CM10194 Computer Systems Architecture 1 - Coursework 2

Wordcount = 1433

Introduction

Fire is an unanticipated event which can lead to undesirable results such as a great loss to social wealth and human life. With the rapid growth of the UK population, it is more than necessary to ensure that every household has a fully functioning fire alarm system. Despite the government efforts of introducing the smoke and carbon monoxide regulations in October 2022. According to the national government website, there has been an 11% increase in fire incidents compared to the previous year, which corresponds to 584,881 incidents in 2022.

Most fires are caused in residential homes, which means that there are greater negative externalities. As one simple fire outbreak can have a domino effect on all the other households nearby. As we have seen with the tragic incident of Grenfell Tower in 2017, which has caused the government to increase spending and invest on more fire proofed material for towers all over London.

Furthermore, such spending on more improved materials incurs a huge time lag as these investments tend to take years for it to be complete, especially since more than 1 building needs to be refurbished. This is due to the high levels of planning needed before execution. Meaning a short-term risk is still present.

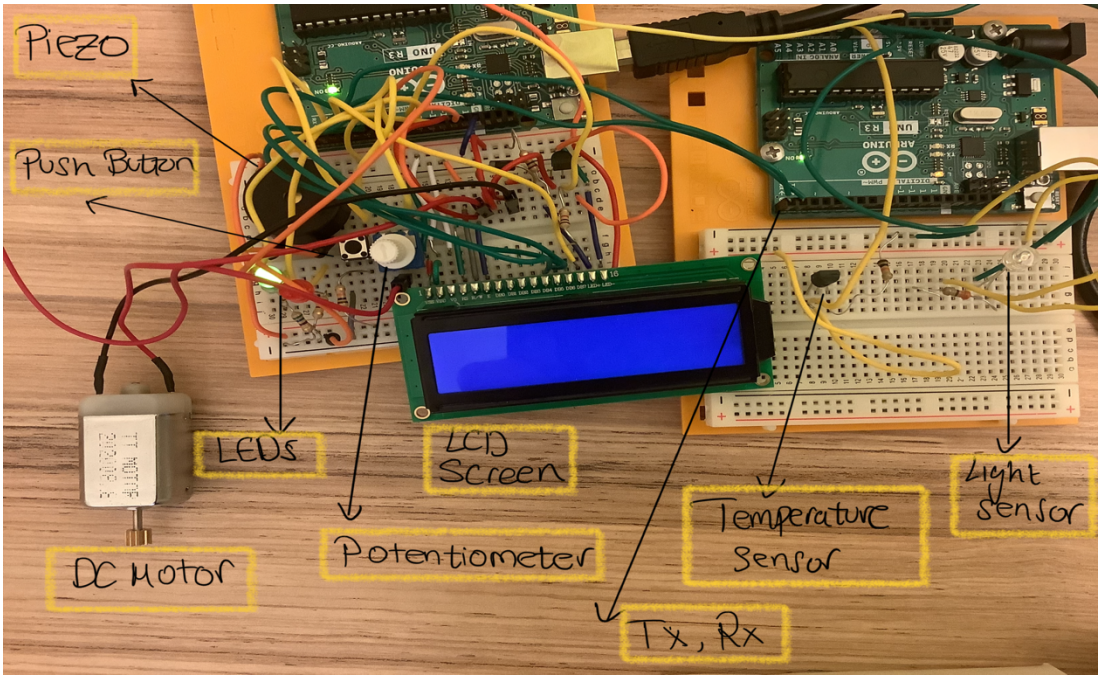
Therefore, this has given us the initiative to create a portable fire alarm system. To allow individuals to take with them in their daily travels. Hence, ensuring that no matter what location they're in, the system can detect a potential fire. Meaning, this prototype can provide an alternative solution, as it gives the individuals the right to to control their own safety. As well as a cost-effective measure as the system can be powered up using a 5V battery port.

Methodology

Equipment:

1) Light sensor
2) Temperature sensor
3) Potentiometer
4) Push Button
5) LCD
6) LEDS
7) Piezo
8) DC Motor

Figure 1



As we can see from figure 1, we have 4 inputs which consist of a temperature and light sensor which are connected to the Arduino 1 breadboard, the transmitter. Whilst the other inputs are a push button and a potentiometer. Where the push button initializes the system and acts as an on/off switch. The potentiometer was added to be able to control the speed of the outputs. Further explanation below.

The initial phase of our project was that the transmitter Arduino will measure the different values of the temperature and light in the room. Sending this data through the serial port via serial communication. With the help of connecting the Tx and Rx between the two Arduinos and having a common ground. We chose this method instead of I2C communication because this is dependent on wireless transmission and only beneficial when one is trying to communicate with multiple Arduinos.

Furthermore, to increase the likelihood of capturing a true fire signal we coded the room temperature and light sensor as a compound rule. Where both individual sensors must be above a certain threshold to trigger the outputs. This will trigger the fire alarm, as well as automatic messages repeatedly being shown on the screen until there is no longer a fire. This is to inform the individual that they need to act quickly.

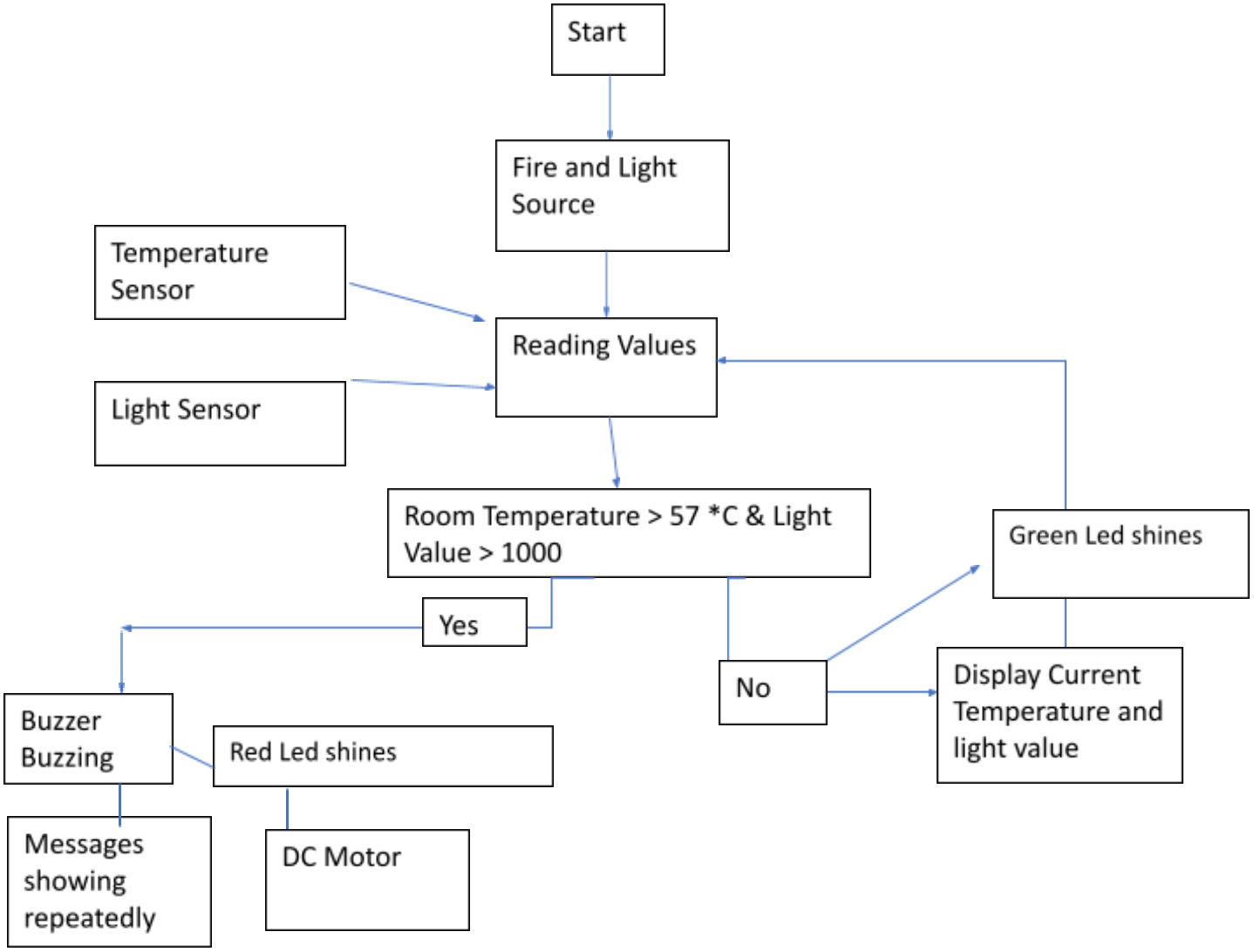
From figure 1, our outputs which are the LCD, dc motor, piezo buzzer and LED will only run if the temperature is greater than 57 degrees and light is greater than 1000 lux. These will simultaneously occur at the same time where the user can change the duration with the potentiometer. The reason why we added this is because the user can manually change the severance of the situation by being able to adjust the duration of the outputs. For example,

a shorter duration means that the outputs are acting with higher urgencies. To represent the mood and alert people quicker.

In addition, the dc motor was added as a proxy to represent a water sprinkler. Where it would be connected to a hose of water.

As we can see from figure 2, if one of the sensors were below their respective threshold, then the fire alarm outputs will not be triggered. Meaning that the LCD screen will only display the current temperature and lux of the room. As well as shining the green LED to represent that there is no fire being detected. A possible scenario is when the user could have just turned on their heaters during the night. Therefore, this makes our project robust, as a fire being lit up corresponds to high abnormal light and temperature values.

The methodology is shown in the algorithm below:



Problems

Since our project was based on serial communication via the use of Tx and Rx transmission, me and my partner decided to have most of the inputs in one Arduino and all the outputs in another Arduino. Hence, the more outputs added, resulted in a much greater accumulation of problems. This is because the different outputs were dependent on the same input, meaning when the second Arduino received the data of exceeding a certain temperature and light. There was a slow reaction in triggering all the outputs due to a time lag between sending the values to the serial port and reading them. Which was caused by each individual delay of the outputs. As the use of the delays will stop the program from running any other part, until the delay time has been completed. Whilst the transmitter Arduino kept sending data.

Therefore, this caused the two Arduinos to not operate in sync. Which was not suitable, as when the temperature sensor and light sensor decreased below their threshold values, the only output that was supposed to trigger was the LCD, as it should have displayed the normal temperature. However, due to the several delays in reading the data in the serial port, all the outputs were still triggering until all the past values in the serial port were read. As a result, if this was implemented in a real-life scenario, then the fire alarm system would have reacted too late or been on for a greater amount of time when there is no longer a present fire. Hence, sending mix signals to the user.

Therefore, the way we resolved this matter was by creating a general Millis function where it included all the states of the outputs, as it can either be on or off. Doing this, enabled all the outputs to function under one interval if enough time has elapsed from the previous program time. Thus, preventing separate outputs from overriding at different times.

Please see snapshot of code below:

```
if (val1>= 57 && val2>=1000)
{
  time_from_start = millis();
  output_interval=0;
  output_interval= output_interval + map(analogRead(Pot1),0,1023,100,1000);

  if((time_from_start - time_previous) > output_interval) // for all outputs to be under one time interval
  {
    time_previous = time_from_start;
    if(red_state ==0 && buzzer_state ==0 && motor_state ==0) // condition if outputs are off
    {
      // turn the outputs back on
      red_state =1;
      buzzer_state=1;
      motor_state=1;
      green_state=0;
    }
    else
    {
      red_state =0;
      buzzer_state=0;
      motor_state=0;
      green_state=0;
    }

   // either on/off depending on previous states
   digitalWrite(greenPin,green_state);
   digitalWrite(redPin,red_state);
   digitalWrite(motorPin,motor_state);
   if (buzzer_state == 1){
      tone(buzzer,1000 ,1000);
   }
    else{
      digitalWrite(buzzer,buzzer_state);
    }
```

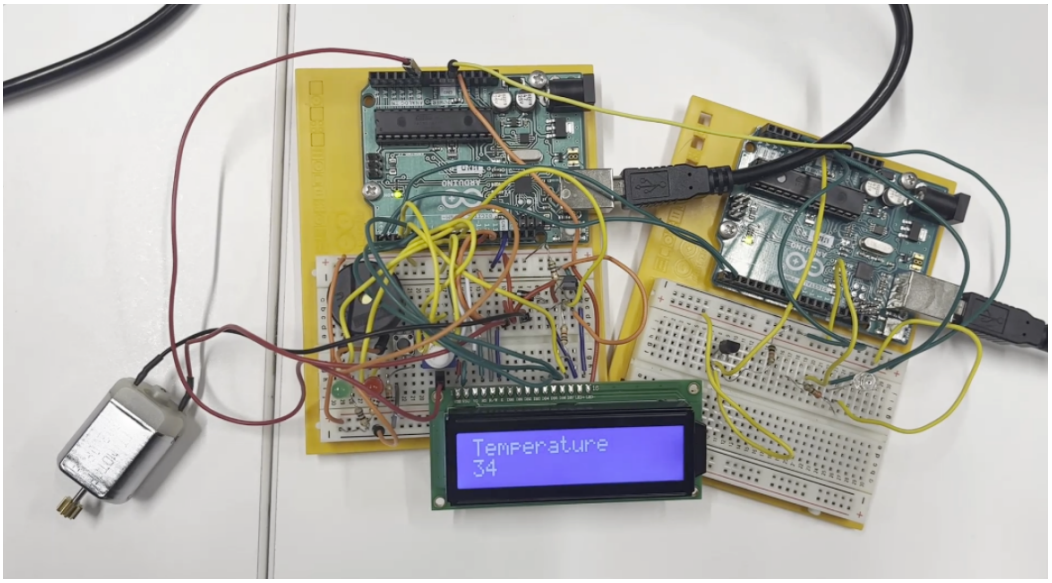
Originally when the solution was implemented, the transmitter Arduino would constantly send chars “FT” (to represent Fire True) or “FF” (to represent Fire False) and the receiver Arduino would pick up the messages and then act accordingly, however transmitting multiple chars in addition to the sensor values caused delay issues as the time taken to process all the messages was too slow compared to the time the receiver Arduino had to respond, which caused incorrect outputs by the system. We overcame this problem by reducing the messages sent to only sending the readings from the sensors with delimiters to differentiate the two readings. Once the receiver Arduino picked up the values, an if condition was ran on the readings to determine if there is a fire or not.

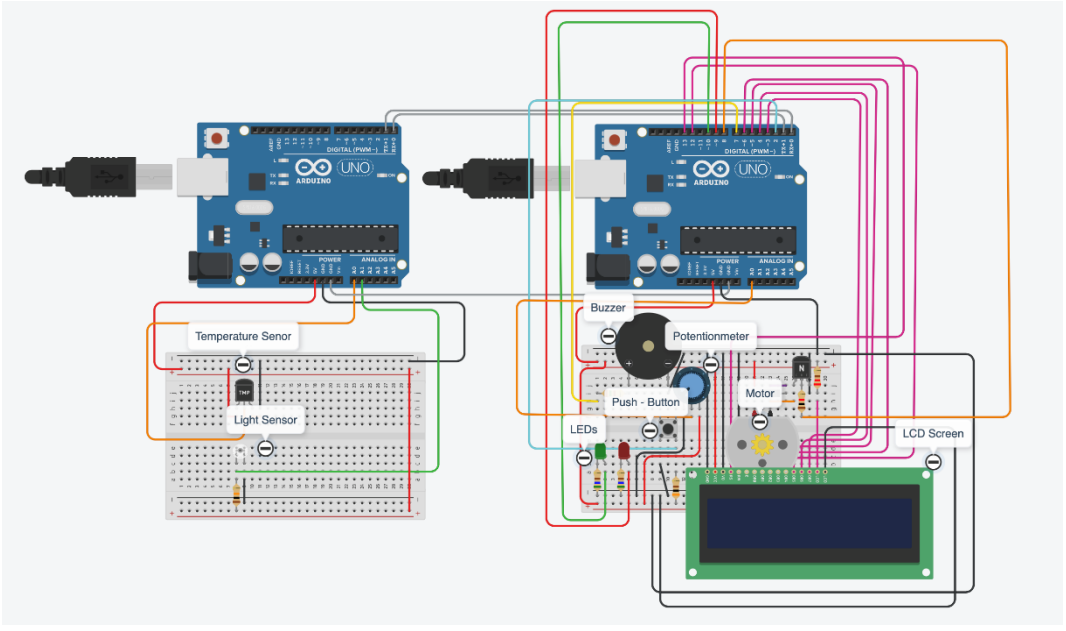
Another problem we encountered during our development process is that the DC motor was drawing too much current to directly run off the Arduino IO pin. We overcame this problem by implementing a transistor in series with the motor connected to +5V and ground as it functions as a switch that the Arduino can control. Additionally, the DC motor can produce reverse voltage spikes which can be harmful to the transistor, therefore we added a diode to protect against that.

Conclusion

Overall, we believe we have the solid foundations to be able to extend this project. As, our prototype is overly simplified where we made the assumptions that there were only two external factors which would identify a fire. However, one improvement could be capturing a gas sensor and using a GSM module. Where we could be able to send SMS messages to alert others and get help. Nonetheless, this was a great project in our opinion because we were able to take two Arduinos and potentially save lives.

Appendix:


Temperature
34


Temperature Senor
Light Sensor
Buzzer
Potentionmeter
Push - Button
Motor
LEDs
LCD Screen

Schematic view taken from Tinker cad where we built the program