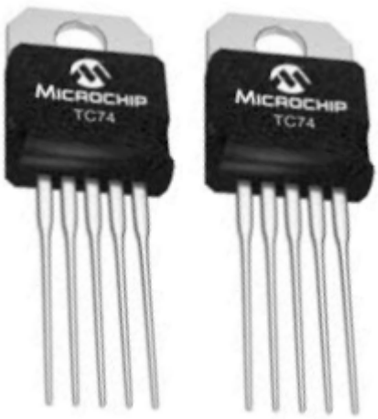
How to Connect Multiple I^{2}C devices to an Arduino Microcontroller



In this project, we will show how to connect multiple I^{2}C devices to an arduino microcontroller.

The I^{2}C bus is a bus which enables high-speed two-way communication between devices while using a minimal number of I/O pins to facilitate communication.

An I^{2}C bus is controlled by a master device (usually a microcontroller) and contains one ore more slave devices that receive information from the master device.

The I^{2}C protocol was created by Phillips in the early 1980s and was standardized and adopted widespread in the 1990s. The protocol is known as the "two-wire" protocol because 2 lines are used for communication. These 2 lines are the clock line and data line.

I^{2}C protocol can use multiple devices that all share the same communication lines: a clock signal (SCL) and a bidirectional data line used for sending information back and forth between the master and slave (SDA).

In order to work, the 2 lines of the I^{2}C, the clock and data lines, need pull-up resistors to the positive voltage source.

The I^{2}C bus allows multiple slave devices to share communication lines with a single master device. The Arduino acts as the master device. The bus master is responsible for initiating all communications. Slave devices cannot initiate communications; they only respond to requests that are sent by the master device. This prevents multiple slave devices from all trying to communicate at the same thing, causing garbled messages.

When a command or request is sent out by the master device, it is received by all slave devices on the bus. Each I^{2}C device has a unique 7-bit address, or ID number. When communication is initiated by the master device, a

device ID is transmitted. I^2C slave devices react to data on the bus only when it is drected at their ID number. Since all the devices are receiving all the data transmitted by the master device, each device has to have a unique ID number so that the master can speak to a particular slave device.

Some I^2C devices have selectable addresses whereas others come form the manufacturer with a fixed address.

An example of an I^2C device that has fixed addresses is the TC74 temperature sensor. This means that the addresses are fixed by the manufacturer. However, just because it is fixed doesn't mean you can't have multiple TC74 sensors connected together. It's possible to connect possible sensors to an I^2C bus by purchasing this IC with eight different ID numbers. So you could connect up to eight of them on a single bus. Each of the 8 lines having different addresses.

Other I^2C, such as the AD7414 and AD7415, have address select (AS) pins that allow you to configure the I^2C address of the device.

the basic steps for controlling any I^2C device are as follows.

1. The Master device sends a start bit
2. The Master sends 7-bit slave address of the slave device it wants to talk to
3. The Master sends a read (1) or write (0) bit depending on whether it wants to write data into an I^2C device's register or if it wants to read from one of the I^2C device's registers.
4. The Slave device then responds with an "acknowledge" or ACK bit (a logic low)
5. In write mode, the master sends 1 byte of information at a time, and slave responds with ACKs. In read mode, the master receives 1 byte of information at a time and sends an ACK to the slave after each byte
6. When communication has been completed, the master sends a stop bit

Components Needed

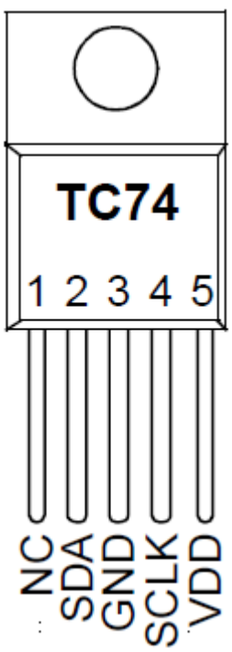
- TC74A0-5.0VAT sensor
- TC74A1-5.0VAT sensor
- 2 4.7KΩ resistors
- Arduino microcontroller

The TC74 temperature we use specifically is the TC74A0-5.0VAT and the TC74A1-5.0VAT sensor.

These sensors both run on 5V on power.

The only difference between these sensors is that they have addresses. Being that we are connecting 2 TC74 sensors to an arduino, they need to have distinguishable addresses so that the arduino can differentiate how to communicate with each of these sensors. If we bought 2 TC74A0s, they would have the same address

The TC74 sensor has 5 pins.



The NC pin is just a Not Connected pin. It doesn't connect to our circuit.

The SDA is the serial data pin. This is how the arduino microcontroller and the temperature sensor share data with each other. Data is shared bidirectionally.

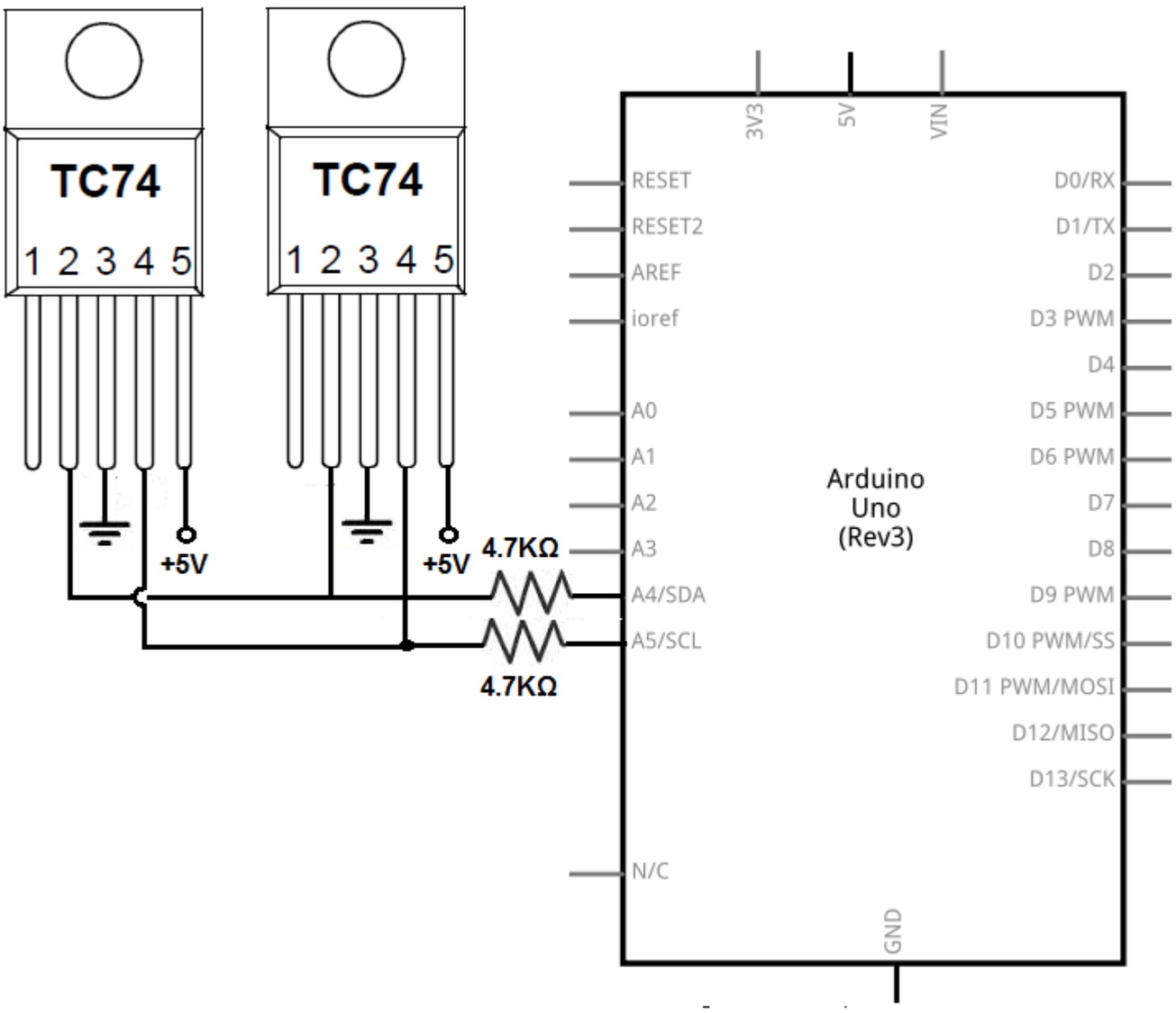
The third pin is the power ground pin.

The fourth pin is the SCLK pin. This allows for communication on a clock signal.

And the fifth pin is V_{DD}, which gets connected +5V.

Circuit Connecting 2 I^2C TC74 Temperature Sensors to an Arduino Microcontroller

The I^2C temperature sensors circuit we will build with an Arduino is shown below.



In this circuit, the hardware connections are very simple.

Pin 1 is NC, which means Not Connected. So we leave that pin unconnected.

Pin 2 is the SDA pin, which is the Serial Data pin. This transmits data bidirectionally between the master device and the slave device. Both SDA pins of the 2 sensors connect to analog pin 4 on the arduino.

Pin 3 is the power ground, so they connect to the ground terminal of the arduino.

Pin 4 is the SCLK pin, which is the Serial Clock pin. This pin clocks data into and out of the TC74 sensor. Both SCLK pins of the 2 sensors connect to pin 4 on the arduino.

Pin 5 is the V_{DD} pin, which is the positive voltage power source for the sensor. These are tied together from both sensors and connect to the +5V pin of the arduino.

And these complete the hardware connections from the arduino microcontroller to the I^2C sensors.

Code

The code so that we can connect the I^2C TC74 sensors to an arduino microcontroller is shown below.

Arduino's I^2C communication library is called the Wire library. With this library, you can easily write to and read from I^2C devices.

```
#include <Wire.h>

int address_sensor1= 72; //binary equivalent is 1001000

int address_sensor2= 73; //binary equivalent is 1001001

void setup(){

Serial.begin(9600); //this creates the Serial Monitor
Wire.begin(); //this creates a Wire object
}

void loop(){
Wire.beginTransmission(address_sensor1); //Send a request to begin communication with the device at the specified address

Wire.write(0); //Sends a bit asking for register 0, the data register of the TC74 sensor

Wire.endTransmission(); //this ends transmission of data from the arduino to the temperature sensor

//this now reads the temperature from the TC74 sensor
Wire.requestFrom(address_sensor1, 1);//this requests 1 byte from the specified address

while(Wire.available() == 0);
int celsius1= Wire.read();

int fahrenheit1= round(celsius1 * 9.0/5.0 + 32.0);

Serial.print("Temperature sensor 1:");
Serial.print(celsius1);
Serial.print("degrees celsius ");
Serial.print(fahrenheit1);
Serial.print(" degrees Fahrenheit");

delay(2000);
Wire.beginTransmission(address_sensor2); //Send a request to begin communication with the device at the specified address

Wire.write(0); //Sends a bit asking for register 0, the data register of the TC74 sensor

Wire.endTransmission(); //this ends transmission of data from the arduino to the temperature sensor

//this now reads the temperature from the TC74 sensor
Wire.requestFrom(address_sensor2, 1);//this requests 1 byte from the specified address

while(Wire.available() == 0);
int celsius2= Wire.read();

int fahrenheit2= round(celsius2 * 9.0/5.0 + 32.0);

Serial.print("Temperature sensor 2:"); Serial.print(celsius2);
Serial.print("degrees celsius ");
Serial.print(fahrenheit2);
Serial.print(" degrees Fahrenheit");

delay(2000);
}
```

First, we import the Wire library, which is the library for communicating with I^2C devices.

Next, we initialize the value of the address which we want to communicate. The TC74 sensors that run on 5V comes in 8 different IC packages. These are TC74A0 to TC74A7. Depending on which TC74 sensor you use determines the address that we will initialize to. The TC74 datasheet has all the addresses listed for all the various TC74s. Therefore, you just need to look up the address on the datasheet. Being that we are using a TC74A0, the address is 1001000; the decimal equivalent of this is 72. Therefore, we initialize the first address to 72. For the TC74A1 sensor, the address is 1001001; the decimal equivalent of this is 73. Therefore, we initialize the second address to 73. Therefore, we can now address both sensors independently.

Next, we have the setup() function. This creates the Serial monitor where we can see the reading of our temperature. And we create a Wire object, which allows for communication between the master and slave devices.

Next, we have our loop() function. We begin a transmission in which we communicate with the first sensor at the address we initially specified. This allows the master device to communicate with that slave device (at that address). Since we specify the first address, which is sensor 1 (address 72), we communicate first with that sensor. Next, we communicate with the second sensor (at address 73).

Next, we send a bit asking for register 0 of the TC74, which is the data register of the sensor. We will then read from this data register to find out the temperature. We then end the transmission of data from the master to the slave device.

We then request 1 byte from the TC74 sensor. This value will come in the value celsius. We then calculate the fahrenheit based on this celsius value. And then print out the values in both celsius and fahrenehit.

We give 2-second delay for the next sensor.

We then repeat the same process for the second sensor. We create a print statement as "Temperature Sensor 2:" so that we know it's the second sensor. We did the same for the first sensor.

With a circuit such as this, we can address each of the sensor, obtain temperatures with each of the sensors, so that we can see whether they are similar in output. If we have another device that's measuring the temperature that is highly accurate, we can see which of the TC74 sensors are closer in value to that temperature sensor. It's a way we can gauge which temperature sensor is more accurate.

And this is how multiple I^2C devices can be connected to an arduino microcontroller.

Related Resources

How to Build a Digital Potentiometer Circuit Using a MCP4131

Comments