**Turbidity Slave Documentation**

By Aaron Nightingale (night030@umn.edu)

**Summary:**

This program contains the code for a turbidity sensor. The turbidity sensor is an I2C slave device, and can communicate with another Arduino or data logger using I2C. This turbidity slave program contains functions that measure the transmitted light, scattered light, and the dark counts from each light-to-frequency sensor. In addition, this program has functions that average these readings and sends them to a master I2C device.

**Using This Program**

1. Open the “turbidity\_slave.ino” program in the Arduino IDE.
2. Next, upload this program to a 3.3v compatible ATMega328 chip.
   * Notice: You may need to burn the bootloader to the chip first. See the following link for details on how to do that:
   * <https://www.arduino.cc/en/Tutorial/ArduinoToBreadboard>
3. Once the program has been uploaded to the chip, connect the chip to the turbidity sensor circuitry. (Schematics of this circuitry can be found in the following Google Drive folder: “Turbidity Sensor Senior Design 2018” => “Final Documentation”)
4. Double check the circuit connections.
5. Now the turbidity sensor is ready to be used! Connect the turbidity sensor to a 3.3v Arduino running the “i2c-master.ino” program to view the turbidity measurements.
   * For reference, the connections are shown in figure 1 below:
   * See “I2C Master Documentation” PDF for more details on how to set this up.

|  |  |  |
| --- | --- | --- |
| **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*

**“turbidity\_slave.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 SENSOR1 2
  + This parameter defines the pin that is connected to sensor #1. (Light to frequency sensor #1).
* #define SENSOR2 3
  + This parameter defines the pin that is connected to sensor #2. (Light to frequency sensor #2).
* #define L1\_HIGH 5
  + This parameter defines the pin that controls when LED #1 is turned on.
* #define L1\_OFF 6
  + This constant defines the pin that controls when LED #1 is turned off.
* #define L2\_HIGH 7
  + This parameter defines the pin that controls when LED #2 is turned on.
* #define L2\_OFF 8
  + This constant defines the pin that controls when LED #2 is turned off.
* #define BUSY 13
  + This constant defines the location of the busy pin. This pin is held high when the turbidity sensor is busy taking a new measurement. The turbidity sensor should not be interrupted over I2C while a measurement is in progress.
* #define TYPES 6
  + This parameter defines the size of the array which stores the measurement results. The turbidity sensor measures 6 different parameters, which are stored in the “result[]” array.
* #define REPEATS 5
  + This parameter defines the number of measurements that should be averaged for each new reading.
* const char MODE01[] = {'M','O','D','E','0','1',0};
  + This constant defines the “MODE01” I2C command. This command is not implemented in the turbidity sensor, but can be used as an example of how to implement custom I2C commands into the program.
* const char MODE02[] = {'M','O','D','E','0','2',0};
  + This constant defines the “MODE02” I2C command. This command is not implemented in the turbidity sensor, but can be used as an example of how to implement custom I2C commands into the program.
* const char MODE03[] = {'M','O','D','E','0','3',0};
  + This constant defines the “MODE03” I2C command. This command is not implemented in the turbidity sensor, but can be used as an example of how to implement custom I2C commands into the program.
* const char UPDATE[] = {'U','P','D','A','T','E',0};
  + This constant defines the “UPDATE” command. This command tells the turbidity sensor to take a new turbidity reading.

**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.
* volatile uint8\_t go;
  + This variable stores the number of seconds remaining in the frequency measurement process.
* volatile uint32\_t count;
  + This variable stores the number of rising edges that were counted on the interrupt pin.
* float measurements[TYPES][REPEATS]
  + This array holds all of the measurements from the light to frequency sensors. The “TYPES” dimension corresponds to the different measurements, while the “REPEATS” dimension corresponds to separate sets of measurements. The “REPEATS” dimension is usually 5, meaning that 5 measurements are averaged. The average of the measurements in this array is stored in the “result[]” array, which is then sent to the data logger.
* float result[TYPES];
  + 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 contents of this array are sent to the I2C master device (data logger) over I2C in a byte by byte process.
* bool needsUpdate;
  + This variable is TRUE whenever the sensor needs to take a new measurement. Long functions cannot be used in the I2C interrupt routines, which is why this variable is used.

**Functions**

* void setup()
  + This function is run before the main loop. It initializes the I2C interface and sets up the timers used for measurements. The input and output pins are set in this function as well.
* ISR(TIMER1\_COMPA\_vect)
  + This is an interrupt service routine for timer #1 within the Arduino microcontroller. This routine is called whenever the value in the timer register is equal to the value of some compare register. This function is used to measure the counts on the light to frequency sensors for some duration of seconds.
* void freq\_counter()
  + This function is also an interrupt routine, and is called whenever there is a rising edge on some pin. This function simply increments the “count” variable with each rising edge. This function can be used on different pins of the Arduino microcontroller using the “attachInterrupt()” function.
* void LED1\_OFF()
  + This function sets the “L1\_OFF” pin high in order to turn LED #1 off.
* void LED1\_LOW()
  + This function sets LED #1 to the low-intensity setting. This is done by setting the “L1\_OFF” pin and the “L1\_HIGH” pin low.
* void LED1\_HIGH()
  + This function sets LED #1 to the high-intensity setting. This is done by setting the “L1\_HIGH” pin high and setting the “L1\_OFF” pin low.
* void LED2\_OFF()
  + This function sets the “L2\_OFF” pin high in order to turn LED #2 off.
* void LED2\_LOW()
  + This function sets LED #2 to the low-intensity setting. This is done by setting the “L2\_OFF” pin and the “L2\_HIGH” pin low.
* void LED2\_HIGH()
  + This function sets LED #2 to the high-intensity setting. This is done by setting the “L2\_HIGH” pin high and setting the “L2\_OFF” pin low.
* uint32\_t SENSE1(uint8\_t seconds)
  + This function reads the pulses from light-to-frequency sensor #1. The pulses are counted for a time duration specified in the argument of the function. The number of pulses counted is returned as an unsigned 32 bit integer.
* uint32\_t SENSE2(uint8\_t seconds)
  + This function reads the pulses from light-to-frequency sensor #2. The pulses are counted for a time duration specified in the argument of the function. The number of pulses counted is returned as an unsigned 32 bit integer.
* float READ\_SENSOR1()
  + This function uses the “SENSE1()” function to calculate the frequency of the pulses from sensor #1. The frequency of the pulses from light-to-frequency sensor #1 is returned as a float-type variable.
* float READ\_SENSOR2()
  + This function uses the “SENSE2()” function to calculate the frequency of the pulses from sensor #2. The frequency of the pulses from light-to-frequency sensor #2 is returned as a float-type variable.
* float SENSOR1\_DARK()
  + This function turns each LED off, and measures the frequency from light-to-frequency sensor #1. The frequency is returned as a float-type variable.
* float SENSOR2\_DARK()
  + This function turns each LED off, and measures the frequency from light-to-frequency sensor #2. The frequency is returned as a float-type variable.
* float LED1\_TRANSMISSION()
  + This function measures the transmitted light from LED #1. This is done by setting LED #1 to be in the low intensity mode. This is accomplished using the “LED1\_LOW()” function. Afterwards, the frequency is measured using the “READ\_SENSOR1()” function. The intensity of the transmitted light is proportional to the frequency of the light-to-frequency sensor, which is returned as a float-type variable in hertz.
  + Note: Increasing the brightness of the LED can cause the light-to-frequency sensor to produce more pulses than the microcontroller can handle. For this reason, the LED has a low intensity mode.
* float LED1\_SCATTER()
  + This function measures the 90˚ side-scattered light from LED #1. To do this, LED #1 is set to operate in the high-intensity mode. This is done by calling the “LED1\_HIGH()” function. Next, the frequency of the light-to-frequency sensor is measured using the “READ\_SENSOR2()” command. The intensity of the side-scattered light is proportional to the frequency of the light-to-frequency sensor, which is returned as a float-type variable in hertz.
* float LED2\_TRANSMISSION()
  + This function measures the transmitted light from LED #2. This is done by setting LED #2 to be in the low intensity mode. This is accomplished using the “LED1\_LOW()” function. Afterwards, the frequency is measured using the “READ\_SENSOR2()” function. The intensity of the transmitted light is proportional to the frequency of the light-to-frequency sensor, which is returned as a float-type variable in hertz.
  + Note: Increasing the brightness of the LED can cause the light-to-frequency sensor to produce more pulses than the microcontroller can handle. For this reason, the LED has a low intensity mode.
* float LED2\_SCATTER()
  + This function measures the 90˚ side-scattered light from LED #2. To do this, LED #2 is set to operate in the high-intensity mode. This is done by calling the “LED2\_HIGH()” function. Next, the frequency of the light-to-frequency sensor is measured using the “READ\_SENSOR1()” command. The intensity of the side-scattered light is proportional to the frequency of the light-to-frequency sensor, which is returned as a float-type variable in hertz.
* void measure()
  + This function measures the dark counts, transmitted light, and the side scattered light from each light-to-frequency sensor. This is a total of 6 frequency measurements, which are saved to the “measurements[][]” array. These measurements are repeated several times, as specified by the “REPEATS” constant. In our setup, 5 sets of measurements are taken each time that this function is called.
* void average()
  + This function calculates the average frequency of each of the 6 different frequency measurements (dark counts, transmission, side scatter, etc.). The average values are saved to the “result[]” array, and are later sent to the data logger over I2C.
* void requestEvent()
  + This function is an interrupt service routine for the I2C interface. This function is called whenever the master device requests data from the turbidity sensor. This function sends the contents of the “result[]” array, byte by byte, to the master device using the I2C interface.
* void receiveEvent(int howMany)
  + This function is an interrupt service routine for the I2C interface. This function is called whenever the master sends data to the turbidity sensor. The data logger can send various commands to the turbidity sensor. This function interprets and executes the received command.
  + The only command that is implemented right now is the “UPDATE” command. When the turbidity sensor receives the character array corresponding to this command over I2C, the “needsUpdate” variable is set to be true. This causes the turbidity sensor to take a new set of measurements.
* void loop()
  + This is the main loop of the turbidity sensor program. In this loop, the turbidity sensor takes a new set of measurements if the “needsUpdate” variable is true. After the new measurements are taken and averaged, the “needsUpdate” variable is set to false. This loop waits 1ms and then repeats itself.