LED Matrix Controller with Custom Device Driver

Skills: C (programming language), Device Drivers, SPI

---

1. Define

Project Overview

The goal of this project is to design and implement a custom Linux device driver to control an **8x8 LED matrix** using the **MAX7219 driver chip** on a **Raspberry Pi**. Communication with the LED matrix will occur via the **SPI (Serial Peripheral Interface)** protocol. The project also includes the development of a **user-space C program** to interact with the device driver and manage the LED display, allowing for real-time updates and dynamic pattern changes.

Learning Objectives

- Gain experience in writing custom Linux device drivers.
- Understand the fundamentals of the **SPI** communication protocol.
- Control **GPIO pins** from the Linux kernel and user-space.
- Interface with hardware using **C** programming in both kernel and user-space.

2. Design

Hardware Design

Components:

1. **Raspberry Pi 3 Model B** (or equivalent)
2. **8x8 LED Matrix** (common-cathode)
3. **MAX7219** LED driver chip (for controlling the LED matrix)
4. **Wires, Breadboard, and Power Supply**

Connection Plan:

- The **MAX7219** will interface with the **Raspberry Pi** through its **SPI** interface.
- The Raspberry Pi will send SPI commands to control the LED matrix through the MAX7219.

Wiring:

MAX7219 Pin	Raspberry Pi Pin	GPIO Pin
VCC	5V Power(pin 2)	-
GND	Ground (pin 6)	-
DIN	SPI MOSI(Pin 19)	GPIO 12 (MOSI)
CLK	SPI SCLK (Pin 23)	GPIO 11 (SCLK)
CS	SPI CE0 (Pin 24)	GPIO 8 (CE0)

The 8x8 LED matrix will connect to the MAX7219 chip, which simplifies control by reducing the number of GPIO pins required.

Software Design

Driver Structure:

- **SPI Communication**: The MAX7219 will be controlled using the SPI protocol, allowing the Raspberry Pi to send commands for lighting up specific LEDs on the matrix.
- **Kernel Driver**:
  - The driver will initialize the SPI communication interface.
  - It will register a character device to allow communication between the user-space program and the driver.
  - It will provide **ioctl()** or **write()** system calls for setting patterns or adjusting brightness.

User-Space Application:

- A **C program** will run in user-space and interact with the custom device driver. It will:
  - Send SPI commands to the driver to change the LED matrix pattern.
  - Use simple command-line inputs to switch between patterns.

3. Development

Step 1: Enable SPI on Raspberry Pi

Before proceeding, ensure SPI is enabled on your Raspberry Pi by running:

```
Sudo raspi-config
```

Navigate to **Interface Options** and enable **SPI**.

Step 2: Write the Device Driver

1. **Set up the SPI Interface**: The driver should initialize SPI and prepare for communication with the MAX7219 chip.
2. **Initialize MAX7219**:
   - In the driver's **probe()** function (triggered when the driver is loaded), initialize the MAX7219 by sending configuration data (e.g., brightness, scan limit).
3. **Character Device Registration**:
   - Register a **character device** that will be used by the user-space application to communicate with the driver.
4. **SPI Communication from Driver**:
   - Implement the **write()** or **ioctl()** system calls to send data to the MAX7219 over SPI.

Sample Skeleton for SPI Driver:

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/spi/spi.h>
#include <linux/fs.h>
#include <linux/uaccess.h>

#define DEVICE_NAME "led_matrix"
#define CLASS_NAME "led_matrix_class"
#define SPI_BUS 0
#define SPI_BUS_CS1 0
#define SPI_BUS_SPEED 1000000 // 1 MHz

// Commands for MAX7219
#define MAX7219_REG_NOOP   0x00
#define MAX7219_REG_DIGIT0 0x01
#define MAX7219_REG_DIGIT1 0x02
#define MAX7219_REG_DIGIT2 0x03
#define MAX7219_REG_DIGIT3 0x04
#define MAX7219_REG_DIGIT4 0x05
#define MAX7219_REG_DIGIT5 0x06
#define MAX7219_REG_DIGIT6 0x07
#define MAX7219_REG_DIGIT7 0x08
#define MAX7219_REG_DECODE 0x09
#define MAX7219_REG_INTENSITY 0x0A
#define MAX7219_REG_SCANLIMIT 0x0B
#define MAX7219_REG_SHUTDOWN 0x0C
#define MAX7219_REG_DISPLAYTEST 0x0F

// SPI Device
static struct spi_device *led_matrix_spi_device;

// Device file major number
static int major_number;
static struct class* led_matrix_class = NULL;
static struct device* led_matrix_device = NULL;

// Send data to MAX7219 via SPI
static int max7219_send(unsigned char address, unsigned char data)
{
    unsigned char tx_buf[2];
    struct spi_transfer transfer = {
        .tx_buf = tx_buf,
        .len = 2,
```

```
        .speed_hz = SPI_BUS_SPEED,
        .bits_per_word = 8,
    };

    tx_buf[0] = address;
    tx_buf[1] = data;

    // Send the message
    return spi_sync_transfer(led_matrix_spi_device, &transfer, 1);
}

// Initialize MAX7219 LED driver
static void max7219_init(void)
{
    max7219_send(MAX7219_REG_SHUTDOWN, 0x01);  // Turn on
    max7219_send(MAX7219_REG_DECODE, 0x00);    // No decode for matrix
    max7219_send(MAX7219_REG_SCANLIMIT, 0x07); // Scan all digits (8 digits)
    max7219_send(MAX7219_REG_INTENSITY, 0x0F); // Maximum intensity
    max7219_send(MAX7219_REG_DISPLAYTEST, 0x00); // No display test
}

// Write function to update the matrix (Example to turn on specific LEDs)
static ssize_t led_matrix_write(struct file *file, const char __user
*buffer, size_t len, loff_t *offset)
{
    char data[8];

    if (len != 8)
        return -EINVAL;

    if (copy_from_user(data, buffer, 8))
        return -EFAULT;

    for (int i = 0; i < 8; i++) {
        max7219_send(MAX7219_REG_DIGIT0 + i, data[i]);
    }

    return len;
}

// Open function
static int led_matrix_open(struct inode *inode, struct file *file)
{
    pr_info("LED Matrix device opened\n");
    return 0;
}

// Release function
```

```c
static int led_matrix_release(struct inode *inode, struct file *file)
{
    pr_info("LED Matrix device closed\n");
    return 0;
}

// File operations structure
static struct file_operations fops = {
    .owner = THIS_MODULE,
    .write = led_matrix_write,
    .open = led_matrix_open,
    .release = led_matrix_release,
};

// Probe function for SPI
static int led_matrix_probe(struct spi_device *spi_device)
{
    led_matrix_spi_device = spi_device;

    // Register the device
    major_number = register_chrdev(0, DEVICE_NAME, &fops);
    if (major_number < 0) {
        pr_err("Failed to register a major number\n");
        return major_number;
    }

    // Create the device class
    led_matrix_class = class_create(THIS_MODULE, CLASS_NAME);
    if (IS_ERR(led_matrix_class)) {
        unregister_chrdev(major_number, DEVICE_NAME);
        pr_err("Failed to create device class\n");
        return PTR_ERR(led_matrix_class);
    }

    // Create the device
            led_matrix_device   =   device_create(led_matrix_class,   NULL,
MKDEV(major_number, 0), NULL, DEVICE_NAME);
    if (IS_ERR(led_matrix_device)) {
        class_destroy(led_matrix_class);
        unregister_chrdev(major_number, DEVICE_NAME);
        pr_err("Failed to create the device\n");
        return PTR_ERR(led_matrix_device);
    }

    // Initialize the MAX7219
    max7219_init();
    pr_info("LED Matrix SPI device initialized\n");
```

```c
    return 0;
}

// Remove function for SPI
static int led_matrix_remove(struct spi_device *spi_device)
{
    device_destroy(led_matrix_class, MKDEV(major_number, 0));
    class_unregister(led_matrix_class);
    class_destroy(led_matrix_class);
    unregister_chrdev(major_number, DEVICE_NAME);

    pr_info("LED Matrix SPI device removed\n");
    return 0;
}

// SPI driver structure
static struct spi_driver led_matrix_spi_driver = {
    .driver = {
        .name = "led_matrix_driver",
        .owner = THIS_MODULE,
    },
    .probe = led_matrix_probe,
    .remove = led_matrix_remove,
};

// Module initialization
static int __init led_matrix_init(void)
{
    pr_info("LED Matrix Module Init\n");
    return spi_register_driver(&led_matrix_spi_driver);
}

// Module exit
static void __exit led_matrix_exit(void)
{
    spi_unregister_driver(&led_matrix_spi_driver);
    pr_info("LED Matrix Module Exit\n");
}

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Tamil selvan");
MODULE_DESCRIPTION("Custom LED Matrix Driver with SPI on Raspberry Pi");
MODULE_VERSION("1.0");

module_init(led_matrix_init);
module_exit(led_matrix_exit);
```

Step 3: Write User-Space Application (C Program)

A simple C application that writes patterns to the device driver using the **write()** system call:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

#define LED_MATRIX_DEVICE "/dev/led_matrix"

// 8x8 LED matrix patterns
unsigned char smiley_face[8] = {
    0b00111100, //   ****
    0b01000010, // *      *
    0b10100101, // * *  * *
    0b10000001, // *      *
    0b10100101, // * *  * *
    0b10011001, // *  **  *
    0b01000010, //  *    *
    0b00111100  //   ****
};

unsigned char heart_shape[8] = {
    0b00000000, //
    0b01100110, //  **  **
    0b11111111, // ********
    0b11111111, // ********
    0b11111111, // ********
    0b01111110, //  ******
    0b00111100, //   ****
    0b00011000  //    **
};

// Function to write data to the LED matrix device
void write_pattern_to_led_matrix(unsigned char *pattern)
{
    int fd = open(LED_MATRIX_DEVICE, O_WRONLY);
    if (fd == -1) {
        perror("Failed to open LED matrix device");
        exit(EXIT_FAILURE);
    }

    // Write the 8-byte pattern to the device
    ssize_t result = write(fd, pattern, 8);
    if (result != 8) {
        perror("Failed to write pattern to LED matrix");
```

```
    } else {
        printf("Pattern successfully written to LED matrix.\n");
    }

    close(fd);
}

int main()
{
    int choice;

    printf("LED Matrix Controller\n");
    printf("Choose a pattern to display:\n");
    printf("1. Smiley Face\n");
    printf("2. Heart Shape\n");
    printf("3. Exit\n");

    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            write_pattern_to_led_matrix(smiley_face);
            break;
        case 2:
            write_pattern_to_led_matrix(heart_shape);
            break;
        case 3:
            printf("Exiting...\n");
            exit(EXIT_SUCCESS);
        default:
            printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

This program sends an 8x8 LED pattern to the driver to display on the matrix.

4. Debug

Step 1: Debugging the Kernel Module

- **Kernel Logs**: Use `dmesg` to check for driver initialization errors:

  dmesg | tail

- **SPI Communication**: Check SPI communication by probing the SPI bus using:

  sudo spidev_test -D /dev/spidev0.0

  This tool can confirm whether SPI communication is correctly set up**.**

  Check the spi enabled in raspberry pi
  sudo raspi-config

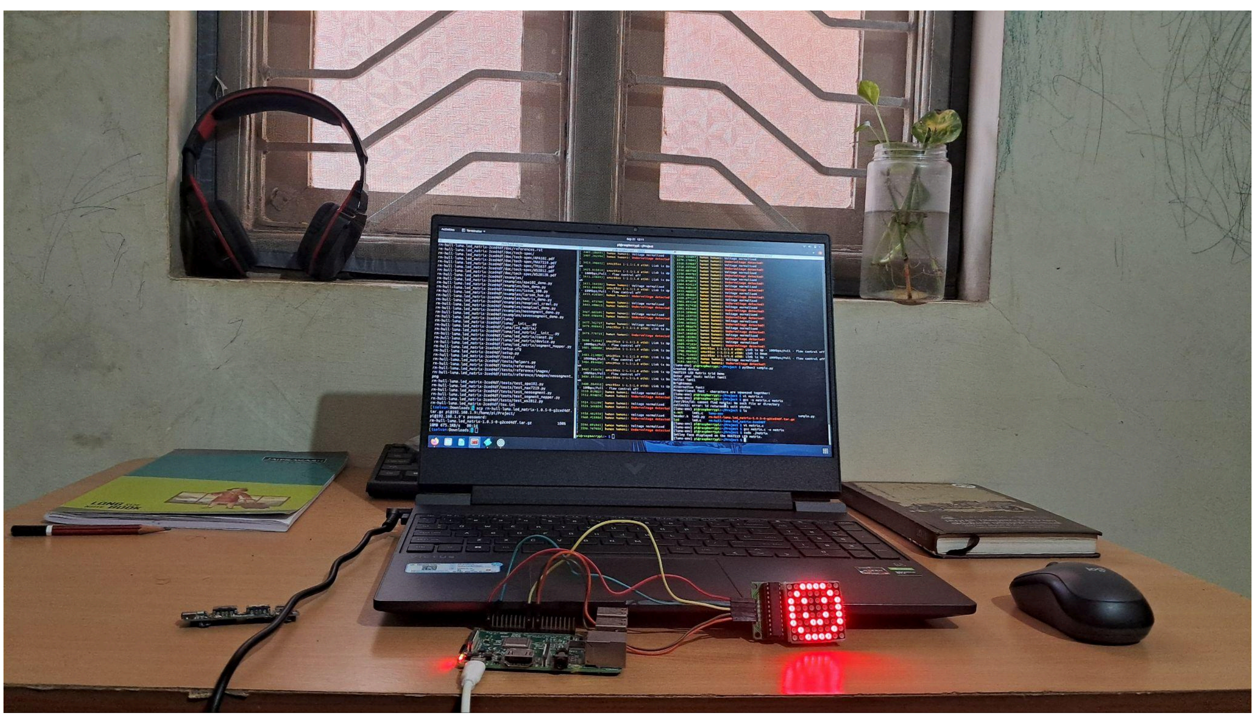
- **Driver Insertion/Removal**: Use `insmod` and `rmmod` to insert and remove the driver.

  sudo insmod led_matrix_drv.ko

  sudo rmmod led_matrix_drv

Step 2: Debugging User-Space Application

- **System Call Errors**: Check for errors during **open()**, **write()**, or **ioctl()** system calls.
- **Pattern Testing**: Test different LED matrix patterns and ensure they display correctly.



5. Future Enhancements

I2C and UART Integration:

- After mastering SPI, you can extend the project to interface additional sensors or devices using I2C or UART.
- For example, use I2C to connect temperature sensors and display the readings on the LED matrix.

Scaling the Project:

- You can expand the matrix by daisy-chaining multiple MAX7219 chips to control larger matrices.

This project provides a deep dive into **custom Linux device driver development** using **C** and familiarizes with communication protocols such as **SPI**. By implementing both the kernel-space driver and the user-space application, gain a comprehensive understanding of embedded systems programming and protocol handling.

Cross Compilation;

cross-compiling and deploying **custom device driver** and **user-space program** for the **Raspberry Pi 3B**.

Step-by-Step Approach for Cross-Compilation:

1. **Set Up Development Environment on the Host Machine:**
    - Install the necessary cross-compilation toolchain for Raspberry Pi on the host machine (Ubuntu or similar).
    - Toolchain: `gcc-arm-linux-gnueabi` for 32-bit systems or `gcc-aarch64-linux-gnu` for 64-bit systems.

```
sudo apt install gcc-aarch64-linux-gnu
```

2. **Set Up Kernel Headers on Raspberry Pi:**
    - Install the appropriate kernel headers on the Raspberry Pi to ensure compatibility with your custom driver.

```
sudo apt install raspberrypi-kernel-headers
```

3. **Write the Driver Code on the Host Machine:**

- Develop your Linux kernel module (driver program) on the host machine.
- Ensure the correct file paths and necessary headers are included.
- Create a Makefile to support cross-compilation using the correct toolchain.

Example Makefile snippet:

```
CROSS_COMPILE = arm-linux-gnueabihf-KERNELDIR ?= /path/to/raspberry/kernel/headers
PWD := $(shell pwd)
obj-m := your_driver.o
all:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) ARCH=arm CROSS_COMPILE=$(CROSS_COMPILE) modules
clean:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) clean
```

4. **Cross-Compile the Kernel Module:**
   - Use the Makefile and cross-compile the driver on the host machine.

```
make ARCH=arm CROSS_COMPILE=gcc-aarch64-linux-gnu
```

5. **Transfer the Compiled Driver to Raspberry Pi:**
   - Use `scp` or a similar method to transfer the compiled `.ko` (kernel object) file to the Raspberry Pi.

```
scp led_matrix_drv.ko pi@raspberrypi:/home/pi/Project
```

6. **Install and Load the Driver on Raspberry Pi:**
   - SSH into the Raspberry Pi and install the driver using `insmod`.

```
sudo insmod /home/pi/led_matrix_drv.ko
```

   - Check if the driver is loaded using `lsmod` and inspect `/dev/` for device nodes created by the driver.
7. **Write and Cross-Compile the User-Space Program:**
   - Develop the user-space C program to interact with the device driver.
   - Use the same cross-compilation toolchain to compile the user-space application.

```
arm-linux-gnueabihf-gcc -o led_matrix_app user_app.c
```

8. **Transfer the User-Space Program to Raspberry Pi:**
   - Use `scp` to send the compiled user-space program to the Raspberry Pi.

```
scp led_matrix_app pi@raspberrypi:/home/pi/Project
```

9. **Run the User-Space Program on the Raspberry Pi:**
    - Run the user-space program to interact with the driver.

```
sudo ./led_matrix_app
```

10. **Test and Debug:**
    - Check kernel logs with `dmesg` for any messages or errors related to the driver.
    - Ensure the driver and user-space program are functioning as expected.