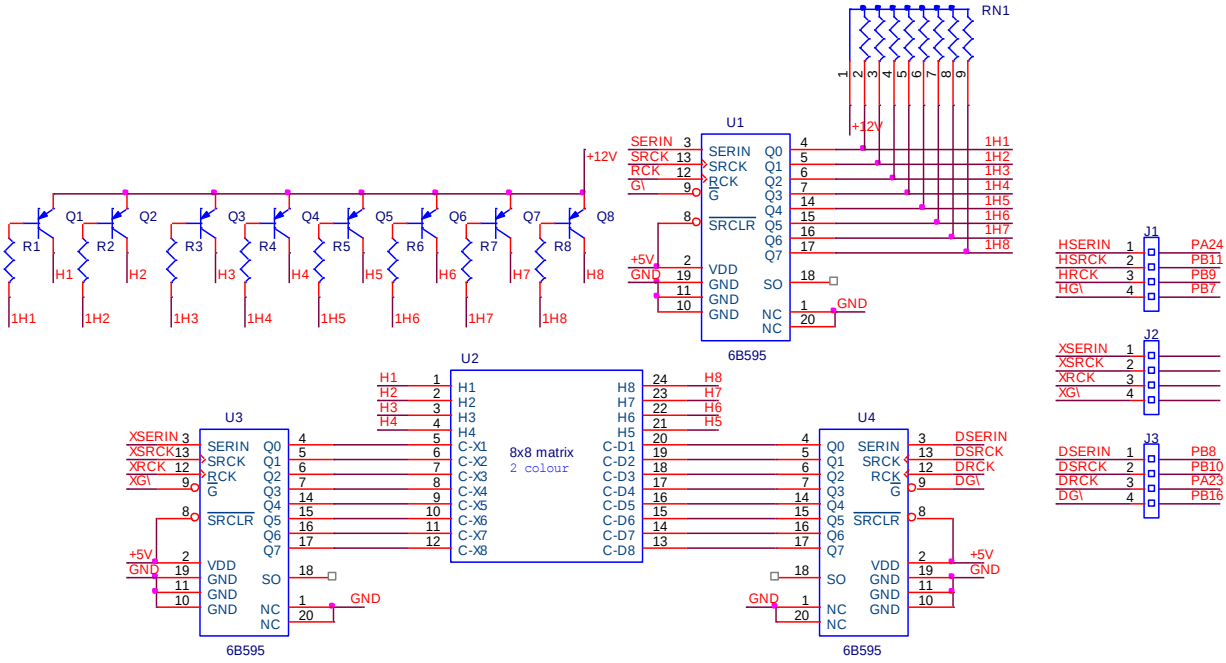
`<ký tự muốn hiển thị>` là ký tự muốn hiển thị trên ma trận Led.

b. Phân công nhiệm vụ:

- ***Driver:*** Nhận dữ liệu là mã Asscii của ký tự muốn hiển thị từ User Application thông qua hàm `write()`, chuyển đổi mã Asscii này sang mã ma trận (một ký tự mã Asscii tương ứng với 5 byte của mã ma trận) và quét module ma trận Led theo các mã ma trận này.
- ***User Application:*** Nhận tham số từ người dùng và chuyển tham số này sang cho Driver.

II. Thực hiện:

1. Kết nối phần cứng theo sơ đồ sau:



Hình 4-18- Sơ đồ kết nối LED ma trân 8x8.

2. Viết chương trình:

- ***Driver:*** Tên Matrix_Led_Display_dev.c

/ khai báo các thư viện cần thiết trong chương trình */*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <mach/at91_tc.h>
#include <asm/gpio.h>
#include <asm/atomic.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
#include <asm/uaccess.h>
#include <linux/clk.h>
#include <linux/irq.h>
#include <linux/time.h>
#include <linux/jiffies.h>
#include <linux/sched.h>
#include <linux/delay.h>
```

/ khai báo tên Driver và tên thiết bị */*

```
#define DRVNAME      "matrix_led_dev"
#define DEVNAME      "matrix_led"
```

/ khai báo các chân dùng để điều khiển modum ma trận Led */*

```
#define DATA_C          AT91_PIN_PB8
#define CLOCK_C         AT91_PIN_PB10
#define LATCH_C         AT91_PIN_PA23
#define OE_C            AT91_PIN_PB16
#define DATA_R          AT91_PIN_PA24
#define CLOCK_R         AT91_PIN_PB11
#define LATCH_R         AT91_PIN_PB9
#define OE_R            AT91_PIN_PB7
```

/ các lệnh tạo mức High và Low cho các chân điều khiển module Led */*

```
#define SET_DATA_C()          gpio_set_value(DATA_C,1)
#define SET_CLOCK_C()         gpio_set_value(CLOCK_C,1)
#define SET_LATCH_C()         gpio_set_value(LATCH_C,1)
#define SET_OE_C()            gpio_set_value(OE_C,1)
#define SET_DATA_R()          gpio_set_value(DATA_R,1)
```

```
#define SET_CLOCK_R()          gpio_set_value(CLOCK_R,1)
#define SET_LATCH_R()          gpio_set_value(LATCH_R,1)
#define SET_OE_R()             gpio_set_value(OE_R,1)

#define CLEAR_DATA_C()         gpio_set_value(DATA_C,0)
#define CLEAR_CLOCK_C()        gpio_set_value(CLOCK_C,0)
#define CLEAR_LATCH_C()        gpio_set_value(LATCH_C,0)
#define CLEAR_OE_C()           gpio_set_value(OE_C,0)
#define CLEAR_DATA_R()         gpio_set_value(DATA_R,0)
#define CLEAR_CLOCK_R()        gpio_set_value(CLOCK_R,0)
#define CLEAR_LATCH_R()        gpio_set_value(LATCH_R,0)
#define CLEAR_OE_R()           gpio_set_value(OE_R,0)
```

/ khai báo các biến cần dùng trong chương trình */*

```
static atomic_t matrix_led_open_cnt = ATOMIC_INIT(1);
int i=0;
```

/ DataDisplay là mảng chứa dữ liệu hiển thị ra ma trận Led (dữ liệu dịch ra IC 6B595 điều khiển hàng), mảng có 8 phần tử ứng với dữ liệu hàng của 8 cột ma trận Led */*

```
unsigned char DataDisplay[8];
unsigned char MatrixCode[5];
```

*/*mảng ColumnCode tạo dữ liệu một bit dịch chuyển trên 8 cột của ma trận Led */*

```
unsigned char
ColumnCode[]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
```

/ bảng mã font Asscii sang ma trận Led*/*

```
unsigned char font[]={
0xFF,0xFF,0xFF,0xFF,0xFF,      //SPACE    0
0xFF,0xFF,0xA0,0xFF,0xFF,      //!        1
0xFF,0xFF,0xF8,0xF4,0xFF,      //'        2
0xEB,0x80,0xEB,0x80,0xEB,      //#        3
0xDB,0xD5,0x80,0xD5,0xED,      //$        4
0xD8,0xEA,0x94,0xAB,0x8D,      //%        5
0xC9,0xB6,0xA9,0xDF,0xAF,      //&        6
```

```
0xFF,0xFF,0xF8,0xF4,0xFF,     //'        7
0xFF,0xE3,0xDD,0xBE,0xFF,     //(        8
0xFF,0xBE,0xDD,0xE3,0xFF,     //)        9
0xD5,0xE3,0x80,0xE3,0xD5,     //*        10
0xF7,0xF7,0xC1,0xF7,0xF7,     //+        11
0xFF,0xA7,0xC7,0xFF,0xFF,     //,        12
0xF7,0xF7,0xF7,0xF7,0xF7,     //-        13
0xFF,0x9F,0x9F,0xFF,0xFF,     //x        14
0xFF,0xC9,0xC9,0xFF,0xFF,     ///        15
0xC1,0xAE,0xB6,0xBA,0xC1,     //0        16
0xFF,0xBD,0x80,0xBF,0xFF,     //1        17
0x8D,0xB6,0xB6,0xB6,0xB9,     //2        18
0xDD,0xBE,0xB6,0xB6,0xC9,     //3        19
0xE7,0xEB,0xED,0x80,0xEF,     //4        20
0xD8,0xBA,0xBA,0xBA,0xC6,     //5        21
0xC3,0xB5,0xB6,0xB6,0xCF,     //6        22
0xFE,0x8E,0xF6,0xFA,0xFC,     //7        23
0xC9,0xB6,0xB6,0xB6,0xC9,     //8        24
0xF9,0xB6,0xB6,0xD6,0xE1,     //9        25
0xFF,0xC9,0xC9,0xFF,0xFF,     //:        26
0xFF,0xA4,0xC4,0xFF,0xFF,     ////       27
0xF7,0xEB,0xDD,0xBE,0xFF,     //<        28
0xEB,0xEB,0xEB,0xEB,0xEB,     //=        29
0xFF,0xBE,0xDD,0xEB,0xF7,     //>        30
0xFD,0xFE,0xAE,0xF6,0xF9,     //?        31
0xCD,0xB6,0x8E,0xBE,0xC1,     //@        32
0x83,0xF5,0xF6,0xF5,0x83,     //A        33
0xBE,0x80,0xB6,0xB6,0xC9,     //B        34
0xC1,0xBE,0xBE,0xBE,0xDD,     //C        35
0xBE,0x80,0xBE,0xBE,0xC1,     //D        36
0x80,0xB6,0xB6,0xB6,0xBE,     //E        37
0x80,0xF6,0xF6,0xFE,0xFE,     //F        38
0xC1,0xBE,0xB6,0xB6,0xC5,     //G        39
0x80,0xF7,0xF7,0xF7,0x80,     //H        40
0xFF,0xBE,0x80,0xBE,0xFF,     //I        41
```

```
0xDF,0xBF,0xBE,0xC0,0xFE,     //J        42
0x80,0xF7,0xEB,0xDD,0xBE,     //K        43
0x80,0xBF,0xBF,0xBF,0xFF,     //L        44
0x80,0xFD,0xF3,0xFD,0x80,     //M        45
0x80,0xFD,0xFB,0xF7,0x80,     //N        46
0xC1,0xBE,0xBE,0xBE,0xC1,     //O        47
0x80,0xF6,0xF6,0xF6,0xF9,     //P        48
0xC1,0xBE,0xAE,0xDE,0xA1,     //Q        49
0x80,0xF6,0xE6,0xD6,0xB9,     //R        50
0xD9,0xB6,0xB6,0xB6,0xCD,     //S        51
0xFE,0xFE,0x80,0xFE,0xFE,     //T        52
0xC0,0xBF,0xBF,0xBF,0xC0,     //U        53
0xE0,0xDF,0xBF,0xDF,0xE0,     //V        54
0xC0,0xBF,0xCF,0xBF,0xC0,     //W        55
0x9C,0xEB,0xF7,0xEB,0x9C,     //X        56
0xFC,0xFB,0x87,0xFB,0xFC,     //Y        57
0x9E,0xAE,0xB6,0xBA,0xBC,     //Z        58
0xFF,0x80,0xBE,0xBE,0xFF,     //[        59
0xFD,0xFB,0xF7,0xEF,0xDF,     //\        60
0xFF,0xBE,0xBE,0x80,0xFF,     //]        61
0xFB,0xFD,0xFE,0xFD,0xFB,     //^        62
0x7F,0x7F,0x7F,0x7F,0x7F,     //_        63
0xFF,0xFF,0xF8,0xF4,0xFF,     //'        64
0xDF,0xAB,0xAB,0xAB,0xC7,     //a        65
0x80,0xC7,0xBB,0xBB,0xC7,     //b        66
0xFF,0xC7,0xBB,0xBB,0xBB,     //c        67
0xC7,0xBB,0xBB,0xC7,0x80,     //d        68
0xC7,0xAB,0xAB,0xAB,0xF7,     //e        69
0xF7,0x81,0xF6,0xF6,0xFD,     //f        70
0xF7,0xAB,0xAB,0xAB,0xC3,     //g        71
0x80,0xF7,0xFB,0xFB,0x87,     //h        72
0xFF,0xBB,0x82,0xBF,0xFF,     //I        73
0xDF,0xBF,0xBB,0xC2,0xFF,     //j        74
0xFF,0x80,0xEF,0xD7,0xBB,     //k        75
0xFF,0xBE,0x80,0xBF,0xFF,     //l        76
```

```
0x83,0xFB,0x87,0xFB,0x87,     //m        77
0x83,0xF7,0xFB,0xFB,0x87,     //n        78
0xC7,0xBB,0xBB,0xBB,0xC7,     //o        79
0x83,0xEB,0xEB,0xEB,0xF7,     //p        80
0xF7,0xEB,0xEB,0xEB,0x83,     //q        81
0x83,0xF7,0xFB,0xFB,0xF7,     //r        82
0xB7,0xAB,0xAB,0xAB,0xDB,     //s        83
0xFF,0xFB,0xC0,0xBB,0xBB,     //t        84
0xC3,0xBF,0xBF,0xDF,0x83,     //u        85
0xE3,0xDF,0xBF,0xDF,0xE3,     //v        86
0xC3,0xBF,0xCF,0xBF,0xC3,     //w        87
0xBB,0xD7,0xEF,0xD7,0xBB,     //x        88
0xF3,0xAF,0xAF,0xAF,0xC3,     //y        89
0xBB,0x9B,0xAB,0xB3,0xBB,     //z        90
0xFB,0xE1,0xE0,0xE1,0xFB,     //^        91
0xE3,0xE3,0xC1,0xE3,0xF7,     //->       92
0xF7,0xE3,0xC1,0xE3,0xE3,     //<-       93
0xEF,0xC3,0x83,0xC3,0xEF,     //         94
0xFF,0xFF,0xFF,0xFF,0xFF //BLANK CHAR 95
};

void __iomem *at91tc0_base;
struct clk *at91tc0_clk;
```

*/*chương trình viết dữ liệu hàng ra các port của IC 6B595 điều khiển hàng */*

```
void row_write_data(unsigned char data)
{
   int j;
   for (j=0;j<8;j++)
   {
```

*/ *set hoặc clear chân DATA_R tùy thuộc vào giá trị của data */*

```
      (data&(1<<j))? SET_DATA_R():CLEAR_DATA_R();
```

/ tạo xung ở chân CLOCK_R để dịch dữ liệu ra IC 6B595 */*

```
      SET_CLOCK_R();
      CLEAR_CLOCK_R();
```

```
    }
/* tạo xung ở chân LATCH_R để chốt dữ liệu trong IC 6B595 */
    SET_LATCH_R();
    CLEAR_LATCH_R();
}
/*chương trình viết dữ liệu cột ra các port của IC 6 B595 điều  khiển cột */
void column_write_data(unsigned char data)
{
    int j;
    for (j=0;j<8;j++)
    {
      (data&(1<<j))? SET_DATA_C():CLEAR_DATA_C();
      SET_CLOCK_C();
      CLEAR_CLOCK_C();
    }
    SET_LATCH_C();
    CLEAR_LATCH_C();
}
/* chương trình phục vụ ngắt, cứ 1 ms chương trình phục vụ ngắt  này sẽ được thực
hiện một lần */
static irqreturn_t at91tc0_isr(int irq, void *dev_id) {
    int status;
    // Read TC0 status register to reset RC compare status.
    status = ioread32(at91tc0_base + AT91_TC_SR);
/*set chân OE_C và OE_R để cách ly dữ liệu bên trong 6B595 và  dữ liệu đã xuất ra
các chân của 6B595 */
    SET_OE_C();
    SET_OE_R();
/* viết dữ liệu hàng và cột tương ứng ra module ma trận Led */
    row_write_data(DataDisplay[i]);
    column_write_data(ColumnCode[i]);
/*clear chân OE_C và OE_R để xuất dữ liệu bên trong 6B595 ra  các chân dữ liệu
của 6B595 */
```

```
   CLEAR_OE_C();
   CLEAR_OE_R();
/* biến i = 0 ứng với vị trí cột đầu tiên của ma trận Led, i =  1 ứng với vị trí cột thứ 2
của ma trận Led, tương tự i = 7 ứng  với vị trí cột cuối cùng của ma trận Led */
   i++;
/* khi vị trí quét đến cột thứ 8 của ma trận Led thì ta quay  lại vị trí thứ nhất */
   if (i==8) {
         i = 0;
   }
   return IRQ_HANDLED;
}
/* chương trình chuyển mã Asscii sang mã ma trận Led */
void asscii_to_matrix_led (unsigned char  data)
{
   MatrixCode[0] = font[data * 5 +0];
   MatrixCode[1] = font[data * 5 +1];
   MatrixCode[2] = font[data * 5 +2];
   MatrixCode[3] = font[data * 5 +3];
   MatrixCode[4] = font[data * 5 +4];
}
/* chương trình nhận dữ liệu từ User Application */
static ssize_t matrix_led_write (struct file *filp, unsigned char
__iomem buf[], size_t bufsize, loff_t *f_pos)
{
   unsigned char write_buf[1];
   int write_size = 0;
/*nhận dữ liệu từ user và chép vào write_buf */
   if (copy_from_user (write_buf, buf, 1) != 0)
   {
         return -EFAULT;
   }
   else
   {
```

```
        write_size = bufsize;
```

*/*trong bảng mã Asscii ký tự khoảng trắng có giá trị là 32, nhưng trong bảng font ma trận Led ký tự khoảng trắng lại là 5 byte đầu tiên trong mảng nên khi chuyển từ mã Asscii sang mã ma trận Led chúng ta phải dùng (mã Asscii – 32) */*

```
        asscii_to_matrix_led(write_buf[0]-32);

         printk("asscii to matrix led %d complete\n",
write_buf[0]);
```

/ chép mã ma trận của ký tự muốn hiển thị vào mảng hiển thị*/*

```
        DataDisplay[0] = ~0xFF;
        DataDisplay[1] = ~MatrixCode[0];
        DataDisplay[2] = ~MatrixCode[1];
        DataDisplay[3] = ~MatrixCode[2];
        DataDisplay[4] = ~MatrixCode[3];
        DataDisplay[5] = ~MatrixCode[4];
        DataDisplay[6] = ~0xFF;
        DataDisplay[7] = ~0xFF;

   }
   return write_size;
}

static int
matrix_led_open(struct inode *inode, struct file *file)
{
   int result = 0;
   unsigned int dev_minor = MINOR(inode->i_rdev);
   if (!atomic_dec_and_test(&matrix_led_open_cnt)) {
      atomic_inc(&matrix_led_open_cnt);
      printk(KERN_ERR DRVNAME ": Device with minor ID %d already in
      use\n", dev_minor);
      result = -EBUSY;
      goto out;
   }
```

```
out:
   return result;
}

static int
matrix_led_close(struct inode * inode, struct file * file)
{
   smp_mb__before_atomic_inc();
   atomic_inc(&matrix_led_open_cnt);

   return 0;
}

struct file_operations matrix_led_fops = {
   .write     = matrix_led_write,
   .open    = matrix_led_open,
   .release = matrix_led_close,
};

static struct miscdevice matrix_led_dev = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name          = "matrix_led",
        .fops          = &matrix_led_fops,
};

static int __init
matrix_led_mod_init(void)
{
   int ret=0;
   gpio_request (DATA_C, NULL);
   gpio_request (CLOCK_C, NULL);
   gpio_request (LATCH_C, NULL);
   gpio_request (OE_C, NULL);
   gpio_request (DATA_R, NULL);
   gpio_request (CLOCK_R, NULL);
```

```
    gpio_request (LATCH_R, NULL);
    gpio_request (OE_R, NULL);

    at91_set_GPIO_periph (DATA_C, 1);
    at91_set_GPIO_periph (CLOCK_C, 1);
    at91_set_GPIO_periph (LATCH_C, 1);
    at91_set_GPIO_periph (OE_C, 1);
    at91_set_GPIO_periph (DATA_R, 1);
    at91_set_GPIO_periph (CLOCK_R, 1);
    at91_set_GPIO_periph (LATCH_R, 1);
    at91_set_GPIO_periph (OE_R, 1);


    gpio_direction_output(DATA_C, 0);
    gpio_direction_output(CLOCK_C, 0);
    gpio_direction_output(LATCH_C, 0);
    gpio_direction_output(OE_C, 0);
    gpio_direction_output(DATA_R, 0);
    gpio_direction_output(CLOCK_R, 0);
    gpio_direction_output(LATCH_R, 0);
    gpio_direction_output(OE_R, 0);

    at91tc0_clk = clk_get(NULL, // Device pointer - not required.
                    "tc0_clk"); // Clock name.
    clk_enable(at91tc0_clk);
    at91tc0_base = ioremap_nocache(AT91SAM9260_BASE_TC0,
    64);
  if (at91tc0_base == NULL)
     {
         printk(KERN_INFO "at91adc: TC0 memory mapping failed\n");
         ret = -EACCES;
         goto exit_5;
     }
```

/ Configure TC0 in waveform mode, TIMER_CLK1 and to generate interrupt on RC compare.*

Load 50000 to RC so that with TIMER_CLK1 = MCK/2 = 50MHz, the interrupt will be

*generated every 1/50MHz * 50000 = 20nS * 50000 = 1 milli second.*

NOTE: Even though AT91_TC_RC is a 32-bit register, only 16-bits are programmble.

```
*/
iowrite32(50000, (at91tc0_base + AT91_TC_RC));
iowrite32((AT91_TC_WAVE | AT91_TC_WAVESEL_UP_AUTO), (at91tc0_base
+ AT91_TC_CMR));
iowrite32(AT91_TC_CPCS, (at91tc0_base + AT91_TC_IER));
iowrite32((AT91_TC_SWTRG | AT91_TC_CLKEN), (at91tc0_base +
AT91_TC_CCR));
```

*/*Install interrupt for TC0*/*

```
ret = request_irq(AT91SAM9260_ID_TC0, // Interrupt number
at91tc0_isr, // Pointer to the interrupt sub-routine
0, // Flags - fast, shared or contributing to entropy pool
"matrix_led_irq", // Device name to show as owner in /proc/interrupts
NULL); // Private data for shared interrupts
   if (ret != 0)
   {
         printk(KERN_INFO "matrix_led_irq: Timer interrupt request
         failed\n");
         ret = -EBUSY;
         goto exit_6;
   }
   misc_register(&matrix_led_dev);
   printk(KERN_INFO "matrix_led: Loaded module\n");
   return ret;
   exit_6:
```

```
    iounmap(at91tc0_base);
    exit_5:
    clk_disable(at91tc0_clk);
    return ret;
}

static void __exit
matrix_led_mod_exit(void)
{
    iounmap(at91tc0_base);
    clk_disable(at91tc0_clk);
    // Free TC0 IRQ.
    free_irq(AT91SAM9260_ID_TC0, //Interrupt number
    NULL); // Private data for shared interrupts
    misc_deregister(&matrix_led_dev);
    printk(KERN_INFO "matrix_led: Unloaded module\n");
}

module_init (matrix_led_mod_init);
module_exit (matrix_led_mod_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("TranCamNhan");
MODULE_DESCRIPTION("Character device for for generic gpio api");
```

- ***User Application:*** Tên Matrix_Led_Display_app.c

```
/* khai báo các thư viện cần dùng trong chương trình */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
```

```
#include <pthread.h>
/* khai báo các biến cần dùng trong chương trình */
int matrix_led_fd;
unsigned long int ioctl_buf[3]={0,0,0};
unsigned char write_buf[1];
/* chương trình in ra màn hình cú pháp lệnh để hiển thị chữ ra  ma trận Led */
void print_usage(){
   printf("matrixled_app <letter>\n");
   exit(0);
}
/* chương trình chính */
int main(int argc, unsigned char **argv)
{
   int res;
   /* mở tập tin thiết bị trước khi thao tác */
   if ((matrix_led_fd = open("/dev/matrix_led", O_RDWR)) < 0)
   {
   /* nếu quá trình mở tập tin thất bại thì in ra màn hình báo lỗi  cho người dùng biết
   và trả về mã lỗi */
         printf("Error whilst opening /dev/matrix_led device\n");
         return -1;
   }

   if (argc != 2)
   {
   /* khi người dùng sai cú pháp lệnh thì in ra hướng dẫn cú pháp  lệnh */
     print_usage();
   }
   else
/* nếu người dùng đúng cú pháp lệnh thì tiến hành chép dữ liệu  vào write_buf[0] và
truyền qua cho driver bằng hàm write()*/
   write_buf[0] = argv[1][0];
   write(matrix_led_fd, write_buf, 1);
```

```
printf ("User: sent %s to driver\n",argv[1][0]);
return 0;
    }
```

3. Biên dịch và thực thi chương trình:

Biên dịch chương trình, nạp vào Kit KM9260 và chạy chương trình.

III. Mở rộng dự án:

Vì module ma trận Led chỉ có 8 hàng và 8 cột nên dự án này chỉ có thể xuất mỗi lần 1 ký tự. Để xuất được nhiều ký tự hơn, cũng như có thể hiển thị được một chuỗi ký tự, một câu thì chúng ta phải dùng thêm hiệu ứng dịch chữ. Chúng ta có thể dịch chữ từ trái sang phải hoặc từ phải sang trái.

Chương trình sau đây có thể hiển thị một chuỗi ký tự bất kỳ nhập từ bàn phím với ký tự khoảng trắng là dấu "_" (vì cầu trúc lệnh của chương trình main trong User Application, nếu dùng khoảng trắng thì chương trình main sẽ hiểu là các tham số argv khác nhau, khuyết diểm này sẽ được khắc phục ở cuối dự án này), có thể sử dụng hiệu ứng dịch trái, dịch phải, đứng yên và có thể thay đổi tốc độ dịch chữ.

Cú pháp lệnh khi sử dụng như sau:

`<tên chương trình> <tên lệnh> <tham số>`

Trong đó:

`<tên chương trình>` là tên chương trình sau khi biên dịch

`<tên lệnh>` là các lệnh dùng để điều khiển ma trận Led. Có các lệnh như:

`update_content`: thay đổi nội dung muốn hiển thị, lênh này có tham số là chuỗi ký tự muốn hiển thị.

`update_speed`: thay đổi tốc độ dịch chữ, lệnh này có tham số là tốc độ dịch chữ thay đổi từ 1 đến 100. Ứng với giá trị 1 thì tốc độ dịch chữ là chậm nhất, 100 là tốc độ dịch chữ là nhanh nhất

`shift_left`: hiệu ứng dịch trái, lệnh này không có tham số.

`shift_right`: hiệu ứng dịch trái, lệnh này không có tham số.

`pause`: hiệu ứng đứng yên hình ảnh đang được hiển thị, lệnh này không có tham số.

`<tham số>` là tham số của lệnh.

❖ **Chương trình:**

- ***Driver***: Tên Matrix_Led_Shift_Display_dev.c

/ khai báo các thư viện cần thiết cho chương trình*/*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <mach/at91_tc.h>
#include <asm/gpio.h>
#include <asm/atomic.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
#include <asm/uaccess.h>
#include <linux/clk.h>
#include <linux/irq.h>
#include <linux/time.h>
#include <linux/jiffies.h>
#include <linux/sched.h>
#include <linux/delay.h>
```

/ khai báo tên thiết bị và tên driver */*

```
#define DRVNAME      "matrix_led_dev"
#define DEVNAME      "matrix_led"
```

/ khai báo các chân của vi xử lý dùng trong dự án này */*

```
#define DATA_C          AT91_PIN_PB8
#define CLOCK_C         AT91_PIN_PB10
#defineLATCH_C          AT91_PIN_PA23
#define OE_C            AT91_PIN_PB16
#define DATA_R          AT91_PIN_PA24
#define CLOCK_R         AT91_PIN_PB11
#define LATCH_R         AT91_PIN_PB9
#define OE_R            AT91_PIN_PB7
```

*/*lệnh set và clear các chân đã khai báo phần trên */*

```
#define SET_DATA_C()    gpio_set_value(DATA_C,1)
```

```
#define SET_CLOCK_C()    gpio_set_value(CLOCK_C,1)
#define SET_LATCH_C()    gpio_set_value(LATCH_C,1)
#define SET_OE_C()       gpio_set_value(OE_C,1)
#define SET_DATA_R()     gpio_set_value(DATA_R,1)
#define SET_CLOCK_R()    gpio_set_value(CLOCK_R,1)
#define SET_LATCH_R()    gpio_set_value(LATCH_R,1)
#define SET_OE_R()       gpio_set_value(OE_R,1)
#define CLEAR_DATA_C()   gpio_set_value(DATA_C,0)
#define CLEAR_CLOCK_C()  gpio_set_value(CLOCK_C,0)
#define CLEAR_LATCH_C()  gpio_set_value(LATCH_C,0)
#define CLEAR_OE_C()     gpio_set_value(OE_C,0)
#define CLEAR_DATA_R()   gpio_set_value(DATA_R,0)
#define CLEAR_CLOCK_R()  gpio_set_value(CLOCK_R,0)
#define CLEAR_LATCH_R()  gpio_set_value(LATCH_R,0)
#define CLEAR_OE_R()     gpio_set_value(OE_R,0)
```

*/*định nghĩa các lệnh ioctl dùng trong chương trình */*

```
#define MATRIX_LED_DEV_MAGIC  'B'
#define SHIFT_LEFT _IOWR(MATRIX_LED_DEV_MAGIC, 1,unsigned long)
#define SHIFT_RIGHT _IOWR(MATRIX_LED_DEV_MAGIC, 2,unsigned long)
#define UPDATE_SPEED _IOWR(MATRIX_LED_DEV_MAGIC, 3,unsigned long)
#define PAUSE    _IOWR(MATRIX_LED_DEV_MAGIC, 4,unsigned long)
```

/ Khai báo các biến cần thiết trong chương trình */*

```
static atomic_t matrix_led_open_cnt = ATOMIC_INIT(1);
int letter,i=0,cycle=0,slip=0,shift=0,shift_speed = 50,pause=0;
unsigned char DataDisplay[100];
unsigned char MatrixCode[5];
unsigned char ColumnCode[] ={0x80, 0x40, 0x20, 0x10, 0x08, 0x04,
0x02, 0x01};
unsigned char font[] = {
0xFF,0xFF,0xFF,0xFF,0xFF,//SPACE    0
0xFF,0xFF,0xA0,0xFF,0xFF,//!    1
0xFF,0xFF,0xF8,0xF4,0xFF,//'    2
0xEB,0x80,0xEB,0x80,0xEB,//#    3
0xDB,0xD5,0x80,0xD5,0xED,//$    4
```

```
0xD8,0xEA,0x94,0xAB,0x8D,//%    5
0xC9,0xB6,0xA9,0xDF,0xAF,//&    6
0xFF,0xFF,0xF8,0xF4,0xFF,//'    7
0xFF,0xE3,0xDD,0xBE,0xFF,//(    8
0xFF,0xBE,0xDD,0xE3,0xFF,//)    9
0xD5,0xE3,0x80,0xE3,0xD5,//*    10
0xF7,0xF7,0xC1,0xF7,0xF7,//+    11
0xFF,0xA7,0xC7,0xFF,0xFF,//,    12
0xF7,0xF7,0xF7,0xF7,0xF7,//-    13
0xFF,0x9F,0x9F,0xFF,0xFF,//x    14
0xFF,0xC9,0xC9,0xFF,0xFF,///    15
0xC1,0xAE,0xB6,0xBA,0xC1,//0    16
0xFF,0xBD,0x80,0xBF,0xFF,//1    17
0x8D,0xB6,0xB6,0xB6,0xB9,//2    18
0xDD,0xBE,0xB6,0xB6,0xC9,//3    19
0xE7,0xEB,0xED,0x80,0xEF,//4    20
0xD8,0xBA,0xBA,0xBA,0xC6,//5    21
0xC3,0xB5,0xB6,0xB6,0xCF,//6    22
0xFE,0x8E,0xF6,0xFA,0xFC,//7    23
0xC9,0xB6,0xB6,0xB6,0xC9,//8    24
0xF9,0xB6,0xB6,0xD6,0xE1,//9    25
0xFF,0xC9,0xC9,0xFF,0xFF,//:    26
0xFF,0xA4,0xC4,0xFF,0xFF,////   27
0xF7,0xEB,0xDD,0xBE,0xFF,//<    28
0xEB,0xEB,0xEB,0xEB,0xEB,//=    29
0xFF,0xBE,0xDD,0xEB,0xF7,//>    30
0xFD,0xFE,0xAE,0xF6,0xF9,//?    31
0xCD,0xB6,0x8E,0xBE,0xC1,//@    32
0x83,0xF5,0xF6,0xF5,0x83,//A    33
0xBE,0x80,0xB6,0xB6,0xC9,//B    34
0xC1,0xBE,0xBE,0xBE,0xDD,//C    35
0xBE,0x80,0xBE,0xBE,0xC1,//D    36
0x80,0xB6,0xB6,0xB6,0xBE,//E    37
0x80,0xF6,0xF6,0xFE,0xFE,//F    38
0xC1,0xBE,0xB6,0xB6,0xC5,//G    39
```

```
0x80,0xF7,0xF7,0xF7,0x80,//H  40
0xFF,0xBE,0x80,0xBE,0xFF,//I  41
0xDF,0xBF,0xBE,0xC0,0xFE,//J  42
0x80,0xF7,0xEB,0xDD,0xBE,//K    43
0x80,0xBF,0xBF,0xBF,0xFF,//L    44
0x80,0xFD,0xF3,0xFD,0x80,//M    45
0x80,0xFD,0xFB,0xF7,0x80,//N    46
0xC1,0xBE,0xBE,0xBE,0xC1,//O    47
0x80,0xF6,0xF6,0xF6,0xF9,//P    48
0xC1,0xBE,0xAE,0xDE,0xA1,//Q    49
0x80,0xF6,0xE6,0xD6,0xB9,//R    50
0xD9,0xB6,0xB6,0xB6,0xCD,//S    51
0xFE,0xFE,0x80,0xFE,0xFE,//T    52
0xC0,0xBF,0xBF,0xBF,0xC0,//U    53
0xE0,0xDF,0xBF,0xDF,0xE0,//V    54
0xC0,0xBF,0xCF,0xBF,0xC0,//W    55
0x9C,0xEB,0xF7,0xEB,0x9C,//X    56
0xFC,0xFB,0x87,0xFB,0xFC,//Y    57
0x9E,0xAE,0xB6,0xBA,0xBC,//Z    58
0xFF,0x80,0xBE,0xBE,0xFF,//[    59
0xFD,0xFB,0xF7,0xEF,0xDF,//\    60
0xFF,0xBE,0xBE,0x80,0xFF,//]    61
0xFB,0xFD,0xFE,0xFD,0xFB,//^    62
0x7F,0x7F,0x7F,0x7F,0x7F,//_    63
0xFF,0xFF,0xF8,0xF4,0xFF,//'    64
0xDF,0xAB,0xAB,0xAB,0xC7,//a    65
0x80,0xC7,0xBB,0xBB,0xC7,//b  66
0xFF,0xC7,0xBB,0xBB,0xBB,//c  67
0xC7,0xBB,0xBB,0xC7,0x80,//d  68
0xC7,0xAB,0xAB,0xAB,0xF7,//e    69
0xF7,0x81,0xF6,0xF6,0xFD,//f  70
0xF7,0xAB,0xAB,0xAB,0xC3,//g    71
0x80,0xF7,0xFB,0xFB,0x87,//h    72
0xFF,0xBB,0x82,0xBF,0xFF,//i    73
0xDF,0xBF,0xBB,0xC2,0xFF,//j    74
```

```
0xFF,0x80,0xEF,0xD7,0xBB,//k    75
0xFF,0xBE,0x80,0xBF,0xFF,//l    76
0x83,0xFB,0x87,0xFB,0x87,//m    77
0x83,0xF7,0xFB,0xFB,0x87,//n    78
0xC7,0xBB,0xBB,0xBB,0xC7,//o    79
0x83,0xEB,0xEB,0xEB,0xF7,//p    80
0xF7,0xEB,0xEB,0xEB,0x83,//q    81
0x83,0xF7,0xFB,0xFB,0xF7,//r    82
0xB7,0xAB,0xAB,0xAB,0xDB,//s    83
0xFF,0xFB,0xC0,0xBB,0xBB,//t    84
0xC3,0xBF,0xBF,0xDF,0x83,//u    85
0xE3,0xDF,0xBF,0xDF,0xE3,//v    86
0xC3,0xBF,0xCF,0xBF,0xC3,//w    87
0xBB,0xD7,0xEF,0xD7,0xBB,//x    88
0xF3,0xAF,0xAF,0xAF,0xC3,//y    89
0xBB,0x9B,0xAB,0xB3,0xBB,//z    90
0xFB,0xE1,0xE0,0xE1,0xFB,//^    91
0xE3,0xE3,0xC1,0xE3,0xF7,//->   93
0xF7,0xE3,0xC1,0xE3,0xE3,//<-   93
0xEF,0xC3,0x83,0xC3,0xEF,//     94
0xFF,0xFF,0xFF,0xFF,0xFF//BLANK CHAR 95
};
void __iomem *at91tc0_base;
struct clk *at91tc0_clk;
void row_write_data(unsigned char data) {
   int j;
   for (j=0;j<8;j++){
     (data&(128>>j))? SET_DATA_R():CLEAR_DATA_R();
     SET_CLOCK_R();
     CLEAR_CLOCK_R();
   }
   SET_LATCH_R();
   CLEAR_LATCH_R();
}
```

```
void column_write_data(unsigned char data) {
   int j;
   for (j=0;j<8;j++){
     (data&(1<<j))? SET_DATA_C():CLEAR_DATA_C();
     SET_CLOCK_C();
     CLEAR_CLOCK_C();
   }
   SET_LATCH_C();
   CLEAR_LATCH_C();
}
static irqreturn_t at91tc0_isr(int irq, void *dev_id) {
   int status;
   // Read TC0 status register to reset RC compare status.
   status = ioread32(at91tc0_base + AT91_TC_SR);
   if (shift!=0) {
   if (cycle < shift_speed) {
    SET_OE_C();
    SET_OE_R();
    row_write_data(DataDisplay[slip+i]);
    column_write_data(ColumnCode[i]);
    CLEAR_OE_C();
    CLEAR_OE_R();
    i++;
    if (i==8) {
         i = 0;
         cycle++;
    }
   } else {
    cycle = 0;
    if (pause==0) {
     if (shift==1) {
      slip++;
      if (slip == letter*5+8) {
       slip = 0;
```

```
        }
       } else {
        slip--;
        if (slip == 0) {
          slip = letter*5+8;
        }
       }
      }
     }

    }
     return IRQ_HANDLED;
}

void asscii_to_matrix_led (unsigned char  data) {
   MatrixCode[0] = font[data * 5 +0];
   MatrixCode[1] = font[data * 5 +1];
   MatrixCode[2] = font[data * 5 +2];
   MatrixCode[3] = font[data * 5 +3];
   MatrixCode[4] = font[data * 5 +4];
}

static int matrix_led_ioctl (struct inode *inode, struct file *file,
unsigned int cmd, unsigned long arg[]){
 int retval;
 switch (cmd) {
  case SHIFT_LEFT:
    shift = 1;
   pause = 0;
  break;
  case SHIFT_RIGHT:
    shift = 2;
   pause = 0;
  break;
  case UPDATE_SPEED: shift_speed = 101 - arg[0];
```

```
  break;
  case PAUSE: pause =1;
  break;
  default:
  printk ("Driver: Don't have this operation \n");
  retval = -EINVAL;
  break;
  }
  return retval;
}

static ssize_t matrix_led_write (struct file *filp, unsigned char
__iomem buf[], size_t bufsize, loff_t *f_pos) {
   unsigned char write_buf[100] ;
   int write_size = 0,letter_byte;
   if (copy_from_user (write_buf, buf, bufsize) != 0) {
        return -EFAULT;
   } else {
         write_size = bufsize;
         printk("write size: %d\n",write_size);
         DataDisplay[0] = 0x0;
         DataDisplay[1] = 0x0;
         DataDisplay[2] = 0x0;
         DataDisplay[3] = 0x0;
         DataDisplay[4] = 0x0;
         DataDisplay[5] = 0x0;
         DataDisplay[6] = 0x0;
         DataDisplay[7] = 0x0;

         for (letter=0;letter < write_size;letter++){
           if (write_buf[letter] == 95) {
             write_buf[letter] = 32;
           }
          asscii_to_matrix_led(write_buf[letter]-32);
           for(letter_byte=0;letter_byte < 5; letter_byte) {
```

```
            DataDisplay[letter*5+8+letter_byte] =
            ~MatrixCode[letter_byte];
          }
        }
         DataDisplay[letter*5+8+0] = 0x0;
         DataDisplay[letter*5+8+1] = 0x0;
         DataDisplay[letter*5+8+2] = 0x0;
         DataDisplay[letter*5+8+3] = 0x0;
         DataDisplay[letter*5+8+4] = 0x0;
         DataDisplay[letter*5+8+5] = 0x0;
         DataDisplay[letter*5+8+6] = 0x0;
         DataDisplay[letter*5+8+7] = 0x0;
         slip = 0;
         cycle = 0;
         i = 0;
   }
   return write_size;
}

static int
matrix_led_open(struct inode *inode, struct file *file) {
   int result = 0;
   unsigned int dev_minor = MINOR(inode->i_rdev);
   if (!atomic_dec_and_test(&matrix_led_open_cnt)) {
      atomic_inc(&matrix_led_open_cnt);
      printk(KERN_ERR DRVNAME ": Device with minor ID %d already in
      use\n", dev_minor);
      result = -EBUSY;
      goto out;
   }
out:
   return result;
}

static int
```

```
matrix_led_close(struct inode * inode, struct file * file) {
   smp_mb__before_atomic_inc();
   atomic_inc(&matrix_led_open_cnt);
   return 0;
}

struct file_operations matrix_led_fops = {
   .write    = matrix_led_write,
   .ioctl   = matrix_led_ioctl,
   .open    = matrix_led_open,
   .release = matrix_led_close,
};

static struct miscdevice matrix_led_dev = {
        .minor         = MISC_DYNAMIC_MINOR,
        .name         = "matrix_led",
        .fops         = &matrix_led_fops,
};

static int __init
matrix_led_mod_init(void) {
   int ret=0;
   gpio_request (DATA_C, NULL);
   gpio_request (CLOCK_C, NULL);
   gpio_request (LATCH_C, NULL);
   gpio_request (OE_C, NULL);
   gpio_request (DATA_R, NULL);
   gpio_request (CLOCK_R, NULL);
   gpio_request (LATCH_R, NULL);
   gpio_request (OE_R, NULL);

   at91_set_GPIO_periph (DATA_C, 1);
   at91_set_GPIO_periph (CLOCK_C, 1);
   at91_set_GPIO_periph (LATCH_C, 1);
   at91_set_GPIO_periph (OE_C, 1);
```

```
at91_set_GPIO_periph (DATA_R, 1);
at91_set_GPIO_periph (CLOCK_R, 1);
at91_set_GPIO_periph (LATCH_R, 1);
at91_set_GPIO_periph (OE_R, 1);


gpio_direction_output(DATA_C, 0);
gpio_direction_output(CLOCK_C, 0);
gpio_direction_output(LATCH_C, 0);
gpio_direction_output(OE_C, 0);
gpio_direction_output(DATA_R, 0);
gpio_direction_output(CLOCK_R, 0);
gpio_direction_output(LATCH_R, 0);
gpio_direction_output(OE_R, 0);

at91tc0_clk = clk_get(NULL,//Device pointer - not required.
                      "tc0_clk"); // Clock name.
 clk_enable(at91tc0_clk);
at91tc0_base = ioremap_nocache(AT91SAM9260_BASE_TC0,
64);
if (at91tc0_base == NULL) {
     printk(KERN_INFO "TC0 memory mapping failed\n");
     ret = -EACCES;
     goto exit_5;
  }
iowrite32(50000, (at91tc0_base + AT91_TC_RC));
iowrite32((AT91_TC_WAVE | AT91_TC_WAVESEL_UP_AUTO), (at91tc0_base
+ AT91_TC_CMR));
iowrite32(AT91_TC_CPCS, (at91tc0_base + AT91_TC_IER));
iowrite32((AT91_TC_SWTRG | AT91_TC_CLKEN), (at91tc0_base +
AT91_TC_CCR));

//Install interrupt for TC0.
ret = request_irq(    AT91SAM9260_ID_TC0,
```

```
                        at91tc0_isr, 0,
                        "matrix_led_irq",NULL);
   if (ret != 0) {
         printk(KERN_INFO "matrix_led_irq: Timer interrupt request
         failed\n");
         ret = -EBUSY;
         goto exit_6;
   }

   misc_register(&matrix_led_dev);
   printk(KERN_INFO "matrix_led: Loaded module\n");
   return ret;
   exit_6:
   iounmap(at91tc0_base);
   exit_5:
   clk_disable(at91tc0_clk);
   return ret;
}

static void __exit
matrix_led_mod_exit(void){
   iounmap(at91tc0_base);
   clk_disable(at91tc0_clk);
   // Free TC0 IRQ.
   free_irq(  AT91SAM9260_ID_TC0, // Interrupt number
              NULL); // Private data for shared interrupts
   misc_deregister(&matrix_led_dev);
   printk(KERN_INFO "matrix_led: Unloaded module\n");
}
```

```
module_init (matrix_led_mod_init);
module_exit (matrix_led_mod_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("TranCamNhan");
MODULE_DESCRIPTION("Character device for for generic gpio api");
```

- ***User Application:*** Tên Matrix_Led_Shift_Display_app.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
#include <pthread.h>
#define MATRIX_LED_DEV_MAGIC  'B'
#define SHIFT_LEFT    _IOWR(MATRIX_LED_DEV_MAGIC, 1,unsigned long)
#define SHIFT_RIGHT   _IOWR(MATRIX_LED_DEV_MAGIC, 2,unsigned long)
#define UPDATE_SPEED  _IOWR(MATRIX_LED_DEV_MAGIC, 3,unsigned long)
#define PAUSE    _IOWR(MATRIX_LED_DEV_MAGIC, 4,unsigned long)
int matrix_led_fd;
unsigned long ioctl_buf[3]={0,0,0};
unsigned char write_buf[100];
void
print_usage(){
   printf("matrixled_app update_content <sentence which you want to
   display on MatrixLed, Space = '_'>\n");
   printf("matrixled_app  update_speed  <speed from 1 to 100>\n");
   printf("matrixled_app  shift_left\n");
   printf("matrixled_app  shift_right\n");
   printf("matrixled_app  pause\n");
 exit(0);
}
```

```
int
main(int argc, unsigned char **argv) {
   int res,i;
   if ((matrix_led_fd = open("/dev/matrix_led", O_RDWR)) < 0) {
        printf("Error whilst opening /dev/matrix_led device\n");
        return -1;
   }

   if (argc == 2) {
     if (!strcmp (argv[1],"pause")){
      ioctl(matrix_led_fd,PAUSE,ioctl_buf);
     } else if (!strcmp(argv[1],"shift_left")){
      ioctl(matrix_led_fd,SHIFT_LEFT,ioctl_buf);
     } else if (!strcmp(argv[1],"shift_right")){
      ioctl(matrix_led_fd,SHIFT_RIGHT,ioctl_buf);
     }else print_usage();
     } else if (argc == 3) {
     if (!strcmp(argv[1],"update_speed")){
        ioctl_buf[0] = atoi(argv[2]);
        ioctl(matrix_led_fd,UPDATE_SPEED,ioctl_buf);
     }else if (!strcmp(argv[1],"update_content")){
       for (i=0;i<strlen(argv[2]);i++){
        write_buf[i] = argv[2][i];
       }
       write(matrix_led_fd, write_buf, strlen(argv[2]));
       printf("Update Content Complete: <%s> \n",argv[2]);
     }else print_usage();
   }else print_usage();
   return 0;
}
```

IV. Kết luận và bài tập:

1. Kết luận:

Trong dự án này chúng ta đã sử dụng phương pháp quét để hiển thị một ký tự bất kỳ, một chuỗi ký tự bất kỳ trên ma trận Led 8X8. Chương trình của chúng ta có 2 phần là Driver và User Application.

Như đã nói trong phần trên, trong chương trình này khi chúng ta muốn hiển thị khoảng trắng trên ma trận Led thì chúng ta phải nhập dữ liệu là dấu « _ » nên sẽ gây ra sự khó chịu khi sử dụng đối với người dùng chương trình và chúng ta không thể hiển thị dấu « _ » trên ma trận Led được. Để khắc phục vấn đề này, thì trong phần lệnh update_content thay vì chúng ta nhập dữ liệu vào tham số argv[2] của lệnh main thì chúng ta nạp dữ liệu này vào trong một biến kiểu chuỗi bằng lệnh gets(). Sau đó truyền biến này sang cho Driver, lúc đó chúng ta có thể khắc phục được khuyết điểm của chương trình trên.

Ngoài ra chương trình trên chỉ hiển thị ma trận Led một màu. Vì vậy chúng ta có thể mở rộng ứng dụng bằng cách viết thêm Driver hiển thị hai màu dựa vào Driver trên.

2. Bài tập :

1. Viết chương trình (Driver và User Application) hiển thị dữ liệu trên ma trận Led một màu có các hiệu ứng dịch trái, dịch phải, dứng yên, thay đổi tốc độ dịch và nhập dữ liệu muốn hiển thị bằng lệnh gets().
2. Viết chương trình (Driver và User Application) hiển thị dữ liệu trên ma trận Led hai màu có các hiệu ứng dịch trái, dịch phải, dứng yên, thay đổi tốc độ dịch, chọn được màu chữ hiển thị và nhập dữ liệu muốn hiển thị bằng lệnh gets().

BÀI 7

GIAO TIẾP ĐIỀU KHIỂN ADC0809

I. Phác thảo dự án:

ADC là một trong những kỹ thuật quan trọng của một hệ thống điều khiển. Kỹ thuật này chuyển tín hiệu tương tự sang tín hiệu số để vi xử lý có thể xử lý được tín hiệu và điều khiển hệ thống.

Trong AT91SAM9260 có tích hợp sẵn một bộ ADC 4 kênh 10 bit, nên chúng ta có thể sử dụng bộ ADC này cho các ứng dụng của mình. Tuy nhiên trong dự án này chúng ta sẽ điều khiển và đọc dữ liệu ADC0809 với mục đích tập viết chương trình Driver vừa điều khiển, vừa nhận giá dữ liệu từ thiết bi bên ngoài. Đồng thời chúng ta cũng sẽ ôn lại cách điều khiển và đọc dữ liệu ADC0809.

a. Yêu cầu dự án:

Viết Driver và User application điều khiển, đọc dữ liệu từ 8 kênh của ADC0809 và hiển thị lên màn hình Terminal giá trị đọc được.

Cú pháp lệnh như sau:

```
./<tên chương trình> <kênh muốn đọc>
```

Trong đó:

`<tên chươn trình>` là tên chương trình sau khi biên dịch

`<kênh muốn đọc>` là số kênh ADC muốn đọc giá trị digital, có giá trị từ 0 đến 7.

b. Phân công nhiệm vụ:

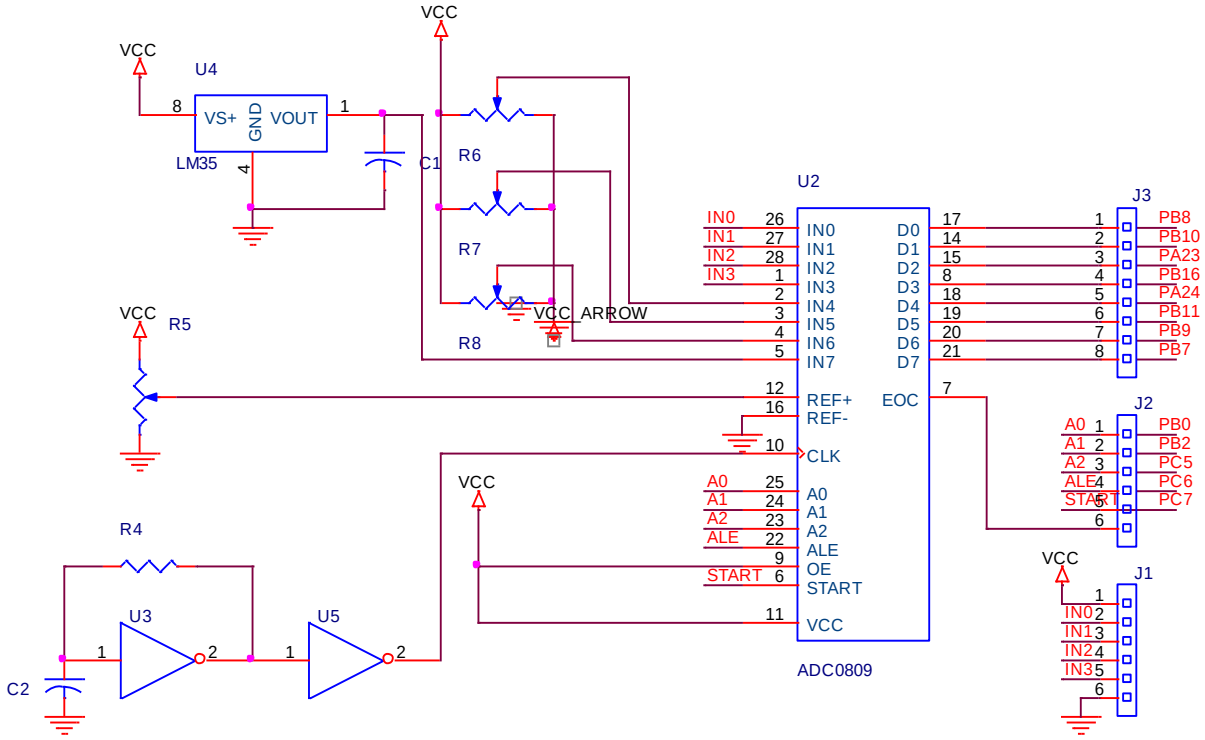
- ***Driver:*** tên chương trình là `adc0809_dev.c`

- Driver sẽ có nhiệm vụ điều khiển ADC0809 chuyển đổi tín hiệu analog sang số tại kênh mà User application yêu cầu và lưu vào một mảng có 10 phần tử, mỗi phần tử sẽ chứa giá trị digital của một lần chuyển đổi, lần chuyển đổi thứ 11 sẽ ghi đè lên giá trị của lần chuyển đổi đầu tiên tại phần tử thứ nhất của mảng. Cứ sau 1 ms thì Driver điều khiển ADC0809 chuyển đổi một lần.

- Khi User Application yêu cầu lệnh read() thì Driver sẽ tính trung bình cộng giá trị của 10 phần tử của mảng chứa giá trị chuyển đổi và gửi giá trị trung bình cộng này sang cho User Application.
  - ***Application:*** tên chương trình là `adc0809_app.c`
- User Application sẽ nhận số kênh muốn chuyển đổi từ người dùng và truyền số kênh này sang cho Driver, sau đó yêu cầu Driver gửi giá trị chuyển đổi đã tính toán. User xuất ra màn hình terminal giá trị nhận được từ Driver.

II. Thực hiện:

a. Kết nối phần cứng :



Hình 4-19- Sơ đồ kết nối ADC0809.

b. Chương trình Driver:

*/*Khai báo các thư viện cần thiết cho chương trình*/*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <mach/at91_tc.h>
#include <asm/gpio.h>
#include <asm/atomic.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
#include <asm/uaccess.h>
#include <linux/clk.h>
#include <linux/irq.h>
#include <linux/time.h>
#include <linux/jiffies.h>
#include <linux/sched.h>
#include <linux/delay.h>
#include <linux/timer.h>
```

*/*Khai bao tên Driver và tên thiết bị*/*

```
#define DRVNAME      "adc0809_dev"
#define DEVNAME      "adc0809"
```

*/*Định nghĩa các chân cần dùng trong chương trình*/*

```
#define D0              AT91_PIN_PB8
#define D1              AT91_PIN_PB10
#define D2              AT91_PIN_PA23
#define D3              AT91_PIN_PB16
#define D4              AT91_PIN_PA24
#define D5              AT91_PIN_PB11
#define D6              AT91_PIN_PB9
#define D7              AT91_PIN_PB7
#define A0              AT91_PIN_PB0
#define A1              AT91_PIN_PB2
#define A2              AT91_PIN_PC5
```

```
#define ALE             AT91_PIN_PC6
#define START           AT91_PIN_PC7
```

*/*Định nghĩa các lệnh set và clear cho các chân điều khiển thiết bị*/*

```
#define SET_A0()                    gpio_set_value(A0,1)
#define SET_A1()                    gpio_set_value(A1,1)
#define SET_A2()                    gpio_set_value(A2,1)
#define SET_ALE()                   gpio_set_value(ALE,1)
#define SET_START()                 gpio_set_value(START,1)

#define CLEAR_A0()                  gpio_set_value(A0,0)
#define CLEAR_A1()                  gpio_set_value(A1,0)
#define CLEAR_A2()                  gpio_set_value(A2,0)
#define CLEAR_ALE()                 gpio_set_value(ALE,0)
#define CLEAR_START()               gpio_set_value(START,0)
```

*/*Khai bao các biến cần dùng trong chương trình*/*

```
static atomic_t ADC0809_open_cnt = ATOMIC_INIT(1);
unsigned char Data_ADC[10];
unsigned char chanel =0;
int i=0;
struct timer_list my_timer;
```

*/*Chương trình điều khiển ADC chọn kênh cần chuyển đổi*/*

```
void choose_chanel(unsigned char data)
{
   (data&(1<<0))? SET_A0():CLEAR_A0();
   (data&(1<<1))? SET_A1():CLEAR_A1();
   (data&(1<<2))? SET_A2():CLEAR_A2();
}
```

*/*Chương trình chuyển một số nhị phân sang số thập phân*/*

```
unsigned char binary_to_decimal (unsigned char bit7, unsigned char
bit6, unsigned char bit5, unsigned char bit4, unsigned char bit3,
unsigned char bit2, unsigned char bit1, unsigned char bit0)
{
  unsigned char decimal_number;
```

```
  decimal_number = 1*bit0 + 2*bit1 + 4*bit2 + 8*bit3 + 16*bit4 +
32*bit5 + 64*bit6 + 128*bit7;
  return decimal_number;
}
```

/ Chương trình phục vụ ngắt timer ảo, cứ sau 1ms thì chương trình này sẽ được thực thi một lần, chương trình này có nhiệm vụ điều khiển ADC0809 chuyển đổi tín hiệu analog sang digital tại kênh mà User Application yêu cầu, sau đó đọc dữ liệu digital này và lưu vào mảng Data_ADC[]*/*

```
void
read_ADC0809_irq(unsigned long data)
{
  unsigned char d0,d1,d2,d3,d4,d5,d6,d7;
  choose_chanel(chanel);
  SET_ALE();
  SET_START();
  CLEAR_ALE();
  CLEAR_START();
  udelay(100);

  if(gpio_get_value(D0) ==0)
    d0=0;
  else
    d0 =1;
   if(gpio_get_value(D1) ==0)
    d1=0;
  else
    d1 =1;
   if(gpio_get_value(D2) ==0)
    d2=0;
  else
    d2 =1;
   if(gpio_get_value(D3) ==0)
    d3=0;
  else
```

```
      d3 =1;
    if(gpio_get_value(D4) ==0)
      d4=0;
    else
      d4 =1;
    if(gpio_get_value(D5) ==0)
      d5=0;
    else
      d5 =1;
    if(gpio_get_value(D6) ==0)
      d6=0;
    else
      d6 =1;
    if(gpio_get_value(D7) ==0)
      d7=0;
    else
      d7 =1;

    Data_ADC[i] = binary_to_decimal(d7,d6,d5,d4,d3,d2,d1,d0);
    i++;
    if(i==10)
    i=0;
    mod_timer (&my_timer, jiffies + 1);
}
```

*/*Chương trình write(), User Application sẽ truyền số kênh cần chuyển đổi cho Driver thông qua lệnh này*/*

```
static ssize_t ADC0809_write (struct file *filp, int __iomem buf[],
size_t bufsize, loff_t *f_pos)
{
   unsigned char driver_write_buf[1] ;
   int write_size = 0;
   if (copy_from_user (driver_write_buf, buf, bufsize) != 0)
         return -EFAULT;
   else
```

```
    {
    write_size = bufsize;
    chanel = driver_write_buf[0];
      }
    return write_size;
}
```

*/*Lệnh read(), Lệnh này sẽ gửi cho User Application giá trị chuyển đổi đã được tính trung bình cộng*/*

```
static ssize_t ADC0809_read (struct file *filp, unsigned char
__iomem buf[], size_t bufsize, loff_t *f_pos)
{
  unsigned char driver_read_buf[1] ={0};
  int sum =0 ;
  int j;
  for(j=0;j<10;j++)
  {
  sum = sum+Data_ADC[j];
  }
  driver_read_buf[0] = sum/10;
  if(copy_to_user(buf,driver_read_buf,bufsize) != 0)
  {
   printk("Can't read Driver \n");
   return -EFAULT;
  }
}
```

*/*Lệnh open()*/*

```
static int
ADC0809_open(struct inode *inode, struct file *file)
{
    int result = 0;
    unsigned int dev_minor = MINOR(inode->i_rdev);
    if (!atomic_dec_and_test(&ADC0809_open_cnt)) {
       atomic_inc(&ADC0809_open_cnt);
```

```
        printk(KERN_ERR DRVNAME ": Device with minor ID %d already in
use\n", dev_minor);
        result = -EBUSY;
        goto out;
    }
out:
    return result;
}
```

*/*Lệnh close()*/*

```
static int
ADC0809_close(struct inode * inode, struct file * file)
{
    smp_mb__before_atomic_inc();
    atomic_inc(&ADC0809_open_cnt);
    return 0;
}

struct file_operations ADC0809_fops = {
    .read    = ADC0809_read,
    .write   = ADC0809_write,
    .open    = ADC0809_open,
    .release = ADC0809_close,
};

static struct miscdevice ADC0809_dev = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name          = "ADC0809",
        .fops          = &ADC0809_fops,
};

static int __init
ADC0809_mod_init(void)
{
    int ret=0;
```

```
gpio_request (D0, NULL);
gpio_request (D1, NULL);
gpio_request (D2, NULL);
gpio_request (D3, NULL);
gpio_request (D4, NULL);
gpio_request (D5, NULL);
gpio_request (D6, NULL);
gpio_request (D7, NULL);

at91_set_GPIO_periph (D0, 1);
at91_set_GPIO_periph (D1, 1);
at91_set_GPIO_periph (D2, 1);
at91_set_GPIO_periph (D3, 1);
at91_set_GPIO_periph (D4, 1);
at91_set_GPIO_periph (D5, 1);
at91_set_GPIO_periph (D6, 1);
at91_set_GPIO_periph (D7, 1);


 gpio_direction_input(D0);
gpio_direction_input(D1);
gpio_direction_input(D2);
gpio_direction_input(D3);
gpio_direction_input(D4);
gpio_direction_input(D5);
gpio_direction_input(D6);
gpio_direction_input(D7);

gpio_request (A0, NULL);
gpio_request (A1, NULL);
gpio_request (A2, NULL);
gpio_request (ALE, NULL);
gpio_request (START, NULL);

at91_set_GPIO_periph (A0, 1);
```

```
    at91_set_GPIO_periph (A1, 1);
    at91_set_GPIO_periph (A2, 1);
    at91_set_GPIO_periph (ALE, 1);
    at91_set_GPIO_periph (START, 1);

    gpio_direction_output(A0, 0);
    gpio_direction_output(A1, 0);
    gpio_direction_output(A2, 0);
    gpio_direction_output(ALE, 0);
    gpio_direction_output(START, 0);

    init_timer (&my_timer);
    my_timer.expires = jiffies  + 1;
    my_timer.data = 0;
    my_timer.function = read_ADC0809_irq;
    add_timer (&my_timer);

    misc_register(&ADC0809_dev);
    printk(KERN_INFO "ADC0809: Loaded module\n");
    return ret;
}

static void __exit
ADC0809_mod_exit(void)
{
    del_timer_sync(&my_timer);
    misc_deregister(&ADC0809_dev);
    printk(KERN_INFO "ADC0809: Unloaded module\n");
}

module_init (ADC0809_mod_init);
module_exit (ADC0809_mod_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("TranCamNhan");
```

```
MODULE_DESCRIPTION("Character device for for generic gpio api");
```

c. Chương trình User Application:

*/*Khai báo các thư viện cần thiết cho chương trình*/*

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
#include <pthread.h>
#include <unistd.h>

int ADC0809_fd;
```

*/*Chương trình in ra màn hình cú pháp sử dụng lệnh*/*

```
int print_usage(void)
{
  printf("./ADC0809_app <chanel_number>\n");
  return -1;
}
```

*/*Chương trình chính*/*

```
int
main(int argc, char **argv)
{
   unsigned char user_read_buf[1];
   unsigned char user_write_buf[1];
   /*Mở thiết bị*/
   if ((ADC0809_fd = open("/dev/ADC0809", O_RDWR)) < 0)
   {
        printf("Error whilst opening /dev/ADC0809 device\n");
        return -1;
   }

   user_write_buf[0]= atoi(argv[1]);
```

```
    if (argc != 2)
      print_usage();
    else
      {
        if(user_write_buf[0] > 7)
          printf("You have to choose from chanel 0 to chanel 7\n");
        else
          {
            write(ADC0809_fd,user_write_buf,1);
            /*Thời gian để Driver chuyển đổi và tính toán xong*/
            usleep(50000);
            read(ADC0809_fd, user_read_buf,1);
            printf("adc chanel %d value: %d\n", user_write_buf[0],
user_read_buf[0]);
          }
      }
    return 0;
}
```

III. Mở rộng dự án:

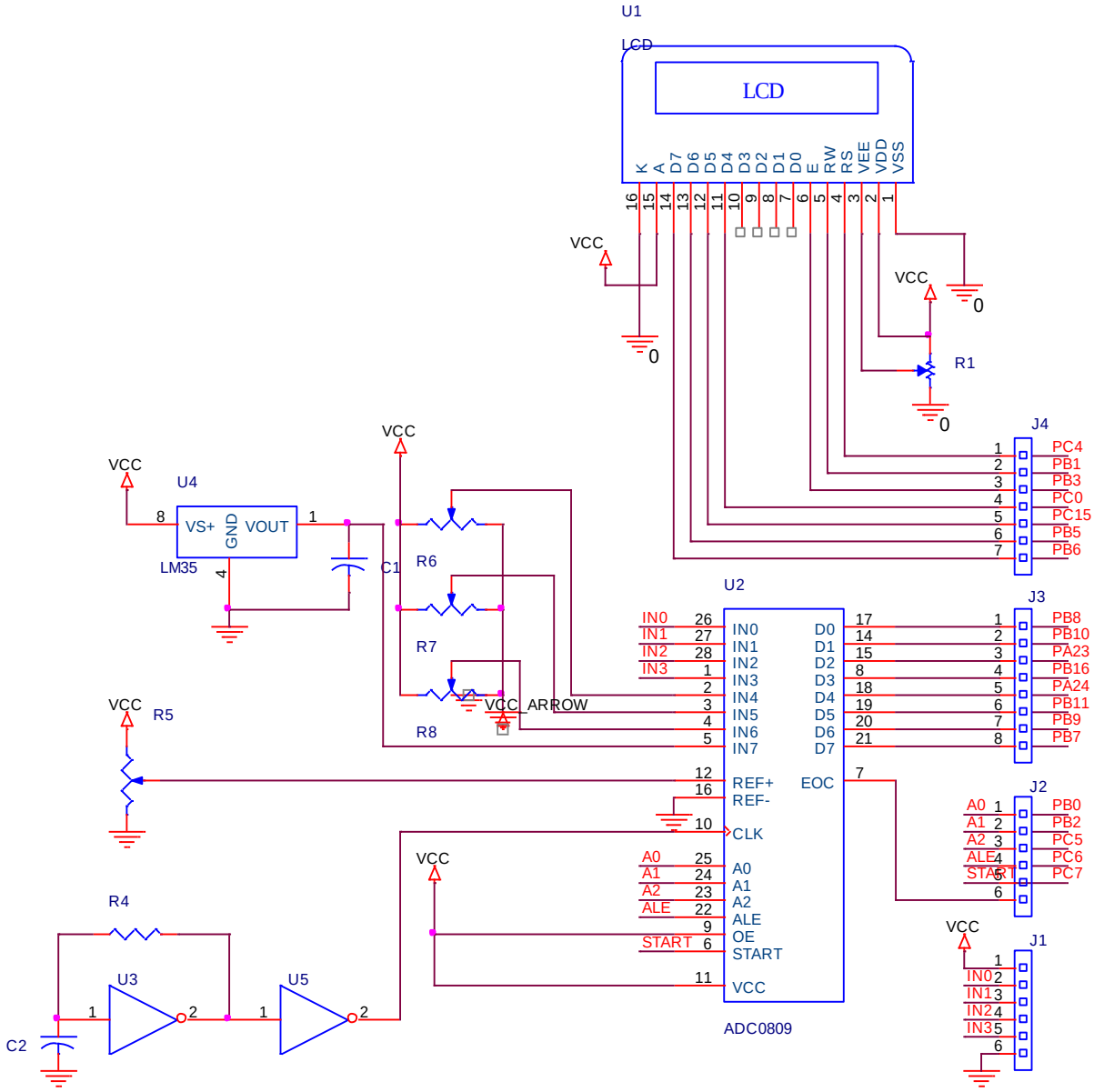
Cũng với yêu cầu điều khiển và đọc dữ liệu của ADC0809 tại 8 kênh, nhưng bây giờ không hiển thị dữ liệu đọc được lên màn hình terminal nữa mà hiển thị lên LCD 20X2.

Với dự án này thì chúng ta kết hợp thêm Driver của LCD. Như vậy trong dự án này chúng ta dùng 2 driver: một driver của ADC và một driver của LCD.

Vì chương trình User Application để ra lệnh cho Driver LCD khá phức tạp, nên trong dự án này chúng ta sẽ viết User Application điều khiển Driver LCD thành một tập tin riêng và User Application điều khiển Driver ADC thành một tập tin riêng. Tuy là viết thành hai tập tin nhưng thực chất chỉ có một chương trình User Application duy nhất. Chúng ta sẽ tập làm quen với cách viết User Application như thế này. Phương pháp viết chương trình như thế này sẽ chúng cho chúng ta quản lý được chương trình, tạo sự thuận lợi cho quá trình kiểm tra.

Trong hai tập tin User Application thì tập tin User Application điều khiển ADC là tập tin chính nên sẽ chứa hàm main(), còn tập tin User Application điều khiển LCD là tập tin phụ. Tập tin chính sẽ include tập tin phụ.

a. Kết nối phần cứng:



Hình 4-20- Sơ đồ kết nối ADC0809 và LCD.

b. ***Chương trình Driver LCD:*** có tên là `lcd_dev.c`

Chương trình này đã được trình bày trong bài module LCD nên trong phần này sẽ không trình bày lại. Nhưng trong Driver LCD này các chân điều khiển LCD lại trùng với các chân điều khiển ADC nên chúng ta phải sửa lại các định nghĩa chân điều khiển của chương trình như sau:

```
#define LCD_RW_PIN              AT91_PIN_PB1
#define LCD_EN_PIN              AT91_PIN_PB3
#define LCD_RS_PIN              AT91_PIN_PC4
#define LCD_D4_PIN              AT91_PIN_PC0
#define LCD_D5_PIN              AT91_PIN_PC15
#define LCD_D6_PIN              AT91_PIN_PB5
#define LCD_D7_PIN              AT91_PIN_PB6
```

c. Chương trình Driver ADC: có tên là `adc0809_dev.c`

Đã trình bày ở phần trên.

d. ***Chương trình User App chính:*** có tên là `ADC0809_display_lcd_app.c`

```
/*Khai báo các thư viện cần thiết cho chương trình*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/ioctl.h>
#include <pthread.h>
#include <unistd.h>
/* include User Application phụ, chú ý User Application chính và User Application phụ phải nằm trong một thư mục khi biên dịch chương trình */
#include "lcd_app.c"

int ADC0809_fd;
int lcd_fd;
/*Chương trình in ra màn hình hướng dẫn sử dụng lệnh*/
int print_usage(void)
```

```
{
  printf("./ADC0809_display_lcd_app <chanel_number>\n");
  return -1;
}

int
main(int argc, char **argv)
{
   unsigned char user_read_buf[1];
   unsigned char user_write_buf[1];
   /*Mở thiết bị ADC0809*/
   if ((ADC0809_fd = open("/dev/ADC0809", O_RDWR)) < 0)
   {
        printf("Error whilst opening /dev/ADC0809 device\n");
        return -1;
   }
   /*Mở thiết bị LCD*/
   if ((lcd_fd = open("/dev/lcd_dev", O_RDWR)) < 0)
   {
        printf("Error whilst opening /dev/lcd_dev device\n");
        return -1;
   }

   user_write_buf[0]= atoi(argv[1]);
   if (argc != 2)
     print_usage();
   else
     {
       if(user_write_buf[0] > 7)
         printf("You have to choose from chanel 0 to chanel 7\n");
       else
          {
           lcd_Clear();
           Display_Print_Data(lcd_fd,"Digital Data is:",0,0);
```

```
            while(1)
            {
             write(ADC0809_fd,user_write_buf,1);
             sleep(1);
             read(ADC0809_fd, user_read_buf,1);
             Display_Number(lcd_fd,user_read_buf[0],1,0);
             printf("adc chanel %d value: %d\n", user_write_buf[0],
             user_read_buf[0]);
             }
           }
      }
   return 0;
}
```

e. ***Chương trình User Application phụ:*** có tên là `lcd_app.c`

*/*Khai báo các thư viện cần thiết trong chương trình*/*

```
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <getopt.h>
#include <fcntl.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <linux/ioctl.h>
```

*/*Định nghĩa các lệnh ioctl*/*

```
#define IOC_LCDDEVICE_MAGIC   'B'
#define LCD_CONTROL_WRITE          _IO(IOC_LCDDEVICE_MAGIC, 15)
#define LCD_DATA_WRITE             _IO(IOC_LCDDEVICE_MAGIC, 16)
#define LCD_INIT                   _IO(IOC_LCDDEVICE_MAGIC, 17)
#define LCD_HOME                   _IO(IOC_LCDDEVICE_MAGIC, 18)
#define LCD_CLEAR                  _IO(IOC_LCDDEVICE_MAGIC, 19)
#define LCD_DISP_ON_C              _IO(IOC_LCDDEVICE_MAGIC, 21)
```

```
#define LCD_DISP_OFF_C              _IO(IOC_LCDDEVICE_MAGIC, 22)
#define LCD_CUR_MOV_LEFT_C          _IO(IOC_LCDDEVICE_MAGIC, 23)
#define LCD_CUR_MOV_RIGHT_C         _IO(IOC_LCDDEVICE_MAGIC, 24)
#define LCD_DIS_MOV_LEFT_C          _IO(IOC_LCDDEVICE_MAGIC, 25)
#define LCD_DIS_MOV_RIGHT_C         _IO(IOC_LCDDEVICE_MAGIC, 29)
#define LCD_DELAY_MSEC              _IO(IOC_LCDDEVICE_MAGIC, 30)
#define LCD_DELAY_USEC              _IO(IOC_LCDDEVICE_MAGIC, 31)
```

*/*Định nghĩa các thông số cho LCD*/*

```
#define LCD_LINE0_ADDR   0x00
#define LCD_LINE1_ADDR   0x40
#define LCD_DD_RAM_PTR   0x80

#define lcd_Control_Write() ioctl(lcd_fd, LCD_CONTROL_WRITE, data)
#define lcd_Data_Write()    ioctl(lcd_fd, LCD_DATA_WRITE, data)
#define lcd_Clear()         ioctl(lcd_fd, LCD_CLEAR)
```

/ Chương trình di chuyển con trỏ của LCD đến vị trí hàng x và cột y*/*

```
void lcd_Goto_XY(int lcd_fd,uint8_t x, uint8_t y)
{
   register uint8_t DDRAMAddr;
   switch(x)
   {
   case 0: DDRAMAddr = LCD_LINE0_ADDR+y; break;
   case 1: DDRAMAddr = LCD_LINE1_ADDR+y; break;
   default: DDRAMAddr = LCD_LINE0_ADDR+y;
   }
   ioctl(lcd_fd, LCD_CONTROL_WRITE, LCD_DD_RAM_PTR | DDRAMAddr);

}
```

*/*Chương trình hiển thị một chuỗi ký tự ra LCD với vị trí đầu tiên tại hàng x cột y*/*

```
void Display_Print_Data(int lcd_fd,char *string, uint8_t x, uint8_t
y)
{
   lcd_Goto_XY(lcd_fd,x,y);
```

```
   while (*string)  {
         ioctl(lcd_fd, LCD_DATA_WRITE, *string++);
   }
}
```

*/*Chương trình hiển thị một ký tự (mã asscii) ra LCD tại vị trí hàng x cột y*/*

```
void Display_Print_Char(int lcd_fd,int data, uint8_t x, uint8_t y) {
   lcd_Goto_XY(lcd_fd,x,y);
   lcd_Data_Write ();
}
```

*/*Chương trình hiển thị một số có giá trị từ 0 đến 999 ra LCD tại vị trí hàng h_pos và cột start_v_pos*/*

```
void Display_Number(int lcd_fd,int number, int h_pos, int
start_v_pos) {
 Display_Print_Char(lcd_fd,(number/100)+48,h_pos,start_v_pos);
  Display_Print_Char(lcd_fd,((number%100)/10)+48,h_pos,start_v_pos+1
);
  Display_Print_Char(lcd_fd,(number%10)+48,h_pos,start_v_pos+2);
}
```

IV.Kết luận và bài tập:

a. Kết luận:

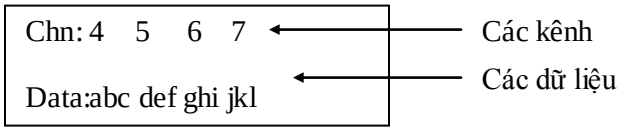
Vậy trong dự án này chúng ta đã điều khiển và đọc được ADC0809, hiển thị giá trị này lên màn hình terminal và lên LCD. Trong thực tế người ta thường ứng dụng dự án này để đo nhiệt độ bằng cách nối các kênh ngõ vào với IC LM35.

Một lần nữa chúng ta lại dùng hai Driver cho một ứng dụng, như vậy chúng ta có thể thấy được điểm mạnh của việc lập trình nhúng.

Ngoài ra trong phần này chúng ta đã làm quen với cách viết chương trình chia làm nhiều tập tin khác nhau.

b. Bài tập:

1. Ứng dụng Driver ADC và LCD trên viết chương trình đọc giá trị chuyển đổi của 4 kênh ADC là 4,5,6,7 và hiển thị lên LCD theo cấu trúc sau:



2. Viết Driver và Application đo nhiệt độ của một lò nung, khi nhiệt độ của lò nung giảm đến dưới 100^{o}C thì bật hệ thống nung để gia nhiệt, khi nhiệt độ đến 120^{o}C thì tắt hệ thống nung.

BÀI 8

GIAO TIẾP ĐIỀU KHIỂN
MODULE I2C

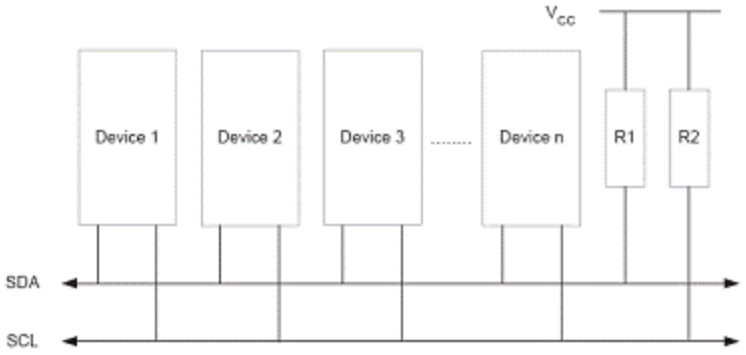
A- TỔNG QUAN VỀ I2C:

I. Giới thiệu I^2C:

I2C là chuẩn truyền thông nối tiếp do hãng điện tử Phiilip xây dựng vào năm 1990. I2C có khả năng truyền nối tiếp đa chủ, nghĩa là trong một bus có thể có nhiều hơn một thiết bị làm Master. Chuẩn I2C thường được tìm thấy trong những thiết bị điện tử như EEPROM 24CXX, realtime DS1307,... và một số thiết bị khác.

(Các kiến thức về I^2C được trình bài sau đây được sưu tầm từ trang web www.hocavr.com).

Chuẩn I2C được kết nối theo dạng sau:



Hình 4-21- Sơ đồ kết nối trong hệ thống giao tiếp theo chuẩn I2C.

Trong chuẩn I2C có hai đường dây dùng để truyền và nhận dữ liệu: SDA, và SCL. Trong đó SDA là đường dữ liệu truyền nhận. Dữ liệu truyền nhận là các bit 0 và 1 tương ứng với mức thấp và mức cao. Vị trí bit trong dự liệu được đồng bộ hòa bởi xung clock thông qua đường SCL. Như vậy đường SCL là đường cung cấp xung dồng bộ dữ liệu. Cả hai đường SCL và SDA điều có cấu hình cực góp hở (Open-collector) vì thế khi được lắp đặt trong hê thống cần phải có điện trỏ kéo lên Vcc để tạo mức thấp và mức cao trong quá trình truyền dữ liệu.

Các thiết bị trong truyền theo chuẩn I2C trong cùng một hệ thống được kết nối song song, mỗi thiết bị điều có dịa chỉ riêng dùng để phân biệt với nhau trong quá trình truyền

và nhận dữ liệu. Tại một thời điểm chỉ có hai thiết bị, một thiết bị đóng vai trò là slave và một thiết bị đóng vai trò là master, truyền và nhận dữ liệu. Các thiết bị khác trong cùng một bus bị vô hiệu hóa.

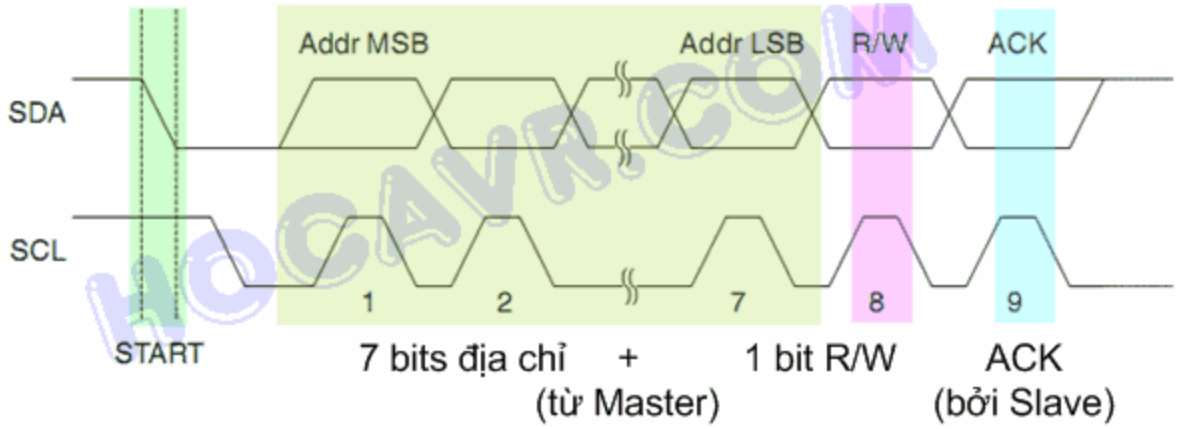
II. Các thuật ngữ và giao thức truyền trong I^2C:

Sau đây là các thuật ngữ thường được sử dụng trong chế độ truyền nối tiếp I2C, trong mỗi thuật ngữ chúng ta sẽ tìm hiểu về định nghĩa và vai trò trong giao thức. Trên cơ sở đó sẽ hiểu rõ cách thức hoạt động của chuẩn I2C:

- ***Master:*** Là thiết bị khởi động quá trình truyền nhận. Thiết bị này sẽ phát ra các bit dữ liệu trên đường SDA và xung đồng bộ trên đường SCL. (Trong giáo trình này Master là chip ARM9260).
- ***Slave:*** Là thiết bị phục vụ cho Master khi được yêu cầu. Mỗi một thiết bị có một địa chỉ riêng. Địa chỉ thiết bị được quy định như sau: Mỗi loại thiết bị khác nhau (chẳng hạn EEPROM, Realtime, hay ADC, ...) nhà sản xuất sẽ quy định một địa chỉ khác nhau, được gọi là địa chỉ thiết bị. Bên cạnh đó còn có địa chỉ phân biệt giữa các thiết bị cùng loại (Chẳng hạn nhiều EEPROM 24CXX được kết nối chung với nhau), địa chỉ này được người sử dụng thiết bị quy định.
- ***SDA-Serial Data:*** Là đường truyền dữ liệu nối tiếp. Đường này sẽ đảm nhận nhiệm vụ truyền thông tin về địa chỉ và dữ liệu theo thứ tự từng bit. Lưu ý trong chuẩn I2C thì bit MSB được truyền trước nhất, cuối cùng là LSB, cách truyền này ngược với chuẩn truyền nối tiếp UART.
- ***SCL-Serial clock:*** Là đường giữ nhịp cho truyền và nhận dữ liệu. Nhịp đồng bộ dữ liệu do thiết bị đóng vai trò là Master phát ra. Quy định của quá trình truyền dữ liệu này là: Tại thời diểm mức cao của SCL dữ liệu trên chân SDA không được thay đổi trạng thái. Dữ liệu trên chân SDA chỉ được quyền thay đổi trạng thái khi xung SCL ở mức thấp. Khi chân SCL mức cao, nếu SDA thay đổi trạng thái thì thiết bị I2C nhận ra là các bit Start hoặc Stop;
- ***START transmition:*** Là trạng thái bắt đầu quá trình truyền và nhận dữ liệu được quy định: trạng thái chân SCL mức cao và chân SDA chuyển trạng thái từ mức cao xuống mức thấp, cạnh xuống.

- ***STOP stransmition:*** Là trạng thái kết thúc quá trình truyền và nhận dữ liệu được quy định: trạng thái chân SCL mức cao và chân SDA chuyển trạng thái từ mức thấp lên mức cao;
- ***Address:*** Là địa chỉ thiết bị, thông thường được quy định có 7 bit;
- ***Data:*** Là dữ liệu truyền nhận, thông thường được quy định có 8 bit;
- ***Repeat Start-Bắt đầu lặp lại:*** Khoảng giữa hai trạng thái Start và Stop là khoảng bận của đường truyền, các Master khác không được tác động vào đường truyền trong khoảng này. Trường hợp sau khi kết thúc truyền nhận mà Master không gửi trạng thái Stop mà gởi thêm trạng thái Start để tiếp tục truyền nhận dữ liệu, quá trình này gọi là Repeat Start. Trường hợp này xuất hiện khi Master muốn lấy hoặc truyền dữ liệu liên tiếp từ các Slave.
- ***Address Packet Format-Định dạng gói địa chỉ:*** Trong mạng I2C tất cả các thiết bị đều có thể là Master hay Slave. Mỗi thiết bị có một địa chỉ cố định gọi là Device address. Khi một Master muốn giao tiếp với một Slave, trước hết nó sẽ tạo ra một trạng thái START tiếp theo sẽ gửi địa chỉ thiết bị của Slave cần giao tiếp trên đường truyền, thì thế xuất hiện thuật ngữ gói địa chỉ (Address packet). Gói địa chỉ trong I2C có định dạng 9 bits trong đó 7 bits đầu (gọi là SLA, được gửi liền sau START) chứa địa chỉ Slave (Một số trương hợp địa chỉ có 10 bits), một bit READ/WRITE và một bit ACK (Acknowledge) xác nhận thông tin địa chỉ. Địa chỉ có độ dài 7 bits nên số thiết bị tối đa có thể giao tiếp là 128 thiết bị. Nhưng có một số địa chỉ không được hiểu là địa chỉ thiết bị. Các địa chỉ đó là 1111xxx (tức là các địa chỉ lớn hơn hoặc bằng 120 không được dùng). Riêng địa chỉ 0 được dùng cho cuộc gọi chung. Bit READ/WRITE được truyền theo sau 7 bits địa chỉ là biết báo cho Slave biết là Master muốn đọc hay ghi dữ liệu vào Slave. Nếu bit này bằng 0 (mức thấp) thì quá trình ghi dữ liệu từ Master đến Slave được yêu cầu, nếu bit này bằng 1 thì Master muốn đọc dữ liệu từ Slave về. 8 bits trên (SLA+RD/WR) được Master phát ra sau khi phát START, nếu một Slave trên mạng nhận ra rằng địa chỉ mà Master yêu cầu trùng khớp với địa chỉ của mình, no sẽ đáp trả lại Master bằng cách đáp trả lại bắng tín hiệu xác nhân ACK bằng cách kéo chân SDA xuống thấp trong xung giữ nhịp thứ 9. Ngược lại nếu không có Slave đáp ứng lại, thì chân SDA vẫn giữ ở mức cao trong xung

giữ nhịp thứ 9 thì gọi lả tín hiệu không xác nhận NOT ACK, lúc này Master cần phải có những ứng xử phù hợp tùy theo mỗi trường hợp cụ thể, ví dụ Master có thể gởi trạng thái STOP và sau đó phát lại địa chỉ Slave khác... Như vậy, trong 9 bit của gói địa chỉ thì chỉ có 8 bits được gửi từ Master, bit còn lại là do Slave. Ví dụ Master muốn yêu cầu "đọc" dữ liệu từ Slave có địa chỉ 43, nó cần phát ra cho Slave một byte như sau: (43<<1)+1, trong đó (43<<1) là địch số 43 về bên trái 1 vị trí vì 7 bits địa chỉ nằm ở các vị trí cao trong gói địa chỉ, sau đó cộng giá trị này với 1 tức là quá trình đọc dư liệu được yêu cầu. Hình bên dưới mô tả quá trình phát gói địa chỉ thiết bị từ Master đến Slave;

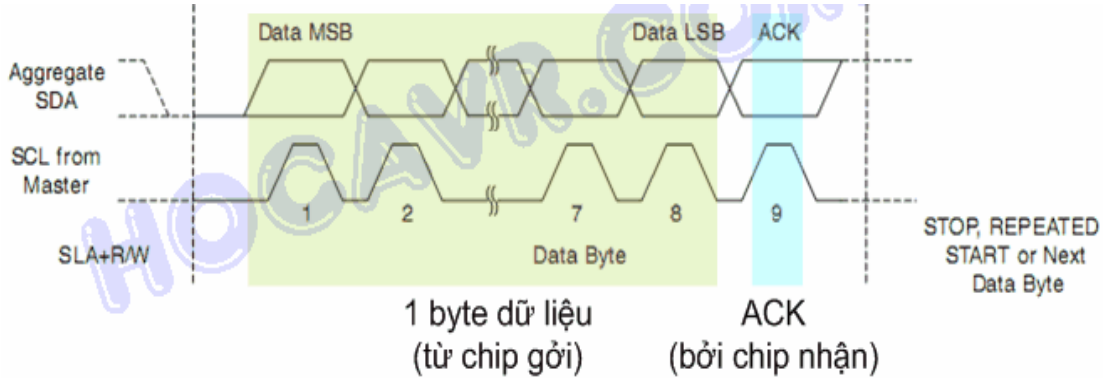


Hình 4-22- Giản đồ xung định dạng gói địa chỉ trong giao tiếp I2C.

- ***General call- Cuộc gọi chung:*** Khi Master phát đi gói địa chỉ có dạng 0 (thực chất là 0+W) tức nó muốn thực hiện một cuộc gọi chung đến tất cả các Slave. Tất nhiên cho phép hay không cho phép là do cuộc gọi chung quyết định. Nếu Slave được cài đặt cho phép cuộc gọi chung, chúng sẽ đáp lại Master bằng bit ACK. Cuộc gọi chung thường xuảy ra khi Master muốn gởi dữ liệu chung đến tất cả các Slave. Chú ý là cuộc gọi chung có dạng 0+R là vô nghĩa vì không bao giờ có trường hợp lả Master nhận dữ liệu từ tất cả các Slave vào cùng một thời điểm.

- ***Data Packet Format-Định dạng gói dữ liệu:*** Sau khi địa chỉ đã được phát đi, Slave sẽ đáp ứng lại Master bằng ACK thì quá trình truyền nhận dữ liệu bắt đầu giữa cặp Master và Slave này. Tuỳ vào bit R/W trong gói địa chỉ, dữ liệu có thể được truyền theo hướng từ Master đến Slave hay từ Slave đến Master. Dù di chuyển theo hướng nào thì gói dữ liệu luôn bao gồm 9 bits trong đó 8 bits đầu là dữ liệu, 1 bit cuối là ACK. 8 bits dữ

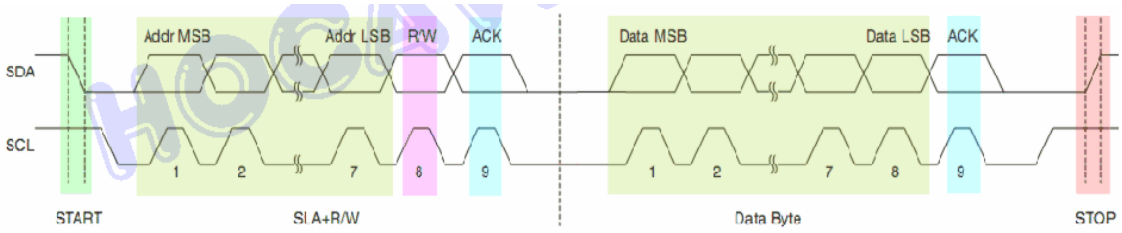
liệu do thiết bị phát gửi và bit ACK do thiết bị nhận tạo ra. Ví dụ Master tiến hành gửi dữ liệu đến Slave, nó sẽ phát ra 8 bits dữ liệu, Slave nhận và phát ra ACK (kéo SDA xuống 0 ở chân thứ 9) sau đó Master sẽ quyết định gởi tiếp byte dữ liệu khác hay không. Nếu Slave phát tín hiệu NOT ACK (không tác động chân SDA xuống mức thấp ở xung giữ nhịp thứ 9) sau khi nhận dữ liệu thì Master sẽ kết thúc quá trình gửi bằng cách phát đi trạng thái STOP. Hình bên dưới mô tả định dạng gói dữ liệu trong I2C.



Hình 4-23- Định dạng gói dữ liệu trong giao tiếp I2C.

- ***Phối hợp gói địa chỉ và gói dữ liệu:***

Một quá trình truyền nhận thường được bắt đầu từ Master. Master phát đi một bit trạng thái START sau đó gửi gói địa chỉ SLA+R/W trên đường truyền. Tiếp theo nếu có một Slave đáp ứng lại, dữ liệu có thể truyền nhận liên tiếp trên đường truyền (1 hoặc nhiều byte liên tiếp). Khung truyền thông thường được mô tả như hình bên dưới.



Hình 4-24- Định dạng phối hợp gói địa chỉ và dữ liệu trong giao tiếp I2C.

Multi-Master Bus-Đường truyền đa thiết bị chủ: Như đã trình bày ở trên, I2C là chuẩn truyền thông đa thiết bị chủ, nghĩa là tại một thời điểm có thể có nhiều hơn 1 thiết bị làm Master nếu các thiết bị này phát ra bit trạng thái START cùng một lúc. Nếu các Master có cùng yêu cầu và thao tác đối với Slave thì chúng có thể cùng tồn tại và quá trình truyền nhận có thể thành công. Tuy nhiên trong đa số trường hợp sẽ có một số Master bị

thất lạc. Một Master bị thất lạc khi nó truyền/nhận một mức cao trong khi một Master khác truyền/nhận mức thấp.

III. Kết luận:

Chúng ta đã tìm hiểu những kiến thức căn bản nhất về chuẩn giao tiếp I2C, trong đó bao gồm định nghĩa về các thuật ngữ, cấu trúc gói địa chỉ và gói dữ liệu. Thế nhưng đây chỉ mới là các kiến thức căn bản chung nhất giúp cho chúng ta hiểu nguyên lý hoạt động của hệ thống giao tiếp I2C. Trong những bài sau, chúng ta sẽ tìm hiểu cách mà hệ thống linux dùng để quản lý đường truyền I2C giao tiếp với các thiết bị Slave khác. Đồng thời sẽ đưa ra những bài tập ví dụ viết chương trình user giao tiếp với eeprom 24c08. Trên cơ sở những phương pháp lập trình này người học có thể áp dụng điều khiển các thiết bị hoạt đông theo chuẩn I2C khác.

B- I2C TRONG LINUX:

I. Giới thiệu:

Giao tiếp theo chuẩn I2C được chia làm 2 thành phần, khởi tạo giao thức truyền từ *Master* và quy định giao thức nhận của *Slave*. Ở đây chip AT91SAM9260 sẽ đóng vai trò là *Master* được điều khiển bởi hệ điều hành nhúng *Linux*. Và *Slave* là các thiết bị ngoại vi do hệ hệ thống *Linux* điều khiển. Giao thức truyền của *Master* phải phù hợp với giao thức nhận của *Slave* thì chúng mới có thể thực hiện thành công việc truyền và nhận dữ liệu với nhau.

Bài này sẽ nghiên cứu các giao thức mà hệ điều hành *Linux* hỗ trợ giao tiếp với nhiều thiết bị ngoại vi khác nhau. Để sau khi hiểu nguyên lý hoạt động của từng thiết bị *Slave* chúng ta có thể áp dụng vào điều khiển dễ dàng.

Các kiến thức có liên quan đến chuẩn I2C của hệ điều hành *Linux* được viết rất đầy đủ trong thư mục Documentation/I2C của mã nguồn *kernel*. Những thông tin sau đây được biên soạn từ tài liệu này, nếu có những vấn đề không được trình bày rõ các bạn có thể tham khảo thêm.

Linux quy định những thuật ngữ sau để thuận tiện cho việc mô tả các giao thức I2C:

`S (1 bit) :`	Start bit
`P (1 bit) :`	Stop bit
`Rd/Wr (1 bit) :`	Read/Write bit. Rd = 1, Wr = 0.
`A, NA (1 bit) :`	Bit ACK hoặc bit NOT ACK.
`Addr (7 bits):`	Là 7 bits địa chỉ thiết bị (device address). Địa chỉ này có thể mở rộng lên thành 10 bits.
`Comm (8 bits):`	Là 8 bits địa chỉ của thanh ghi trong thiết bị. Mỗi thiết bị bao gồm có nhiều ô nhớ khác nhau, mỗi ô nhớ chứa một thông tin riêng và có duy nhất một địa chỉ.
`Data (8 bits):`	Là 8 bits dữ liệu truyền/nhận, trong chế độ truyền theo byte. Trong chế độ truyền theo word, được hiểu là 8 bits cao DataHigh hoặc 8 bits thấp DataLow.

`Count (8 bits):` Là 8 bits quy định số bytes của khối dữ liệu trong chế độ truyền/nhận dữ liệu theo khối.

Lưu ý: Những tham số chứa trong dấu ngoặc vuông [..] được hiểu là được truyền từ thiết bị *Slave* đến *Master* ngược lại được truyền từ *Master* đến *Slave*. Ví dụ:

`Data` : Là dữ liệu truyền từ *Master* đến *Slave*;

`[Data]`: Là dữ liệu truyền từ *Slave* đến *Master*;

Linux xây dựng một giao thức để giao tiếp với chuẩn I2C mang tên SMBus (System Management Bus) để giao tiếp với các thiết bị được quy định theo chuẩn I2C. Có nhiều thiết bị ngoại vi khác nhau thì cũng sẽ có nhiều cấu trúc giao tiếp khác nhau tương ứng với từng chức năng truy xuất mà thiết bị hỗ trợ. Nhiệm vụ của người lập trình là phải lựa chọn giao thức SMBus phù hợp để giao tiếp với thiết bị cần điều khiển. Phần tiếp theo sẽ trình bày các giao thức SMBus thường được sử dụng mà *Linux* hỗ trợ.

II. Các giao diện hàm trong driver *I2C*:

a. Giới thiệu về driver I2C trong Linux:

Thông thường các thiết bị I2C được điều khiển bởi lớp *kernel*, thông qua các lệnh được định nghĩa trong mã nguồn *Linux*. Thế nhưng chúng ta cũng có thể sử dụng những giao diện được định nghĩa sẵn trong *driver* I2C do *Linux* hỗ trợ. Nghĩa là thay vì điều khiển trực tiếp thông qua các hàm trong *kernel*, chúng ta sẽ điều khiển gián tiếp thông qua các giao diện hàm trong *driver*. Các giao diện hàm này sẽ gọi các hàm trong *kernel* hỗ trợ để điều khiển *Slave* theo yêu cầu từ *user*.

Driver I2C được chứa trong thư mục `drivers/I2C` của mã nguồn *Linux*. Thông thường chúng ta sử dụng các hàm trong tập tin *driver* `I2C-device.c`. Trong tập tin `I2C-dev.c` định nghĩa các giao diện hàm như: `read()`, `write()` và `ioctl()` và một số những giao diện khác phục vụ cho đóng và mở tập tin thiết bị. Trong đó `read()` và `write()` dùng để đọc và ghi vào thiết bị *Slave* theo dạng từng khối. Ngoài ra bằng cách dùng giao diện hàm `ioctl()` chúng ta có thể thực hiện tất cả các theo tác đọc/ghi trên thiết bị *Slave* và một số những thao tác khác như định địa chỉ thiết bị, kiểm tra các hàm chức năng, chọn chế độ giao tiếp 8 bits hay 10 bits địa chỉ.

Tiếp theo chúng ta sẽ tìm hiểu các giao diện `read()`, `write()` và `ioctl()`.

b. Giao tiếp với thiết bị I2C ***Slave thông qua driver*** I2C***:***

Để thuận tiện cho việc tìm hiểu và ứng dụng các hàm giao diện sao cho thuận lợi nhất, chúng ta sẽ tìm hiểu theo hướng nghiên cứu các bước thực hiện và giải thích từng đoạn chương trình ví dụ. Các đoạn chương trình ví dụ này nằm trong *user application*, sử dụng các hàm hỗ trợ trong *driver* I2C.

Bước 1: Đầu tiên muốn giao tiếp với Bus I2C chúng ta phải xác định sự tồn tại của giao diện tập tin thiết bị trong thư mục `/dev/`. Thường thì tập tin thiết bị này có tên là `/dev/i2c-0`. Nếu không có tập tin thiết bị này trong thưc mục `/dev/` chúng ta phải biên dịch lại mã nguồn *kernel* và check vào mục biên dịch *driver* I2C. Sau khi biên dịch xong, cài đặt vào kit và khởi động lại hệ thống.

Bước 2: Trong *user application*, mở tập tin thiết bị I2C-0 chuẩn bị thao tác;

```
/*Biến lưu số mô tả tập tin khi thiết bị được mở*/
int fd_I2C;
/*Gọi giao diện hàm mở tập tin thiết bị*/
fd_I2C = open("/dev/I2C-0, O_RDWR);
/*Kiểm tra lỗi trong quá trình mở tập tin thiết bị*/
if (fd_I2C < 0) {
 /* ERROR HANDLING; you can check errno to see what went  wrong */
/*Nếu có lỗi xảy ra thì thoát chương trình đang gọi*/
 exit(1);
}
```

Bước 3: Xác định địa chỉ thiết bị muốn giao tiếp;

Bằng cách gọi giao diện hàm `ioctl()` với tham số I2C_*SLAVE* như sau:

```
/*Biến lưu địa chỉ thiết bị cần giao tiếp*/
int addr = 0x40;
/*Gọi hàm ioctl() để xác định địa chỉ thiết bị với số định danh lệnh  là I2C_SLAVE*/
 if (ioctl(fd_I2C, I2C_SLAVE, addr) < 0) {
/*ERROR HANDLING; you can check errno to see what went wrong*/
```

```
/*Thoát khỏi chương trình khi có lỗi xảy ra*/
    exit(1);
  }
```

Bước 4: Sử dụng các hàm giao diện `read()`, `write()` và `ioctl()` để truyền/nhận dữ liệu với *Slave*.

- Dùng giao diện hàm read để đọc dữ liệu từ *Slave*:

```
/*Khai báo biến đệm lưu giá trị trả về khi gọi hàm read()*/
char buf[10];
/*Gọi hàm read() đọc dữ liệu hiện tại của thiết bị Slave, kích thước đọc về là 1 byte*/
if (read(fd_I2C, buf, 1) != 1) {
/* ERROR HANDLING: I2C transaction failed */
/*Trong trường hợp có lỗi xảy ra thoát khỏi chương trình thực thi*/
exit (1);
}
/*Lúc này dữ liệu đã chứa trong bộ nhớ đệm buf[0]*/
```

Cấu trúc giao thức này như sau:

S Addr Rd [A] [Data] NA P

Trong trường hợp đọc về nhiều byte liên tiếp, chúng ta cũng dùng giao diện hàm `read()` nhưng với độ dài n: `read(fd_I2C, buf, n);` Khi đó cấu trúc của giao thức ngày như sau:

S Addr Rd [A] [Data] [A] [Data] ... [A] [Data] NA P

***Trong một số thiết bị Slave dùng* I2C*, có hỗ trợ một thanh ghi dùng để lưu địa chỉ hiện tại truy xuất dữ liệu, do đó khi gọi hàm read() dữ liệu đọc về sẽ là nội dung của ô nhớ có địa chỉ lưu trong thanh ghi này. Khi đọc 1 byte thì nội dung của thanh ghi địa chỉ này sẽ tăng lên 1 đơn vị.*

- Dùng giao diện hàm write để ghi dữ liệu vào *Slave*:

Chúng ta cũng có thể dùng giao diện hàm `write()` để ghi dữ liệu và *Slave*. Khi đó cấu trúc của giao thức này như sau:

- Trong trường hợp ghi 1 byte vào thiết bị *Slave*:

S Addr Wr [A] Data [A] P

- Trong trường hợp ghi nhiều byte vào thiết bị *Slave*:

S Addr Wr [A] Comm [A] Data [A] ... Data [A] P

Ví dụ sau sẽ minh họa cách ghi bộ nhớ đệm có độ dài 3 bytes vào thiết bị *Slave*:

```
buf[0] = Data0;
buf[1] = Data1;
buf[2] = Data2;
if (write(fd_I2C, buf, 3) ! =3) {
/* ERROR HANDLING: I2C transaction failed */
}
```

- Dùng giao diện hàm `ioctl()` truyền/nhận dữ liệu và các chức năng điều khiển khác trong *driver* I2C, mỗi chức năng sẽ tương ứng với các tham số lệnh khác nhau.
  - `ioctl(file, I2C_SLAVE, long addr)` : Dùng để thay đổi địa chỉ thiết bị muốn giao tiếp. Địa chỉ này là giá trị địa chỉ khi chưa thêm bit R/W vào vị trí bit thứ 8 (LSB) của gói địa chỉ.
  - `ioctl(file, I2C_TENBIT, long select)`: Dùng để quy định chế độ giao tiếp 8 bits hay 10 bits địa chỉ. Chế độ giao tiếp địa chỉ 10 bits nếu `select` khác 0, chế độ giao tiếp địa chỉ 7 bit nếu `select` bằng 0. Và mặc định là chế độ giao tiếp 7 bits địa chỉ. Hàm này chỉ hợp lệ khi *driver* I2C có hỗ trợ `I2C_FUNC_10BIT_ADDR`.
  - `ioctl(file, I2C_PEC, long select)`: Dùng để khởi động hay tắt chế độ kiểm tra lỗi trong quá trình truyền. Bật chế độ kiểm tra lỗi khi `select` khác 0, tắt chế độ kiểm tra lỗi khi `select` bằng 0. Mặc định là tắt kiểm tra lỗi. Hàm này chỉ hợp lệ khi *driver* có hỗ trợ I2C_FUNC_SMBUS_PEC;
  - `ioctl(file, I2C_FUNCS, unsigned long *funcs)`: Dùng để kiểm tra chức năng *funcs của *driver* I2C có hỗ trợ hay không.
  - `ioctl(file, I2C_RDWR, struct I2C_rdwr_ioctl_data *msgset)`: Dùng để kết hợp hay quá trình đọc và ghi trong có ngăn cách bởi trạng thái STOP. Hàm chỉ hợp lệ khi *driver* có hỗ trợ chức năng I2C_FUNC_I2C. Hàm

này có tham số là `struct I2C_rdwr_ioctl_data *msgset`. Với cấu trúc I2C_rdwr_ioctl_data được định nghĩa như sau:

```
struct I2C_rdwr_ioctl_data {
      struct I2C_msg *msgs;   /* Con trỏ đến mảng dữ liệu của I2C */
      int nmsgs;              /* Số lượng I2C_msg muốn đọc và ghi */
  }
```

với cấu trúc struct I2C_msg lại được định nghĩa như sau:

```
struct I2C_msg {
  __u16 addr;
  __u16 flags;
#define I2C_M_TEN                       0x0010
#define I2C_M_RD                        0x0001
#define I2C_M_NOSTART                   0x4000
#define I2C_M_REV_DIR_ADDR              0x2000
#define I2C_M_IGNORE_NAK                0x1000
#define I2C_M_NO_RD_ACK                 0x0800
#define I2C_M_RECV_LEN                  0x0400
  __u16 len;
  __u8 * buf;
 }
```

Như vậy mỗi một phần tử `msg[]` sẽ chứa một con trỏ khác, con trỏ này chính là bộ đệm đọc hay ghi tùy thuộc vào giá trị của cờ `I2C_M_RD` được bật tương ứng cho từng `msg[]`. Bên cạnh đó `msg[]` còn có thông số địa chỉ thiết bị, chiều dài bộ nhớ đệm, ...

- `ioctl(file, I2C_SMBUS, struct I2C_smbus_ioctl_data *args)`: Bản thân hàm này không gọi thực thi một chức năng cụ thể. Mà nó sẽ gọi thực thi một trong nhựng hàm sau đây tùy vào các tham số chứa trong *arg. Các chức năng được sử dụng tùy theo quy định của tham số *args.

III. Các giao thức SMBus:

1. Giao thức SMBus Quick Command:

Được định nghĩa theo dạng hàm sau:

```
__s32 I2C_smbus_write_quick(int file, __u8 value);
```

Giao thức này chỉ truyền một bit cho thiết bị *Slave* nằm tại vị trí của bit R/W có cấu trúc như sau:

A Addr Rd/Wr [A] P

2. Giao thức SMBus Receive Byte:

Được định nghĩa theo dạng hàm sau:

```
__s32 I2C_smbus_read_byte(int file);
```

Giao thức này ra lệnh đọc một byte tại địa chỉ hiện tại từ thiết bị *Slave*. Thông thường lệnh này được dùng sau các lệnh khác dùng để quy định địa chỉ muốn đọc. Giao thức có cấu trúc như sau:

S Addr Rd [A] [Data] NA P

3. Giao thức SMBus Send Byte:

Được định nghĩa theo dạng hàm sau:

```
__s32 I2C_smbus_write_byte(int file, __u8 value);
```

Giao thức này dùng để truyền một byte xuống thiết bị *Slave* có cấu trúc:

S Addr Wr [A] Data [A] P

4. Giao thức SMBus Read Byte:

Được định nghĩa theo dạng hàm sau:

```
__s32 I2C_smbus_read_byte_data(int file, __u8 command);
```

Giao thức này dùng để nhận một byte có địa chỉ thanh ghi là command trong thiết bị *Slave*. Giao thức này tương đương với hai giao thức send byte và receive byte hoạt động liên tiếp nhưng không có bit STOP ngăn cách giữa hai giao thức này. Cấu trúc của giao thức SMBus như sau:

S Addr Wr [A] Comm [A] S Addr Rd [A] [Data] NA P

5. Giao thức SMBus Read Word:

Được định nghĩa theo dạng hàm sau:

```
__s32 I2C_smbus_read_word_data(int file, __u8 command);
```

Giao thức này dùng để đọc một word data, bao gồm byte thấp và byte cao bắt đầu tại địa chỉ command. Giao thức có cấu trúc:

S Addr Wr [A] Comm [A] S Addr Rd [A] [DataLow] A [DataHigh] NA P

6. Giao thức SMBus Write Byte:

Được định nghịa theo dạng hàm sau:

```
__s32 I2C_smbus_write_byte_data(int file, __u8 command, __u8 value);
```

Giao thức này dùng để ghi một byte dữ liệu đến thanh ghi có địa chỉ command có cấu trúc như sau:

S Addr Wr [A] Comm [A] Data [A] P

7. Giao thức SMBus Write Word:

Được định nghĩa theo dạng hàm sau:

```
__s32 I2C_smbus_write_word_data(int file, __u8 command, __u16
value);
```

Giao thức này dùng để ghi một word dữ liệu có 2 byte, HighByte và LowByte, đến thanh ghi có địa chỉ bắt dầu là command. Giao thức có cấu trúc:

S Addr Wr [A] Comm [A] DataLow [A] DataHigh [A] P

8. Giao thức SMBus Block Read:

Giao thức này được định nghĩa theo dạng hàm sau:

```
__s32 I2C_smbus_read_block_data(int file, __u8 command, __u8
*values);
```

Dùng để đọc từ thiết bị *Slave* một khối dữ liệu có chiều dài lên tới 32 bytes (Tùy theo khả năng của thiết bị *Slave*) có địa chỉ bắt đầu từ command với số byte đọc được quy định trong tham số count. Giao thức thức này có cấu trúc sau:

S Addr Wr [A] Comm [A]
S Addr Rd [A] [Count] A [Data] A [Data] A ... A [Data] NA P

9. Giao thức SMBus Block Write:

Được định nghĩa theo dạng hàm sau:

```
__s32 I2C_smbus_write_block_data(int file, __u8 command, __u8
length,
__u8 *values);
```

Giao thức này dùng để ghi một khối dữ liệu có chiều dài quy định bởi length (tối đa là 32 bits) tại địa chỉ bắt đầu là command. Giao thức có cấu trúc như sau:

```
S Addr Wr [A] Comm [A] Count [A] Data [A] Data [A] ... [A] Data [A] P
```

***Trên đây là 9 giao thức căn bản và thường sử dụng nhất trong giao tiếp theo chuẩn I2C. Đây chỉ mới là những giao thức do Linux quy định sẵn cho Master khi muốn thao tác với thiết bị Slave. Để giao tiếp thành công với mỗi thiết bị, trước tiên chúng ta phải tìm hiểu quy định về cách thức giao tiếp của thiết bị đó, sau đó sẽ áp dụng một hay nhiều giao thức SMBus trên để thực hiện một tác vụ cụ thể do thiết bị Slave hỗ trợ.*

IV. Kết luận:

Trong bài này chúng ta đã nghiên cứu những nguyên lý và các bước căn bản về thao tác truyền dữ liệu theo chuẩn I2C trong *driver* I2C_dev. *Driver* I2C do *Linux* hỗ trợ bao gồm tất cả những giao thức phù hợp với các thiết bị *Slave* khác nhau. Trong các bài sau, chúng ta sẽ đi tìm hiểu giao thức truyền nhận của một số thiết bị *Slave* đồng thời sẽ áp dụng các hàm trong *driver* I2C điều khiển truy xuất dữ liệu từ các thiết bị này.

C- THỰC HÀNH GIAO TIẾP EEPROM I2C 24C08:

I. Giới thiệu chung về EEPROM 24c08:

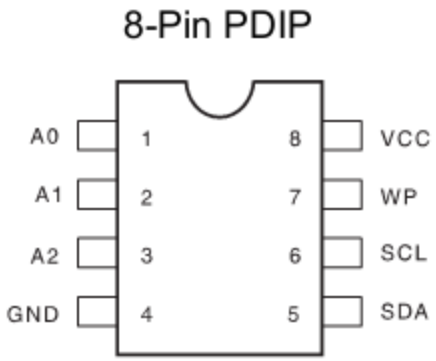
**Để quyển sách này có sự tập trung và đúng hướng, trong phần này chúng tôi không trình bày một cách chi tiết cấu tạo và nguyên lý hoạt động mà chỉ trình bày giao thức truyền dữ liệu theo chuẩn I2C của EEPROM 24C08. Để từ đó áp dụng những lệnh đã học trong hai bài trước vào điều khiển thành công thiết bị Slave này.

1. Mô tả:

EEPROM 24C08 có những chức năng sau:

- Dung lượng 8Kb (1KB);
- Là ROM có thể lập trình và xóa bằng xung điện;
- Chế độ truyền theo chuẩn I2C;

2. Sơ đồ chân:



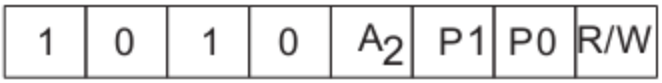
Hình 4-25- Sơ đồ chân của EEPROM 24c08.

Trong đó:

- A0, A1, A2: Là 3 chân địa chỉ ngõ vào dùng để chọn địa chỉ phân biệt nhiều eeprom ghép song song với nhau. Đối với eeprom 24C08 chỉ có 1 chân A2 được phép chọn lựa địa chỉ. Như vậy chỉ có 2 eeprom 24C08 được nối chung với nhau trên 1 bus I2C.
- SDA và SCL: Là chân dữ liệu và chân clock trong chuẩn I2C;
- WP: Là chân bảo vệ chống ghi vào eeprom;
- VCC và GND là hai chân cấp nguồn cho eeprom;

3. *Địa chỉ thiết bị:*

Địa chỉ thiết bị eeprom 24C08 được quy định như trong hình vẽ sau:



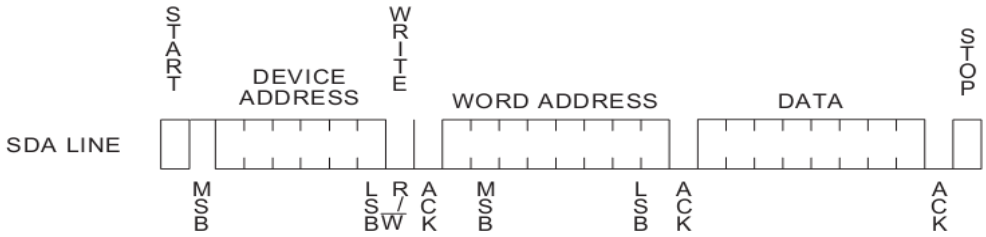
Hình 4-26- Địa chỉ thiết bị trong eeprom 24c08.

4. *Giao thức ghi dữ liệu:*

EEPROM 24Cxx có hai chế độ ghi dữ liệu, ghi theo từng byte và ghi theo từng block.

a. *Ghi theo từng byte:*

Chế độ này được minh họa bằng giao thức sau:

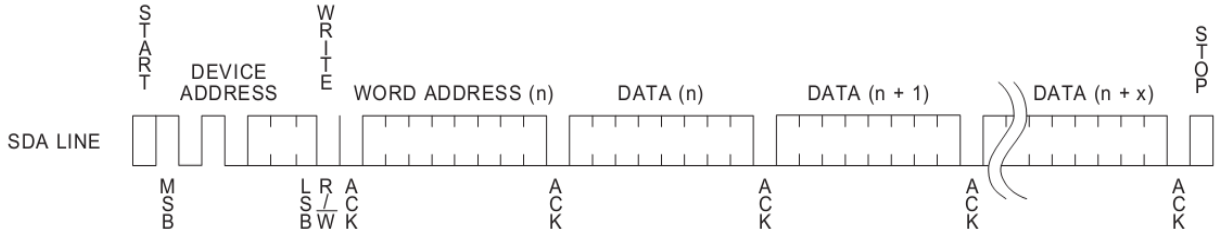


Hình 4-27-

Giãn đồ xung trong chế độ ghi theo từng byte dữ liệu trong eeprom 24c08.

b. *Ghi theo từng block:*

Chế độ này được minh họa bằng giao thức sau:



Hình 4-28-

Giãn đồ xung trong chế độ ghi theo từng khối dữ liệu trong eeprom 24c08.

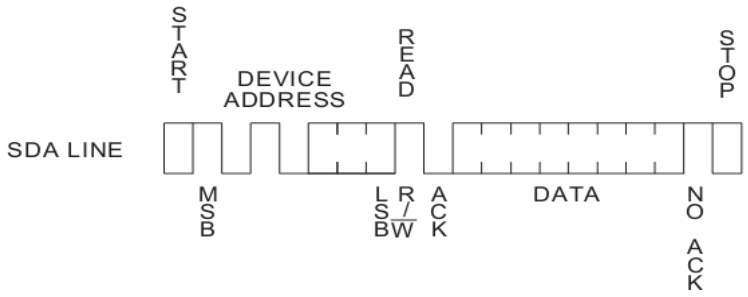
***EEPROM 24C08 có thể hỗ trợ chế độ ghi dữ liệu theo trang có độ dài lên tới 16 bytes.*

5. *Giao thức đọc dữ liệu:*

EEPROM 24Cxx có 3 chế độ đọc dữ liệu, đọc tại địa chỉ hiện tại, đọc ngẫu nhiên và đọc theo tuần tự;

a. *Đọc tại địa chỉ hiện tại:*

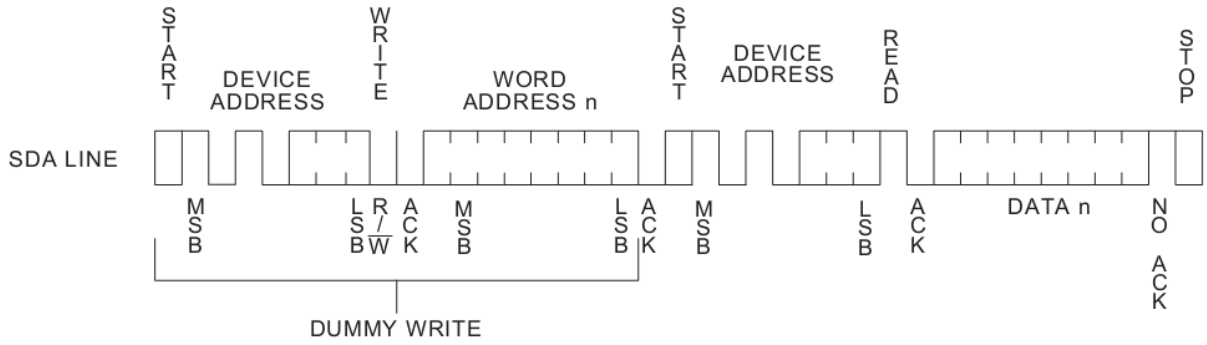
Chế độ này được minh họa bằng giao thức sau:



Hình 4-29- Giãn đồ xung đọc tại địa chỉ hiện tại của 24c08.

b. *Đọc ngẫu nhiên:*

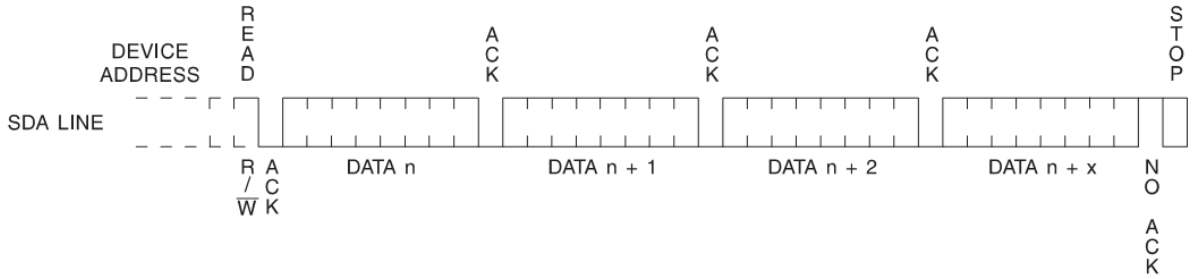
Chế độ này được minh họa bằng giao thức sau:



Hình 4-30- Giãn đồ xung đọc ngẫu nhiên trong eeprom 24c08.

c. *Đọc theo tuần tự:*

Chế độ này được minh họa bằng giao thức sau:



Hình 4-31- Giãn đồ xung đọc theo chế độ tuần tự trong eeprom 24c08.

II. Dự án điều khiển EEPROM 24C08:

1. *Phác thảo dự án:*

Mục đích của dự án là làm cho người học có khả năng sử dụng những hàm trong driver I2C hỗ trợ để điều khiển EEPROM 24C08. Bên cạnh đó còn giúp người học rèn luyện cách viết một chương trình user application sử dụng thư viện liên kết động.

a. *Yêu cầu dự án:*

Chương trình được xây dựng thao tác với eeprom 24Cxx với các chức năng sau:

- Ghi một chuỗi thông tin bao gồm 256 bytes vào các ô nhớ trong eeprom 24Cxx bắt đầu từ 0 kết thúc 255. Để thực hiện chức năng này, người sử dụng chương trình nhập câu lệnh thực thi theo cú pháp sau:

  ```
  ./<tên chương trình> write_numbers
  ```

- Đọc lần lượt thông tin của các ô nhớ có địa chỉ từ 0 đến 255 trong eeprom 24Cxx xuất ra màn hình hiển thị. Để thực hiện chức năng này người sử dụng chương trình phải nhập câu lệnh thực thi theo cú pháp sau:

  ```
  ./<tên chương trình> read_numbers
  ```

- Ghi chuỗi ký tự được nhập từ người dùng vào eeprom bắt đầu từ ô nhớ có địa chỉ 00h, số ký tự được ghi phụ thuộc vào chiều dài của chuỗi ký tự. Để thực hiện chức năng này, người sử dụng chương trình phải nhập câu lệnh thực thi theo cú pháp sau:

  ```
  ./<tên chương trình> write_string
  ```

- Đọc chuỗi ký tự từ eeprom bắt đầu từ ô nhớ có địa chỉ 00h, số ký tự muốn đọc do người sử dụng chương trình quy định. Để thực hiện chức năng này người dùng phải nhập câu lệnh thực thi theo cú pháp sau:

```
./<tên chương trình> read_string <số ký tự muốn đọc>
```

b. Phân công nhiệm vụ:

- *Driver:*

Sử dụng driver I2C đã được hỗ trợ sẵn trong kernel. (Các hàm chức năng hỗ trợ trong driver I2C đã được chúng tôi trình bày kỹ trong bài trước).

- *Application:*

Chương trình trong user được xây dựng thành nhiều "lớp" con khác nhau, được chia thành các tập tin như: 24cXX.h, 24cXX.c và eeprom.c;

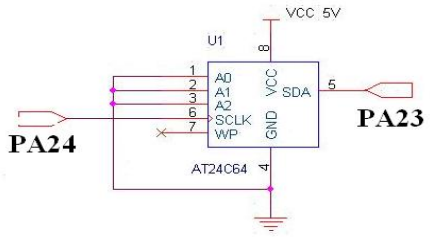
- Tập tin chương trình chính mang tên eeprom.c chứa hàm main() được khai báo dưới dạng cấu trúc tham số để thu thập thông tin từ người dùng. eeprom.c gọi các hàm được định nghĩa trong các tập tin khác (Chủ yếu là tập tin 24cXX.h). eeprom.c thực hiện 4 nhiệm vụ:
    - Mở tập tin thiết bị driver I2C mang tên I2C-0 trong thư mục /dev/ sau đó quy định các thông số như: Địa chỉ Slave thiết bị, số bits địa chỉ thanh ghi, ... lưu vào cấu trúc eeprom để sử dụng trong những lần tiếp theo.

  Tùy theo tham số lựa chọn của người dùng mà thực hiện một trong 4 chức năng sau:
    - Ghi lần lượt các số từ 00h đến FFh đến các ô nhớ có địa chỉ từ 00h đến FFh trong eeprom 24Cxx;
    - Đọc nội dung của các ô nhớ từ 00h đến FFh lần lượt ghi ra màn hình hiển thị;
    - Ghi chuỗi ký tự được nhập từ người dùng vào eeprom 24Cxx, với giới hạn số ký tự do eeprom quy định. Vị trí ghi bắt đầu từ địa chỉ 00h.
    - Đọc chuỗi ký tự được lưu trong epprom ghi ra màn hình hiển thị, với kích thước đọc do người lập trình quy định (nhập từ tham số người dùng);

2. Thực hiện:

a. Kết nối phần cứng:

Các bạn kết nối phần cứng theo sơ đồ sau đây:



Hình 4-32- Sơ đồ kết nối eeprom 24c08.

b. Chương trình dirver:

Driver được sử dụng trong dự án này mang tên I2C-0 trong thư mục /dev/. Những chức năng lệnh trong driver được trình bày kỹ trong bài trước.

c. Chương trình application:

- Tập tin chương trình chính: eeprom.c

```
/*Khai báo thư viện cho các hàm cần dùng trong chương trình*/
#include <stdio.h>
#include <fcntl.h>
#include <getopt.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "24cXX.h" /*Gán tập tin định nghĩa 24cXX.h vào chương trình chính */
/*Hàm in hướng dẫn cho người dùng trong trường hợp nhập sai cú pháp*/
void use(void)
{
      printf("./i2c  read_numbers|read_string|write_numbers|write_string
      (<number for read_string>|<string_to_write_string>)");
      exit(1);
```

```
}
```
*/*Hàm thực hiện chức năng đọc nội dung của eeprom struct eeprom *e, từ ô nhớ có địa chỉ int addr, kích thước muốn đọc là int size. Mỗi lần đọc, thông tin được xuất ra màn hình dưới dạng số hex.*/*
```
static int read_from_eeprom(struct eeprom *e, int addr, int size)
{
```
*/*Khai báo biến lưu ký tự đọc về và biến đếm để đọc về một khối ký tự*/*
```
int ch, i;
```
*/*Vòng lặp đọc thông tin từ eeprom tại địa chỉ addr đến size, sau khi đọc tăng giá trị addr lên 1 đơn vị*/*
```
for(i = 0; i < size; ++i, ++addr)
{
```
*/*Đọc thông tin từ eeprom tại địa chỉ addr đồng thời kiểm tra lỗi trong quá trình đọc*/*
```
if((ch = eeprom_read_byte(e, addr)) < 0)
```
*/*In ra thông báo trong trường hợp có lỗi xảy ra*/*
```
printf("read error\n");
```
*/*Cứ mỗi 16 lần in thông tin thì xuống hàng một lần*/*
```
if( (i % 16) == 0 )
printf("\n %.4x|  ", addr);
```
*/*Cứ mỗi 8 lần tin thông tin thì thêm một khoảng trắng*/*
```
else if( (i % 8) == 0 )
printf("  ");
printf("%.2x ", ch);
}
```
*/*In hai ký tự xuống hàng khi kết thúc quá trình đọc thông tin*/*
```
printf("\n\n");
return 0;
}
```
*/*Hàm thực hiện chức năng ghi số từ 00 đến ff bắt đầu từ địa chỉ addr*/*
```
static int write_to_eeprom(struct eeprom *e, int addr)
{
```

```
	/*Biến đếm điều khiển vòng lặp ghi dữ liệu vào eeprom*/
	int i;
	/*Vòng lặp từ 0 đến 256, mỗi lần đọc tăng giá trị addr lên 1 đơn vị chuẩn bị
	cho lần đọc tiếp theo*/
	for(i=0; i<256; i++, addr++)
	{
		/*Ghi thông tin ra màn hình trước khi ghi vào eeprom*/
		if( (i % 16) == 0 )
			printf("\n %.4x|  ", addr);
		else if( (i % 8) == 0 )
			printf("  ");
		printf("%.2x ", i);
		/*Ghi dữ liệu vào eeprom tại địa chỉ addr, kiểm tra lỗi trong quá trình ghi
		dữ liệu*/
		if(eeprom_write_byte(e, addr, i)<0)
			printf("write error\n");
	}
	/*In ký tự xuống dòng khi quá trình đọc kết thúc*/
	printf("\n");
	return 0;
}
/*Hàm thực hiện chức năng ghi chuỗi thông tin từ người dùng vào eeprom*/
int test_write_ee(struct eeprom *e, char *data, int size)
{
	int i;
	/*Vòng lặp ghi từng ký tự trong chuỗi char *data vào eeprom tại địa chỉ i*/
	for(i=0;i<size;i++){
		/*Ghi ký tự data[i] vào địa chỉ i trong eeprom đồng thời kiểm tra lỗi trong
		quá trình ghi dữ liệu*/
		if(eeprom_write_byte(e, i, data[i])<0){
			printf("write error\n");
			return -1;
```

```
            }
      }
      /*In thông báo khi quá trình ghi kết thúc*/
      printf("Writing finishes!");
      printf("\n");
      return 0;
}
```

*/*Hàm thực hiện chức năng đọc ký tự từ eeprom bắt đầu từ địa chỉ 00, kích thước là size*/*

```
int test_read_ee(struct eeprom *e, int size)
{
      int i;
      char ch;
      /*Đọc ký tự từ 0 đến size ghi ra màn hình hiển thị*/
      for(i=0;i<size;i++)
      {
            if((ch = eeprom_read_byte(e, i)) < 0) {
                  printf("read error\n");
                  return -1;
            }
            /*Dùng hàm putchat() để ghi ký tự ra màn hình hiển thị*/
            putchar(ch);
      }
      printf("\n");
      return 0;
}
int main(int argc,char** argv)
{
      /*Khai báo cấu trúc eeprom lưu những thông tin loại eeprom, số bit địa chỉ thanh ghi, .. cấu trúc được định nghĩa trong tập tin 24cXX.h*/
      struct eeprom e;
      /*Số ký tự nhập vào*/
      int number_of_char;
```

*/*Thông báo mở tập tin thiết bị*/*

```
printf("Open /dev/i2c-0 with 8 bit mode\n");
```

*/*Mở tập tin thiết bị i2c, cập nhật địa chỉ eeprom, hàm eeprom được định nghĩa trong tập tin 24cXX.c và 24cXX.h*/*

```
if(eeprom_open("/dev/i2c-0",0x50,EEPROM_TYPE_8BIT_ADDR, &e) < 0)
      printf("unable to open eeprom device file \n");
```

*/*Trong trường hợp thực hiện chức năng đọc giá trị ô nhớ từ 00h đến ffh ghi ra màn hình hiển thị*/*

```
if (!strcmp(argv[1],"read_numbers")){
      fprintf(stderr,"Reading 256 bytes from 0x0\n");
      read_from_eeprom(&e,0,256);
```

*/*Trong trường hợp thực hiện chức năng 1, ghi số từ 00 đến ff lên các ô nhớ trong eeprom bắt đầu từ địa chỉ 00h*/*

```
} else if (!strcmp(argv[1],"write_numbers")) {
      fprintf(stderr, "Writing 0x00-0xff into 24C08 \n");
      write_to_eeprom(&e,0);
      printf("Writing finishes!");
```

*/*Trong trường hợp thực hiện chức năng đọc chuỗi thông tin từ eeprom hiển thị ra màn hình*/*

```
} else if (!strcmp(argv[1],"read_string")) {
      number_of_char = atoi(argv[2]);
      printf("Reading eeprom from address 0x00 with size %d
      \n",number_of_char);
      printf ("The read string is: ");
      test_read_ee(&e,number_of_char);
```

*/*Trong trường hợp thực hiện chức năng ghi chuỗi thông tin từ người dùng vào eeprom*/*

```
} else if (!strcmp(argv[1],"write_string")) {
      number_of_char = strlen(argv[2]);
      fprintf(stderr, "Writing eeprom from address 0x00 with size
      %d \n",number_of_char);
      test_write_ee(&e,argv[2],number_of_char);
```

```
        /*Trong trường hợp có lỗi xảy ra do cú pháp*/
        } else {
                use();
                exit(1);
        }
        eeprom_close(&e);
        return 0;
}
```

Hai tập tin sau được dùng để định nghĩa những hàm được sử dụng trong eeprom.c, truy xuất trực tiếp đến các hàm trong driver I2C-0 để điều khiển eeprom. Nếu có nhu cầu các bạn có thể đọc thêm để nghiên cứu ý nghĩa của từng hàm trong tập tin 24cXX.c. Nếu không chúng ta sẽ sử dụng những hàm chức năng được định nghĩa trong tập tin 24cXX.h như là những hàm hỗ trợ sẵn trong thư viện liên kết tĩnh.

- Tập tin thư viện: 24Cxx.h

```
/*Khai báo tập tin định nghĩa 24cXX.h*/
#ifndef _24CXX_H_
#define _24CXX_H_
/*Gán thư viện của driver i2c-dev cho chuẩn i2c*/
#include <linux/i2c-dev.h>
#include <linux/i2c.h>
/*Định nghĩa hằng số quy định chế độ truy xuất địa chỉ*/
#define EEPROM_TYPE_UNKNOWN0
#define EEPROM_TYPE_8BIT_ADDR    1/*Chế độ truy xuất địa chỉ 8 bits*/
#define EEPROM_TYPE_16BIT_ADDR   2/*Chế độ truy xuất địa chỉ 16 bits*/
/*Định nghĩa cấu trúc eeprom
Cấu trúc này bao gồm những thông tin sau:
char *dev: con trỏ char lưu tên đường dẫn thiết bị
int addr: Biến lưu địa chỉ thiết bị Slave
int fd: Biến lưu số mô tả tập tin thiết bị khi được mở
int type: Biến lưu kiểu eeprom truy xuất*/
```

```
struct eeprom
{
      char *dev;       // device file i.e. /dev/i2c-N
      int addr;        // i2c address
      int fd;          // file descriptor
      int type;        // eeprom type
};
/*
 Hàm eeprom_open() dùng để mở tập tin thiết bị Slave eeprom;
 Bao gồm các thông số sau:
 char *dev_fqn: Đường dẫn đến tập tin thiết bị muốn mở;
 int addr: Địa chỉ của thiết bị Slave muốn mở;
 int type: Loại eeprom cần truy xuất;
  struct eeprom: Sau khi mở thành công tập tin thiết bị với đia chỉ addr, loại eeprom
  type, ... Các thông tin này sẽ được cập nhật trong  các field tương ứng của cấu trúc
  struct eeprom;
 */
int eeprom_open(char *dev_fqn, int addr, int type, struct eeprom*);
/*
  Hàm eeprom_close() dùng để đóng thiết bị eeprom sau khi truy xuất xong;
  Hàm bao gồm các tham số sau:
  struct eeprom *e: Là con trỏ thiết bị eeprom muốn đóng, sau khi đóng những thông tin
  trong các field của thiết bị *e được khôi phục lại trạng thái ban đầu;
*/
int eeprom_close(struct eeprom *e);
/*
  Hàm eeprom_read_byte() dùng để đọc một byte dữ liệu có địa chỉ __u16 mem_addr
  của thiết bị struct eeprom* e; Giá trị đọc được sẽ trả về làm giá trị của hàm;
```

Lưu ý: Trước khi sử dụng hàm này, phải đảm bảo là thiết bị struct eeprom e đã được cập nhật địa chỉ trước đó bằng cách gọi hàm eeprom_oepn() và hàm này sẽ gọi thực thi hàm ioctl(fd, I2C_SLAVE, address); Được hỗ trợ trong driver /dev/i2c-N*

*/

```
int eeprom_read_byte(struct eeprom* e, __u16 mem_addr);
```

/*

*Hàm eeprom_read_current_byte() đọc giá trị ô nhớ có địa chỉ hiện tại chứa trong thanh ghi địa chỉ của eeprom struct eeprom *e;*

*/

```
int eeprom_read_current_byte(struct eeprom *e);
```

/*

*Hàm eeprom_write_byte() dùng để ghi giá trị __u8 data vào ô nhớ có địa chỉ __u16 mem_addr của thiết bị struct eeprom *c;*

*/

```
int eeprom_write_byte(struct eeprom *e, __u16 mem_addr, __u8 data);
#endif
```

- Tập tin chương trình con: 24cXX.c

*/*Tập tin 24cXX có nhiệm vụ định nghĩa những hàm đã khai báo trong tập tin 24cXX.h*/*

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <linux/fs.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <assert.h>
#include <string.h>
#include "24cXX.h"
```

*/*Hàm i2c_smbus_access() định nghĩa một giao thức tổng quát trong truyền|nhận dữ liệu thông qua chuẩn i2c. Nó bao hàm tất cả những giao thức khác, việc lựa chọn giao thức thực hiện do các tham số trong hàm quy định;*

Hàm bao gồm những tham số căn bản như sau:

int file: Số mô tả tập tin thiết bị được mở trong hệ thống;

char read_write: Cờ cho biết là lệnh đọc hay ghi vào Slave;

__u8 command: Byte dữ liệu muốn ghi vào Slave; Thường thì command có vai trò là địa chỉ thanh ghi trong thiết bị muốn truy xuất;

int size: Là kích thước tính theo bytes của khối dữ liệu muốn truy xuất;

*union i2c_smbus_data *data: Là cấu trúc dữ liệu truyền nhận trong smBus i2c;*/*

```
static inline __s32
i2c_smbus_access(int file, char read_write, __u8 command,int size,
union i2c_smbus_data *data){
      /*Khai báo biến lưu cấu trúc dữ liệu truyền nhận trong ioctl*/
      struct i2c_smbus_ioctl_data args;
      /*Cập nhật thông tin đọc hay ghi*/
      args.read_write = read_write;//read/write
      /*Cập nhật thông tin command cho câu lệnh*/
      args.command = command;
      /*Cập nhật kích thước dữ liệu muốn đọc hay ghi*/
      args.size = size;//size of data
      /*Lưu dữ liệu trả về trong trường hợp đọc, Dữ liệu muốn ghi trong  trường hợp ghi */
      args.data = data;
      return ioctl(file,I2C_SMBUS,&args);
}
/*
  Hàm i2c_smbus_read_byte() đọc về giá trị của ô nhớ có địa chỉ lưu trong thanh ghi địa chỉ của eeprom; Hàm có tham số là int file: Là số mô tả tập tin thiết bị được mở.
*/
static inline __s32
```

```
i2c_smbus_read_byte(int file){
      /*Biến cấu trúc dữ liệu truy xuất từ thiết bị I2C*/
      union i2c_smbus_data data;
      if(i2c_smbus_access(file,I2C_SMBUS_READ,0,I2C_SMBUS_BYTE,&data))
            return -1;
      else
            return 0x0FF & data.byte;
}
```

*/*Hàm i2c_smbus_write_byte() dùng để ghi dữ liệu có chiều dài 1 byte vào eeprom nhằm mục đích là cập nhật giá trị cho thanh ghi địa chỉ trong eeprom;*

Hàm có các tham số như sau:

int file: Số mô tả tập tin thiết bị được mở;

__u8 value: Giá trị muốn ghi vào eeprom, nói dúng hơn là địa chỉ muốn cập nhật/*

```
static inline __s32
i2c_smbus_write_byte(int file, __u8 value)
{
      return i2c_smbus_access(file,I2C_SMBUS_WRITE,value,
                              I2C_SMBUS_BYTE,NULL);
}
```

*/*Hàm i2c_smbus_read_byte_data() có nhiệm vụ đọc giá trị của ô nhớ có địa chỉ cụ thể trong eeprom;*

Hàm có các tham số sau:

int file: Số mô tả tập tin thiết bị được mở;

__u8 command: Địa chỉ ô nhớ muốn đọc;/*

```
static inline __s32
i2c_smbus_read_byte_data(int file, __u8 command){
      union i2c_smbus_data data;
      if(i2c_smbus_access(file,I2C_SMBUS_READ,command,I2C_SMBUS_BYTE_DA
      TA,&data))
            return -1;
      else
            return 0x0FF & data.byte;
```

```
}
```

/ *i2c_smbus_write_byte_data() dùng để ghi 1 byte vào ô nhớ có địa chỉ cụ thể trong eeprom;*

Các tham số cụ thể hàm như sau:

int file: Số mô tả tập tin thiết bị đang dược mở để thao tác;

__u8 command: Địa chỉ thanh ghi muốn ghi dữ liệu;

__u8 value: Giá trị dữ liệu muốn ghi;

**/*

```
static inline __s32
i2c_smbus_write_byte_data(int file, __u8 command,__u8 value){
      union i2c_smbus_data data;
      data.byte = value;
      return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
                              I2C_SMBUS_BYTE_DATA, &data);
}
```

*/*Hàm i2c_smbus_read_word_data() cũng tương tự như hàm i2c_smbus_read_byte_data() nhưng dữ liệu trả về là một word có 32 bits*/*

```
static inline __s32 i2c_smbus_read_word_data(int file, __u8 command)
{
      union i2c_smbus_data data;
      if(i2c_smbus_access(file,I2C_SMBUS_READ,command,I2C_SMBUS_WORD_DA
      TA,&data))
            return -1;
      else
            return 0x0FFFF & data.word;
}
```

*/*Hàm i2c_smbus_write_word_data cũng tương tự như hàm i2c_smbus_write_byte_data() nhưng dữ liệu trả về là một word có 32 bits */*

```
static inline __s32 i2c_smbus_write_word_data(int file, __u8
command,__u16 value)
{
```

```
      union i2c_smbus_data data;
      data.word = value;
      return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
                              I2C_SMBUS_WORD_DATA, &data);
}
```

*/*Định nghĩa hàm ghi 1 byte vào thiết bị struct eeprom *e*/*

```
static int i2c_write_1b(struct eeprom *e, __u8 buf)
{
      int r;
```

*/*we must simulate a plain I2C byte write with SMBus functions*/*

```
      r = i2c_smbus_write_byte(e->fd, buf);
      if(r < 0)
fprintf(stderr,"Error i2c_write_1b: %s\n", strerror(errno));
      usleep(10);
      return r;
}
```

*/*Định nghĩa hàm ghi 2 bytes vào thiết bị struct eeprom *e*/*

```
static int i2c_write_2b(struct eeprom *e, __u8 buf[2])
{
      int r;
```

//we must simulate a plain I2C byte write with SMBus functions

```
      r = i2c_smbus_write_byte_data(e->fd, buf[0], buf[1]);
      if(r < 0)
      fprintf(stderr, "Error i2c_write_2b: %s\n", strerror(errno));
      usleep(10);
      return r;
}
```

*/*Định nghĩa hàm ghi 2 bytes vào thiết bị struct eeprom *e*/*

```
static int i2c_write_3b(struct eeprom *e, __u8 buf[3])
{
      int r;
```

// we must simulate a plain I2C byte write with SMBus functions

//the __u16 data field will be byte swapped by the SMBus protocol

```
	r = i2c_smbus_write_word_data(e->fd, buf[0], buf[2] << 8 |
	buf[1]);
	if(r < 0)
	fprintf(stderr, "Error i2c_write_3b: %s\n", strerror(errno));
	usleep(10);
	return r;
}
```

*/*Định nghĩa hàm mở thiết bị struct eeprom *e*/*

```
int
eeprom_open(char *dev_fqn, int addr, int type, struct eeprom* e)
{
```

*/*Định nghĩa các biến lưu thông tin về hàm hỗ trợ của i2c: funcs;*

Số mô tả tập tin thiết bị fd;

Biến lưu mã lỗi trả về khi truy xuất: r/*

```
	int funcs, fd, r;
```

*/*Xóa các thông tin trong cấu trúc struct eepom *e*/*

```
	e->fd = e->addr = 0;
	e->dev = 0;
```

*/*Mở tập tin thiết bị theo đường dẫn dev_fqn*/*

```
	fd = open(dev_fqn, O_RDWR);
```

*/*Kiểm tra lỗi trong quá trình mở thiết bị/*

```
	if(fd <= 0)
	{
		fprintf(stderr, "Error eeprom_open: %s\n",
		strerror(errno));
		return -1;
	}
```

*/*Gọi hàm ioctl trong driver kiểm tra những hàm trong driver hỗ trợ*/*

```
	if((r = ioctl(fd, I2C_FUNCS, &funcs) < 0))
	{
	fprintf(stderr, "Error eeprom_open: %s\n", strerror(errno));
	return -1;
```

```
	}
	/*Lần lượt kiểm tra những hàm trong driver hỗ trợ*/
CHECK_I2C_FUNC( funcs, I2C_FUNC_SMBUS_READ_BYTE );
CHECK_I2C_FUNC( funcs, I2C_FUNC_SMBUS_WRITE_BYTE );
CHECK_I2C_FUNC( funcs, I2C_FUNC_SMBUS_READ_BYTE_DATA );
CHECK_I2C_FUNC( funcs, I2C_FUNC_SMBUS_WRITE_BYTE_DATA );
CHECK_I2C_FUNC( funcs, I2C_FUNC_SMBUS_READ_WORD_DATA );
CHECK_I2C_FUNC( funcs, I2C_FUNC_SMBUS_WRITE_WORD_DATA );
	/*Gọi hàm ioctl() quy định địa chỉ của thiết bị muốn truy xuất*/
	if( ( r = ioctl(fd, I2C_SLAVE, addr)) < 0)
	{
	fprintf(stderr, "Error eeprom_open: %s\n", strerror(errno));
	return -1;
	}
	/*Cập nhật các thông tin sau khi mở thiết bị thành công vào cấu trúc struct eeprom *e để phục vụ cho những lần truy xuất thiết bị trong những lần tiếp theo*/
	e->fd = fd;
	e->addr = addr;
	e->dev = dev_fqn;
	e->type = type;
	return 0;
}
/*Định nghĩa hàm đóng đóng thiết bị struct eeprom *e*/
int eeprom_close(struct eeprom *e)
{
	/*Gọi giao diện hàm close đóng tập tin thiết bị*/
	close(e->fd);
	/*Khôi phục lại các thông tin ban đầu trong cấu trúc struct eeprom *e*/
	e->fd = -1;
	e->dev = 0;
	e->type = EEPROM_TYPE_UNKNOWN;
	return 0;
}
```

```
/*Các hàm hỗ trợ đọc dữ liệu từ eeprom*/
/*Hàm đọc giá trị trong ô nhớ có địa chỉ mem_addr*/
int eeprom_read_byte(struct eeprom* e, __u16 mem_addr)
{
    /*Biến lưu mã lỗi trả về cho hàm trong trường hợp có lỗi xảy ra  nếu không có lỗi thì rả về giá trị của ô nhớ đọc được*/
    int r;
    /*Gọi hàm ioctl() có chức năng xóa bộ nhớ đệm đọc trong kernel*/
    ioctl(e->fd, BLKFLSBUF);
    /*Kiểm tra từng loại eeprom là loại địa chỉ 8 bits hay 16 bits*/
    /*Trong trường hợp là loại eeprom có địa chỉ 8 bits*/
    if(e->type == EEPROM_TYPE_8BIT_ADDR)
    {
        /*Chỉ lấy 8 bits thấp của mem_addr lưu vào bộ đệm địa chỉ*/
        __u8 buf =  mem_addr & 0x0ff;
        /*Gọi hàm ghi 1 byte địa chỉ vào eeprom*/
        r = i2c_write_1b(e, buf);
    /*Trong trường hợp epprom loại 16 bits địa chỉ*/
    } else if(e->type == EEPROM_TYPE_16BIT_ADDR) {
    /*Định nghĩa mảng 2 biến 8 bits lưu byte địa chỉ thấp và byte địa  chỉ cao*/
    __u8 buf[2] = { (mem_addr >> 8) & 0x0ff, mem_addr & 0x0ff };
/*Ghi mảng 2 bytes địa chỉ vào eeprom bằng cách gọi hàm  i2c_write_2b()*/
    r = i2c_write_2b(e, buf);
/*Các trường hợp còn lại in ra lỗi vì không có loại eeprom được hỗ  trợ*/
    } else {
        fprintf(stderr, "ERR: unknown eeprom type\n");
        return -1;
    }
/*Nếu ghi địa chỉ bị lỗi thì thoát chương trình và in ra mã lỗi*/
    if (r < 0)
        return r;
/*Nếu không có lỗi xảy ra tiếp tục đọc giá trị tại ô nhớ vừa cập nhật  địa chỉ*/
```

```
	r = i2c_smbus_read_byte(e->fd);
	return r;
}
```

*/*Hàm hỗ trợ eeprom ghi một byte vào địa chỉ cụ thể trong eeprom*/*

```
int
eeprom_write_byte(struct eeprom *e, __u16 mem_addr, __u8 data)
{
```

*/*Ban đầu xác định địa chỉ cần ghi dữ liệu*/*

*/*Trong trường hợp eeprom có địa chỉ 8 bits*/*

```
	if(e->type == EEPROM_TYPE_8BIT_ADDR) {
```

*/*Ghi 2 bytes, byte đầu tiên là địa chỉ của ô nhớ; byte tiếp theo là dữ liệu cần ghi*/*

*/*Khai báo mảng lưu 2 bytes thông tin*/*

```
		__u8 buf[2] = { mem_addr & 0x00ff, data };
```

*/*Gọi hàm ghi 2 bytes vào eeprom*/*

```
		return i2c_write_2b(e, buf);
```

*/*Trong trường hợp eeprom có địa chỉ 16 bits*/*

*/*Đầu tiên ghi 2 bytes (16 bits) địa chỉ vào eeprom cuối cùng ghi 1 bytes dữ liệu*/*

```
	} else if(e->type == EEPROM_TYPE_16BIT_ADDR) {
```

*/*Khai báo mảng chứa 3 bytes thông tin*/*

```
		__u8 buf[3] =
	{ (mem_addr >> 8) & 0x00ff, mem_addr & 0x00ff, data };
```

*/*Gọi hàm ghi 3 bytes vào eeprom*/*

```
		return i2c_write_3b(e, buf);
	}
```

*/*Các trường hợp khác không thuộc loại eeprom được hỗ trợ*/*

```
	fprintf(stderr, "ERR: unknown eeprom type\n");
	return -1;
}
```

3. Biên dịch và thực thi chương trình:

Biên dịch chương trình bằng câu lệnh sau:

```
arm-none-linux-gnueabi-gcc eeprom.c –o eeprom
```

Chép chương trình đã biên dịch vào kit và tiến hành kiểm tra các chức năng mà chương trình hỗ trợ.

III. Kết luận:

Đến đây chúng ta đã hoàn thành việc ứng dụng những giao diện hàm trong driver I2C do linux hỗ trợ để điều khiển thành công một thiết bị Slave là eeprom 24c08. Bằng cách áp dụng kỹ thuật tương tự trong chương trình, chúng ta có thể tự mình xây dựng các chương trình khác điều khiển tất cả những thiết bị hoạt động theo chuẩn I2C. Do thời gian có hạn nên chúng tôi chỉ đưa ra một chương trình ví dụ về I2C.

Trong những bài tiếp theo chúng ta cũng sẽ nghiên cứu một driver truyền nối tiếp khác khác đó là driver truyền nối tiếp theo chuẩn UART.

BÀI 9

GIAO TIẾP ĐIỀU KHIỂN
ADC ON CHIP

A- TỔNG QUAN VỀ ADC ON CHIP:

I. Mô tả chung:

a. ***Đặc tính tổng quát:*** *ADC tích hợp trong vi điều khiển AT91SAM9260 có những đặc tính nổi bật sau:*

- 2 kênh chuyển đổi tương tự số;
- Độ phân giải có thể lựa chọn 8 bits hoặc 10 bits;
- Tốc độ chuyển đổi ở chế độ 10 bits là 312K sample/sec; Chuyển đổi theo phương pháp xắp xỉ liên tiếp;
- Có thể cho phép không cho phép đối với từng kênh chuyển đổi;
- Nguồn xung kích có thể lựa chọn: Hardware-Software trigger; Externel trigger pin; Timer/Counter 0 to 2 output;

b. Các chân được sử dụng trong module ADC:

Ký hiệu	***Mô tả***
VDDANA	Nguồn cung cấp cho module analog;
ADVREF	Điện áp tham chiếu;
AD0-AD1	Kênh ngỏ vào tương tự 0 và 1;
ADTRG	Nguồn xung trigger bên ngoài;

Bảng 4-1- Các chân được sử dụng trong module ADC.

II. Đặc tính hoạt động:

ADC sử dụng xung ADC để thực hiện quá trình chuyển đổi của mình. Tốc độ xung chuyển đổi có thể được thay đổi tùy theo mục đích bằng các bits lựa chọn tần số PRESCAL trong thanh ghi Mode Register (ADC_MR). Tốc độ chuyển đổi có thể nằm trong khoảng từ MCK/2 khi PRESCAL=0 đến MCK/128 khi PRESCAL=63.

Giá trị điện áp chuyển đổi nằm trong khoảng từ 0V đến giá trị diện áp trên chân ADVREF và sử dụng phương pháp chuyển đổi ADC xắp xỉ liên tiếp.

Độ phân giải của ADC có thể lựa chọn giữa hai giá trị là 8 bits và 10 bits. Chế độ chuyển đổi ADC 8 bits được cài đặt bằng cách set bit LOWRES trong thanh ghi ADC Mode (ADC_MR). Giá trị chuyển đổi ADC có thể được đọc trong 8 bits thấp của thanh ghi Channel Data Register x (ADC_CDRx). Chế độ chuyển đổi ADC 10 bits được cài dặt bằng cách clear bit LOWRES trong thanh ghi ADC_MR. Lúc này giá trị chuyển đổi ADC được đọc trong 2 bytes thấp và cao của thanh ghi ADC_CDRx. (Trong user interface của module).

Quá trình chuyển dổi ADC được thực hiện theo các bước sau:

- Khởi tạo chân ngõ vào tương tự theo chế độ kéo xuống mức 0;
- Chọn mode hoạt động cho ADC;
- Reset lại ADC bằng cách set bit SWRST trong thanh ghi ADC_CR;
- Cho phép hay không cho phép các kênh chuyển đổi ADC hoạt động;
- Tạo một xung trigger cho bit START trong thanh ghi ADC_CR;
- Chờ cho đến khi xuất hiện mức cao của bit DRDY trong thanh ghi ADC_SR;
- Đọc giá trị chuyển đổi trong các thanh ghi tương ứng với từng kênh chuyển đổi;

Trong đó 3 bước cuối cùng được thực hiện liên tục để cập nhật dữ liệu chuyển đổi.

III. Một số cộng thức quan trọng:

a. Công thức tính tộc độ xung ADC:

ADCClock = MCK/((PRESCAL+1)*2);

b. Công thức tính thời gian khởi động:

Start Up Time = (STARTUP + 1)*8/ADCClock;

c. Công thức tính thời gian lấy mẫu và giữ:

Sample & Hold Time = (SHTIM + 1) / ADCClock;

B- ĐIỀU KHIỂN NHIỆT ĐỘ DÙNG ADC ON CHIP:

I. Phác thảo dự án:

Với những kiến thức về ADC On Chip trong bài trước chúng ta sẽ kết hợp với những lệnh truy xuất thanh ghi trong linux để thiết kế một ứng dụng đơn giản đo nhiệt dộ dùng LM35 hiển thị trên LED 7 đoạn.

a. Yêu cầu dự án:

Yêu cầu của dự án này như sau:

- Người sử dụng nhập vào hai thông số nhiệt độ, nhiệt độ giới hạn trên và nhiệt độ giới hạn dưới;
- Định thời mỗi 1s cập nhật nhiệt độ hiện tại một lần;
- Điều khiển một LED sáng tắt theo quy luật:
  + LED sáng khi nhiệt độ hiện tại lớn hơn hoặc bằng nhiệt độ giới hạn trên;
  + LED tắt khi nhiệt độ hiện tại nhỏ hơn hoặc bằng nhiệt độ giới hạn dưới;

b. Phân công nhiệm vụ:

- *Chương trình driver:*

Chương trình sử dụng 2 driver tương ứng với hai module khác nhau: Driver điều khiển đọc giá tri chuyển đổi ADC trong CHIP và một driver điều khiển quét LED 7 đoạn. Nhiệm vụ cụ thể của từng driver như sau:

- *Driver ADC*: Có tên at91adc_dev.ko

- Khởi tạo ADC trong CHIP;
- Sử dụng giao diện hàm read() để truyền nhiệt độ chuyển đã được chuyển đổi sang user. Khi gọi hàm read() Driver sẽ thực hiện những công việc sau:
  - Kích hoạt bộ chuyển đổi ADC hoạt động;
  - Chờ cho đến khi ADC chuyển đổi xong;
  - Đọc giá trị chuyển đổi;
  - Truyền sang user giá trị đọc được;
- Sử dung giao diện ioctl() để SET và CLEAR một chân gpio để điều khiển động cơ. Giao diện hàm ioctl() có hai số định danh lệnh: ADC_SET_MOTOR và ADC_CLEAR_MOTOR tương ứng với set và clear chân GPIO.

➢ *Driver LED 7 Đoạn*: Có tên là led_seg_dev.ko

Thực hiện quét 8 led 7 đoạn hiển thị những giá trị nhiệt độ: Nhiệt độ giới hạn trên, nhiệt độ giới hạn dưới và nhiệt độ hiện tại;

- Sử dụng giao diện hàm write() để nhận các thông tin từ nhiệt độ từ user hiển thị ra led;
- Sử dụng phương pháp ngắt timer mềm để cập nhật quét LED hiển thị; Phương pháp điều khiển này cũng tương tự như trong bài thực hành led 7 đoạn.

Nhiệt độ hiển thị trên LEDs theo dạng sau:

`<AA> < BB> <CC>`

Trong đó:

`<AA>` là giá trị nhiệt độ giới hạn dưới; (Giới hạn từ 00 đến 99);

`<BB>` là giá trị nhiệt độ hiện tại, khả năng hiển thị từ 00 đến 99;

`<CC>` là giá trị nhiệt độ giới hạn trên; (Giới hạn hiển thị từ 00 đến 99);

- *Chương trình application:* mang tên at91adc_app.c

Xây dựng chương trình application theo hướng có tham số nhập từ người dùng.

Để chạy chương trình, chúng ta nhập câu lệnh thực thi theo cú pháp sau:

`./at91adc_app <AA> <CC>`

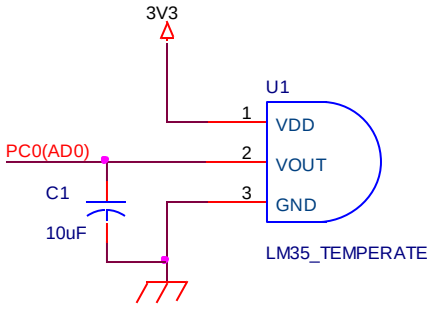
Chương trình application thực hiện những tác vụ:

- Khởi động 2 driver: `at91adc_dev` và `led_seg_dev`;
- Định thời gian 1s cập nhật và so sánh nhiệt độ hiện tại với các nhiệt độ giới hạn trên dưới để điều khiển động cơ cho phù hợp.

II. Thực hiện:

1. Kết nối phần cứng:

Các bạn thực hiện kết nối phần cứng theo sơ đồ sau:



Hình 4-33- Sơ đồ kết nối LM35.

2. Chương trình driver:

- ***Driver ADC:*** Có tên `at91adc_dev.ko`

*/*Khai báo các thư viện cần dùng cho các hàm trong chương trình*/*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <asm/uaccess.h>
#include <mach/gpio.h>
#include <asm/gpio.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
#include <asm/atomic.h>
#include <linux/jiffies.h>
#include <linux/sched.h>
#include <linux/fs.h>
#include <linux/clk.h>
#include <mach/at91_adc.h>

#define DRVNAME      "at91adcDriver"
#define DEVNAME      "at91adcDevice"
```

*/*Định nghĩa số định danh lệnh cho giao diện hàm ioctl của driver ADC*/*

```
#define AT91ADC_DEV_MAGIC  'B'
```

```
/*Số định danh lệnh bật motor*/
#define AT91ADC_SET_MOTOR        _IO(AT91ADC_DEV_MAGIC, 10)
/*Số định danh lệnh tắt motor*/
#define AT91ADC_CLEAR_MOTOR      _IO(AT91ADC_DEV_MAGIC, 11)
/*Định nghĩa chân gpio cho chân điều khiển motor*/
#define GPIO_MOTOR    AT91_PIN_PC15
/*Định nghĩa dạng rút gọn cho lệnh set clear gpio*/
#define at91adc_set_motor()      gpio_set_value(GPIO_MOTOR,1)
#define at91adc_clear_motor()    gpio_set_value(GPIO_MOTOR,0)
/*Khai báo biến con trỏ chứa địa chỉ nền của các thanh ghi thao tác  với ADC on chip*/
void __iomem *at91adc_base;
/*Khai báo biến con trỏ cấu trúc clock cấp xung cho ADC hoạt động*/
struct clk *at91adc_clk;
static atomic_t at91adc_open_cnt = ATOMIC_INIT(1);
/*Hàm hỗ trợ đọc dữ liệu từ ADC */
unsigned int
at91adc_read_current_value(void) {
/*Khai báo mảng chứa 100 giá trị đọc được từ ADC*/
     int current_value[100];
/*Khai báo biến lưu giá trị trung bình của các giá trị trong mảng  current_value[100]*/
     int average=0;
/*Biến điều khiển hỗ trợ cho vòng lặp*/
     int i;
/*Vòng lặp hỗ trợ cho việc đọc các dữ liệu của ADC*/
     for (i = 0; i<100; i++) {

/*Tạo xung trigger cho bit START trong thanh ghi ADC_CR*/
     iowrite32(AT91_ADC_START, (at91adc_base + AT91_ADC_CR));
/*Chờ cho đến khi quá trình chuyển đổi hoàn thành, nghĩa là khi bit  ADC_DRDY được
set lên mức cao*/
while((ioread32(at91adc_base+AT91_ADC_SR)&AT91_ADC_DRDY)==0)
schedule();
```

*/*Chép giá trị đã chuyển đổi chứa trong thanh ghi ADC_CHR(0) vào một phần từ trong mảng bộ nhớ đệm trong kernel*/*

```
	current_value[i]=ioread32(at91adc_base + AT91_ADC_CHR(0));
}
```

*/*Tính tổng các giá trị chứa trong mảng*/*

```
	for (i=0; i<100; i++) {
		average += current_value[i];
	}
```

*/*Trả về giá trị trung bình của các phần tử trong mảng*/*

```
	return average/100;
}
```

*/*Khai báo và định nghĩa giao diện hàm read để cung cấp thông tin giá trị chuyển đổi cho user application*/*

```
static ssize_t at91adc_read (struct file *filp, unsigned char __iomem
buf[], size_t bufsize, loff_t *f_pos)
{
```

*/*Khai báo bộ đệm cho hàm read*/*

```
	unsigned int buf_read[1];
```

*/*Gọi hàm hỗ trợ đọc dữ liệu đã được lập trình ở trên*/*

```
	buf_read [0]=at91adc_read_current_value();
```

*/*Gọi hàm truyền dữ liệu sang user application khi có yêu cầu có kiểm tra lỗi trong quá trình truyền*/*

```
	if (copy_to_user(buf, buf_read, bufsize) != 0) {
	printk ("Error whilst copying to user\n");
	return -1;
	}
	return 2;
}
```

*/*Giao diện hàm ioctl() thực hiện set và clear các chân điều khiển động cơ theo yêu cầu từ user application*/*

```
static int
at91adc_ioctl(struct inode * inode, struct file * file, unsigned int
cmd, unsigned long *arg)
{
      int retval;
      switch (cmd)
      {
/*Trong trường hợp lệnh set chân điều khiển động cơ lên mức 1*/
      case AT91ADC_SET_MOTOR:
            at91adc_set_motor();
            break;
/*Trong trường hợp lệnh set chân điều khiển động cơ xuống mức 0*/
      case AT91ADC_CLEAR_MOTOR:
            at91adc_clear_motor();
            break;
/*Trong trường hợp không có lệnh hỗ trợ trả về mã lỗi*/
      default:
            retval = -EINVAL;
            break;
      }
      return retval;
}
/*Khai báo và định nghĩa giao diện hàm open*/
static int
at91adc_open(struct inode *inode, struct file *file)
{
   int result = 0;
   unsigned int dev_minor = MINOR(inode->i_rdev);
   if (!atomic_dec_and_test(&at91adc_open_cnt)) {
      atomic_dec(&at91adc_open_cnt);
      printk(KERN_ERR DRVNAME ": Device with minor ID %d already in
      use\n", dev_minor);
      result = -EBUSY;
      goto out;
```

```
    }
out:
    return result;
}
```

```
/*Khai báo và định nghĩa giao diện hàm close*/
static int
at91adc_close(struct inode * inode, struct file * file)
{
      smp_mb__before_atomic_inc();
      atomic_inc(&at91adc_open_cnt);
      return 0;
}
struct file_operations at91adc_fops = {
   .read      = at91adc_read,
   .ioctl    = at91adc_ioctl,
   .open      = at91adc_open,
   .release   = at91adc_close,
};
static struct miscdevice at91adc_dev = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name          = "at91adc_dev",
        .fops          = &at91adc_fops,
};

static int __init
at91adc_mod_init(void)
{
      int ret;
      /*Khởi tạo cấu hình làm việc cho ADC on chip*/
      /*Yêu cầu tạo xung cho ADC hoạt động */
      at91adc_clk = clk_get(NULL, /*Device pointer - not required.*/
                            "adc_clk"); /*Clock name*/
      /*Cho phép nguồn xung hoạt động*/
      clk_enable(at91adc_clk);
      /*Định vị con trỏ nền adc vào địa chỉ nền vật lý của các thanh ghi điều khiển
      adc*/
```

```
at91adc_base=ioremap_nocache(AT91SAM9260_BASE_ADC,/*Physical
address*/
64);/*Number of bytes to be mapped*/
/*Kiểm tra lỗi trong quá trình định vị*/
if (at91adc_base == NULL)
{
      printk(KERN_INFO "at91adc: ADC memory mapping failed\n");
      ret = -EACCES;
      goto exit_3;
}
/*Khởi tạo chân gpio ngõ vào analog có điện trở kéo xuống*/
at91_set_A_periph(AT91_PIN_PC0, 0);
/*Reset ADC bằng cách set bit ADC_SWRST*/
iowrite32(AT91_ADC_SWRST, (at91adc_base + AT91_ADC_CR));
/*Cho phép kênh 0 của bộ chuyển đổi ADC hoạt động*/
iowrite32(AT91_ADC_CH(0), (at91adc_base + AT91_ADC_CHER));
/*Cài dặt các chế độ hoạt động cho ADC*/
/*Tần số cực đại = 5MHz = MCK / ((PRESCAL+1) * 2)
/*PRESCAL = ((MCK / 5MHz) / 2) -1 = ((100MHz / 5MHz)/2)-1) = 9
/*Thời gian start up cực đại = 15uS = (STARTUP+1)*8/ADC_CLOCK
/*STARTUP = ((15uS*ADC_CLOK)/8)-1 = ((15uS*5MHz)/8)-1 = 9
/*Minimum hold time = 1.2uS = (SHTIM+1)/ADC_CLOCK
/*SHTIM = (1.2uS*ADC_CLOCK)-1 = (1.2uS*5MHz)-1 = 5, Use 9 to  /*ensure
2uS hold time.
/*Enable sleep mode and hardware trigger from TIOA output from  TC0.*/
iowrite32((AT91_ADC_SHTIM_(9)|AT91_ADC_STARTUP_(9)|
AT91_ADC_PRESCAL_(9)|AT91_ADC_SLEEP|AT91_ADC_TRGEN),
(at91adc_base + AT91_ADC_MR));
/*Khởi tạo chân gpio theo chế độ ngõ ra*/
gpio_request(GPIO_MOTOR, NULL);
at91_set_GPIO_periph (GPIO_MOTOR, 1);
```

```
	gpio_direction_output(GPIO_MOTOR, 0);
	printk(KERN_INFO "at91adc: Loaded module\n");
	printk(KERN_ALERT "Welcome to our at91adc world\n");
	return misc_register(&at91adc_dev);
	exit_3:
	clk_disable(at91adc_clk);
	return ret;
}
```

*/*Khai báo và định nghĩa hàm exit thực hiện khi tháo driver ra khỏi hệ thống*/*

```
static void __exit
at91adc_mod_exit(void)
{
```

*/*Vô hiệu hóa hoạt động của xung clock*/*

```
	clk_disable(at91adc_clk);
```

*/*Giải phóng con trỏ thanh ghi nền điều khiển ADC*/*

```
	iounmap(at91adc_base);
```

*/*In ra thông báo cho người dùng*/*

```
	printk(KERN_ALERT "Goodbye for all best\n");
	printk(KERN_INFO "at91adc: Unloaded module\n");
	misc_deregister(&at91adc_dev);
}
module_init (at91adc_mod_init);
module_exit (at91adc_mod_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Coolwarmboy / OpenWrt");
MODULE_DESCRIPTION("Character device for for generic at91adc driver");
```

- ***Driver LED 7 Đoạn:*** Có tên là led_seg_dev.ko

*/*Driver điều khiển led 7 đoạn đã dược chúng tôi giải thích kỹ trong bài thực hành điều khiển LED 7 đoạn, ở đây chỉ thay đổi các chân gpio và cấu trúc chương trình cho phù hợp với dự án*/*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/init.h>
```

```
#include <mach/at91_tc.h>
#include <asm/gpio.h>
#include <asm/atomic.h>
#include <linux/genhd.h>
#include <linux/miscdevice.h>
#include <asm/uaccess.h>
#include <linux/interrupt.h>
#include <linux/clk.h>
#include <linux/irq.h>
#include <linux/time.h>
#include <linux/jiffies.h>
#include <linux/sched.h>
#include <linux/delay.h>

#define DRVNAME      "led_seg_dev"
#define DEVNAME      "led_seg"

/*-------------Port Control-----------*/
#define P00                AT91_PIN_PB0
#define P01                AT91_PIN_PB2
#define P02                AT91_PIN_PC5
#define P03                AT91_PIN_PC6
#define P04                AT91_PIN_PC7
#define P05                AT91_PIN_PC4
#define P06                AT91_PIN_PB3
#define P07                AT91_PIN_PB1

#define A             AT91_PIN_PB7
#define B             AT91_PIN_PB9
#define C             AT91_PIN_PB11
#define D             AT91_PIN_PA24
#define E             AT91_PIN_PB16
#define F             AT91_PIN_PA23
#define G             AT91_PIN_PB10
```

```
/*Basic commands*/
#define SET_P00()                    gpio_set_value(P00,1)
#define SET_P01()                    gpio_set_value(P01,1)
#define SET_P02()                    gpio_set_value(P02,1)
#define SET_P03()                    gpio_set_value(P03,1)
#define SET_P04()                    gpio_set_value(P04,1)
#define SET_P05()                    gpio_set_value(P05,1)
#define SET_P06()                    gpio_set_value(P06,1)
#define SET_P07()                    gpio_set_value(P07,1)

#define CLEAR_P00()                  gpio_set_value(P00,0)
#define CLEAR_P01()                  gpio_set_value(P01,0)
#define CLEAR_P02()                  gpio_set_value(P02,0)
#define CLEAR_P03()                  gpio_set_value(P03,0)
#define CLEAR_P04()                  gpio_set_value(P04,0)
#define CLEAR_P05()                  gpio_set_value(P05,0)
#define CLEAR_P06()                  gpio_set_value(P06,0)
#define CLEAR_P07()                  gpio_set_value(P07,0)

#define SET_A()                      gpio_set_value(A,1)
#define SET_B()                      gpio_set_value(B,1)
#define SET_C()                      gpio_set_value(C,1)
#define SET_D()                      gpio_set_value(D,1)
#define SET_E()                      gpio_set_value(E,1)
#define SET_F()                      gpio_set_value(F,1)
#define SET_G()                      gpio_set_value(G,1)

#define CLEAR_A()                    gpio_set_value(A,0)
#define CLEAR_B()                    gpio_set_value(B,0)
#define CLEAR_C()                    gpio_set_value(C,0)
#define CLEAR_D()                    gpio_set_value(D,0)
#define CLEAR_E()                    gpio_set_value(E,0)
#define CLEAR_F()                    gpio_set_value(F,0)
#define CLEAR_G()                    gpio_set_value(G,0)
```

```
#define CYCLE    1

#define LED_SEG_DEV_MAGIC  'B'
#define LED_SEG_UPDATE      _IO(LED_SEG_DEV_MAGIC, 12)
```

*/*Counter is 1, if the device is not opened and zero (or less) if opened.*/*

```
static atomic_t led_seg_open_cnt = ATOMIC_INIT(1);
unsigned char DataDisplay[8]={0,1,2,3,4,5,6,7};
unsigned char   SevSegCode[] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92,
0x82,0xF8,0x80,0x90,0x3F,0x77,0xFF};
int i;
```

*/*Khai báo cấu trúc timer "mềm"*/*

```
struct timer_list my_timer;
void led_seg_write_data_active_led(char data)
{
      (data&(1<<0))? SET_P00():CLEAR_P00();
      (data&(1<<1))? SET_P01():CLEAR_P01();
      (data&(1<<2))? SET_P02():CLEAR_P02();
      (data&(1<<3))? SET_P03():CLEAR_P03();
      (data&(1<<4))? SET_P04():CLEAR_P04();
      (data&(1<<5))? SET_P05():CLEAR_P05();
      (data&(1<<6))? SET_P06():CLEAR_P06();
      (data&(1<<7))? SET_P07():CLEAR_P07();
}
void led_seg_write_data_led(char data)
{
      (data&(1<<0))? SET_A():CLEAR_A();
      (data&(1<<1))? SET_B():CLEAR_B();
      (data&(1<<2))? SET_C():CLEAR_C();
      (data&(1<<3))? SET_D():CLEAR_D();
      (data&(1<<4))? SET_E():CLEAR_E();
      (data&(1<<5))? SET_F():CLEAR_F();
      (data&(1<<6))? SET_G():CLEAR_G();
```

```
}
void active_led_choice(char number) {
      led_seg_write_data_active_led(~(1<<(number)));
}
void data_led_stransmitt (char data) {
      led_seg_write_data_led (SevSegCode[data]);
}
void sweep_led_time_display(int hh, int mm, int ss) {
      DataDisplay[0] = ss%10;
      DataDisplay[1] = ss/10;
      DataDisplay[2] = 10;
      DataDisplay[3] = mm%10;
      DataDisplay[4] = mm/10;
      DataDisplay[5] = 10;
      DataDisplay[6] = hh%10;
      DataDisplay[7] = hh/10;
}
/*Hàm thực thi quét LED khi có ngắt xảy ra*/
void my_timer_function (unsigned long data) {
      /*Xuất mã 7 đoạn thứ của dữ liệu thứ i ra LED 7 đoạn thứ i*/
      data_led_stransmitt(DataDisplay[i]);
      /*Cho phép LED 7 đoạn thứ i tích cực*/
      active_led_choice(i);
      /*Tăng biến i lên 1 đơn vị*/
      i++;
      /*Giới hạn số LED hiển thị là 8*/
      if (i==8) i = 0;
      /*Cài đặt lại thời gian ngắt cho timer là CYCLE=1(ms)*/
      mod_timer (&my_timer, jiffies + CYCLE);
}
/*buf[0] nhiệt độ giới hạn trên; buf[1] nhiệt độ giới hạn dưới*/
static ssize_t led_seg_write (struct file *filp, unsigned char __iomem
buf[], size_t bufsize, loff_t *f_pos)
```

```
{
      unsigned char write_buf[2];
      int write_size = 0;
      int i;
      if (copy_from_user (write_buf, buf, bufsize) != 0) {
            return -EFAULT;
      } else {
            write_size = bufsize;
      }
```

*/*Cập nhật hiển thị LED*/*

```
      DataDisplay[0] = write_buf[0] % 10;
      DataDisplay[1] = write_buf[0] / 10;
      DataDisplay[2] = 12; 
```

*/*Ký tự khoảng trắng trong LED 7 đoạn*/*

```
      DataDisplay[5] = 12; 
```

*/*Ký tự khoảng trắng trong LED 7 đoạn*/*

```
      DataDisplay[6] = write_buf[1] % 10;
      DataDisplay[7] = write_buf[1] / 10;

      return write_size;
}
```

*/*Khai báo và định nghĩa giao diện hàm ioctl() phục vụ cho hàm cập nhật dữ liệu hiện tại hiển thị trên led*/*

```
static int
led_seg_ioctl(struct inode * inode, struct file * file, unsigned int
cmd, unsigned int arg)
{
   int retval;
   switch (cmd)
   {
```

*/*Trong trường hợp lệnh update dữ liệu hiển thị trên led 7 đoạn*/*

```
   case LED_SEG_UPDATE:
```

*/*Giải mã số nguyên từ 00 đến 99 sang 2 số BCD chục và đơn vị, cập nhật thông tin cho mảng dữ liệu hiển thị*/*

```
            DataDisplay[3] = arg % 10;
```

```
            DataDisplay[4] = arg / 10;
    break;
    default:
            retval = -EINVAL;
            break;
    }
    return retval;
}

static int
led_seg_open(struct inode *inode, struct file *file)
{
    int result = 0;
    unsigned int dev_minor = MINOR(inode->i_rdev);
    if (!atomic_dec_and_test(&led_seg_open_cnt)) {
      atomic_inc(&led_seg_open_cnt);
      printk(KERN_ERR DRVNAME ": Device with minor ID %d already in
      use\n", dev_minor);
      result = -EBUSY;
      goto out;
    }
out:
    return result;
}

static int
led_seg_close(struct inode * inode, struct file * file)
{
    smp_mb__before_atomic_inc();
    atomic_inc(&led_seg_open_cnt);
    return 0;
}

struct file_operations led_seg_fops = {
    .write  = led_seg_write,
```

```
    .ioctl   = led_seg_ioctl,
    .open    = led_seg_open,
    .release = led_seg_close,
};

static struct miscdevice led_seg_dev = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name          = "led_seg_dev",
        .fops          = &led_seg_fops,
};

static int __init
led_seg_mod_init(void)
{
      int ret=0;
      gpio_request (P00, NULL);
      gpio_request (P01, NULL);
      gpio_request (P02, NULL);
      gpio_request (P03, NULL);
      gpio_request (P04, NULL);
      gpio_request (P05, NULL);
      gpio_request (P06, NULL);
      gpio_request (P07, NULL);

      at91_set_GPIO_periph (P00, 1);
      at91_set_GPIO_periph (P01, 1);
      at91_set_GPIO_periph (P02, 1);
      at91_set_GPIO_periph (P03, 1);
      at91_set_GPIO_periph (P04, 1);
      at91_set_GPIO_periph (P05, 1);
      at91_set_GPIO_periph (P06, 1);
      at91_set_GPIO_periph (P07, 1);

     gpio_direction_output(P00, 0);
      gpio_direction_output(P01, 0);
```

```
gpio_direction_output(P02, 0);
gpio_direction_output(P03, 0);
gpio_direction_output(P04, 0);
gpio_direction_output(P05, 0);
gpio_direction_output(P06, 0);
gpio_direction_output(P07, 0);

gpio_request (A, NULL);
gpio_request (B, NULL);
gpio_request (C, NULL);
gpio_request (D, NULL);
gpio_request (E, NULL);
gpio_request (F, NULL);
gpio_request (G, NULL);

at91_set_GPIO_periph (A, 1);
at91_set_GPIO_periph (B, 1);
at91_set_GPIO_periph (C, 1);
at91_set_GPIO_periph (D, 1);
at91_set_GPIO_periph (E, 1);
at91_set_GPIO_periph (F, 1);
at91_set_GPIO_periph (G, 1);
gpio_direction_output(A, 0);
gpio_direction_output(B, 0);
gpio_direction_output(C, 0);
gpio_direction_output(D, 0);
gpio_direction_output(E, 0);
gpio_direction_output(F, 0);
gpio_direction_output(G, 0);
```

*/*Khởi tạo timer "mềm" ngắt với chu kỳ thời gian là 1ms*/*

*/*Khởi tạo timer từ cấu trúc timer đã định nghĩa*/*

```
init_timer (&my_timer);
```

*/*Cài dặt thời gian sinh ra ngắt là CYCLE=1 (ms)*/*

```
my_timer.expires = jiffies  + CYCLE;
```

```
	/*Cung cấp dữ liệu cho hàm phục vụ ngắt*/
	my_timer.data = 0;
	/*Gán con trỏ hàm phục vụ ngắt cho timer khi có ngắt xảy ra*/
	my_timer.function = my_timer_function;
	/*Thực hiện cài đặt timer vào hệ thống*/
	add_timer (&my_timer);

	misc_register(&led_seg_dev);
	printk(KERN_INFO "led_seg: Loaded module\n");
	return ret;
}
static void __exit
led_seg_mod_exit(void)
{
/*Xóa đồng bộ timer ra khỏi hệ thống để không làm ảnh hưởng đến các  lần khởi tạo timer tiếp theo*/
	del_timer_sync(&my_timer);
	misc_deregister(&led_seg_dev);
	printk(KERN_INFO "led_seg: Unloaded module\n");
}
module_init (led_seg_mod_init);
module_exit (led_seg_mod_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("coolwarmboy");
MODULE_DESCRIPTION("Character device for for generic gpio api");
```

3. *Chương trình application:*

*/*Khai báo thư viện dùng cho các hàm trong chương trình*/*

```
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <fcntl.h>
#include <linux/ioctl.h>

#include <sys/types.h>
#include <sys/stat.h>
```

*/*Khai báo các số định danh lệnh cần dùng cho driver ADC và LED 7 đoạn*/*

*/*Số định danh lệnh dùng cho giao diện hàm ioctl() của driver at91adc_dev*/*

```
#define AT91ADC_DEV_MAGIC  'B'
#define AT91ADC_SET_MOTOR            _IO(AT91ADC_DEV_MAGIC, 10)
#define AT91ADC_CLEAR_MOTOR          _IO(AT91ADC_DEV_MAGIC, 11)
```

*/*Số định danh lệnh dùng cho giao diện hàm ioctl() của driver led_seg_dev*/*

```
#define LED_SEG_DEV_MAGIC  'B'
#define LED_SEG_UPDATE               _IO(LED_SEG_DEV_MAGIC, 12)
```

*/*Khai báo các biến để lưu các nhiệt độ giới hạn nhập từ người dùng*/*

```
unsigned int HighestTemp, LowestTemp;
```

*/*Các biến lưu số mô tả tập tin thiết bị khi được mở*/*

```
int fd_at91adc_dev, fd_led_seg_dev;
```

*/*Chương trình con cập nhật nhiệt độ hiện tại từ driver adc và truyền sang driver LED 7 đoạn*/*

```
void check_update_cur_temp (void) {
```

*/*Bộ nhớ đệm lưu giá trị trả về từ driver adc*/*

```
     unsigned char current_value[2];
```

*/*Biến lưu nhiệt độ hiện tại sau khi chuyển đổi*/*

```
     int cur_temp;
```

*/*Biến lưu giá trị trả về từ driver dưới dạng int*/*

```
        int cur_value;
/*Gọi giao diện hàm read đọc thông tin chuyển đổi từ driver adc*/
        read(fd_at91adc_dev, current_value, 2);
/*Chuyển byte thấp và byte cao của bộ đệm thành số int*/
        cur_value = current_value[1]*256 + current_value[0];
/*Chuyển đổi thông tin chuyển đổi từ driver adc sang nhiệt độ
Số 0.31867 được tính bằng công thức sau: Vref/(2^10-1)
trong đó Vref là giá trị đo được từ chân VREFP; 10 là độ phân giải 10 bits của ADC on chip*/
        cur_temp = cur_value * 0.31867;
/*Gọi giao diện hàm ioctl() của driver LED 7 đoạn cập nhật hiển thị nhiệt độ hiện tại lên LEDs*/
        ioctl(fd_led_seg_dev, LED_SEG_UPDATE, cur_temp);
/*So sánh nhiệt độ hiện tại với các nhiệt độ giới hạn để tắt|mở động cơ cho thích hợp*/
        if (cur_temp >= HighestTemp) {
/*Trong trường hợp nhiệt độ hiện tại vượt quá nhiệt độ giới hạn trên*/
                /*In ra thông báo cho người dùng*/
                printf("Motor is turned on\n");
                /*Gọi hàm ioctl() của driver adc set pin điều khiển motor*/
                ioctl(fd_at91adc_dev, AT91ADC_SET_MOTOR);
        } else if (cur_temp <= LowestTemp) {
                /*Trong trường hợp nhiệt độ hiện tại xuống thấp dưới nhiệt độ giới hạn dưới*/
                printf("Motor is turned off\n");
                /*Gọi giao diện hàm ioctl() clear pin điều khiển motor*/
                ioctl(fd_at91adc_dev, AT91ADC_CLEAR_MOTOR);
        }
}
```

```
/*Chương trình chính khai báo dưới dạng có tham số nhập vào từ người  dùng */
int
main(int argc, char **argv)
{
      /*Bộ đệm lưu giá trị cần truyền sang driver LED 7 đoạn hiển thị hai nhiệt độ
      giới hạn trên và dưới */
      /*buf[0] highest temp; buf[1] lowest temp*/
      unsigned char buf_write[2] = {34, 35};
      /*Mở hai tập tin thiết bị trước khi thao tác*/
      fd_at91adc_dev = open("/dev/at91adc_dev",O_RDWR);
      fd_led_seg_dev = open("/dev/led_seg_dev",O_RDWR);
      /*Cập nhật các thông tin về nhiệt độ giới hạn*/
      HighestTemp = atoi(argv[2]);     /*Highest Temperature value*/
      LowestTemp = atoi(argv[1]);      /*Lowest Temperature value*/
      /*Lưu các giá trị giới hạn vào bộ đệm ghi sang driver hiển thị LED 7 đoạn*/
      buf_write[0] = HighestTemp;
      buf_write[1] = LowestTemp;
      /*Gọi giao diện hàm write() ghi dữ liệu trong bộ đệm sang driver  LED 7 đoạn*/
      write(fd_led_seg_dev,buf_write,2);
      /*Dùng vòng lặp liên tục cập nhật điều khiển động cơ, hiển thị nhiệt độ ra LED
      với chu kỳ là 1s*/
      while (1) {
            check_update_cur_temp();
            sleep(1);
      }
      exit (0);
}
```

III. Kết luận và bài tập:

a. Kết luận:

Bên cạnh dùng IC 0809 làm thiết bị chuyển đổi tương tự-số chúng ta cũng có thể dùng module ADC tích hợp sẵn trong CHIP vi điều khiển. Như vậy sẽ tiếc kiệm chi phí và rút gọn được số chân IO điều khiển để dùng vào công việc khác. Bên cạnh đó dùng ADC on chip chúng ta có thể thực hiện được nhiều chức năng tiện lợi hơn, độ chính xác có thể linh động thay đổi tùy theo yêu cầu chương trình ứng dụng.

Trong bài này chúng ta đã sửa lại driver hiển thị LED 7 đoạn ứng dụng timer mềm để quét LED đồng thời đã ứng dụng phương pháp lập trình character device driver vào viết driver cho ADC on chip. Kết hợp 2 driver để điều khiển và hiển thị thông tin trên LED 7 đoạn. Người học có thể thay đổi dự án hiển thị trên những thiết bị khác hay ứng dụng driver vào việc lấy mẫu tín hiệu, ... lúc đó ứng dụng sẽ phức tạp hơn. Do thời gian thực hiện cuốn giáo trình này có hạn nên chúng tôi không đi sâu vào nghiên cứu những ứng dụng này.

b. Bài tập:

1. Làm lại yêu cầu của dự án trên, nhưng hiển thị các thông tin các nhiệt độ lên LCD. (Áp dụng LCD driver đã viết trong bài thực hành LCD);
2. Tối ưu hóa dự án trên theo yêu cầu sau:

- Thông tin về các nhiệt độ giới hạn được lưu trong EEPROM 24C08;
- Khi chương trình được mở, nó sẽ cập nhật hai giá trị nhiệt độ giới hạn từ EEPROM;
- Thông tin về các giá trị nhiệt độ nếu thay đổi sẽ được cập nhật vào EEPROM;

BÀI 10

GIAO TIẾP ĐIỀU KHIỂN
UART ON CHIP

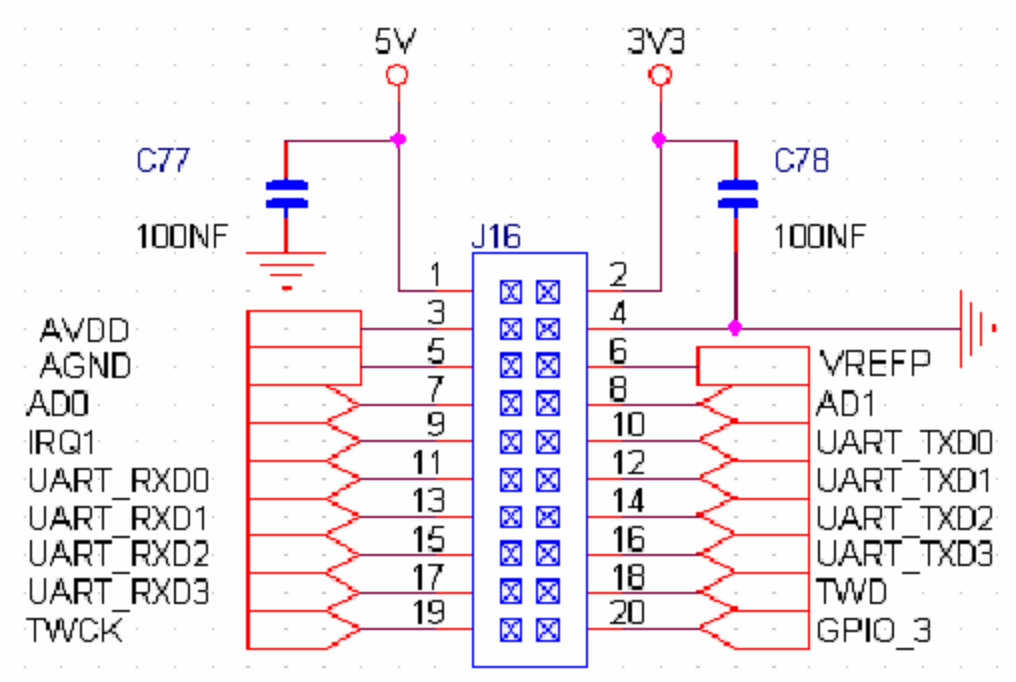
A- KHÁI QUÁT VỀ UART TRONG KM9260:

I. Tổng quan:

Trong các bài trước chúng ta đã học hai chuẩn truyền nối tiếp đó là chuẩn I2C và chuẩn 1-Wire. Trong đó chuẩn I2C được truyền theo cơ chế đồng bộ, nghĩa là có xung đồng bộ giữ nhịp trong quá trình truyền|nhận thông tin. Chuẩn 1-Wire do truyền nhân dữ liệu chỉ thông qua một dây nên không có xung đồng bộ giữ nhịp trong quá trình truyền. Thay vào đó là dựa vào trục thời gian để xác định các bit dữ liệu. Trong phần này chúng ta sẽ nghiên cứu chuẩn truyền UART (Truyền nối tiếp bất đồng bộ) được tích hợp trong module UART của vi điều khiển AT91SAM9260. Cũng giống như chuẩn 1-Wire chuẩn UART phân biệt các bit truyền|nhận theo thời gian, nghĩa là mỗi vị trí bit sẽ tương ứng với từng thời điểm khác nhau. Nhưng khác với chuẩn 1-Wire, chuẩn UART có 2 dây, một dây truyền TXD và một dây nhận RXD nên có thể truyền và nhận đồng thời nhưng 1-Wire thì chỉ có một dây bao hàm cả hai nhiệm vụ truyền và nhận nên hay công việc này phải được tiến hành khác thời điểm.

(Các kiến thức sau đây về UART trong KM9260 được trình bày đựa vào tài liệu đính kèm với kit nếu có vấn đề thắc mắc các bạn có thể tham khảo trong tài liệu này)

Chip vi điều khiển AT91SAM9260 có 6 kênh UART trong đó bao gồm một kênh DBUG (Được nối với thiết bị đầu cuối chính) và 5 kênh truyền nhận dữ liệu nối tiếp phụ: UART0 đến UART4. Trong kit KM9260, các kênh này đều được đưa ra ngoài để kết nối với các thiết bị ngoại vi. Sơ đồ ngõ ra của các kênh UART này như sau:



Hình 4-34- Sơ đồ kết nối Jump 16 của kit km9260, các chân kết nối UART.

Quyển sách này chỉ chú trọng vào điều khiển các cổng UART thông qua giao diện tập tin thiết bị được hệ thống linux hỗ trợ sẵn chứ không nhằm vào điều khiển trực tiếp các thanh ghi trong phần cứng. Tương tác điều khiển truyền nhận và các lệnh cấu hình đều được thực hiên thông qua các lệnh đọc và ghi tập tin (các giao diện hàm `read()` `write()` và `ioctl()`). Để thực hiện công việc này, chúng ta có hai cách. Hoặc là dùng những câu lệnh trong môi trường shell hoặc là dùng những lệnh C được định nghĩa trong thư viện termios.h.

II. Thao tác UART trong shell:

Để điều khiển được module UART, chương trình ứng dụng cần phải tác động lên các tập tin thiết bị tương ứng với module đó. Các module UART được quy định tên như sau:

UART	Device File
DBGU	ttyS0
UART0	ttyS1
UART1	ttyS2
UART2	ttyS3
UART3	ttyS4
UART4	ttyS5

Bảng 4-2- Các moduel UART được quy định trong linux.

Tất cả các tập tin thiết bị đều chứa trong thư mục `/dev/` của cấu trúc root file system. Để xem danh sách thiết bị UART được hỗ trợ trong hệ thống, chúng ta dùng câu lệnh shell sau:

```
# ls /dev/ | grep "ttyS"
```

Khi đó linux sẽ liệt kê các tập tin thiết bị có 5 ký tự đầu dạng "`ttyS`":

```
ttyS0
ttyS1
ttyS2
```

Như vậy hệ thống chỉ hỗ trợ 3 module UART. Chúng ta có thể đăng ký thêm thiết bị ttyS3, ttyS4, ttyS5 bằng cách thêm những dòng sau trong tập tin *board-sam9260ek.c* của cấu trúc mã nguồn kernel nằm trong thư mục `/arch/arm/mach-at91/`:

```
/* UART2 on  ttyS3. (Rx & Tx only) */
at91_registe_uart(AT91SAM9260_ID_US2, 3, 0);
 /* UART3 on  ttyS4. (Rx & Tx only) */
at91_register_uart(AT91SAM9260_ID_US3, 4, 0);
/* UART4 on  ttyS5. (Rx & Tx only) */
 at91_register_uart(AT91SAM9260_ID_US4, 5, 0);
```

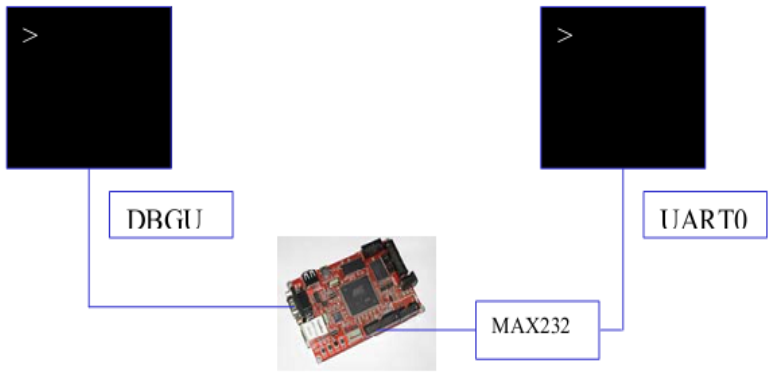
bắt đầu từ dòng 67.

Công việc tiếp theo là biên dịch và nạp lại kernel cho hệ thống. Các bước biên dịch và nạp kernel được trình bày trong phần lập trình nhúng căn bản.

Tiếp theo chúng ta sẽ tiến hành kiểm tra hoạt động của một kệnh UART mà hệ thống hỗ trợ. Để cụ thể, chúng ta sẽ tiến hành kiểm tra hoạt động của kênh UART0 (Tập tin thiết bị trong hệ thống có tên là ttyS1).

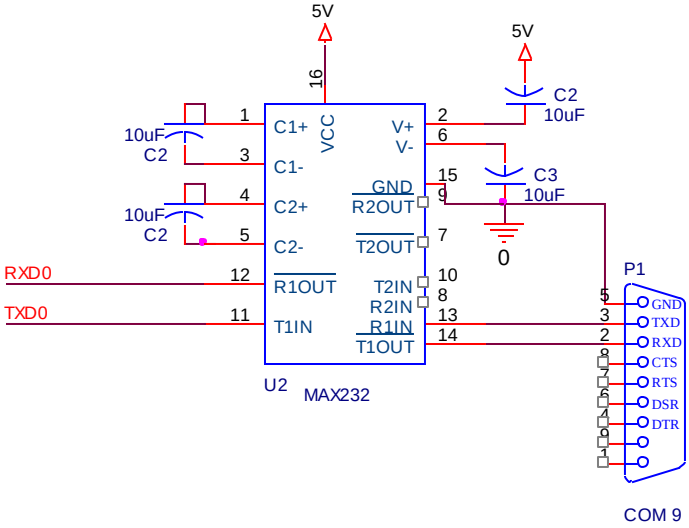
- Kết nối phần cứng theo sơ đồ sau:

a. Sơ đồ khối:



Hình 4-34- Sơ đồ khối giao tiếp giữa 2 kênh UART.

b. Sơ đồ nguyên lý phần MAX232 và UART0:



Hình 4-35- Sơ đồ kết nối kênh UART0.

Với cấu hình kết nối trên ta thấy xuất hiện cùng lúc 2 terminal (thiết bị đầu cuối), terminal thứ nhất là DBUG được nối với máy tính thông qua cổng COM 9 được thiết kế trên kit (Hoặc được đăng nhập thông qua cổng ethenet RJ45) terminal thứ hai là UART0 được nối với máy tính thông qua kênh UART0 (ttyS1). Ta có thể mở cùng lúc hai màn hình console putty, mỗi lần mở chúng ta sẽ cung cấp những thông số cần thiết. Console thứ nhất cùa DBUG được mở bằng COM1 tốc độ 115200 bps, console thứ hai UART0 được được mở bằng COM 2 tốc độ 9600bps.

***Lưu ý: Kênh UART0 kết nối với cổng COM 2 của máy tính phải thông qua IC chuyển đổi mức điện áp MAX232 như sơ đồ mạch nguyên lý trên, không thể nào được nối trực tiếp.*

Như vậy 2 console này có thể truyền dữ liệu qua lại với nhau. Console DBUG đóng vai trò là Master ra lệnh truyền hay nhận chuỗi thông tin qua|từ console UART0 đóng vai trò là Slave. Ta sẽ dùng những câu lệnh shell để thực hiện hai chức năng này từ console Master DBUG.

***Console DBUG ngoài việc đăng nhập qua cổng COM, nó còn có thể đăng nhập thông qua cổng ethenet. Như vậy console Master được hiểu là nơi đăng nhập và thực thi chương trình.*

Trước tiên kiểm tra tốc độ baund hiện tại của kênh UART0 bằng câu lệnh shell sau (lưu ý lệnh này chỉ thực hiện trên console DBUG).

```
# stty –F /dev/ttyS1
```

Lúc này linux sẽ cho ra kết quả sau:

```
speed 9600 baunds
```

Tốc độ này đúng với tốc độ đã quy định trong bài tập tiến hành, tuy nhiên nếu muốn thay đổi tốc độ baund (chẳng hạn là 115200 bps) ta có thể dùng câu lệnh sau:

```
# stty –F /dev/ttyS1 115200
```

Để truyền một chuỗi thông tin từ console Master sang console Slave, ta thực hiện lệnh sau:

```
# echo "Hello Slave" > /dev/ttyS1
```

Lúc này bên Slave sẽ nhận được và hiển thị ra thông tin:

```
Hello Slave
```

Để nhận chuỗi thông tin từ console Slave, ta thực hiện câu lệnh:

```
# cat /dev/ttyS1

_
```

Lúc này console Master sẽ đợi cho đến khi bên phía Slave phát ký tự sau đó mới in ra màn hình và cứ tiếp tục đợi thông tin. Trong quá trình đợi Master không làm bất kỳ công việc nào khác, đôi khi việc này không được mong đợi. Để quá trình đợi vẫn được tiến

hành mà Master có thể làm những công việc khác, chúng ta dùng lệnh "&" để đem công việc đợi vào hậu tiến trình.

```
# cat /dev/ttyS1&
#
```

***Những câu lệnh trên có thể áp dụng cho tất cả các kênh UART, ngay cả kênh DBUG.*

Dùng những lệnh trong môi trường shell có ưu điểm là đơn giản, dễ thực hiện nhưng ít khi được sử dụng rộng rãi vì bản thân những lệnh này là những chương trình được lập trình sẵn chứa trong thư viện chung của linux. Khi gọi những lệnh này, linux sẽ tốn thời gian khởi tạo tiến trình mới hơn nữa cấu trúc lệnh không linh động. Trong bài sau chúng ta sẽ nghiên cứu một phương pháp khác để giao tiếp với các kênh UART là dùng những giao diện hàm C trong user application tương tác với driver UART.

B- CÁC HÀM HỖ TRỢ UART TRONG USER APPLICATION:

I. Thư viện termios.h và cấu trúc struct termios:

Trong chương trình ứng dụng (user application) không thể tác động trực tiếp đến các thanh ghi để điều khiển UART mà phải thông qua driver. Driver của UART là những tập tin thiết bị có dạng "ttySx" trong thư mục /dev/ của cấu trúc root file system. Các tập tin thiết bị này cũng được sử dụng giống như những tập tin thiết bị khác, cũng dùng các giao diện hàm read(), write() và ioctl() để tương tác truyền nhận thông tin và cấu hình thiết bị. Bên cạnh đó chúng ta còn dùng những hàm chuyên dụng được hỗ trợ trong thư viện termios.h.

Thư viên termios.h là một tập hợp những hàm Unix API dùng cho terminal IO. Các hàm trong thư viện hỗ trợ 2 chế độ hoạt động.

1. Canonical mode:

Thích hợp cho các thiết bị giao tiếp theo kiểu line-by-line . Kiểu giao tiếp line-by-line là kiểu giao tiếp có đơn vị thông tin là từng dòng ký tự. Mỗi dòng được phân biệt nhau bởi một trong những ký tự đặc biệt trong bảng mã ascii như: Ký tự xuống dòng (NL), ký tự kết thúc tập tin (EOF) hoặc ký tự kết thúc một dòng thông tin (EOL). Các ký tự này là những ký tự đặc biệt không được hiển thị trong dòng thông tin. Chính vì thế, khi gọi hàm nhận thông tin từ một thiết bị UART nào đó, hàm chỉ trả về chuỗi thông tin khi thiết bị truyền phát ra một trong những ký tự đặc biệt trên.

2. Non-canonical mode:

Dùng cho kiểu giao tiếp các ký tự riêng lẽ. Trong kiểu giao tiếp này, một đơn vị thông tin được quy định bởi hai thông số, MIN và TIME. Hai tham số này là hai trường thông tin của mảng c_cc (sẽ được trình bày sau) trong cấu trúc struct termios. MIN được hiểu là số bytes tối thiểu trong một đơn vị thông tin. TIME là thời gian truyền nhận (đơn vị 0.1s) giữa hai bytes liên tiếp. Một đơn vị thông tin được hoàn thành khi hai tham số MIN và TIME được thõa mãn.

3. Cấu trúc termios:

Cấu trúc struct termios là nơi trung gian chứa các thông số của cổng truyền nối tiếp UART. Đây là nơi chứa các thông số hiện tại của kênh UART và đồng thời cũng là nơi

chứa các thông số mới cần cập nhật cho kênh UART này. Cấu trúc struct termios bao gồm những trường thông tin chứa giá trị của các cờ định dạng cổng UART.

Cấu trúc struct termios được định nghĩa như sau:

```
struct termios {
  tcflag_t c_iflag, /*input specific flags (bitmask) */
  tcflag_t c_oflag, /*output specific flags (bitmask) */
  tcflag_t c_cflag, /*control flags (bitmask) */
  tcflag_t c_lflag, /*local flags (bitmask) */
  cc_t     c_cc[NCCS], /* special characters */ };
```

Chúng ta sẽ lần lượt giải thích từng nội dung của cấu trúc này theo hướng nghiên cứu cách sử dụng những chức năng mà mỗi trường thông tin hỗ trợ trong việc lập trình kênh UART.

***Những chức năng được trình bày là những chức năng phổ biến được sử dụng trong những ứng dụng đơn giản. Nhằm mục đích phục vụ cho các bài tập ví dụ trong những bài thực hành tiếp theo.*

a. Thông tin về các thông số điều khiển:

Được định nghĩa trong trường thông tin c_cflag, bao gồm các thông số điều khiển tốc độ baud, số bits truyền nhận, bit parity, stop bits và những tham số điều khiển dòng dữ liệu bằng phần cứng thông qua hai bits RTS và CTS. Mỗi thông số được định nghĩa dưới dạng cờ thông tin. Các cờ được tổng hợp lại với nhau thông qua phép toán logic OR để tạo thành thông tin tổng quát cho thiết bị.

- *Cài đặt tốc độ baud:*

Thông tin về tốc độ baud được quy định bởi các cờ sau:

Ký hiệu	Ý nghĩa
CBAUD	Bit mask for baud rate
B0	0 baud (drop DTR)
B50	50 baud
B75	75 baud
B110	110 baud
B134	134.5 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud
B2400	2400 baud
B4800	4800 baud
B9600	9600 baud
B19200	19200 baud
B38400	38400 baud
B57600	57,600 baud
B76800	76,800 baud
B115200	115,200 baud

Bảng 4-3- Các tốc độ baud được hỗ trợ trong linux.

Chúng ta sử dụng 4 hàm được hỗ trợ trong thư viện termios.h là `cfsetospeed()`, `cfsetispeed()` và `tcgetattr()` và `tcsetattr()` để cài đặt tốc độ baud như sau:

```
/*Định nghĩa cấu trúc termios để lưu thuộc tính hiện tại của kênh UART*/
struct termios option;
/*Lấy về các thông số hiện tại của kệnh UART*/
tcgetattr(uart_fd, &options);
/*Cài đặt tốc độ baud của kênh hiện tại thành 115200 baud*/
cfsetispeed(&options, B115200);
cfsetospeed(&options, B115200);
/*Cho phép nhận và cài đặt chế độ local*/
options.c_cflag |= (CLOCAL | CREAD);
/*Gọi hàm cài đặt các thông số trên cho kênh UART hiện tại có số định danh lưu trong uart_fd*/
tcsetattr(uart_fd, TCSANOW, &options);
```

Trong đoạn chương trình trên, chúng ta dùng hàm `tcgetattr()` để lấy về các thông số hiện tại của kênh `uart_fd` và lưu vào các trường thông tin trong cấu trúc `struct termios` được định nghĩa từ trước như sau: `struct termios options`; Sau khi cài đặt tốc độ `baud`, kích hoạt chế độ local và cho phép nhận chúng ta gọi hàm `tcsetattr()` để cài đặt các thuộc tính mới chứa trong cấu trúc `termios options` vào kênh uart hiện tại là `uart_fd`. Chúng ta sử dụng tham số `TCSANOW` để cài đặt các thông số ngay lập tức mà không cần phải chờ kết thúc việc truyền nhận trước đó. Chúng ta có 3 tham số có thể cài đặt cho hàm `tcsetattr()` tùy theo yêu cầu sử dụng:

Ký hiệu	Ý nghĩa
TCSANOW	Cập nhật ngay lập tức không chờ kết thúc truyền nhận dữ liệu
TCSADRAIN	Cập nhật sẽ được thực thi ngay sau khi việc truyền nhận kết thúc.
TCSAFLUSH	Lưu trữ tạm thời các bộ đệm truyền nhận và cập nhật các thông số mới

Bảng 4-4- Các tham số cài đặt cho hàm tcsetattr().

- *Cài đặt số bits truyền nhận:*

Thông tin về số bit truyền nhận được quy định trong các cờ sau:

Ký hiệu	Ý nghĩa
CSIZE	Các bits che cờ quy định chiều dài ký tự.
CS5	5 data bits
CS6	6 data bits
CS7	7 data bits
CS8	8 data bits

Bảng 4-5- Chiều dài dữ dữ liệu truyền nhận UART trong linux.

Để cài đặt số bits truyền nhận, chúng ta tiến hành theo các bước sau:

*/*Cài đặt kênh uart_fd theo chế độ truyền nhật 8 bits dữ liệu*/*

*/*Bước này được tiến hành sau khi cập nhật các thông số hiện tại của kệnh uart_fd*/*

*/*Che bits quy định chiều dài dữ liệu truyền nhận*/*

```
options.c_cflag &= ~CSIZE;
/*Quy định chiều dài dữ liệu truyền nhận là 8 bits*/
options.c_cflag |= CS8;
/*Các bước tiếp theo có thể cài đặt các thông số khác hoặc gọi  hàm cập nhật các thông số cho kênh uart_fd*/
...
```

Trong đoạn chương trình trên, đầu tiên ta tiến hành che các bits quy định chiều dài dữ liệu của kệnh UART cũ bằng phép toán logic AND (&) giữa trường thông tin c_cflag với dữ liệu đảo của cờ che thông tin chiều dài dữ liệu CSIZE. Cuối cùng, quy định chiều dài dữ liệu mới bằng cách dùng phép toán OR giữa trường thông tin c_cflag với cờ quy định chiều dài dữ liệu, ở đây là 8 bits CS8.

- *Cài đặt kiểm tra parity:*

Các thông tin quy định bit parity được quy định bởi những bits cờ sau:

Ký hiệu	Ý nghĩa
PARENB	Cho phép parity, ngược lại không cho phép.
PARODD	Parity lẽ, ngược lại parity chẵn.

Bảng 4-6- Các tham số cài đặt kiểm tra parity của kênh UART.

Do không có các hàm hỗ trợ cài đặt parity trong thư viện termios.h nên chúng ta phải sử dụng các hàm logic để cài đặt các thông số này. Các bước được tiến hành theo câu lệnh sau:

- Trong trường hợp không có bit kiểm tra parity:

```
options.c_cflag &= ~PARENB; //Vô hiệu hóa parity
options.c_cflag &= ~CSTOPB; //1 bit stop
options.c_cflag &= ~CSIZE; //Che các bits kích thước
options.c_cflag |= CS8; //Kích thước dữ liệu là 8 bits
```

- Trong trường hợp parity chẵn:

```
options.c_cflag |= PARENB;  //Cho phép kiểm tra parity
options.c_cflag &= ~PARODD; //Chuyển sang chế độ parity chẵn
options.c_cflag &= ~CSTOPB; //Một bit stop
options.c_cflag &= ~CSIZE; //Che bits quy định kích thước
options.c_cflag |= CS7; //Kích thước dữ liệu là 7 bits
```

- Trong trường hợp parity lẽ:

```
options.c_cflag |= PARENB; //Cho phép kiểm tra parity
options.c_cflag |= PARODD; //Chuyển sang chế độ parity lẽ
options.c_cflag &= ~CSTOPB; //Một bit stop
options.c_cflag &= ~CSIZE;//Che các bits quy định kích thước
options.c_cflag |= CS7; //Kích thước dữ liệu là 7 bits
```

- Trong trương hợp space parity:

```
options.c_cflag &= ~PARENB; //Vô hiệu hóa kiểm tra parity
options.c_cflag &= ~CSTOPB; //Một bit stop
options.c_cflag &= ~CSIZE; //Che các bits quy định kích thước
options.c_cflag |= CS8; //Kích thước dữ liệu là 7 bits
```

- *Cài đặt điều khiển dòng dữ liệu bằng phần cứng:*

Một số phiên bản của UNIX hỗ trợ chế độ kiểm soát dòng dữ liệu thông qua phần cứng. Đó là dựa vào tín hiệu phát ra từ hai chân RTS và CTS cùa chuẩn RS232. Cờ điều khiển kích hoạt hay vô hiệu hóa chế độ này là: CNEW_RTSCTS hay CRTSCTS.

Nếu muốn kích hoạt chế độ này ta, dùng dòng lệnh sau:

```
options.c_cflag |= CNEW_RTSCTS; //Ta cũng có thể thay bằng CRTSCTS
```

Nếu muốn vô hiệu hóa chế độ này, ta dùng dùng dòng lệnh sau:

```
options.c_cflag &= ~CNEW_RTSCTS; //Ta cũng có thể thay bằng CRTSCTS
```

b. Thông tin về các thông số cục bộ:

Các thông số tổng quát này được quy định bởi các cờ trong trường thông tin c_lflag trong cấu trúc struct termios. Các thông số này quy định cách quản lý luồng thông tin ngõ vào của driver UART. Nghĩa là luồng thông tin ngõ vào được quy định là chế độ canonical hay raw(non-canonical).

- *Chọn chế độ ngõ vào là canonical:*

Chế độ ngõ vào canonical được giải thích kỹ trong phần đầu nên sẽ không nhắc lại. Các cờ thao tác trong chế độ này là:

Constant	Description
ICANON	Enable canonical input (else raw)

ECHO	Enable echoing of input character
ECHOE	Echo erase character as BS-SP-BS

Bảng 4-7- Các cờ liên quan đến chế độ canonical.

Để chọn chế độ canonical input, ta dùng dòng lệnh sau:

```
options.c_lflag |= (ICANON | ECHO | ECHOE);
```

- *Chọn chế độ ngõ vào là raw(non-canonical):*

Trong chế độ này ta thường vô hiệu hóa chức năng so sánh và thực thi tín hiệu ngắt từ luồng thông tin bằng cờ ISIG và vô hiệu hóa chức năng hiển thị các luồng thông tin bằng hai cờ ECHO, ECHOE.

Ta thực hiện dòng lệnh sau:

```
options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
```

c. Thông tin về các thông số ngõ vào:

Thông tin này được quy định trong trường c_iflag của cấu trúc termios dùng để điều khiển tất cả các quá trình xử lý tín hiệu ngõ vào. Các chức năng chính của trường thông tin này bao gồm:

- *Cài đặt các thông số parity ngõ vào:*

Để cài đặt chế độ kiểm tra parity ngõ vào trước tiên ta phải cho phép chế độ kiểm tra parity trong trường thông tin điều khiển c_cflag (cờ PARENB). Những cờ liên quan đến công việc này là:

Ký hiệu	Ý nghĩa
INPCK	Enable parity check
IGNPAR	Ignore parity errors
PARMRK	Mark parity errors
ISTRIP	Strip parity bits

Bảng 4-8- Các cờ liên quan đến cài đặt parity ngõ vào.

Ta tiến hành dòng lệnh sau để cho phép cờ INPCK và ISTRIP hoạt động:

```
options.c_iflag |= (INPCK | ISTRIP);
```

Trong đó, cờ ISTRIP có chức năng: Nếu được bật thị giá trị của mỗi byte dữ liệu là 7 bits đầu tiên, ngược lại giá trị của mỗi byte dữ liệu được chứa trong 8 bits.

- *Cài chế độ điều khiển dòng dữ liệu bằng phần mềm:*

Điều khiển dòng dữ liệu bằng phần mềm nghĩa là dòng thông tin được điều khiển bắt đầu hay kết thúc được qui định bởi hai ký tự đặc biệt START và STOP. Các cờ qui định chế độ này là:

Ký hiệu	Ý nghĩa
IXON	Enable software flow control (outgoing)
IXOFF	Enable software flow control (incoming)
IXANY	Allow any character to start flow again

Bảng 4-9- Các cờ liên quan đến điều khiển dòng dữ liệu bằng phần mềm.

Để bật chế độ này, chúng ta tiến hành dòng lệnh sau:

```
options.c_iflag |= (IXON | IXOFF | IXANY);
```

Để tắt chế độ này chúng ta tiến hành dòng lệnh sau:

```
options.c_iflag &= ~(IXON | IXOFF | IXANY);
```

d. Thông tin về các thông số ngõ ra:

Được điều khiển bởi trường thông tin c_oflag dùng để qui định luồng thông tin ngõ ra. Chức năng chính của trường thông tin này là chọn chế độ ngõ ra dạng canonical hoặc raw. Cờ duy nhất để điều khiển chức năng này là OPOST (Postprocess output (not set = raw output).

Để chọn chế độ ngõ ra là process output, ta tiến hành câu lệnh sau:

```
options.c_oflag |= OPOST;
```

Để chọn chế độ ngõ ra là raw, ta tiến hành câu lệnh sau:

```
options.c_oflag &= ~OPOST;
```

e. Thông tin về các ký tự đặc biệt:

Giá trị mã ascii của các ký tự đặc biệt được chứa trong các phần tử của trường thông tin mảng c_cc của cấu trúc termios. Ý nghĩa của các phần tử này được qui định trong bảng sau:

Ký hiệu phần tử		
Canonical	**Non-canonical**	
Mode	**Mode**	**Mô tả**
VEOF		EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR	VINTR	INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

Bảng 4-10- Các thành phần trong mảng thông tin c_cc.

Đoạn chương trình sau mô tả cách qui định các ký tự đặc biệt trong UART:

```
newtio.c_cc[VINTR]    = 0; /*Ctrl-c*/
newtio.c_cc[VQUIT]    = 0; /*Ctrl-\*/
newtio.c_cc[VERASE]   = 0; /*del*/
newtio.c_cc[VKILL]    = 0; /*@*/
newtio.c_cc[VEOF]     = 0; /*Ctrl-d*/
/*inter-character timer unused*/
newtio.c_cc[VTIME]    = 0;
/*blocking read until 1 character arrives*/
newtio.c_cc[VMIN]     = 1;
```

```
newtio.c_cc[VSWTC]    = 0; /* '\0' */
newtio.c_cc[VSTART]   = 0; /* Ctrl-q */
newtio.c_cc[VSTOP]    = 0; /* Ctrl-s */
newtio.c_cc[VSUSP]    = 0; /* Ctrl-z */
newtio.c_cc[VEOL]     = 0; /* '\0' */
newtio.c_cc[VREPRINT] = 0; /* Ctrl-r */
newtio.c_cc[VDISCARD] = 0; /* Ctrl-u */
newtio.c_cc[VWERASE]  = 0; /* Ctrl-w */
newtio.c_cc[VLNEXT]   = 0; /* Ctrl-v */
newtio.c_cc[VEOL2]    = 0; /* '\0' */
```

Các giá trị trên chỉ là tham khảo, người học cần lựa chọn những ký tự điều khiển thích hợp sau cho không ảnh hưởng đến hoạt động của chương trình ứng dụng.

II. Các hàm hỗ trợ UART trong termios.h:

a. ***Giao diện hàm open:*** Có cú pháp như sau:

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

Hàm `open()` dùng để mở tập tin thiết bị có đường dẫn trong `const char *path`, chế độ truy xuất được quy định trong tham số int oflag. Giá trị trả về là số mô tả tập tin nếu quá trình thực thi không bị lỗi, ngược lại trả về giá trị âm.

Ví dụ:

Để mở tập tin thiết bị UART có đường dẫn `/dev/ttyS1`; các chế độ truy xuất là cho phép đọc ghi và không thuộc màn hình console ta thự hiện lệnh sau:

```
/*Khai báo biến lưu số mô tả tập tin trả về*/
int fd;
/*Gọi giao diện hàm open() để mở tập tin thiết bị*/
fd = open(/dev/ttyS1, O_RDWR  | O_NOCTTY);
/*Kiểm tra lỗi trong quá trình mở tập tin thiết bị*/
if (fd<0) { printf ("Open device ttyS0 failed\n); exit(1);}
```

b. ***Giao diện hàm close():*** Cú pháp của hàm này như sau:

```
#include <unistd.h>
int close(fd);
```

Hàm có chức năng đóng tập tin thiết bị đang mở trong hệ thống có số mô tả tập tin là fd. Kết quả trả về là 0 nếu đóng thành công, là -1 nếu đóng không thành công.

c. ***Giao diện hàm read():*** Có cú pháp như sau:

```
#include <unistd.h>
ssize_t read(itn fd, void *buf, size_t count);
```

Hàm dùng để đọc về thông tin từ thiết bị `UART` có số mô tả tập tin là `fd`, lưu vào vùng nhớ đệm có con trỏ buf, kích thước dữ liệu muốn đọc là `count (bytes)`. Hàm trả về giá trị là số bytes đọc được nếu quá trình đọc thành công. Nếu `count` lớn hơn `SSIZE_MAX` thì giá trị trả về không xác định. Nếu đọc không thành công giá trị trả về là -1.

d. ***Giao diện hàm write():*** Hàm có cú pháp như sau:

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

Hàm dùng để ghi lượng thông tin có chiều dài `count (bytes)` chứa trong con trỏ `buf` vào thiết bị UART có số mô tả tập tin thiết bị là `fd`.

Giá trị trả về cảu hàm:

- Là số byte(s) ghi hoàn tất nếu quá trình ghi thành công.
- Là -1 nếu quá trình ghi bị lỗi.

e. ***Hàm tcgetattr():*** Hàm có cú pháp như sau:

```
#include <termios.h>
int tcgetattr (int fd, struct termios *termios_p);
```

Hàm tcgetattr dùng lấy về thuộc tính của kênh UART đang mở có số mô tả tập tin thiết bị là `fd`. Thông tin về các thuộc tính được lưu và con trỏ cấu trúc `termios` là `termios_p`.

Hàm trả về giá trị dương nếu hàm thực thi thành công, ngược lại sẽ trả về giá trị âm.

f. ***Hàm tcsetattr():*** Hàm có cú pháp như sau:

```
#include <termios.h>
int tcsetattr(int fd, int optional-actions, const struct termios
*termios_p);
```

Hàm làm nhiệm vụ cập nhật thông tin cấu hình kênh UART chứa trong con trỏ cấu trúc `termios termios_p` vào kênh UART đang mở có số mô tả tập tin là fd với yêu cầu chứa trong biến `int optional-actions`.

C- BÀI THỰC HÀNH UART 1:

I. Phác thảo dự án:

Sau khi nghiên hai nội dung đầu tiên trong bài UART, người học đã lĩnh hội được những kiến thức có liên quan đến cách quản lý và lắp đặt module UART trong kit KM9260 về phần cứng cũng như phần mềm. Truyền UART trong Linux được phân làm hai chế độ truyền, truyền theo định hướng dòng ký tự (canonical mode) và theo định hướng từng khối ký tự (non-canonical mode). Người học sẽ thực hành truyền dữ liệu sử dụng module UART lần lượt theo hai chế độ trên.

a. Yêu cầu dự án:

Trong dự án này, ta sẽ thực hành truyền dữ liệu theo chế độ canonical (định hướng dòng ký tự). Thông tin sẽ được truyền từ màn hình terminal_1 (sử dụng kênh UART0-ttyS1) đến terminal_0 (sử dụng kênh UART DEBUG-ttyS0) theo chế độ từng dòng, mỗi dòng được truyền đi khi từ terminal_1 ta nhấn phím Enter.

b. Phân công nhiệm vụ:

- ***Driver:***

Sử dụng driver UART được hỗ trợ trong Linux với chức năng và những lệnh hỗ trợ được giải thích trong hai phần trước. Sau đây sẽ nhắc lại một một số những kiến thức về chế độ truyền nối tiếp UART theo chế độ canonical-Truyền nối tiếp theo định hướng dòng ký tự.

Kiểu giao tiếp line-by-line là kiểu giao tiếp có đơn vị thông tin là từng dòng ký tự. Mỗi dòng được phân biệt nhau bởi một trong những ký tự đặc biệt trong bảng mã ascii như: Ký tự xuống dòng (NL), ký tự kết thúc tập tin (EOF) hoặc ký tự kết thúc một dòng thông tin (EOL). Các ký tự này là những ký tự đặc biệt không được hiển thị trong dòng thông tin. Chính vì thế, khi gọi hàm nhận thông tin từ một thiết bị UART nào đó, hàm chỉ trả về chuỗi thông tin khi thiết-bị-truyền phát ra một trong những ký tự đặc biệt trên.

- ***Application:*** Chương trình mang tên `test_serial_app_1.c`

Chương trình ứng dụng có những nhiệm vụ chính như sau:

- Mở và cấu hình driver điều khiển kênh UART0 (ttyS1) trong Linux kernel, với:
  + Tốc độ baud: 115200 bps;
  + Chế độ truyền theo dòng ký tự (canonical mode);
  + Số bits truyền nhận 8 bits;
  + Kết nối trực tiếp không qua modem;

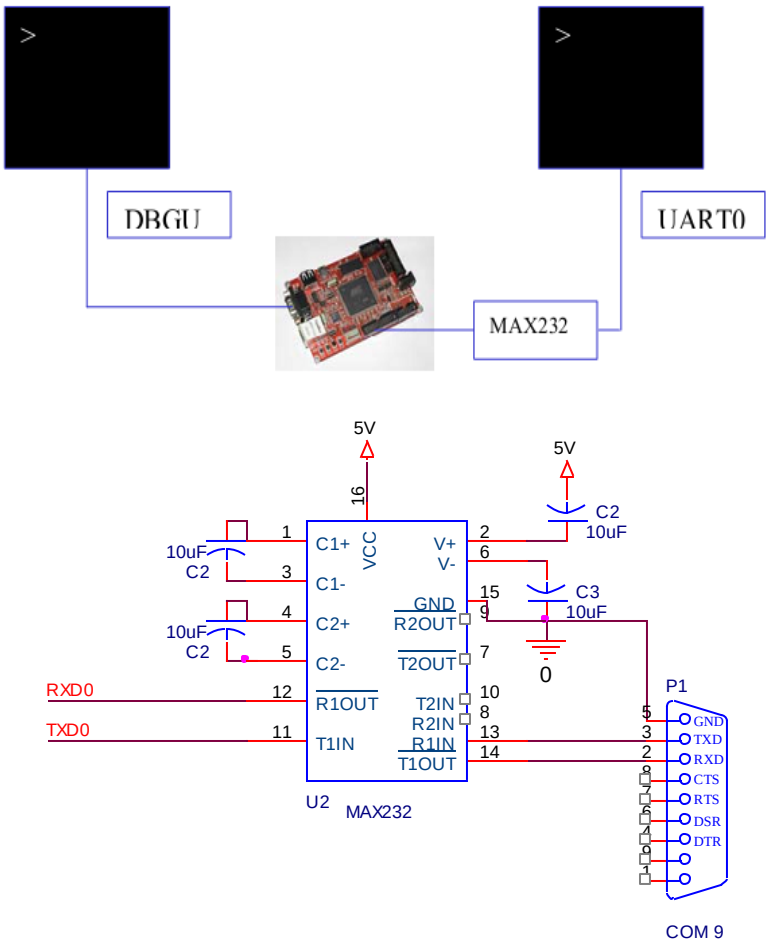
+ Cho phép nhận dữ liệu từ bên ngoài;
+ Bỏ qua kiểm tra parity;
+ Và một số những thông tin cấu hình khác sẽ được giải thích trong quá trình viết mã lệnh.

- Gọi giao diện hàm read() để đọc chuỗi ký tự (có chiều dài tối đa 255 ký tự) được phát ra từ terminal_1 (ttyS1). Nếu chưa có thông tin, chương trình sẽ thực hiện đợi tại giao diện hàm read(). Khi nhận được chuỗi thông tin, nếu không phải ký tự đầu tiên của chuỗi là 'z' thì chương trình sẽ tiếp tục đọc chuỗi thông tin tiếp theo từ terminal_1.

II. Thực hiện:

a. Kết nối phần cứng:

Người học kết nối phần cứng theo yêu cầu sau:



Hình 4-36- Sơ đồ kết nối giao tiếp giữa hai kênh UART0 và DEBUG

b. Chương trình application: Chương trình được mang tên test_serial_app_1.c

```
/*Khai báo thư viện cho các hàm cần dùng trong chương trình*/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
/*Thư viện phục vụ giao tiếp với các thiết bị đầu cuối*/
#include <termios.h>
#include <stdio.h>

/*
Định nghĩa hằng số lưu tốc độ baud cần cập nhật cho kênh UART là 115200 bps.
*/
#define BAUDRATE B115200
/*
Định nghĩa đến đường dẫn tập tin thiết bị là ttyS1 (kênh UART1)
*/
#define MODEMDEVICE "/dev/ttyS1"
/*
Định nghĩa các giá trị TRUE và FALSE sử dụng trong giá trị điều kiện.
*/
#define FALSE 0
#define TRUE 1
/*
Biến lưu cờ thoát khỏi vòng lặp, giá trị mặc định là FALSE;
*/
volatile int STOP=FALSE;
/*
Khai báo hàm main() dưới dạng không có tham số và giá trị trả về;
*/
main(){
/*
Biến lưu số mô tả tập tin thiết bị, giá trị mã lỗi trả về khi thực thi lệnh;
*/
int fd,c, res;
```

```
/*
Khai báo cấu trúc termios bao gồm hai biến lưu thông tin cấu hình kênh UART
trước(oldtio) và sau(newtio) khi chỉnh sửa;
*/
struct termios oldtio,newtio;
/*
Bộ nhớ đệm dùng để chứa dữ liệu đọc về từ kênh UART. Bộ nhớ đệm này có kích
thước tối đa là 255 ký tự;
*/
char buf[255];
/*
Mở tập tin thiết bị driver cho UART với chế độ truy xuất là read|write và không
chịu sự điều khiển của tty.
Trả về số mô tả tập tin lưu trong biến fd
*/
fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
/*
Kiểm tra lỗi trong quá trình mở tập tin thiết bị.
*/
if (fd <0) {
/*Thông báo cho người lập trình biết khi có lỗi xảy ra*/
      printf("Error while opening the device");
/*Trả về mã lỗi là -1*/
      exit(-1);
}
/*Lưu thuộc tính của kênh UART vào biến cấu trúc oldtio bằng cách gọi hàm
tcgetattr()*/
tcgetattr(fd,&oldtio);
/*Xóa các giá trị thuộc tính cũ trong biến lưu thuộc tính của kênh UART là newtio
bằng cách dùng hàm bzero().
Hàm bzero() được định nghĩa trong thư viện strings.h có cú pháp sử dụng như
sau:
#include <strings.h>
```

*void bzero(void *s, size_t n);*

Làm nhiệm vụ đặt n giá trị 0 vào khối có địa chỉ bắt đầu được quy định bởi con trỏ s.

*/

```
bzero(&newtio, sizeof(newtio));
```

/*

Khai báo lại trường thông tin c_cflag của cấu trúc termios newtio để xác định các thuộc tính truyền nhận UART.

BAUDRATE: Cài đặt thông số tốc độ baud được quy định trong phần định nghĩa các hằng số, ngoài ra chúng ta có thể cài đặt tốc độ baud bằng hai hàm cfsetispeed và cfsetospeed.

CS8 : Quy định chiều dài dữ liệu truyền và nhận theo cấu trúc 8n1 (8bit,no parity,1 stopbit)

CLOCAL : Cài đặt chế độ kết nối theo kiểu cục bộ không qua modem

CREAD : Cho phép nhận thông tin

*/

*/*Thực hiện cài đặt các thông số trường c_cflag*/*

```
newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
```

/*

Cài đặt các thông số cho ngõ vào, nhận dữ liệu từ terminal_1

IGNPAR : Bỏ qua kiểm tra parity trong chế độ ngõ vào

ICRNL : Chuyển đổi ký tự CR sang NL (Trong trường hợp ngược lại ký tự CR không kết thúc dòng thông tin)

Ký tự NL có thể được vô hiệu hóa trong chế độ UART raw (Không có xử lý ký tự điều khiển từ ngõ vào.

*/

```
newtio.c_iflag = IGNPAR | ICRNL;
```

/*

Chế độ ngõ ra theo định hướng ký tự raw mode;

*/

```
newtio.c_oflag = 0;
```

/*

ICANON : Cài đặt chế độ truyền nhận theo chế độ canonical;

/

```
newtio.c_lflag = ICANON;
```

/

Cài đặt không sử dụng những ký tự điều khiển xuất hiện trên dòng thông tin ngõ vào;

*Những ký tự được đặt trong /**/ là những ký tự mặc định được qui định cho từng trường hợp;*

Ta cài đặt giá trị 0 tương ứng với không sử dụng các ký tự điều khiển tương ứng trong từng trường hợp;

Ý nghĩa của các hằng số quy định trong trường thông tin c_cc của cấu trúc termios được trình bày trong phần lý thuyết serial;

/

```
newtio.c_cc[VINTR]    = 0;     /* Ctrl-c */
newtio.c_cc[VQUIT]    = 0;     /* Ctrl-\ */
newtio.c_cc[VERASE]   = 0;     /* del */
newtio.c_cc[VKILL]    = 0;     /* @ */
newtio.c_cc[VEOF]     = 4;     /* Ctrl-d */
newtio.c_cc[VTIME]    = 0;     /* inter-character timer unused */
newtio.c_cc[VMIN]     = 1;     /* blocking read until 1 character
arrives */
newtio.c_cc[VSWTC]    = 0;     /* '\0' */
newtio.c_cc[VSTART]   = 0;     /* Ctrl-q */
newtio.c_cc[VSTOP]    = 0;     /* Ctrl-s */
newtio.c_cc[VSUSP]    = 0;     /* Ctrl-z */
newtio.c_cc[VEOL]     = 0;     /* '\0' */
newtio.c_cc[VREPRINT] = 0;     /* Ctrl-r */
newtio.c_cc[VDISCARD] = 0;     /* Ctrl-u */
newtio.c_cc[VWERASE]  = 0;     /* Ctrl-w */
newtio.c_cc[VLNEXT]   = 0;     /* Ctrl-v */
newtio.c_cc[VEOL2]    = 0;     /* '\0' */
```

*/*Xóa thông tin được lưu trước đó trong đường truyền*/*

```
tcflush(fd, TCIFLUSH);
```

```
/*Gọi hàm tcsetattr() để cài đặt những thông tin quy định ở phần trên vào kênh UART*/
tcsetattr(fd,TCSANOW,&newtio);

/*
Vòng lặp thực hiện nhiệm vụ của chương trình application.
Thông tin được nhận và truyền theo từng dòng.
Chương trình chỉ kết thúc khi ký tự đầu tiên trong chuỗi thông tin nhận được là 'z'.
Chuỗi thông tin tối đa có chiều dài 255 ký tự.
*/
while (STOP==FALSE) {
/*
Gọi giao diện hàm read() để nhận chuỗi thông tin nhận được từ terminal_1.
Khi quá trình đọc chưa thành công thì chương trình sẽ chờ tại giao diện hàm read() cho đến khi nhận thành công.
Quá trình đọc chỉ thành công khi nhận được ký tự điều khiển kết thúc việc truyền dữ liệu từ terminal_1.
Khi quá trình đọc thành công thì giao diện hàm read() sẽ trả về số ký tự đọc được từ terminal_1 (bao gồm cả những ký tự điều khiển).
*/
res = read(fd,buf,255);
/*Cho ký tự cuối cùng trong mảng bằng 0 để quy định là một chuỗi ký tự hoàn chỉnh khi đó ta có thể sử dụng hàm printf() để in chuỗi thông tin này ra*/
buf[res]=0;
/*Sử dụng hàm printf() để xuất chuỗi thông tin đọc được từ terminal_1 và số ký tự đọc được.*/
printf(":%s:%d\n", buf, res);
/*Kiểm tra điều kiện kết thúc chương trình. CHương trình kết thúc khi ký tự đầu tiên trong chuỗi thông tin nhận được là ký tự 'z'*/
if (buf[0]=='z') STOP=TRUE;
}
/*Phục hồi lại trạng thái trước đó từ oldtio
```

```
Các thông số được kích hoạt ngay lập tức*/
tcsetattr(fd,TCSANOW,&oldtio);
}
```

c. Thực thi chương trình:

- Biên dịch chương trình ứng dụng bằng câu lệnh sau:

  `arm-none-linux-gnueabi-gcc test_serial_app_1.c -o test_serial_app_1`

  Tập tin ngõ ra có tên test_serial_app_1 được chép vào kit.

- Chương trình test_serial_app_1 được thực thi trong kit bằng dòng lệnh:

  */*Chuyển đổi mode cho tập tin vừa chép vào hệ thống thực thi trong mọi user */*

  `chmode 777 test_serial_app_1`

  */*Thực thi chương trình*/*

  `./test_serial_app_1`

  (Lúc này chương trình sẽ chờ cho đến khi có dòng thông tin nhận được)

- Tại terminal_1 ta nhập dòng thông tin sau:

  "Hello terminal_0↵ "

- Tại terminal_0 ta thấy xuất hiện dòng thông tin sau:

  `: Hello terminal_0:19`

  _

  (Chương trình chờ cho đến khi xuất hiện ký tự 'z' tại vị trí đầu tiên của dòng thông tin tiếp theo)

III. Kết luận và bài tập:

a. Kết luận:

Đến đây người học đã biết cách sử dụng UART trong chế độ truyền dữ liệu theo định hướng dòng ký tự. Ưu điểm của chế độ này là thông tin được truyền liên tục theo từng khối làm giảm sự ngắt quãng giữa các đơn vị dữ liệu trong quá trình truyền nhưng chế độ này không phù hợp trong các ứng dụng đòi hỏi truyền các số liệu riêng biệt. Đối với các ứng dụng như thế thì ta thường dùng chế độ truyền dữ liệu theo định hướng ký tự (raw-mode). Trong bài thực hành tiếp theo, người học sẽ tiếp cận với UART theo chế độ truyền này.

b. Bài tập:

1. Người học tự nghiên cứu và sửa những thông số của chương trình, biên dịch và kiểm tra kết quả thực thi. Từ đó rút ra được những kinh nghiệm lập trình riêng cho mình.

   (Gợi ý: Người học có thể sửa các thông số sau:

- Số bit truyền nhận;
- Tốc độ truyền nhận;
- Quy định các ký tự điều khiển nào đó trong trường thông tin c_cc bằng các ký tự trong bảng mã ascii, truyền dữ liệu từ terminal_1 sang terminal_0 trong đó có sử dụng các ký tự điều khiển này sau đó quan sát kết quả thực thi để biết ý nghĩa thực thi của các ký tự điều khiển trong trường thông tin c_cc cảu cấu trúc termios.
- ...

2. Viết chương trình ứng dụng chạy trên user điều khiển led sáng tắt theo yêu cầu sau:

- Thông tin điều khiển được truyền từ terminal_1 (terminal không thuộc console).
- Nếu nhận được chuỗi thông tin “tat led↵” thì led được kết nối với chân gpio làm ngõ ra tắt.
- Nếu nhận đuọc chuỗi thông tin “sang led↵” thì led trên sáng.

  (Người học có thể nghĩ ra các ứng dụng khác làm bài tập rèn luyện, chẳng hạn xuất chuỗi thông tin nhận được từ terminal_1 ra màn hình LCD 16x2)

D-BÀI THỰC HÀNH UART 2:

I. Phác thảo dự án:

Trong bài thực hành này chúng ta sẽ tiến hành diều khiển kênh UART0 với chế độ truyền theo từng khối ký tự (non-canonical mode). Chế độ truyền UART theo khối được trình bày trong phần lý thuyết UART thế nhưng chúng ta sẽ nhắc lại một cách khái quát những kiến thức quan trong sau đây.

Chế độ truyền UART theo non-canonical mode dùng cho kiểu giao tiếp các khối ký tự riêng lẽ. Trong kiểu giao tiếp này, một đơn vị (khối) thông tin được quy định bởi hai thông số, MIN và TIME. Hai tham số này là hai trường thông tin của mảng c_cc (sẽ được trình bày sau) trong cấu trúc struct termios. MIN được hiểu là số bytes tối thiểu trong một đơn vị thông tin. TIME là thời gian truyền nhận tối đa (đơn vị 0.1s) giữa hai bytes liên tiếp. Một đơn vị thông tin được hoàn thành khi hai tham số MIN và TIME được thõa mãn.

Như vậy có 4 trường hợp xảy ra:

- *Trong trường hợp MIN>0 và TIME=0,*

  Trường hợp này tham số TIME=0 nghĩa là UART không quan tâm đến thời gian tối đa giữa 2 bytes liên tiếp, thời gian nhận giữa hai bytes có thể dài ngắn tùy thích. Kênh UART sử dụng chỉ quan tâm đến số bytes quy định bởi tham số MIN. Khi số bytes nhận bằng với số MIN thì kênh UART sẽ thông báo quá trình đọc thành công.

- *Trong trường hợp MIN=0 và TIME>0,*

  Trong trường hợp này thì giá trị MIN không được quan tâm. Quá trình đọc chỉ thành công khi nhận được một byte dữ liệu hoặc sau khi gọi giao diện hàm read() TIME*0.1s. Lưu ý, lúc này thời gian TIME không còn là thời gian nhận 2 bytes liên tiếp mà trở thành thời gian tính từ khi gọi hàm read() cho đến khi quá trình đọc thành công.

- *Trong trường hợp MIN>0 và TIME>0,*

  Trong trường hợp này, TIME được xem như là timer được kích hoạt kể từ khi nhận được 1 byte đầu tiên. Timer này được reset sau mỗi lần nhận được một byte thông tin. Nếu số lượng thông tin nhận được đạt đến giá trị MIN(bytes) trước khi thời gian TIME*0.1s kết thúc thì quá trình đọc thành công. Nếu thời gian timer đạt được giá trị TIME trước khi số bytes nhận đạt giá trị MIN thì quá trình đọc

cũng thành công nhưng lúc này dữ liệu trả về là số byte đã nhận thành công trước đó.

- *Trong trường hợp MIN=0 và TIME=0,*

  Trong trường hợp này, hai giá trị TIME và MIN đều không được quan tâm. Nghĩa là bất kỳ bộ đệm có dữ liệu hay không, khi ta gọi giao diện hàm read() thì hàm sẽ được thực thi ngay lập tức. Trong trường hợp dữ liệu có sẵn, thì quá trình đọc thành công, giá trị trả về là số bytes đọc được. Trong trường hợp không có dữ liệu, quá trình đọc cũng thành công, nhưng giá trị trả về là 0, tương ứng với số byte đọc được là 0 byte.

a. Yêu cầu dự án:

Cũng tương tự như bài thực hành UART trong chế độ canonical, dự án này cũng thực hiện truyền dữ liệu qua lại giữa hai terminal, terminal_1 (sử dụng kênh UART0) và terminal_0 (Sử dụng kênh UART DEBUG). Nhưng khác với bài thực hành trước ở chỗ, thông tin nhận được không theo từng dòng ký tự mà theo từng block 5 ký tự. Nghĩa là, khi terminal_1 truyền đủ 5 ký tự cho terminal_0 thì terminal_0 lập tức nhận và xuất ra màn hình hiển thị. Thông tin hiển thị bao gồm những ký tự nhận được và số ký tự nhận được.

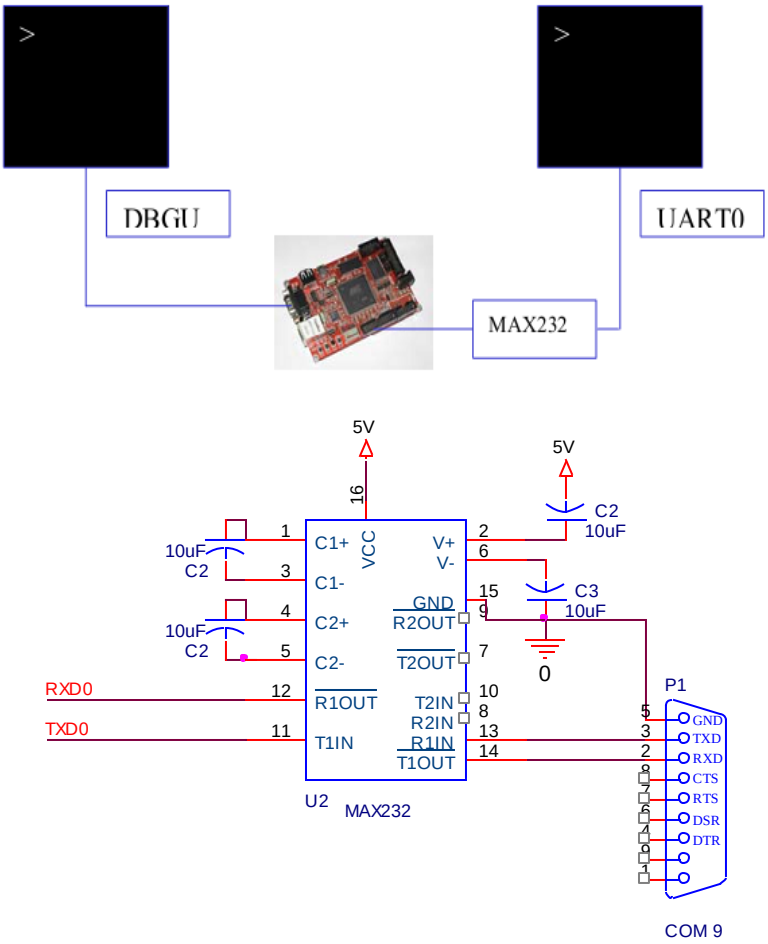
b. Phân công nhiệm vụ:

- **Driver:** Sử dụng driver ttyS1 cùa kênh UART0.
- **Application:** Chương trình ứng dụng có tên `test_serial_app_2.c`
  - Cài đặt các thông số cho kênh UART0 tương tự như bài trước;
  - Liên tục gọi giao diện hàm read() để đọc thông tin nhận được từ terminal_1 cho đến khi ký tự đầu tiên trong khối là ‘z’;
  - Chương trình sẽ dừng ngay tại giao diện hàm read() cho đến khi quá trình đọc thành công.

II. Thực hiện:

a. Kết nối phần cứng:

Kết nối phần cứng theo sơ đồ sau:



Hình 4-37- Sơ đồ kết nối giao tiếp giữa hai kênh UART0 và DEBUG.

b. Chương trình application:

*/*Chương trình application mang tên test_serial_app_2.c*/*

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

#define BAUDRATE B115200
#define MODEMDEVICE "/dev/ttyS1"
#define FALSE 0
#define TRUE 1
volatile int STOP=FALSE;
```

```
/*Chương trình chính được khai báo không có tham số và giá trị trả về*/
main()
{
int fd, res;
struct termios oldtio,newtio;
char buf[255];
fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
if (fd <0) {
   printf("Error while opening the device\n");
   exit(-1);
}
tcgetattr(fd,&oldtio); /* save current port settings */
bzero(&newtio, sizeof(newtio));
newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
newtio.c_iflag = IGNPAR;
newtio.c_oflag = 0;
/*Cài đặt chế độ ngõ vào là non-canonical mode*/
newtio.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
/*Cài đặt hai giá trị TIME và MIN trong trường thông tin c_cc*/
/*Không sử dụng timer cài đặt thời gian tối đa giữa hai lần nhận dữ  liệu*/
newtio.c_cc[VTIME]    = 0;
/*Cài đặt kích thước khối dữ liệu là 5 bytes*/
newtio.c_cc[VMIN]     = 5;
tcflush(fd, TCIFLUSH);
/*Cập nhật các thuộc tính cho kênh UART với yêu cầu thực thi ngay lập  tức*/
tcsetattr(fd,TCSANOW,&newtio);
/*Vòng lặp thực thi các yêu cầu thuật toán trong dự án*/
/*Vòng lặp liên tục gọi giao diện hàm read() đọc thông tin từ UART*/
while (STOP==FALSE) {
/*Mặc dù đọc về 255 ký tự nhưng thực ra số lượng ký tự trong mỗi lần đọc là 5, vì
thông tin được đọc theo từng khối thông tin*/
   res = read(fd,buf,255);
   buf[res]=0;
   printf(":%s:%d\n", buf, res);
   if (buf[0]=='z') STOP=TRUE;
```

```
}
/*Phục hồi trạng thái trước đó của kênh UART đã sử dụng*/
tcsetattr(fd,TCSANOW,&oldtio);
}
```

c. Thực thi chương trình:

- Biên dịch chương trình ứng dụng bằng câu lệnh sau:

  ```
  arm-none-linux-gnueabi-gcc test_serial_app_2.c -o test_serial_app_2
  ```

  Tập tin ngõ ra có tên test_serial_app_2 được chép vào kit.

- Chương trình test_serial_app_2 được thực thi trong kit bằng dòng lệnh:

  /*Chuyển đổi mode cho tập tin vừa chép vào hệ thống thực thi trong mọi user */

  ```
  chmode 777 test_serial_app_2
  ```

  /*Thực thi chương trình*/

  ```
  ./test_serial_app_2
  ```

  (Lúc này chương trình sẽ chờ cho đến khi có dòng thông tin nhận được)

- Tại terminal_1 ta nhập dòng thông tin sau:

  ```
  "0123456789"
  ```

- Tại terminal_0 ta thấy xuất hiện dòng thông tin sau:

  ```
  : 01234:5
  : 56789:5
  _
  ```

  Như vậy không chờ nhấn enter, thông tin vẫn được truyền từ terminal_1 sang terminal_0 miễn sao đủ số lượng 5 bytes.

  Chương trình hoạt động theo chế độ này vẫn có thể nhận được các ký tự điều khiển nhưng không hiển thị ra terminal console. Ta thử nhấn 5 lần phím enter tại terminal_1 thông tin hiển thị trong terminal_0 như sau:

  ```
  ::5
  _
  ```

  (Chương trình chờ cho đến khi xuất hiện ký tự ‘z’ tại vị trí đầu tiên của khối thông tin tiếp theo)

III. Kết luận và bài tập:

a. Kết luận:

Trong bài thực hành này người học đã điều khiển thành công kênh UART theo chế độ non-canonical, truyền dữ liệu theo từng khối thông tin. Thông tin được truyền đi theo từng khối, chiều dài của khối có thể thay đổi được, chế độ này phù hợp với các ứng dụng điều khiển không phù hợp với các ứng dụng truyền thông tin thông báo.

b. Bài tập:

1. Người học thay đổi các thông số TIME và MIN sao đó biên dịch và kiểm tra kết quả, qua đó có thể hiểu rõ hơn ý nghĩa của hai tham số này và rút ra kinh nghiệm lập trình.

E- BÀI THỰC HÀNH UART 3:

I. Phác thảo dự án:

Trong hai bài trước chúng ta đã tìm hiểu hai chế độ truyền dữ liệu nối tiếp theo chuẩn UART, với mỗi chế độ ta thấy có một điểm chung: Khi gọi giao diện hàm read() trước khi nhận dữ liệu thành công, chương trình gọi sẽ dừng lại tại vị trí hàm cho đến khi hoàn thành sau đó mới tiếp tục thực hiện những câu lệnh tiếp theo. Điều này đôi khi không thuận lợi, làm tiêu tốn thời gian hoạt động của tiến trình hiện tại và ảnh hưởng đến hoạt động của các tác vụ khác cùng tuyến (thread) và tiến trình (process). Để khắc phục nhược diểm này ta có hai cách:

- ***Cách 1:*** Chúng ta khai báo hai tuyến (thread) tồn tại trong một tiến trình. Một tuyến làm nhiệm vụ liên tục đọc thông tin trong kênh UART bằng cách gọi hàm read(). Các tuyến khác thực hiện các nhiệm vụ xử lý thông tin nhận được để thực thi điều khiển hoặc hiển thị.
- ***Cách 2:*** Khai báo một tín hiệu cho kênh truyền dữ liệu nối tiếp. Tín hiệu này sẽ phát sinh khi dữ liệu truyền|nhận được thực hiện thành công. Lúc này tiến trình cài đặt cho tín hiệu sẽ được thực thi, tiến trình này sẽ gọi giao diện hàm read() hay write() để đọc thông tin nhận được hoặc truyền dữ liệu mới.

Trong bài này chúng ta sẽ thực hành cách thứ hai là tạo và xử lý tín hiệu được tạo bởi kênh UART. Tín hiệu ngắt có thể được khởi tạo trong cả hai chế độ truyền canonical và non-canonical. Do đó bài thực hành này sẽ thực hiện lại hai ví dụ trong hai bài trước nhưng sử dụng kỹ thuật tạo tín hiệu ngắt này với hai chế độ truyền trong UART.

a. Yêu cầu dự án:

Dự án này được chia thành hai chương trình. Chương trình A tạo và xử lý tín hiệu ngắt trong UART sử dụng chế độ canonical, chương trình B tạo và xử lý tín hiệu ngắt trong UART sử dụng chế độ non_canonical.

b. Phân công nhiệm vụ:

- **Driver:** Sử dụng driver UART ttyS1 trong linux.
- **Application:**

Hai chương trình với hai chế độ làm việc khác nhau nhưng có cùng chức năng:

- Khai báo các thông số cần thiết cho kênh UART cần điều khiển.

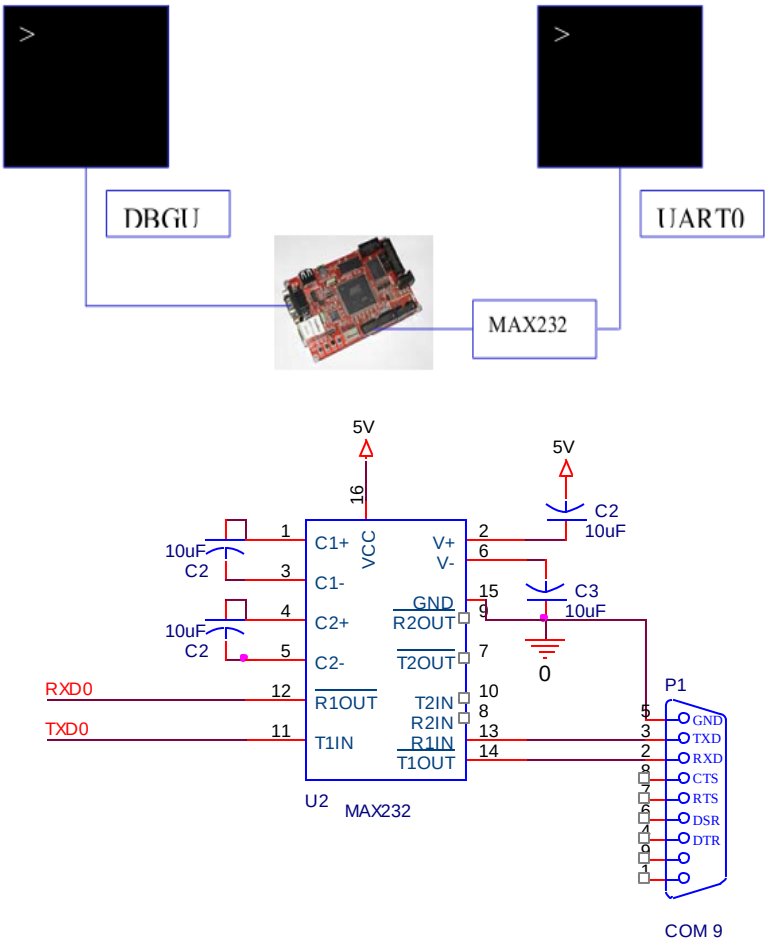
- Khai báo tín hiệu xử lý ngắt cho chương trình đang thực thi, gắn hàm xử lý tín hiệu ngắt cho tín hiệu bằng hàm sigaction() (đã được nghiên cứu trong phần lý thuyết lập trình nhúng nâng cao).
- Hàm xử lý tín hiệu ngắt làm nhiệm vụ:
  + In ra thông báo cho biết đang thực thi hàm xử lý tín hiệu ngắt;
  + Mở khóa cho chương trình main() in thông tin ra màn hình hiển thị;
- Sau khi khởi tạo các thông số cho kênh UART, khai báo tín hiệu cho phép tiến trình nhận tín hiệu ngắt và khai báo hàm phục vụ ngắt cho tín hiệu bằng hàm sigaction() chương trình chính sử dụng vòng lặp vô tận in ra chuỗi dấu "." chờ tín hiệu ngắt xảy ra, mở khóa hiện dữ liệu nhận được ra màn hình. Chương trình chỉ kết thúc khi nhận được ký tự 'z' tại vị trí đầu tiên của bộ đệm.

  ***Trong thực tế thì tại thời điểm này chương trình chính sẽ thực thi những tác vụ khác mang tính chất liên tục, lặp lại nhiều lần.*

II. Thực hiện:

a. Kết nối phần cứng:

Thực hiện kết nối phần cứng theo sơ đồ sau: (*Cũng tương tự như những bài thực hành trước*).



Hình 4-38- Sơ đồ kết nối giữa hai kênh UART0 và DEBUG.

b. Chương trình application:

- *Chương trình application theo chế độ canonical:* có tên là test_serial_3_a.c

*/*Khai báo các thư viện cần thiết cho các hàm trong chương trình */*

```
#include <termios.h> /*Thư viện cho các hàm hỗ trợ UART*/
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h> /*Thư viện cho xử lý các tin hiệu trong user*/
#include <sys/types.h>
```

*/*Hằng số lưu tốc độ baud là 115200 bps*/*

```
#define BAUDRATE B115200
```

```
/*Hằng số lưu tên thiết bị driver là kênh UART0*/
#define MODEMDEVICE "/dev/ttyS1"
#define FALSE 0
#define TRUE 1
 /*Khai báo các biến toàn cục cho chương trình chính và chương trình phục vụ ngắt*/
 /*Biến lưu khóa để thoát khỏi vòng lặp*/
volatile int STOP=FALSE;
/*Biến lưu số mô tả tập tin cho hàm open() và giá trị trả về cho hàm read()*/
int fd,res;
/*Định nghĩa hàm phục vụ tín hiệu ngắt SIGIO*/
void signal_handler_IO (int status);
/*Mảng ký tự lưu trữ thông tin nhân được từ giao diện hàm read()*/
char buf[255];
/*Khai báo chương trình chính dạng không có tham số và có giá trị trả về*/
int
main() {
/*Khai báo cấu trúc struct termios lưu thuộc tính cấu hình cho kênh UART
Biến cấu trúc oldtio dùng lưu thuộc tính hiện tại của kênh UART
Biến cấu trúc newtio dùng lưu thuộc tính mới của kênh UART*/
struct termios oldtio,newtio;
/*Khai báo cấu trúc sigaction cho việc khai báo và xử lý tín hiệu ngắt*/
struct sigaction saio;
/*Đầu tiên mở tập tin thiết bị UART là ttyS1 với các thông số liên quan*/
fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK);
/*Kiểm tra lỗi trong quá trình mở tập tim thiết bị*/
if (fd <0) {perror(MODEMDEVICE); exit(-1); }

/* Cài đặt những thông số cho các trương thông tin trước khi cài đặt xử lý tín hiệu ngắt*/
/*Cài đặt hàm thực thi ngắt cho tín hiệu ngắt*/
saio.sa_handler = signal_handler_IO;
/*Xóa những cờ thông tin*/
sigemptyset(&saio.sa_mask);
saio.sa_flags = 0;
```

```
/*Gọi hàm sigaction() để gán cấu trúc sigaction vào tín hiệu SIGIO*/
sigaction(SIGIO,&saio,NULL);

/* Cho phép tiến trình đang thực thi nhận tín hiệu SIGIO trong user*/
fcntl(fd, F_SETOWN, getpid());
/*Thực hiện đồng bộ hóa số mô tả tập tin*/
fcntl(fd, F_SETFL, FASYNC);
/*Các bước sau cài đặt các thông số cho kênh UART cần giao tiếp*/
/*Lấy về cấu hình hiện tại câu kênh UART phục vụ cho việc phục hồi*/
tcgetattr(fd,&oldtio);
/* Cài đặt các thông số cho chế độ ngõ vào là canonical*/
newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
newtio.c_iflag = IGNPAR | ICRNL;
newtio.c_oflag = 0;
/*Chế độ giao tiếp là canonical mode*/
newtio.c_lflag = ICANON;
newtio.c_cc[VMIN]=1;
newtio.c_cc[VTIME]=0;
tcflush(fd, TCIFLUSH);
/*Cài đặt thông số mới cho kênh AURT cần giao tiếp*/
tcsetattr(fd,TCSANOW,&newtio);

/*Vòng lặp vô tận chờ quá trình ngắt xảy ra, chương trình chỉ thoát khỏi  vòng lặp
này khi biến khóa STOP=TRUE*/
while (STOP==FALSE) {
   /*In ra ký tự "." Báo hiệu không có dữ liệu truyền đến*/
   printf(".\n");
   /*Trì hoãn thời gian 1s*/
   sleep(1);
   /*Kiểm tra xem có tín hiệu ngắt xảy ra hay chưa*/
   if (wait_flag==FALSE) {
   /*Khóa đã được mở*/
         /*Đọc dữ liệu từ bộ nhớ đệm của kênh UART*/
         res = read(fd,buf,255);
         buf[res]=0;
```

```
            printf(":%s:%d\n", buf, res);
            /*Kiểm tra điều kiện thoát chương trình*/
            if (buf[0] == 'z') STOP=TRUE;
            /*Khóa lại việc đọc dữ liệu từ vùng đệm*/
            wait_flag = TRUE;
        }
    }
    /* Khôi phục lại thông số gốc của kênh UART */
    tcsetattr(fd,TCSANOW,&oldtio);
    }
    /*Định nghĩa hàm phục vụ ngắt thực thi yêu cầu nêu trong dự án*/
    void signal_handler_IO (int status) {
        /*In thông báo ra màn hình hiển thị là nhận được tín hiệu báo quá trình nhận dữ
        liệu thành công*/
        printf("received SIGIO signal.\n");
        /*Tháo khóa cho phép chương trình chính ghi thông tin ra màn hình hiển thị*/
        wait_flag = FALSE;
    }
```

- *Chương trình application theo chế độ non-canonical:*

Chương trình cũng tương tự như trong chế độ canonical, chỉ khác một số thông tin khai báo chế độ hoạt động cho kênh UART. Chương trình được viết như sau:

```
/*Những chú thích sẽ được ghi khi cần thiết*/
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/types.h>

#define BAUDRATE B115200
#define MODEMDEVICE "/dev/ttyS1"
#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;
int fd,c, res;
```

```
void signal_handler_IO (int status);
int wait_flag=TRUE;
int fd,c, res;
char buf[255];

main() {
struct termios oldtio,newtio;
struct sigaction saio;

fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK);
if (fd <0) {perror(MODEMDEVICE); exit(-1); }
saio.sa_handler = signal_handler_IO;
sigemptyset(&saio.sa_mask);
saio.sa_flags = 0;
sigaction(SIGIO,&saio,NULL);
fcntl(fd, F_SETOWN, getpid());
fcntl(fd, F_SETFL, FASYNC);
tcgetattr(fd,&oldtio);
newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
newtio.c_iflag = IGNPAR | ICRNL;
newtio.c_oflag = 0;
/*Cài đặt kênh UART thành chế độ non-canonical*/
newtio.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
/*Vô hiệu hòa timer nhận giữa hai bytes liên tiếp*/
newtio.c_cc[VTIME]    = 0;
/*Chiều dài khối dữ liệu nhận là 5 bytes*/
newtio.c_cc[VMIN]     = 5;
tcflush(fd, TCIFLUSH);
tcsetattr(fd,TCSANOW,&newtio);
while (STOP==FALSE) {
   printf(".\n");
   sleep(1);
   if (wait_flag==FALSE) {
        res = read(fd,buf,255);
        buf[res]=0;
        printf(":%s:%d\n", buf, res);
        if (buf[0] == 'z') STOP=TRUE;
        wait_flag = TRUE;
```

```
    }
}
tcsetattr(fd,TCSANOW,&oldtio);
}
void signal_handler_IO (int status) {
   printf("Received SIGIO signal.\n");
   wait_flag = FALSE;
}
```

c. Thực thi chương trình:

Hai chương trình trên được biên dịch và thực thi tương tự như những bước thực hiện trong bài trước.

Người học quan sát và kiểm tra kết quả chương trình khi chạy trên hệ thống. Từ đó, rút ra được những kinh nghiệm lập trình. Người học thay đổi một trong những tham số của chương trình, biên dịch thực thi và kiểm tra sự thay đổi.

III. Mở rộng chương trình:

Trong hai ví dụ trước, chúng ta sử dụng kỹ thuật bẫy tín hiệu SIGIO để đồng bộ hóa việc đọc dữ liệu từ kênh UART khi đã sẵn sàng. Sau đây ta sẽ mở rộng thêm việc đọc dữ liệu từ kênh UART bằng kỹ thuật chi tuyến (thread) thực hiện, không cần khai báo sử dụng ngắt tín hiệu, sẽ dành cho mục đích khác.

Do các lệnh trong ví dụ này đều được lặp lại từ hai ví dụ trên nên chỉ đưa ra mã chương trình mà không giải thích, những đoạn chương trình quan trọng sẽ được chúng tôi giải thích kỹ trong từng dòng lệnh.

Chương trình ví dụ:

```
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/types.h>

#define BAUDRATE B115200
#define MODEMDEVICE "/dev/ttyS1"
#define FALSE 0
#define TRUE 1
```

```
volatile int STOP=FALSE;
int flag = FALSE;
int fd, res;
/*Định nghĩa hàm thực thi cho tuyến con*/
void *child_thread(void *data);
int fd,res;
char buf[255];

int
main() {
struct termios oldtio,newtio;
/*Tạo biến pthread lưu trữ thông tin của tuyến được tạo*/
pthread_t a_thread;
/*Biến lưu số định danh của tuyến*/
int thread_id;
/*Mở tập tin thiết bị với chế độ read write, không cho phép hiển thị lên màn hình
console và chờ cho đến khi dữ liệu đầy đủ mớ thoát (trong quá trình sử dụng giao
diện hàm read()*/
fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
if (fd <0) {perror(MODEMDEVICE); return -1; }

tcgetattr(fd,&oldtio);
newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
newtio.c_iflag = IGNPAR | ICRNL;
newtio.c_oflag = 0;
newtio.c_lflag = 0;
newtio.c_cc[VTIME]    = 0;
newtio.c_cc[VMIN]     = 2;
tcflush(fd, TCIFLUSH);
tcsetattr(fd,TCSANOW,&newtio);
/*sử dụng hàm pthread_create() để tạo tuyến con hoạt động song song với tuyến
chính*/
thread_id = pthread_create (&a_thread, NULL, child_thread, NULL);
/*Vòng lặp vô tận liện tục in ra ký tữ "." Đồng thời kiểm tra điều kiện thoát*/
while (1) {
   printf(".\n");
```

```
      sleep(1);
      if (buf[0] == 'z') return 0;
   }
   tcsetattr(fd,TCSANOW,&oldtio);
   }
```

*/*Chương trình thực thi của tuyến con, nhiệm vụ của tuyến con là đọc tập tin thiết bị, chờ cho đến khi việc nhận dữ liệu hoàn thành cuối cùng in dữ liệu và số ký tự nhận được ra màn hình hiển thị*/*

```
   void *child_thread(void *data) {
      while (1)  {
                  res = read(fd, buf, 255);
                  buf[res]=0;
                  printf(":%s:%d\n", buf, res);
                  flag = FALSE;
      }
   }
```

***Chương trình sử dụng kỹ thuật lập trình đa tuyến để kiểm tra việc nhận dữ liệu có ưu và nhược điểm sau đây:*

- *Ưu điểm của cách này là dễ thực hiện. Chương trình không cần phải khai báo các thông số cho việc xử lý tín hiệu ngắt, mà chỉ cần sử dụng hàm pthread_create() để đưa một tuyến con vào hoạt động.*
- *Nhược điểm là thời gian đáp ứng khi nhận được tín hiệu nhận thành công dữ liệu từ kênh UART. Thời gian tối thiểu trong kỹ thuật chia tuyến là thời gian chuyển đổi qua lại giữa các tuyến với nhau. Trong kỹ thuật bẫy tín hiệu, thì thời gian tối thiểu là thời gian chuyển từ chương trình chính sang chương trình thực thi ngắt. Rõ ràng khoảng thời gian dùng kỹ thuật bẫy tín hiệu luôn ngắn hơn rất nhiều so với khoảng thời gian khi dùng kỹ thuật chia tuyến.*

IV.Kết luận:

Trong bài thực hành này chúng ta đã thực hành điều khiển đồng bộ hóa trong việc nhận dữ liệu từ kênh UART trong hai chế độ. Quá trình truyền dữ liệu cũng được tiến hành tương tự. Để đồng bộ ta có hai cách thực hiện: dùng kỹ thuật bẫy tín hiệu và chia tiến trình thao tác với kênh UART. Mỗi phương pháp đều có ưu và nhược điểm riêng tùy theo yêu cầu của từng ứng dụng cụ thể.

Đến đây chúng ta cũng đã kết thúc phần nghiên cứu về chế độ truyền nối tiếp UART trong linux hoạt động trên kit KM9260. Những ví dụ được trình bày chỉ là những thao tác cơ bản nhất, người học có thể ứng dụng vào nhiều ứng dụng điều khiển khác nhau như giao tiếp máy tính, giao tiếp với các vi điều khiển khác theo chuẩn UART...