Aquasense

By: Jordan Reichhardt / Kevin Swanson / John Mort

EE 329 Section 01

Fall Quarter

Experiment Final Project EE 329: Custom Project

Post Lab Report

November 24, 2024

Instructor: John Penvenne

Behavior Description

Aquasense is comprised of three main microcontroller units: a base station ESP32, a transmitter station ESP32, and an STM32 Nucleo L4 for user input. The ESP32s communicate with each other through wireless LoRa communication, rated for a distance of up to 2 miles (tested successfully up to 0.5 miles). The system begins with the user interacting via a 4x3 keypad connected to the STM32. Initially, the user presses 1 to enter Tank Mode (with future development planned to use 2 for Tide Mode). The system then prompts the user to enter the tank's height and maximum water level. This configuration data is transmitted from the STM32 to the base station ESP32 over I2C.

Once the configuration is complete, the transmitter station ESP32 begins reading water levels from the ultrasonic sensor. This data is sent via LoRa to the base station ESP32. The base station processes the data and displays the results in a user-friendly format on an attached OLED screen. The displayed information includes the water level in meters and centimeters ("x.x m, xx.x cm") and the tank's fill percentage. The system continuously repeats this process, updating the display in real-time, until power is lost or the system is manually reset.

System Specifications

Transmitter Unit (ESP32-S3)	
Supply Voltage	5V (typical), 4.7-6 V (Min-Max)
Current Draw	48.5 mA (typical), 61.2 mA (LoRa TX)

Base Station	
Supply Voltage	5V (typical), 4.7-6 V (Min-Max)
Current Draw	82mA (Min) - 94 mA (Max)

LoRa SX1262	
Max TX Power	21 dBm
Max Receiving Sensitivity	-134 dBm 470~510 MHz, 863~928 MHz (915 MHz in U.S.)
Spreading Factor	SF7 (higher data throughput, reduced range)
TX Power	14 dBm (moderate power)
Bandwidth	125 kHz (lower data throughput, power consumption, robust)
Preamble	8 symbols (short-moderate length, for lower interference environment)

Ultrasonic Rangefinder	
Sonic Frequency	40 kHz
Measuring Angle	15°
Max Range	4 m
Min Range	2 cm
Accuracy	±3mm (ideal, perpendicular from target)*
Trigger Signal	10 us TTL pulse

I2C Operation	
Clock Speed:	100 kHz Standard Mode
Logic Level	3.3V (oversized R = 2 kOhm pullups utilized)
Peripheral ESP32 ADDR	0x51 (7-bit address)

* Oblique surfaces greatly impact accuracy, and the sensor's accuracy is also affected by extreme temperature and pressure.

Code Planning & Software Architecture

STM32 Pseudocode:

Main.c Pseudocode:

```
Setup:
  - Initialize HAL
  - Configure System Clock
  - Initialize GPIOs for I2C (PB8: SCL, PB9: SDA)
  - Initialize Keypad

Main loop:
  - While true:
    - Call `Grab_KeyPress`
    - Store result in `number_to_send`
    - Send `number_to_send` over I2C
```

I2C.c Pseudocode:

```
START

GPIO_I2C1_Init:
  - Enable GPIOB clock
  - Set PB8 and PB9 as alternate function
  - Configure PB8, PB9 as open-drain
  - Set medium speed for PB8, PB9
  - Disable pull-up/pull-down for PB8, PB9

I2C1_Init:
  - Enable I2C1 clock
  - Disable I2C1 peripheral (clear)
  - Enable analog filter
  - Disable digital filter
  - Set I2C1 timing to 100 kHz
  - Configure 7-bit addressing
  - Enable I2C1 peripheral

I2C_SendNumber:
  - While bus is busy and timeout > 0:
    - Delay 1 ms
    - Decrease timeout
    - If timeout == 0:
```

        - Break

    - Configure transaction:
      - Set peripheral address
      - Set byte count
      - Enable auto end
      - Set start condition

    - While TXIS flag is unset and timeout > 0:
      - Delay 1 ms
      - Decrease timeout
      - If timeout == 0:
        - Exit function

    - Send `number` via I2C1->TXDR

    - While TC flag is unset and timeout > 0:
      - Delay 1 ms
      - Decrease timeout
      - If timeout == 0:
        - Break

    - Clear STOP condition flag

Keypad.c Pseudocode:

  Keypad_Config:
    - Enable GPIO clock for keypad
    - Clear mode registers for pins
    - Set column pins as output
    - Configure pins to push-pull
    - Enable pull-down resistors
    - Set columns to high speed

  Keypad_IsAnyKeyPressed:
    - Set all columns high
    - Wait for signal settling
    - If any row is high:
      - Return true
    - Else:
      - Return false

Keypad_WhichKeyIsPressed:
- Set all columns high
- For each row:
  - If row is high:
    - Set all columns low
    - For each column:
      - Set current column high
      - If row and column are high:
        - Encode and return key
- Return NO_KEYPRESS

debounce:
- If key matches stable key:
  - Increment counter
  - If counter exceeds threshold:
    - Return true
- Else:
  - Update stable key
  - Reset counter

Grab_KeyPress:
- Loop:
  - Turn ON LED
  - If any key is pressed:
    - Detect key
    - If key is stable:
      - If key is valid:
        - Turn off all LEDs
        - Set LEDs for feedback
        - Wait until key is released
        - Turn off LEDs

ESP32 Base Station Pseudocode:

Initialization:
1. Initialize I2C Communication:
   - Set device address to `0x51`.
   - Assign `SDA_PIN` and `SCL_PIN`.
   - Register event handlers for:

- onReceive: Store incoming keypad values.
- onRequest: Respond with the current data buffer.

2. Initialize LoRa Communication:
   - Set frequency: 915 MHz.
   - Set TX Power: 14 dBm.
   - Set Bandwidth: 125 kHz.
   - Set Spreading Factor: SF7.
   - Set Coding Rate: 4/5.
   - Set Preamble Length: 8 symbols.
   - Register handler for RX Done event.
3. Print "Press 1 to enter Tank Mode."

Main Loop:
1. If Tank Mode is Inactive:
   - Wait for user to press `1`.
   - Activate Tank Mode.
2. If Tank Mode is Active but Not Configured:
   - Prompt user to enter tank height.
   - Prompt user to enter max water level.
   - Calculate offset: `tankHeight - maxWaterLevel`.
   - Mark tank as configured.
3. If Tank Mode is Configured:
   - Enter LoRa RX Mode.
   - Wait for incoming LoRa data.
   - Process received data.
4. Add short delay for stability.

Functions:
1. I2C Event Handlers:
   - onReceive:
     - Store received I2C data in buffer.
     - Update the keypad value.
   - onRequest:
     - Send the buffer's data back to the master.
2. Keypad Data Processing:
   - Store Keypad Value:
     - Save the first byte of the buffer as the keypad value.
3. LoRa Event Handlers:
   - OnRxDone:
     - Save received payload and RSSI.

- Parse payload as sensor reading.
- Adjust sensor reading using pre-configured offset.
- Print adjusted value.
- Set system to idle.

4. Tank Configuration:
   - Prompt user for tank height and max water level:
     - Collect 5 digits from keypad.
     - Parse digits into meters, centimeters, and fractional cm.
     - Convert parsed values to centimeters.
   - Calculate offset: `tankHeight - maxWaterLevel`.
5. Sensor Reading Adjustment:
   - Adjust sensor reading using offset.
   - Calculate fill percentage:
     - `(tankHeight - sensorReading) / maxWaterLevel * 100`.
   - Print fill percentage.

ESP32 Remote Unit Pseudocode:

Initialization:
1. Set up Serial communication at 115200 baud.
2. Initialize the Heltec LoRa module:
   - Set frequency: 915 MHz.
   - Set TX Power: 5 dBm.
   - Set Bandwidth: 125 kHz.
   - Set Spreading Factor: SF7.
   - Set Coding Rate: 4/5.
   - Set Preamble Length: 8 symbols.
   - Configure TxDone and TxTimeout event handlers.
3. Configure ultrasonic sensor pins:
   - Set `trigPin` as OUTPUT.
   - Set `echoPin` as INPUT.

Main Loop:
1. If LoRa is idle:
   - Delay for 1 second.
   - Measure the distance using the ultrasonic sensor:
     - Send a 10-microsecond pulse on `trigPin`.
     - Measure echo pulse duration on `echoPin`.
     - Calculate distance in centimeters:
       `distance = (duration * 0.034) / 2`.

- Format the distance as a string.
- Print the distance and formatted string to Serial.
- Send the distance over LoRa.
- Set `lora_idle` to false.

2. Call `Radio.IrqProcess()` to handle LoRa IRQs.

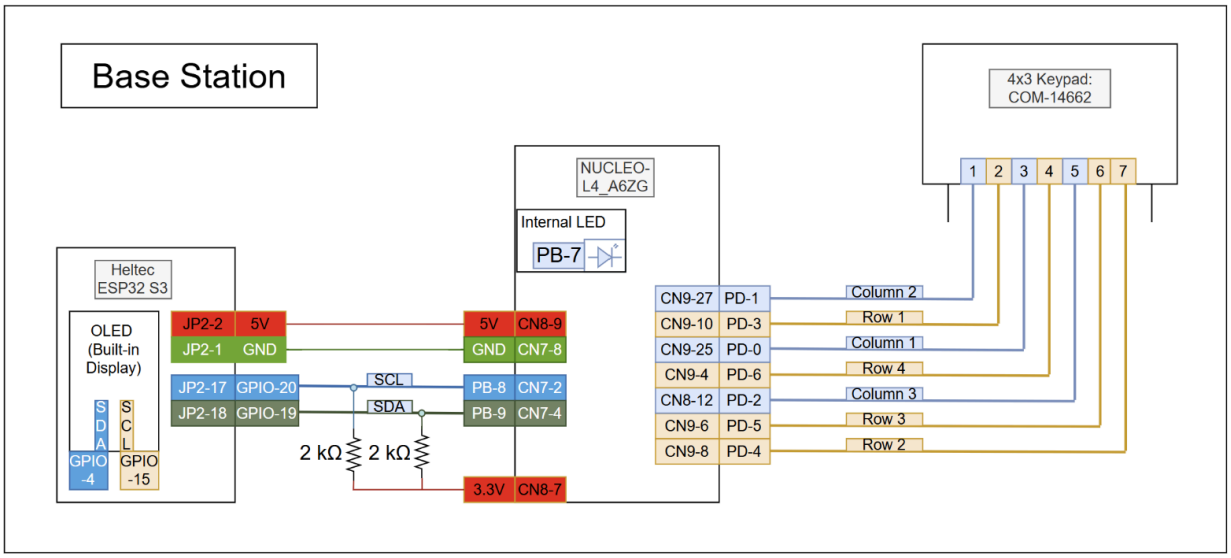
Functions:

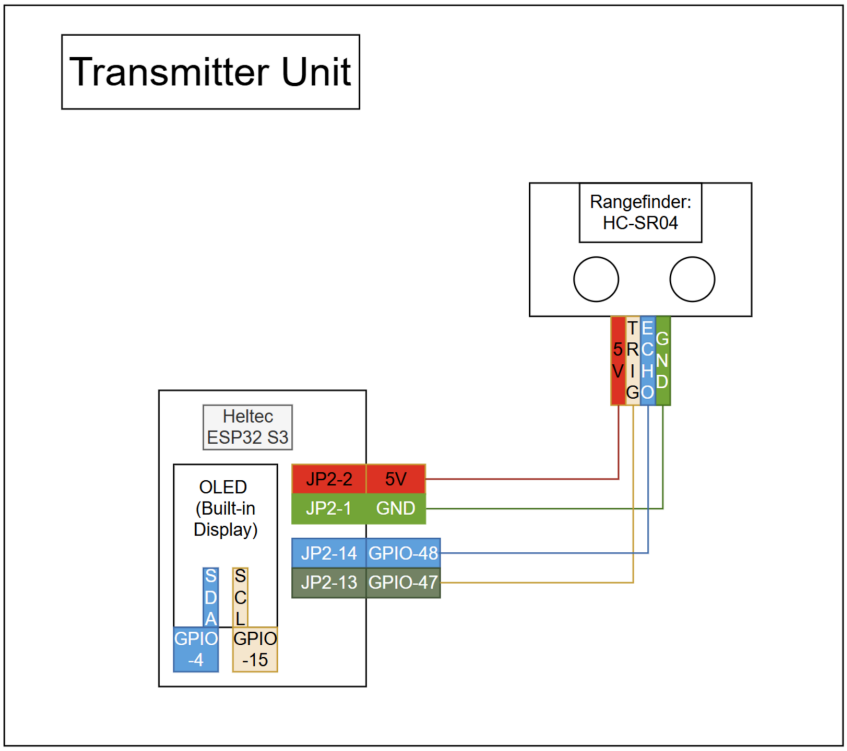
1. getSDist:
   - Clear the `trigPin` by setting it LOW.
   - Send a 10-microsecond HIGH pulse on `trigPin`.
   - Measure the echo pulse duration using `pulseIn`.
   - Calculate the distance:
     `distance = (duration * 0.034) / 2`.
   - Print the distance to Serial and return it.

2. OnTxDone:
   - Print "TX done..." to Serial.
   - Set `lora_idle` to true.

3. OnTxTimeout:
   - Put the LoRa module to sleep.
   - Print "TX Timeout..." to Serial.
   - Set `lora_idle` to true.

System Schematic





Attribution

Personnel	Jordan Reichhardt	John Mort	Kevin Swanson
Project Contributions:	● I2C development and debugging ● ESP32 code integration for both sender and receiver - Connecting I2C for ESP32 and Nucleo - Debugging sender and receiver code - Integrating ultrasonic code and calculations into sender code	● Hardware/assembly ● System schematic ● System specifications ● I2C development and keypad integration on STM32 ● Debugging I2C on ESP32	● Solar Panel Power R&D ○ Soldering ○ Splicing cables ○ Voltage testing ○ Circuit Assembly ● Bill of Materials

Appendix

A: Bill of Materials

qty	part type	designator/ id	package/ footprint	description	manufacturer	manufacturer p/n	vendor : p/n	unit $	$ qty	cog $	purch $	notes
1	MCU	MCU1	L4A6ZG-ND	NUCLEO-144 STM32L4A6 ZG EVAL BRD	STMicroelectronics	NUCLEO-L4A6ZG	DK: 497-NUCLEO-L4A6ZG-ND	$19.99	1	$19.99	$19.99	STM NUCLEO-L4A6ZG
1	Keypad Switch	Switch1	1136-ND	SWITCH KEYPAD 12KEY NON-ILLUM	Adafruit Industries LLC	419	DK: 419	$3.95	1	$3.95	$3.95	4x3 Keypad Membrane
1	Sensor	Sensor1	0.79" x 0.79" x 0.79"	Ultrasonic Ranging Module 13ft	DONGKER	1670	AZ: B0C5QB4RLR	$13.99	1	$13.99	$13.99	Ultrasonic Sensor

2	MCU	MCU2	LoRa V3	915MHz ESP32S3 WiFi BT Development Board	Heltec	S3FN8	AZ: B08629526P	$58.09	1	$58.09	$58.09	Part of Bundle with header pins, antenna, cases, connector cables
1	Sensor	Sensor2	ETFE	6V 2W Solar Panel	Voltaic	P126	AF: 5366	$20.95	1	$20.95	$20.95	Solar Panel
1	INTEG. CKT	CKT1	1.6" x 1.3" x 0.08"	USB / DC / Solar Lithium Ion/Polymer charger	Adafruit Industries LLC	390	AF: 390	$17.50	1	$17.50	$17.50	Charging Circuit
1	INTEG. CKT	CKT2	1.1" x .9" x .1"	PowerBoost 1000 Basic	Adafruit Industries LLC	2030	AF: 2030	$14.95	1	$14.95	$14.95	Power Boost Circuit
1	Thermistor	Therm1	0.13”	Precision Epoxy Thermistor	Adafruit Industries LLC	3950	AF: 372	$4.00	1	$4.00	$4.00	Thermal Resistor
1	Button Switch	Button1	0.7" x 0.7" x 1.15"	Illuminated Pushbutton - White Latching On/Off Switch	Adafruit Industries LLC	1478	AF: 1478	$1.95	1	$1.95	$1.95	Illuminating Push Button
1	Battery	Bat1	2.71" x 0.71"	Lithium Ion Cylindrical Battery - 3.7v 2200mAh	Adafruit Industries LLC	1781	AF: 1781	$9.95	1	$9.95	$9.95	Rechargeable Battery
									TOTAL	$165.32	$165.32	
							Vendor Key					
							DK: DigiKey					
							AZ: Amazon					
							AF: Adafruit					

B: References

[1] Instructables, “Solar Class: USB Charger,” *Instructables*, Oct. 26, 2017. https://www.instructables.com/Solar-USB-Charger-2/ (accessed Nov. 25, 2024).
[2] Espressif, “ESP32-S3 Series Datasheet 2.4 GHz Wi-Fi + Bluetooth Including,” Sep. 2024. Accessed: Nov. 25, 2024. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf

[3] Espressif, “ESP32-S3 Technical Reference Manual,” Apr. 2024. Accessed: Nov. 25, 2024. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_technical_reference_manual_en.pdf

[4] Heltec, “HTIT-WB32LA_V3 LoRa Node Development Kit,” Aug. 2022. Accessed: Nov. 25, 2024. [Online]. Available: https://resource.heltec.cn/download/WiFi_LoRa_32_V3/HTIT-WB32LA_V3(Rev1.1).pdf

[5] Squix, “Angularclient,” *Squix.ch*, 2024. https://oleddisplay.squix.ch/ (accessed Nov. 25, 2024).

[6] Shug, “Heltec ESP32 Series Quick Start — esp32 latest documentation,” *Heltec.org*, 2022. https://docs.heltec.org/en/node/esp32/esp32_general_docs/quick_start.html (accessed Nov. 25, 2024).

C: Source Code

STM32 Code:

```
/* ------------------------------------------------------------------------*/
 main.h
/*-------------------------------------------------------------------------*/
 ***************************************************************************
 * @file           : main.h
 * @brief          : Header for main.c file.
 *                   This file contains the common defines of the application.
 ***************************************************************************
 * @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.
 *
 ***************************************************************************
 */
#ifndef __MAIN_H
#define __MAIN_H
#ifdef __cplusplus
extern "C" {
#endif
#include "stm32l4xx_hal.h"
void Error_Handler(void);
void SystemClock_Config(void);
void Delay_ms(uint32_t ms);
/* USER CODE BEGIN EFP */
/* USER CODE END EFP */
#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
#define LD3_Pin GPIO_PIN_14
#define LD3_GPIO_Port GPIOB
#define USB_OverCurrent_Pin GPIO_PIN_5
#define USB_OverCurrent_GPIO_Port GPIOG
#define USB_PowerSwitchOn_Pin GPIO_PIN_6
#define USB_PowerSwitchOn_GPIO_Port GPIOG
#define STLK_RX_Pin GPIO_PIN_7
#define STLK_RX_GPIO_Port GPIOG
#define STLK_TX_Pin GPIO_PIN_8
#define STLK_TX_GPIO_Port GPIOG
```

```
#define USB_SOF_Pin GPIO_PIN_8
#define USB_SOF_GPIO_Port GPIOA
#define USB_VBUS_Pin GPIO_PIN_9
#define USB_VBUS_GPIO_Port GPIOA
#define USB_ID_Pin GPIO_PIN_10
#define USB_ID_GPIO_Port GPIOA
#define USB_DM_Pin GPIO_PIN_11
#define USB_DM_GPIO_Port GPIOA
#define USB_DP_Pin GPIO_PIN_12
#define USB_DP_GPIO_Port GPIOA
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
#define LD2_Pin GPIO_PIN_7
#define LD2_GPIO_Port GPIOB
#ifdef __cplusplus
}
#endif
#endif /* __MAIN_H */
/* ----------------------------------------------------------------------------
*/



/*-----------------------------------------------------------------------------*/
 main.c
/*-----------------------------------------------------------------------------*/

* @file main.c
* @brief Main application for interfacing a keypad and transmitting key presses
*  via I2C.
*
* This module is the entry point for the application. It initializes hardware
* peripherals, handles the main execution loop, and coordinates the keypad and
* I2C communication functionality. The application detects key presses from a
* 4x3 keypad and sends the corresponding key value to an ESP32 via I2C.
*
* Hardware Connections:
* Keypad Columns (Outputs):** GPIO pins PD0, PD1, PD2
* Keypad Rows (Inputs):** GPIO pins PD3, PD4, PD5, PD6
* I2C SDA: GPIO pin PB9
* I2C SCL: GPIO pin PB8
* I2C Slave Address: 0x51
* Status LED: GPIO pin PB7
*
* Functionality:
* - Detect and debounce key presses from the keypad.
```

```
* - Encode key presses into numerical values or control characters.
* - Transmit the detected key value to an I2C slave device.
* - Provide system clock configuration and delay functionality.
* ------------------------------------------------------------------------------
*/
#include "main.h"
#include "i2c.h"
#include "keypad.h"
/*
*-------------------------------------------------------------------------------
* Function    : main
* Purpose     : Initializes Hal, system clock, GPIOS, I2C, and Keypad module
*
* Inputs      : None
* Outputs     : None
* Authors     : John Mort, Kevin Swanson, Jordan Reichhardt
* Version     : 1.2
* Date        : 2024-12-05
* ------------------------------------------------------------------------------
*/
int main(void) {
   // Initialize HAL, clock, and GPIO
   HAL_Init();
   SystemClock_Config();
   GPIO_I2C1_Init();
   I2C1_Init();
   Keypad_Config();
   // Configure LED on PB7
   RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
   GPIOB->MODER &= ~(GPIO_MODER_MODE7); // Clear mode bits
   GPIOB->MODER |= (1 << GPIO_MODER_MODE7_Pos); // Set as output
   uint8_t number_to_send = 6;
   while (1) {
       number_to_send = Grab_KeyPress();  // Detect key press
       I2C_SendNumber(number_to_send);   // Send the pressed key
       Delay_ms(10);  // Delay for stability
   }
}
/*
*-------------------------------------------------------------------------------
* Function    : Delay_ms
* Purpose     : Delay function in ms
* Inputs      : ms - Delay duration in milliseconds.
* Outputs     : None
* Authors     : Kevin Swanson, Jordan Reichhardt, John Mort
* Version     : 1.2
* Date        : 2024-12-05
* ------------------------------------------------------------------------------ */
void Delay_ms(uint32_t ms) {
```

```
    SysTick->LOAD = 16000 - 1; // 1 ms delay at 16 MHz
    SysTick->VAL = 0;          // Clear current value
    SysTick->CTRL = 5;         // Enable SysTick with processor clock
    for (uint32_t i = 0; i < ms; i++) {
        while (!(SysTick->CTRL & 0x10000)); // Wait for COUNTFLAG
    }
    SysTick->CTRL = 0; // Disable SysTick
}
/*
*-----------------------------------------------------------------------
* Function    : SystemClock_Config
* Purpose     : Configures the system clock.
* Inputs      : None
* Outputs     : None
* Authors     : Kevin Swanson, Jordan Reichhardt, John Mort
* Version     : 1.2
* Date        : 2024-12-01
* ---------------------------------------------------------------------- */
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();
    }
    // Initialize the RCC Oscillators according to the specified parameters
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
    RCC_OscInitStruct.MSIState = RCC_MSI_ON;
    RCC_OscInitStruct.MSICalibrationValue = 0;
    RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_8;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
        Error_Handler();
    }
    // Initialize 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_0) != HAL_OK) {
        Error_Handler();
    }
}
void Error_Handler(void) {
    __disable_irq();
    while (1) {
```

```
    }
}
```

```
/*-------------------------------------------------------------------------*/
 I2C.h
/*-------------------------------------------------------------------------*/
#ifndef I2C_H
#define I2C_H
#include "stm32l4xx.h"
// I2C Address
#define I2C_SLAVE_ADDRESS 0x51
#define I2C_TIMEOUT_MS 10  // Timeout for I2C operations
// Function Prototypes
void GPIO_I2C1_Init(void);
void I2C1_Init(void);
void I2C_SendNumber(uint8_t number);
#endif /* I2C_H */
```

```
/*-------------------------------------------------------------------------*/
 I2C.c
/*-------------------------------------------------------------------------**
@file i2c.c* @brief Header file for I2C module configuration and data
transmission.*
* This module has function to integrate I2C communication with main STM code so
taht the keypad instructions can be sent to the base unit ESP32 this will then
trigger the sender (remote) ESP32 to send ultrasonic data.
--------------------------------------------------------------------------
*/
#include "i2c.h"
#include "main.h"
/*
*-------------------------------------------------------------------------
* Function    : GPIO_I2C1_Init
* Purpose     : Configures PB8 (SLC) and PB9 (SDA)
* Inputs      : None
* Outputs     : None
* Authors     : John Mort, Jordan Reichhardt, Kevin Swanson
* Version     : 1.2
* Date        : 2024-12-01
*------------------------------------------------------------------------- */
void GPIO_I2C1_Init(void) {
   RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN; // Enable GPIOB clock
   // Configure PB8 (SCL) and PB9 (SDA) as alternate function (AF4)
   GPIOB->MODER &= ~(GPIO_MODER_MODE8 | GPIO_MODER_MODE9);
```

```
    GPIOB->MODER |= (GPIO_MODER_MODE8_1 | GPIO_MODER_MODE9_1); // AF mode
    GPIOB->OTYPER |= (GPIO_OTYPER_OT8 | GPIO_OTYPER_OT9);       // Open-drain
    GPIOB->OSPEEDR |= (GPIO_OSPEEDR_OSPEED8 | GPIO_OSPEEDR_OSPEED9); // Medium
speed
    GPIOB->PUPDR &= ~(GPIO_PUPDR_PUPD8 | GPIO_PUPDR_PUPD9);     // No
pull-up/pull-down
    GPIOB->AFR[1] |= (4 << GPIO_AFRH_AFSEL8_Pos) | (4 << GPIO_AFRH_AFSEL9_Pos);
// AF4
}
/* ---------------------------------------------------------------------------
* Function    : I2C1_Init
* Purpose     : Initializes the I2C1 peripheral with 400 kHz settings.
* Inputs      : None
* Outputs     : None
* Authors     : Jordan Reichhardt, John Mort, Kevin Swanson
* Version     : 1.2
* Date        : 2024-12-01
*--------------------------------------------------------------------------- */
void I2C1_Init(void) {
    RCC->APB1ENR1 |= RCC_APB1ENR1_I2C1EN; // Enable I2C1 clock
    I2C1->CR1 &= ~I2C_CR1_PE;             // Disable I2C peripheral
    I2C1->CR1 &= ~I2C_CR1_ANFOFF;         // Enable analog filter
    I2C1->CR1 &= ~I2C_CR1_DNF;            // Disable digital filter
    I2C1->TIMINGR = 0x00303D5B;           // Timing for 16 MHz SYSCLK
    I2C1->CR2 &= ~I2C_CR2_ADD10;          // 7-bit addressing
    I2C1->CR1 |= I2C_CR1_PE;              // Enable I2C peripheral
}
/* ---------------------------------------------------------------------------
* Function    : I2C_SendNumber
* Purpose     : Sends a single byte (number) to the ESP32 over I2C.
* Inputs      : number - Byte to send over I2C.
* Outputs     : None
* Authors     : John Mort, Kevin Swanson, Jordan Reichhardt
* Version     : 1.2
* Date        : 2024-12-01
*--------------------------------------------------------------------------- */
void I2C_SendNumber(uint8_t number) {
    uint32_t timeout = I2C_TIMEOUT_MS;
    // Wait until the I2C bus is free or timeout
    while ((I2C1->ISR & I2C_ISR_BUSY) && timeout > 0) {
        Delay_ms(1);
        timeout--;
    }
    if (timeout == 0) {
        return;  // Exit if the bus is busy
    }
    // Configure the I2C transaction
    I2C1->CR2 = (I2C_SLAVE_ADDRESS << 1)   // Set slave address (7-bit) for
write
```

```
                | (1 << I2C_CR2_NBYTES_Pos) // 1 byte to send
                | I2C_CR2_AUTOEND;          // Generate STOP after transmission
    I2C1->CR2 &= ~I2C_CR2_RD_WRN;           // Write mode
    I2C1->CR2 |= I2C_CR2_START;             // Generate START condition
    // Wait for TXIS flag
    timeout = I2C_TIMEOUT_MS;
    while (!(I2C1->ISR & I2C_ISR_TXIS) && timeout > 0) {
        Delay_ms(1);
        timeout--;
    }
    if (timeout == 0) {
        return;  // Exit if TXIS flag is not set
    }
    // Send data byte
    I2C1->TXDR = number;
    // Wait for TC flag
    timeout = I2C_TIMEOUT_MS;
    while (!(I2C1->ISR & I2C_ISR_TC) && timeout > 0) {
        Delay_ms(1);
        timeout--;
    }
    if (timeout == 0) {
        return;  // Exit if TC flag is not set
    }
    // Clear the STOP condition flag
    I2C1->ICR = I2C_ICR_STOPCF;
}


/*-----------------------------------------------------------------------------*/
 keypad.h
/*-----------------------------------------------------------------------------*/
/* USER CODE END Header */
#ifndef INC_KEYPAD_H_
#define INC_KEYPAD_H_
#define clr (0xf)
#define KYPD_Port (GPIOD)                   // port used by keypad
// function prototyping
int Keypad_IsAnyKeyPressed(void);
int Keypad_WhichKeyIsPressed(void);
void Keypad_Config(void);
int Grab_KeyPress(void);
int debounce(int key);
#define deb_cnt (8)
/*
Keypad Pins left-to-right:
1, 2, 3, 4, 5, 6, 7
Rows (top-to-bottom): 2, 7, 6, 4
Columns (left-to-right):  3, 1, 5
```

```
*/
// Columns
#define c1 (1<<0)
#define c2 (1<<1)
#define c3 (1<<2)
#define COL_PINS (c1|c2|c3)
// Rows
#define r1 (1<<3)
#define r2 (1<<4)
#define r3 (1<<5)
#define r4 (1<<6)
#define ROW_PINS (r1|r2|r3|r4)
#define KEY_ZERO (11)          //checks output of special function
#define CODE_ZERO (0)          //LED value of 0
#define KEY_STAR (10)          //checks output of special function
#define CODE_STAR (14)         //LED value of 14
#define KEY_POUND (12)         //checks output of special function
#define CODE_POUND (15)  //LED value of 15
#define NO_KEYPRESS (-1)
#define NUM_COLS (3)
#define NUM_ROWS (4)
#define SETTLE (1000)
#define TRUE (1)
#define FALSE (0)
#endif /* INC_KEYPAD_H_ */


/* ------------------------------------------------------------------------*/
 keypad.c
/*-------------------------------------------------------------------------*/*
@file KeyPad.c
* @brief Keypad interfacing and key press detection functionality for Aquasense
project.
* This module configures the GPIO for keypad interaction and includes functions
* to detect, debounce, and encode key presses. It operates with a 4x3 keypad,
* where key values are transmitted as encoded numeric or control characters.
* ### Hardware Connections:
* - **Keypad Columns (Outputs):** GPIO pins PD0, PD1, PD2
* - **Keypad Rows (Inputs):** GPIO pins PD3, PD4, PD5, PD6
* - **Status LED:** GPIO pin PB7
*
* ### Functionality:
* - Configures GPIO pins for keypad columns and rows.
* - Detects key presses and ensures stability via debouncing.
* - Encodes key presses into usable numeric or control codes.
*
-------------------------------------------------------------------------*/#
include "KeyPad.h"
#include "main.h"
```

```
/*
*-----------------------------------------------------------------------------
* Function    : Keypad_Config
* Purpose     : Configures GPIO for keypad interaction (columns and rows).
* Inputs      : None
* Outputs     : None
* Authors     : Kevin Swanson, Jordan Reichhardt, John Mort
* Blame       : Adapted from A5 keypad controls (William Leiker, John Mort).
* Version     : 1.2
* Date        : 2024-12-05
*----------------------------------------------------------------------------- */
void Keypad_Config(void) {
RCC->AHB2ENR |= (RCC_AHB2ENR_GPIODEN); // Enable GPIO clock for keypad
// Clear mode registers for all pins   KYPD_Port->MODER &= ~(GPIO_MODER_MODE0 |
GPIO_MODER_MODE1 |
```

```
                        GPIO_MODER_MODE2 |
                        GPIO_MODER_MODE3 |
                        GPIO_MODER_MODE4 |
                        GPIO_MODER_MODE5 |
                        GPIO_MODER_MODE6 |
                        GPIO_MODER_MODE7);
    // Set columns as output
    KYPD_Port->MODER |= (GPIO_MODER_MODE0_0 |
                         GPIO_MODER_MODE1_0 |
                         GPIO_MODER_MODE2_0);
    // Set all pins to push-pull mode
    KYPD_Port->OTYPER &= ~(GPIO_OTYPER_OT0 |
                           GPIO_OTYPER_OT1 |
                           GPIO_OTYPER_OT2 |
                           GPIO_OTYPER_OT3 |
                           GPIO_OTYPER_OT4 |
                           GPIO_OTYPER_OT5 |
                           GPIO_OTYPER_OT6 |
                           GPIO_OTYPER_OT7);
    // Clear and set pull-down resistors for all pins
    KYPD_Port->PUPDR &= ~(GPIO_PUPDR_PUPD0 |
                          GPIO_PUPDR_PUPD1 |
                          GPIO_PUPDR_PUPD2 |
                          GPIO_PUPDR_PUPD3 |
                          GPIO_PUPDR_PUPD4 |
                          GPIO_PUPDR_PUPD5 |
                          GPIO_PUPDR_PUPD6 |
                          GPIO_PUPDR_PUPD7);
    KYPD_Port->PUPDR |= (GPIO_PUPDR_PUPD0_1 |
                         GPIO_PUPDR_PUPD1_1 |
                         GPIO_PUPDR_PUPD2_1 |
                         GPIO_PUPDR_PUPD3_1 |
                         GPIO_PUPDR_PUPD4_1 |
                         GPIO_PUPDR_PUPD5_1 |
                         GPIO_PUPDR_PUPD6_1 |
                         GPIO_PUPDR_PUPD7_1);
    // Set columns to very high speed
    KYPD_Port->OSPEEDR |= ((3 << GPIO_OSPEEDR_OSPEED0_Pos) |
                           (3 << GPIO_OSPEEDR_OSPEED1_Pos) |
                           (3 << GPIO_OSPEEDR_OSPEED2_Pos));
}
/*
*-----------------------------------------------------------------------------
* Function    : Keypad_IsAnyKeyPressed
* Purpose     : Checks if any key is pressed on the keypad.
* Inputs      : None
* Outputs     : TRUE if a key is pressed, FALSE otherwise.
* Authors     : Jordan Reichhardt, John Mort, Kevin Swanson
* Blame       : Lacks software debounce; may detect spurious key presses.
```

```
* Version     : 1.2
* Date        : 2024-12-05
*------------------------------------------------------------------------- */
int Keypad_IsAnyKeyPressed(void) {
   KYPD_Port->BSRR = COL_PINS; // Set all columns high
   for (uint16_t idx = 0; idx < SETTLE; idx++); // Wait for signals to settle
   if ((KYPD_Port->IDR & ROW_PINS) != 0) { // Check if any row is high
       return TRUE;
   }
   return FALSE;
}
/*
*-------------------------------------------------------------------------
* Function    : Keypad_WhichKeyIsPressed
* Purpose     : Determines which key on the keypad is pressed and encodes it.
* Inputs      : None
* Outputs     : Encoded key ID or NO_KEYPRESS.
* Authors     : John Mort, Kevin Swanson, Jordan Reichhardt
* Blame       : Lacks robust error handling for overlapping key presses.
* Version     : 1.2
* Date        : 2024-12-05
*------------------------------------------------------------------------- */
int Keypad_WhichKeyIsPressed(void) {
   int8_t iRow = 0, iCol = 0, iKey = 0;
   int8_t bGotKey = 0;
   KYPD_Port->BSRR = COL_PINS; // Set all columns high
   for (iRow = 0; iRow < NUM_ROWS; iRow++) { // Check all rows
       if (KYPD_Port->IDR & (1 << (3 + iRow))) { // Key press in iRow
           KYPD_Port->BRR = COL_PINS; // Set all columns low
           for (iCol = 0; iCol < NUM_COLS; iCol++) { // Check all columns
               KYPD_Port->BSRR = (1 << iCol); // Set current column high
               if (KYPD_Port->IDR & (1 << (3 + iRow))) { // Key press in iCol
                   bGotKey = 1;
                   break;
               }
           }
           if (bGotKey) break;
       }
   }
   if (bGotKey) {
       iKey = (iRow * NUM_COLS) + iCol + 1; // Encode numeric keys
       switch (iKey) {
           case KEY_STAR: iKey = CODE_STAR; break;
           case KEY_ZERO: iKey = CODE_ZERO; break;
           case KEY_POUND: iKey = CODE_POUND; break;
           default: break;
       }
       return iKey;
   }
```

```
    return NO_KEYPRESS; // No key press detected
}
/*
*-----------------------------------------------------------------------
* Function    : debounce
* Purpose     : Stabilizes key press detection by ensuring consistent input.
* Inputs      : key - Key ID to debounce.
* Outputs     : TRUE if the key is stable, FALSE otherwise.
* Authors     : Kevin Swanson, Jordan Reichhardt, John Mort
* Blame       : May fail with rapid key press and release.
* Version     : 1.2
* Date        : 2024-12-05
*----------------------------------------------------------------------- */
int debounce(int key) {
    static int stable_key = -1;
    static int cnt = 0;
    if (key == stable_key) {
        cnt++;
        if (cnt >= deb_cnt) {
            return TRUE;
        }
    } else {
        stable_key = key;
        cnt = 0;
    }
    return FALSE;
}
/*
*-----------------------------------------------------------------------
* Function    : Grab_KeyPress
* Purpose     : Captures a stable key press from the keypad.
* Inputs      : None
* Outputs     : Encoded key value or NO_KEYPRESS.
* Authors     : Jordan Reichhardt, John Mort, Kevin Swanson
* Blame       : Busy waiting; no multitasking support.
* Version     : 1.2
* Date        : 2024-12-05
*----------------------------------------------------------------------- */
int Grab_KeyPress(void) {
    int key = 0;
    while (1) {
        GPIOB->ODR |= (1 << 7); // Turn ON LED
        if (Keypad_IsAnyKeyPressed()) {
            key = Keypad_WhichKeyIsPressed();
            if (debounce(key)) {
                if (key != NO_KEYPRESS) {
                    GPIOC->BRR = 0xF; // Turn off all LEDs
                    GPIOC->BSRR |= (key); // Set LEDs for feedback
                    while (Keypad_IsAnyKeyPressed()); // Wait for release
```

```
                    GPIOB->ODR &= ~(1 << 7); // Turn OFF LED
                    return key;
                }
            }
        }
    }
}
```

ESP32 Code:

```
/* ------------------------------------------------------------------------
* Function    : onReceive
* Purpose     : Manages the I2C receive event and stores received data in a
buffer.
* Inputs      : Length of the received data.
* Outputs     : None
* Authors     : Jordan Reichhardt, John Mort, Kevin Swanson
* Blame       : Used Chat GPT to outline how to store data in a buffer
* Version     : 1.2
* Date        : 2024-12-05.
* ------------------------------------------------------------------------
*/
void onReceive(int len) {
    memset((void *)rxBuffer, 0, sizeof(rxBuffer)); // Clear the buffer
    dataLength = 0;

    while (Wire.available()) {
        size_t length = dataLength;
        if (length < sizeof(rxBuffer)) { // Prevent thw buffer from
overflowing
            rxBuffer[length] = Wire.read();
            dataLength = length + 1; //increment data length
        }
    }

    storeKeypadValue(); // Store the value pressed on keypad on STM32
}

/*
------------------------------------------------------------------------
* Function    : onRequest
* Purpose     : Manages I2C request event and sends data from the buffer.
```

```
* Inputs      : None
* Outputs     : None
* Authors     : Jordan Reichhardt, John Mort, Kevin Swanson
* Version     : 1.2
* Date        : 2024-12-05
* ----------------------------------------------------------------------------
*/
void onRequest() {
    Wire.write((const uint8_t *)rxBuffer, dataLength);
}

/*
------------------------------------------------------------------------------
* Function    : storeKeypadValue
* Purpose     : Stores the first value from the I2C buffer in order to key
keypad values coming from STM32 in a global variable.
* Inputs      : None
* Outputs     : None
* Authors     : Jordan Reichhardt, John Mort, Kevin Swanson
* Blame       : Used ChatGPT to structure buffer array and integrate
storeKeypadValue with onRecieve function.
* Version     : 1.2
* Date        : 2024-12-05
* ----------------------------------------------------------------------------
*/
void storeKeypadValue() {
    if (dataLength > 0) {
        keypadValue = rxBuffer[0]; // Store first value from buffer
        Serial.printf("Keypad Value Stored: %d\n", keypadValue);
    } else {
        keypadValue = -1; // Reset keypad value if data isn't valid
    }
}

/*
------------------------------------------------------------------------------
* Function    : OnRxDone
* Purpose     : Processes the data from the sender if the receive event over
LORA has gone through
* Inputs      : payload - Pointer to the received data from LoRa.
```

```
*                size - Size of data from LoRa.
*                rssi - Received Signal Strength Indicator.
*                snr - Signal-to-Noise Ratio.
* Outputs     : None
* Authors     : Jordan Reichhardt, John Mort, Kevin Swanson
* Blame       : Function was adapted from Arduino IDE example code for LORA
receivers.
* Version     : 1.2
* Date        : 2024-12-05
*overflow risk.
* ---------------------------------------------------------------------------
*/
void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr) {
    rssi = rssi;
    rxSize = size;
    memcpy(rxpacket, payload, size);
    rxpacket[size] = '\0';
    Radio.Sleep();
    Serial.printf("\r\nReceived packet \"%s\" with RSSI %d, length %d\r\n",
rxpacket, rssi, rxSize);

    // Adjust sensor reading with the offset for tank % calculation before
printing
    float sensorReading = atof(rxpacket); //string to float
    float adjustedReading = adjustSensorReading(sensorReading);
    Serial.printf("Adjusted Sensor Reading: %.2f cm\n", adjustedReading);

    lora_idle = true;
}

/*
---------------------------------------------------------------------------
* Function    : setup
* Purpose     : Initializes the I2C and LoRa communication so that
communication can function between all three microcontrollers
* Inputs      : None
* Outputs     : None
* Authors     : Jordan Reichhardt, John Mort, Kevin Swanson
* Blame       : Adapted from Arduino IDE example LoRa receiver code
* Version     : 1.2
```

```
* Date         : 2024-12-05
* ------------------------------------------------------------------------
*/
void setup() {
    Serial.begin(115200);

    setupI2C();  // Initialize I2C
    setupLoRa(); // Initialize LoRa

    Serial.println("Press 1 to enter Tank Mode");
}

/*
--------------------------------------------------------------------------
* Function     : loop
* Purpose      : Main code to link LoRa communication and calculations for tank
metrics based on ultrasonic data being received over LoRa
* Inputs       : None
* Outputs      : None
* Authors      : Jordan Reichhardt, Kevin Swanson, John Mort
* Blame        : Used ChatGPT to structure integration of LoRa data into
conditional statement
* Version      : 1.2
* Date         : 2024-12-05
* ------------------------------------------------------------------------
*/
void loop() {
    if (!tankModeActive) {
        if (keypadValue == 1) { // Check if the keypad input is '1'
            tankModeActive = true;
            keypadValue = -1; // Reset keypad value
            Serial.println("Tank Mode Activated!");
        }
    } else if (!tankConfigured) {
        configureTankAndWaterLevel();
        tankConfigured = true;
    } else {
        if (lora_idle) {
            lora_idle = false;
            Serial.println("Requesting data from sender...");
```

```
            Radio.Rx(0); // Enter RX mode
        }
        Radio.IrqProcess(); // Process LoRa IRQ
    }

    delay(100); // Small delay for stability
}

/*
-------------------------------------------------------------------------------
* Function    : setupI2C
* Purpose     : Initializes I2C communication so that ESP32 can connect to
user imput connected to STM32
* Inputs      : None
* Outputs     : None
* Authors     : John Mort, Jordan Reichhardt, Kevin Swanson
* Blame       : Inspired by Heltec ESP32 I2C scanner example
* Version     : 1.4
* Date        : 2024-12-05
* -----------------------------------------------------------------------------
*/
void setupI2C() {
    if (!Wire.begin(I2C_DEV_ADDR, SDA_PIN, SCL_PIN, 100000)) {
        Serial.println("Failed to initialize I2C slave!");
        while (1);
    }
    Serial.println("I2C slave initialized successfully.");

    Wire.onReceive(onReceive);
    Wire.onRequest(onRequest);
}

/*
-------------------------------------------------------------------------------
* Function    : setupLoRa
* Purpose     : Configure the LoRa module for communication between receiver
(base station) and sender (remote station).
* Inputs      : None
* Outputs     : None
* Authors     : Jordan Reichhardt, John Mort, Kevin Swanson
```

```
* Blame       : Used Arduino IDE example ESP32 sender code to structure this
function
* Version     : 1.2
* Date        : 2024-12-05
* ------------------------------------------------------------------------
*/
void setupLoRa() {
    Mcu.begin(HELTEC_BOARD, SLOW_CLK_TPYE);
    RadioEvents.RxDone = OnRxDone;

    Radio.Init(&RadioEvents);
    Radio.SetChannel(RF_FREQUENCY);
    Radio.SetRxConfig(MODEM_LORA, LORA_BANDWIDTH, LORA_SPREADING_FACTOR,
                      LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,
                      LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON,
                      0, true, 0, 0, LORA_IQ_INVERSION_ON, true);
    Serial.println("LoRa receiver initialized.");
}

/*
------------------------------------------------------------------------
* Function    : configureTankAndWaterLevel
* Purpose     :Based on user input on keypad for tank height and max water
level this function calculates the offset from sensor to max water level and
zeros out the sensor.
* Inputs      : None
* Outputs     : None
* Authors     : Kevin Swanson, Jordan Reichhardt, John Mort
* Version     : 1.2
* Date        : 2024-12-05
* ------------------------------------------------------------------------
*/
void configureTankAndWaterLevel() {
    tankHeight = getTankOrWaterLevel("Enter tank height (m, cm, cm, .cm):");
    Serial.printf("Tank Height Configured: %.2f cm\n", tankHeight);

    maxWaterLevel = getTankOrWaterLevel("Enter max water level (m, cm, cm,
.cm):");
    Serial.printf("Max Water Level Configured: %.2f cm\n", maxWaterLevel);
```

```
    offset = tankHeight - maxWaterLevel;
    Serial.printf("Offset Calculated: %.2f cm\n", offset);
}

/*
-------------------------------------------------------------------------------
* Function    : getTankOrWaterLevel
* Purpose     : Retrieves user input for tank height or max water level via
keypad data sent over I2C communication.
* Inputs      : prompt - Message to display on serial monitor for user input.
* Outputs     : float - Tank height or water level (cm).
* Authors     : John Mort,Jordan Reichhardt,Kevin Swanson
* Blame       : Used ChatGPT to structure code for calculating 5 digit user
input, and to combine that into a single value
* Version     : 1.2
* Date        : 2024-12-05

* -------------------------------------------------------------------------
*/
float getTankOrWaterLevel(const char *prompt) {
    Serial.println(prompt);
    int heightM = 0;    // store meter
    int heightCm = 0;  // store cm
    float heightDecimal = 0.0; // store .cm
    int inputDigits[5] = {-1, -1, -1, -1, -1}; // stores 5 keypad digits
    // Collect 5 digits from the keypad
    for (int i = 0; i < 5; i++) {
        while (keypadValue == -1) {
            delay(100); // delay for input
        }
        inputDigits[i] = keypadValue;
        Serial.printf("Received digit: %d\n", keypadValue);
        keypadValue = -1; // Reset after keypad value saved
    }

    // Store first two digits (meter part)
    heightM = inputDigits[0] * 10 + inputDigits[1];

    // Store second two digits (cm part)
    heightCm = inputDigits[2] * 10 + inputDigits[3];
```

```
    // Store last digit (.cm part)
    heightDecimal = inputDigits[4] / 10.0;

    // Print full value
    Serial.printf("Parsed Value: %dm %dcm %.1fmm\n", heightM, heightCm,
heightDecimal * 10);

    // Combine into a single float value in centimeters
    return (heightM * 100.0) + heightCm + heightDecimal; // Convert meters to
cm
}

/*
-------------------------------------------------------------------------------
* Function    : adjustSensorReading
* Purpose     : Calculate water level based on sensor reading and calculated
offset
* Inputs      : sensorReading - Ultrasonic sensor reading from sender (cm).
* Outputs     : float - Adjusted sensor reading(cm).
* Authors     : Jordan Reichhardt, John Mort, Kevin Swanson
* Version     : 1.2
* Date        : 2024-12-05
* -----------------------------------------------------------------------------
*/
float adjustSensorReading(float sensorReading) {
    float tankPerC = ((tankHeight - sensorReading)/maxWaterLevel) * 100;
    Serial.println("Water Tank %Full: " + String(tankPerC) + "%");
    return tankHeight - sensorReading;
}
```

Sender ESP32 unit (remote tank station):

```
/* USER CODE BEGIN Header */
/*******************************************************************************
**
* File: Ultrasonic_LoRa_Communication (sender)
*******************************************************************************
*
```

```
* @file : Ultrasonic_LoRa_Communication.cpp
* @brief : This file implements a LoRa communication system with ultrasonic
*          distance measurement.
* project : EE 329 Current Measurement Project
* authors : John Mort, Jordan Reichhardt, Kevin Swanson
* version : 1.2
* date : 2024-12-05
* compiler : Arduino IDE
* target : ESP32 LoRa V3
*
*******************************************************************************
*
* OPERATION:
*  - The system sends distance measurements from the ultrasonic sensor over
LoRa communication.
*  - Ultrasonic sensor readings are collected from sensor, formatted, and
transmitted to receiver (base station).
*******************************************************************************
*
* WIRING:
* GPIO19 - I2C SDA
* GPIO20 - I2C SCL
* GPIO47 - Ultrasonic Sensor Trigger
* GPIO48 - Ultrasonic Sensor Echo
*******************************************************************************
*
* REVISION HISTORY:
* 1.0 241128 KS/JR/JM Initial version for testing of sending and recieving
* 1.2 241204 KB/JR Updated for ultrasonic integration into the sender.
*******************************************************************************
*/

/*
-------------------------------------------------------------------------------
* Include Headers
* -----------------------------------------------------------------------------
*/
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include <Wire.h>
```

```
#include "HT_SSD1306Wire.h"

/*
---------------------------------------------------------------------------
* Macro Definitions (many are default for ESP32 LoRa communication)
* -------------------------------------------------------------------------
*/
// LoRa configuration
#define RF_FREQUENCY                                915000000 // Hz
#define TX_OUTPUT_POWER                             5        // dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
1: 250 kHz, 2: 500 kHz]
#define LORA_SPREADING_FACTOR                       7         // [SF7..SF12]
#define LORA_CODINGRATE                             1         // [1: 4/5, 2:
4/6, 3: 4/7, 4: 4/8]
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx
and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 30 // Define the payload
size here

// Ultrasonic sensor configuration
#define trigPin 47 // GPIO pin to trigger ultrasonic rangefinder
#define echoPin 48 // GPIO pin to receive echo from rangefinder

/*
---------------------------------------------------------------------------
* Global Variables
* -------------------------------------------------------------------------
*/
char txpacket[BUFFER_SIZE];
bool lora_idle = true;
static RadioEvents_t RadioEvents;

//Initializes function for the LoRa and ultrasonic pins
void setup();
```

```
//Initialize main loop
void loop();

//Initialize Lora transmission function
void OnTxDone(void);

//Initialize Lora timeout
void OnTxTimeout(void);

//Initialize ultrasonic calculations function
int getSDist();

/**********************************************************************
**
* Function    : setup
* Purpose     : Initializes the LoRa and ultrasonic sensor connection to ESP32
* Inputs      : None
* Outputs     : None
* Authors     : Kevin Swanson, Jordan Reichhardt, John Mort
* Blame       : Structured based on Arduino IDE example ESP32 Lora V3 sender
code
* Version     : 1.1
* Date        : 2024-12-05
*
**********************************************************************
*/
void setup() {
    Serial.begin(115200);
    Mcu.begin(HELTEC_BOARD, SLOW_CLK_TPYE);

    // Initialize LoRa
    RadioEvents.TxDone = OnTxDone;
    RadioEvents.TxTimeout = OnTxTimeout;
    Radio.Init(&RadioEvents);
    Radio.SetChannel(RF_FREQUENCY);
    Radio.SetTxConfig(MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                      LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                      LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                      true, 0, 0, LORA_IQ_INVERSION_ON, 3000);
```

```
    // Set ultrasonic sensor connections to ESP32
    pinMode(trigPin, OUTPUT); // Trigger pin as output
    pinMode(echoPin, INPUT);  // Echo pin as input
}

/*****************************************************************************
**
* Function    : loop
* Purpose     : Main application loop. Takes ultrasonic sensor data and sends
it over LoRa to the receiver (base unit).
* Inputs      : None
* Outputs     : None
* Authors     :  John Mort, Kevin Swanson, Jordan Reichhardt
* Version     : 1.1
* Date        : 2024-12-03
*
*****************************************************************************
*/
void loop() {
    if (lora_idle == true) {
        delay(1000); // Send data one time a second

        // Get the ultrasonic distance
        int distance = getSDist();

        // Format the distance as string so it can be sent over LoRa
        sprintf(txpacket, "%d", distance);

        // Send to serial monitor for receiver for viewing before transmission
over LoRa
        Serial.printf("\r\nSending packet: \"%s\", length: %d\r\n", txpacket,
strlen(txpacket));

        // Send via LoRa
        Radio.Send((uint8_t *)txpacket, strlen(txpacket));
        lora_idle = false;
    }

    Radio.IrqProcess();
}
```

```
/*******************************************************************************
**
* Function    : OnTxDone
* Purpose     : Callback for LoRa transmission.
* Inputs      : None
* Outputs     : None
* Authors     : Jordan Reichhardt, John Mort, Kevin Swanson
* Blame       : Adapted from Arduino IDE EPS32 LoRa V3 example sender code
* Version     : 1.1
* Date        : 2024-12-03
*
*******************************************************************************
*/
void OnTxDone(void) {
    Serial.println("TX done...");
    lora_idle = true;
}

/*******************************************************************************
**
* Function    : OnTxTimeout
* Purpose     : Callback for LoRa transmission timeout.
* Inputs      : None
* Outputs     : None
* Authors     : John Mort, Jordan Reichhardt, Kevin Swanson
* Blame       : Inspired by the Heltec ESP32-S3 “LoRa Sender” example
* Version     : 1.1
* Date        : 2024-12-03

*******************************************************************************
*/
void OnTxTimeout(void) {
    Radio.Sleep();
    Serial.println("TX Timeout...");
    lora_idle = true;
}

/*******************************************************************************
**
```

```
* Function     : getSDist
* Purpose      : Sends a pulse to the ultrasonic sensor and calculates the
distance to object that noise is bouncing off of.
* Inputs       : None
* Outputs      : int - Distance in centimeters.
 Authors      : Jordan Reichhardt, John Mort, Kevin Swanson
* Blame        :
* Version      : 1.1
* Date         : 2024-12-03

**************************************************************************
*/
int getSDist() {
    long duration = 0;  // Store pulse duration
    int distance = 0;   // Store calculated distance

    // Clear the trigPin (set low)
    digitalWrite(trigPin, LOW);
    delayMicroseconds(5);

    // Send 10-microsecond pulse to trigPin
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);a
    digitalWrite(trigPin, LOW);

    // Measure the duration of echo
    duration = pulseIn(echoPin, HIGH);

    // Calculate distance in cm (sound: 0.034 cm/us)
    distance = duration * 0.034 / 2;

    // Print the distance to Serial Monitor
    Serial.print("Distance = ");
    Serial.print(distance);
    Serial.println(" cm");

    delay(50); // Short delay for sensor stability
    return distance;
}
```