Tracking Movement with the Triaxial Accelerometer

Alice Seaborn

Contact Information

Alice Triskele Seaborn
Electrical and Computer Engineering Department
Caterpillar College of Engineering and Technology
Bradley University

Harper Hall #xxx
1312 West Main Street
Peoria, IL, 61606, USA

Phone: 1+ (858) xxx-xxxx
E-Mail: xxx@mail.bradley.edu

Table of Contents

1 Project Proposal

1.1 Overview

My project aims to interface a ten degree-of-freedom (10-DOF) breakout board's accelerometer (LSM303) with the BeagleBone microcomputer by taking advantage of the Inter-Integrated Circuit (I^2C) communication protocol. Once the data from the sensors is acquired locally on the microcomputer it will parse and manipulate the raw data into a usable form through C code. This will allow the user to implement this system on various platforms to track motion data and determine trajectories in real time from robots working on a two-dimensional (2D) plane to rockets flying on complex three-dimensional (3D) paths. This will be accomplished through processing the three-axis accelerometer and possibly gyroscope data in combination with the magnetometer data.

I am personally fit for this project as I have extensive experience with the BeagleBone Black platform as well as knowledge of the C programming techniques necessary to implement the design specified above. I do not have experience with the I^2C communication protocol but I am confident that interfacing the hardware will be a minor portion of the work required to complete this project. Ultimately, this project will be a worthwhile learning experience as I will gain experience with common and useful hardware and the protocols which govern their use.

1.2 Specifications

The motion tracking program plans to successfully communicate with the LSM303 breakout board and measure the output through I^2C, which includes parsing the data packets being sent over the communication bus, and then to process that data into a usable form such that trajectories may be derived and information regarding the motion of the breakout board may be interpreted. This will require the following deliverables to be achieved:

Use I^2C to communicate between the breakout board and the BeagleBone

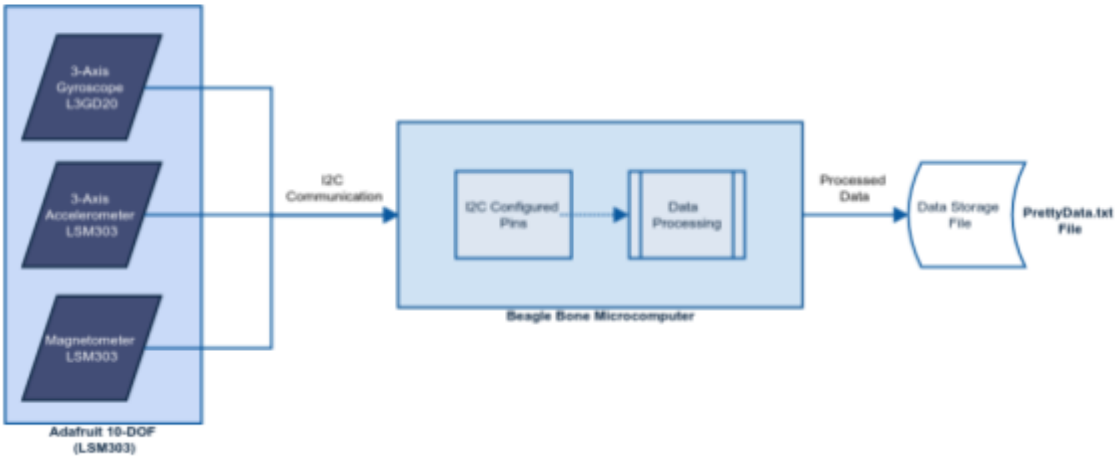
Collect the input data through the GPIO pins and save said data into a raw file

Parse the raw data file into a readable/usable form

This may ultimately be more difficult than it appears as the microcomputer may need to account for noise within the data it receives or signals that are extremely deviant as a result of the margin of error related to the breakout board. The sorting algorithm will face parsing this data as well as interpreting what data to add to the data file. This may be a complicated task.

In the future I hope to develop a program which will take the data file as input and output a graph of the movement data in either 3D or 2D along with critical data associated with the movement of the object, as this would be useful in a flight controller.

1.3 Block Diagram



The system will be primarily designed as a data processing unit and thus much of the effort will be spent interfacing the LSM303 to the BeagleBone via the I^2C communication protocol to secure the data and then processing said data via the data processing program. This means that there are basically two stages of this process to complete, namely to retrieve and then to process the LSM303 data. I will access the accelerometer, gyroscope, and magnetometer via the I^2C ports available on the breakout board and interface them to the BeagleBone via the on-board General Purpose Input/Output (GPIO) pins on the microcomputer.

First I must configure the Beagle Bone to handle the I^2C data then to solder the L3GD20 breakout board so that it will fit onto a mini breadboard. This way the microcomputer is configured to receive I^2C data and the pins are easily accessible and ready to be configured to the microcomputer itself. Once the GPIOs are connected to the breakout board and the board is powered I can begin to read the I^2C data and the interpretation process may begin. The GPIOs will be read by the Data Processing program and processed into usable data which will then be written onto a memory storage unit of some type, most likely a USB or a Micro-SD card.

As mentioned in section 1.2, I plan to develop an interpretation algorithm which can return a graphic representation of the data provided by the above illustrated process, extending the flight controllers utility.

1.4 Time-line and Milestones

As of April 10th the hardware required for this project including the BeagleBone Green Microcomputer as well as the LSM303 breakout board have been ordered and the expected delivery date is on or before April 15th. This means that over the weekend before April 18th it is within the realm of possibility that the I^2C data be successfully communicated using the hardware acquired.

The second stage is truly the filtering, interpreting, and storing of the received data into the storage medium that best fits the BeagleBone (USB or Micro-SD). This will consume several days of research but the timeline is definitely achievable. After the data parser is complete the project will have reached its conclusion and the application of that data will become the sole focus of the project, this defines the implementation

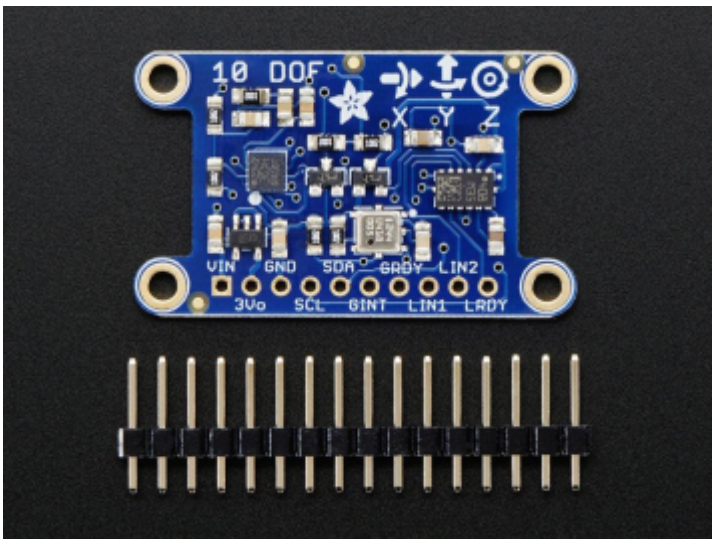
phase. From here the results will be compiled, the process refined to be reliable enough to present, and the final report will be drafted in time for the submission times.

2 Project Report

2.1 Introduction

Inter-integrated-circuit communication (I^2C) was initially designed to allow components to communicate with one or more masters in a rapid and reliable manner. This protocol requires the use of a data and clock wire such that the clock may synchronize the devices in the network and the data line provides the means for information transfer. Surprisingly, a single I^2C bus may support up to 1008 devices, making a successful implementation of the protocol a very useful technological asset.[1] Additionally, the I^2C bus supports high speeds of data transfer, making the protocol more than sufficient for applications which require high data-sample refresh speeds.

Devices capable of communicating through I^2C are diverse and include higher level logical components such as complex sensors along with lower level signaling units which communicate temperature and control data. Among them are affordable digital triaxial accelerometers and gyroscopes which can detect the forces of gravity and report angular position data respectively. The introduction of the affordable accelerometer into the maker-market has brought with it a wide range of capabilities which were previously inconceivable in the fields of robotics, aviation, and automation. With these powerful components, engineers may now design complex systems capable of self-balancing, detecting engine vibration, and even analyzing flight data in a data recorder.[2] Taking advantage of this device requires implementing a system which can read as well as write to the I^2C bus and then interpret that data into a usable form such that conclusions may be drawn regarding the system on which it is implemented.



It is the aim of this design project to produce a system which can interact efficiently with the accelerometer's I^2C data bus and then process the information gained therein to provide useful feedback for the user. This will be accomplished by generating C code capable of altering the settings on board the LSM303 accelerometer to configure the data that it captures and the manner in which it communicates it. From there a C program to read data from the bus, as opposed to writing to the bus, must be created and a data storage process must be designed therein. A separate program, written in Python, will then process the data and graph the results such that the user will have access to a friendly interface in

1 https://learn.sparkfun.com/tutorials/i2c#res Maximum I2C devices

2 http://www.gcdataconcepts.com/examples.html Applications of accelerometers

which to interpret the results of the sample. This progression between setup/writing to the bus, reading from the bus, and then interpreting data will be very logical and ideal as this is a proper representation of the scientific method.

The report on this project will seek to overview the process through which the product was produced, list and describe implementation results, explain any difficulties encountered therein, and then suggest ways in which this project could continue to develop into the future. The design process, described above, was implemented quite successfully and the progression between writing, reading, and processing the data was effective in assisting the research of I^2C and the methods necessary to make use of it. Although the results of the product were unique in comparison to the expected functionalities of the system, the product performed efficiently and the resultant data was what was desired.

2.2 Background

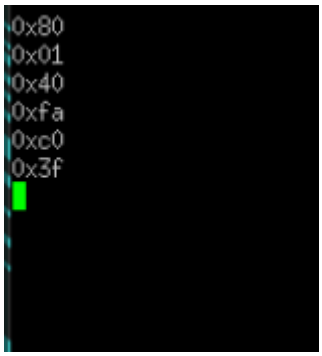
The protocol for I^2C has become quite popular and consequently many programs and tool sets are available for one to access the data bus of an I^2C capable device. One of these tool sets are the i2ctool packages available in the default Debian distribution present on all BeagleBone products. Included in this package are functions to detect devices connected to an I^2C bus, read the data bus on an I^2C device, set registers on the device bus to configure settings, and read specific locations on the device bus. These tools will be mentioned throughout the report as they were crucial for interacting with the device, its settings, and the information its utilities supplied the user.

```
root:~# i2cdump -y 1 0x19
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 33    ...............3
10: a1 1e a0 24 02 32 d9 a0 23 19 60 74 e0 20 80 00    ???$?2??#?`t? ?.
20: 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20    ?..............
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 33    ...............3
90: a1 1e a0 24 02 32 d9 a0 23 19 60 74 e0 20 80 00    ???$?2??#?`t? ?.
a0: 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20    ?..............
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
```

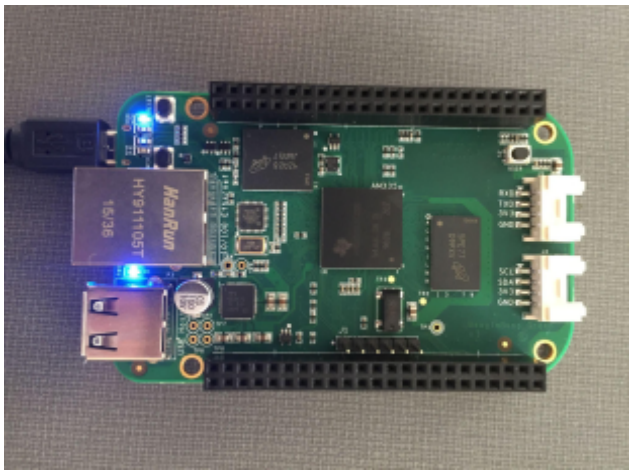
The data bus of the I^2C device was densely packed with information and settings. To read the bus the user could take advantage of the i2cdump function available in the i2ctools package mentioned above. The data bus from the accelerometer presents a lot of information. The addresses of interest included 0x20 and 0x28 – 0x2D and reported the states for the refresh setting and the axis accelerometer data respectively. By altering the 0x20 value using i2cset, the refresh rate could be maximized to provide the maximum amount of data to addresses 0x28 – 0x2D, which were later read using C code and analyzed through data conversion and graphing programs developed in Python.

When choosing which of the two languages (C language or Python) to compute the tasks included in this process, the question became a matter of what each language was best suited for. When dealing with lower level architecture and processes, C language is more efficient and faster at performing its given task. However, Python excels in performing higher level analysis as it has superior libraries, making the development process easier and thus allowing the developer to realize new heights in programming which may have been previously unreachable in C language. Ergo, as a consequence of the performance benefits of each language, C was chosen to rapidly and reliably interact with the I^2C device and compile the raw data while Python was tasked with analyzing and graphing said data into an interpretable form.

2.3 Hardware



The hardware components used throughout this project include the LSM303 accelerometer as part of the Adafruit 10-DOF breakout board and the BeagleBoneBlack embedded Linux microcomputer from the beagle-board product line. By interfacing the accelerometer with the microcomputer, the data was easily compiled and sampled through code executed locally on the BeagleBone machine. Determining which data segments were of use and which were irrelevant to the project was made possible through the data-sheet provided for the LSM303 accelerometer. Specifically, the table 17 “Register address map” (page 23), the CTRL_REG1_A (20h) linear acceleration register description (page 25), and acceleration data sections 7.1.9 – 7.1.11 were crucial in understanding the intricacies of the component. Using the information described in these sections, code could be designed around the accelerometer's features.



The BeagleBoneBlack microcomputer was used alongside the BeagleBoneGreen, a similar product designed with the same embedded Linux platform that could be acquired at a lower price. The BeagleBoneGreen was used during the early stages of the research out of necessity as the BeagleBoneBlack was not available. Ultimately the BeagleBoneBlack was received and used for the final stages of the project as it has superior documentation and electrical specifications.

2.4 Design Procedure

As dictated by the CTRL_REG1_A (20h) register, the default data refresh rate for the LSM303 triaxial accelerometer is set to 0Hz. This means that the data will remain static unless the settings for this register are altered. Using the i2ctools i2cset function, the 0x20 register was altered to a desired refreshing frequency. This was accomplished using the SETUP.c program included in this submission. This program takes user input and prints to a string to form the i2cset command before using the system() function to send the preformed command to the terminal for execution. This reduced the error the user may encounter due to typos, as the i2ctools can damage the device or the I^2C bus itself if used improperly.

Now that the device is properly configured, a program was developed to read from the data bus and collect the desired accelerometer data, which is now being refreshed at the specified rate, before saving it into a file for later analysis. To read from the bus, a process of similar fashion to that of SETUP.c was employed in a function named BUS_READ. The program forms i2cget commands for each desired data segment and routes the return data, which is typically echoed to the terminal screen through the stdout stream, to a file by the name of RAWdata.txt. This data arrived as a hexadecimal representation of a 2's complement binary number and was saved in a column.

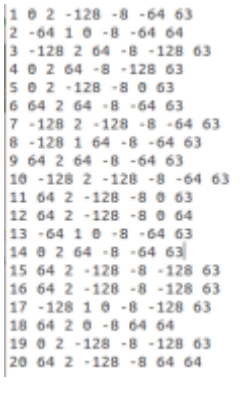
From here a function named LSAVE captured the data present in RAWdata.txt and saved it locally in a linked list designed specifically to meet the needs and architecture of the data received from the device. In addition to the axis data, time stamps were appended to each entry of the linked list. Considering that a rapid refresh of the data was necessary to gather useful results, a refresh rate was implemented in the main function to gather a total of ten samples per given second of the total duration of the program's execution, which was specified by the user as an argument sent to main upon the execution of the program. Once the duration of time had elapsed, the program would dump the contents of the local linked list into a formatted file named PrettyData.txt for future analysis.

With the formatted data compiled it was time to convert the values from hexadecimal to binary and then to a signed decimal so it could be graphed with ease. A legitimate attempt was made to accomplish this task in C language and although the CONVERT function in PARSE.c does successfully convert the hexadecimal data to its desired decimal destination, a programming bug was found but never solved in which the data was exported as a hexadecimal regardless of its conversion. However, as conversion increased the time complexity of the C program, a conversion process developed in the python analysis program was logical as the time complexity of post-implementation analysis was less important than the time complexity of the data sampling, in which the accuracy of the time stamps were crucial in relating the reality of the physical system to the data samples taken of said system.

The PrettyData.txt file was exported from the device using a remote file transfer program where it was analyzed by a python program which converted the hexadecimal values to their decimal equivalents and saved the data into a temporary file named Test.txt. This data was then analyzed by a graphing program which generated graphs of the acceleration for user analysis and interpretation.

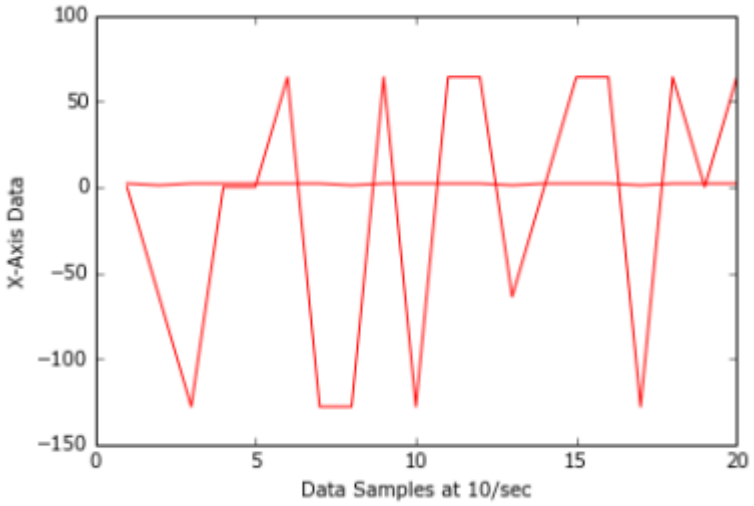
2.5 Implementation Results

The data sampling program produced by the research project was quite effective in taking data samples from the device data bus. Unfortunately, the time complexity of the program interfered with the accuracy of the time stamps; however, this deviation was not so great as to compromise the program's utility and thus, while it remains an anomaly deserving of further study, the program remains effective for short-term data samples of twenty seconds or less. Concordantly, the program is not without its bugs and areas for improvement in data sampling efficiency. For example, the CONVERT function, while effective in converting hexadecimal representations of 2's complement numbers to a signed decimal value, did not complete the task of providing a converted data file and thus is additionally deserving of further study.



The analysis of the data file using the python programming language rectified the conversion failures of the C program by not only converting the data but also providing graphs of acceleration in a separate program. The axis data had both LOW and HIGH components, although the significance of the HIGH component (Blue) is less understood, it has become clear that the LOW component (Red) is more accurate in reflecting the g-forces in the system, which in this sample was moving more gradually than is depicted by the HIGH component.

Data from all axes graphed into a single frame could be more easily analyzed and conclusions regarding the movement of the system in three-dimensional space could be more easily drawn. The system, which was slowly transformed about all axes, visually demonstrates movement in all axes. Here the X-axis (Red), Y-axis (Blue), and Z-axis (Green) easily reflect the g-forces of the system as a whole.



2.6 Conclusion

The programs resulting from the research project are effective at taking short-term data samples of a system moving in three-dimensional space and then analyzing those samples to produce readable graphs from which conclusions about the system may be drawn. Given the amount of data which the system is capable of collecting, it is possible to integrate the data file to achieve an output of the positional data rather than the acceleration. When used in combination with a triaxial gyroscope sensor to relay angular data, these programs could effectively develop a flight data recorder or localized navigation system for drone and robotics applications alike.

Although the programs suffer from several setbacks, including time-complexity, they accomplish the given task of communicating to and from an I^2C device bus using C language. Applying such concepts as linked list, file manipulation, user-defined functions, pointers, bitwise operations, and arguments sent to main, the resulting program is effective in not only altering settings on board an I^2C device through editing registers but is also effective in gathering data from the device bus for compilation and later analysis. This required additional research in communications protocols, advanced data types, data streams, and file operations to complete the project and produce the desired outcome.

Furthermore, the data gathered in the aforementioned C program was later analyzed through the implementation of python code to not only convert but also graph the data into a figure for the use to evaluate.

This required research into the use of data type conversions and the graphing functions included in the matplotlib library. By providing the user with illustrations of the movement of an object in three-dimensional space, many technological applications become available.

More is to be discovered within the complexities of both the I^2C protocol and the LSM303 accelerometer alike. Through additional research, I am confident that the time-complexity of the C code could be significantly reduced and thus the resolution of the resulting data samples improved. With time it is my desire to increase the resolution of the accelerometer sample data such that detecting the subtler forces of vibration will become within the range of the product. Additionally, with the integration of a triaxial gyroscope, which is also an I^2C device, many more technological applications will be within range of this project's grasp.