ECE422: FINAL PROJECT REPORT

[Digital Acquisition Board and App

With Free-RTOS]

Student Name: [Swapnil Acharya]

Date: [12/14/2018]

Contents

[Background 1](#_Toc532814836)

[Goals and Objectives 1](#_Toc532814837)

[System Requirements, design specifications 1](#_Toc532814838)

[Design and Implementation 3](#_Toc532814839)

[Results 10](#_Toc532814840)

[Development Journal 18](#_Toc532814841)

[Analysis and Discussion 23](#_Toc532814842)

[Conclusion 23](#_Toc532814843)

[Acknowledgement 24](#_Toc532814844)

[Appendix A 24](#_Toc532814845)

# Background

Microcontroller nowadays are now foundation of majority of digital systems available today. However, a system is not complete without a Graphical user interface that allows users to communicate with and display the results of the microcontroller. So, this project will help understand how microcontroller is programmed and used with a graphical user interface and used in Industry.

# Goals and Objectives

The purpose of this project is to Create a C# graphical User Interface (GUI) that can interface with a Microcontroller (MCU). The microcontroller should be able to perform various task such as Analog to Digital Conversion, Pulse Width Modulation, read status of a Digital Input, drive a pin high for certain amount of time and Led Control. The commands to MCU to perform these tasks will be sent from GUI. A serial communication device will allow GUI to interface with MCU.

# System Requirements, design specifications

**System Requirements**

## The developer shall establish and document protocols for communication between Microcontroller (MCU) and Graphical User Interface (GUI).

## The developer shall develop encoders and parsers to facilitate smooth processing of communication protocols.

## The developer shall write a driver for MCU which is able to drive a LED high or low.

## The developer shall write driver for MCU which is able to produce a periodic waveform of 50% duty cycle, pulse width shall be provided by user.

## The developer shall write a driver for MCU which is able to drive an output pin high, where user inputs duration (in number of seconds). The timing shall start when user enables the output. When the output pin’s active duration equals or exceeds the duration, the output pin shall revert to default state (off state).

## The developer shall develop a driver which can report the remaining time of the timed output.

## The developer shall write a driver for Analog to Digital Converter Peripheral (ADC). The developer shall decide which sampling method is being used. The sampling methods can either be On-demand (sample when command is received) or Background (sample periodically in the background. Returns the most recent ADC value when command is received).

## The developer shall write a driver for a Digital Input pin which will allow status of a digital input to be read.

## The developer shall design a C# GUI which will use the communication protocol established in 1 to send commands (request packets) and accept response packets from MCU and displays the results of those commands.

## The MCU source code shall implement Free RTOS.

**Design Specification:**

## The communication protocol for both Microcontroller (MCU) and Graphical User Interface (GUI) was designed.

## An encoder function was designed on both GUI and MCU which would contain operation flags, delimiters, and the command from GUI/ response from MCU then turn this into a single data packet (a string made up of flags, delimiters and operation payloads).

## A parser was developed on both GUI and MCU side which would do operations based on data packets, extract the payload between delimiters from data packet when necessary and send response packet back to GUI.

## A LED control driver was written on MCU side which can drive an output pin high or low.

## Using steps provided by datasheet, a Pulse Width Modulation Driver was written on MCU side which can produce a square wave. The driver accepts pulse per second as use input then converts it to a period value which is then used to update period value of required registers. To maintain 50% duty cycle one of register was always assigned half of the value assigned to other register.

## Using Application Program Interface’s provided by Free Real Time Operating System (Free-RTOS) a timer Interrupt was implemented to Handle Timed output operation. The designated output pin is driven high when the timed output enable command is received. The designated output pin is driven low when the timer expires.

## On demand Sampling method was chosen. The driver provided by datasheet for Analog to Digital Converter Peripheral for pic24F was modified and used so that the channel number can be selected based on user input from GUI.

## A driver was written on MCU side which can read the status of a digital input pin.

## The developer shall write driver for MCU which is able to produce a periodic waveform of 50% duty cycle, pulse width shall be provided by user.

## The GUI source code was written using C# programming language in Visual Studio Community IDE whereas MCU’s source code was written in C programming language in MP lab-X IDE.

## Majority of MCU’s source was implement using Free RTOS

# Design and Implementation

## System Blocks

The system block Diagram (See Appendix- A, Fig 1) shows how individual components interact with each other. In software side, the individual components accept commands from user and the commands get encoded into data packet and are sent to Hardware Side’s Serial Communication Device. The serial communication acts like a bridge between hardware and software components. The serial communication device then transmits the data packets to MCU and then based on the data packet MCU perform various operations. The After performing the requested operation the MCU responds to GUI with response packets which are sent to serial communication device which then send the packets to GUI.

## Schematics

The Schematic (See Appendix-A, Fig 6) shows how all the hardware components are connected. The microcontroller (MCU). MCU’s Pin 2 is connect to green led which toggles every 1000ms, indicating MCU is functioning properly. MCU’s Pin 3 is connected to red led is received. MCU’s Pin 6 is connected to yellow led which turns on for requested time when timed output enabled command is receive and then turns off. MCU’s pin 15 is connected to a push button in a pull up configuration. This pin reads the status of push button when digital input status command is received. MCU’s Pin 26 outputs a square wave with 50 % duty cycle, whose width can be changed. MCU’s Pin 25, 24, 23 are designated as ADC channel 1,2 and 3 respectively, all three of them are connected to a Potentiometer’s wiper. When ADC command is received, one of these ADC pins read the analog voltage in the wiper and convert them to raw values. Finally, MCU’s pin 18 (uart transmit) and 17 (uart receive) are connected to a uart to usb’s receive and transmit pin respectively which is used to receive and responds to command sent from GUI.

## Software Components

Data Transmission:

The protocol written for data transmission between MCU and GUI:

LED Operation

GUI TO MCU

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload1 | Delimiter2 | End Flag |
| L | ^ | '1' if (ON)  '0' if ( OFF) | & | @ |
| 1 Byte | 1 Byte | 1 Byte | 1 Byte | 1 Byte |

MCU TO GUI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload1 | Delimiter2 | End Flag |
| L | ^ | '1' if Successful  '0' if Not Success | & | # |
| 1 Byte | 1 Byte | 1 Byte | 1 Byte | 1 Byte |

Timed Output Operation

GUI TO MCU

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload | Delimiter2 | End Flag |
| N if (Enable)  F if (Disable) | ^ | Time in seconds (string) | & | @ |
| 1 Byte | 1 Byte | Length of Payload | 1 Byte | 1 Byte |

MCU TO GUI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload | Delimiter2 | End Flag |
| T | ^ | '1' if Successful  '0' if Not Success | & | # |
| 1 Byte | 1 Byte | 1 Byte | 1 byte | 1 Byte |

Timed Output Status Operation

GUI TO MCU

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload | Delimiter2 | End Flag |
| S | ^ | ‘X’ (don’t care) | & | @ |
| 1 Byte | 1 Byte | 1 Byte | 1 Byte | 1 Byte |

MCU TO GUI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload | Delimiter2 | End Flag |
| S | ^ | “High” if High  “Low” if Low | & | # |
| 1 Byte | 1 Byte | Length of payload | 1 byte | 1 Byte |

Timed Output Remaining Time Operation

GUI TO MCU

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload | Delimiter2 | End Flag |
| R | ^ | ‘X’ don’t care | & | @ |
| 1 Byte | 1 Byte | Length of payload | 1 byte | 1 Byte |

MCU TO GUI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload | Delimiter2 | End Flag |
| R | ^ | Remaining Times | & | # |
| 1 Byte | 1 Byte | Length of payload | 1 byte | 1 Byte |

Digital Input

GUI TO MCU

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload | Delimiter2 | End Flag |
| D | ^ | ‘X’  (Don’t Care) | & | @ |
| 1 Byte | 1 Byte | 1 Byte | 1 Byte | 1 Byte |

MCU TO GUI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload | Delimiter2 | End Flag |
| D | ^ | “High” if High  “Low” if Low | & | # |
| 1 Byte | 1 Byte | Length of payload | 1 byte | 1 Byte |

PWM

GUI TO MCU

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload | Delimiter2 | End Flag |
| I if Enable  J if Disable | ^ | Pulse in seconds (String) | & | @ |
| 1 Byte | 1 Byte | Length of Payload | 1 byte | 1 Byte |

MCU TO GUI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload | Delimiter2 | End Flag |
| P | ^ | '1' if Successful  '0’ if Not Success | & | # |
| 1 Byte | 1 Byte | 1 Byte | 1 byte | 1 Byte |

ADC

GUI TO MCU

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation Payload | Delimiter2 | End Flag |
| A | ^ | Channel No (String)  From 1 to …Nth channel | & | @ |
| 1 Byte | 1 Byte | Length of Payload | 1 byte | 1 Byte |

MCU TO GUI

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Operation Flag | Delimiter1 | Operation  Payload 1 | Delimiter2 | Operation Payload2 | Delimiter 3 | End Flag |
| A | ^ | Channel No (String)  From 1 to …Nth channel | & | Raw ADC Value (String) | \* | # |
| 1 Byte | 1 Byte | Length of Payload | 1 Byte | Length of payload | 1 Byte | 1 Byte |

MCU FIRMWARE: (See Figure -16, Figure -17, Appendix A for Component Dependencies)

Free RTOS:  
It is a real-time operating system kernel for embedded devices that has been ported to 35 microcontroller platforms. Free-RTOS is designed to be small and simple. The kernel itself consists of only three C files. To make the code readable, easy to port, and maintainable, it is written mostly in C, but there are a few assembly functions included where needed (mostly in architecture-specific scheduler routines). Free RTOS method used in this project include multiple tasks and software timers.

ADC.c, .h  
This source and header files contain initializations and implementations for ADC module. This source file also contains the function that starts, stops sampling and converts analog value to digital based on analog channel and delay between start and stop sampling.

Timed Output.c,.h  
This source and header file contains implementation for timed output module. This file contains Software timer and software timer handler provided by Free RTOS.

dataPacketHelpers.c,.h

This source and header file contain implementation related to data packet encoding and payload extraction and header flags.

LED.c,.h

This source and header file contain implementation related to Led controls and status.

C# GUI:

When user presses certain commands on GUI the depending upon command a data packet which includes operation flag, payload if necessary, delimiters and end flags get encoded into a string and this packet is sent to MCU though Serial Communication Port.

When the end flag of a response packet from MCU is received, based on operation flag in the data packet certain functions are triggered in a separate threads (process) that update Gui textboxes or other elements.

## Application Workflow

Parser and Encoder: (See Appendix -A, Fig 3 and 4 respectively For Flowchart)

The application first initializes all the tasks and then keeps looping on the respective task that been scheduled to run. Majority of the application is focused on one task which handles all the commands received.

As shown by flowchart, the application stored stores all the commands into a buffer. When the end flag is received, based on the operation flag (part of command packet) respective operation are performed.

In general, when an end flag is received, then depending upon operation flag, the payload is extracted from data packet and necessary operation are performed. Once the operations are performed then the result of those operations are stored in a response packet and then sent to GUI.

## Analog to Digital Conversion

Successive-approximation-register (SAR) analog-to-digital converters (ADCs) are frequently the architecture of choice for medium-to-high-resolution applications with sample rates under 5 mega samples per second (Msps). Resolution for SAR ADCs most commonly ranges from 8 to 16 bits, and they provide low power consumption as well as a small form factor. This combination of features makes these ADCs ideal for a wide variety of applications, such as portable/battery-powered instruments, pen digitizers, industrial controls, and data/signal acquisition.  
As the name implies, the SAR ADC basically implements a binary search algorithm. Therefore, while the internal circuitry may be running at several megahertz (MHz), the ADC sample rate is a fraction of that number due to the successive-approximation algorithm.

The AD Converter used in this project is based on Successive Approximation Register (SAR) architecture. In this project raw ADC values are used instead of converting raw ADC values to their respective voltage readings. However, they could be easily converted in to voltage readings. The resolution of AD converter used in this project is 10-bit and the reference voltage used is 3.3 V. The corresponding voltage reading would be given by formula

e

## C# Application

Parser and Encoder: (See Appendix -A, Fig 5 and 2 respectively For Flowchart)

The C# Gui allows user to send various commands to Microcontroller. The C# GUI code composer class to display what data is being sent from Microcontroller to Gui in character and hex format. The Gui also allows user to manually type commands and send commands like a terminal. When user presses any buttons in the C# GUI, the respective command is encoded into a data packet string which contains operation flags, delimiters, payload and end flag. Finally, that string is sent to Microcontroller.

The C# GUI has its own Serial Port Data receive handler which gets triggered when data is received. As the flowchart shows all the data received is stored into a data packet buffer (string). Then code scans the string to see whether an end flag has been received. If an end flag has been received, then code checks for operation flag on the same string and depending upon operation flag on the string the code invoke separate threads to update various GUI elements like textboxes and others.

# Results

Test Cases (For Pictures Refer To Fig 7-15 in Appendix – A)

| Test ID | Action | Expected outcome | Pass/Fail | Points |
| --- | --- | --- | --- | --- |
| 1.1 | On C# GUI, click “LED on” button | LED on breadboard turned on | Pass | 1 |
| 1.2 | On C# GUI, click “LED off” button | LED on breadboard turned off | Pass | 1 |
| 1.3 | On C# GUI, click “LED on” button twice | LED on breadboard turned on | Pass | 1 |
|  |  |  |  |  |
| 2.1 | On C# GUI, under Timed Output:  Enter 2 in the text box | Value 2 appeared in text box, no other actions are triggered | Pass | 1 |
| 2.2 | Follows test 2.1, click “Enabled” button | The corresponding output on breadboard is activated for 2 seconds +/- 100ms and then reverts to original state. | Pass | 1 |
| 2.3 | Have stopwatch timing device ready.  On C# GUI, under Timed Output:  Enter 10 in the text box, click “Enabled” button and start stopwatch.  5 second +/- 500ms after the button-click, click “Enabled” button.  Stop the stopwatch when the corresponding output reverts to original state. | Total time for output to be activated should be **15** +/- 500ms. | Pass | 2 |
| 2.4 | Have stopwatch timing device ready.  On C# GUI, under Timed Output:  Enter 20 in the text box, click “Enabled” button and start stopwatch.  Then enter 2 in the text box.  5 second +/- 500ms after the button-click, click “Enabled” button.  Stop the stopwatch when the corresponding output reverts to original state. | Total time for output to be activated should be **7** +/- 500ms. | Pass | 2 |
| 2.5 | On C# GUI, under Timed Output:  Enter 10 in the text box, click “Enable” button and start stopwatch.  Click ‘Status’ button at 1 second +/- 200 ms interval until the corresponding pin revert to original state. | On C# GUI, the corresponding text box should display the remaining number of seconds until the output pin is reverted to original state. | Pass | 2 |
| 2.6 | On C# GUI, under Timed Output:  Enter 10 in the text box, click “Enable” button and start stopwatch.  On 1 second +/- 500ms after output is activated, click ‘Disable’ button | The corresponding output on breadboard should revert to original state within 100ms of the button click. | Pass | 2 |
|  |  |  |  |  |
| 3.1 | On C# GUI, under Periodic Output:  Enter number in the text box, click “Enable” button  Under Periodic output textbox enter 100 (pulse/second) and then click enable | The corresponding output shall produce periodic output according to the following specifications:  The output period should be around 10 ms (100 Hz). | Pass | 1 |
| 3.2 | Test case 3.1 to be reviewed by instructor | The test case content must pass instructor’s review process. | Pass | 2 |
|  |  |  |  |  |
| 4.1 | Setup variable voltage circuit to produce voltage range between 0V and 3.3V  Measure voltage level rail-to-rail on voltage meter. | Minimum voltage is +0V.  Maximum voltage is +3.3V. | Pass | 1 |
| 4.2 | Adjust variable voltage to 1V +/- 0.1V.  On C# GUI, click ‘Read’ button | The corresponding ADC value should be displayed on the corresponding text box. | Pass | 2 |
| 4.3 | Enter any number between 401-409 (pulse/second) in periodic textbox, then click enable | The output periodic waveform should have a period around 2.43ms  (f = 410 Hz) | Pass | 2 |
| 4.4 | Supply an input voltage between 3.3 and 3.5 V to the designated ADC channel 1 pin. Then click channel 1 button. | The RAW Adc value should be 1023 (ADC steps) | Pass | 2 |
|  |  |  |  |  |
| E1.1 to  E1.3 | Enter 2000 in periodic textbox. Then click enable | The output periodic waveform should be around 449 ms (f = 2.2 KHz) with around 43% duty cycle | Pass | 10% of total |
|  |  |  |  |  |
| E2.1 to  E2.3 | Enter 100 in Timed Output textbox. Then click enable. After 10s (±500ms), Enter 10 in Timed Output textbox. Then click enable. | Total time for output to be driven high should be around 20s (±500ms), | Pass | 10% of total |
|  |  |  |  |  |
| R1.1 | Random test to be requested by instructor.  Test area:   * start / restart application | C# application behave the same on each restart/shutdown – no crash | Pass | -2 if failed |
| R1.2 | Random test to be requested by instructor  Test area:   * application behavior when serial cable is disconnected | C# Application remain responsive.  C# application gracefully ignore user action | Pass | -2 if failed |
| R1.3 | Random test to be requested by instructor   * Follow up on test cases if the instructor picks up something unexpected |  | Pass | -2 if failed |

# Development Journal

Include your development logs here.

|  |  |
| --- | --- |
| Date/Time | Description |
| 11/21/18  12:00 PM  12:33 PM  1:00 PM  2:00 PM  3:00 PM | Project Started  Project Created  Coroutine no working  Added hearbeat.c, taskSerial.c,.h and update mcc generated files  Led using in heartbeat Coroutine not working |
| 11/24/18  12:01 PM  1:15 PM  1:38 PM  1:51 PM  2:36 PM  3:09 PM  3:20 PM  3:27 PM | fixed, greed led issue (was not initialized as output pin)  added PWM.c, Datasheet reading for PWM driver.  added OCM1 initializations  added timer2 initializations and helpers  modified timer2 initialization, added function to calculate period  added pin for OCM1, initialized timer 2 and OCM1  updated PWM.c  changed PWM pin from 10 to 12 |
| 11/25/2018  9:06 AM  9:10 AM  11:24 AM  11:37 AM  11:40 AM  11:45 AM  11:52 AM  12:55 PM  2:00 PM  2:16 PM  2:33 PM  2:43 PM  4:33 PM  5:35 PM  5:53 PM | PWM driver added, PWM working  removed timer 3 driver from PWM.h  ADC driver started  added ADC initializations  added ADC reading based on channel  added init ADC to system.c  update ADC.h  update pins for uart, ocm1  added adc input pins  only 1 out of 3 adc channel working  added all 3 PWM channel  work on adc restarted  adc all channels working  added timer 2 init, interrupt isr  fixed timer and heartbeat issue |
| 11/26/2018  9:58 AM  10:17 AM  1:03 PM  1:09 PM | work on timed output started  added elapsed time function in timerInt.c  added task timed output  removed task timed output |
| 11/29/2018  2:15 PM  2:21 PM  3:03 PM  3:52 PM  4:10 PM | reduced pwm channels from 3 to 1  updated OCM1 functions  changed pwm source to timer  updated function types in ocm.c,.h  updated timer drivers |
| 12/01/2018  11:55 AM  12:11 PM  12:18 PM  1:06 PM  1:13 PM  1:15 PM  1:23 PM  1:25 PM  1:27 PM  1:30 PM  1:31 PM  2:09 PM  2:16 PM  2:37 PM  3:44 PM  4:23 PM  16:32 PM  5:38 PM | changes made in taskUart.c  added data packet header macros  removed dpacket.h , update misc.h with flags  updated pin manager.h,.c with new pins  updated LED.c,.h  updated LED.c,.h  updated pin manger to support more analog pins  updated pin manager.c to support rb12  updated led.c to support rb12  heartbeat not working  fixed: wrong led placement  updated data packet flags in mic.h  led operation not working  led operation working  encoder not working  updated encoder  LED ON/OFF functional  OCM functional |
| 12/02/2018  11:42 AM  12:05 PM  12:16 PM  12:21 PM  1:15 PM  1:22 PM  2:05 PM  2:16 PM  2:19 PM  2:31 PM  6:01 PM  6:22 PM  6:39 PM  6:42 PM  7:10 PM  7:22 PM  7:44 PM  7:50 PM | added encoder and updated command strings  added delegate method in gui  added multiple delegates  added contents to delegates  basic commands send through gui working  updated gui to directly send command on click  added parser method  gui not interfacing with MCU  end flag fixed  parser working, all but timed output and status work  AC implement in task UART  ADC somewhat working  added Digital input operation in taskUART.c  digital input functional  working on timed output  updated timer2interrupt driver.c  added timed output operation and acknowledge  added timed output and status codes |
| 12/03/2018  9:18 AM  9:37 AM  10:04 AM  10:31 AM  10:49 AM  11:09 AM  12:08 PM  12:59 PM  1:11 PM  1:19 PM  1:34 PM  6:37 PM  6:55 PM  7:10 PM  7:24 PM  8:08 PM | wok on timed output started again  decided to use RTOS timers  rtos timer somewhat working  updated rtos timer and not compiling  added timed output driver.c,.h  moved all RTOS timer implementation to timed Output driver.c  added xtimerchangeperiod  added timedotuput apis to taskUART  timed output without relcik functional  timed output working  timed output status works, period not working  timer based timed output limited  updated misc.c and added task timed output  updated task timed output, task uart, misc.c  task timed output not compiling  task timed output compiling |
| 12/04/2018  08:54 AM  09:13 AM  2:43 PM  2:46 PM  3:10 PM  6:55 PM  7:05 PM  8:45 PM  9:21 PM | updated UART pins for GA702  created new project for KM202, work on UART  uart1 and 2 working for KM202  pins updated, uart1 and 2 working for km202  added buffers to uart1 and uart2  updated timed output  updated timed output period = 0 condition  added remaining time operation for timed output  added remaining time operation working |
| 12/06/2018  12:34 PM  3:12 PM  3:19 PM  4:24 PM | refactored directories for normal and extra credit  multiple uart problem fixed  uart1 and uart2 communicating  script somewhat working |
| 12/09/2018  10:39 PM  4:00 PM | extra credit not working  extra credit not working completely |

# Analysis and Discussion

Led turned on/off when led on/off command is received. The led didn’t toggle or turn off when turn on command is received more than once. The textboxes displayed accurate value that was being entered by user. The timed output drived output pin high and then turned it off after requested time. When timed output was clicked twice the output pin did not drive low, it stayed high for the new requested period. The remaining time of timed output is displayed accurately by the system. The status of digital input pin was read accurately by the system. The period waveform worked well for pulse per second values less than 650. However, it did not produce accurate and precise waveform for values above 650. It was usually off by a factor of 0-20 pulse per second. The raw ADC (ADC Steps) values are read well however they fluctuate between ±3. The heartbeat coroutine worked well to indicate that MCU is working well.

Since specific range for periodic output waveform was not given, the period output waveform did not quite achieve was it was meant to, but it worked well. The clock source used for period wave form was based on 16-bit timer. Whenever, values higher than 16-bit, despite being prescaled were used it did not perform quite well. Other than this the all the system requirements and design specification were met.

Major roadblock was to get timed output to work. Hardware timer had limitation of duration. However, Free-RTOS’s powerful tool, software timers came quite handy. The use of software timer solved this problem as they were able handle larger values for timed output.

The system can be used by using advance features of Free-RTOS like queues and others. More threads (tasks) could have been used to achieve optimal performance. Multiple tasks would have made Background ADC sampling very easy. In depth analysis and research on AD converters would have helped achieving more out of ADC modules.

This system performs most tasks of a general system. This system can be sued to test sensors, check PWM based motors, indicators and many other applications where prototyping is a must.

This system heavily relies on serial communication and can easily be disrupted by disturbing serial communication protocol. The data packets can be modified by injecting undesired payload or modifying delimiters. The data packets need more layers of security and echo back system to ensure correct transmission has taken place.

# Conclusion

A Digital Data Acquisition system with Free-Real time operating system was designed and developed. The system’s two major components the Graphical User Interface (GUI) and Microcontroller (MCU) communicated with each other using serial communication protocol via a UART to USB bridge. The MCU receives command from the GUI and can perform various tasks such as pulse width modulation, analog sensor readings, indicator controls (led), timed output and read status of digital inputs. This system introduced Real timed operating system on an Embedded system which can be vastly expanded to achieve maximum out of an Embedded system.

# Acknowledgement

1. PIC24FJ256GA702 and PIC24FV16KM202 Datasheet, for help in Writing Drivers

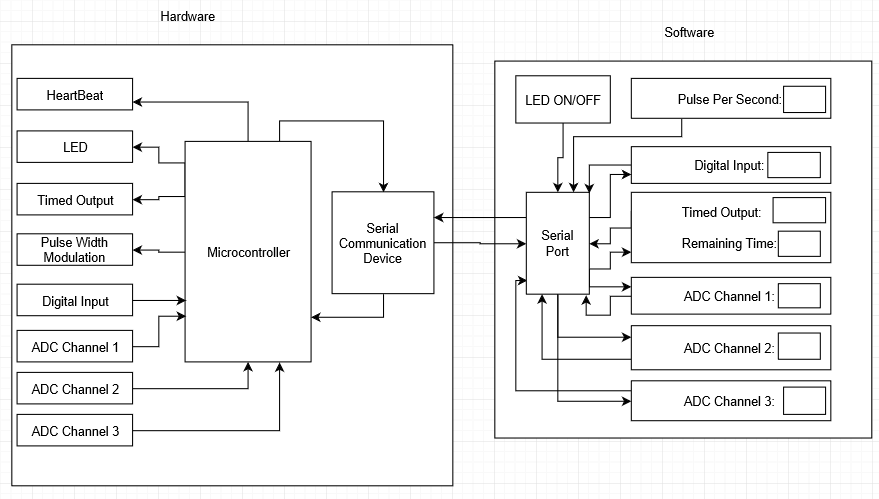
2. PIC24F Family Reference Manual(s), for help in Writing Drivers

3. Microchip Code Configuration for help in Writing Drivers

3. Instructor Peng Lei, for providing Data Acquisition C# Gui, protocol design and troubleshooting

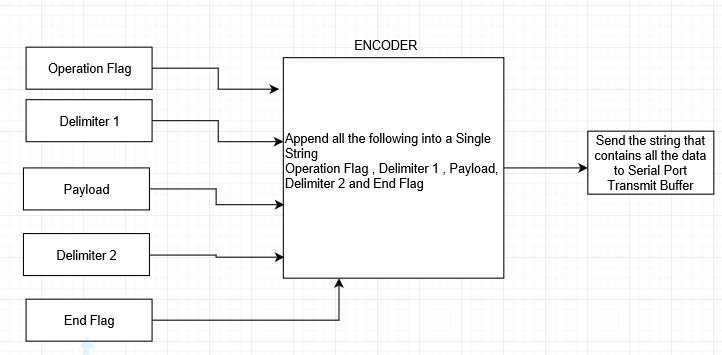
# Appendix A

**System Block Diagram**



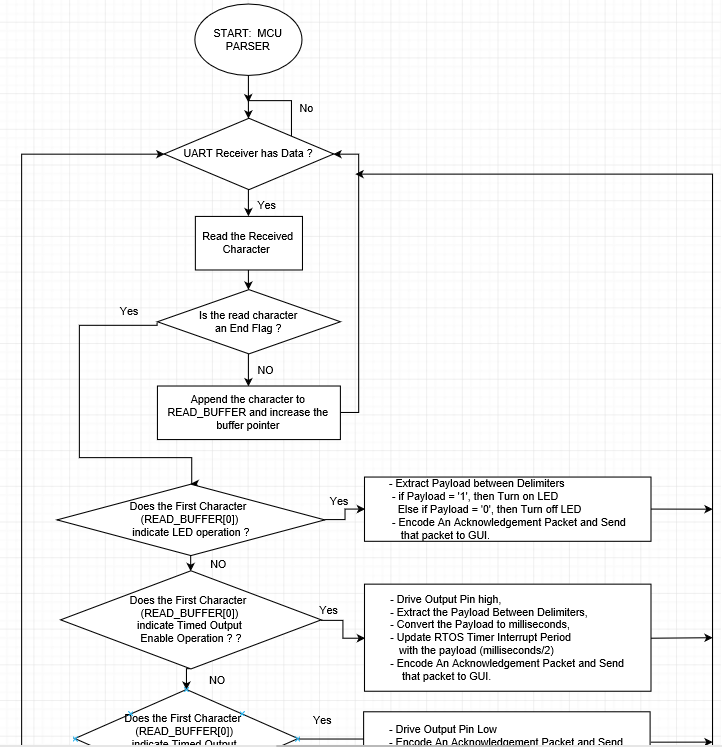
**Fig 1: System Block Diagram**

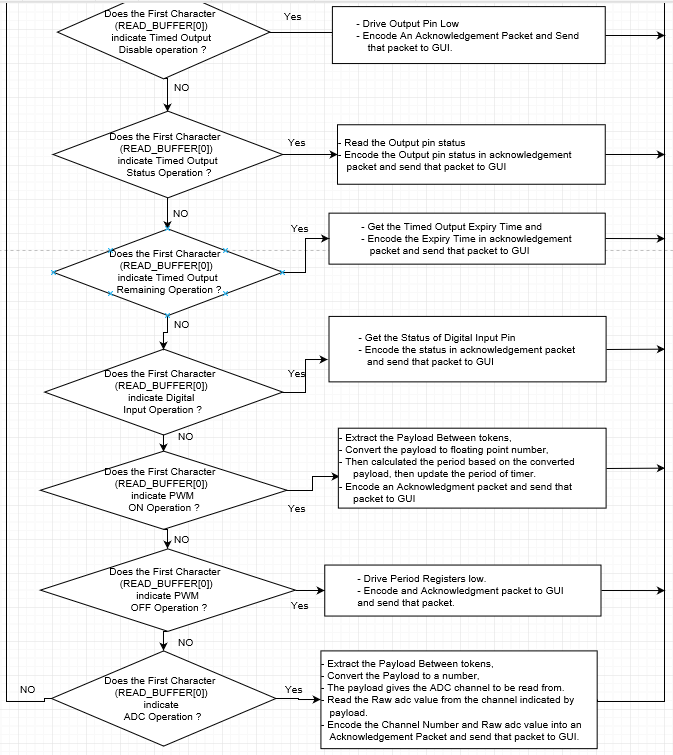
**GUI ENCODER**



**Fig 2: GUI Encoder**

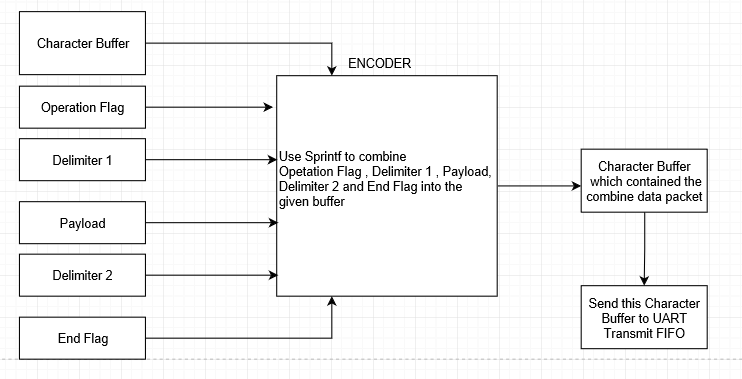
**MCU PARSER**





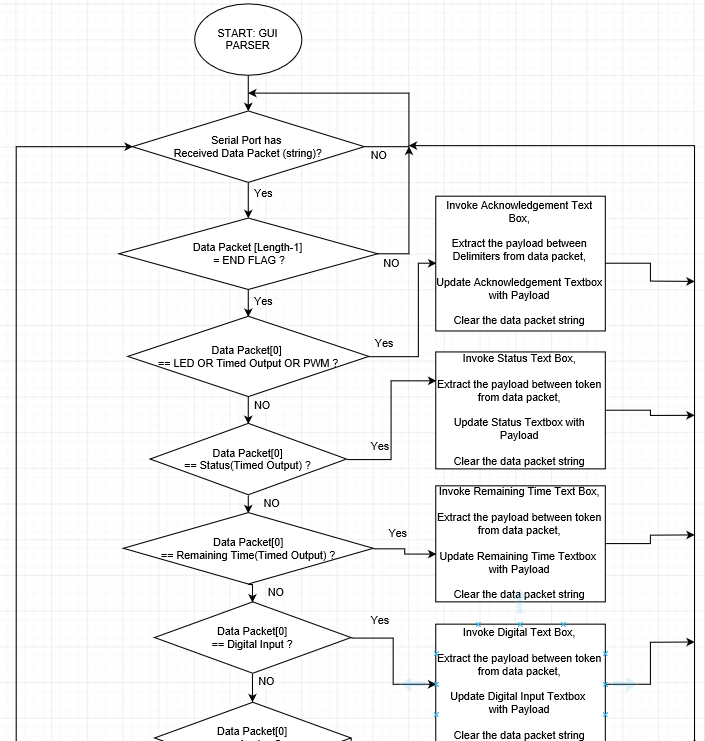
**Fig 3: MCU Parser**

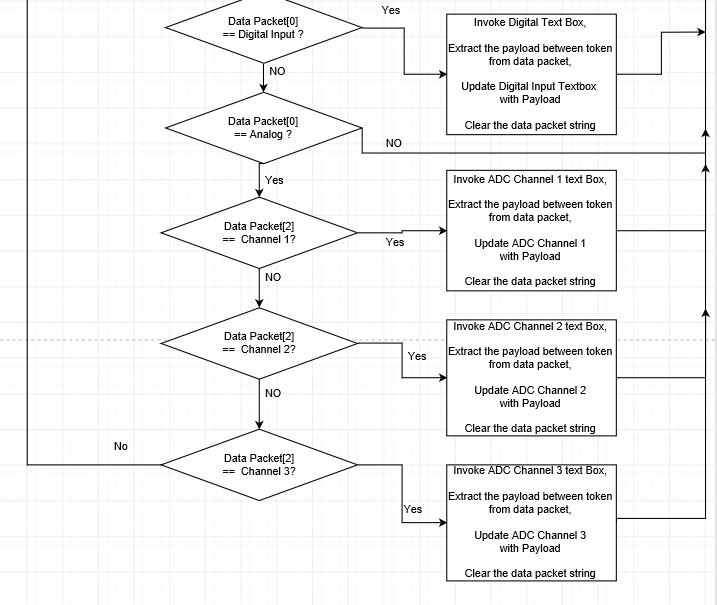
**MCU ENCODER**



**Fig 4: MCU Encoder**

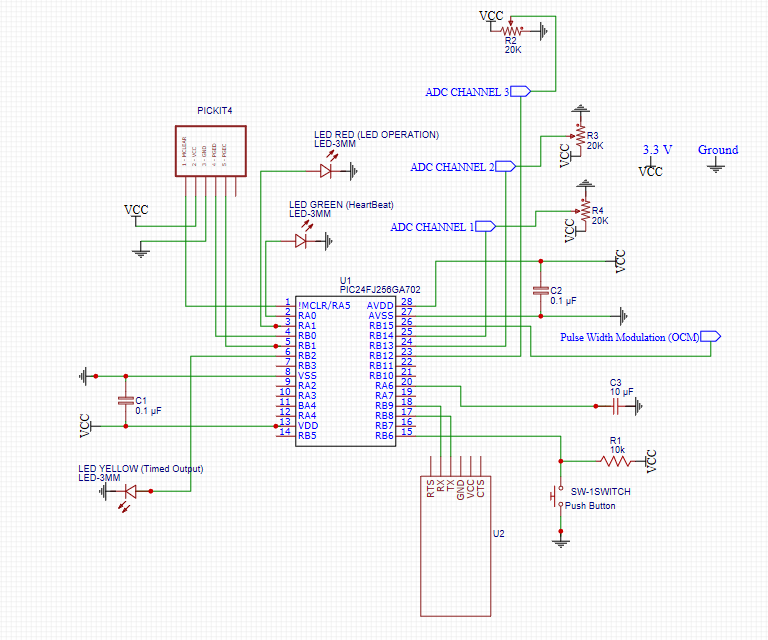
**GUI PARSER**





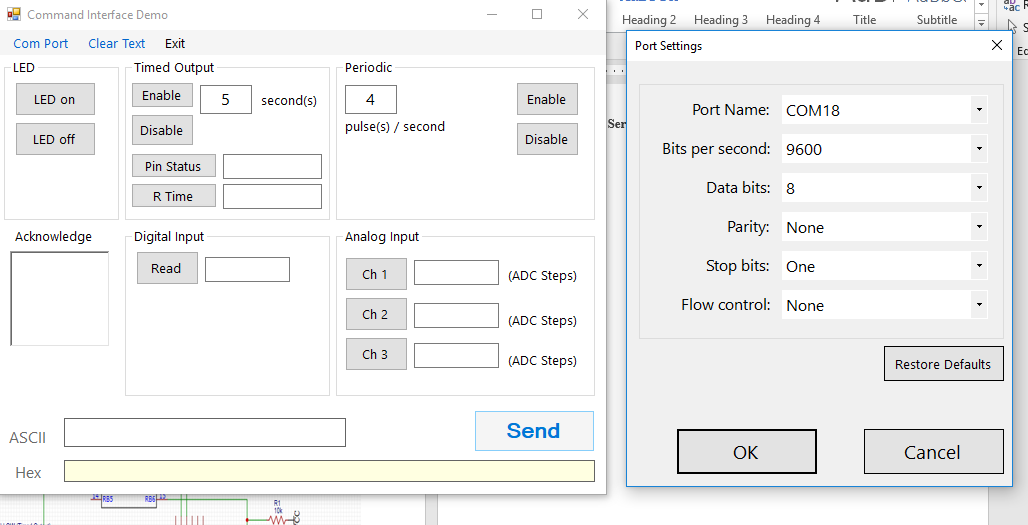
**Fig 5: GUI Parser**

**Hardware Schematic:**

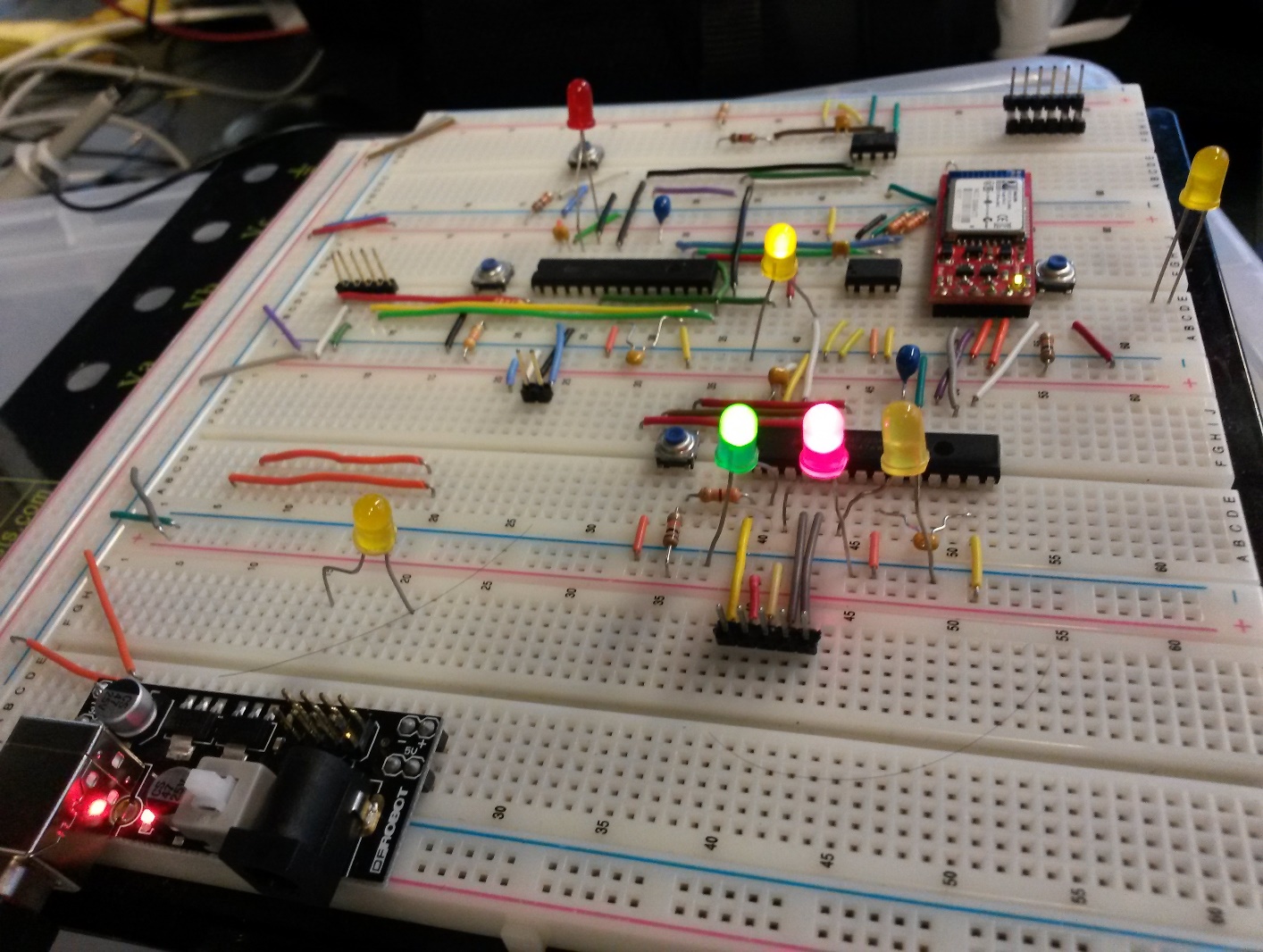
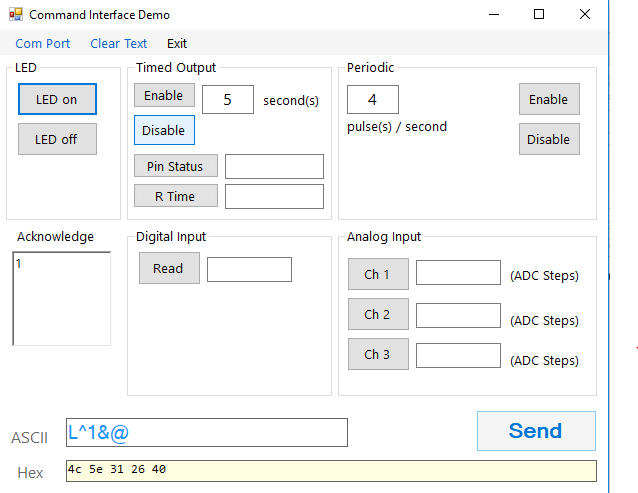


**Fig 6: Hardware Schematic**

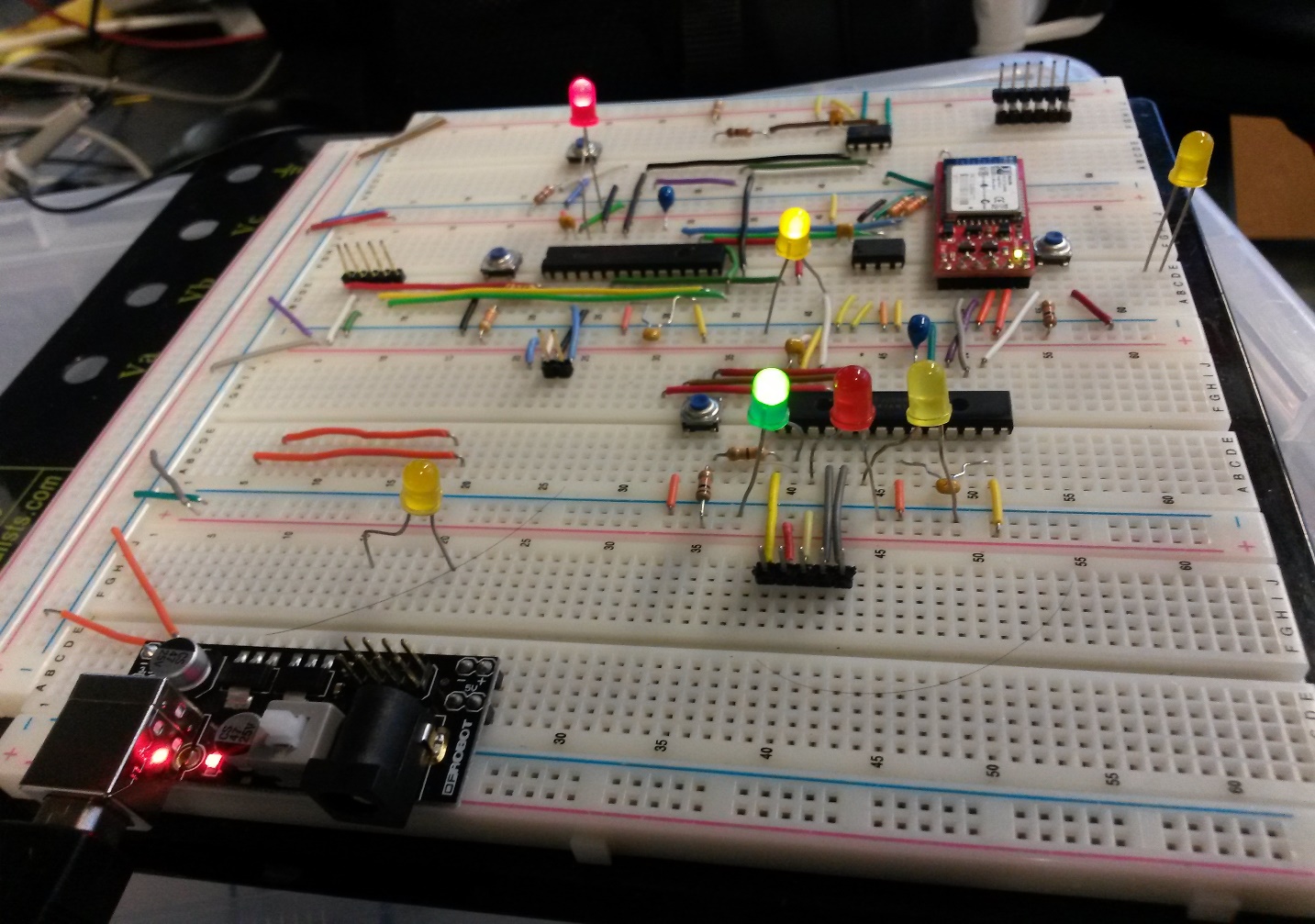
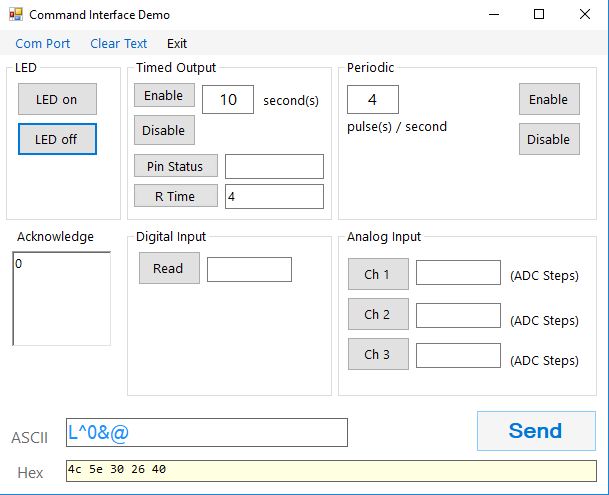
**Test Results:**



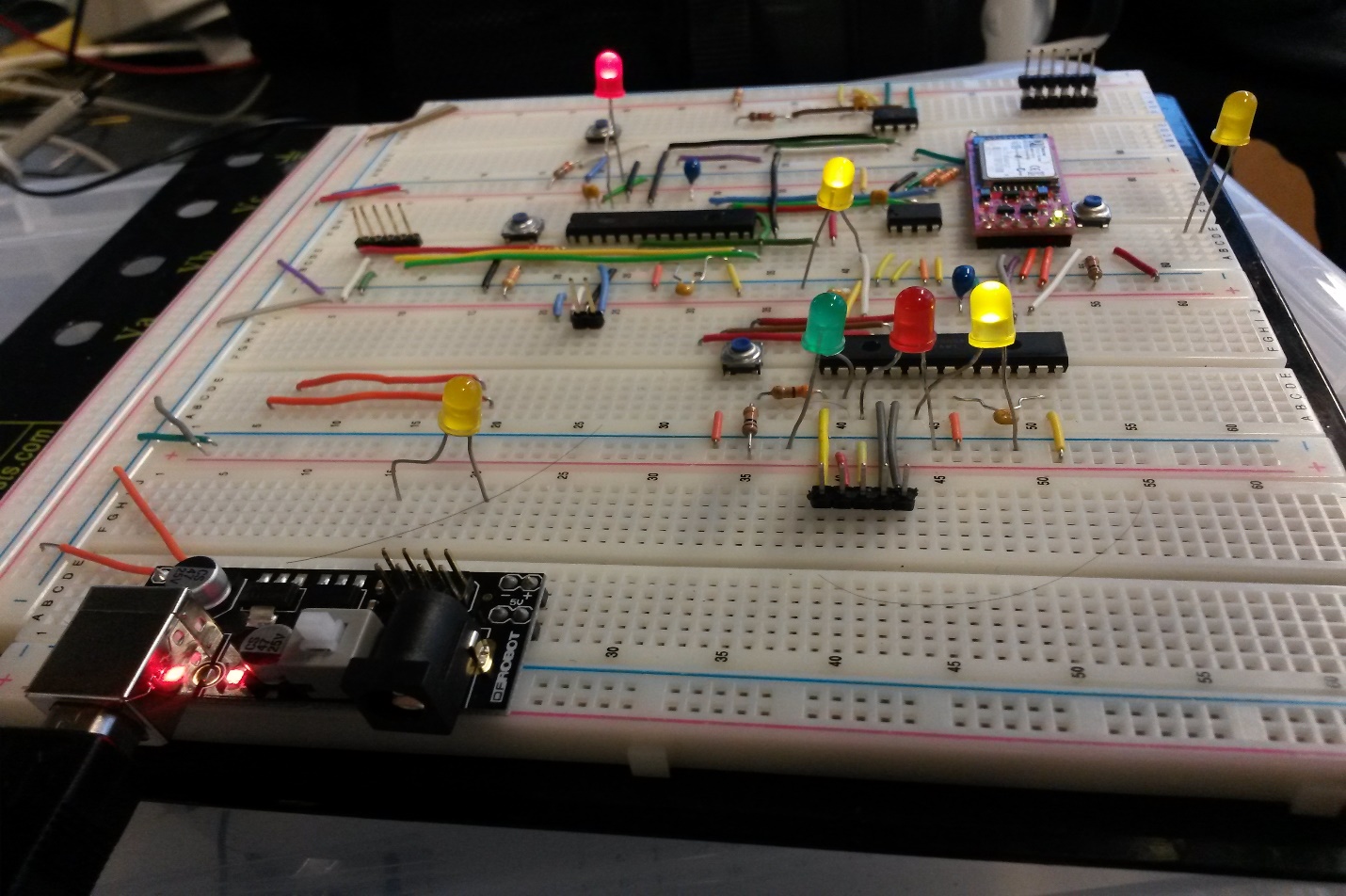
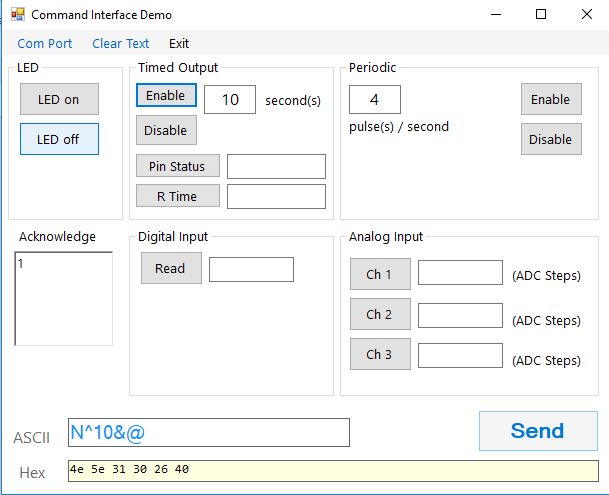
**Fig 7: GUI Connecting to Serial Communication Device:**



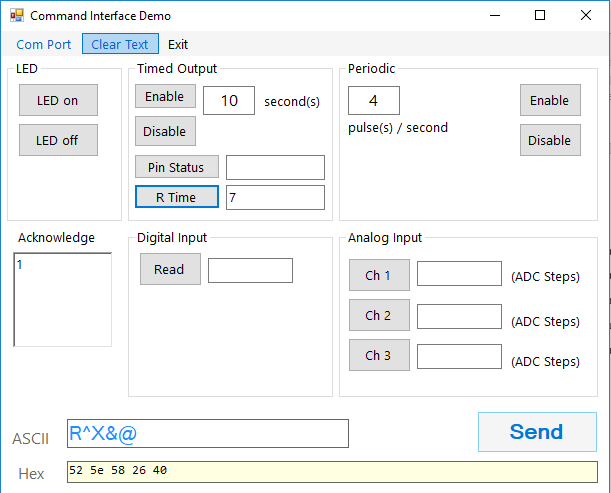
**Fig 8: Tuning Led On (Red Led turned On)**



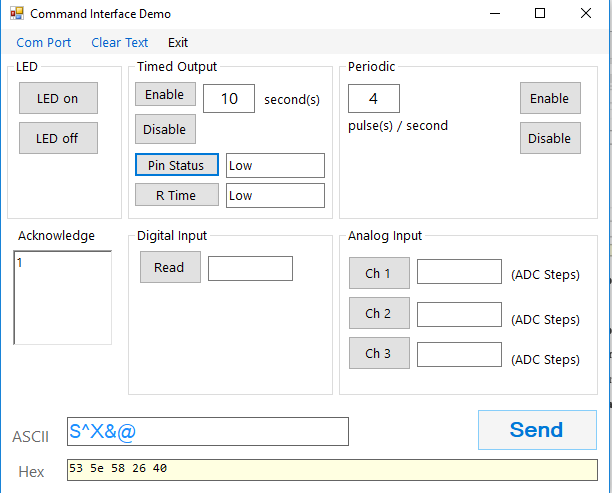
**Fig 9: Turning Led off (Red Led turned Off)**



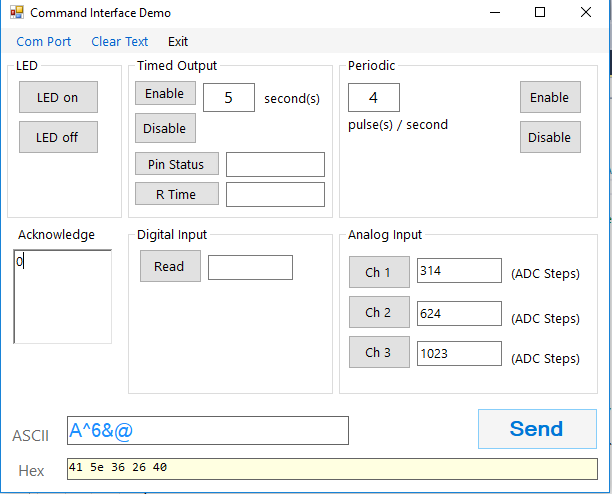
**Fig 10: Enabling Timed Output (Yellow Led turned on )**



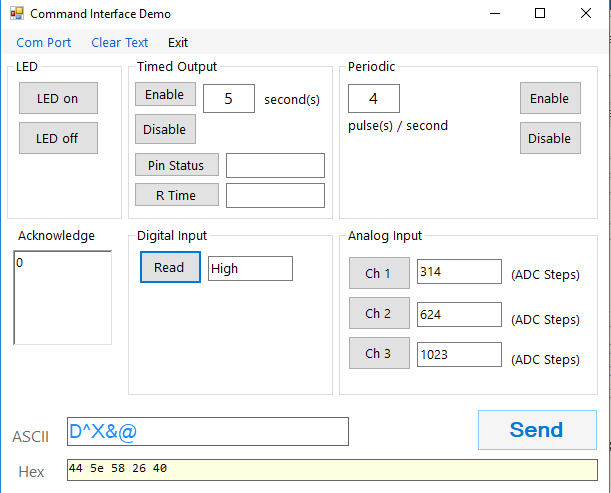
**Fig 10: Timed Output Remaining Time**



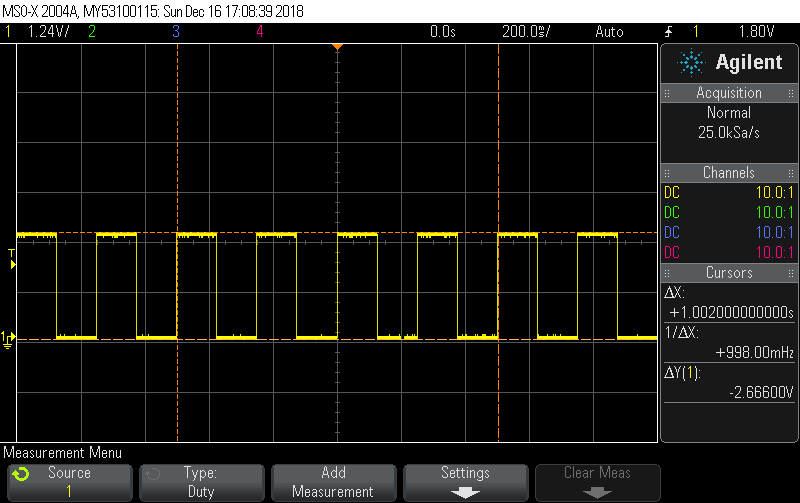
**Fig 11: Timed Output Disable or Expired**



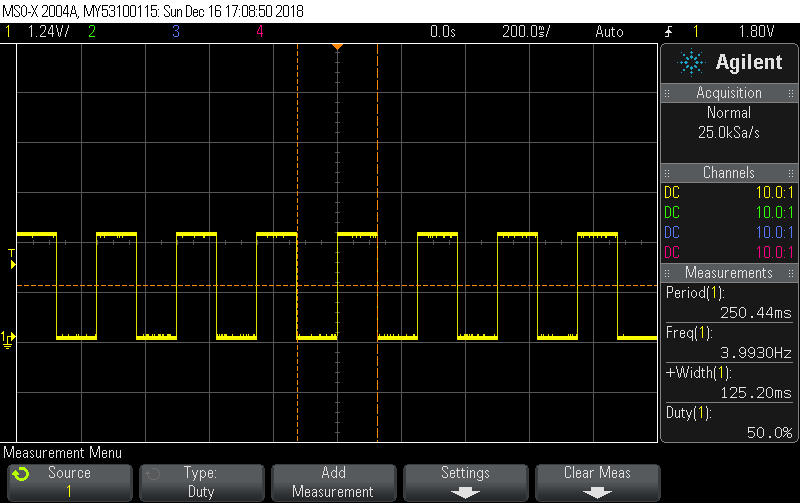
**Fig 12: ADC Reading For 1.0 V (Ch1), 2.0 V(ch2), 3.3 V(Ch3)**



**Fig 13: Digital Input Status Reading**

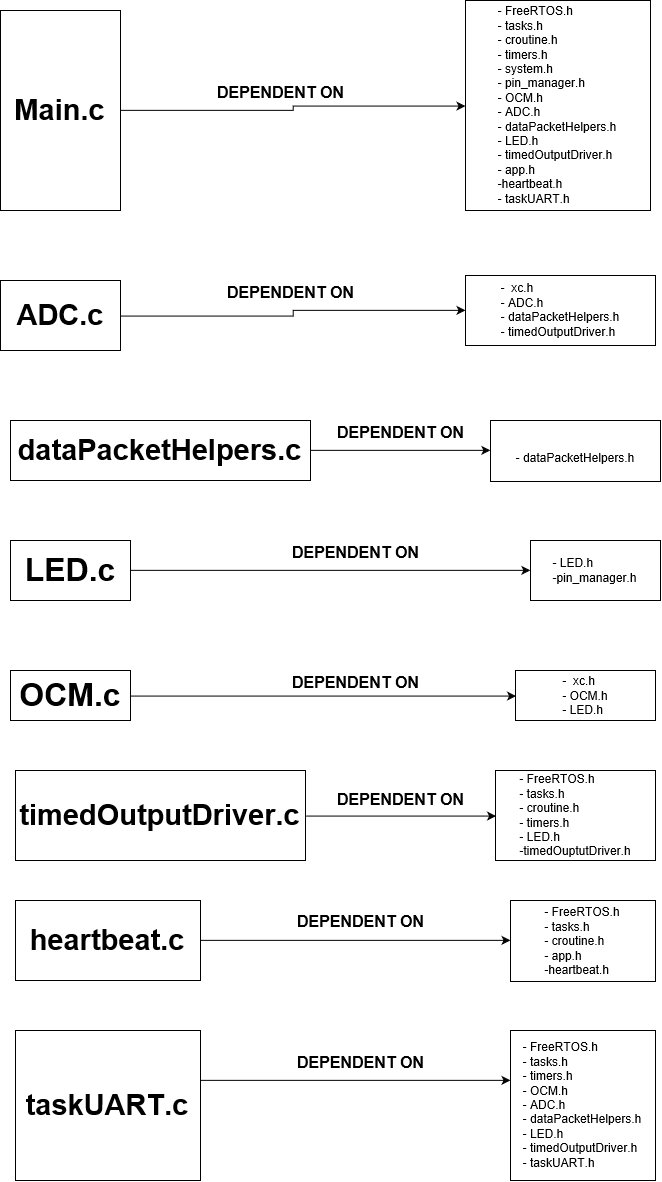


**Fig 14: Periodic Waveform (4 pulses/second)**

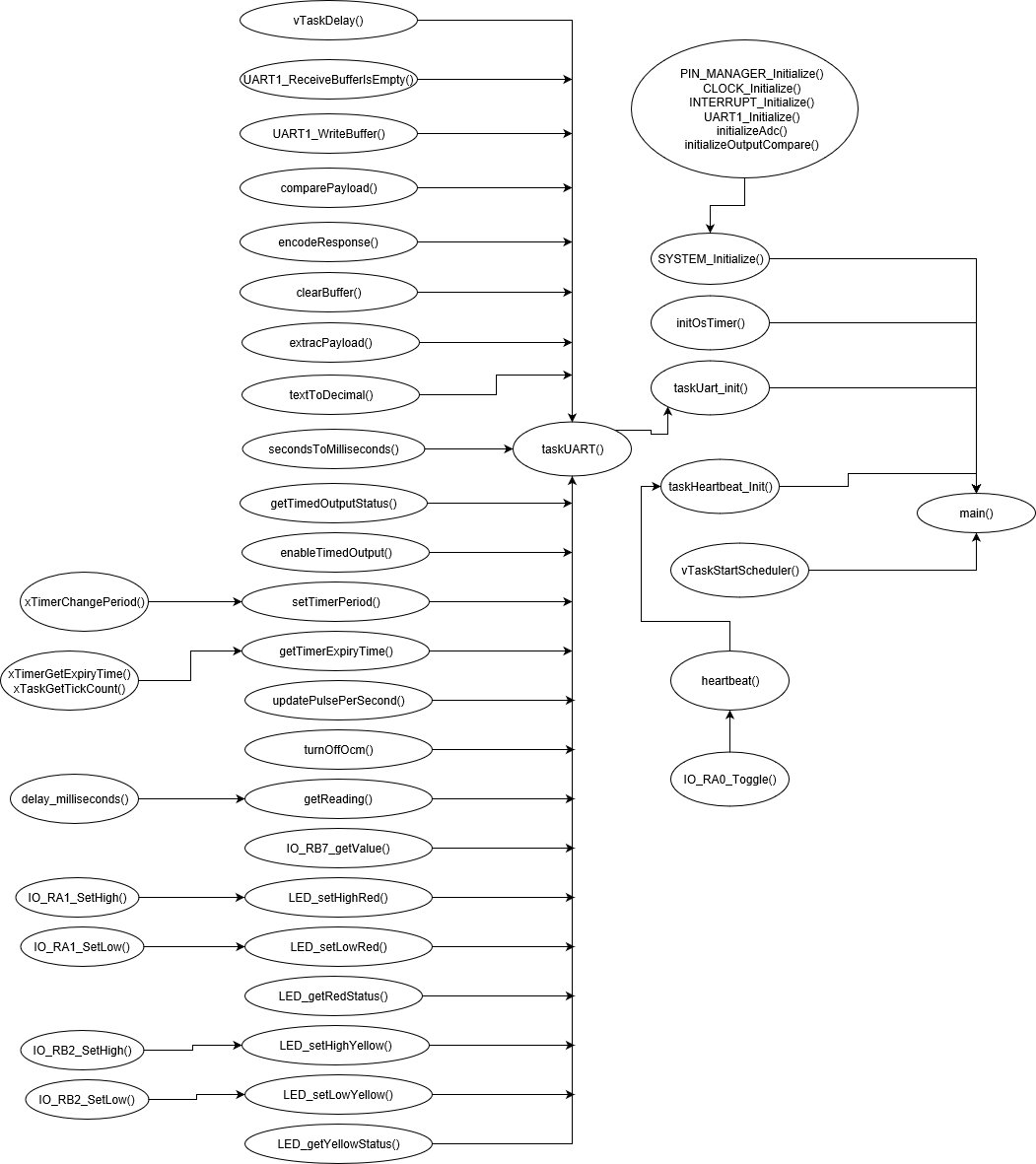


**Fig 15: Periodic Waveform 4Hz**

**Dependencies**



**Fig 16: Component Dependencies\_1**



**Fig 17: Component Dependencies\_2**