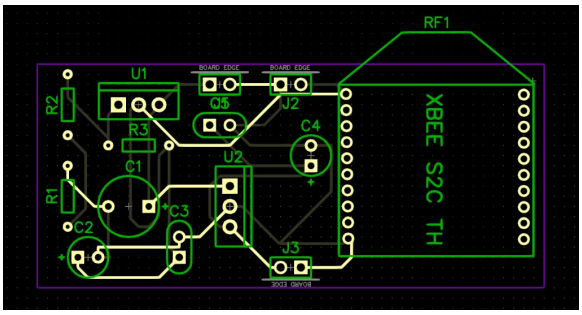
Oswego State University of New York
Electrical and Computer Engineering Department



ECE 492
Automated Home Security

By
Ashish Kharka, Jonathan Castillo, Nathan Loucks

Supervisor:
Dr. Mario Bkassiny

Submitted in Partial Fulfillment of the Requirement for the B.Sc. Degree,
Electrical and Computer Engineering Department

Spring 2020

I Acknowledgments

We would like to express our gratitude to Dr. Mario Bkassiny, for his continuous support and guidance.

Our sincere thanks goes to Dennis Quill for his help and advice on parts as well as his expertise working with Raspberry Pi devices.

Finally we would like to thank the ECE department as a whole for providing the resources necessary to make this research possible despite having to work remotely due to the unique circumstances of the spring 2020 semester. Without the planning and preparation made by the ECE department this project and many others would not have been possible.

Abstract

Abstract — Modern security systems more often than are not very consumer friendly in their business practices. Most home security companies charge a large amount of money for initial installation and some other companies require a subscription based payment method to have your home secure. With the use of a couple inexpensive electronic components one can implement a stand alone, completely wireless security system.

Introduction

Home security systems are becoming a staple for home owners in America. The expansion of the embedded systems market over the past few decades has made microprocessors and embedded RF communication commonplace in today's society. Most cell phones today contain several of these devices with WiFi, Bluetooth, GSM, and even satellite navigation right in your pocket. In the home, many people are using "Smart Home" devices such as voice assistants, automated thermostats and light switches. As these devices have gained a significant amount of popularity in recent years, "Smart" home security is also a rapidly growing trend.

There are a plethora of viral videos out there showing footage of criminal activity occurring right outside someone's home, without their home security system they may have never been alerted to any potential danger.

In a study released in 2010[1] by the U.S. Department of Justice, Bureau of Justice Statistics, an estimated 3.7 million burglaries occurred annually over preceding years. About 12% of homes burglarized while the occupant was present resulted in the homeowner facing an offender armed with a firearm or other deadly weapon.

A security system of any complexity is a good step in the pursuit of ensuring that you and your loved ones are protected home. The autonomous nature of our system has the added advantage of operating in a stand-alone fashion, sparing the homeowner the trouble of setting an alarm or monitoring the sensors them self.

Most modern security systems have sensors that can detect carbon monoxide and motion or glass-break, and record and send live video for the user to view remotely. This provides a safer living environment for the occupants when they're home or away, as well as peace of mind. Major security companies often require a service or subscription that a user pays periodically, as well as relatively complicated installation.

An inexpensive and easy to use home security system would let the user take control of their own home security. Making erstwhile wired sensors operate over a wireless channel would allow for the greatest ease of use for a typical consumer.

Using XBee RF modules and a Raspberry Pi, one can create their own automated home security system with the aforementioned functionalities. The XBee modules are used as communication devices that report sensor data back to a main unit, housing a system that processes the data and responds accordingly.The Raspberry Pi will serve as the main control unit to coordinate the signals coming from each sensor module, as well as transmit or store video depending on user configuration.. The goal of this project is to create an automated home security system that is inexpensive yet delivers the user a range of key capabilities to help protect their home and loved ones from preventable harm.

Student Statement

As one of the final steps in the pursuit of our degrees, this project has been a long but rewarding process. The Spring 2020 semester was significantly impacted by the COVID-19 outbreak.

All of us, students and staff, were required to come up contingencies on short notice to facilitate the completion of our projects. As a result we needed to find solutions not only to the problems initially proposed, but also to the new problems created by the aberrant situation.

Our erstwhile well planned timeline was thrown out the window. Luckily, we had been ahead of schedule prior to the transition to remote collaboration. Within a short time we had a tentative plan to bring our system to fruition. Ultimately our end product lacks some functionality that was initially proposed due to a variety of extenuating factors, beyond our control.

Contents

List of Figures

II Research and Design

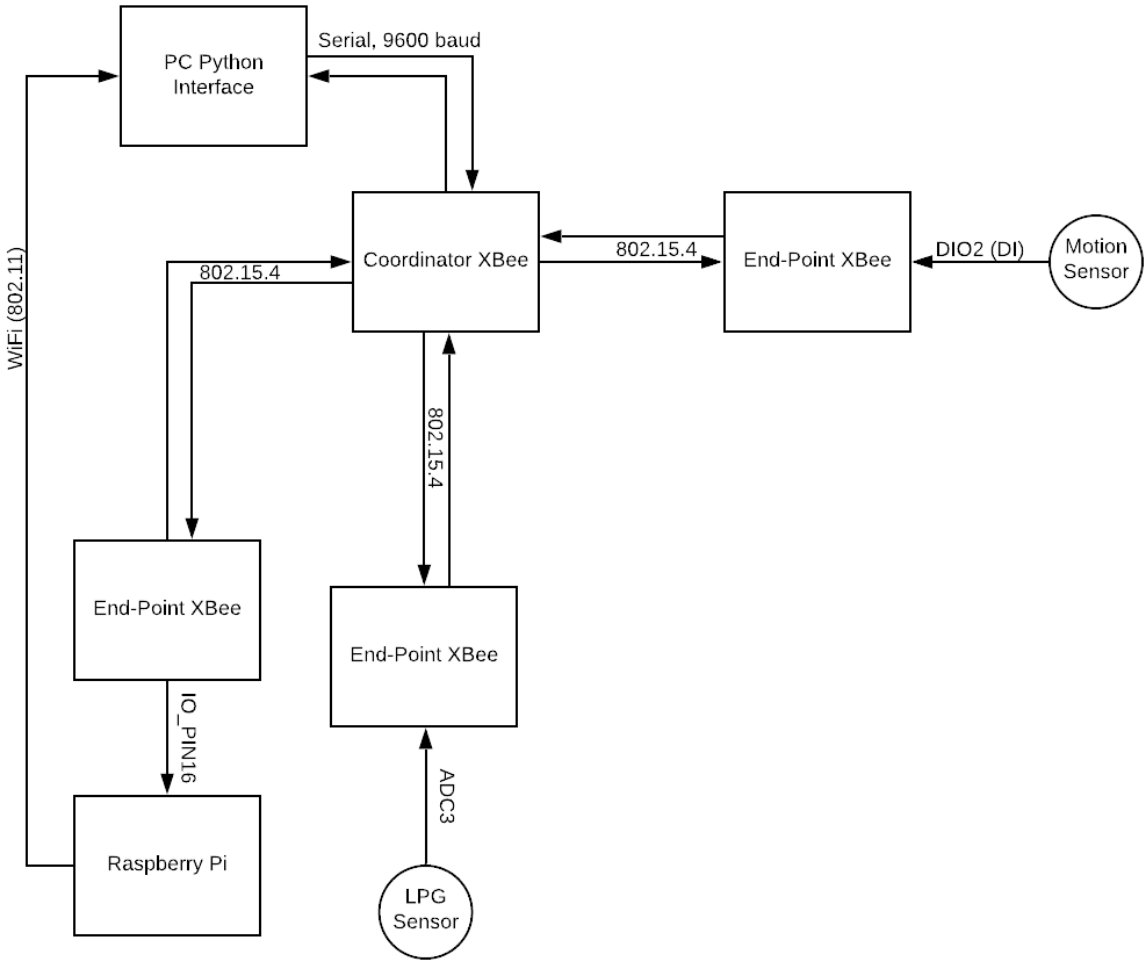


Figure 1: Basic Functional Diagram of System

The XBee S2C Module

For the heart of this wireless system we decided to use the XBee IEEE 802.15.4 RF modules manufactured by Digi International. These light weight, low-power modules are idea for this type of system given the desired ranges and performance characteristics we were looking for. To make a suitable home security system, the transfer of information from point A to B must be fast but reliable, and without errors or interference.

IEEE 802.15.4 standard is a member of the 802.15 group of standards for wireless personal area networks (WPAN) and is used for a wide variety of applications. In our case, the implementation of this standard in the XBee modules provides an adequate range, about 200 ft (60 m) in an indoor environment, as well as an adequate data transfer speed of 250 Kbps over the 2.4 GHz RF channel.

The nodes we designed around the XBee modules are battery operated and thus must have relatively low power consumption. The XBee modules themselves draw a maximum of 45 mA when transmitting, 31 mA when receiving, and less than 1 μA when in the powered down state.

The XBee modules come complete with 15 I/O pins and multiple analog to digital converters. This makes them the perfect choice for data acquisition of remote sensors.

XBee modules are available in a variety of form factors. We chose the XB24CAPIT form factor for our designs. Its antenna is contained within the PCB, while this may have less range than a wire antenna or U.fl add-on type of antenna, it provides excellent range for our application with minimal complexity. This model XBee is through-hole mounted, making initial designs and testing much easier that with a surface mount variety, though these occupy a smaller rectangular footprint.

All of these considerations taken into account we decided that the XBee-XB24CAPIT would be the ideal communication module for the system we have designed.

API vs Transparent Mode

	API	Transparent
	Sends wireless data to multiple destinations.	Provides a simple interface that makes it easy to get started with XBee devices.
[2]	Can set or read the configuration of remote XBee devices in the network.	what you send is exactly what other modules get, and vice versa.
	Can transmit data to one or multiple destinations	Works very well for two-way communication between XBee devices.

Raspberry Pi

The Raspberry Pi 3 Model A+ is the latest product in the Raspberry Pi 3 range, weighing in at just 29 g. Like the Raspberry Pi 3 Model B+, it boasts a 64-bit quad core processor running at 1.4 GHz, dual-band 2.4 GHz and 5 GHz wireless LAN, and Bluetooth 4.2/BLE. The dual-band wireless LAN comes with modular compliance certification, allowing the board to be designed into end products with significantly reduced wireless LAN compliance testing, improving both cost and time to market. The Raspberry Pi 3 Model A+ has the same mechanical footprint as the older Raspberry Pi 1 Model A+.

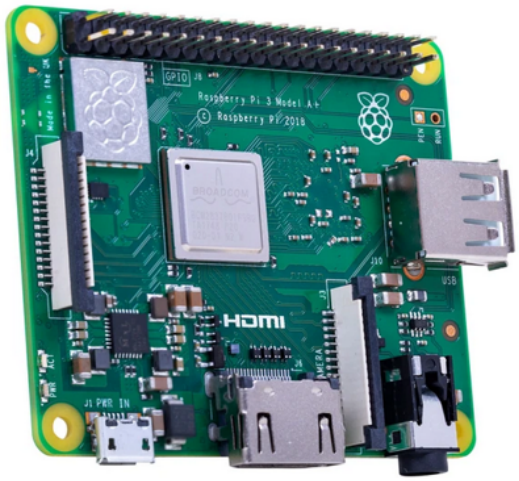


Figure 2: Raspberry Pi 3 Model A+

Raspberry Pi Camera (Hardware and Software)

The Raspberry Pi Camera v2 is the new official camera board released by the Raspberry Pi Foundation. The Raspberry Pi Camera Module v2 is a high quality 8 megapixel Sony IMX219 image-sensor custom

Figure 3: Pi NOIR Camera unit

designed add-on board for Raspberry Pi, featuring a fixed focus lens. It's capable of 3280 x 2464 pixel static images, and also supports 1080p30, 720p60 and 640x480p90 video. It attaches to Pi by way of one of the small sockets on the board upper surface and uses the dedicated CSi interface, designed especially for interfacing to cameras.

PyAudio

To begin the raspberry pi will receive a trigger from the Xbee to activate the camera. The camera would then take several pictures or a video file. The goal was to take a video of whoever approached the front door sensed by the motion sensor. The motion sensor would then communicate in tandem with the Xbee device and send a signal to the other Xbee device on the Raspberry Pi. The Xbee is connected to pin 16 on the Raspberry Pi and that will then start the camera to record the person at the door. Once the camera has captured enough video or pictures, the raspberry pi will then send the files to the main computer for viewing through the GUI.

Since the camera component does not work, we are replacing with recording audio whenever pin 16 is activated the code to record the audio via the microphone for whenever a person is at the door will be the main process instead. Through the implementation of python code and the triggers from the Xbee the raspberry pi will record audio of whenever motion is detected at the door and then save an audio file of the encounter which will then be automatically transferred over to the main computer over the network via WinSCP and the use of the planned GUI.

Supplying Power

The next consideration to be made was supplying power. In the pursuit of making out devices truly wireless, we were determined to have all the end-point devices be completely battery operated. Though the XBee RF modules themselves use minimal power even while transmitting, an efficient power supply would be necessary for prolonged operation.

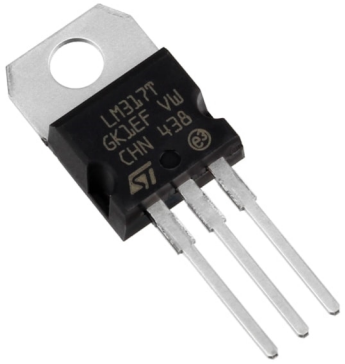


Figure 4: LM317 Linear Regulator (TH)

LM317 Linear Voltage Regulator The LM317 is an IC designed to maintain a constant voltage level. Voltage regulators usually have an input voltage range, but for the LM317 voltage regulator, the range is not specified because it has no ground connection. It has an output voltage range from 1.2 V to 37 V at 1.5 A. Since it is a linear regulator, the output voltage is always lower than input. The LM317 is used in our project to power the XBee modules which require a supply voltage between 2.1 V and 3.6 V. We designed a circuit incorporating the LM317 to output 3.3 V, which is ideal to power our devices.

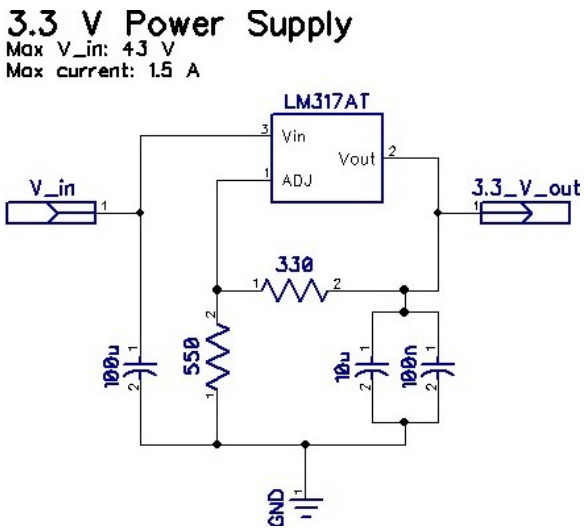


Figure 5: Complete power supply circuit.

As in figure 5, the values we know are $V_o = 1.2$ V, $V_{out} = 3.3$ V and setting R1 = 330 Ω. The value of Iadj is very small hence it can have considered negligible. The value of R2 came to 550 Ω. We then added the C1 = 100 μF, C2 = 0.1 μF and C3 = 10 μF for the improvement of transient response of a system to change from steady state.

LM7805 5V Voltage Regulator To power our devices that require a 5 volt source such as the MQ-6 LP gas sensor, we chose to design a circuit around the LM7805 voltage regulator. This regulator has an input range of 5 to 18 volts, ideal for out 6 volt battery packs, and produces a constant 5 volt output.

Environment Monitoring Hardware

MQ6 Liquid Petroleum Gas Sensor

The MQ-6 LPG Sensor is a highly sensitive, metal oxide based detector[3], designed to detect different petroleum derived gases, such as propane, butane and other flammable gasses including natural gas and methane. The sensor's conductivity changes proportional to the gas concentration in contact with the Tin Dioxide (SnO_2) sensing material within the shroud of the unit. The relatively low cost and wide range of detection make it an excellent choice for our system. The MQ-6 sensor operates at 5-volts.

Parallax Passive Infra-red Motion Sensor

To trigger subsequent elements of the proposed system we needed a motion detecting sensor. An infra red motion sensor works through applying the concept of pyroelectricity, a characteristic of materials that will generate a potential difference when heated. In this case the heat source is reflected infra red light. When there is no movement, current through a resistor in series with the pyroelectric material's electrodes will be constant. When movement occurs in the field of view of the infra red source, the amount of reflected light will change causing a change in the pyroelectric voltage. This change in voltage causes the current through the circuit to change, this is measured triggers the output accordingly.[4]

(a) MQ6 Liquid Petroleum Gas Sensor Package (b) MQ6 Liquid Petroleum Gas Sensor Diagram

Figure 6: MQ6 Gas Sensor

Figure 7: PIR Motion Sensing Module

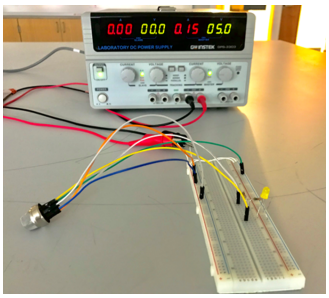
In our case, we desire a sensor such that negligible movement will be ignored. If this input was to trigger a camera, picking up a very small object's movement or similar "false alarms" would result in an over abundance of images taking up memory on the users pc or the system itself. Moreover if this device triggered an alarm, false alarms would be unacceptable in most cases. Most infra-red motion sensors on the market today are all-in-one units that take this into consideration by only raising a digital output high when a sufficiently large current change in the pyroelectric circuit is measured by the sensor module.

For our system we chose to use a Parallax passive infra-red *(PIR)* motion sensor. This PIR motion sensor operates from 3 volts to 6 volts meaning it can utilize the same power source as the XBee modules. The XBee will read the digital signal generated by the PIR sensor to trigger the rest of the system remotely.

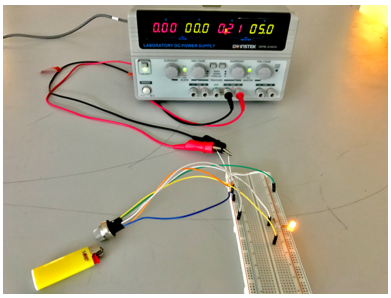
III Hardware Testing

Testing the MQ6 gas sensor

Next, we wanted to test this sensor independently. To do this we first started off with reading the data-sheet to find out the appropriate voltage input and pin layout. We found out that it operates at 5 volts with a minimum of 0 volts. We supplied 5 volts from the power supply to our breadboard and we used an LED to our output pin to see if it detects any gas.



(a) LED on via pin 18



(b) LED off via pin 18

Figure 8: Simple LED activation over 802.15.4 (2)

Properly connected, the gas sensor produced quantifiable results. Using a common butane lighter, we tested the sensors ability to detect ambient gases. The LED was able to indicate the presence of LPG, thus we now know our gas sensing components are functional.

Since the XBee modules analog to digital converter requires an input in the range of 1.2 volts, we must reduce the 5 volt output of the gas sensor down to this level to be quantified and processed. To do this a simple voltage divider was designed (1). With voltages constant and one resistor value constant we only need to solve for one unknown (2).

$$V_{\text{out}} = \frac{R1}{R1 + R2} * V_{\text{in}} \tag{1}$$

Solving for R1:

$$1.2 = \frac{5100}{R1 + 5100} * 5 \tag{2}$$
$$R1 = 16150\Omega$$

Testing the PIR Motion Sensor

The Parallax sensor allows us to choose between two approximate effective ranges, 15 feet or 30 feet. In testing the average rage, the results were about 8 ft and 15 feet respectively in a dense indoor environment. In tests conducted without the protective plastic cover over the sensor, range improved by about 2 to 4 feet in either case.

The range setting is changed with the manual moving of a jumper that connects a center pin with a second pin either on its left or right. In essence this is a crude switch. To yield as flexible a system as possible, a small DIP switch has been designed to toggle this setting. This allows the user to decide between a "Closer" or "Farther" setting depending on which is best suited to their operating environment. This would be set to the "Closer" option by default.

IV Software Testing

Note: full source code is at the end of this document.

Python, Primary Programming Language

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high-level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on Windows 2000 and later.

We will use Python to control each module via the coordinator XBee module that will be attached to a PC. Its versatility provides the ability to interface lower level controls on the XBee with a high level graphical interface all in one language.

Setting up Python to Interface with XBee over Serial Port

There are a handful of python libraries that are required to interface with the XBee device over USB serial. Most importantly, the Serial package allows configurable two way data-flow to serial devices from a python program.
The following packages were used in our systems python interface and data processing:

```
import serial           serial interface lib
import binascii           for converting between number systems
import time             time lib for cuttent times/performing operatons on times
import numpy as np        gui feature creation lib


"""The following lines are used for processing live data and creating plots"""
port matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib import style
mport math
import csv
```

A readily available python package allows for easy interfacing with the camera unit. This package provides a pure Python interface to the Raspberry Pi camera module for Python 2.7 (or above) or Python 3.2 (or above). It can configure resolution and give different dimensions to the video or picture files. It can also take photos under varying conditions if given the right parameters for a camera to take a good picture in a variety of different conditions including poor lighting. The code for the camera is as follows:

```
# A test of the cameras functions
from time import sleep
from picamera import PiCamera

camera = PiCamera()
camera.capture('/home/pi/Desktop/test.jpg') #saves camera picture to file location specified


# Camera trigger implementations
import RPi.GPIO as GPIO
from picamera import PiCamera
from time import sleep
GPIO.setmode(GPIO.BOARD)
GPIO.setup(16, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #Pull down to make pin read active high
GPIO.input(16) #pin 16 being read
if GPIO.input(16):
camera = PiCamera()
camera.capture('/home/pi/Desktop/test.jpg')
print('input high/camera on')
else:
```

```
camera = PiCamera()
camera.capture('/home/pi/Desktop/testfail.jpg')
print('input low/camera off')
GPIO.cleanup()
```

Explanation of Raspberry Pi Code Functions

The audiotest.py decsription:

This code establishes the audio streaming parameters to properly record and retain the data one has to set various parameters such as the resolution and sampling rate of the audio stream. This code records audio via a USB audio recording device and is set to record for 30 seconds. This then saves the file under the name test1.wav and will be placed in the main directory of the raspberry pi.

The devicecheck.py description:

This code just checks to see what USB audio devices are connected and recognized by the python pyaudio module. It should return the device name and what channel it is occupying. This information is needed to help set up the rest of the parameters in the audiotest.py code.

The cameratrigger.py description:

This code demonstrates how the general purpose input and output pins on the raspberry pi would have been implemented to read active high on pin 16 to be able to receive a signal from the Xbee device. When the pin receives 3.3V from the XBee device then the camera will trigger and take a picture and save it in the file directory specified.

The cameratest.py description:

Simple program to run the camera and save it to a specified file location.

Audiotriggercode.py description:

This code implements the GPIO from the Raspberry pi in preparation to interact with the Xbee device when it is being sent the trigger.

Syncscript.txt description:

This establishes an SFTP protocol relationship with the two file locations specified and synchronizes the folders via the synchronize command.

Syncthescript.bat description:

This helps execute the script that is programmed in the notepad.txt "syncscript."

ZigBee Mesh Protocol

There are three types of node in a mesh network. The coordinator who creates a network. Once the network is created other nodes can join in. Then there is a router node which routes the data sent by the end device. Then the information is sent over to the destination which is the coordinator. This coordinator node receives all the data send by the router node which is connected to the end device which are mainly sensor. Advantage of using a mesh network is to messages can be received more quickly if the route is short. It can also manage high traffic, as multiple nodes can transmit data between one another.

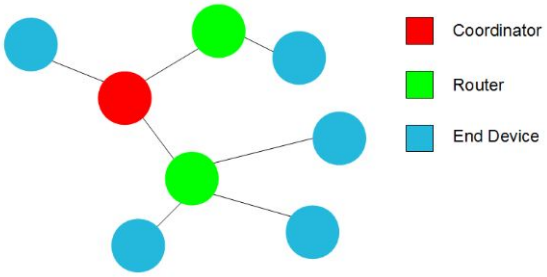


Figure 9: Mesh Network Visualized

Creating the Wireless Network between the Coordinator and Router modules

We wanted to create a wireless network between the coordinator and router modules this time. Firstly, we started off with soldering our XBee Explorer to connect our router modules on top of them. The reason we chose Xbee Explorer was because it will allow us to solder the input power voltage and the output pins that will be connected to the sensors. We used the application XCTU to help us set up each Xbee as a coordinator or router modules. We first set our coordinator in API mode that will allow us to communicate directly with the sensors via router modules. At the same time, we set the other three router modules to be in a AT or Transparent mode. This will allow sensors to send data directly to the coordinator. We also made sure all the XBee modules used the same Channel and Personal area network (PAN) ID.

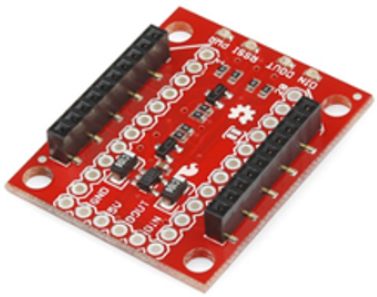
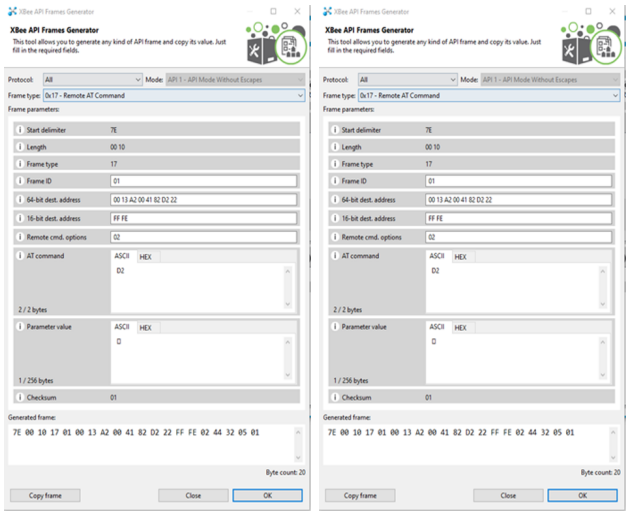


Figure 10: Small XBee interface to be used in each module

Controlling the Router XBee with XCTU Remote AT Command

We decided to test our router XBee remotely by sending some AT Commands. We used the XCTU application to first create the Remote AT Command to turn on and off pin 18 of the router XBee. To see if we receive any output, we put an LED to see if the pin was turned on or not. In the figure below you can see the LED turning on and off via Pin 18 of the XBee device. The pictures help visualize what the pin is doing corresponding to the LED.



(a) Turn On Pin 18 (b) Turn Off Pin 18

Figure 11: Simple LED activation over 802.15.4

(a) LED on via pin 18 (b) LED off via pin 18

Figure 12: Simple LED activation over 802.15.4 (2)

XBee Digital In/Out setup *(DIO)*

The XBee devices have the capability to detect a change in a digital input signal. This edge detection allows the remote motion sensing module to transmit its data only if the signal changes. The coordinator will receive two packets, the first being a reading of logic high, the second being a reading of logic low once the sensor has stopped detecting any movement. In this way the unit does not need to send samples at a continuous rate, which saves power.

To enable the change detection feature on DIO3, we need to set the hexadecimal value in the setting DIO change detect to 0x4. As seen on the right side of Figure 13 this will enable the change detection on bit 2 only, which corresponds to DIO3.

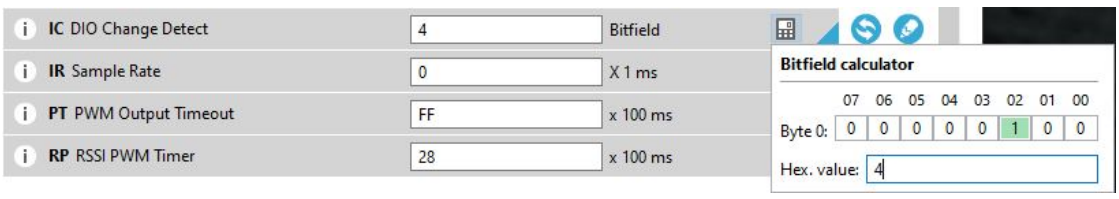


Figure 13: Change detection parameters in XCTU

When initially attempting to read data from the XBee's DIO pin, the value read would always read high. This reading was verified with a multimeter as well as on the scope. After some trouble shooting it was discovered that the issue lie in the I/O settings of the XBee, which has the ability to pull its

digital inputs or outputs high or low based on a bit mask.

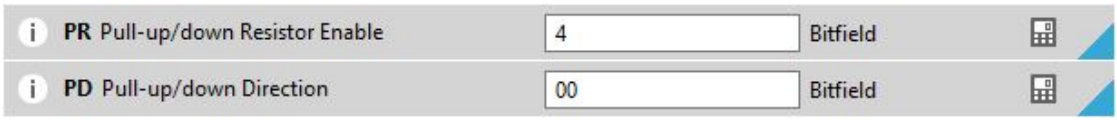


Figure 14: Bit masking settings for pull-up/pull-down resistors.

In XCTU the mask had been set to pull all DIO pins high, which was not allowing a low input to be seen. Once re-masked, pulling the DIO pin used for the sensor reading down by default, the values were read correctly.

Digital I/O Interfacing with Python

The next step was to interact with the system through via python. After some initial testing of functions and learning the syntax a small GUI (Figure 15) was created to display red when tripped and green when no movement was present.

Figure 15: Motion Sensor Interface

```
# setup gui elements ---------

window = tk.Tk(screenName="Test Name", baseName=None, className='Tk', useTk=1)
window.title("Motion Sensor Console")
motion_canvas = tk.Canvas(window, bg="green", width=50, height=50)
btn_ON_OFF = tk.Button(window,
                activebackground="grey",
                text="Off",
                fg='white',
                bg='red'
               )
# window title banner
window_banner = tk.Label(window,
                 text="Motion Sensor Console",
                 fg='grey',
                 bg='black',
                 relief="solid",
                 font=("Arial", 30, "normal")
                )
```

Then, by establishing a serial connection with the coordinator, the whole network can be controlled and modified from a python program. Reading the digital input from the remote sensor, its is converted into a usable format, and if a rising edge was detected the program enter the triggered state and display a a red block. When a falling edge is received, the program will reset to a normal state.

Primary GUI

This graphical user interface (GUI) was simply developed as a user friendly software. This allow users to operate and control electronic device through manipulation of buttons, gestures, and mouse. In our project, the users would have the capability of turning on/off the system, control the cameras, and turn on/off the alarm. The users have this feature for their ease and comfort. This can act accordingly to any difficult situations. The code below is a prototype of the GUI we were thinking of.

```
    labell=Label(window,text="HOME AUTOMATION", fg='blue', bg='yellow',
    relief="solid", font=("stencil",30,"bold")).place(x=139,y=22)

    #def main():
    #print('OLA')

    button1: Button=Button(window,text="TURN ON SYSTEM",fg='red', bg='purple',
    relief=RAISED,font=("arial",14,"bold"))#,command=main())
    button1.place(x=100,y=90)
    button1.config(height=3, width=22)

    button2=Button(window,text="TURN OFF SYSTEM",fg='red',bg='brown',
    relief=RAISED,font=("arial",14,"bold"))
    button2.place(x=300,y=90)
    button2.config(height=3, width=22)

    button3=Button(window,text="TURN ON ALARM",fg='red',bg='brown',
    relief=RAISED,font=("arial",14,"bold"))
    button3.place(x=100,y=165)
    button3.config(height=3, width=22)

    button4=Button(window,text="TURN OFF ALARM",fg='red',bg='brown',
    relief=RAISED,font=("arial",14,"bold"))
    button4.place(x=300,y=165)
    button4.config(height=3, width=22)

    button5=Button(window,text="TURN ON CAMERA",fg='red',bg='brown',
    relief=RAISED,font=("arial",14,"bold"))
    button5.place(x=100,y=238)
    button5.config(height=3, width=22)

    button6=Button(window,text="TURN OFF CAMERA",fg='red',bg='brown',
    relief=RAISED,font=("arial",14,"bold"))
    button6.place(x=300,y=238)
    button6.config(height=3, width=22)

    window.mainloop()
```

When using a GUI in python the window runs in an infinite loop, constantly updating. This prevents the program from executing anything while the *"mainloop()"* is running. To avoid this issue. The gui will setup the window environment and then enter a recursive function that reads and interpenetrates

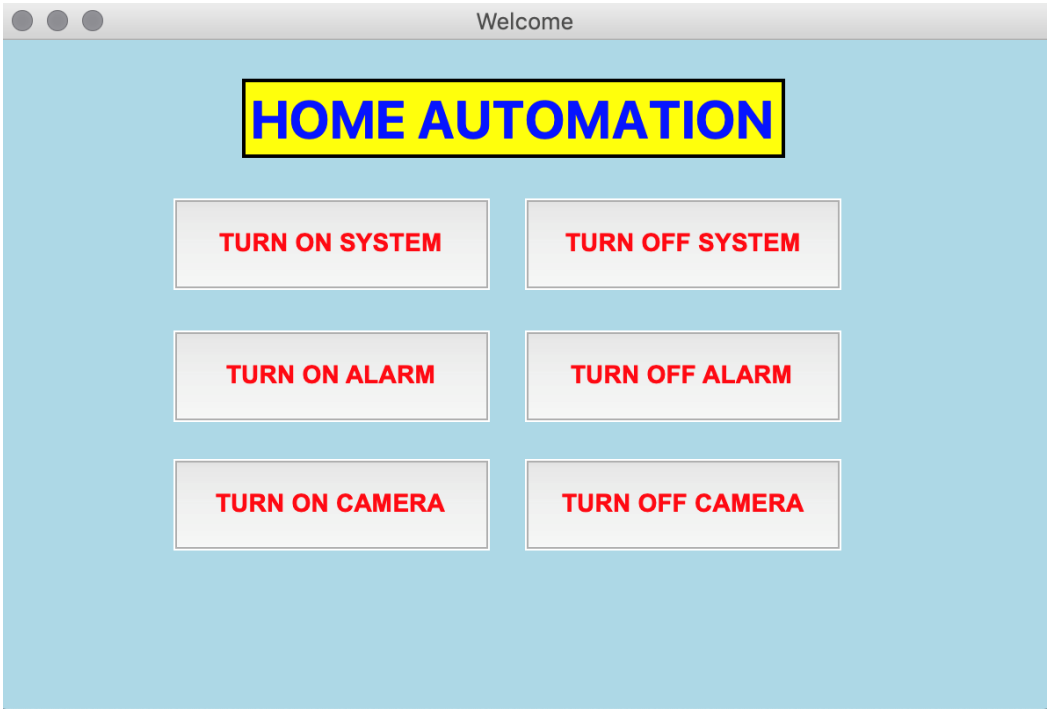


Figure 16: Main System GUI

the digital data being received and updates the window elements accordingly.

```
        window_banner.pack()
        motion_canvas.pack()
        print("re-paint")
        window.update()
        print("set window interrupt")
        window.after(0, get_motion_status()) # recursive function
```

If the remote module was sending samples at a constant rate this is much simpler, missed packets are less of a concern. The change detect digital IO configuration will only send a data packet when an event occurs. This means that the python program must spend most of its time waiting for data to arrive from the serial port. By setting a timeout in the serial setup, the program will wait for data for a set time before executing the rest of the loop. If the timeout was indefinite the window would not update properly. If the timeout is too short, the program will miss packets that arrive while the rest of the function is executing. With a timeout slightly less that the 2 second high-time of the sensor itself, the program spend the majority of its time listening for data and will miss very few data transmissions, while still updating the window elements at a reasonable rate.
The below code block shows the recursive function written to check for motion sensor data and update the display:

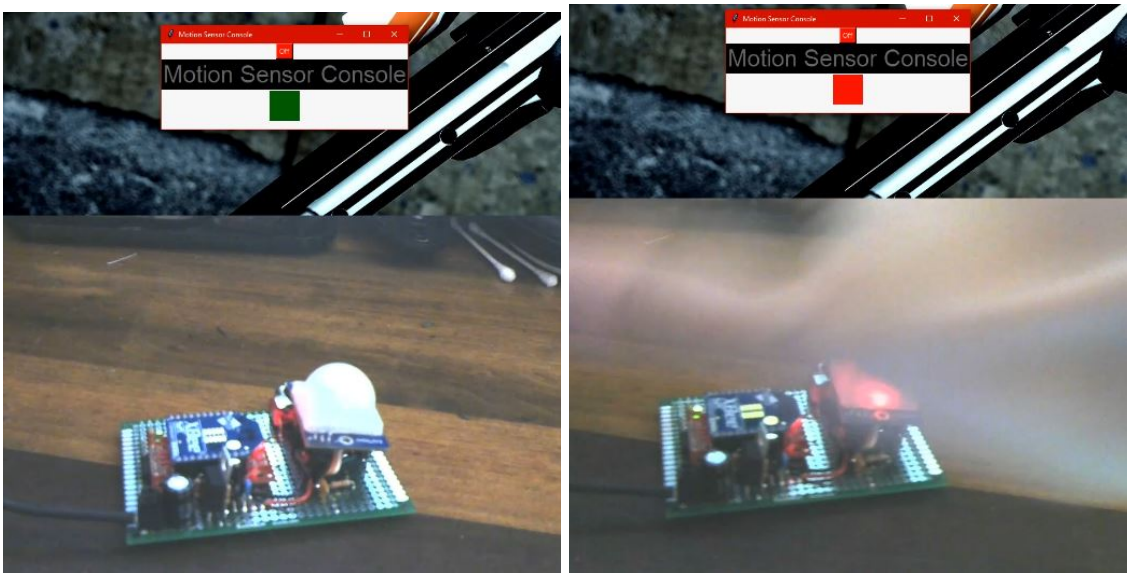
```
        def get_motion_status():
           data_raw = ''
           initial_red_time = 0
           ser1 = serial.Serial('COM3', 9600, timeout=1.75)
           data_raw = ser1.read(14)

           if data_raw == '':
              ser1.close()
              window.after(0, get_motion_status())
              window.update()
           else:
              try:
                 data_hex = binascii.hexlify(data_raw).decode('utf-8')
                 D2 = data_hex[22:26] # extract desired bits
                 base_ten_val =int(D2, 16)
                 # print("this sample:")
                 print(base_ten_val) # view data

              except ValueError:
```

```
            # print("caught valueError")
            ser1.close()
            # print("re-paint")
            window.update()

        else:
            print("Data received")
            if base_ten_val == 4:
            print("set indicator: red")
            motion_canvas.config(bg="red")
            print("re-paint")
            initial_red_time = time.now()
            window.update()
            elif base_ten_val == 0 or '':
            print("set indicator: green")
            motion_canvas.config(bg="green")
            print("re-paint")
            initial_red_time = 0
            window.update()
        finally:
            ser1.close()
            # print("re-paint")
            window.update()
            # print("set window interrupt")
            window.after(0, get_motion_status())
```



(a) Safe (b) Triggered

Figure 17: Motion Sensor GUI functionality

To account for any data that is missed a contingency was added to the program such that if the program is in the "tripped" state for too long it will automatically reset to "safe." If the sensor is being tripped it will be set to the "tripped" state again almost immediately. Remaining tripped for too long indicate that a rising edge was received but a falling edge was not. If no movement occurs after a missed falling edge the program would remain in the tripped state indefinitely.

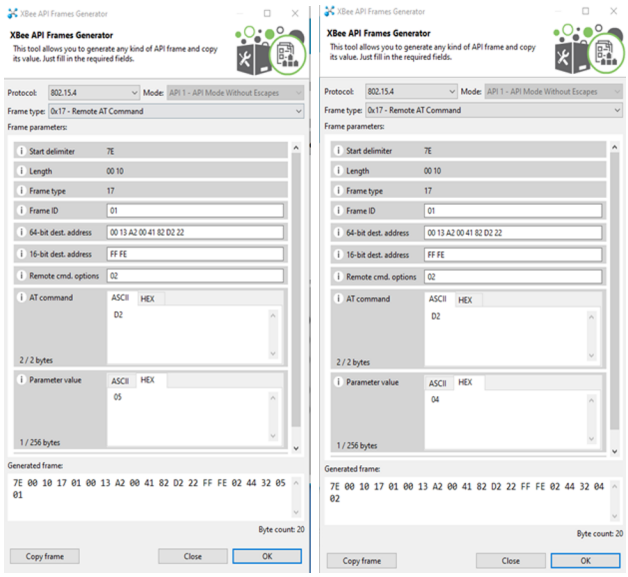
```
    delta_t = time.now() - initial_red_time
    threshold_time = time.time(0, 0, 10) # threshold for missed low == 10 sec

    if delta_t - threshold_time < 0:
        """IF time after a high signal received exceeds 10 seconds, reset. If a falling edge missed program will remain high until the
            falling edge of subsequent trigger. If tripped value will return to tripped state on the next loop until 10 seconds have
            passed again. """

        print("set indicator: green")
        motion_canvas.config(bg="green")
        print("re-paint")
        initial_red_time = 0
        window.update()
```

Controlling XBee module using Python

Since we now know that remote router XBee can be controlled by the coordinator XBee wireless, we decided to use python program for the same task. First, we connected our coordinator XBee module to the computer and set it to API mode. Then we powered up the router module and now we know that it has been connected in a wireless mesh network. We used the XCTU application to generate the same Remote AT Command frame to control the pin 18 of the router module. Then opened the python program to start writing our program in a new script file. API frame generated to Turn on & Off pin 18:



(a) Turn On Pin 18 (b) Turn Off Pin 18

Figure 18: API Frames created in XCTU

```
#Configure AD2 and DIO2 to control pin-18

import serial
import binascii
import time

ser = serial.Serial(COM3)
ser.baudrate = 9600

#XCTU API Frame remote AT command used to control pin-18
#Power ON
turnOn = "7E 00 01 17 01 00 13 A2 00 41 82 D2 22 FF FE 02 44 32 05 01"
#Power OFF
turnOff = "7E 00 01 17 01 00 13 A2 00 41 82 D2 22 FF FE 02 44 32 04 02"

#Use binascii to convert hex instruction to binary
messageOn = "".join(turnOn.split())
on = binascii.unhexlify(messageOn)

messageOff = "".join(turnOff.split())
off = binascii.unhexlify(messageOff)

#Test write commands, flash led at 5 sec intervals
count = 0
while (count < 10)
   ser.write(on)
   time.sleep(5)
   ser.write(off)
   count = count + 1
```

Gas Sensor Interfacing & Data Processing

Since the MQ-6 LPG sensor is an analog device, all sampled data must be converted to digital data by an analog to digital converter before transmission. As a result of this, all data received by python will be binary voltage values which must be converted into a usable format.

Threshold Detection

After the sensor started to give us the correct voltages, we decided to use python interface to plot us few graphs for us in real time. This way we have more visual knowledge and meaning about the data the sensor is sending to us. The code to plot graph is shown below.

```
#initializing the function to open the XBee Coordinator
ser1=serial.Serial()
ser1.baudrate=9600 #speed
ser1.port='COM3'  #port address

#figure for plotting the Time Vs Votlage
fig = plt.figure()
ax=fig.add_subplot(1,1,1)

time=[]
volt=[]

#open the serial port of the coordinator for incomming signal from the gas sensor
ser1.open()

#Periodically from funcAnimation
def animate(i,time,volt):
   data=ser1.read(14)
   v=binascii.hexlify(data).decode('utf-8') #decode the data to Hexadecimal
   D2=v[22:26]                              #select the Voltage range from the Hexadecimal
   Dec2=int(D2,16)                          #covert the Hexadecimal to Decimal value
   Voltage=1.2*Dec2/1023                    #formula to finally to convert decimal to voltage reading

   time.append(dt.datetime.now().strftime('%H:%M:%S')) #Display current time
   volt.append(Voltage)

   time=time[-20:] #Trim and updated the 20 cureent incoming data
   volt=volt[-20:] #Trim and updated the 20 cureent incoming data

   #Below is the style, axis label, and intervals for the graph
   ax.clear()
    ax.plot(time,volt)
   plt.plot()
   plt.xticks(rotation=45, ha='right')
   plt.subplots_adjust(bottom=0.30)
   plt.title('Voltage over Time')
   plt.ylabel('Voltage(V)')
   plt.xlabel('Time')
   plt.ylim(0,1.3)
   plt.grid(color='black',linestyle='dotted')
   plt.axhline(0.35)

   ani=animation.FuncAnimation(fig,animate,fargs=(time,volt), interval=1000)
   plt.show()
   ser1.close()
```

Finding the Average, Mean, and Variance of sampled voltages

The voltages we were receiving from the sensor were finally what we expected. Our next goal was to find the threshold of the sensor so that we could make a visual cut off of the concentration of gas in terms of PPM and voltages in real time. We started off my reading the voltages from the sensor and computing the mean, average, and variance from the sensor directly.
The following Program was written to attain these values:

```
#Importing the modules
import serial
import binascii
import time
import numpy as np

#initializing the function to open the XBee Coordinator
```

```
ser1=serial.Serial()
ser1.baudrate=9600 #speed
ser1.port='COM3'   #port address
ser1.open()        #open port

#For Loop to keep reading data
for i in range(15):
data=ser1.read(14)
v=binascii.hexlify(data).decode('utf-8') #decode the data to Hexadecimal
print(v)

D2=v[22:26] #Select the Voltage range from the Hexadecimal
Dec2=int(D2,16) #covert the Hexadecimal to Decimal value
print(D2)

Voltage=1.2*Dec2/1023 # formula to finally to convert decimal to voltage reading
print(volt)

average=np.average(volt) #compute the average of the voltages
mean=np.mean(volt)      #compute the mean of the voltages
vari=np.var(volt)       #compute the variance of the voltages

print('The Average of the Sensor is:')
print(average)
print('The Mean of the Sensor is:')
print(mean)
print('The Variance of the Sensor is')
print(vari)

ser1.close()       #close port
```

```
Output:
   The Average of the Sensor is:
   0.255
   The Mean of the Sensor is:
   0.255
   The Variance of the Sensor is:
   0.0
```

Calculating True Parts-Per-Million *(PPM)*

Parts per million is very useful way of describing small amounts of concentration compared to the larger amount of concentration. In our project we will be using PPM to compare our concentration of any type of gas from our MQ-6 sensor. We are mainly targeting LPG.

Using documentation provided in the manufacturer data sheet (MDS) for the MQ-6 (Appendix VIII.3) the system can be calibrated to read a very accurate PPM of select ambient gasses.

The ordinate is resistance ratio of the sensor (Rs/Ro) the abscissa is concentration of gases. Rs means resistance in target gas with different concentration, R0 means resistance of sensor in clean air. All tests are finished under standard test conditions. The x axis is in ppm while y axis is $\frac{R_s}{R_o}$. This ratio can be calculated using Equation 3, where m is the slope and b is the y- intercept which can be solved using any two points from the graphs in Figure 20, see Table 1.

$$Log_10(\frac{R_s}{R_o}) = m \times Log_10(PPM) + b \tag{3}$$

$\frac{R_s}{R_o}$	PPM
1	1000
0.39	10,000
...	...

Table 1: $\frac{R_s}{R_o}$ vs. PPM

Using the Equation 3 with the values from Table 1 we can generate a system of two equations with which we can solve for m and b.

$$Log(1) = m \times Log(1000) + b$$

$$Log(0.39) = m \times Log(10,000) + b$$
$$Solve:$$
$$m = -0.41, b = 1.23$$

Now we can solve for the PPM:

$$Log(\frac{R_s}{R_o}) = m \times Log(PPM) + b$$

$$\frac{R_s}{R_o} = 10^{-}0.41 \times log(10) \times PPM + 1.23$$
$$\frac{R_s}{R_o} = 10^{1}.23 \times 10^{-}0.41 \times log(10) \times PPM$$
$$\frac{R_s}{R_o} = 16.982 \times PPM^{-}0.41$$

Our main objective is to find a relation between the output sensor voltage V_{out} and $\frac{R_s}{R_o}$. We start by assuming that the circuit is made up of series resistance Rs and RL voltage divider of R1 and R2 as seen in Figure 19 giving us equation 4.

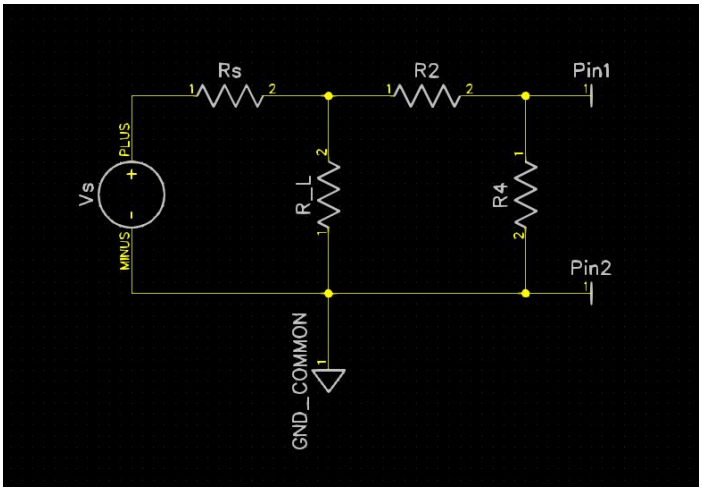


Figure 19: Restive Circuit

$$\frac{V_1 - V_s}{R_s} + \frac{V_1}{RL} + \frac{V_1 - V_o ut}{R_1} = 0 \tag{4}$$
$$Known: R_1 = 16150\Omega, R_2 = 5100\Omega, RL = 20000\Omega, V_s = 5v$$
$$\frac{V1 - Vs}{Rs} + \frac{V1}{RL} + \frac{V1 - Vout}{R1} = 0$$
$$\frac{V1}{RL} + \frac{V1 - Vout}{R1} = -\frac{V1 - Vs}{Rs}$$
$$\frac{V1R1 + V1RL - VoutRL}{R1 * RL} = -\frac{V1 - Vs}{Rs}$$
$$Rs = -\frac{(V1 - Vs)(R1 * RL)}{V1 * R1 + V1 * RL - Vout * RL}$$
$$Rs = -\frac{(V1 - Vs)(R1 * RL)}{V1 * R1 + V1 * RL - Vout * RL}$$
$$Rs = -\frac{(V1 * R1 * RL - Vs * R1 * RL)}{V1 * R1 + V1 * RL - Vout * RL}$$
$$Rs = \frac{Vs * R1 * RL - V1 * R1 * RL}{V1 * R1 + V1 * RL - Vout * RL} \tag{5}$$

From the graph, we can note that the resistance ratio in fresh air is a constant:

$$R_s = R_o$$

To find the Ro we will have to find the value of Rs in fresh air. So by taking the average of analog reading from the gas sensor and converting it into voltage reading, Average voltage reading from the sensor we found in clean air = 0.225 volts.
Therefore:

$$V1 = \frac{R1+R2}{R2} * \text{Vout}$$
$$V1 = \frac{16150 + 5100}{5100} * 0.225$$
$$V1 = 0.9375 volts$$

Solving for Rs from equation 5

$$Rs = \frac{5 * 16150 * 20000 - 0.9375 * 16150 * 20000}{0.9375 * 16150 + 0.9375 * 20000 - 0.225 * 20000}$$
$$Rs = 44645.9704\Omega = R_o \text{ (in clean air)}$$

The value of Ro can be obtained only numerically. It is equal to the value of Rs in clean air condition. The value of Ro will remain constant and no need to be recomputed every time. Finally using equation 3

$$Log_10(\frac{Rs}{Ro}) = m \times log_10(PPM) + b$$
$$m * Log(PPM) = Log_10(\frac{Rs}{Ro}) - b$$
$$\text{PPM} = 10^{\frac{\text{Log10}\left(\frac{Rs}{Ro}\right) - \text{b}}{m}}$$
$$\text{PPM} = 10^{\frac{\text{Log10}\left(\frac{Rs}{Ro}\right) - 1.23}{-0.41}}$$
$$PPM = 1000$$

Where

$$Ro = 44645.9704\Omega$$

and

$$Rs = \frac{Vs * R1 * RL - V1 * R1 * RL}{V1 * R1 + V1 * RL - Vout * RL}$$
$$\text{with changing } V_1$$

Therefore the PPM Equation will be,

$$PPM = 10^{\frac{Log_10(\frac{Rs}{Ro}) - b}{m}}$$

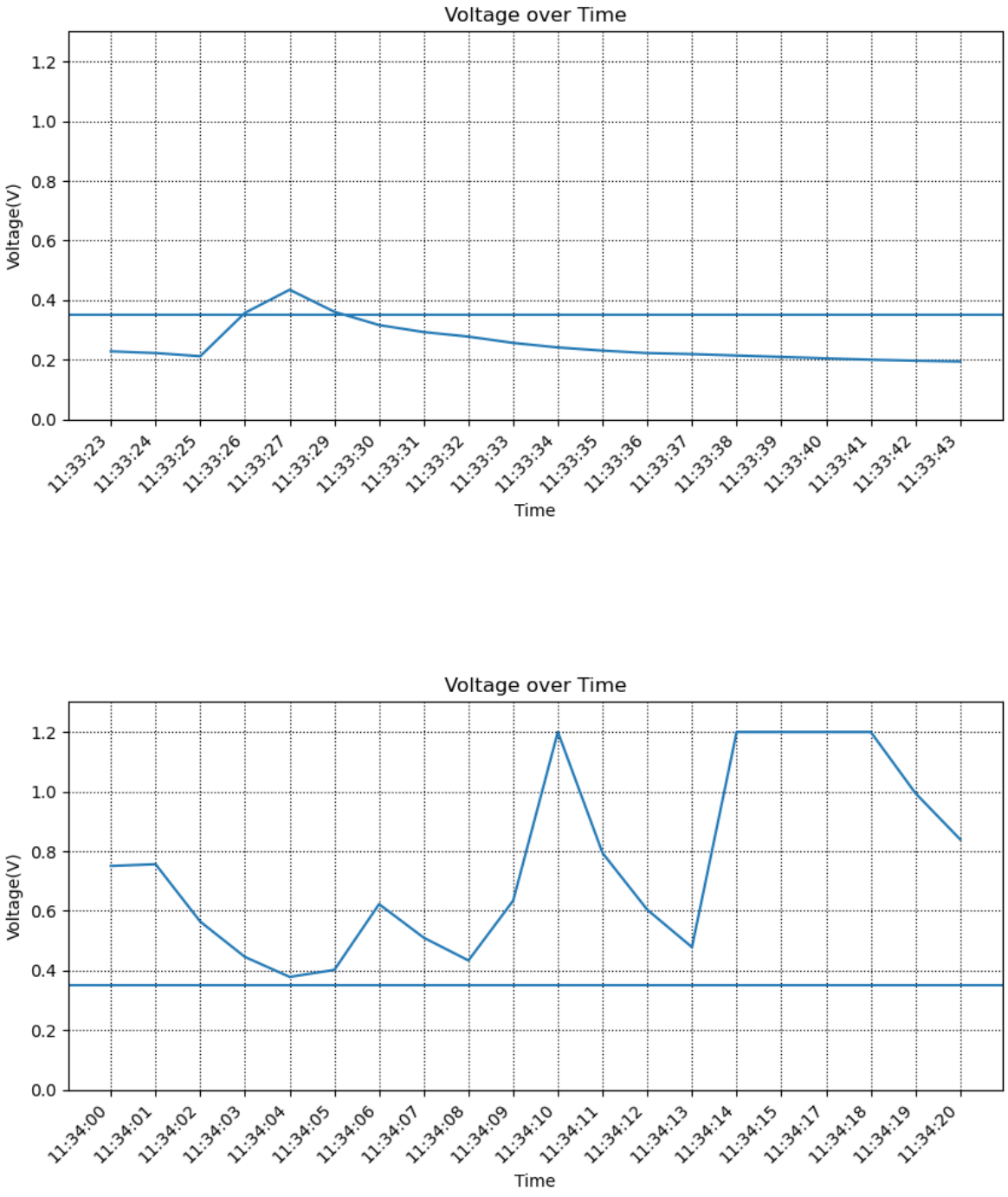
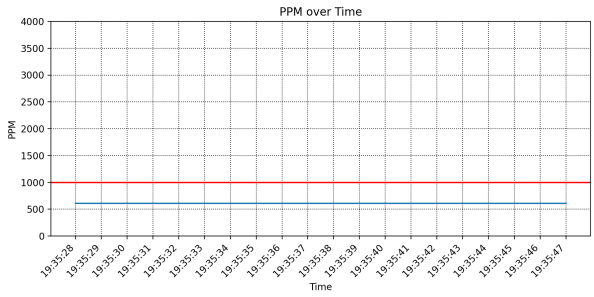
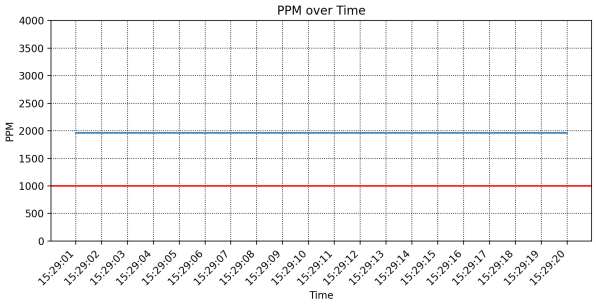


Figure 20: two of several sets of data, Live python plot



(a) PPM below critical threshold



(b) PPM above critical threshold

V Physical Construction of End-point Devices

The modules themselves will be constructed on 6x8 cm perf-board and will be contained in a housing. Each module will contain all circuitry necessary including power, data acquisition and transmission.

The gas sensor module requires two power supplies, one 3.3 volts to power the XBee itself and 5 volts to operate the Tin Dioxide (SnO2) gas sensor.

Our system consists of 4 separate modules:

- Coordinator Module - Connects directly to the PC, connects to user interface and controls messages between subsequent end point modules.
- LP Gas Module - A liquid petroleum gas sensor to detect gas leaks in the home. Can detect a wide range of flammable gasses commonly found a typical home. (Figure 22 & 23a)
- Motion Sensing Module - The motion sensing module will detect movement and trigger a system response to suit. The Motion module is stand alone to allow the as much freedom of placement as possible. (Figure 23b)
- Door Camera/Lock Module - This module, when directed to do so, can take an image of the area as well as lock the door automatically. This is the most complicated module in the system and includes a Raspberry Pi to transfer frames over the local area network.

The completed modules are fully independent and occupy a small footprint, despite being assembled on prototyping board.

A custom schematic diagram, PCB layout (Figure24) and 3D model (Figure 25) were created to demonstrate the small potential form factor of the units themselves. With surface mount components, including the XBee S2C surface mount model could be used to manufacture extremely effective sensing devices in an incredibly small package.

Note: The pattern (PCB footprint) of the XBee S2C through-hole model has been designed accurately up to 1 mil (0.001 inches). CAD libraries often do not have the component pattern or schematic of most specialized ICs and other components. If you wish yo use it in a PCB design the schematic, pattern, pin layout/numbering/types, and dimensions must all be measured or documented to a high precision. I used the dimensions found in the technical drawings in the XBee S2C manufacturer data sheet to design the schematic and pattern for the XBee device.

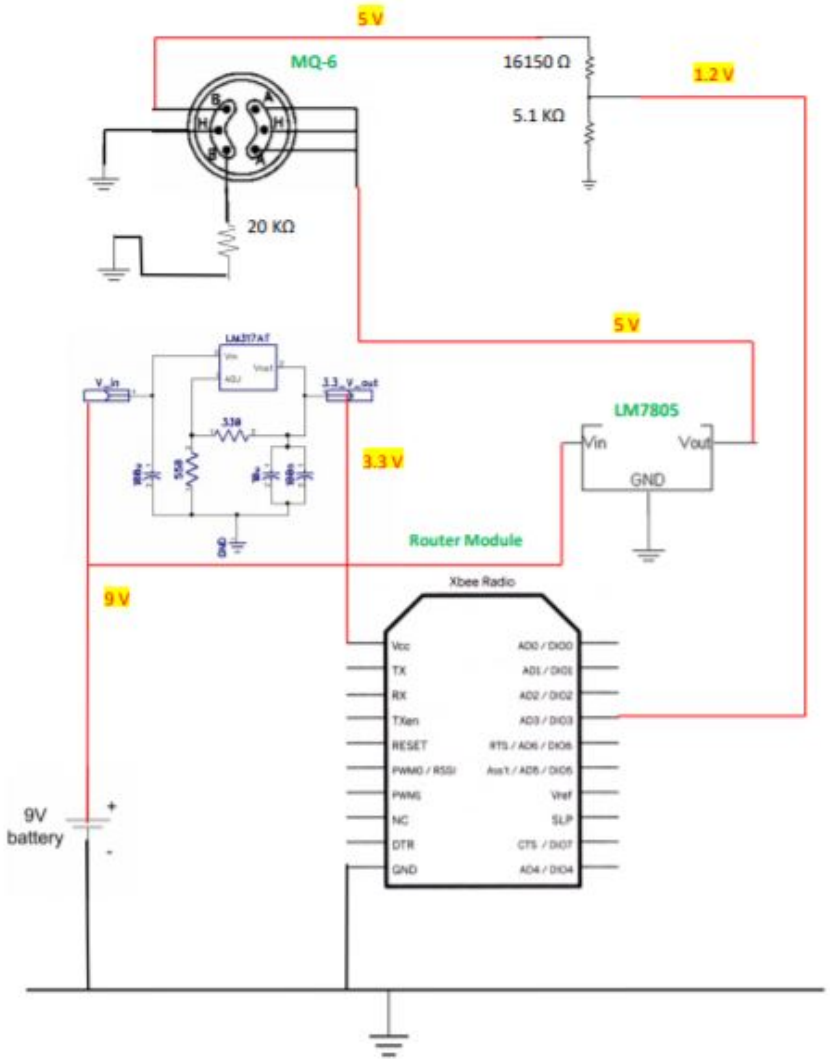
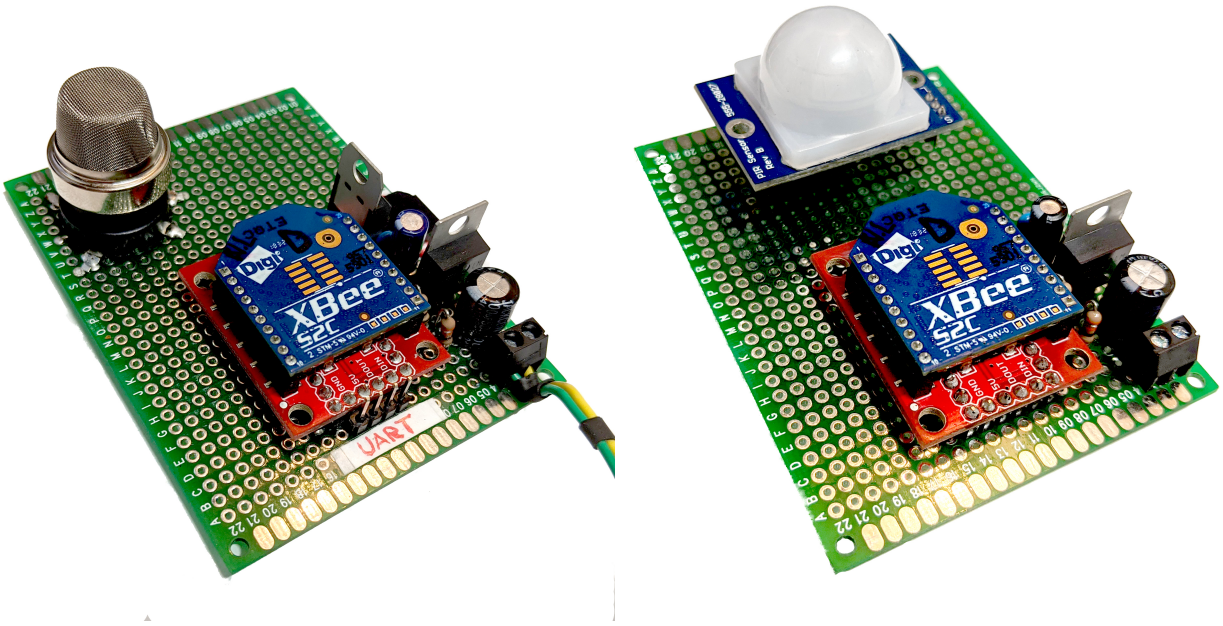


Figure 22: Complete gas Sensing unit circuitry.



(a) Gas Module (b) Motion Module

Figure 23: Internals of Gas (a), and Motion (b) sensing units

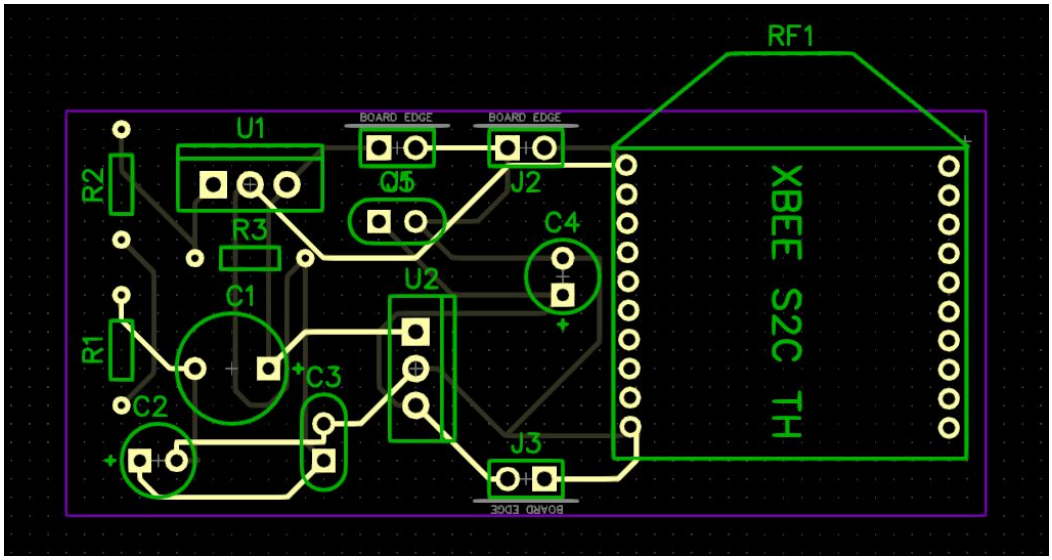


Figure 24: PCB Layout, 2 layer + top silk screen

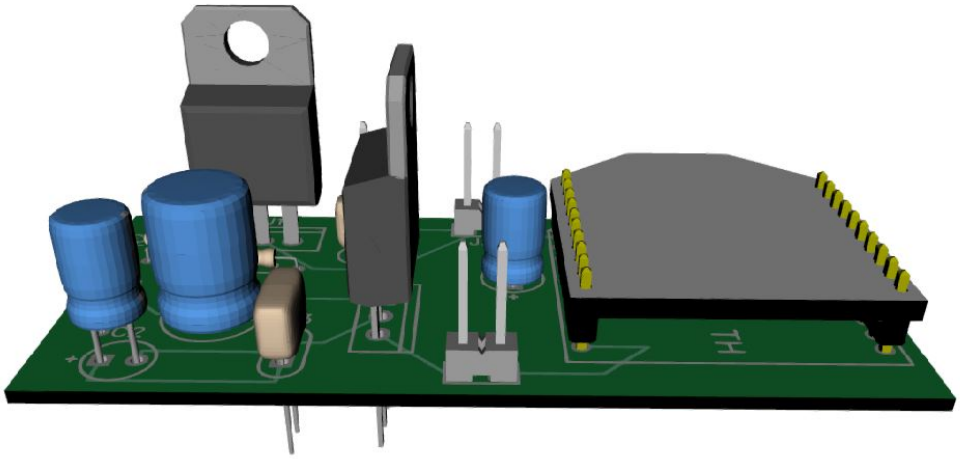


Figure 25: CAD Generated 3D Model of PCB including 3.3 & 5 V power supplies, test points, XBee Module

VI Design Difficulties

Testing Raspberry Pi & Camera

Camera difficulties Camera Component sends a camera control callback error and does not take and save pictures anymore. The OS of the raspberry pi has been reset and re-downloaded to the raspberry pi. There is a visible trigger to the camera activating when using the Pi NOIR camera board. There is a red light on the board of the camera that signifies when it is turned on or activated. The camera when activated through the command window will sometimes create the file but have no data inside of it. Upon receiving this error it seems to slow down the raspberry pi board significantly that it requires a restart. The raspberry pi has been test with three different camera modules and all of them give back the same callback control error. Raspberry Pi has configuration settings to activate and recognize when a camera is connected, however the issues may seem to arise with a faulty port on the raspberry pi itself.

```
mmal: Received unexpected camera control callback event, 0x4f525245
```

Figure 26: Error from camera module

When attempting to switch to audio recording instead of using the camera the only devices available were 3.5mm jack from a headset with built in microphone. The assumption was to connect to the audio jack output of the raspberry pi, and record and listen through the headset itself. However, this is impossible due to the specifications of the raspberry pi the audio jack is meant for output only. The only way to record audio was using a USB microphone connected to the Raspberry Pi device itself. The other problem was unfortunately due to limitations the GUI was not implemented due to not having components being able to come together, so manual transfer of file is required but is shown that it can be implemented to work automatically as a python script.

PyAudio

To begin the raspberry pi will receive a trigger from the XBee to activate the camera. The camera would then take several pictures or a video file. The goal was to take a video of whoever approached the front door sensed by the motion sensor. The motion sensor would then communicate in tandem with the XBee device and send a signal to the other XBee device on the Raspberry Pi. The XBee is connected to pin 16 on the Raspberry Pi and that will then start the camera to record the person at the door. Once the camera has captured enough video or pictures, the raspberry pi will then send the files to the main computer for viewing through the GUI.

Since the camera component does not work, we are replacing with recording audio whenever pin 16 is activated the code to record the audio via the microphone for whenever a person is at the door will be the main process instead. Through the implementation of python code and the triggers from the XBee the raspberry pi will record audio of whenever motion is detected at the door and then save an audio file of the encounter which will then be automatically transferred over to the main computer over the network via WinSCP and the use of the planned GUI.

PIR Motion Sensor

In initial testing of the sensor, the output received at the XBee itself was subject to some instability. This was solved by adding an RC filter between the digital input of the XBee and the output of the sensor module. This eliminated a DC component that would sometimes be present in the signal.

Remote Collaboration / COVID-19 Disruption

The sudden disruption of the project by the COVID-19 outbreak caused a great deal of additional work that, while necessary, consumed a lot of extra man hours. The switch to remote collaboration also meant we could not test our system as a whole. We divided all feasible tasks

Additionally the transition to completely remote collaboration demanded a high level of organization to keep various source code files, report documents, and all other files and supporting documents up to date.

To keep the LaTeX version of the report up to date, as well as to keep track of versioning, all included documents, source files, and images were stored in a GitHub repository. This allowed us to always download not only an always up to date pdf version of this report, but all other included files necessary to make changes. This ensures that a clone of the repository will always be open-able and compile-able.
The repository containing this report can be found and viewed at:
`https://github.com/nrloucks/CapstoneReport`

Performance Results & Conclusion

Performance Results

As demonstrated in the preceding sections of this report. Each element of our system functioned as-intended.

The Gas detection unit was able to accurately measure not only gas presence, but also accurate part-per-million values for common liquid petroleum based gasses. This data was able to be transmitted and operated on in the python GUI.

The Raspberry Pi and supporting module were able to interface with each other to trigger acquisition and transmission of various forms of media.

The motion detection unit was able to detect movement at a variety of ranges, detect movement without constant sampling, transmit digital data used to trigger a response in the python GUI, as well as compensate for erroneous operations.

Conclusion

Overall this project was a success. Despite design challenges we were able to troubleshoot or find alternative solutions to new problems quickly and effectively. The change to remote learning took a toll on the rate at which we could accomplish certain aspects of the project, however through online collaboration we were able to successfully implement each aspect of our project.

It was an excellent learning experience that utilized a combination of skills and techniques learned in the Electrical and Computer Engineering program. It also shows a big project may be tackled by a team of engineers working in the field.

VII Cost Analysis and Bill of Materials

RefDes	Name/PN	Description	Value	Quantity	UNIT PRICE	EXT PRICE
C1	UVZ1H101MPD	CAPACITOR RADIAL 100U 50V +/-20%	100u	4	$0.31	$1.24
C2	EEUFR1H100	CAPACITOR RADIAL 10U 50V +/-20%	10u	4	$0.29	$1.16
C3	C315C104M5U5TA7301	CAPACITOR RADIAL 100N 50V +/-20%	100n	4	$0.24	$0.96
C4	EEUFR1H100	CAPACITOR RADIAL 10U 50V +/-20%	10u	4	$0.29	$1.16
C5	C315C104M5U5TA7301	CAPACITOR RADIAL 100N 50V +/-20%	100n	4	$0.24	$0.96
J1	644456-2	CONN HEADER MALE 1X2		4	$0.77	$3.08
J2	644456-2	CONN HEADER MALE 1X2		4	$0.77	$3.08
J3	644456-2	CONN HEADER MALE 1X2		4	$0.77	$3.08
R1	RES330	RES330 330OHM 1/4W 5% AXIAL	330 Ohm	4	$0.10	$0.40
R2	RES220	RES220 220 OHM 1/4W 5% AXIAL	220 Ohm	4	$0.10	$0.40
R3	RES330	RES330330 OHM 1/4W 5% AXIAL	330 Ohm	4	$0.10	$0.40
U1	LM317T	IC REG LIN POS ADJ 1.5A TO220AB		4	$0.71	$2.84
U2	LM7805CT	PMIC REG LINEAR 5V 1A TO220-3		1	$1.54	$1.54
RF1	XB24CDMPIT-001	XBEE/XBEE-PRO S2C ZIGBEE RF MODULE		1	$24.99	$24.99
RF2	XB24CDMPIT-001	XBEE/XBEE-PRO S2C ZIGBEE RF MODULE		1	$24.99	$24.99
RF3	XB24CDMPIT-001	XBEE/XBEE-PRO S2C ZIGBEE RF MODULE		1	$24.99	$24.99
RF4	XB24CDMPIT-001	XBEE/XBEE-PRO S2C ZIGBEE RF MODULE		1	$24.99	$24.99
	1568-1075-ND	Board Xbee Explorer		4	$10.95	$43.80
U3	555-28027-ND	Parallax PIR motion		1	$15.00	$15.00
U4	SEN-09405/1568-1411-ND	LPG Sensor - MQ-6		1	$4.95	$4.95
	RASPBERRY PI 3 A+	Raspberry Pi 3 Model A+		1	$37.44	$37.44
	913-2664	PiCamera Module		1	$29.90	$29.90
U5	CST-931AP/102-1458-ND	"Buzzer magnetic 2-4v"		1	$1.06	$1.06
				TOTAL PRICE		$252.41

Table 2: Cost analysis and bill of materials.

Bibliography

[1] S. Catalano, *Bureau of Justice Statistics Special Report National Crime Victimization Survey, Victimization During Household Burglary NCJ 227379*. 810 Seventh Street NW 2nd Floor Washington DC 20531: U.S. Department of Justice Office of Justice Programs, 2010. [Online]. Available: `https://www.bjs.gov/content/pub/ascii/vdhb.txt`.

[2] D. International, "Xbee s2c/s2c-pro manufacturer data sheet,"

[3] A. Dey, "Semiconductor metal oxide gas sensors: A review," 2018.

[4] J. G. Webster, *The Measurement, Instrumentation and Sensors Handbook*. 2000 Corporate Blvd. N.W. Boca Raton Florida 33431: CRC Press LLC, 1999, pp. 32–116.

VIII Complete Source Code

Main GUI Code & Gas Sensor Code

```
Part 1:
#The Following code is to read extract right voltages from the sensor and compute its mean,average, and variance
#Importing the modules
import serial
import binascii
import time
import numpy as np

#initializing the function to open the XBee Coordinator
ser1=serial.Serial()
ser1.baudrate=9600 #speed
ser1.port='COM3'   #port address
ser1.open()        #open port

#For Loop to keep reading data
for i in range(15):
data=ser1.read(14)
v=binascii.hexlify(data).decode('utf-8') #decode the data to Hexadecimal
print(v)

D2=v[22:26] #Select the Voltage range from the Hexadecimal
Dec2=int(D2,16) #covert the Hexadecimal to Decimal value
print(D2)

Voltage=1.2*Dec2/1023 # formula to finally to convert decimal to voltage reading
print(volt)

average=np.average(volt) #compute the average of the voltages
mean=np.mean(volt)      #compute the mean of the voltages
vari=np.var(volt)       #compute the variance of the voltages

print('The Average of the Sensor is:')
print(average)
print('The Mean of the Sensor is:')
print(mean)
print('The Variance of the Sensor is')
print(vari)
ser1.close()       #close port

Part 2:
#This program is to read right voltages from the sensor and plot it in real time (Voltage VS Time)

#Importing the Modules
import serial
import binascii
import datetime as dt
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib import style

#initializing the function to open the XBee Coordinator
ser1=serial.Serial()
ser1.baudrate=9600 #speed
ser1.port='COM3'  #port address

#figure for plotting the Time Vs Votlage
fig = plt.figure()
ax=fig.add_subplot(1,1,1)

time=[]
volt=[]

#open the serial port of the coordinator for incomming signal from the gas sensor
ser1.open()

#Periodically from funcAnimation
def animate(i,time,volt):
data=ser1.read(14)
v=binascii.hexlify(data).decode('utf-8') #decode the data to Hexadecimal
D2=v[22:26]                          #select the Voltage range from the Hexadecimal
Dec2=int(D2,16)                      #covert the Hexadecimal to Decimal value
Voltage=1.2*Dec2/1023                #formula to finally to convert decimal to voltage reading

time.append(dt.datetime.now().strftime('%H:%M:%S')) #Display current time
volt.append(Voltage)
```

```
time=time[-20:] #Trim and updated the 20 cureent incoming data
volt=volt[-20:] #Trim and updated the 20 cureent incoming data

#Below is the style, axis label, and intervals for the graph
ax.clear()
ax.plot(time,volt)
plt.plot()
plt.xticks(rotation=45, ha='right')
plt.subplots_adjust(bottom=0.30)
plt.title('Voltage over Time')
plt.ylabel('Voltage(V)')
plt.xlabel('Time')
plt.ylim(0,1.3)
plt.grid(color='black',linestyle='dotted')
plt.axhline(0.35)

ani=animation.FuncAnimation(fig,animate,fargs=(time,volt), interval=1000)
plt.show()
ser1.close()
```

Part 3:

```
#This program is to read right voltages from the sensor and plot it in real time (PPM VS Time)
#Importing the Modules
import serial
import binascii
import datetime as dt
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib import style
import math
import csv

#initializing the function to open the XBee Coordinator
ser1=serial.Serial()
ser1.baudrate=9600 #speed
ser1.port='COM3' #port address
ser1.open()

#figure for plotting Time VS PPM
fig = plt.figure()
ax=fig.add_subplot(1,1,1)

#array for storing data
time = []
ppm=[]

#defining constants
Ro=99996.85
b=1.23
m=-0.41

#Periodically from funcAnimation
def animate(i,time,ppm):
data=ser1.read(14)
v=binascii.hexlify(data).decode('utf-8') #decode the data to Hexadecimal
D2=v[22:26]                              #select the Voltage range from the Hexadecimal
Dec2=int(D2,16)                          #covert the Hexadecimal to Decimal value
Voltage=1.2*Dec2/1023                    #formula to finally to convert decimal to voltage reading

#v1 for changing Vout
v1=4.167*Voltage

Rs=(1615000000-(v1*323000000))/((v1*16150)+(v1*20000)-(Voltage*20000)) #changing Rs
x=math.log10(Rs/Ro)                                                    #logarthimc conputation
PPM = 1000*(10**((x-b)/m))                                             #Formula to find PPM
ppm.append(PPM)

time.append(dt.datetime.now().strftime('%H:%M:%S')) #Display current time
time=time[-20:] #Trim and updated the 20 cureent incoming data
volt=volt[-20:] #Trim and updated the 20 cureent incoming data

#Below is the style, axis label, and intervals for the graph
ax.clear()
ax.plot(time,ppm)
plt.plot()
plt.xticks(rotation=45, ha='right')
plt.subplots_adjust(bottom=0.30)
plt.title('PPM over Time')
plt.ylabel('PPM')
plt.xlabel('Time')
plt.axhline(y=1000, color='r', linestyle='-')
plt.ylim([0,4000])
plt.grid(color='black',linestyle='dotted')

ani=animation.FuncAnimation(fig,animate,fargs=(time,ppm), interval=1000)
```

```
    plt.show()
    ser1.close()
```

RaspberryPi Source Code

```
from time import sleep
from picamera import PiCamera

camera = PiCamera()
camera.capture('/home/pi/Desktop/test.jpg') #saves camera picture to file location specified

import RPi.GPIO as GPIO
from picamera import PiCamera
from time import sleep
GPIO.setmode(GPIO.BOARD)
GPIO.setup(16, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #Pull down to make pin read active high
GPIO.input(16) #pin 16 being read
if GPIO.input(16):
camera = PiCamera()
camera.capture('/home/pi/Desktop/test.jpg')
print('input high/camera on')
else:
camera = PiCamera()
camera.capture('/home/pi/Desktop/testfail.jpg')
print('input low/camera off')
GPIO.cleanup()

#############################################################################################################

import pyaudio
p = pyaudio.PyAudio()
for ii in range(p.get_device_count()):
print(p.get_device_info_by_index(ii).get('name')) #test for device connection

#############################################################################################################

option batch abort
option confirm off
open sftp://pi:poop@192.168.0.9 -hostkey="ssh-ed25519 255 47:19:56:74:72:d0:4f:5a:aa:76:8a:57:4b:09:af:d6"
synchronize local C:\FileTransferTesting /home/pi/file
exit

winscp.com /script=syncscript.txt
pause

#############################################################################################################

import pyaudio
import wave

form_1 = pyaudio.paInt16 # 16-bit resolution
chans = 1 # 1 channel
samp_rate = 44100 # 44.1kHz sampling rate
chunk = 4096 # 2^12 samples for buffer
record_secs = 30 # seconds to record
dev_index = 2 # device index found by p.get_device_info_by_index(ii)
wav_output_filename = 'test1.wav' # name of .wav file

audio = pyaudio.PyAudio() # create pyaudio instantiation

# create pyaudio stream
stream = audio.open(format = form_1,rate = samp_rate,channels = chans, \
input_device_index = dev_index,input = True, \
frames_per_buffer=chunk)
print("recording")
frames = []

# loop through stream and append audio chunks to frame array
for ii in range(0,int((samp_rate/chunk)*record_secs)):
data = stream.read(chunk)
frames.append(data)

print("finished recording")

# stop the stream, close it, and terminate the pyaudio instantiation
stream.stop_stream()
stream.close()
audio.terminate()

# save the audio frames as .wav file
wavefile = wave.open(wav_output_filename,'wb')
wavefile.setnchannels(chans)
wavefile.setsampwidth(audio.get_sample_size(form_1))
```

```
wavefile.setframerate(samp_rate)
wavefile.writeframes(b''.join(frames))
wavefile.close()

#############################################################################################################

import RPi.GPIO as GPIO #imports GPIO module to interface with general purpose input and output pins
import pyaudio          #imports pyaduio python interface to interact with audio stream
import wave          #imports

GPIO.setmode(GPIO.BOARD)
GPIO.setup(16, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #Pull down to make pin read active high
GPIO.input(16)

form_1 = pyaudio.paInt16 # 16-bit resolution
chans = 1 # channel 1
samp_rate = 44100 # 44.1kHz sampling rate
chunk = 4096 # 2^12 samples for buffer
record_secs = 5 # seconds to record
dev_index = 2 # device index found by p.get_device_info_by_index(ii)
wav_output_filename = '/home/pi/file/audioexample.wav' # name of .wav file and saves to specific file location
if GPIO.input(16):
audio = pyaudio.PyAudio() # create pyaudio instantiation

# create pyaudio stream
stream = audio.open(format = form_1,rate = samp_rate,channels = chans, \
input_device_index = dev_index,input = True, \
frames_per_buffer=chunk)
print("recording")
frames = []

# loop through stream and append audio chunks to frame array
for ii in range(0,int((samp_rate/chunk)*record_secs)):
data = stream.read(chunk)
frames.append(data)

print("Done recording")

# stop the stream, close it, and terminate the pyaudio instantiation
stream.stop_stream()
stream.close()
audio.terminate()

# save the audio frames as .wav file
wavefile = wave.open(wav_output_filename,'wb')
wavefile.setnchannels(chans)
wavefile.setsampwidth(audio.get_sample_size(form_1))
wavefile.setframerate(samp_rate)
wavefile.writeframes(b''.join(frames))
wavefile.close()

else:
print ('failed')
GPIO.cleanup()

#############################################################################################################
```

Motion Sensor Control & Data Acquisition

```
import serial
import binascii
import tkinter as tk
from digi import *

# variables ---
state = 0
is_tripped = 0

# setup gui elements ---------

window = tk.Tk(screenName="Test Name", baseName=None, className='Tk', useTk=1)
window.title("Motion Sensor Console")
motion_canvas = tk.Canvas(window, bg="green", width=50, height=50)
btn_ON_OFF = tk.Button(window,
                activebackground="grey",
                text="Off",
                fg='white',
                bg='red'
                )
# window title banner
window_banner = tk.Label(window,
                text="Motion Sensor Console",
                fg='grey',
```

```
                bg='black',
                relief="solid",
                font=("Arial", 30, "normal")
                )


# function definitions -------


def set_disabled():
window.config(motion_canvas, bg="grey")


def get_motion_status():
data_raw = ''
# print("enter get motion status")
ser1 = serial.Serial('COM3', 9600, timeout=1.25)
# print("serial opened, waiting for data")
data_raw = ser1.read(14)
if data_raw == '':
   ser1.close()
   window.after(0, get_motion_status())
   window.update()
else:
   try:
      data_hex = binascii.hexlify(data_raw).decode('utf-8')
      D2 = data_hex[22:26] # extract desired bits
      base_ten_val =int(D2, 16)
      # print("this sample:")
      print(base_ten_val) # view data

   except ValueError:
      # print("caught valueError")
      ser1.close()
      # print("re-paint")
      window.update()

   else:
      print("Data received")
      if base_ten_val == 4:
         print("set indicator: red")
         motion_canvas.config(bg="red")
         print("re-paint")
         window.update()
      elif base_ten_val == 0 or '':
         print("set indicator: green")
         motion_canvas.config(bg="green")
         print("re-paint")
         window.update()
   finally:
      ser1.close()
      # print("re-paint")
      window.update()
      # print("set window interrupt")
      window.after(0, get_motion_status())

# runs once -> calls recursive "get_motion_status"


btn_ON_OFF.pack()
window_banner.pack()
motion_canvas.pack()
print("re-paint")
window.update()
print("set window interrupt")
window.after(0, get_motion_status())
```

Appendices

Appendix A

Manufacturer Data Sheets

TECHNICAL DATA MQ-6 GAS SENSOR

FEATURES

* High sensitivity to LPG, iso-butane, propane
* Small sensitivity to alcohol, smoke.
* Fast response . * Stable and long life * Simple drive circuit

APPLICATION

They are used in gas leakage detecting equipments in family and industry, are suitable for detecting of LPG, iso-butane, propane, LNG, avoid the noise of alcohol and cooking fumes and cigarette smoke.

SPECIFICATIONS

A. Standard work condition

Symbol	Parameter name	Technical condition	Remarks
Vc	Circuit voltage	5V±0.1	AC OR DC
V_H	Heating voltage	5V±0.1	ACOR DC
P_L	Load resistance	20KΩ	
R_H	Heater resistance	33Ω ±5%	Room Tem
P_H	Heating consumption	less than 750mw	

B. Environment condition

Symbol	Parameter name	Technical condition	Remarks
Tao	Using Tem	-10℃-50℃	
Tas	Storage Tem	-20℃-70℃	
R_H	Related humidity	less than 95%Rh	
O_2	Oxygen concentration	21%(standard condition)Oxygen concentration can affect sensitivity	minimum value is over 2%

C. Sensitivity characteristic

Symbol	Parameter name	Technical parameter	Remarks
Rs	Sensing Resistance	10KΩ - 60KΩ (1000ppm LPG)	Detecting concentration scope: 200-10000ppm LPG , iso-butane, propane, LNG
α (1000ppm/ 4000ppm LPG)	Concentration slope rate	≤0.6	
Standard detecting condition	Temp: 20℃±2℃ Vc:5V±0.1 Humidity: 65%±5% Vh: 5V±0.1		
Preheat time	Over 24 hour		

D. Strucyure and configuration, basic measuring circuit

	Parts	Materials
1	Gas sensing layer	SnO_2
2	Electrode	Au
3	Electrode line	Pt
4	Heater coil	Ni-Cr alloy
5	Tubular ceramic	Al_2O_3
6	Anti-explosion network	Stainless steel gauze (SUS316 100-mesh)
7	Clamp ring	Copper plating Ni
8	Resin base	Bakelite
9	Tube Pin	Copper plating Ni

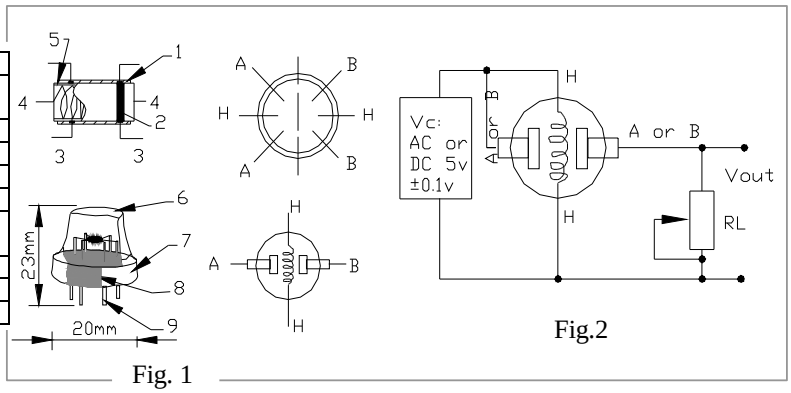
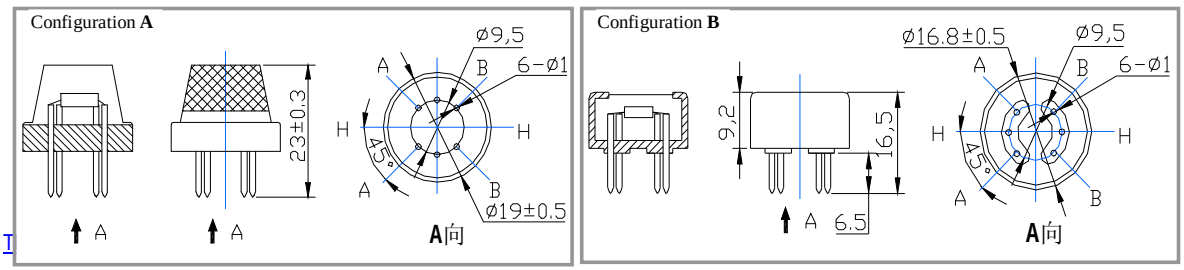


Fig. 1 Fig.2



Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

PIR Sensor (#555-28027)

The PIR (Passive Infra-Red) Sensor is a pyroelectric device that detects motion by sensing changes in the infrared (radiant heat) levels emitted by surrounding objects. This motion can be detected by checking for a sudden change in the surrounding IR pattern. When motion is detected the PIR sensor outputs a high signal on its output pin. This logic signal can be read by a microcontroller or used to drive an external load; see the source current limits in the features list below.

NOTE: Rev B of this sensor provides many updates and improvements from Rev A. If your PIR Sensor's PCB does not read "Rev B," please use the information found in the Revision History section on page 5. For the Wide Angle PIR Sensor, search for product "28032" at www.parallax.com.

Features

- Detect a person up to approximately 30 ft away, or up to 15 ft away in reduced sensitivity mode
- Jumper selects normal operation or reduced sensitivity
- Source current up to 12 mA @ 3 V, 23 mA @ 5 V
- Onboard LEDs light up the lens for fast visual feedback when movement is detected
- Mounting holes for #2 sized screws
- 3-pin SIP header ready for breadboard or through-hole projects
- Small size makes it easy to conceal
- Easy interface to any microcontroller

Key Specifications

- Power Requirements: 3 to 6 VDC; 130 µA idle, 3 mA active (no load)
- Communication: Single bit high/low output
- Operating temperature: 32 to 122 °F (0 to 50 °C)
- Dimensions: 1.41 x 1.0 x 0.8 in (35.8 x 25.4 x 20.3 cm)

Application Ideas

- Motion-activated nightlight
- Alarm systems
- Holiday animated props

LM117, LM317-N Wide Temperature Three-Pin Adjustable Regulator

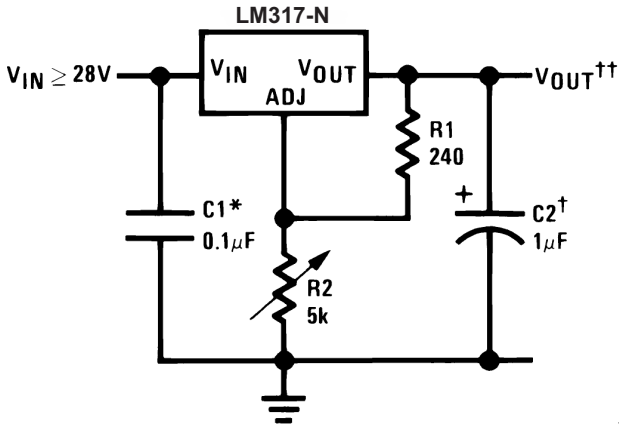
1 Features

- Typ. 0.1% Load Regulation
- Typ. 0.01%/V Line Regulation
- 1.5-A Output Current
- Adjustable Output Down to 1.25 V
- Current Limit Constant With Temperature
- 80-dB Ripple Rejection
- Short-Circuit Protected Output
- −55°C to 150°C Operating Temperature Range (LM117)

2 Applications

- Automotive LED Lighting
- Battery Chargers
- Post Regulation for Switching Supplies
- Constant Current Regulators
- Microprocessor Supplies

Typical Application



*Needed if device is more than 6 inches from filter capacitors.

†Optional—improves transient response

$$_{\dagger\dagger} V_{OUT} = 1.25\ V \left(1 + \frac{R2}{R1}\right) + I_{ADJ}\left(R_2\right)$$

3 Description

The LM117 and LM317-N series of adjustable 3-pin positive voltage regulators are capable of supplying in excess of 1.5 A over a 1.25-V to 37-V output range and a wide temperature range. They are exceptionally easy to use and require only two external resistors to set the output voltage. Further, both line and load regulation are better than standard fixed regulators.

The LM117 and LM317-N offer full overload protection such as current limit, thermal overload protection and safe area protection. All overload protection circuitry remains fully functional even if the adjustment terminal is disconnected.

Typically, no capacitors are needed unless the device is situated more than 6 inches from the input filter capacitors, in which case an input bypass is needed. An optional output capacitor can be added to improve transient response. The adjustment terminal can be bypassed to achieve very high ripple rejection ratios that are difficult to achieve with standard 3-terminal regulators.

Because the regulator is *floating* and detects only the input-to-output differential voltage, supplies of several hundred volts can be regulated as long as the maximum input-to-output differential is not exceeded. That is, avoid short-circuiting the output.

By connecting a fixed resistor between the adjustment pin and output, the LM117 and LM317-N can be also used as a precision current regulator. Supplies with electronic shutdown can be achieved by clamping the adjustment terminal to ground, which programs the output to 1.25 V where most loads draw little current.

For applications requiring greater output current, see data sheets for LM150 series (3 A), SNVS772, and LM138 series (5 A), SNVS771. For the negative complement, see LM137 (SNVS778) series data sheet.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
LM117	TO-3 (2)	38.94 mm x 25.40 mm
	TO (3)	8.255 mm × 8.255 mm
LM317-N	TO-3 (2)	38.94 mm x 25.40 mm
	TO-220 (3)	14.986 mm × 10.16 mm
	TO-263 (3)	10.18 mm × 8.41 mm
	SOT-223 (4)	6.50 mm × 3.50 mm
	TO (3)	8.255 mm × 8.255 mm
	TO-252 (3)	6.58 mm × 6.10 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Is Now Part of

ON Semiconductor®

To learn more about ON Semiconductor, please visit our website at
www.onsemi.com

Appendix C

Included Reference

Materials Science & Engineering B 229 (2018) 206–217

Contents lists available at ScienceDirect

Materials Science & Engineering B

journal homepage: www.elsevier.com/locate/mseb

Review

Semiconductor metal oxide gas sensors: A review

Ananya Dey

NMIMS Mukesh Patel School of Technology Management & Engineering, Mumbai, India

ARTICLE INFO

Keywords:
Types of gas sensors
Ammonia sensing
Semiconductor metal oxide
Sensitivity
Selectivity
Stability
Dopant induced variations

ABSTRACT

This review paper encompasses a detailed study of semiconductor metal oxide (SMO) gas sensors. It provides for a detailed comparison of SMO gas sensors with other gas sensors, especially for ammonia gas sensing. Different parameters which affect the performance (sensitivity, selectivity and stability) of SMO gas sensors are discussed here under. This paper also gives an insight about the dopant or impurity induced variations in the SMO materials used for gas sensing. It is concluded that dopants enhance the properties of SMOs for gas sensing applications by changing their microstructure and morphology, activation energy, electronic structure or band gap of the metal oxides. In some cases, dopants create defects in SMOs by generating oxygen vacancy or by forming solid solutions. These defects enhance the gas sensing properties. Different nanostructures (nanowires, nanotubes, heterojunctions), other than nanopowders have also been studied in this review. At the end, examples of SMOs are given to illustrate the potential use of different SMO materials for gas sensing.

1. Introduction

Detection and monitoring of flammable, toxic and exhaust gases are important for both energy saving as well as environmental protection [1,72]. Gas sensors have been in use for monitoring flammable as well as toxic gases in domestic and industrial environment [1]. The cheap, reliable, small and low power-consuming gas sensors are in great demand due to the wide range of applications. With the increasing demand for better gas sensors of higher selectivity and sensitivity, rigorous efforts are in progress to find more suitable material with required surface and bulk properties [13]. SMO gas sensors are generating interest as these materials fulfill the requirement of an ideal sensor to a very good extent.

Semiconductor metal oxide (SMO) gas sensors are the most investigated group of gas sensors [3] and recently the SMOs, having size in the range of 1 nm–100 nm, are being increasingly used for gas sensing due to their size dependent properties. Nanomaterials are unique because of their mechanical, optical, electrical, catalytic and magnetic properties. Apart from this, these materials also possess high surface area per unit mass. Further, new physical and chemical properties emerge when particles are in nanometer scale. The specific surface area as well as surface to volume ratio increase drastically when the size of the material decreases. Also, the movement of electrons and holes in semiconductor nanomaterials are affected by size and geometry of the materials [8]. High crystalline structure, ability of noble metal doping, and competitive production rate increase the demand of production for nanoparticles for gas sensors development [51].

This review article is focused on types of gas sensors and their comparison, factors affecting the sensitivity, selectivity and stability of SMO gas sensors, gas sensing mechanism of the above group of gas sensors. Furthermore the prospect of NH_3 sensing by different sensors are reviewed and compared with SMO gas sensors. Dopant or impurity induced variations which improve the properties of SMO materials for gas sensing applications is also being considered in this review.

2. Types of gas sensors

Over the past decades, many types of gas sensors have been developed based on different sensing materials and methods. Accordingly, the gas sensors are classified as catalytic combustion, electrochemical, thermal conductive, infrared absorption, paramagnetic, solid electrolyte and metal oxide semiconductor sensors [22]. Liu et al. [35] has classified the gas sensors based on their sensing methods and divided them to two groups: (a) methods based on variation in electrical properties and (b) methods based on variation in other properties. Materials like semiconductor metal oxides (SMO), carbon nanotubes and polymers are able to sense gas based on variation in electrical properties. The other variations are optic, acoustic, gas chromatographic and calorimetric. Comini [10] classified the gas sensors according to the measurement methods as (1) DC conductometric gas sensors (2) Field-Effect-Transistors (FET) based gas sensors (3) Photoluminescence (PL) based gas sensors. A comparison of various types of gas sensors is given in Table 1 and has been studied by Korotcenkov [29].

E-mail address: ananya.dey@nmims.edu.

https://doi.org/10.1016/j.mseb.2017.12.036
Received 31 July 2017; Received in revised form 22 November 2017; Accepted 27 December 2017
Available online 05 January 2018