ECE422: FINAL PROJECT REPORT

Digital Acquisition Board and App

Student Name: Anthony Bloch

Date: 12/14/18

Contents

[Background 1](#_Toc532562845)

[Goals and Objectives 1](#_Toc532562846)

[System Requirements, design specifications 1](#_Toc532562847)

[Design and Implementation 2](#_Toc532562848)

[Results 4](#_Toc532562849)

[Development Journal 4](#_Toc532562850)

[Analysis and Discussion 4](#_Toc532562851)

[Conclusion 5](#_Toc532562852)

[Acknowledgement 5](#_Toc532562853)

[Appendix A 5](#_Toc532562854)

# Background

The ECE422 course aims to teach microcontroller system design. Throughout this semester, the labs have covered the PIC24FV16KM202 and the PIC24FJ256GA705 chips, controlling LEDs with the chip, receiving input from push buttons or serial communication connected to the chip, SPI, I2C, analog to digital conversion (ADC), and FreeRTOS. MPLab has been used as the integrated design environment (IDE), and the PIC chips were programmed using C code. A lab and the midterm have also been done which involved developing a graphical user interface (GUI) for a vending machine using C# and Visual Studio. Having completed several smaller projects, it is now time to develop a larger project that incorporates the lessons learned from the previous labs and provides an opportunity to develop code that is organized into multiple source files.

# Goals and Objectives

The goal of this project is to implement an embedded systems application and a GUI that together provide the user with a digital acquisition board and app. The user will be able to turn on/off a green LED. The user will be able to enable a yellow LED to turn on and remain on for a time before turning off. The user can disable the yellow LED to turn it off at any time. The user can also check the status of the yellow LED. The user will be able enable/disable a red LED to blink with a given frequency and a 50% duty cycle. Another green LED will blink constantly with a frequency of 1 Hz and a 50% duty cycle. The user will be able prompt for a reading of a single digital input channel and three analog input channels. The embedded systems application will be programmed in C on the PIC24FJ256GA705, and the GUI will be written in C# in Visual Studio. The embedded systems application will perform most of the operations, while the GUI takes user input and displays information to the user. The GUI and the application will connect through UART which will allow for the passing of information between the two.

# System Requirements, design specifications

Below are the system requirements for this project, organized into software, hardware, and operations requirements.

## Software

The embedded systems application is programmed in C using MPLab. The code is organized into several files: main.c, uart.h/.c, gpio.h/.c, adc.h/.c, taskLED.h/.c, taskUART.h/.c, and global\_variables.h. The GUI is programmed in C# using Visual Studio primarily in FormMain.cs.

## Hardware

The application is downloaded to a PIC24FJ256GA705 chip. The PIC chip is set up on a breadboard and connects to four diode LEDs, two push buttons, and a voltage divider circuit for analog inputs. The GUI is run on a Surface Pro 3. The PIC chip connects to the Surface Pro 3 through the LC231X UART to micro-usb converter.

## Operations

Both the GUI and the application are in ready state. The GUI updates the embedded systems application when the Send button is pushed by the user. This prompts a protocol to be sent to the PIC chip which decodes the protocol and responds accordingly. If the GUI receives a protocol, it processes and displays the information appropriately. The application reports the status of the yellow LED, the status of the digital input, and the values of the analog channels when it is notified by the PIC chip to do so. The PIC chip can, on command, turn on/off the green LED, turn on the yellow LED for a period of time before turning it off, disable (turn off) the yellow LED manually, enable/disable the red LED to blink on/off at a given frequency specified on the GUI at a 50% duty cycle, read the digital input from one of the push buttons, read three analog channels (typically connected to the voltage divider circuit), and reset when the other push button is pushed. The PIC chip continuously blinks a second LED on/off at a frequency of 1 Hz and with a 50% duty cycle.

# Design and Implementation

## System Blocks

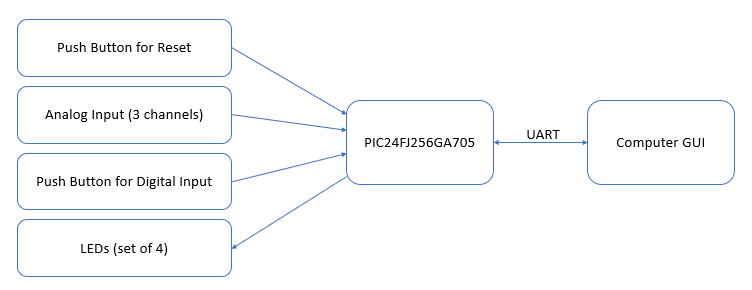


Figure : System Block Diagram

The main components of the systems include the PIC24FJ256GA705 chip, four LEDs, two push buttons, three input analog channels, and a personal computer. The LEDs are connected as output on the PIC chip to perform as described in the operations section above. The push buttons are inputs for reset and digital input. The analog channels are also configured as inputs. The PIC chip connects to the computer GUI through UART, with both transmit and receive pins.

## Schematics

The main system components, except the computer, are assembled on a breadboard along with other components such as resistors and capacitors. The schematic is shown below and includes pinouts to the LC231X for UART communication to the GUI and the RJ-12 for MPLab programming.

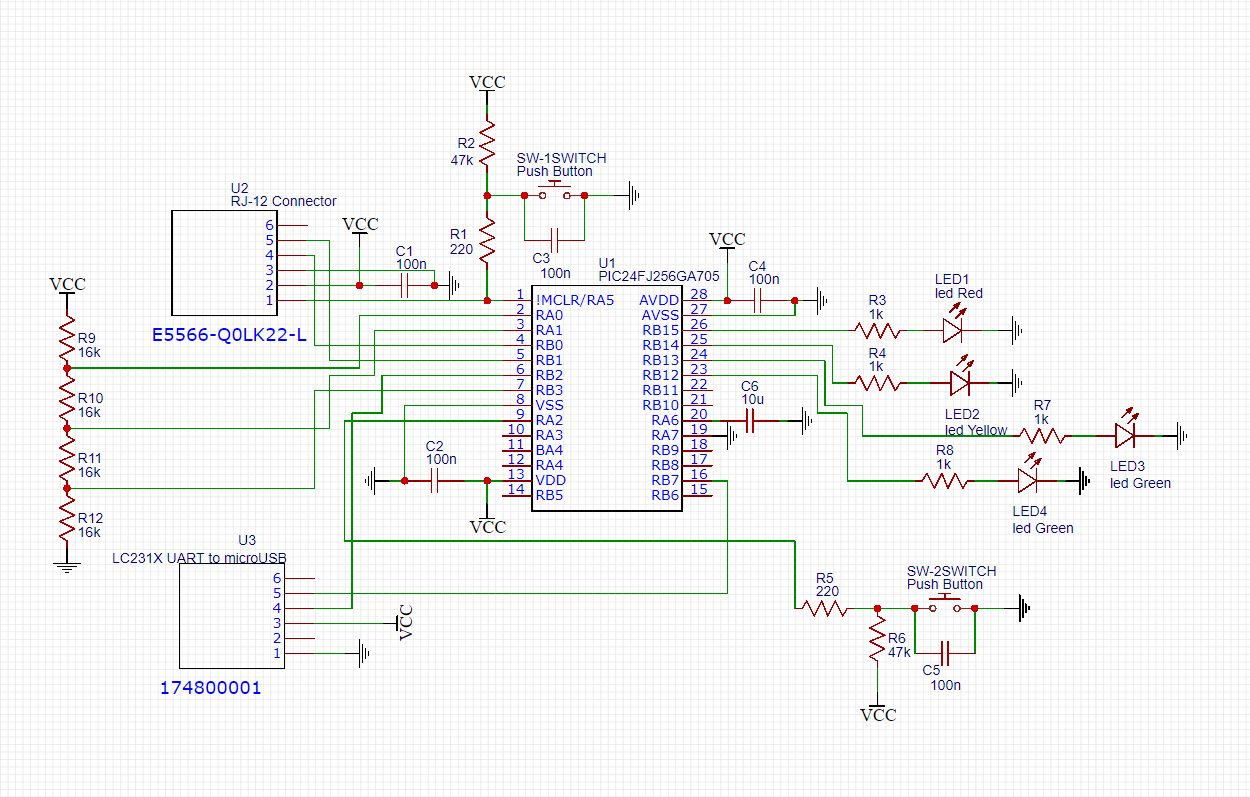


Figure : Breadboard Schematic

## Software Components

The general flow of the embedded systems application is to initialize the GPIO, the ADC, UART communication, and the various variables used by the application and then to initialize with freeRTOS three LED coroutines (heartbeat, timed output, and periodic output) and a task for UART. In the UART task, there is an endless loop where the application waits until a new begin protocol character is received through UART from the GUI. When the new protocol character is received, it transmits back an acknowledgement character and then waits for the rest of the protocol to be sent. Each character received before the end of protocol character is saved to a protocol buffer. The first character in the buffer is the mode, and the rest are used to set variables’ values. The different responses the application can have in response depending on the mode are to turn on/off a green LED, to enable/disable timed output on a yellow LED, to enable/disable a periodic pulse on a red LED, to read the status of the timed output LED, to read the status of a digital input, and to read the values of three analog input channels. When checking a status or reading values, a return message is transmitted to the GUI so that it be updated appropriately.

The GUI is set up to wait for an interrupt to trigger once it loads up. Interrupts trigger when buttons are pressed or when data is received once the COM port is opened for UART communication. The user can select from a drop-down menu to connect or disconnect the COM port. When any button, except for “Send” is pressed, the protocol textbox is updated with the coded message. When “Send” is pressed, the begin protocol character is sent, and then, after an acknowledgement is received from the PIC24, the protocol message followed by the end of protocol character are transmitted. When the COM port is open and data is received through UART, the message is read byte by byte from the begin protocol character to the end protocol character. The first character after the begin of protocol character is received, is saved as the mode. The rest of the characters received before the end of protocol character is received are saved to a string. Depending on the mode, the appropriate textbox is updated with the string information. If the acknowledge character is received instead of the begin of protocol character, a flag variable is raised.

The flowcharts for the design described above can be found in appendix A.

## Application Configuration and Workflow

The UART is configured with output on pin 16 and input on pin 6 of the PIC chip. U1BRG is set to 103 which corresponds to 4MHz frequency and 9600 baud rate with BRGH set to 1. Priority is set to 1. Receive and transmit are enabled. When the receive interrupt is triggered, the receive register is read and saved to a global character variable, and a global variable flag is raised to show that a new character was received. If the character is ‘<’, a global new protocol flag is raised. The interrupt flag is cleared. The uart handling is implemented in uart.h and timer.c. These files have three functions: uart\_init(), U1RXinterrupt(), and uart\_send() for initialization, interrupt handling, and transmitting a character. There are four global variables instantiated here which hold the last received character, a flag signifying if a new character has been received, a flag signifying is a new protocol is being sent, and the next character to be transmitted when send\_char() is called next.

The GPIO is configured with RP12, 13, 14, and 15 as output and the rest as input. RP12, 13, 14, and 15 and RA2 are set as digital. There are six functions: gpio\_init(), turn\_on(int LED) which turns an LED, turn\_off(int LED) which turns off an LED, toggle(int LED) which toggles an LED, status(int LED) which checks if an LED is on or off, and check\_digital() which checks if RA2 is on or off.

For ADC initialization, AD1CON1 is 0, AD1CSSL is 0, AD1CON2 is 0, and AD1CON3 is 0x2, and then ADON is set to 1. There are two functions: adc\_init(), and check\_analog(int channel) which returns the value of a given analog channel. ADC is set up to sample manually. When check\_analog(int channel) is called, AD1CHS is set equal to the channel parameter. SAMP is set to 1 to begin sampling. After a delay, SAMP is set to 0 to stop sampling. It waits while DONE is 0 (i.e. the ADC is converting the samples). Finally, ADC1BUF0 is returned as the value of that ADC reading. No manipulation of the raw data is done.

Main.c initializes GPIO, ADC, and UART, and then starts a UART task in FreeRTOS along with three LED coroutines.

The UART task is set with the highest priority. It turns off for a millisecond before checking if a new protocol is being received. If a new protocol is being received, it receives the entire protocol into a protocol buffer and then processes it. Depending on the mode character in the protocol, the PIC24 will respond accordingly. If the mode is ‘a’, the green LED is turned on. If the mode is ‘b’, the green LED is turned off. If the mode is ‘c’, the timed output is enabled, the timed output counter is set to 0, and the rest of the characters in the buffer are converted to their corresponding integer value and saved as the timed output duration. If the mode is ‘d’, the yellow LED is turned off, and the timed output is disabled. If the mode is ‘e’, if the yellow LED is on, “<e1>” is transmitted via UART, but, if the yellow LED is off, “<e0>” is transmitted. If the mode is ‘f’, the periodic output is enabled and the rest of the characters in the buffer are converted to their corresponding integer value and saved as the periodic pulse width. If the mode is ‘g’, the periodic output is disabled, and the red LED is turned off. If the mode is ‘h’, if the digital input is on, “<h1>” is transmitted via UART, but, if the yellow LED is off, “<h0>” is transmitted. If the mode is ‘i’, channel 1 on pin 2 (AN0) is checked and the value is transmitted via UART (e.g. <i0123>). If the mode is ‘j’, channel 2 on pin 3 (AN1) is checked and the value is transmitted via UART (e.g. <j1023>). If the mode is ‘k’, channel 3 on pin 7 (AN5) is checked and the value is transmitted via UART (e.g. <k0900>).

Three co-routines are setup for LEDs: timed output, periodic output, and heartbeat. Timed output has priority 1 and index 0. Periodic output has priority 2 and index 1. Heartbeat has priority 3 and index 2. The heartbeat co-routine delays for 0.5 seconds and then toggles a green LED. The periodic output co-routine delays for 0.1 seconds if disabled or 0.5 / pulse width seconds if enabled. When enabled it toggles the red LED after the delay. The timed output co-routine delays 1 second. If it is enabled, it checks the value of the timed output counter. If the counter is 0, it turns on the yellow LED. If the counter is greater than or equal to the timed output duration, it turns off the yellow LED and disables timed output. Before delaying again, it increments the timed output counter.

Global.h contains global variables referenced and set in the different co-routines and UART task. These include periodic pulse width, periodic status (enabled/disabled), timed duration, timed counter, and timed status.

## Analog to Digital Conversion

No transformation is done. The raw data is sent back to the GUI as is.

## C# Application

There are many buttons and textboxes in the GUI which are described in the following. There is a drop down option “Com Port” that allows the user to “Connect” the COM port for UART communication . When “LED on” is pushed, ‘a’ appears in the ASCII textbox. When “LED off” is pushed ‘b’ appears in the ASCII textbox. The seconds(s) and pulse(s)/second textboxes can be edited. When “Enabled” under timed output is pushed, ‘c’ followed by the value in the second(s) textbox appears in the ASCII textbox. If the value in the second(s) textbox is not an integer, nothing happens when “Enabled” is pushed. When “Disabled” under timed output is pressed, ‘d’ appears in the ASCII textbox. When “Status” is pressed, ‘e’ appears in the ASCII textbox. When “Enabled” under periodic is pushed, ‘f’ followed by the value in the pulse(s) /second textbox appears in the ASCII textbox. If the value in the pulse(s) / second textbox is not an integer, nothing happens when “Enabled” is pushed. When “Disabled” under periodic is pressed, ‘g’ appears in the ASCII textbox. When “Read” is pushed, ‘h’ appears in the ASCII textbox. When “CH1”, “CH2”, or “CH3” is pushed, ‘i’, ‘j’, or ‘k’ respectively appears in the ASCII textbox. The Hex textbox always reflect the ASCII textbox as hex values. When “Send” is pushed, ‘<’ is transmitted via UART. While the acknowledgement flag is not raised, it waits. Then the values in the ASCII textbox are transmitted followed by ‘>’.

For the UART received interrupt, if ‘<’ is received, the next character is saved as the mode. The remaining characters received before ‘>’ is received are saved to rx\_string. If the mode is ‘e’ and rx\_string is ‘1’, “ON” is displayed in the status textbox under timed output. If the mode is ‘e’ and rx\_string is ‘0’, “OFF” is displayed in the status textbox under timed output. If the mode is ‘h’ and rx\_string is ‘1’, “ON” is displayed in the status textbox under digital input. If the mode is ‘h’ and rx\_string is ‘0’, “OFF” is displayed in the status textbox under digital input. If the mode is ‘i’, rx\_string is displayed in the CH1 textbox. If the mode is ‘k’, rx\_string is displayed in the CH2 textbox. If the mode is ‘k’, rx\_string is displayed in the CH3 textbox. If ‘!’ is received instead of ‘<’, the acknowledgement flag is raised. If any other character is received instead of ‘<’, nothing happens.

# Results

Include screen capture of your GUI at runtime, your system’s expected and actual limitations (such as maximum input signal frequency). The sampling frequencies you can get, the DFT resolution, the minimum and maximum frequency you can display in time plot and DFT. Etc.

The system runs as expected. The periodic output can support a range of frequencies from 1 to 100 Hz. The analog input channel values range from 0 to 1023 corresponding to ground and supply voltage on the PIC24. The timed output is tested and is seen to work for a range of times from 1 s to 20 s. The test case procedure is used for check-off. All predefined tests are verified as fully functional. The student defined tests for enabling the periodic pulse for 4 pulses per second and then for disabling the periodic pulse (tests 3.1 and 3.2) are verified as functional. The other student defined tests for pushing CH1 and CH2 (tests 4.3 and 4.4) are verified as functional. The instructor defined tests for restart of C# application not crashing the system (R1.1) and for testing digital input on/off (R1.3) are verified as functional. The instructor defined test to remove the Rx pin to the GUI and not crash the GUI (R1.2) was not passed, because the GUI froze.

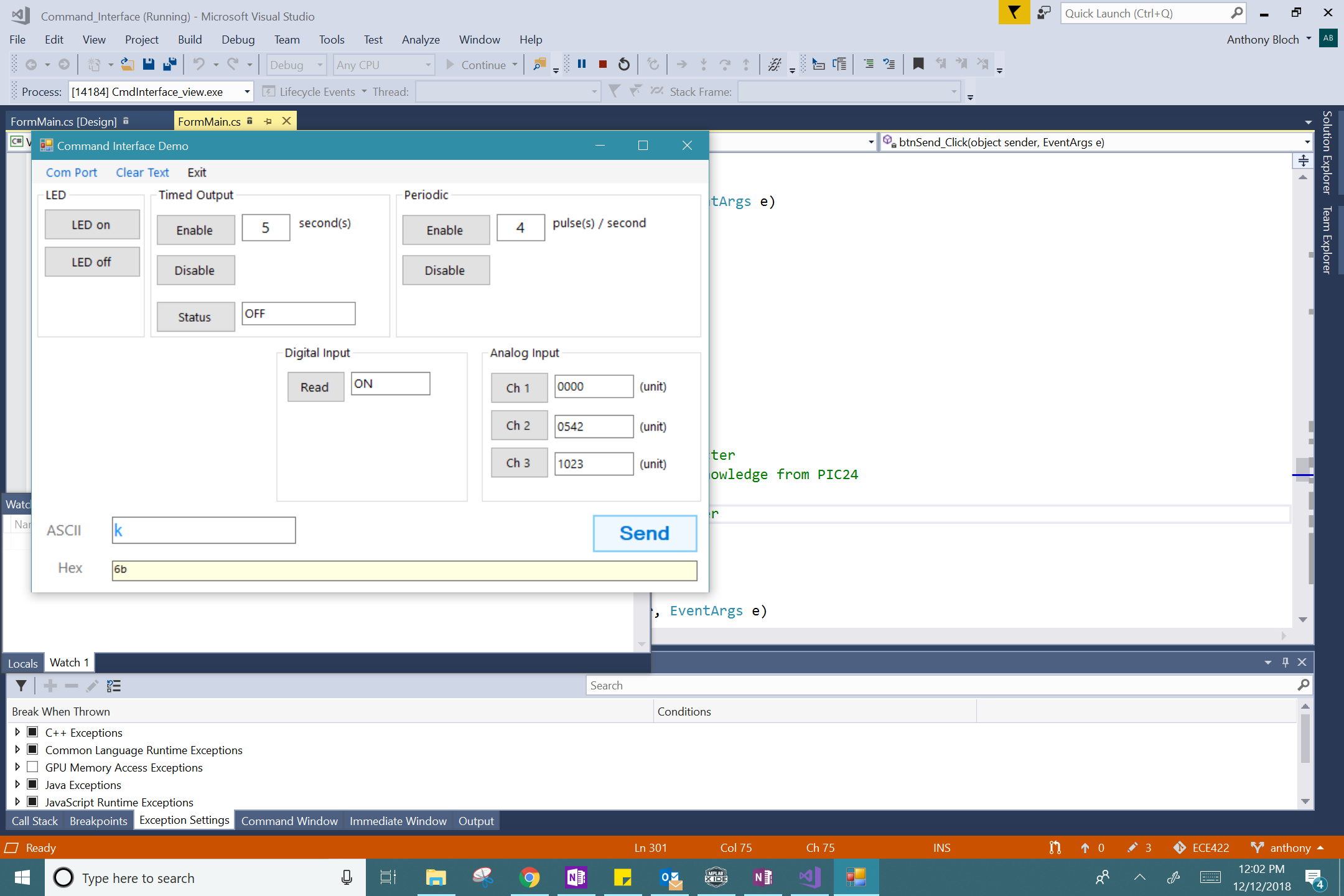


Figure : GUI Running

# Development Journal

Include your development logs here.

|  |  |
| --- | --- |
| Date | Description |
| 11/25/18 | Start time: 3:45 PM   * Read through lab handout * Design for protocol   + Operation commands to microcontroller:     - Change pulse width of waveform -> 'p' followed by integer for width       * P->5 for pulse width seconds     - Toggle waveform on/off -> 'w'     - Change timed output duration -> 'd' followed by integer for duration of timed output     - Toggle timed output on/off -> 't'     - Read input -> 'i' followed by d or a , 1, 2, or 3 channel   + Info sent back from microcontroller     - Waveform status -> 'w' followed by '1' or '0' for ON or OFF followed by pulse width (integer)     - Timed output status -> 't' followed by '1' or '0' for ON or OFF followed by time value (integer)     - Input value -> 'i' followed by value of input (float sent digit by digit)   + Note: all commands are ended with a '!' as an end of string character   End time: 4:05 PM |
| 11/26/18 | Start time: 5:25 PM   * Started microcontroller code in mplab ->Lab10.x   + Receives uart and sends back character shifted one * Started looking at gui template * Discussed protocol design with professor   End time: 7:10 PM |
| 11/28/18 | Start time: 9:00 AM   * Added LED toggle with each uart reception * Looked at gui more to better understand how commands are send   End time: 9:30 AM  Start time: 12:45 PM   * Reworked protocol referencing existing GUI   + LED     - LED on -> a -> !       * Ex: a!     - LED off -> b -> !       * Ex: b!   + Timed Output     - Enable -> c -> duration (decimal digits left to right) -> !       * Ex: c12! (12 seconds duration)     - Disable -> d -> !       * Ex: d!     - Status -> e -> !       * Ex: e!   + Periodic     - Enable -> f -> duration (decimal digits left to right) -> !       * Ex: f24! (24 pulses/second)     - Disable -> g -> !       * Ex: g!   + Digital Input     - Read -> h -> !       * Ex: h!   + Analog Input     - Ch 1 -> i -> !       * Ex: i!     - Ch 2 -> j -> !       * Ex: j!     - Ch 3 -> k -> !       * Ex: k! * Pushed version 1.0 to git   + GUI set up to send proper protocols to PIC24   + PIC24 set up to receive protocol     - Can process protocols for turning on/off a single LED (green)   + PIC project     - Files Added       * Main.c       * App.h/.c       * Gpio.h/.c       * Uart.h.c   + GUI     - Files added       * Template from D2L   End time: 2 PM  Begin time: 7:35 PM   * Implemented pulse led and timed output led * Changed protocol to begin with '<' and end with '>' * Saved version 1.1   + PIC project     - Files Added       * Timer.h/.c     - Added LED pulsed and LED timed output functionality     - Added status reporting of LED timed output   + Protocols now begin with '<' and end with '>' * TODO: digital and analog readings   End time: 12:30 AM |
| 11/29/18 | Begin time: 6:50 PM   * Pushed version 1.2   + PIC project     - Added functionality to read digital input (button) - verified as working     - Added functionality to read analog, but it needs more work   + Full communication between GUI and PIC is functional   End time: 9 PM |
| 11/30/18 | Begin time: 11:30 AM   * Incorporated FreeRTOS into project * Pushed Version 1.3 to git   + PIC project     - Added files for FreeRTOS tasks       * taskUART.h/.c       * taskLEDs.h/.c     - UART doesn't always receive command (perhaps because of delay). May need receive buffer instead of character for uart   End time: 3:30 PM  Begin time: 6:30 PM   * Created coroutines for timed and periodic output * Uart is now working   + Set up synchronization sequence     - GUI sends '<' and then waits for '!' before sending the rest of the protocol     - PIC waits till '<' is received (flag raised in uart.c) and then sends '!' before receiving/processing the rest of the protocol     - TODO: timed output -> currently both timed and periodic LEDs blink simultaneously (like periodic)   End time: 8:12 PM |
| 12/6/18 | Begin time: 10:37 PM   * Tried to set up ADC, but configuration doesn't seem to be right…gets stuck when sampling   End time: 11:37 PM |
| 12/7/18 | Begin time: 9 AM   * Set up single ADC on AN5   End time: 10 AM  Begin time: 11 AM   * Completed ADC * Added heartbeat constant running task * Fine tuned timing of LEDs * Pushed version 2.0   + PIC24     - Added Adc.h/.c     - Added 3rd coroutine   End time 1:10 PM |
| 12/10/18 | Begin time: 12 PM   * Final calibration, dry run of check off   End time: 2:43 PM |
| 12/12/18 | Begin time: 11:20 AM   * Commented code * Pushed version 2.1 with commented code * Disassembled hardware and turned it in   End time: 12:40 |
| 12/14/18 | Begin time: 2:00 PM   * Worked on report   End time: 3:00 PM |
| 12/18/18 |  |

# Analysis and Discussion

As all but one of the tests are successful, the system can be considered to meet the design specifications well. For the test that failed where the GUI crashed, this could be fixed by removing the while loop in the send function that waits for the acknowledgement flag to be raised. Instead, of waiting for the flag to be raised in the send function, the protocol could be saved to a string variable. When processing the data received by UART and the acknowledgement character is received, the previously saved protocol string could then be transmitted. This should prevent the GUI from freezing if the Rx pin is disconnected. The ADC has a 10-bit resolution (1024 values) which is about 3 mV since Vcc is 3.3 volts. The results of the ADC readings reflected well to the voltage they represented. The value read compared to 1023 (max value), was the same proportion as the actual voltage (read by voltmeter) compared to 3.3 V.

One challenge was synchronizing the UART task with the GUI. For the other co-routines to run, the UART task required a delay for it to sleep. At first this allowed the PIC24 to miss some messages sent by the GUI because the acknowledgement wasn’t set up that way originally. At first the PIC24 just checked if a new character was received and if it was ‘<’ (the begin of protocol character). If this was sent while the UART task was sleeping, it would not get processed. As a solution, a new protocol flag was added with the acknowledgement from the PIC24 to the GUI. While this led to the GUI freezing when the Rx pin is disconnected, a solution to this issue is described above.

The major limitation seen in the system is the maximum frequency possible for the periodic pulse which is 100 Hz. This is due to the way tasks and co-routines are set up. The UART task is set with the highest priority, and all tasks and co-routines are set to sleep for at least a small period of time. The 100 Hz limit is the result of having to process the other tasks and co-routines. Taking turns switching between each of them takes a minimum amount of time resulting in the maximum frequency of 100 Hz. One improvement that could be made is to use semaphores to control the tasks and co-routines.

# Conclusion

Using Visual Studio and MPLab, a GUI in C# and an embedded systems application in C for the PIC24FJ256GA705 to form a digital acquisition board and app to control four LEDs, to read a digital input, and to read three analog inputs. The system is tested and found to meet the design specifications mostly well. Experience is gained in working with ADC, UART, and GPIO on the PIC24.

# Acknowledgement

I would like to thank Raymond Lei for teaching ECE422 wherein I learned or further developed the concepts and practices necessary to complete this project. I would also like to thank Josh Benoit for his help throughout the semester when I was having trouble on the labs that paved the way for this final project.

# Appendix A

Section 1: Flowcharts

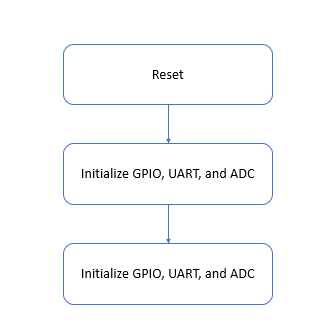


Figure 4: Flowchart for Main.c

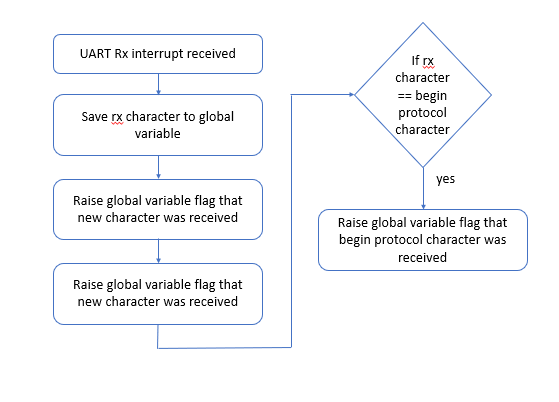


Figure 5: Flowchart for UART interrupt on PIC24

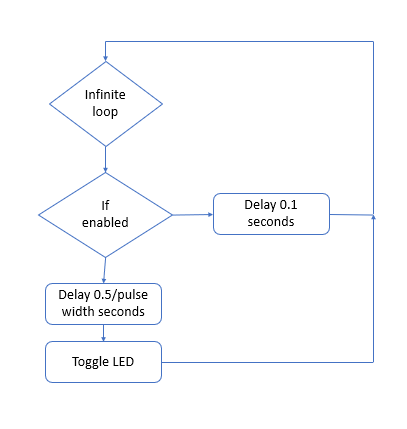


Figure 6: Flowchart for Periodic Co-Routine

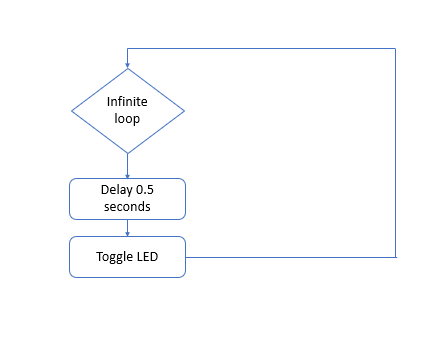


Figure 7: Flowchart for Heartbeat Co-Routine

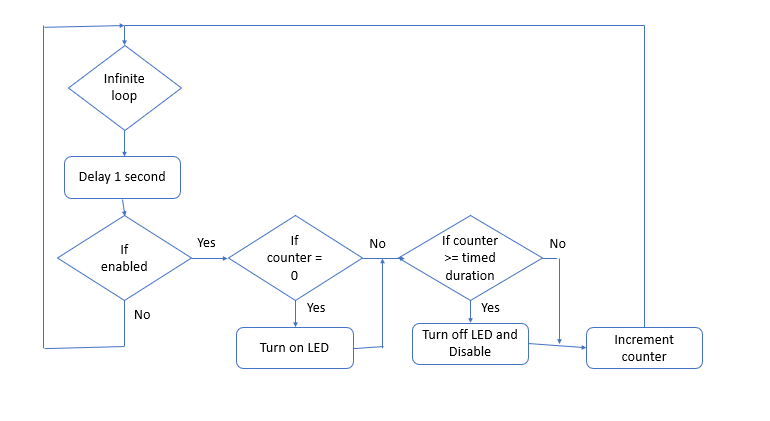


Figure 8: Flowchart for Timed Output Co-Routine

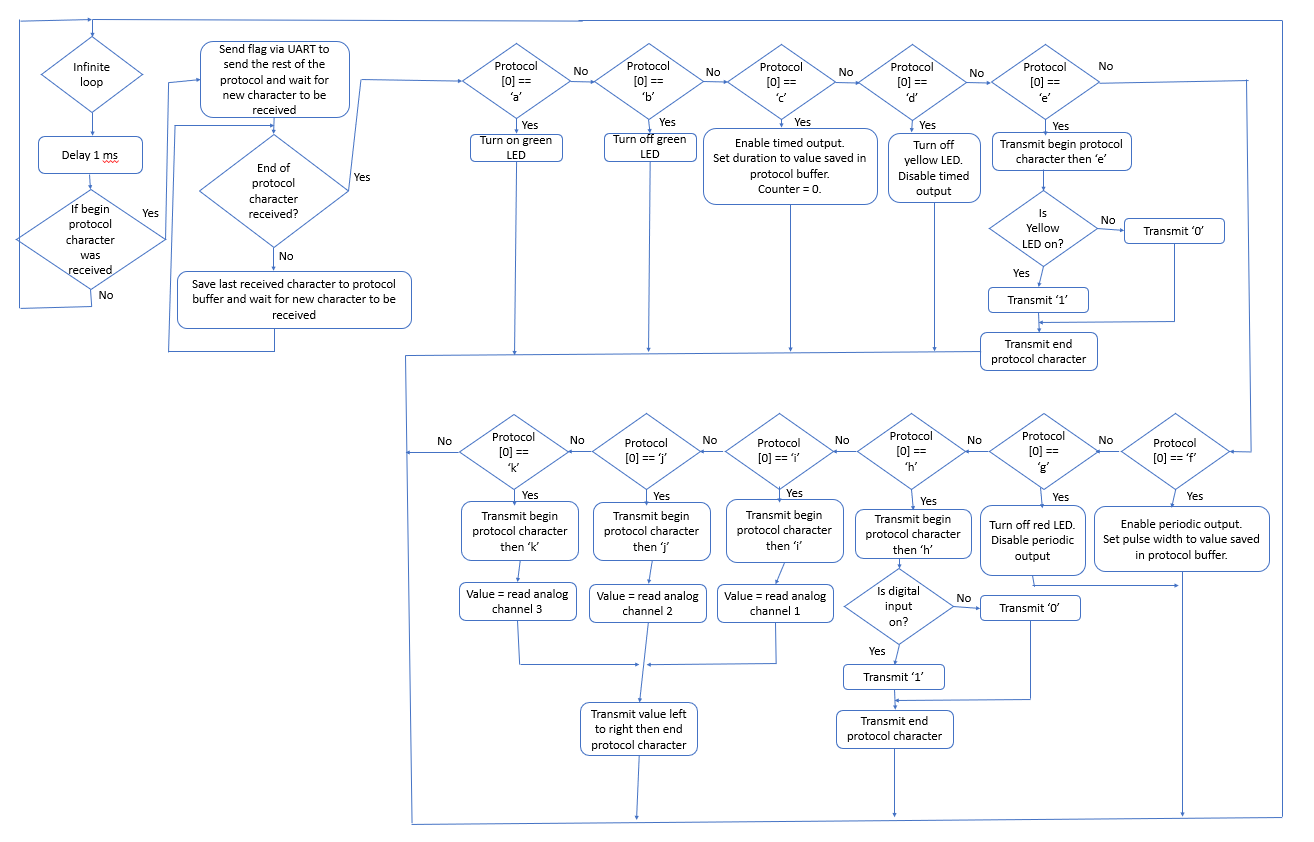


Figure 9: Flowchart for UART task

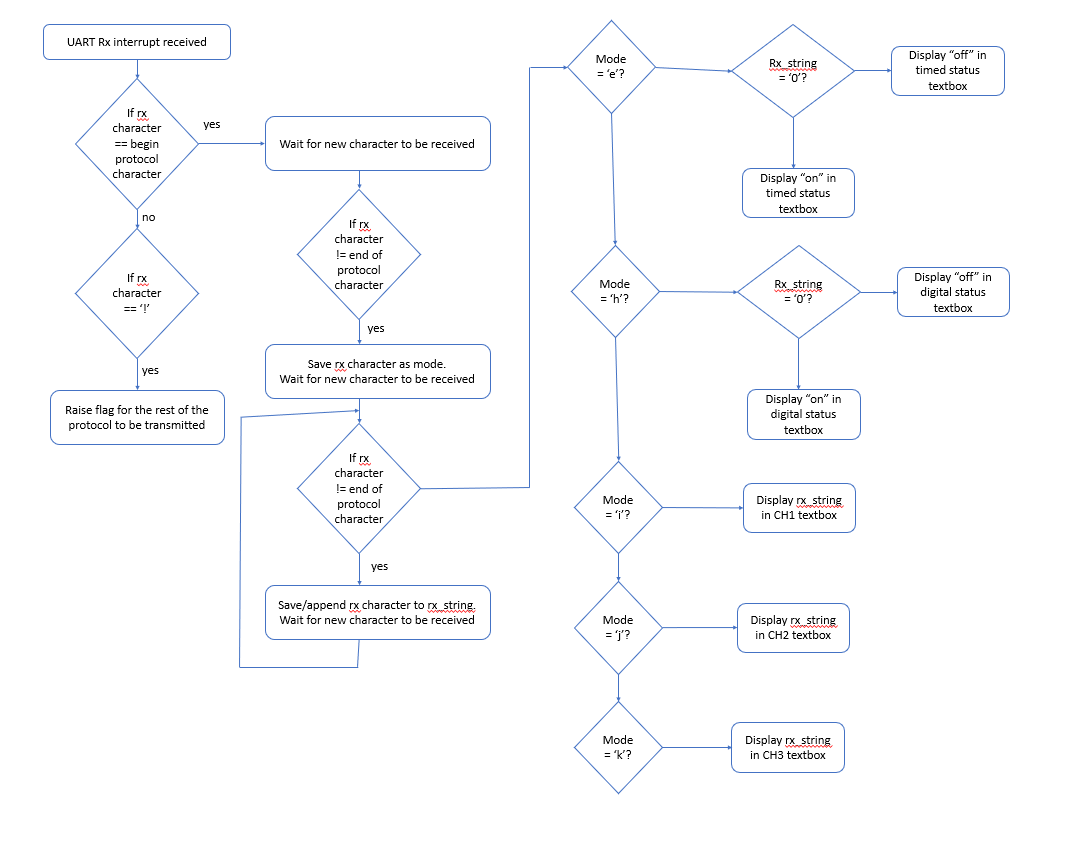


Figure 10: Flowchart for UART Interrupt on GUI

Section 2: Code

**FormMain.cs**

/\*

\* File: FormMain.cs

\* Author: Anthony Bloch

\* Comments: This is the main for Lab10 for ECE422

\* This program runs alongside a program running on the PIC24FJ256GA2017 connected via uart to this gui.

\* Protocols are sent from the GUI which are processed and responded to accordingly by the PIC24.

\* Functionality includes:

\* constantly blink LED1 with period of 1 second

\* prompted by protocol...

\* turn on/off LED2

\* turn on LED3 for a period of time after which it turns off

\* this LED can be disabled while it is on, and its status can also be read

\* enable LED4 such that is blinks on/off at a given frequency

\* this LED can be disabled such that it remains off

\* read a digital input button

\* read 3 analog channel inputs

\* reset capability through push button

\* information is sent back to the GUI via uart as needed

\*

\* Revision history:

\*/

using System;

using System.IO.Ports;

using System.Windows.Forms;

using CmdInterface.common;

namespace CmdInterface

{

public partial class FormMain : Form, IView\_FormMain

{

private FormPortSettings \_formPortSettings;

private PortSettingsEntity \_defaultSettings, \_currentSettings;

private CmdComposer \_composer;

private int new\_ch = 0;

private char ch\_rx;

private char mode\_rx;

private string str\_rx;

private int count\_rx;

private char ch\_tx;

private int ready\_to\_send\_protocol;

public FormMain()

{

InitializeComponent();

//Initialize UI visual elements

connectToolStripMenuItem.Enabled = true;

disconnectToolStripMenuItem.Enabled = false;

//tbAscii.ReadOnly = true;

}

private void FormMain\_Load(object sender, EventArgs e)

{

\_defaultSettings = new PortSettingsEntity(); //Object instance containing default settings

\_currentSettings = new PortSettingsEntity(); //Object instance storing current settings from user

InitializeComPort(\_defaultSettings);

InitializeComPort(\_currentSettings);

\_composer = new CmdComposer(this);

}

private void FormMain\_FormClosing(object sender, FormClosingEventArgs e)

{

if (serialPort1.IsOpen)

{

serialPort1.Close();

}

}

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void InitializeComPort(PortSettingsEntity handle)

{

string[] port\_list = System.IO.Ports.SerialPort.GetPortNames();

if (port\_list.Length > 0)

{

handle.PortName = port\_list[0]; //Default com port is the first one on the list

handle.BaudRate = 9600;

handle.DataBits = 8;

handle.StopBits = System.IO.Ports.StopBits.One;

handle.Parity = System.IO.Ports.Parity.None;

}

}

private void Connect(PortSettingsEntity handle)

{

if (handle != null)

{

//Configure serial port

serialPort1.PortName = handle.PortName;

serialPort1.BaudRate = (int)handle.BaudRate;

serialPort1.DataBits = handle.DataBits;

serialPort1.Parity = handle.Parity;

serialPort1.Handshake = handle.Handshake;

serialPort1.ReadTimeout = handle.ReadTimeout;

serialPort1.WriteTimeout = handle.WriteTimeout;

serialPort1.DiscardNull = handle.DiscardNull;

serialPort1.RtsEnable = handle.RtsEnable;

serialPort1.NewLine = handle.Newline;

serialPort1.Encoding = handle.Encoding;

try

{

if (serialPort1.IsOpen == false)

{

serialPort1.Open(); //attempt to open the configured serial port

if (serialPort1.IsOpen)

{

//Com Port connected

connectToolStripMenuItem.Enabled = false;

disconnectToolStripMenuItem.Enabled = true;

tbAscii.ReadOnly = false;

}

}

}

catch (Exception)

{

MessageBox.Show("Unable to open serial port");

}

}

}

private void serialPort1\_DataReceived(object sender, SerialDataReceivedEventArgs e)

{

//Note: SerialPort object operates on a seperate thread.

// Therefore, DataReceived event can not interact directly with other WinForm controls.

// Doing so will cause cross-thread action exception.

// In order to display received data to UI, delegate method must be used.

//Read data from serial port object

int temp;

temp = serialPort1.ReadByte();

ch\_rx = (char)temp;

if(ch\_rx == '<') // begin protocol character

{

count\_rx = 0;

str\_rx = "";

temp = serialPort1.ReadByte();

ch\_rx = (char)temp;

while (ch\_rx != '>') // keep receiving protocol until end protocol character is received

{

if(count\_rx == 0)

{

mode\_rx = ch\_rx;

}

else

{

str\_rx += ch\_rx.ToString();

}

count\_rx++;

temp = serialPort1.ReadByte();

ch\_rx = (char)temp;

}

}

else if(ch\_rx == '!') // response from PIC24 that it is ready for protocol

{

ready\_to\_send\_protocol = 1;

}

else

{

;

}

this.Invoke(new EventHandler(ShowText));

}

private void ShowText(object sender, EventArgs e) // called after UART data received

{

if (count\_rx > 0)

{

switch (mode\_rx)

{

case 'e': // status of timed output

{

if (count\_rx > 1)

{

if (str\_rx == "1")

{

tbStatus.Text = "ON";

}

else

{

tbStatus.Text = "OFF";

}

}

break;

}

case 'h': // digital read

{

if (count\_rx > 1)

{

if (str\_rx == "1")

{

tbInputStatus.Text = "ON";

}

else

{

tbInputStatus.Text = "OFF";

}

}

break;

}

case 'i': //read analog channel 1

{

if (count\_rx > 1)

{

tbStatusAN1.Text = str\_rx;

}

break;

}

case 'j': // read analog channel 2

{

if (count\_rx > 1)

{

tbStatusAN2.Text = str\_rx;

}

break;

}

case 'k': // read analog channel 3

{

if (count\_rx > 1)

{

tbStatusAN3.Text = str\_rx;

}

break;

}

default:

{

break;

}

}

}

}

private void connectToolStripMenuItem\_Click(object sender, EventArgs e)

{

//handle serial port connect

Connect(\_currentSettings);

}

private void disconnectToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (serialPort1.IsOpen)

{

//Disconnect serial port

serialPort1.Close();

connectToolStripMenuItem.Enabled = true;

disconnectToolStripMenuItem.Enabled = false;

tbAscii.ReadOnly = true;

}

}

private void tbAscii\_TextChanged(object sender, EventArgs e)

{

byte[] buffer = common.Utility.StringToByteArray(tbAscii.Text);

UpdateHexTextbox((buffer));

}

public void ShowCommand(string command\_string)

{

tbAscii.Text = command\_string;

}

private void UpdateHexTextbox(byte[] buffer)

{

tbHex.Text = common.Utility.ByteArrayToString(buffer);

tbHex.SelectionStart = tbHex.Text.Length;

tbHex.ScrollToCaret();

tbHex.Refresh();

}

private void clearTextToolStripMenuItem\_Click(object sender, EventArgs e)

{

tbAscii.Clear();

}

private void btnSend\_Click(object sender, EventArgs e)

{

if (serialPort1.IsOpen)

{

if (string.IsNullOrEmpty(tbAscii.Text) == false)

{

//Write string directly to serial port object

ready\_to\_send\_protocol = 0;

serialPort1.Write("<"); // send begin protocol character

while (ready\_to\_send\_protocol == 0); // wait for acknowledge from PIC24

serialPort1.Write(tbAscii.Text);

serialPort1.Write(">"); // send end protocol character

}

}

}

private void settingsToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (\_formPortSettings == null)

{

\_formPortSettings = new FormPortSettings();

}

\_formPortSettings.Initialize(\_defaultSettings, \_currentSettings);

if (\_formPortSettings.ShowDialog() == DialogResult.OK)

{

//apply settings

\_currentSettings = \_formPortSettings.NewSettings;

}

}

private void btnLedOff\_Click(object sender, EventArgs e)

{

\_composer.GenericCmd("b");

}

private void btnLedOn\_Click(object sender, EventArgs e)

{

\_composer.GenericCmd("a");

}

private void btnEnableTimed\_Click(object sender, EventArgs e)

{

int parsedValue;

if (int.TryParse(textBoxTime1.Text, out parsedValue))

{

\_composer.GenericCmd("c" + textBoxTime1.Text);

}

else

{

\_composer.GenericCmd("");

}

}

private void btnDisableTimed\_Click(object sender, EventArgs e)

{

\_composer.GenericCmd("d");

}

private void btnEnablePeriodic\_Click(object sender, EventArgs e)

{

int parsedValue;

if (int.TryParse(tbPeriodic.Text, out parsedValue))

{

\_composer.GenericCmd("f" + tbPeriodic.Text);

}

else

{

\_composer.GenericCmd("");

}

}

private void btnDisabledPeriodic\_Click(object sender, EventArgs e)

{

\_composer.GenericCmd("g");

}

private void btnReadInput\_Click(object sender, EventArgs e)

{

\_composer.GenericCmd("h");

}

private void btnCh1\_Click(object sender, EventArgs e)

{

\_composer.GenericCmd("i");

}

private void btnCh2\_Click(object sender, EventArgs e)

{

\_composer.GenericCmd("j");

}

private void tbStatus\_TextChanged(object sender, EventArgs e)

{

}

private void tbPeriodic\_TextChanged(object sender, EventArgs e)

{

}

private void lblSecond\_Click(object sender, EventArgs e)

{

}

private void btnStatus\_Click(object sender, EventArgs e)

{

\_composer.GenericCmd("e");

}

private void tbInputStatus\_TextChanged(object sender, EventArgs e)

{

}

private void tbStatusAN1\_TextChanged(object sender, EventArgs e)

{

}

private void btnCh3\_Click(object sender, EventArgs e)

{

\_composer.GenericCmd("k");

}

}

}

**adc.h**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: adc.h

\* Author: Anthony Bloch

\* Comments: handles ADC for PIC24

\* Revision history:

\*/

// This is a guard condition so that contents of this file are not included

// more than once.

#ifndef ADC\_H

#define ADC\_H

#include <xc.h> // include processor files - each processor file is guarded.

void adc\_init(void);

int check\_analog(int channel);

#endif /\* ADC\_H \*/

**adc.c**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: adc.c

\* Author: Anthony Bloch

\* Comments: handles ADC for PIC24

\* Revision history:

\*/

#include "adc.h"

void delay(int delayVal)

{

unsigned int i;

for (i = 0; i < delayVal; i++)

{

;

}

}

void adc\_init(void)

{

//initialize adc for manual sampling

AD1CON1 = 0;

AD1CSSL = 0;

AD1CON2 = 0;

AD1CON3 = 0x2;

AD1CON1bits.ADON = 1; // turn on ADC

}

int check\_analog(int channel)

{

AD1CHS = channel; // set channel

AD1CON1bits.SAMP = 1; // start sampling

delay(100000);

AD1CON1bits.SAMP = 0; // stop sampling

while(AD1CON1bits.DONE == 0);

return ADC1BUF0;

}

**global\_variables.h**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: global\_variables.h

\* Author: Anthony Bloch

\* Comments: contains global variables used within the project and across various tasks

\* Revision history:

\*/

// This is a guard condition so that contents of this file are not included

// more than once.

#ifndef GLOBAL\_VARIABLES\_H

#define GLOBAL\_VARIABLES\_H

#include <xc.h> // include processor files - each processor file is guarded.

extern int periodic\_pulse\_width;

extern int periodic\_status;

extern int periodic\_count;

extern int led\_red\_status;

extern int timed\_status;

extern int timed\_duration;

extern int timed\_count;

extern int led\_yellow\_status;

#endif /\* GLOBAL\_VARIABLES\_H \*/

**gpio.h**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: gpio.h

\* Author: Anthony Bloch

\* Comments: handles GPIO on PIC24

\* Revision history:

\*/

// This is a guard condition so that contents of this file are not included

// more than once.

#ifndef GPIO\_H

#define GPIO\_H

#include <xc.h> // include processor files - each processor file is guarded.

#define ON 1

#define OFF 0

#define LED\_GREEN 13

#define LED\_YELLOW 14

#define LED\_RED 15

#define LED\_POWER 12

#define ENABLED 1

#define DISABLED 0

void gpio\_init(void);

void turn\_on(int LED);

void turn\_off(int LED);

void toggle(int LED);

int status(int LED);

int check\_digital(void);

#endif /\* GPIO\_H \*/

**gpio.c**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: gpio.c

\* Author: Anthony Bloch

\* Comments: handles GPIO for PIC24

\* Revision history:

\*/

#include "gpio.h"

void gpio\_init(void)

{

LATB = 0x0000; // initialize output to be off

TRISB = 0x0FFF; // RP12, 13, 14, 15 are output, the rest are input

TRISA = 0xFFFF; // all are input

ANSB = 0x0FFF; //RP12, 13, 14, 15 are digital

ANSA = 0xFFFB; // RA2 is digital

}

//turn on RPx where x is LED

void turn\_on(int LED)

{

LATB |= 0x0001 << LED;

}

//turn off RPx where x is LED

void turn\_off(int LED)

{

LATB &= (~(0x0001 << LED));

}

//toggle RPx where x is LED

void toggle(int LED)

{

if(((LATB >> LED)&0x0001) == 0) // check if LED is off

{

turn\_on(LED);

}

else

{

turn\_off(LED);

}

}

//checks the status of RPx where x is LED

int status(int LED)

{

int status;

if(((LATB >> LED)&0x0001) == 0) // check if LED is off

{

status = 0;

}

else

{

status = 1;

}

return status;

}

//checks digital input at RA2

int check\_digital(void)

{

return PORTAbits.RA2;

}

**main.c**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: main.c

\* Author: Anthony Bloch

\* Comments: This is the main for Lab10 for ECE422

\* This program runs on the PIC24FJ256GA2017 along with a GUI connected via uart.

\* Protocols are sent from the GUI which are processed and responded to accordingly by the PIC24.

\* Functionality includes:

\* constantly blink LED1 with period of 1 second

\* prompted by protocol...

\* turn on/off LED2

\* turn on LED3 for a period of time after which it turns off

\* this LED can be disabled while it is on, and its status can also be read

\* enable LED4 such that is blinks on/off at a given frequency

\* this LED can be disabled such that it remains off

\* read a digital input button

\* read 3 analog channel inputs

\* reset capability through push button

\* information is sent back to the GUI via uart as needed

\*

\* Revision history:

\* version 1.0

§ GUI set up to send proper protocols to PIC24

§ PIC24 set up to receive protocol

□ Can process protocols for turning on/off a single LED (green)

§ PIC project

□ Files Added

® Main.c

® App.h/.c

® Gpio.h/.c

® Uart.h.c

§ GUI

□ Files added

Template from D2L

\*version 1.1

§ PIC project

□ Files Added

® Timer.h/.c

□ Added LED pulsed and LED timed output functionality

□ Added status reporting of LED timed output

Protocols now begin with '<' and end with '>'

\* version 1.2

§ PIC project

□ Added functionality to read digital input (button) - verified as working

□ Added functionality to read analog, but it needs more work

Full communication between GUI and PIC is functional

\* Version 1.3 to git

§ PIC project

□ Added files for FreeRTOS tasks

® taskUART.h/.c

® taskLEDs.h/.c

UART doesn't always receive command (perhaps because of delay). May need receive buffer instead of character for uart

\* version 2.0

§ PIC24

□ Added Adc.h/.c

Added 3rd coroutine

\* version 2.1

\* commented code

\*/

#include <stddef.h> // Defines NULL

#include <stdbool.h> // Defines true

#include <stdlib.h> // Defines EXIT\_FAILURE

#include "FreeRTOS.h"

#include "croutine.h"

#include "task.h"

#include "../mcc\_generated\_files/system.h"

#include "../mcc\_generated\_files/pin\_manager.h"

//RTOS tasks include

#include "app.h"

#include "taskUART.h"

//#include "taskTimed.h"

#include "taskLED.h"

#include "gpio.h"

#include "uart.h"

#include "adc.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Private Macros

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* The number of flash co-routines to create. \*/

#define mainNUM\_FLASH\_COROUTINES (3)

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Section: Main Entry Point

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main( void )

{

// initialize the device

SYSTEM\_Initialize();

gpio\_init();

uart\_init();

adc\_init();

/\*while(1)

{

temp2 = 0;

temp = check\_analog(temp2);

temp2 = 0;

temp2 = 1;

temp = check\_analog(temp2);

temp2 = 0;

temp2 = 5;

temp = check\_analog(temp2);

temp2 = 0;

}\*/

//=========================================================================

// Board initialization

//=========================================================================

//=========================================================================

// Set application's initial state

//=========================================================================

//=========================================================================

// Application Task initialization

//=========================================================================

taskUART\_init();

taskLED\_init(mainNUM\_FLASH\_COROUTINES);

//taskPeriodic\_init(mainNUM\_FLASH\_COROUTINES);

//=========================================================================

// FreeRTOS scheduler

//=========================================================================

vTaskStartScheduler();

/\* If all is well then this line will never be reached. If it is reached

then it is likely that there was insufficient (FreeRTOS) heap memory space

to create the idle task. This may have been trapped by the malloc() failed

hook function, if one is configured.

\*/

while (1)

{

// Add your application code

}

return -1;

}

**taskLED.h**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: taskLED.h

\* Author: Anthony Bloch

\* Comments: runs tasks for heartbeat, timed, and periodic LEDs

\* Revision history:

\*/

#ifndef TASK\_LED\_H

#define TASK\_LED\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

#define crf\_TIMED\_INDEX (0)

#define crf\_TIMED\_PRIORITY (1)

#define crf\_PERIODIC\_INDEX (1)

#define crf\_PERIODIC\_PRIORITY (2)

#define crf\_HEARTBEAT\_INDEX (2)

#define crf\_HEARTBEAT\_PRIORITY (3)

extern void taskLED\_init(unsigned portBASE\_TYPE uxPriority);

extern void taskTimed\_Execute(void);

extern void taskPeriodic\_Execute(void);

extern void taskHeartbeat\_Execute(void);

extern void prvTimedCoRoutine(CoRoutineHandle\_t xHandle, unsigned portBASE\_TYPE uxIndex);

extern void prvPeriodicCoRoutine(CoRoutineHandle\_t xHandle, unsigned portBASE\_TYPE uxIndex);

extern void prvHeartbeatCoRoutine(CoRoutineHandle\_t xHandle, unsigned portBASE\_TYPE uxIndex);

#ifdef \_\_cplusplus

}

#endif

#endif /\* TASK\_LED\_H \*/

**taskLED.c**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: taskLED.c

\* Author: Anthony Bloch

\* Comments: runs tasks for heartbeat, timed, and periodic LEDs

\* Revision history:

\*/

#include <stddef.h> // Defines NULL

#include <stdbool.h> // Defines true

#include <stdlib.h> // Defines EXIT\_FAILURE

#include "FreeRTOS.h"

#include "croutine.h"

#include "task.h"

#include "../mcc\_generated\_files/system.h"

#include "../mcc\_generated\_files/pin\_manager.h"

//RTOS tasks include

#include "app.h"

#include "taskUART.h"

#include "taskLED.h"

#include "gpio.h"

#include "global\_variables.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FreeRTOS Task implementation: Co-routine for periodic pulse output when enabled

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void prvPeriodicCoRoutine(CoRoutineHandle\_t xHandle, unsigned portBASE\_TYPE uxIndex)

{

/\* Co-routines MUST start with a call to crSTART. \*/

crSTART(xHandle);

for (;;)

{

if(periodic\_status == DISABLED)

{

crDELAY(xHandle, Time\_MillisecondsToTicks(100));

}

else

{

crDELAY(xHandle, Time\_MillisecondsToTicks(2500/periodic\_pulse\_width));

}

if(periodic\_status == ENABLED)

{

taskPeriodic\_Execute();

}

}

/\* Co-routines MUST end with a call to crEND. \*/

crEND();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FreeRTOS Task implementation: Co-routine for timed output when enabled

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void prvTimedCoRoutine(CoRoutineHandle\_t xHandle, unsigned portBASE\_TYPE uxIndex)

{

/\* Co-routines MUST start with a call to crSTART. \*/

crSTART(xHandle);

for (;;)

{

crDELAY(xHandle, Time\_MillisecondsToTicks(50));

if(timed\_status == ENABLED)

{

taskTimed\_Execute();

}

}

/\* Co-routines MUST end with a call to crEND. \*/

crEND();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FreeRTOS Task implementation: Co-routine for heartbeat

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void prvHeartbeatCoRoutine(CoRoutineHandle\_t xHandle, unsigned portBASE\_TYPE uxIndex)

{

/\* Co-routines MUST start with a call to crSTART. \*/

crSTART(xHandle);

for (;;)

{

crDELAY(xHandle, Time\_MillisecondsToTicks(249));

taskHeartbeat\_Execute();

}

/\* Co-routines MUST end with a call to crEND. \*/

crEND();

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Private functions implementation

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void taskLED\_init(unsigned portBASE\_TYPE uxNumberToCreate)

{

xCoRoutineCreate(prvPeriodicCoRoutine, crf\_PERIODIC\_PRIORITY, crf\_PERIODIC\_INDEX);

xCoRoutineCreate(prvTimedCoRoutine, crf\_TIMED\_PRIORITY, crf\_TIMED\_INDEX);

xCoRoutineCreate(prvHeartbeatCoRoutine, crf\_HEARTBEAT\_PRIORITY, crf\_HEARTBEAT\_INDEX);

}

//turns on LED to begin, and turns it off when the appropriate time has passed

void taskTimed\_Execute(void)

{

portENTER\_CRITICAL();

{

if(timed\_count == 0)

{

turn\_on(LED\_YELLOW);

}

else if(timed\_count >= timed\_duration)

{

turn\_off(LED\_YELLOW);

timed\_status = DISABLED;

}

timed\_count++;

}

portEXIT\_CRITICAL();

}

// toggles periodic pulse LED

void taskPeriodic\_Execute(void)

{

portENTER\_CRITICAL();

{

toggle(LED\_RED);

}

portEXIT\_CRITICAL();

}

// toggles the heartbeat LED

void taskHeartbeat\_Execute(void)

{

portENTER\_CRITICAL();

{

toggle(LED\_POWER);

}

portEXIT\_CRITICAL();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

End of File

\*/

**taskUART.h**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: taskUART.h

\* Author: Anthony Bloch

\* Comments: transceiver task for uart. Receives protocol via uart and processes them accordingly

\* Revision history:

\*/

#ifndef TASK\_UART\_H

#define TASK\_UART\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

extern void taskUART\_init(void);

#ifdef \_\_cplusplus

}

#endif

#endif /\* TASK\_UART\_H \*/

**taskUART.c**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: taskUART.c

\* Author: Anthony Bloch

\* Comments: transceiver task for uart. Receives protocol via uart and processes them accordingly

\* Revision history:

\*/

#include <stdlib.h>

#include "FreeRTOS.h"

#include "timers.h"

#include "task.h"

#include "uart.h"

#include "gpio.h"

#include "adc.h"

#include <math.h>

#include "global\_variables.h"

#include "taskUART.h"

#include "../mcc\_generated\_files/pin\_manager.h"

#define taskUART\_PRIORITY 0

#define taskUART\_STACK\_SIZE 512

#define MAX\_PROTOCOL\_LENGTH 8

#define PROTOCOL\_BEGIN '<'

#define PROTOCOL\_END '>'

#define ENABLED 1

#define DISABLED 0

#define CH\_1 0

#define CH\_2 1

#define CH\_3 5

char protocol[MAX\_PROTOCOL\_LENGTH];

int protocol\_length;

int periodic\_pulse\_width = 1;

int periodic\_status = DISABLED;

int periodic\_count = 0;

int led\_red\_status = OFF;

int timed\_status = DISABLED;

int timed\_duration = 1;

int timed\_count = 0;

int led\_yellow\_status = OFF;

void receive\_protocol(void)

{

ch\_tx = '!'; // signal gui to send rest of command

uart\_send();

ch\_new = 0;

while(ch\_new == 0); // wait for first character of protocol

protocol\_length = 0;

while(ch\_rx != PROTOCOL\_END) // get protocol characters until end of string ('!') is indicated

{

if(protocol\_length < MAX\_PROTOCOL\_LENGTH) // save only the first MAX\_PROTOCOL\_LENGTH characters of protocol

{

protocol[protocol\_length] = ch\_rx; // save current character of protocol

protocol\_length++;

}

ch\_new = 0;

while(ch\_new == 0); // wait for next character in protocol

}

ch\_new = 0;

}

void process\_protocol(void)

{

int i, value;

if(protocol > 0) // protects against receiving only '!'

{

switch(protocol[0])

{

case 'a': // turn on LED

{

turn\_on(LED\_GREEN);

break;

}

case 'b': // turn off LED

{

turn\_off(LED\_GREEN);

break;

}

case 'c': // enable timed output

{

timed\_status = ENABLED;

timed\_duration = 0;

timed\_count = 0;

for(i = 1; i < protocol\_length; i++)

{

timed\_duration += (int)(protocol[i]-'0')\*(int)pow(10, (protocol\_length-i));

}

break;

}

case 'd': // disable timed output

{

timed\_status = DISABLED;

turn\_off(LED\_YELLOW);

break;

}

case 'e': // check status of timed output

{

ch\_tx = PROTOCOL\_BEGIN;

uart\_send();

ch\_tx = 'e';

uart\_send();

if(status(LED\_YELLOW) == 1)

{

ch\_tx = '1';

}

else

{

ch\_tx = '0';

}

uart\_send();

ch\_tx = PROTOCOL\_END;

uart\_send();

break;

}

case 'f': // enable periodic output

{

periodic\_status = ENABLED;

periodic\_pulse\_width = 0;

periodic\_count = 0;

for(i = 1; i < protocol\_length; i++)

{

periodic\_pulse\_width += (int)(protocol[i]-'0')\*(int)pow(10, (protocol\_length-i));

}

break;

}

case 'g': // disable periodic output

{

periodic\_status = DISABLED;

turn\_off(LED\_RED);

break;

}

case 'h': // read digital

{

ch\_tx = PROTOCOL\_BEGIN;

uart\_send();

ch\_tx = 'h';

uart\_send();

if(check\_digital() == 1)

{

ch\_tx = '1';

}

else

{

ch\_tx = '0';

}

uart\_send();

ch\_tx = PROTOCOL\_END;

uart\_send();

break;

}

case 'i': // read adc channel 1

{

value = check\_analog(CH\_1);

ch\_tx = PROTOCOL\_BEGIN;

uart\_send();

ch\_tx = 'i';

uart\_send();

i = 1000;

while(i > 0)

{

ch\_tx = value/i + '0';

uart\_send();

value = value % i;

i = i/10;

}

ch\_tx = PROTOCOL\_END;

uart\_send();

break;

}

case 'j': // read adc channel 2

{

value = check\_analog(CH\_2);

ch\_tx = PROTOCOL\_BEGIN;

uart\_send();

ch\_tx = 'j';

uart\_send();

i = 1000;

while(i > 0)

{

ch\_tx = value/i + '0';

uart\_send();

value = value % i;

i = i/10;

}

ch\_tx = PROTOCOL\_END;

uart\_send();

break;

}

case 'k': // read adc channel 3

{

value = check\_analog(CH\_3);

ch\_tx = PROTOCOL\_BEGIN;

uart\_send();

ch\_tx = 'k';

uart\_send();

i = 1000;

while(i > 0)

{

ch\_tx = value/i + '0';

uart\_send();

value = value % i;

i = i/10;

}

ch\_tx = PROTOCOL\_END;

uart\_send();

break;

}

default:

{

break;

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Private functions prototype

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static portTASK\_FUNCTION(vTaskUART, pvParameters);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FreeRTOS Task implementation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static inline void vCreatNewTasks(unsigned portBASE\_TYPE uxPriority)

{

xTaskCreate(vTaskUART, (char const\*)"taskName", (uint16\_t)taskUART\_STACK\_SIZE/(uint16\_t)(2), NULL, uxPriority, (TaskHandle\_t \*) NULL );

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Public functions implementation.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void taskUART\_init(void)

{

vCreatNewTasks(taskUART\_PRIORITY);

}

static portTASK\_FUNCTION(vTaskUART, pvParameters)

{

/\* Just to stop compiler warnings. \*/

(void) pvParameters;

//===========================================

//Task entrance

//===========================================

while(1)

{

if(protocol\_new == 1)

{

if(ch\_rx == PROTOCOL\_BEGIN) //signifies beginning of protocol

{

receive\_protocol();

process\_protocol();

}

protocol\_new = 0;

}

vTaskDelay(1);

}

}

**uart.h**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: uart.h

\* Author: Anthony Bloch

\* Comments: provides uart functionality to PIC24FJ256GA2017

\* Revision history:

\*/

// This is a guard condition so that contents of this file are not included

// more than once.

#ifndef UART\_H

#define UART\_H

#include <xc.h> // include processor files - each processor file is guarded.

extern char ch\_rx;

extern char ch\_tx;

extern int ch\_new;

extern int protocol\_new;

void uart\_init(void);

void uart\_send(void);

void \_\_attribute\_\_ ((interrupt, no\_auto\_psv)) \_U1RXInterrupt(void);

#endif /\* UART\_H \*/

**uart.c**

/\* Microchip Technology Inc. and its subsidiaries. You may use this software

\* and any derivatives exclusively with Microchip products.

\*

\* THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER

\* EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED

\* WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A

\* PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION

\* WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*

\* IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,

\* INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND

\* WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS

\* BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE

\* FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS

\* IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF

\* ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

\*

\* MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE

\* TERMS.

\*/

/\*

\* File: uart.c

\* Author: Anthony Bloch

\* Comments: provides uart functionality to PIC24FJ256GA2017

\* Revision history:

\*/

#include "uart.h"

#include "gpio.h"

char ch\_rx;

char ch\_tx;

int ch\_new = 0;

int protocol\_new = 0;

void uart\_init(void)

{

RPINR18bits.U1RXR = 2; //U1RX assigned to RP2

RPOR3bits.RP7R = 3; // U1TX assigned to RP7

U1BRG = 103; // 4 MHz and 9600 baud rate with BRGH = 1

ANSBbits.ANSB2 = 0; // turn off analog for Rx

U1MODE = 0;

U1MODEbits.BRGH = 1; // high speed -> 4 clocks per bit period

IEC0bits.U1RXIE = 1; // Rx interrupt enabled

IPC2bits.U1RXIP = 1; // priority

IFS0bits.U1RXIF = 0; // clear interrupt flag

U1MODEbits.UARTEN = 1; // enabling UART ON bit

U1STAbits.UTXEN = 1; // Transmit enabled

}

void uart\_send(void)

{

while(U1STAbits.UTXBF == 1); // wait if send buffer is full

U1TXREG = ch\_tx;

}

void \_\_attribute\_\_((interrupt, no\_auto\_psv)) \_U1RXInterrupt(void)

{

ch\_rx = U1RXREG; // read data received by UART

ch\_new = 1; // raise flag that new character was received

if(ch\_rx == '<')

{

protocol\_new = 1; // raise flag that new protocol is being sent

}

IFS0bits.U1RXIF = 0; // clear interrupt flag

}