**I2C Master Documentation**

By Aaron Nightingale (night030@umn.edu)

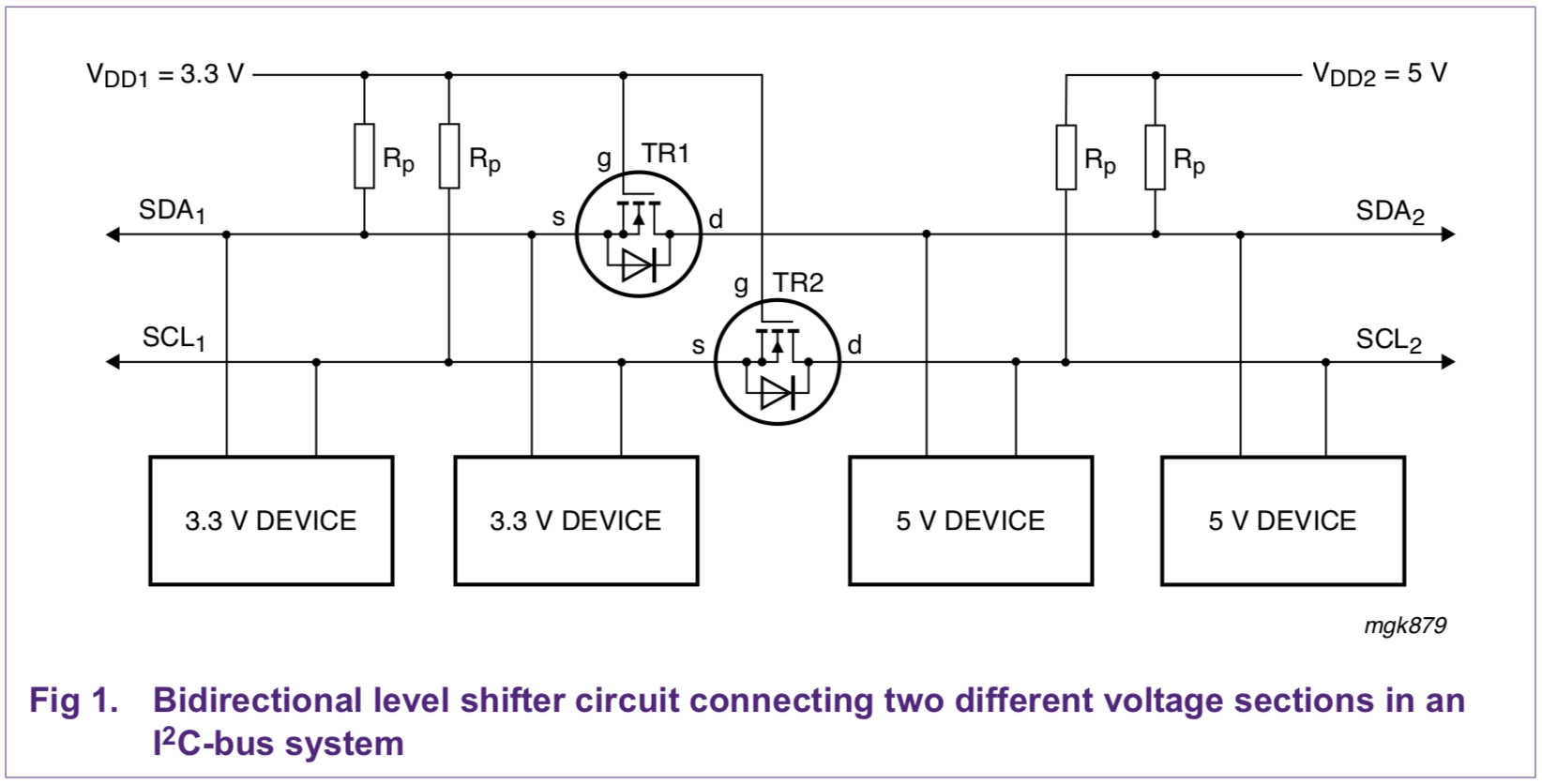
**Summary:**

The program titled “i2c-master.ino” is a bare-bones program that enables an Arduino to act as an I2C master device, and request readings from the turbidity sensor (slave device). This program is much easier to understand than the “Turbidity.h” and “TurbidityHelper.h” classes which are based on the modular sensors library. This program was frequently used when testing the turbidity sensor.

**Using This Program**

This program was used to test the turbidity sensor and do simple demonstrations. This program is intended to be programmed to an Arduino, which acts like a master device. Using this program, the Arduino can tell the turbidity sensor to take a new reading and display the results of that measurement on the serial monitor.

1. To use this program, upload it to a 3.3v Arduino device. (Note that 5v Arduinos could be used as well, but a level shifter is required. Figure 1 shows the type of level shifter that was used in our preliminary prototyping).
   * NXP has great documentation on I2C level shifters. General purpose N-Channel MOSFETS like the 2N7000 can be used. See the following website for more information:
   * <https://www.nxp.com/docs/en/application-note/AN10441.pdf>
2. Next, unplug the Arduino from the computer, and connect it to the turbidity sensor. The connections are shown in figure 2 below.
3. Verify the connections, and plug the Arduino back into the computer.
4. Select the serial port corresponding to the master Arduino. Then open the serial monitor and set the baud rate to 9600 bit/s.
5. You should see “Sensor #8 found.” displayed on the serial monitor. If not, double check the wiring, and make sure that the turbidity sensor is programmed with the “turbidity\_slave.ino” program.
6. Various commands can now be sent to the turbidity sensor. For example, type “UPDATE” into the text box of the serial monitor and press enter. This will tell the turbidity sensor to take a new turbidity measurement.
7. The measurement process takes about 8 minutes. Once it is complete, the measurement results are displayed on the serial monitor. This includes the calculated water turbidity in NTU.
8. Now the turbidity sensor is up and running, and communicating with an I2C “master” Arduino!



*Figure 1 - I2C level shifter circuit*

|  |  |  |
| --- | --- | --- |
| **Master Arduino Connection** | **Turbidity Sensor Connection** | **Purpose** |
| +3.3v | Red | Sensor power |
| GND | Black | Sensor ground |
| A4 (SDA) | Blue | I2C data line |
| A5 (SCL) | Yellow | I2C clock line |
| N / A | Green | Sensor busy pin (Optional) |

*Figure 2 - Connections between the I2C master Arduino and the turbidity sensor*

**“i2c-master.ino” Reference**

**Constants**

* #define SLAVE\_ADDR 8
  + This constant defines the address of the turbidity sensor. If the address of the turbidity sensor is changed in “turbidity\_slave.ino”, then this constant must be changed to properly communicate with the turbidity sensor.
* #define DATA\_LEN 6
  + This constant defines the size of the response from the turbidity sensor. The turbidity sensor sends 6 float-type variables, causing the data length constant to be 6.
* #define COMMAND\_LEN 7
  + The data logger can send various commands to the turbidity sensor. The commands are sent as a character array, and the length of said array is 7 bytes. Note that the last byte must be a zero.
* #define RES\_DARK1 0
  + The turbidity sensor returns an array of 6 float-type variables to the datalogger. The float at position 0 of this array is the number of dark counts from sensor #1. The units for this are Hertz.
* #define RES\_TRAN1 2
  + The turbidity sensor returns an array of 6 float-type variables to the datalogger. The float at position 2 of this array is the number of transmission counts from LED #1. The units for this are Hertz.
* #define RES\_SCAT1 3
  + The turbidity sensor returns an array of 6 float-type variables to the datalogger. The float at position 3 of this array is the number of side scatter counts from LED #1. The units for this are Hertz.
* #define RES\_DARK2 1
  + The turbidity sensor returns an array of 6 float-type variables to the datalogger. The float at position 1 of this array is the number of dark counts from sensor #2. The units for this are Hertz.
* #define RES\_TRAN2 4
  + The turbidity sensor returns an array of 6 float-type variables to the datalogger. The float at position 4 of this array is the number of transmission counts from LED #2. The units for this are Hertz.
* #define RES\_SCAT2 5
  + The turbidity sensor returns an array of 6 float-type variables to the datalogger. The float at position 5 of this array is the number of side scatter counts from LED #2. The units for this are Hertz.

**Global Variables**

* char command[COMMAND\_LEN];
  + This array contains the command that will be sent to the turbidity sensor. This array is a character array.
* float response[DATA\_LEN];
  + This array contains the response from the turbidity sensor. This array consists of 6 float-type variables. Using these 6 variables, the turbidity of the water can be calculated. The variable at each position of the array is given by the “RES\_XXX” constants. (Note that RES stands for RESponse array).

**Functions**

* void setup()
  + This is the setup function of the I2C master program. In this function, the I2C bus and the serial port are initialized. After that, the function checks to see if there is a turbidity sensor connected to the I2C bus using the “sensorPresent()” function.
* void loop()
  + This is the main loop of the program. The function begins by reading the current values of the sensor using the “sensorRead()” function. Next, this data is displayed on the serial monitor using the “dumpFormattedData()” function. Afterwards, the program waits until the user has entered a command into the serial monitor. If a valid command is entered, then this command is sent to the turbidity sensor using I2C. In most cases, the “UPDATE” command is sent to the turbidity sensor, causing it to take a new turbidity reading. This measurement process takes about 8 minutes, so an 8 minute delay is added to this function. Then, the function loops back to the start and repeats.
* void getSerialCommand()
  + When this program is running, and the master Arduino device is connected to a computer, then the user may send commands to the turbidity sensor using the serial monitor. This function reads data from the serial monitor and puts it into the “command[]” array. For example, if a user types “UPDATE” into the serial monitor, then this function parses that string into a character array {‘U’,’P’,’D’,’A’,’T’,’E’,’0’}, which can be sent to the turbidity sensor using I2C.
* bool sensorPresent(int addr)
  + This function checks if there is a sensor present at the desired address. This function is used before any data transactions with the turbidity sensor. TRUE is returned if an I2C sensor was found at the specified address.
* void sensorRead()
  + This function requests data from the turbidity sensor using I2C. The turbidity sensor sends 6 float-type variables one byte at a time. This function reads the data from the turbidity sensor, byte by byte, and stores this data in the “response[]” array.
* void sensorWrite()
  + This function sends the contents of the “command[]” array to the slave turbidity sensor. By changing the contents of the “command[]” array, different commands can be sent to the turbidity sensor. For example, if the array consists of “{‘U’,’P’,’D’,’A’,’T’,’E’,0}, then the update command is issued and the turbidity sensor begins taking a new measurement.
* void debug(int i)
  + This is a basic function that is used for debugging. When this function is called, “DEBUG: #” is written to the serial monitor, where the value if the argument “i” is represented by the # sign.
* void dumpSensorData()
  + This function dumps the contents of the “response[]” array to the serial monitor. Each float is displayed on a new line. This function was primarily used for debugging purposes.
* float calculate\_turbidity()
  + This function calculates the turbidity using the data in the “response[]” array. The turbidity is returned as a float-type variable. This function can be changed to more accurately calibrate the turbidity sensor.
* void dumpFormattedData()
  + This function dumps the contents of the “response[]” array to the serial monitor. However, unlike the “dumpSensorData()” function, this function displays a short description of what each number is. This function was used for debugging and simple demonstrations of the turbidity sensor.