DESIGN AND DEVELOPMENT OF IOT ENABLED COST AND ENERGY EFFICIENT WATER FLOW METERS

Sarwan Shah

HABIB UNIVERSITY
KARACHI, PAKISTAN
2021

Contents

List of Figures

List of Tables

Design and Development of WiFi Enabled Cost and Energy Efficient Water Flow Meters

Sarwan Shah[1], Junaid Ahmed Memon[2], and Dr Hassaan Furqan Khan[3]

[1] Student Research Assistant, Electrical Engineering,
[2]Lecturer, Electrical Engineering and Computer Engineering,
[3]Assitant Professor, Integrated Sciences and Mathematics,
[*]at Dhanani School of Science and Engineering, Habib University, Karachi, Pakistan.

Introduction

The work presented in this report has been conducted over a period of 4 months from February 2020 to May 2020. The work done so far can be considered to be progressing towards the completion of a first draft for the below proposed solution.

The idea of developing a solution or device that can measure the flow rate of water in a pipe with high accuracy to get accurate estimates of water volume consumption is not entirely novel.

While there exist several solutions globally that cater to the need for metering water connections or pipes, one bottleneck in availing any of these is that most of them are expensive and availability within Pakistan is limited. This makes them unfeasible for large scale implementation locally. Another major short-coming with most of these solutions is that they are not IoT (Internet of Things) enabled. They are unable to transmit their readings over a network and require taking manual readings.

These factors have motivated us to develop an integrated solution that fulfils our required needs while maintaining cost-effectiveness and longevity. These requirements can be summarized as follows:

- A device that can measure the flow rate of water passing through a cross-section of pipe with a high degree of accuracy and digitally transmit this.
- A metering system that can reliably and continuously receive, store, process and transmit this information across a WiFi network for an extended period of time.
- An online server and user-end software to organize, log, visualize and post-process incoming data from a network of such systems for analysis.

The next section provides a detailed account of the methodologies, materials and devices being utilized in designing the solution, their rationale, specifications, and how they integrate in the overall design. Following that we touch upon remaining work and future plans, finally moving towards the conclusion of the work achieved so far.

System Design Overview

The diagram below captures the layout of our system's latest design, its integral components, and how they interact with each other:

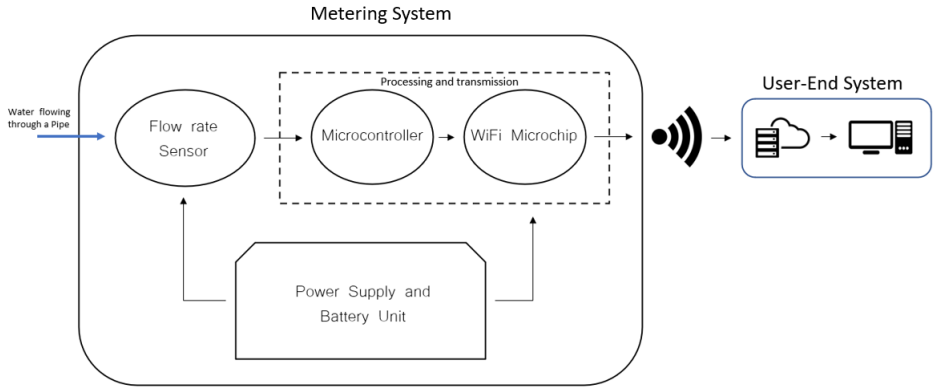

Figure 1. System Diagram

Metering System:

Water Flow Rate Measurement

The G1-FS400A Hall-Effect type flow rate sensor was chosen as the sensor of choice for measuring the flow rate water. The reasons for that include their relative ease of use in-terms of their working principle, coupling with microcontrollers, available sizes and integration with household water pipes. Furthermore, they also offer a fairly high degree of accuracy (up to 97%), easy calibration, and are widely available in the market under a reasonable cost margin. The table below summarizes the specification for the utilized flow sensor:

Parameter	Rating
Size	1"
Min. Working Voltage	4.5 V
Max. Working Current	15 mA
Working Voltage Range	5 – 24 V
Flow Rate Range	1 – 60 L/min
Load Capacity	10 mA
Operating Temperature	80 °C
Liquid Temperature	120 °C
Operating Humidity	35%-90%RH
Water Pressure	1.2 Mpa
Storage Temperature	-25 °C -80 °C
Pulse Frequency to Flow Rate Constant	4.8
Storage Humidity	25%-95%RH

Table 1. Flow Sensor Specifications

The rated pulse frequency to flow rate constant for the given sensor was **4.8** and gave an accuracy of about **93%** which was determined by experimentation on a setup- designed using an Arduino, graphical user-interface application-, stopwatch and a fixed 4 litre volume of water. By repeated trail and error of changing the constant's value, a maximum accuracy of about **97%** was achieved at the value of **4.75**, hence, successfully calibrating the sensor. This can be seen in Figure 6.

Flow Rate Constant	Value	Accuracy
Rated	4.8	93%
Calibrated	4.75	97%

Table 2. Calibration Results

Figure 2. Testing and Calibration Setup

Processing Unit

The processing unit or microcontroller being utilized for the Metering system operations is the AVR Attiny85. We initially started with the NodeMCU Development Board, however, it's high power-consumption and large form-factor in contrast to its powerful processor, large memory, WiFi chip and several I/O Pins played out to be a major downside for our limited purpose of processing simple pulses from the flow sensor to measure and store the flow-rate values.

On the other hand, the Attiny85 allows major cutbacks in power consumption and form-factor to our advantage, as we aim to battery-power the system for an extended period of time. Albeit, at the expense of not having WiFi transmission capabilities, but this is efficiently taken care of by our transmission unit.

Micro-controller	Nominal Current Draw
NodeMCU - Normal Operation	200 mA @ 3.3V VCC
Attiny85 - Normal Operation	2 mA @ 3.3V VCC

In addition the Attiny85 is also responsible for keeping check of the battery's capacity or voltage level via its built in ADC (Analog to Digital Converter) to allow triggering necessary mechanisms that allows the battery to charge, alert the user-end about potential shutdowns, update LED status's and back-up essential data.

Furthermore, the Attiny85 is also responsible for triggering the transmission controller and transmitting data to it for transmission over WiFi to the user-end system, as the Attiny85 itself lacks any mechanism to transmit over the WiFi.

Transmission Unit

For the transmission of data to the user-end system we are utilizing an ESP8266 low-cost IoT WiFi Micro-chip. This seamlessly integrates with the Attiny85, operating in slave-mode, allowing the Attiny85 to periodically transmit data to it for transmission over WiFi, which further minimizes its average power-consumption.

Micro-controller	Nominal Current Draw
ESP8266 - Normal Operation	80 mA @ 3.3V VCC
ESP8266 - Deep Sleep Mode	0.02 mA @ 3.3V VCC

The ESP8266 is be programmed to utilize the increasingly popular MQTT messaging protocol for transmission of data. The MQTT is an open OASIS and ISO standard lightweight, publish-subscribe network protocol that allows transmission of messages between multiple IoT devices at a very low-latency. The protocol being lightweight and having several online services that provide dashboards for managing data incoming from multiple devices is an excellent choice for our task which will involve managing a network of such devices.

Battery and Power Supply Unit

Our finalized battery and power supply setup is best summarized by the circuit diagram provided below:

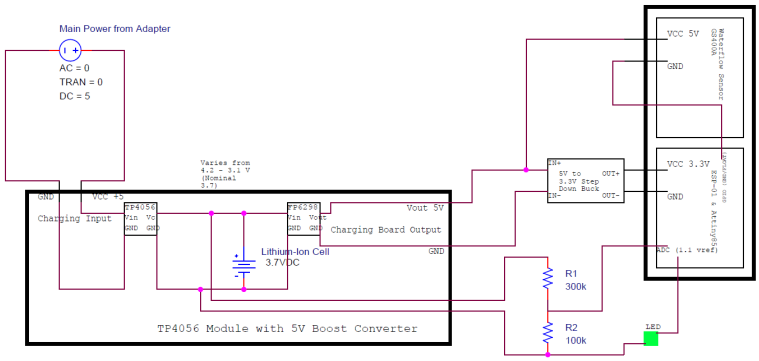

Figure 3. Power Supply Circuit Diagram

Our system's remote nature and its requirement to run continuously requires that its power-supply be uninterruptible, while also being able to provide a large enough battery back-up to support it through long power outages, which are common feature of localities in the city of Karachi.

The battery utilized in the power supply setup is the 18650 Single Cell Lithium-ion Battery with over-charging protection. The batteries relatively flat discharge curves, high energy density, high number of discharge cycles, and quick-charging, in addition, to our setup's low current and a shallow discharge cycle requirement make it the best choice for our Metering application; leaving options such as the Lead Acid batteries better suited for more heavy-duty standby applications. However, on the contrary the Lithium-ion battery does pose some challenges with regard to it's charging, which requires a special 3 step process: Pre-charge, Constant Current Charge and Constant Voltage Charge. These are characterized by the graph below:

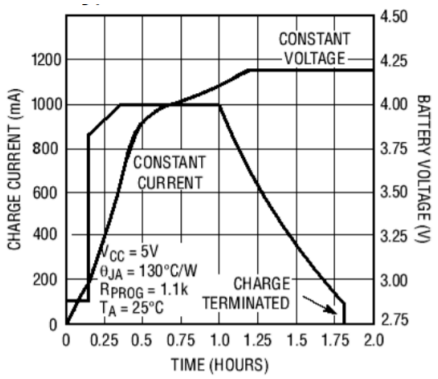

Figure 4. TP4056 Charging Characteristics from Datasheet

To cater to this end of a specialized charging process we are utilizing the TP4056 1A Standalone Linear Li-ion Battery Charging chip module which has a low cost and does this efficiently.

Module Parameter	Rating
Charging Current	1000 mA
Quinescent Current Draw	200 uA

The module we are using integrates this chip with a 5V boost-converter that supplies power to both: our flow sensor and an efficient 3.3 V MIC5219 regulator chip, which in-turn is responsible for supplying power to the Attiny85 and ESP8266.

The only challenge around this IC is that if the battery is discharged below 3 volts, the charger only supplies a trickle charge current of about 100 mA to main line on the main line as a part of the pre-charge process. This main line is shared by both the load and the battery. This trickle charge continues until it the battery charges and recovers above the threshold 3 voltage, after which it jumps back to 1A. As such, if in this state a load consuming around 100 mA is connected to the circuit, the battery will never be able to recover as the load will be consuming all the charge.

To solve this issue we are utilizing the Attiny85 analog to digital converter, which using a voltage divider circuit will keep check of the battery's voltage level. If the voltage level nears the 3 volt threshold, the Attiny85 will back-up and shutdown all essentials operations, minimizing current consumption, until the battery's voltage level has recovered above a set threshold.

Power Supply Circuit Testing

The circuit shown in figure 3 was tested using module a that integrated the TP4056 IC with a 5V regulator, while a micro-USB cable acted as the mains supply. The power supply was able to supply the battery with a charge current of **540 mA**, suggesting the R_{prog} resistor to the TP4056 on this particular module was set to **2K ohms**, as speculated. This will be set to **1.2K ohms** in our design ensuring a charge current of **1000 mA**.

A **22 ohm** resistive-load was also attached to the circuit that reduced the batteries charge current to **339 mA**, given the load drew a current of **240 mA**. This verified that the circuit could support a simultaneous supply to both the load and battery. In addition, the mains supply was removed abruptly while measuring the voltage and current reading on the load; no noticeable current or voltage drop was observed on the digital multi-meter, confirming the uninterruptible nature of the circuit proposed.

Lastly, the battery was allowed to charge to maximum capacity at which point an led switched green, the charge current dropped to a negligible value **65 uA**, and a voltage of **4.25 V** was observed. After this the battery was allowed to discharge below **2.9 V** using the same **22 ohm** resistance. Below this point, the load current dropped to zero not allowing the battery to further discharge. After this, when the mains supply was turned on, the load observed a current of **64 mA**, while no-current was observed to the batter until once the **22 ohm** was increased to **220 ohm**, reducing the load current draw to **23 mA**. All of these results were consistent with our theoretical findings and expectations within fair bounds.

Power Consumption Analysis and Optimizations

Our application at the Metering end will be dependent on a battery for its operation partially in some cases, while completely in others. This required we that have a fairly good estimate of our system's power (or current, which is directly related) consumption in-order to gauge the battery capacity required and how long it would last. The table below lists the current consumption of the various under-consideration components:

Component	Current Draw
G1-FS400A	15 mA
Attiny85 - Normal Operation	2 mA @ 3.3V VCC
ESP8266 - Normal Operation	80 mA @ 3.3V CC
TP4056 + Boost Converter Module - No load	200 uA
MIC5219 Regulator - No load	30 uA

Note: All these values have been measured directly of the components using a digital multi-meter while they were under operation individually and have been cross-verified against the values given in their data-sheets.

This sums up to an estimated current consumption of about **98 mA** if all components are under continuous operation. Adding another 10% in value to remove any room for underestimation results in **108 mA**. At this rate our typical single cell Lithium-ion battery rated at **2200 mAH** would only last about **20 hours** (2200/108).

This number of hours may suffice in situations where the main's power supply is easily accessible and in an area that does not face frequent power outages, however, a number in the order of thousands of hours is required for uninterrupted operation if it is not accessible and power outages are common. Even adding more battery cells cannot make a significant difference.

To solve this issue we have utilize software optimizations on our Attiny85 and ESP8266 chips, which involve:

- Keeping the Attiny85 in sleep-mode continuously and only running it in normal mode for a fraction of a second to perform three essential operations: checking battery level, checking time-escaped since last transmission, and enabling power to flow sensor, taking a reading from it and storing it.
- The ESP8266 remains in deep-sleep continuously and is only triggered into normal operation by the Attiny85 to bulk transmit data for a few seconds every 600 seconds.

Component	Nominal Current Draw	Avg. Est. Current Draw Per Second
ESP8266 - WiFi Operation (10 s every 600 seconds)	80 mA	1.5 mA
ESP8266 - Deep Sleep Mode (Continuously)	0.02 mA	0.02 mA
Attiny85 - Normal operation (50 ms each second)	2 mA	0.08 mA
Attiny85 - Sleep Mode (950 ms each sec)	0.02 mA	0.02 mA
G1-FS400 - Normal operation (10 ms each second)	15 mA	0.2 mA

We can see these optimizations drastically reduce our estimated current consumption to **1.8 mA**, while removing room for underestimation we round it off to **2 mA**. At this rate our typical single cell Lithium-ion battery rated at **2200 mAH** last's about **1100 hours** or about **45 days** and adding more battery cells can make a significant difference.

User-End System:

The user-end system can be said to consist of 3 components: a Thingsboard Dashboard, a Local Monitoring Graphical User Interface, and a Python Logging Script.

Thingsboard Dashboard

Thingsboard is an open-source IoT platform for device and data management. We are utilizing their free of cost community edition dashboard as an online monitoring station for our network of water flow meter devices. All our incoming data will be flowing through this central hub, as this service seamlessly integrates with our ESP8266 on the Metering-end, which is using the Thingsboard's MMQT service for transmitting data over WiFi with low-latency.

With a 1000 messages per device (or node) per second limit on the free-edition and a plethora of other features, this dashboard out-features any other open-source IoT service provider we could find, such as the Adafruit IO, which doesn't allow more than 1 message per second. The dashboards key features can be summarized as follows:

- The ability to display stats for multiple devices, such as live plot of incoming data, activity status, last connect and disconnect times, location maps etc. These stats will be critical in identifying any faults or downtimes across the network.
- The ability of allowing multiple users to access the dashboard and setting their permission level as per our need
- The ability to apply 'Rule Chains' to incoming data, which allow to filter and post-process incoming data for analysis. A feature that can be handy in the future scope of the project.
- An extensive API (application programming interface) that allows data to be easily accessed by external programs.

Local Monitoring Graphical User Interface

This graphical user-interface application was developed in Python using the PyQt library. This graphical user-interface was initially developed to calibrate the flow sensor and understand & process the data incoming from of the Metering-end system. It is now fully-integrated with the Thingsboard API, and will be serving to display data from multiple devices locally on any computer in a coherent manner. It's key features are:

- Displaying flow-rate for a device over time, in addition to other application critical information such as the volume flown, instantaneous flow rate, average flow rate, and time escaped.

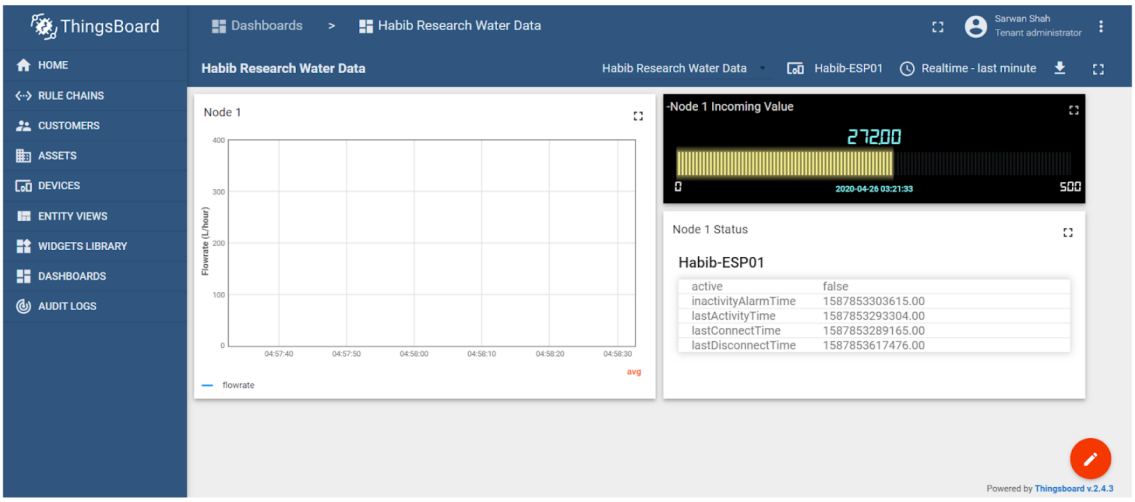

Figure 5. Thingsboard dashboard showing the status and data from one device

- The ability to switch between multiple devices.
- The ability of starting, stopping, and resetting the display of incoming data.
- The continuous logging of incoming data from multiple nodes.

Figure 6. Graphical User-Interface Application

Python Logging Script

This is a script programmed in Python that is fully-integrated with the Thingsboard API and can continuously log all the essential data incoming from the network of flow meter devices in a sorted and coherent manner into files and folders.It's key features are:

- Continuously logging data with timestamps in a format that allows easy importing to excel.
- Continuously organizing data in folders based on date and device ID's
- Can be easily run on any Python-supported servers.

Work Completed

- About 80% of the draft design for the Metering-end system has been finalized. This has involved: the testing and calibration of the G1-FS400A flow sensor; a technical feasibility assessment and testing of the microcontroller (Attiny85) and networking chip (ESP8266) of choice; a detailed analysis of the systems power need; the designing of an optimized and need-specific uninterruptible power supply circuitry; and the designing and feasibility assessment of a software-based theoretical power optimization framework.
- About 90% of the software and networking needs for the user-end system have been finalized. This has involved: development and testing a graphical user-interface application in Python; the integration of the MMQT protocol with the GUI application; the setting up and testing of the Thingsboard dashboard; the integration of the Thingsboard API with the GUI Application; and the development of Python logging script integrated with the Thingsboard API.

Remaining Work

- Implementing, testing and validating the proposed software-based power optimization in Metering-end system.
- The software and hardware integration of all Metering-end system components into a fully-functional prototype.
- Extensive testing of fully-functional Metering-end system prototype
- Further revisions and optimizations to first draft design of Metering-end system.
- Integration and testing of Thingsboard dashboard, local monitoring GUI and Python logging script with fully-functional prototype for multiple interval-based data streams.
- Further revision and optimization of user-end system features as per needs and requirements.

Conclusion

We have made a reasonable amount of progress in these last 4 months, despite the unfortunate and disabling circumstances that were a consequence of the COVID-19 pandemic. The system's current design covers all of our initially proposed needs and goals in a highly efficient and optimized manner, while ensuring cost-effectiveness too. We are beginning to see a mature engineering system take form, whose true reliability, durability and usefulness should be reflected once put to test in the form of a fully-functional prototype.

Design and Development of GSM Enabled Cost and Energy Efficient Water Flow Meters

Sarwan Shah[1], Junaid Ahmed Memon[2], and Dr Hassaan Furqan Khan[3]

[1] Student Research Assistant, Electrical Engineering,
[2]Lecturer, Electrical Engineering and Computer Engineering,
[3]Assitant Professor, Integrated Sciences and Mathematics,
[*]at Dhanani School of Science and Engineering, Habib University, Karachi, Pakistan.

Introduction

In the previous iteration of our developmental work on the design of an IoT enabled cost and energy efficient water flow meter device we concluded with a design that utilized WiFi in a power optimized setting for extended operation. However, the following limitations were identified with the proposed design:

- The proposed system was only limited to places with WiFi access, and hence could not be utilized in settings beyond it. This dependency was further strained by regular access to electricity, as WiFi directly depended on it.
- Measurements were only sampled for a small duration in relatively larger interval between measurements, which lead to a loss of data and accuracy.
- There was a significant lag between the time measurements were made by the water flow meter and the data was received on the server side. This would be worsened by any failure related system resets, leading to inaccuracies in the temporality of the data.

While the limitation of WiFi might not have posed a serious challenge for a study of water demand sough out in a developed countries, however, concerning a developing country and a city like Karachi where both: access to WiFi and electricity can be scarce in localities, we identified it would hinder the spatial scope of the study in the city. Similarly, we found it critical to have accurate information on the temporal changes in data to be able to identify and study temporal trends in water demand.

The work presented ahead has been conducted over a period of 4 months from September 2020 to December 2020. It has been aimed at developing and implementing a system design that can cater to the aforementioned limitations and hence contribute to improving the proposed study.

In the next section we will provide a detailed account of the new system's design and implementation, the improvements, their rationale, specifications, limitations, and how the entirety of it integrates with the our existing design. After that we will address the work that remains, potential improvements, and future considerations, finally concluding with a short summary of the proposed system design and implementation.

System Design Overview

The diagram below captures the layout of our system's latest design, its integral components, and how they interact with each other:

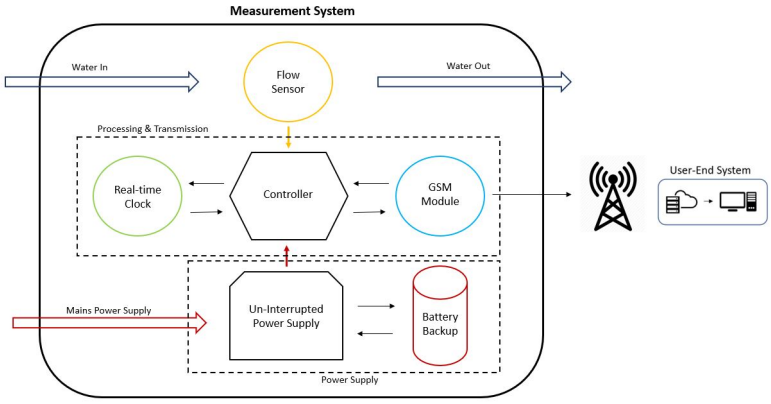

Figure 7. System Diagram

To summarize our current system flow, the microcontroller acts as a central hub which records data from the real-time clock and flow sensor at a very precise defined intervals, and continues doing this for a larger defined interval. Once this larger interval has passed, it communicates these sets of flow readings along with their timestamps to the GSM module which transmits them together as a single message, after which upon success, it clears it memory and repeats the process. In the event of a failure, it will continue to attempt to re-transmit that message until a successful transmission is achieved.

There are two major design changes reflected in this new system design in comparison to what was presented in figure 9. Firstly, we have introduced a real-time clock module which allows keeping precise track of the time when each reading is recorded. Furthermore, we have replaced the WiFi transmission chip with a GSM module for transmission, allowing our system to transmit from all locations where cellular service is available. This provides an over-arching functional structure of our system, and we will now continue to expand on each component of our metering system with individual detail.

Metering System:

Processing Unit

In our previous design we utilized the Attiny85 to act as our main controlling unit. While although sufficient in its capabilities for our previous design, it posed the following constraints when it came to catering to the design limitations mentioned in the introduction:

1. Memory limited to 512KB for storage, processing and incorporating other improvements.This would particularly be a problem when introducing timestamps a data point in addition to the flow rate.
2. Only 6 general purpose I/O pins (GPIO) for interfacing other modules, leaving little room for upgrades by interfacing new components.
3. Only 2 on-chip timers, out which only 1 is independently usable. (These will be critical for time tracking and measurement applications, and their need will be discussed ahead in the report.)

It will become more clear as to why these were limitations as we progress through the remainder of the report. A cost-benefit analysis was performed for deciding a new microcontroller, as shown in figure 8, and the Atmega328p was decided to be the most optimum due its availability, high resource support, and fair specifications in terms of memory, GPIO's and timers. The trade-off being a small compromise in-terms of the form factor and power consumption.

Microcontroller	Architecture	Program Memory (KBs)	SRAM (Bytes)	EEPROM (Bytes)	I/O Count	CPU Speed (MHz)	Op. Voltage (V)	Op. Current (mA)	Communication (UART -- SPI -- I2C)	Price (PKR)	Availability
Attiny85	8-bit	8	512	512	6	8	1.8 - 5	1 -- 5	0 -- 1 -- 1	PKR 200	High
Attiny1614/16/17 & Attiny3216/17	8 bit	16 & 32	2048	256	14/20/24	20	1.8 - 5	1 -- 5	1 -- 1 -- 1	NA	Not available online
Atmega328p	8-bit	32	2048	1024	28	16	2.7 - 5.5	1 -- 12	1 -- 2 -- 1	PKR 270	High
Atmega2560	8-bit	64	8192	4096	54	16	1.8 - 5.5	2 -- 24	4 -- 5 -- 1	PKR 950	Low
Atmega4809 (Nano Package)	8-bit	48	6144	256	48	20	1.8 - 5.5	0.5 -- 9.5	3 -- 1 -- 1	PKR 2800	Low
STM32	32-bit	64/128	20480	NA	32	72	2.0 - 3.6	5 -- 50	3 -- 2 -- 2 (+ CAN Bus & USB)	PKR 450	High

Comparision

Microcontroller	Pros	Cons		
Attiny85	low cost, small form factor, low power consumption, easy availability	low memory, limited I/O pins (less room for expansion), high support resources	Color Code:	Least Desirable
Attiny1614/16/17 & Attiny3216/17	low cost, small form factor, low power consumption, fair memory, fair I/O	not available in Pakistan, limited support resources		
Atmega328p	low cost, low power connsumption, fair I/O, very high support resource	moderate memory		
Atmega2560	large form factor, high memory, high I/O, high support resources	expensive, large form factor, moderate power consumption, low availability		
Atmega4809 (Nano Package)	m factor, high memory, high I/O, dedicated RTC, fair support resources, low power con	expensive, low availability		Most Desireble
STM32	*5V tolerant pins, 16-bit PWM resolution (AVR 8-bit), huge SRAM, RTC, CAN bus	limited arduino IDE resource support, high power consumption		

Figure 8. Microcontroller Cost-Benefit Analysis

Water Flow Rate Measurement

The water flow measurement system in terms of hardware remains unchanged in the new system design. It utilizes the same G1-FS400A hall-effect type flow rate sensor.

Some new observations were made with regard to the use of this particular type of flow sensor. Upon being tested after a 5 months gap, during which the sensor was exposed to residual water, the sensor was no longer giving flow rate values that were consistent with the ground truth (4 Litres over 50 seconds) it was previously tested and calibrated against (see table 2).

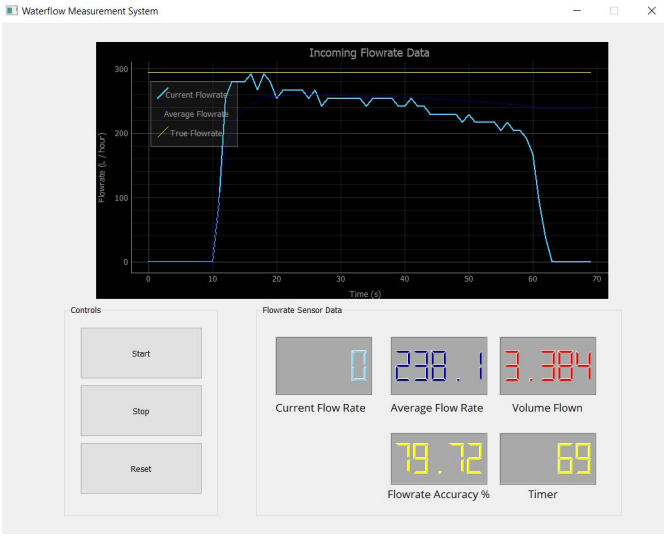

Figure 9. Flowmeter Test

The flow rate's graphical results show a plausible trend in the above test, suggesting that flow meter might only require some software calibration, but it is worth mentioning that certain inconsistencies, such as a considerable difference in the average flow rate between tests, were also observed. However, it should be clear that these observations required further testing for verification to identify the root of the issue.

This testing could not be carried out due to time constraints, and hence a new flow sensor was acquired to curb the issue. The newly acquired flow sensor was calibrated to the of following specifications:

Flow Rate Constant	Value	Accuracy
Rated	4.8	85.9%
Calibrated	4.0	98.9%

Table 3. Calibration Results

In addition, an a theoretical study of flow rate measurement was also carried out. The purpose being to formulate a generalized mathematical model that equated the flow rate with variables such as pipe size, number of flow sensor pedals, and pedal radius

to determine the flow constant. The following equation was derived:

$$f = \frac{n}{2\pi r}\frac{(1.1111x10^{-6})60}{\pi d^2}Q \tag{1}$$

where f is the frequency from the hall sensor, n is the number of flow sensor pedals, r is the radius of the flow sensor pedals, d is the diameter of the pipe, and Q is the volumetric flow rate in litres per hour. These variables were determined for our current sensor and setup, and the following relation was determined:

Variable	Value
Number of Pedals	8
Pedal Radius	0.01 m
Pipe Diameter	0.0254 m^2

Table 4. Sensor & Piping Specifications

$$f = 4.18Q \tag{2}$$

Hence, the theoretical flow constant for our sensor and system was determined to be **4.18**, which is different from our rated and calibrated values of **4.0** & **4.8** respectively. This difference can be attributed to number of factors such as approximations and idealizations in the theoretical derivation, utilized coupling mechanism in the real system or the tilt in the piping mechanism.

Lastly, a more robust approach towards the measurement of flow was incorporated in the new design. In the previous iterations, the flow rate, or more specifically, the frequency generated by the flow sensor was only sampled at specific intervals, depending on how the microcontroller was programmed. This gave rise to two issues:

1. A loss of accuracy in the measurement, as each measurement would only reflect the flow rate at the instant or during the duration of sampling, which would be limited, as the microcontroller will need to cater to other task periodically
2. There would be a loss of data if the sampling interval needs to be increased. Increasing the sampling interval is a desirable feature as it would allow the microcontroller to cater to other tasks of importance for extended periods and would be significant in allowing power consumption related optimizations to the system.

To cater to this end, the timers on the Atmega328p were utilized. Timers are integrated circuits that are embedded within the micro controller and run independent of its core functions. They are often used to keep track of time in such systems, however, they can also be configured to behave as counters, in which case they can count the number of pulses an external signal feeds on their input.

This feature allows them to be utilized for frequency measurement applications, where the frequency of a signal is essentially the count registered by the timer in one second. In addition, another on-chip timer is used to keep precise track of the time period for which the pulses have been counted. The advantages of using this scheme of timers for frequency measurement of the flow sensor can be summarized as follows:

- They operate independent of the microcontrollers core operations, and hence, can be used to the continuously sample the frequency, and consequently flow rate, from the flow sensor. This ensures no data is lost.
- It helps improve the accuracy, as no data is lost, and there is complete information about the flow of water at every instant of time .
- It allows us to increase the sampling interval without losing data, allowing us to increase our transmission time and introduce power consumption related optimizations in the future.

The following table summarizes some of the limitations on the timers:
It was also ensured through study that non of the existing code or libraries - SoftwareSerial (used for communication with GSM Module) & RTCLib (for communication with real-time clock module) - were in conflict with the timers, as some libraries tend to utilize them.

Sampling Interval	Value
Minimum	64 μs
Maximum	4.36 years

Table 5. Theoretical Sampling Interval Limits

Real-time Clock (RTC) Unit

The real-time module consists of a commercially available TinyRTC module that integrates the popular DS1307 RTC chip, AT24C32 EEPROMM, and a coin cell backup slot. The following are the specifications for the module:

Parameter	Value/Type
Operating Voltage	3 - 5.5 V
Communication	I^2C
On-board Storage	56 bytes + 32 Kbits EEPROMM
Low Powered	Yes
Battery Backup	Yes
Back-up Power Consumption	500 nA (Rated)

Table 6. TinyRTC Module Specifications

The module is responsible for providing the microcontroller with a timestamp of the instant a measurement or reading is recorded. This is then added to packet of data, along side the flow rate measurement, providing a temporal reference for each reading.
Another attractive feature the module offers is a coin-cell battery back-up, which in the event of a failure puts the RTC in back-up mode in which it consumers also low as **500 nA**, allowing for a theoretical back-up of **50 years**. Hence, allowing our the system to maintain a track of time in the event of a restart or power failure.

Transmission Unit

The transmission unit utilizes a SIM800C module, which consists of a Quad-band GSM/GPRS chip. The following table summarizes the specifications of this module:

Parameter	Value/Type
Operating Voltage	3.4 - 4.4 V
Communication	AT Commands via UART(Tx/Rx)
On-board Storage	32 Mbits SRAM
Idle Current Consumption (GSM850 Mode)	13.8 mA @ 4V (Rated)
SMS Transmission Current Consumption	800 mA @ 5V (Observed)
Peak Transmission Current Consumption	2A @ 4V (Rated)
Sleep Mode	0.6 - 1.5 mA @ 4V (Rated)

Table 7. SIM800C Module Specifications

Our microcontroller, the Atmega328p, only has one UART for communication. This was currently utilized for transmitting information between the PC and microcontroller for programming it, receiving information relevant to debugging and for visualizing the program flow.

To cater to the GSM Module another UART was required. To this end, a library known as ”SoftwareSerial” was used. SoftwareSerial uses code to mimick a UART on any 2 selected Atmega328p’s GPIO’s at a baudrate of 9600 bps. All communication is mediated by AT commands which are transmitted serially, which is a set standard for cellular networking devices.

Each AT command sent is responded by an echoed response which contains a copy of the sent command, in-addition to a response providing information about any results followed by an ’OK’, indicating successful execution.

Since all this communication happens serially, responses cannot be directly used to control changes in the program flow. To this end, a programming framework has also been developed that allows for the efficient sending and response extraction of AT commands. This allows for a more stronger control over the program flow. For example, we now instruct the program to only move ahead in the process once there is a confirmation of successful execution (which can take a few milliseconds) from the GSM module, or how to change the program flow in the event of an error.

Furthermore, powering up the module requires that it hard reset pin get pulled down. This was not possible to simulate using the Atmega328's GPIO's directly, leading to unpredictable behavior. Hence, a small transistor circuit was developed to pull-down the hard-reset pin on the SIM800C module, and when there was high pulse from the Atmega328p for 1.5 ¡ t ¡ 2 seconds, it would reset the module forcing it to start.

Prototyping

Circuit Design

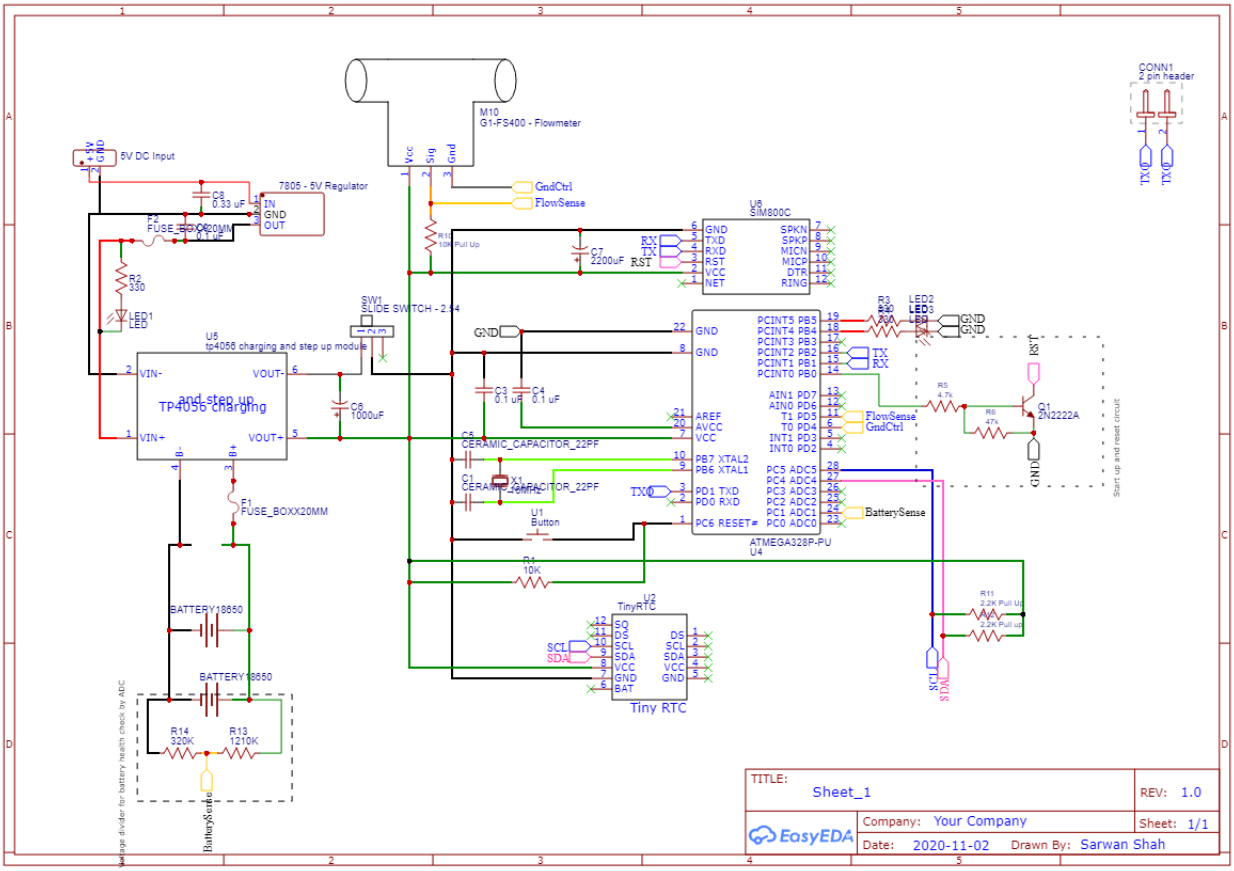

Figure 10. Circuit Schematic