Smart Home Application Project

Amit's Graduation Project

Prepared By

Areej Helmy

Yousef Elsherif

Supervised By

Eng. Karim El-Nahal

2020/2021

Introduction:

AVR micro-controllers provide a great platform to create a magnificent projects. Smart home application build by ATMEGA32 micro-controller which is based on AVR. With the arrival of the **IoT** and the need for control, devices now need to do more than perform the basic functions for which they are built, they need to be capable of communicating with other devices like a mobile phone among others. There are different communication systems which can be adapted for communication between devices, they include systems like WiFi, RF, Bluetooth among several others. Our focus will be on communication over Bluetooth. Using Bluetooth has many advantages, it’s secure, easy to us It works in short distance range (i.e. up to 10mtrs.) and its available.

This system mechanism operation depends on interfacing two ECU connected via SPI. One act like master another act as actuator. The master receives commands which have been sent from a mobile application via the Bluetooth. Those commands will transfer the actuator via the communication protocol to execute those commands.

In this case the best communication protocol is the I2C (Inter-Integrated Circuit) or SPI (Serial Peripheral Interface). Both protocols define communications between a master and one or more actuator devices. While interchangeable in some scenarios, each protocol has its specific advantages and drawbacks, making the choice of which to use is important for an elegant and highly functional device.

While I2C’s wiring simplicity and standardization are significant advantages, SPI has some attractive features as well.

1. Communication:

Separate MISO/MOSI data lines mean that it’s capable of full-duplex communication, as opposed to I2C’s half-duplex operation, meaning that data send and receive transmissions must alternate.

2. Speed:

I2C originally defined data transfer rates at 100kbps, though we have seen it bump up to 400kbps or even up to 5Mbps in Ultra Fast-mode. SPI, however, does not define a top—or any—communications speed, and can be implemented at speeds of 10 Mbps or more.

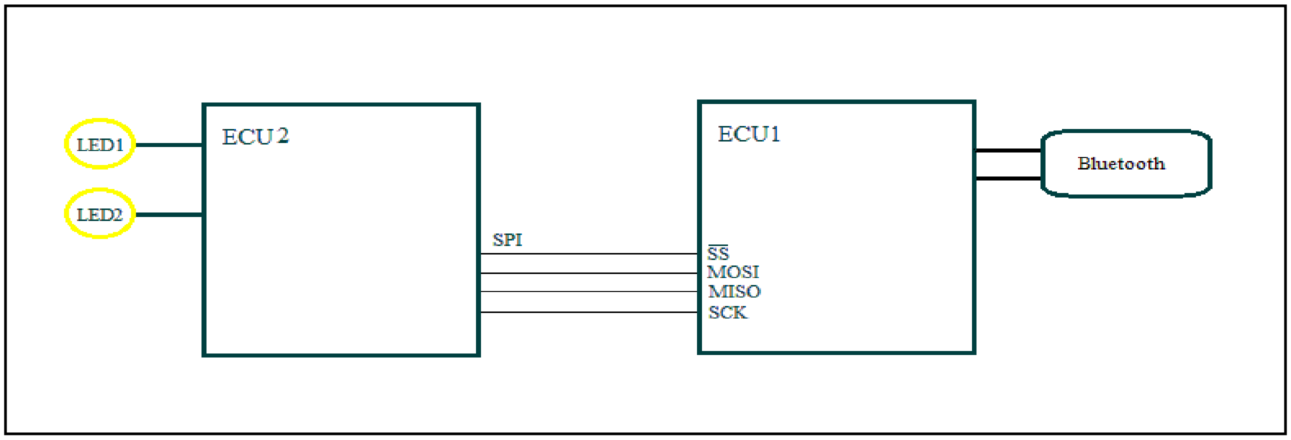
SPI’s advantages make it appropriate for applications like reading/writing data to-from ECUs, or any other application where data transfer speed is essential. Conversely, I2C tends to be strong when sending rather simple control signals to multiple peripherals, such as reading from a real-time clock or adjusting the volume of a remote speaker. Users can also utilize SPI for remote speaker control, though SPI's more complex wiring requirements may make it a less attractive option.

As a very simplified rule of thumb, you might say that SPI is a better choice for the small number of peripherals that need to transfer a large amount of data. I2C is the best option if you need to control many peripherals, especially if you are transferring a small amount of data to each one. Your application, of course, may be different, and designers should also consider what devices are available that support only one protocol or the other.

The proposed system:

For simplicity, the goal of the project is to switch two LEDs connected to the ECU2 using the mobile app. We will be using the **HC05 Bluetooth, two microcontrollers and wired (SPI).** The HC0 Bluetooth Module has 4 pins and communicates with a microcontroller via serial interface.

Circuit Connections:



In the circuit, Micro controller's boards, LEDs need to represent output, HC-05 to receive command from mobile application. When the circuit is initialized the master ECU initialize USART library to receive command from the Bluetooth module. The commands are typed into the smartphone application terminal (as shown in below figure) and then sent via Bluetooth (HC-05) attached to the main microcontroller.

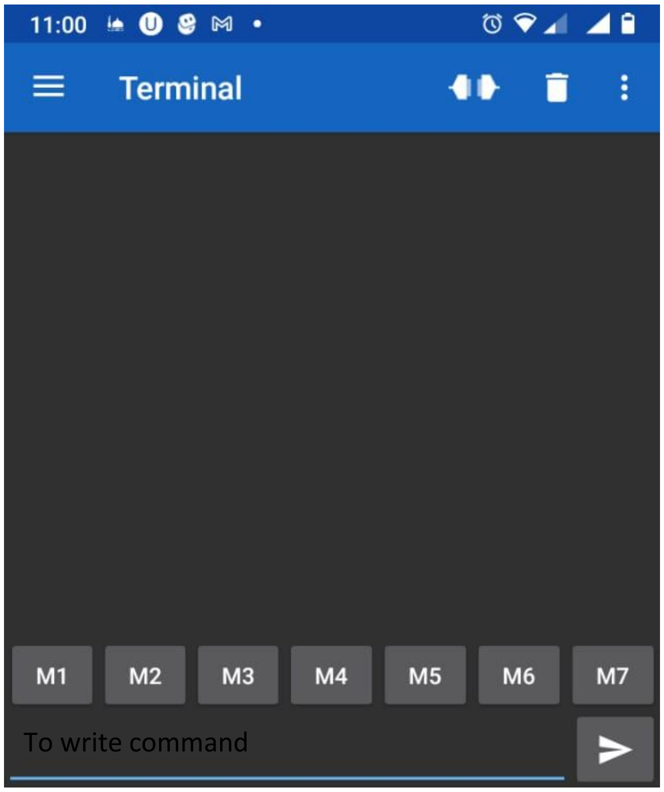


Figure 1smartphone application terminal

Simulation:

Simulation of a system in which the master microcontroller receives commands sent from a mobile application via HC-05 by USART protocol and then sends those commands to the sub-microcontroller via the SPI protocol to execute those commands.

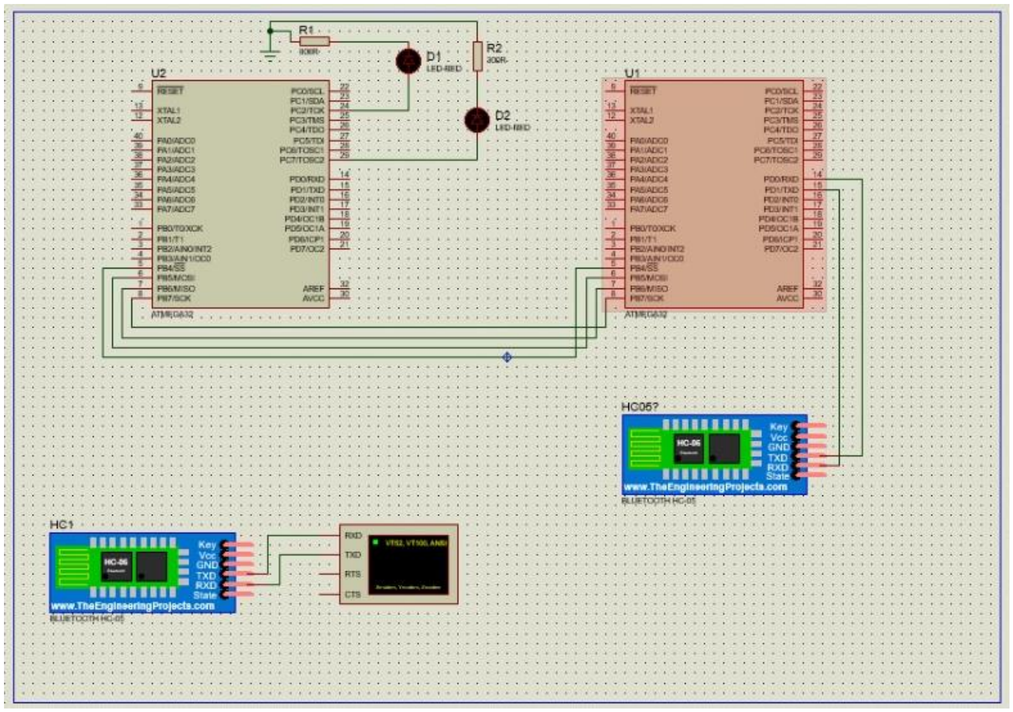


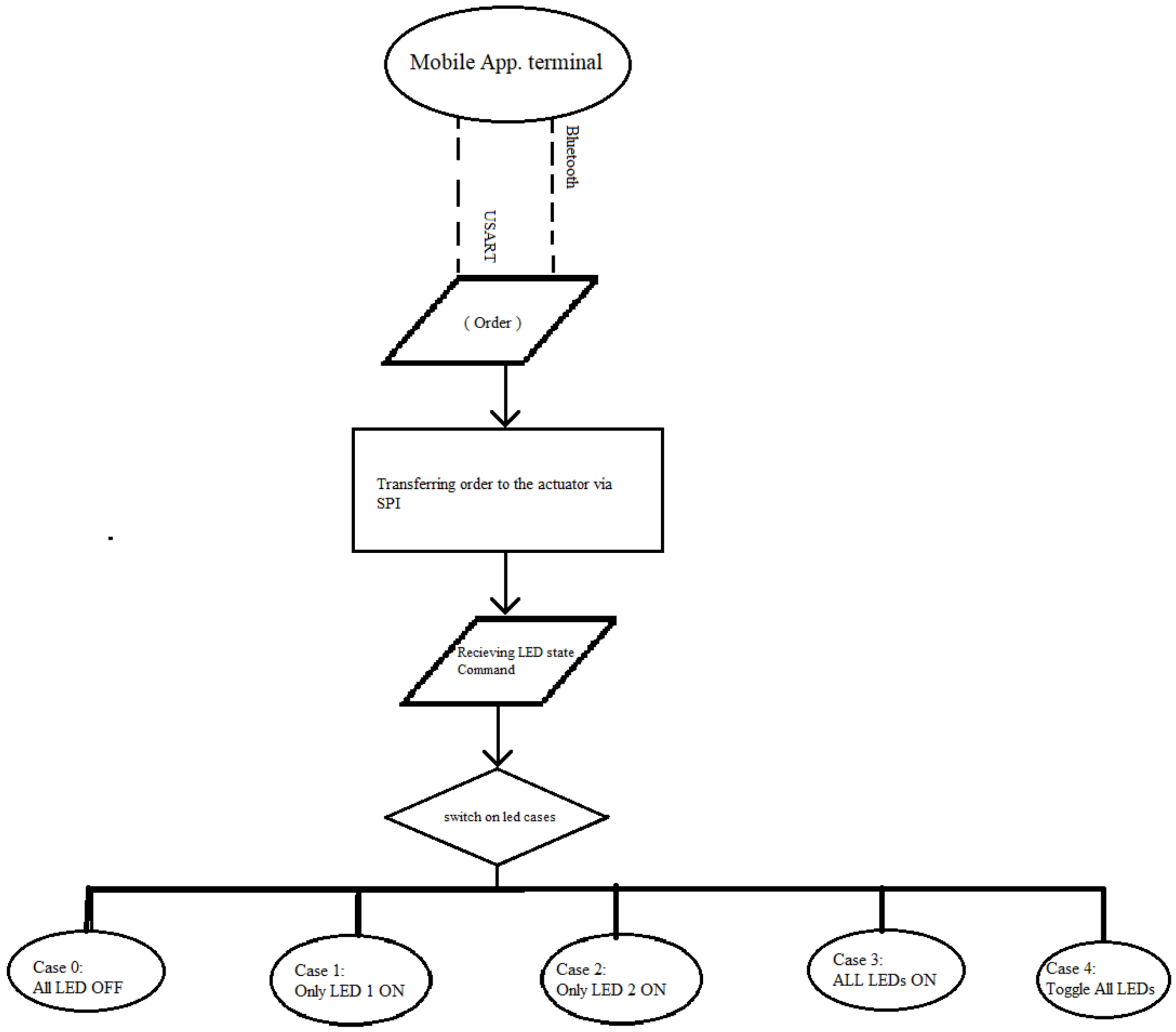
Figure 2smart home project simulation

Main features:

- Use the Serial Bluetooth Terminal smartphone app to send control commands to the main microcontroller
- Commands and their Functions:
  - → 0 (Default: All LEDs OFF)
  - → 1 (LED 1 ON)
  - → 2 (LED1 OFF & LED 2 ON)
  - → 3 (All LEDs ON)
  - → 4 (Toggle All LEDs)
- The HC-05 Bluetooth module passes the commands sent from the smartphone to the main microcontroller through the USART protocol.

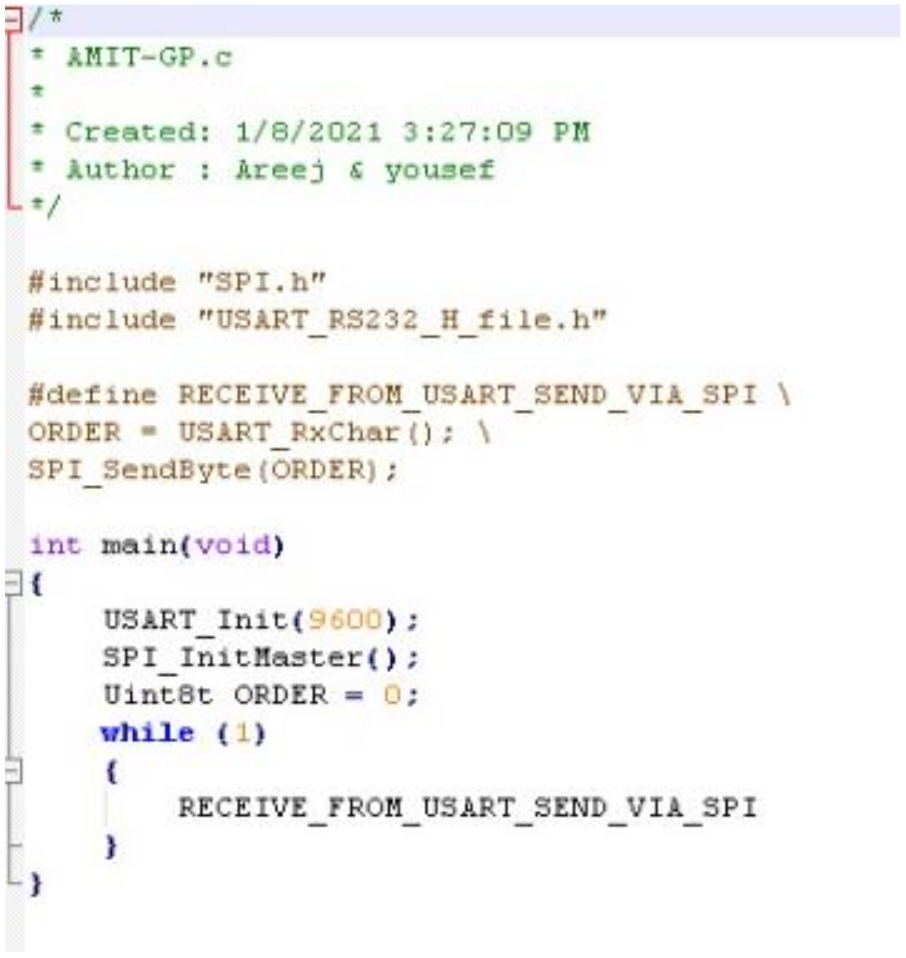
- The master microcontroller re-transmits the commands via SPI to the sub-microcontroller.
- The sub-microcontroller receives the commands via SPI protocol.
- The sub microcontroller performs the specified function for each command

Flow chart:



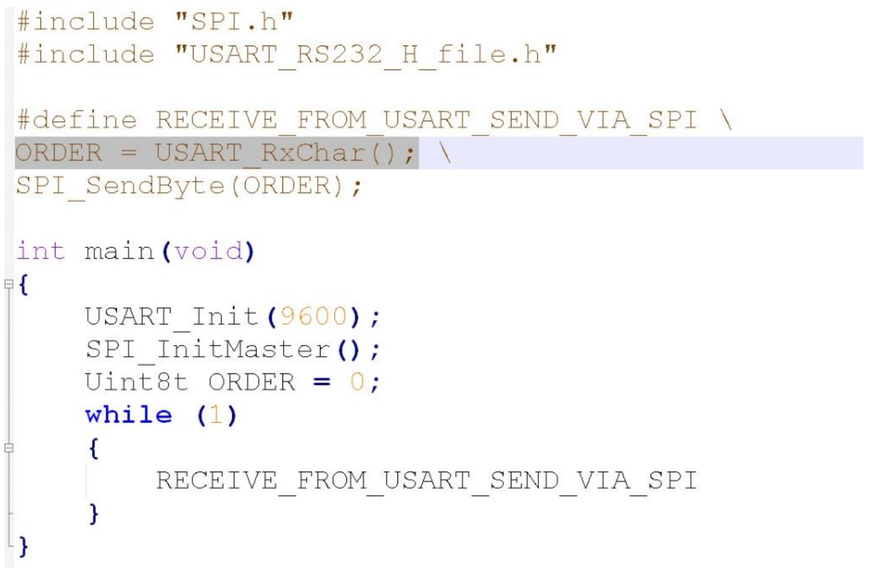
Programming Guide:

This .c file contain the main().c of the master ECU1 which receive the order from Bluetooth and send it to the actuator via SPI.it implanted by micros because it will be build by preprocessor and it will be faster.

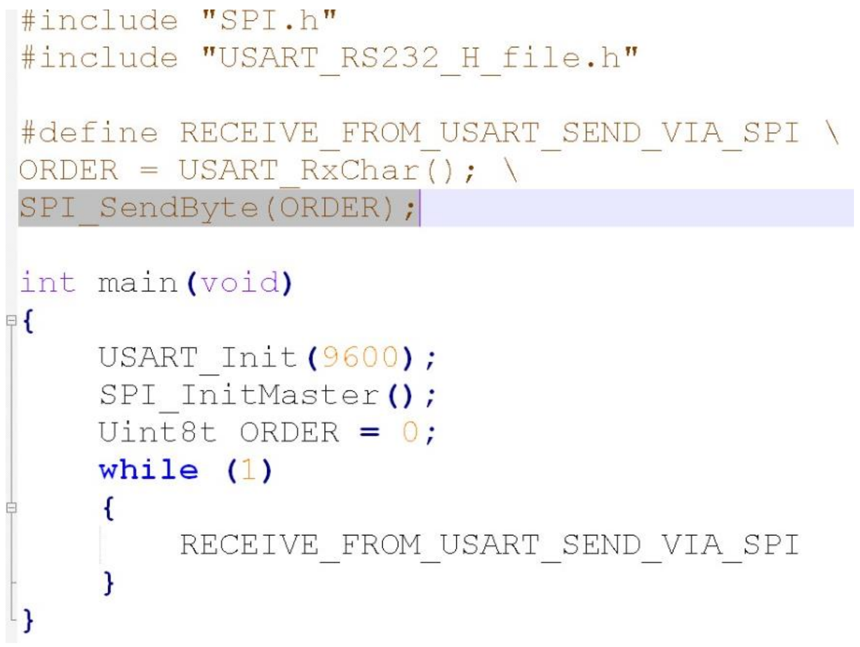


```
/*
 * AMIT-GP.c
 *
 * Created: 1/8/2021 3:27:09 PM
 * Author : Areej & yousef
 */

#include "SPI.h"
#include "USART_RS232_H_file.h"

#define RECEIVE_FROM_USART_SEND_VIA_SPI \
ORDER = USART_RxChar(); \
SPI_SendByte(ORDER);

int main(void)
{
    USART_Init(9600);
    SPI_InitMaster();
    Uint8t ORDER = 0;
    while (1)
    {
        RECEIVE_FROM_USART_SEND_VIA_SPI
    }
}
```

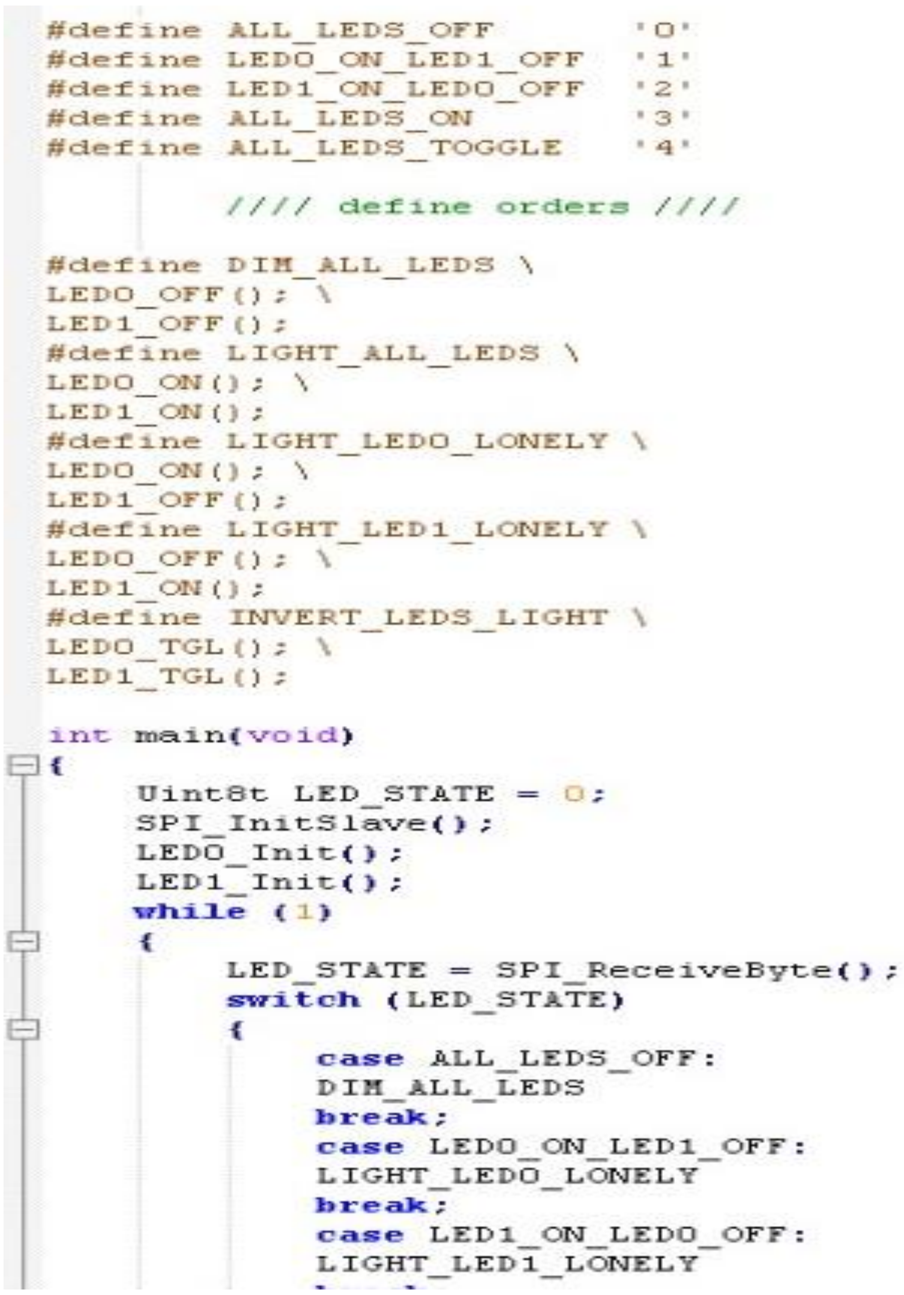
Figure 3master main()



```c
#include "SPI.h"
#include "USART_RS232_H_file.h"

#define RECEIVE_FROM_USART_SEND_VIA_SPI \
ORDER = USART_RxChar(); \
SPI_SendByte(ORDER);

int main(void)
{
    USART_Init(9600);
    SPI_InitMaster();
    Uint8t ORDER = 0;
    while (1)
    {
        RECEIVE_FROM_USART_SEND_VIA_SPI
    }
}
```

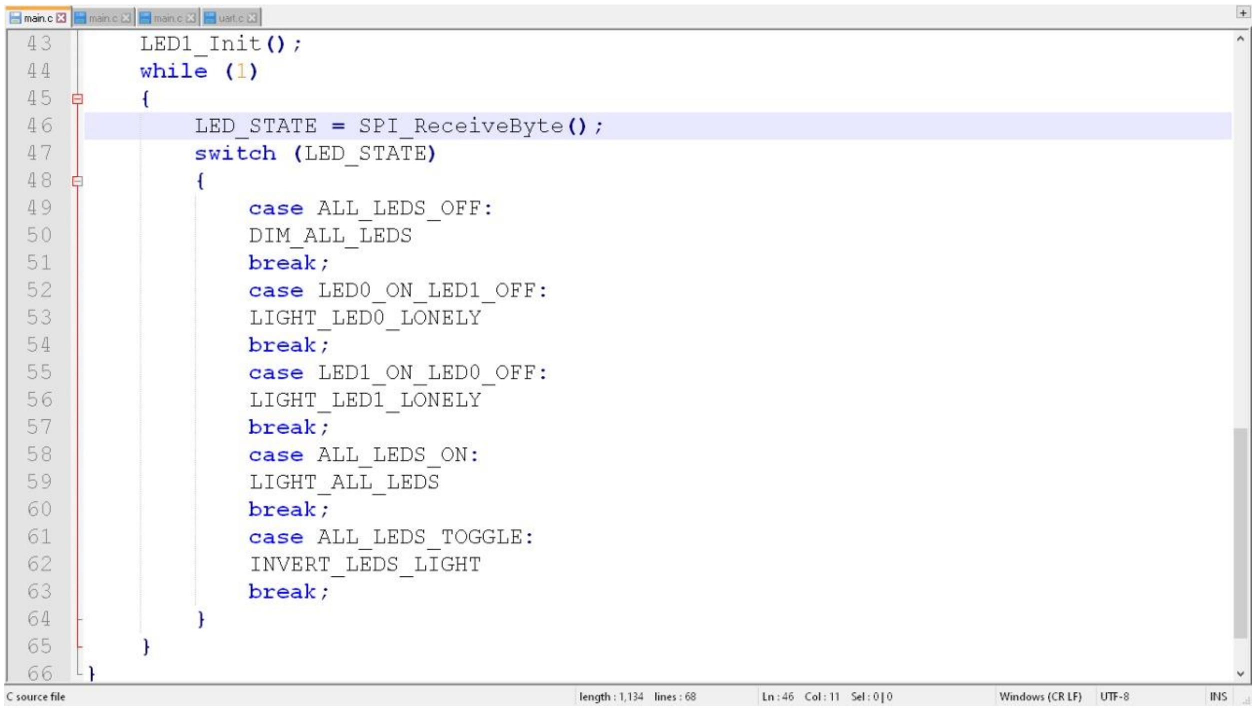


```c
#include "SPI.h"
#include "USART_RS232_H_file.h"

#define RECEIVE_FROM_USART_SEND_VIA_SPI \
ORDER = USART_RxChar(); \
SPI_SendByte(ORDER);

int main(void)
{
    USART_Init(9600);
    SPI_InitMaster();
    Uint8t ORDER = 0;
    while (1)
    {
        RECEIVE_FROM_USART_SEND_VIA_SPI
    }
}
```



```
#define ALL_LEDS_OFF        '0'
#define LED0_ON_LED1_OFF    '1'
#define LED1_ON_LED0_OFF    '2'
#define ALL_LEDS_ON         '3'
#define ALL_LEDS_TOGGLE     '4'

        //// define orders ////

#define DIM_ALL_LEDS \
LED0_OFF(); \
LED1_OFF();
#define LIGHT_ALL_LEDS \
LED0_ON(); \
LED1_ON();
#define LIGHT_LED0_LONELY \
LED0_ON(); \
LED1_OFF();
#define LIGHT_LED1_LONELY \
LED0_OFF(); \
LED1_ON();
#define INVERT_LEDS_LIGHT \
LED0_TGL(); \
LED1_TGL();

int main(void)
{
    Uint8t LED_STATE = 0;
    SPI_InitSlave();
    LED0_Init();
    LED1_Init();
    while (1)
    {
        LED_STATE = SPI_ReceiveByte();
        switch (LED_STATE)
        {
            case ALL_LEDS_OFF:
            DIM_ALL_LEDS
            break;
            case LED0_ON_LED1_OFF:
            LIGHT_LED0_LONELY
            break;
            case LED1_ON_LED0_OFF:
            LIGHT_LED1_LONELY
```

The next one is the actuator main(). Which predefined all LEDs operation into numbers too make it easier to deal with, then rename each case of function to increase readability. Then receiving the commend from the master and store it in LED state to compare the led state to match the proposed order.



```c
    LED1_Init();
    while (1)
    {
        LED_STATE = SPI_ReceiveByte();
        switch (LED_STATE)
        {
            case ALL_LEDS_OFF:
            DIM_ALL_LEDS
            break;
            case LED0_ON_LED1_OFF:
            LIGHT_LED0_LONELY
            break;
            case LED1_ON_LED0_OFF:
            LIGHT_LED1_LONELY
            break;
            case ALL_LEDS_ON:
            LIGHT_ALL_LEDS
            break;
            case ALL_LEDS_TOGGLE:
            INVERT_LEDS_LIGHT
            break;
        }
    }
}
```