

Laboratory 6 – Digital Sensors, Analog Keypad

1. The use of a digital sensor

Most microcontroller based applications use inputs from different analog or digital sensors. Here we will briefly study the datasheet of digital sensor, in order to see it's characteristics, if it is suited for the desired application and how it can be configured and interfaced to the microcontroller.

a. Read the Datasheet

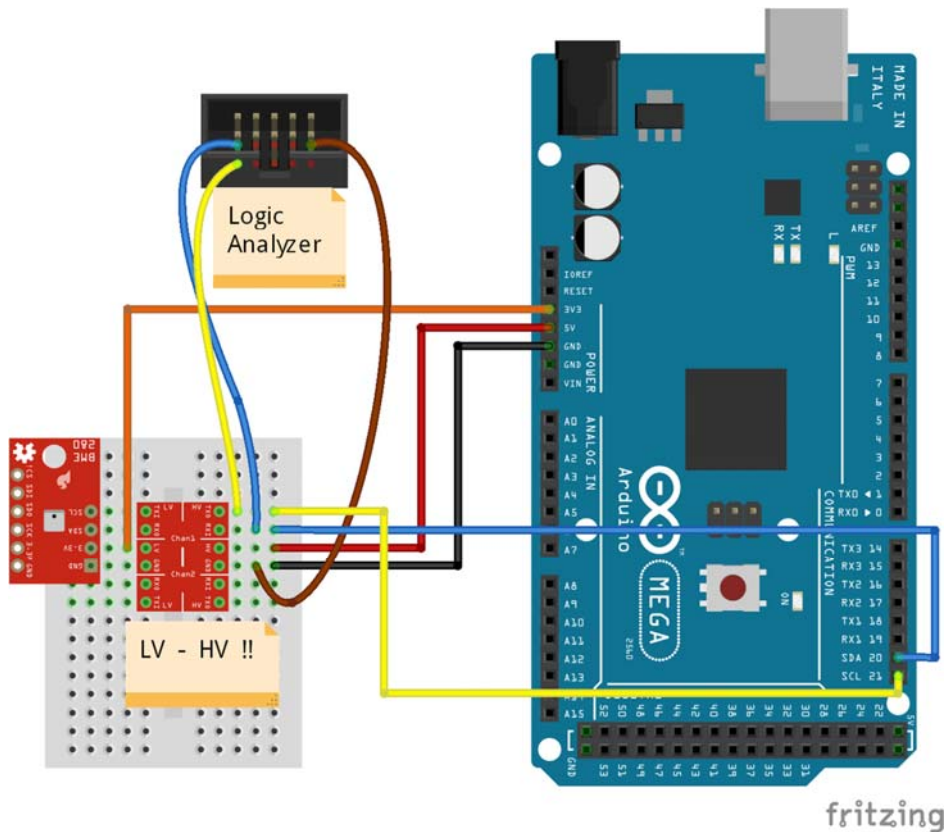
We will use the Bosch BME280, a very precise environment sensor, that provides the temperature, atmospheric pressure and air humidity. This sensor has two communication interfaces: SPI (Serial Peripheral Interface) and I2C (Inter-Integrated Circuit). We will use the I2C interface which was studied in the previous laboratory.

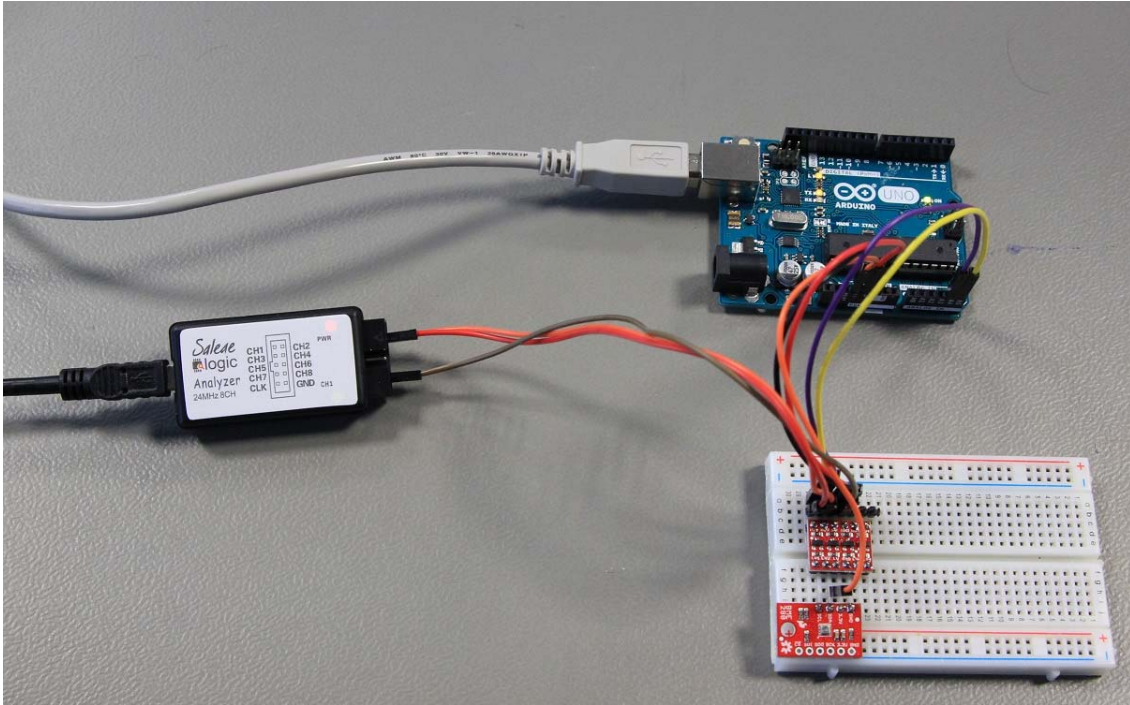
Open the datasheet from the link provide below and search for:

- Supply voltage
- Sensibility and precision of measurement
- I2C interface usage
- How to interpret the provided data

Datasheet: https://ae-bst.resource.bosch.com/media/tech/media/datasheets/BST-BME280_DS001-11.pdf

b. Electrical Connections





As you could see in the datasheet, the sensor has a maximum operating voltage of 3.6 V, and the Arduino board is powered at 5V; the HIGH level of the digital pins is also 5V. If one would connect the sensor directly to the Arduino board, the sensor will be damaged (burned). In order to prevent this, we will use a Logic Level Shifter between the sensor and the Arduino board, for adapting the voltage of the I2C digital interface. For powering up the sensor we will use the 3.3 V output from the Arduino board.

c. Read Temperature example

```
#include <Wire.h>

#define BME280_ADDRESS          (0x77) // I2C address of the sensor

#define BME280_REGISTER_CHIPID  (0xD0) // ChipID register address
#define BME280_REGISTER_CONTROLHUMID (0xF2) // Humidity configuration register address
#define BME280_REGISTER_CONTROL  (0xF4) // Measurement configuration register address
#define BME280_REGISTER_TEMPDATA (0xFA) // Temperature data register address

typedef struct
{
  uint16_t dig_T1; int16_t dig_T2; int16_t dig_T3;
  uint16_t dig_P1; int16_t dig_P2; int16_t dig_P3; int16_t dig_P4; int16_t dig_P5; int16_t dig_P6;
  int16_t dig_P7; int16_t dig_P8; int16_t dig_P9;
  uint8_t dig_H1; int16_t dig_H2; uint8_t dig_H3; int16_t dig_H4; int16_t dig_H5; int8_t dig_H6;
} bme280_calib_data;

// Calibration factors. Usually these are read from the ROM memory of the sensor.
bme280_calib_data bme280_calib = {28398, 26500, 50, 36953, -10772, 3024, 7550, 6, -7, 9900, -10230, 4285,
75, 368, 0, 302, 0, 30};
int32_t t_fine;

// Compensation function from the Datasheet, page 23.
// https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME280_DS001-11.pdf
float compensateTemperature(int32_t adc_T)
{
  int32_t var1, var2;
  adc_T >>= 4;

  var1 = (((adc_T >> 3) - ((int32_t)_bme280_calib.dig_T1 << 1)) *
          ((int32_t)_bme280_calib.dig_T2) >> 11;
```

```
var2 = (((((adc_T>>4) - ((int32_t)_bme280_calib.dig_T1)) *
          ((adc_T>>4) - ((int32_t)_bme280_calib.dig_T1))) >> 12) *
        ((int32_t)_bme280_calib.dig_T3)) >> 14;

t_fine = var1 + var2;

float T = (t_fine * 5 + 128) >> 8;
return T/100;
}

void setup() {
  Serial.begin(9600);
  Wire.begin();
  Serial.println("BME280 test");
  delay(500);

  // Read Register Example
  // We read the CHIPID
  Wire.beginTransmission((uint8_t)BME280_ADDRESS);
  Wire.write((uint8_t)BME280_REGISTER_CHIPID);
  Wire.endTransmission();
  Wire.requestFrom((uint8_t)BME280_ADDRESS, (byte)1);
  uint8_t value = Wire.read();
  Serial.print("Registru Chip_Id (0xD0): ");
  Serial.println(value);

  // Write Register Example
  // Set the oversampling factor for humidity to 16x
  Wire.beginTransmission((uint8_t)BME280_ADDRESS);
  Wire.write((uint8_t)BME280_REGISTER_CONTROLHUMID);
  Wire.write((uint8_t)0x05);
  Wire.endTransmission();

  // Set the oversampling factor for temperature and pressure to 16x + Normal mode (Not sleep)
  Wire.beginTransmission((uint8_t)BME280_ADDRESS);
  Wire.write((uint8_t)BME280_REGISTER_CONTROL);
  Wire.write((uint8_t)0xB7);
  Wire.endTransmission();
}

void loop() {

  // Read Temperature
  Wire.beginTransmission((uint8_t)BME280_ADDRESS);
  Wire.write((uint8_t)BME280_REGISTER_TEMPDATA);
  Wire.endTransmission();

  // The raw temperature is found in registers 0xFA, 0xFB, 0xFC.
  // We can read the registers sequentially; the address is auto incremented
  Wire.requestFrom((uint8_t)BME280_ADDRESS, (byte)3);
  uint32_t raw_temperature;
  raw_temperature = Wire.read();
  raw_temperature <<= 8;
  raw_temperature |= Wire.read();
  raw_temperature <<= 8;
  raw_temperature |= Wire.read();

  // Raw temperature is encoded on 20 bits
  Serial.print("Temperatura raw: ");
  Serial.println(raw_temperature >> 4);

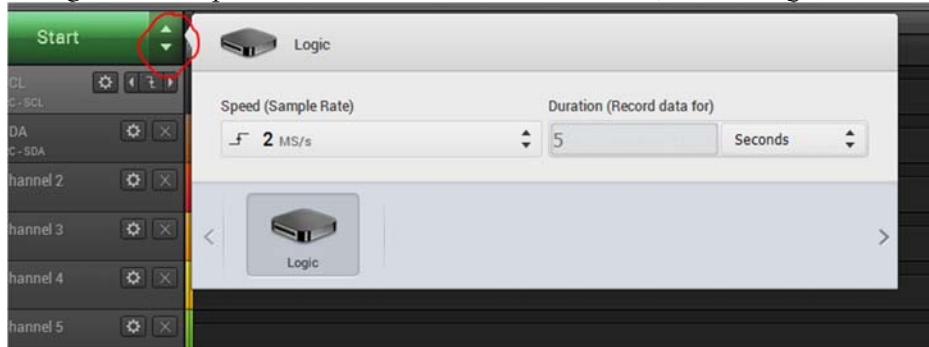
  // Compensate temperature according to the calibration factors
  float temperature = compensateTemperature(raw_temperature);
  Serial.print("Temperatura: ");
  Serial.println(temperature);

  delay (1000);
}
```

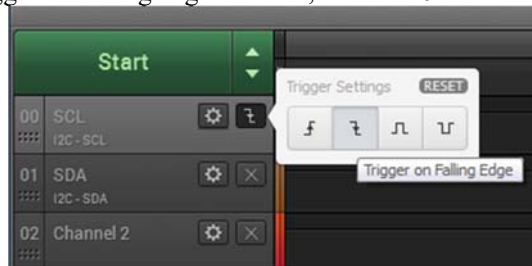
d. Using the Logic Analyzer

The logic analyzer is used to visualize the data traffic between more devices and is especially useful in problem solving. Follow the next steps for visualizing the traffic towards the sensor:

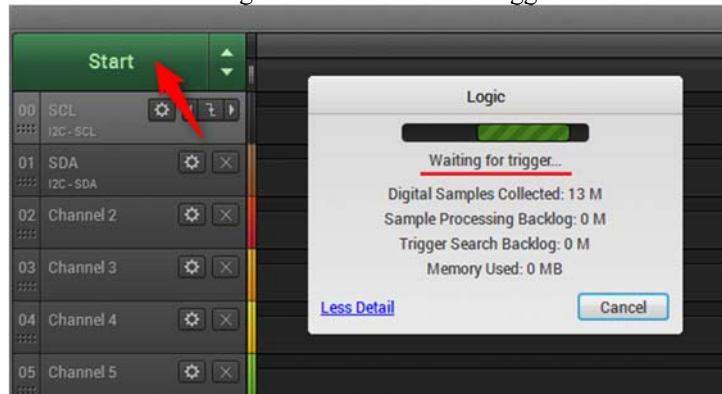
- Connect the logic analyzer as displayed in the above figures. Check that the SCL line is connected to CH0 and SDA to CH1.
- Leave the Arduino board disconnected from the PC
- Configure the Sample Rate to 2MS/s and the duration to 5s, as in the figure below:



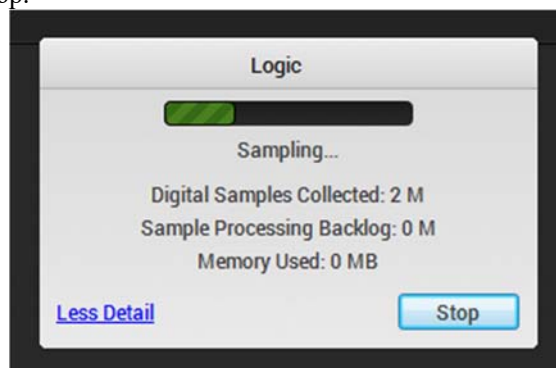
- Configure the trigger on falling edge for SCL, Channel 0.



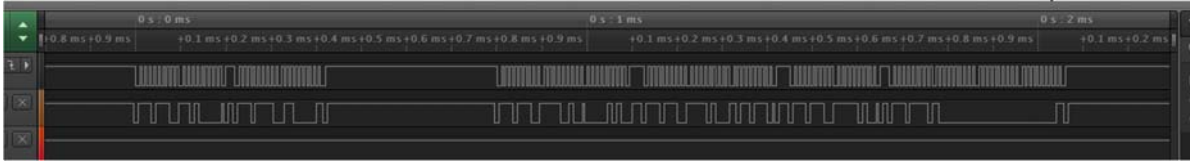
- Press the start button. The “Logic” will wait for the Trigger



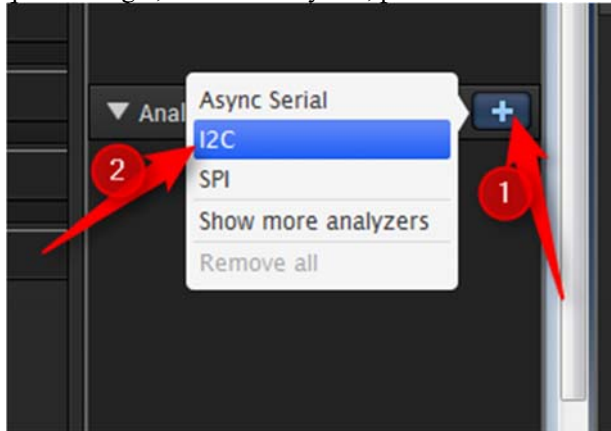
- Connect the Arduino board to power supply (PC). The recording should start now and after 5 s to automatically stop.



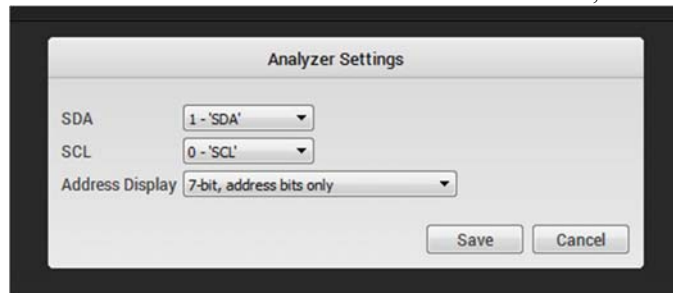
- We can see the traffic between the Arduino board and the sensor. Now we can analyze it.



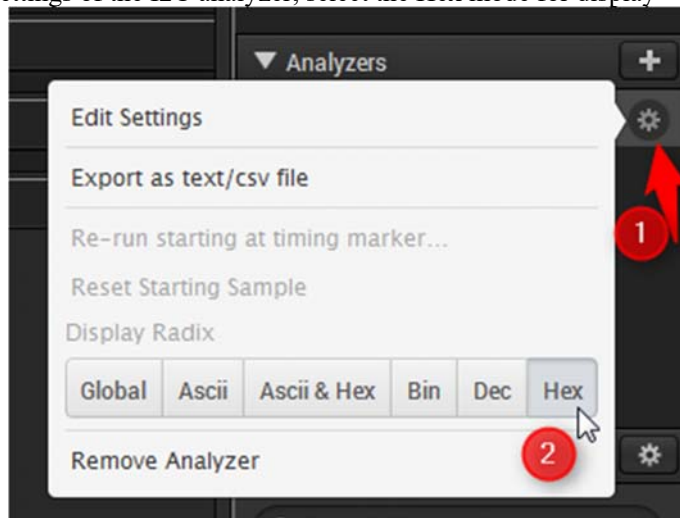
- From the right part of Logic, section Analyzers, press '+' and add the I2C analyzer



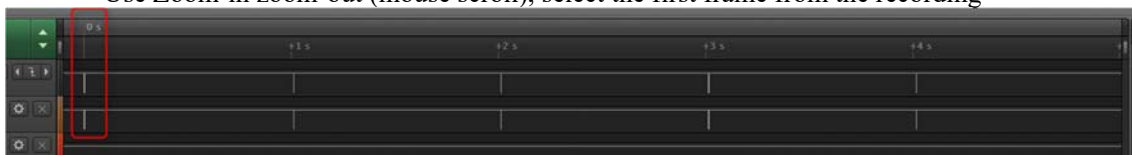
- Be sure that SDA is allocated to channel 1 and SCL to channel 0; set the address to 7-bits



- From the settings of the I2C analyzer, select the Hex mode for display



- Use Zoom-in zoom-out (mouse scroll), select the first frame from the recording



- See how the ChipID register is read; compare the decoded data from the Logic analyzer with the code from the setup function that reads the ChipID register



```
// Exemplu citire registru
// Citim CHIPID
Wire.beginTransmission((uint8_t)BME280_ADDRESS); // BME280_ADDRESS = 0x77
Wire.write((uint8_t)BME280_REGISTER_CHIPID);    // CHIPID      = 0xD0
Wire.endTransmission();
Wire.requestFrom((uint8_t)BME280_ADDRESS, (byte)1);
uint8_t value = Wire.read();
Serial.print("Registru Chip_Id (0xD0): ");
Serial.println(value);
```

- Search for one of the transactions that read the temperature and compare it with the code.



e. Read the pressure and humidity

Read the pressure and humidity by using the compensation functions below. For this you should look-up the addresses for the pressure and humidity registers, from the sensor's datasheet. Be careful at the width of these registers.

```
// Compensation function from the Datasheet, page 23.
float compensatePressure(int32_t adc_P) {
    int64_t var1, var2, p;

    adc_P >>= 4;
    var1 = ((int64_t)t_fine) - 128000;
    var2 = var1 * var1 * (int64_t)_bme280_calib.dig_P6;
    var2 = var2 + ((var1*(int64_t)_bme280_calib.dig_P5)<<17);
    var2 = var2 + (((int64_t)_bme280_calib.dig_P4)<<35);
    var1 = ((var1 * var1 * (int64_t)_bme280_calib.dig_P3)>>8) +
            ((var1 * (int64_t)_bme280_calib.dig_P2)<<12);
    var1 = (((((int64_t)1)<<47)+var1))*((int64_t)_bme280_calib.dig_P1)>>33;

    if (var1 == 0) {
        return 0;
    }
    p = 1048576 - adc_P;
    p = (((p<<31) - var2)*3125) / var1;
    var1 = (((int64_t)_bme280_calib.dig_P9) * (p>>13) * (p>>13)) >> 25;
    var2 = (((int64_t)_bme280_calib.dig_P8) * p) >> 19;

    p = ((p + var1 + var2) >> 8) + (((int64_t)_bme280_calib.dig_P7)<<4);
    return (float)p/256;
}

// Functia de compensare din Datasheet, de la pag. 23
float compensateHumidity(int32_t adc_H) {
    int32_t v_x1_u32r;

    v_x1_u32r = (t_fine - ((int32_t)76800));

    v_x1_u32r = (((((adc_H << 14) - ((int32_t)_bme280_calib.dig_H4) << 20) -
                    (((int32_t)_bme280_calib.dig_H5) * v_x1_u32r) + ((int32_t)16384)) >> 15) *
                ((((((v_x1_u32r * ((int32_t)_bme280_calib.dig_H6)) >> 10) *
                    (((v_x1_u32r * ((int32_t)_bme280_calib.dig_H3)) >> 11) + ((int32_t)32768)))) >> 10) +
                    ((int32_t)2097152)) * ((int32_t)_bme280_calib.dig_H2) + 8192) >> 14));
```

```

v_x1_u32r = (v_x1_u32r - (((((v_x1_u32r >> 15) * (v_x1_u32r >> 15)) >> 7) *
((int32_t)_bme280_calib.dig_H1) >> 4));

v_x1_u32r = (v_x1_u32r < 0) ? 0 : v_x1_u32r;
v_x1_u32r = (v_x1_u32r > 419430400) ? 419430400 : v_x1_u32r;
float h = (v_x1_u32r >> 12);
return h / 1024.0;
}

```

2. Analog Keypad

In the second of the laboratory we will use an analog keypad. This is a first example for using the analog to digital converter which comes with the AVR microcontroller from the Arduino boards; in the next laboratory we will use other examples with analog sensors. For now, it is enough to know that an analog value is read by using the **analogRead(pinNb)** instruction.

The analog keypad is depicted in the next figure.

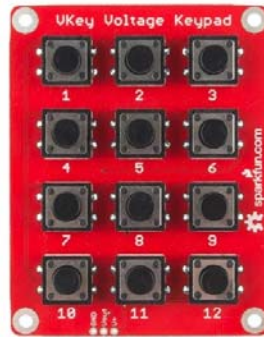


Fig 1. Analog Keypad

We will not go into details about the functionality of this component. The important thing to remember is that when pressing a key from the keypad, a voltage is transmitted on the Vout pin. In the next figure you can find the nominal Vout voltage for each key.

Key Number	V _{out} (5V supply)	V _{out} (3.3V supply)
None	.057	.053
12	.198	.194
11	.396	.388
10	.596	.583
9	.794	.776
8	.992	.970
7	1.190	1.164
6	1.388	1.358
5	1.585	1.551
4	1.781	1.744
3	1.979	1.938
2	2.176	2.131
1	2.372	2.323

Fig 2. Voltage values corresponding to each pressed key

If more than one key is pressed, Vout will correspond to the key with the highest numerical value.

For this example, you need an Arduino board with LCD shield and an analog keypad. See the next image for realizing the setup (connect the VCC and GND keypad pins to 5V and GND on the Arduino board, and the Vout output pin of the keypad to the A1 input from the Arduino board).

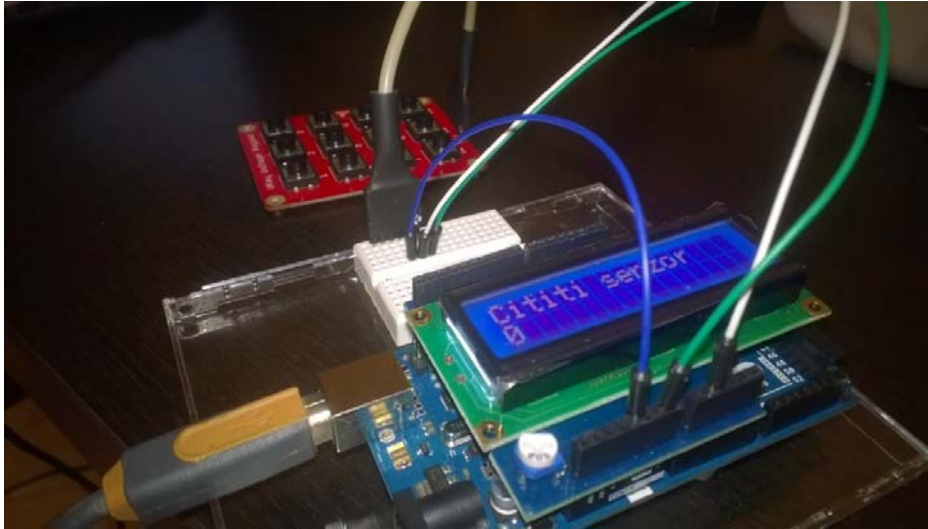


Fig 3. Analog Keypad setup

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
//use an intermediate value for clearing the LCD screen only when a new keypad is pressed
int valIntermediar;

void setup()
{
  //set the reference voltage to 5V
  analogReference(DEFAULT); //setarea tensiunii de referinta la tensiunea default
  lcd.begin(16, 2);
  lcd.setCursor(0,0);
  lcd.print("Cititi senzor");
  pinMode(A1, INPUT); // set the A1 analogic pin as input
  digitalWrite(A1, HIGH); //activate the pull up resistor
  //initialize the intermediate value
  valIntermediar = 1000;
}

void loop()
{
  int val = analogRead(A1); //read the analog value
  //get the key corresponding to the analog voltage level
  val = getTasta(val);
  if(valIntermediar != val)
  {
    valIntermediar = val;
    lcd.clear();//clear screen
    lcd.setCursor(0,0);
    lcd.print("Cititi senzor");
    lcd.setCursor(0,1);//display the new value
    lcd.print(valIntermediar);
  }
}
/*
This function gets an analog value as parameter and returns the corresponding key for thi
value. Because the analog signal is not filtered nor stabilized it continuously fluctuates.
For obtaining only our desired keypad values we build a hysteresis. These intervals have
been experimentally chosen, based on the voltage levels presented in the keypad's datasheet.
*/
int getTasta(int val)
{
  if(val > 50 && val < 80)return 12;//key 12 is pressed
```



```
else if(val > 90 && val < 120) return 11; // key 11 is pressed
else if(val > 120 && val < 160) return 10; // and so on
else if(val > 160 && val < 200) return 9;
else if(val > 200 && val < 240) return 8;
else if(val > 240 && val < 290) return 7;
else if(val > 290 && val < 320) return 6;
else if(val > 320 && val < 370) return 5;
else if(val > 370 && val < 410) return 4;
else if(val > 410 && val < 450) return 3;
else if(val > 450 && val < 490) return 2;
else if(val > 490 && val < 530) return 1;
else return 0; //for any other value return 0 - no key is pressed
}
```

!!! Attention

A problem with the analog sensors is that the read values fluctuate. This can be fixed in hardware by using a capacitor or a low pass filter for eliminating the spike effects, or it can be fixed in software by reading more values and computing the average or median of these values.

Individual work

1. Implement and test the examples from this laboratory.
2. Add the functionalities for reading the pressure and humidity from the Bosch BME280 sensor. Lift the sensor by 50 cm or more above the table and see the change in atmospheric pressure. Breathe hot air towards the sensor and observe the change in temperature and humidity.
3. Create a simple thermostat. Read the desired temperature and humidity values from the analog keypad. Turn on the heat, air conditioner or humidifier until you reach the desired parameters.