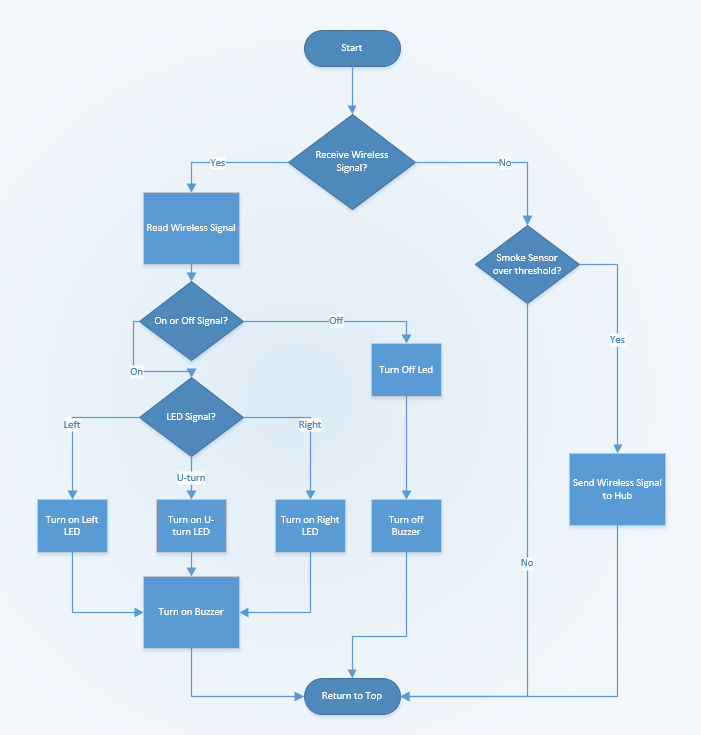
**(ADDED TO PAPER)**

**Fire Alarm Software Overview**



The software for our fire alarm system will be written using the Arduino independent development environment. The Arduino IDE uses the built in Arduino programming language to access GPIO pins and perform other operations. The Arduino programming language is actually a set of C and C++ functions that can be called from your code. The Arduino IDE performs minor changes on the Arduino sketch that is written such as automatic generation of function prototypes and then passes the code directly into a general C and C++ compiler. The compiler that is used is the avr-g++ compiler.

The software for our fire alarm will start with checking if the fire alarm has received a wireless signal from the Raspberry Pi hub over the Xbee wireless module. The hub will be receiving signals from fire alarms and using this information to calculate the return signals for each alarm. Each alarm will be checking for its’ specific return signal telling it which led to turn on and if it should turn the buzzer on.

If a wireless signal has not been received, the program will then go on to checking the response from the MQ2 Smoke Sensor. The pin that the MQ2 sensor is connected to will need to be set to input using Arduino’s pinMode() function. Once the MQ2 sensors have been configured, we will choose a default input value and a threshold value. This threshold value will be the value that the smoke sensor will send when it senses a smoke emergency. Our program will then check if MQ2 sensor is sending our fire alarm a signal that is higher than our threshold value. The program will use Arduino’s analogRead() function to read the input from the sensor and save it into an integer variable. If the alarm reads that this sensor reading is below the threshold value, no action will be performed and the program will go back to the top of the loop. If the alarm reads that this smoke sensor reading is above the threshold value, however, the alarm will send out a wireless signal over the Xbee Module to alert the system that this fire alarm is going off. This wireless signal will be received by the hub for calculation and the hub will then send out separate responses to each of the fire alarms. After a wireless signal has been sent out by the fire alarm, the alarm will restart its programming loop.

If a wireless signal has been received by the fire alarm Xbee module, the first action that the fire alarm will take will be to decode the message. This means reading the wireless message and setting the proper variables accordingly. The first part of this process will be to check if the message received via the Xbee module will be a “On” signal or an “Off’ signal.

Once the fire alarm detects that an “On” signal has been received from the Hub, the fire alarm will go into checking which LED it has been commanded to turn on. There will be three options for this. The fire alarm will get the choice of turning on the left LED, the right LED, and a U-Turn LED. Each LED will be connected to different pins on the ATMega 328 microprocessor. The Arduino language comes with a function called digitalWrite which sends a signal out to a pin on the ATMega. This function takes in a pin number and either a HIGH or LOW signal as input arguments. The pins for each LED will be defined at the top of the Arduino program. If the left LED is connected to pin 13, for example, we will have the statement “int leftLED = 13” at the top of the program. These LED pines will also need to be set as output using Arduino’s pinMode() function. So if the fire alarm gets the signal to turn on the left LED, the program will run the line “digitalWrite(leftLED, HIGH).” If the fire alarm gets the signal to turn on the right LED, the program will run the line “digitalWrite(rightLED, HIGH).” Lastly, if the fire alarm receives a signal to turn on the U-Turn led, the line “digitalWrite(uTurnLED, HIGH)” will be run.

The last part of receiving an “On” signal via the Xbee module will be to turn on the buzzer connected to the ATMega 328. Thankfully, the Arduino programming language comes with a function for controlling the sound that it output via a buzzer. The tone() function generates a square wave for a specified frequency and duty cycle for a specified pin. For the ATMega328, the tone function will be able to set a minimum frequency of 31 Hz and a maximum frequency of 65 kHz. The tone function also takes in a optional duration parameter which is specified in milliseconds. The pin that the buzzer is on will be defined in the same way that the LEDs are defined. That pin will also need to be set to output using Arduino’s pinMode() function. If our fire alarm was going to set the buzzer to 4kHz and a duration of 1 tenth of a second, the line “tone(buzzer, 4000, 100)” would be run.

If the original signal received by the Xbee module was not an “On” signal and was instead an “Off” signal, a different section of code would be run. The “Off” signal will be sent out by the Hub to all alarms when a reset function is called. This function would turn off the LEDs of a fire alarm as well as turn off the buzzer that is connected to the ATmega 328. LEDs are turned off by the system in a very similar way to being turned on. The same digitalWrite() function is called, just with the LOW parameter instead of the HIGH parameter. In order to make the processing an easier process for the Hub, the fire alarm will not need to know which LED it needs to turn off. Instead, it will just run a turn off function on all the pins connected to LEDs. To turn off the left led, the program will run the line “digitalWrite(leftLED, LOW).” To turn off the right LED, the program will then run the line “digitalWrite(rightLED, LOW).” Lastly, to turn off the U-Turn LED, “digitalWrite(uTurnLED, LOW)” will be run.

The last part of the turn off signal will be turning off the buzzer. Thankfully, the Arduino programming language also comes with a designated function for this. The noTone() function provided by Arduino is specifically designed to stop the generation of a square wave that was triggered by the tone() function. It has only one input parameter. This parameter is the pin that you would like to turn off. In order for the fire alarm to turn off the signal to the buzzer, the line “noTone(buzzer)” will be run. After this is executed, this is the final step in handling an off signal so the program will go back to running its overarching loop that checks for a wireless message or MQ2 sensor reading.