Centrale Nantes

MAC : 2^{nd} Lab's Report
Getting started with GPIOs

1^{st} year Embedded Systems Engineering

By

EL KHAYDER Zakaria

SAOUTI Rayan

Professor

BRIDAY Mikael

November 26, 2023

Session

2023-2024

Made with LaTeX

Contents

List of Tables

Listings

1 Driving an LED

1.1 Simple enough, right?

Well, yes and no!

We are all used to Arduino, it is mostly the first-ever microcontroller we have used, and there is a strong reason why that is the case: Arduino provides a very simple and easy API to control the hardware, such as `digitalWrite`, `digitalRead` and `pinMode`. Such simple APIs do not come with STM32 out-of-the-box, instead, we have to directly mutate the value of specific memory locations to change the behavior of our controller.

So what should we do? Easy, we build our own ease-of-life utilities!

1.2 Taking it a level-higher

We can create the three functions `digitalRead`, `digitalWrite`, and `pinMode` ourselves, and thankfully, they were already provided 😄.

1.3 Hello, LED!

The next program is basically `Hello, World!` for micro-controllers, we are gonna make an LED blink! Yes, very exciting, We know 😂.

```
// This is the original Code provided with the project boilerplate

#include "stm32f3xx.h"
#include "pinAccess.h"

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

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

int main(void) {
  setup();
  while (1) {
    digitalWrite(GPIOB,3,1);
```

```
    wait();
    digitalWrite(GPIOB,3,0);
    wait();
  }
}
```

Listing 1: Blinking LED

Given we still haven't looked into STM32 internal timers and clock, we just use a loop to waste some of the CPU processing time, hence delaying the next code execution. This solution is neither efficient nor precise, but for now, it gets the job done.

1.4 Hello... Button?

Now, we will try to instead of just toggling the LED on and off, make it controlled by a push button.

Let's start with something easy, like turning it on when we click the button, and off when we release it.

Thanks to our `pinAccess.h` library, which defines a lot of functions that make our life easier, one of them is `digitalRead`, which simplifies - as the name suggests - reading from a digital input pin.

Reading an input value is as simple as writing this single line of code:

```
    int value = digitalRead(_port, _pin);
```

But not that fast bud... We have to tell the CPU that the corresponding pin in the corresponding port is an input, and if required, if it needs a pull-up or pull-down resistor.
Thankfully, this is also easy thanks to `pinAccess.h`, our lord and savior.

```
    pinMode(GPIOB, 1, INPUT_PULLUP);
```

In this example, we declare that the second pin of port B (PB1) is an input, with a pull-up resistor.

Given the button has a pull-up resistor, meaning it is using active low-logic, so if we need to check if the button is clicked, we need to invert the `digitalRead` return value.

1.5 A very cool trick

We can take advantage of C++ macro compiler substitution to clean a bit of our code, using #define directives.

```
    #define Button GPIOB, 1
    pinMode(Button, INPUT_PULLUP);
```

```

    pinMode(GPIOB, 1, INPUT_PULLUP);
```

The two chunks of code above are literally equivalent once the compiler goes through macro substitution.

We can clearly notice that the code on the left is much cleaner, scalable, and understandable than the one on the right.

1.6 Maybe light up the LED now?

Ok, Ok... Chill man, We are on it.

We already have our building blocks, the only thing left is putting them together.

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

#define LED GPIOB, 3
#define Button GPIOB, 1

void main(void) {
    pinMode(Button, INPUT_PULLUP);
    pinMode(LED, OUTPUT);

    while(1) {
        digitalWrite(LED, !digitalRead(Button));
    }
}
```

Listing 2: Simple LED

Simple, right? It is almost magical!

1.7 Making it bistable

Yeah, yeah, our code works and all, but wouldn't it be cooler if we could instead of turning on the LED when we push, to be able to toggle it on/off instead? So let's do that.

We are too lazy and already too late to explain how the system works, **understanding it is left as an exercise for the reader**.

We will be using multiple buttons in the next chapter, so while we are here, we might as well encapsulate the button logic inside a single class that we can reuse.

```
class Button
{
    enum State
    {
        RELEASED,
```

```
        PUSHING,
        PUSHED,
        RELEASING
    };

public:
    Button(GPIO_TypeDef *port, unsigned char pin) : _port(port),
                                                    _pin(pin)
    {
        pinMode(_port, _pin, INPUT_PULLUP);
    }

    void Update()
    {
        int value = digitalRead(_port, _pin);

        switch (_state)
        {
        case RELEASED:
            if (value == 0)
            {
                _state = PUSHING;
            }
            break;

        case PUSHING:
            _state = PUSHED;
            break;

        case PUSHED:
            if (value == 1)
            {
                _state = RELEASING;
            }
            break;

        case RELEASING:
            _state = RELEASED;
            break;

        default:
            break;
        }
    }

    int JustClicked()
    {
        return _state == PUSHING;
    }
```

```
55
56 private:
57     GPIO_TypeDef *_port;
58     unsigned char _pin;
59
60     State _state = State::RELEASED;
61 };
```

Listing 3: Button class

The `JustClicked` function will return true on the falling edge (not really), helping us detect when the button was just clicked, and run the code a single time.

Now the only thing left is to inset our newly crafted wisdom into our existing code.

```
1     #include "stm32f3xx.h"
2     #include "pinAccess.h"
3
4     #define LED GPIOB, 3
5     #define ButtonInput GPIOB, 1
6
7     /* Button class definition (redacted to save vertical space) */
8
9     void main(void) {
10        pinMode(LED, OUTPUT);
11        Button button(ButtonInput); // Class constructor takes care of pinMode
12
13        int ledState = 0;
14
15        while(1) {
16            if(button.JustClicked())
17                ledState = !ledState;
18
19            digitalWrite(LED, ledState);
20        }
21    }
```

Listing 4: Bistable LED control

2 Charlieplexing

In this chapter, we won't get into how Charlieplexing works. The most explanation provided is gonna be the next truth table. The reader is expected to do his own research (it is not very complicated already).

2.1 Truth Table

L_0	L_1	L_2	L_3	L_4	L_5	CH_0	CH_1	CH_2
1	0	0	0	0	0	1	0	Z
0	1	0	0	0	0	0	1	Z
0	0	1	0	0	0	Z	1	0
0	0	0	1	0	0	Z	0	1
0	0	0	0	1	0	1	Z	0
0	0	0	0	0	1	0	Z	1

Table 1: 6 LEDs Charlieplexing Truth Table

2.2 Driving a single led

We can write a function that lights up an LED given its index (from 0 to 5). We already put down the truth table above, the only thing left is translating it to C++ code.

```
#define CH0 GPIOB, 7
#define CH1 GPIOF, 0
#define CH2 GPIOF, 1

int setLed(int id)
{
    if (id > 5) // Verify input
        return -1;

    switch (id)
    {
    case 0:
        // CH0
        pinMode(CH0, OUTPUT);
        digitalWrite(CH0, 1);
        // CH1
        pinMode(CH1, OUTPUT);
        digitalWrite(CH1, 0);
        // CH2
        pinMode(CH2, INPUT);
        break;

    case 1:
        // CH0
        pinMode(CH0, OUTPUT);
        digitalWrite(CH0, 0);
        // CH1
        pinMode(CH1, OUTPUT);
        digitalWrite(CH1, 1);
        // CH2
        pinMode(CH2, INPUT);
        break;
```

```
33
34      case 2:
35          // CH0
36          pinMode(CH0, INPUT);
37          // CH1
38          pinMode(CH1, OUTPUT);
39          digitalWrite(CH1, 1);
40          // CH2
41          pinMode(CH2, OUTPUT);
42          digitalWrite(CH2, 0);
43          break;
44
45      case 3:
46          // CH0
47          pinMode(CH0, INPUT);
48          // CH1
49          pinMode(CH1, OUTPUT);
50          digitalWrite(CH1, 0);
51          // CH2
52          pinMode(CH2, OUTPUT);
53          digitalWrite(CH2, 1);
54          break;
55
56      case 4:
57          // CH0
58          pinMode(CH0, OUTPUT);
59          digitalWrite(CH0, 1);
60          // CH1
61          pinMode(CH1, INPUT);
62          // CH2
63          pinMode(CH2, OUTPUT);
64          digitalWrite(CH2, 0);
65          break;
66
67      case 5:
68          // CH0
69          pinMode(CH0, OUTPUT);
70          digitalWrite(CH0, 0);
71          // CH1
72          pinMode(CH1, INPUT);
73          // CH2
74          pinMode(CH2, OUTPUT);
75          digitalWrite(CH2, 1);
76          break;
77      }
78
79      return 0;
80  }
```

Listing 5: Charlieplexing a single LED

2.3 Driving the whole set of LEDs

We can create a function that will take a 6-bit mask as input (can't have only 6 bits, so we will take a full byte) and light up each LED where there is a 1 in its corresponding bit.

Actually, this is very simple to implement, we only have to loop from 0 to 5 (included) and check the bit in that index using masking operations.

```
void charlieplexing(uint8_t mask)
{
    for (uint8_t i = 0; i < 6; i++)
    {
        if (mask & (1U << i))
            setLed(I);
            // Adding a slight delay to make the LEDs light brighter
            volatile int j = 0;
            for (j = 0; j < 1000; j++);
    }
}
```

Listing 6: Charlieplexing with a mask

2.4 Going a step further. More Buttons!

We can create a simple program that lights all the LEDs except one, and we can change the position of the turned off led right or left by clicking on one of the two buttons:

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

```
//... The rest of the code, redacted to save space

int main(void)
{
    Button rightButton(ButtonRight), leftButton(ButtonLeft);

    uint8_t shift = 0;

    while (1)
    {
        rightButton.Update();
        leftButton.Update();

        if (rightButton.JustClicked())
        {
            if (shift == 5)
```

```
            {
                shift = 0;
            }
            else
            {
                shift++;
            }
        }
        else if (leftButton.JustClicked())
        {
            if (shift == 0)
            {
                shift = 5;
            }
            else
            {
                shift--;
            }
        }

        charlieplexing(0xFF & ~(1 << shift));
    }
}
```

Listing 7: The LED game

Resources

The code files, and this report's source code, are available on this GitHub repository: elkhayder/sec1-tp-mac