Household Water Monitoring System
ECE 445 Design Document - Fall 2024

Daniel Baker, Jack Walberer, Advait Renduchintala
Professor: Kejie Fang
TA: Pusong Li

Contents

1 Introduction

1.1 Problem

The problem we want to tackle involves the lack of transparency and control over water usage in apartment settings. During some months' utility bills, our cost for water consumption is extremely high, and we're not sure why this is the case. This problem drove our curiosity as this seemed like a general issue for many renters. Apartments typically charge tenants for utilities, which include water consumption, but tenants are often unaware of where their water usage is occurring and how much water each part of their apartment is using. This lack of visibility often leads to confusion when utility bills are unexpectedly high. Without detailed information, it becomes difficult to identify and manage specific areas of overuse, which can be particularly frustrating for tenants trying to reduce their water consumption.

Furthermore, water overuse is not just a financial concern, as it's also an environmental issue. Excessive water consumption depletes natural resources, increases strain on local water supplies, and contributes to environmental degradation. In a broader sense, water conservation is very important for sustainability and preserving ecosystems. Reducing unnecessary water use can help reduce the effects of water shortages and ensure that natural water resources are available for future generations.

Our goal is to design a device to address both of these issues by providing detailed insights into where water is being used within an apartment. By developing a device that incorporates each individual faucet, we can accurately measure and track water consumption in real-time, allowing renters to pinpoint specific sources of water usage. Hopefully, this data will allow users to make informed decisions about their water consumption habits, helping them to not only save money on utility bills but also contribute to environmental sustainability through responsible water usage.

1.2 Solution

Our product is a smart device designed to be easily attached to the end of faucets around the house, allowing users to monitor water usage at each faucet in real-time. The device uses ultrasonic sensors to track the amount of water dispensed from each faucet. As soon as the faucet is turned on, the device begins measuring the flow of water, and this data is displayed on the attached LCD screen. The screen provides count of the total volume of water that has been used since the faucet was activated. This enables users to have immediate feedback on their water consumption every time they use the sink, which promotes more mindful water usage.

Our system is also scalable, allowing multiple devices to be installed on various faucets throughout our apartment. All these devices are connected to a centralized dashboard, which is accessible via a website. The dashboard provides a detailed breakdown of water consumption for each faucet, including both specific and aggregated data over a given time period (such as a month). For example, users can see how much water was used by the kitchen faucet compared to the bathroom sink, allowing them to identify high-usage areas. The dashboard will also display facts like the kitchen sink accounted for 50% of total household water usage, with 100 gallons used in a month.

1.3 Visual Aid

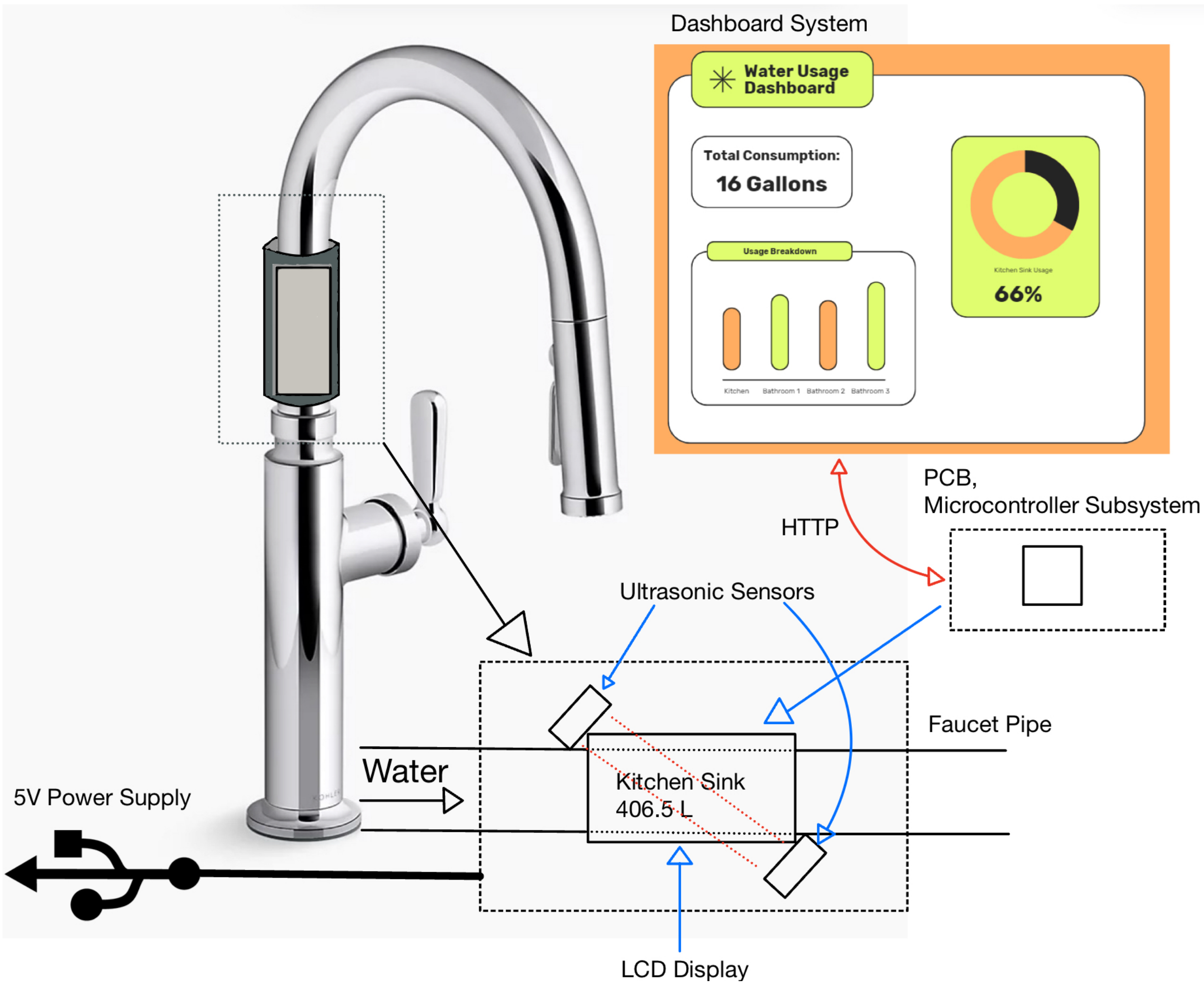

Figure 1: Visual Aid of the Household Water Monitoring System

1.4 High-Level Requirements

1. Ultrasonic Sensing and Microcontroller Subsystems can accurately measure and calculate flow rate and total water usage, determine flow with less than a 10% difference of the real amount of water dispensed. We will measure the real amount of water dispensed using a measurement cup.

2. The Front-End, API, Database, and Microcontroller Subsytems must provide conistent and available water usage data. The webpage must load (fetch water usage data from database) and display the dashboard within 1 second for 90% of requests assuming a standard internet connection (10 Mbps). The database and dashboard data must be consistent with the microcontroller data displayed on the LCD screen.

3. The Household Water Monitoring System must be resilient to network interruptions and power fluctuations, ensuring continued operation and data integrity across all subsystems. If network connectivity is lost, the Microcontroller Subsystem will continue collecting water usage data and store it in the Flash Memory. Once reconnected, the Microcontroller Subsystem must automatically synchronize the stored data with the Database Subsystem, no matter the time of day. This differs from the automatic daily synchronization of water usage data from the Monitoring System. Additionally, the Front-End Dashboard should display an alert indicating any data gaps caused by outages, ensuring users are aware of any temporary data inconsistency. This will be verified through stress testing, simulating both network and power interruptions.

2 Design

2.1 Physical Design

The physical design of the household water monitoring system focuses on the monitoring device, which is designed for easy attachment to household pipes. The device will be secured using two semicircle clamps positioned on either side of the pipe and held together with zip ties, as shown in figure 2.

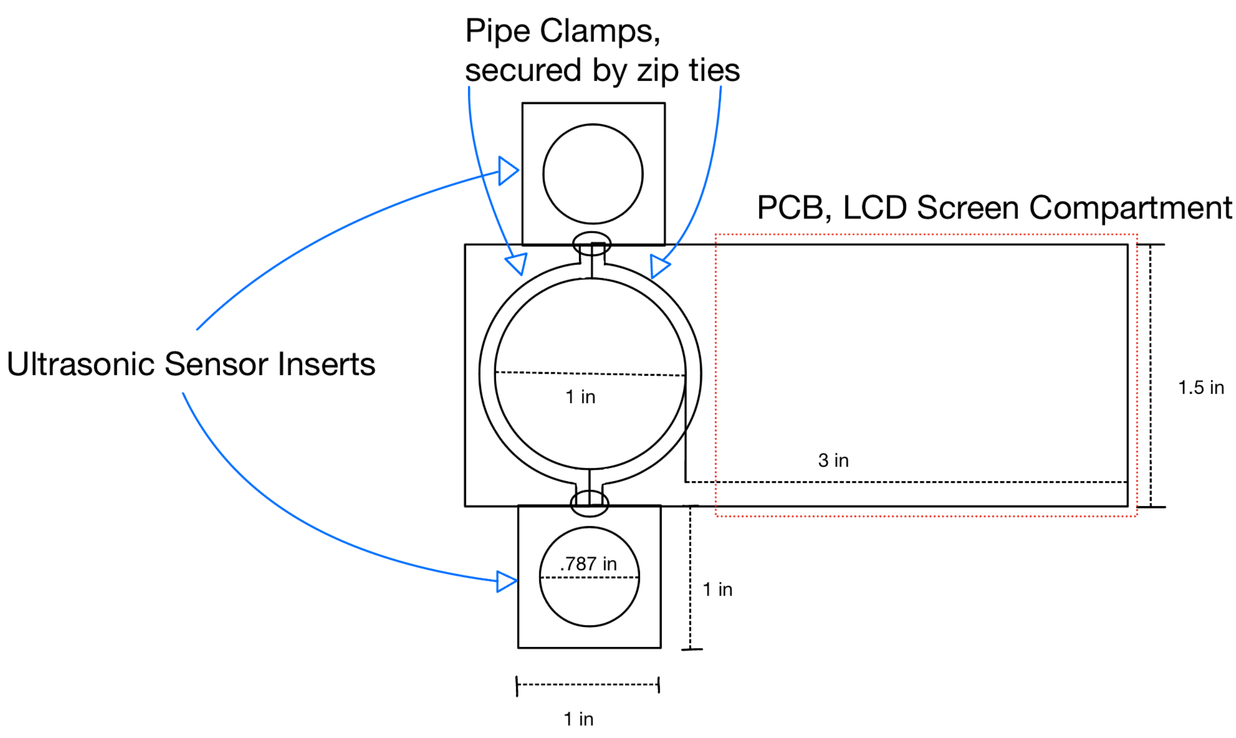

Figure 2: Top view of the Household Water Monitoring System

To optimize the accuracy of ultrasonic sensor measurements, the design is asymmetrical. Three sides of the device will be positioned close to the pipe to maintain a tight fit and avoid unnecessary air gaps between the sensors and the pipe. The fourth side of the device will protrude outward, making space for the PCB (Printed Circuit Board) and the LCD screen, as depicted in figures 2 and 3.

As shown in figure 3, the ultrasonic sensors will be mounted at approximately a 45-degree angle relative to the pipe. Inserts will secure the disk-shaped sensors in place, ensuring stability and accuracy in the sensor measurements during water flow detection. This angled configuration helps in improving the precision of time-of-flight measurements between the sensors.

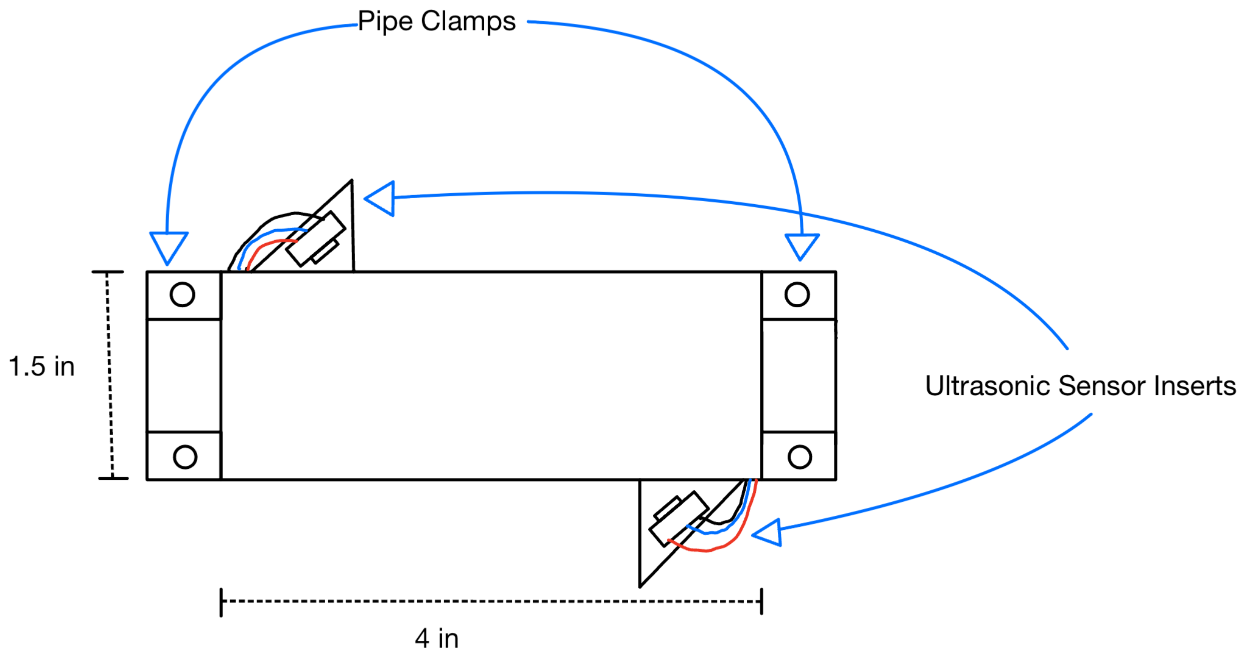

Figure 3: Side view of the Household Water Monitoring System

2.2 Block Diagram

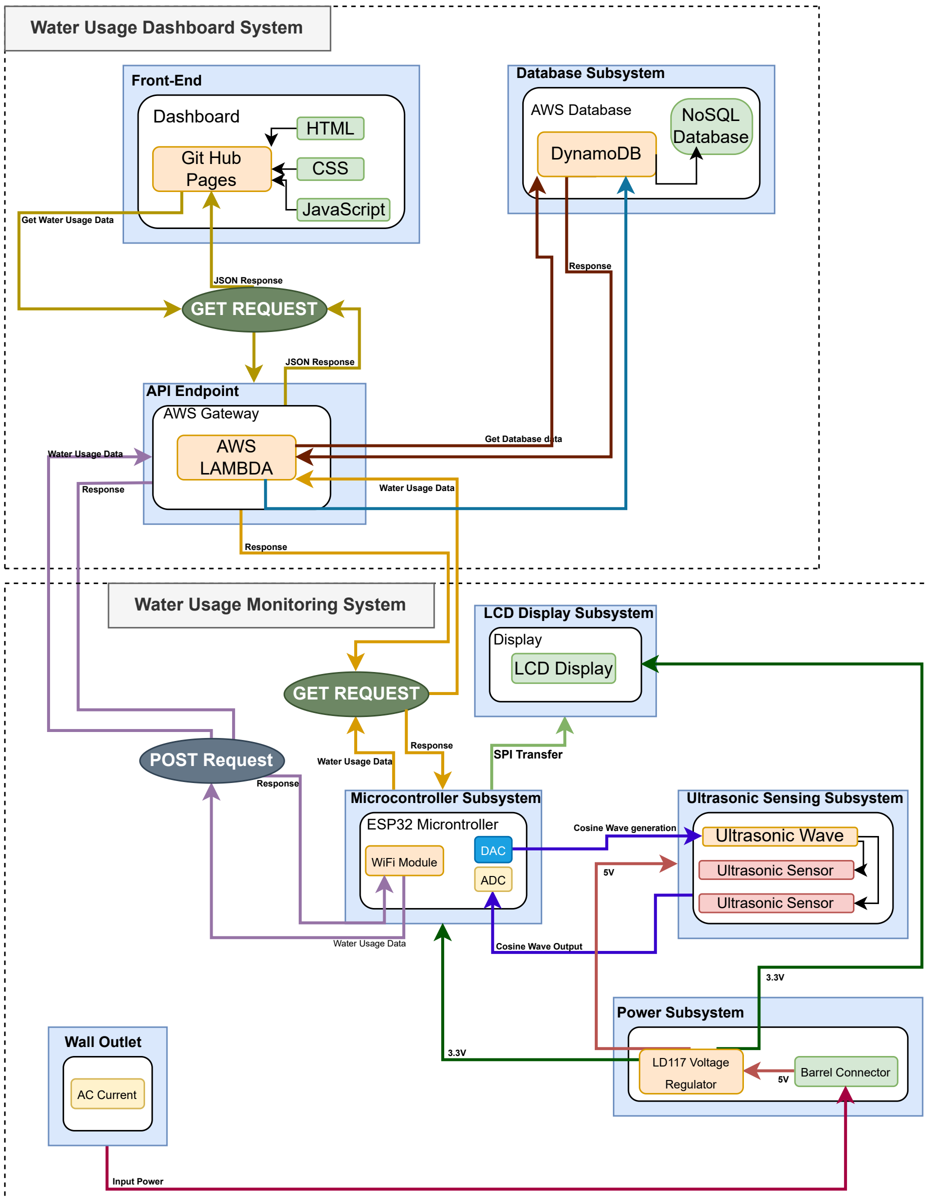

Figure 4: Block Diagram of the Household Water Monitoring System

2.3 Subsystem Block Descriptions, Requirements, and Verifications

2.3.1 Dashboard System - Front-End

This subsystem consists of a web-based dashboard hosted on GitHub Pages, designed to display water usage data collected from various household monitoring devices. The dashboard provides a user-friendly interface that allows users to view detailed statistics of their water consumption over time.

The dashboard will populate upon refresh of the page, where it will request water usage data from backend services. The dashboard will initiate an HTTP GET request, where AWS Lambda will serve as the intermediary service to fulfill the request.

After AWS Lambda gathers the water usage monitoring data from the database, it will respond with a POST Request to the web-based dashboard. The web page will update its table to reflect this new information of water usage per device. [9][10]

Table 1: Front-End Subsystem - Requirements and Verification.

Requirement	Verification
• The webpage must load and display the dashboard within 1 second for 90% of requests on a standard internet connection (10 Mbps)	• Open the dashboard and initiate a data fetch from the API. • Use browser developer tools (e.g., Chrome DevTools) to measure page load time and verify it is 1000ms or below • Repeat test across various devices such as laptops, desktops, and mobile phones.
• When the user loads or initiates a refresh of the page, the Front-End Dashboard must perform an HTTP GET request to a URL given by AWS Lambda to indicate its need for updated water usage data.	• Open the dashboard and initiate a data fetch using HTTP GET Request to AWS Lambda URL. • Cross-check the GET request on AWS Lambda's developer tools and verify the endpoint was hit, and the GET request is of the correct format.
• The dashboard must update daily, given that AWS Lambda is forwarding POST Requests of water usage data that was given from the Database Subsystem.	• Verify API Endpoint and Database Subsystems are operating correctly through our requirement and verification tables for those respective subsystems. • Check a POST request is hitting AWS Lambda's endpoint for the dashboard. • Verify the JSON body within the POST Request is correctly transferred to the web page's HTML document element to ensure consistency. • Verify the dashboard displays the water usage that is consistent with the HTML document element for the water usage data.
• The frontend must render properly on mobile devices with a screen width of at least 320px.	• Use browser developer tools to simulate mobile screens and verify proper display on different resolutions. • Test on actual mobile devices (e.g., smartphones, tablets) with varying screen sizes. • Verify the JSON body within the POST Request is correctly transferred to the web page's HTML document element to ensure consistency. • Ensure that all content is accessible and the dashboard layout adapts to smaller screens without horizontal scrolling.

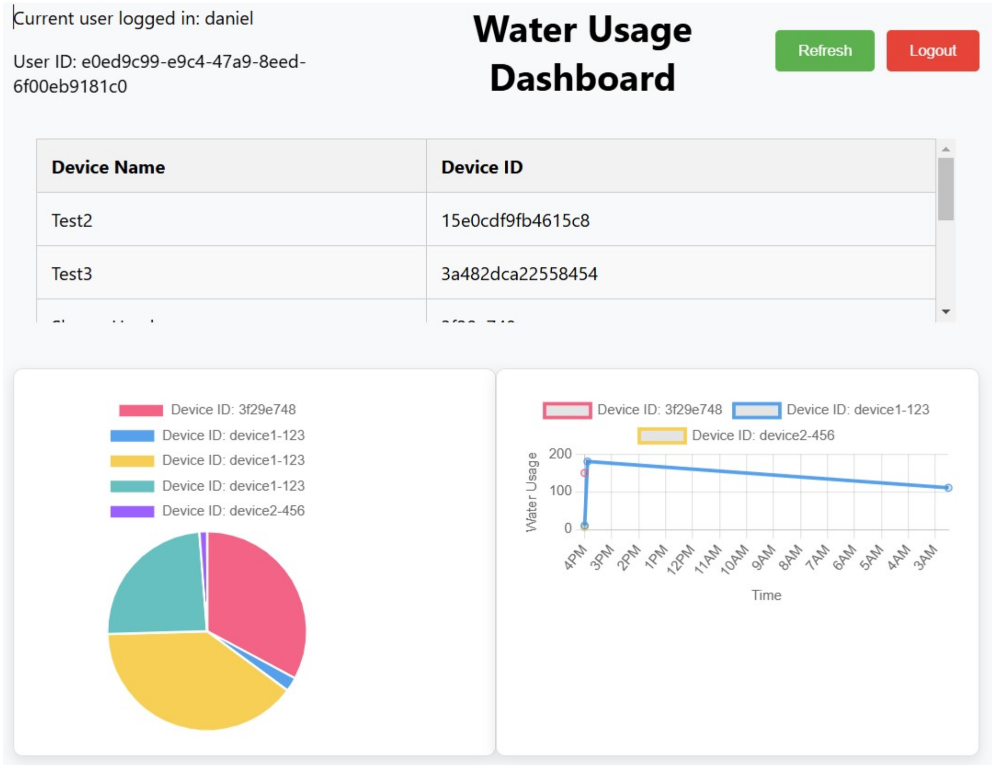

Figure 5: User view of the front end dashboard

2.3.2 Dashboard System - API Endpoint

The API Endpoint Subsystem serves as the core communication hub between the frontend dashboard, the microcontroller (ESP32), and the database (DynamoDB). The core component of this subsystem is AWS Lambda, which processes incoming requests and interacts with the database, ensuring efficient data transfer between the components of the water monitoring system.

To begin with, when the system records water usage data, the ESP32 microcontroller calculates the water flow based on the ultrasonic sensors' input (this is done using C or C++). The microcontroller then sends the data to the API Endpoint Subsystem using a POST request. This request is directed to an API Gateway to ensure that the correct Lambda function is triggered. The API Gateway is configured to handle RESTful HTTP requests (GET and POST) from both the ESP32 and the frontend. Once the POST request is received, the AWS Lambda function is invoked. Lambda serves as a serverless backend, meaning it runs code only when an API request is made, ensuring cost-efficiency and scalability. The POST request includes data such as the device ID and water usage. The Lambda function processes this data and stores it in AWS DynamoDB, a NoSQL database optimized for high-speed data access and scalability. The data is stored in a structured format, typically with a primary key, we will be using DeviceID, and a timestamp for tracking water usage over time. Similarly, when the frontend dashboard or the ESP32 requests usage data via a GET request, the API Gateway routes the request to another Lambda function, which queries DynamoDB for the relevant water usage data. Lambda retrieves the data, formats it as JSON, and returns it through the API Gateway to the requesting client (either the frontend or the microcontroller), which then displays the information to the user. This system architecture ensures that all communication between the components is handled efficiently, with AWS Lambda acting as the middleman for processing and relaying data, ensuring a seamless flow of information between the monitoring devices, database, and dashboard. [5][7][9]

Table 2: API Endpoint Subsystem - Requirements and Verification.

Requirement	Verification
• The API Endpoint Subsystem must handle GET & POST requests from the ESP32 microcontroller within a maximum latency of 200 milliseconds under normal operation.	• Set up a GET & POST request from the ESP32 to AWS Lambda using a known payload. • Use a network traffic monitoring tool (e.g., Wireshark) or measure timestamps in the code to calculate the time from the request being sent to Lambda's response. • Record the latency and ensure it is less than or equal to 200-300 milliseconds for 95% of requests.
• The Lambda function must process and store each POST request payload in DynamoDB with a success rate of 99% or higher.	• Send 100 POST requests from the ESP32 with a known payload to AWS Lambda. • Verify each entry in DynamoDB to ensure all data was correctly stored. • Ensure at least 99% of the requests are successfully stored in DynamoDB.
• The API Endpoint Subsystem must return data to the ESP32 or frontend in JSON format with correct data structure (DeviceID, Timestamp, WaterUsed).	• Send a GET request to Lambda. • Capture the response and check the data format. • Ensure that the response contains the correct JSON structure: {"DeviceID": "ID", "Timestamp": "timestamp", "WaterUsed": "value"}.
• The API Endpoint Subsystem must return a status code of 200 for all successful requests and appropriate error codes (400, 500) for failed requests.	• Send a mix of valid and invalid GET and POST requests to AWS Lambda. • Monitor the returned status codes. • Ensure that successful requests return status code 200 and invalid requests return appropriate error codes such as 400 or 500.

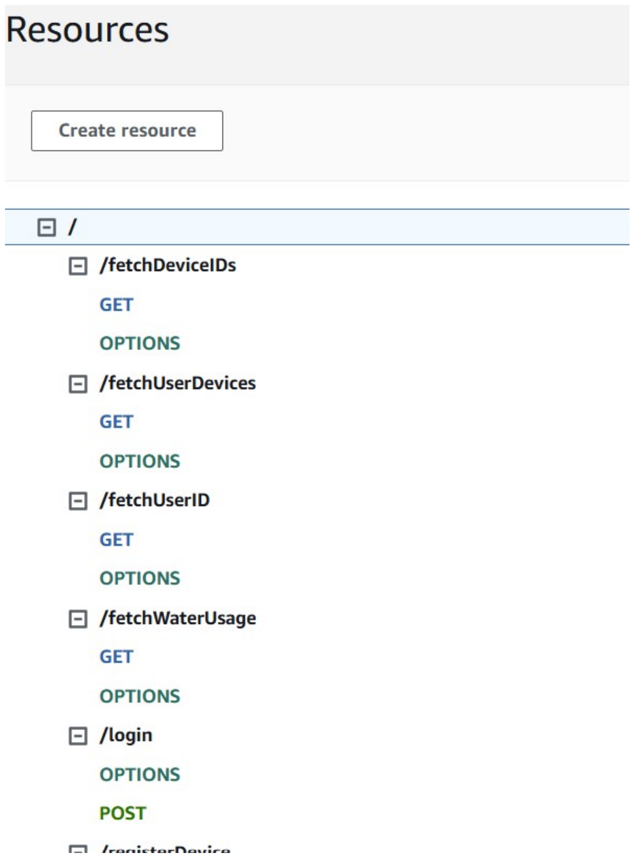

Figure 6: List of some of the API endpoints we are using through AWS API Gateway

2.3.3 Dashboard System - Database

The Database Subsystem for the water monitoring system is responsible for storing and retrieving water usage data from our household. The primary technology used for this subsystem is AWS DynamoDB, a highly scalable, low-latency NoSQL database. DynamoDB is ideal for handling the frequent write and read operations that are generated by the water monitoring devices and the dashboard, making it suitable for this application.

Each time the ESP32 microcontroller calculates water usage data, it sends a POST request to the API Endpoint Subsystem, which processes the data and stores it in DynamoDB. The database is designed with a primary key structure, where each entry has a unique DeviceID and a Timestamp as the sort key. This allows efficient tracking of water usage over time for each device, making it easy to retrieve historical usage data for a specific faucet. The database also stores additional data like the user-defined names of the devices (e.g., "Kitchen Faucet", "Bathroom Sink"), allowing for better tracking in the user dashboard. Data retrieval is performed using GET requests from both the ESP32 (to display usage on the LCD) and the frontend dashboard. DynamoDB will query to retrieve data based on specific time ranges or devices, allowing the frontend to display both individual and aggregated water usage. The system is designed to handle large volumes of data with minimal latency, ensuring the dashboard provides near-real-time updates of water consumption. [3][6][8][10]

Table 3: Database Subsystem - Requirements and Verification.

Requirement	Verification
• The database must be able to handle at least 1000 writes per hour without data loss or corruption.	• Simulate 1000 POST requests within an hour using Postman. • Verify that all records are correctly stored in DynamoDB by querying for the total count. • Ensure that no data is lost or corrupted after the test.
• The database must support querying based on DeviceID and Timestamp with correct filtering.	• Store water usage data with different DeviceID values and timestamps. • Send a GET request that queries DynamoDB for data from a specific device and time range. • Verify that the query returns the correct set of data, filtered by DeviceID and time range.
• The database must ensure data availability with an uptime of 99.9% or higher.	• Use AWS CloudWatch to observe uptime over a period of one month. • Verify that the database remains available 99.9% of the time. • Review logs for any downtime events and confirm they are within acceptable limits.

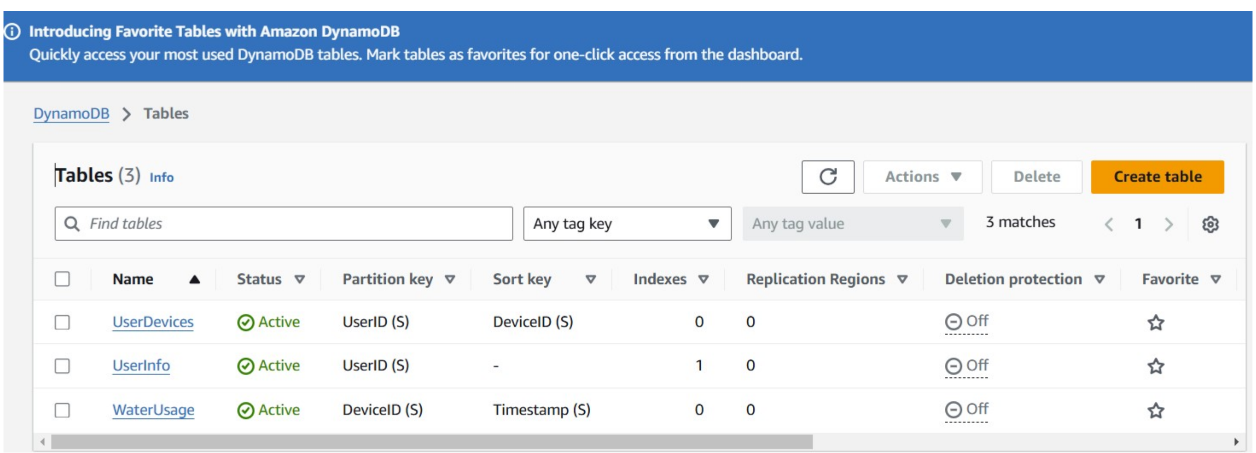

Figure 7: List of the three tables we are using in AWS DynamoDB

2.3.4 Monitoring System - Microcontroller Subsystem

The microcontroller subsystem in this project utilizes the ESP32, a 3.3V system that serves as the central control unit for data collection, signal processing, and communication within the monitoring subsystem of the household water usage monitoring system.

The ESP32's built-in 2.4 GHz Wi-Fi capability will be used to send HTTP POST requests to AWS Lambda's endpoint. This POST Request will have a JSON body containing update water usage data for this specific monitoring device.

The ESP32's built-in 4 MB flash storage will be used to store the water usage data in order to always display the current usage to the device's LCD screen. This storage will also be useful in maintaining data consistency with the DynamoDB database in the case of network failure.

The ESP32's Digital-to-Analog Converter (DAC) will generate sine wave signals to drive the ultrasonic sensing system. Since the sensors are both transmitters and receivers, the ESP32's Analog-to-Digital Converter (ADC) will listen for signal reception. The ESP32 will do onboard signal processing to the raw ADC data in order to filter out noise and accurately calculate transmission time between sensors through the water pipe. This will involve techniques such as threshold detection and Fast Fourier Transform.

The transmission time found through signal processing will be directly proportional to the flow rate of water. We can use it to find the flow rate using the equation below:

$$v = [c^2 * \Delta T]/[2 * L * cos(\theta)].$$

Taking the integral over time of this flow rate will show us the total water usage. [4][5]

Table 4: Microcontroller Subsystem - Requirements and Verification.

Requirement	Verification
• The ESP32 must generate a sine wave using the DAC to drive the ultrasonic sensing system.	• Obtain an ESP32 breakout board and program the DAC to generate a sine wave. • Measure the DAC output using an oscilloscope to ensure it generates a sine wave with the correct frequency.
• The ESP32 must process the signals from the ultrasonic sensors and calculate transmission time between the sensors.	• Using a signal generator, mock an ultrasonic sensor receive signal • Test the ADC input to verify the reception of the signal from the generator • Confirm the signal processing by checking the processed data for noise reduction and accurate threshold detection. • Use a logic analyzer to verify the ESP32 calculates the correct transmission time by comparing it to expected values from controlled test cases.
• The ESP32 must store water usage data in its 4 MB flash memory and ensure continuous data availability to the 2.4GHz wireless system and LCD screen.	• Simulate water usage data gathering by hardcoding water usage into the flash memory. • Confirm that the stored data can be correctly retrieved and displayed on the LCD screen. • Confirm that the stored data can be correctly transformed to a JSON body within a POST Request to be sent by the wireless system.
• The ESP32 must send HTTP POST requests to AWS Lambda with the water usage data.	• Simulate a POST Request with a JSON body of hardcoded water usage data • Confirm on AWS Lambda that the endpoint is being hit with the same JSON body • Simulate network failure (not receiving OK response, or receiving 400/500 response), and verify ESP32 resents POST Request

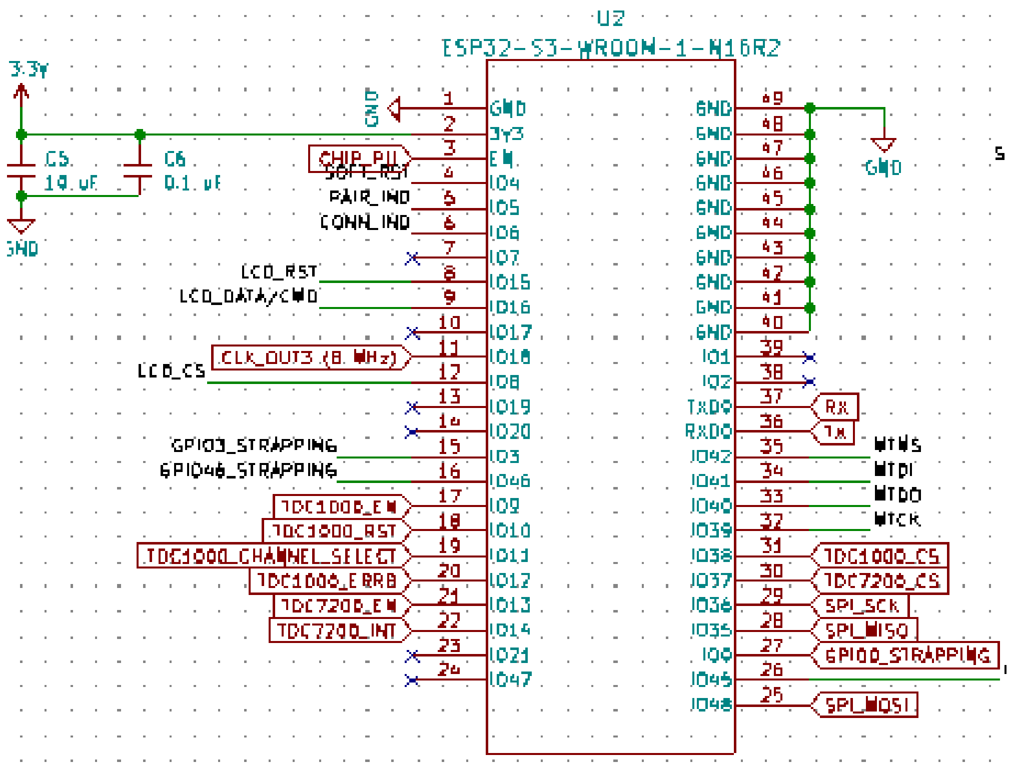

Figure 8: Schematic of the ESP32-S3-WROOM-1 that we will be using, along with its connections to other subsystems.

2.3.5 Monitoring System - Power Subsystem

The Power Subsystem is designed to provide reliable and efficient power distribution to all components within the a Monitoring Device of household water monitoring system.

The Power Subsystem receives a 5V input from a USB power source to be used by the LCD screen and MCP6001 Op-Amp. It will also utilize the LD1117-3.3V linear voltage regulator to additonally provide a 3.3V supply for the ESP32 Microcontroller.

The Power Subsystem will include a jumper-selectable voltage for signal input/output for the ultrasonic sensing system. While the datasheet uses 3.3V for its example signal, we know that a 5V signal amplified by the MCP6001 Op-Amp could be more effective when deriving transmission time, ultimately leading to a more accurate flow measurement. Thus, we will incorporate jumpers on each of the signal paths to the ultrasonic sensors, shorting the one that we select. [14]

Table 5: Power Subsystem - Requirements and Verification.

Requirement	Verification
• The Power Subsystem must provide stable 3.3V and 5V power supplies.	• Measure the 5V input voltage and the output voltage of the LD1117-3.3V linear regulator using a multimeter to verify steady 5V and 3.3V outputs.
• The jumper-selectable signals for the ultrasonic sensors must each provide reliable 3.3V and 5V input/output signals.	g • For the 3.3V signal path, measure the output of the ADC of the ESP32 using an oscilloscope. • For the 5V signal path, measure the output of the MCP6001 Op-Amp using an oscilloscope. • Verify the switching between signal path configurations through shorting jumpers by measuring the signal at the input of the ultrasonic sensors using an oscilloscope.

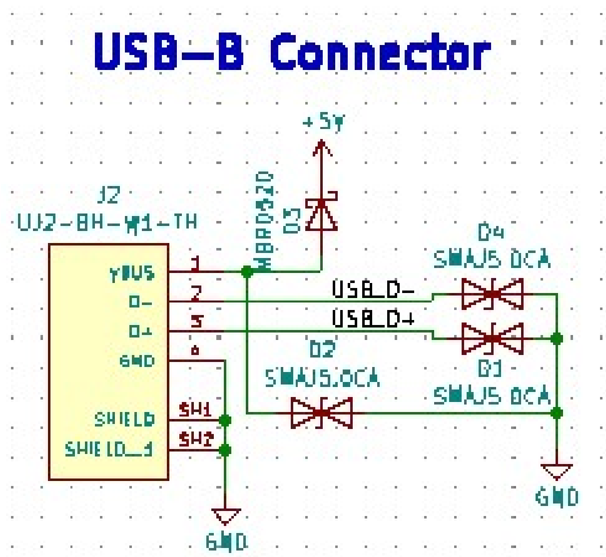

Figure 9: Schematic of the USB-B connector, power connection for the USB to Linear Voltage Regulator

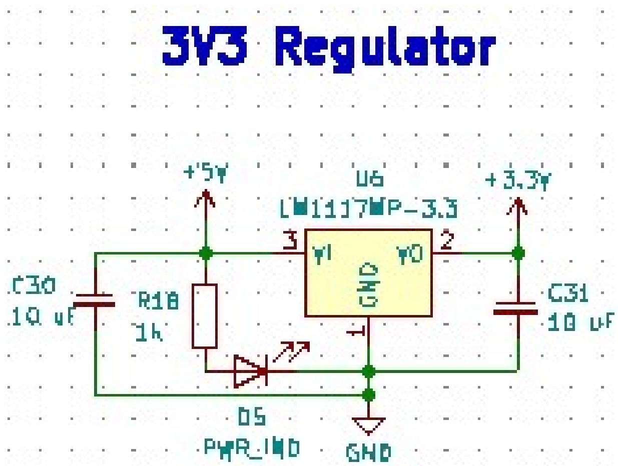

Figure 10: Schematic of the LD1117 3.3V Linear Regulator, showing it down convert the 5V source to a 3.3V one.

2.3.6 Monitoring System - LCD Display Subsystem

The LCD Subsystem in the water monitoring device is responsible for displaying water usage data. This subsystem uses a simple LCD screen that allows users to see how much water has been used since the faucet was activated. It reads data through the SPI (Serial Peripheral Interface) protocol, which is used for communication between the ESP32 microcontroller and the LCD.

The SPI protocol is a fast, synchronous serial communication method that allows reliable data transfer between the microcontroller and the LCD screen. It operates in a master-slave configuration, where the ESP32 acts as the master and the LCD as the slave. The SPI bus uses four lines: MOSI (Master Out Slave In), MISO (Master In Slave Out), SCLK (Serial Clock), and CS (Chip Select). The microcontroller sends water usage data through the MOSI line, synchronized by the SCLK, and the data is received and displayed on the LCD screen in real time.

Since the LCD only needs to display numerical data, the simplicity of SPI ensures that our LCD Subsystem can operate smoothly and display accurate water usage readings. [11]

Table 6: LCD Display Subsystem - Requirements and Verification.

Requirement	Verification
• The LCD must correctly display water usage values in a clear, readable format with no data corruption for at least 99% of updates.	• Send a series of known values (100 test cases) to the LCD using SPI. • Manually verify the accuracy of the displayed values on the LCD. • Ensure that at least 99% of the values are displayed correctly without errors.
• The LCD must interface correctly with the ESP32 over a 4-wire SPI configuration (MOSI, MISO, SCLK, CS).	• Connect the ESP32 to the LCD using the 4-wire SPI configuration. •Send data from the ESP32 to the LCD and verify that the data is correctly displayed. • Ensure that the communication works without issues through the 4-wire interface.

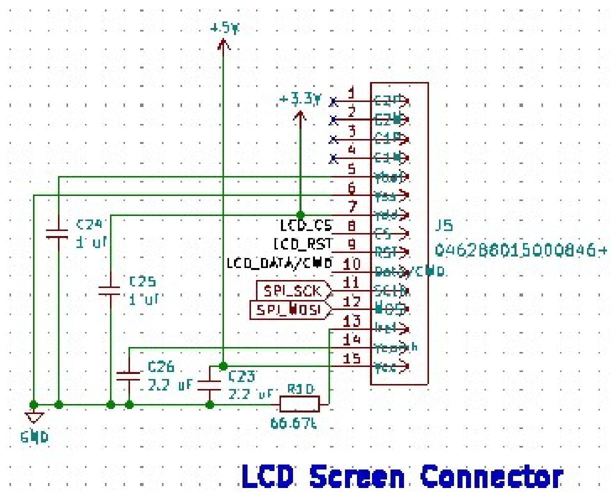

Figure 11: Schematic of the LCD screen's connections, showing how it will communicate with the Microcontroller Subsystem through SPI.

2.3.7 Monitoring System - Ultrasonic Sensing Subsystem

There will be two ultrasonic sensors, each of them will be able to send and receive signals. The sensors will be placed on opposite sides of the faucet at an angle of 45 degrees to the faucet, while facing each other.

The ESP 32 will send a signal from its DAC, a sine function with frequency 1 MHZ and magnitude 3.3V, to the sensors. This signal will then be amplified by an op amp. The ultrasonic sensors can take input voltages up to 50Vpp so the closer the voltage is to that the more accurate the measurements can be. The input voltage will not be that high, but the op amp should get it above 5V. After receiving signals from the DAC, the sensor the received it will send the wave through the faucet and received by the ADC pin. Then using a different DAC, but the same signal, and a different ADC pin, another trial will be run, but the signal will be sent in the opposite direction. In both these trials, the time that the waves is travelling for will be measured. These measurements will be used to calculate the time difference in the upstream travel time and downstream travel time. Using the time difference we can calculate the velocity of the flow, and then the flow rate.

Components of this subsystem include just the ultrasonic sensors and whatever casing they will held in on the faucet. We do not yet know how the sensors will be connected to the board, but once we have the sensors we will know if we need header pins or if we can solder directly on the PCB. [4][12][13]

Table 7: Ultrasonic Sensing Subsystem - Requirements and Verification.

Requirement	Verification
• Receive a sinusoidal signal of the proper magnutide from the DAC of the ESP 32	• Using the breakout ESP 32, we will send a signal to the sensors while they are not connected to the facuet. • Measure the sensor output using an oscilloscope to ensure it generates a sine wave with the and expected travel time based on the speed of sound in air
• Send a signal from one ultrasonic sensor to the other through a medium	• Using a signal generator or ESP32 send a voltage and measure the reciever voltage on an oscilloscope • Place a the sensors on either side of the sink, with no water flow to make sure the waves can pass through the material
• Must have a mechanism that holds the devices at an angle of 45 degress to the faucet, and confirm they are lined up	• Use a protractor to confirm the angle is correct • Send a test signal, with no water flow, through the pipe to make sure they are lined up
• The Ultrasonic sensors will be able to send and recieve signals in both directs, and send the output to the ESP32 for time calculations to happen	• Make sure signals can be sent in both directions by connecting one input pin to DAC1 and the other sensors to DAC2 and then one output pin to ACD1 and one to ADC2 • Send a signal from DAC1, recieve at ADC1, then wait a second and send one from DAC2 to AD2, make sure each of the ADC pins recieved a signal using the oscilloscope

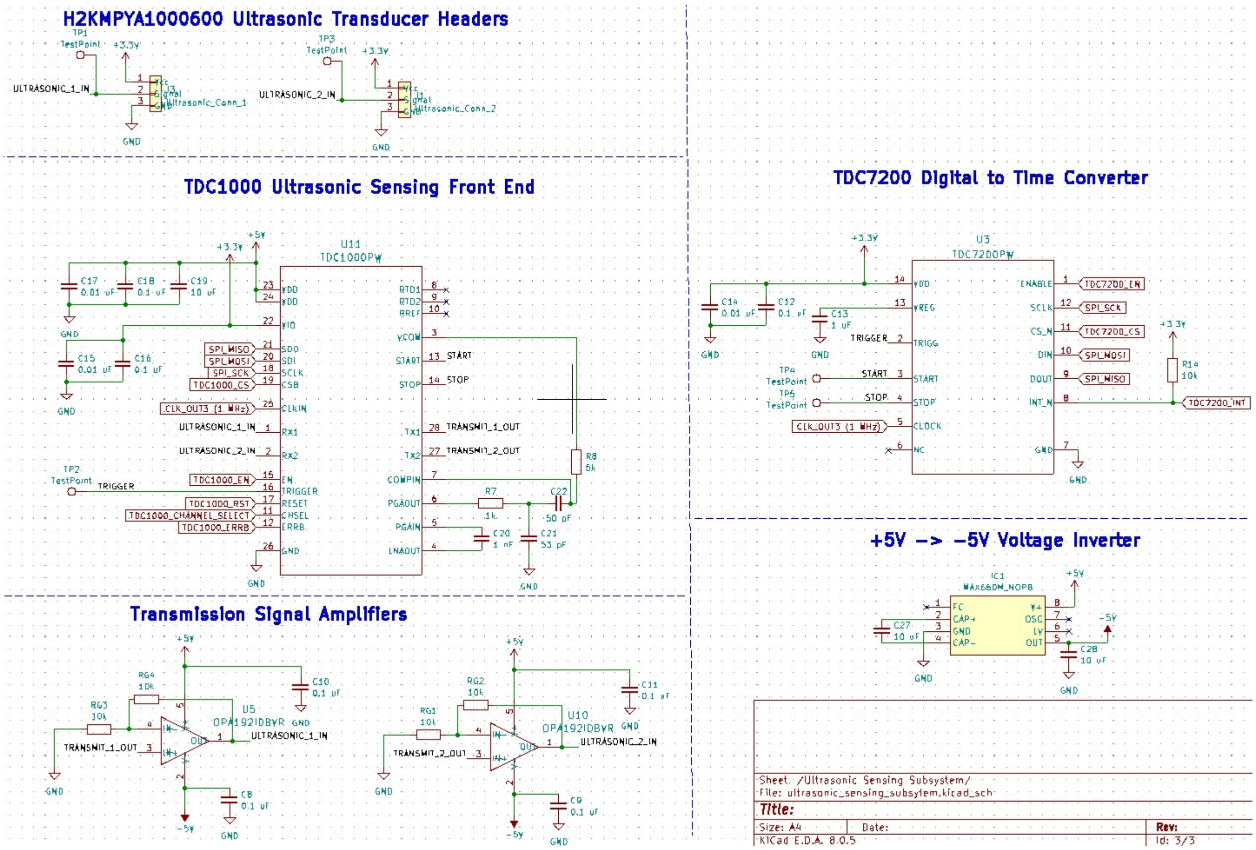

Figure 12: Schematic of the Ultrasonic Subsystem, showing the ultrasonic sensor headers, the TDC1000 Ultrasonic Front end and its SPI communication with the Time to Digital Converter TDC7200. The OPA192 Op Amps and its required -5V source from the MAX660 are also shown, used for amplifying the ultrasonic signal.

2.4 Tolerance Analysis

Our project needs to measure amount of water coming out of a faucet. The main reason this measurement would be off is timing. Two signals are sent across the ultrasonic sensors from the DACs to the ADCs. The timing of these signals will give us the information we need to calculate flow rate of the water. When a signal is sent from the DAC and received at the ADC there will be some internal delay. The ESP32 runs at 80MHz. Using some calculations below, here is the maximum amount of clock cycles that the timer can be off by to ensure a measurement that is within 20 percent of the true value.

The approximate time it takes for a sine wave to cross the faucet is found by taking the speed of sound through various mediums times the distance of the medium. The faucet this product will be build for has .25in aluminum as a pipe encircling a .5in diameter spout for water. Since the sensors are at 45 degrees, the distance of travel through the faucet is root 2 times these values times the velocity of sound in the medium. For the calculations the distances were changed meters. 6320 m/s is the speed of sound in aluminum, and 1482 m/s is the speed of sound in water. We need the flow rate of water up stream. Flow velocity can be found by

$$Q = vA$$

where Q is flow rate, v is velocity and A is cross sectional area. Based on the average faucet producing 2 gallons per minute (GpM) and the cross sectional area of the water spout to be pi*(.25in*.25in). In SI units this is-

$$1.26 * 10^{-4} m^3/s = v * 1.266 * 10^{-4} m^2$$

Then, v = 0.995 m/s. So the velocity of the signal in water up stream is 1482 - .995 or about 1481 m/s.

$$(\sqrt{2} * .00635m * (1/6320m/s)) + (\sqrt{2} * .0127m * (1/1481m/s)) = t1$$

To get the time down stream we can just add the v flow to the velocity in water. So we get 1483 m/s.

$$(\sqrt{2} * .00635m * (1/6320m/s)) + (\sqrt{2} * .0127m * (1/1483m/s)) = t2$$

Take the difference of these terms.

$$t1 = 1.35482 * 10^{-5}$$

$$t2 = 1.35319 * 10^{-5}$$

The time difference is on the scale of 10^{-8}, and that is the same time scale of our clock cycle, which is 80MHz, so we will be adding an external device to measure time differences. Because right now with a time difference of about 2-3 clock cycles, the error would be very high. Because in this case, the time difference is $1.6 * 10^-8$ seconds, and the clock has a rising edge every $1.25 * 10^{-8}$ seconds. So

$$(1.6 - 1.25)/((1.6 + 1.25) * .5) = 25\%$$

This error is simply too high, so we will add an external device that can have a faster clock cycle allowing us to gather accurate data.

Without additional timing chips, this tolerance is not acceptable. However, with the addition of the TDC1000 and TDC7200, we can measure time differences down to 55 pico seconds. Based on the approximate calculations above, the time difference in upstream vs downstream travel time is $2.6x10^{-8}$. With just the ESP 32 this is only about a 2 clock cycle difference so error is too high. $2.6x10^{-8}/55x10^{-12}$ is more than 450 clock cycles different. If we assume 5 clock cycles of delay, this will still result in error of less than 5% based in timing.

3 Cost and Schedule

3.1 Cost Analysis

Item (Hyperlinked)	Manufacturer	Qty	Price	Total
CP2102 USB to UART Bridge	Silicon Labs	1	$4.44	$4.44
ESP32-S3-WROOM-1-N16	Espressif Systems	1	$3.48	$3.48
TDC7200PWR Time to Digital Converter	Texas Instruments	1	$2.58	$2.58
TDC1000PW Ultrasonic Front End	Texas Instruments	1	$5.41	$5.41
OPA192IDBVR Op Amp	Texas Instruments	2	$2.90	$5.80
OLED Display	Crystal Fontz	1	$6.31	$6.31
SS8050-G BJT	Comchip Technology	2	$0.17	$0.34
H2KMPYA1000600 Ultrasonic Sensor	Unictron Technologies	2	$15.63	$31.26
MAX660 Voltage Inverter	Texas Intruments	1	$1.72	$1.72
Various 0805 Capacitors	KYOCERA AVX	30	$0.54	$16.20
Various 0805 Resistors	Bourns	20	$0.10	$2.00
LD1117 3.3V Linear Regulator	STMicroelectronics	1	$0.79	$0.79
USB-B Connector	Same Sky	1	$0.59	$0.59
AWS DynamoDB	Amazon Web Services	1	$0.59	$0.59
AWS Lambda	Amazon Web Services	1	$0.59	$0.59
AWS API Gateway	Amazon Web Services	1	$0.59	$0.59

Item (Hyperlink)	Company	Cost per User per Month (assuming 264,000 requests/month estimate)
AWS DynamoDB	Amazon Web Services	$0.001
AWS Lambda	Amazon Web Services	$0.00
AWS API Gateway	Amazon Web Services	$0.92

For Labor, we expect the costs to be ($45/hr) * (2.5) * (60 hours) = $6,750 per team member. The total costs in our bill of materials is $79.43 + Sales Tax (10%) + Shipping (5%), so our total cost for parts will be $91.23 with tax and shipping. After incorporating the costs of labor as well for 3 teammates, we believe the total cost will be $20,341.23.

3.2 Schedule

Week	Task	Person
Week of 10/7	Order parts for breadboarding	Everyone
	Set up AWS basics	Advait
	Begin PCB design for design + PCB review	Daniel + Jack
	Computation for wave that we want to transmit through transducers	Jack
Week of 10/14	Construct Front End	Advait + Daniel
	Breadboard ESP32 and begin integration with WIFI	Jack + Advait
	Submit PCB first round order with optional components	Daniel + Jack
Week of 10/21	Breadboard ESP32 and begin integration with WIFI	Advait
	CAD mounts and device encasing	Daniel + Jack
	PCB revisions & Pass Audit !!	Daniel
	3d print mounts draft	Advait
Week of 10/28	Solder PCB with ordered parts	Daniel
	Test transducers working with the device mounts	Jack + Advait
	Integrate Databse and API protocols with PCB	Advait + Daniel
Week of 11/4	Modify device encasing/mounts	Advait
	Revise finalized design	Everyone
	Test accuracy of water flow detection	Jack
Week of 11/11	Ensure all test cases for software work (including edge)	Daniel + Advait
	Modify computations to align better accuracy	Jack
Week of 11/18	Tweaking minor bugs (if any) and practicing presentation	All of us
Week of 12/2	Final Presentation	All of Us

Figure 13: Schedule

4 Ethics and Safety

Our project raises important ethical and safety concerns that we have addressed comprehensively to ensure the protection of users and the environment. First, one of the primary safety risks involves the potential for electrocution, as the device operates near water. To take account of this, the electronics will be housed away from the water source, ensuring no water can enter the sensitive areas. We will also use Ground Fault Circuit Interrupter (GFCI) outlets to further protect users from electrical hazards. Additionally, the system will be designed to operate at low voltage, reducing the severity of any potential electric shocks.

Data privacy and confidentiality are also critical ethical considerations since the system collects water usage data from households. To protect user privacy, we will implement secure data encryption for all data transmissions, along with robust access control mechanisms such as multi-factor authentication. These measures ensure that user data remains confidential and protected from unauthorized access or misuse.

Environmental sustainability is another key concern, given the potential ecological impact of electronic devices. To address this, we will use materials that minimize the presence of harmful substances like lead. The device casing and components will also be designed with recyclable materials to ensure environmentally responsible disposal at the end of the product's life cycle.

Ethical considerations regarding the collection and use of data will be managed by providing users with clear information about what data is collected and how it is used. In sum, our project's design incorporates strong safeguards to ensure user safety, protect privacy, and promote sustainability, demonstrating a commitment to ethical responsibility throughout the system's development and deployment. [1][2]

5 Citations

1 "ACM Code of Ethics and Professional Conduct." *Association for Computing Machinery*, https://www.acm.org/code-of-ethics. Accessed 3 Oct. 2024.

2 "IEEE Code of Ethics." *Institute of Electrical and Electronics Engineers*, https://www.ieee.org/about/corporate/governance/p7-8.html. Accessed 3 Oct. 2024.

3 "Programming with the Low-Level API." *Amazon DynamoDB Developer Guide*, Amazon, https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.LowLevelAPI.html. Accessed 3 Oct. 2024.

4 Synnes, Endre. "ESP32 and Ultrasonic Sensor Tutorial." *YouTube*, 19 June 2020, https://www.youtube.com/watch?v=TzbImff5KOO.

5 Synnes, Endre. "ESP32 Data Logging with AWS." *YouTube*, 9 Apr. 2021, https://www.youtube.com/watch?v=x5TcGHUahN8.

6 "Amazon CloudWatch." *Amazon Web Services*, https://aws.amazon.com/cloudwatch/. Accessed 3 Oct. 2024.

7 "Postman API Client." *Postman*, https://www.postman.com/product/apiclient/. Accessed 3 Oct. 2024.

8 "S3 Data Import Validation." *Amazon DynamoDB Developer Guide*, Amazon, https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/S3DataImport.Validation.htm. Accessed 3 Oct. 2024.

9 *Wireshark User's Guide*. Wireshark Foundation, https://www.wireshark.org/docs/wsughtml/. Accessed 3 Oct. 2024.

10 "Amazon DynamoDB and AWS Lambda." *Amazon Web Services*, https://aws.amazon.com/dynamodb/ and https://aws.amazon.com/lambda. Accessed 3 Oct. 2024.

11 "ESP32 LCD API Reference." *Espressif Systems Documentation*, Espressif, https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/lcd.html. Accessed 3 Oct. 2024.

12 "The Basics of Ultrasonic Sensors." *SameSky Devices*, 3 Oct. 2024, https://www.sameskydevices.com
basics-of-ultrasonic-sensors#:≃Ultrasonic%20sensors%20emit%20a%20chirp,to%20bounce%20off%2
Accessed 3 Oct. 2024.

13 "How Ultrasonic Sensors Work — Science in 5." *YouTube*, uploaded by World Health Organization, 3 Feb. 2020, https://www.youtube.com/watch?v=JRKlR4YgMHw. Accessed 3 Oct. 2024.

14 "LD1117: Low Dropout Voltage Regulator." *STMicroelectronics*, https://www.st.com/en/power-management/ld1117.html. Accessed 10 Oct. 2024.