Microprocessor Systems

ELE 271

Laboratory 3:

BLINKER

Due 20FEB2023

By: Nick Bean , Toussaint Falangani , [Taylor Brookes](mailto:taylor.brookes@maine.edu)

**Introduction**

The purpose of this laboratory experiment is to learn how we can utilize general purpose input/output (GPIO) pins for digital output. GPIO is a standard interface used to connect microcontrollers to other electronic devices. For example, it can be used with sensors, diodes, displays and System on chip modules etc. For our purposes, we will be utilizing a logic analyzer and our microcontroller’s built in green LED connected to Port A pin 5 (PA.5) to observe its output. In our lab, we will also utilize the Reset Clock Control, the Advanced High-Performance Bus, MODER (the pin’s mode), OTYPER (push-pull or open-drain), OSPEEDR (how quickly the device responds to an input voltage), PUPDR (pull-up or pull-down, which defines the default voltage of the pin), and ODR (output data register).

**Part 1**

The goal of part 1 is to enable the GPIO PA.5 pin and turn on and off the LED by toggling bit 5 of GPIOA output data register. In order for us to enable GPIO PORT A, we use the statement from pg 357 of the textbook using the example as reference.

RCC -> AHB2ERN |= RCC\_AHB2ENR\_GPIOAEN;

RCC -> AHB2ERN |= 0x00000001;

RCC stands for Reset Clock Control. It controls the internal clock to each peripheral on the microcontroller. AHB2ERN stands for Advanced High-performance Bus 2 Enable Register. It allows us to use the functions within the bus. Setting RCC -> AHB2ERN or-equals to 0x00000001 enables us to use port A, which is where PA.5 is located. The arrow indicates that AHB2ERN is a subset inside of RCC that we are addressing.

GPIOA -> MODER &= ~(3UL<<10);

GPIOA -> MODER |= 1UL<<10;

MODER sets PA.5 as a digital output. This means that when something occurs, PA.5 will change its state (voltage) due to the input of something. The first line clears the mode bits on PA.5 (which is the 10 in the code, because each pin has two bits associated with it). The second line sets the two bits as a digital output. ~3UL means the complementary of 3 unsigned long. This sets the bits to 00 initially. 1UL then sets the two bits to 01. Moder has 4 different outputs, our focus is on digital output:

* (00) **Input mode**: detects whether an external voltage signal is higher or lower than a predetermined threshold.
* (01) **Output mode:** Controls the voltage on the pin
* (10) **Analog mode :**  performs digital to analog or analog to digital conversion
* (11) **Alternate function mode :** complex functions such as SPI data pin, and timer PWM output

GPIOA -> OTYPER &= ~(1UL<<5);

OTYPER defines the configuration of PA.5. It can be push-pull or open-drain. According to the Reference Manual,

* 0 = Push-Pull
* 1 = Open-Drain

Push-pull mode allows the pin to supply and absorb current. However, a GPIO pin in open-drain (also called collector) mode can only absorb current. By default the pin will be in push-pull mode, but we want to make sure that it is set to 0 so we manually set it. We use (5) instead of (10) because there is only 1 bit associated with this state, instead of 2 for the other ones.

GPIOA -> OSPEEDR &= ~(3UL<<10);

OSPEEDR controls how quickly the pin responds to inputs or outputs. The values are:

* 00 = Low Speed
* 01 = Medium Speed
* 10 = Fast Speed
* 11 = High Speed

The lower the speed, the less power is used, and vice versa for higher speeds. By default, in our class, we use Low Speed. ~(3<<10) sets the two bits of PA.5 to 00 because ~(3) = ~(11) = (00).

GPIOA -> PUPDR &= ~(3UL<<10);

PUPDR is how we select pull-up or pull-down. A Pull-up resistor is used to make the default state of the digital pin as High or to the logic level (in the above image it is 5V) and a Pull-Down resistor does exactly opposite, it makes the default state of the digital pin as Low (0V). We’re told to set it to no pull-up or pull-down, so we set the two bits to 00. The bit configuration for the states are:

* 00 = no pull-up or pull-down
* 01 = pull-up
* 10 = pull-down
* 11 = reserved

GPIOA -> ODR |= 1<<5;

Inside the main function, we declare the ODR. ODR is the output data register. We set it to 1 on PA.5 to enable the register, as it only has 1 bit associated with it. In a push-pull setting, if the bit value is 1, the output voltage on its corresponding GPIO pin is high; if the bit value is 0, the output voltage then is low.

while (1) {

int i;

for (i = 0; i < 1000000; i++);

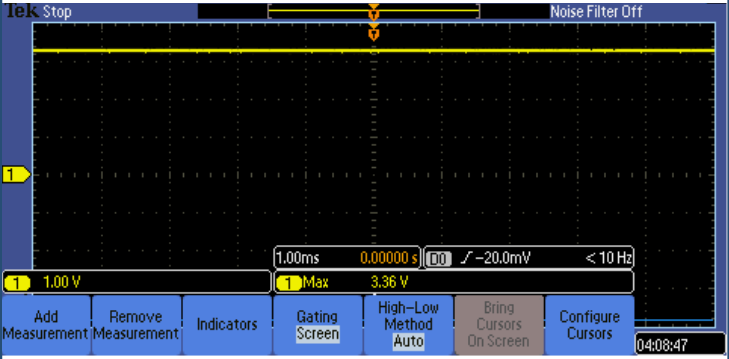
GPIOA -> ODR ^= 1UL<<5;

}

Our main while loop is how we toggle the LED. We first initialize the index variable “i”, then increment it by one until it reaches 1,000,000. This is the delay for each time the LED blinks. the caret operator ^= is XOR, exclusive-OR. It will be false when none of the bits or all of the bits are on, and true otherwise. This is our “toggle” command, which blinks the LED once by turning it on then off immediately afterwards. When ODR is high, the LED is off because both bits are 1. when ODR is low, the LED is on because it satisfies the XOR bitwise operator.

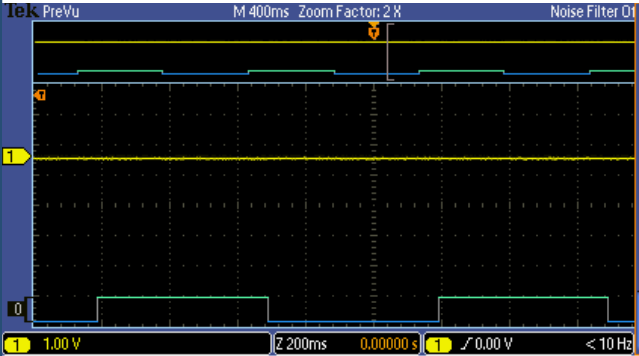
**Part 2**

The goal of part two is to determine the voltage level of a high output on a GPIO pin by connecting PA.5 to the oscilloscope and measuring the voltage. To do this, we used the oscilloscope to display the output voltage. As you can see below in the digital display, the output voltage remains constant at 3.36V. This makes sense because there are only two possible voltages for the stm32, 5V via USB and 3.3V via a rail of the 5V line.



**Part 3**

The goal of part 3 is to Adjust the count value the signal on PA.5 has an approximately 1 second period, and then use the logic analyzer to display the signal and validate your result. We found that in part 2, setting the count value to 1,000,000 gave us an approximately 8 second period. So, to create a 1 second period, we simply set the count value from 1,000,000 to 125,000. To confirm this, we used the oscilloscope. Each vertical line is an increment of 200ms, as displayed. Counting the number boxes between the beginning of each HIGH signal (when it changes from high to low then high again), we come up to 5 boxes. 5 \* 200ms = 1sec.



while (1) {

int i;

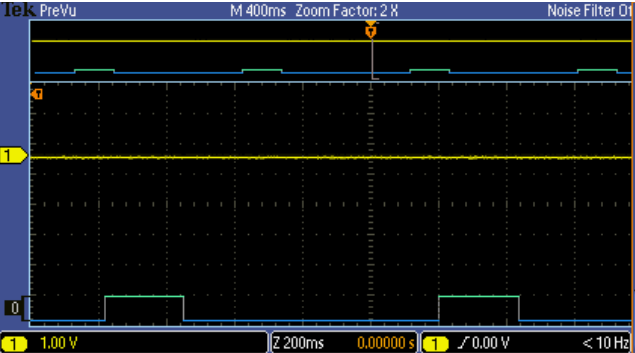
for (i = 0; i < 125000; i++);

GPIOA -> ODR ^= 1UL<<5;

}

**Part 4**

The goal of part 4 is to modify the code to vary the duty cycle of the output signal. As per the instruction example, our goal is to maintain a 1 second period while turning the LED ON for 0.25 seconds and OFF for 0.75 seconds. We created a moseDot() method to do this for us, and then called it in the main while loop. GPIOA -> ODR |= (1UL<<5) turns the LED on, while GPIOA -> ODR &= ~(1UL<<5) turns the LED off. This occurs because PA.5 is the ODR pin we are using. “|=” will always be true because we are using 1UL, which will turn on the LED< while “&= ~()” will always be false because it compares ODR to 0 and uses the bitwise AND operator. Unintuitively, using the for loop to count to 125,000 / 2 and then 125,000 \* 3 / 2 gives us a period of 1 second. We first turn on the LED, then count for 0.25sec, then turn off the LED, then count for 0.75sec to force the LED to stay off for the desired amount of time.



void morseDot () {

int i;

GPIOA -> ODR |= (1UL<<5);

for (i = 0; i < 125000 / 2; i++);

GPIOA -> ODR &= ~(1UL<<5);

for (i = 0; i < 125000 \* 3 / 2; i++);

}

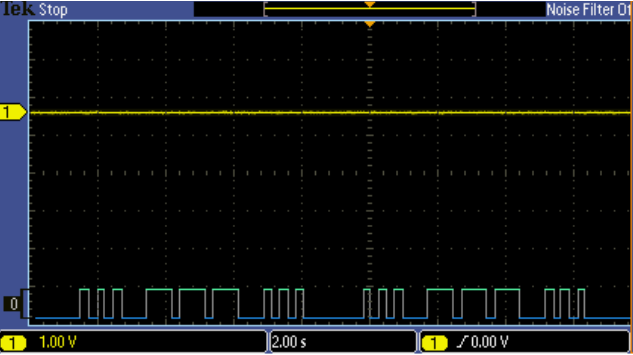
**Part 5**

The goal of part 5 is to blink the LED in Morse code SOS. Before blinking the LED, we first have to understand the blinking pattern of Morse code. Below is a list of the time units used in morse code:

* dot = one unit on
* dash = three units on
* between dashes and dots = one unit off
* between letters = three units off
* between words = seven units off

S is 3 dots, then O is 3 dashes, then S is another 3 dots. Between each letter we want to run our betweenLetters() method, then at the end of the SOS message we want to run our betweenWords() method. To do this, we created methods for each function, excluding the function “between dashes and dots”, because we can implement this in the “dot” and “dash” methods, making the code easier to understand. Interestingly, we chose to understand what value the code counts up until in this part to better understand what is going on “under the hood”. Our oscilloscope image shows that our program functions as it should. Included after the image is the code for our methods and the main while loop that repeats the SOS message. In table below, our betweenLetters and betweenWords methods are 2 and 6 units respectively because after each dot or dash we include a 1 unit off signal, which is run before betweenLetters and betweenWords, generating our needed 3 and 7 unit off signals in total, respectively.

| method name | hex count value | decimal count value | seconds on/off |
| --- | --- | --- | --- |
| morseDot | 0x0000 F424 | 62,500 | 0.25/0.25 |
| morseDash | 0x0002 DC6C | 187,500 | 0.75/0.25 |
| betweenLetters | 0x0001 E848 | 125,000 | 0/0.5 |
| betweenWords | 0x0005 B8D7 | 375,000 | 0/1.5 |



void morseDot () {

int i;

GPIOA -> ODR |= (1UL<<5);

for (i = 0; i < 125000 / 2; i++);

GPIOA -> ODR &= ~(1UL<<5);

for (i = 0; i < 125000 / 2; i++);

}

void morseDash () {

int i;

GPIOA -> ODR |= (1UL<<5);

for (i = 0; i < 125000 \* 3 / 2; i++);

GPIOA -> ODR &= ~(1UL<<5);

for (i = 0; i < 125000 / 2; i++);

}

void betweenLetter () {

int i;

GPIOA -> ODR &= ~(1UL<<5);

for (i = 0; i < 125000 \* 2 / 2; i++);

}

void betweenWord () {

int i;

GPIOA -> ODR &= ~(1UL<<5);

for (i = 0; i < 125000 \* 6 / 2; i++);

}

while (1) {

morseDot();

morseDot();

morseDot();

betweenLetter();

morseDash();

morseDash();

morseDash();

betweenLetter();

morseDot();

morseDot();

morseDot();

betweenWord();

}

**Conclusion**

Our goal in this lab was to learn about the GPIO pins for digital output in addition to utilizing the logic analyzer without stm32’s LED on PA.5. First, we initialized all the functions necessary to use PA.5 as an output. Next, we blinked the LED by comparing the ODR bit to 1 and then utilized a For-loop to count up to a certain number to delay the next blink. This is repeated indefinitely due to a dead while loop. In part 2, we displayed the output voltage of PA.5 as 3.36V. In part 3, we changed the blink period from 8 seconds to 1 second by changing the value of the counter in the for loop from 1,000,000 to 125,000. In part 4, we modified the on-off period to be 0.25sec on and 0.75sec off by using modified for-loop counters of 125,000 / 2 and 125,000 \* 3 / 2 respectively, in addition to changing the operator from ^= to |= and &= ~() respectively. Intuitively, 125,000 producing a 1 second delay while 125,000 / 2 producing a 0.25 second delay doesn’t make sense, and we aren’t sure why this is the case. When the for loop counting to 125,00 is implemented in its own function, it is for a duration of 0.5 seconds. However, when it is directly implemented in the dead while loop, it is for a duration of 1 second. In part 5, we created methods to output the SOS message via our LED. This was accomplished by modifying the on and off LED code to suit the time units of the message itself of dots, dashes, time between letters, and time between words.