LAB REPORT: EMBEDDED COMPUTING

LAB I: GETTING STARTED WITH GPIOS

Tailei WANG, Hao DENG

Electric Vehicle Propulsion and Control (E-PiCo)

ÉCOLE CENTRALE DE NANTES

January 2022

1 Objective

The objective of this first lab is start an application development process by trying to stay as close as possible to the hardware.

The development interface is deliberately reduced to a minimum (no IDE), because IDE may hide a significant part of the complexity of the development.

This lab is split in 2 parts:

- The first one focuses on getting started with a very simple application to use debugging tools (gdb).
- The second part focuses on the development of a driver to control a set of leds that are connected using the charlieplexing method.

2 Getting started

Question 1 *update the squeleton provided so that the green led is light on (without using the ST symbol provided). compile and flash your application.*

All the specific files are in the `sys/` folder:

- `startup_ARMCM4.c` the startup file (define the interrupt vector and the reset Handler)
- `startup_clock.c` configure the clock tree at startup.
- `stm32f303K8.ld` is the link script, used by the linker to allocate the code/data in memory
- `CMSIS/Device/ST/STM32F3xx/Include/stm32f303x8.h` contains the symbol definitions provided by ST for this chip. This is a 12000 lines files... you will have to use the search capability of your favorite editor.

The code is given as below:

```c
#include "stm32f3xx.h"
#include "pinAccess.h"

void wait() {
    volatile int i = 0;
    for (i = 0; i < 2000000; i++);
}

void setup() {
    pinMode(GPIOB,3,OUTPUT);
}

/* main function */
int main(void) {
    setup();
    /* Infinite loop */
    while (1) {
        /* Add application code here */
        digitalWrite(GPIOB,3,1);
        wait();
        digitalWrite(GPIOB,3,0);
        wait();
    }
}
```

After applying this code, it can be seen that the led turns on and off periodically.

Question 2 *update your application so that the led is light on only when the button (D6) is pushed. Your peripheral accesses should use symbol definitions.*

Firstly, considering the connection between Button D6 and Pin PB1 of the MCU, Pin PB1 is supposed to be configured as input. And the pull-up resistor is mandatory since the state is in high impedance when the button is released. The setup function is:

```c
void setup() {
    //input clock = 64MHz
    RCC->AHBENR |= RCC_AHBENR_GPIOBEN_Msk;
    __asm("nop");
    //pinMode(GPIOB,3,OUTPUT);
    GPIOB->MODER &= ~GPIO_MODER_MODER3_Msk;
    GPIOB->MODER |= 1 << GPIO_MODER_MODER3_Pos;
    //pinMode(GPIOB,1,INPUT_PULLUP);
    GPIOB->MODER &= ~GPIO_MODER_MODER1_Msk;
    GPIOB->PUPDR &= ~GPIO_PUPDR_PUPDR1_Msk;
    GPIOB->PUPDR |= 1 << GPIO_PUPDR_PUPDR1_Pos;
}
```

Secondly, IDR register is used to read the state of PB1 and give value to ODR of PB3. The main function is written as below:

```c
/* main function */
int main(void) {
    setup();
    /* Infinite loop */
    while (1) {
        /* Release the button */
        if((GPIOB->IDR & 0x2U)){
            GPIOB->ODR |= 1 << 3;   //LED on
        }
        /* Push the button */
        else {
            GPIOB->ODR &= ~(1 << 3);//LED off
        }
    }
}
```

Question 3 *add a breakpoint in your application, just after the C source line that lights the led on. Using GDB command only, update the state of the LED using only GDB commands.*

The management of the push button can be done by a non blocking finite state machine (with states `RELEASED`, `PUSHING`, `PUSHED` and `RELESASING`). The code is shown as below:

```c
enum PBState {RELEASED, PUSHING, PUSHED, RELEASING};
enum PBState managePB1(){
    static enum PBState state = RELEASED;
    switch(state) {
        case RELEASED: if((GPIOB->IDR & 0x2) == 0) state = PUSHING; break;
        case PUSHING: state = PUSHED; break;
        case PUSHED: if(GPIOB->IDR & 0x2) state = RELEASING; break;
        case RELEASING: state = RELEASED; break;
    }
    return state;
}
```

Question 4 *update your application so that the led is toggled each time we push the button.*

Based on the FSM introduced in Question 3, the main function can be developed to:

```
/* delay function */
void wait() {
    volatile int i = 0;
    for (i = 0; i < 2000000; i++);
}

/* main function */
int main(void) {
    setup();
    enum PBState state1;
    /* Infinite loop */
    while (1) {
        state1 = managePB1();
        if(state1 == PUSHING){
            if(GPIOB->ODR & 1 << 3 ){
                GPIOB->ODR &= ~(1 << 3);
                wait();
            }
            else{
                GPIOB->ODR |= 1 << 3;
                wait();
            }
        }
    }
}
```

3 Charlieplexing

Charlieplexing is a technique for driving a multiplexed display in which relatively few I/O pins on a microcontroller are used e.g. to drive an array of LEDs.

The lab board has 6 leds L0 to L5 connected to 3 I/Os, using the schematic in figure 1.

Only one led can light on at a time, and we have to use the 3-states outputs of the MCU pins (0, 1 or high impedance = Z).

Question 5 *What is the truth table of the system, with the 3 inputs from the MCU, and the 6 output leds?*

To light up one led at one single time, there are 6 states of this system. See Table 1.

3.1 Driving a single led

We can write a simple application that lights on the next led each time the button is pushed.

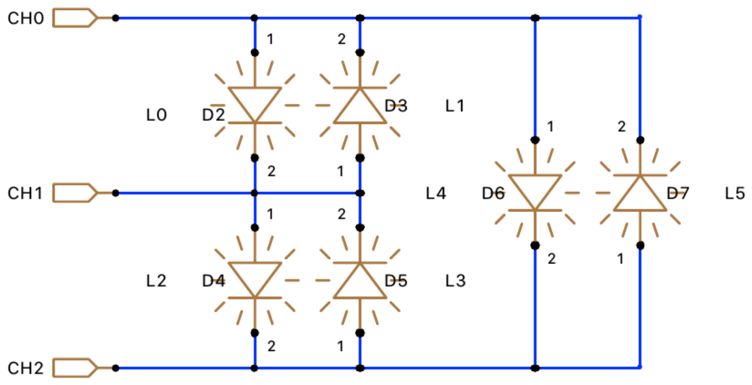

Figure 1: Schematic of the Charlieplexing leds of the lab board.

Table 1: Truth table of the Charlieplexing leds

LED (on)	D3	D7	D2	D6	D5	D4
CH0	0	0	1	1	Z	Z
CH1	1	Z	0	Z	0	1
CH2	Z	1	Z	0	1	0

Question 6 *implement and test the basic charlieplexing application. Take a particular attention to check the arguments at the start of the function (few values are OK for* `ledId`*).*

In terms of Table 1 and the schematic of NUCLEO F303K8, the full code is written as below. To get a high impedance state, the pin is supposed to be set to input mode without push-pull resistors.

```
void setLed(int ledId) {
    if (ledId == 2) {
        //CH0 PB7=1
        GPIOB->MODER &= ~GPIO_MODER_MODER7_Msk;
        GPIOB->MODER |= 1 << GPIO_MODER_MODER7_Pos;
        GPIOB->ODR |= 1 << 7;
        //CH1 PF0=0
        GPIOF->MODER &= ~GPIO_MODER_MODER0_Msk;
        GPIOF->MODER |= 1 << GPIO_MODER_MODER0_Pos;
        GPIOF->ODR &= 0;
        //CH2 PF1=Z
        GPIOF->MODER &= ~GPIO_MODER_MODER1_Msk;
    }
    else if (ledId == 3) {
        //CH0 PB7=0
        GPIOB->MODER &= ~GPIO_MODER_MODER7_Msk;
        GPIOB->MODER |= 1 << GPIO_MODER_MODER7_Pos;
        GPIOB->ODR &= 0 << 7;
        //CH1 PF0=1
        GPIOF->MODER &= ~GPIO_MODER_MODER0_Msk;
        GPIOF->MODER |= 1 << GPIO_MODER_MODER0_Pos;
        GPIOF->ODR |= 1;
        //CH2 PF1=Z
        GPIOF->MODER &= ~GPIO_MODER_MODER1_Msk;
    }
    else if (ledId == 4) {
        //CH0 PB7=Z
        GPIOB->MODER &= ~GPIO_MODER_MODER7_Msk;
        //CH1 PF0=1
        GPIOF->MODER &= ~GPIO_MODER_MODER0_Msk;
        GPIOF->MODER |= 1 << GPIO_MODER_MODER0_Pos;
        GPIOF->ODR |= 1;
        //CH2 PF1=0
        GPIOF->MODER &= ~GPIO_MODER_MODER1_Msk;
        GPIOF->MODER |= 1 << GPIO_MODER_MODER1_Pos;
        GPIOF->ODR &= 0 <<1;
    }
    else if (ledId == 5) {
        //CH0 PB7=Z
        GPIOB->MODER &= ~GPIO_MODER_MODER7_Msk;
        //CH1 PF0=0
        GPIOF->MODER &= ~GPIO_MODER_MODER0_Msk;
        GPIOF->MODER |= 1 << GPIO_MODER_MODER0_Pos;
        GPIOF->ODR &= 0;
        //CH2 PF1=1
        GPIOF->MODER &= ~GPIO_MODER_MODER1_Msk;
        GPIOF->MODER |= 1 << GPIO_MODER_MODER1_Pos;
        GPIOF->ODR |= 1 <<1;
    }
    else if (ledId ==6) {
        //CH0 PB7=1
        GPIOB->MODER &= ~GPIO_MODER_MODER7_Msk;
        GPIOB->MODER |= 1 << GPIO_MODER_MODER7_Pos;
        GPIOB->ODR |= 1 << 7;
        //CH1 PF0=Z
        GPIOF->MODER &= ~GPIO_MODER_MODER0_Msk;
        //CH2 PF1=0
        GPIOF->MODER &= ~GPIO_MODER_MODER1_Msk;
        GPIOF->MODER |= 1 << GPIO_MODER_MODER1_Pos;
        GPIOF->ODR &= 0 <<1;
    }
    else if (ledId ==7) {
        //CH0 PB7=0
        GPIOB->MODER &= ~GPIO_MODER_MODER7_Msk;
        GPIOB->MODER |= 1 << GPIO_MODER_MODER7_Pos;
        GPIOB->ODR &= 0 << 7;
        //CH1 PF0=Z
        GPIOF->MODER &= ~GPIO_MODER_MODER0_Msk;
        //CH2 PF1=1
        GPIOF->MODER &= ~GPIO_MODER_MODER1_Msk;
        GPIOF->MODER |= 1 << GPIO_MODER_MODER1_Pos;
        GPIOF->ODR |= 1 <<1;
    }
}
```

3.2 Driving the whole set of leds

We now want to make a fast scanning function that allows the display of several LEDs at the same time, thanks to the retinal persistence.

The function will be:

`void charlieplexing ( uint8_t mask );`

The first low 6 bits of `mask`) are associated to the state of one led. This function should call the `setLed()` function many times, but should not access the hardware directly.

The `charlieplexing()` function should be called very often in the main loop of the application. It may be implemented in 2 ways:

- the function updates all the leds at each call;
- the function updates only one led at each call (it will result in a better display quality, as each led will light the same time if the loop duration is constant).

Question 7 *Make a simple application that lights all the leds except L0 in the setup, and:*

- *shift the led off to the right when we push the button D6*
- *shift the led off the left when we push the button D5*

The full code is programmed as below:

```c
#include "stm32f3xx.h"
#include "pinAccess.h"

enum PBState {RELEASED, PUSHING, PUSHED, RELEASING};

void wait() {
    volatile int i = 0;
    for (i = 0; i < 2000000; i++);
}

void setLed(int ledId);

void setup() {
    RCC->AHBENR |= RCC_AHBENR_GPIOBEN_Msk;
    __asm("nop");
    RCC->AHBENR |= RCC_AHBENR_GPIOFEN_Msk;
    __asm("nop");
    //pinMode(GPIOB,1,INPUT_PULLUP);
    GPIOB->MODER &= ~GPIO_MODER_MODER1_Msk;
    GPIOB->PUPDR &= ~GPIO_PUPDR_PUPDR1_Msk;
    GPIOB->PUPDR |= 1 << GPIO_PUPDR_PUPDR1_Pos;
    //pinMode(GPIOB,6,INPUT_PULLUP);
    GPIOB->MODER &= ~GPIO_MODER_MODER6_Msk;
    GPIOB->PUPDR &= ~GPIO_PUPDR_PUPDR6_Msk;
    GPIOB->PUPDR |= 1 << GPIO_PUPDR_PUPDR6_Pos;
}

enum PBState managePB12(){
    static enum PBState state2 = RELEASED;
    switch(state2) {
        case RELEASED: if((GPIOB->IDR & (1<<6)) == 0) state2 = PUSHING; break;
        case PUSHING: state2 = PUSHED; break;
        case PUSHED: if(GPIOB->IDR & 0x64U) state2 = RELEASING; break;
        case RELEASING: state2 = RELEASED; break;
    }
    return state2;
}

enum PBState managePB1(){
    static enum PBState state = RELEASED;
    switch(state) {
        case RELEASED: if((GPIOB->IDR & 0x2) == 0) state = PUSHING; break;
        case PUSHING: state = PUSHED; break;
        case PUSHED: if(GPIOB->IDR & 0x2) state = RELEASING; break;
        case RELEASING: state = RELEASED; break;
    }
    return state;
}

/* main function */
int main(void) {
    setup();
    enum PBState state1;
    enum PBState state2;
    int ledId=2;
    /* Infinite loop */
    while (1) {
        state1 = managePB1();
        state2 = managePB12();
        if(state1 == PUSHED){
            if(ledId>7){
                ledId=2;
            }
            setLed(ledId);
            wait();
            ledId++;
        }
        if(state2 == PUSHED){
            if(ledId<2){
                ledId=7;
            }
            setLed(ledId);
            wait();
            ledId--;
        }
    }
}
```