CEC322 - Section 1

Lab 2 - Joystick and LEDs with STM32CubeMX

Lab start date: 01/30/19

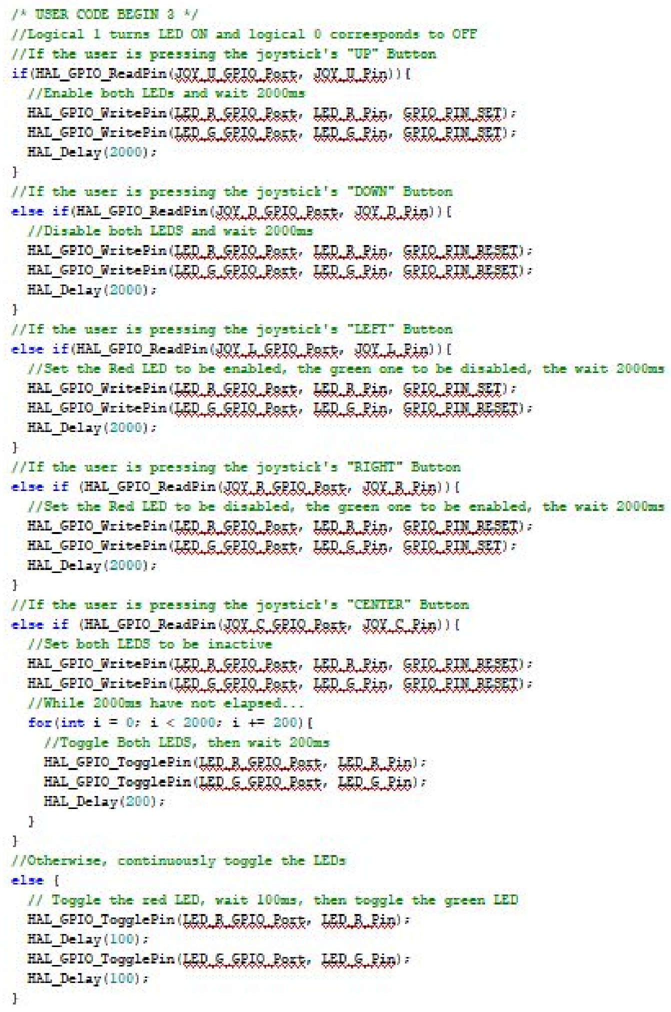
Report data: 02/06/19

Luis Mora and Tyler Wise

Introduction:

This lab's purpose is to introduce students to building and running code on the microprocessor. Additionally, this lab introduces students to using the GPIO functionality of the microprocessor and how to get a project setup for this purpose using STM32CubeMX. In order to complete this, students need a working installation of Keil, STM32CubeMX, and all of the drivers and support software required to connect the microprocessor to Keil.

Code Snippet:



```c
/* USER CODE BEGIN 3 */
//Logical 1 turns LED ON and logical 0 corresponds to OFF
//If the user is pressing the joystick's "UP" Button
if(HAL_GPIO_ReadPin(JOY_U_GPIO_Port, JOY_U_Pin)){
  //Enable both LEDs and wait 2000ms
  HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin, GPIO_PIN_SET);
  HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_SET);
  HAL_Delay(2000);
}
//If the user is pressing the joystick's "DOWN" Button
else if(HAL_GPIO_ReadPin(JOY_D_GPIO_Port, JOY_D_Pin)){
  //Disable both LEDS and wait 2000ms
  HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_RESET);
  HAL_Delay(2000);
}
//If the user is pressing the joystick's "LEFT" Button
else if(HAL_GPIO_ReadPin(JOY_L_GPIO_Port, JOY_L_Pin)){
  //Set the Red LED to be enabled, the green one to be disabled, the wait 2000ms
  HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin, GPIO_PIN_SET);
  HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_RESET);
  HAL_Delay(2000);
}
//If the user is pressing the joystick's "RIGHT" Button
else if (HAL_GPIO_ReadPin(JOY_R_GPIO_Port, JOY_R_Pin)){
  //Set the Red LED to be disabled, the green one to be enabled, the wait 2000ms
  HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_SET);
  HAL_Delay(2000);
}
//If the user is pressing the joystick's "CENTER" Button
else if (HAL_GPIO_ReadPin(JOY_C_GPIO_Port, JOY_C_Pin)){
  //Set both LEDS to be inactive
  HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_RESET);
  //While 2000ms have not elapsed...
  for(int i = 0; i < 2000; i += 200){
    //Toggle Both LEDS, then wait 200ms
    HAL_GPIO_TogglePin(LED_R_GPIO_Port, LED_R_Pin);
    HAL_GPIO_TogglePin(LED_G_GPIO_Port, LED_G_Pin);
    HAL_Delay(200);
  }
}
//Otherwise, continuously toggle the LEDs
else {
  // Toggle the red LED, wait 100ms, then toggle the green LED
  HAL_GPIO_TogglePin(LED_R_GPIO_Port, LED_R_Pin);
  HAL_Delay(100);
  HAL_GPIO_TogglePin(LED_G_GPIO_Port, LED_G_Pin);
  HAL_Delay(100);
}
```

Results:
One of the functions used in this lab is **HAL_GPIO_TogglePin**. The important bit of code in this function is **GPIOx->ODR ^= GPIO_Pin;** What this does is set the value of the output data register to be the result of XORing it's previous value with the mask specified. This results in toggling the pin(s) on the port between active and inactive based on the supplied GPIO_Pin mask. This operation is used because the result of a single bit XORed with a 1, which is the value of the mask, results in getting the opposite value of that bit. It also works very well because it only affects the single bit without modifying any other data of the register.

For this lab, it is extremely important that the GPIO pins for the LEDs to be set to "push-pull" mode. This is critical because the absence of an external current renders the open-drain mode useless. Push-Pull allows the output pin to read HIGH and LOW depending on the logical 1 or 0, respectively, supplied by the program. An open-drain circuit will pull the current to the ground, if the logical output is a 0, as to have the output pin read a 0. An open-drain circuit will read a logical 1 as floating: this means it cannot supply any current and is in a state of high impedance. The LEDs in our board would never turn on if an open-drain method is approached.

It is also crucial for the GPIO pins used for the joystick keys on the STM32L476VG boards to be set to "pull-down" mode. The benefit of doing this is that the value output by the GPIO port is either a "high" or "low" voltage rather than a floating value that is very susceptible to noise in the circuitry which can also cause damage do the chips. Additionally, pull down/ pull up circuitry allows us to control how our program will read an input of high impedance. For our lab, we used a pull down resistor as it allowed this state of no input to be read as a logical 0 by our program. This fact is very convenient when thinking about the logical design of the application to run on the board. With that said, it is not necessary to explicitly tie the center button of the joystick to a pull-down resistor as it already is, internally, by default to ensure the microprocessor's default functionality of the center key being a "wake-up" button.