V I S I O N
Camera Controlled System for Mechatronic Football

Final Report

Electrical Engineering Senior Design - Spring 2013

University of Notre Dame

Lucas De la Fuente Munita

Bryan Dimas

Nicholas Ferruolo

Ryan Hames

Kevin Shetler

Table of Contents

1 Introduction

The Vision project aims to design a camera control system that would be able to track robotic football players on a playing field. Mechatronic football involves the programming and design of robots to play a game of football. Over the years the project has evolved and it now involves robots that can launch a football to be caught by another robot. However, this now creates a large margin of error in terms of completing passes. This opens the door for new technology to be implemented in order to improve the mechatronic football game. With the Vision camera control system, live position feedback sent to the robots would decrease the margin of error in passing and create a more exciting mechatronic football game. This camera control system project could also lead the way for the robots to be computer controlled. The system can even applied further than the mechatronic football game; it could be used in other robotic sport game, with security cameras, and even for autonomous cars.

2 Detailed Systems Requirements

The control system takes bitmap information from the camera via an Universal Asynchronous Receive/ Transmit (UART) interface where the image information is processed with an algorithm to determine location of the players. The board then wirelessly sends the locations to the robots as well as a computer terminal for the user to read. The embedded intelligence has enough processing power to process the images and derive the x-y coordinates of the objects. It also works in real time, continuously taking in bitmaps, processing them, and broadcasting the coordinates. It is essential that the system does not experience lag, for if it does the information will be inaccurate and not improve the margin of error that we are trying to reduce.

The board itself is powered by plugging it into the wall using an extension cord from the top of the platform holding the board. The system takes anywhere from 5-9V and regulates it to 3.3V in order to power the camera and the board. The whole system is powered by this connection. The microcontroller controls the camera, asking for new bitmaps as needed. This is fairly often given the pace of robotic football.

The algorithm that processes the bitmap consists of several steps. First it finds the contours of objects so that it can define what an object is and what the background is. It then filters out the colors by converting the image to the gray-scale so that it can find all of the red objects for one team and all of the green objects for the other team. The program then marks the centers of each object with a red cross for the red objects and a green cross for the green objects. The coordinates of each object will also be displayed next to these crosses so that the user can read their location and the coordinates can be transmitted to the robots.

The board design also includes a wireless ZigBee interface which is connected to the board via a Serial Peripheral Interface (SPI). The wireless communication is responsible for communicating the information from the board to the robots and to a terminal. Users read off a terminal the team number and position of the robots. The packets of data are sent continuously from the source to receiver. Therefore the wireless interface supports the control system working in real time. This accurate, moment by moment information gives the users greater control over the game as well as the robots a feedback system to know where they are.

The user interface of this system is a computer terminal running PuTTY, which interfaces to the microcontroller through the UART. The team number and x-y coordinates are expressed in a table in the terminal window. The table is switching between displaying team one objects and team two objects. This eliminates the distraction of scrolling text and allows for the data to be read with greater ease.

The system is installed by suspending the camera and microcontroller from a wooden support. This allows the camera to face squarely down toward the field. The camera is connected to the microcontroller with sexed connectors of four and six pins. The camera must be at the appropriate height to measure the entire field. For the demonstration, the camera is about 35 inches off the ground which covers a 20 x 24 inch area. The board is programmed before being placed over the field.

3 Detailed Project Description

3.1 System Theory of Operation:

The camera system will be suspended over the playing field with the whole field in view. It runs continuously so the users can have real time information about the robots. The camera system communicates with the robots, providing locations using IEEE 802.15.4. The CMUcam4 sends a 640x480 bitmap images in RGB565 binary formatted data to the board via the UART lines. The PIC32 microchip receives this bitmap image and stores it as an image array. The PIC32 then runs its algorithm to identify colors (players) and provide their exact location on the image. Afterwards the PIC32 converts pixel coordinates to distance coordinates and sends the object team number and coordinates wirelessly using ZigBee. Each data packet contains the coordinates of every object on a certain team.

3.2 System Block Diagram:

As stated above, the camera system can be broken up into four subsystems. The CMUCam4 subsystem deals with capturing an image and sending a bitmap of that image to the computer. The board/microchip subsystem controls the camera sending and receiving images and storing and processing the images. The image tracking algorithm subsystem is responsible for determining the contours of objects to define what a continuous object is, filter the color to find the differently colored pieces, and determine the coordinates of the objects that are being tracked. The wireless communication subsystem uses the SPI interface to communicate to the ZigBee board and is responsible for sending the identification and location of the objects being tracked. This information is sent to every robot and to a terminal so a user can read it.

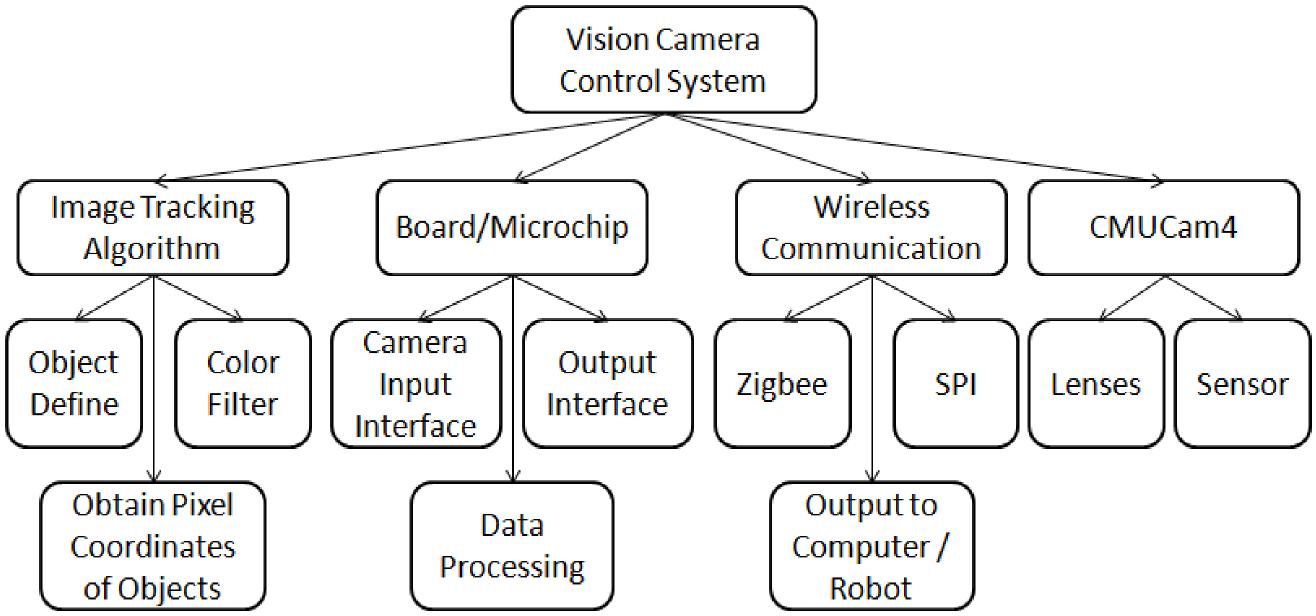


Figure 1: System block diagram

3.3 Subsystem 1 – CMUCam4:

The first subsystem in the Vision camera control system is most importantly, the camera. The

CMUcam4 is a fully programmable embedded computer vision sensor. It contains a Parallax P8X32A processor that is connected to an Omnivision 9665 CMOS camera sensor module. The CMUcam4 is open source programmable and is low-cost; low power that doesn't require a PC to program. The CMUcam4 can perform various on-board vision processing tasks that include color tracking, image statistics, and a histogram in real time. However, for our purposes, we treated the camera as a black box that only supplies an RGB bitmap.

CMUcam4 Key Features

· Fully open source and programmable
· VGA resolution (640x480) RGB565/YUV655 color sensor
· Image processing rate of 30 frames per second
· Raw image dumps over serial (640:320:160:180) x (480:420:120:60) image resolution
· Segmented image capture for tracking visualization (over serial)
· I/O Interface TTL UART (up to 250,000 baud -19200 baud by default)

CMUcam4 Applications

· Automotive (lane detection)
· Buildings (occupancy sensing, light metering)
· Education (interactive toys, video display)
· Manufacturing (product inspection)
· Robotics (robot navigation, object detection, object recognition, object tracking, servo control)
· Surveillance (digital camera, data logging, sensor networks)

As the above features and applications show, the CMUcam4 was an ideal choice for what we set out to accomplish. The CMUcam4 works extremely well with robotics, and can accomplish all the features that we want in object detection, object recognition, and object tracking. The main reason we chose the CMUcam4 was because of the UART serial communication. We needed a camera that could communicate with a PIC32 microchip via I2C, UART, or SPI. After researching various cameras and camera sensors we came to the conclusion that the CMUcam4 would be provide an adequate image for our purpose and could be ported with a PIC32. Since it is also low cost, we could scale our system to cover a larger area and provide more accurate data due to the higher resolution that would be supplied by using multiple cameras.

We only planned for the CMUcam4 to send bitmap images to the PIC32 but it also has more functions than sending a bitmap image. It currently has over fifty commands. Commands sent to the camera can adjust to brightness and contrast, auto gain and auto white balance, run in black and white and negative mode, change baud rate, run television commands, color tracking for up to eight objects, and save to an SD slot. So the CMUcam has the object tracking capabilities already but we wanted to go learn about the algorithms ourselves. It also gives us more freedom to analyze images instead of relying on the tracking capabilities of the camera.

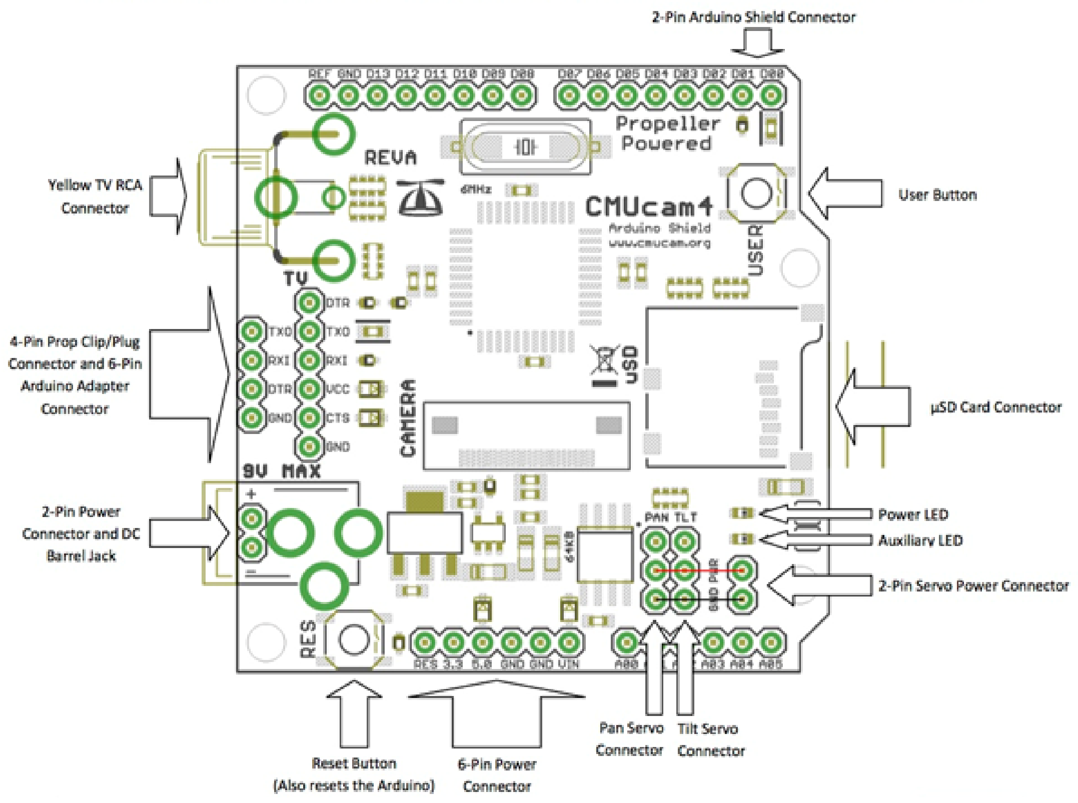

2-Pin Arduino Shield Connector
Yellow TV RCA Connector
User Button
4-Pin Prop Clip/Plug Connector and 6-Pin Arduino Adapter Connector
µSD Card Connector
2-Pin Power Connector and DC Barrel Jack
Power LED
Auxiliary LED
2-Pin Servo Power Connector
Reset Button (Also resets the Arduino)
6-Pin Power Connector
Pan Servo Connector
Tilt Servo Connector
CMUcam4
Propeller Powered
REVA
TV
CAMERA
USER
RES
9V MAX

Fig. 2 CMUcam4 Board Layout

CMUcam4 functions are enabled by porting in the portable serial and timer wrapper library. CMUcam4 is typically programmed by using Arduino programming environment but for Senior Design, Arduino's are not allowed. Since CMUcam4 is open source, it means that anyone can program and communicate with the CMUcam4 as long as they have a C/C++ environment and a microcontroller/microchip with serial communication. For our project, we chose the PIC32 microchip as we had the most experience working with MPLAB X integrated development environment.

The first step in communicating and sending commands to the camera is downloading and porting in the portable serial and timer wrapper library. This header file and cpp file contain the functions necessary for the camera to communicate with a device using serial lines. The functions in these files are written in C++ code but the functions are meant for Arduino's. So same functions in these files had to be rewritten in order to be compatible with the PIC32 and use the standard libraries in MPLAB X.

The way the code flows the main source file will call the CMUcom4 (communication) header file and the CMUcom4 cpp file (functions library). The CMUcom4 communication header file defines many important communication features such as the buffer size, the serial port, bits and strings. These are done through wrapper functions. The CMUcom4 functions library file was where the most changes had to be made. Many of the functions within the file were written in Arduino C++. This means that although the functions are part of the standard C++ libraries, most of the code is preprocessed, so any other microcontroller/microchip would be unable to compile the code because it is missing precompiled code. So we had to rewrite all the functions in Arduino C++ to common C++ code that can be used for the Microchip. The functions library therefore was rewritten in to open the PIC32 serial port to the CMUcam4, read a byte from the PIC32 serial port, write a character to the PIC32, write a string to the PIC32, sending data, and a timer. We threw out functions meant for Arduino that we did not need such as the flush, peek, and closing the serial port.

So the main file, would call the header file and functions library, among other header files, in order to send commands to the CMUcam4 through the PIC32 UART and receive data in return. Sending commands to the PIC32 was simple as each command to the camera is only two characters are needed to make the camera accomplish a function. So for our purpose we needed an RGB bitmap image of 640 x 480. So the command to the camera "Send Frame" and would be sent as "SF 0 0." We write the characters using the printf C function terminated with a /r (which would tell the camera it was the end of the command). The camera would then send an RGB bitmap as an array of characters through the serial port back to the PIC32. In order to accomplish the command, header file and functions library need to be in the project folder and need to be rewritten for the PIC32 to function, otherwise it would not work.

Using the PIC32 kit board we tested this process. We would connect the camera RX1 and TX1 ports to the UART 6 port on the kit board. We would then port in the proper files into MPLAB X and compile the code. We would program the PIC32 using the Pickit 3 programmer, which would connect to the PIC32. The PIC32 would then run this code and send the command to the camera, which would then receive the data and store it. For our designed board, we would communicate to UART 4 serial lines. Program the PIC32 and upload the same code using the Pickit3. Subsystem 2 is processed by the data received.

We followed the steps described above in order to program the CMUcam4 through the PIC32 via UART serial lines. The first problem we ran into was converting the functions and language CMUcom4.cpp and CMUcom4.h files from Arduino based C++ language to C++. Functions that were associated in those files called functions that were not available in the standard library in MPLAB X. There were also functions that we did not understand and could not replicate. Therefore we rewrote the entire header and cpp file, eliminating all the code associated with each function, and putting in our own functions related to the PIC32 that would accomplish the same thing. After a long period in trial and error and consulting other programmers, we finally got our code to compile and build without any errors. We then moved on to the next step which was to connect the CMUcam4 to the PIC32 Kit board and send a simple command to the camera. We initially attempted to try this by using Putty program on the computer to send a command and receive data in return, but Putty returns a /n after each command and CMUcam4 needs a /r after each command. So we attempted to send commands directly to the camera without any visual aid (Putty) and not knowing if we receive data in return. Our CMUcam4 project would compile and would program the PIC32 kit board to send the command. We would not receive any data in return. So we checked the signal on the PIC32 kit board and CMUcam4 and we would detect the signal being sent to the CMUcam4. We detected that the CMUcam4 was working, because it would have a busy signal, that could be stopped if the Reset button was held. This was the road block that we were unable to overcome. After countless efforts to rethink, and approach the same process from a different angle, our efforts were not met with success and we could not program and receive data from the CMUcam4. This is one of the reasons why our system did not work as originally planned.

3.4 Subsystem 2 – Board/Microchip:

The board requires a power source of 3.3V so as to keep the entire system running. This voltage will be enough to power the camera, board, and communication system without issue. The board must be plugged in using a wall charger provided to power the system.

The board also contains several ports to the various components of the system. One of these ports connects to the CMUcam4 through a 4 pin header labeled CMUCAM on the board to the six pin header on the CMUcam4 board. Only 3 connections need to be made, between the transmit and receive on the camera and board and to ground. Another port connects to the wireless communication system by using a 10 pin header labeled ZIGBEE to connect the wireless communication system to the SPI interface from the microcontroller. There is also a port called TEST1 on the board that can be used to test the functionality of the board and for troubleshooting. This three pin header connects to the microcontroller in a manner such that one would be able to make sure that the microcontroller is functional and use it for debugging. A port for an FPGA is also built into the board if someone would want to use that instead of processing the images on the microcontroller itself. The 4 pin header labeled FPGA1 on the board can be used to make a connection to any FPGA that one may wish to use

Below in Figure 3 is a picture of the board that has been finished, but does not have the part soldered into place yet.

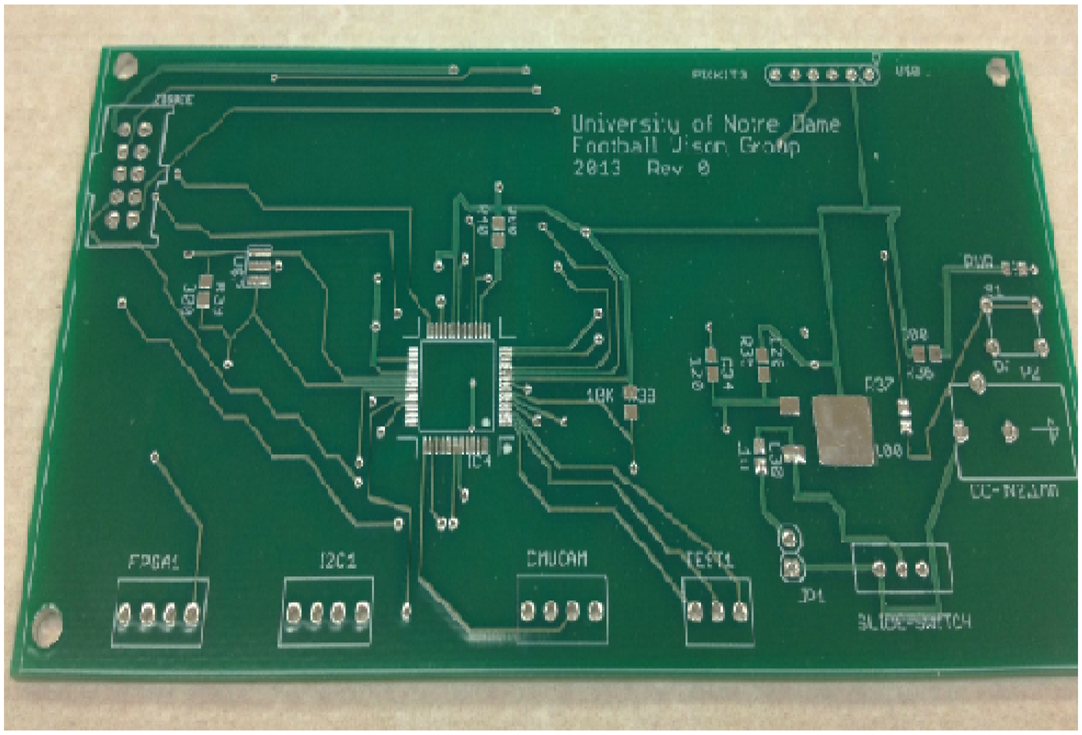


Figure 3: Picture of board

The microcontroller that was used is the PIC32MX695F512L microcontroller. We chose this microcontroller because we had experience using it and because it would have enough power to store and process the images that would be sent to it. The microcontroller is shown below in Figure 4.

Figure 4: Picture of the PIC32MX695F512L microcontroller

3.5 Subsystem 3 – Image Tracking Algorithms:

The image tracking algorithms were first developed in Matlab, using the Image Processing Toolbox. Our objective was to make a function that was simple enough to be implemented in a microcontroller while also complex enough to be able to recognize a number of elements and be able to split them into two categories, that is, two distinct football teams.

We discovered that the Image Processing Toolbox provided great functions for tracking objects in still cameras. In general, to tracking images, given a still camera, requires not only to identify an object but also to recognize it in future frames; hence in order to do this we must address each of these issues separately. The former - identify the presence of an object - is done through a combination of several methods like edge identification and background comparison. The latter is done through several predictive methods which address different levels of complexity, like changes in light (a camera overlooking a street) or objects that disappear at times because of blocking objects.

Given the range of possible approaches and the necessity of our project we decided that a color filter would be the most appropriate way to achieve the result we wanted. For the two different teams we identified them by color and established a certain threshold that distinguishes them from the rest. Even better, we used the colors red and green avoiding complex calculations in the microcontroller because pure red and green channels are in the bitmap file.

The final algorithm works as follows:

-After obtaining the image it extracts either the red channel or the green channels and compares it to the gray scale of the image. This is a pixel-wise operation that yields another bitmap with areas where red or green is more intense.

-The resulting bitmap is put through a binary threshold filter, which makes pixels above the threshold white and the rest of them black. This process creates a binary file with our areas of interest in white.

-A contour function is applied. This function labels any threshold above certain number of pixels and returns the center of each figure found. Since both red and green channels are analyzed in parallel this algorithm is applied twice to different binary bitmaps yielding two variable vectors in which each vector represents a team and each element is the coordinate of an object found.

3.6 Subsystem 4 – Wireless Communication:

The centers of the contours must now be turned into positions on the field of play and transmitted to the robots. For the demonstration, a square inch is 20 x 20 pixels, the entire field being 20 x 24 inches. Therefore, simply dividing the y-coordinate pixel position by 20 provides the number of inches the object is up from the bottom. Likewise, dividing the x-coordinate pixel position by 20 gives the number of inches from the left hand side of the field.

Applied to a larger scale, we calculated that a 16’ x 20’ rectangle could reasonably be covered by one camera. To encompass the entire field, four cameras will adequately cover everything. These bitmaps from each camera would be concatenated to make one large bitmap of high resolution. While the mounting and positioning of the cameras in Stepan Center could offer some logistical challenges, other than this the camera system is easily scalable to a very large field barring processor limitations.

Now the team identifier, x-coordinate, and y-coordinate must be broadcast to the robots. We decided to use a ZigBee transmitter which follows IEEE 802.15.4 standard. The ZigBee transmitter interfaces to the microcontroller using SPI. Under this serial communication method, the microcontroller was the master with the ZigBee as a slave. They shared a common clock provided by the microcontroller and operated in eight bit mode. The clock is divided so that the baud rate is 57600, but it can be adjusted if needed. As the code indicates, to communicate with the Zigbee card first the SPI must be initialized, then certain registers in the Zigbee daughter card must be set so it is initialized. After this, commands are sent out by writing to a state register.

Figure 7-1. Basic Operating Mode State Diagram (for State Transition Timing Data Refer to Table 7-1)

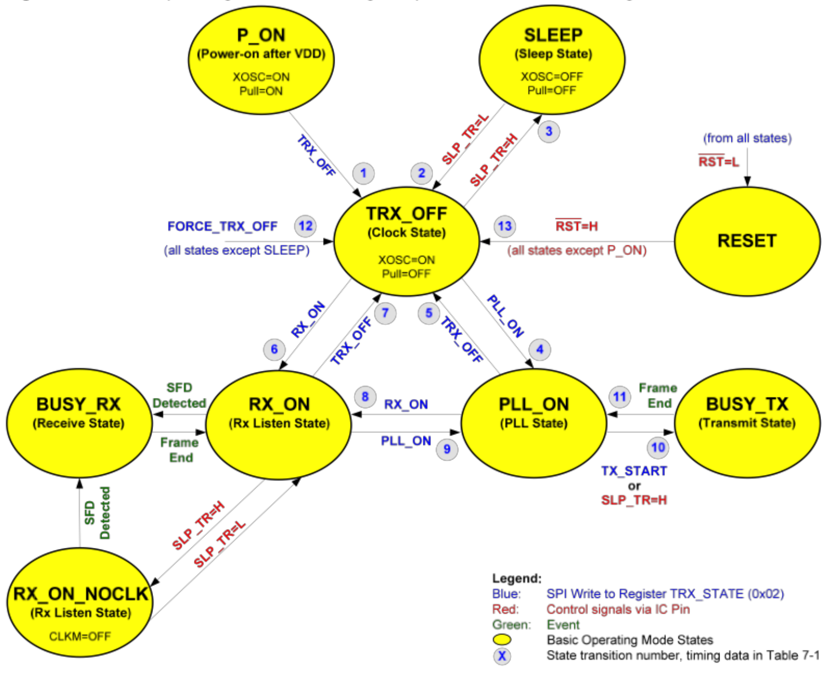


Figure 5: ZigBee Module State Diagram

The primary states used in our system are PLL_ON, BUSY_TX, RX_ON, and BUSY_RX. The command to change states was issued by writing to the TRX_STATE register. You could then check which state the system was in by reading the TRX_STATUS register. For transmission the buffer was written to (different method than adjusting registers), PLL_ON was initiated, and once the antenna locked, a transmission was started by writing TX_START. Likewise for receive, the interrupt was generated when an incoming transmission was sensed, and then the receiver would transition from RX_ON to BUSY_RX.

The information was transmitted using the structure outlined by 802.15.4. As exhibited by figure six, there are many parameters that must be sent before the actual payload data. These precursors such as the PAN ID are set in the registers to ensure that it is properly transmitting, but they are also sent along with the data.

Figure 8-2 IEEE 802.15.4-2003 Frame Format – MAC Layer Frame Structure

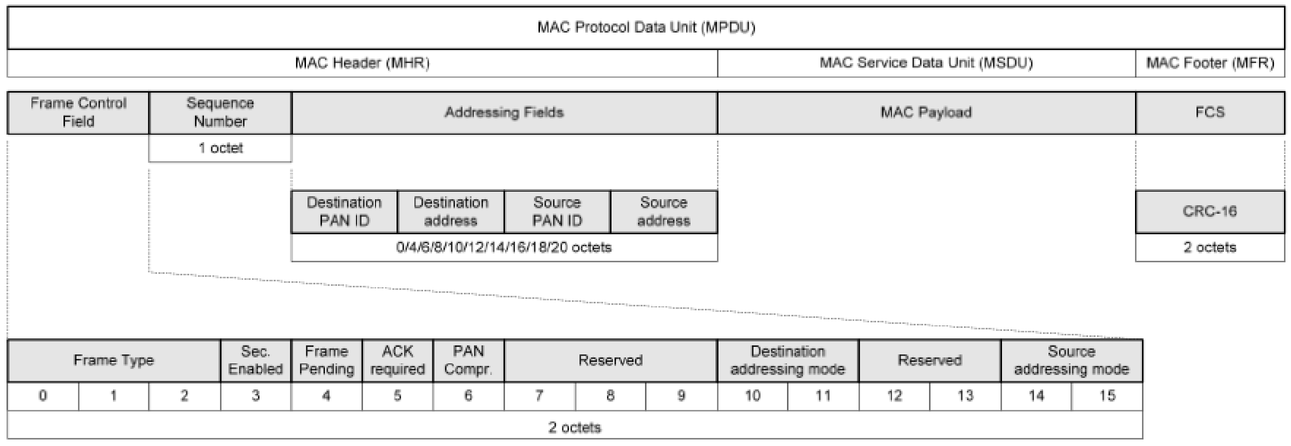
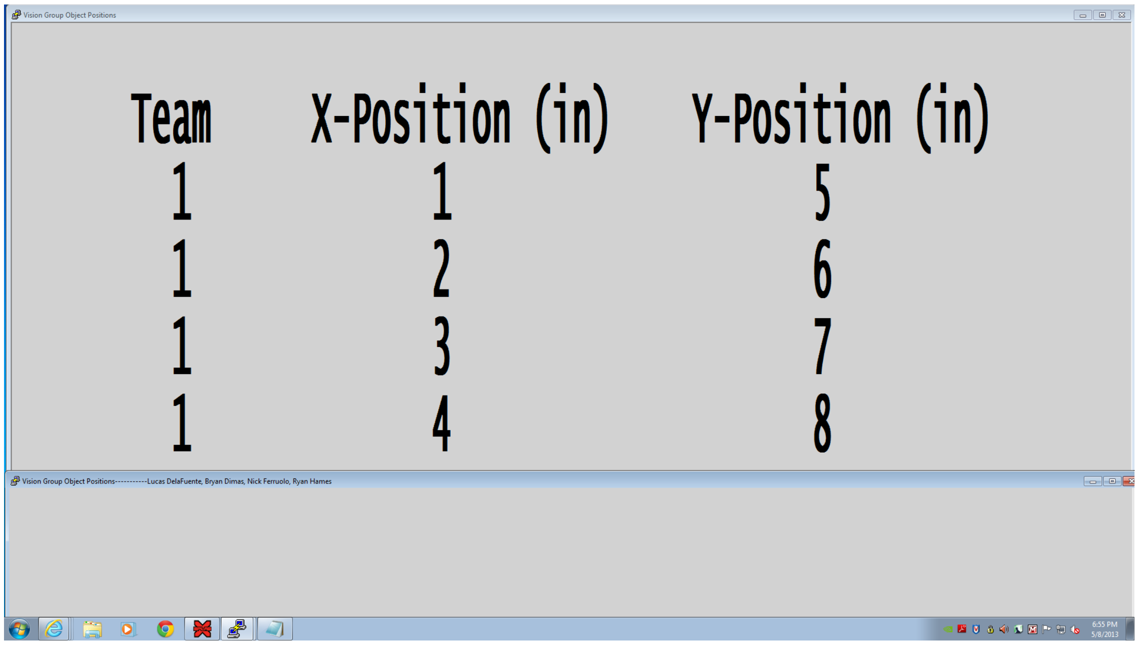


Figure 6: Transmission Frame Structure

The payload data was of the form # payload bytes,Team #, x1,y1,x2,y2,x3,y3,x4,y4. Once the positions are received, the robots could go on to make decisions based on this data. For demonstration purposes, we displayed the team positions on the terminal with PuTTY via the microcontroller UART lines at a baud rate of 57600. Below is a screenshot from the demonstration, notice how the PuTTY settings are different from normal. The configuration settings can be found in the appendix.


Vision Group Object Positions
Vision Group Object Positions----------Lucas DelaFuente, Bryan Dimas, Nick Ferruolo, Ryan Hames
6:55 PM
5/8/2013

```
Team    X-Position (in)    Y-Position (in)
 1            1                  5
 1            2                  6
 1            3                  7
 1            4                  8
```

Figure 7: Terminal Window

4 Systems Integration Testing

4.1 Testing the Subsystems:

To test the functionality of our color tracking algorithm we used Matlab again, this time in combination with the camera integrated in our laptops; this methodology allowed us to test the tracking of several objects and how to adjust the filter's thresholds under different light conditions as well as see how each filter changed the original image.

For the wireless distribution of the positions, the send and receive functions were tested with ZigBee sender and sniffer modules.   These modules either sent a known packet of data, or would listen to and display received data via PuTTY. If data is not being displayed on the screen, the UART connection may be compromised. Ensure that the UART is properly connected and that the baud rate is 57600. Once it was apparent that either the send or listen function worked, it was possible to test one with the other. This was the end result functionality and final test, the board attached to the camera took the positions and sent them to the receiver board which displayed them on the terminal using PuTTY. The SPI was tested using USBee software which would record and decipher SPI signals.

4.2 Testing the Overall System:

In order to run the camera control system as a whole the user needs to make sure that the subsystems are communicating in the right manner. To make sure of this the most logical and simple method is to follow the path of the original image step by step. This one needs to start from the setup and check for the position of the camera and that its field of vision is appropriate. Next we need to check the camera is working and connected to the microchip. At this point we can use it’s alternative port to see that the camera sees confirming that both the range is correct and that the camera is working. Next comes the microcontroller and the wireless systems. While both of these systems are separate, it is easier to test them together by a computer that receives the transmissions and checks for its contents. Depending on the information we can make sure that different levels of processing are being completed. If it is sending information then the wireless communication system is working, and if this information comes in the form of pairs of coordinates then the microcontroller is working as well. Finally if all the above is functioning we can have a test player in the field and correlate its position with a coordinate, and if this matches the whole system is running correctly.

5 Users Manual/Installation Manual

The Installation manual's objective is to provide an accessible source of information regarding the overall installation, use and troubleshooting of our product. It is important to notice that the system as a whole is composed of several subsystems that have complexities of their own and because of this, it is possible that some of the issues regarding the overall functioning of the product have their explanation in a particular subsystem; it is important to be able to recognize this type of error and refer to the functioning of that specific subsystem right away.

5.1 How to Install Product:

The first step to installing the vision system is to make all the necessary connections. This includes connecting the CMUCam4 to the microcontroller board that will transmit the camera's data. Also, the receiving board should be connected to a computer or another screen where the data will be displayed. All boards need to be connected to a DC power source. The camera also has to option to be connected via an RCA cable to a television or monitor to display what the camera is seeing.

One thing that is important to note is that the code may need to be adjusted for different light conditions. The object detection algorithm sets color thresholds to determine whether or not a certain pixel can be considered that color. The optimal threshold value for finding the objects will vary in different lighting conditions. The objects will not be found correctly if the algorithm thinks there is more or less of a particular color. These threshold values for each color can be adjusted in the algorithm code.

5.2 How to Set Up Product:

The product needs to be set up by placing the system underneath the structure above a playing field. For our testing we placed it about 3 feet above a table in order to get the playing field of a table. In the future, with upgrades and a better camera, the system can be placed much higher, about 30 feet above the ground to cover a large area of the field.

During the set up, the camera, board and Zigbee needs to be facing down towards the field. This is to ensure that the camera is capturing the best image and that the wireless communication is not being interfered with the structure that holds it.

5.3 Expected Product Operation:

When the product is working correctly, the images captured by the camera will be shown on a screen, if hooked up. The team number and position will be transmitted to the wireless receiver, which will display them on PuTTY set to a baud rate of 57600. The transmitted information is also available to be received by the robots themselves.

5.4 Troubleshooting:

In order to solve issues effectively the first step is recognize the subsystem or communication step which is failing. The easiest process to identify which subsystem is erroneous is to analyze them separately, that is, to follow the steps outlined in ‘Testing the overall System’. Depending on where the problem "starts" the responses will be different. There are, however, some common errors that will avoid the complications that a full analysis entails. For example, one simple and common mistake could be in the light conditions of the camera. If the camera is used in a very dark environment it is very likely that the camera will not recognize an object at all, and the opposite might happen as well creating in this case "non-existing players". Correct light conditions or a modification of the threshold will solve this issue promptly. Another common issue might be the impossibility to receive any wireless signaling from the wireless communication system, this might be due to a number of reasons some of them external to the product itself and should be checked before intervening in the subsystems. For example, the presence of other wireless signals in the same frequency or the use of a receiver too far away from the microcontroller could cause problems with receiving the data from the wireless communication system. These are just particular examples that illustrate that understanding the source of your problem is the most useful tool always. If the problem can not be easily fixed, please rely on the an overall analysis of all subsystems.

6 To-Market Design Changes

There are several changes that should be made before sending this project to-market. One of these possible changes would be to build a more appropriate storing system for the camera system. We should develop some sort of container to house all of the boards and the camera that would be suspended over the playing field as opposed to the having all of the boards out in the open. This would give the product a sleeker and cleaner design, and it would also make the product easier to transport or move around because there would only be the box as opposed to all the boards and components.

Another possible change that could be made to the design would be to design a battery for the board instead of just plugging it into the wall or to a USB charger. This would allow for the system to be set up easier because there wouldn't be wires running all over the place just to power the board. The system could be placed above the field and there would not have to be any outside connections to the system. There would be no risk of obstruction from any wires with the field or objects being tracked. This would also make the product look cleaner, sleeker, be more convenient, and more self contained.

A possible change that could be made to the design would be integrate the system so multiple cameras could be used together to cover the whole field rather than just the one hanging overhead. This would allow the cameras to focus on less ground to produce better accuracy in measurements. It would also make tracking of all of the objects easier because there would be less of them in each frame for each camera. There would be a need to develop the code so that it knows where each camera is in relation to another and then be able to combine them all together so that they are like one image of the field. These changes would allow larger fields to have objects tracked on them and make the product more versatile rather than forcing all of the fields to be the same size to work properly.

7 Conclusion

Mechatronic Football is an exciting program that has a lot of room for improvement in order to make the performance by the robots more exciting. The camera system will allow users to create football plays and use the system to complete plays at a higher rate of succession. The camera system is meant to improve the communication and tracking of robots, so that when the quarterback robot launches the football, the receiver robot will be able to catch it. While this was the first version in attempting to create a system compatible with Mechatronic Football, this camera-controlled system also has other capabilities outside of robotics. The system can be used in buildings to track location and movement of people, it can be used with systems that launch objects at moving objects, and it can be used to transmit the location of colored objects. Although our project was not successful in transmitting the RGB bit map image, we were successful in developing colored object tracking algorithms and transmitting the array wirelessly. To overcome these obstacles, more research should be done to properly code and send commands to the camera through a PIC32 and the MPLAB X C++ compiler. The microchip is also not powerful enough to handle all the coding information to send commands to the camera, receive the data, process the data, then send it through ZigBee. Our system contains an extra connection that can communicate with an FPGA through a UART serial connection. Recommendation for FPGA would be an Altera DE2 as it would be able to handle the code functions necessary to use all the algorithms necessary for colored object tracking.

If this system reaches a commercial grade, much more work would need to be added in order to upgrade the camera system and finding a more powerful camera. One option would be to redesign the CMUcam4, but use the camera sensor to its full power. The camera sensor on the CMUcam4 is only capturing about 1/8th of its capability. The highest image the CMUcam4 can get is a 640 x 480 but the Omni camera sensor can go to 5120 x 3840. So the Omni camera sensor can be redesigned with its own board and serial functionality to function like a CMUcam4 but to its full power. However, that would require a more powerful microchip and processing power. If the FPGA is added, it would require a different programming environment outside of MPLAB X and it would have to port into the

MPLAB X - PIC32 to be able to communicate with it. We feel this is a good step forward in order to developing a better, more powerful camera controlled system that can meet the fast and demanding needs of mechatronic football. Although we were unable to demonstrate a fully functional prototype, all the subsystems and pieces are there to make the system work, it is just a matter of debugging, and upgrading the code functions.

8 Appendices

ORIGINAL PORTABLE SERIAL AND TIMER WRAPPER LIBRARY – C++ ARDUINO CPP FILE

```
/***************************************************************************//**
* @file
* Portable serial and timer wrapper library.
*
* @version @n 1.1
* @date @n 2/7/2013
*
* @authors @n Kwabena W. Agyeman & Christopher J. Leaf
* @copyright @n (c) 2013 Kwabena W. Agyeman & Christopher J. Leaf
* @n All rights reserved - Please see the end of the file for the terms of use
*
* @par Update History:
* @n v0.1 - Beta code - 3/20/2012
* @n v0.9 - Original release - 4/18/2012
* @n v1.0 - Documented and updated release - 8/3/2012
* @n v1.1 - Added support for the Arduino Due, fixed the send frame command,
            and fixed a number of compile time warnings - 2/7/2013.
*******************************************************************************/

#include "CMUcom4.h"

/*******************************************************************************
* Constructor Functions
*******************************************************************************/

CMUcom4::CMUcom4()
{
            _port = CMUCOM4_SERIAL;
}

CMUcom4::CMUcom4(int port)
{
            _port = port;
```

```
}

/*******************************************************************************
* Public Functions
*******************************************************************************/

void CMUcom4::begin(unsigned long baud)
{
        delayMilliseconds(CMUCOM4_BEGIN_DELAY);

#if defined(__AVR_ATmega1280__) || \
        defined(__AVR_ATmega2560__) || \
        defined(__SAM3X8E__)
        switch(_port)
        {
        case CMUCOM4_SERIAL1: Serial1.begin(baud); break;
        case CMUCOM4_SERIAL2: Serial2.begin(baud); break;
        case CMUCOM4_SERIAL3: Serial3.begin(baud); break;
        default: Serial.begin(baud); break;
        }
#else
        Serial.begin(baud);
#endif

        delayMilliseconds(CMUCOM4_BEGIN_DELAY);
}

void CMUcom4::end()
{
        delayMilliseconds(CMUCOM4_END_DELAY);

#if defined(__AVR_ATmega1280__) || \
        defined(__AVR_ATmega2560__) || \
        defined(__SAM3X8E__)
        switch(_port)
        {
        case CMUCOM4_SERIAL1: Serial1.end(); break;
        case CMUCOM4_SERIAL2: Serial2.end(); break;
        case CMUCOM4_SERIAL3: Serial3.end(); break;
        default: Serial.end(); break;
        }
#else
        Serial.end();
#endif

        delayMilliseconds(CMUCOM4_END_DELAY);
}

int CMUcom4::read()
{
#if defined(__AVR_ATmega1280__) || \
        defined(__AVR_ATmega2560__) || \
        defined(__SAM3X8E__)
        switch(_port)
        {
        case CMUCOM4_SERIAL1: return Serial1.read(); break;
        case CMUCOM4_SERIAL2: return Serial2.read(); break;
        case CMUCOM4_SERIAL3: return Serial3.read(); break;
        default: return Serial.read(); break;
        }
#else
```

```
		return Serial.read();
#endif
}

size_t CMUcom4::write(uint8_t c)
{
#if defined(__AVR_ATmega1280__) || \
		defined(__AVR_ATmega2560__) || \
		defined(__SAM3X8E__)
		switch(_port)
		{
		case CMUCOM4_SERIAL1: return Serial1.write(c); break;
		case CMUCOM4_SERIAL2: return Serial2.write(c); break;
		case CMUCOM4_SERIAL3: return Serial3.write(c); break;
		default: return Serial.write(c); break;
		}
#else
		return Serial.write(c);
#endif
}

size_t CMUcom4::write(const char * str)
{
#if defined(__AVR_ATmega1280__) || \
		defined(__AVR_ATmega2560__) || \
		defined(__SAM3X8E__)
		switch(_port)
		{
		case CMUCOM4_SERIAL1: return Serial1.write(str); break;
		case CMUCOM4_SERIAL2: return Serial2.write(str); break;
		case CMUCOM4_SERIAL3: return Serial3.write(str); break;
		default: return Serial.write(str); break;
		}
#else
		return Serial.write(str);
#endif
}

size_t CMUcom4::write(const uint8_t * buffer, size_t size)
{
#if defined(__AVR_ATmega1280__) || \
		defined(__AVR_ATmega2560__) || \
		defined(__SAM3X8E__)
		switch(_port)
		{
		case CMUCOM4_SERIAL1: return Serial1.write(buffer, size); break;
		case CMUCOM4_SERIAL2: return Serial2.write(buffer, size); break;
		case CMUCOM4_SERIAL3: return Serial3.write(buffer, size); break;
		default: return Serial.write(buffer, size); break;
		}
#else
		return Serial.write(buffer, size);
#endif
}

int CMUcom4::available()
{
#if defined(__AVR_ATmega1280__) || \
		defined(__AVR_ATmega2560__) || \
		defined(__SAM3X8E__)
		switch(_port)
```

```
        {
        case CMUCOM4_SERIAL1: return Serial1.available(); break;
        case CMUCOM4_SERIAL2: return Serial2.available(); break;
        case CMUCOM4_SERIAL3: return Serial3.available(); break;
        default: return Serial.available(); break;
        }
#else
        return Serial.available();
#endif
}

void CMUcom4::flush()
{
#if defined(__AVR_ATmega1280__) || \
        defined(__AVR_ATmega2560__) || \
        defined(__SAM3X8E__)
        switch(_port)
        {
        case CMUCOM4_SERIAL1: Serial1.flush(); break;
        case CMUCOM4_SERIAL2: Serial2.flush(); break;
        case CMUCOM4_SERIAL3: Serial3.flush(); break;
        default: Serial.flush(); break;
        }
#else
        Serial.flush();
#endif
}

int CMUcom4::peek()
{
#if defined(__AVR_ATmega1280__) || \
        defined(__AVR_ATmega2560__) || \
        defined(__SAM3X8E__)
        switch(_port)
        {
        case CMUCOM4_SERIAL1: return Serial1.peek(); break;
        case CMUCOM4_SERIAL2: return Serial2.peek(); break;
        case CMUCOM4_SERIAL3: return Serial3.peek(); break;
        default: return Serial.peek(); break;
        }
#else
        return Serial.peek();
#endif
}

void CMUcom4::delayMilliseconds(unsigned long ms)
{
        return delay(ms);
}

unsigned long CMUcom4::milliseconds()
{
        return millis();
}

/***************************************************************************//**
* @file
* @par MIT License - TERMS OF USE:
* @n Permission is hereby granted, free of charge, to any person obtaining a
* copy of this software and associated documentation files (the "Software"), to
* deal in the Software without restriction, including without limitation the
* rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
* sell copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
* @n
* @n The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
* @n
* @n THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
******************************************************************************/
```

MODIFIED PORTABLE SERIAL AND TIMER WRAPPER LIBRARY – C++ PIC32 FILE

```
/***************************************************************************//**
* @file
* Portable serial and timer wrapper library.
*
* @version @n 1.1
* @date @n 2/7/2013
*
* @authors @n Kwabena W. Agyeman & Christopher J. Leaf
* @copyright @n (c) 2013 Kwabena W. Agyeman & Christopher J. Leaf
* @n All rights reserved - Please see the end of the file for the terms of use
*
* @par Update History:
* @n v0.1 - Beta code - 3/20/2012
* @n v0.9 - Original release - 4/18/2012
* @n v1.0 - Documented and updated release - 8/3/2012
* @n v1.1 - Added support for the Arduino Due, fixed the send frame command,
         and fixed a number of compile time warnings - 2/7/2013.
*******************************************************************************/

#include "CMUcom4.h"
#include <stdbool.h>
#include <xc.h>
#include <stdlib.h>
#include <stdio.h>


/*******************************************************************************
* Constructor Functions
*******************************************************************************/

CMUcom4::CMUcom4()
{
  _port = CMUCOM4_SERIAL;
}

CMUcom4::CMUcom4(int port)
{
  _port = port;
}

/*******************************************************************************
* Public Functions
*******************************************************************************/

void begin(unsigned long baud)
```

```
{
  //code need to open the PICs serial port to the CMUcam4
            U3MODEbits.BRGH=1;
            U3BRG = 128; // Set Baud rate
            U3MODEbits.PDSEL=0;
            U3MODEbits.STSEL=0;
            U3STAbits.UTXEN=1;
            U3STAbits.URXEN=1;
            U3MODEbits.ON=1;
}



void end()
{
  //code need to close the PIC?s serial port to the CMUcam4
}



int read()
{
            while (U3STAbits.URXDA == 0)
            {
            }

            return(U3RXREG);

  //code needed to read a byte from the PIC?s serial port... should return ?1 if no byte
}



int write(uint8_t c) {
            while (U3STAbits.UTXBF == 1){
          TXSTAbits.TXEN=0;// disable transmission
          TXREG=c;          // load txreg with data
           TXSTAbits.TXEN=1;    // enable transmission
 while(TXSTAbits.TRMT==0) // wait here till transmit complete
 {
  Nop();
 }
            }

            U3TXREG = c;

  //code needed to write a character to PIC32
}

int write(const char * str)
{

  //code needed to write a string to PIC32


}

void write(const uint8_t * buffer, size_t size)
{

            //code needed to write a buffer to PIC32

  void SendD(const char *buffer, UINT32 size)

  while(size)
  {
        while(!UARTTransmitterIsReady(UART_MODULE_ID))
            ;

        UARTSendDataByte(UART_MODULE_ID, *buffer);

        buffer++;
        size--;
    }

    while(!UARTTransmissionHasCompleted(UART_MODULE_ID))
        ;

}

/*int available()
{
    int incomingByte = 0;           // for incoming serial data

    void setup() {

                void begin();           // opens serial port
        }

    void loop() {

                // send data only when you receive data:
                if (available() > 0) {
                            // read the incoming byte:
                            incomingByte = int read();

                }
} */


 //Get the number of bytes (characters) available for reading from the serial port.
 //This is data that's already arrived and stored in the serial receive buffer
 //(which holds 64 bytes). available() inherits from the Stream utility class.

       //need to check the available function on PIC32 and C++
//}

//int flush()
//{
    //fflush(port)

    // code to waits for the transmission of outgoing serial data to complete.
//}

//int peek()
//{

   // x = peek(a)


  //return 0;

       //Returns the next byte (character) of incoming serial data without
                   //removing it from the internal serial buffer. That is, successive
                   //calls to peek() will return the same character, as will the next
                   //call to read(). peek() inherits from the Stream utility class.
//}

/*void delayMilliseconds(unsigned long ms)
{
    //create a sleep function for the camera
    return delay(ms);
```

```
}

unsigned long milliseconds()
{
   //create a timer in millseconds
   return millis();
}
*/

/***************************************************************************//**
* @file
* @par MIT License - TERMS OF USE:
* @n Permission is hereby granted, free of charge, to any person obtaining a
* copy of this software and associated documentation files (the "Software"), to
* deal in the Software without restriction, including without limitation the
* rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
* sell copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
* @n
* @n The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
* @n
* @n THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*******************************************************************************/
```

ORIGINAL PORTABLE SERIAL AND TIMER WRAPPER LIBRARY – C++ ARDUINO HEADER FILE

```
/***************************************************************************//**
* @file
* Portable serial and timer wrapper library.
*
* @version @n 1.1
* @date @n 2/7/2013
*
* @authors @n Kwabena W. Agyeman & Christopher J. Leaf
* @copyright @n (c) 2013 Kwabena W. Agyeman & Christopher J. Leaf
* @n All rights reserved - Please see the end of the file for the terms of use
*
* @par Update History:
* @n v0.1 - Beta code - 3/20/2012
* @n v0.9 - Original release - 4/18/2012
* @n v1.0 - Documented and updated release - 8/3/2012
* @n v1.1 - Added support for the Arduino Due, fixed the send frame command,
           and fixed a number of compile time warnings - 2/7/2013.
*******************************************************************************/

#ifndef _CMUCOM4_H_
#define _CMUCOM4_H_

/**@cond CMUCOM4_PRIVATE*****************************************************/

// Handle Arduino Library renaming.
#if defined(ARDUINO) && (ARDUINO >= 100)
```

```
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

// Try to save RAM for non-Mega boards.
#if defined(__AVR_ATmega1280__) || \
        defined(__AVR_ATmega2560__) || \
        defined(__SAM3X8E__)
#define CMUCOM4_INPUT_BUFFER_SIZE   256 ///< Responce input buffer size.
#define CMUCOM4_OUTPUT_BUFFER_SIZE  256 ///< Command output buffer size.
#else
#define CMUCOM4_INPUT_BUFFER_SIZE   160 ///< Responce input buffer size.
#define CMUCOM4_OUTPUT_BUFFER_SIZE  96 ///< Command output buffer size.
#endif

/***************************************************************************//**
* This function macro expands whatever argument name that was passed to this
* function macro into a string. @par For example:
* <tt>@#define ARDUINO 100</tt> @n
* <tt>%CMUCOM4_N_TO_S(ARDUINO)</tt> exapands to @c "ARDUINO"
*******************************************************************************/
#define CMUCOM4_N_TO_S(x)              #x

/***************************************************************************//**
* This function macro expands whatever argument value that was passed to this
* function macro into a string. @par For example:
* <tt>@#define ARDUINO 100</tt> @n
* <tt>%CMUCOM4_V_TO_S(ARDUINO)</tt> exapands to @c "100"
*******************************************************************************/
#define CMUCOM4_V_TO_S(x)          CMUCOM4_N_TO_S(x)

/***************************************************************************//**
* Default firmware startup baud rate number.
*******************************************************************************/
#define CMUCOM4_SLOW_BAUD_RATE              19200

/***************************************************************************//**
* Default firmware startup baud rate string.
*******************************************************************************/
#define CMUCOM4_SLOW_BR_STRING      CMUCOM4_V_TO_S(CMUCOM4_SLOW_BAUD_RATE)

/***************************************************************************//**
* Version 1.01 firmware and below maximum baud rate number.
*******************************************************************************/
#define CMUCOM4_MEDIUM_BAUD_RATE         115200

/***************************************************************************//**
* Version 1.01 firmware and below maximum baud rate string.
*******************************************************************************/
#define CMUCOM4_MEDIUM_BR_STRING    CMUCOM4_V_TO_S(CMUCOM4_MEDIUM_BAUD_RATE)

/***************************************************************************//**
* Version 1.02 firmware and above maximum baud rate number.
*******************************************************************************/
#define CMUCOM4_FAST_BAUD_RATE              250000

/***************************************************************************//**
* Version 1.02 firmware and above maximum baud rate string.
*******************************************************************************/
#define CMUCOM4_FAST_BR_STRING      CMUCOM4_V_TO_S(CMUCOM4_FAST_BAUD_RATE)
```

```
/***************************************************************************//**
* Default firmware startup stop bits number.
*******************************************************************************/
#define CMUCOM4_SLOW_STOP_BITS               0

/***************************************************************************//**
* Default firmware startup stop bits string.
*******************************************************************************/
#define CMUCOM4_SLOW_SB_STRING     CMUCOM4_V_TO_S(CMUCOM4_SLOW_STOP_BITS)

/***************************************************************************//**
* Version 1.01 firmware and below necessary stop bits number.
*******************************************************************************/
#define CMUCOM4_MEDIUM_STOP_BITS             0

/***************************************************************************//**
* Version 1.01 firmware and below necessary stop bits string.
*******************************************************************************/
#define CMUCOM4_MEDIUM_SB_STRING   CMUCOM4_V_TO_S(CMUCOM4_MEDIUM_STOP_BITS)

/***************************************************************************//**
* Version 1.02 firmware and above necessary stop bits number.
*******************************************************************************/
#define CMUCOM4_FAST_STOP_BITS               0

/***************************************************************************//**
* Version 1.02 firmware and above necessary stop bits string.
*******************************************************************************/
#define CMUCOM4_FAST_SB_STRING     CMUCOM4_V_TO_S(CMUCOM4_FAST_STOP_BITS)

/***************************************************************************//**
* Serial CMUcom4::begin() post delay in milliseconds.
*******************************************************************************/
#define CMUCOM4_BEGIN_DELAY      1

/***************************************************************************//**
* Serial CMUcom4::end() post delay in milliseconds.
*******************************************************************************/
#define CMUCOM4_END_DELAY        1

/**@endcond*******************************************************************/

/***************************************************************************//**
* This is a convenient macro for specifying the Serial port when initializing a
* CMUcam4 or CMUcom4 object.
*******************************************************************************/
#define CMUCOM4_SERIAL           0

/***************************************************************************//**
* This is a convenient macro for specifying the Serial1 port on an Arduino Mega
* when initializing a CMUcam4 or CMUcom4 object.
*******************************************************************************/
#define CMUCOM4_SERIAL1          1

/***************************************************************************//**
* This is a convenient macro for specifying the Serial2 port on an Arduino Mega
* when initializing a CMUcam4 or CMUcom4 object.
*******************************************************************************/
#define CMUCOM4_SERIAL2     2
```

```
/***************************************************************************//**
* This is a convenient macro for specifying the Serial3 port on an Arduino Mega
* when initializing a CMUcam4 or CMUcom4 object.
*******************************************************************************/
#define CMUCOM4_SERIAL3                  3

/***************************************************************************//**
* This is a hardware abstraction layer for the %CMUcam4 class. The %CMUcom4
* class targets the Ardunio prototyping platform by default.
*******************************************************************************/
class CMUcom4
{

public:

/***************************************************************************//**
* Initialize the %CMUcom4 object to use the default Serial port.
*******************************************************************************/
CMUcom4();

/***************************************************************************//**
* Initialize the %CMUcom4 object to use the @c port Serial port.
* @param [in] port The port.
* @see CMUCOM4_SERIAL
* @see CMUCOM4_SERIAL1
* @see CMUCOM4_SERIAL2
* @see CMUCOM4_SERIAL3
*******************************************************************************/
CMUcom4(int port);

/***************************************************************************//**
* Arduino Serial.begin() wrapper.
* @param [in] baud In bits per second.
* @see http://arduino.cc/en/Serial/Begin
*******************************************************************************/
void begin(unsigned long baud);

/***************************************************************************//**
* Arduino Serial.end() wrapper.
* @see http://arduino.cc/en/Serial/End
*******************************************************************************/
void end();

/***************************************************************************//**
* Arduino Serial.read() wrapper.
* @return The first byte of incoming serial data.
* @see http://arduino.cc/en/Serial/Read
*******************************************************************************/
int read();

/***************************************************************************//**
* Arduino Serial.write() wrapper.
* @param [in] buffer An array to send as a series of bytes.
* @param [in] size The size of the buffer.
* @return The number of bytes written.
* @see http://arduino.cc/en/Serial/Write
*******************************************************************************/
size_t write(const uint8_t * buffer, size_t size);

/***************************************************************************//**
* Arduino Serial.write() wrapper.
* @param [in] str A string to send as a series of bytes.
* @return The number of bytes written.
* @see http://arduino.cc/en/Serial/Write
*******************************************************************************/
size_t write(const char * str);

/*******************************************************************************//**
* Arduino Serial.write() wrapper.
* @param [in] c A character to send as a single byte.
* @return The number of bytes written.
* @see http://arduino.cc/en/Serial/Write
*******************************************************************************/
size_t write(uint8_t c);

/*******************************************************************************//**
* Arduino Serial.available() wrapper.
* @return The number of bytes available to be read.
* @see http://arduino.cc/en/Serial/Available
*******************************************************************************/
int available();

/*******************************************************************************//**
* Arduino Serial.flush() wrapper.
* @see http://arduino.cc/en/Serial/Flush
*******************************************************************************/
void flush();

/*******************************************************************************//**
* Arduino Serial.peek() wrapper.
* @return The first byte of incoming serial data available.
* @see http://arduino.cc/en/Serial/Peek
*******************************************************************************/
int peek();

/*******************************************************************************//**
* Arduino delay() wrapper.
* @param [in] ms The number of milliseconds to pause for.
* @see http://arduino.cc/en/Reference/Delay
*******************************************************************************/
void delayMilliseconds(unsigned long ms);

/*******************************************************************************//**
* Arduino millis() wrapper.
* @return Number of milliseconds since the program started.
* @see http://arduino.cc/en/Reference/Millis
*******************************************************************************/
unsigned long milliseconds();

private:

/*******************************************************************************//**
* Selected serial port storage.
* @see CMUCOM4_SERIAL1
* @see CMUCOM4_SERIAL2
* @see CMUCOM4_SERIAL3
*******************************************************************************/
int _port;
};

#endif
```

```
/***************************************************************************//**
* @file
* @par MIT License - TERMS OF USE:
* @n Permission is hereby granted, free of charge, to any person obtaining a
* copy of this software and associated documentation files (the "Software"), to
* deal in the Software without restriction, including without limitation the
* rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
* sell copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
* @n
* @n The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
* @n
* @n THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*******************************************************************************/
```

MODIFIED PORTABLE SERIAL AND TIMER WRAPPER LIBRARY – PIC32 HEADER FILE

```
/***************************************************************************//**
* @file
* Portable serial and timer wrapper library.
*
* @version @n 1.1
* @date @n 2/7/2013
*
* @authors @n Kwabena W. Agyeman & Christopher J. Leaf
* @copyright @n (c) 2013 Kwabena W. Agyeman & Christopher J. Leaf
* @n All rights reserved - Please see the end of the file for the terms of use
*
* @par Update History:
* @n v0.1 - Beta code - 3/20/2012
* @n v0.9 - Original release - 4/18/2012
* @n v1.0 - Documented and updated release - 8/3/2012
* @n v1.1 - Added support for the Arduino Due, fixed the send frame command,
          and fixed a number of compile time warnings - 2/7/2013.
*******************************************************************************/
#include <plib.h>


#ifndef _CMUCOM4_H_
#define _CMUCOM4_H_

#if defined (__32MX695F512H__)

// Configuration Bit settings
// SYSCLK = 80 MHz (8MHz Crystal / FPLLIDIV * FPLLMUL / FPLLODIV)
// PBCLK = 80 MHz (SYSCLK / FPBDIV)


/**@cond CMUCOM4_PRIVATE*******************************************************/



#define CMUCOM4_INPUT_BUFFER_SIZE   256 ///< Responce input buffer size.
#define CMUCOM4_OUTPUT_BUFFER_SIZE  256 ///< Command output buffer size.
//#else
```

```
//#define CMUCOM4_INPUT_BUFFER_SIZE   160 ///< Responce input buffer size.
//#define CMUCOM4_OUTPUT_BUFFER_SIZE  96 ///< Command output buffer size.
//#endif

/***************************************************************************//**
* This function macro expands whatever argument name that was passed to this
* function macro into a string. @par For example:
* <tt>@#define ARDUINO 100</tt> @n
* <tt>%CMUCOM4_N_TO_S(ARDUINO)</tt> exapands to @c "ARDUINO"
*******************************************************************************/
#define CMUCOM4_N_TO_S(x)          #x

/***************************************************************************//**
* This function macro expands whatever argument value that was passed to this
* function macro into a string. @par For example:
* <tt>@#define ARDUINO 100</tt> @n
* <tt>%CMUCOM4_V_TO_S(ARDUINO)</tt> exapands to @c "100"
*******************************************************************************/
#define CMUCOM4_V_TO_S(x)          CMUCOM4_N_TO_S(x)

/***************************************************************************//**
* Default firmware startup baud rate number.
*******************************************************************************/
#define CMUCOM4_SLOW_BAUD_RATE      19200

/***************************************************************************//**
* Default firmware startup baud rate string.
*******************************************************************************/
#define CMUCOM4_SLOW_BR_STRING      CMUCOM4_V_TO_S(CMUCOM4_SLOW_BAUD_RATE)

/***************************************************************************//**
* Version 1.01 firmware and below maximum baud rate number.
*******************************************************************************/
#define CMUCOM4_MEDIUM_BAUD_RATE    115200

/***************************************************************************//**
* Version 1.01 firmware and below maximum baud rate string.
*******************************************************************************/
#define CMUCOM4_MEDIUM_BR_STRING    CMUCOM4_V_TO_S(CMUCOM4_MEDIUM_BAUD_RATE)

/***************************************************************************//**
* Version 1.02 firmware and above maximum baud rate number.
*******************************************************************************/
#define CMUCOM4_FAST_BAUD_RATE      250000

/***************************************************************************//**
* Version 1.02 firmware and above maximum baud rate string.
*******************************************************************************/
#define CMUCOM4_FAST_BR_STRING      CMUCOM4_V_TO_S(CMUCOM4_FAST_BAUD_RATE)

/***************************************************************************//**
* Default firmware startup stop bits number.
*******************************************************************************/
#define CMUCOM4_SLOW_STOP_BITS      0

/***************************************************************************//**
* Default firmware startup stop bits string.
*******************************************************************************/
#define CMUCOM4_SLOW_SB_STRING      CMUCOM4_V_TO_S(CMUCOM4_SLOW_STOP_BITS)

/***************************************************************************//**
* Version 1.01 firmware and below necessary stop bits number.
*******************************************************************************/
#define CMUCOM4_MEDIUM_STOP_BITS    0

/***************************************************************************//**
```

```
* Version 1.01 firmware and below necessary stop bits string.
*******************************************************************************/
#define CMUCOM4_MEDIUM_SB_STRING    CMUCOM4_V_TO_S(CMUCOM4_MEDIUM_STOP_BITS)

/***************************************************************************//**
* Version 1.02 firmware and above necessary stop bits number.
*******************************************************************************/
#define CMUCOM4_FAST_STOP_BITS      0

/***************************************************************************//**
* Version 1.02 firmware and above necessary stop bits string.
*******************************************************************************/
#define CMUCOM4_FAST_SB_STRING      CMUCOM4_V_TO_S(CMUCOM4_FAST_STOP_BITS)

/***************************************************************************//**
* Serial CMUcom4::begin() post delay in milliseconds.
*******************************************************************************/
#define CMUCOM4_BEGIN_DELAY         1

/***************************************************************************//**
* Serial CMUcom4::end() post delay in milliseconds.
*******************************************************************************/
#define CMUCOM4_END_DELAY           1

/**@endcond*******************************************************************/

/***************************************************************************//**
* This is a convenient macro for specifying the Serial port when initializing a
* CMUcam4 or CMUcom4 object.
*******************************************************************************/
#define CMUCOM4_SERIAL              0

/***************************************************************************//**
* This is a convenient macro for specifying the Serial1 port on an Arduino Mega
* when initializing a CMUcam4 or CMUcom4 object.
*******************************************************************************/
#define CMUCOM4_SERIAL1             1

/***************************************************************************//**
* This is a convenient macro for specifying the Serial2 port on an Arduino Mega
* when initializing a CMUcam4 or CMUcom4 object.
*******************************************************************************/
#define CMUCOM4_SERIAL2             2

/***************************************************************************//**
* This is a convenient macro for specifying the Serial3 port on an Arduino Mega
* when initializing a CMUcam4 or CMUcom4 object.
*******************************************************************************/
#define CMUCOM4_SERIAL3             3

/***************************************************************************//**
* This is a hardware abstraction layer for the %CMUcam4 class. The %CMUcom4
* class targets the Ardunio prototyping platform by default.
*******************************************************************************/
class CMUcom4
{

public:

/***************************************************************************//**
* Initialize the %CMUcom4 object to use the default Serial port.
*******************************************************************************/
CMUcom4();

/***************************************************************************//**
* Initialize the %CMUcom4 object to use the @c port Serial port.
* @param [in] port The port.
* @see CMUCOM4_SERIAL
* @see CMUCOM4_SERIAL1
* @see CMUCOM4_SERIAL2
* @see CMUCOM4_SERIAL3
*******************************************************************************/
CMUcom4(int port);

/***************************************************************************//**
* Arduino Serial.begin() wrapper.
* @param [in] baud In bits per second.
* @see http://arduino.cc/en/Serial/Begin
*******************************************************************************/
void begin(unsigned long baud);

/***************************************************************************//**
* Arduino Serial.end() wrapper.
* @see http://arduino.cc/en/Serial/End
*******************************************************************************/
void end(void);

/***************************************************************************//**
* Arduino Serial.read() wrapper.
* @return The first byte of incoming serial data.
* @see http://arduino.cc/en/Serial/Read
*******************************************************************************/
int read(void);

/***************************************************************************//**
* Arduino Serial.write() wrapper.
* @param [in] buffer An array to send as a series of bytes.
* @param [in] size The size of the buffer.
* @return The number of bytes written.
* @see http://arduino.cc/en/Serial/Write
*******************************************************************************/

int write(const uint8_t *, size_t);

/***************************************************************************//**
* Arduino Serial.write() wrapper.
* @param [in] str A string to send as a series of bytes.
* @return The number of bytes written.
* @see http://arduino.cc/en/Serial/Write
*******************************************************************************/
int write(const char * str);

/***************************************************************************//**
* Arduino Serial.write() wrapper.
* @param [in] c A character to send as a single byte.
* @return The number of bytes written.
* @see http://arduino.cc/en/Serial/Write
*******************************************************************************/

size_t write(uint8_t c);

/***************************************************************************//**
* Arduino Serial.available() wrapper.
* @return The number of bytes available to be read.
* @see http://arduino.cc/en/Serial/Available
*******************************************************************************/
int available();

/***************************************************************************//**
* Arduino Serial.flush() wrapper.
* @see http://arduino.cc/en/Serial/Flush
*******************************************************************************/
```

```
//void flush();

/*******************************************************************************//**
* Arduino Serial.peek() wrapper.
* @return The first byte of incoming serial data available.
* @see http://arduino.cc/en/Serial/Peek
*******************************************************************************/
//int peek();

/*******************************************************************************//**
* Arduino delay() wrapper.
* @param [in] ms The number of milliseconds to pause for.
* @see http://arduino.cc/en/Reference/Delay
*******************************************************************************/
//void delayMilliseconds(unsigned long ms);

/*******************************************************************************//**
* Arduino millis() wrapper.
* @return Number of milliseconds since the program started.
* @see http://arduino.cc/en/Reference/Millis
*******************************************************************************/
//unsigned long milliseconds();

private:

/*******************************************************************************//**
* Selected serial port storage.
* @see CMUCOM4_SERIAL1
* @see CMUCOM4_SERIAL2
* @see CMUCOM4_SERIAL3
*******************************************************************************/
int _port;

};

#endif

/*******************************************************************************//**
* @file
* @par MIT License - TERMS OF USE:
* @n Permission is hereby granted, free of charge, to any person obtaining a
* copy of this software and associated documentation files (the "Software"), to
* deal in the Software without restriction, including without limitation the
* rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
* sell copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
* @n
* @n The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
* @n
* @n THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*******************************************************************************
```

UART PIC32 MAIN FILE

```
/*
 * File:   Assignment8.c
 * Author: nferruol
 *
 * Created on November 26, 2012, 4:33 PM
 */

#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include <sys/attribs.h>
#include <plib.h>
#include "configbits.h"
#include "CMUcom4.h"

/*
char uart3Read(void)
{
  while (U3STAbits.URXDA == 0)
  {
  }

  return(U3RXREG);
}*/

/*void uart3Write(char a)
{
   while (U3STAbits.UTXBF == 1)
   {
   }

   U3TXREG = a;
 }*/

/*void _mon_putc(char c){
    while (U3STAbits.UTXBF);
    U3TXREG = c;
} */

int main(void)
{
                uint8_t c;

                uart_init(BAUD_RATE);
                while (1) {
                                if (uart_available()) {
                                                c = uart_getchar();
                                                uart_print("Byte: ");
                                                uart_putchar(c);
                                                uart_putchar('\r');
                                                uart_putchar('\n');
                                }
                }
}
```

Matlab DetectRed.m function

```
 %% Now to track red objects in real time
    % we have to subtract the red component
    % from the grayscale image to extract the red components in the image.
    diff_im = imsubtract(data(:,:,1), rgb2gray(data));
    %Use a median filter to filter out noise
```

```
diff_im = medfilt2(diff_im, [3 3]);
% Convert the resulting grayscale image into a binary image.
diff_im = im2bw(diff_im,0.19);
% Remove all those pixels less than 300px
diff_im = bwareaopen(diff_im,300);
% Label all the connected components in the image.
bw = bwlabel(diff_im, 8);

% Here we do the image blob analysis.
% We get a set of properties for each labeled region.
stats = regionprops(bw, 'BoundingBox', 'Centroid');

%% Now to track green objects in real time
% we have to subtract the red component
% from the grayscale image to extract the red components in the image.
diff_im2 = imsubtract(data(:,:,2), rgb2gray(data));
%Use a median filter to filter out noise
diff_im2 = medfilt2(diff_im2, [3 3]);
% Convert the resulting grayscale image into a binary image.
diff_im2 = im2bw(diff_im2,0.05);
% Remove all those pixels less than 300px
diff_im2 = bwareaopen(diff_im2,300);
% Label all the connected components in the image.
bw2 = bwlabel(diff_im2, 8);

% Here we do the image blob analysis.
% We get a set of properties for each labeled region.
stats2 = regionprops(bw2, 'BoundingBox', 'Centroid');

%% Display the image
coder.extrinsic('imshow','hold on','rectangle','plot','text','set','hold off');

imshow(data)

hold on

%This is a loop to bound the red objects in a rectangular box.
for object = 1:length(stats)
   %for red
   bb = stats(object).BoundingBox;
   bc = stats(object).Centroid;
   rectangle('Position',bb,'EdgeColor','r','LineWidth',2)
   plot(bc(1),bc(2), '-m+')
   a=text(bc(1)+15,bc(2), strcat('X: ', num2str(round(bc(1))), '    Y: ', num2str(round(bc(2)))));
   set(a, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 12, 'Color', 'yellow');
end

for object = 1:length(stats2)
   %for green
   bb2 = stats2(object).BoundingBox;
   bc2 = stats2(object).Centroid;
   rectangle('Position',bb2,'EdgeColor','g','LineWidth',2)
   plot(bc2(1),bc2(2), '-m+')
   a=text(bc2(1)+15,bc2(2), strcat('X: ', num2str(round(bc2(1))), '    Y: ', num2str(round(bc2(2)))));
   set(a, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 12, 'Color', 'yellow');
end

hold off
```

Matlab RedTrackObject.m Test Program

```
a = imaqhwinfo;
allStats = struct([]);
%[camera_name, camera_id, format] = getCameraInfo(a);


% Capture the video frames using the videoinput function
% You have to replace the resolution & your installed adaptor name.
%vid = videoinput(camera_name, camera_id, format);
vid = videoinput('winvideo',2,'YUY2_1024x768');
%vid = videoinput('matrox',,'Port_#0002.Hub_#0004');

% Set the properties of the video object
set(vid, 'FramesPerTrigger', Inf);
set(vid, 'ReturnedColorspace', 'rgb')
vid.FrameGrabInterval = 8;


%start the video aquisition here
start(vid)

flushdata(vid);

%call function that process data frame by frame

while(vid.FramesAcquired<=400)

    % Get the snapshot of the current frame
    data = getsnapshot(vid);

    [diff_im,bw,stats,diff_im2,bw2,stats2]=DetectRed(data);

    % Create a construct that stores all stats

    allStats([vid.FramesAcquired]).red = stats;
    allStats([vid.FramesAcquired]).green = stats2;

end
% Stop the video aquisition.
stop(vid);

% Flush all the image data stored in the memory buffer.
flushdata(vid);

% Clear all variables
%clear all
sprintf('%s','That was all about Image tracking :) ')
```

Microcontroller initialization configbits_1.h

```
/*
 * File:   configbits_1.h
 * Author: Mike
 *
 * Created on October 9, 2012, 1:50 PM
 */

#ifndef CONFIGBITS_H
#define     CONFIGBITS_H
```

```
/* 20 MHz crystal run at 80 mhz

 peripher clock = at 10 MHz (80 MHz/8)

 */

#pragma config FNOSC = FRCPLL // Oscillator selection
#pragma config POSCMOD = OFF // Primary oscillator mode
#pragma config FPLLIDIV = DIV_5 // PLL input divider (20 -> 4)
#pragma config FPLLMUL = MUL_20 // PLL multiplier  ( 4x20 = 80)
#pragma config FPLLODIV = DIV_1 // PLL output divider
#pragma config FPBDIV = DIV_8 // Peripheral bus clock divider 10 mhz
#pragma config FSOSCEN = OFF // Secondary oscillator enable
/* Clock control settings
*/
#pragma config IESO = OFF // Internal/external clock switchover
#pragma config FCKSM = CSDCMD // Clock switching (CSx)/Clock monitor (CMx)
#pragma config OSCIOFNC = OFF // Clock output on OSCO pin enable
/* USB Settings
*/
#pragma config UPLLEN = ON // USB PLL enable
#pragma config UPLLIDIV = DIV_2 // USB PLL input divider
#pragma config FVBUSONIO = OFF // VBUS pin control
#pragma config FUSBIDIO = OFF // USBID pin control
/* Other Peripheral Device settings
*/
#pragma config FWDTEN = OFF // Watchdog timer enable
#pragma config WDTPS = PS1024 // Watchdog timer post-scaler
#pragma config FSRSSEL = PRIORITY_7 // SRS interrupt priority

#pragma config        ICESEL    = ICS_PGx1              // ICE pin selection
#endif      /* CONFIGBITS_H */
```

Zigbee_1.h

```
/*
 * File:   Zigbee_1.h
 * Author: nferruol
 *
 *
 */

#ifndef ZIGBEE_H
#define     ZIGBEE_H

#ifdef       __cplusplus
extern "C" {
#endif



//Define Registers
#define TRX_STATUS          0x01
#define TRX_STATE           0x02
#define TRX_CTRL_0          0x03
#define PHY_TX_PWR          0x05
#define PHY_RSSI           0x06
#define PHY_ED_LEVEL        0x07
#define PHY_CC_CCA          0x08
#define PHY_CCA_THRES       0x09
```

```
#define IRQ_MASK            0x0E
#define IRQ_STATUS          0x0F
#define VREG_CTRL           0x10
#define BATMON              0x11
#define XOSC_CTRL           0x12
#define PLL_CF              0x1A
#define PLL_DCU             0x1B
#define PART_NUM            0x1C
#define VERSION_NUM         0x1D
#define MAN_ID_0            0x1E
#define MAN_ID_1            0x1F
#define SHORT_ADDR_0        0x20
#define SHORT_ADDR_1        0x21
#define PAN_ID_0            0x22
#define PAN_ID_1            0x23
#define IEEE_ADDR_0         0x24
#define IEEE_ADDR_1         0x25
#define IEEE_ADDR_2         0x26
#define IEEE_ADDR_3         0x27
#define IEEE_ADDR_4         0x28
#define IEEE_ADDR_5         0x29
#define IEEE_ADDR_6         0x2A
#define IEEE_ADDR_7         0x2B
#define XAH_CTRL            0x2C
#define CSMA_SEED_0         0x2D
#define CSMA_SEED_1         0x2E

//States written to TRX_STATE
#define TX_START            0x02
#define FORCE_TRX_OFF       0x03
#define RX_ON               0x06
#define TRX_OFF             0x08
#define PLL_ON              0x09

//States read out of TRX_STATUS
#define P_ON                0x00
#define BUSY_RX             0x01
#define BUSY_TX             0x02
#define RX_ON               0x06
#define TRX_OFF             0x08
#define PLL_ON              0x09
#define SLEEP               0x0F
#define STATE_TRANSITION    0x1F

//Define pins
#define sel                 LATFbits.LATF5
#define rst                 LATBbits.LATB5
#define slp                 LATBbits.LATB4

 //Function Prototypes
char check();
void send(char);
void SPI_initialize();
void UART_initialize();
void final(int);
void ZIGBEE_initialize();
void ZIGBEE_pll();
void ZIGBEE_transmit(int, int,int, int,int, int,int, int,int);
void ZIGBEE_receive();
void ZIGBEE_write_register (int, int);
int ZIGBEE_read_register (int);
```

```
void ZIGBEE_write_buffer (int, int,int, int,int, int,int, int,int);
void ZIGBEE_read_buffer();
int ZIGBEE_check_interrupt();



#ifdef        __cplusplus
}
#endif

#endif       /* ZIGBEE_H */
```

ZIGBEE_FUNCTIONS.c

```
/*
 * File:   ZIGBEE_FUNCTIONS.c
 * Author: nferruol
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include <plib.h>
#include <sys/attribs.h>
#include "Zigbee_1.h"

void UART_initialize (){
    //Set all bits to digital
    AD1PCFG = 0xFFFF;

    U4MODEbits.BRGH=1;
    U4BRG = 42; // Set Baud rate 9600= 259 57600=42
    U4MODEbits.PDSEL=0;
    U4MODEbits.STSEL=0;
    U4STAbits.UTXEN=1;
    U4STAbits.URXEN=1;
    U4MODEbits.ON=1;
    U4MODEbits.UARTEN = 0x01;

}

char check(void)
{
  while (U4STAbits.URXDA == 0)
  {
  }

  return(U4RXREG);

}

void send(char a)
{
   while (U4STAbits.UTXBF == 1)
   {
   }

   U4TXREG = a;

 }
 void _mon_putc (char c)
 {   while (U4STAbits.UTXBF);
     U4TXREG = c;
 }

void SPI_initialize (){

    //set rst, slp, sel to outputs
    TRISFbits.TRISF5 = 0;       //sel
    TRISBbits.TRISB4 = 0;       //slp
    TRISBbits.TRISB5 = 0;       //rst

    //set rst, sel high  set slp low
    rst = 1;
    sel = 1;
    slp = 0;

    //disable interrupts
    IEC1bits.SPI2RXIE = 0;
    IEC1bits.SPI2EIE  = 0;
    IEC1bits.SPI2TXIE = 0;

    //stops and resets SPI2
    SPI2CONbits.ON = 0;

    //clears the buffer
    int clear = SPI2BUF;

    //standard mode
    SPI2CONbits.ENHBUF = 0;

    //use FPB/4 clock frequency Fsck = FPB/(2*(SPIxBRG+1))    FPB = 40MHz
    SPI2BRG = 42;


    //Turn SPI2 on, 8 bits transfer, master mode
    SPI2CONbits.MODE32 = 0;
    SPI2CONbits.MODE16 = 0;
    SPI2CONbits.MSTEN = 1;      //Master mode
    SPI2CONbits.MSSEN = 0;      //Manually do slave select
    SPI2CONbits.SSEN = 0;       //Not slave mode
    SPI2CONbits.CKP = 0;        //Low idle, high active
    SPI2CONbits.CKE = 1;        //Change output on falling edge
    SPI2CONbits.SMP = 0;        //Sample MISO at middle of clock pulse
    SPI2CONbits.DISSDO = 0;     //MISO is enabled
    //Must be final command
    SPI2CONbits.ON = 1;
    //Ready to transmit or receive via SPI2BUF
}



void ZIGBEE_initialize(){
    rst = 0;
    int k;
    for  (k =0; k<100000;k++)
    {}
    rst = 1;

    //ZIGBEE will start up in P_ON state or has been reset
    //Set up IF from ZIGBEE
    TRISDbits.TRISD8 = 1;

    //Put in TRX_OFF state (clock state)
    ZIGBEE_write_register(TRX_STATE, TRX_OFF);

    //Set channel with PHY_CC_CCA channel #11, 2405MHz 0B on bits 4:0
    ZIGBEE_write_register(PHY_CC_CCA, 0x2B);

    //Enable pertinant interupts TRX_END, PLL_LOCK
    ZIGBEE_write_register(IRQ_MASK, 0b00001001);

    //Turns on automatic FCS appending
    ZIGBEE_write_register(PHY_TX_PWR, 0xC6);

    //Set short address to 0123
    ZIGBEE_write_register(SHORT_ADDR_1, 0x01);
    ZIGBEE_write_register(SHORT_ADDR_0, 0x23);

    //Set pan id to 4567
    ZIGBEE_write_register(PAN_ID_1, 0x45);
    ZIGBEE_write_register(PAN_ID_0, 0x67);
}


void ZIGBEE_pll(){
    //Put in PLL_ON state (ready state)
    ZIGBEE_write_register(TRX_STATE, PLL_ON);

    int k;
    for  (k =0; k<1000;k++)
    {}

    return;
}

void ZIGBEE_transmit(int message1, int message2, int message3, int message4, int message5, int message6, int message7, int
message8, int message9)
{
    //Write info to be sent to buffer
    ZIGBEE_write_buffer(message1, message2, message3,message4,message5,message6,message7,message8,message9);

    //Put in Transmit state
    ZIGBEE_write_register(TRX_STATE, TX_START);

    //Return to PLL_ON
    ZIGBEE_pll();
}

void ZIGBEE_receive (){
    //Put in Receive state
    ZIGBEE_write_register(TRX_STATE, RX_ON);

    //Wait for message to be delivered
    while(!PORTDbits.RD8) {}
    int result = ZIGBEE_check_interrupt();
    if(result == 1) //Wrong interrupt, reset
    {
        ZIGBEE_initialize;
        ZIGBEE_pll;
        ZIGBEE_receive();
    }
    else

    ZIGBEE_read_buffer();

    return;
}
void ZIGBEE_write_register (int address, int message){
    int receive;
    sel = 0;

    //or write command with address
    SPI2BUF = (0b11000000 | address);
    //wait for full recieve buffer so you can write again
    while (!SPI2STATbits.SPIRBF) {}
    receive = SPI2BUF;

    //write message to given register
    SPI2BUF = message;
    //wait for full recieve buffer to confirm transmission
    while (!SPI2STATbits.SPIRBF) {}
    receive = SPI2BUF;

    sel = 1;
}

int ZIGBEE_read_register (int address){
    int receive;
    int message;
    sel = 0;

    //or read command with address
    SPI2BUF = (0b10000000 | address);
   //wait for full recieve buffer so you can write again
    while (!SPI2STATbits.SPIRBF) {}
    receive = SPI2BUF;

    //0x00 to get response
    SPI2BUF = 0x00;
    //wait for full recieve buffer
    while (!SPI2STATbits.SPIRBF) {}
    message = SPI2BUF;

    sel = 1;
    return message;
}

void ZIGBEE_write_buffer(int message1, int message2, int message3, int message4, int message5, int message6, int message7, int
message8, int message9) {
    //can include longer messages just need to send it as 8 bit packages message, message1, message2 etc. Be sure to change payload bytes
    sel = 0;
    //set number of payload bytes using 8 bits
    int payload = 0x0A;

    //give write to buffer command
    SPI2BUF = 0b01100000;
    //wait for full recieve buffer to confirm transmission
    while (!SPI2STATbits.SPIRBF) {};
    int receive = SPI2BUF;

    //give number of payload bytes
```

```
 int numbytes = payload + 13;
 SPI2BUF = numbytes;
 //wait for full recieve buffer to confirm transmission
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 //write MAC Frame Format

 SPI2BUF = 0x01;          //Frame Control 1st
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

SPI2BUF = 0x88;          //Frame Control 2nd
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 SPI2BUF = 0x00;          //Sequence Number
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 SPI2BUF = 0xEF;          //Destination PAN ID 1
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 SPI2BUF = 0xEF;          //Destination PAN ID 2
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 SPI2BUF = 0xCD;          //Destination address 1
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 SPI2BUF = 0xAB;          //Destination address 2
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 SPI2BUF = 0x67;          //Source PAN ID 1
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 SPI2BUF = 0x45;          //Source PAN ID 2
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 SPI2BUF = 0x23;          //Source address 1
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 SPI2BUF = 0x01;          //Source address 2
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 //write payload byte 1 to ZIGBEE buffer
 SPI2BUF = message1;
 //wait for full recieve buffer to confirm transmission
 while (!SPI2STATbits.SPIRBF) {};
 receive = SPI2BUF;

 //write payload byte 2 to ZIGBEE buffer
 SPI2BUF = message2;
 //wait for full recieve buffer to confirm transmission
```

```
    while (!SPI2STATbits.SPIRBF) {};
    receive = SPI2BUF;

    //write payload byte 3 to ZIGBEE buffer
    SPI2BUF = message3;
    //wait for full recieve buffer to confirm transmission
    while (!SPI2STATbits.SPIRBF) {};
    receive = SPI2BUF;

    //4
    SPI2BUF = message4;
    while (!SPI2STATbits.SPIRBF) {};
    receive = SPI2BUF;

    //5
    SPI2BUF = message5;
    while (!SPI2STATbits.SPIRBF) {};
    receive = SPI2BUF;

    //6
    SPI2BUF = message6;
    while (!SPI2STATbits.SPIRBF) {};
    receive = SPI2BUF;

    //7
    SPI2BUF = message7;
    while (!SPI2STATbits.SPIRBF) {};
    receive = SPI2BUF;

    //8
    SPI2BUF = message8;
    while (!SPI2STATbits.SPIRBF) {};
    receive = SPI2BUF;

    //9
    SPI2BUF = message9;
    while (!SPI2STATbits.SPIRBF) {};
    receive = SPI2BUF;

    sel = 1;
}
void ZIGBEE_read_buffer(){
    int numbytes, message1, message2, message3, message4;
    int message5, message6, message7, message8, message9;
    sel = 0;

    //give read buffer command
    SPI2BUF = 0b00100000;
    //wait for full recieve buffer to confirm transmission
    while (!SPI2STATbits.SPIRBF) {};
    int receive = SPI2BUF;

    //Number of payload bytes
    SPI2BUF = 0x00;
    while (!SPI2STATbits.SPIRBF) {};
    numbytes = SPI2BUF;

    //read MAC Frame Format

    SPI2BUF = 0x00;             //Frame Control 1st
    while (!SPI2STATbits.SPIRBF) {};
```

```
int receive1 = SPI2BUF;

SPI2BUF = 0x00;            //Frame Control 2nd
while (!SPI2STATbits.SPIRBF) {};
int receive2 = SPI2BUF;

SPI2BUF = 0x00;             //Sequence Number
while (!SPI2STATbits.SPIRBF) {};
receive = SPI2BUF;

SPI2BUF = 0xEF;             //Destination PAN ID 1
while (!SPI2STATbits.SPIRBF) {};
receive = SPI2BUF;

SPI2BUF = 0xEF;             //Destination PAN ID 2
while (!SPI2STATbits.SPIRBF) {};
receive = SPI2BUF;

SPI2BUF = 0xCD;             //Destination address 1
while (!SPI2STATbits.SPIRBF) {};
receive = SPI2BUF;

SPI2BUF = 0xAB;             //Destination address 2
while (!SPI2STATbits.SPIRBF) {};
receive = SPI2BUF;

SPI2BUF = 0x67;             //Source PAN ID 1
while (!SPI2STATbits.SPIRBF) {};
receive = SPI2BUF;

SPI2BUF = 0x45;             //Source PAN ID 2
while (!SPI2STATbits.SPIRBF) {};
receive = SPI2BUF;

SPI2BUF = 0x23;             //Source address 1
while (!SPI2STATbits.SPIRBF) {};
receive = SPI2BUF;

SPI2BUF = 0x01;             //Source address 2
while (!SPI2STATbits.SPIRBF) {};
receive = SPI2BUF;

//Data 1
SPI2BUF = 0x00;
//wait for full recieve buffer
while (!SPI2STATbits.SPIRBF) {};
message1 = SPI2BUF;

//Data 2
SPI2BUF = 0x00;
//wait for full recieve buffer
while (!SPI2STATbits.SPIRBF) {};
message2 = SPI2BUF;

 //Data 3
SPI2BUF = 0x00;
//wait for full recieve buffer
while (!SPI2STATbits.SPIRBF) {};
message3 = SPI2BUF;

 //Data 4
```

```
    SPI2BUF = 0x00;
    //wait for full recieve buffer
    while (!SPI2STATbits.SPIRBF) {};
    message4 = SPI2BUF;

     //Data 5
    SPI2BUF = 0x00;
    //wait for full recieve buffer
    while (!SPI2STATbits.SPIRBF) {};
    message5 = SPI2BUF;

     //Data 6
    SPI2BUF = 0x00;
    //wait for full recieve buffer
    while (!SPI2STATbits.SPIRBF) {};
    message6 = SPI2BUF;

     //Data 7
    SPI2BUF = 0x00;
    //wait for full recieve buffer
    while (!SPI2STATbits.SPIRBF) {};
    message7 = SPI2BUF;

     //Data 8
    SPI2BUF = 0x00;
    //wait for full recieve buffer
    while (!SPI2STATbits.SPIRBF) {};
    message8 = SPI2BUF;

     //Data 9
    SPI2BUF = 0x00;
    //wait for full recieve buffer
    while (!SPI2STATbits.SPIRBF) {};
    message9 = SPI2BUF;

    sel = 1;
    printf("%7s %19s %18s \n", "Team", "X-Position (in)", "Y-Position (in)");
    printf("%6d %12d %18d \n", message1, message2, message3);
    printf("%6d %12d %18d \n", message1, message4, message5);
    printf("%6d %12d %18d \n", message1, message6, message7);
    printf("%6d %12d %18d \n", message1, message8, message9);

    return;
}

int ZIGBEE_check_interrupt()
{
    int result = 4;
    int problem = ZIGBEE_read_register(IRQ_STATUS);
    //pll locked
    if ((problem & 0x01) == 0x01)
    {
        result = 1;

    }

    //transmission sent, buffer empty OR message received, buffer full
    if ((problem & 0x08) == 0x08)
    {
        result = 2;
    }

    return result;
}
```

ZIGBEE_SEND.c

```
/*
 * File:   ZIGBEE_SEND.c
 * Author: Nick Ferruolo
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include <sys/attribs.h>
#include <plib.h>
#include<p32xxxx.h>
#include "Zigbee_1.h"
#include "configbits_1.h"

void main(){
    int k;
    UART_initialize();
    SPI_initialize();
    ZIGBEE_initialize();
    ZIGBEE_pll();

    while (1) {

    ZIGBEE_transmit(1, 1,5, 2,6, 3,7,  4,8);
    ZIGBEE_transmit(2, 9,13, 10,14, 11,15, 12,16);
    for (k=0;k<5000000;k++){}
    ZIGBEE_transmit(1, 17,21, 18,22, 19,23, 20,24);
    for (k=0;k<5000000;k++){}
    ZIGBEE_transmit(2, 25,29, 26,30, 27,31, 28,32);
    for (k=0;k<5000000;k++){}
    ZIGBEE_transmit(1, 33,37, 34,38, 35,39, 36,40);
    for (k=0;k<5000000;k++){}
    ZIGBEE_transmit(2, 41,45, 42,46, 43,47, 44,48);
    for (k=0;k<5000000;k++){}
    ZIGBEE_transmit(1, 49,53, 50,54, 51,55, 52,56);
    for (k=0;k<5000000;k++){}
    ZIGBEE_transmit(2, 57,61, 58,62, 59,63, 60,64);
    for (k=0;k<5000000;k++){}
    ZIGBEE_transmit(1, 65,69, 66,70, 67,71, 68,72);
    for (k=0;k<5000000;k++){}
    ZIGBEE_transmit(2, 73,77, 74,78, 75,79, 76,80);
    for (k=0;k<5000000;k++){}
    ZIGBEE_transmit(1, 81,85, 82,86, 83,87, 84,88);
    for (k=0;k<5000000;k++){}
    ZIGBEE_transmit(2, 89,93, 90,94, 91,95, 92,96);
    for (k=0;k<5000000;k++){}

    }
}
```

ZIGBEE_RECEIVE.c

```
/*
```

```
 * File:   ZIGBEE_RECEIVE.c
 * Author: Nick Ferruolo
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include <sys/attribs.h>
#include <plib.h>
#include<p32xxxx.h>
#include "Zigbee_1.h"
#include "configbits_1.h"

void main(){
   UART_initialize();
   SPI_initialize();
   ZIGBEE_initialize();
   ZIGBEE_pll();

   while(1){
      ZIGBEE_receive();
   }
}
```