Grand Valley State University

ScholarWorks@GVSU

Masters Theses

Graduate Research and Creative Practice

12-2018

Power Optimization of Solar Powered Standalone Wireless Sensor System

Rajan Amatya

Grand Valley State University

Follow this and additional works at: https://scholarworks.gvsu.edu/theses

Part of the Power and Energy Commons

ScholarWorks Citation

Amatya, Rajan, "Power Optimization of Solar Powered Standalone Wireless Sensor System" (2018). *Masters Theses*. 915.
https://scholarworks.gvsu.edu/theses/915

This Thesis is brought to you for free and open access by the Graduate Research and Creative Practice at ScholarWorks@GVSU. It has been accepted for inclusion in Masters Theses by an authorized administrator of ScholarWorks@GVSU. For more information, please contact scholarworks@gvsu.edu.

Power Optimization of Solar Powered Standalone Wireless Sensor System

Rajan Amatya

A Thesis Submitted to the Graduate Faculty of

GRAND VALLEY STATE UNIVERSITY

In

Partial Fulfillment of the Requirements

For the Degree of

Master of Science in Electrical & Computer Engineering

School of Engineering

December 2018

Acknowledgement

I would like to extend my sincere gratitude to Dr. Robert Bossemeyer, for constantly guiding and supporting me to complete this thesis work. I am very thankful to Dr. Heidi Jiao and Dr. Nabeeh Kandalaft for providing necessary feedback and suggestion to improve my research. I am thankful to Dr. Bossemeyer and Dr. Jiao for the Embedded System Interface class, where I was introduced to Texas Instruments MSP432 board, including analog and digital design techniques. I would like to thank my lab supervisor Ryan Aldridge for building the LTC3106 board and helping me debugging the board. Lastly, I would like to thank my graduate advisor Dr. Shabbir Choudhuri, for guiding me throughout my entire master's program.

Abstract

Wireless sensor systems are common in applications where harsh environmental conditions or remote locations make it difficult to run wires. Having these sensor systems self-powered is essential as it is difficult for battery replacement. Using solar as an alternative source of energy can be a medium to charge the battery in such scenarios. With an increasing number of portable devices powered by battery; more and more research is focused on low-power design technique. For this thesis, a microcontroller (Texas Instruments MSP432) was used along with a digital sensor (Bosch BME280) to provide temperature, pressure and humidity information, and Wi-Fi module (Espressif ESP8266) to transmit the information to the Internet where it was posted to a spreadsheet on a web server. Power levels were measured with different modes in order to compare the power consumption of the MSP432. In addition, a solar charging circuit was designed to provide management of a Lithium-ion battery to explore the feasibility of operating this Internet connected sensor system entirely off the power grid. Incremental steps were taken to reduce the power consumption by the system and a system was designed to run with the lowest power consumption. The power requirements for each component connected to the microcontroller were calculated and then optimized. The minimum current required during transmission was reduced by 25.71%. A 2000mAh battery was selected for the final design to maintain system operation during extended dark hours. Solar panels were able to charge the battery whenever sunlight was available with the intent to keep the system running continuously on battery power. The system was tested for 20 days, and it ran without any interruption.

Table of Contents

List of Tables

List of Figures

Abbreviations

API	:	Application Programming Interface
ARM	:	Advanced RISC Machines
ADC	:	Analog to Digital Conversion
AP	:	Access Point
CS	:	Chip Select
DCO	:	Device Configuration Overlay
ESD	:	Electrostatic Discharge
GPIO	:	General Purpose Input Output
HFXT	:	High Frequency Crystal Oscillator
IDE	:	Integrated Development Environment
IC	:	Integrated Circuits
I/O	:	Input Output
IrDA	:	Infrared Data Association
IEEEE	:	Institute of Electrical & Electronics Engineering
JTAG	:	Joint Test Action Group
LCD	:	Liquid Crystal Display
LDO	:	Low Dropout Regulator
LFXT	:	Low Frequency Crystal Oscillator
LPM	:	Low Power Mode
MSPS	:	Mega Samples Per Second
MODOSC	:	Module Oscillator
MIMO	:	Multiple Input Multiple Output

P2P	:	Point to Point
PV	:	Photo Voltaic
PWM	:	Pulse Width Modulation
PCM	:	Pulse Width Modulation
PSM	:	Power Supply System
RO	:	Reference Oscillator
RTC	:	Real Time Clock
SYSOC	:	System Oscillator
SAR	:	Successive Approximation
SPI	:	Serial Peripheral Interface
SRAM	:	Static Random-Access Memory
SDA	:	Serial Data Acquisition
SCL	:	Serial Clock
STC	:	Standard Testing Condition
TCP/IP	:	Transmission Control Protocol Internet Protocol
TFT	:	Thin Film Transistor
ULP	:	Ultra-Low Power
UART	:	Universal Asynchronous Receiver
VLFO	:	Very Low Frequency Oscillator
Wi-Fi	:	Wireless Fidelity

1. Introduction

Background

Wireless sensor systems are designed to collect data using dedicated sensors to monitor and to record environmental or humanitarian physical conditions [1]. Wireless sensor systems are used for measuring different conditions such as temperature, sound, humidity, pressure, pollution levels, water level and so on. When the weather condition is harsh, it is not possible to reach a location and take the required measurements. It is difficult to run wires to all the places. In such locations, the wireless sensor system is important.

As it is difficult to run wires to some remote sensor systems, it is also difficult to power such devices. A battery usually powers these systems. The limited battery life demands the need of self-powering system or a means of charging the battery. Self-powering systems harvest ambient energy from the environment that is usually unused. There are an abundant number of resources in the environment from which power can be generated. Some common environmental energy sources include light, air flow, heat, vibrations, acoustics, and chemical reactions. These power sources can also be used to recharge the battery.

As the device is being powered by an environmental energy, one major challenge would be to minimize the power consumption of the system. Minimizing the power consumption is necessary due to the limitation on the amount of energy that can be generated from the environment. The size of the energy collector plays a crucial role in the amount of energy to be generated. For instance, a photovoltaic solar cell only uses about 17% of the available energy from the sunlight [2]. Hence to generate more electricity multiple solar cells are required combined in an array increasing the size of the energy collector. Minimizing the size of the system is also a major concern to place the system conveniently in various locations.

The need for low power requirements becomes crucial when determining the lifetime of the device. Over time, batteries start losing the power density, hence need to be replaced for proper continuous operation. One option could be using a rechargeable battery, however even rechargeable batteries lose their power density over time [1]. Another option for power storage could be to use a supercapacitor. Supercapacitors are similar to conventional capacitors with much higher capacitors on the order of hundred millifarads to several farads [3]. Ultimately the selection of an energy storage device is dependent on the expected lifetime of the final application.

Research Goal

The goal of this research is to design and study the feasibility of operation of a sensor system consisting of MSP432 Launchpad with ESP8266 Wi-Fi module. The other goal is to optimize the designed system, so it can operate autonomously.

Scope

The scope of this thesis is to build a system using MSP432 as a controller to transmit the data of BME280 sensor to a Google Spreadsheets using ESP8266 Wi-Fi module. The system was designed by investigating different communication protocols for communicating MSP432 with the BME280 sensor and ESP8266 Wi-Fi module. The MSP432 datasheet and the Technical Reference manual were explored to make the control device consume the lowest possible power. The ESP8266 and BME280 datasheets were explored for optimization of the overall power consumption of the system. The power consumption before and after the optimization was observed. The optimized system was powered from the Li-ion rechargeable battery. A small solar panel was chosen to charge the battery in a normal lighting condition. LTC3106 buck-boost converter was chosen to manage power of the system. The optimized system was tested for a 20-day period to make sure the system is power independent and self-sustainable.

Thesis Organization

This thesis is organized into six chapters. This chapter includes the research goals and provides the background information about the wireless sensor system and the need for energy optimization of the system. It also defines the objectives of this research. Chapter 2 describes different hardware components and the software tools used to build the system and measure power consumption. Chapter 3 discusses different optimization techniques available for reduced power operation of the MSP432 and different power modes that can be applied to meet the research goals. Chapter 4 explains the process involved in designing the system, power optimization process, and managing power to the system. Chapter 5 includes different tests performed and the results obtained during those tests. Chapter 6 outlines conclusions made from the testing and provides insight into probable future work to further decrease the power requirements of the wireless sensor system and improve the battery power sub-system.

2. Description of Components

Hardware Description

Different hardware components were used for this thesis. A microcontroller, Texas Instruments MSP432 was used to collect data from a sensor and send it over to a wireless connection. A sensor module (Bosch BME280) interfaced to the MSP432 was used to measure temperature, pressure, and humidity from the environment. A Wi-Fi module (Espressif ESP8266) interfaced to the MSP432 was used for transmitting the sensor readings to a router connected to the Internet where they are posted to a Google Sheets document. A 1.8" Liquid Crystal Display with 128*160 color pixels was used to display the readings of the system. A buck-boost voltage converter (Linear Technology LTC3106 IC) was used to manage power to the system where solar panels were used as a primary source and a Li-ion rechargeable battery as a secondary source. More details about the hardware components are listed below.

2.1.1 MSP432 Launchpad

The MSP432 Launchpad development kit enables users to develop high-performance applications using low power modes [4]. It includes MSP432 microcontroller from Texas Instruments, an onboard emulator with Energy Trace feature which helps in debugging the program without any additional tools and measures total energy consumption of the system [4][5]. Figure 1 is the MSP432 Launchpad board.

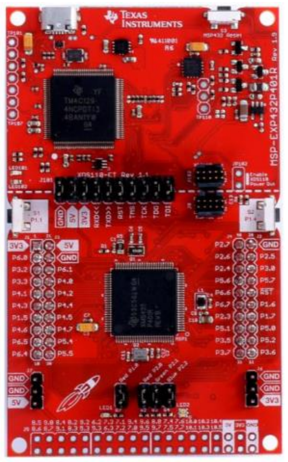


Figure 1: MSP432 Launchpad Diagram[6]

The Launchpad development kit includes an MSP432 microcontroller. Its core is an ARM Cortex M4F processor that provides a high performance, low-power, low-cost platform. ARM technology is in use in 95% of smartphones, 80% of digital cameras and 35% of all electronic devices according to the ARM company database, 2017 [7].

Some of the key features of the microcontroller that was essential for choosing this device for this thesis are:

- Flexible clocking features.
- 256 KB flash main memory
- 64KB SRAM and 32KB ROM with MSP432 Peripheral Driver Libraries.
- Ultra-Low Power Operating Modes
- Four 16-bit timers with capture/compare/PWM, two 32-bit timers and Real Time Clock (RTC)
- Up to eight serial communication channels (I^2C, Serial Peripheral Interface (SPI), UART, and IrDA)
- 14- Bit 1-MSPS SAR ADC that includes Differential and Single- Ended Inputs

- Internal Voltage Reference with 10-ppm/°C
- Ultra-low leakage I/O s (+-20nA)
- Up to 48 I/O s with interrupt and wake-up capability

The Launchpad development board integrates an onboard debug probe. This eliminates the need for expensive programmers. It is a simpler low-cost debug probe that supports nearly all TI ARM device derivatives. The on-board circuitry on the XDS110 debug probe was used to measure the energy consumption of the system. The hardware circuitry provides high-accuracy energy consumption with a low bandwidth current and power readings. The energy profiling range covers from 1-µA to 75-mA current draw, above which the tool displays an overcurrent alert and shutdown. The sampling time for this tool is about 500µsec. The energy trace feature can be easily accessed through Code Composer Studio, the software that was used for implementing and debugging the program. Most of the energy measurement are done using this feature. More details about the MSP432 Launchpad is included in Appendix A.

2.1.2 ESP8266

The ESP8266 is a low-cost Wi-Fi Module with full TCP/IP protocol stack and has a microcontroller capability [8][9]. The ESP8266 is capable of hosting an application or offloading all Wi-Fi networking functions from another application processor.

Figure 2: ESP8266 breakout board

Each ESP8266 module comes pre-programmed with an AT command set firmware, meaning it can be interfaced to a microcontroller and connected to the Internet [8]. Although it has a powerful onboard processing capability which allows to integrate it to different sensors and other devices through its interface pins, the device was used merely to provide the MSP432 with wireless access to the Internet.

For this thesis, ESP8266 was used for adding Wi-Fi functionality to the MSP432 via a UART serial connection. The ESP8266 was configured to work on station mode enabling the module to be connected to a Wi-Fi network. The sensor readings were transmitted to a workstation wirelessly via TCP/IP Client-Server module. Client-Server computing is a model in which client and server computers communicate with each other in a network. A server takes request from a client computer and shares its resources and a client is a computing device that initiates contact with a server to make use of sharable resource. The ESP8266 acts as a client getting connected to an access point (router) with access to the Internet as shown in Figure 3. The HTTP request sent from the MSP432 to receive/transmit data from the Internet is handled by the TCP/IP network to the server to post sensor readings.

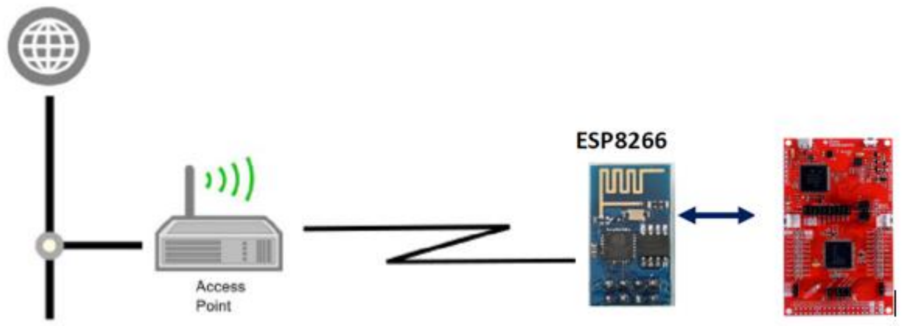


Figure 3: Communication path between MSP432 to the Internet via ESP8266

2.1.3 BME280

BME280 is a module that integrates three precision sensors from Bosch. A breakout board that includes this module makes it easy to interface with a microcontroller. The BME280 is a combined digital humidity, pressure and temperature sensor designed for smartphone applications [10]. The precision sensor from Bosch is the best low-cost sensing solution for measuring humidity with ±3% accuracy, barometric pressure with ±1hPa absolute accuracy, and temperature with ±1.0°C accuracy [11]. Its small dimensions and low power consumption allow the implementation of battery-driven devices such as smartphones, GPS modules or watches. Figure 4 shows the BME280 sensor on a small PCB.

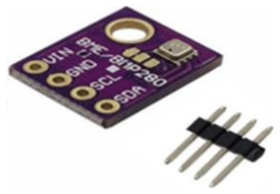


Figure 4: BME280 sensor on a breakout board

BME sensor supports 3 different modes of operations: sleep mode, forced mode and normal mode. For weather monitoring applications forced mode is recommended in the datasheet. In the forced mode, a single measurement is performed, and the sensor goes to sleep. To get the measurement result, the data registers can be read. Figure 5 shows the timing diagram for the forced mode in BME280.

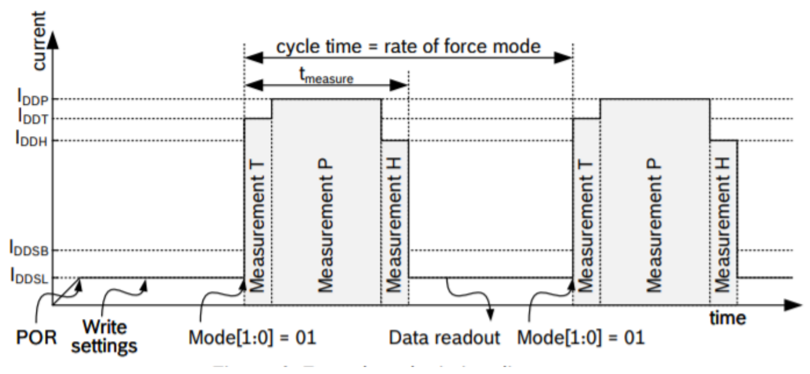


Figure 5: Forced mode timing diagram [11]

The measurement period of BME280 consists of temperature, pressure and humidity readings with selectable oversampling. The temperature and pressure values are passed through an optional IIR filter which removes the short-term fluctuations in pressure.

The data was read using the burst read which resulted in an uncompensated value of readings. The BME280 API, available from Bosch Sensortec, was used to compensate the values. The data from the BME was read using I^2C communication. The I^2C interface consists of the serial clock (SCL) and serial data (SDA) lines. Both lines must be connected to V_{CC} through a pull-up resistor. The current master generates the clock signal.

2.1.4 ST7735 DISPLAY

The ST7735 is a 1.8" diagonal TFT display. It has a microSD interface for storing files and images. It uses a 4-wire SPI to communicate and has its own pixel-addressable frame buffer [12]. The ST7735 can display full 18-bit, 128*160 color pixels. Figure 6 shows the ST7735 display. It includes a transistor Q1 in the breakout board, which is a built-in-driver transistor for backlight. This transistor was used as a switch to control the backlight of the LCD, which helped in turning off the LCD when not required and eventually saving power.

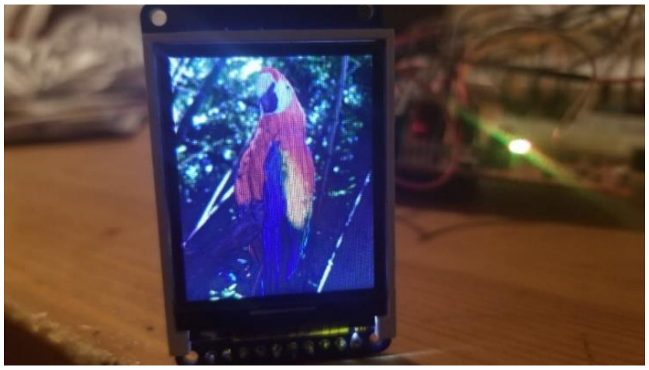


Figure 6: ST7735 display

The main purpose for using the display was to display the real-time readings of the sensors. The breakout board also includes SD card slot for storing the data. The data is being stored in a Google Spreadsheets in a continuous interval, so the SD card feature was not explored. The SD card could be helpful in storing the data in cases when the Wi-Fi connection is lost in case of a low battery. It can be used as an alternative way of storing the data. A synchronous serial communication protocol (SPI) was used for sending data to the display.

2.1.5 LTC3106

The LTC3106 is a low voltage buck-boost DC/DC converter with automatic Power Path management optimized for multisource, low power systems [13]. It can be powered from 2 sources, and when the primary source is unavailable, it switches to the backup power source. It is compatible with either rechargeable or primary cell batteries and can charge the battery whenever there is an energy surplus available. The output voltage is programmed digitally, reducing the required number of external components. It draws 1.6 µA current at no load.

For this research, LTC3106 is used for managing power to the system. A solar panel is connected as a primary source and a Lithium-ion rechargeable battery as a secondary source. The MSP432 board requires 3.3 Volt power supply which was powered by 2 Volt solar panel and 3.7

Volt Li-ion rechargeable battery. More details about different pin connection of LTC3106 is included in Appendix B.

2.1.6 Solar Panels

Solar panels are devices that convert light into electricity. A collection of solar cells makes a solar panel. Many small solar cells spread over a large area can work together to provide enough power to be useful. The more light that strikes a cell, the more electricity it produces.

Solar cell is basically a p-n junction diode but constructed a little bit differently than conventional p-n junction diode. When a semiconductor is exposed to light it can exhibit photovoltaic effect, thus are also called photovoltaics (PV) cells. Regardless of size, a typical solar cell produces about 0.5-0.6-volt DC under open-circuit, no load conditions [14]. The power output of the solar cell depends on its efficiency and size (surface area) and is proportional to the intensity of sunlight striking the surface of a cell [14].

Photovoltaic or solar cells are connected electrically to each other in series to produce higher voltages or in parallel to produce higher currents. Photovoltaic module consists of PV cell circuits sealed in an environmentally protective laminate and are the fundamental building blocks of the PV systems. A photovoltaic panel includes one or more PV modules assembled as a pre-wired, field-installable unit. A solar array is the complete power-generating unit, consisting of any number of solar module and panels. This cycle can be understood more clearly from Figure 7.

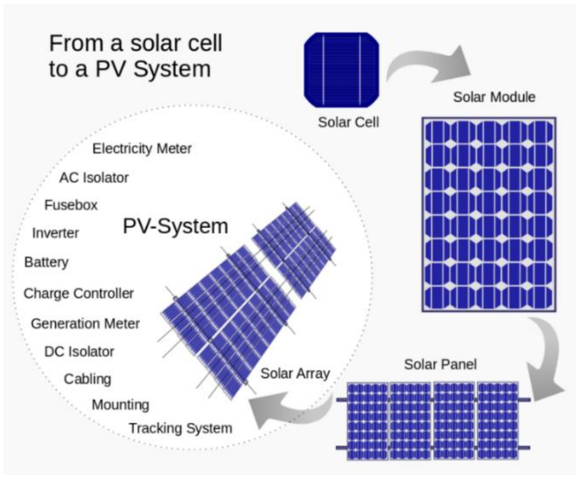


Figure 7: PV System [15]

The performance of these solar modules and arrays are rated according to their maximum power output in watts under Standard Test Conditions (STC). Standard Test Conditions are defined by a module (cell) operating temperature of 25° C (77°F), incident solar irradiance level of 1000 W/m^2, and under Air Mass 1.5 spectral distribution [16]. However, the actual performance of the solar modules and arrays is usually 85 to 90 percent of STC rating. This is because the conditions are not always typical of how these modules operate in the field. The photovoltaic modules available today are extremely safe and reliable products with minimal failure rates. The projected service life of these modules are about 20 to 30 years. For this research, solar energy was used as a primary source to power the system. The excess energy coming from the solar cells was used to charge the battery. For this system, a 2 Volt, 160 mA [17] solar panel was chosen to power the system.

2.1.7 Batteries

An electrical battery is a combination of two or more electrochemical cells used to convert stored chemical energy into electrical energy [18]. Batteries are a common power source for many households and industrial applications. The basic unit inside a battery is called a cell and consists of three main components. There are two electrodes (electric terminal) and a chemical called an electrolyte between them. The difference between a battery and a cell is simply that two or more cells are hooked up in a battery, so their power adds together.

The Li-ion rechargeable battery was selected for the system which requires 100 mA currents. The Li-ion rechargeable batteries are commonly available in 4.2 Volts and was available in different current ratings. Just a single Li-ion battery could supply the required voltage and current. Thus, it was an optimum choice for the system.

Software Description

Code Composer Studio was the Integrated Development Environment (IDE) used for this project. The code was written in C programming language. The compiler version used was TI v18.1.1.LTS. The energy optimizing features of CCS is detailed below.

Energy Trace Technology

Energy Trace Technology is an energy-based code analysis tool that measures and displays the application's energy profile and helps to optimize it for ultra-low power applications [19]. MSP432 devices include built-in Energy Trace technology allows real-time monitoring of many internal device states while user program code executes. They enable analog energy measurement to determine the energy consumption of an application. The energy trace technology is available for all MSP432 devices with selected debuggers, including the Code Composer Studio IDE.

Debuggers with Energy Trace Technology support include a unique way to continuously measure the energy supplied to a target microcontroller that differs considerably from a well-known method of amplifying and sampling the voltage drop over a shunt resistor. A software-controlled dc-dc converter is used to generate the target power supply. A built-in-on-the-fly calibration circuit defines the energy equivalent of a single dc-dc charge pulse. The figure below shows the energy measurement principle. Periods with a small number of charge pulses per time unit indicate low energy consumption and thus low current flow. Periods with a high number of charge pulses per time unit indicate high energy consumption and high current consumption. Each charge pulse leads to a rise of the output voltage V_{OUT}, which results in an unavoidable voltage ripple typical to all dc-dc converters [20].

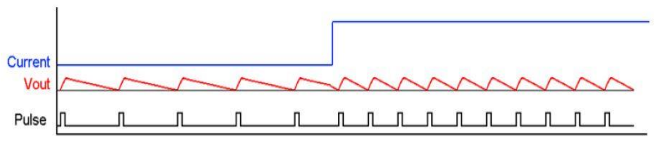


Figure 8: Pulse density and current flow

By sampling continuously even the shortest device activity that consumes energy contributes to the overall recorded energy.

ULP Advisor

ULP (Ultra-Low Power) Advisor is a tool for guiding developers to write efficient code to fully utilize the ultra-low power features of MSP microcontrollers. The ULP advisor checks the code against a thorough ULP checklist to reduce the current consumption as much as possible [21]. At build time, ULP Advisor provides notification and remarks to highlight areas of the code that can be optimized further for reducing the power consumption. The integration of ULP Advisor with the Energy Trace is helpful in the optimization process. Figure 9 demonstrates the integration of ULP Advisor and Energy Trace Technology in the development process.

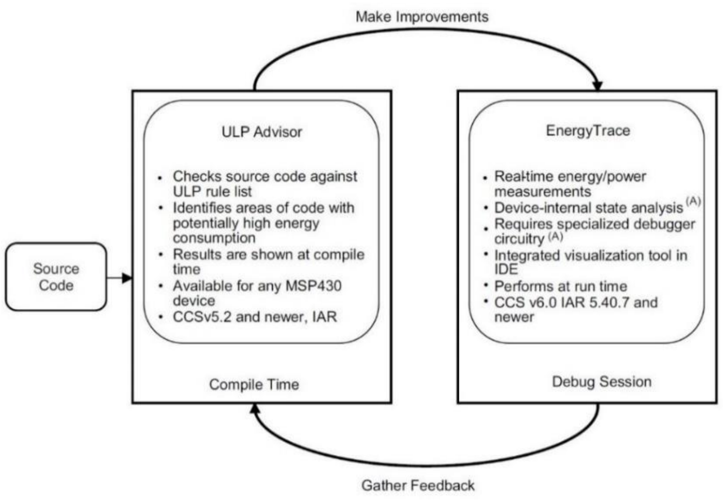
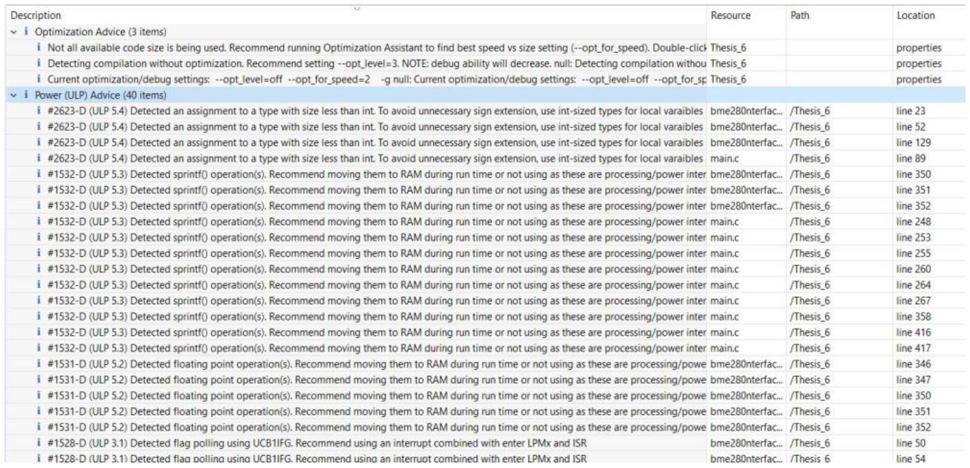

Make Improvements
Source Code
ULP Advisor
• Checks source code against ULP rule list
• Identifies areas of code with potentially high energy consumption
• Results are shown at compile time
• Available for any MSP430 device
• CCSv5.2 and newer, IAR
Compile Time
EnergyTrace
• Realtime energy/power measurements
• Device-internal state analysis (A)
• Requires specialized debugger circuitry (A)
• Integrated visualization tool in IDE
• Performs at run time
• CCS v6.0 IAR 5.40.7 and newer
Debug Session
Gather Feedback

Figure 9: ULP Advisor and Energy Trace Technology [22]

There were a few optimizations done in the program following the ULP Advisor guidance. Some of them were changing the type of the variables to a smaller size, setting the uninitialized ports as outputs. Figure 10 is the advice given by the ULP while running the code.



Description	Resource	Path	Location
Optimization Advice (3 items)			
Not all available code size is being used. Recommend running Optimization Assistant to find best speed vs size setting (--opt_for_speed). Double-click	Thesis_6		properties
Detecting compilation without optimization. Recommend setting --opt_level=3. NOTE: debug ability will decrease. null: Detecting compilation withou	Thesis_6		properties
Current optimization/debug settings: --opt_level=off --opt_for_speed=2 -g null: Current optimization/debug settings: --opt_level=off --opt_for_sp	Thesis_6		properties
Power (ULP) Advice (40 items)			
#2623-D (ULP 5.4) Detected an assignment to a type with size less than int. To avoid unnecessary sign extension, use int-sized types for local varaibles	bme280nterfac...	/Thesis_6	line 23
#2623-D (ULP 5.4) Detected an assignment to a type with size less than int. To avoid unnecessary sign extension, use int-sized types for local varaibles	bme280nterfac...	/Thesis_6	line 52
#2623-D (ULP 5.4) Detected an assignment to a type with size less than int. To avoid unnecessary sign extension, use int-sized types for local varaibles	bme280nterfac...	/Thesis_6	line 129
#2623-D (ULP 5.4) Detected an assignment to a type with size less than int. To avoid unnecessary sign extension, use int-sized types for local varaibles	main.c	/Thesis_6	line 89
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	bme280nterfac...	/Thesis_6	line 350
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	bme280nterfac...	/Thesis_6	line 351
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	bme280nterfac...	/Thesis_6	line 352
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	main.c	/Thesis_6	line 248
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	main.c	/Thesis_6	line 253
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	main.c	/Thesis_6	line 255
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	main.c	/Thesis_6	line 260
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	main.c	/Thesis_6	line 264
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	main.c	/Thesis_6	line 267
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	main.c	/Thesis_6	line 358
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	main.c	/Thesis_6	line 416
#1532-D (ULP 5.3) Detected sprintf() operation(s). Recommend moving them to RAM during run time or not using as these are processing/power inter	main.c	/Thesis_6	line 417
#1531-D (ULP 5.2) Detected floating point operation(s). Recommend moving them to RAM during run time or not using as these are processing/powe	bme280nterfac...	/Thesis_6	line 346
#1531-D (ULP 5.2) Detected floating point operation(s). Recommend moving them to RAM during run time or not using as these are processing/powe	bme280nterfac...	/Thesis_6	line 347
#1531-D (ULP 5.2) Detected floating point operation(s). Recommend moving them to RAM during run time or not using as these are processing/powe	bme280nterfac...	/Thesis_6	line 350
#1531-D (ULP 5.2) Detected floating point operation(s). Recommend moving them to RAM during run time or not using as these are processing/powe	bme280nterfac...	/Thesis_6	line 351
#1531-D (ULP 5.2) Detected floating point operation(s). Recommend moving them to RAM during run time or not using as these are processing/powe	bme280nterfac...	/Thesis_6	line 352
#1528-D (ULP 3.1) Detected flag polling using UCB1IFG. Recommend using an interrupt combined with enter LPMx and ISR	bme280nterfac...	/Thesis_6	line 50
#1528-D (ULP 3.1) Detected flag polling using UCB1IFG. Recommend using an interrupt combined with enter LPMx and ISR	bme280nterfac...	/Thesis 6	line 54

Figure 10: Advice given by ULP Advisor for optimization

Some of the common issues the ULP Advisor found were uninitialized GPIO ports, improper initializations of functions, type of the variables.

3. Optimization Techniques

MSP432 Optimizations

With growing complexity in the microcontroller (MCU) applications, minimizing the overall energy consumption of a system is one of the most challenging problems [23]. Multiple aspects, such as the hardware components used onboard and the application software, must be considered [23]. Some obvious generic technique, such as reducing the frequency, might not significantly reduce the energy consumption independently but taken as a whole, the result might be significant, as there are many interdependencies across these components [23]. The MSP432 microcontroller includes several power enhancements features to reduce the overall power consumption. The device provides various options and power configurations that enable developers to optimize the power consumption for a specific application.

3.1.1 Reducing operating voltage

Power is the product of current and voltage. By reducing the supply voltage in certain applications, power consumption can be reduced. The constraint here is that the minimum voltage requirement is met.

3.1.2 Reducing the operating frequency

Power and current consumption are directly proportional to the operating frequency. In most cases, higher operating frequency means the CPU can execute the codes and complete the task faster. In a real-time scenario, many applications are time-dependent or event-driven. There are cases when CPU is running faster in an idle loop waiting for a certain event to trigger. The CPU can spend a lot of time waiting for serial data to come in at a lower baud rate. This can consume additional power which can be reduced by reducing the operating frequency.

3.1.3 Maximizing the sleep time

There are two modes generally considered in low power designs- active mode and low power mode. During active mode, it executes the designed tasks. The low power mode is the period where there is minimal activity, other than timekeeping or waiting for an interrupt or event to wake up to the active mode. Low power mode consumes less current compared to active mode. There are different low power modes in the MSP432 microcontroller where current consumption can be as low as 700 nA, while active mode current can be up to several milliamps. Maximizing the low power mode can significantly reduce the power consumption.

3.1.4 Minimizing the transition time

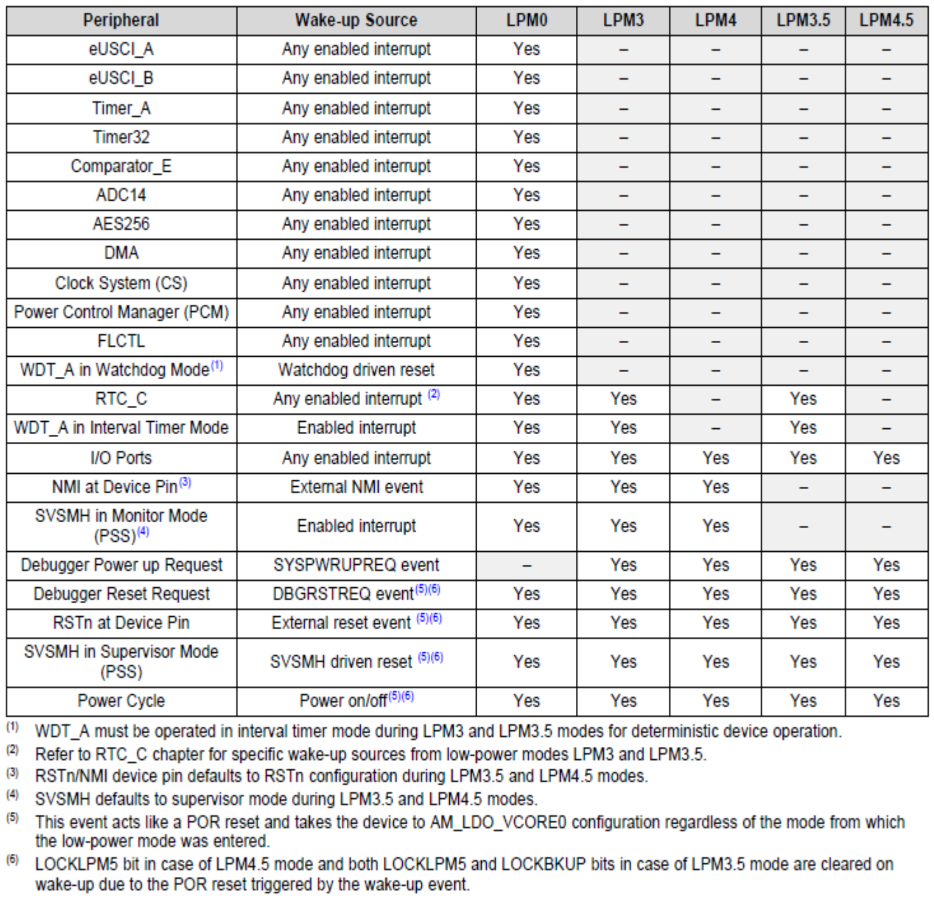
Beside time allocation for active and low-power modes, some applications might unknowingly spend a considerable amount of time transitioning between these power modes. If this transition time goes unnoticed, this might contribute to an increase in the total energy consumption. For power optimization, it is essential to identify the system transitions to determine if they can be reduced or removed.

3.1.5 Solving intermodular dependencies

The previously mentioned vectors are all possible options to optimize individually. These vectors might have some interdependencies for a given microcontroller platform. Optimizing a vector might affect another vector. For example, reducing the frequency of might increase the duty cycle of the system which might lead to more power consumption. Thus, it is even more important to consider the intermodular dependencies and determine optimal conditions of settings for a given platform. When clock was reduced to work on low-frequency mode it did not support the LCD, thus 48MHz clock source was chosen.

Power Modes

The MSP432 device supports several power modes. Active modes are the power modes in which CPU execution is possible. LMP0, LMP3, LPM4 and LPMx.5 modes do not allow the CPU execution. Figure 11 illustrates different peripherals and their wake-up sources for different power modes.



Peripheral	Wake-up Source	LPM0	LPM3	LPM4	LPM3.5	LPM4.5
eUSCI_A	Any enabled interrupt	Yes	–	–	–	–
eUSCI_B	Any enabled interrupt	Yes	–	–	–	–
Timer_A	Any enabled interrupt	Yes	–	–	–	–
Timer32	Any enabled interrupt	Yes	–	–	–	–
Comparator_E	Any enabled interrupt	Yes	–	–	–	–
ADC14	Any enabled interrupt	Yes	–	–	–	–
AES256	Any enabled interrupt	Yes	–	–	–	–
DMA	Any enabled interrupt	Yes	–	–	–	–
Clock System (CS)	Any enabled interrupt	Yes	–	–	–	–
Power Control Manager (PCM)	Any enabled interrupt	Yes	–	–	–	–
FLCTL	Any enabled interrupt	Yes	–	–	–	–
WDT_A in Watchdog Mode[(1)]	Watchdog driven reset	Yes	–	–	–	–
RTC_C	Any enabled interrupt [(2)]	Yes	Yes	–	Yes	–
WDT_A in Interval Timer Mode	Enabled interrupt	Yes	Yes	–	Yes	–
I/O Ports	Any enabled interrupt	Yes	Yes	Yes	Yes	Yes
NMI at Device Pin[(3)]	External NMI event	Yes	Yes	Yes	–	–
SVSMH in Monitor Mode (PSS)[(4)]	Enabled interrupt	Yes	Yes	Yes	–	–
Debugger Power up Request	SYSPWRUPREQ event	–	Yes	Yes	Yes	Yes
Debugger Reset Request	DBGRSTREQ event[(5)(6)]	Yes	Yes	Yes	Yes	Yes
RSTn at Device Pin	External reset event [(5)(6)]	Yes	Yes	Yes	Yes	Yes
SVSMH in Supervisor Mode (PSS)	SVSMH driven reset [(5)(6)]	Yes	Yes	Yes	Yes	Yes
Power Cycle	Power on/off[(5)(6)]	Yes	Yes	Yes	Yes	Yes

(1) WDT_A must be operated in interval timer mode during LPM3 and LPM3.5 modes for deterministic device operation.
(2) Refer to RTC_C chapter for specific wake-up sources from low-power modes LPM3 and LPM3.5.
(3) RSTn/NMI device pin defaults to RSTn configuration during LPM3.5 and LPM4.5 modes.
(4) SVSMH defaults to supervisor mode during LPM3.5 and LPM4.5 modes.
(5) This event acts like a POR reset and takes the device to AM_LDO_VCORE0 configuration regardless of the mode from which the low-power mode was entered.
(6) LOCKLPM5 bit in case of LPM4.5 mode and both LOCKLPM5 and LOCKBKUP bits in case of LPM3.5 mode are cleared on wake-up due to the POR reset triggered by the wake-up event.

Figure 11: Wake up sources from Low Power Modes [5]

For this research, the system was running on active mode when the system was working and LPM3 mode when staying idle. In active mode, the display and the BME run perfectly. Initially, the lowest power consumption mode (LPM3.5) was tried to set up when the system was in a standby condition. The system was unable to wake up from the LMP3.5 mode. The example code available in the MSP432 library was tried and it was not running properly. So, the LMP3.5 mode was discarded and LPM3 mode was chosen. It was one step below the lowest power mode. The system ran properly without any problem in this power mode.

The PCM Control 0 Register (PCMCTL0) is the primary mechanism for changing the power modes. The PCMCTL0 contains 2 fields:

- Active Mode Request (AMR): It is used to change from one active mode to another.
- Low Power Mode Request (LPMR): It is used to enter different low power modes such as LPM3, LPM4, LPM3.5, and LPM4.5.

The AMR has the highest priority over the LMPR settings. The low power modes are entered when proper values are written to LPMR regardless of AMR settings. The Current Power Mode (CPM) bits that reside in the PCMCTL0 registers are read-only and reflects the current power mode of the system. CPM is updated when the power mode request is completed.

4. Design and Implementation of the System

The design process involved 3 sub parts i.e. to build a system, optimize the power consumed and self-sustainable. The readings of the sensor were displayed on the LCD. The block diagram of the major component is shown in Figure 12.

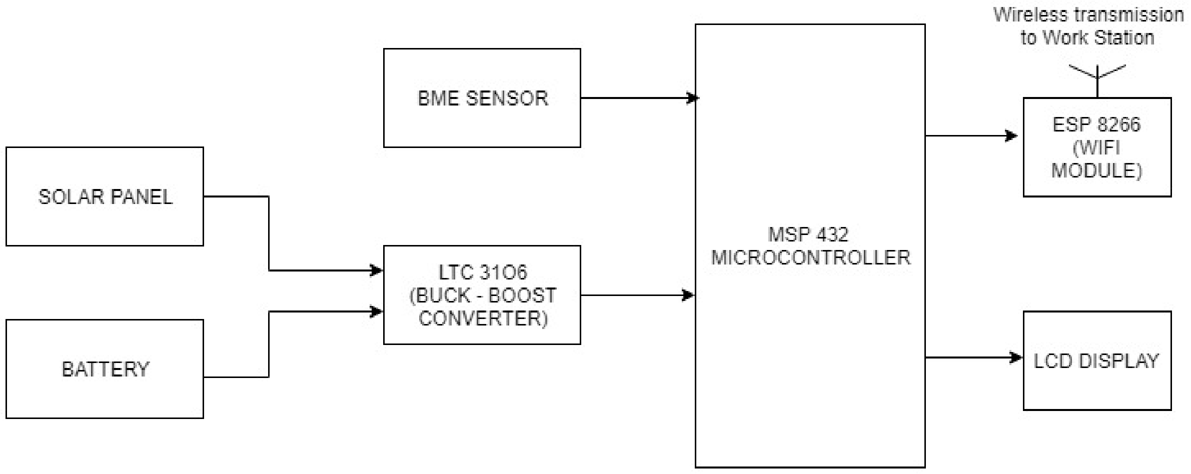


Figure 12: General block diagram of the overall system

The objective of this work is to optimize the overall power consumption dissipated by the system. Different hardware and software optimization techniques were applied to reduce the overall power consumption. After the power optimization process, a buck-boost converter was selected to reduce power using a 2 Volt solar panel and a 3.7 Volt Li-ion rechargeable battery. The LTC3106 IC was used to manage power of the system when there was enough sunlight, as well as in the absence of sunlight, and have the system run continuously without any interruption.

4.1 Reading the information from Sensors and transmitting to the Google Documents

Figure 13 shows the hardware set of the system. The major components used for the initial condition were MSP432 microcontroller, BME 280 sensor, LCD and ESP8266 Wi-Fi module.

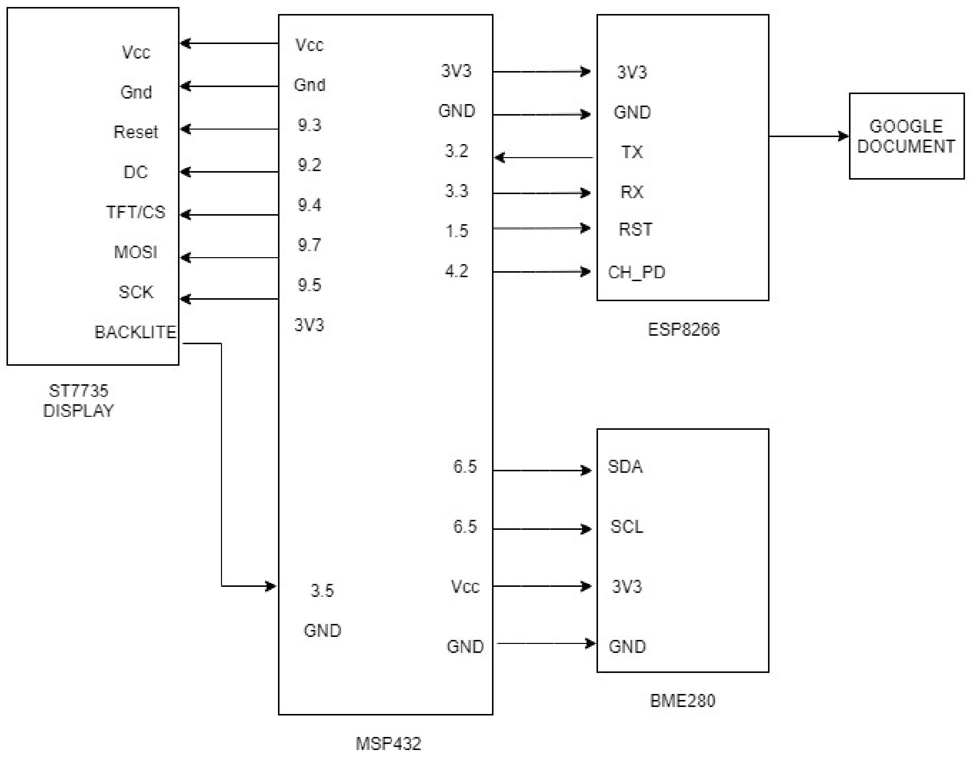


Figure 13: Block diagram to read value from sensor, display and post data in Spreadsheet

4.1.1 Connecting MSP432 to BME sensor

The first step in the application was to read the values from the BME sensor. The BME280 communicated with MSP432 using the I^2C protocol. The eUSCI module was enabled for the MSP432. The baud rate for the eUSCI was set to 100KHz meaning that the rate at which bits are sent are synchronized with a 100KHz clock signal in the MSP432. GPIO pins 6.4 and 6.5 were used for communication between MSP432 and the BME280 sensor beside the supply and Ground. The pin 6.4 could read I^2C SDA channel and the pin 6.5 was able to read the I^2C SCL channel.

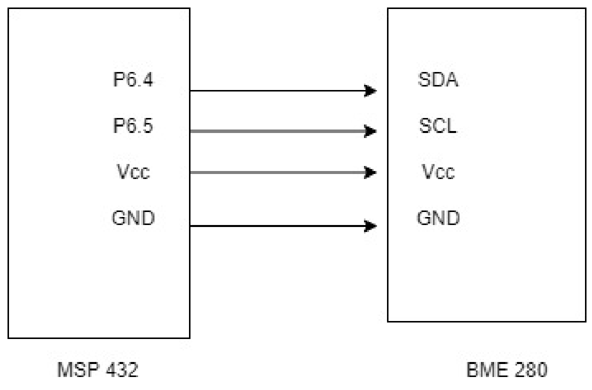


Figure 14: Block diagram of MSP432 connection with BME280

For reading the data stored in the register of the BME280, the eUSCI port on the MSP432 was configured to communicate with an I^2C slave device. A block diagram of the MSP432 serial port is shown in Figure 15. The clock source was selected to create the signal that drives the I^2C clock line in UCxSCL. The slave address was loaded into the Slave Address buffer that was automatically loaded into the Transmit Shift Register and shifted out the I^2C SDA line following the start bit. Once the flag was set indicating the slave address and read/write bit had been sent, the data was loaded to the Transmit Buffer, automatically transferred into the shift register, and put on the SDA line bit by bit in synchrony with the SCL transitions. The shift register output drive the MOSFET that pulled down the SDA line. The clock was pulled down by another MOSFET driven by the clock generator.

To read the data stored in a register on the BME280 IC, the I^2C interface first sends the device address and the write bit. Then when the BME280 acknowledges its address, the MSP432 master sends the register address to it. After the slave acknowledges, the MSP432 master then sends a stop signal, switches to read mode and puts the device address on the bus along with the read/write bit set to read. The BME280 then puts the contents of its byte wide register located at the register address on the bus. The MSP432 I^2C bus master collects the bits in the receive shift register and transfers them to the receive buffer. When filled, the receive buffer sets a flag that is monitored by the MSP432 that then releases the bus and stores the value. It was now ready for the next transaction. This process can be seen in Figure 15.

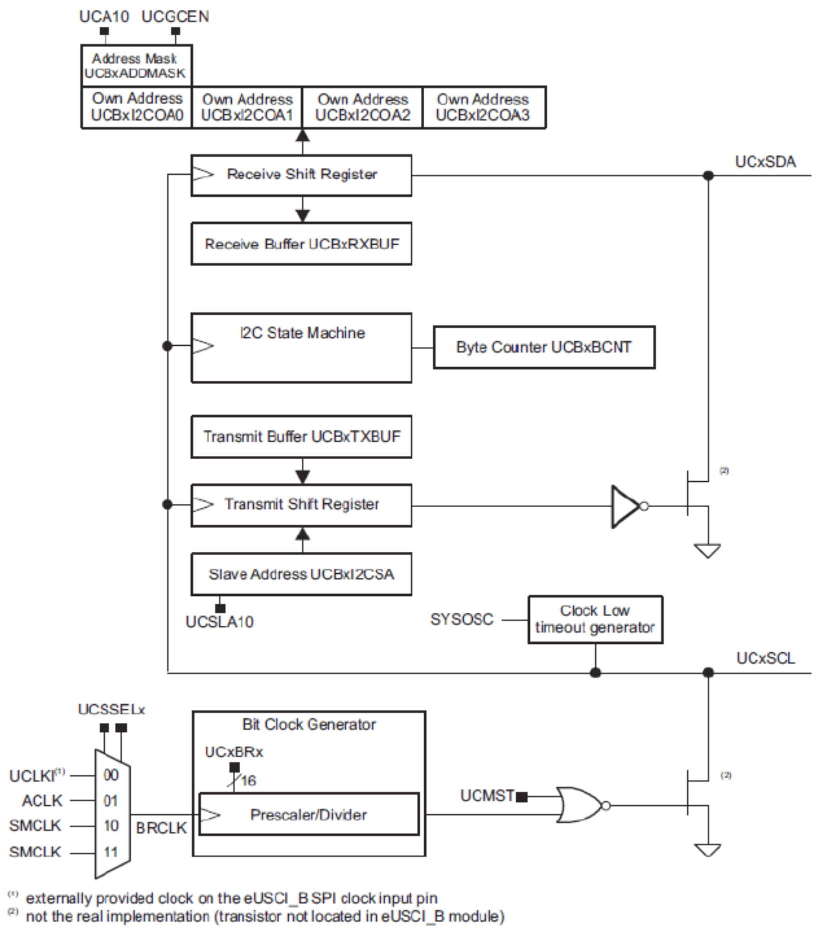


Figure 15: MSP432 eUSCI in I^{2}C mode

4.1.2 Connecting LCD to MSP 432

The MSP432 microcontroller was connected to the LCD using the SPI interface. The LCD was used to display the values from the sensors, and the date and time in the RTC clock. The figure below shows the pin connection of the MSP432 and the ST7735 display. For the SPI communication SCK was connected to P9.5 (UCA3CLK), MOSI to P9.7(UCA3SIMO), TFT_CS to P9.4 (UCA3STE) respectively. The reset, backlight, and DC(Data/Command) signal was provided using GPIOs. The baud rate was set to 4 MHz which means that the rate at which bits are sent are synchronized with a 4 MHz clock signal originating in the MSP432.

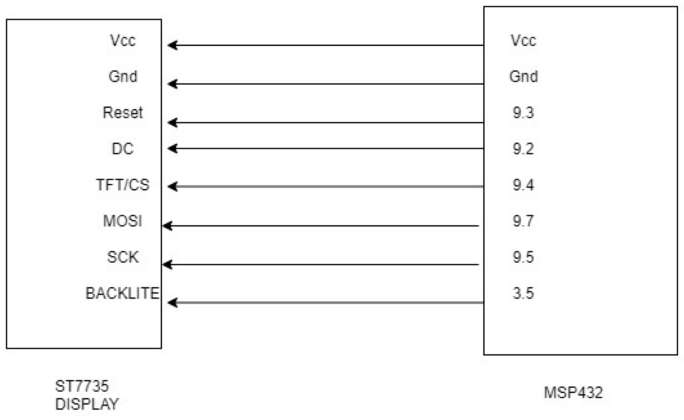


Figure 16: Block diagram for connection of ST7735 display with MSP432

The interface between the LCD module and the MSP432 used a synchronous serial communication protocol (SPI). It was implemented by configuring the eUSCI port on the MSP432 for sending data in synchrony with the clock signal to enable the LCD module to receive data. In addition, the MSP432 was configured to reset the LCD module upon startup to be configured to received data or commands. To display color pixels on the screen, the 16-bit data was sent two bytes at a time per pixel with the color information for each pixel encoded in red, green and blue intensity. A preceding command points to a memory location in the LCD module where the following pixel information was sent as data is to be stored incrementally in RAM memory on the LCD module. The LCD module electronics then controlled the color and intensity of pixels from information encoded in the bytes stored in its RAM memory space. Pixel location on the LCD was mapped directly from the pixel location in RAM. The reading from the sensor i.e. the temperature, pressure and the humidity value was displayed on the LCD. A GPIO pin was used to control a transistor switch that in turn-controlled power to the backlight of the display. When the GPIO pin was high the display was on and when it was low the display was off.

4.1.3 Connecting ESP8266 to MSP432

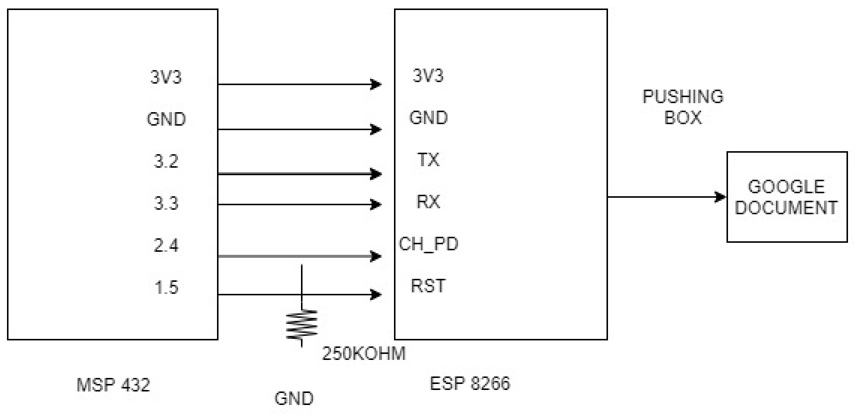


Figure 17: Block diagram for MSP432 connection with ESP8266

The ESP8266 was connected to MSP432 in UART mode, as shown in Figure 17. UART A2 serial port was used to communicate between the MSP432 and ESP8266. The GPIO pins of the MSP432 controlled the CH_PD and RST pins. The CH_PD pin was used to power down the ESP8266 when it was not transmitting and a pull-down resistor of 250KOHM was used so that the pin was not left floating. The RST pin was used as a reset for the ESP8266. AT commands were sent from the code for the MSP432 to communicate with the ESP8266. They are standard command sets developed to control modems. AT stands for attention. The ESP8266 was used as a station mode to connect to the microcontroller to the Internet and then send the reading values of the sensor to the Google Spreadsheet.

To send values of the sensor readings to the Google spreadsheet pushingbox.com API was used. The pushingbox.com API acted as an intermediate in posting the data to a Google spreadsheet. Initially, a google spreadsheet was accessed through an API URL. For ESP8266

URL redirection was not a straightforward process. URL redirection is a technique that sending a user from one URL to another. The ESP8266 needs to correctly decode the header information received from the first server to extract the redirect URL and make a second GET request to the new server. An HTTP redirect was created at pushingbox.com and a TCP request was sent with the sensor readings. The pushingbox scenario directs the necessary request to the Google spreadsheet API and formats it as an HTTPS for security reasons.

Table 1 shows the data that was sent to the Google spreadsheet. For testing purposes, the values were transmitted every 3 minutes.

Table 1: Values being posted on Google Spreadsheet

Date	**Humidity(%rh)**	**Temperature(ºC)**	**Pressure (inHg)**
10/12/2018 12:36:44	49.11	19.57	30.05
10/12/2018 12:39:39	49.21	19.79	30.05
10/12/2018 12:42:42	47.89	19.67	30.03
10/12/2018 12:45:49	47.56	19.7	30.03
10/12/2018 12:48:44	47.69	19.65	30.04
10/12/2018 12:51:52	46.27	20.57	29.97
10/12/2018 12:54:50	45.96	20.65	29.97
10/12/2018 12:57:50	45.96	20.65	29.97
10/12/2018 13:00:45	45.02	20.96	29.97
10/12/2018 13:03:45	45.29	21.01	29.97
10/13/2018 13:06:33	37.99	20.75	30.04
10/13/2018 13:09:39	38.25	20.81	30.04
10/13/2018 13:12:39	40.24	20.95	30.04
10/13/2018 13:15:07	43.39	21.44	30.05
10/13/2018 13:18:10	46.9	21.54	30.05
10/13/2018 13:21:32	49.13	21.46	30.05
10/13/2018 13:24:03	27.91	22.42	29.98
10/13/2018 13:27:57	27.95	22.52	29.98
10/15/2018 13:30:22	41.14	24.01	30.18

After designing a proper working system, the current consumed was measured using a BK Precision 1761 DC Power Supply [24]. The minimum current required to run the system was 105 mA. It must be realized that at this stage all components were drawing current from the power supply.

4.1.4 Software Flow

Figure 18 shows the flow chart of the system.

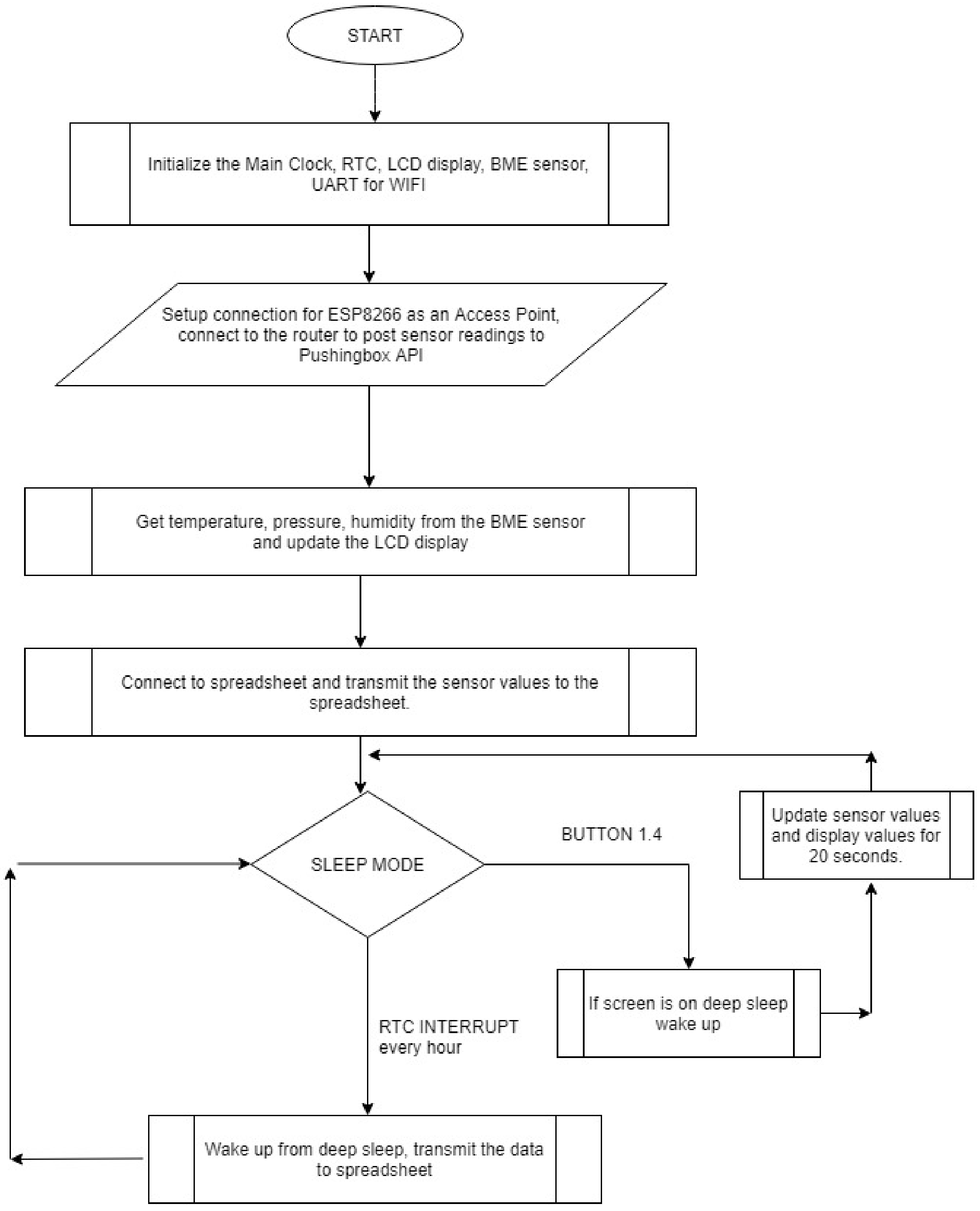
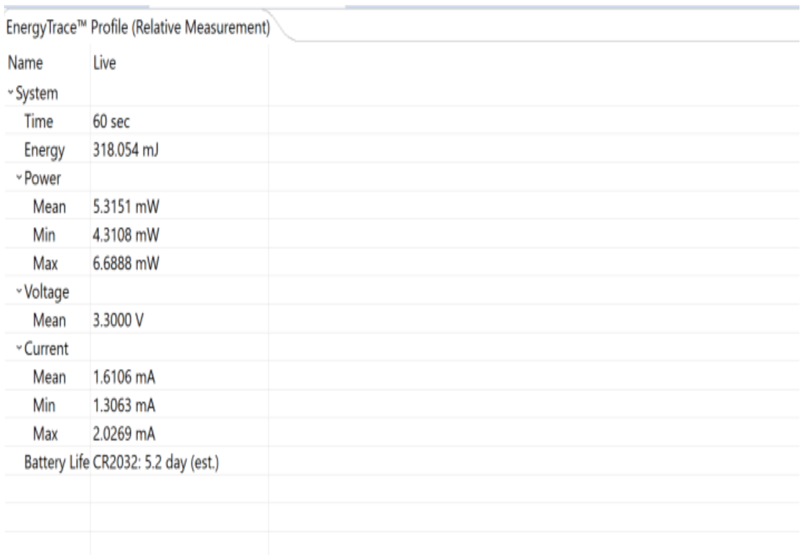

START
Initialize the Main Clock, RTC, LCD display, BME sensor, UART for WIFI
Setup connection for ESP8266 as an Access Point, connect to the router to post sensor readings to Pushingbox API
Get temperature, pressure, humidity from the BME sensor and update the LCD display
Connect to spreadsheet and transmit the sensor values to the spreadsheet.
SLEEP MODE
BUTTON 1.4
Update sensor values and display values for 20 seconds.
If screen is on deep sleep wake up
RTC INTERRUPT every hour
Wake up from deep sleep, transmit the data to spreadsheet

Figure 18: Software flow for the main program

4.2 Optimization in the Power consumed by the system

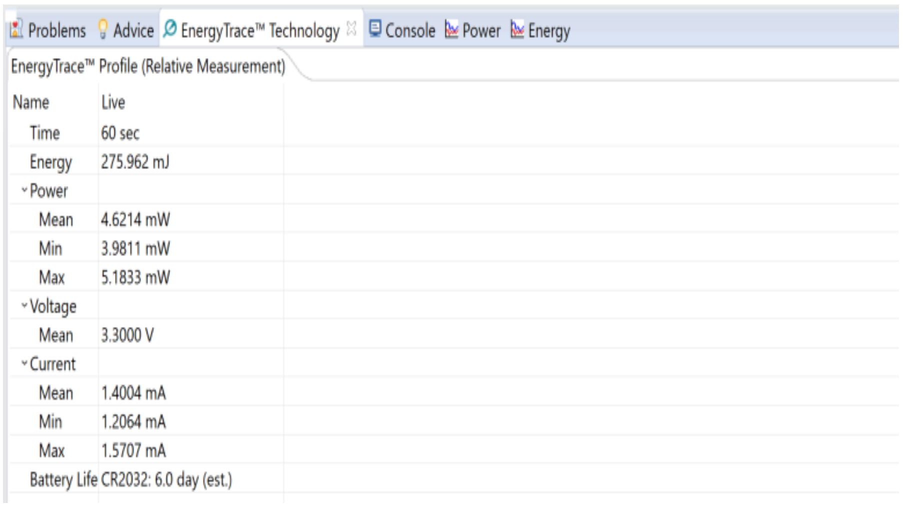
Initially, without any optimization technique, the minimum current required to run the system was around 105 mA when all components were turned on and functioning. The MSP432 power consumption was measured as a standalone device without any connectivity. Initially, the current consumption was about 1.6 mA.


EnergyTrace™ Profile (Relative Measurement)

Name	Live
˅System	
Time	60 sec
Energy	318.054 mJ
˅Power	
Mean	5.3151 mW
Min	4.3108 mW
Max	6.6888 mW
˅Voltage	
Mean	3.3000 V
˅Current	
Mean	1.6106 mA
Min	1.3063 mA
Max	2.0269 mA
Battery Life	CR2032: 5.2 day (est.)

Figure 19: Energy consumed by the MSP432 when nothing was connected

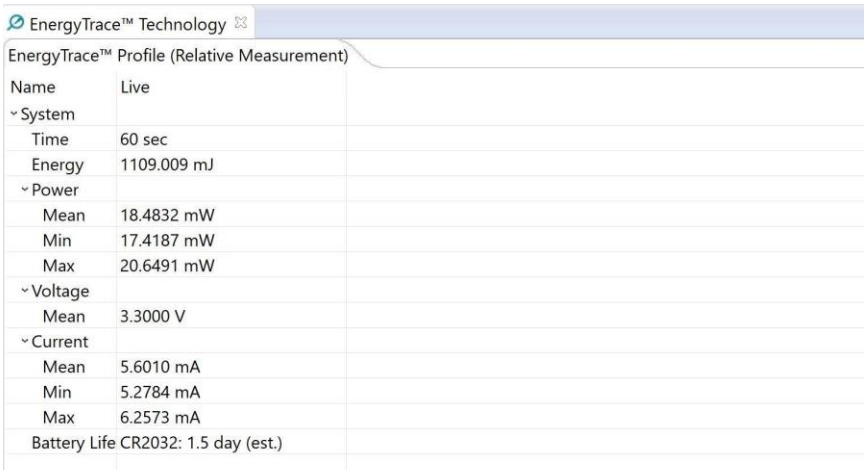
The first step in the optimization was to reduce the power consumed by the unused I/O port pins. This was carried out by setting all the pins as output. To prevent the floating input and to reduce the power consumption, the unused I/O pins should be configured as I/O function, output direction and left unconnected on the PC board. There was a slight reduction in the power consumed by noise on the input pins due to rapid switching as shown in Figure 20.



EnergyTrace™ Profile (Relative Measurement)	
Name	Live
Time	60 sec
Energy	275.962 mJ
Power	
Mean	4.6214 mW
Min	3.9811 mW
Max	5.1833 mW
Voltage	
Mean	3.3000 V
Current	
Mean	1.4004 mA
Min	1.2064 mA
Max	1.5707 mA
Battery Life	CR2032: 6.0 day (est.)

Figure 20: Energy consumed by the MSP432 when setting all unused pins as output

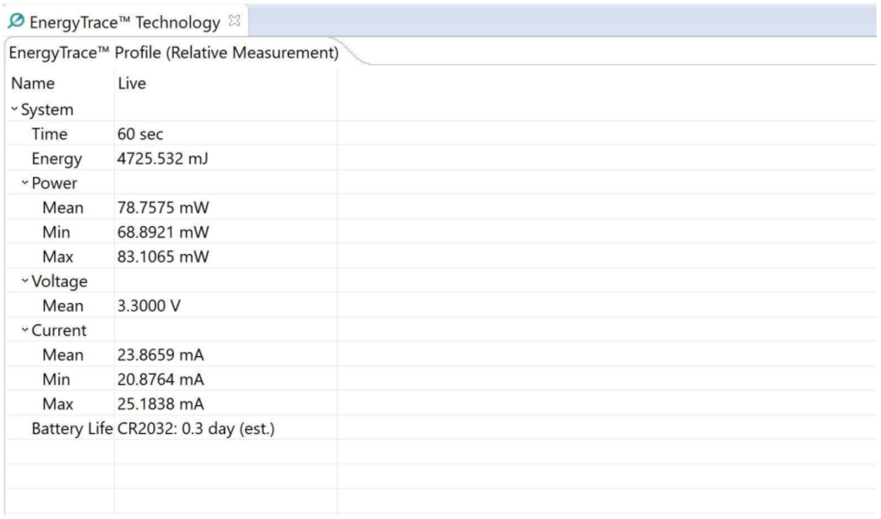
Next the BME sensor was added to the system. The I^{2}C communication required a clock. The current consumption of the system when the BME was connected increased to 5.6 mA. Here, 48MHz external high-frequency crystal oscillator was used to set up the clock. There was no change in the current consumption when the internal DCO was used as the source for the clock. The SMCLK was set to 12MHz. The readings can be seen in Figure 21.



EnergyTrace™ Profile (Relative Measurement)	
Name	Live
System	
Time	60 sec
Energy	1109.009 mJ
Power	
Mean	18.4832 mW
Min	17.4187 mW
Max	20.6491 mW
Voltage	
Mean	3.3000 V
Current	
Mean	5.6010 mA
Min	5.2784 mA
Max	6.2573 mA
Battery Life	CR2032: 1.5 day (est.)

Figure 21: Energy consumed by the MSP432 when BME is connected in the system

The next addition to the system was the display. The LCD was added to display the readings from the BME sensor. The LCD worked on SPI communication and it required a high clock to display 128*160 color pixels. The LCD refresh was the limiting factor on how slowly the microcontroller clock could operate as it was clocking the LCD controller IC. To generate an RGB color 3 segments are required. These 3 segments individually pass light through a red, green and a blue filter to make a group of segments or an RGB pixel, i.e., 128*3=384 segments (columns) and 160 rows are required. Displays can drive 3 segments (1 pixel) per clock cycle. Thus, the 48MHz external high-frequency crystal was used to set up the clock. When the LCD was introduced to display the readings from the sensor the system consumed around 24 mA of current.



Name	Live
System	
Time	60 sec
Energy	4725.532 mJ
Power	
Mean	78.7575 mW
Min	68.8921 mW
Max	83.1065 mW
Voltage	
Mean	3.3000 V
Current	
Mean	23.8659 mA
Min	20.8764 mA
Max	25.1838 mA
Battery Life	CR2032: 0.3 day (est.)

Figure 22: Energy consumed when LCD is introduced to the system

For reducing the current consumption, the backlight of the LCD was controlled by a GPIO pin of the MSP432. The LCD backlight was controlled from the interrupt of the push button in the microcontroller. The push button was used for turning on and off the LCD backlight.

The graph below represents the power consumption by the system when the backlight of LCD was turned on and turned off. Initially when the program started the backlight of the LCD was on and it consumed around 24 mA (75mW) power. When the backlight of display was turned off from the push button the power consumption reduced to around 8.81 mA (27mW). By controlling the backlight of the display, the current consumption can be reduced drastically as seen in Figure 23.

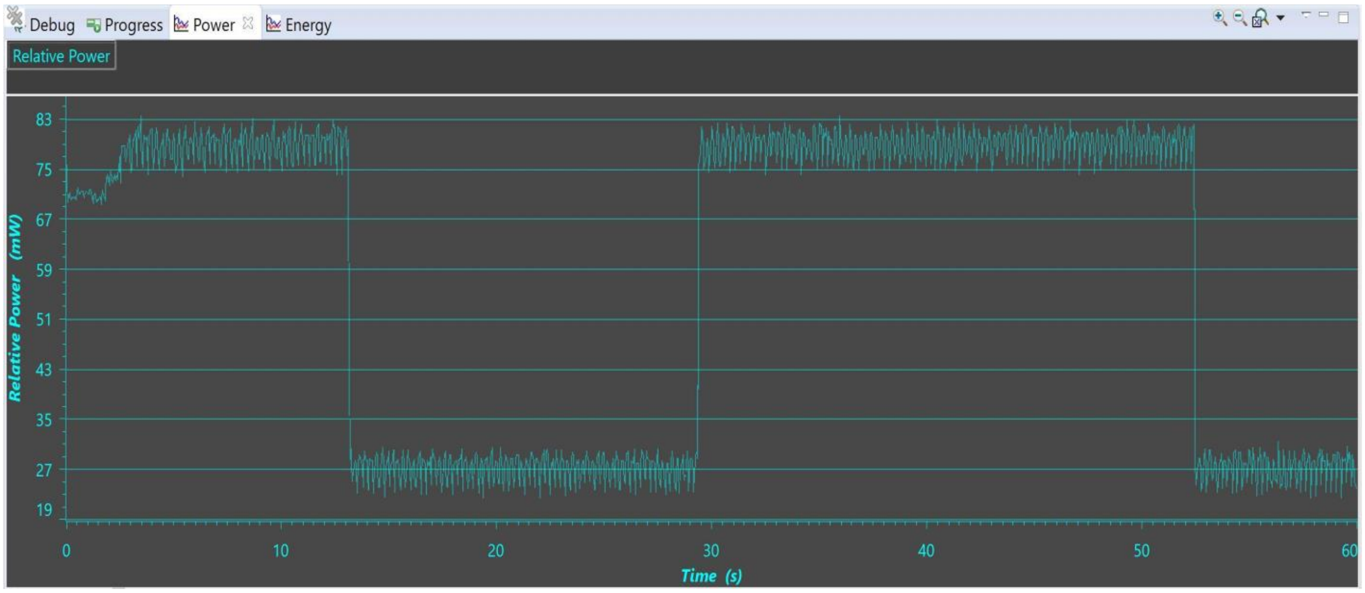
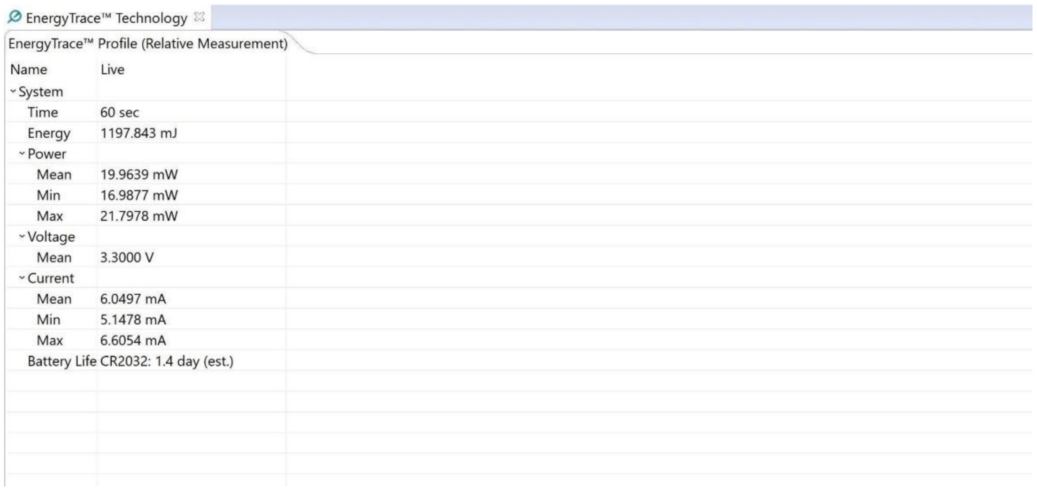


Figure 23: Relative power consumption by the system when the backlight is on and off

When the backlight of the LCD was off, display off command was sent to the LCD to put the display register off. The current consumption reduced to around 6 mA when display off command was sent. The current readings after sending display off command to LCD is seen in Figure 24.



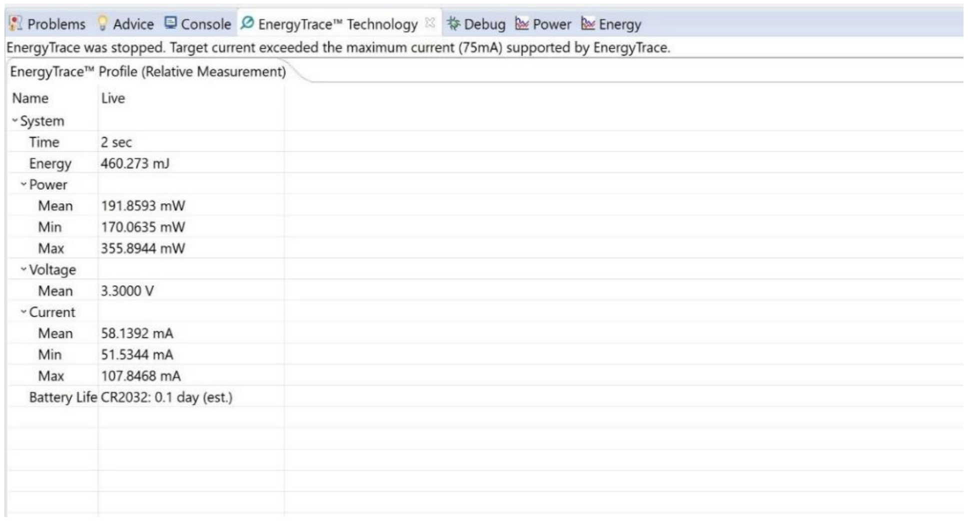
EnergyTrace™ Technology

EnergyTrace™ Profile (Relative Measurement)

Name	Live
System	
Time	60 sec
Energy	1197.843 mJ
Power	
Mean	19.9639 mW
Min	16.9877 mW
Max	21.7978 mW
Voltage	
Mean	3.3000 V
Current	
Mean	6.0497 mA
Min	5.1478 mA
Max	6.6054 mA
Battery Life	CR2032: 1.4 day (est.)

Figure 24: Energy consumed when the LCD was turned off

After integration of the LCD on the system an ESP8266 module was integrated to communicate with the Internet. The objective was to post the value obtained from the sensor to a Google document. The Energy Trace Software was not able to capture the energy consumed when ESP8266 was integrated to the system. The system consumed more than 75 mA current, as the Energy Trace Software stopped showing an error that it cannot measure current above 75 mA.



EnergyTrace was stopped. Target current exceeded the maximum current (75mA) supported by EnergyTrace.

EnergyTrace™ Profile (Relative Measurement)

Name	Live
System	
Time	2 sec
Energy	460.273 mJ
Power	
Mean	191.8593 mW
Min	170.0635 mW
Max	355.8944 mW
Voltage	
Mean	3.3000 V
Current	
Mean	58.1392 mA
Min	51.5344 mA
Max	107.8468 mA
Battery Life	CR2032: 0.1 day (est.)

Figure 25: EnergyTrace stopped

Realizing this condition, the system was powered externally through a DC power supply [24] which had the capability to measure the current consumption. When the current consumption was measured the entire system consumed about 95 mA in the startup. The system consumed 95 mA during the start and then settled to about 26 mA. The current consumption increased when the system was transmitting to the Google Document.

Figure 26 shows the test setup to measure the power consumption by the ESP8266 using a DC power supply. There are 2 readings seen of which only the right-side reading is being used. The power was supplied from the right side of the power source. The reading seen on the right side is the current consumption reading. The left side power supply was not used.

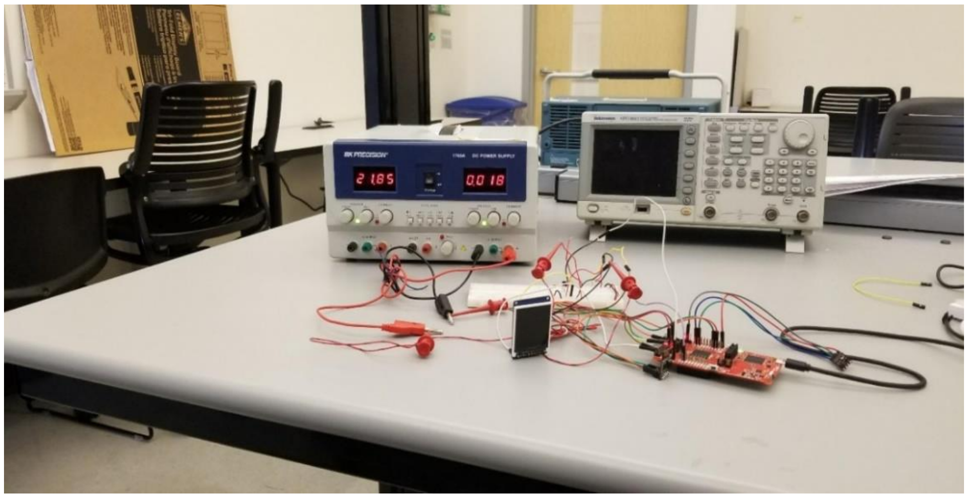


Figure 26: Readings for ESP8266 taken from DC power supply

To reduce the current consumed by the ESP8266, the CH_PD pin was set to power down mode. The MSP432 GPIO pin was used to provide the CH_PD signal to the ESP8266. The CH_PD was high during the transmission of data and was set to low when not needed to transmit data. This reduced the current consumed by the ESP8266 that stays in a “modem-sleep” mode between transmissions by default. The current consumption by the system was reduced to around 6 mA when CH_PD was set to low, and LCD was off.

The ESP8266 current was monitored in more detail to measure the surge current drawn by the ESP8266 when it is powered, as the CH_PD signal goes high. The current consumed by the ESP8266 was monitored when the system was waking up from the power down mode. This would give us a clear idea of how much current is required to start up the system or during wake-up from sleep mode. To monitor the current required by the ESP8266, the ADC in the MSP432 Launchpad was used. A 1 OHM resistor was used to measure the differential voltage that could be converted to current going across the ESP8266 power pin. The push button in the MSP432 was used to change the state of the CH_PD and then start the sampling. The sampling rate was set to 10KHz and the samples for the first 200 milliseconds was stored in a buffer. Then the buffer was read and displayed using the serial monitor to figure out the surge current.

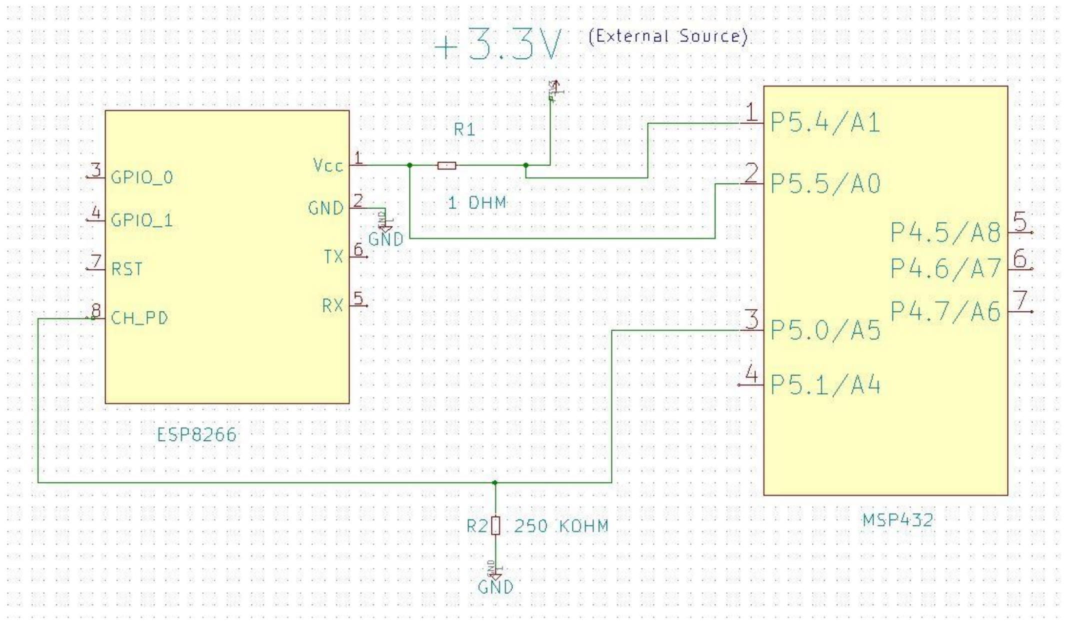


Figure 27: Current measurement of ESP8266 when waking up form power down mode

Figure 27 shows the circuit diagram for differential current measurement across 1 OHM resistor when the CH_PD was set high from the powered down mode. The current was measured by triggering the ADC periodically using a timer. The sampling was done for 10,000 times per second for a 0.1ms sample resolution to monitor the current surge during the ESP8266 power on. This current was observed using a serial monitor. A short interval of 20 samples was captured before the CH_PD line goes high. This was done to establish a baseline current to the ESP8266 prior to power on. Tera Term was the software using as a serial monitor. The eUSCI A UART module of the MSP432 was used at a baud rate of 115200 to communicate with the Tera Term. Based on the readings observed in a serial monitor a graph is plotted in Figure 28.

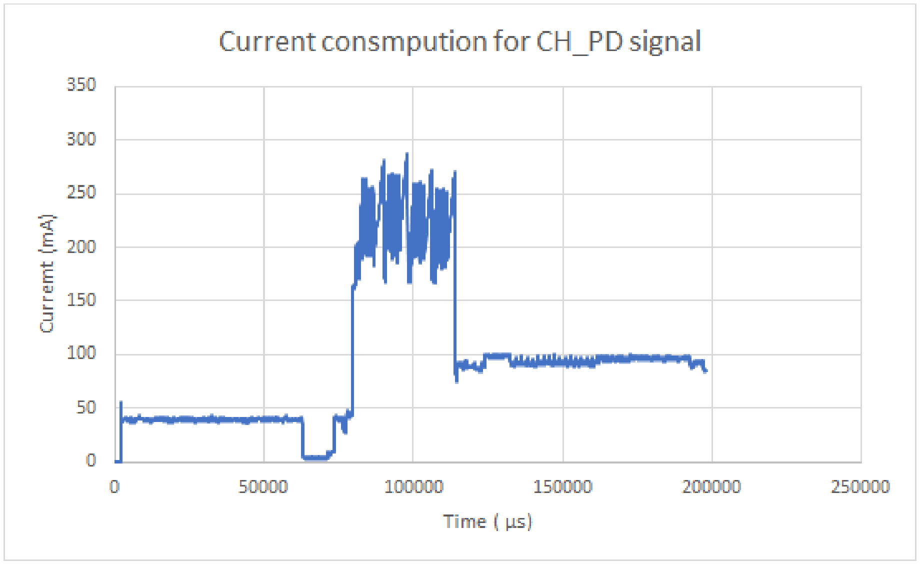


Figure 28: Current consumption for the ESP8266 when CH_PD signal goes high

The ESP8266 required surge currents during the startup of the system. The ESP8266 consumed up to 280 mA currents for about 50 milliseconds. A supercapacitor [25] was placed in the output side of the LTC3106 board to provide the required surge current consumed by the ESP8266 for a very short period.

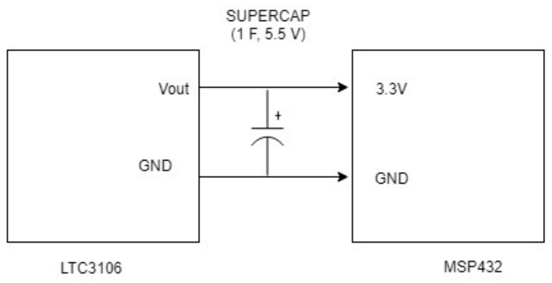


Figure 29: Adding supercapacitor to provide the surge currents

A 1 Farad, 5.5 Volt supercapacitor was chosen. The supercapacitor was chosen with a low leakage current so that it does not unnecessarily drain the battery. It has a low internal resistance, providing high power density capability. The supercapacitors can store and release energy almost instantly, so it gets charged slowly during normal operation of the system and provide the surge current required for the short period of time. It helped the system to run continuously by providing the surge current when required.

When the system was in an idle condition the MSP432 was set to a Low Power Mode. Different power modes were applied to determine the current drawn by the system and LPM3 mode was selected for the idle condition. The active mode had everything operating in normal condition. When the system was in LPM0 mode the CPU was not operating but all the other peripherals were operating. The current reduced to about 5.43 mA compared to earlier 6.05 mA in active mode. When setting the system to LPM3 mode the display was not functioning as the CPU remained inactive and only RTC clock was operating in a very low frequency of 32.76 KHz. The high-frequency clock was disabled. The display was turned off before entering LPM3 mode by sending the display off command to the LCD.

When the push button of the MSP432 is pressed the system wakes up from the deep sleep mode and updates the sensor reading and displays. The backlight of display was set to turn off after 20 seconds so that a display was on for a sufficient time for a user to read the sensor readings. After displaying the values, the system goes to deep sleep mode.

Every hour the system wakes up to transmit data to google sheet and once the transmission process is completed it goes to deep sleep. The backlight of the LCD is off in this process so there is nothing being displayed on the screen. The process goes on continuously and the sensor values are updated in the Google Sheet in every hour. Current consumption for the system in different power modes after the transmission process is complete and the backlight of the display is turned off is shown in Table 2.

Table 2: Current consumption in different power modes

Power Mode	**Current Consumption (mA)**
Active Mode	6.05
LPM0	5.43
LPM3	less than 1

The current consumption in the entire process explained above can be summed up in Table 3. Here for this table, the surge current of the ESP8266 is not included and the average current consumption of the ESP8266 was taken. The value was observed from the DC power supply. For other readings, the values were taken based on the Energy Trace.

Table 3: Average consumption in different steps in chronological order of the system

Mode	**Current in milliampere(mA)**	**Power Consumed By**
When MSP432 is turned ON	1.6	MSP432
All pins are set as Output	1.2	MSP432
When BME is included in the system	5.43	MSP432+ BME
When LCD is included in the system	23	MSP432+ BME+ LCD
When LCD backlight is turned Off	6.05	MSP432+ BME+ LCD
ESP included in the system, System is running, and backlight of LCD is ON	40	MSP432+ BME+ LCD+ ESP8266
Transmitting to Google Document, backlight ON in LCD	94	MSP432+ BME+ LCD+ ESP8266
Transmission is completed, LCD backlight is turned OFF	16	MSP432+ BME+ LCD+ ESP8266
The CH_PD of ESP8266 is powered low	6.05	MSP432+ BME+LCD
The system goes to LPM3 mode, only RTC clock operating	Less than 1	MSP432+BME
Transmitting to Google Document, backlight OFF in LCD	78	MSP432+ BME+ ESP8266

Table 3 is the stepwise optimization that was followed to reduce the current consumption. After all the optimization was complete, the current was measured for different modes for the running system. The energy consumption is displayed in Figure 30, when the system was running.

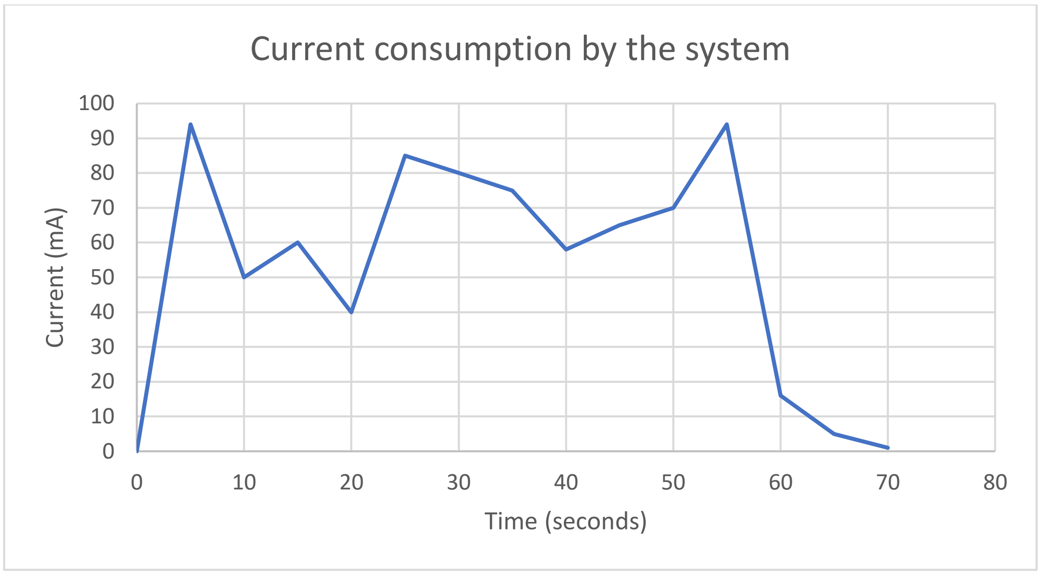


Figure 30: Current consumption for first minute of the system running

Figure 30 shows the current consumption reading for the first 70 seconds after the system was turned on, based on the DC power supply that was used to measure the current. The system consumed 94 mA current maximum when the ESP8266 started and the backlight of the display was on. Then the current consumption was irregular for some time which can be explained as ESP8266 was trying to connect to the internet. After getting connected to the internet the current consumption reduced to 40 mA. The display was on during this period. Then, the sensor values were posted to the google spreadsheet which increased the current consumption to 94 mA and settled down to 16 mA. When the CH_PD was off this value got reduced to about 6 mA currents. The system went to deep sleep mode with display turning off. In this mode, there was very minimal current consumption of less than 1 mA.

4.3 Calculations

After the optimization process the minimum current required during transmission was reduced from 105 mA to about 78 mA. The system was set to sleep mode whenever it was needed. The power required to the system was properly managed. The LCD was included in the system for user convenience. The backlight of the LCD was turned off after it was on for 20 seconds so that the backlight of LCD does not drain unnecessary currents.

Minimum current calculations,

As discussed earlier, the minimum current required for transmission:

Without any optimization = 105 mA (1)

After optimization = 78 mA (2)

Current reduced $= \dfrac{Initial\ value - Final\ Value}{Initial\ value} * 100\ \%$

$= \dfrac{105mA - 78mA}{105mA} * 100\%$

$= 25.71\ \%$ (3)

The minimum current requirement for running the system can be reduced by 25.71% by managing the individual components.

Duty cycle for peak current demand:

The duty cycle for the peak current demand is the ratio of time for the peak current to the time between the transmissions.

Duty Cycle for peak current $= \dfrac{Peak\ current\ \ required\ time}{Time\ gap\ between\ transmission}$ (4)

$= \dfrac{50\ milliseconds}{1\ hour}$

$= \dfrac{0.05\ seconds}{60 * 60\ seconds}$

$$= 0.0001388 \tag{5}$$

The duty cycle for the peak current demand is 0.0001388.

4.4 Using LTC3106 IC and solar panel to charge the system

For this process LTC3106 IC, a uxcell 2 Volt, 160 mA solar panel [17] and a PKCELL LP552035, a 3.7 Volt, 350mAh Li-ion rechargeable battery [26] were chosen. The solar panel was selected as the primary source and the battery as the secondary source for the LTC3106 IC. A printed circuit board of the LTC3106, shown in Figure 31 was used to generate the required current [27]. For this thesis, the pins were connected as according to Table 4.

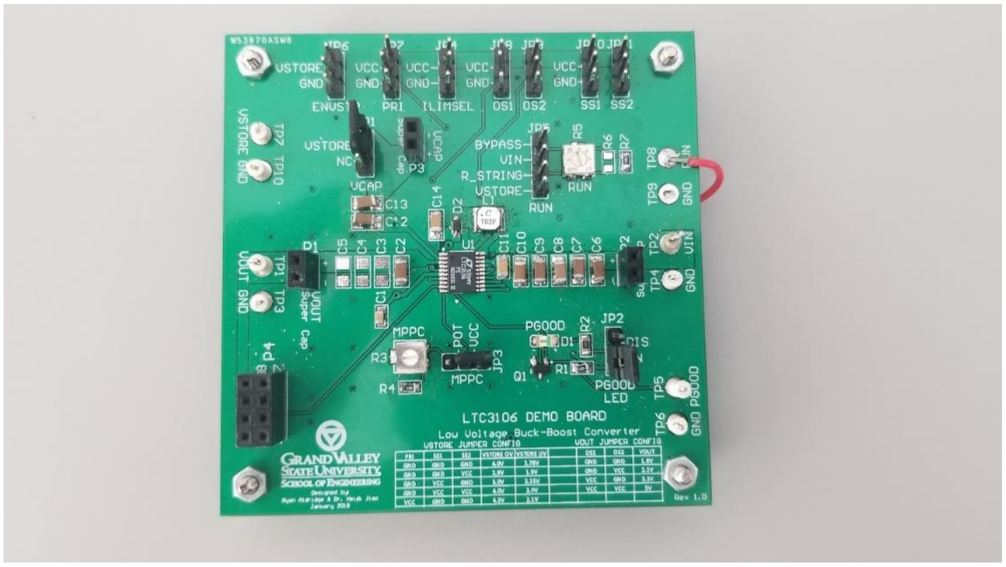


Figure 31: LTC3106 printed circuit board

Table 4: Different pin connection with their function for the LTC3106 IC

PIN Number	PIN	Function	Connections
1	V_{STORE}	Secondary Supply Input	Rechargeable battery connected
2	V_{CAP}	V_{STORE} isolation pin	Tied to V_{STORE}
3	V_{OUT}	Programmable Output Voltage	Output
4	NC	No Connection	Left Unconnected
5	Vaux	Auxiliary Voltage	Connected to GND.
6	V_{CC}	Internal supply rail to power internal connection	
7	OS1	V_{OUT} Select Programming Input	Connected to V_{CC}
8	OS2	V_{OUT} Select Programming Input	Connected to GND.
9	PGOOD	Power Good Indicator	
10	MPP	Set Maximum Power Point Control	Connected to V_{CC}
11	SS1	V_{STORE} Select Programming Input	Connected to GND.
12	SS2	V_{STORE} Select Programming Input	Connected to GND.
13	PRI	Primary Battery Enable Input	Connected to GND.
14	ILIMSEL	Current Limit Input Select	Tied to Vcc.
15	RUN	Input to enable IC and to set custom Vin undervoltage threshold	Tied to Vin.
16	ENVSTR	Enable V_{STORE} Input	Tie to V_{STORE}
17	GND	Internal ground connection	
18	Vin	Main supply input.	Connected to solar power
19	SW2	Buck- boost convertor switch pins.	
20	SW1	Buck- boost convertor switch pins.	

The pins in LTC3106 are connected in such a way that the solar panel was set as a primary source and the rechargeable battery as a secondary source. The aim was to make sure that if the solar energy was enough to make the overall system run, the excess energy was used to charge the battery. The system would run via battery in the absence of solar power. Two solar panels were connected in parallel to get the required amount of current for the load during the daytime.

The ADC was used for measuring the voltage from the solar panel, LTC and the battery and the current from the solar panel and the LTC. The single ended mode of the ADC was used for measuring the voltage. The voltage from the solar cells and the MSP432 was measured with the MSP432 ADC but for the battery voltage, a voltage divider circuit and a buffer op-amp were used to drop down the maximum voltage of the battery that could go as high as 4.2 Volts when fully charged below 3.3 Volts. The MSP432 pins can measure a maximum of 3.3 Volts. A 10 KOHM and a 36 KOHM resistors were used for the voltage divider circuit to drop down the voltage. An LM324 op-amp was used as a buffer to match the impedance level of the battery and the MSP432. The obtained value was converted to the equivalent battery voltage using a linear regression equation. The equation was generated with a known voltage source values and then was verified. For current measurement, 1 OHM resistor was placed as shown in the figure below and the differential voltage across each resistor was measured using the differential mode of ADC of the MSP432. The battery current was not measured directly but the current available to charge the battery can be calculated using the solar current and LTC current. When there was no solar energy available, the battery was sourcing the current to the LTC. When solar energy was available there were some current left for charging the battery, after boosting the output voltage. The currents and voltages were displayed on the system LCD.

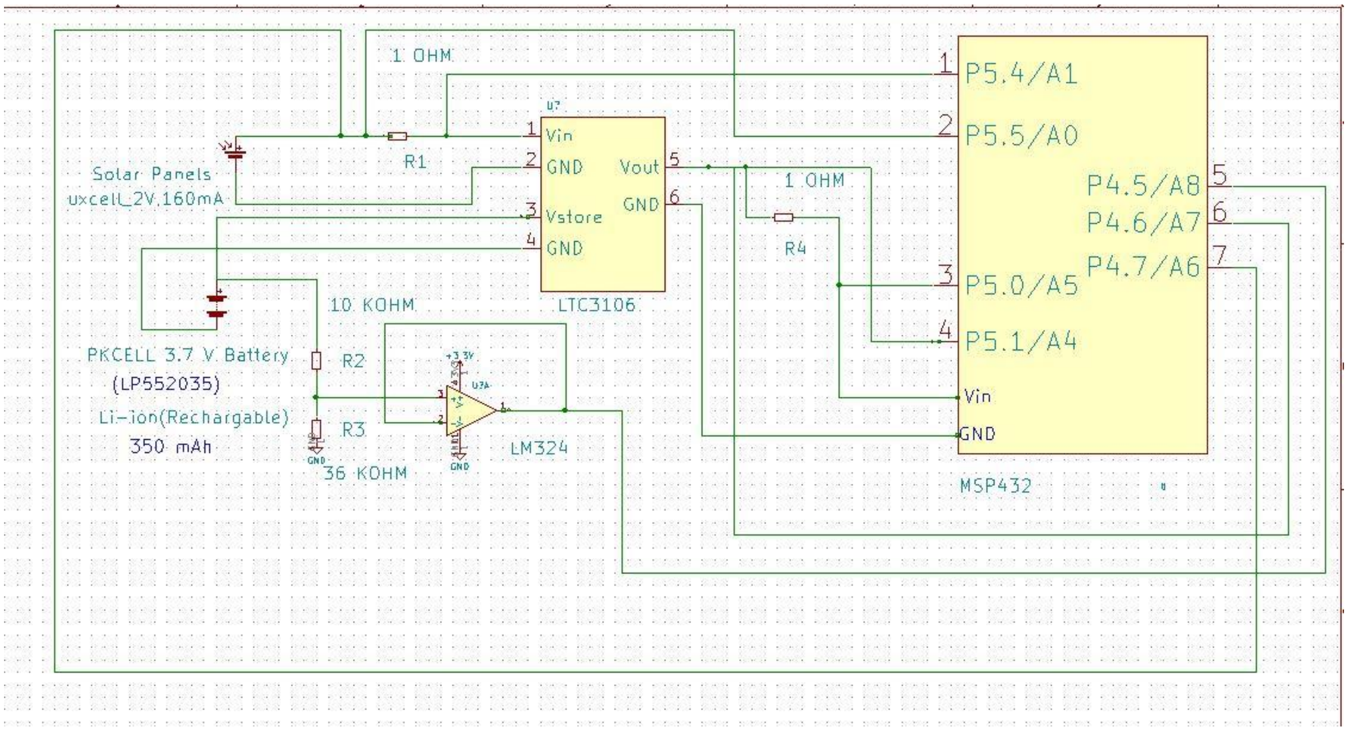
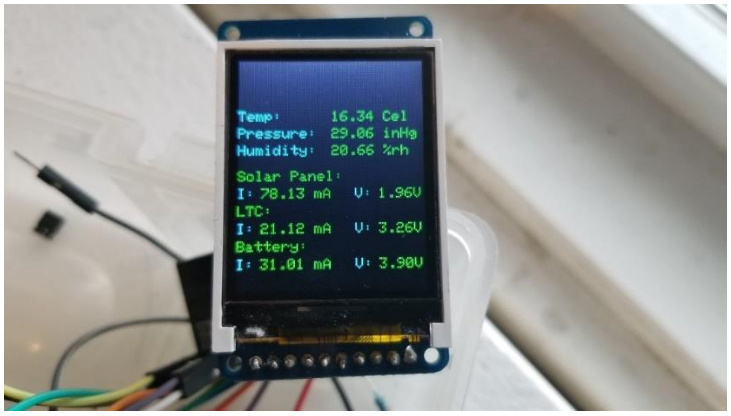


Figure 32: ADC measurement circuit for solar, battery and LTC output

The readings were shown in Figure 33. The reading shows that when two solar panels are connected in parallel, they are supplying 78.13 mA of the current of which 21.12 mA is sourced to the load and 31.01 mA sourced to the battery. The voltages for the solar panels, load and battery are 1.96 V, 3.26 V, and 3.9 V respectively. The values were updated every five seconds using the RTC clock of the MSP432.



```
Temp:        16.34 Cel
Pressure:    29.06 inHg
Humidity:    20.66 %rh

Solar Panel:
I: 78.13 mA   V: 1.96V
LTC:
I: 21.12 mA   V: 3.26V
Battery:
I: 31.01 mA   V: 3.90V
```

Figure 33: Display showing the ADC result values and BME sensor

The ADC values were verified using digital multimeters. First, the voltage values from the ADC were verified. Three voltmeters were placed parallel to the solar voltage, LTC voltage, and battery voltage. Figure 34 shows the voltage reading verification. The voltmeter and the display were showing almost the same readings.

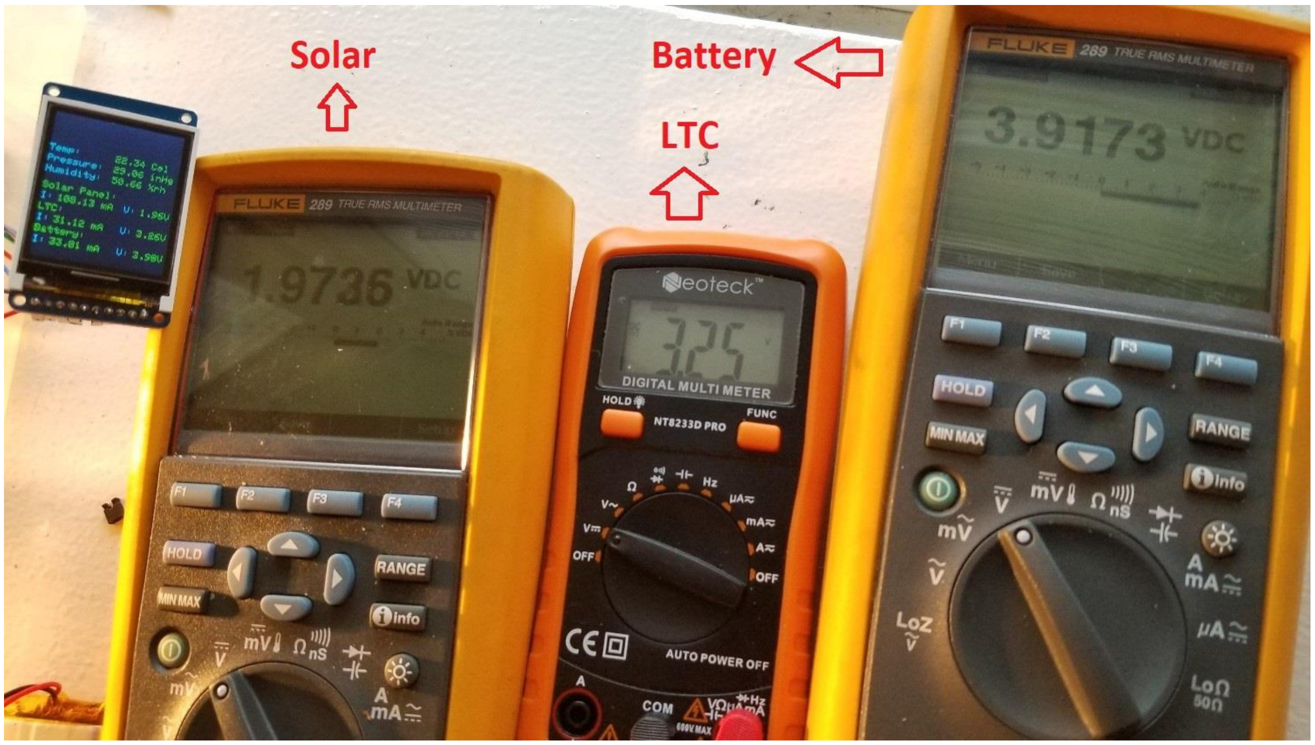


Figure 34: Voltage reading verification using multimeters

After verifying the voltage readings, current readings were also verified by placing multimeters in series to the solar panels, LTC output and battery. The current readings in the display were almost similar to the readings obtained in the multimeters. The battery current reading was discarded as it was not measured directly. The circuit diagram of the final system is shown in Figure 36.

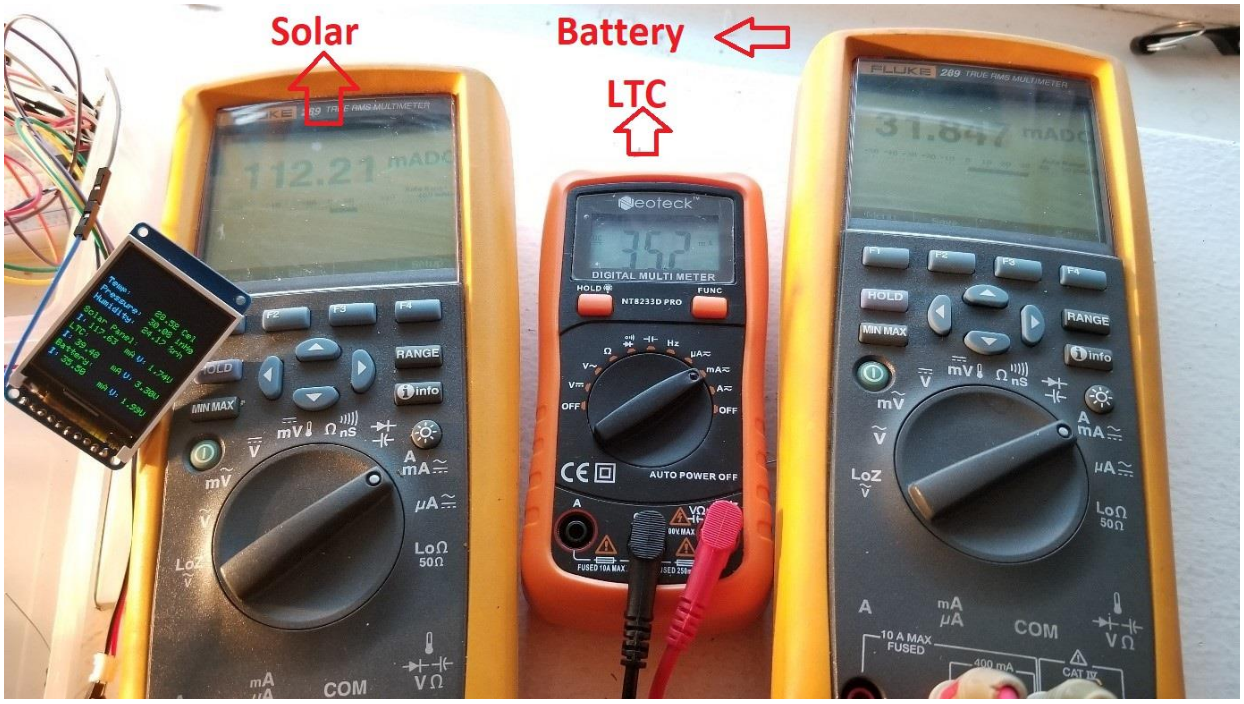


Figure 35: Current reading verification using multimeters

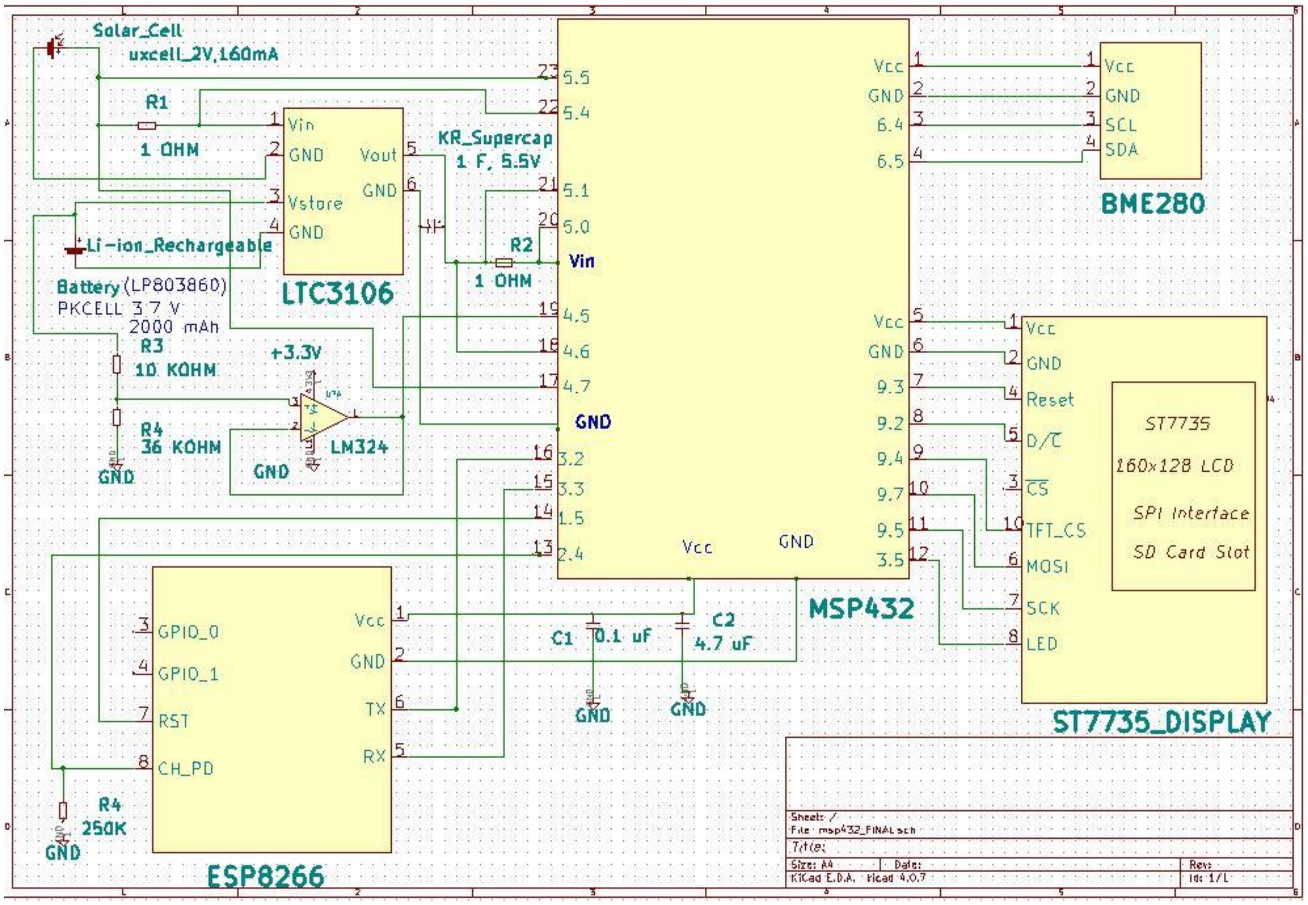


Figure 36: Complete circuit diagram of the system

5. Observations & Results

For this research, a rechargeable Li-ion battery was chosen as a secondary source to power the system. A 2000mAh Ni-MH battery was tested, but it was mostly available in 1.2 Volts, thus required multiple batteries connected in series for providing the required current. While connecting the batteries in series the output voltage gets added. While testing 2 Ni-MH batteries in series, it was only providing current for some time, but not enough time required for ESP8266 during its setup process in the beginning. Even though supercapacitors provided the surge current the initial Wi-Fi connectivity time could last up to a minute, as a result, the system was not able to startup all the time. Thus, Li-ion battery was a better option.

The currents drawn by different load resistors were observed when powered from a full charged 3.7 Volt, 350mAh Li-ion battery serving as a secondary source to the LTC3106. From the observation across different load resistor, it can be observed that the LTC3106 has an output current limitation of 180 mA when the LTC is powered from a Li-ion battery that when fully charged has an output voltage of 4.2V.

Table 5: Current readings for various loads on the LTC3106

INPUT SUPPLIED		Expected Current	OUTPUT	
Voltage (Volts)	Resistance	Current(mA)	Voltage (Volts)	Current(mA)
4.2	220 KΩ	0.015	3.27	0.015
4.2	22 KΩ	0.15	3.26	0.14
4.2	10 KΩ	0.33	3.26	0.32
4.2	4.7 KΩ	0.7	3.26	0.68
4.2	1 KΩ	3.3	3.26	3.27
4.2	470 Ω	7.02	3.26	6.81
4.2	220 Ω	15	3.25	14.25
4.2	100 Ω	33	3.25	30.4
4.2	47 Ω	70.21	3.24	59.9
4.2	20 Ω	165	3.24	120.8
4.2	10 Ω	330	3.221	187.7
4.2	1 Ω (1 watt)	3300	1.59	197.9
4.2	1Ω (2 watt)	3300	1.625	198.2

After the integration of the system described in the previous chapter, a test was done by placing the entire system in outdoor sunlight. A voltmeter was connected to measure the voltage coming from the solar panel and the voltage coming out from the LTC. The light meter LT300, from Extech Instruments [24] was used to measure the light intensity. When the intensity of sunlight was about 22600 Lux, the system was running from a 2 Volt solar panel and was able to charge a 3.7 Volt, 350mAh Li-ion rechargeable battery. The solar panel was supplying around 120 mA current where the excessive current was used up in charging the rechargeable battery. The current and voltage were recorded using multimeters.

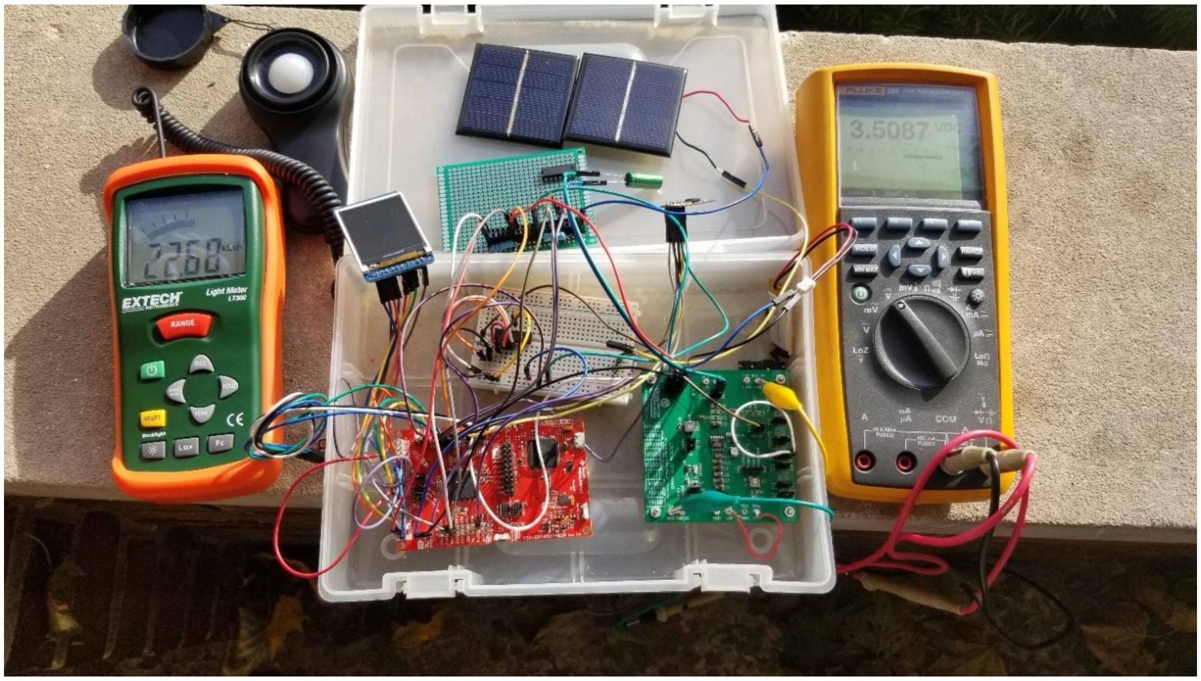


Figure 37: Test setup in sunlight

Another test was done by placing the solar panel in the 500-Watt twin head halogen lights [28]. The intensity of the light was not as good as it was in bright sunlight. The luminous intensity that could be received was 15,000 Lux. The voltage from the solar was intermittent as a result the system was not able to start up all the time just from the solar power. Whenever the solar voltage was above 2.08 Volts the solar power was able to start the system up and was able to charge the battery as well. Adding a supercapacitor in the V_{OUT} pin in the LTC3106 was able to get the system run in the regular intervals. As the solar intensity was not consistent the supercapacitor provided the surge current that was required for the ESP8266. Just using the solar power without using a battery did not seem appropriate for running the system where less light is available. With a Li-ion rechargeable battery as a secondary source, the system started more smoothly.

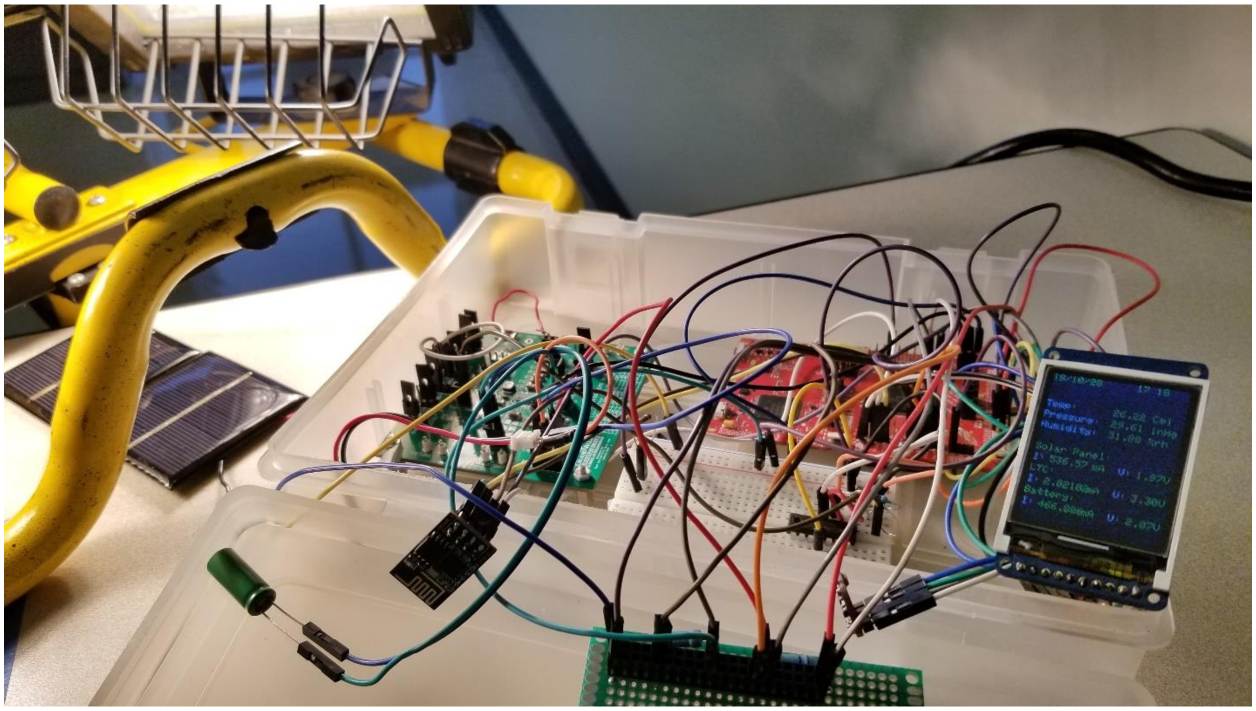


Figure 38: Test setup with solar panel placed near 500-watt twin head lights

Another test was done with placing the entire system in a ELH halogen [28] lightbox. There was not much change in intensity compared to the previous testing condition. It was difficult to get just the solar panels to start the system. But when the system was powered from battery, the solar panels were able to charge the battery and then provide enough current to run the system.

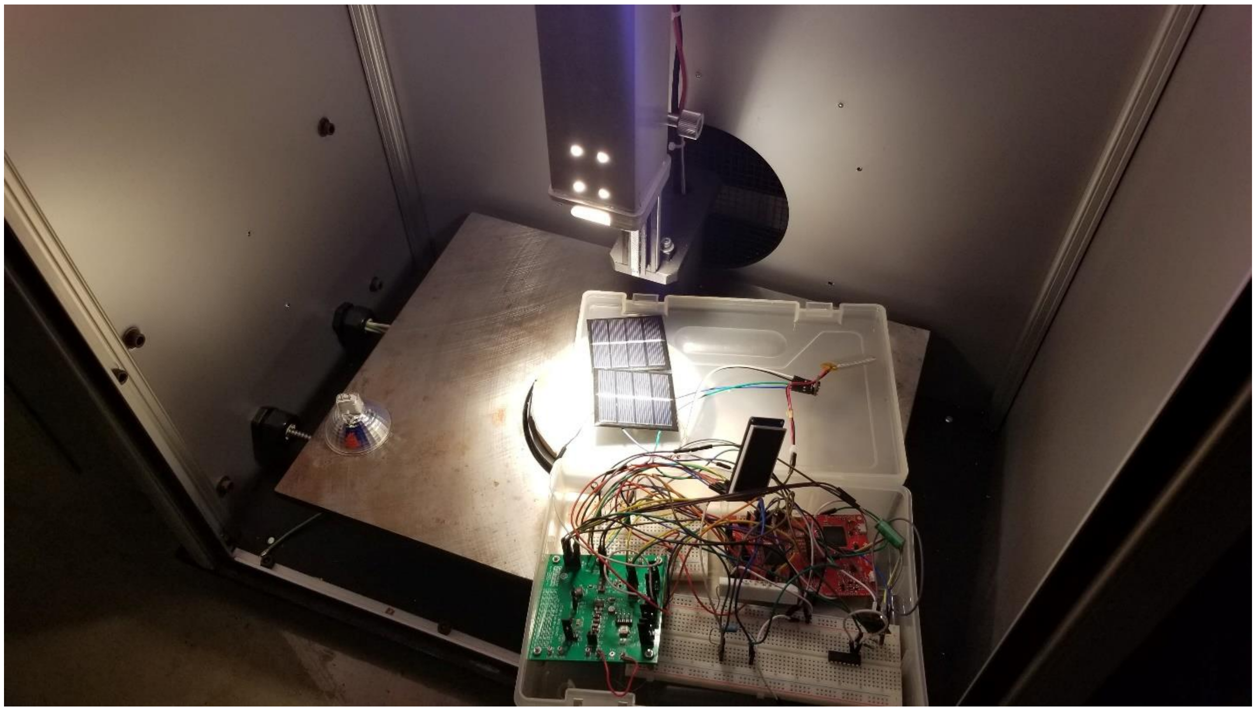


Figure 39: Test setup inside an ELH halogen light box

Then, the program was changed to set the ESP8266 to transmit only when there were 3.7 Volts or above in the battery. When the battery voltage fell below 3.7 Volts the ESP8266 would skip transmission and wait for the battery to charge from the solar panel before starting the transmission. With enough voltage available in the battery the data was transmitted in every hour to the Google Spreadsheets. The battery voltage was constantly monitored so that the system could space out the transmission when the voltage drops below 3.7 Volts.

Discharging of battery:

A 350mAh battery was used to power the system without the solar primary source to recharge it. The battery lasted for 193 hours before the ESP8266 stopped transmitting.

Based on this observation the average current consumed by the system,

$$\text{Average Current} = \frac{Battery\ Capacity}{Total\ time} \tag{6}$$

$$= \frac{350\ mAh}{193\ hrs}$$

$$= 1.81 \text{ mA} \tag{7}$$

A 2 Volt, 160mA solar panel was tested to figure out the charging time required for a Li-ion rechargeable battery. A 350mAh Li-ion rechargeable battery was observed for finding the charging rate in a light source with the intensity of 15000 Lux. The charging time was 30 hours 27 minutes.

$$\text{Charging time for 350mAh Li-ion battery} = \text{30 hours 27 minutes} \tag{8}$$

The charging time was too long to observe the charging phenomena. A LIR2450 [29], 100 mA Li-ion coin cell rechargeable battery was chosen. It took around 500 minutes to charge the battery fully from 2.7 Volts when the light intensity was 15000 Lux. The light was provided by 500-Watt twin head halogen lights [28] and the light meter used was LT300 [24]. The ADC in the MSP432 was used to measure the current coming from the solar panel and the voltage of the battery. The detail about the ADC setup is in the previous chapter. The readings were printed every minute in a serial monitor like it was done for the measurement of current for the CH_PD signal. The data was verified by connecting multimeter to measure the current and voltage. An ammeter was placed in series with the solar panel and the input source of the LTC3106. A voltmeter was placed parallel with the V_{STORE} voltage. Figure 40 shows the charging rate for the battery as observed in the serial monitor.

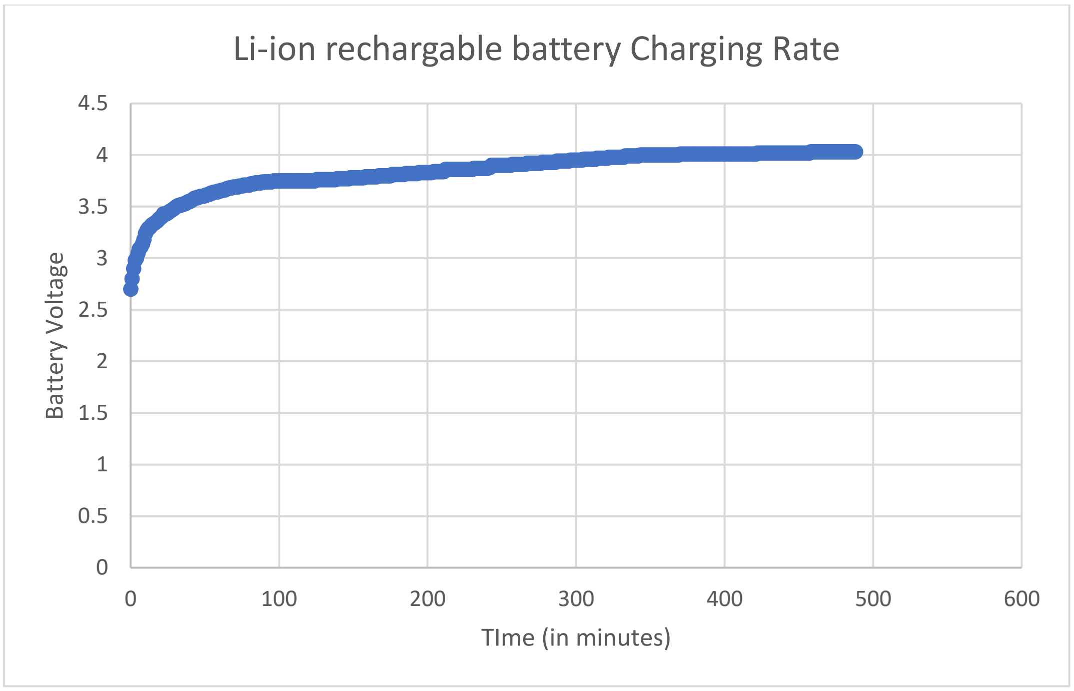


Figure 40: Charging rate for LIR2450, Li-ion rechargeable battery

After testing in the controlled environment, the charging rate was observed for different lighting conditions and it was almost constant under all the conditions. The solar panel provided a constant current at the rate of 37mAh.

Table 6: Readings during charging LIR2450 battery on various light conditions

Condition	**Intensity of light**	**Battery Charge Time**
Bright Sunny	Above 50000 Lux	8hrs 20 minutes
Sunny	40000-50000 Lux	8hrs 30 minutes
ELH Halogen Light Box	25000 Lux	8hrs 35 minutes
ELH Halogen Light Box	15000 Lux	8hrs 45 minutes
500-watt Twin Headed Light Source	15000-20000 Lux	8hrs 45 minutes

Ideal condition calculations:

Based on the readings for charging and discharging time for the Li-ion rechargeable battery

Charging time for 350mAh battery = 30 hours 27 minutes

Discharging time for 350mAh battery = 193 hours

Since the battery charging time is quicker than battery discharging time and the battery is charging when the solar current exceeds the system load, it can be inferred that the system should run continuously provided 12 hours of a sunny day and 12 hours of dark while transmitting the data every hour and then going to sleep. The system was tested in a controlled lighting condition where for 12 hours the lights were turned on and for 12 hours it was off. Two solar panels were connected in parallel as a primary source and a 350mAh battery as a secondary source. The system was able to transmit the data every hour and sleep in between transmission. The current consumption was minimal during sleep times, as a result, the solar panels were able to charge the battery when the lights were on. The table below is the readings seen for a day when for 12 hours the light was turned on and for 12 hours the lights were off. The sensor was read just after the ESP8266 was powered up and were ready to transmit the data.

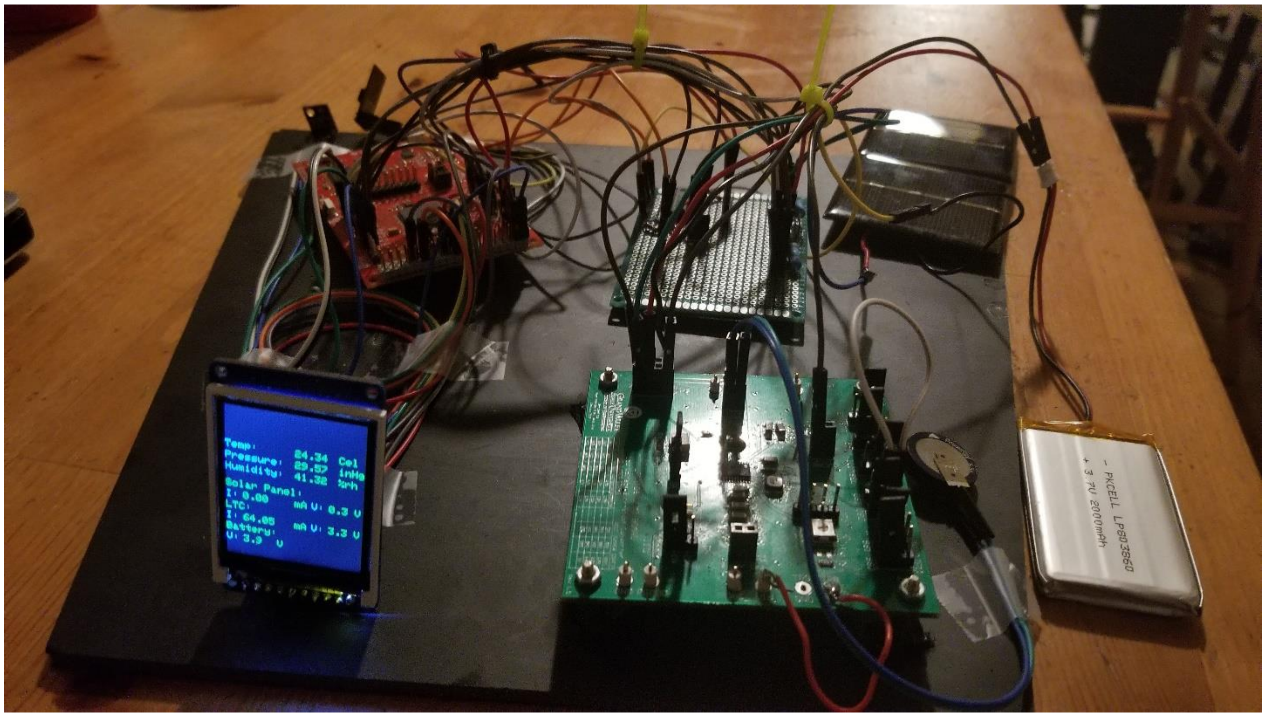
Table 7: Data collected for 24 hours with lights on and off

Date/ Time (mm:dd:yy hh:mm)	**Humidity (%rh)**	**Temp(⁰C)**	**Pressure(inHg)**	**Solar Voltage(V)**	**Solar Current(mA)**	**LTC Voltage (V)**	**LTC Current(mA)**	**Battery Voltage (V)**
11/2/2018 10:37	44.28	16.07	29.88	1.81	106.23	3.3	53.7	4.0
11/2/2018 11:37	43.56	16.41	29.88	2.23	108.83	3.3	53.28	4.0
11/2/2018 12:37	43.76	16.65	29.89	2.12	107.21	3.3	47.99	4.0
11/2/2018 13:37	46.33	16.64	29.89	2.13	105.56	3.3	57.09	4.0
11/2/2018 14:37	45.25	16.59	29.87	2.12	108.26	3.3	52.77	4.0
11/2/2018 15:37	43.87	16.43	29.86	2.17	109.57	3.3	52.77	4.0
11/2/2018 16:37	45.1	16.78	29.86	2.18	108.83	3.3	52.77	4.0
11/2/2018 17:37	41.7	17.17	29.87	2.24	109.26	3.3	54.91	4.0
11/2/2018 18:37	40.95	17.16	29.88	2.18	98.12	3.3	49.88	4.0
11/2/2018 19:37	42.36	17.78	29.88	1.86	103.26	3.3	46.03	4.0
11/2/2018 20:37	40.5	18.52	29.9	1.94	107.26	3.3	53.22	4.0
11/2/2018 21:37	42.67	18.31	29.92	2.08	108.12	3.3	54.16	4.0
11/2/2018 22:37	40.6	18.17	29.91	0	0	3.3	51.77	4.0
11/2/2018 23:37	39.91	17.71	29.92	0	0	3.3	52.77	4.0
11/3/2018 0:37	40.53	17.58	29.92	0	0	3.3	35.71	3.9
11/3/2018 1:37	40.3	17.83	29.92	0	0	3.3	52.77	3.9
11/3/2018 2:37	39.77	17.86	29.92	0	0	3.3	50.76	3.9
11/3/2018 3:37	40.03	17.6	29.94	0	0	3.3	52.77	3.9
11/3/2018 4:37	40.74	17.27	29.95	0	0	3.3	52.77	3.9
11/3/2018 5:38	40.79	17.41	29.96	0	0	3.3	52.77	3.9
11/3/2018 6:38	39.59	17.69	29.99	0	0	3.3	44.19	3.9
11/3/2018 7:38	39.79	17.5	30.01	0	0	3.3	52.12	3.9
11/3/2018 8:38	40.53	17.16	30.03	0	0	3.3	58.15	3.9
11/3/2018 9:38	41.21	17.11	30.07	0	0	3.3	47.29	3.9

Data was collected for the first 12 hours. Lights were on and the solar panel was able to power up the system and charge the battery with the excess energy. The temperature, pressure, and humidity were observed as well as the voltage and current of the solar panel, LTC and battery. When the lights were on the solar panel supplied currents to run the load and was also able to charge the battery.

The solar voltage was fluctuating around 2 Volts when the lights were on. Most of the time it was above 2 Volts but there were times it fell below 2 Volts. The LTC output voltage value was constant 3.3 Volts. The battery was charging very slowly in the presence of solar panels. The battery had 4.0 Volts during the start of the system and its voltage increased slightly to 4.01 Volts. But after the lights were removed the voltage got reduced slowly.

Later the system was tested in a regular environment where it could receive sunlight during the day hours. The battery powered the system during the night time and solar panels recharged the battery when there was sunlight. The system ran through the battery power, and when sunlight was available it was able to compensate the charge lost by the battery. For this final setup, an LP803860 a PKCELL 2000mAh Li-ion rechargeable battery [26] was chosen so that the battery could last for longer days when there is less sunlight.



```
Temp:
Pressure:   24.34  Cel
Humidity:   29.57  inHg
            41.32  %rh
Solar Panel:
I: 0.00
LTC:        mA V: 0.3 V
I: 64.05
Battery:    mA V: 3.3 V
V: 3.9   V
```

Figure 41: Test setup for the final system in a normal environment

With changing the battery to a 2000mAh, the system was tested for 20 days period. The system ran continuously for 20 days without any interruption. There was very minimal sunlight available during the testing condition but whenever there was sunlight the solar panels were able to charge the rechargeable battery. Table 8 shows the snippet for data collected over the period.

Table 8: Data snippet recorded during the test in normal condition

Date/ Time (mm:dd:yy hh:mm)	**Humidity (%rh)**	**Temp(⁰C)**	**Pressure(inHg)**	**Solar Voltage(V)**	**Solar Current (mA)**	**LTC Voltage (V)**	**LTC Current(mA)**	**Battery Voltage (V)**
11/9/2018 9:43:03	26.55	22.97	30.19	0.5	0	3.3	46.18	4.0
11/9/2018 10:43:07	22.25	23.95	30.23	0.5	0	3.3	33.84	4.0
11/9/2018 11:43:17	23.68	22.32	30.27	0.5	0	3.3	47.18	4.0
11/9/2018 12:43:28	23.22	23.2	30.32	0.3	6.45	3.3	52.74	4.0
11/9/2018 13:43:35	18.69	25.53	30.32	2.2	33.44	3.3	46.52	4.0
11/9/2018 14:43:27	28.56	24.03	30.02	0	0	3.3	54.08	4.0
11/9/2018 12:43:28	25.12	22.16	30.05	0.4	0	3.3	13.29	4.0
11/9/2018 15:43:35	24.22	22.84	30.05	0.7	1.21	3.3	32.22	4.0
...								
…								
…								
11/18/2018 10:44:26	17.38	27.24	30.59	2	30.4	3.3	35.71	3.9
11/18/2018 11:44:28	17.3	27.34	30.59	2.3	24.98	3.3	17.72	3.9
11/18/2018 12:44:25	16.31	29.6	30.58	2.3	27.98	3.3	34.43	4.0
11/18/2018 13:44:23	16.22	29.78	30.57	0.7	3.22	3.3	42.42	3.9
11/18/2018 14:44:22	17.88	25.7	30.54	1	4.83	3.3	37.25	3.9
11/18/2018 15:44:25	`16.89	24.01	29.98	0.6	0.4	3.3	24.57	3.9
11/18/2018 16:44:23	16.35	21.47	29.96	0.8	0	3.3	44.5	3.9
…								
…								
…								
11/28/2018 19:47:46	35.48	21.81	30.06	0.7	0	3.3	53.12	3.9
11/28/2018 20:47:45	34.24	20.16	30.24	0.7	0	3.3	45.66	3.9
11/28/2018 21:47:45	33.18	20.27	30.12	0.7	0	3.3	28.12	3.9
11/28/2018 22:47:45	33.28	20.41	29.13	0.7	0	3.3	31.11	3.9
11/28/2018 23:47:45	35.48	21.81	30.06	0.7	0	3.3	33.15	3.9

6. Conclusion

6.1 Summary

A system was designed to collect- the temperature, pressure and humidity data sensor. The data was transmitted to a workstation in a remote location by connecting to a router with an Internet connection. The current consumption of the system was measured, and successive steps were taken to optimize the overall system. The process started with optimization in the MSP432 board, followed by BME sensor, LCD display and the Wi-Fi module. The sleep mode feature of the MSP432 was used to set the system to sleep when the system is not transmitting any data or displaying the values. The minimum current required during transmission was reduced by 25.71% after the optimization process.

The LTC3106 was used for managing power to the system. The solar panel was used as a primary source and a Li-ion battery as a secondary source. The current and voltage coming from the solar panels, coming out form the LTC3106 and the voltage from the battery was measured using the ADC in the MSP432 board. When solar energy was available, the LTC was consuming some current to boost the voltage, some of the difference was available to charge the battery. The battery charging current should be in the opposite polarity from the current supplied by the battery.

Based on observations on different lighting condition a 350mAh battery was chosen for testing the charging and discharging rate. The system transmitted the sensor readings to the Google Spreadsheets continuously for about 8 days (193 hours) on battery power alone with an average current consumption of 1.81 mA due to the low duty cycle of 0.0001388 of peak current demand. Then the system was tested in a controlled environment where the charge lost during dark hours was compensated by the solar power received during day hours. After making sure the system runs smoothly in a controlled environment the system was placed in a normal environment where the solar energy was able to charge the system whenever sunlight was available. For the final system a 2000mAh battery was chosen to compensate longer dark hours. The final system was tested for a period of 20 days. The system ran continuously without any interruption.

6.2 Future Work

The data was transmitted to the Google documents every hour. By logging data to the SD card more frequently than it is posted in a Google Sheet the values can be read more often than hourly basis. Power savings has always been a major concern in the present technology world. This research was performed on the MSP432 microcontroller. Different ways were analyzed to save power. Similar techniques can be applied to other types of microcontroller to figure out the most optimized case for a required scenario. This research work was solely involved in optimizing the power and was done on a breadboard with different kinds of configurations. This work can be taken into a next level by building a product that can be useful in the real-world application. By replacing the sensor and using different other sensors the data recording and transmitting can be useful in different industrial and other applications.

The ESP8266 is a power consuming device and most of the power in this system was consumed by the ESP8266 device during transmission of data and its startup. Different other possibilities can be explored to figure out better alternatives for this Wi-Fi module.

Appendix A

This chapter includes more detail about the MSP432 Launchpad board used for this research. It describes about the Launchpad, including the block diagram and functional description.

MSP432 Launchpad

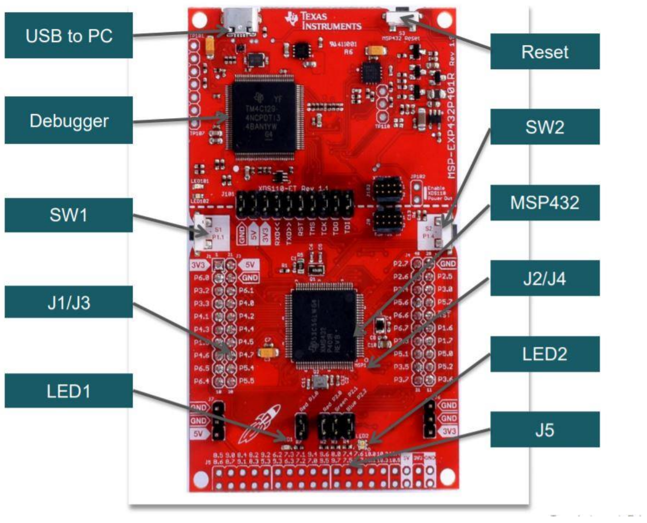
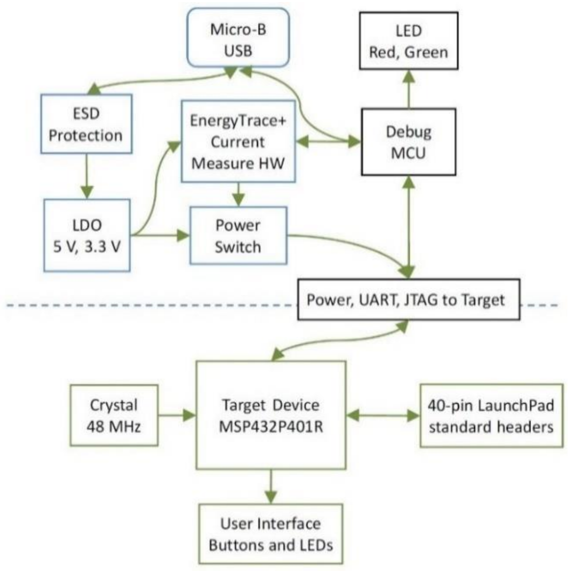


Figure A1: MSP432 Launchpad board

The MSP 432 Launchpad board includes different pins as shown in Figure A1. The jumper J1 to J4 are the 40 pins booster pack connector. Booster pack is the modular plug-in board that fits on top of the launchpad as seen in Figure A1. These pins are used to make connections to the sensor and Wi-Fi modules. The regular function of the pins is to perform I/O functions however most of the pins have one or more other functions.

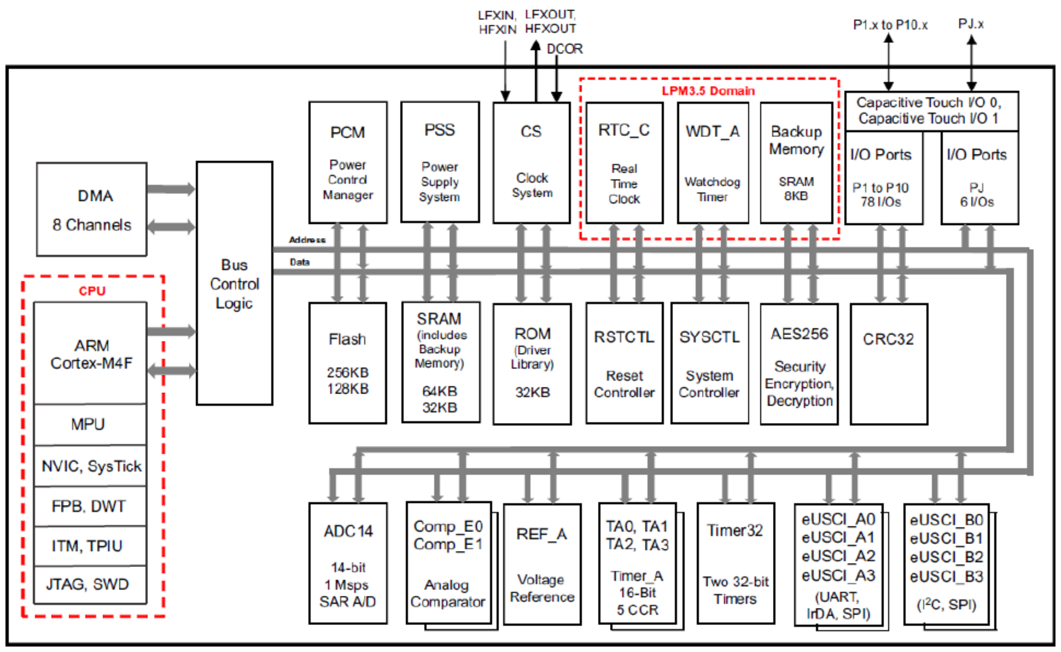
The launchpad also includes two push buttons and two LEDs for user interaction. Figure A2 shows the general block diagram for the MSP432 Launchpad. It includes how the components are linked with each other inside the launchpad.



Micro-B USB
LED Red, Green
ESD Protection
EnergyTrace+ Current Measure HW
Debug MCU
LDO 5 V, 3.3 V
Power Switch
Power, UART, JTAG to Target
Crystal 48 MHz
Target Device MSP432P401R
40-pin LaunchPad standard headers
User Interface Buttons and LEDs

Figure A2: Block diagram of Launchpad ***[5]***

Figure A2 shows the functional block diagram of the MSP432 devices and how they are interconnected on the IC. The Cortex-M4F processor is built on a high-performance core with 3-stage pipeline Harvard architecture making it suitable for demanding embedded applications.



LFXIN, HFXIN
LFXOUT, HFXOUT
DCOR
P1.x to P10.x
PJ.x
LPM3.5 Domain
PCM Power Control Manager
PSS Power Supply System
CS Clock System
RTC_C Real Time Clock
WDT_A Watchdog Timer
Backup Memory SRAM 8KB
Capacitive Touch I/O 0, Capacitive Touch I/O 1
I/O Ports P1 to P10 78 I/Os
I/O Ports PJ 6 I/Os
DMA 8 Channels
Bus Control Logic
Address
Data
CPU
ARM Cortex-M4F
MPU
NVIC, SysTick
FPB, DWT
ITM, TPIU
JTAG, SWD
Flash 256KB 128KB
SRAM (includes Backup Memory) 64KB 32KB
ROM (Driver Library) 32KB
RSTCTL Reset Controller
SYSCTL System Controller
AES256 Security Encryption, Decryption
CRC32
ADC14 14-bit 1 Msps SAR A/D
Comp_E0 Comp_E1 Analog Comparator
REF_A Voltage Reference
TA0, TA1 TA2, TA3 Timer_A 16-Bit 5 CCR
Timer32 Two 32-bit Timers
eUSCI_A0 eUSCI_A1 eUSCI_A2 eUSCI_A3 (UART, IrDA, SPI)
eUSCI_B0 eUSCI_B1 eUSCI_B2 eUSCI_B3 (I²C, SPI)

Figure A3: Functional block diagram of MSP432 device [30]

Appendix B

This chapter includes the description of the LTC3106 board used for this thesis. It details about different pins in the LTC3106 board and its functions.

LTC3106 Breakout Board

A 20-pin breakout board was used for connecting different pins of the LTC3106. The pins for the LTC3106 and its features are detailed below:

1) NC: No Connect. It is not connected electrically. It can be connected to PCB ground or left floating.
2) V_{OUT}: It is a programmable output voltage pin. It must be connected with at least 22µF low ESR capacitor to GND. The size of the capacitor is dependent on the application.
3) V_{AUX}: This pin is a generated voltage rail used for powering the internal circuitry It must be connected to a 2.2µF minimum ceramic capacitor to GND.
4) V_{CC}: This is an internal supply rail which is used for powering the internal circuitry. It must be decoupled with a 0.1µF ceramic capacitor.
5) OS1, OS2: These pins are used for programming the V_{OUT} selectable input. The output voltage can be generated into 4 different values based on the following selection in.

Table B1: Output Voltage Selection

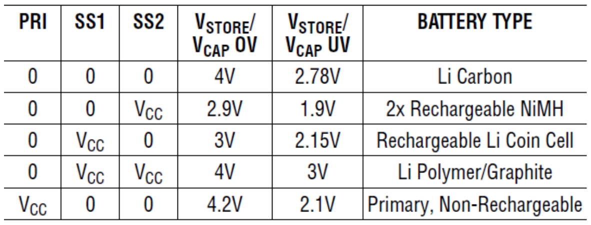
OS1	OS2	OUTPUT VOLTAGE
0	0	1.8V
0	V_{CC}	2.2V
V_{CC}	0	3.3V
V_{CC}	V_{CC}	5V

6) PGOOD: Power Good Indicator. This pin is pulled to ground if the V_{OUT} falls 8% below the programmed voltage.

7) MPP: It is a Maximum Power Point control pin.

8) SS1, SS2: These pins are for the V_{STORE} select programmable inputs. Based on Table the pins need to be connected.

Table B2: V_{STORE} voltage selection

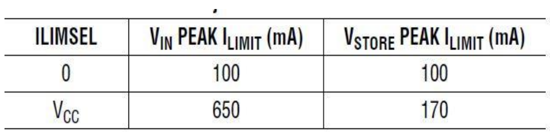


PRI	SS1	SS2	V_{STORE}/V_{CAP} OV	V_{STORE}/V_{CAP} UV	BATTERY TYPE
0	0	0	4V	2.78V	Li Carbon
0	0	V_{CC}	2.9V	1.9V	2x Rechargeable NiMH
0	V_{CC}	0	3V	2.15V	Rechargeable Li Coin Cell
0	V_{CC}	V_{CC}	4V	3V	Li Polymer/Graphite
V_{CC}	0	0	4.2V	2.1V	Primary, Non-Rechargeable

9) PRI: Primary Battery Enable Input. This pin is connected to V_{cc} for enabling the use of a non-rechargeable battery and to disable V_{STORE} pin charge capability. The pin is connected to GND to use a secondary battery and enable charging.

10) ILLIMSEL: This pin is used for selecting the Current Limit Input. The pin is configured for the different current condition as the current requirement shown in Table B3.

Table B3: Current Limit Adjustment



ILIMSEL	V_{IN} PEAK I_{LIMIT} (mA)	V_{STORE} PEAK I_{LIMIT} (mA)
0	100	100
V_{CC}	650	170

11) RUN: This pin is an input to enable the IC and set custom V_{IN} undervoltage thresholds. A voltage greater than 400mV will enable certain IC function. The threshold set at 600mV enables V_{IN} as input.

12) ENVSTR: This pin enables V_{STORE} as input. It is tied to V_{STORE} to enable it as a backup input. It is tied to GND to disable V_{STORE} as a backup input source.

13) GND: This pin is the ground pin and is connected to PCB ground for internal electrical ground connection.

14) V_{IN}: This is the main supply input pin. It must be decoupled with a minimum 10uF capacitor.

15) SW1, SW2: These pins are Buck-Boost Converter switch pins. An inductor is connected between these two pins.

16) V_{STORE}: This pin is connected with the secondary input supply. A rechargeable battery may be connected with this pin to GND to power the system when the input voltage is lost. This pin must be tied to V_{CAP} for primary or high capacity secondary battery application.

17) V_{CAP}: This pin isolates V_{STORE} from the decoupling capacitor for low capacity backup batteries.

Appendix C

Source Code

Main.c

```c
//******************************************************************************
/* MSP432 - Main file
* This is the main file for this project.
* Different header files are included which includes different functions used for this research like
initialization of pins, clock, LCD display, BME sensor.
* The program check the battery voltage and if it is above 3.7V then it turns on the ESP8266 and updates
the sensor readings to Google Spreadsheet
*If battery is below 3.7V it displays the reading of the sensor. After displaying the reading the system
goes to sleep mode. A state machine is designed to handle the operation for different steps in this
program.
*The sensor reads the value from the function in the file bmeinterfacing.h
*RTC interrupt handler is for the Real Time Clock interrupts
*Port1 interrupt handler is for the interrupt received from the button press to turn on and off the LCD
* When the battery voltage falls below 3.7V, it stops transmitting and only transmit when the battery
voltage is again above 3.7V making the system run continuously without any interruption.

//*****************************************************************************


/*Header files*/
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>

#include "msp.h"
#include "driverlib.h"

/*Different header files created for running the project*/
#include "bme280Interfacing.h"
#include "i2c.h"
#include "uart.h"
#include "ST7735.h"
#include "rtc.h"
#include "clk.h"
#include "gpio.h"


void getDateAndTime();
void Delay1ms();
uint8_t  writecommand(uint8_t c);

typedef enum {
    screen1=0,
    screen2
}_screen;
_screen screen= screen1;


/*Variables created to store different values*/
uint8_t U2RXData;
bool timeset= 0;
int twoSecondToggle = 0;
bool timeBlinker = 0;
bool spreadsheet =0;
int sensorCount=0;
int screenOff=0;
int wakeup=0;
int less_battery=0;
int connection_entered=0;
int gotdata=0;
int batteryVoltage= 0;

RTC_C_Calendar newTime; //to store updated time
RTC_C_Calendar currentTime;  //to store updated time

char receivingBuffer[2500];
char timeBuffer[32];
int bufferposition = 0;
char displayBuff [20];
char at_commands [100];
char get_command[300];

/*SSID username and password*/
char SSID [] = "SM-G950U66E";
char password [] = "6163041458";

/*Device id to connect to pushingbox.com*/
char deviceId []= "v76E7A8A355A371B";

#define ST7735_DIS0N   0x29
#define ST7735_DISPOFF  0x28

/*Readings to be displayed on the screen*/
volatile char temperature[7];
volatile char pressure[7];
volatile char humidity[7];
volatile char current_solar[9];
volatile char voltage[7];
volatile char current_ltc[9];
volatile char voltage2[7];
volatile char voltage3[7];
volatile char current_battery[9];

/*State machine for different modes*/
typedef enum  {CheckAdc, ESPReset, ConnectionSetUp, SetESPMode, ConnectToRouter,
TCPConnect,LCDinit,UpdateData,END,Display,ConnectSpreadSheet} states;
states CurrentState;

/*Delay 10milliseconds*/
void DelayWait10ms(uint32_t n){
  Delay1ms(n*10);
}

/*Function to load values to the display*/
void display(int row, int col, char *str, int front, int back, int size)
{
    int8_t i = 0;
    for(i = 0; str[i]!= '\0';i++)
    {
        ST7735_DrawCharS(row*8 +6*i*size, col * 6*size , str[i], front, back, size);
    }

}

/*UART A2 handler
* UART A2 handler is used for communication of the MSP432 with the ESP8266
* It receives the values from the web and store it in a receiving buffer
*/
void EUSCIA2_IRQHandler(void)
{
    uint32_t status = MAP_UART_getEnabledInterruptStatus(EUSCI_A2_BASE);
    MAP_UART_clearInterruptFlag(EUSCI_A2_BASE,status);

    if (status & EUSCI_A_IE_RXIE)
    {

        U2RXData = MAP_UART_receiveData(EUSCI_A2_BASE);
```

```c
        if(bufferposition >= 2500)
            bufferposition = 0;
        receivingBuffer [bufferposition++] = U2RXData;
        while(!(UCA0IFG & UCTXIFG));
        UCA0TXBUF = U2RXData;

    }
}

/*Function to check if the received string is OK*/
int checkforOK()
{
     char* charPointer = strstr(receivingBuffer,"OK");
    //lastIndex = bufferposition;
   if (charPointer != NULL)
        return 1;
    else
        return 0;

}

/*Function to check if the received string is CLOSED*/
int checkforClosed()
{
    //printf("Response: %s",receivingBuffer);
    char* charPointer = strstr(receivingBuffer,"CLOSED");
    //lastIndex = bufferposition;

    if (charPointer != NULL)
        return 1;
    else
        return 0;

}

/*Function to check if the received string is ERROR*/
int checkforError()
{
    //printf("Response: %s",receivingBuffer);
    char* charPointer = strstr(receivingBuffer,"ERROR");
    //lastIndex = bufferposition;

    if (charPointer != NULL)
        return 1;
    else
        return 0;

}

/*Function to check if the received string is ALREADY CONNECTED*/
int alreadyConnected()
{
    char* charPointer = strstr(receivingBuffer,"ALREADY CONNECTED");
    //lastIndex = bufferposition;

    if (charPointer != NULL)
        return 1;
    else
        return 0;
}

/*Function to check if the received string is IPD51*/
char* getTime()
{
    char *timePointer = strstr(receivingBuffer,"+IPD,51:");
    if (timePointer != NULL)
        return timePointer;
    else
        return 0;
```

```
}

/*Function stores the value of Date and Time in Time buffer*/
void updateDateAndTime(){
    char* pointintTotime;
    pointintTotime = getTime();
    if(pointintTotime){
        memcpy(timeBuffer,pointintTotime,32);
        getDateAndTime();
    }

}

/*Function is used to convert the ASCII values received to actual values in appropriate time zone*/
void getDateAndTime()
{
    newTime.year = (int)(timeBuffer[15]-48)*10+(int)(timeBuffer[16]-48);
    newTime.month = (int)(timeBuffer[18]-48)*10+(int)(timeBuffer[19]-48);
    newTime.dayOfmonth = (int)(timeBuffer[21]-48)*10+(int)(timeBuffer[22]-48);
    newTime.hours = (int)(timeBuffer[24]-48)*10+(int)(timeBuffer[25]-48);
    newTime.minutes = (int)(timeBuffer[27]-48)*10+(int)(timeBuffer[28]-48);

    if (newTime.hours == 0)
    {
        newTime.hours = 24;
        newTime.dayOfmonth -=1;
    }
    newTime.hours -= 4;

    MAP_RTC_C_initCalendar(&newTime, RTC_C_FORMAT_BINARY);//passing the entered input in RTC in binary
format
    /* Specify an interrupt to assert every minute */
    MAP_RTC_C_setCalendarEvent(RTC_C_CALENDAREVENT_MINUTECHANGE);
    MAP_RTC_C_startClock();
    timeset = 1;
    DelayWait10ms(5);
    ST7735_FillScreen(ST7735_BLACK);
    CurrentState = ConnectSpreadSheet;
}


/*Function check if the recieved string is IPD*/
int checkForIPD()
{
    char* ipdPointer = strstr(receivingBuffer,"+IPD");
    if (ipdPointer != NULL)
            return 1;
        else
            return 0;

}

/*Function to send character to ESP8266*/
void send_to_ESP8266(char *sendStr)
{   int i;
   // printf(" Sending: %s",sendStr);
    for(i=0;i<strlen(sendStr);i++)
        MAP_UART_transmitData(EUSCI_A2_BASE,sendStr[i]);
}

/*Function to reset Buffer*/
void resetBuffer()
{
    memset(receivingBuffer,0,2500);
    bufferposition = 0;
}

/*Function used to display data in the LCD screen*/
void displayData(){
    if(screen==screen1) //when the backlight of display is ON
        if(gotdata==1)
        {
            gotdata=2;
            ST7735_FillScreen(ST7735_BLACK);

        }
       writecommand(ST7735_DISON); //to turn on the display
       MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN5); //to turn on the backlight of the display

       /*Command to print values on the display*/
       display(0,6,"Temp:", ST7735_CYAN,ST7735_BLACK,1);
       display(8,6,temperature, ST7735_GREEN,ST7735_BLACK,1);
       display(13,6,"Cel", ST7735_GREEN,ST7735_BLACK,1);
       display(0,8,"Pressure:",ST7735_CYAN,ST7735_BLACK,1);
       display(8,8,pressure, ST7735_GREEN,ST7735_BLACK,1);
       display(13,8,"inHg", ST7735_GREEN,ST7735_BLACK,1);
       display(0,10,"Humidity:",ST7735_CYAN,ST7735_BLACK,1);
       display(8,10,humidity, ST7735_GREEN,ST7735_BLACK,1);
       display(13,10,"%rh", ST7735_GREEN,ST7735_BLACK,1);

       display(0,13, "Solar Panel:", ST7735_GREEN,ST7735_BLACK,1);
       display(0,15,"I:",ST7735_CYAN,ST7735_BLACK,1);
       display(2,15,"      ", ST7735_GREEN,ST7735_BLACK,1);
       display(2,15,current_solar, ST7735_GREEN,ST7735_BLACK,1);
       display(8,15, "mA", ST7735_GREEN,ST7735_BLACK,1);
       display(10,15,"V:",ST7735_CYAN,ST7735_BLACK,1);
       display(12,15,voltage, ST7735_GREEN,ST7735_BLACK,1);
       display(15,15,"V", ST7735_GREEN,ST7735_BLACK,1);
       display(0,17, "LTC:", ST7735_GREEN,ST7735_BLACK,1);
       display(0,19,"I:",ST7735_CYAN,ST7735_BLACK,1);
       display(2,19,"        ", ST7735_GREEN,ST7735_BLACK,1);
       display(2,19,current_ltc, ST7735_GREEN,ST7735_BLACK,1);
       display(8,19,"mA", ST7735_GREEN,ST7735_BLACK,1);

       display(10,19,"V:",ST7735_CYAN,ST7735_BLACK,1);
       display(12,19,voltage2, ST7735_GREEN,ST7735_BLACK,1);
       display(15,19,"V", ST7735_GREEN,ST7735_BLACK,1);

       display(0,21, "Battery:", ST7735_GREEN,ST7735_BLACK,1);
       display(0,23,"V:",ST7735_CYAN,ST7735_BLACK,1);
       display(2,23,voltage3, ST7735_GREEN,ST7735_BLACK,1);
       display(6,23,"V", ST7735_GREEN,ST7735_BLACK,1);

       MAP_PCM_gotoLPM0(); //send the system to Low Power Mode 0

         }

           else if(screen== screen2) //when backlight of dispaly is OFF
           {
             MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN5); //set the backlight OFF
             writecommand(ST7735_DISPOFF); //set the display OFF
              MAP_PCM_gotoLPM3(); //send the system to Low Power Mode 3 mode
        }
}

/* To initialize the Chip Select*/
void CS_Init(void)
{
    FlashCtl_setWaitState(FLASH_BANK0, 2);
    FlashCtl_setWaitState(FLASH_BANK1, 2);
    PCM_setCoreVoltageLevel(PCM_VCORE1);
    CS_setDCOCenteredFrequency(CS_DCO_FREQUENCY_24);
}

/*ESP8266 HardReset*/
void ESP8266_HardReset(void)
{
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN5);
    DelayWait10ms(10);
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN5);
}

/*Main function
*/

void main(void)
{

        WDTCTL = WDTPW | WDTHOLD;        // Stop watchdog timer
        boardInit();                     //intialization for GPIO pins as output
        clk_init();                      //clock initialization
        port6_init();
        CS_Init();
        port1_init();

        rtc_init();                      //initialize rtc clock
        MAP_Interrupt_enableMaster();
        adcInit();                       //initialize adc
        bme280Init();                    //initialize bme
        ST7735_InitR(INITR_REDTAB);      //enabling LCD

        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN5); //gpio pin for controlling ST7735 display
        backlight
        MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN5); //gpio pin for reset of ESP8266
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN4);//gpio pin to control ch_pd signal

        uartA2_init();
        DelayWait10ms(5);

        CurrentState = CheckAdc;


         while(1)
            {

             /*State Machine for different cases*/
             switch(CurrentState){

                case CheckAdc:
                {
                  getSensorValues(&temperature,&pressure,&humidity,&current_solar, &voltage, &current_ltc,
        &voltage2, &voltage3, &current_battery);
                    DelayWait10ms(5);
                    batteryVoltage= (int)(voltage3[0]-48);
                    /* Check if battery volatge is enought to starupthe system*/
                    if(batteryVoltage>=3.7)
                    {
                        CurrentState=  ESPReset;
                    }
                   else
                    {
                        CurrentState= END;
                       less_battery=1;
                    }

                    resetBuffer();
                    break;

                }
                case ESPReset:
                {
                    {
                        /*Hard Reset ESP8266*/
                        ESP8266_HardReset();
                        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN4);
                        DelayWait10ms(5);
                        ST7735_FillScreen(ST7735_BLACK);
                        CurrentState =ConnectionSetUp;
                        connection_entered=1;
                    }

                    resetBuffer();
                    break;

                  }

                    case ConnectionSetUp:
                         {
                         less_battery=2;
                         DelayWait10ms(5);

                          display(0,7,"Loading",ST7735_GREEN,ST7735_BLACK,2);

                          send_to_ESP8266("AT\r\n");
                          DelayWait10ms(5);
                          if(checkforOK())
                          {
                              send_to_ESP8266("AT+RST\r\n");
                              CurrentState = SetESPMode;
                              DelayWait10ms(5);

                          }
                          resetBuffer();
                          break;
                         }

                    case SetESPMode:
                    {
                        send_to_ESP8266("AT+CWMODE=1\r\n");

                        DelayWait10ms(5);
                        if(checkforOK())
                          {
                               CurrentState = ConnectToRouter;

                              DelayWait10ms(5);
                          }
                        resetBuffer();

                          break;
                    }
                    /*Connect to the Router*/
                    case ConnectToRouter:
                    {
                        sprintf(at_commands,"AT+CWJAP=\"%s\",\"%s\"\r\n",SSID,password);
                        send_to_ESP8266(at_commands);
                        DelayWait10ms(2000);
                        if(checkforOK())
                        {
                            CurrentState = TCPConnect;
                            DelayWait10ms(200);

                        }
                        resetBuffer();
                        break;

                    }

                    /*TCP Connection*/
                    case TCPConnect:
                    {
                        while(!timeset){
                            send_to_ESP8266("AT+CIPSTART=\"TCP\",\"time.nist.gov\",13\r\n");
                            DelayWait10ms(200);

                            if (alreadyConnected())
                            {
                                send_to_ESP8266("AT+CIPCLOSE\r\n");
                            }
                            if(checkforClosed())
                            {
                                updateDateAndTime();
                            }
                            else
                            {
                                DelayWait10ms(300);
                            }

                    }
                        resetBuffer();
                        break;
                    }

                    case ConnectSpreadSheet:
                    {
                      /*Check if battery voltage is above 3.7 before transmission to Google Spreadsheet*/
                        if( batteryVoltage< 3,7)
                        {
                            CurrentState= END;
                            resetBuffer();
                            break;
                        }

                        clk_init();
                        uartA2_init();

                        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN4);

                         /*Get the readings from the sensor, voltage and current*/

                         getSensorValues(&temperature,&pressure,&humidity,&current_solar, &voltage,
                         &current_ltc, &voltage2, &voltage3, &current_battery);

                         DelayWait10ms(5);


                         /* Send TCP connection request to pushingbox API*/
                        send_to_ESP8266("AT+CIPSTART=\"TCP\",\"api.pushingbox.com\",80\r\n");
                        DelayWait10ms(200);
                        if(checkforOK())
                        {
                            CurrentState = UpdateData;
                            DelayWait10ms(5);
                        }
                        resetBuffer();
                        break;

                     }

                    case UpdateData:
                    {

                         sprintf(get_command,"GET
                         /pushingbox?devid=%s&humidityData=%s&celData=%s&fehrData=%s&voltageSolar=%s&curre
                         ntSolar=%s&voltageLTC=%s&currentLTC=%s&voltageBattery=%s HTTP/1.1\r\nHost:
                         api.pushingbox.com\r\nUser-Agent: ESP8266/1.0\r\nConnection:close
                         \r\n\r\n",deviceId,humidity,temperature,pressure,voltage,current_solar,voltage2,
                         current_ltc,voltage3);

                        /* Send values to the pushing box API */
                        sprintf(at_commands,"AT+CIPSEND=%d\r\n",strlen(get_command));
                        send_to_ESP8266(at_commands);
                        DelayWait10ms(10);
                        send_to_ESP8266(get_command);
                        DelayWait10ms(200);
```

```
                        /*Set the CH_PD low after transmission of readings*/
                        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN4);
                        connection_entered=0;
                        spreadsheet=0;
                        gotdata=1;
                            CurrentState = END;
                        resetBuffer();
                        break;
                    }

                    case END:
                    {



                        /*Get sensor, voltage and current readings*/

                         getSensorValues(&temperature,&pressure,&humidity,&current_solar, &voltage,
                         &current_ltc, &voltage2, &voltage3, &current_battery);
                        batteryVoltage= (int)(voltage3[0]-48);
                       /*Condition check when the system is not transmitting*/
                        if(less_battery==1)
                        {
                            if(batteryVoltage>=3.7)
                            {
                                CurrentState= ESPReset; //reset state
                                writecommand(ST7735_DISON);
                                MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN5);
                                DelayWait10ms(5);
                                resetBuffer();
                                break;
                            }
                        }
                        displayData();
                        break;
                }
                    default:
                        break;
                }
            }
}

/*RTC clock interrupt handler function
//RTC interrupt is used to time keeping for different purpose of this research. It counts for 20 seconds
for turning the backlight off.
//Minute interrupt is used for counting 60 minutes for posting data to the spreadsheet.
*/
void RTC_C_IRQHandler(void)
{
    uint32_t status;

    status = MAP_RTC_C_getEnabledInterruptStatus();
    MAP_RTC_C_clearInterruptFlag(status);


    if (status & RTC_C_CLOCK_READ_READY_INTERRUPT)//taking interrupt in each second
    {
        currentTime = MAP_RTC_C_getCalendarTime();//storing the updated value
        timeBlinker = !timeBlinker;
        if(connection_entered!=1)
             {
            if(spreadsheet!=1)
            {
            timerDelay++;

        if(timerDelay==2)
        {
           timerDelay=0;
```

```
            CurrentState=END;
         }
            }
        screenOff++;
             if(screenOff==20)
             {
               screenOff=0;
            if(screen== screen1)
            {
               screen=screen2;
              }
               }
            }
       }


    if (status & RTC_C_TIME_EVENT_INTERRUPT)
    {//taking interrupt in each minute
        twoSecondToggle++; //counter to count 60 seconds
        if(twoSecondToggle == 60)
        {
            twoSecondToggle = 0;
            if(less_battery!=1)
              {
                spreadsheet= 1;
                CurrentState = ConnectSpreadSheet;
              }
        }
    }

    if (status & RTC_C_CLOCK_ALARM_INTERRUPT)//interrupts when time reaches alarm time
    {

    }

}


/* GPIO ISR
This interrupt handler is used to receive an interrupt from the GPIO pin in Port 1.
Based on the interrupt form the Pin1 and Pin4 the LCD backlight display is turned on and off.
*/

void PORT1_IRQHandler(void)
{
    uint32_t status;
    status = MAP_GPIO_getEnabledInterruptStatus(GPIO_PORT_P1);
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P1, status);

    /* Toggling the output on the LED */
    if(status & GPIO_PIN1)

    {
        if(screen==screen1)
        {
            screen= screen2;
        }

    }

    if(status & GPIO_PIN4)
    {
        if(screen==screen2)
        {
            screen=screen1;
        }
    }
}
```

BMEINTERFACING.C

```c
/*
 * bme280nterfacing.c
 *
 *  Created on: Mar 8, 2017
 *      Author: Rajan
 *
 */
#include "bme280Interfacing.h"
#include "driverlib.h"
#include <string.h>
#include <math.h>
#include <stdio.h>


s8 BME280_I2C_bus_write(u8 dev_addr, u8 reg_addr, u8 *reg_data, u8 cnt)
{
    s32 iError = BME280_INIT_VALUE;
    u8 array[I2C_BUFFER_LEN];
    u8 stringpos = BME280_INIT_VALUE;
    array[BME280_INIT_VALUE] = reg_addr;
    uint8_t i;
    uint16_t rtnval, debugdump;

    for (stringpos = BME280_INIT_VALUE; stringpos < cnt; stringpos++) {
        array[stringpos + BME280_DATA_INDEX] = *(reg_data + stringpos);
    }
    /*
    * Please take the below function as your reference for
    * write the data using I2C communication
    * "IERROR = I2C_WRITE_STRING(DEV_ADDR, array, cnt+1)"
    * add your I2C write function here
    * iError is an return value of I2C read function
    * Please select your valid return value
    * In the driver SUCCESS defined as 0
    * and FAILURE defined as -1
    * Note :
    * This is a full duplex operation,
    * The first read data is discarded, for that extra write operation
    * have to be initiated. For that cnt+1 operation done in the I2C write string function
    * For more information please refer data sheet SPI communication:
    */

    while(UCB1STATW&0x0010){};         // wait for I2C ready
    UCB1CTLW0 |= 0x0001;               // hold the eUSCI module in reset mode
    UCB1TBCNT = cnt+1;                     // generate stop condition after this many bytes
    UCB1CTLW0 &= ~0x0001;              // enable eUSCI module
    UCB1I2CSA = dev_addr;              // I2CCSA[6:0] is slave address
    UCB1CTLW0 = ((UCB1CTLW0&~0x0004)   // clear bit2 (UCTXSTP) for no transmit stop condition
                                       // set bit1 (UCTXSTT) for transmit start condition
                  | 0x0012);           // set bit4 (UCTR) for transmit mode
    while((UCB1IFG&0x0002) == 0){};    // wait for slave address sent

    for(i=0; i<cnt; i++) {
        UCB1TXBUF = array[i]&0xFF;         // TXBUF[7:0] is data
        while((UCB1IFG&0x0002) == 0){      // wait for data sent
            if(UCB1IFG&0x0030){              // bit5 set on not-acknowledge; bit4 set on arbitration lost
                debugdump = UCB1IFG;           // snapshot flag register for calling program
                I2C_Init();                    // reset to known state
                return iError=-1;
            }
        }
    }
    UCB1TXBUF = array[i]&0xFF;         // TXBUF[7:0] is last data
    while(UCB1STATW&0x0010){            // wait for I2C idle
      if(UCB1IFG&0x0030){               // bit5 set on not-acknowledge; bit4 set on arbitration lost
        debugdump = UCB1IFG;            // snapshot flag register for calling program
        I2C_Init();                     // reset to known state
        return iError=-1;
      }
    }
    return iError=0;
}



/* \Brief: The function is used as I2C bus read
*  \Return : Status of the I2C read
*  \param dev_addr : The device address of the sensor
*  \param reg_addr : Address of the first register, will data is going to be read
*  \param reg_data : This data read from the sensor, which is hold in an array
*  \param cnt : The no of data byte of to be read
*/
s8 BME280_I2C_bus_read(u8 dev_addr, u8 reg_addr, u8 *reg_data, u8 cnt)
{
   s32 iError = BME280_INIT_VALUE;
   uint8_t i;
   uint16_t rtnval, debugdump;
   u8 array[I2C_BUFFER_LEN] = {BME280_INIT_VALUE};
   u8 stringpos = BME280_INIT_VALUE;
   array[BME280_INIT_VALUE] = reg_addr;
   /* Please take the below function as your reference
    * for read the data using I2C communication
    * add your I2C read function here.
    * "IERROR = I2C_WRITE_READ_STRING(DEV_ADDR, ARRAY, ARRAY, 1, CNT)"
    * iError is an return value of write function
    * Please select your valid return value
    * In the driver SUCCESS defined as 0
    * and FAILURE defined as -1
    */

   // set pointer to register address
   while(UCB1STATW&0x0010){};          // wait for I2C ready
   UCB1CTLW0 |= 0x0001;                // hold the eUSCI module in reset mode
   UCB1TBCNT = 1;                      // generate stop condition after this many bytes
   UCB1CTLW0 &= ~0x0001;               // enable eUSCI module
   UCB1I2CSA = dev_addr;                  // I2CCSA[6:0] is slave address
   UCB1CTLW0 = ((UCB1CTLW0&~0x0004)    // clear bit2 (UCTXSTP) for no transmit stop condition
                                       // set bit1 (UCTXSTT) for transmit start condition
                 | 0x0012);            // set bit4 (UCTR) for transmit mode
   while(UCB1CTLW0&0x0002){};          // wait for slave address sent
   UCB1TXBUF = reg_addr&0xFF;             // TXBUF[7:0] is data
   while(UCB1STATW&0x0010){            // wait for I2C idle
     if(UCB1IFG&0x0030){               // bit5 set on not-acknowledge; bit4 set on arbitration lost
       debugdump = UCB1IFG;            // snapshot flag register for calling program
       I2C_Init();                     // reset to known state
       return iError=-1;
     }
   }

   // receive bytes from registers on BME280 device

   while(UCB1STATW&0x0010){};          // wait for I2C ready
   UCB1CTLW0 |= 0x0001;                // hold the eUSCI module in reset mode
   UCB1TBCNT = cnt;                      // generate stop condition after this many bytes
   UCB1CTLW0 &= ~0x0001;               // enable eUSCI module
   UCB1I2CSA = dev_addr;                  // I2CCSA[6:0] is slave address
   UCB1CTLW0 = ((UCB1CTLW0&~0x0014)    // clear bit4 (UCTR) for receive mode
                                       // clear bit2 (UCTXSTP) for no transmit stop condition
                 | 0x0002);            // set bit1 (UCTXSTT) for transmit start condition
   for(i=0; i<cnt; i++) {
       while((UCB1IFG&0x0001) == 0){       // wait for complete character received
         if(UCB1IFG&0x0030){               // bit5 set on not-acknowledge; bit4 set on arbitration lost
```

```
          I2C_Init();                    // reset to known state
          return 0xFFFF;
        }
      }
      *reg_data++= UCB1RXBUF&0xFF;            // get the reply
   }

   return (s8)iError;
}

/*  Brief : The delay routine
*  \param : delay in ms
*/

/* delay */
void BME280_delay_msek(u32 msek)
{
 Delay1ms(10);
 }


/*-------------------------------------------------------------------------*
*   The following function is used to map the I2C bus read, write, delay and
*   device address with global structure bme280
*-------------------------------------------------------------------------*/
s8 I2C_routine(void) {
/*-------------------------------------------------------------------------*
*  By using bme280 the following structure parameter can be accessed
*  Bus write function pointer: BME280_WR_FUNC_PTR
*  Bus read function pointer: BME280_RD_FUNC_PTR
*  Delay function pointer: delay_msec
*  I2C address: dev_addr
*--------------------------------------------------------------------------*/
   bme280.bus_write = BME280_I2C_bus_write;
   bme280.bus_read = BME280_I2C_bus_read;
   bme280.dev_addr = BME280_I2C_ADDRESS1;
   bme280.delay_msec = BME280_delay_msek;

   return BME280_INIT_VALUE;
}


void bme280Init()
{
    I2C_Init();  // initialize eUSCI
    memset(resultsBuffer,0x00, 16);

        /* The variable used to assign the standby time*/
        v_stand_by_time_u8 = BME280_INIT_VALUE;
        /* The variable used to read uncompensated temperature*/
         v_data_uncomp_temp_s32 = BME280_INIT_VALUE;
        /* The variable used to read uncompensated pressure*/
         v_data_uncomp_pres_s32 = BME280_INIT_VALUE;
        /* The variable used to read uncompensated pressure*/
         v_data_uncomp_hum_s32 = BME280_INIT_VALUE;
        /* The variable used to read compensated temperature*/
         u32 v_comp_temp_s32[2] = {BME280_INIT_VALUE, BME280_INIT_VALUE};
        /* The variable used to read compensated pressure*/
         u32 v_comp_press_u32[2] = {BME280_INIT_VALUE, BME280_INIT_VALUE};
        /* The variable used to read compensated humidity*/
         u32 v_comp_humidity_u32[2] = {BME280_INIT_VALUE, BME280_INIT_VALUE};

        /* result of communication results*/
        s32  com_rslt = ERROR;

     /*********************** START INITIALIZATION ************************/
      /*    Based on the user need configure I2C or SPI interface.
      * It is example code to explain how to use the bme280 API*/
        I2C_routine();
        /*SPI_routine();*/
    /*---------------------------------------------------------------------------*
     *  This function used to assign the value/reference of
     *  the following parameters
     *  I2C address
     *  Bus Write
     *  Bus read
     *  Chip id
    *-------------------------------------------------------------------------*/
        com_rslt = bme280_init(&bme280);

        /*  For initialization it is required to set the mode of
         *  the sensor as "NORMAL"
         *  data acquisition/read/write is possible in this mode
         *  by using the below API able to set the power mode as NORMAL*/
        /* Set the power mode as NORMAL*/
        com_rslt += bme280_set_power_mode(BME280_NORMAL_MODE);
        /*  For reading the pressure, humidity and temperature data it is required to
         *  set the OSS setting of humidity, pressure and temperature
         * The "BME280_CTRLHUM_REG_OSRSH" register sets the humidity
         * data acquisition options of the device.
         * changes to this registers only become effective after a write operation to
         * "BME280_CTRLMEAS_REG" register.
         * In the code automated reading and writing of "BME280_CTRLHUM_REG_OSRSH"
         * register first set the "BME280_CTRLHUM_REG_OSRSH" and then read and write
         * the "BME280_CTRLMEAS_REG" register in the function*/
        com_rslt += bme280_set_oversamp_humidity(BME280_OVERSAMP_1X);

        /* set the pressure oversampling*/
        com_rslt += bme280_set_oversamp_pressure(BME280_OVERSAMP_1X);
        /* set the temperature oversampling*/
        com_rslt += bme280_set_oversamp_temperature(BME280_OVERSAMP_1X);
    /*---------------------------------------------------------------------------*/
    /*-------------------------------------------------------------------------*
    ************************* START GET and SET FUNCTIONS DATA ****************
    *---------------------------------------------------------------------------*/
        /* This API used to Write the standby time of the sensor input
         *  value have to be given
         *  Normal mode comprises an automated perpetual cycling between an (active)
         *  Measurement period and an (inactive) standby period.
         *  The standby time is determined by the contents of the register t_sb.
         *  Standby time can be set using BME280_STANDBYTIME_125_MS.
         *  Usage Hint : bme280_set_standbydur(BME280_STANDBYTIME_125_MS)*/

      com_rslt += bme280_set_standby_durn(BME280_STANDBY_TIME_1_MS);
    //    com_rslt += bme280_set_standby_durn(BME280_STANDBY_TIME_1000_MS);

        /* This API used to read back the written value of standby time*/
        com_rslt += bme280_get_standby_durn(&v_stand_by_time_u8);
    /*-----------------------------------------------------------------*
    ************************* END GET and SET FUNCTIONS ****************
    *------------------------------------------------------------------*/
        // get registers

        BME280_I2C_bus_read(p_bme280->dev_addr, 0xF2, regData, 12);
    /************************* END INITIALIZATION *************************/
        /*---------------------------------------------------------------------*
        ************************* START DE-INITIALIZATION **********************
        *-----------------------------------------------------------------------*/
            /*  For de-initialization it is required to set the mode of
             *  the sensor as "SLEEP"
             *  the device reaches the lowest power consumption only
             *  In SLEEP mode no measurements are performed
             *  All registers are accessible
             *  by using the below API able to set the power mode as SLEEP*/
             /* Set the power mode as SLEEP*/
            com_rslt += bme280_set_power_mode(BME280_SLEEP_MODE);
        /*-------------------------------------------------------------------*
        ************************* END DE-INITIALIZATION *********************
```

```
        *---------------------------------------------------------------------*/

}

void getSensorValues(char *temperature,char *pressure,char *humidity, char *light, char *voltage, char
*light2, char *voltage2, char *voltage3, char *light3)
{
    com_rslt= bme280_set_power_mode(0x01); // set power mode to forced to generate reading
    Delay1ms(10);
    if(timerDelay==0)
    {

    // get registers

    BME280_I2C_bus_read(p_bme280->dev_addr, 0xF2, regData, 12);

    /*---------------------------------------------------------------------*
    ************ START READ UNCOMPENSATED PRESSURE, TEMPERATURE
    AND HUMIDITY DATA ********
    *---------------------------------------------------------------------*/
        /* API is used to read the uncompensated temperature*/
        com_rslt += bme280_read_uncomp_temperature(&v_data_uncomp_temp_s32);

        /* API is used to read the uncompensated pressure*/
        com_rslt += bme280_read_uncomp_pressure(&v_data_uncomp_pres_s32);

        /* API is used to read the uncompensated humidity*/
        com_rslt += bme280_read_uncomp_humidity(&v_data_uncomp_hum_s32);

        /* API is used to read the uncompensated temperature,pressure
        and humidity data */
//            com_rslt += bme280_read_uncomp_pressure_temperature_humidity(
//            &v_data_uncomp_temp_s32, &v_data_uncomp_pres_s32, &v_data_uncomp_hum_s32);
    /*---------------------------------------------------------------------*
    ************ END READ UNCOMPENSATED PRESSURE AND TEMPERATURE********
    *---------------------------------------------------------------------*/

    /*---------------------------------------------------------------------*

    ************ START READ COMPENSATED PRESSURE, TEMPERATURE
    AND HUMIDITY DATA ********
    *---------------------------------------------------------------------*/
        /* API is used to compute the compensated temperature*/
        v_comp_temp_s32[0] = bme280_compensate_temperature_int32(
                v_data_uncomp_temp_s32);

        /* API is used to compute the compensated pressure*/
        v_comp_press_u32[0] = bme280_compensate_pressure_int32(
                v_data_uncomp_pres_s32);

        /* API is used to compute the compensated humidity*/
        v_comp_humidity_u32[0] = bme280_compensate_humidity_int32(
                v_data_uncomp_hum_s32);

        /* API is used to read the compensated temperature, humidity and pressure*/
//            com_rslt += bme280_read_pressure_temperature_humidity(
//            &v_comp_press_u32[1], &v_comp_temp_s32[1],  &v_comp_humidity_u32[1]);
    /*---------------------------------------------------------------------*
    ************ END READ COMPENSATED PRESSURE, TEMPERATURE AND HUMIDITY ********
    *---------------------------------------------------------------------*/
        uint32_t t = v_comp_temp_s32[0];
        uint32_t p = v_comp_press_u32[0];
        uint32_t h = v_comp_humidity_u32[0];
    /*---------------------------------------------------------------------*/
        pressureCompensation = v_comp_press_u32[0] + 2340;

        temperature_bme280 = (float) v_comp_temp_s32[0] * 0.01;
        pressure_bme280 = (float) pressureCompensation *0.01 * 0.02953;
        humidity_bme280 = (float) v_comp_humidity_u32[0] /1024;
        sprintf(temperature,"%0.2f",temperature_bme280);
        sprintf(pressure,"%0.2f",pressure_bme280);
        sprintf(humidity,"%0.2f",humidity_bme280);

        MAP_ADC14_toggleConversionTrigger();

       sprintf(light,"%0.2f",current);
       sprintf(voltage,"%0.1f",voltage_read);
       sprintf(light2,"%0.2f",current2);
       sprintf(voltage2,"%0.1f",voltage_read2);
       sprintf(voltage3,"%0.1f",battery_volt);
       sprintf(light3,"%0.2f",current3);
       timerDelay++;
    }
}


/* ADC Interrupt Handler. This handler is called whenever there is a conversion
 * that is finished for ADC_MEM0.
 * The converted value is stored in different buffer for required conversion process
 */
void ADC14_IRQHandler(void)
{
    uint64_t status = MAP_ADC14_getEnabledInterruptStatus();
    MAP_ADC14_clearInterruptFlag(status);

    if (ADC_INT8 & status)
    {
        MAP_ADC14_getMultiSequenceResult(resultsBuffer);
      //  flag=1;

        /* Getting values for current and voltage from the result buffer*/
        curADCResult0= (int16_t)resultsBuffer[0]-8192;

        current= (float)((((curADCResult0*3.3)/8192)*1000));

        if(current<0)
        {
            current=0;
        }

        curADCResult1= (int16_t)resultsBuffer[4]-8192;

        current2= (float)((((curADCResult1*3.3)/8192)*1000));
        if(current2<=0)
        {
                  current2=0;
        }

        current3= (float)(current - current2);
        voltage_read= (resultsBuffer[6]*3.3)/16384;
        voltage_read2= (resultsBuffer[7]*3.3)/16384;
        voltage_read3= (resultsBuffer[8]*3.3)/16384;
        battery_volt= 1.1045*voltage_read3+1.6589;
                  }
}
```

ADC.C

```
/*
 * adc.c
 *
 *  Created on: Jul 27, 2018
 *      Author: Rajan
 */


#include "clk.h"
#include "driverlib.h"

void adcInit(){

    /* Initializing ADC (MCLK/1/4) */
        MAP_ADC14_enableModule();
        MAP_ADC14_initModule(ADC_CLOCKSOURCE_MCLK , ADC_PREDIVIDER_4, ADC_DIVIDER_6,
                0);

        /* Configuring ADC Memory in Multisequence mode */
        MAP_ADC14_configureMultiSequenceMode (ADC_MEM0, ADC_MEM8, false);


        /* Setting up GPIO pins as analog inputs */
                /* Pin 5 & Pin 4 for getting current for the Solar, Pin 0 & Pin 1 for measuring current
        from LTC */
        MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                    GPIO_PIN5 | GPIO_PIN4| GPIO_PIN0| GPIO_PIN1, GPIO_TERTIARY_MODULE_FUNCTION);

        /* Setting up GPIO pins as analog inputs */
        /* Port 4, pin 5,6,7 for measuring voltage from solar, ltc and battery */
           MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P4,
                    GPIO_PIN6 | GPIO_PIN7| GPIO_PIN5, GPIO_TERTIARY_MODULE_FUNCTION);


    /* Setting A0 & A1 and A4& A5 in differential mode */
    MAP_ADC14_configureConversionMemory(ADC_MEM0, ADC_VREFPOS_AVCC_VREFNEG_VSS  ,
    ADC_INPUT_A0, true);

    MAP_ADC14_configureConversionMemory(ADC_MEM4, ADC_VREFPOS_AVCC_VREFNEG_VSS  ,
    ADC_INPUT_A4, true);


    /* Setting A6, A7 & A8 in single ended mode */
    MAP_ADC14_configureConversionMemory(ADC_MEM6, ADC_VREFPOS_AVCC_VREFNEG_VSS  ,
    ADC_INPUT_A6, false);

    MAP_ADC14_configureConversionMemory(ADC_MEM7, ADC_VREFPOS_AVCC_VREFNEG_VSS  ,
        ADC_INPUT_A7, false);

    MAP_ADC14_configureConversionMemory(ADC_MEM8, ADC_VREFPOS_AVCC_VREFNEG_VSS  ,
           ADC_INPUT_A8, false);

    /* Configuring Sample Timer */
    MAP_ADC14_enableSampleTimer(ADC_AUTOMATIC_ITERATION);

    /* Enabling interrupts */
    MAP_ADC14_enableInterrupt(ADC_INT8);

    /* Enabling/Toggling Conversion */
    MAP_ADC14_enableConversion();
    MAP_ADC14_toggleConversionTrigger();
    MAP_Interrupt_enableInterrupt(INT_ADC14);

}
```

CLK.C

```
* clk.c
 *
 *  Created on: Feb 7, 2018
 *      Author: Rajan
 */


#include "driverlib.h"
#include "debug.h"


uint32_t g_SMCLKfreq;
uint32_t g_MCLKfreq;

static void clockInit48MHzXTL(void)
{  // sets the clock module to use the external 48 MHz crystal

    /* Configuring pins for peripheral/crystal usage */
    MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_PJ,
            GPIO_PIN3 | GPIO_PIN2, GPIO_PRIMARY_MODULE_FUNCTION);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);

    CS_setExternalClockSourceFrequency(32000,48000000); // enables getMCLK, getSMCLK to know externally
set frequencies

    /* Starting HFXT in non-bypass mode without a timeout. Before we start
     * we have to change VCORE to 1 to support the 48MHz frequency */
    MAP_PCM_setCoreVoltageLevel(PCM_VCORE1);
    MAP_FlashCtl_setWaitState(FLASH_BANK0, 2);
    MAP_FlashCtl_setWaitState(FLASH_BANK1, 2);
    CS_startHFXT(false);  // false means that there are no timeouts set, will return when stable

    /* Initializing MCLK to HFXT (effectively 48MHz) */
    MAP_CS_initClockSignal(CS_MCLK, CS_HFXTCLK_SELECT, CS_CLOCK_DIVIDER_1);
}

void clk_init(){
    clockInit48MHzXTL();  // set up the clock to use the crystal oscillator on the Launchpad
    MAP_CS_initClockSignal(CS_MCLK, CS_HFXTCLK_SELECT, CS_CLOCK_DIVIDER_1);    /* MCLK = 48 Mhz*/
    MAP_CS_initClockSignal(CS_SMCLK, CS_HFXTCLK_SELECT, CS_CLOCK_DIVIDER_4); /* SMCLK = 48/4 = 12Mhz*/
    g_SMCLKfreq=MAP_CS_getSMCLK();  // get SMCLK value to verify it was set correctly
    g_MCLKfreq=MAP_CS_getMCLK();  // get MCLK value
   }
/*
```

UART.C

```
/*
 * uart.c
 *
 *  Created on: Feb 21, 2018
 *      Author: Rajan
 */

#include "driverlib.h"

const eUSCI_UART_Config uartConfig =
{
        EUSCI_A_UART_CLOCKSOURCE_SMCLK,          // SMCLK Clock Source
        6,                                      // BRDIV = 26
        8,                                       // UCxBRF = 0
        0x11,                                       // UCxBRS = 0
        EUSCI_A_UART_NO_PARITY,                  // No Parity
        EUSCI_A_UART_LSB_FIRST,                  // MSB First
        EUSCI_A_UART_ONE_STOP_BIT,               // One stop bit
        EUSCI_A_UART_MODE,                       // UART mode
        EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION  // Low Frequency Mode
};


void uartA0_init()
```

```
{
    /* Selecting P1.2 and P1.3 in UART mode. */
      MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1,
            GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);

       /* Configuring UART Module */
       MAP_UART_initModule(EUSCI_A0_BASE, &uartConfig);

       /* Enable UART module */
       MAP_UART_enableModule(EUSCI_A0_BASE);

       UART_enableInterrupt(EUSCI_A0_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT);
       Interrupt_enableInterrupt(INT_EUSCIA0);

}


void uartA2_init()
{
          /* Selecting P3.2 and P3.3 in UART mode. */
           MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P3,
                   GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);
           /* Configuring UART Module */
           MAP_UART_initModule(EUSCI_A2_BASE, &uartConfig);

           /* Enable UART module */
           MAP_UART_enableModule(EUSCI_A2_BASE);

           UART_enableInterrupt(EUSCI_A2_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT);
           Interrupt_enableInterrupt(INT_EUSCIA2);

           /* Configuring P1.1 as an input and enabling interrupts */
              MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);
              MAP_GPIO_clearInterruptFlag(GPIO_PORT_P1, GPIO_PIN1);
              MAP_GPIO_enableInterrupt(GPIO_PORT_P1, GPIO_PIN1);
              MAP_Interrupt_enableInterrupt(INT_PORT1);

}
```

RTC.C

```
/*
 * rtc.c
 *
 *  Created on: Feb 21, 2018
 *      Author: Rajan
 */

#include "driverlib.h"

void rtc_init()
{

    /* Specify an interrupt to assert every minute */
    MAP_RTC_C_setCalendarEvent(RTC_C_CALENDAREVENT_MINUTECHANGE);

    /* Enable interrupt for RTC Ready Status, which asserts when the RTC
     * Calendar registers are ready to read.
     * Also, enable interrupts for the Calendar alarm and Calendar event. */
    MAP_RTC_C_clearInterruptFlag(
            RTC_C_CLOCK_READ_READY_INTERRUPT | RTC_C_TIME_EVENT_INTERRUPT
                    | RTC_C_CLOCK_ALARM_INTERRUPT);
    MAP_RTC_C_enableInterrupt(
            RTC_C_CLOCK_READ_READY_INTERRUPT | RTC_C_TIME_EVENT_INTERRUPT
                    | RTC_C_CLOCK_ALARM_INTERRUPT);

    MAP_Interrupt_enableInterrupt(INT_RTC_C);
    RTC_C->AMINHR= 0x00;
    MAP_RTC_C_startClock();
}
```

GPIO.C

```
/*
 * gpio.c
 *
 *  Created on: Feb 21, 2018
 *      Author: Rajan
 */

#include "driverlib.h"
#include <stdio.h>


void port6_init(void)
{
    /* Selecting P6.4 for Reset*/
     MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P6, GPIO_PIN4);

     MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6,
                      GPIO_PIN4 + GPIO_PIN5, GPIO_PRIMARY_MODULE_FUNCTION);
}


void port1_init(void)
{
    /* Configuring P1.1 as an input and enabling interrupts */
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1| GPIO_PIN4);
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P1, GPIO_PIN1|GPIO_PIN4);
    MAP_GPIO_enableInterrupt(GPIO_PORT_P1, GPIO_PIN1|GPIO_PIN4);
    MAP_Interrupt_enableInterrupt(INT_PORT1);
    /* Configuring P3.0 as output and P3.5 (switch) for LCD display intensity control */
       MAP_GPIO_setAsOutputPin(GPIO_PORT_P3, GPIO_PIN5);
}


void boardInit(void)
{
    // GPIO Port Configuration for lowest power configuration
        P1->OUT = 0x00; P1->DIR = 0xFF;
        P2->OUT = 0x00; P2->DIR = 0xFF;
        P3->OUT = 0x00; P3->DIR = 0xFF;
       // P4->OUT = 0x00; P4->DIR = 0xFF;
       // P5->OUT = 0x00; P5->DIR = 0xFF;
        P6->OUT = 0x00; P6->DIR = 0xFF;
        P7->OUT = 0x00; P7->DIR = 0xFF;
        P8->OUT = 0x00; P8->DIR = 0xFF;
        P9->OUT = 0x00; P9->DIR = 0xFF;
        P10->OUT = 0x00; P10->DIR = 0xFF;
        PJ->OUT = 0x00; PJ->DIR = 0xFF;
}
```

Some other codes used:

- BME.c [2016 Bosch Sesortech GmBh)
- ST7735.c [Adrafruit 1.8” SPI library)

References:

[1] S. Roundy, P. K. Wright, and J. M. Rabaey, “Energy scavenging for wireless sensor networks,” *Norwell*, pp. 45–47, 2003.

[2] S. Philipps, F. Ise, W. Warmuth, P. Conferences, and C. GmbH, “PHOTOVOLTAICS REPORT,” 2018. [Online]. Available: www.ise.fraunhofer.de. [Accessed: 28-Aug-2018].

[3] “Cap-XX, “GW1 Series – Double Layer Supercapacitor,” 2006.

[4] “MSP-EXP432P401R SimpleLink^{TM} MSP432P401R high-precision ADC LaunchPad^{TM} Development Kit | TI.com.” [Online]. Available: http://www.ti.com/tool/MSP-EXP432P401R. [Accessed: 27-Aug-2018].

[5] “MSP432P4xx SimpleLink^{TM} Microcontrollers Technical Reference Manual,” 2015.

[6] “TI-RSLK Texas Instruments Robotics System Learning Kit.”

[7] “Architecting a Smarter World – Arm.” [Online]. Available: https://www.arm.com/. [Accessed: 24-Oct-2018].

[8] “ESP8266EX Datasheet.” [Online]. Available: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[9] “ESP8266 WiFi Module Quick Start Guide.” [Online]. Available: https://www.mpja.com/download/esp8266 wifi module quick start guide v 1.0.4.pdf.

[10] “BME280.” [Online]. Available: https://www.bosch-sensortec.com/bst/products/all_products/bme280. [Accessed: 27-Aug-2018].

[11] “datasheet BME280.” [Online]. Available: https://www.digchip.com/datasheets/parts/datasheet/1727/BME280-pdf.php. [Accessed: 27-Aug-2018].

[12] “1.8 Color TFT LCD display with MicroSD Card Breakout [ST7735R] ID: 358 - $19.95 : Adafruit Industries, Unique &amp; fun DIY electronics and kits.” [Online]. Available: https://www.adafruit.com/product/358. [Accessed: 27-Aug-2018].

[13] L. Technology Corporation, “LTC3106 - 300mA Low Voltage Buck-Boost Converter with PowerPath and 1.6µA Quiescent Current.”

[14] “Working Principle of Photodiode, Characteristics And Applications.” [Online]. Available: https://www.elprocus.com/photodiode-working-principle-applications/. [Accessed: 28-Aug-2018].

[15] “User:Rfassbind - Wikimedia Commons.” [Online]. Available:

https://commons.wikimedia.org/wiki/User:Rfassbind. [Accessed: 28-Aug-2018].

[16] "How do batteries work? A simple introduction - Explain that Stuff." [Online]. Available: https://www.explainthatstuff.com/batteries.html. [Accessed: 28-Aug-2018].

[17] "Amazon.com: uxcell 5Pcs 2V 160mA Poly Mini Solar Cell Panel Module DIY for Phone Light Toys Charger 60mm x 60mm: Automotive." [Online]. Available: https://www.amazon.com/gp/product/B073Y3KFKV/ref=oh_aui_detailpage_o04_s00?ie=UTF8&psc=1. [Accessed: 21-Nov-2018].

[18] "Battery Reference Book - Thomas P J Crompton - Google Books." [Online]. Available: https://books.google.ca/books?id=QmVR7qiB5AUC&lpg=PA11&ots=ckHhIPVdcC&dq=battery one or more cells&pg=PA11#v=onepage&q&f=false. [Accessed: 28-Aug-2018].

[19] "EnergyTrace for MSP432 - Texas Instruments Wiki." [Online]. Available: http://processors.wiki.ti.com/index.php/EnergyTrace_for_MSP432. [Accessed: 28-Aug-2018].

[20] "Code Composer Studio Integrated Development Environment | TI.com." [Online]. Available: http://www.ti.com/tools-software/ccs.html. [Accessed: 28-Aug-2018].

[21] "How Do Breakpoints Work - Texas Instruments Wiki." [Online]. Available: http://processors.wiki.ti.com/index.php/How_Do_Breakpoints_Work. [Accessed: 29-Aug-2018].

[22] B. Finch and W. Goh MSP, "Application Report MSP430^{TM} Advanced Power Optimizations: ULP Advisor^{TM} Software and EnergyTrace^{TM} Technology," 2014.

[23] "Designing an Ultra-Low-Power (ULP) Application With SimpleLink^{TM} MSP432^{TM} Microcontrollers Application Report," 2015.

[24] "Model 1761, Triple Output DC Power Supplies - B&amp;K Precision." [Online]. Available: http://www.bkprecision.com/products/power-supplies/1761-4-digit-triple-output-dc-power-supply-20-35v-0-3a-12-65v-5a.html. [Accessed: 02-Dec-2018].

[25] "KR Supercapacitors Coin cells," 2016.

[26] L.-P. Battery and X. Li, "Li-Polymer Battery Technology Specification Customer Part name Wenfei liang," 2015.

[27] R. Aldridge and H. Jiao, "LTC3106 Evaluation Board," Grand Rapids, MI, 2018.

[28] "Optical &amp; medical equipment Halogen reflector Special Lamps Halogen reflector lamps-proven reliability."

[29] "Power Stream Li2450 datasheet."

[30] "MSP432 datasheet," 2015.