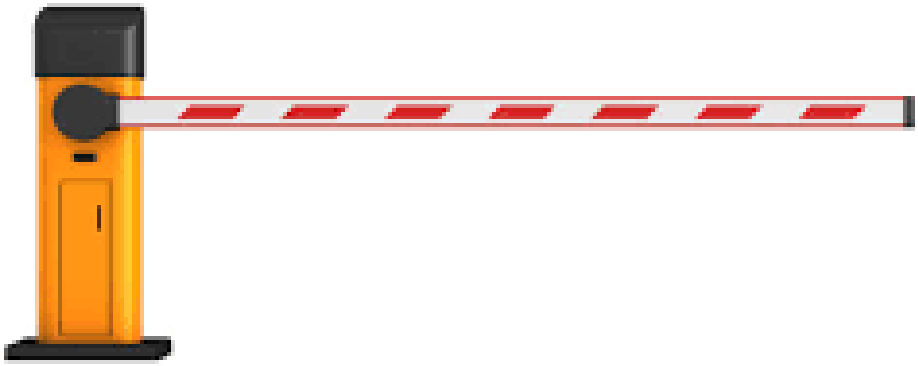
Automatic Parking Gate

Francisco Irazaba

CPE 316 | Section 1 | Fall Quarter
Professor Hummel
12/9/24



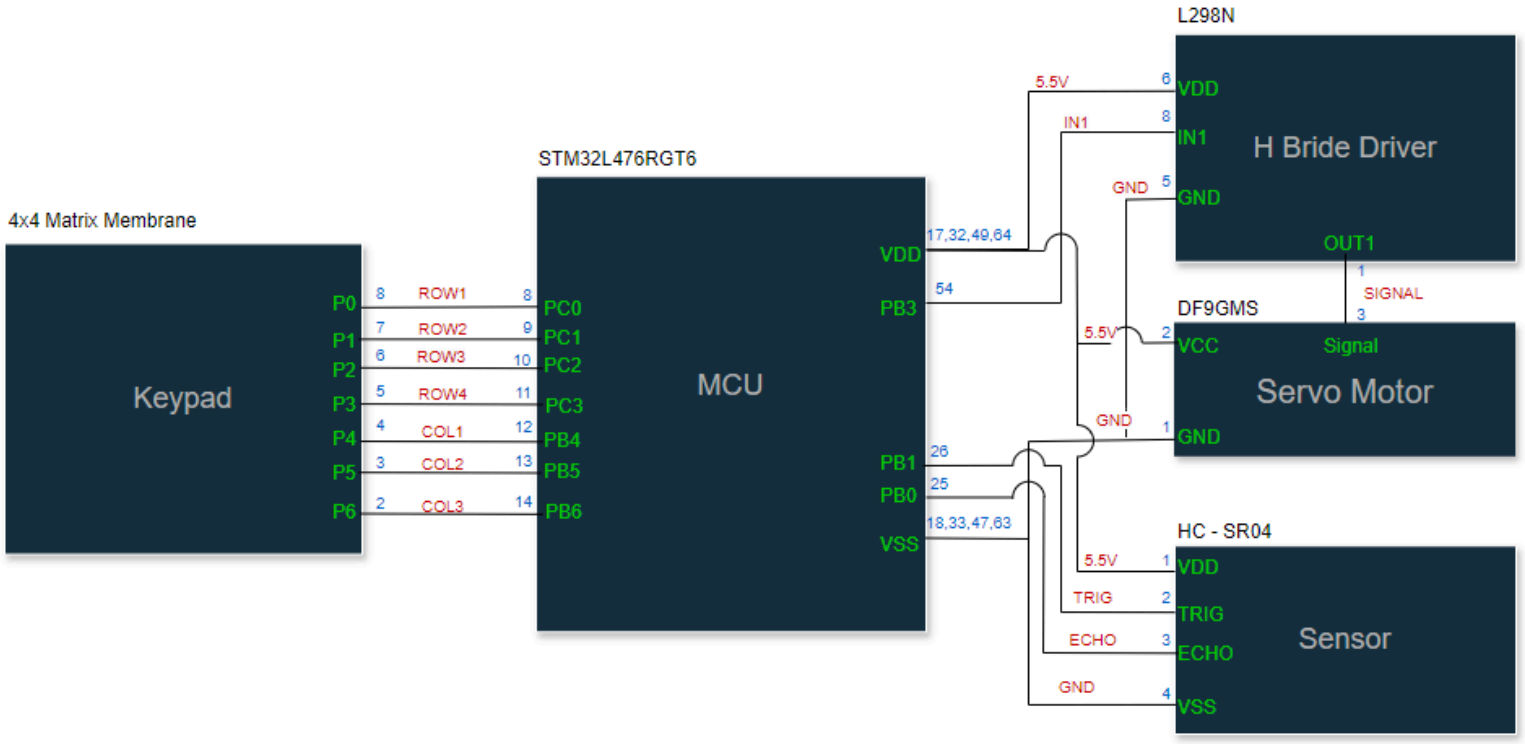
Behavior Description

The automatic parking gate replicates a secure access mechanism by generating a unique four-digit passcode for each user. When a vehicle is detected via the ultrasonic sensor, the system prompts the user to enter the displayed passcode. If the entered passcode is correct, the gate opens to grant access; otherwise, access is denied, and the user is prompted to re-enter the passcode. A new passcode is generated for every detection to ensure security and simulate real-world functionality.

System Specification

Specification	Details
Microcontroller	STM32L476RGT6
Dual H-Bridge Motor Driver	L298N
Ultrasonic Sensor	HC–SR04
Power Supply Voltage	3.3V DC (USB or external adapter)
Power Consumption	265.8 mW
Clock Frequency (MCU)	48MHz
Sensor Detection Range	8.84cm
UART Baud Rate	115,200 baud
User Interface	4x3 Keypad
Connectors	USB for data connection
Supported Protocols	USB for remote control

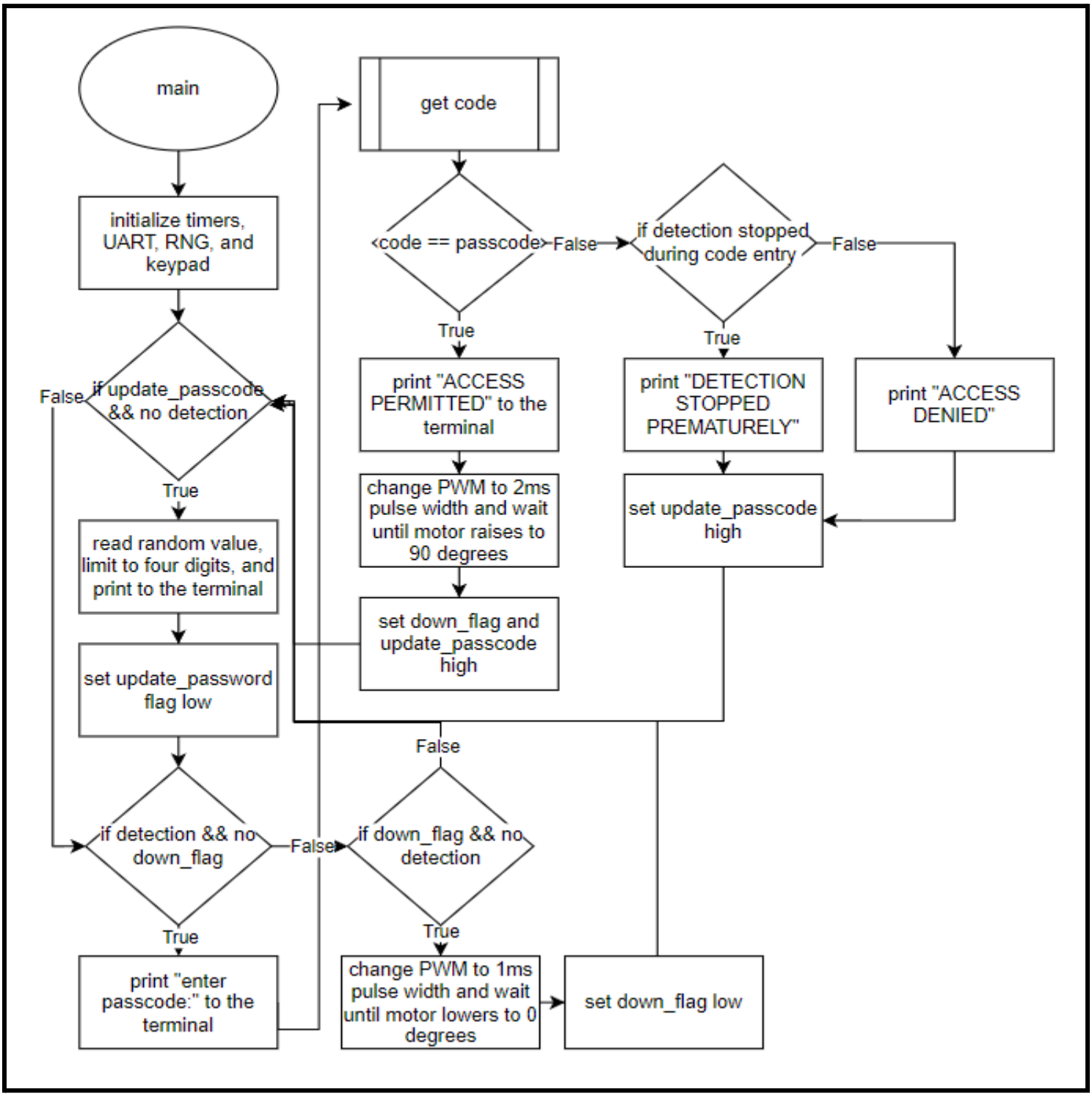
System Schematic

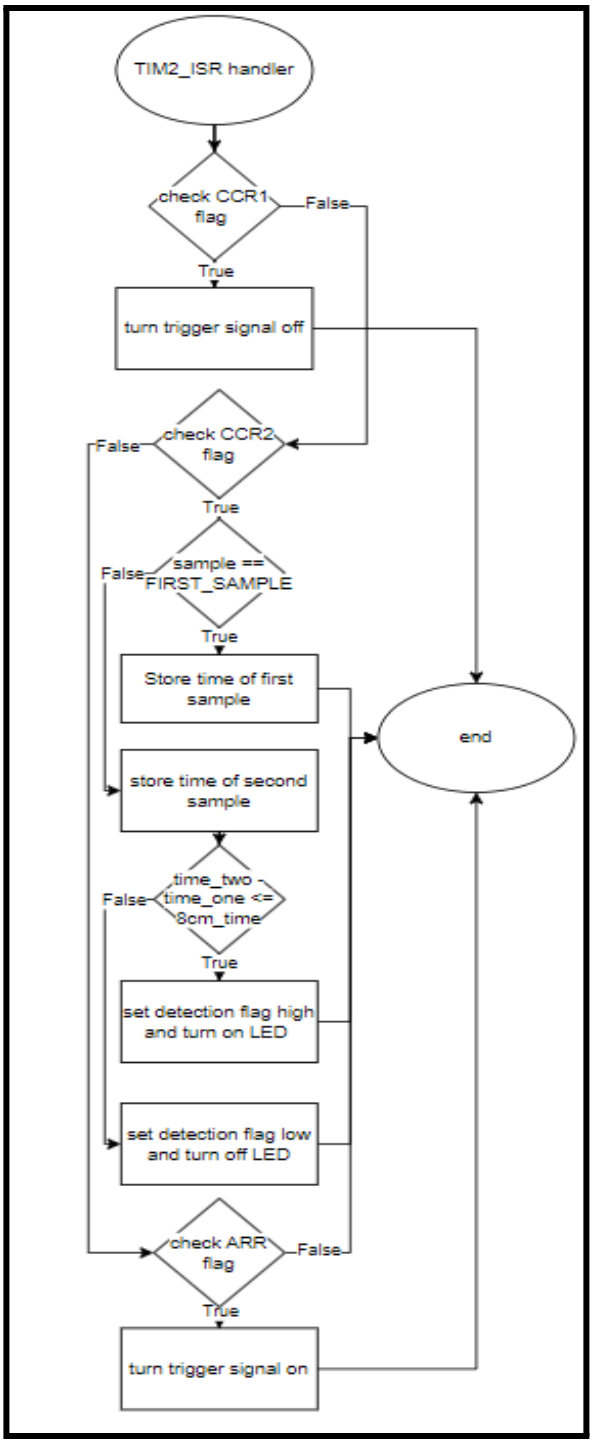


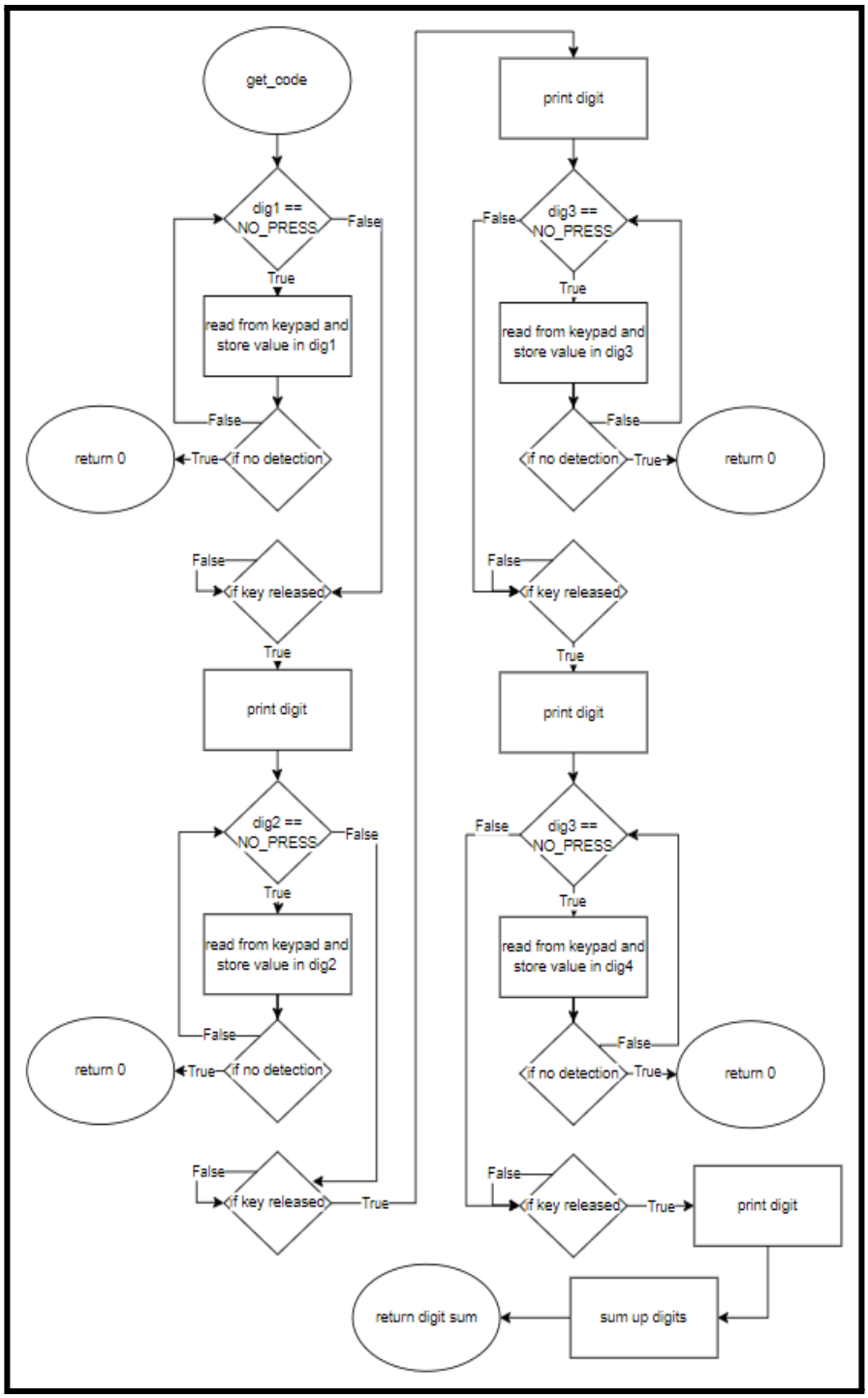
Software Architecture

The program logic for the automatic parking gate begins by initializing the timers, UART, RNG, and keypad for interface, object detection, and gate movement. After initialization, the program generates a random number to display on the terminal and sets a flag that the password is updated. Then the program checks the detection flag. The detection flag is set when the echo signal from the sensor takes the equivalent of 8cm to return. The user is then prompted to enter the password on the terminal via the keypad. The program reads the four-digit code and compares it to the RNG passcode. If the codes do not match, “ACCESS DENIED” is printed to the terminal and the user is prompted to enter the password again. If the detection stops during the code entry “DETECTION STOPPED PREMATURELY” is printed to the terminal and the passcode is updated. Once the codes match, “ACCESS PERMITTED” is printed to the terminal and the PWM sent to the motor is changed to have a 2ms pulse width; raising the motor until it is about 90 degrees. After the gate is at the 90-degree point, a down flag is set to signal that the gate is ready to lower. The gate waits until the detection stops, and then once the detection flag is off, the program waits for around two seconds to let the object pass, then the PWM sent to the motor is changed to have a 1ms pulse width until the motor reaches 0 degrees. Once the gate returns to its initial position, the flag for updating the passcode is set, preparing the system for the next detection cycle.

Flowcharts:


main
initialize timers, UART, RNG, and keypad
if update_passcode && no detection
False
True
read random value, limit to four digits, and print to the terminal
set update_password flag low
if detection && no down_flag
False
True
print "enter passcode:" to the terminal
get code
<code == passcode>
False
True
print "ACCESS PERMITTED" to the terminal
change PWM to 2ms pulse width and wait until motor raises to 90 degrees
set down_flag and update_passcode high
if detection stopped during code entry
False
True
print "DETECTION STOPPED PREMATURELY"
print "ACCESS DENIED"
set update_passcode high
if down_flag && no detection
False
True
change PWM to 1ms pulse width and wait until motor lowers to 0 degrees
set down_flag low





User's Manual

Safety Information:

- Avoid physical obstruction of the gate during operation.
- Do not tamper with sensors or motor housing.
- Keep the device dry and away from extreme temperatures.

Components Overview:

- Sensor - Detects the presence of objects.
- Keypad - Used to input the passcode.
- LEDs - Indicate detection status.
- Motor - Operates the gate.
- MCU - Runs the program and controls peripherals.

Setting Up the Device:

Power On:

- Connect the MCU to a power source via USB

Power Off:

- Disconnect the USB from the MCU to cut the power source

Operation Instructions:

Step 1: Initializing the System:

- Once powered on, the system will automatically generate a random four-digit passcode displayed on the terminal.
- The sensor starts monitoring for object detection.

Step 2: Detection and Passcode Input:

- When an object is detected (the sensor indicates proximity within 8 cm), the system prompts the user to enter the displayed passcode via the keypad.
- LED indicate:
    - On: Sensor detection
    - Off: No sensor detection

Step 3: Gate Movement:

- Correct Passcode:
    - The gate motor lifts the gate to a 90° position and "ACCESS PERMITTED" is displayed on the terminal.
- Incorrect Passcode or No Input:
    - "ACCESS DENIED" is displayed on the terminal.
- Interrupted Detection:
    - If detection stops while entering the code, "DETECTION STOPPED PREMATURELY" is displayed, and the passcode resets.

Step 4: Gate Lowering:

- After the object passes and detection is off, the system waits for 2 seconds before lowering the gate to its original position.

Connecting to the Device:

The device supports RS-232 communication for terminal-based output and debugging. Ensure the following settings:

- Baud Rate: 115,200
- Data Bits: 8
- Stop Bits: 1
- Parity: None
- Flow Control: None

Specifications:

Power Supply Voltage:

- 3.3V DC (USB or external adapter)

Sensor Range:

- Up to 8 cm detection.

Motor Operation:

- 2ms PWM (raise), 1.5ms PWM (neutral), 1ms PWM (lower).

Passcode:

- Four-digit randomly generated code.

LED Indicator:

- Detection status.

Troubleshooting:

Problem	**Possible Cause**	**Solution**
The gate does not open or close	Motor not powered or obstructed	Check the power source and clear obstructions
Passcode not accepted	Incorrect passcode input	Ensure correct code is inputted
Detection not triggered	Sensor malfunction	Check and clean the sensor
LED not lighting up	Device not powered	Ensure the proper USB connection

Appendices

References:

1. *HC-SR04 Ultrasonic Sensor Module User Guide*, *www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf*. *Accessed 9 Dec. 2024.*
2. *Digikey, media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/SER0043_Web.pdf. Accessed 7 Dec. 2024.*
3. *"L298N Motor Driver Module." Components101, components101.com/modules/l293n-motor-driver-module. Accessed 7 Dec. 2024.*
4. *RM0351 Reference Manual*, www.st.com/resource/en/reference_manual/rm0351-stm32l47xxx-stm32l48xxx-stm32l49xxx-and-stm32l4axxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf. Accessed 18 Nov. 2024.

Source Code:

main.c

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2024 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "timer.h"
#include "keypad.h"
#include "UART.h"
RNG_HandleTypeDef hrng;
#define RAISE_TIME 3000000
#define LOWER_TIME 3200000
#define WAIT_TIME 8000000
#define FIRST_SAMPLE 0
#define SECOND_SAMPLE 1
#define MAX_ECHO_TIME 24610 //~8.84cm
#define FLAG 1
#define NO_FLAG 0
#define FOUR_DIG_LIMITER 10000
//function prototypes
```

```
void sensor_init(void);
void motor_init(void);
void SystemClock_Config(void);
static void MX_RNG_Init(void);
//global vars
uint8_t detection_flag = NO_FLAG;
uint8_t update_passcode = FLAG;
uint8_t sample = FIRST_SAMPLE;
uint32_t time_one = 0;
uint32_t time_two = 0;
int main(void)
{
 /* MCU Configuration--------------------------------------------------------*/
 HAL_Init();
 SystemClock_Config();
 //initialize peripherals
 TIM2_init();
 TIM3_init();
 UART_init();
 sensor_init();
 motor_init();
 keypad_init();
 LED_init();
 MX_RNG_Init();
 //local vars
 uint8_t down_flag = NO_FLAG;
 uint32_t passcode;
 while (1)
 {
            if(update_passcode && !detection_flag){
                  //check flags before reading RNG data
                  while(!(RNG->SR & RNG_SR_DRDY));
                  // Read random data and limit it to a four-digit passcode
                  uint32_t raw_random_number = RNG->DR;
                  passcode = raw_random_number % FOUR_DIG_LIMITER;
                  char passcode_str[6];
                  //reset screen and cursor
                  UART_print("\x1B[2J");
                  UART_print("\x1B[H");
                  //print RNG passcode
                  UART_print("Passcode: ");
                  sprintf(passcode_str, "%04u", passcode);
                  UART_print(passcode_str);
                  //set flag low
                  update_passcode = NO_FLAG;
            }
            if (detection_flag && !down_flag){
                  //reset screen and cursor
                  UART_print("\x1B[2J");
                  UART_print("\x1B[H");
                  //print access status
                  UART_print("Enter Passcode: ");
                  //get the four digit code entered by the user
                  uint16_t code = get_code();
                  //check that code is the same as random could provided
                  if(code == passcode){
                        //reset screen and cursor
                        UART_print("\x1B[2J");
                        UART_print("\x1B[H");
                        //print access status
                        UART_print("ACCESS PERMITTED");
```

```
                        //set pulse time to 2ms to raise gate
                        TIM3->ARR = FREQUENCY;
                        TIM3->CCR4 = DUTY_CYCLE_RAISE;
                        //wait for gate to hit 90 degrees
                        for(int i=0; i<RAISE_TIME; i++);
                        //set pulse time to 1.5ms for neutral
                        TIM3->ARR = FREQUENCY;
                        TIM3->CCR4 = DUTY_CYCLE_NEUTRAL;
                        //delay to see access status
                        for(uint32_t i = 0; i < WAIT_TIME; i++);
                        //set flags
                        down_flag = FLAG;
                        update_passcode = FLAG;
                        }
                        else if(code == 0){
                              //reset screen and cursor
                              UART_print("\x1B[2J");
                              UART_print("\x1B[H");
                              //print access status
                              UART_print("DETECTION STOPPED PREMATURELY");
                              //delay to see access status
                              for(uint32_t i = 0; i < WAIT_TIME; i++);
                              //set flag
                              update_passcode = FLAG;
                        }
                         else{
                              //reset screen and cursor
                              UART_print("\x1B[2J");
                              UART_print("\x1B[H");
                              //print access status
                              UART_print("ACCESS DENIED");
                              //delay to see access status
                              for(uint32_t i = 0; i < WAIT_TIME; i++);
                              //set flag
                              update_passcode = FLAG;
                        }
            }
            else if(down_flag && !detection_flag){
                  //wait for ~2 second
                  for(int i=0; i<WAIT_TIME; i++);
                  //set pulse time to 1ms to lower gate
                  TIM3->ARR = FREQUENCY;
                  TIM3->CCR4 = DUTY_CYCLE_LOWER;
                  //wait for gate to lower to 0 degrees
                  for(int i=0; i<LOWER_TIME; i++);
                  //set pulse time to 1.5ms for neutral
                  TIM3->ARR = FREQUENCY;
                  TIM3->CCR4 = DUTY_CYCLE_NEUTRAL;
                  //set flag
                  down_flag = NO_FLAG;
            }
      }
}
void TIM2_IRQHandler(void){
      // check for CC1 flag
      if (TIM2->SR & TIM_SR_CC1IF){
            //turn off trigger
            GPIOB->ODR &= ~(GPIO_ODR_OD0);
            //clear and update CCR1 flag
            TIM2->SR &= ~(TIM_SR_CC1IF);
      }
```

```
	// check for CC2 flag
	else if (TIM2->SR & TIM_SR_CC2IF){
		if(sample == FIRST_SAMPLE){
			//get current time
			time_one = TIM2->CNT;
			//increment sample
			sample++;
			//falling edge polarity
			TIM2->CCER |= TIM_CCER_CC2P;
		}
		else if (sample == SECOND_SAMPLE){
			//get current time
			time_two = TIM2->CNT;
			//increment sample
			sample = FIRST_SAMPLE;
			if(time_two - time_one <= MAX_ECHO_TIME){
				detection_flag = FLAG;
				GPIOA->ODR |= (GPIO_ODR_OD5);
			}
			else{
				detection_flag = NO_FLAG;
				GPIOA->ODR &= ~(GPIO_ODR_OD5);
			}
			//rising edge polarity
			TIM2->CCER &= ~TIM_CCER_CC2P;
		}
		//clear and update CCR1 flag
		TIM2->SR &= ~(TIM_SR_CC2IF);
	}
	// check for update event flag
	else if (TIM2->SR & TIM_SR_UIF){
		//turn on trigger
		GPIOB->ODR |= (GPIO_ODR_OD0);
		// clear update event interrupt flag
		TIM2->SR &= ~(TIM_SR_UIF);
	}
}
void LED_init(void){
	// Enable GPIOA Clock
	RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
	// Set MODER to output
	GPIOA->MODER &= ~(GPIO_MODER_MODE5);
	GPIOA->MODER |= (GPIO_MODER_MODE5_0);
	//set push-pull output type
	GPIOA->OTYPER &= ~(GPIO_OTYPER_OT5);
	//no PUPD
	GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPD5);
	//set to high speed
	GPIOA->OSPEEDR |= (GPIO_OSPEEDR_OSPEED5);
}
void sensor_init(void){
	//configure GPIOB clock
	RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOBEN);
	/*----- Configure PB0 as sensor trigger -----*/
	//setup MODER as output
	GPIOB->MODER &= ~(GPIO_MODER_MODE0);
	GPIOB->MODER |= (GPIO_MODER_MODE0_0);
	//set push-pull
	GPIOB->OTYPER &= ~(GPIO_OTYPER_OT0);
	//no pull-up/pull-down
	GPIOB->PUPDR &= ~(GPIO_PUPDR_PUPD0);
```

```
	//set to high speed
	GPIOB->OSPEEDR |= (GPIO_OSPEEDR_OSPEED0);
	/*----- Configure PB3 as sensor echo -----*/
	//set to alternate function
	GPIOB->MODER &= ~(GPIO_MODER_MODER3);
	GPIOB->MODER |= (GPIO_MODER_MODE3_1);
	GPIOB->AFR[0] |= (1 << GPIO_AFRL_AFSEL3_Pos);
	//set push-pull
	GPIOB->OTYPER &= ~(GPIO_OTYPER_OT3);
	//pull-down
	GPIOB->PUPDR &= ~(GPIO_PUPDR_PUPD3);
	GPIOB->PUPDR |= (GPIO_PUPDR_PUPD3_1);
}
void motor_init(void){
	//configure GPIOB clock
	RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOBEN);
	/*---- Configure PB2 as motor signal-----*/
	//setup as alternate function
	GPIOB->MODER &= ~GPIO_MODER_MODE1;
	GPIOB->MODER |= GPIO_MODER_MODE1_1;
	GPIOB->AFR[0] &= ~GPIO_AFRL_AFSEL1;
	GPIOB->AFR[0] |= (2 << GPIO_AFRL_AFSEL1_Pos);
}
/**
 * @brief: Function to read 4 key-presses
 * @retval: int16_t
 */
uint16_t get_code(void){
	//configure variables
	uint16_t dig_total = 0;
	int16_t dig1 = NO_PRESS;
	int16_t dig2 = NO_PRESS;
	int16_t dig3 = NO_PRESS;
	int16_t dig4 = NO_PRESS;
	while(dig1 == NO_PRESS){ //wait for key-press
	  dig1 = keypad_func(); //read key-press of first digit
	  //leave if detection stops
	  if(!detection_flag){
		return NO_FLAG;
	  }
	}
	while(keypad_func()!= NO_PRESS);//wait for key release
	//print first digit
	char dig1_str[2];
	sprintf(dig1_str, "%u", dig1);
	UART_print(dig1_str);
	while(dig2 == NO_PRESS){ //wait for second key-press
	  dig2 = keypad_func(); //read key-press of second digit
	  //leave if detection stops
	  if(!detection_flag){
		return NO_FLAG;
	  }
	}
	while(keypad_func()!= NO_PRESS);//wait for key release
	//print second digit
	char dig2_str[2];
	sprintf(dig2_str, "%u", dig2);
	UART_print(dig2_str);
	while(dig3 == NO_PRESS){ //wait for third key-press
	  dig3 = keypad_func(); //read key-press of third digit
	  //leave if detection stops
```

```
        if(!detection_flag){
              return NO_FLAG;
        }
      }
      while(keypad_func()!= NO_PRESS); //wait for key release
      //print third digit
      char dig3_str[2];
      sprintf(dig3_str, "%u", dig3);
      UART_print(dig3_str);
      while(dig4 == NO_PRESS){ //wait for fourth key-press
        dig4 = keypad_func(); //read key-press of fourth digit
        //leave if detection stops
        if(!detection_flag){
              return NO_FLAG;
        }
      }
      while(keypad_func()!= NO_PRESS); //wait for key release
      //print fourth digit
      char dig4_str[2];
      sprintf(dig4_str, "%u", dig4);
      UART_print(dig4_str);
      //combine digits to get code entered
      dig_total = (dig1 * 1000) + (dig2 * 100) + (dig3 * 10) + dig4;
      return dig_total;
}
/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 /** Configure the main internal regulator output voltage
 */
 if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
 {
   Error_Handler();
 }
 /** Initializes the RCC Oscillators according to the specified parameters
 * in the RCC_OscInitTypeDef structure.
 */
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
 RCC_OscInitStruct.MSIState = RCC_MSI_ON;
 RCC_OscInitStruct.MSICalibrationValue = 0;
 RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_11;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
 {
   Error_Handler();
 }
 /** Initializes the CPU, AHB and APB buses clocks
 */
 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                               |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
 {
```

```
    Error_Handler();
  }
}
/**
  * @brief RNG Initialization Function
  * @param None
  * @retval None
  */
static void MX_RNG_Init(void)
{
  /* USER CODE BEGIN RNG_Init 0 */
  /* USER CODE END RNG_Init 0 */
  /* USER CODE BEGIN RNG_Init 1 */
  /* USER CODE END RNG_Init 1 */
  hrng.Instance = RNG;
  if (HAL_RNG_Init(&hrng) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN RNG_Init 2 */
  /* USER CODE END RNG_Init 2 */
}
/* USER CODE BEGIN 4 */
/* USER CODE END 4 */
/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

keypad.h

```
/*
* keypad.h
*
*  Created on: Oct 23, 2024
*      Author: firaz
*/
#ifndef INC_KEYPAD_H_
#define INC_KEYPAD_H_
#define SRC_KEYPAD_H_
#define NUM_OF_ROWS 4  // 4-row keypad
#define NUM_OF_COLS 3  // 3-column keypad
#define NO_PRESS (int8_t) -1//signfies no button was pressed
#define ASTERISK (int8_t) 10 //signifies asterisk keypress
#define POUND (int8_t) 11 //signifies pound keypress
void keypad_init(void);
int8_t keypad_func(void);
int8_t calculate_key(int8_t row,int8_t col);
uint16_t get_code(void);
#endif /* INC_KEYPAD_H_ */
```

keypad.c

```
/*
* keypad.c
*
*  Created on: Oct 23, 2024
*      Author: firaz
*/
#include "main.h"
#include "keypad.h"
#define SRC_KEYPAD_H_
#define NUM_OF_ROWS 4  // 4-row keypad
#define NUM_OF_COLS 3  // 3-column keypad
#define NO_PRESS (int8_t) -1//signfies no button was pressed
#define ASTERISK (int8_t) 10 //signifies asterisk keypress
#define POUND (int8_t) 11 //signifies pound keypress
/**
 * @brief: function to initialize keypad ports and set columns to 1
 * @retval: None
 */
void keypad_init(void){
        //set clock for GPIOA, GPIOB, and GPIOC
        RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOBEN | RCC_AHB2ENR_GPIOCEN);
        /*------------------- Configure PB4-PB6 for GPIOB column input -------------------*/
        //setup MODER for columns input
        GPIOB->MODER &= ~(GPIO_MODER_MODE4 | GPIO_MODER_MODE5 | GPIO_MODER_MODE6);
        //setup pull down resistor to avoid floating
        GPIOB->PUPDR &= ~(GPIO_PUPDR_PUPD4 |GPIO_PUPDR_PUPD5 | GPIO_PUPDR_PUPD6);
        GPIOB->PUPDR |= (GPIO_PUPDR_PUPD4_1 |GPIO_PUPDR_PUPD5_1 | GPIO_PUPDR_PUPD6_1);
        /*------------------- Configure PC0-PC3 for GPIOC row output -------------------*/
        //setup MODER for row output
        GPIOC->MODER &= ~(GPIO_MODER_MODE0 | GPIO_MODER_MODE1 | GPIO_MODER_MODE2 |
GPIO_MODER_MODE3);
        GPIOC->MODER |= (GPIO_MODER_MODE0_0 | GPIO_MODER_MODE1_0 | GPIO_MODER_MODE2_0 |
GPIO_MODER_MODE3_0);
        //set push pull output type
        GPIOC->OTYPER  &=  ~(GPIO_OTYPER_OT0 | GPIO_OTYPER_OT1 | GPIO_OTYPER_OT2 |
GPIO_OTYPER_OT3);
```

```
        //no PUPD
        GPIOC->PUPDR |= (GPIO_PUPDR_PUPD0_1 |GPIO_PUPDR_PUPD1_1 | GPIO_PUPDR_PUPD2_1 |
GPIO_PUPDR_PUPD3_1);
        //set to high speed
        GPIOC->OSPEEDR |= (GPIO_OSPEEDR_OSPEED0_Msk | GPIO_OSPEEDR_OSPEED1_Msk
                    | GPIO_OSPEEDR_OSPEED2_Msk | GPIO_OSPEEDR_OSPEED3_Msk);
        /*------------------- Initialize rows-------------------*/
        //set rows to 1
        GPIOC->ODR |= (GPIO_ODR_OD0 | GPIO_ODR_OD1 | GPIO_ODR_OD2 | GPIO_ODR_OD3);
}
/**
 * @brief Helper function to calculate the key for keypad_func
 * @retval int8_t
 */
int8_t calculate_key(int8_t row,int8_t col){
      //2D array to represent keypad
      int8_t keypad[NUM_OF_ROWS][NUM_OF_COLS] = {
          {1, 2, 3},
          {4, 5, 6},
          {7, 8, 9},
          {ASTERISK, 0, POUND}
      };
            return keypad[row][col];  // Return the character for the pressed key
}
/**
 * @brief: function to check for key-press and return key value if pressed
 * @retval: int8_t
 */
int8_t keypad_func(void){
      int8_t pressed_row = NO_PRESS;
   int8_t pressed_col = NO_PRESS;
   int16_t idr_value = GPIOB->IDR;
   //read cols
   idr_value &= (GPIO_IDR_ID4  | GPIO_IDR_ID5 | GPIO_IDR_ID6);
   if(!((idr_value == 0b10000) || (idr_value == 0b100000) || (idr_value == 0b1000000))){
      //set rows back to zero for next press
      GPIOC->ODR |= (GPIO_ODR_OD0 | GPIO_ODR_OD1 | GPIO_ODR_OD2 | GPIO_ODR_OD3);
      return NO_PRESS;
   }
   //cycle through rows to determine key-press
   for(int row = 0; row < NUM_OF_ROWS; row++){
       GPIOC->ODR &= ~(GPIO_ODR_OD0 | GPIO_ODR_OD1 | GPIO_ODR_OD2 | GPIO_ODR_OD3);
       GPIOC->ODR |= (1<<row);
       idr_value = GPIOB->IDR;
       //checks to see if we get correct idr, stores row, and associated col based on col idr
       if(idr_value & GPIO_IDR_ID4 ){ //check column 0
           pressed_col = 0;
           pressed_row = row;
           break;
       }
       if(idr_value & GPIO_IDR_ID5 ){ //check column 1
           pressed_col = 1;
           pressed_row = row;
           break;
       }
       if(idr_value & GPIO_IDR_ID6){ //check column 3
           pressed_col = 2;
           pressed_row = row;
           break;
       }
   }
```

```
   if(pressed_row == NO_PRESS || pressed_col == NO_PRESS){
      //set rows back to zero for next press
      GPIOC->ODR |= (GPIO_ODR_OD0 | GPIO_ODR_OD1 | GPIO_ODR_OD2 | GPIO_ODR_OD3);
      return NO_PRESS;
   }
   else{
      //set rows back to zero for next press
      GPIOC->ODR |= (GPIO_ODR_OD0 | GPIO_ODR_OD1 | GPIO_ODR_OD2 | GPIO_ODR_OD3);
      return calculate_key(pressed_row, pressed_col);
   }
}
```

timer.h

```
/*
* timer.h
*
*  Created on: Nov 7, 2024
*      Author: firaz
*/
#ifndef INC_TIMER_H_
#define INC_TIMER_H_
#define MEASUREMENT_TIME 2879999 //48MHz x 60ms
#define TRIGGER_TIME 479 //48MHz x 10us
#define PRESCALER 3999
#define DUTY_CYCLE_LOWER 12 //6.5%
#define DUTY_CYCLE_NEUTRAL 18 //7.5%
#define DUTY_CYCLE_RAISE 24 //8.5%
#define FREQUENCY 240 //48MHz/3999 x 1/50Hz
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void TIM2_init(void);
void TIM3_init(void);
#endif /* INC_TIMER_H_ */
```

timer.c

```
/*
* timer.c
*
*  Created on: Nov 7, 2024
*      Author: firaz
*/
#include "main.h"
#include "timer.h"
void TIM2_init(void){
      //configure TIM2 clock
      RCC->APB1ENR1 |= (RCC_APB1ENR1_TIM2EN);
      //set TIM2 to count up
      TIM2->CR1 &= ~(TIM_CR1_DIR);
      //set measuring time
      TIM2->ARR = MEASUREMENT_TIME;
      //set time of trigger event
      TIM2->CCR1 = TRIGGER_TIME;
      //configure channel 2 as input
      TIM2->CCMR1 |= TIM_CCMR1_CC2S_0;
      //rising edge polarity
```

```
	TIM2->CCER &= ~TIM_CCER_CC2P;
	//Enable capture on Channel 2
	TIM2->CCER |= TIM_CCER_CC2E;
	//enable update event interrupt in TIM2
	TIM2->DIER |= (TIM_DIER_UIE | TIM_DIER_CC1IE | TIM_DIER_CC2IE);
	//clear the flag before starting
	TIM2->SR &= ~(TIM_SR_UIF | TIM_SR_CC1IF | TIM_SR_CC2IF);
	//start timer
	TIM2->CR1 |= TIM_CR1_CEN;
	//enable TIM2 in NVIC
	NVIC_EnableIRQ(TIM2_IRQn);
	//enable interrupts globally
	__enable_irq();
}
void TIM3_init(void){
	//configure TIM3 clock
	RCC->APB1ENR1 |= (RCC_APB1ENR1_TIM3EN);
	// Prescaler of 0 (40MHz clock)
	TIM3->PSC = PRESCALER;
	//set PWM Mode 1 (OC4M = 110)
	TIM3->CCMR2 &= ~TIM_CCMR2_OC4M_Msk;
	TIM3->CCMR2 |= (6 << TIM_CCMR2_OC4M_Pos);
	//enable preload for CCR4
	TIM3->CCMR2 |= TIM_CCMR2_OC4PE;
	//enable timere 3 channel 4
	TIM3->CCER |= TIM_CCER_CC4E;
	//set to 50Hz
	TIM3->ARR = FREQUENCY;
	//set 7.5% duty cycle
	TIM3->CCR4 = DUTY_CYCLE_NEUTRAL;
	//enable ARR preload
	TIM3->CR1 |= TIM_CR1_ARPE;
	//generate an update event to load registers
	TIM3->EGR |= TIM_EGR_UG;
	//start the timer
	TIM3->CR1 |= TIM_CR1_CEN;
}
```

UART.h

```
/*
* UART.h
*
*  Created on: Nov 6, 2024
*      Author: firaz
*/
#ifndef INC_UART_H_
#define INC_UART_H_
#define CLOCK_SPEED 48000000
#define BAUD_RATE 115200
void UART_init(void);
void UART_print(char *out_str);
#endif /* INC_UART_H_ */
```

UART.c

```
/*
* UART.c
*
```

```
*  Created on: Nov 6, 2024
*      Author: firaz
*/
#include "main.h"
#include "UART.h"
void UART_print(char *out_str){
	//check string
	if(out_str == NULL){
		return;
	}
	//check character isn't null terminator
	while(*out_str != '\0'){
		//wait for transmission flag
		while(!(USART2->ISR & USART_ISR_TXE));
		USART2->TDR = *out_str; //write string to USART
		out_str++; //increment string pointer
	}
}
void UART_init(void){
	//Enable clock for GPIOA and USART
	RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
	RCC->APB1ENR1 |= RCC_APB1ENR1_USART2EN;
	//Set GPIOA2 & GPIOA3 to alternate function
	GPIOA->MODER &= ~(GPIO_MODER_MODE2 | GPIO_MODER_MODE3);
	GPIOA->MODER |= (GPIO_MODER_MODE2_1 | GPIO_MODER_MODE3_1);
	//Enable alternate functionality
	GPIOA-> AFR[0] &= ~(GPIO_AFRL_AFSEL2 | GPIO_AFRL_AFSEL3);
	GPIOA-> AFR[0] |= ((0x7UL << GPIO_AFRL_AFSEL2_Pos) | (0x7UL << GPIO_AFRL_AFSEL3_Pos));
	//Set no PUPD
	GPIOA->OTYPER &= ~(GPIO_OTYPER_OT2 | GPIO_OTYPER_OT3);
	GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPD2 | GPIO_PUPDR_PUPD3);
	//Set GPIOA speed to high
	GPIOA->OSPEEDR |= (GPIO_OSPEEDR_OSPEED2 | GPIO_OSPEEDR_OSPEED3);
	//USART setup
	//Disable UE to set M0, M1, BRR, STOP,
	USART2->CR1 &= ~(USART_CR1_UE);
	//Set M0, M1 for 8 bit word
	USART2->CR1 &= ~(USART_CR1_M1 | USART_CR1_M0);
	//Set stop bit to 1 bit
	USART2->CR2 &= ~(USART_CR2_STOP);
	//BRR set baud rate 48MHz/115.2Kbps
	USART2->BRR = (CLOCK_SPEED/BAUD_RATE);
	//RE - receiver enable
	USART2->CR1 |= (USART_CR1_RE);
	//Set oversampling mode
	USART2->CR1 &= ~(USART_CR1_OVER8);
	//UE - USART enable
	USART2->CR1 |= (USART_CR1_UE);
	//TE - transmit enable  RE - Receive enable
	USART2->CR1 |= (USART_CR1_TE | USART_CR1_RE);
}
```