Light Bucket

Final Report

Team Members :

Adan Hernandez

David Brown

Kassandra Flores

Oliver Bolosan

CECS Senior Project Design II

Instructor: Eric Hernandez

12/16/2016

Table of Contents

Introduction

Project Light Bucket allows any person to use a phone application to pick a celestial body and have their selection be sent to an onboard system that will calculate the appropriate altitude and azimuth for an alt-az mount .These calculated coordinates will then be used to automatically configure a table top telescope to point to the right coordinates in the sky.

Project Overview

Light Bucket is, at its core, a click-and-view telescope. Using a mobile application developed for android devices, the user can select a series of celestial objects to point the telescope at. The telescope will automatically rotate the scope and the base of the telescope toward the celestial body for easy viewing,

A standard table top telescope will be outfitted with two stepper motors to help position it towards a selected of celestial body that is viewable from the user's location. The Light Bucket android application will allow the user to interface with the telescope.

There are three main factors to this solution: user location, time of day, and Julian day from Julian day 2000. These factors play a pivotal roles into determining the location of our celestial body. They will also be responsible for maintaining the view of the object.

We will use a wireless Bluetooth communication to transmit the user data for positioning. The data that is obtained from the user will be used in a series of equations to determine the altitude and azimuth degree values that are needed when position the alt-az telescope.

Technology Overview

Stepper Motors

Stepper motors are motors that move in discrete steps. They have multiple coils that are organized in groups called "phases". By energizing each phase in sequence, the motor will rotate, one step at a time. With a computer controlled stepping you can achieve very precise positioning and/or speed control.

Bluetooth Communication

Bluetooth communication is wireless technology that allows data to be transferred between multiple devices over a short amount of distance. Devices connected over Bluetooth can alternate between slave

and master roles. Transmission of data is made using short-wavelength radio waves ranging from 2.4 to 2.485 GHZ

Typical modern day applications of Bluetooth: Bluetooth headphones, wireless controllers like the ones on Nintendo WII and PlayStation 3.replacement of RS-232 in GPS receivers, test equipment, bar code scanner.

Android Applications

Android mobile applications are written in Extensible Markup Language (XML) and Java. The layout, also called Activities, is defined by the XML. Application Program Interfaces (API) are also defined in the XML. We will be using the Android's GPS and the Google Maps API to obtain the user's latitude and longitude. The Java files are responsible for much of the event handling when the user interacts with the application.

Microcontroller

A microcontroller is a single integrated circuit that contains a central processing unit (CPU). RAM, ROM, and programmable input and output pins. They are typically designed for doing one specific task and having this task be defined in its ROM.

Microcontrollers are most notably used as embedded systems in larger machinery and can typically be found in the following systems: phones, engine control systems, toys and power tools.

3D Modeling

In 3D computer graphics, 3D modeling (or three-dimensional modeling) is the process of developing a mathematical representation of any three-dimensional surface of an object (either inanimate or living) via specialized software. The product is called a 3D model. It can be displayed as a two-dimensional image through a process called 3D rendering or used in a computer simulation of physical phenomena. The model can also be physically created using 3D printing devices.

The graphics software used for this project was Autodesk 123D.

Project Objectives

1. To be able to select any celestial body on the Android app
2. To be able to transmit the user's gps coordinates from the android app to the microcontroller via Bluetooth.

3. To be able to transmit which celestial body was selected on the android app to the microcontroller via Bluetooth
4. To have the microcontroller determine how much it needs to drive the stepper motors to have the telescope facing in the correct direction.
5. To have all components of the project work in the following order : 1) Have the user's gps location sent to the microcontroller, 2) Allow the user to select which celestial body they want to view on the android app and have that information sent to the microcontroller , 3) Have the microcontroller determine the necessary calculations needed to know how much the telescope must be moved, 4) have the microcontroller send the correct signaling to the stepper motor controller to make the telescope move to determined position it needs to be in.

Achieved

The following numbers in the format "#" refer to the above numbers in project objectives

1. Project Objective "2" was completed. We were able to receive the needed information from the developed android application that was needed to calculate altitude and azimuth degrees.
2. Project Object "4" was completed. The TM4C123GH6PM was able to use the received data given by the Bluetooth module and be able to convert it into the required altitude and azimuth degrees needed to move the telescope
3. Project Objective "5" was completed. The users GPS location was sent successfully to the microcontroller. The microcontroller was able to utilize the sent data and then was able to control the signaling needed on the stepper motor drivers to move the two attached stepper motors.

Uncomplete

The following numbers in the format "#" refer to the above numbers in project objectives

1. Project Objective "1" was incomplete. Due to mishaps that stalled production of the android app and Bluetooth communication between the TM4C123GH6PM, only the moon is able to be selected for viewing.
2. Project Objective "3" was incomplete. Since the user android application only allowed for the moon to be selected, there was no need in implementing a checker for which object was selected for viewing

Overall Project Completion

The Light Bucket project succeeded in positioning the telescope to the approximately calculated altitude and azimuth coordinates.

The android application to Bluetooth module worked accordingly; all the required data was transmitted. As well as the TM4C123H6PM was able to use this transmitted data to calculate the altitude and azimuth need for the alt-az telescope.

Overall, even though the project ended up not allowing for more than the moon to be selected, a proof of concept was achieved.

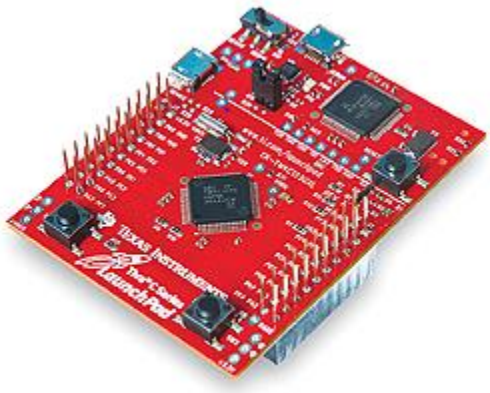
Hardware Design

Celestron Table Top Telescope

- Price: $69.95
- Model : Celestron Cosmos FirstScope Telescope
- Product Dimensions: 25.4 x 20.3 x 25.4 cm7
- Aperture: 76mm aperture Newtonian reflector
- Weight: 4.40 lbs

- Type of telescope mount : Altazimuth mount

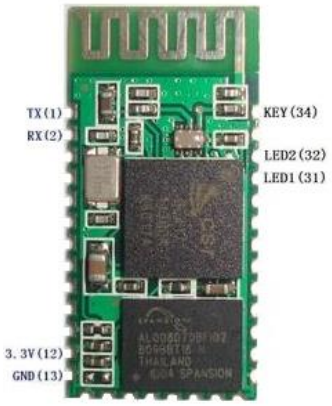
TM4C123GH6PM Microcontroller



- Price :$39.95
- Processor :32-bit ARM® Cortex™-M4 80MHz processor core
- Flash Ram: 256 kb
- SRAM: 32 kb
- 43 GPIO pins
- 12 ADC channels
- 8 UART channels
- 12 bit ADC Resolution
- Max Speed: 80 MHZ
- Input Voltage: 3.3v

Purpose: The TM4C123GH6PM will control the stepper motors to move the telescope to the required position and will receive data retrieved from an attached HC-05 Bluetooth module to determine how much to step the stepper motors.

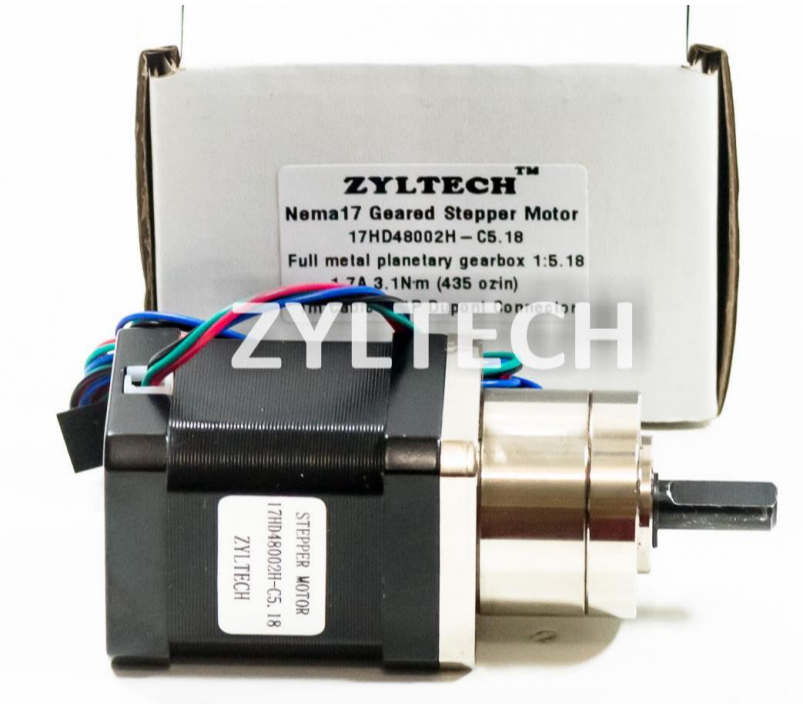
HC-05 Bluetooth Module



- Price :$3.35
- Input Voltage: 3.3v
- UART interface with programmable baud rate
- Data Bits: 8
- No Parity
- Stop Bit : 1
- Default Baud Rate: 38400
- Supported Baud Rates : 9600,19200,38400,57600,115200,230400,460800.

Purpose: This Bluetooth module will be connected to the TM4C123GH6PM microcontroller. Its purpose will be to be able to have data sent via Bluetooth from the android app to the microcontroller.

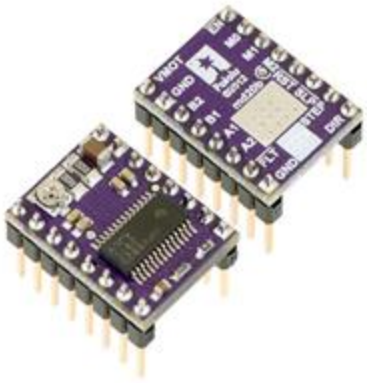
NEMA 17 Stepper Motors



- Price: $26.00
- Gear ratio: 1:5.18
- Step Angle: 1.8
- Step Accuracy: 5%
- Rated Voltage: 3.1 V
- Weight: 518 g

Purpose: To be used to move the telescope to a set location. One stepper motor will be used to control the altitude and one stepper motor will be used to control the right azimuth of the telescope. Stepper motors were selected instead of dc motors because the angle in which the telescope points will be more precise and more easily controlled than if the motors were DC motors

DRV8825 Stepper Motor Driver



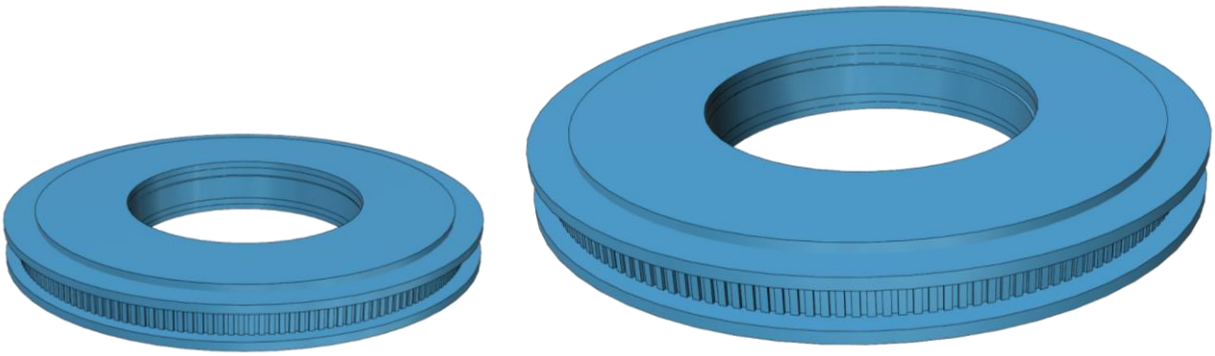
- Price: $8.95
- Manufacture : Polulu
- Operating Voltage : 8.2v
- Maximum Operating Voltage : 45v
- Maximum Current Phase : 2.2A
- Minimum Logic Voltage : 2.5v
- Maximum Logic Voltage : 5.25v
- Step Resolution : 1/2 , 1/4, 1/8, 1/16, 1/32

Purpose : To drive an individual stepper motor The step amount of each motor will be determined by the level logic present on the STEP pin and the Step resolution will be set by pins MO , M1, M2.

3D Gear Design

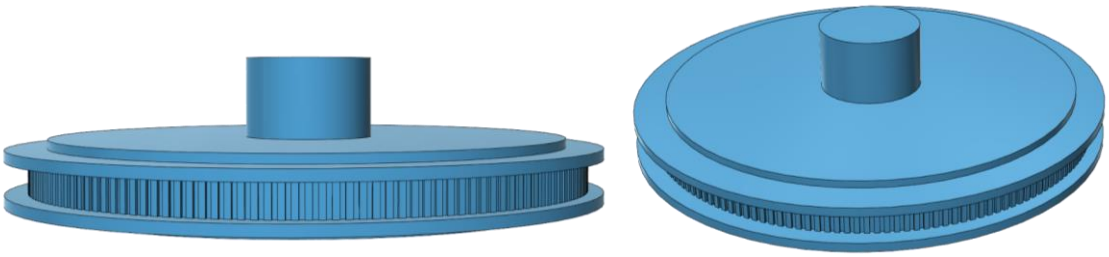
Gear 1

This Gear was designed to fit around a protruding piece on the side of the Celestron telescope. Friction holds this pulley in place and rotates the telescope when it is moved. This pulley was designed to work with a MXL belt.



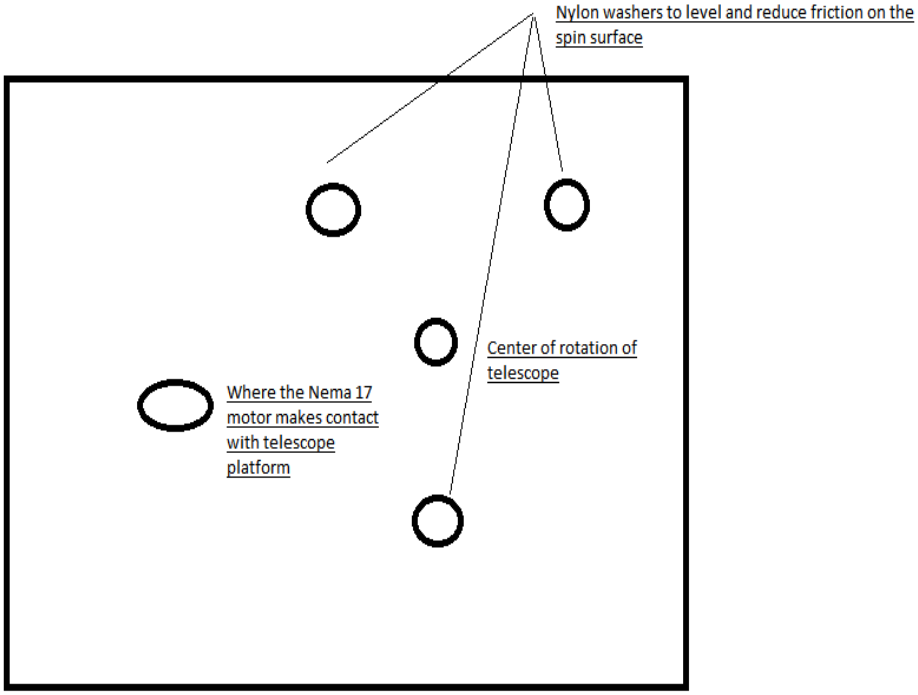
Gear 2

This pulley was designed to be driven by a nema 17 stepper motor.



Structural Design

Top View of Light Bucket Stand



Power Supply

Hardware	Voltage Requirements

TM4C123GH6PM Microcontroller	5 v
NEMA 17 Stepper Motor	3.1 v
HC-05 Bluetooth Module	3.3 v
DRV8833 Dual H-Bridge Motor Driver	8.2 v - 54 v

Power Source Connections

1. TM4C123GH6PM will receive voltage from 5v usb port
2. Voltage to the Bluetooth module HC-05 will come from 3.3v voltage pin on the TM4C123GH6PM microcontroller.
3. Voltage to each stepper motor driver will come from 12V Tenergy 2000mAh NiMh Battery Pack

Note: All stepper motor driver voltage and current where provided by in lab variable output current and voltage power supply boxes during production of project. Actual use of Tenergy battery was only stated and selected as an appropriate battery pack for the drivers based on specifications but never tested due to battery pack not arriving in time.

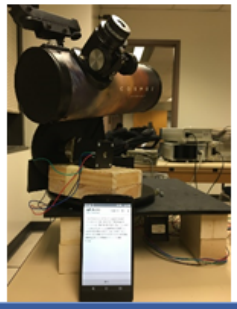
12V Tenergy 2000mAh NiMH Battery Pack

Components	10x AA NimH 2000mAh Cells
Connector	Bare Lead
Weight	10oz
Dimensions	50mm(width); 29mm (Height); 72mm (Length)

Bill of Materials

Light Bucket

Assembly Name :	
Assembly Number :	
Assembly Revision :	2
Approval Date :	
Part Count :	16
Total Cost :	$263.45



Part #	Part Name	Description	Qty	Units	Picture	Unit Cost	Cost
21024	Celestron Cosmos FirstScope Telescope	76 mm reflector optical tube Altazimuth mount	1	1		$ 69.95	$ 69.95
TM4C123GH6PM	TI Tiva Microcontroller		1	1		$ 39.95	$ 39.95
17HD48002H	Zyltech Nema17 Stepper Motor w/ planetary gearbox	Stepper Motor	2	2		$ 25.25	$ 50.50
HC-05	Bluetooth Module	Input Voltage: 3.3v UART interface with programmable baud rate	1	1		$ 3.35	$ 3.35
DRV8825	Dual H-Bridge Motor Driver	45 V maximum supply voltage. Adjustable current control	2	2		$ 2.94	$ 5.88
0	Custom 3D printed pulleys		2	2		$ 7.50	$ 15.00
	Seaboard High Density Polyethylene Sheet	1/4 inch Thick	1	1		$ 16.95	$ 16.95
	Wood Blocks		4	4		$ 5.00	$ 20.00
	MXL Timing Belt	Belt Pitch 2.032mm Width 6mm	1	1		$ 21.88	$ 21.88
	Tenergy Battery Pack	12V 2000mAh	1	1		$ 19.99	$ 19.99
							$ -
	Total		**16**				**$263.45**

Theory

The following are terms and equations that were used when calculating the altitude and azimuth degrees needed for the alt-az telescope.

Right Ascension: The angular distance measured eastward along the celestial equator from the vernal equinox to the hour circle of the point in question.

Declination: The angular distance of a point North or south of the celestial equator.

Latitude: The angular distance of a place north or south of the earth's equator.

Longitude : Thee angular distance of a place east or west of the meridian at Greenwich, England.

Day number(Julian Date): The Julian date is used to find the number of days since the fundamental epoch J2000. Calculations are as follows:

Julian date = ((UniversalTimeHour + (universalTimeInMinutes/60))/(24) + CurrentDayToBeginningOfYearInCurrentMonth + CurrentDayMonth + CurrentDaySinceJan2000(Jan1))

Local Sidereal time: The sidereal time is the right ascension of the points crossing the meridian. This is used as well as the time for all observers with the same geographical longitude. Equation is as follows:

*LST acronym for local sideral time

LST = (100.46 + 0.98567*JulianDate-Longitude + 15*(universalTimeInDecimalForm))

If the LST is greater value than 360, we subtract LST = LST - 360

Hour angle: The hour angle is one of the coordinates used in the equatorial coordinate system to give a direction of a point in the celestial sphere, in our case the moon. Value must be positive. If value is not positive, then we add 360, in the 0 to 360 range.

Hour Angle = Local Sidereal Time - Right Ascension

Pacific standard time to universal time: We live in southern California, so the time zone for us would be the pacific standard time. We need to convert this to universal time.

Universal Time: Universal time is a time standard for earth's rotation. Calculations are as follows:

UTC = pacificStandardTime + 7

If current time is not daylight savings, then equation changes as follows:

UTC = pacificStandardTime + 8

Altitude: The height of an object or point in relation to sea level or ground level.

Calculating Altitude from Declination , Latitude and Hour Angle :

sin(altitude) = sin(declination)*sin(latitude)+cos(declination)*cos(latitude)*cos(hour angle)

ALT = asin(altitude)
* ALT being the final value of altitude

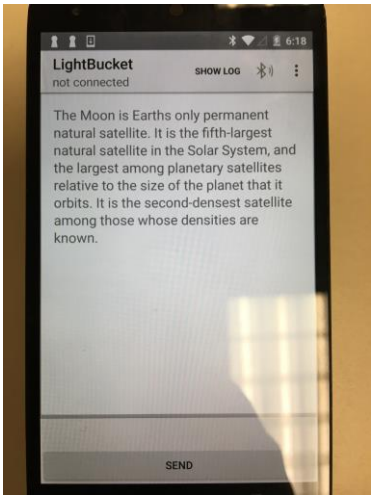
Azimuth: Geographical coordinates of the point on Earth, from which the object is seen directly in zenith.

Calculating Azimuth from Declination , Altitude and Latitude:

$$\cos(az0) = \frac{\sin(\text{Declination}) - \sin(\text{altitude})*\sin(\text{latitude})}{\cos(\text{altitude})*\cos(\text{latitude})}$$

az1 = acos(az0)

If sin(hour angle) is negative, then Azimuth = az1, otherwise
Azimuth = 360 - az1

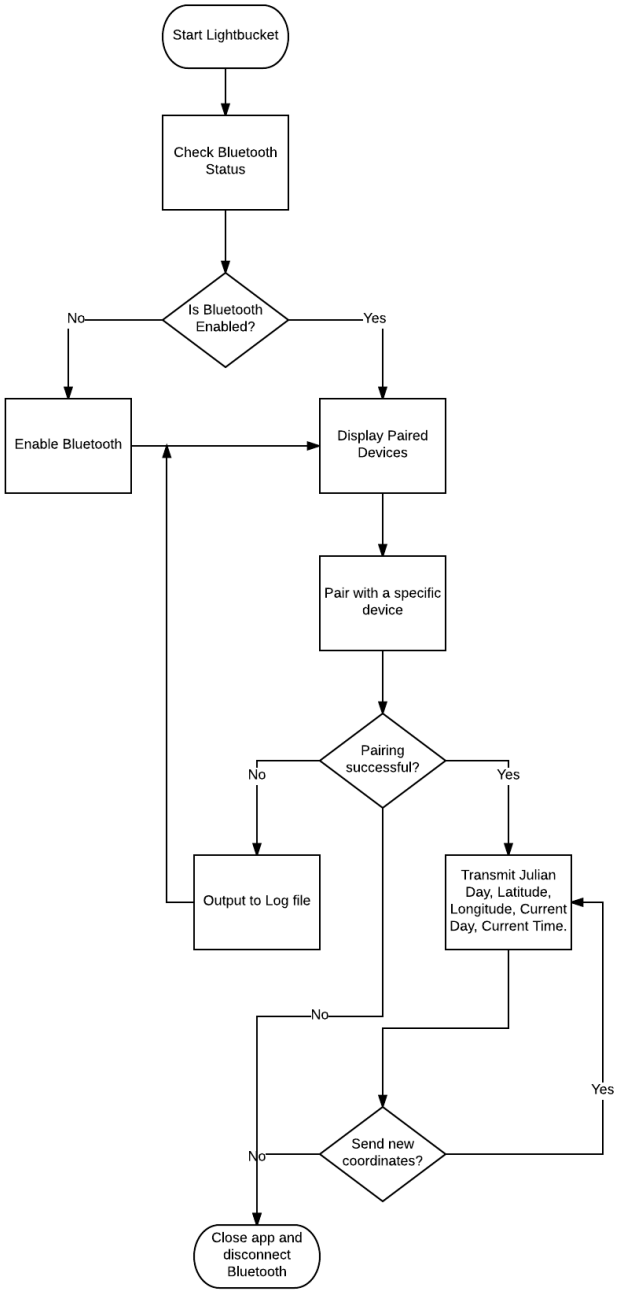


User Interface

The user interface for LightBucket is an Android application that can connect to the telescope's on-board Bluetooth module and transmit data to the

telescope. The data string consists of the julian day, latitude, longitude, current date, and current time. The on-board microcontroller would then take that received data and compute the Alt-Az coordinates. This would then result in pivoting the telescope toward the moon ready for easy-viewing.

User Interface Software Flow



User Interface Source Code

```
//User Interface Source Code
```

```
import android.app.ActionBar;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentActivity;
import android.text.TextUtils;
import android.view.KeyEvent;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.view.inputmethod.EditorInfo;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import com.example.android.common.logger.Log;
```

```
import java.util.Date;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.GregorianCalendar;



/**

* This fragment controls Bluetooth to communicate with other devices.

*/

public class BluetoothChatFragment extends Fragment {


   private static final String TAG = "BluetoothChatFragment";


   // Intent request codes

   private static final int REQUEST_CONNECT_DEVICE_SECURE = 1;

   private static final int REQUEST_CONNECT_DEVICE_INSECURE = 2;

   private static final int REQUEST_ENABLE_BT = 3;


   private Button mSendButton;


   /**

       * Name of the connected device

       */

   private String mConnectedDeviceName = null;


   /**

       * String buffer for outgoing messages

       */
```

```
private StringBuffer mOutStringBuffer;


/**
    * Local Bluetooth adapter
    */
private BluetoothAdapter mBluetoothAdapter = null;


/**
    * Member object for the chat services
    */
private BluetoothChatService mChatService = null;


@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);
    // Get local Bluetooth adapter
    mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();


    // If the adapter is null, then Bluetooth is not supported
    if (mBluetoothAdapter == null)
    {
    FragmentActivity activity = getActivity();
    Toast.makeText(activity, "Bluetooth is not available", Toast.LENGTH_LONG).show();
    activity.finish();
    }
}
```

```
@Override
public void onStart()
{
    super.onStart();
    // If BT is not on, request that it be enabled.
    // setupChat() will then be called during onActivityResult
    if (!mBluetoothAdapter.isEnabled())
    {
    Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
    // Otherwise, setup the chat session
    }
    else if (mChatService == null)
    {
    setupChat();
    }
}

@Override
public void onDestroy()
{
    super.onDestroy();
    if (mChatService != null)
    {
        mChatService.stop();
    }
```

```
}

@Override
public void onResume()
{
    super.onResume();

    // Performing this check in onResume() covers the case in which BT was
    // not enabled during onStart(), so we were paused to enable it...
    // onResume() will be called when ACTION_REQUEST_ENABLE activity returns.
    if (mChatService != null)
    {
    // Only if the state is STATE_NONE, do we know that we haven't started already
    if (mChatService.getState() == BluetoothChatService.STATE_NONE)
    {
            // Start the Bluetooth chat services
             mChatService.start();
    }
    }
}

@Override
public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
                         @Nullable Bundle savedInstanceState)
{
    return inflater.inflate(R.layout.fragment_bluetooth_chat, container, false);
}
```

```
    @Override
    public void onViewCreated(View view, @Nullable Bundle savedInstanceState)
    {
        //mConversationView = (ListView) view.findViewById(R.id.in);
        //mOutEditText = (EditText) view.findViewById(R.id.edit_text_out);
        mSendButton = (Button) view.findViewById(R.id.button_send);
    }

    /**
        * Set up the UI and background operations for chat.
        */
    private void setupChat()
    {
        Log.d(TAG, "setupChat()");

        // Initialize the array adapter for the conversation thread
        //mConversationArrayAdapter = new ArrayAdapter<String>(getActivity(),
R.layout.message);

        //mConversationView.setAdapter(mConversationArrayAdapter);

        // Initialize the compose field with a listener for the return key
        //mOutEditText.setOnEditorActionListener(mWriteListener);

        // Initialize the send button with a listener that for click events
        mSendButton.setOnClickListener(new View.OnClickListener()
        {
        public void onClick(View v)
```

```
        {
                // Send a message using content of the edit text widget
                View view = getView();
                if (null != view)
                {
                        //TextView textView = (TextView)
view.findViewById(R.id.edit_text_out);
                        //String message = textView.getText().toString();
                        //sendMessage(message);
                     sendMessage();
                }
            }
        });

        // Initialize the BluetoothChatService to perform bluetooth connections
        mChatService = new BluetoothChatService(getActivity(), mHandler);

        // Initialize the buffer for outgoing messages
        mOutStringBuffer = new StringBuffer("");
    }

    /**
        * Makes this device discoverable.
        */
    private void ensureDiscoverable()
    {
        if (mBluetoothAdapter.getScanMode() !=
                BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE)
```

```
        {
        Intent discoverableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
            discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
            startActivity(discoverableIntent);
        }
    }

    /**
        * Sends a message.
        *
        //* @param message A string of text to send.
        */
    private void sendMessage()
    {
        float Day = 0;
        GregorianCalendar gc = new GregorianCalendar();
        int year = gc.get(Calendar.YEAR);
        int month = gc.get(Calendar.MONTH);
        int day = gc.get(Calendar.DAY_OF_MONTH);
        Day = julianDay(year, month, day) - 2451543.5f;
        // Check that we're actually connected before trying anything
        if (mChatService.getState() != BluetoothChatService.STATE_CONNECTED)
        {
        Toast.makeText(getActivity(), R.string.not_connected, Toast.LENGTH_SHORT).show();
        return;
        }
        String latitude = "33.7838";
```

```
        String longitude = "118.1141";
        String jd = Float.toString(Day);
        String currentDateTimeString = new SimpleDateFormat("yyyy-MM-
dd'x'HH:mm:ss").format(new Date());
        // Get the message bytes and tell the BluetoothService to write
        byte[] send = (jd + "x" + latitude + "x" + longitude + "x" + currentDateTimeString +
"x" + "\r").getBytes();
        mChatService.write(send);

        // Reset out string buffer to zero and clear the edit text field
        mOutStringBuffer.setLength(0);
    }

    /**
        * The action listener for the EditText widget, to listen for the return key
        */
    private TextView.OnEditorActionListener mWriteListener
        = new TextView.OnEditorActionListener()
    {
        public boolean onEditorAction(TextView view, int actionId, KeyEvent event)
        {
        // If the action is a key-up event on the return key, send the message
        if (actionId == EditorInfo.IME_NULL && event.getAction() == KeyEvent.ACTION_UP)
        {
                String message = view.getText().toString();
               //sendMessage(message);
                sendMessage();
        }
        return true;
```

```
    }
};


/**
    * Updates the status on the action bar.
    *
    * @param resId a string resource ID
    */
private void setStatus(int resId)
{
    FragmentActivity activity = getActivity();
    if (null == activity)
    {
    return;
    }
    final ActionBar actionBar = activity.getActionBar();
    if (null == actionBar)
    {
        return;
    }
    actionBar.setSubtitle(resId);
}


/**
    * Updates the status on the action bar.
    *
    * @param subTitle status
    */
```

```
    private void setStatus(CharSequence subTitle)
    {
        FragmentActivity activity = getActivity();
        if (null == activity) {
        return;
        }
        final ActionBar actionBar = activity.getActionBar();
        if (null == actionBar)
        {
        return;
        }
        actionBar.setSubtitle(subTitle);
    }

    /**
        * The Handler that gets information back from the Bluetooth Service
        */
    private final Handler mHandler = new Handler()
    {
        @Override
        public void handleMessage(Message msg) {
            FragmentActivity activity = getActivity();
        switch (msg.what) {
                case Constants.MESSAGE_STATE_CHANGE:
                        switch (msg.arg1) {
                          case BluetoothChatService.STATE_CONNECTED:
                              setStatus(getString(R.string.title_connected_to,
mConnectedDeviceName));
```

```
                    //mConversationArrayAdapter.clear();
                    break;
                case BluetoothChatService.STATE_CONNECTING:
                    setStatus(R.string.title_connecting);
                    break;
                case BluetoothChatService.STATE_LISTEN:
                case BluetoothChatService.STATE_NONE:
                    setStatus(R.string.title_not_connected);
                    break;
                }
                break;
            case Constants.MESSAGE_WRITE:
                byte[] writeBuf = (byte[]) msg.obj;
                // construct a string from the buffer
                String writeMessage = new String(writeBuf);
                //mConversationArrayAdapter.add("Me:  " + writeMessage);
                break;
            case Constants.MESSAGE_READ:
                byte[] readBuf = (byte[]) msg.obj;
                // construct a string from the valid bytes in the buffer
                String readMessage = new String(readBuf, 0, msg.arg1);
                //mConversationArrayAdapter.add(mConnectedDeviceName + ":  " +
readMessage);
                break;
            case Constants.MESSAGE_DEVICE_NAME:
                // save the connected device's name
                mConnectedDeviceName = msg.getData().getString(Constants.DEVICE_NAME);
                if (null != activity)
```

```
                    {
                      Toast.makeText(activity, "Connected to "
                            + mConnectedDeviceName, Toast.LENGTH_SHORT).show();
                    }
                    break;
            case Constants.MESSAGE_TOAST:
                    if (null != activity)
                    {
                      Toast.makeText(activity, msg.getData().getString(Constants.TOAST),
                            Toast.LENGTH_SHORT).show();
                    }
                    break;
    }
    }
};


public void onActivityResult(int requestCode, int resultCode, Intent data)
{
    switch (requestCode)
    {
    case REQUEST_CONNECT_DEVICE_SECURE:
            // When DeviceListActivity returns with a device to connect
            if (resultCode == Activity.RESULT_OK)
            {
                 connectDevice(data, true);
            }
            break;
    case REQUEST_CONNECT_DEVICE_INSECURE:
```

```
            // When DeviceListActivity returns with a device to connect
            if (resultCode == Activity.RESULT_OK)
            {
                 connectDevice(data, false);
            }
            break;
        case REQUEST_ENABLE_BT:
            // When the request to enable Bluetooth returns
            if (resultCode == Activity.RESULT_OK)
            {
                   // Bluetooth is now enabled, so set up a chat session
                 setupChat();
            }
            else
            {
                   // User did not enable Bluetooth or an error occurred
                 Log.d(TAG, "BT not enabled");
                 Toast.makeText(getActivity(), R.string.bt_not_enabled_leaving,
                         Toast.LENGTH_SHORT).show();
                 getActivity().finish();
            }
        }
    }

    /**
        * Establish connection with other divice
        *
```

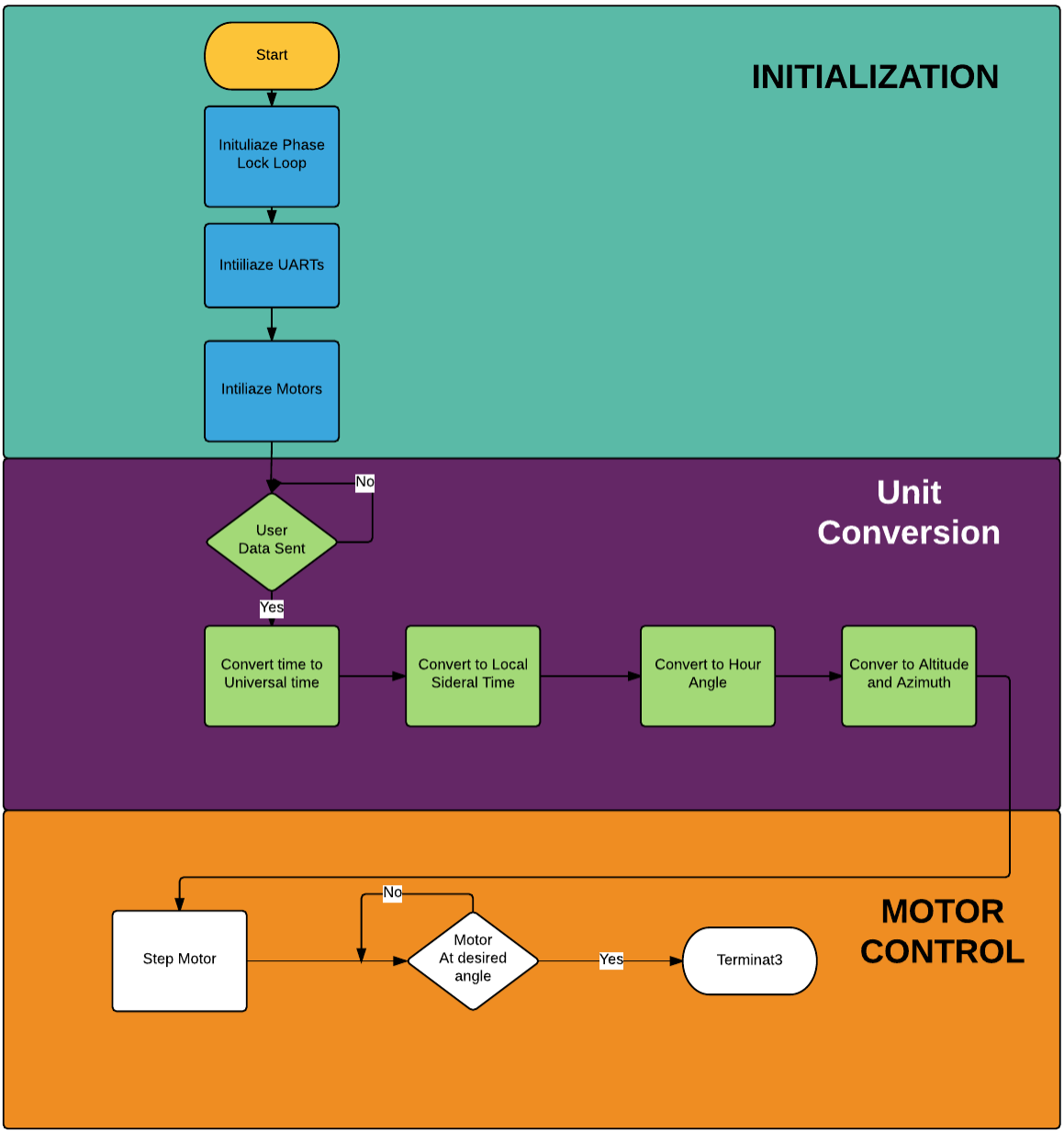
```
        * @param data   An {@link Intent} with {@link
DeviceListActivity#EXTRA_DEVICE_ADDRESS} extra.
        * @param secure Socket Security type - Secure (true) , Insecure (false)
        */
    private void connectDevice(Intent data, boolean secure)
    {
        // Get the device MAC address
        String address = data.getExtras()
                .getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
        // Get the BluetoothDevice object
        BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
        // Attempt to connect to the device
        mChatService.connect(device, secure);
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
    {
        inflater.inflate(R.menu.bluetooth_chat, menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        switch (item.getItemId())
        {
        case R.id.secure_connect_scan:
        {
```

```
            // Launch the DeviceListActivity to see devices and do scan
             Intent serverIntent = new Intent(getActivity(), DeviceListActivity.class);
             startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE_SECURE);
            return true;
        }
        case R.id.insecure_connect_scan:
        {
            // Launch the DeviceListActivity to see devices and do scan
             Intent serverIntent = new Intent(getActivity(), DeviceListActivity.class);
             startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE_INSECURE);
            return true;
        }
        case R.id.discoverable:
        {
            // Ensure this device is discoverable by others
             ensureDiscoverable();
            return true;
        }
        }
        return false;
    }
    public static float julianDay(int year, int month, int day)
    {
        float a = (14 - month) / 12;
        float y = year + 4800 - a;
        float m = month + 12 * a - 3;
        float jdn = day + (153 * m + 2)/5 + 365*y + y/4 - y/100 + y/400 - 32045;
        return jdn + 29.5f;
```

```
    }
}
```

Software

Software Flow



Source Code

StarLord.h

```
#include <math.h>
```

```
////////////////////////////////////////////////////////
// Calculating RA ( Right Ascension in Degrees)
// Step 1 : convert to Decimal Hrs
// Step 2 : multilply those decimal hrs by 15 to convert to degrees
//
// (Using presentation day as example)
// ex)RA : 5h 42m 48s
//      time to decimal hrs
//      5 + 42/60 + 48/3600 = 5.713333333
//
//      decimal hrs to degrees
//      5.713333333*15 = 85.7 <-- Right Ascension in degrees
////////////////////////////////////////////////////////////////////////////
// UNITS FOR DAY DEC 13, 2016 8:30 pm --> CHANGE IF USING ON A DIFFRENT DAY<--
#define RA  86.217//69.7750// Right Ascension
#define DEC 16.43;// Declination
////////////////////////////////////////////////////////////////////////////
//////////////////////////
// Local Sideral TIme
//////////////////////////
float LST(float longtitude , float jday, float ut_dec);
//////////////////////////
// Azimuth Coordinates
//////////////////////////
float AZ ( float HA  , float DECr, float LAT, float ALT);
//////////////////////////
// Altitude Coordinates
//////////////////////////
float ALT ( float HA, float DECr , float LAT);
//////////////////////////
// Hour Angle
//////////////////////////
float HS ( float RAr, float LST);
//////////////////////////
// Universal Time Converter
//////////////////////////
float UTC ( float hrs , float mins , int dst);
```

```
////////////////////////////
// Julian Day Converter
////////////////////////////
float J2000Days( int ut_hr , int ut_mr, float MDecr , float Doffsetr , float jyearr);
```

StarLord.c

```
//////////////////////////////////////////////////////////////////////////
// Moon
// Date of Coordinates: 12/13/2016
// Right Ascension : 5:42:48 (hh:mm:ss) (85.7 degrees)
// Declination         : 18° 11'53''
//////////////////////////////////////////////////////////////////////////
// User  Longtitude and LAatitude ( provided by app)
// Latitude   (approx.) : 33.783764
```

```
// Longtitude (approx.) : -118.110426
//
// time must be in civilian time , not military time
// UTC = PST + 7 or if not dalyight savings ime UTC = PST + 8
//
//
//
////////////////////////////////////////////////////////////////////////
// Alitude (alt) : the up and down of the scope
// Azimuth (az)  : the left and right of the base of the telescope mount
////////////////////////////////////////////////////////////////////////
#include "StarLord.h"



float n ;

//////////////////////
// Global Variables
//////////////////////
float DJD200 ; // Days from Julian Date

////////////////////////////////////////////////
// Calculate Days since Julian year 2000
// Input : Universal time hours (ut_hr)
//                              Universal time minutes (ut_mr)
//      Current days since begning of year to first day of month ( MDecr)
//        Current days since beginning of the month (Doffsetr)
//      Current Days since the first day of year 2000 ( Jan 1)
// Output : Days that have passed since Jan 1, 2000
////////////////////////////////////////////////////
float J2000Days( int ut_hr , int ut_mr, float MDecr , float Doffsetr , float jyearr)
{
 return ( ((ut_hr+(ut_mr/60))/24) + MDecr+ Doffsetr+jyearr);
}


////////////////////////////////////////////////////////////////////////////
// Universal Time Converter to decimal    ( ONLY PST )
```

```
// Input : hrs - current hour (STD)
//                            mins - current minutes (STD)
//                            dst  - (1) if daylight savings time , (0) if not daylight savings
time
// Output : Universal time in decimal form
////////////////////////////////////////////////////////////////////////////////
float UTC ( float hrs , float mins , int dst)
{
        float u_hrs  = hrs;

         u_hrs = u_hrs + 8 ;// to convert to Universal time
        if ( u_hrs > 24 )
        {
                u_hrs  = u_hrs -24;
        }
                return  (u_hrs+(float)(mins/60));


}

////////////////////////////////////////////////////////////////////////////////
// LOCAL SIDREAL TIME : ( best explanation for it)
// Suppose you have a sunny morning. Put a stick in the ground, and watch the shadow.
// The shadow will get shorter and shorter - and then start to get longer and longer.
// The time corresponding to the shortest shadow is your local noon.
// We reckon a Solar day as (roughly) the mean time between two local noons,
// and we call this 24 hours of time.
// Input :   longtitude -- current locations longtitude
//           jday      -- Julian days since JD2000
//           ut_dec    -- Current time in Universal time in decimal form
// Output: local sideral time for the spicified location and , time and jday
////////////////////////////////////////////////////////////////////////////////
float LST(float longtitude , float jday, float ut_dec)
{
        n =(100.46 + 0.98567*jday-longtitude + 15*(ut_dec));
        if ( n > 360 )
        {
        do
        {
                n = n- 360;
```

```
        } while ( n > 360);
        }
        return (n <0) ? (n + 360.0) : n;
}
////////////////////////////////////////////////////////////////////////////
// Hour Angle:
// used in the equatorial coordinate system to give the direction of a
// point on the celestial sphere.
//
// HR must be positive
// If HR is negative , add 360 tobe in the 0 to 360 range.
// RA must be in degrees
// Input : LST - Local Sidreal time
//      RAr  - Right Ascnesion
// Output: Hour Angle
////////////////////////////////////////////////////////////////////////////

float HS ( float RAr, float LST)
{
        float HS  = LST - RAr;
        if ( HS < 0)
        {
                return (HS + 360.0);
        }
        else
        {
                return HS;
        }
}

////////////////////////////////////////////////////////////////////////////
// HOUR and DECLINATION to ALT
// Calculate the Altitude from
// Hour angle and Declinaton
// Input : Hour Angle  - HA
//         Declination - DEC
//                           Latitude      - LAT
// Output : Altitude
////////////////////////////////////////////////////////////////////////////
```

```
float ALT( float HA, float DECr , float LAT)
{
  float ALT;
        ALT  =  (sinf(DECr * 0.0174533)*sinf(LAT* 0.0174533))+(cosf(DECr*
0.0174533)*(cosf(LAT*0.0174533)*cosf(HA*0.0174533)));
        // input is always in radians , output of sinf is in radians
        ALT  =  asinf(ALT)/ 0.0174533; // division by  0.017533 gets the radians back to degrees
        return ALT;
}
////////////////////////////////////////////////////////////////////////////////
// HOUR and DECLINATION to AZ
// Calculate th e Azimuth from Hour angle and declination
// Hour angle and Declinaton
// Input : Hour Angle  - HA
//         Declination - DEC
//                           Latitude      - LAT
//                           Altitude      - ALT
// Output : Azimuth
////////////////////////////////////////////////////////////////////////////////
float AZ ( float HA  , float DECr, float LAT, float ALT)
{
        float AZ;
        AZ  = (sinf(DECr * 0.0174533) - (sinf(ALT * 0.0174533)*sinf(LAT * 0.0174533)))/ (cosf(ALT
* 0.0174533)*cosf(LAT * 0.0174533));

        if( sinf(HA * 0.0174533) <0 )
        {
                return acosf(AZ)/ 0.0174533;
        }
        else
        {
                return (360.0-((acosf(AZ))/0.0174533));
        }

}
```

Motor.h

```
////////////////////////////////////////////////////////////////////////////////
//                                          DRV8825 BREAKOUT BOARD PIN OUT
//
//
// 1) EN:  Name: Enable
//                  (Negative Logic)
//                    HIGH-Disable
//                    LOW-Enable*
//
// 2) MO,M1,M2 : Step Resolution Setting
//
// 3) RESET: Enable or Disable H Bridge Output
//                    LOW-Disbale*
//                    HIGH-Enable
//
// 4) SLEEP: ENable/Disable low power sleep mode
```

```
//                    LOW-Disbale*
//                    HIGH-Enable
// 5) STEP: LOW -> HIGH, move one step
//
// 6) LOW / HIGH switches direction
//
// 7) VMOT: Motor Power (12- 24v)
//
// 8) FAULT : unused (?)
////////////////////////////////////////////////////////////


////////////////////////////////////////////////////
//             Motor Initillization
// -- Setup pins for motor connections
// Input  : none
// Output : none
////////////////////////////////////////////////////
void MotorInit(void);


////////////////////////////////////////////////////
//             STEP PINS INITILIZATION
// - 3 pins needed to select step resolution
// - all 3 pins need to be output pins
void MotorStepInit(void);



////////////////////////////////////////////////////
//               STEP RESOLUTION
// DV8885 Breakout Board step resoluation pins :
//             M0 ,  M1, M2
//
// Step Angle Logic:
// M0   |  M1  | M02  | Resolution
// --------------------------------
// Low  |  Low | Low  | Full step
// High |  Low | Low  | Half step
```

```
// Low  | High | Low  | 1/4 step
// High | High | Low  | 1/8 step
// Low  |  Low | High | 1/16 step
// High |  Low | High | 1/32 step
// Low  | High | High | 1/32 step
// High | High | High | 1/32 step
//////////////////////////////////////////////////
void MotorStepResolution(int);



//////////////////////////////////////////////////
//      MOTOR DIRECTION
//////////////////////////////////////////////////
void MotorDirection(int);



//////////////////////////////////////////////////
//      MOVING OF MOTOR
// - signal the DRV8825 to begin motor stepping
//////////////////////////////////////////////////
void MotorMoveB(void);
void MotorMoveA(void);
int movescopeamount(float );//
//////////////////////////////////////////////////
//      converting degrees to how many time to
//      0scillate pin
//////////////////////////////////////////////////
int moveamount(float);

//////////////////////////////////////////////////
//      ENABLE DRV8825
// - set enable
// - disable reset
// - disable sleep
//////////////////////////////////////////////////
void EnableMotor(void);
```

```
void PortEInit(void);
////////////////////////////////////////////////
//                    Intialize Systic
//- used to st up the timing per step
////////////////////////////////////////////////
void SysTick_Init(void);
void SysTick_Wait(unsigned long delay);
```

Motor.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
/////////////////////////////////////////
//                                  PORT B DECLARATIONS //
/////////////////////////////////////////
//#define GPIO_PORTB_DATA_BITS_R  (*((volatile unsigned long *)0x40005000))
//#define GPIO_PORTB_DATA_R          (*((volatile unsigned long *)0x400053FC))
//#define GPIO_PORTB_DIR_R           (*((volatile unsigned long *)0x40005400))
//#define GPIO_PORTB_IM_R            (*((volatile unsigned long *)0x40005410))
//#define GPIO_PORTB_RIS_R        (*((volatile unsigned long *)0x40005414))
//#define GPIO_PORTB_MIS_R           (*((volatile unsigned long *)0x40005418))
//#define GPIO_PORTB_ICR_R           (*((volatile unsigned long *)0x4000541C))
//#define GPIO_PORTB_AFSEL_R         (*((volatile unsigned long *)0x40005420))
//#define GPIO_PORTB_DEN_R           (*((volatile unsigned long *)0x4000551C))
//#define GPIO_PORTB_LOCK_R          (*((volatile unsigned long *)0x40005520))
//#define GPIO_PORTB_CR_R            (*((volatile unsigned long *)0x40005524))
//#define GPIO_PORTB_AMSEL_R         (*((volatile unsigned long *)0x40005528))
//#define GPIO_PORTB_PCTL_R          (*((volatile unsigned long *)0x4000552C))
//#define GPIO_PORTB_PUR_R           (*((volatile unsigned long *)0x40005510))
/////////////////////////////////////////
//                    PORT D DECLARATIONS //
```

```
//////////////////////////////////////////
//#define GPIO_PORTD_DATA_R           (*((volatile unsigned long *)0x400073FC))
//#define GPIO_PORTD_DIR_R            (*((volatile unsigned long *)0x40007400))
//#define GPIO_PORTD_AFSEL_R          (*((volatile unsigned long *)0x40007420))
//#define GPIO_PORTD_DEN_R            (*((volatile unsigned long *)0x4000751C))
//#define GPIO_PORTD_CR_R             (*((volatile unsigned long *)0x40007524))
//#define GPIO_PORTD_AMSEL_R          (*((volatile unsigned long *)0x40007528))
//#define GPIO_PORTD_PCTL_R           (*((volatile unsigned long *)0x4000752C))
//#define GPIO_PORTD_PUR_R            (*((volatile unsigned long *)0x40007510))


#define NVIC_ST_CTRL_R        (*((volatile unsigned long *)0xE000E010))
#define NVIC_ST_RELOAD_R      (*((volatile unsigned long *)0xE000E014))
#define NVIC_ST_CURRENT_R     (*((volatile unsigned long *)0xE000E018))
#define NVIC_ST_CTRL_COUNT    0x00010000  // Count flag
#define NVIC_ST_CTRL_CLK_SRC  0x00000004  // Clock Source
#define NVIC_ST_CTRL_INTEN    0x00000002  // Interrupt enable
#define NVIC_ST_CTRL_ENABLE   0x00000001  // Counter mode
#define NVIC_ST_RELOAD_M      0x00FFFFFF  // Counter load value
//////////////////////////////////////////
//                    CLOCK DECLARATIONS            //
//////////////////////////////////////////
#define SYSCTL_RCGC1_R        (*((volatile unsigned long *)0x400FE104))
#define SYSCTL_RCGC2_R        (*((volatile unsigned long *)0x400FE108))
#define SYSCTL_RCGC1_UART0    0x00000001  // UART0 Clock Gating Control
#define SYSCTL_RCGC2_GPIOB    0x00000002  // Port B Clock Gating Control
#define SYSCTL_PRGPIO_R       (*((volatile unsigned long *)0x400FEA08))
//////////////////////////////////////////
//             STEP RESOLUTION                                    //
//////////////////////////////////////////
#define FULLSTEP           1 //  1
#define HALFSTEP           2 //  1/2
#define FOUTHSTEP          3 //  1/4
#define EIGHTSTEP          4 //  1/8
#define SIXTEENTHSTEP      5 //  1/16
#define THIRTYTWOSTEP_v0   6 //  1/32
#define THIRTYTWOSTEP_v1   7 //  1/32
#define THIRTYTWOSTEP_v2   8 //  1/32
```

```
#define FORWARD 1
#define DOWWARD 0

#define STEPA_PIN   (*((volatile unsigned long *)0x40005004))
#define DIR_PIN    (*((volatile unsigned long *)0x40005008))
#define SLEEP_PIN  (*((volatile unsigned long *)0x40005010))
#define STEPB_PIN   (*((volatile unsigned long *)0x40005020))
#define RESET_PIN  (*((volatile unsigned long *)0x40005200))

#define MO_PIN     (*((volatile unsigned long *)0x40007004))
#define M1_PIN     (*((volatile unsigned long *)0x40007008))
#define M2_PIN     (*((volatile unsigned long *)0x40007010))
//////////////////////////////////////////
// PIN CONNECTIONS
//
// MO   : PD0
// M1   : PD1
// M2   : PD2
// STEPA : PE0
// SLEEP : PB2
// STEPB : PE3
//
#include "Motor.h"
#include "tm4c123gh6pm.h"

void MotorInit()
{
        volatile unsigned long delay;
  SYSCTL_RCGC2_R       |= 0x00000002; // 1) activate clock for Port B
  while((SYSCTL_PRGPIO_R&0x02) == 0) {};      // allow time for clock to start
  GPIO_PORTB_AMSEL_R &= 0x00;         // 2) disable analog on Port B
  GPIO_PORTB_PCTL_R  = 0x00000000;  // 3) PCTL GPIO on PB0,PB1,PB2,PB3,PB7
        GPIO_PORTB_DIR_R    |= 0x8F;           // 4) Output Pins : PB0,PB1,PB2, PB3, PB7
  GPIO_PORTB_AFSEL_R &= 0x00;         // 5) disable alt funct on PB7-0
  GPIO_PORTB_PUR_R    &= 0x00;         // 6) disable pull-up on Port B
  GPIO_PORTB_DEN_R    |= 0x8F;         // 7) enable digital I/O on PB0,PB1,PB2, PB3, PB7

}
```

```
void PortEInit()
{
        volatile unsigned long delay;
  SYSCTL_RCGC2_R       |= 0x00000010; // 1) activate clock for Port E
  while((SYSCTL_PRGPIO_R&0x10) == 0) {};      // allow time for clock to start
  GPIO_PORTE_AMSEL_R &= 0x00;         // 2) disable analog on Port E
  GPIO_PORTE_PCTL_R  &= 0x00000000; // 3) No special functionality
        GPIO_PORTE_DIR_R    |= 0x1F;           // 4) Output Pins : PE0,PE1,PE2, PE3, PE7
  GPIO_PORTE_AFSEL_R &= 0x00;         // 5) disable alt funct on PB7-0
  GPIO_PORTE_PUR_R    &= 0x00;         // 6) disable pull-up on Port E
  GPIO_PORTE_DEN_R    |= 0x1F;         // 7) enable digital I/O on PE0,PE1,PE2, PE3, PE7

}
void MotorStepInit()
{
        volatile unsigned long delay;
  SYSCTL_RCGC2_R       |= 0x00000008; // 1) activate clock for Port D
  while((SYSCTL_PRGPIO_R&0x08) == 0) {};      // allow time for clock to start
  GPIO_PORTD_AMSEL_R &= 0x00;         // 2) disable analog on Port D
  GPIO_PORTD_PCTL_R  = 0x00000000;  // 3) PCTL GPIO on PD0-2
        GPIO_PORTD_DIR_R    |= 0x07;           // 4) Output Pins : PD0,PD1,PD2,
  GPIO_PORTD_AFSEL_R &= 0x00;         // 5) disable alt funct on PD2-0
  GPIO_PORTD_PUR_R    &= 0x00;         // 6) disable pull-up on Port D
  GPIO_PORTD_DEN_R    |= 0x07;         // 7) enable digital I/O on PD0,PD1,PD2
}
void MotorStepResolution(int step)
{
        switch (step)
        {
                case FULLSTEP :        GPIO_PORTD_DATA_R = 0x00; break; //M0 = LOW  | M1 = LOW  |
M2 = LOW

                case HALFSTEP :        GPIO_PORTD_DATA_R = 0x01; break; //M0 = HIGH | M1 = LOW  |
M2 = LOW

                case FOUTHSTEP:        GPIO_PORTD_DATA_R = 0x02; break; //M0 = LOW  | M1 = HIGH |
M2 = LOW

                case EIGHTSTEP:        GPIO_PORTD_DATA_R = 0x03; break; //M0 = HIGH | M1 = HIGH |
M2 = LOW
```

```
                case SIXTEENTHSTEP:    GPIO_PORTD_DATA_R = 0x04; break; //M0 = LOW  | M1 = LOW  |
M2 = HIGH

                case THIRTYTWOSTEP_v0: GPIO_PORTD_DATA_R = 0x05; break; //M0 = HIGH | M1 = LOW  |
M2 = HIGH

                case THIRTYTWOSTEP_v1: GPIO_PORTD_DATA_R = 0x06; break; //M0 = LOW  | M1 = HIGH |
M2 = HIGH

                case THIRTYTWOSTEP_v2: GPIO_PORTD_DATA_R = 0x07; break; //M0 = HIGH | M1 = HIGH |
M2 = HIGH



                // Default is set to Half Step
                default:   GPIO_PORTD_DATA_R = 0x01; break; //M0 = HIGH | M1 = LOW  | M2 = LOW
        }
}

/*               SYSTICK INITIATION                        */
////////////////////////////////////////////////////////////
void SysTick_Init(void){
  NVIC_ST_CTRL_R = 0;                // disable SysTick during setup
  NVIC_ST_RELOAD_R = NVIC_ST_RELOAD_M;  // maximum reload value
  NVIC_ST_CURRENT_R = 0;             // any write to current clears it
                                     // enable SysTick with core clock
  NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC;
}

////////////////////////////////////////////////////////////
void SysTick_Wait(unsigned long delay){
  volatile unsigned long elapsedTime;
  unsigned long startTime = NVIC_ST_CURRENT_R;
  do{
        elapsedTime = (startTime-NVIC_ST_CURRENT_R)&0x00FFFFFF;
   }
  while(elapsedTime <= delay);
}
// 10000us equals 10ms
void SysTick_Wait10ms(unsigned long delay){
  unsigned long i;
  for(i=0; i<delay; i++){
```

```
        SysTick_Wait(800000);  // wait 10ms
  }
}
//////////////////////////////////////
// Motor Direction
//////////////////////////////////////
void MotorDirection( int dirc)
{
        DIR_PIN = 0;
}


void EnableMotor()
{
        SLEEP_PIN = 0; //enable(0), disable(1)
        GPIO_PORTB_DATA_R = 0x80; //enable(1), disable(0) CHECK THIS IF DOESNT WORK

}
//////////////////////////////////////////
//Base movement
//////////////////////////////////////////
void MotorMoveB()
{
        SysTick_Wait(6040);
  GPIO_PORTE_DATA_R ^=  0x8;//(~STEP_PIN & 0x08);//masking PE3 bit
}
//////////////////////////////////////////
// Scope Movement
//////////////////////////////////////////
void MotorMoveA()
{
        SysTick_Wait(6040);
        GPIO_PORTE_DATA_R ^=  0x1;//(~STEP_PIN & 0x01);//masking PE0 bit
}
//////////////////////////////////////////////////////////////////
// Conversion of azimuth degrees into base motor step amounts
//////////////////////////////////////////////////////////////////
int moveamount( float var)
{
```

```
        int approximate;
        approximate  =  (int)(var / .00010875 );
        approximate = (int)(approximate / 3 );
        return approximate;
}
////////////////////////////////////////////////////////////////
// Conversion of altitude degrees into scope motor step amounts
////////////////////////////////////////////////////////////////
int movescopeamount(float var) {

return (int)(( var / .0003618)/15);

}
```

UART.h

```
// U0Rx (VCP receive) connected to PA0
// U0Tx (VCP transmit) connected to PA1

// standard ASCII symbols
#define CR   0x0D
#define LF   0x0A
#define BS   0x08
#define ESC  0x1B
#define SP   0x20
#define DEL  0x7F
////////////////////////////////////////////////////
/////////////////////////////////////////
// DISPLAYING INPUTTED CHARCTER ON UART 1
/////////////////////////////////////////
void UART1_OutDec(unsigned long data);
///////////////////////////////
// ACCEPTING FLOATS INTO UART
///////////////////////////////
float UART1_InUFloatX(void);
///////////////////////////////
// CONVERTING TIME INTO HOUR , MINUTES AND SECONDS
///////////////////////////////
void getTimeBreak(int *h, int *m, int *s, float time);
////////////////////////////////////////////////////

//------------UART_Init------------
// Initialize the UART for 115,200 baud rate (assuming 50 MHz clock),
// 8 bit word length, no parity bits, one stop bit, FIFOs enabled
// Input: none
// Output: none
void UART0_Init(void);

//------------UART_InChar------------
// Wait for new serial port input
// Input: none
// Output: ASCII code for key typed
```

```
unsigned char UART0_InChar(void);

//------------UART_OutChar------------
// Output 8-bit to serial port
// Input: letter is an 8-bit ASCII character to be transferred
// Output: none
void UART0_OutChar(unsigned char data);

//------------UART_OutString------------
// Output String (NULL termination)
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
void UART0_OutString(char *pt);

//------------UART_InUDec------------
// InUDec accepts ASCII input in unsigned decimal format
//      and converts to a 32-bit unsigned number
//      valid range is 0 to 4294967295 (2^32-1)
// Input: none
// Output: 32-bit unsigned number
// If you enter a number above 4294967295, it will return an incorrect value
// Backspace will remove last digit typed
unsigned long UART0_InUDec(void);

//-----------------------UART_OutUDec-----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void UART0_OutUDec(unsigned long n);

//---------------------UART_InUHex----------------------------------------
// Accepts ASCII input in unsigned hexadecimal (base 16) format
// Input: none
// Output: 32-bit unsigned number
// No '$' or '0x' need be entered, just the 1 to 8 hex digits
// It will convert lower case a-f to uppercase A-F
//      and converts to a 16 bit unsigned number
//      value range is 0 to FFFFFFFF
```

```
// If you enter a number above FFFFFFFF, it will return an incorrect value
// Backspace will remove last digit typed
unsigned long UART0_InUHex(void);

//-------------------------UART_OutUHex----------------------------
// Output a 32-bit number in unsigned hexadecimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1 to 8 digits with no space before or after
void UART0_OutUHex(unsigned long number);

//------------UART_InString------------
// Accepts ASCII characters from the serial port
//      and adds them to a string until <enter> is typed
//      or until max length of the string is reached.
// It echoes each character as it is inputted.
// If a backspace is inputted, the string is modified
//      and the backspace is echoed
// terminates the string with a null character
// uses busy-waiting synchronization on RDRF
// Input: pointer to empty buffer, size of buffer
// Output: Null terminated string
void UART0_InString(char *bufPt, unsigned short max);


////////////////////////////////////////////////////////////////////////////
//------------UART_Init------------
// Initialize the UART for 115,200 baud rate (assuming 50 MHz clock),
// 8 bit word length, no parity bits, one stop bit, FIFOs enabled
// Input: none
// Output: none
void UART1_Init(void);

//------------UART_InChar------------
// Wait for new serial port input
// Input: none
// Output: ASCII code for key typed
unsigned char UART1_InChar(void);
```

```
//------------UART_OutChar------------
// Output 8-bit to serial port
// Input: letter is an 8-bit ASCII character to be transferred
// Output: none
void UART1_OutChar(unsigned char data);

//------------UART_OutString------------
// Output String (NULL termination)
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
void UART1_OutString(char *pt);

//------------UART_InUDec------------
// InUDec accepts ASCII input in unsigned decimal format
//      and converts to a 32-bit unsigned number
//      valid range is 0 to 4294967295 (2^32-1)
// Input: none
// Output: 32-bit unsigned number
// If you enter a number above 4294967295, it will return an incorrect value
// Backspace will remove last digit typed
unsigned long UART1_InUDec(void);

//-----------------------UART_OutUDec-----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void UART1_OutUDec(unsigned long n);

//---------------------UART_InUHex----------------------------------------
// Accepts ASCII input in unsigned hexadecimal (base 16) format
// Input: none
// Output: 32-bit unsigned number
// No '$' or '0x' need be entered, just the 1 to 8 hex digits
// It will convert lower case a-f to uppercase A-F
//      and converts to a 16 bit unsigned number
//      value range is 0 to FFFFFFFF
// If you enter a number above FFFFFFFF, it will return an incorrect value
// Backspace will remove last digit typed
```

```
unsigned long UART1_InUHex(void);

//--------------------------UART_OutUHex----------------------------
// Output a 32-bit number in unsigned hexadecimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1 to 8 digits with no space before or after
void UART1_OutUHex(unsigned long number);

//------------UART_InString------------
// Accepts ASCII characters from the serial port
//      and adds them to a string until <enter> is typed
//      or until max length of the string is reached.
// It echoes each character as it is inputted.
// If a backspace is inputted, the string is modified
//      and the backspace is echoed
// terminates the string with a null character
// uses busy-waiting synchronization on RDRF
// Input: pointer to empty buffer, size of buffer
// Output: Null terminated string
void UART1_InString(char *bufPt, unsigned short max);
```

UART.c

```
#include "UART.h"


#define GPIO_PORTA_AFSEL_R    (*((volatile unsigned long *)0x40004420))
#define GPIO_PORTA_DEN_R      (*((volatile unsigned long *)0x4000451C))
#define GPIO_PORTA_AMSEL_R    (*((volatile unsigned long *)0x40004528))
#define GPIO_PORTA_PCTL_R     (*((volatile unsigned long *)0x4000452C))
#define UART0_DR_R            (*((volatile unsigned long *)0x4000C000))
#define UART0_FR_R            (*((volatile unsigned long *)0x4000C018))
```

```
#define UART0_IBRD_R          (*((volatile unsigned long *)0x4000C024))
#define UART0_FBRD_R          (*((volatile unsigned long *)0x4000C028))
#define UART0_LCRH_R          (*((volatile unsigned long *)0x4000C02C))
#define UART0_CTL_R           (*((volatile unsigned long *)0x4000C030))
#define UART_FR_TXFF          0x00000020  // UART Transmit FIFO Full
#define UART_FR_RXFE          0x00000010  // UART Receive FIFO Empty
#define UART_LCRH_WLEN_8      0x00000060  // 8 bit word length
#define UART_LCRH_FEN         0x00000010  // UART Enable FIFOs
#define UART_CTL_UARTEN       0x00000001  // UART Enable
#define SYSCTL_RCGC1_R        (*((volatile unsigned long *)0x400FE104))
#define SYSCTL_RCGC2_R        (*((volatile unsigned long *)0x400FE108))
#define SYSCTL_RCGC1_UART0      0x00000001  // UART0 Clock Gating Control
#define SYSCTL_RCGC2_GPIOA    0x00000001  // port A Clock Gating Control

//------------UART_Init------------
// Initialize the UART for 115,200 baud rate (assuming 50 MHz UART clock),
// 8 bit word length, no parity bits, one stop bit, FIFOs enabled
// Input: none
// Output: none
void UART0_Init(void){
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_UART0; // activate UART0
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA; // activate port A
  UART0_CTL_R &= ~UART_CTL_UARTEN;    // disable UART
  UART0_IBRD_R = 43;                  // IBRD = int(50,000,000 / (16 * 115,200)) = int(27.1267)
  UART0_FBRD_R = 26;                      // FBRD = int(0.1267 * 64 + 0.5) = 8
                                      // 8 bit word length (no parity bits, one stop bit, FIFOs)


  UART0_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN );
  UART0_CTL_R |= UART_CTL_UARTEN;     // enable UART
  GPIO_PORTA_AFSEL_R |= 0x03;         // enable alt funct on PA1-0
  GPIO_PORTA_DEN_R |= 0x03;           // enable digital I/O on PA1-0
                                      // configure PA1-0 as UART
  GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R&0xFFFFFF00)+0x00000011;
  GPIO_PORTA_AMSEL_R &= ~0x03;        // disable analog functionality on PA
}
//////////////////////////////////////////////////////////////////////////////////////////
/////////
#define GPIO_PORTB_AFSEL_R    (*((volatile unsigned long *)0x40005420))
#define GPIO_PORTB_DEN_R      (*((volatile unsigned long *)0x4000551C))
```

```
#define GPIO_PORTB_AMSEL_R    (*((volatile unsigned long *)0x40005528))
#define GPIO_PORTB_PCTL_R     (*((volatile unsigned long *)0x4000552C))
#define UART1_DR_R            (*((volatile unsigned long *)0x4000D000))
#define UART1_FR_R            (*((volatile unsigned long *)0x4000D018))
#define UART1_IBRD_R          (*((volatile unsigned long *)0x4000D024))
#define UART1_FBRD_R          (*((volatile unsigned long *)0x4000D028))
#define UART1_LCRH_R          (*((volatile unsigned long *)0x4000D02C))
#define UART1_CTL_R           (*((volatile unsigned long *)0x4000D030))
#define UART_FR_TXFF          0x00000020  // UART Transmit FIFO Full
#define UART_FR_RXFE          0x00000010  // UART Receive FIFO Empty
#define UART_LCRH_WLEN_8      0x00000060  // 8 bit word length
#define UART_LCRH_FEN         0x00000010  // UART Enable FIFOs
#define UART_CTL_UARTEN       0x00000001  // UART Enable
#define SYSCTL_RCGC1_R        (*((volatile unsigned long *)0x400FE104))
#define SYSCTL_RCGC2_R        (*((volatile unsigned long *)0x400FE108))
#define SYSCTL_RCGC1_UART1    0x00000002  // UART1 Clock Gating Control
#define SYSCTL_RCGC2_GPIOB    0x00000002  // port B Clock Gating Control

//------------UART_Init------------
// Initialize the UART for 115,200 baud rate (assuming 50 MHz UART clock),
// 8 bit word length, no parity bits, one stop bit, FIFOs enabled
// Input: none
// Output: none
void UART1_Init(void){
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_UART1; // activate UART1
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOB; // activate port B
  UART1_CTL_R &= ~UART_CTL_UARTEN;    // disable UART
  UART1_IBRD_R = 43;                  // IBRD = int(50,000,000 / (16 * 38400)) = int(27.1267)
  UART1_FBRD_R = 26;                      // FBRD = int(0.1267 * 64 + 0.5) = 8
                                      // 8 bit word length (no parity bits, one stop bit, FIFOs)
  UART1_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN );
  UART1_CTL_R |= UART_CTL_UARTEN;     // enable UART
  GPIO_PORTB_AFSEL_R |= 0x03;              // enable alt funct on PB1-0
  GPIO_PORTB_DEN_R |= 0x03;           // enable digital I/O on PB1-0
                                      // configure PA1-0 as UART
  GPIO_PORTB_PCTL_R = (GPIO_PORTB_PCTL_R&0xFFFFFF00)+0x00000011;
  GPIO_PORTB_AMSEL_R &= ~0x03;        // disable analog functionality on PB
}
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////
/////////

//------------UART_InChar------------
// Wait for new serial port input
// Input: none
// Output: ASCII code for key typed
unsigned char UART0_InChar(void){
  while((UART0_FR_R&UART_FR_RXFE) != 0);
  return((unsigned char)(UART0_DR_R&0xFF));
}
unsigned char UART1_InChar(void){
  while((UART1_FR_R&UART_FR_RXFE) != 0);
  return((unsigned char)(UART1_DR_R&0xFF));
}
//------------UART_OutChar------------
// Output 8-bit to serial port
// Input: letter is an 8-bit ASCII character to be transferred
// Output: none
void UART0_OutChar(unsigned char data){
                while((UART0_FR_R&UART_FR_TXFF) != 0);
  UART0_DR_R = data;
}

//////////////////////////////////////////////////////////////////////////////////////////

//------------UART_OutString------------
// Output String (NULL termination)
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
void UART0_OutString(char *pt){
  while(*pt){
        UART0_OutChar(*pt);
        pt++;
  }
}

//------------UART_InUDec------------
```

```
// InUDec accepts ASCII input in unsigned decimal format
//      and converts to a 32-bit unsigned number
//      valid range is 0 to 4294967295 (2^32-1)
// Input: none
// Output: 32-bit unsigned number
// If you enter a number above 4294967295, it will return an incorrect value
// Backspace will remove last digit typed
unsigned long UART0_InUDec(void){
unsigned long number=0, length=0;
char character;
  character = UART0_InChar();
  while(character != CR){ // accepts until <enter> is typed
// The next line checks that the input is a digit, 0-9.
// If the character is not 0-9, it is ignored and not echoed
        if((character>='0') && (character<='9')) {
        number = 10*number+(character-'0');   // this line overflows if above 4294967295
        length++;
        UART0_OutChar(character);
        }
// If the input is a backspace, then the return number is
// changed and a backspace is outputted to the screen
        else if((character==BS) && length){
        number /= 10;
        length--;
        UART0_OutChar(character);
        }
        character = UART0_InChar();
  }
  return number;
}

//-----------------------UART_OutUDec-----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void UART0_OutUDec(unsigned long n){
// This function uses recursion to convert decimal number
//   of unspecified length as an ASCII string
```

```
  if(n >= 10){
        UART0_OutUDec(n/10);
        n = n%10;
  }
  UART0_OutChar(n+'0'); /* n is between 0 and 9 */
}

//---------------------UART_InUHex----------------------------------------
// Accepts ASCII input in unsigned hexadecimal (base 16) format
// Input: none
// Output: 32-bit unsigned number
// No '$' or '0x' need be entered, just the 1 to 8 hex digits
// It will convert lower case a-f to uppercase A-F
//      and converts to a 16 bit unsigned number
//      value range is 0 to FFFFFFFF
// If you enter a number above FFFFFFFF, it will return an incorrect value
// Backspace will remove last digit typed
unsigned long UART0_InUHex(void){
unsigned long number=0, digit, length=0;
char character;
  character = UART0_InChar();
  while(character != CR){
        digit = 0x10; // assume bad
        if((character>='0') && (character<='9')){
        digit = character-'0';
        }
        else if((character>='A') && (character<='F')){
        digit = (character-'A')+0xA;
        }
        else if((character>='a') && (character<='f')){
        digit = (character-'a')+0xA;
        }
// If the character is not 0-9 or A-F, it is ignored and not echoed
        if(digit <= 0xF){
        number = number*0x10+digit;
        length++;
        UART0_OutChar(character);
        }
// Backspace outputted and return value changed if a backspace is inputted
```

```
        else if((character==BS) && length){
        number /= 0x10;
        length--;
        UART0_OutChar(character);
        }
        character = UART0_InChar();
  }
  return number;
}

//-------------------------UART_OutUHex----------------------------
// Output a 32-bit number in unsigned hexadecimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1 to 8 digits with no space before or after
void UART0_OutUHex(unsigned long number){
// This function uses recursion to convert the number of
//   unspecified length as an ASCII string
  if(number >= 0x10){
        UART0_OutUHex(number/0x10);
        UART0_OutUHex(number%0x10);
  }
  else{
        if(number < 0xA){
        UART0_OutChar(number+'0');
        }
        else{
        UART0_OutChar((number-0x0A)+'A');
        }
  }
}

//------------UART_InString------------
// Accepts ASCII characters from the serial port
//      and adds them to a string until <enter> is typed
//      or until max length of the string is reached.
// It echoes each character as it is inputted.
// If a backspace is inputted, the string is modified
//      and the backspace is echoed
```

```
// terminates the string with a null character
// uses busy-waiting synchronization on RDRF
// Input: pointer to empty buffer, size of buffer
// Output: Null terminated string
void UART0_InString(char *bufPt, unsigned short max) {
int length=0;
char character;
  character = UART0_InChar();
  while(character != CR){
        if(character == BS){
        if(length){
        bufPt--;
        length--;
         UART0_OutChar(BS);
        }
        }
        else if(length < max){
        *bufPt = character;
        bufPt++;
        length++;
        UART0_OutChar(character);
        }
        character = UART0_InChar();
  }
  *bufPt = 0;
}


//////////////////////////////////////////////////////////////////////////////
//------------UART_InChar------------
// Wait for new serial port input
// Input: none
// Output: ASCII code for key typed

//------------UART_OutChar------------
// Output 8-bit to serial port
// Input: letter is an 8-bit ASCII character to be transferred
// Output: none
void UART1_OutChar(unsigned char data){
```

```
  while((UART1_FR_R&UART_FR_TXFF) != 0);
  UART1_DR_R = data;
}


//------------UART_OutString------------
// Output String (NULL termination)
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
void UART1_OutString(char *pt){
  while(*pt){
        UART1_OutChar(*pt);
        pt++;
  }
        UART1_OutChar(CR);
}

//------------UART_InUDec------------
// InUDec accepts ASCII input in unsigned decimal format
//      and converts to a 32-bit unsigned number
//      valid range is 0 to 4294967295 (2^32-1)
// Input: none
// Output: 32-bit unsigned number
// If you enter a number above 4294967295, it will return an incorrect value
// Backspace will remove last digit typed
unsigned long UART1_InUDec(void){
unsigned long number=0, length=0;
char character;
  character = UART1_InChar();
  while(character != CR){ // accepts until <enter> is typed
// The next line checks that the input is a digit, 0-9.
// If the character is not 0-9, it is ignored and not echoed
        if((character>='0') && (character<='9')) {
        number = 10*number+(character-'0');   // this line overflows if above 4294967295
        length++;
        UART1_OutChar(character);
        }
// If the input is a backspace, then the return number is
// changed and a backspace is outputted to the screen
```

```
        else if((character==BS) && length){
        number /= 10;
        length--;
        UART1_OutChar(character);
        }
        character = UART1_InChar();
  }
  return number;
}

//-----------------------UART_OutUDec-----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void UART1_OutUDec(unsigned long n){
// This function uses recursion to convert decimal number
//   of unspecified length as an ASCII string
  if(n >= 10){
        UART1_OutUDec(n/10);
        n = n%10;
  }
  UART1_OutChar(n+'0'); /* n is between 0 and 9 */
}

//---------------------UART_InUHex----------------------------------------
// Accepts ASCII input in unsigned hexadecimal (base 16) format
// Input: none
// Output: 32-bit unsigned number
// No '$' or '0x' need be entered, just the 1 to 8 hex digits
// It will convert lower case a-f to uppercase A-F
//      and converts to a 16 bit unsigned number
//      value range is 0 to FFFFFFFF
// If you enter a number above FFFFFFFF, it will return an incorrect value
// Backspace will remove last digit typed
unsigned long UART1_InUHex(void){
unsigned long number=0, digit, length=0;
char character;
  character = UART1_InChar();
```

```
  while(character != CR){
        digit = 0x10; // assume bad
        if((character>='0') && (character<='9')){
        digit = character-'0';
        }
        else if((character>='A') && (character<='F')){
        digit = (character-'A')+0xA;
        }
        else if((character>='a') && (character<='f')){
        digit = (character-'a')+0xA;
        }
// If the character is not 0-9 or A-F, it is ignored and not echoed
        if(digit <= 0xF){
        number = number*0x10+digit;
        length++;
        UART1_OutChar(character);
        }
// Backspace outputted and return value changed if a backspace is inputted
        else if((character==BS) && length){
        number /= 0x10;
        length--;
        UART1_OutChar(character);
        }
        character = UART1_InChar();
  }
  return number;
}

//-------------------------UART_OutUHex----------------------------
// Output a 32-bit number in unsigned hexadecimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1 to 8 digits with no space before or after
void UART1_OutUHex(unsigned long number){
// This function uses recursion to convert the number of
//   unspecified length as an ASCII string
  if(number >= 0x10){
        UART1_OutUHex(number/0x10);
        UART1_OutUHex(number%0x10);
```

```
  }
  else{
        if(number < 0xA){
        UART1_OutChar(number+'0');
        }
        else{
        UART1_OutChar((number-0x0A)+'A');
        }
  }
}

//------------UART_InString------------
// Accepts ASCII characters from the serial port
//      and adds them to a string until <enter> is typed
//      or until max length of the string is reached.
// It echoes each character as it is inputted.
// If a backspace is inputted, the string is modified
//      and the backspace is echoed
// terminates the string with a null character
// uses busy-waiting synchronization on RDRF
// Input: pointer to empty buffer, size of buffer
// Output: Null terminated string
void UART1_InString(char *bufPt, unsigned short max) {
int length=0;
char character;
  character = UART1_InChar();
  while(character != CR){
        if(character == BS){
        if(length){
        bufPt--;
        length--;
         UART1_OutChar(BS);
        }
        }
        else if(length < max){
        *bufPt = character;
        bufPt++;
        length++;
        UART1_OutChar(character);
```

```
        }
        character = UART1_InChar();
  }
  *bufPt = 0;
}

//////////////////////////////////////////////
//             UART In Float
// Accepts ASCII characters and converts them into their respective
// integer value and adds them to a float until 'x' is typed
//      or until max length of the string is reached.
 float UART1_InUFloatX(void){
 float number=0, length=0;
 char character;
 int frac_flag = 0;// fraction flag
 int frac = 10;        // nth places init
 character = UART1_InChar();//wait until recieve numbers
  while(character != 'x'){ // accepts until 'x' is typed
// The next line checks that the input is a digit, 0-9.
// If the character is not 0-9, it is ignored and not echoed
                if(character == '.')
                {
                        frac_flag = 1;//faction portion of value incoming
                }
        if((character>='0') && (character<='9') && frac_flag == 0) {
        number = 10*number+(character-'0');   // this line overflows if above 4294967295
        length++;
        UART1_OutChar(character);
        }
// adding the fractional components to the value
                if((character>='0') && (character<='9') && frac_flag == 1)
                {
                        number = number+((float)(character-'0')/frac);   // this line overflows if
above 4294967295
        length++;
                        frac = frac*10;
        UART1_OutChar(character);
                }
// If the input is a backspace, then the return number is
// changed and a backspace is outputted to the screen
```

```
        else if((character==BS) && length){
        number /= 10;
        length--;
        UART1_OutChar(character);
        }
        character = UART1_InChar();
  }
  return number;
}

//////////////////////////////////////////////
//              Converting time format hh: mm : ss to
//      seperate values: Hours , minutes
//              and seconds recursivley
//////////////////////////////////////////////
void getTimeBreak(int *h, int *m, int *s, float time)
{
        if ((time / 100)>0)
        {
                getTimeBreak(h, m, s, time / 100);
        }
        if (*h == 0)
        {
                *h = (int)time;
        }
        else if (*m == 0)
        {
                *m = (int)(time - *h * 100);
        }
        else if (*s == 0)
        {
                *s = (int)(time - (*h * 10000) - (*m * 100));
        }
}
```

PLL.c

```
#include "PLL.h"

// The #define statement SYSDIV2 in PLL.h
// initializes the PLL to the desired frequency.
```

```
// bus frequency is 400MHz/(SYSDIV2+1) = 400MHz/(7+1) = 50 MHz
// see the table at the end of this file

#define SYSCTL_RIS_R          (*((volatile unsigned long *)0x400FE050))
#define SYSCTL_RIS_PLLLRIS    0x00000040  // PLL Lock Raw Interrupt Status
#define SYSCTL_RCC_R          (*((volatile unsigned long *)0x400FE060))
#define SYSCTL_RCC_XTAL_M      0x000007C0  // Crystal Value
#define SYSCTL_RCC_XTAL_6MHZ  0x000002C0  // 6 MHz Crystal
#define SYSCTL_RCC_XTAL_8MHZ  0x00000380  // 8 MHz Crystal
#define SYSCTL_RCC_XTAL_16MHZ   0x00000540  // 16 MHz Crystal
#define SYSCTL_RCC2_R         (*((volatile unsigned long *)0x400FE070))
#define SYSCTL_RCC2_USERCC2   0x80000000  // Use RCC2
#define SYSCTL_RCC2_DIV400    0x40000000  // Divide PLL as 400 MHz vs. 200
                                          // MHz
#define SYSCTL_RCC2_SYSDIV2_M   0x1F800000  // System Clock Divisor 2
#define SYSCTL_RCC2_SYSDIV2LSB  0x00400000  // Additional LSB for SYSDIV2
#define SYSCTL_RCC2_PWRDN2    0x00002000  // Power-Down PLL 2
#define SYSCTL_RCC2_BYPASS2   0x00000800  // PLL Bypass 2
#define SYSCTL_RCC2_OSCSRC2_M   0x00000070  // Oscillator Source 2
#define SYSCTL_RCC2_OSCSRC2_MO  0x00000000  // MOSC

// configure the system to get its clock from the PLL
void PLL_Init(void){
  // 0) configure the system to use RCC2 for advanced features
  //    such as 400 MHz PLL and non-integer System Clock Divisor
  SYSCTL_RCC2_R |= SYSCTL_RCC2_USERCC2;
  // 1) bypass PLL while initializing
  SYSCTL_RCC2_R |= SYSCTL_RCC2_BYPASS2;
  // 2) select the crystal value and oscillator source
  SYSCTL_RCC_R &= ~SYSCTL_RCC_XTAL_M;   // clear XTAL field
  SYSCTL_RCC_R += SYSCTL_RCC_XTAL_16MHZ;// configure for 16 MHz crystal
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_OSCSRC2_M;// clear oscillator source field
  SYSCTL_RCC2_R += SYSCTL_RCC2_OSCSRC2_MO;// configure for main oscillator source
  // 3) activate PLL by clearing PWRDN
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_PWRDN2;
  // 4) set the desired system divider and the system divider least significant bit
  SYSCTL_RCC2_R |= SYSCTL_RCC2_DIV400;  // use 400 MHz PLL
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R&~0x1FC00000) // clear system clock divider field
```

```
              + (SYSDIV2<<22);      // configure for 80 MHz clock
  // 5) wait for the PLL to lock by polling PLLLRIS
  while((SYSCTL_RIS_R&SYSCTL_RIS_PLLLRIS)==0){};
  // 6) enable use of PLL by clearing BYPASS
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_BYPASS2;
}


/*
SYSDIV2  Divisor  Clock (MHz)
 0      1       reserved
 1      2       reserved
 2      3       reserved
 3      4       reserved
 4      5       80.000
 5      6       66.667
 6      7       reserved
 7      8       50.000
 8      9       44.444
 9      10      40.000
 10     11      36.364
 11     12      33.333
 12     13      30.769
 13     14      28.571
 14       15      26.667
 15     16      25.000
 16     17      23.529
 17     18      22.222
 18     19      21.053
 19     20      20.000
 20     21      19.048
 21     22      18.182
 22     23      17.391
 23     24      16.667
 24     25      16.000
 25     26      15.385
 26     27      14.815
 27     28      14.286
 28     29      13.793
```

29	30	13.333
30	31	12.903
31	32	12.500
32	33	12.121
33	34	11.765
34	35	11.429
35	36	11.111
36	37	10.811
37	38	10.526
38	39	10.256
39	40	10.000
40	41	9.756
41	42	9.524
42	43	9.302
43	44	9.091
44	45	8.889
45	46	8.696
46	47	8.511
47	48	8.333
48	49	8.163
49	50	8.000
50	51	7.843
51	52	7.692
52	53	7.547
53	54	7.407
54	55	7.273
55	56	7.143
56	57	7.018
57	58	6.897
58	59	6.780
59	60	6.667
60	61	6.557
61	62	6.452
62	63	6.349
63	64	6.250
64	65	6.154
65	66	6.061
66	67	5.970
67	68	5.882

68	69	5.797
69	70	5.714
70	71	5.634
71	72	5.556
72	73	5.479
73	74	5.405
74	75	5.333
75	76	5.263
76	77	5.195
77	78	5.128
78	79	5.063
79	80	5.000
80	81	4.938
81	82	4.878
82	83	4.819
83	84	4.762
84	85	4.706
85	86	4.651
86	87	4.598
87	88	4.545
88	89	4.494
89	90	4.444
90	91	4.396
91	92	4.348
92	93	4.301
93	94	4.255
94	95	4.211
95	96	4.167
96	97	4.124
97	98	4.082
98	99	4.040
99	100	4.000
100	101	3.960
101	102	3.922
102	103	3.883
103	104	3.846
104	105	3.810
105	106	3.774
106	107	3.738

```
 107    108     3.704
 108    109     3.670
 109    110     3.636
 110    111     3.604
 111    112     3.571
 112    113     3.540
 113    114     3.509
 114    115     3.478
 115    116     3.448
 116    117     3.419
 117    118     3.390
 118    119     3.361
 119    120     3.333
 120    121     3.306
 121    122     3.279
 122    123     3.252
 123    124     3.226
 124    125     3.200
 125    126     3.175
 126    127     3.150
 127    128     3.125
*/
```

PLL.h

```
// The #define statement SYSDIV2 initializes
// the PLL to the desired frequency.
#define SYSDIV2 4
// bus frequency is 400MHz/(SYSDIV2+1) = 400MHz/(7+1) = 50 MHz

// configure the system to get its clock from the PLL
void PLL_Init(void);


/*
SYSDIV2  Divisor  Clock (MHz)
 0      1       reserved
 1      2       reserved
 2      3       reserved
```

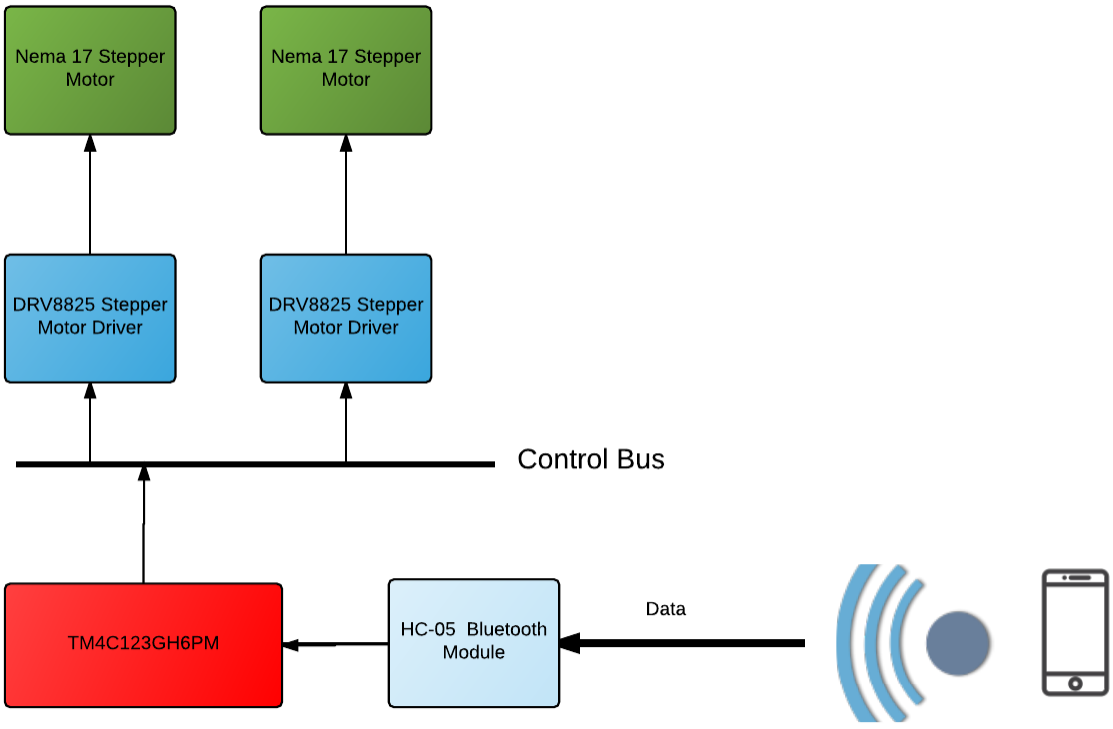
3	4	reserved
4	5	80.000
5	6	66.667
6	7	reserved
7	8	50.000
8	9	44.444
9	10	40.000
10	11	36.364
11	12	33.333
12	13	30.769
13	14	28.571
14	15	26.667
15	16	25.000
16	17	23.529
17	18	22.222
18	19	21.053
19	20	20.000
20	21	19.048
21	22	18.182
22	23	17.391
23	24	16.667
24	25	16.000
25	26	15.385
26	27	14.815
27	28	14.286
28	29	13.793
29	30	13.333
30	31	12.903
31	32	12.500
32	33	12.121
33	34	11.765
34	35	11.429
35	36	11.111
36	37	10.811
37	38	10.526
38	39	10.256
39	40	10.000
40	41	9.756
41	42	9.524

42	43	9.302
43	44	9.091
44	45	8.889
45	46	8.696
46	47	8.511
47	48	8.333
48	49	8.163
49	50	8.000
50	51	7.843
51	52	7.692
52	53	7.547
53	54	7.407
54	55	7.273
55	56	7.143
56	57	7.018
57	58	6.897
58	59	6.780
59	60	6.667
60	61	6.557
61	62	6.452
62	63	6.349
63	64	6.250
64	65	6.154
65	66	6.061
66	67	5.970
67	68	5.882
68	69	5.797
69	70	5.714
70	71	5.634
71	72	5.556
72	73	5.479
73	74	5.405
74	75	5.333
75	76	5.263
76	77	5.195
77	78	5.128
78	79	5.063
79	80	5.000
80	81	4.938

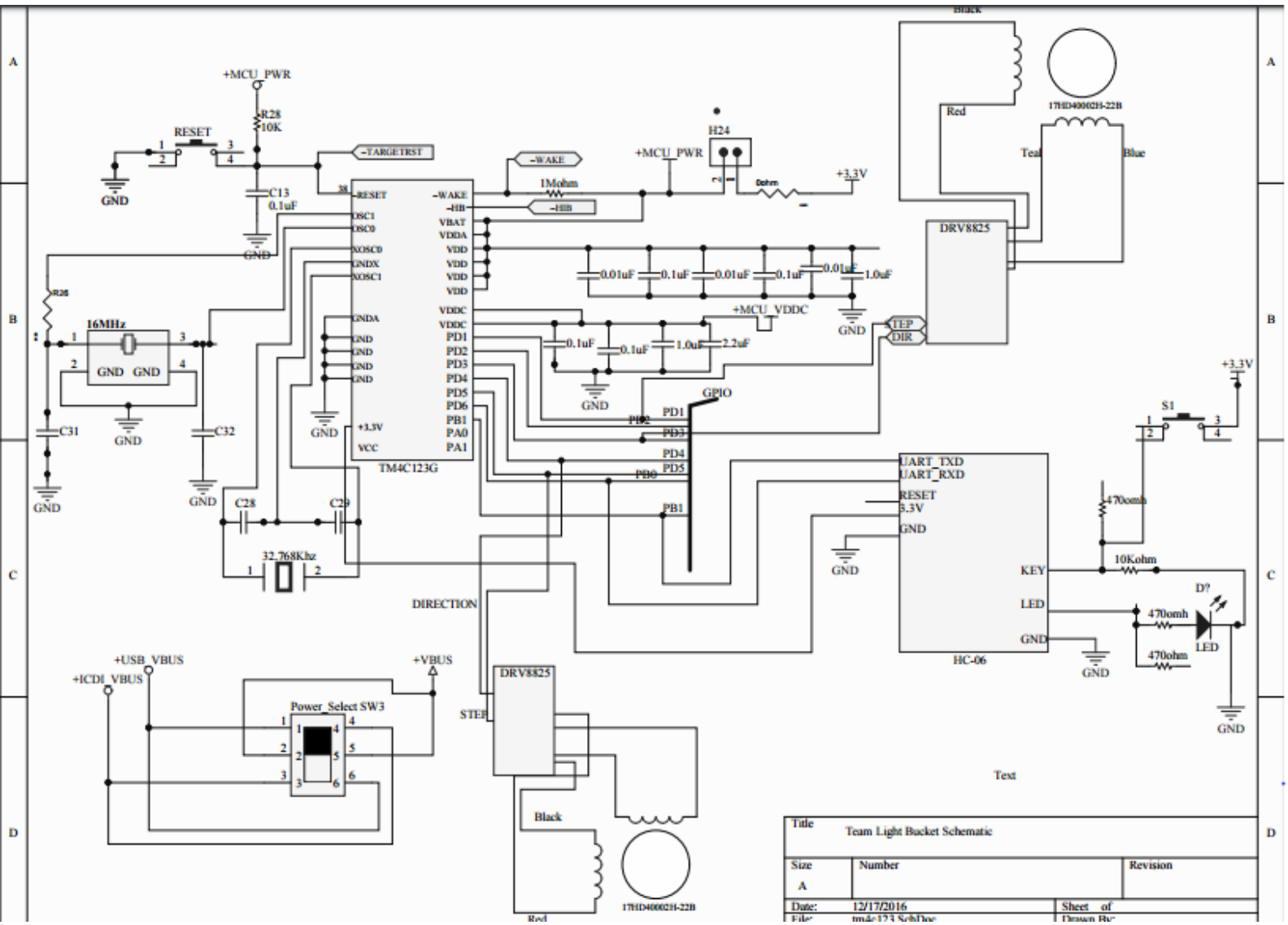
81	82	4.878
82	83	4.819
83	84	4.762
84	85	4.706
85	86	4.651
86	87	4.598
87	88	4.545
88	89	4.494
89	90	4.444
90	91	4.396
91	92	4.348
92	93	4.301
93	94	4.255
94	95	4.211
95	96	4.167
96	97	4.124
97	98	4.082
98	99	4.040
99	100	4.000
100	101	3.960
101	102	3.922
102	103	3.883
103	104	3.846
104	105	3.810
105	106	3.774
106	107	3.738
107	108	3.704
108	109	3.670
109	110	3.636
110	111	3.604
111	112	3.571
112	113	3.540
113	114	3.509
114	115	3.478
115	116	3.448
116	117	3.419
117	118	3.390
118	119	3.361
119	120	3.333

```
 120    121      3.306
 121    122      3.279
 122    123      3.252
 123      124      3.226
 124    125      3.200
 125    126      3.175
 126    127      3.150
 127    128      3.125
*/
```

Block Diagram



Schematic


+MCU_PWR
R28
10K
RESET
GND
C13
0.1uF
~TARGETRST
~WAKE
+MCU_PWR
H24
1Mohm
+3.3V
~HIB
TM4C123G
RESET
OSC1
OSC0
VBAT
VDDA
VDD
VDDC
PD1
PD2
PD3
PD4
PD5
PD6
PB1
PA0
PA1
+3.3V
VCC
0.01uF
0.1uF
1.0uF
2.2uF
+MCU_VDDC
R26
16MHz
C31
C32
C28
C29
32.768Khz
GPIO
DIRECTION
STEP
DIR
DRV8825
Red
Teal
Blue
Black
17HD4000I2H-22B
UART_TXD
UART_RXD
RESET
3.3V
GND
HC-06
KEY
LED
S1
470omh
10Kohm
D?
470ohm
+USB_VBUS
+ICDI_VBUS
+VBUS
Power_Select SW3
Text
Title
Team Light Bucket Schematic
Size
A
Number
Revision
Date:
12/17/2016
Sheet of

Demonstration Videos

1. Base and Scope Movement : See attached video : “Alititude and Azimuth coordinate movement”
2. Android app and telescope control : See attached video : “AppVideo”

Resources

1. The Android Open Source Project (2014). Android SDK 4.0.4 (IceCreamSandwich). Available from Bluetooth Chat Sample
2. Converting RA and DEC to ALT and AZ2(1998) by Keith Unit Conversion Link
3. Embedded Systems: Introduction to ARM Cortex-M Microcontrollers by Jonathan Valvano.Chapter 11 Serial Interfacing
4. DRV8825 Specifications DRV8825