Microcontroller & Interfacing

CE205T & EE243T

	CLO-2		CLO-3					Total
Part	B	C	A	D	E	F	G	
Marks	50	50	50	50	50	50	50	
Obt.								

Water Level Monitoring & Control

Group # [CE-03]

1: Awwab Samad

2: Muhammad Abubakar Saif

3: Furqan Omer

A. Overview [CLO-3, 50 Marks]

The Water Level Monitoring & Control project aims to develop an embedded system using the STM32F401 microcontroller to monitor the water level in a container and provide control functionality. The system employs various sensors to measure the water level, an LCD display to show the readings, and an alarm to alert users when the water level exceeds a set threshold. In the advanced implementation, a solenoid valve is used to regulate water inflow, and temperature control is achieved through a heating element. The project involves integrating hardware components, developing control algorithms, and implementing the necessary interfacing and communication protocols. (Nikita B Jape, 2022)

GOALS

- Measure and display the water level in millimeters using suitable sensors and an LCD module.
- Set a configurable threshold for the water level alarm and activate the alarm when the threshold is exceeded.
- Control the inflow of water into the container using a solenoid valve based on the water level readings.
- Implement temperature control by switching a heating element on or off based on the sensed temperature.

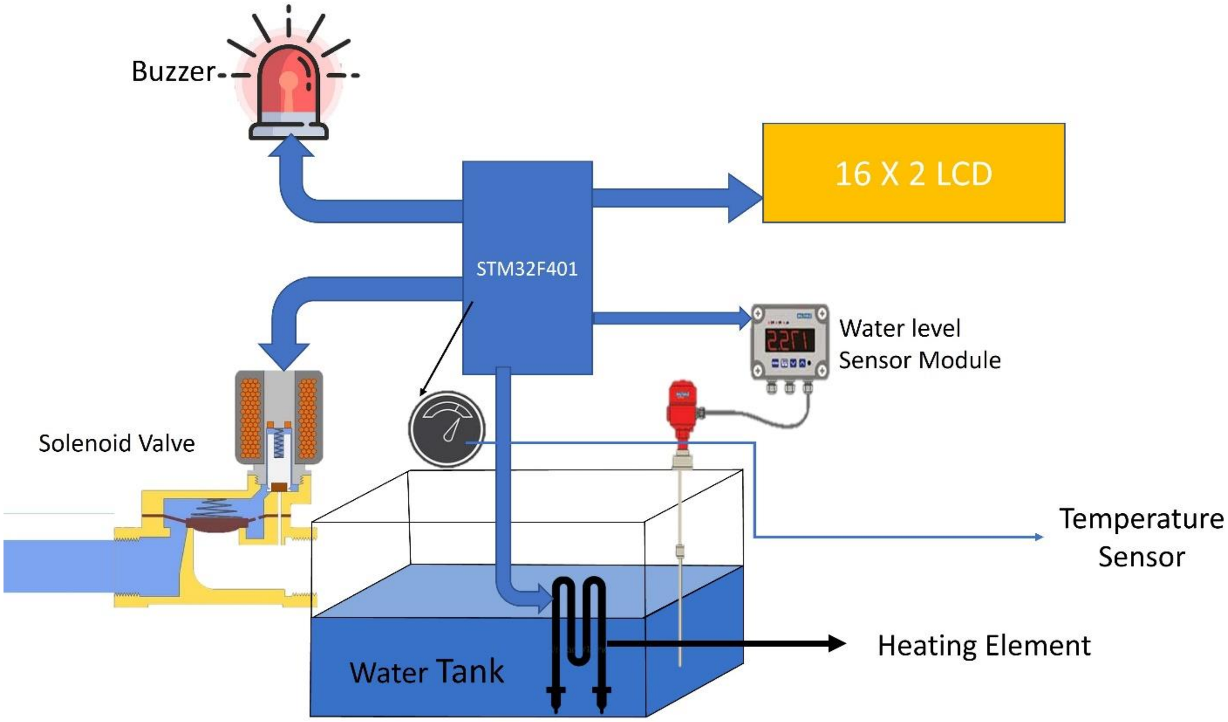
B. List of Components Used [CLO-2, 50 Marks]

Sr#	Component	Cost (Rs.) (Unit Price)	Link to Dataset	Operating Principle
1	STM32F401CDU6	990	ST Datasheet	Microcontroller
2	Water Level Sensor	75	Datasheet	Pressure/Resistive
3	LCD Display 16 x 2	470	Datasheet	Liquid Crystal Display
4	Potentiometer	29	Datasheet	Variable Resistor
5	Buzzer	150	Datasheet	Sound Generation
6	Solenoid Valve	Not bought yet		Electrically controlled valve
7	5V Relay	45	Datasheet	Electrical isolation/control
8	Heating Element	400	-	Resistive Heating Element
9	Tactile Push Buttons	10	Datasheet	
10	LM35	150	Datasheet	IC temperature device
11	ST-Link V2	1300	Datasheet	Programming

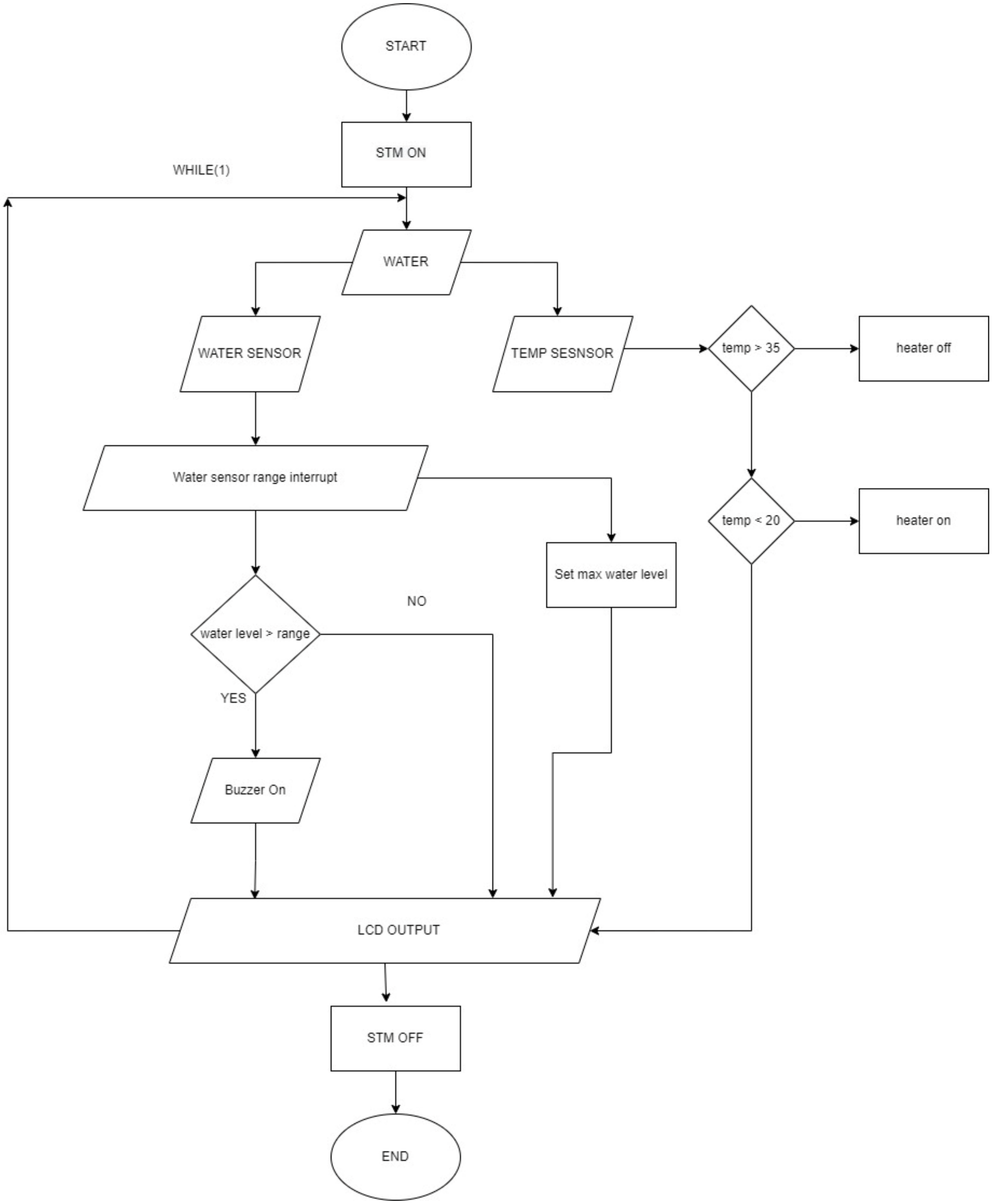
C. Peripherals of STM Microcontroller being used [CLO-2, 50 Marks]

1. GPIO Pins: Used for interfacing with various components such as the water level sensor, LCD display, potentiometer, buzzer, solenoid valve, and heating element.
2. ADC (Analog-to-Digital Converter): Utilized to read the analog voltage value from the potentiometer for configuring the water level threshold.
3. Timers: Employed for generating precise timing intervals and PWM signals required for controlling the solenoid valve and heating element.
4. USART/UART: Possible communication interface for debugging and sending data to external devices if needed.

D. Block Diagram/Schematic [CLO-3, 50 Marks]



E. Flow Chart (Required at the time of final submission) [CLO-3, 50 Marks]



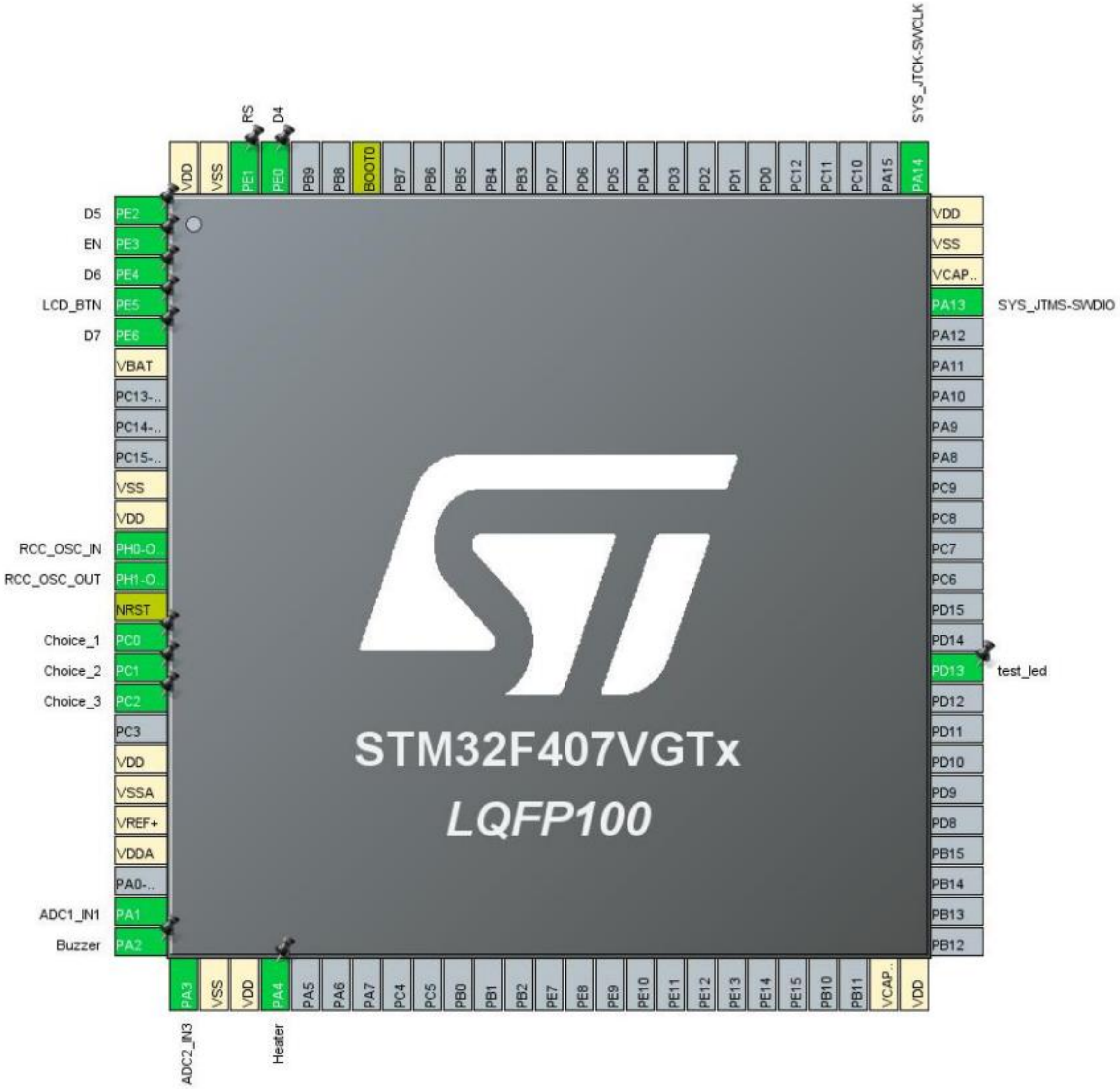
F. CEP (Project Complexity) Attributes - Describe Briefly [CLO-3, 50 Marks]

Attribute	Description	Complexity Level in your project
WP1: Depth of knowledge	The project shall involve in-depth engineering knowledge related to the area of Microprocessors, Microcontrollers & Interfacing [WK-4, Engineering Specialization].	Interfacing Techniques: Methods to interface microcontrollers with sensors, actuators, and peripherals using protocols like I2C, SPI, and UART. Sensor Interfacing: Analog and digital sensor interfaces, signal conditioning, and techniques for accurate sensor readings. Peripheral Utilization: Configuring and controlling GPIOs, timers, UART, and ADC to interface with various components
WP2: Range of conflicting requirements	The project has multiple conflicting requirements in terms of optimal usage of peripheral resources available on a Microcontroller.	• Limited number of interrupt lines available on the microcontroller for handling multiple events simultaneously. • Limited number of I/O lines, requiring careful allocation for interfacing with various components. • Conflicting timing requirements between different peripherals, such as LCD updates, sensor readings, and control operations.
WP5 Extent of applicable codes	The projects expose the students to broadly defined problems which require the development of codes that may be partially outside those encompassed by well-documented standards.	• Implementing interrupt-driven routines for timely sensor readings and alarm activations. • Designing control algorithms to regulate the solenoid valve and heating element based on sensor inputs. • Synchronizing multiple tasks, such as sensor readings, LCD updates, and control operations, within a real-time framework.
WP7 Interdependence	The projects shall have multiple components at the hardware and software level.	• Proper integration of sensors, LCD, potentiometer, buzzer, solenoid valve, and heating element with the microcontroller, ensuring correct wiring, signal conditioning, and compatibility. • Coordinating communication and control between multiple components while optimizing the utilization of microcontroller peripherals.

		- Balancing the timing requirements of different tasks to avoid conflicts and ensure efficient operation of the system.

G. Code [CLO-3, 50 Marks]

Pin Configuration:



Code:

```
// IDE: STM32CUBEIDE (Authors, 2023)
/* Includes -----------------------------------------------------------------
-*/
#include "main.h"

/* Private includes ---------------------------------------------------------
-*/
/* USER CODE BEGIN Includes */
#include "lcd.h"
#include "stdio.h"
/* USER CODE END Includes */

/* Private typedef ----------------------------------------------------------
-*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----------------------------------------------------------
-*/
/* USER CODE BEGIN PD */
int display_state = 1;
int temp;
int input = 30;
/* USER CODE END PD */

/* Private macro ------------------------------------------------------------
-*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables --------------------------------------------------------
-*/
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;

TIM_HandleTypeDef htim1;

/* USER CODE BEGIN PV */
long map(long x, long in_min, long in_max, long out_min, long out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
/* USER CODE END PV */

/* Private function prototypes ----------------------------------------------
-*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
static void MX_ADC1_Init(void);
static void MX_ADC2_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */
/* Private user code --------------------------------------------------------
-*/
/* USER CODE BEGIN 0 */
int ADCConfig(void){
      HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 100);
    uint16_t adcVal = HAL_ADC_GetValue(&hadc1);
    int maping = map(adcVal, 0, 4095, 0, 40);
    HAL_ADC_Stop(&hadc1);
    HAL_Delay(500);
    return maping;

}

//int ADC2Config(void){
//    HAL_ADC_Start(&hadc2);
//    HAL_ADC_PollForConversion(&hadc2, 100);
//    uint16_t adcVal = HAL_ADC_GetValue(&hadc2);
//
//    temp = (adcVal*500.0)/4095.0;
//    HAL_ADC_Stop(&hadc2);
//    HAL_Delay(500);
//    return adcVal;
//
//}

void lcd_initial(){ //reference: (Craftem, 2020)
        lcd_clear();
        lcd_cursorShow(0);
        //lcd_setCursor(0, 3);
        lcd_printf("Water Level");
        lcd_2ndLine();
        //lcd_setCursor(1, 0);
        lcd_printf("Sensor Module");
        HAL_Delay(2000);
        lcd_clear();
        lcd_1stLine();
        //lcd_setCursor(0, 0);
        lcd_printf("Project by:Awwab");
        lcd_2ndLine();
        lcd_printf("Abubakar, Furqan");
        lcd_clear();

        HAL_Delay(2000);
}


/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
```

```
      char txt[30];
      char txt1[30];
      char txt2[30];
      int adcVal = 0;
  /* USER CODE END 1 */

  /* MCU Configuration-------------------------------------------------------
-*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
  int value;
  int voltageOut;
  int quotient, remainder, tempA, tempB;
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM1_Init();
  MX_ADC1_Init(); // (Yakub, 2018)
  MX_ADC2_Init();
  /* USER CODE BEGIN 2 */
  lcd_init_4bits(RS_GPIO_Port, RS_Pin, EN_Pin, D4_GPIO_Port, D4_Pin, D5_Pin,
D6_Pin, D7_Pin);
  HAL_Delay(100);

  HAL_Delay(2000);
  //lcd_clear();
  lcd_initial();
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  //lcd_printf("Level:");
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      value = ADCConfig();

        //percent = ((float)adcVal/4000)*100;
        sprintf(txt,"%d", (int) value);
        if (display_state == 1){
              lcd_clear();
              lcd_printf("Level:");
              lcd_setCursor(0,7);
              lcd_printf(txt);
              lcd_setCursor(0, 10);
              lcd_printf("mm");
        } else if (display_state == 0){
                    HAL_ADC_Start(&hadc2);
                  if (HAL_ADC_PollForConversion(&hadc2, 100) == HAL_OK){
                        adcVal = HAL_ADC_GetValue(&hadc2);

                        voltageOut = (adcVal*500)/4096.0;
                        temp = (voltageOut/5);

                        //quotient = temp/10;
                        //remainder = temp % 10;
                        //tempA = quotient + 0x30;
                        sprintf(txt1,"%d", temp);
                        lcd_clear();
                        lcd_printf("Temp:");
                        lcd_setCursor(0,7);
                        lcd_printf(txt1);
                        lcd_setCursor(0, 10);
                        lcd_printf("C");
                  //    HAL_ADC_Stop(&hadc2);
                        //HAL_Delay(500);
                  }

        }
        if (display_state == 2){
            lcd_clear();
            lcd_printf("1");
            lcd_setCursor(0,5);
            lcd_printf("2");
            lcd_setCursor(0,10);
            lcd_printf("3");
            lcd_2ndLine();
            lcd_printf("40");
            lcd_setCursor(1, 5);
            lcd_printf("30");
            lcd_setCursor(1, 10);
            lcd_printf("20");
        }
        if (display_state == 3){
              lcd_clear();
              lcd_printf("Max Level: ");
              lcd_setCursor(0,12);
              sprintf(txt2,"%d", input);
              lcd_printf(txt2);
              display_state = 1;
              HAL_Delay(1000);
        }
        if (value >= input) {
              HAL_GPIO_WritePin(Buzzer_GPIO_Port, Buzzer_Pin, 1);
              HAL_GPIO_WritePin(test_led_GPIO_Port, test_led_Pin, 1);
        } else {
              HAL_GPIO_WritePin(Buzzer_GPIO_Port, Buzzer_Pin, 0);
              HAL_GPIO_WritePin(test_led_GPIO_Port, test_led_Pin, 0);
        }
        if (temp < 25) {
```

```
              HAL_GPIO_WritePin(Heater_GPIO_Port, Heater_Pin, 1);
        }else if (temp > 35) {
              HAL_GPIO_WritePin(Heater_GPIO_Port, Heater_Pin, 0);
        }
        //lcd_printf(txt);
  }
  /* USER CODE END 3 */
}

/**
  * @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
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 72;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  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_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Configure the global features of the ADC (Clock, Resolution, Data
Alignment and number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its corresponding rank
in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) // (Clinic, 2021)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief ADC2 Initialization Function
  * @param None
  * @retval None
  */
```

```
static void MX_ADC2_Init(void)
{

  /* USER CODE BEGIN ADC2_Init 0 */

  /* USER CODE END ADC2_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC2_Init 1 */

  /* USER CODE END ADC2_Init 1 */

  /** Configure the global features of the ADC (Clock, Resolution, Data
Alignment and number of conversion)
  */
  hadc2.Instance = ADC2;
  hadc2.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
  hadc2.Init.Resolution = ADC_RESOLUTION_12B;
  hadc2.Init.ScanConvMode = DISABLE;
  hadc2.Init.ContinuousConvMode = DISABLE;
  hadc2.Init.DiscontinuousConvMode = DISABLE;
  hadc2.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc2.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc2.Init.NbrOfConversion = 1;
  hadc2.Init.DMAContinuousRequests = DISABLE;
  hadc2.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc2) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its corresponding rank
in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_3;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC2_Init 2 */

  /* USER CODE END ADC2_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{
  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 72-1;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 0xffff-1;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
```

```
/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOE, D5_Pin|EN_Pin|D6_Pin|D7_Pin
                        |D4_Pin|RS_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOA, Buzzer_Pin|Heater_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(test_led_GPIO_Port, test_led_Pin, GPIO_PIN_RESET);

/*Configure GPIO pins : D5_Pin EN_Pin D6_Pin D7_Pin
                         D4_Pin RS_Pin */
GPIO_InitStruct.Pin = D5_Pin|EN_Pin|D6_Pin|D7_Pin
                        |D4_Pin|RS_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

/*Configure GPIO pin : LCD_BTN_Pin */
GPIO_InitStruct.Pin = LCD_BTN_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
HAL_GPIO_Init(LCD_BTN_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : Choice_1_Pin Choice_2_Pin */
GPIO_InitStruct.Pin = Choice_1_Pin|Choice_2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_PULLUP;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pin : Choice_3_Pin */
GPIO_InitStruct.Pin = Choice_3_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(Choice_3_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : Buzzer_Pin Heater_Pin */
GPIO_InitStruct.Pin = Buzzer_Pin|Heater_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pin : test_led_Pin */
GPIO_InitStruct.Pin = test_led_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(test_led_GPIO_Port, &GPIO_InitStruct);

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI0_IRQn);

HAL_NVIC_SetPriority(EXTI1_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI1_IRQn);
```

```
  HAL_NVIC_SetPriority(EXTI2_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI2_IRQn);

  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
      if(GPIO_Pin == LCD_BTN_Pin ){
            //HAL_GPIO_TogglePin(test_led_GPIO_Port, test_led_Pin);
            //HAL_GPIO_WritePin(test_led_GPIO_Port, test_led_Pin, 1);
            if (display_state == 1){
            //    lcd_clear();
                  HAL_GPIO_WritePin(test_led_GPIO_Port, test_led_Pin, 1);
                  //lcd_printf("Temp:");
                  display_state = 0;
            }
            else{
                  //HAL_GPIO_TogglePin(test_led_GPIO_Port, test_led_Pin);
                  //lcd_clear();
                  //HAL_GPIO_WritePin(test_led_GPIO_Port, test_led_Pin, 0);
                  //lcd_printf("Level:");
                  display_state = 1;
            }
      }
      if (GPIO_Pin == Choice_1_Pin){
            //HAL_GPIO_TogglePin(test_led_GPIO_Port, test_led_Pin);
            //lcd_clear();
            display_state = 2;
            //HAL_Delay(1000);
      }
      if (GPIO_Pin == Choice_2_Pin) {
            input = 40;
            //HAL_GPIO_TogglePin(test_led_GPIO_Port, test_led_Pin);
            //HAL_Delay(1000);
            display_state = 3;
      }
      if (GPIO_Pin == Choice_3_Pin) {
            input = 30;
            //HAL_GPIO_TogglePin(test_led_GPIO_Port, test_led_Pin);
            display_state = 3;
            //display_state = 1;
      }
      if (GPIO_Pin == Choice_4_Pin){
            input = 20;
            display_state = 3;
      }
}
/* 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 */
```

H.References

Authors, S. W. (2023, March 1). *Introduction to STM32CubeIDE*. Retrieved from STM32 Arm® Cortex® MCU Wiki: https://wiki.stmicroelectronics.cn/stm32mcu/wiki/STM32CubeIDE:Introduction_to_STM32CubeIDE

Clinic, E. (2021, Dec 07). *12V DC Heater Plate 80W, PTC Heater Plate with Arduino temperature monitoring, Solar water heater*. From Youtube: https://www.youtube.com/watch?v=eWrkHLmWQJU

Craftem. (2020, December 2). *Interface LCD Display 16×2 with Stm32*. Retrieved from Youtube: https://www.youtube.com/watch?v=nSopd2cQ0ok

Nikita B Jape, A. B. (2022, February 2). *IOT Based Water Level Monitoring & Controlling System*. From International Journal of Creative Research Thoughts (IJCRT): https://ijcrt.org/papers/IJCRT2202447.pdf

Yakub, M. (2018, Feb 04). *STM32 Nucleo - Keil 5 IDE with CubeMX: Tutorial 3 - ADC Single mode multi-channel*. From Mutex Embedded - Youtube: https://www.youtube.com/watch?v=6U1uyEjoPu0

The code used for lcd was written based on the open-source Arduino LiquidCrystal library
and by referring to the DATASHEET of the lcd, also with the help of
the following YouTube tutorials on LCD 16X2:

(1): 'RC Tractor Guy' YouTube tutorial on the following link:
https://www.youtube.com/watch?v=efi2nlsvbCI

(2): 'Explore Embedded' YouTube tutorial on the following link:
https://www.youtube.com/watch?v=YDJISiPUdA8