WRITING AN I2C DRIVER

**FOR**

**A DC MOTOR CONTROLLER BOARD**

ECE 372 Winter 2020

Copyright

February 22, 2020

Douglas V. Hall

Portland, OR

**INTRODUCTION:**

I2C is a 2-wire serial interface that is used with many currently available devices and modules such as sensor modules, display controllers, motor controllers, and many other types of peripheral controllers. For this reason, most current processors have one or more built-in I2C controllers, so you don’t have to generate the I2C SCL and SDA signals by “bit-banging” GPIO pins. The TI Sitara AM3358 processor you will be using has 3 built-in I2C controllers I2C(0-2). You will use one of these I2C controllers, I2C1, to communicate with an external board that has the interface circuitry for two standard DC motors or a DC stepper motor. Your overall task will be to develop a program that will step the motor some specified number of steps forward and backward under program control. You will develop the overall program in two major parts.

1. In the first part, you will learn how to program an I2C controller on the BeagleBone Black board to correctly generate the desired clock frequency signal and the required data signals on the I2C bus. A section below gives you some hints on how to develop the program for this part of the project. To determine if these signals are being generated correctly, you will use a logic analyzer or oscilloscope to look at them on the I2C bus. Once you have verified that the signals are being generated and correct, you demonstrate these to TA for a signoff on the source code print out for this part. Then you go on to part 2.
2. In Part 2 you develop the commands you send to the PCA9685 motor controller IC on the motor board to initialize it and then have it step the motor the desired number of steps clockwise and counter-clockwise. A later section will give some hints for this part of the project.

All Computer Engineering students and Electrical Engineering students who are reasonably comfortable with basic C programming will implement the majority, if not all, of this program in the C language with the help of the C programming handouts given in class. I really recommend that you implement in C since this is a very good skill to have for many EE jobs. EEs also have the choice of using Code Composer Studio and implementing the program in ARM assembly language as you have done with previous projects. This is very doable. Students have successfully implemented a wide variety of I2C Projects like this in the past with Assembly language. A key point is that all the initial research about I2C, the I2C controller, handshaking with I2C, and how you control the motor are the same for both implementations so most of the work on initializations and algorithms is the same for both approaches.

**NOTE: You MUST do all work by yourself with no help from anyone except the Instructor and the TA. Please do NOT contact companies for assistance. You are not a paying customer and therefore not entitled to ask them to solve your problems.**

**NOTE: YOU MUST KEEP AN AS-YOU-GO LOG OF THE ACTUAL STEPS YOU TOOK, YOUR DATA FINDINGS AT EACH STEP, PROBLEMS ENCOUNTERED, HOW YOU SOLVED THESE PROBLEMS, AND HOW YOU TESTED YOUR PROGRAM.**

# DELIVERABLES

Your documentation for the project should include:

1. A detailed design log that clearly shows all the development steps you took and the results at each step.
2. Clearly written High-Level algorithms and Low-Level algorithms for the Initialization and communication sections for each version.
3. Fully documented source file for the final programs, signed by the TA to verify that your program works and meets specifications. At signoff you will be expected to be able to accurately discuss details of your implementation.
4. **A signed statement that you developed and wrote this program by yourself with no help from anyone except the instructor and/or the T.A. and that you did not give any help to anyone else. (Any evidence of joint work will result in project grades of zeros for all parties involved.)**

**GRADING**

**The Grading Weights for the sections of this project will be the same as for Design Project #1and shown at the end of this document . However, the overall grading will be a little different.**

1. Successful communication with motor controller and controlling the motor using polled handshaking, as in text example, gives 75% maximum.
2. Successful communication with display using Interrupt-based handshaking, 100% maximum.
3. Successful communication with motor controller using Interrupt-based handshaking, and impressive manipulation of motor such as trapezoidal speed control gives 120% maximum.

**REFERENCES**

**Hall Text, Chapter 7 section on I2C.**

**AM335x Sitara™ Processors Technical Reference Manual,**

[**http://www.ti.com/lit/ug/spruh73q/spruh73q.pdf**](http://www.ti.com/lit/ug/spruh73q/spruh73q.pdf)

**NXP PCA9685 16 Channel PWM I2C LED Controller - on class D2L or**

<https://www.nxp.com/products/power-management/lighting-driver-and-controller-ics/ic-led-controllers/16-channel-12-bit-pwm-fm-plus-ic-bus-led-controller:PCA9685>

**SUGGESTED PROCEDURE FOR PART I OF THE PROJECT:**

1. Read through the Hall Chapter 7 section on I2C to get an overview of SCL and SDA signals, Start condition, Stop condition, ACK, etc. Note the order of the data and address bits in a word sent out on the I2C bus.
2. Next, consider some high-level initialization you need to do as follows:
   1. Take a look at a pin diagram for the P9 connector on the BeagleBone Black Board to find the pins identified as I2C1\_SCL and I2C1\_SDA. Note that the default for Pin 17 is the spi0\_cs0 signal, so you will have to find the conf\_ spi0\_cs0 register in the Control Module and change the mode to connect the I2C1\_SCL signal to pin 17. Likewise, you have to find the conf\_ spi0\_d1 register in the Control Module and change the mode as needed to connect the I2C1\_SDA signal to pin 18. **(You do not have to physically connect the motor driver module, since they are already connected correctly.)**
   2. Think about how you turn on the clock for the I2C1 module with the appropriate register in the Clock Module as you did for the timer and UART.

1. Read through the first few pages of the I2C Chapter in the Sitara manual. Study Figure 21-8 to give you another view of the I2C waveforms for the 7-bit addressing mode you will be using. Then pay particular attention to section 21.3.15. to help you make a high-level list of the steps required to initialize the I2C controller. Note how to get the system’s 48MHz clock scaled down to 12 MHz and how to get a 400 Kbps SCL for Fast/Standard mode (F/S). Also note that for the first implementation you will be using polled mode transmit as I did in the Hall Text example, instead of interrupt-based transmission. You will not be using the DMA capability due to the small number of bytes to transfer. (From my own experience I found it best to get control of the hardware and the data flow path with simply polling, that I could step through, instead of starting with the added complexity of the interrupt system.)
2. Read though the Register list in Table 21-8 and mark any registers mentioned in the introductory sections. Then skim through the descriptions for the registers you have marked to find the specified bits you will use to initialize the controller, start a transmission, and transfer the next byte when the controller is ready. I find it useful to print out copies of the descriptions for the registers I am going to be using, so I can write notes on them and include them in my log.
3. Construct your high-level algorithm and then your low-level algorithm for the program sections up to this point. This much is basically generic for communicating with any I2C device on a polled basis except for the steps to read from a slave as I did with the rangefinder in the Hall Text example. You will not be reading from the motor controller, so this is not needed right now.
4. Complete your High-level and Low-level algorithms (note structure of example in Hall Text).
5. Translate your Low-level algorithms to C or assembly language. Set your program to send out the slave address of 0x60 with the Read/Write bit a zero for the test. After a successful compile and load, it is time to run and test the program.
6. To test if the program is generating an SCL signal with the right frequency, connect a scope with a X10 probe to the purple/blue wire coming from the motor controller board. Connect the ground lead from the scope probe to the double stake pins on the right corner of the motor controller breadboard. Set the scope voltage scale on a range appropriate for an I2C signal and the scope time scale on a range reasonable for a 400 KHz signal.
7. When you get a stable signal on the SCL line, verify that the frequency is about 400 KHz.
8. Connect another X10 scope probe the SDA signal that is connected to the yellow wire from the motor controller board. Set the ranges on this channel of the scope to the same ranges as the SCL channel and set it to trigger on start of an I2C message.
9. Run the program from the start and see if the first message is being sent. You should be able to identify the point at which the Start condition is asserted on the bus and, if you remember the order in which the bits are sent out, you should be able to identify the slave address in the message bits. On the screen.
10. Print a copy or take a photo for your log, then print a copy of the final program and demo to TA to get signed off.
11. Congratulate yourself and get started on Part 2. Some hints for part 2 will be provided in a separate document.

Doug Hall

**GRADING KEY FOR ECE372 DESIGN PROJECT#2 D.V. HALL WINTER 2020**

**POSSIBLE SCORES**

**WORKING PROGRAM PT #1 PT #2**

**With TA questions**

**answered correctly 50 \_\_\_\_\_\_\_ 15 \_\_\_\_\_\_\_\_**

**ALGORITHMS**

**(CLEAR AND COMPLETE ) 10 \_\_\_\_\_\_\_ 5 \_\_\_\_\_\_\_\_**

**LOG (DETAILED 15 \_\_\_\_\_\_\_ 5 \_\_\_\_\_\_\_\_**

**AND “AS YOU GO”)**

**Write so that someone else**

**could trace all your thinking**

**and discoveries to your**

**final result.**

**TOTAL for versions 75 \_\_\_\_\_\_\_ 25 \_\_\_\_\_\_\_\_\_**

**Extra Feature up to 20 \_\_\_\_\_\_\_\_\_**

**TOTAL 100 \_\_\_\_\_\_\_\_**