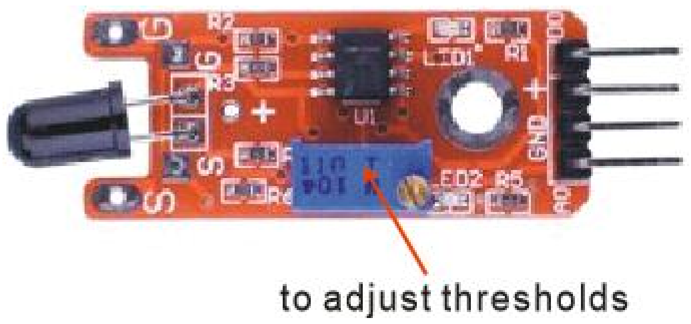
www.kumantech.com

# Flame Sensor



## Overview

Flame-sensing phototransitors are infrared (IR) detectors tuned to the 760nm – 1100nm light range, where open flame has a strong infrared signature. And their effective range depends on the size of the detected flame, and so are found in fire alarms and many other safety systems. They respond to the visible light spectrum as well as to infrared, but are particularly sensitive to the flame spectrum. The flame sensor packages an IR-detecting phototransistor inside a convenience circuit that reports both the intensity of the flame (as an analog output), and whether that intensity exceeds some user-defined threshold (as a digital output). In this experiment, you'll use your Raspberry Pi and both the analog-to-digital converter and an LED to monitor the two outputs of the flame sensor.

## Experimental Materials

| | |
|---|---|
| Raspberry Pi | x1 |
| Breadboard | x1 |
| Flame sensor | x1 |
| ADC0832 | x1 |
| LED (3-pin) | x1 |
| Resistor (330Ω) | x1 |
| Dupont jumper wires | |

## Experimental Procedure

1. If you have not done so already, prepare your development system by installing the Python interpreter, RPi.GPIO library, and wiringPi library as described in READ_ME_FIRST.TXT.

2. Install the ADC0832 analog/digital converter IC, flame sensor, three-pin LED and resistor on your breadboard, and use Dupont jumper wires to connect them to each other and your Raspberry Pi as illustrated in the Wiring Diagram below. Note you will connect only two of the three pins on the LED.

3. Execute the sample stored in this experiment's subfolder.
   If using C, compile and execute the C code:

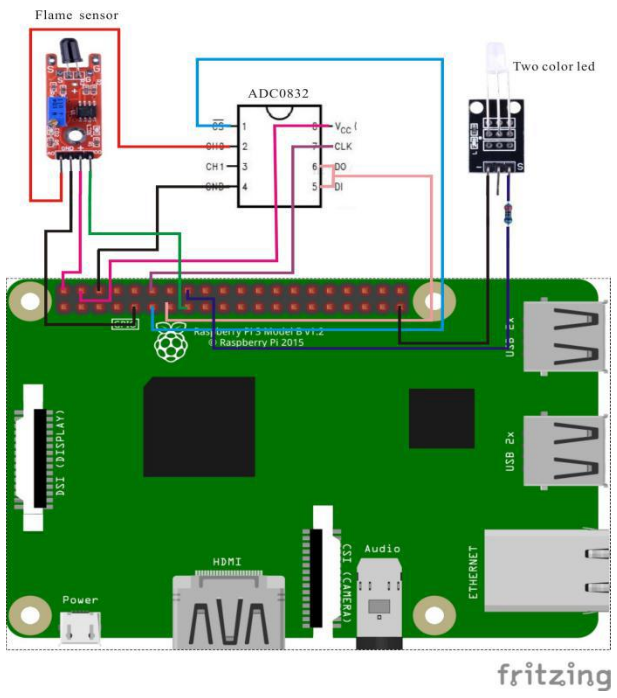
   ```
   cd Code/C
   gcc flameSensor.c -o flameSensor.out -lwiringPi
   ./flameSensor.out
   ```

   If using Python, launch the Python script:

   ```
   cd Code/Python
   python flameSensor.py
   ```

4. Make experimental observations as you approach the sensor with a small open flame (a candle, match, or lighter) from a meter away. The Raspberry Pi reports the increasing analog signal through its command-line interface, and when the flame exceeds the threshold determined by the onboard potentiometer, the LED comes on. Remove the flame to reduce the reported analog strength and extinguish the LED. Don't bring the flame too close to the sensor, as you can damage it with high temperatures. Instead you can adjust the threshold of sensitivity for the LED by varying the onboard potentiometer.

## Wiring Diagram



ADC0382 pin position:

| | | |
|---|---|---|
| CS | ↔ | Raspberry Pi Pin 11 |
| CLK | ↔ | Raspberry Pi Pin 12 |
| DI | ↔ | Raspberry Pi Pin 13 |
| D0 | ↔ | Raspberry Pi Pin 13 |
| CH0 | ↔ | Flame Sensor Pin A0 |
| VCC | ↔ | Raspberry Pi +5V |
| GND | ↔ | Raspberry Pi GND |

Flame Sensor pin position:

| | | |
|---|---|---|
| A0 | ↔ | ADC0382 Pin CH0 |
| D0 | ↔ | Raspberry Pi Pin 15 |
| GND | ↔ | Raspberry Pi GND |
| "+" | ↔ | Raspberry Pi +5V |

LED pin position:

| | | |
|---|---|---|
| "S" | ↔ | Raspberry Pi Pin 16(through resistor) |
| "-" | ↔ | Raspberry Pi GND |

## Sample Code

### Python Code

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import ADC0832
import time
```

```
Flame_DO_Pin = 15
LedPin = 16
thresholdVal = 150

def init():
   GPIO.setmode(GPIO.BOARD)
   GPIO.setup(Flame_DO_Pin, GPIO.IN,
pull_up_down=GPIO.PUD_DOWN)
   GPIO.setup(LedPin, GPIO.OUT)
   ADC0832.setup()

def loop():
    while True:
        global digitalVal, analogVal
        analogVal = ADC0832.getResult(0)
        print 'Current analog value is %d'% analogVal
        GPIO.output(LedPin, GPIO.input(Flame_DO_Pin))
        time.sleep(0.2)

if __name__ == '__main__':
   init()
   try:
      loop()
   except KeyboardInterrupt:
      ADC0832.destroy()
      print 'The end !'
```

## C Code

```
#include <wiringPi.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>

#define     ADC_CS    0
#define     ADC_CLK   1
#define     ADC_DIO   2

#define  Flame_DO_Pin   3
#define  LedPin         4
#define thresholdVal 150
```

```
typedef unsigned char uchar;
typedef unsigned int uint;


uchar get_ADC_Result(void)
{
   uchar i;
   uchar dat1=0, dat2=0;

   digitalWrite(ADC_CS, 0);
   digitalWrite(ADC_CLK,0);
   digitalWrite(ADC_DIO,1);  delayMicroseconds(2);
   digitalWrite(ADC_CLK,1);  delayMicroseconds(2);

   digitalWrite(ADC_CLK,0);
   digitalWrite(ADC_DIO,1);   delayMicroseconds(2);
   digitalWrite(ADC_CLK,1);  delayMicroseconds(2);

   digitalWrite(ADC_CLK,0);
   digitalWrite(ADC_DIO,0);  delayMicroseconds(2);
   digitalWrite(ADC_CLK,1);
   digitalWrite(ADC_DIO,1);   delayMicroseconds(2);
   digitalWrite(ADC_CLK,0);
   digitalWrite(ADC_DIO,1);   delayMicroseconds(2);

   for(i=0;i<8;i++)
   {
      digitalWrite(ADC_CLK,1);  delayMicroseconds(2);
      digitalWrite(ADC_CLK,0);   delayMicroseconds(2);

      pinMode(ADC_DIO, INPUT);
      dat1=dat1<<1 | digitalRead(ADC_DIO);
   }

   for(i=0;i<8;i++)
   {
      dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
      digitalWrite(ADC_CLK,1);  delayMicroseconds(2);
      digitalWrite(ADC_CLK,0);   delayMicroseconds(2);
   }
   digitalWrite(ADC_CS,1);
   pinMode(ADC_DIO, OUTPUT);
   return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
   uchar digitalVal = 1;
   uchar analogVal = 0;
   if(wiringPiSetup() == -1)
   {
      printf("setup wiringPi failed !");
      return -1;
   }


   pinMode(ADC_CS,  OUTPUT);
   pinMode(ADC_CLK, OUTPUT);
   pinMode(Flame_DO_Pin, INPUT);
   pullUpDnControl(Flame_DO_Pin, PUD_DOWN);
   pinMode(LedPin, OUTPUT);

   while(1)
   {
      printf("Current analog value is %d.\n",
get_ADC_Result());
      digitalWrite(LedPin, digitalRead(Flame_DO_Pin));
      delay(200);
   }

   return 0;
}
```