# ELEC335 Lab5

**December 9, 2020**

The objective of this lab is to get you communicate with MCU from PC, and utilize bidirectional data transmission and parsing. You will use C language for the problems unless some parts require *inline assembly*. Use *blinky* project from stm32g0 repo as the starting point for your problems.

## Submission

You should submit the following items organized in a folder:

- **Lab report** - Written in English. Proper cover page, intro, problems, flow charts, block diagrams, schematic diagrams, comments, and theoretical, mathematical work, simulation vs.
- **Source files** - should have proper comments and pin connections if there are any external components.
- **Video link** - A link to video demonstration of your lab (this video also should show yourself.)
- **Elf files** - generated elf file with **debug information**.

Compress the folder to a zip file, and submit that way. Each problem should be in a separate folder. Example folder structure is given below.

```
name_lastname_number_labX/
  name_lastname_report_labX.pdf
  video_link.txt
  problem1/
    problem1.s/c
    problem1.elf
    ...
  problem2/
    problem2.s/c
    problem2.elf
    ...
```

## Problems

### Problem 1

In this problem, you will connect your board to the PC using UART protocol. For this you will need to create an initialization routine for UART, and create send and receive functions. **You should not use interrupts for this assignment.**

Basically create two functions as given below

```
void uart_tx(uint8_t c);
uint8_t uart_rx(void);
```

and in your main loop, call them like

```
while(1) {
  uart_tx(uart_rx());
}
```

## Problem 2

In this problem, you will implement a PWM signal and drive an external LED using varying duty cycles. Your LED should display a triangular pattern, meaning the brightness should linearly increase and decrease in 1 second.

Bonus - After you setup and get your PWM working with the LED, replace the LED with a speaker and see what happens. (Make sure to keep the series resister)

## Problem 3

In this problem, you will implement a PWM signal and drive an LED at different speeds. You will use keypad to set the duty cycle. Pressing 10# will set the duty cycle to 10% and 90# will set the duty cycle to 90%. Any unusal presses should be ignored. (i.e. 9F#)

- The current duty cycle should print every 2 seconds using UART
- The new duty cycle should also print after a button presses.
- Your buttons should not bounce.

## Appendix A - Setting up UART and creating a print() function

Pins PA2 and PA3 are connected to the ST/Link Debugge IC on the back which gets passed to the PC over USB as a Virtual COM port. This can be seen from the schematic of the board. First, we need to determine the USART modules PA2 and PA3 are connected to. From stm32g031k8 datasheet, we can see that we have two USART modules named USART1 and USART2.

Looking at Table 13 - Port A alternate function mapping, we can also see that PA2 and PA3 are connected to USART2 module using Alternate Function 1 mode (AF1). Now that we have the relevant info, we can start initializing steps. Check the RCC, GPIO and USART sections from rm0444 reference manual for exact registers and relevant bits.

1. enable GPIOA and USART2 module clocks from RCC
2. enable and setup pins PA2 and PA3 as alternate function for USART2 module
3. enable receive and transmit from USART2 module
4. set baud rate to 9600 bps assuming clock is running at 16Mhz
5. enable uart module

At this point, if we open a serial port from PC, and send a character from MCU, we should be able to see. Let's write some helpers for this purpose.

6. implement a `printChar` function given as below. Now you can just call `printChar('a');` from your main loop and this should be displayed on your PC terminal
7. implement a `_print` function that will call the `printChar` function for the given string in a for loop given as below. Now you can call `_print(0, "Hello\n", 6);` and it should be displayed on your PC terminal
8. implement a `print` function that will calculate the number of characters in a string and call `_print` function given as below. Now you can just call `print("Hello\n");` and it should be displayed on your PC terminal

```
// step 6 skeleton
void printChar(uint8_t c) {
  // write c to the transmit data register
  // wait until transmit is complete
}
```

```
// step 7 skeleton
int _print(int f, char *ptr, int len)
{
  (void)f; // don't do anthing with f
  // call printChar within a for loop len times
  return len; // return length
}
```

```
// step 8 skeleton
void print(char *s) {
// count number of characters in s string until a null byte comes `\0`
  _print(0, s, length);
}
```

In order to connect to your MC and see these, you can open a serial port from your PC. When you connect the device, it should create a COM port for Windows and /dev/tty* for Unix-like systems (ACM / USB). You can use STM32CubeIDE.

1. From the Console window, press the window icon with plus symbol, then choose *Command Shell Console*
2. For connection type, choose *Serial Port*
3. Hit new and create a connection. Give a connection name, choose correct uart setup, and choose serial port. If your device is connected a COM port should show up here. Choose that.
4. Choose encoding as UTF-8 and hit OK.

You should now have a serial connection to your board.

## Appendix B - Setting up PWM

1. First, we need to pick a pin which is capable of timer functionality. From the stm32g031k8 datasheet pinout, pick one that has a TIMx_CHy functionality listed and from Table 13/14 find the AF number. In this case x is the timer module that we will use and y is the channel that we will activate for PWM.
2. Setup `TIMx` module like you would normally do, but choose PWM mode 1 from `CCMR` register for the relevant y channel. (`CCMR1` has control over channels 1/2, and `CCMR2` has control over channels 3/4)
3. Enable the Capture/Compare output from relevant bit in CCER register.
4. `ARR` register is the period for PWM and `CCRy` register is the duty cycyle for the PWM.
5. Everytime full period happens, it should interrupt and you should decide on the next duty cycle to send.
6. In the interrupt routine, make sure to clear the status register of the timer