

**DUBLIN CITY UNIVERSITY**

**SCHOOL OF ELECTRONIC ENGINEERING**

**A Wireless 3D Embedded RTOS Human Computer Interface**

**Ciarán O’Donnell**

April 2019

BACHELOR OF ENGINEERING

IN

Electronic & Computer Engineering

MAJOR IN

Digital Interaction

Supervised by Dr. Derek Molloy

# Acknowledgements

I would like to thank my supervisor Dr. Noel Murphy for his guidance, enthusiasm and commitment to this project. Thanks are also due to Dr. Conor Brennan for supporting this work to date, etc. Finally I would like to express my deep appreciation to Dr. Derek Molloy for preparing the original Transfer Report template, from which this document is hacked.

**IMPORTANT NOTE ON REFERENCES**

One area that some students fall down on is the area of references. The guideline here is quite simple: either you did the work and wrote the text, or *someone else did*. For any element of your dissertation, even the tiniest element, which falls into the latter category, you must provide as complete a reference as possible, so that another researcher can easily access exactly the same source of information as you have. The desired form for the reference data is usually that used in IEEE journals. Please examine carefully the references used in this document in the References section after section 6 that includes an example of how to reference a document from the Internet. (Please delete this note when using this document as a template). More recent versions of MS Word will allow you to ‘Manage your Sources’ under the ‘References’ tab and Bibtex is great if you are using Latex to prepare your document.

# Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the DCU Academic Integrity and Plagiarism at <https://www4.dcu.ie/sites/default/files/policy/1%20-%20integrity_and_plagiarism_ovpaa_v3.pdf> and IEEE referencing guidelines found at https://loop.dcu.ie/mod/url/view.php?id=448779.

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

This Microsoft Word document file is intended to fulfil two purposes: to act as a template for the generation of a project report for the Final Year Projects in the School of Electronic Engineering’s undergraduate degrees and to give examples and guidelines for the writing of this report. The style and structure proposed here reflect the author’s own personal pre-occupations and any suggestions for corrections or improvements would be very much welcome. It is suggested that you make a copy to act as the template for your document and that you keep a copy of the original (this document) to access the guidelines as required.

Now for some guidelines. The abstract should briefly tell me, the reader, about this document. Basically, having read the abstract, I should be able to determine if it is worthwhile for me to read any further. The abstract should concentrate on the content of this document, not on the state of the art, or the wider implications of the research work of which this is a part. It should delimit the scope of the work by indicating how far you have gone. It should not, however, detail the document on a chapter-by-chapter basis. Remember also, the next thing I am likely to look at after the abstract, is the conclusion, to see exactly what you have achieved. Note that abstracts are generally 100- 250 words, address a more technical audience, and do not typically include recommendations, "bottom line" figures, and comments on significance of the findings. The abstract should include

* An introductory sentence that create interest and draws attention to the topic;
* The project definition and goals;
* The method of solution;
* The results; and,
* The conclusions.

Place the abstract on a separate page single spaced with no indentions.

(Even though the remainder of the document uses 1.5-line spacing, the text in this Roman numeral section is usually single-spaced). This is revision 2.0 of this document. We hope to continue to make improvements to this document as we gain experience of its use.

Table of Contents

[Acknowledgements ii](#_Toc5391252)

[Declaration ii](#_Toc5391253)

[Abstract iv](#_Toc5391254)

[Chapter 1 - Introduction 1](#_Toc5391255)

[Chapter 2 - Technical Background 2](#_Toc5391256)

[2.1 Human Computer Interaction 2](#_Toc5391257)

[2.1.1 Current HCI Devices 2](#_Toc5391258)

[2.1.2 3D HCI Devices 2](#_Toc5391259)

[2.1.3 Interfacing with a Computer 3](#_Toc5391260)

[2.2 Real Time Operating System (RTOS) Embedded System 4](#_Toc5391261)

[2.2.1 Real Time Operating Systems 4](#_Toc5391262)

[2.2.2 Embedded System 4](#_Toc5391263)

[2.3 Three-Dimensional Data 5](#_Toc5391264)

[2.3.1 3D Rotational Data 5](#_Toc5391265)

[2.3.2 3D Positional Data 5](#_Toc5391266)

[2.4 Wireless Communication 6](#_Toc5391267)

[2.4.1 Wireless versus Wired Communication 6](#_Toc5391268)

[2.4.2 Current Standards 6](#_Toc5391269)

[Chapter 3 - Design of the Wireless 3D Embedded RTOS HCI 7](#_Toc5391270)

[3.1 Parts Choice 7](#_Toc5391271)

[3.1.1 MCU 7](#_Toc5391272)

[3.1.2 Accelerometer/Gyroscope 7](#_Toc5391273)

[3.2 TI-RTOS 7](#_Toc5391274)

[3.2.1 Task Construction 7](#_Toc5391275)

[3.2.2 Pin Selection 8](#_Toc5391276)

[3.2.3 TI-RTOS Initial Programming Test 8](#_Toc5391277)

[3.2.4 TI-RTOS Pre-Supplied Libraries 9](#_Toc5391278)

[3.3 Interfacing with the Accelerometer 9](#_Toc5391279)

[3.3.1 Connecting SPI Device 9](#_Toc5391280)

[3.3.2 Connecting I2C Device 10](#_Toc5391281)

[3.3.3 Programming for SPI 11](#_Toc5391282)

[3.3.4 Programming for I2C 12](#_Toc5391283)

[3.3.5 Data Transmission 12](#_Toc5391284)

[3.4 Receiver Device 13](#_Toc5391285)

[3.4.1 Receiving Data 13](#_Toc5391286)

[3.4.2 Host PC Interfacing 13](#_Toc5391287)

[Chapter 4- Implementation and Testing of Wireless 3D Embedded RTOS HCI 14](#_Toc5391288)

[Chapter 5 - Results and Discussion 15](#_Toc5391289)

[Chapter 6 – Ethics 16](#_Toc5391290)

[Chapter 7 - Conclusions and Further Research 17](#_Toc5391291)

[References 18](#_Toc5391292)

[Appendix 20](#_Toc5391293)

[TI-RTOS Flashing LED Code: 20](#_Toc5391294)

[SPI Device ID Code: 21](#_Toc5391295)

[TI Packet Transmission Code 22](#_Toc5391296)

[TI Packet Reciever Code 25](#_Toc5391297)

[Glossary 29](#_Toc5391298)

[This table has been automatically generated using Heading styles (see Chapter 2) – please remove this note for your document; this table is indicative – check the marking rubric to ensure you have covered all the necessary content]

Table of Figures

[**Figure 1.1**. A Nice Picture. 1](#_Toc381621110)

[Figure 2.1 A Sample Figure 10](#_Toc381621111)

# Chapter 1 - Introduction

Computer interaction has become daily life for millions of people around the world. Many computer devices have become a necessity in our day to day lives. Therefore, how people interact with computers has become very important. There are currently several standard ways to interact with computers including keyboard, mouse, monitor and speakers. However recently many more possible methods of interacting with computers have become more viable, including touchscreen interfaces, face and/or eye tracking, speech recognition.

This project will investigate the possibility of using a real-time operating system micro-controller as a base to design a device that allows a user to have 3D interactivity with a computer. This would allow for manipulation of 3D data such as architectural models, medical imagery, or an avatar within a video game environment.

# Chapter 2 - Technical Background

## 2.1 Human Computer Interaction

### 2.1.1 Current HCI Devices

Human computer interaction refers to the ability for a person to give input or take input from a computer device. There are already a vast number of devices that allow this. The most common human computer interface devices include, keyboard, mouse, display and speakers, but any device that a person may use to interact with a computer is a HCI. The device that will be investigated in this report is a HCI that allows a person to input 3D data to the computer.

### 2.1.2 3D HCI Devices

The 3D HCI device investigated within this report should track either 3D positional or rotational data and input to the computer. There have been similar devices designed before. The first major consumer 3D HCI device is the controller of the Nintendo game console called the “Wii”. It can be seen in the devices patent that was granted in the US on 09/09/2008, that the controller uses acceleration data collected before and after a predetermined amount of time to calculate the relative motion of the controller across two axis. [1] This allows the device to communicate rotational data to the console. The controller can also calculate positional data by means of an ultra-violet camera onboard the device. This works in tandem with ultraviolet LEDs placed across from the device as stated in this US patent assigned to Nintendo Co. [2] This gives the device information to determine its 3D positional data along a single axis parallel to the LEDs.

Another method of 3-dimensional human computer interfacing is displayed by Iason Oikonomidis, Nikolaos Kyriazis, and Antonis A. Argyros in their design for 3D Tracking of Hand Articulations. [3] This design uses the Xbox ‘Kinect’ camera and infrared sensor arrays and image processing techniques to quantify 3D information.

These are two examples of 3D HCI devices that show some of the mothods possible to communicate 3D information with a computer. However these devices have their own drawbacks, one such drawback being that they are proprietary in design.

### 2.1.3 Interfacing with a Computer

Communication with the host machine must be completed for the HCI device to work. Most modern HCI devices use universal serial bus in order to communicate with a computer. This report will investigate using USB as the communication bus between an MCU and the host computer for this functionality. Modern operating systems include support for general input devices. Windows allows for input devices to be designed using custom drivers or general human interface device communication over USB like described in Silicon Labs tutorial on the design of HIDs. [3] This tutorial is specifically for Silicon Labs devices, but the concepts apply to all USB HID devices.

Another potential resource for the design of communication between the device and host computer is with the use of Valve Corporation’s ‘Open VR’ SDK. [4] This SDK include repositories to allow the design of VR controllers to work with steam VR. This would allow the device to communicate 3D information to the computer however it would also limit the device to only work within the ‘Steam VR’ ecosystem.

## 2.2 Real Time Operating System (RTOS) Embedded System

### 2.2.1 Real Time Operating Systems

A real time operating system refers to operating system software that has a predetermined amount of clock cycles for all instructions to take place. It is designed as such to allow the device to have fast, highly accurate, and efficient performance for time sensitive operations. The use of a RTOS for a HCI is beneficial as it allows for the fast performance which is important as when a user is interacting with a virtual environment any latencies in the users action and the action occurring on the computer are unwanted and users can likely detect changes in latency as low as 33msec according to a paper by Stephen R. Ellis, Mark J. Young, Bernard D. Adelstein, and Sheryl M. Ehrlich. [5]

### 2.2.2 Embedded System

An embedded system is what shall compute the 3D data and communicate it to the host computer. It is therefore important that the embedded system used is one capable of communication with a computer, computation of the 3D data and one that operates using a real-time operating system. The Texas Instrument Simplink cc2650 meets these requirements and was opted as the platform for this project. [6] This development board is capable of several different methods of wireless communication, as well as having a 48MHz processor which is more than capable of the 3D data computation needed. It has an array of GPIO pins to allow integration of many methods of 3D data collection. The cc2650 also has support for TI’s real time operating system software TI-RTOS.

## 2.3 Three-Dimensional Data

### 2.3.1 3D Rotational Data

Tracking and recording of 3D data onboard a device is most often completed using one or more of the following three technologies, a gyroscope, accelerometer or a magnetometer. Each varies in operation, but all can be used as a method of computing 3D rotational data. A gyroscope is capable of calculating rotational velocity accurately, as it can sense motion relative to that of gravity. This allows for accurate rotational motion data but is susceptible to drift as the device has no way to calculate current rotational position. An accelerometer works via sensing acceleration along a single axis; however three linear accelerometers can be used orientated perpendicular to each other to calculate acceleration in an x, y and z direction and then calculate based off that the orientation of the device against the acceleration due to gravity on the accelerometers. A magnetometer simply senses the direction of a magnetic field and can take advantage of the planets natural magnetic field to calculate the devices rotation in reference to the magnetic north pole. This method is limited and susceptible to magnetic interference. Therefore, this method should most often be incorporated along with one or both of the previous two.

### 2.3.2 3D Positional Data

As well as rotational data of the device, the devices position in 3D space could also be collected and communicated to the host computer. There are various methods this data could be collected including the methods used by Nintendo’s ‘Wii’ and Microsoft’s ‘Kinect’ discussed earlier. Another possible method would be to use distance sensors onboard the device to calculate distances from know objects such as the ground. This method would allow the device to remain a self-contained device without the need of peripherals such as cameras but would be extremely susceptible to interference from other objects. Tracking the 3D position of the device is an interesting area to investigate but will remain a lower priority than the rotational data for the purpose of this report.

## 2.4 Wireless Communication

### 2.4.1 Wireless versus Wired Communication

To communicate with the host PC the device must have some form of connection. The device could connect to the host pc via a wired connection such as USB, or communicate wirelessly. The advantages of wired communication is that it can be simpler and more reliable however it also can become cumbersome. In this scenario where the device shall need to move freely within 3D space a cable would be very inconvenient and cumbersome and therefore this report will investigate the use of wireless communication for this device.

### 2.4.2 Current Standards

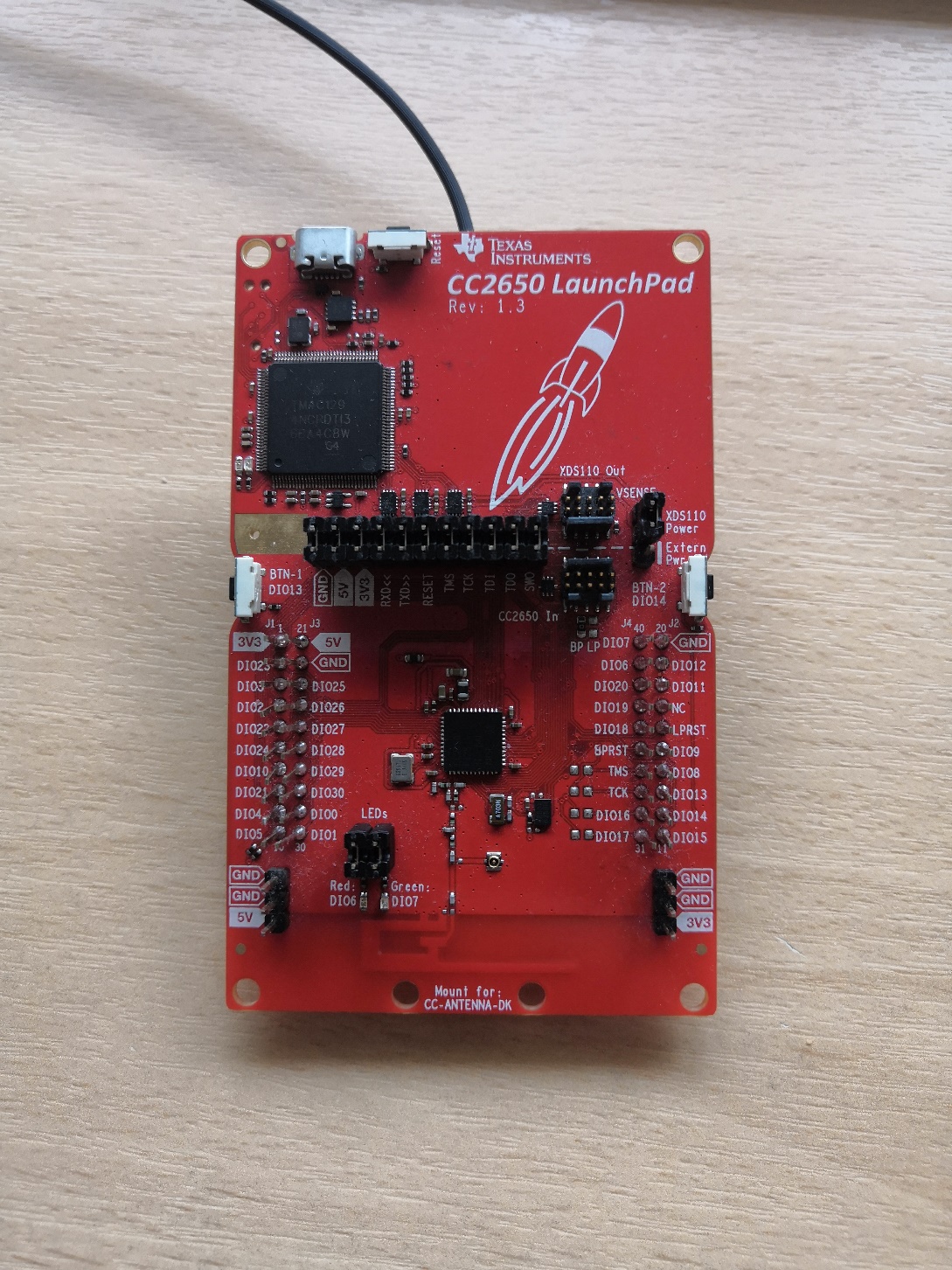
For the device to be functional it will need to be able to wirelessly transfer data to the computer. The method of transportation will have a major impact on the responsiveness of the device. As this device will be used with the computer it is communicating with at all times, the communication will therefore only ever be over a short distance. There are many new communication methods being standardised for short range communication wirelessly. These personal area networks (PANs) can be very fast and low powered ways of sending data point to point. Some of these methods of wireless communication are described within an article by Cheolhee Park and Theodore S. Rappaport. [7] The SimpleLink device described earlier are capable of many of these communication standards including Wi-Fi, Bluetooth, as well as others described on the ti website. [8] One interesting capability of this device is an extremely low powered and low overhead device to device communication between two of these SimpleLink devices. This could allow for one device to track and send 3D data and the other to receive the data and interface with the host computer over a wired connection such as USB. This could drastically simplify the communication with the computer.

# Chapter 3 - Design of the Wireless 3D Embedded RTOS HCI

## 3.1 Parts Choice

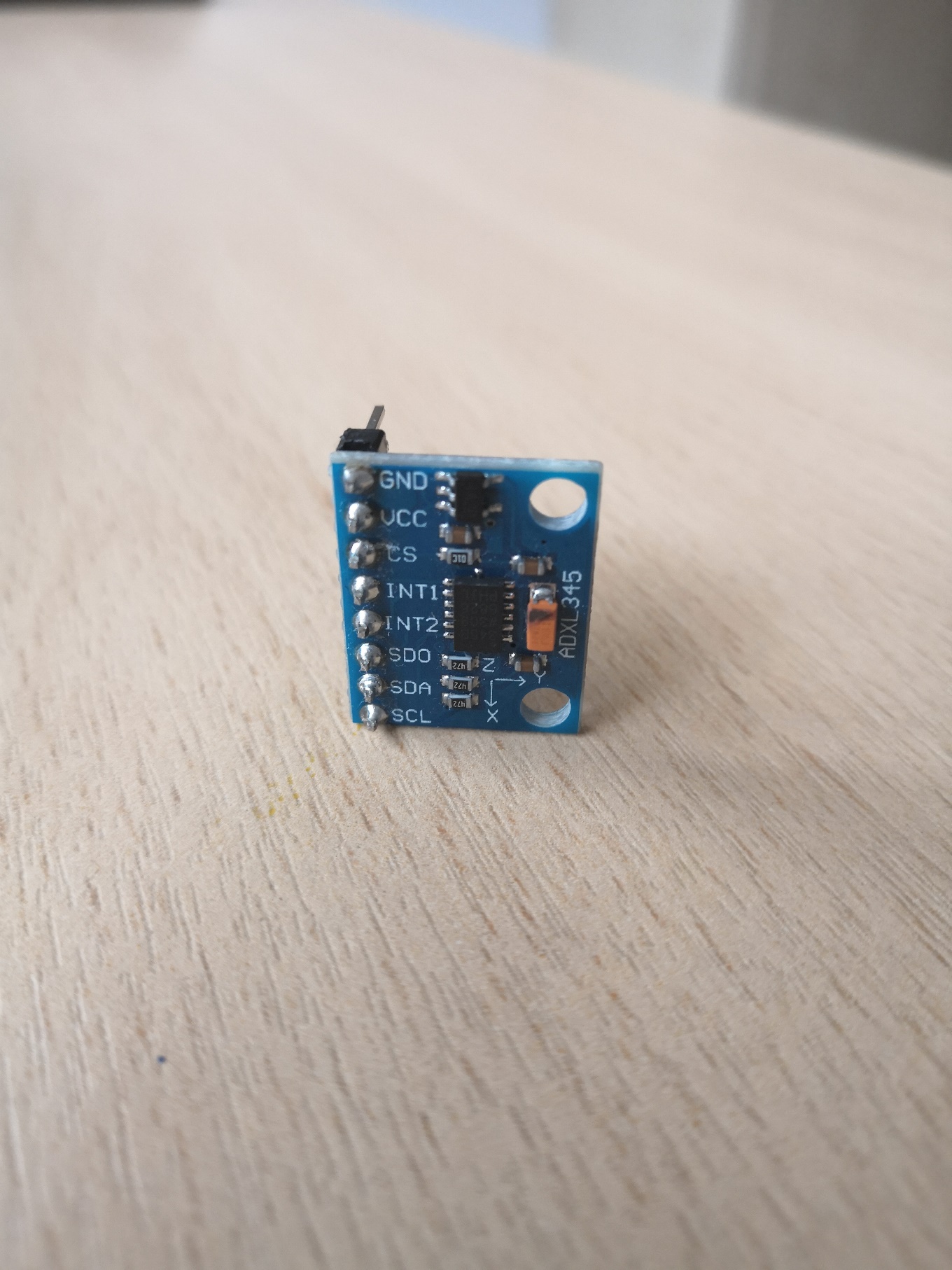
### 3.1.1 MCU

The Texas Instruments cc2650 Launchpad was chosen as the micro controller for the device. For its wireless communication capabilities, adequate computational performance and user accessible to GPIO. Two of these are deployed in a transceiver, receiver pair, with the transceiver interfacing with the accelerometer and sending its data to the receiver. The receiver takes this data and then communicates it to the host computer as a HID instruction.



### 3.1.2 Accelerometer/Gyroscope

For the gyroscope/accelerometer an ADXL345 was chosen. This device was chosen for its ability to communicate using either I2C or SPI interfaces, its ability to accurately measure across 3 axes up to a resolution of 13bits at ±16g, with lower resolution modes ±2g, ±4g, and ±8g, and its low typical power draw of 0.462mW. [9]



## 3.2 TI-RTOS

A major element of this project is working within the real time operating system environment. It is therefore important to understand the concepts of how the specific environment the device will be designed in operates. TI-RTOS is the operating system that is to be used. Its syntax bares some similarities to C and C++ but differs substantially in operation.

### 3.2.1 Task Construction

TI-RTOS works on setting up threaded operations and then allowing the device bios to schedule and run each thread. The threads are constructed as tasks on initialisation of the device. The user must give the task parameters and functionality. Initialize the task and once all tasks and functionality of each task is defined the BIOS runs them all. Tasks must have a defined stack and stack size i.e. the area on the device the instructions will be stored, as well as a priority. This priority will determine the processing time that the bios grants the task. The tasks functionality must also be defined, in a similar way to a function definition in C or C++ and the task is constructed to point towards it.

Task\_Params\_init(&taskParams);

taskParams.stack = notifyTaskStack;

taskParams.stackSize = NOTIFY\_TASK\_STACK\_SIZE;

taskParams.priority = NOTIFY\_TASK\_PRIORITY;

Task\_construct(&notifyTask, NotificationTask\_taskFxn, &taskParams, NULL);

[10]

Once a task is defined and created, once the command BIOS\_start() is ran the threads will begin.

### 3.2.2 Pin Selection

Pin selection is another important element of TI-RTOS to understand in the design of this device. In order to control the pins of the cc2650, they are first declared on a pin config table this table. Then a pin handle is opened linked to the pin config table, and the pins are ‘opened’. This allows the user to now control the pins as they see fit. From that point pins can be controlled by referencing the pin table, the individual pin and then setting the pin value.

**static** PIN\_Handle MyPinHandle;

**static** PIN\_State MyPinState;

PIN\_Config MyPinTable[] = {

Board\_LED0 | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_LOW | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

Board\_LED1 | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_LOW | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

PIN\_TERMINATE

};

/\* Open LED pins \*/

MyPinHandle = **PIN\_open**(&MyPinState, MyPinTable);

**PIN\_setOutputValue**(MyPinHandle, Board\_LED1, 1);

**PIN\_setOutputValue**(MyPinHandle, Board\_LED0, 0);

### 3.2.3 TI-RTOS Initial Programming Test

With these capabilities it is possible to test the TI-RTOS environment with a program similar to that of a hello world, flashing LEDs. This simple application is the first proof of concept of use of the TI-RTOS. A program was written to create a task to control the LEDs on the board which is available in the appendix at the end of this report.



### 3.2.4 TI-RTOS Pre-Supplied Libraries

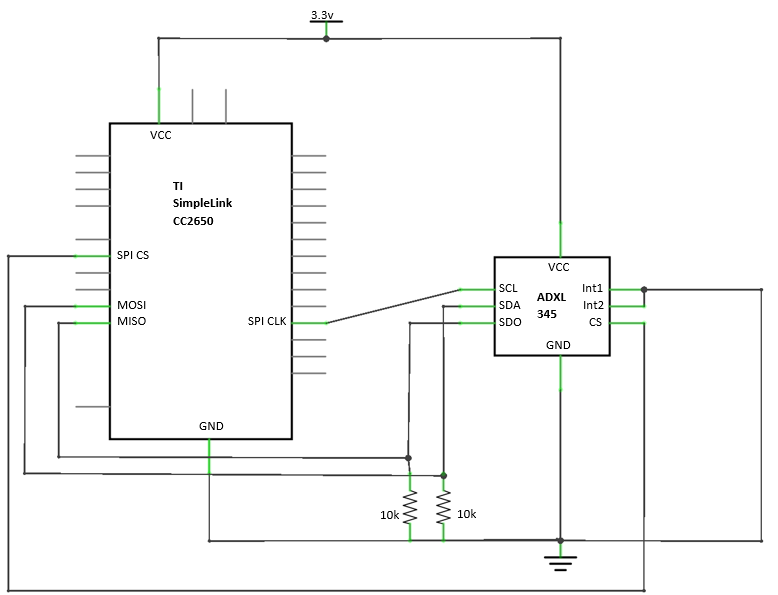
Texas Instruments have several libraries wrote for TI-RTOS that are relevant to this project. There are libraries available for SPI and I2C communication built to simplify interfacing with other devices, SPI.h and I2C.h allow the user to create connections and make data transactions with the device. [11] [12] This should simplify communication with the ADXL345 possible and therefore the functionality of the libraries was studied.

## 3.3 Interfacing with the Accelerometer

The MCU must communicate with the ADXL345 to collect the necessary 3D data. There are two potential communication standards that both the CC2650 and the ADXL345 support, I2C, and SPI. Both possible designs have been detailed below.

### 3.3.1 Connecting SPI Device

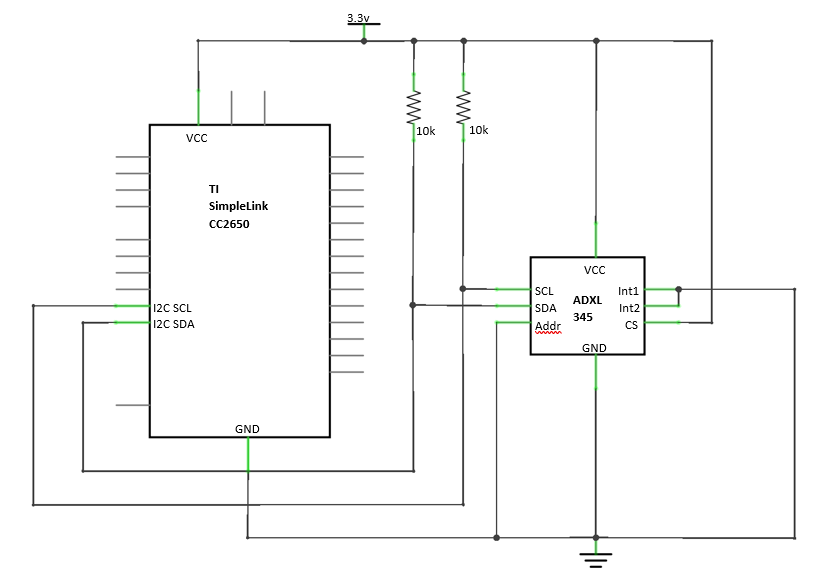
The first design for the communication between the MCU and the ADXL345 is with the use of the SPI Interface. The SPI Interface uses a connection of a clock signal, two data lines, a master out slave in (MOSI) and master in slave out (MISO) as well as a chip select line to activate the slave for communication. The ADXL345 acts as a slave device with the CC2650 acting as the master and controlling communication between them both. The ADXL345 also has two active high interrupt pins as well as a standard supply voltage and ground pin.

The CC2650 can use any of the GPIO to serve for the communication but have pins pre-defined by the SPI library also so these will be the ones used in this design. 

The advantages of SPI over I2C is quicker communication as it requires less data transections due to the fact that I2C does not use a chip select line and instead transmits a slave devices ID to begin communication. Its disadvantages include the need for more GPIO as each device requires its own chip select line, and it has a more complex such as setting the phase and polarity of the clock, whereas in I2C these parameters are defined.

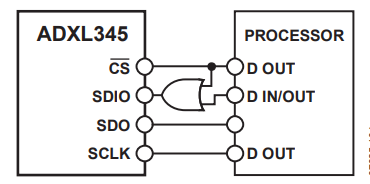
### 3.3.2 Connecting I2C Device

Interfacing with the ADXL345 using I2C uses a clock signal and a single data line. The master device initiates communication via sending the slave devices ID onto the communication line, then the slave acknowledges, and the master continues the communication. For the ADXL345 to have I2C mode activate the chip select line must be high and therefore in this design it is connected to Vcc, also the data lines are pulled high to ensure no floating connections, and the interrupt pins are again connected to ground. The SDO line acts as an alt address controller for I2C so that there are two possible address the device may use based on whether the alt address pin is high or low, so in this design it is also grounded.



Here we can see how the ADXL345 in I2C mode needs less of the GPIO of the MCU to function. Also, if there were more devices to be added they could all communicate over the same two GPIO pins as long as the slave device addresses do not conflict.

Another consideration with the ADXL345 is how the I2C mode is activated via the chip select pin being high. Therefore when using the device in SPI mode there is the potential for a communication on the data line to be considered an I2C communication. The solution to this is described in the data sheet to simply add an OR gate with the chip select and data line before the SDA line of the ADXL345.

 [9]

### 3.3.3 Programming for SPI

SPI communication is done in TI-RTOS via first declaring an SPI Handle object and an SPI Parameters object. These objects handle the connection and store parameter settings respectively. The parameters include the transfer mode, frame format, SPI mode, bitrate, and data size. Once the parameters are set the connection can be opened and set as the handle. Then a transaction object is created with its own parameters for each transaction, including the transmit and receive buffers and the count of data chunks to send. So, a simple SPI transfer is wrote as below:

SPI\_Params\_init(&spiParams); //initialise spi params object

spiParams.transferMode = SPI\_MODE\_BLOCKING; //block thread until transaction complete

spiParams.transferTimeout = SPI\_WAIT\_FOREVER;//wait forever

spiParams.frameFormat = SPI\_POL1\_PHA1;//clock polarity 1 phase 1

spiParams.transferCallbackFxn = NULL; //no callback

spiParams.mode = SPI\_MASTER; //master mode

spiParams.bitRate = 400000; //400MHz

spiParams.dataSize = 8; //data size

handle = SPI\_open(0, &spiParams);//open spiHandle

transmitBuffer[0] = 0xAD;//write to power control

transmitBuffer[1] = 0x08;//turn device on

spiTransaction.count = 2;

spiTransaction.txBuf = transmitBuffer;

spiTransaction.rxBuf = recieveBuffer;

ret = SPI\_transfer(handle, &spiTransaction);

**if** (!ret) {

System\_printf("Unsuccessful SPI transfer");

}

This full program code is available in the appendix at the end of this report.

For this device once, the connection is initialized a threads operation should be to constantly update x, y, and z data and a separate thread handles sending this data to the receiver. The X, Y, and Z data is accessible via their own SPI transactions to the registries 0x32 to 0x37. Each coordinate uses two bytes so multi-byte reads are needed for each.

### 3.3.4 Programming for I2C

The programming for I2C communication is very similar to that of the SPI interface. A connection handler object and parameters objects are made in the same way. The paramaters include slightly different data as clock phase and polarity does not need to be set. Then the transaction is given perameters in a similar way however this time must have a slave address specified.

**I2C\_Params\_init**(&params);

params.transferMode = *I2C\_MODE\_BLOCKING*; //thread waits for transaction

params.transferCallbackFxn = NULL; //no callback needed

params.bitRate = *I2C\_400kHz* ; //400kHz

txBuffer[0] = 0;//Get Device ID

i2cTrans.writeCount = 1; //one byte right (slave address is handled by I2C.H

i2cTrans.writeBuf = txBuffer;

i2cTrans.readCount = 1;

i2cTrans.readBuf = rxBuffer;

i2cTrans.slaveAddress = 0x3A;//slave address when alt address pin is grounded

handle = **I2C\_open**(Board\_I2C, &params);//open connection

**I2C\_transfer**(handle, &i2cTrans);//make transfer

System\_printf("testing %d \n", rxBuffer[0]);

System\_flush();

**I2C\_close**(handle);

Apart from these slight differences the data is then collected identically as stated in the SPI design. Once the thread begins collecting data the XYZ Data can be sent by the transmission thread to the receiver device.

### 3.3.5 Data Transmission

Once the data is received from the ADXL345 it must be sent wirelessly to the receiver. This runs in a separate thread that sends the XYZ data to the receiver continuously. It would be more energy efficient to send the data only when there is a change, but this would add additional latency and complexity and therefore instead the data will be sent at a constant rate and changes in the position will be calculated on the receiver’s end.

TI have sample code for use with the CC2650 on sending packets and therefore this code will be tested and then deployed with the rest of the program and added as necessary. This code is available from the TI-RTOS resource files and also made available in the appendix of this report. [13]

## 3.4 Receiver Device

The receiver device is another CC2650, its operation is to receive the XYZ data from the transmitter, once received it will calculate any movement via a difference in the values to those previously received. Once calculated this difference will then be communicated to a host computer as movement of a human interface device (HID). One thread will receive the data and calculate the change in data as it is ingested, then a separate thread will communicate this to the computer. This will allow for one thread to be near constantly be waiting on updates from the transceiver as the other thread operates the interactions with the computer.

### 3.4.1 Receiving Data

Much like the data transmission, TI also have made available sample code for the recovery of data from a CC2650 to another, therefore this code will again be deployed tested and edited to suit this application. The code is available again from the TI Resource repositories for TI-RTOS and the CC2650 section specifically and again is displayed in the appendix of this report. [13]

### 3.4.2 Host PC Interfacing

Once the data is received and the change in data is calculated this can be communicated to the computer via a serial connection. There have been

#############################Finish l8r

# Chapter 4- Implementation and Testing of Wireless 3D Embedded RTOS HCI

# Chapter 5 - Results and Discussion

# Chapter 6 – Ethics

# Chapter 7 - Conclusions and Further Research

The conclusions chapter is very important in your report. It must conclude your work! It is not a summary of the work in the previous chapters; it must give insight into the value of your work, inform the readers of the impact of your work and should provide directions for future research on your report topic. This chapter allows you a chance to document your own opinions and insights while displaying ingenuity and imagination in choosing possible implementation applications or future directions of your own work.

# 

# References

|  |  |
| --- | --- |
| [1] | K. Sato, “Motion determining apparatus and storage medium having motion determining program stored thereon”. US Patent US 7424388 B2, 9 September 2008. |
| [2] | K. Ohta and K. shi, “Image processing apparatus and storage medium storing image processing program”. US Patent US 2007/0211027 A1, 13 September 2007. |
| [3] | Silicon Labs, “AN249 Human Interface Device Tutorial,” [Online]. Available: https://www.silabs.com/documents/public/application-notes/AN249.pdf. [Accessed 10 February 2019]. |
| [4] | Valve Corporation, “Github.com/ValveSoftware,” 2 February 2019. [Online]. Available: https://github.com/ValveSoftware/openvr. [Accessed 5 March 2019]. |
| [5] | S. R. Ellis, M. J. Young, B. D. Adelstein and S. M. Ehrlich, “DISCRIMINATION OF CHANGES OF LATENCY,” in *Human Factors and Ergonomics Society*, Houston, Texas, 1999. |
| [6] | Texas Instruments, “SimpleLink™ CC2650 wireless MCU LaunchPad™ Development Kit,” 2019. [Online]. Available: http://www.ti.com/tool/launchxl-cc2650#technicaldocuments. [Accessed 10 February 2019]. |
| [7] | P. Cheolhee and S. R. Theodore, “Short-Range Wireless Communications for Next-Generation Networks: UWB, 60 GHz Millimeter-Wave WPAN, and ZigBee,” *IEEE Wireless Communications,* vol. 14, no. 4, pp. 70-78, 2007. |
| [8] | Texas Instruments, “TI SimpleLink MCU Platform,” Texas Instruments, 2018. [Online]. Available: http://www.ti.com/wireless-connectivity/simplelink-solutions/overview/overview.html. [Accessed 2 December 2018]. |
| [9] | Analog Devices, “Data Sheet ADXL345 Rev.E,” 2015. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf. [Accessed 12 February 2019]. |
| [10] | Texas Instruments, “Adding Custom RTOS Task,” 1 October 2015. [Online]. Available: http://processors.wiki.ti.com/index.php/Adding\_Custom\_RTOS\_Task#Anatomy\_of\_an\_RTOS\_Task. [Accessed 12 March 2019]. |
| [11] | Texas Instruments Incorporated, “SPI.h File Reference,” 2017. [Online]. Available: http://dev.ti.com/tirex/content/simplelink\_cc26x2\_sdk\_1\_60\_00\_43/docs/tidrivers/doxygen/html/\_s\_p\_i\_8h.html. [Accessed 12 March 2019]. |
| [12] | Texas Instruments Incorporated, “I2C.h File Reference,” 2016. [Online]. Available: http://software-dl.ti.com/dsps/dsps\_public\_sw/sdo\_sb/targetcontent/tirtos/2\_20\_00\_06/exports/tirtos\_full\_2\_20\_00\_06/products/tidrivers\_full\_2\_20\_00\_08/docs/doxygen/html/\_i2\_c\_8h.html. [Accessed 12 March 2019]. |
| [13] | Texas Instruments, “TI Resouce Explorer - CC2650 LaunchPad,” [Online]. Available: http://dev.ti.com/tirex/#/DevTool/CC2650%20LaunchPad/?link=Development%20Tools%2FKits%20and%20Boards%2FCC2650%20LaunchPad. [Accessed 20 March 2019]. |
| [14] | O. Iason, K. Nikolaos and A. A. Antonis, “Efficient Model-based 3D Tracking of,” in *British Machine Vision Confrenece*, Dundee, 2011. |

# Appendix

## TI-RTOS Flashing LED Code:

/\* XDCtools Header files \*/

**#include** <xdc/std.h>

**#include** <xdc/runtime/System.h>

/\* BIOS Header files \*/

**#include** <ti/sysbios/BIOS.h>

**#include** <ti/sysbios/knl/Clock.h>

**#include** <ti/sysbios/knl/Task.h>

/\* TI-RTOS Header files \*/

**#include** <ti/drivers/PIN.h>

/\* Board Header files \*/

**#include** "Board.h"

**#define** TASKSTACKSIZE 512

Task\_Struct task0Struct;

Char task0Stack[TASKSTACKSIZE];

/\* Pin driver handle \*/

**static** PIN\_Handle MyPinHandle;

**static** PIN\_State MyPinState;

PIN\_Config MyPinTable[] = {

Board\_LED0 | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_LOW | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

Board\_LED1 | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_LOW | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

PIN\_TERMINATE

};

Void **Flashing**(UArg arg0, UArg arg1)

{

**while** (true) {

Task\_sleep((UInt)arg0);

**PIN\_setOutputValue**(MyPinHandle, Board\_LED0, !**PIN\_getOutputValue**(Board\_LED0));

**PIN\_setOutputValue**(MyPinHandle, Board\_LED1, !**PIN\_getOutputValue**(Board\_LED1));

}

}

**int** **main**(**void**)

{

Task\_Params taskParams;

Board\_initGeneral();

/\* Construct Task thread \*/

Task\_Params\_init(&taskParams);

taskParams.arg0 = 1000000 / Clock\_tickPeriod;

taskParams.stackSize = TASKSTACKSIZE;

taskParams.stack = &task0Stack;

Task\_construct(&task0Struct, (Task\_FuncPtr)Flashing, &taskParams, NULL);

/\* Open LED pins \*/

MyPinHandle = **PIN\_open**(&MyPinState, MyPinTable);

**PIN\_setOutputValue**(MyPinHandle, Board\_LED1, 1);

**PIN\_setOutputValue**(MyPinHandle, Board\_LED0, 0);

System\_printf("Starting the LED Flashing");

System\_flush();

BIOS\_start();

**return** (0);

}

## SPI Device ID Code:

**#include** <xdc/std.h>

**#include** <xdc/runtime/System.h>

/\* BIOS Header files \*/

**#include** <ti/sysbios/BIOS.h>

**#include** <ti/sysbios/knl/Task.h>

**#include** <ti/sysbios/knl/Clock.h>

**#include** <ti/drivers/PIN.h>

**#include** <ti/drivers/SPI.h>

**#include** <ti/drivers/GPIO.h>

**#include** <ti/drivers/Watchdog.h>

/\* Board Header files \*/

**#include** "Board.h"

**#define** TASKSTACKSIZE 512

Task\_Struct task0Struct;

Char task0Stack[TASKSTACKSIZE];

**void** **delay\_s**(**int** dly) {

**while** (dly > 0) {

\_\_delay\_cycles(48000000);

dly--;

}

}

**void** **delay\_ms**(**int** dly) {

**while** (dly > 0) {

\_\_delay\_cycles(48000);

dly--;

}

}

Char myTaskStack[1024];

Task\_Struct myTaskStruct;

Void **SPIFxn**(UArg arg0, UArg arg1) {

System\_printf("Start SPI task\n");

SPI\_Handle handle;

SPI\_Params spiParams;

SPI\_Transaction spiTransaction;

uint8\_t transmitBuffer[2];

uint8\_t recieveBuffer[2];

bool ret;

SPI\_Params\_init(&spiParams); //initialise spi params object

spiParams.transferMode = SPI\_MODE\_BLOCKING; //block thread until transaction complete

spiParams.transferTimeout = SPI\_WAIT\_FOREVER;//wait forever

spiParams.frameFormat = SPI\_POL1\_PHA1;//clock polarity 1 phase 1

spiParams.transferCallbackFxn = NULL; //no callback

spiParams.mode = SPI\_MASTER; //master mode

spiParams.bitRate = 400000; //400MHz

spiParams.dataSize = 8; //data size

handle = SPI\_open(0, &spiParams);//open spiHandle

transmitBuffer[0] = 0xAD;//write to power control

transmitBuffer[1] = 0x08;//turn device on

spiTransaction.count = 2;

spiTransaction.txBuf = transmitBuffer;

spiTransaction.rxBuf = recieveBuffer;

ret = SPI\_transfer(handle, &spiTransaction);

**if** (!ret) {

System\_printf("Unsuccessful SPI transfer");

}

delay\_s(3);

**while**(true){

transmitBuffer[0] = 0x80;//read from address 0x00 with read bit 0x80 (devID)

transmitBuffer[1] = 0x00;//fake request for interchange

ret = SPI\_transfer(handle, &spiTransaction);

**if** (!ret) {

System\_printf("Unsuccessful SPI transfer");

}

delay\_s(3);

System\_printf("Devid = %D", recieveBuffer[1]);

System\_flush();

delay\_s(3);

}

}

// ======== SETUP ========

Int **main**(){

System\_printf("Start Setup\n");

Board\_initGeneral();

Board\_initSPI();

/\* Configure task. \*/

Task\_Params taskParams;

Task\_Params\_init(&taskParams);

taskParams.stack = myTaskStack;

taskParams.stackSize = **sizeof**(myTaskStack);

Task\_construct(&myTaskStruct, SPIFxn, &taskParams, NULL);

System\_printf("End Setup\n");

BIOS\_start();

**return** (0);

}

## TI Packet Transmission Code

/\*

\* Copyright (c) 2015-2016, Texas Instruments Incorporated

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* \* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* \* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the distribution.

\*

\* \* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

\* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

\* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

\* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

\* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

\* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

\* OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

\* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR

\* OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

\* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

/\*\*\*\*\* Includes \*\*\*\*\*/

**#include** <stdlib.h>

**#include** <xdc/std.h>

**#include** <xdc/runtime/System.h>

**#include** <ti/sysbios/BIOS.h>

**#include** <ti/sysbios/knl/Task.h>

/\* Drivers \*/

**#include** <ti/drivers/rf/RF.h>

**#include** <ti/drivers/PIN.h>

/\* Board Header files \*/

**#include** "Board.h"

**#include** "smartrf\_settings/smartrf\_settings.h"

/\* Pin driver handle \*/

**static** PIN\_Handle ledPinHandle;

**static** PIN\_State ledPinState;

/\*

\* Application LED pin configuration table:

\* - All LEDs board LEDs are off.

\*/

PIN\_Config pinTable[] =

{

Board\_LED0 | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_LOW | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

Board\_LED1 | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_LOW | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

PIN\_TERMINATE

};

/\*\*\*\*\* Defines \*\*\*\*\*/

**#define** TX\_TASK\_STACK\_SIZE 1024

**#define** TX\_TASK\_PRIORITY 2

/\* Packet TX Configuration \*/

**#define** PAYLOAD\_LENGTH 30

**#define** PACKET\_INTERVAL (uint32\_t)(4000000\*0.5f) /\* Set packet interval to 500ms \*/

/\*\*\*\*\* Prototypes \*\*\*\*\*/

**static** **void** **txTaskFunction**(UArg arg0, UArg arg1);

/\*\*\*\*\* Variable declarations \*\*\*\*\*/

**static** Task\_Params txTaskParams;

Task\_Struct txTask; /\* not static so you can see in ROV \*/

**static** uint8\_t txTaskStack[TX\_TASK\_STACK\_SIZE];

**static** RF\_Object rfObject;

**static** RF\_Handle rfHandle;

uint32\_t time;

**static** uint8\_t packet[PAYLOAD\_LENGTH];

**static** uint16\_t seqNumber;

**static** PIN\_Handle pinHandle;

/\*\*\*\*\* Function definitions \*\*\*\*\*/

**void** **TxTask\_init**(PIN\_Handle inPinHandle)

{

pinHandle = inPinHandle;

Task\_Params\_init(&txTaskParams);

txTaskParams.stackSize = TX\_TASK\_STACK\_SIZE;

txTaskParams.priority = TX\_TASK\_PRIORITY;

txTaskParams.stack = &txTaskStack;

txTaskParams.arg0 = (UInt)1000000;

Task\_construct(&txTask, txTaskFunction, &txTaskParams, NULL);

}

**static** **void** **txTaskFunction**(UArg arg0, UArg arg1)

{

uint32\_t time;

uint8\_t test = 12;

RF\_Params rfParams;

**RF\_Params\_init**(&rfParams);

RF\_cmdPropTx.pktLen = PAYLOAD\_LENGTH;

RF\_cmdPropTx.pPkt = packet;

RF\_cmdPropTx.startTrigger.triggerType = TRIG\_ABSTIME;

RF\_cmdPropTx.startTrigger.pastTrig = 1;

RF\_cmdPropTx.startTime = 0;

/\* Request access to the radio \*/

rfHandle = **RF\_open**(&rfObject, &RF\_prop, (RF\_RadioSetup\*)&RF\_cmdPropRadioDivSetup, &rfParams);

/\* Set the frequency \*/

**RF\_postCmd**(rfHandle, (RF\_Op\*)&RF\_cmdFs, *RF\_PriorityNormal*, NULL, 0);

/\* Get current time \*/

time = **RF\_getCurrentTime**();

**while**(1)

{

/\* Create packet with incrementing sequence number and random payload \*/

packet[0] = (uint8\_t)(8);

packet[1] = (uint8\_t)(seqNumber++);

uint8\_t i;

**for** (i = 2; i < PAYLOAD\_LENGTH; i++)

{

packet[i] = test;

}

/\* Set absolute TX time to utilize automatic power management \*/

time += PACKET\_INTERVAL;

RF\_cmdPropTx.startTime = time;

/\* Send packet \*/

RF\_EventMask result = **RF\_runCmd**(rfHandle, (RF\_Op\*)&RF\_cmdPropTx, *RF\_PriorityNormal*, NULL, 0);

**if** (!(result & RF\_EventLastCmdDone))

{

/\* Error \*/

**while**(1);

}

**PIN\_setOutputValue**(pinHandle, Board\_LED0,!**PIN\_getOutputValue**(Board\_LED0));

}

}

/\*

\* ======== main ========

\*/

**int** **main**(**void**)

{

/\* Call board init functions. \*/

Board\_initGeneral();

/\* Open LED pins \*/

ledPinHandle = **PIN\_open**(&ledPinState, pinTable);

**if**(!ledPinHandle)

{

System\_abort("Error initializing board LED pins\n");

}

/\* Initialize task \*/

TxTask\_init(ledPinHandle);

/\* Start BIOS \*/

BIOS\_start();

**return** (0);

}

## TI Packet Reciever Code

/\*

\* Copyright (c) 2015-2016, Texas Instruments Incorporated

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* \* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* \* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the distribution.

\*

\* \* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

\* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

\* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

\* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

\* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

\* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

\* OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

\* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR

\* OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

\* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

/\*\*\*\*\* Includes \*\*\*\*\*/

**#include** <stdlib.h>

**#include** <xdc/std.h>

**#include** <xdc/cfg/global.h>

**#include** <xdc/runtime/System.h>

**#include** <ti/sysbios/BIOS.h>

**#include** <ti/sysbios/knl/Task.h>

/\* Drivers \*/

**#include** <ti/drivers/rf/RF.h>

**#include** <ti/drivers/PIN.h>

**#include** <driverlib/rf\_prop\_mailbox.h>

/\* Board Header files \*/

**#include** "Board.h"

**#include** "RFQueue.h"

**#include** "smartrf\_settings/smartrf\_settings.h"

**#include** <stdlib.h>

/\* Pin driver handle \*/

**static** PIN\_Handle ledPinHandle;

**static** PIN\_State ledPinState;

/\*

\* Application LED pin configuration table:

\* - All LEDs board LEDs are off.

\*/

PIN\_Config pinTable[] =

{

Board\_LED0 | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_LOW | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

Board\_LED1 | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_LOW | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

PIN\_TERMINATE

};

/\*\*\*\*\* Defines \*\*\*\*\*/

**#define** RX\_TASK\_STACK\_SIZE 1024

**#define** RX\_TASK\_PRIORITY 2

/\* Packet RX Configuration \*/

**#define** DATA\_ENTRY\_HEADER\_SIZE 8 /\* Constant header size of a Generic Data Entry \*/

**#define** MAX\_LENGTH 30 /\* Max length byte the radio will accept \*/

**#define** NUM\_DATA\_ENTRIES 2 /\* NOTE: Only two data entries supported at the moment \*/

**#define** NUM\_APPENDED\_BYTES 2 /\* The Data Entries data field will contain:

\* 1 Header byte (RF\_cmdPropRx.rxConf.bIncludeHdr = 0x1)

\* Max 30 payload bytes

\* 1 status byte (RF\_cmdPropRx.rxConf.bAppendStatus = 0x1) \*/

/\*\*\*\*\* Prototypes \*\*\*\*\*/

**static** **void** **rxTaskFunction**(UArg arg0, UArg arg1);

**static** **void** **callback**(RF\_Handle h, RF\_CmdHandle ch, RF\_EventMask e);

/\*\*\*\*\* Variable declarations \*\*\*\*\*/

**static** Task\_Params rxTaskParams;

Task\_Struct rxTask; /\* not static so you can see in ROV \*/

**static** uint8\_t rxTaskStack[RX\_TASK\_STACK\_SIZE];

**static** RF\_Object rfObject;

**static** RF\_Handle rfHandle;

/\* Buffer which contains all Data Entries for receiving data.

\* Pragmas are needed to make sure this buffer is 4 byte aligned (requirement from the RF Core) \*/

**#if** defined(\_\_TI\_COMPILER\_VERSION\_\_)

**#pragma** DATA\_ALIGN (rxDataEntryBuffer, 4);

**static** uint8\_t rxDataEntryBuffer[RF\_QUEUE\_DATA\_ENTRY\_BUFFER\_SIZE(NUM\_DATA\_ENTRIES,

MAX\_LENGTH,

NUM\_APPENDED\_BYTES)];

**#elif** defined(\_\_IAR\_SYSTEMS\_ICC\_\_)

**#pragma** data\_alignment = 4

**static** uint8\_t rxDataEntryBuffer[RF\_QUEUE\_DATA\_ENTRY\_BUFFER\_SIZE(NUM\_DATA\_ENTRIES,

MAX\_LENGTH,

NUM\_APPENDED\_BYTES)];

**#elif** defined(\_\_GNUC\_\_)

**static** uint8\_t rxDataEntryBuffer [RF\_QUEUE\_DATA\_ENTRY\_BUFFER\_SIZE(NUM\_DATA\_ENTRIES,

MAX\_LENGTH, NUM\_APPENDED\_BYTES)] **\_\_attribute\_\_** ((aligned (4)));

**#else**

**#error** This compiler is not supported.

**#endif**

/\* Receive dataQueue for RF Core to fill in data \*/

**static** dataQueue\_t dataQueue;

**static** rfc\_dataEntryGeneral\_t\* currentDataEntry;

**static** uint8\_t packetLength;

**static** uint8\_t\* packetDataPointer;

**static** PIN\_Handle pinHandle;

**static** uint8\_t packet[MAX\_LENGTH + NUM\_APPENDED\_BYTES - 1]; /\* The length byte is stored in a separate variable \*/

/\*\*\*\*\* Function definitions \*\*\*\*\*/

**void** **RxTask\_init**(PIN\_Handle ledPinHandle) {

pinHandle = ledPinHandle;

Task\_Params\_init(&rxTaskParams);

rxTaskParams.stackSize = RX\_TASK\_STACK\_SIZE;

rxTaskParams.priority = RX\_TASK\_PRIORITY;

rxTaskParams.stack = &rxTaskStack;

rxTaskParams.arg0 = (UInt)1000000;

Task\_construct(&rxTask, rxTaskFunction, &rxTaskParams, NULL);

}

**static** **void** **rxTaskFunction**(UArg arg0, UArg arg1)

{

RF\_Params rfParams;

**RF\_Params\_init**(&rfParams);

**if**( RFQueue\_defineQueue(&dataQueue,

rxDataEntryBuffer,

**sizeof**(rxDataEntryBuffer),

NUM\_DATA\_ENTRIES,

MAX\_LENGTH + NUM\_APPENDED\_BYTES))

{

/\* Failed to allocate space for all data entries \*/

**while**(1);

}

/\* Modify CMD\_PROP\_RX command for application needs \*/

RF\_cmdPropRx.pQueue = &dataQueue; /\* Set the Data Entity queue for received data \*/

RF\_cmdPropRx.rxConf.bAutoFlushIgnored = 1; /\* Discard ignored packets from Rx queue \*/

RF\_cmdPropRx.rxConf.bAutoFlushCrcErr = 1; /\* Discard packets with CRC error from Rx queue \*/

RF\_cmdPropRx.maxPktLen = MAX\_LENGTH; /\* Implement packet length filtering to avoid PROP\_ERROR\_RXBUF \*/

RF\_cmdPropRx.pktConf.bRepeatOk = 1;

RF\_cmdPropRx.pktConf.bRepeatNok = 1;

/\* Request access to the radio \*/

rfHandle = **RF\_open**(&rfObject, &RF\_prop, (RF\_RadioSetup\*)&RF\_cmdPropRadioDivSetup, &rfParams);

/\* Set the frequency \*/

**RF\_postCmd**(rfHandle, (RF\_Op\*)&RF\_cmdFs, *RF\_PriorityNormal*, NULL, 0);

/\* Enter RX mode and stay forever in RX \*/

**RF\_runCmd**(rfHandle, (RF\_Op\*)&RF\_cmdPropRx, *RF\_PriorityNormal*, &callback, IRQ\_RX\_ENTRY\_DONE);

**while**(1);

}

**void** **callback**(RF\_Handle h, RF\_CmdHandle ch, RF\_EventMask e)

{

**if** (e & RF\_EventRxEntryDone)

{

/\* Toggle pin to indicate RX \*/

**PIN\_setOutputValue**(pinHandle, Board\_LED1,!**PIN\_getOutputValue**(Board\_LED1));

/\* Get current unhandled data entry \*/

currentDataEntry = RFQueue\_getDataEntry();

/\* Handle the packet data, located at &currentDataEntry->data:

\* - Length is the first byte with the current configuration

\* - Data starts from the second byte \*/

packetLength = \*(uint8\_t\*)(&currentDataEntry->data);

packetDataPointer = (uint8\_t\*)(&currentDataEntry->data + 1);

System\_printf("Testing %d", \*packetDataPointer);

System\_flush();

/\* Copy the payload + the status byte to the packet variable \*/

**memcpy**(packet, packetDataPointer, (packetLength + 1));

RFQueue\_nextEntry();

}

}

/\*

\* ======== main ========

\*/

**int** **main**(**void**)

{

/\* Call board init functions. \*/

Board\_initGeneral();

System\_printf("Test\n");

System\_flush();

/\* Open LED pins \*/

ledPinHandle = **PIN\_open**(&ledPinState, pinTable);

**if**(!ledPinHandle)

{

System\_abort("Error initializing board LED pins\n");

}

/\* Initialize task \*/

RxTask\_init(ledPinHandle);

/\* Start BIOS \*/

BIOS\_start();

**return** (0);

}

# Glossary

If required.