

# ELEC 2520 Lab 4:

## I2C Communication

### Objective:

Create a program that can read the temperature of the Microchip TC74 using I2C and display it for the user to read by using UART.

Important Settings needed for your program:

- Processor Frequency 16000000UL
- Baud Rate 9600
- 1 Stop Bit
- 8 Data Bits

### References:

[1] TC74 Schematic Diagram from:

<http://www.electroschematics.com/9798/reading-temperatures-i2c-arduino/>

[2] Tiny Serial Digital Thermal Sensor:

<http://ww1.microchip.com/downloads/en/DeviceDoc/21462D.pdf>

### Wiring the Sensor

The TC74 chip is a TO-220 package with 5 pins. A description of the pin mapping of the device is seen below.

TO-220

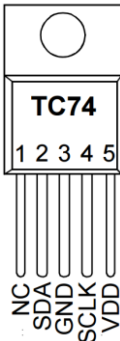
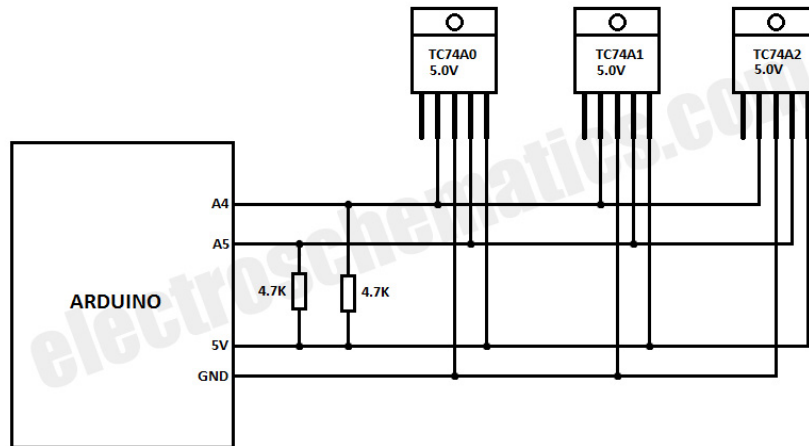


TABLE 2-1: PIN FUNCTION TABLE

Pin No. (5-Pin SOT-23)	Pin No. (5-Pin TO-220)	Symbol	Type	Description
1	1	NC	None	No Internal Connection
2	3	GND	Power	System Ground
3	5	V <sub>DD</sub>	Power	Power Supply Input
4	4	SCLK	Input	SMBus/I <sup>2</sup> C Serial Clock
5	2	SDA	Bidirectional	SMBus/I <sup>2</sup> C Serial Data

Place the device in the breadboard. Connect power and ground to the 5V line of your Atmega328p processor. Finally two resistors need to be connected from the SCLK and SDA lines to the high voltage rail. These are called pull up resistors and you want 2x 4.7 Kohm resistors for this. See below for an example wiring of this. (If you don't have 4.7 kohm resistors, you can use 10K instead)



Example Circuit Implementation Provided by ElectroSchematics.com<sup>[1]</sup>

## Lab Background - I2C

Reading and writing to this device will require knowing the TC74 conventions for a read and write cycle. These are specified below.

### Write Byte Format

S	Address	WR	ACK	Command	ACK	Data	ACK	P
	7 Bits			8 Bits		8 Bits		

Slave Address

Command Byte: selects which register you are writing to.

Data Byte: data goes into the register set by the command byte.

### Read Byte Format

S	Address	WR	ACK	Command	ACK	S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits			7 Bits			8 Bits		

Slave Address

Command Byte: selects which register you are reading from.

Slave Address: repeated due to change in data-flow direction.

Data Byte: reads from the register set by the command byte.

### Receive Byte Format

S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits		

S = START Condition

P = STOP Condition

Shaded = Slave Transmission

Data Byte: reads data from the register commanded by the last Read Byte or Write Byte transmission.

Using this structure, you will first send the address of the device and a write command to program the register you are wanting to read/write. Following that you will send the register you are interested in reading and writing. If you are writing data, you will then immediately send the value you wish to write to that register. If you are going to read data, you will send a new transmission to the device starting with a new start condition, the device address and then a read command (bit). The device will then respond with the register you have programmed it to read.

The I2C address of this device is **0x48**. This is seen in the table below and our specific version of the device is the TC74A0-5.

### SOT-23 Package Marking Codes

SOT-23 (V)	Address	Code	SOT-23 (V)	Address	Code
TC74A0-3.3VCT	1001 000	V0	TC74A0-5.0VCT	1001 000	U0
TC74A1-3.3VCT	1001 001	V1	TC74A1-5.0VCT	1001 001	U1
TC74A2-3.3VCT	1001 010	V2	TC74A2-5.0VCT	1001 010	U2
TC74A3-3.3VCT	1001 011	V3	TC74A3-5.0VCT	1001 011	U3
TC74A4-3.3VCT	1001 100	V4	TC74A4-5.0VCT	1001 100	U4
TC74A5-3.3VCT	1001 101*	V5	TC74A5-5.0VCT	1001 101*	U5
TC74A6-3.3VCT	1001 110	V6	TC74A6-5.0VCT	1001 110	U6
TC74A7-3.3VCT	1001 111	V7	TC74A7-5.0VCT	1001 111	U7

**Note:** \* Default Address

The I2C max frequency this device will use is 100 KHz and cannot be slower than 10 KHz.

SMBus/I <sup>2</sup> C Clock Frequency	f <sub>SMB</sub>	10	—	100	kHz
--	------------------	----	---	-----	-----

There are only two 8-bit registers, a configuration register and a temperature register. See below.

**TABLE 4-1: COMMAND BYTE DESCRIPTION (SMBUS/I<sup>2</sup>C READ\_BYTE AND WRITE\_BYTE)**

Command	Code	Function
RTR	00h	Read Temperature (TEMP)
RWCR	01h	Read/Write Configuration (CONFIG)

**TABLE 4-2: CONFIGURATION REGISTER (CONFIG); 8 BITS, READ/ WRITE)**

Bit	POR	Function	Type	Operation
D[7]	0	STANDBY Switch	Read/ Write	1 = standby, 0 = normal
D[6]	0	Data Ready *	Read Only	1 = ready 0 = not ready
D[5]-D[0]	0	Reserved - Always returns zero when read	N/A	N/A

**Note 1:** \*DATA\_RDY bit RESET at power-up and SHDN enable.

**TABLE 4-5: TC74 REGISTER SET SUMMARY**

Name	Description	POR State	Read	Write
TEMP	Internal Sensor Temperature (2's Complement)	0000 0000b <sup>(1)</sup>	√	N/A
CONFIG	CONFIG Register	0000 0000b	√	√

**Note 1:** The TEMP register will be immediately updated by the A/D converter after the DATA\_RDY Bit goes high.

**Requirements:**

The sample code template will be provided for you.

In this lab, you need to write "C Only" program to read temperature.

All you need to do is to implement these functions below:

`i2c_init();`

`i2c_start();`

`i2c_write();`

`i2c_read();`

`i2c_stop();`

The requirements for each function, please check the notes in code template.

## Code template for Lab 4

```
void setup() {  
    Serial.begin(9600);  
}  
  
void loop() {  
    int Temperature = read_temp(device_address); // You need to define "device_address"  
    Serial.print("Temperature: ");  
    Serial.print(Temperature);  
    Serial.print("C");  
    Serial.print("\n");  
    delay(1000);  
}  
  
/* Read from Temperature Sensor */  
int read_temp(int address) {  
    /* Implementement this function by using functions below:  
    i2c_init();  
    i2c_start();  
    i2c_write();  
    i2c_read();  
    i2c_stop();  
    And return the value of temperture  
    */  
}
```

```
/******
```

Initialization of the I2C bus interface.

```
*****/
```

```
void i2c_init(void)
```

```
{
```

```
    /*
```

```
        1. no prescaler
```

```
        2. SCL clock: 100 kHz clock
```

```
    */
```

```
}
```

```
/******
```

Issues a start condition and sends address and transfer direction.

return 0 = device accessible, 1= failed to access device

```
*****/
```

```
unsigned char i2c_start(unsigned char address)
```

```
{
```

```
    /*
```

```
        1. send START condition
```

```
        2. wait until transmission completed
```

```
        3. check value of TWI Status Register. Mask prescaler bits. If status different from "start" and  
        "repeated start", return 1
```

```
        4. send device address, configure TWCR
```

```
        5. wait until transmission completed and ACK/NACK has been received
```

```
        6. check value of TWI Status Register. Mask prescaler bits. If status different from "SLA+W  
        transmitted, ACK received " and " SLA+R transmitted, ACK received ", return 1 otherwise return 0
```

```
    */
```

```
}
```

```
/******
```

Terminates the data transfer and releases the I2C bus

```
*****/
```

```
void i2c_stop(void)
```

```
{
```

```
    /*
```

```
        1. send stop condition
```

```
        2. wait until stop condition is executed and bus released
```

```
    */
```

```
}
```

```
/******
```

Send one byte to I2C device

Input: byte to be transfered

Return: 0 write successful

1 write failed

```
*****/
```

```
unsigned char i2c_write( unsigned char data )
```

```
{
```

```
    /*
```

```
        1. send data to the previously addressed device
```

```
        2. wait until transmission completed
```

```
        3. check value of TWI Status Register. Mask prescaler bits. If status different from "data  
transmitted, ACK received", return 1 otherwise return 0
```

```
    */
```

```
}
```

```
/******
```

Read one byte from the I2C device, read is followed by a stop condition

Return: byte read from I2C device

```
*****/
```

```
int i2c_read(void)
```

```
{
```

```
/*
```

1. configure TWCR
2. wait until transmission completed
3. return temperature data

```
*/
```

```
}
```