TECHNICAL REPORT

Practicum 4

Embedded systems

Authors: Veselin Atanasov, Wu Feng

Teacher: Hans van Neumen

Date:23.05.2024

ABSTRACT

This report summarizes the use of I2C communication and utilizing it using the Wire.h library for Arduino microcontrollers. The following report concludes that multiple devices can be put on the same I2C bus without interference. The following report also features the process of analyzing datasheets to develop a device driver.

CONTENTS

ASSIGNMENT A

INTRODUCTION

This assignment's goal is to create a robust device driver for the BME 280 digital sensor for temperature, pressure and humidity. The goal is to turn all the raw readings of the sensor into actual data, by carefully analyzing the datasheet of the device and manipulating registers. This exercise also showcases the use of I2C and how it works using the "Wire.h" library.

PROCEDURE

Assignment A starts with creating low level device driver for the BME280 sensor. The BME280 sensor displays humidity, pressure, and temperature. The sensor provides fast response time and high accuracy when displaying its measurements. In this case the Arduino is the master and the sensor is the slave. To create the driver the following functions are necessary:

1. uint8_t BME280_GetID();
2. void BME280_Reset();
3. uint8_t BME280_ReadCtrlHum();
4. void BME280_CtrlHum(uint8_t bitpattern);
5. uint8_t BME280_ReadCtrlMeas();
6. void BME280_CtrlMeas(uint8_t bitpattern);
7. long BME280_ReadTemperature();
8. int BME280_ReadHumidity();
9. long BME280_ReadPressure();

In this case two more functions were implemented to avoid repetition of code. Those functions are Read16FromAddress and Read8FromAddress.

Every function starts with Wire.beginTransmission(0x76) and what it does is starts communication with the sensor, and ends with Wire.endTransmission(). If end transmission is mentioned, because afterwards there is something that must be implemented. It is mentioned here, for that reason it does not get repetitive. This is an exception for BME280_ReadTemperature(), ReadHumidity(), and ReadPressure().

UINT8_T BME280_GETID()

The purpose of this function is to get the ID of the sensor. Before beginning, it is wise to get all the necessary address. The address of the slave is 0x76, given from the datasheet, and the register address to get the ID is 0xD0.

Step 1: Wire.write(0xD0) and what it does is it gives access to the data in register D0, which in this case is to get the ID. Step 2: End the transmission with the slave. Step 3: Wire.requestFrom(0x76, 1) and what it does is request bytes from the slave. After completing all the steps the last step would be Wire.Read() which reads the bytes that are requested from the master, from the specified register address in the slave. In the end the ID that gets read is 0x60.

VOID BME280_RESET();

The purpose of this function is to perform a soft reset. Same as the first function the necessary address will first be found. The address of the slave is 0x76, the register to reset is 0xE0, and to reset the address 0xB6 must be used. The goal of this function is to reset all the registers in the bme responsible for calibration and setting modes and oversampling. After this function is completed, the right modes can be set aswell as the proper oversampling settings by directly writing to the registers responsible for their respective settings.

Step 1: Wire.write(0xE0) and what it does is talk to the register E0, which in this case is the register responsible for the reset function of the sensor. Step 2: Wire.write(0xB6) and what it does is perform a reset, if other than B6 is written to the register it would have no effect.

UINT16_T READ16FROMADDRESS(UINT16_T ADDRESS)

This function reads two 8 bit values from a specified address and combines the into an unsigned 16-bit value. This driver requires multiple readings, so for optimization this function was implemented as well as the Read8FromAddress.

BME280_ADDRESS is the I2C address of the sensor. Step 1: Wire.write(address) and what it does is specify the register address we want to read from. The address parameter tells the sensor which register to read. Step 2: End the transmission with the slave. This step finalizes the process of sending the register address to the sensor. Step 3: Wire.requestFrom(BME280_ADDRESS, 2) and what it does is request 2 bytes of data from the sensor starting from the specified register address. The sensor will send the data from the register. Step 4: uint8_t lsb = Wire.read() and what it does is read the least significant byte (LSB) of the 16-bit data from the sensor. Step 5: uint8_t msb = Wire.read() and what it does is read the most significant byte (MSB) of the 16-bit data from the sensor. Step 7: uint16_t reading = (uint16_t)(msb << 8) | lsb and what it does is combine the MSB and LSB to form a 16-bit value. The MSB is shifted left by 8 bits and then bitwise OR'd with the LSB to form the final 16-bit reading.

UINT8_T READ8FROMADDRESS(UINT16_T ADDRESS)

It gets and address and reads from this specific address in the bme, then it returns the reading.

UINT8_T BME280_READCTRLHUM();

This function accesses the address of the ctrl_hum register and returns the reading to the user in 8-bit format. The user can take this reading and do logical operation to achieve a bit mask that represents the settings the user wants. To take effect, this mask has to be written to the ctrl_hum register using the BME280_CtrlHum(uint8_t bitpattern).

VOID BME280_CTRLHUM(UINT8_T BITPATTERN);

The purpose of this function is to control the humidity. The address of the slave is 0x76, the address to control the humidity is 0xF2.

This function overwrites the data in the register for controlling humidity to the desired bitpattern of the user.

UINT8_T BME280_READCTRLMEAS();

This function works in exactly the same manner as BME280_ReadCtrlHum() the only difference being the address from which data is read.

VOID BME280_CTRLMEAS(UINT8_T BITPATTERN);

The purpose of this function is to control the measurements depending on the desired setting by the user. The address of the slave is 0x76, the address to control the measurements is 0xF4. Bit 7, 6, 5 is to control the oversampling of temperature data, bit 4, 3, 2 is to control the oversampling of pressure data, bit 1, 0 is to control the sensor mode. For an oversampling of 1 and set it to normal mode, the mask that is needed to achieve the desired setting is 0x27.

This function works in exactly the same manner as BME_280_CtrlHum() the only difference being again the address.

LONG BME280_READTEMPERATURE();

The purpose of this function is to read out the temperature data. To measure the temperature, a reading from the sensor is required. To store the temperature readings, three registers are used in the BME-0xFA to 0xFC. This function is of great importance because it calculates the t_fine value which is also involved in the calculations of the humidity and pressure.

Step: 1:All those values are stored in 8 bit format. To create a single reading, those three values must be combined into a single unsigned 32-bit variable, by sorting them from least to most significant. Step 2: To calculate the temperature, some trimming parameters are also necessary. They are stored in the registers of the bme and cannot be changed by the user. They are stored in 8-bit format and have to be combined to a signed 32-bit variable. The trimming parameters for calculating temperature are: dig_T1, dig_T2 and dig_T3. Step 3: After all the necessary data is collected, the temperature is calculated using the formulas provided in the datasheet. Step 4: The output of the calculation is a double that when divided by 100.0, to match the output format, gives the accurate temperature reading.

LONG BME280_READPRESSURE();

The procedure is almost the same. The difference is that the registers that store the actual reading and the addresses of the trimming parameters are different as well as the calculation formula. The reading addresses are 0xF7 to 0xF9 and the addresses of the trimming parameters are 0x90...0x9E. They are named in the same manner as the temperature ones – dig_P1...dig_P9. After calculation is done, the result is converted to double and is divided by 25600.0(to match output format) to get the actual reading in hPa.

LONG BME280_READHUMIDITY();

Again, same methods. The key differences are the addresses, the formula and the formulation of the dig_H values, because of some different positions of the bits. Also the humidity reading are stored in just two registers compared to three that temperature and pressure need.

ASIGNMENT B

INTRODUCTION

This assignment features analyzing example codes and creating applications to have two arduinos communicating in an I2C master- slave configuration.

PROCEDURE

To create the desired program it is important to have basic understanding of registers, addresses, I2C and the Wire.h library. To do this there are example codes for the Wire.h library, that can give a good idea on how to program using this library, in the Arduino IDE.

First the analysis of the example codes is explained and then the code for the exercise.

The first example code is about the master sending data to the slave. Only important thing happening from the master side is that they send bytes over to the slave with Wire.write(). Onto the slave side, as the master sends bytes it triggers the Wire.onReceive(receiveEvent) and depending what handler is inside the function, it will execute. Therefore in this case it will be receiveEvent(), what happens in the function is that it first checks if the slave receives any data from master, once it does it enters a while loop, where then it reads the data with Wire.read() (first the master sends the characters which is usually more than one byte) and then the slave prints all the byte as characters. - after printing the characters, as long as the slave receives data the loop continues.

The second example is about the master receiving data from the slave. The master first execute Wire.requestFrom(8, 6) to request its desired bytes. The 8 is the address of the slave and 6 is the amount of bytes the master is requesting. On the slave side, when the request from master is received then the Wire.onRequest(requestEvent) gets triggered and executes requestEvent() function. The requestEvent() function sends bytes to the master with Wire.write().

After studying the examples code, it is now onto the last part of the assignment. Make a program that allows communication in both directions. The requirements is (1) that master sends repeatedly and increment bytes to the slave. (2) Once the slave receives value that is over 100, it sends back the number 2, otherwise 4. And (3) the slave is only allowed to receive data and return data when the master request, use only Wire.onReceive() and Wire.onRequest().

First in master initialize x to 0 then use Wire.write(x) so that the master sends the data to the slave then immediately increment x. Once the data gets send in slave it will trigger the Wire.onReceive(receiveEvent) and the receiveEvent() function will print out the x value to the serial. Then on the master side it will send a request with Wire.requestFrom(0x42, 6) to the slave, the slave then sends 6 bytes to the master. The slave will keep sending the number 4 until x is over 100.

ASSIGNMENT C

INTRODUCTION

The goal of Assignment C is to exercise the concepts of I2C and the Wire.h library, by creating a simple program that shows how the basics work. This program features the slave sending two values and the slave returning the min and the max value.

PROCEDURE

The first step is to define all the addresses that are required to complete the asssignment(the address of the slave device and the addresses of the registers). In this case the values are 4 and 12. To send those values, the master follows the regular writing procedure for I2C communication. First the address of the slave device is accessed, then the address of the target register and finally the value is written to the register. After the values are sent from the master, he requests two bytes from the slave for the max and the min values and prints the to the serial monitor.

The first step of creating the slave code is the same- defining all the necessary addresses. Then two functions have to be created-one for the onReceive event and on for the onRequest event. In the onReceive event, the slave arduino expects the address of the register, after it receives it, the slaves saves the data to the targeted register by the master(If the address sent by the master matches the register address of the INA register, the value will be saved in the InA variable and if the address matches the INB register's address, the value is saved in the inB variable). In the onRequest event, the slave compares the two variables and sends the results to the master accordingly.

ASSIGNMENT D

INTRODUCTION

The purpose of assignment D is to show that application A and C can work at the same time without any disturbance.

PROCEDURE

There will be two slaves present, the BME280 sensor and an Arduino. First was to integrate the master code from assignment C with assignment A code. The slave Arduino can keep the same code (from assignment C) and it only needs to be powered to work. Both will work according to their functions. As the master reads the humidity, temperature, and pressure, it also reads what value the Arduino (slave) sends.

CONCLUSION

The experiments featured in this practicum summarize the advantages of I2C communication. I2C allows multiple devices on a single bus that can communicate with eachother using the respective device and register addresses. This advantage can be used to design and implement complicated system where no user input is required and processes are automized. This reports also features the importance of analyzing datasheets to create software for different devices. In this case, by analysis of the data sheet of the BME280 sensor and utilization of the Wire.h library, a robust and effective device driver was created that can give the user accurate data like temperature, pressure and humidity that can be controlled by user defined settings.

REFFERENCES

BME280 datasheet – Bosch, February 2024

Wire library description – Arduino, December 2023

Writing technical reports for practicums - Dr. Gerald H. Hinderik, January 2021

TwoWireInterface – Canvas

BME280 – Canvas