Modular Rapid Monitoring System

Timothy Kritzler and Joseph Mintun

Sponsored by:
Martin Engineering, Illinois
Bradley University Electrical and Computer Engineering Department

Liaisons: Dr. Malinowski and Dr. Ahn

May 4, 2016

Executive Summary

The purpose of the Modular Rapid Monitoring System is to monitor the status of large three-phase motors used in equipment designed and sold by Martin Engineering. The motors used in Martin Engineering's products usually operate for long periods of time and in extreme temperatures because of their applications in large scale bulk material systems. Because the failure of one motor can cause the whole system to fail, predicting when a failure is imminent is greatly beneficial. The modular rapid monitoring system, or MRMS, monitors and records electrical and movement parameters of the motor which in the future may lead to development of a system capable of predicting the imminent failure. The MRMS monitors voltage and current of each of the three phases along with vibration data and displays the data in a concise and easy to read fashion. By viewing this data, a technician can determine if a motor failure is imminent and may even give an estimated remaining lifespan.

The MRMS consists of two main subsystems: the sensor interface system, or SIS, and the gateway interface system, or GIS. The SIS collects voltage and current data from an analog to digital converter or ADC along with data from a 3-axis accelerometer communicating through I^2C, a digital communication protocol widely used in microcontrollers. The SIS then scales the data and sends it through UART, a simplified serial communication, to the GIS. The GIS then stores the data in permanent memory. A lightweight web server is hosted on the GIS which displays the data. In order to access the web server, the GIS acts as an access point and creates a Wi-Fi network.

Low cost of the entire system is an important factor in the design. Because of this, cost is the primary factor when choosing. The GIS consists of a Raspberry Pi ($30) with a custom embedded Linux build, a RT5370 USB Wi-Fi dongle ($12), and a 2A USB power adapter ($4). In addition to the low cost, the GIS components were chosen because of the compatibility with each other, and the extensive online developer community support. The SIS consists of an Atmel UC3-A3 prototyping board ($32), an ADXL335 accelerometer ($15), and a 2A USB power adapter ($4). These components were chosen because of the low cost and extensive features. The Atmel UC3-A3 board has an external memory chip on board that can be used to store the data in case there is not enough internal memory on the processor chip itself.

Abstract

The Department of Electrical Engineering has partnered with Martin Engineering to create a prototype of a Modular Rapid Monitoring System. The goal of this project is to produce a monitoring system to enhance diagnostics and detection of imminent machine failures. The historical data of recorded transient responses helps diagnose and troubleshoot problems with the machine. This helps to reduce downtime as well as make the production process safer.

Table of Contents

I. INTRODUCTION

A. Problem Background

Martin Engineering is a small company based in Neponset, IL. They have a long history developing innovative products to move bulk materials in mining and other industries. In order to improve their current products, they want to monitor the voltage, temperature, and revolutions per minute of the motors used to run the equipment. Martin Engineering has a relationship with Bradley University and decided to sponsor a senior project for the Department if Electrical and Computer Engineering. In the 2014-2015 academic year, one team at Bradley University worked on developing such a system. This team successfully developed a rough prototype, but unfortunately it was missing major features. The goal of this project is to develop a more refined proof of concept system by building off of the current prototype.

B. Problem Statement

Martin Engineering has asked for a cost effective modular rapid monitoring system that is broken up into two subsystems: a sensor interface system, or SIS, and a gateway interface system, or GIS. The SIS monitors six ADC channels, one digital from a three-axis accelerometer, and system temperature within milliseconds after power is applied. The SIS must store the data briefly while the GIS is booting, then send the data to the GIS. The GIS then stores data received from the SIS and makes it available to users using a web server by communicating through Wi-Fi.

C. Constraints

- Overall System
  - System must have a free commercial license.
  - System must keep track of total time in powered up state without time
- Gateway Interface System
  - Must start logging data within ten milliseconds
  - Must log data at 600±10 samples per second
  - Must store the first five minutes of data after system power up
  - Must store the most recent five minutes of data and keep it available on next power up
  - Must keep the total time, or alternatively the number of motor revolutions
  - Must keep the minimum and maximum of the temperature, accelerometer data, current, and voltage for the duration of use
- Sensor Interface System
  - Must communicate through Wi-Fi (ad-hoc or infrastructure)
  - Must not transmit unscaled values to user
  - Must contain a Web server to provide read-only access to data

II. STATEMENT OF WORK

A. System Block Diagram

Fig. 1 shows the system block diagram of the modular rapid monitoring system, or MRMS. The MRMS consists of two main systems: the sensor interface system and the gateway interface system. The sensor interface system receives digital input data from the accelerometer and six analog voltages (the three currents are measured by reading voltage across three resistors and using an ohm's law calculation). The sensor interface system then sends the data collected to the gateway interface system which stores the information and makes it available to the user interface.

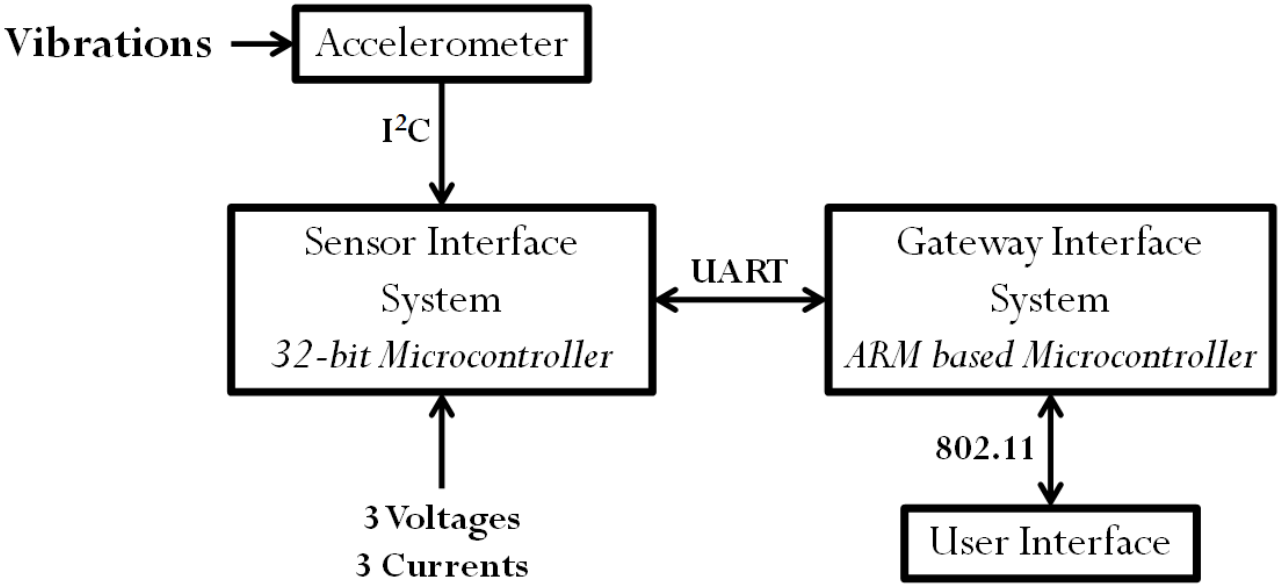


Figure 1: System Block Diagram

Table I: System Block Diagram Input/Output Descriptions

Input	**Description**
Accelerometer	Digital input (I^2C or SPI).
6 Voltages	Converted to digital with Sensor Interface system ADC.
3 Currents	Voltage over resistor is measured and converted to digital with Sensor Interface system ADC then converted back to current internally.
Output	**Description**
User Interface	HTTP Web server accessible with web browser of user's choice.

B. Subsystem Block Diagram

Fig. 2a shows the sensor interface system block diagram. The inputs to the system are acceleration, voltage, and current data. The analog to digital converter, or ADC, converts the voltage and current data into digital data and the accelerometer uses I²C communication. Data from the two parts is stored in a rotary buffer that is sent to the GIS via UART. The system uses a timer and interrupt controller to perform the processes.

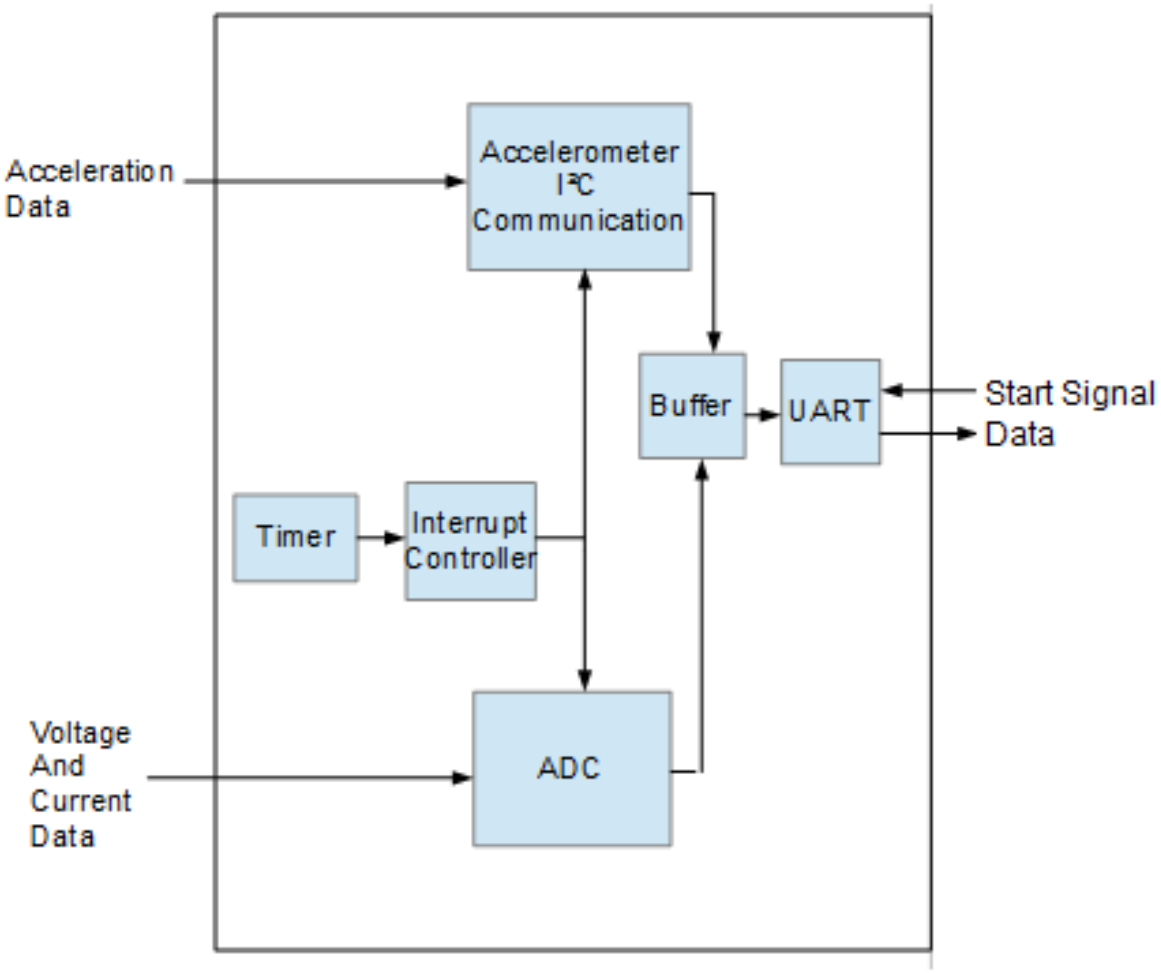


Figure 2a: Sensor Interface System Block Diagram

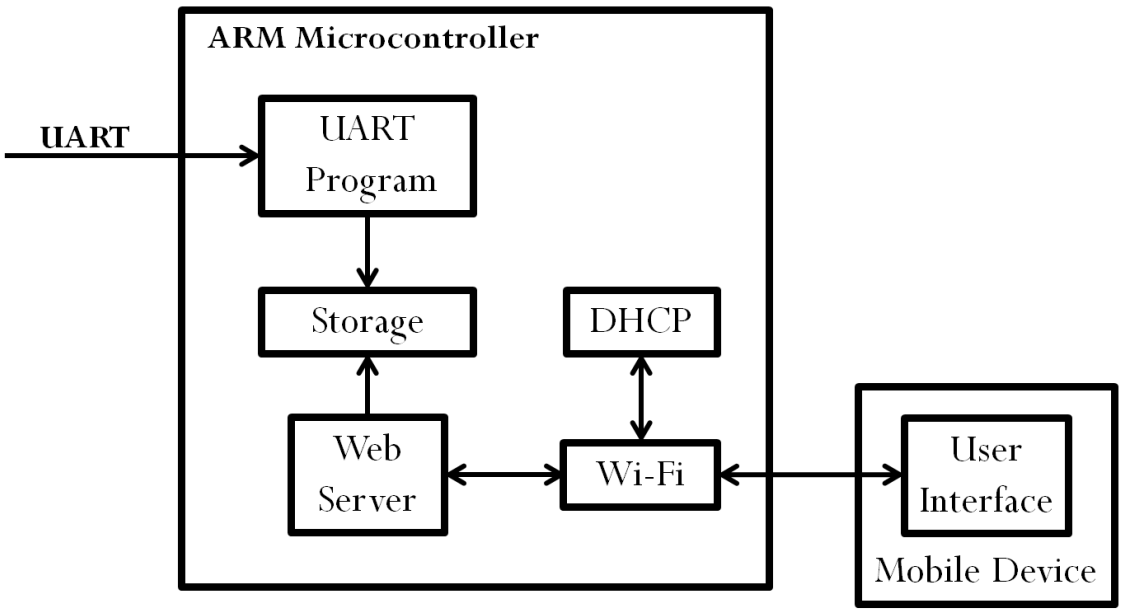


Fig. 2b: Gateway Interface System Block Diagram

Fig. 2b shows a closer look at the gateway interface system. After the data is received from the sensor interface system, the gateway interface system must store it in permanent memory, publish it on web server, and run an 802.11 wireless network to allow connectivity to web server.

C. System State Diagram

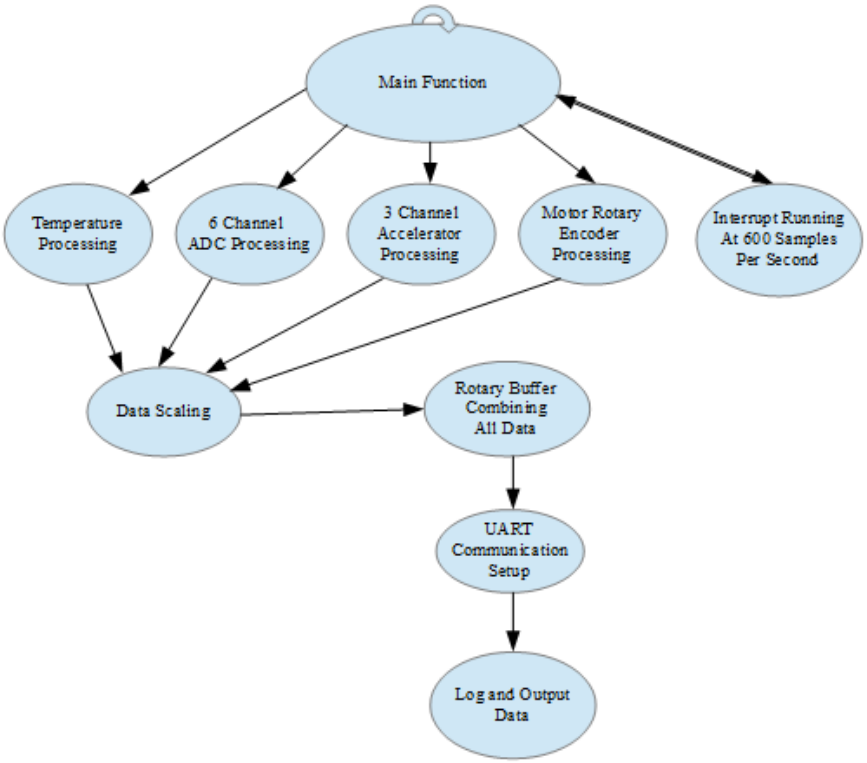


Figure 3: System State Diagram

The “Main Function” runs continuously while power is applied. It has an interrupt running at 600 times a second and output a variable to the “main function”. The “Temperature Processing”, “6 Channel ADC Processing”, and “3 Channel Accelerometer Processing” functions have an input from the “Main Function” which tells them to read inputs from a temperature sensor, 6 A/D channels, and an accelerometer, respectively. They output the data read from the respective inputs. The “Data Scaling” function accepts the data output from “Temperature Processing”, “6 Channel ADC Processing”, and “3 Channel Accelerometer Processing”. It then performs mathematical operations needed to scale data and output the data. The “Rotary Buffer” function accepts the scaled data and stores it for a short amount of time before sending it to the “UART Communication Setup”. Next, “UART Communication Setup” sends the data to be permanently stored and displayed for the user on the Gateway Interface System.

D. Nonfunctional Requirements

Objectives

- The system is low cost: under $350, but the lower the better.
- The system withstands large amounts of vibration. The system is attached to large motors which cannot affect the accuracy of the data logging.
- The data format for the web server is easy for the user to understand. It should be aesthetically pleasing while still displaying all the relevant information.

E. Functional Requirements

Sensor Interface System (SIS)

- The SIS begins to store data within 250 ms after power is applied.
- The SIS measures and temporarily buffers six analog channels (three for voltage and three for current) within a deviation of 0.2 Volts.
- The SIS measures and temporarily buffers three digital accelerations through the use of a three-axis accelerometer.
- The SIS measures and temporarily buffers changes in temperature.
- The SIS calculates total number of motor revolutions from voltages and currents.
- The SIS transmits the measured and buffered data to GIS as soon as GIS is available for receiving them. The transmission parameters are 115,200bps, 8bits, no parity, and 2 stop bits.

Gateway Interface System (GIS)

- The GIS communicates with rapid monitoring system at 119,500 bits per second
- The GIS receives and stores data from SIS and store up to 4 gigabytes of data on a memory card.
- The GIS makes the data available for download on a Web page displayed using a Web server accessible by connecting to a wireless network configured on the GIS.

III. DESIGN APPROACH AND METHOD OF SOLUTION

Martin Engineering has requested for the product design to contain two separate modules that communicate with each other in order to accomplish the final output to a web server. The first part of the design is the sensor interface system, or SIS. The board chosen for this system is an Atmel UC3-A3 Xplained. This board is ideal because of the low cost and the on board external memory. An AVR Dragon programmer is used to transfer firmware written in Atmel Studio to the board. The board is also capable of I^2C communication to the three-axis accelerometer, an ADXL335. This accelerometer was chosen because of its support of multiple communication protocols (SPI and I^2C, both are widely used protocols in microcontrollers) and its low cost.

The second subsystem is the gateway interface system, or GIS. Raspberry Pi was chosen due to the numerous features, powerful processor, vast online developer community support, and low cost. The Linux distribution used previously was not compatible with Wi-Fi, so a different lightweight microcontroller Linux is utilized. This ensures that the previous driver problems do not occur when the RT5370 USB Wi-Fi dongle is interfaced. The SIS communicates with the GIS via UART communication.

There are many different hardware options available for both the SIS and GIS. For the SIS, there are many variations of a 32-bit microcontroller that may have worked for this application. Since there was already an Atmel UC3-A3 Xplained board from the previous year, it was decided that it would remain as the development board. The GIS could have used a Beaglebone Black, but this design would not have met the constraint that the final product must have a free commercial license.

IV. ECONOMIC ANALYSIS

As stated in the constraints, the cost of the final product must be less than $300. The cost of each component that is utilized is shown in Table II. The two boards that are used for the GIS and SIS have the biggest effect on the budget, and still use only about twenty percent of the maximum acceptable price.

Table II: Component Pricing

Part	**Cost**	**Store**
Raspberry Pi	$30.00	Element14 Online Store
Atmel UC3-A3 Xplained	$31.25	Atmel Online Store
ADXL335 Accelerometer	$15.00	Sparkfun Online Store
2x 2A Micro USB Power Adapter	$7.99	Amazon
RT5370 USB Wi-Fi Adapter	$11.99	Amazon

The only expenditures for this project occur in the components. All software used is open source or provided free from manufacturer. This leads to a total cost of $96.23 which is well within the constraint.

V. PROJECT TIMELINE

For the first half of this project, there are five deliverables to ensure that progress is being made. First, the proposal presentation and document must be completed. Next, the webpage and progress presentation occur. Finally, there is an end of semester performance review which marks the halfway point for the project. A detailed list of these deliverables can be found in Appendix C.

A full Gantt chart is also shown in Appendix C. From this chart, the critical path for the project can be found. Since the project was split up the whole time until the final weeks, there are two paths that were fulfilled in order for the project to be successful. These are shown in Table VI of Appendix C with both paths leading to the combined test. For the SIS, interfacing the accelerometer properly takes the most time. For the GIS, optimizing the operating system boot time took the most time because it is critical that it receives data as quickly as possible.

VI. DIVISION OF LABOR

As previously discussed, this project has two main modules. This was taken into consideration when deciding how to divide the tasks. One member of the team developed the sensor interface system while the other developed the gateway interface system. This ensured that the team accomplishes the tasks in the most efficient way possible because it allows for each member to focus on a single specialized piece of hardware. The website that contains information about the project as well as progress updates was modified by both members. The specific task breakdown is shown in Table III.

Table III: Division of Tasks

Subsystem	Task	Timothy Kritzler	Joseph Mintun
Sensor Interface System	Develop ADC controller		X
	Develop Serial Communication with GIS		X
	Optimize Rotary Buffer for the Data		X
	Accelerometer Interfacing		X
	Data Storage During on SIS during GIS Boot		X
	Correct Timings for Sending Data		X
Gateway Interface System	Research and decide base OS	X	
	Interface Wi-Fi	X	
	Develop UART Access program	X	
	Develop lightweight web server	X	
	Optimize boot time	X	
	Optimize Web server GUI	X	
	Combined SIS/GIS Testing and debugging	X	
N/A	Develop Progressive Website	X	X

VII. SOCIETAL AND ENVIRONMENTAL IMPACTS

The main companies impacted by the use of this product are Martin Engineering and any subsidiaries that may receive the device through their company. The extensive data logging system helps to alert these companies to approaching breakdowns which increases the safety of their employees. It also helps the company to pinpoint the source of any problems that occur. This reduces downtime for repairs which helps to maximize productivity.

Since the device is used to sense potentially dangerous breakdowns, the final product must meet the functional and non-functional requirements in all scenarios. Ethically, the team cannot submit any work unless this condition is met due to the liability risk of someone getting injured.

VIII. SUMMARY AND CONCLUSIONS

The modular rapid monitoring system helps to improve the efficiency of repairs and upgrades to the machines that Martin Engineering decides to attach it to. The product contains two main modules: the sensor interface system and the gateway interface system. The SIS, which is utilizing an Atmel UC3-A3 Xplained microcontroller, logs requested data such as temperature, vibrations, current, and voltage. The GIS, using a Raspberry Pi that runs a lightweight Linux, receives this data and print it to a web server. This data can then be compared to data from operating periods where the machine did not experience any problems. This comparison makes troubleshooting the machine much more efficient which saves the company time and valuable resources that could be allocated elsewhere.

IX. REFERENCES

[1] K. Palmer and S. Shelton, "Modular Rapid Monitoring System".

[2] K. Palmer, “Senior Project Laboratory Notebook,” unpublished.

[3] S. Shelton, “Senior Project Laboratory Notebook,” unpublished.

[4] "Raspberry Pi Forum", Tiny Core Linux Forum, 2016. [Online]. Available: http://forum.tinycorelinux.net/index.php/board,57.0.html.

APPENDIX A: TESTING PROCEDURES

A. Gateway Interface System

Testing of GIS subsystem as a wireless Access Point is done by attempting to connect to the network with a PC. In order to be considered "connected", (1) the laptop must be assigned an IP address and (2) a ping from the laptop to the IP address of the GIS must be successful.

Testing of the GIS's web server is first done by attempting to download the index.html file through local loopback by using the wget command. Once that is successful, the website is accessed through the web browser of a laptop connected to the GIS through Wi-Fi.

Testing of the GIS's UART communication starts with a simple test of the interface. A computer running Hyper-terminal connects to the GIS running Minicom. If the interface is working, keyboard characters pressed on one device will appear on the other's terminal screen. Next, a program written in C running on the GIS records received data into a text file stored in the GIS's memory. This program is tested by sending data from Hyper-terminal in the same fashion as before.

Finally, testing of the GIS subsystem as a whole is done by (1) sending data from a computer running Hyper-terminal and (2) accessing the Web page and downloading the data on a computer connected to the GIS's Wi-Fi.

B. Sensor Interface System

Testing of the analog to digital converter for the sensor interface system is done with six potentiometers with varying voltages. The data from the analog to digital converter is compared to the voltages from the potentiometers shown on the oscilloscope. If the system is working properly, the values should be the same.

Testing of the UART communication protocol utilizes a computer running hyper-terminal. The start bit is sent from the hyper-terminal which starts the transfer of data from the rotary buffer. If the program is working correctly, the data from the rotary buffer continues to appear in the terminal.

APPENDIX B: GATEWAY INTERFACE SYSTEM SETUP

Below is the procedure for setting up the Gateway Interface System on the first version of the Raspberry Pi. The newest version of Tiny Core Linux (Version 7.0) is compatible with the Raspberry Pi 2 and Raspberry Pi 3 The following procedure should be compatible for either one with few complications. Tiny Core Linux can be found at http://www.tinycorelinux.net/.

A. Required Materials

The required materials are listed below.

- Raspberry Pi
- MicroSD card 4GB or larger. An 8GB card is used.
- RT5370 USB Wi-Fi dongle
- PC or Virtual Machine Linux. A Virtual Machine running Ubuntu is used.
  NOTE: If a virtual Machine is used, an external USB MicroSD must be used.

B. MicroSD Card Setup and piCore Installation

1. Plug in the MicroSD card and wipe clean using gparted or similar partition manager.

2. Download piCore 7.0 from the following link on the Linux machine. Newer versions should be compatible, but minor complications may occur.

3. Unzip the downloaded file. There should be a .img file along with a checksum and a readme.

4. Using terminal and the "DD" command, flash the image to the MicroSD card. Then run the "sync" command to prevent data loss after ejection. The command used is shown below.

   $ sudo ddb s=1M if=/home/ee/Downloads/piCore7.0.img of=/dev/mmcblk0 && sync

5. Unmount and remount the MicroSD card using the "umount" then the "mount" commands.

6. Using gparted resize the ext4 partition to make larger ~1GB. Then add another ext4 partition filling the remainder of the card. Finally eject the card safely.

7. Put the MicroSD card in the Raspberry Pi and connect the HDMI monitor, USB keyboard, Ethernet, and power. The device should boot and to a console.

8. Install the "nano" text editor with the following command.

   $ tce-load –iw nano.tcz

Then persist the changes using the following command.

$ filetool.sh –b

C. *Wireless Driver Setup*

1. Add the following packages using the *tce-load* command. When finished persist the changes using *filetool.sh –b*.
    a. wifi.tcz
    b. usbutils.tcz
    c. firmware-ralinkwifi.tcz

2. Setup the Wi-Fi driver to load at startup by adding *modprobe –v /sbin/rt2800usb* at the end of the boot script located in /opt/bootlocal.sh using nano. Make sure to run nano as root and persist the change when completed.

3. Reboot the system and check if working by running *iwconfig* and examine for "wlan0". Further testing can include running *wifi.sh* and checking if the device can connect to an AP.

D. *Wireless Access Point Setup*

1. Install "dhcpd.tcz" using the *tce-load* command.
2. Download the hostapd binary using the following command.

    $ wget http://www.adafruit.com/downloads/adafruit_hostapd.zip

3. Unzip and move the file to /home/tc/wifi

4. Create a file in the directory /home/tc/wifi called "dhcpd.conf" and enter the following data.

```
authoritative;
subnet 192.168.42.0 netmask 255.255.255.0 {
range 192.168.42.10 192.168.42.50;
option broadcast-address 192.168.42.255;
option routers 192.168.42.1;
default-lease-time 600;
max-lease-time 7200;
option domain-name "local";
option domain-name-servers 8.8.8.8, 8.8.4.4;}
```

5. Create a file in the directory /home/tc/wifi called "hostapd.conf" and enter the following data.

```
interface=wlan0
driver=rtl871xdrv
ssid=Pi_AP
```

```
hw_mode=g
channel=6
macaddr_acl=0
ignore_broadcast_ssid=0
```

6. Create a file in the directory /home/tc/wifi called “wlan0.sh” and enter the following data.

   ENTER SCRIPT DATA HERE

7. Make the script executable using *$ sudo chmod 755 /home/tc/wifi/wlan0.sh*

8. Persist the changes using *filetool.sh –b*

9. Test by running “wlan0.sh”.

E. *Web Server Setup*

1. Install lighttpd using *$ sudo tce-load –iw lighttpd*

2. Create directories /home/tc/http and /home/tc/http/data

3. Create blank file /home/tc/http/data/data.txt

4. Create file /home/tc/http/lighttpd.conf and insert the following data

```
server.document-root = "/home/tc/http"
server.port = 80
index-file.names = ( "index.html" )
```

5. Create file /home/tc/http/index.html and insert the following data

```
<html>
<head>
<title>UART Motor Data</title>
</head>

<body>

<h1>Please click the link to download the latest motor
data.</h1>
<a href='data/data.txt'>Data</a>

</body>
</html>
```

6. Persist the changes using *filetool.sh –b*

7. Run the server using the following command
   $ sudo lighttpd –f /home/tc/http/lighttpd.conf –m /usr/local/lib/lighttpd

F. *UART Communication Setup*

1. Open /opt/bootlocal.sh as root and comment out any lines referring to “ttyAMA0” and “ttyAMA1”

2. Open /etc/inittab and comment out any lines referring to “ttyAMA0” and “ttyAMA1”

3. Open /opt/.filetool.lst and check if “/etc/inittab” is listed. If not, add at the end.

4. Create directory /home/tc/uart

5. Persist the changes using *filetool.sh –b*

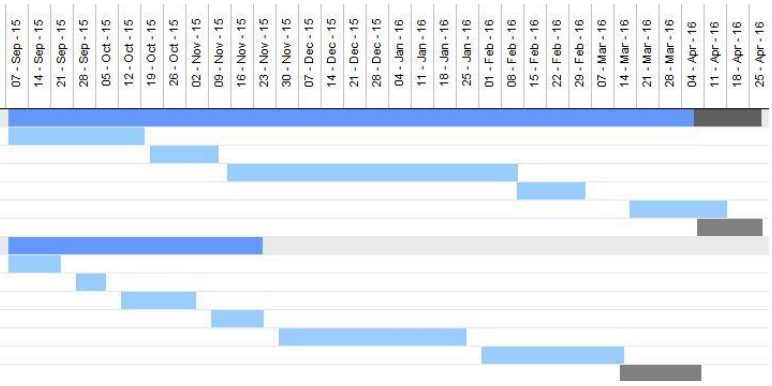
6. Using a virtual machine or linux based PC, copy cross-compiled program to /home/tc/uart
   Note: See Appendix C for UART C code.

7. Run as root by navigating to /home/tc/uart and running *./filename*

APPENDIX C: SCHEDULE

Table VI: Gantt Chart

WBS	Tasks	Task Lead	Start	End	Duration (Days)	% Complete	Working Days	Days Complete	Days Remaining
1	SIS	Joe	9/8/15	4/28/16	234	91%	168	213	21
1.1	Task 1		9/8/15	10/19/15	42	100%	30	42	0
1.2	Task 2		10/22/15	11/11/15	21	100%	15	21	0
1.3	Task 3		11/15/15	2/12/16	90	100%	65	90	0
1.4	Task 4		2/13/16	3/4/16	21	100%	15	21	0
1.5	Task 5		3/19/16	4/17/16	30	100%	20	30	0
1.6	Task 6		4/9/16	4/28/16	20	0%	14	0	20
2	GIS	Tim	9/8/15	11/25/15	79	100%	57	79	0
2.1	Task 7		9/8/15	9/23/15	16	100%	12	16	0
2.2	Task 8		9/29/15	10/7/15	9	100%	7	9	0
2.3	Task 9		10/13/15	11/4/15	23	100%	17	23	0
2.4	Task 10		11/10/15	11/25/15	16	100%	12	16	0
2.5	Task 11		12/1/15	1/27/16	58	100%	42	58	0
2.6	Task 12		2/2/16	3/16/16	44	100%	32	44	0
2.7	Task 13		3/16/16	4/9/16	25	0%	18	0	25



APPENDIX D: CODE

Gateway Interface System

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>

int main()
{
	int uart_filestream = -1;
	uart_filestream = open("/dev/ttyAMA0", O_RDWR | O_NOCTTY);

	if (uart_filestream == -1)
	{
		printf("\nError: Unable to open UART\n");
	}

	struct termios options;
	tcgetattr(uart_filestream, &options);
	options.c_cflag = B115200 | CS8 | CSTOPB | CLOCAL | CREAD;
	options.c_iflag = IGNPAR;
	options.c_oflag = 0;
	options.c_lflag = 0;
	tcflush(uart_filestream, TCIFLUSH);

	tcsetattr(uart_filestream, TCSANOW, &options);

	//Declare variables for opening file
	unsigned char rx_buffer[1024]; //buffer to store received data
	int rx_length = -1; //length of received data
	FILE *fp; //file buffer
	char filename[25]; //name of file
	//int num = 0;

	while(1)
	{
		fp = fopen("/home/tc/http/data/data.txt", "w"); //open file

		if(fp == NULL)
		{
			printf("File %s does not exist.\n", filename);
		}

		int i = 0;
		for (i; i<1000; i++)
		{
				//Receive data
				if (uart_filestream != -1)
				{
					//Read the data
					rx_length = read(uart_filestream, (void*)rx_buffer, 255);
    //name of filestream, buffer to store, max bytes

					if (rx_length < 0) //checks if error
					{
						printf("error\n");
						printf("Error: No bytes received.\n");
						fflush(fp);
					}
					else if (rx_length == 0) //checks for unread data
					{
						printf("Error: No data waiting\n");
						fflush(fp);
					}
					else //data is available
					{
```

```
                                        rx_buffer[rx_length] = '\0'; //adds NULL character
at end

                                        printf("%i bytes read : %s\n", rx_length,
rx_buffer); //displays data on screen
                                        fprintf(fp, "%s\n", rx_buffer); //prints data to
file
                                        fflush(fp);
                                    }
                                }
            }
            fclose(fp);
    }

    close(uart_filestream);
}
```

Sensor Interface System

```
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <asf.h>

#include "sysclk.h"
#include "board.h"
#include "print_funcs.h"
#include "tc.h"
#include "gpio.h"
#include "adc.h"
#include "usart.h"
#include "conf_example.h"

static volatile uint8_t done = 0;
static volatile uint16_t channel = 0;
static volatile uint32_t buffer[6000] = {0};

#define TWIM                (&AVR32_TWIM0)
#define TARGET_ADDRESS      0x32
#define TARGET_ADDR_LGT     7
#define VIRTUALMEM_ADDR     0x123456
#define TWIM_MASTER_SPEED   50000

// GPIO ADC MAPPING
const gpio_map_t ADC_GPIO_MAP = {
    #if defined(ADC_0_CHANNEL)
    {ADC_0_PIN, ADC_0_FUNCTION},
    #endif
    #if defined(ADC_1_CHANNEL)
    {ADC_1_PIN, ADC_1_FUNCTION},
    #endif
    #if defined(ADC_2_CHANNEL)
    {ADC_2_PIN, ADC_2_FUNCTION},
    #endif
    #if defined(ADC_3_CHANNEL)
    {ADC_3_PIN, ADC_3_FUNCTION},
    #endif
    #if defined(ADC_4_CHANNEL)
    {ADC_4_PIN, ADC_4_FUNCTION},
    #endif
    #if defined(ADC_5_CHANNEL)
    {ADC_5_PIN, ADC_5_FUNCTION},
    #endif
};

#if !defined(EXAMPLE_TC) || !defined(EXAMPLE_TC_IRQ)
#error The TC preprocessor configuration to use in this example is missing.
#endif

// Timer Flag
volatile static bool update_timer = true;
```

```
int count = 0;
int i = 0;

#if defined (__GNUC__)
__attribute__((__interrupt__))
#elif defined (__ICCAVR32__)
#pragma handler = EXAMPLE_TC_IRQ_GROUP, 1
__interrupt
#endif

static void tc_irq(void)
{
    // Increment counter
    count++;

    tc_read_sr(EXAMPLE_TC, EXAMPLE_TC_CHANNEL);

    // ADC CHANNELS

    adc_start(&AVR32_ADC);
    buffer[0] = adc_get_value(&AVR32_ADC, ADC_0_CHANNEL);
    adc_get_status(&AVR32_ADC, 0);
    // specify that an interrupt has been raised
    update_timer = true;

    //toggle LED to test timings
    if (count == 600  && adc_get_status(&AVR32_ADC, 0))
    {
            LED_Toggle(LED0);
            count = 0;
    }
}

// ADC interrupt
static void ADC_irq(void)
{
    buffer[i] = (adc_get_value(&AVR32_ADC, channel));
    i++;
    if (i == 6000)
    {
            i = 0;
    }
    if (channel == 5)
    {
            channel = 1;
            done = 1;
    }
    else
    {
            channel++;
    }
}

//TC initialization
static void tc_init(volatile avr32_tc_t *tc)
{
            static const tc_waveform_opt_t waveform_opt = {
            // Channel selection.
            .channel  = EXAMPLE_TC_CHANNEL,
            // Software trigger effect on TIOB.
            .bswtrg   = TC_EVT_EFFECT_NOOP,
            // External event effect on TIOB.
            .beevt    = TC_EVT_EFFECT_NOOP,
            // RC compare effect on TIOB.
            .bcpc     = TC_EVT_EFFECT_NOOP,
            // RB compare effect on TIOB.
            .bcpb     = TC_EVT_EFFECT_NOOP,
            // Software trigger effect on TIOA.
            .aswtrg   = TC_EVT_EFFECT_NOOP,
            // External event effect on TIOA.
            .aeevt    = TC_EVT_EFFECT_NOOP,
```

```
            // RC compare effect on TIOA.
            .acpc     = TC_EVT_EFFECT_NOOP,
            .acpa     = TC_EVT_EFFECT_NOOP,
            /*
             * Waveform selection: Up mode with automatic trigger(reset)
             * on RC compare.
             */
            .wavsel   = TC_WAVEFORM_SEL_UP_MODE_RC_TRIGGER,
            // External event trigger enable.
            .enetrg   = false,
            // External event selection.
            .eevt     = 0,
            // External event edge selection.
            .eevtedg  = TC_SEL_NO_EDGE,
            // Counter disable when RC compare.
            .cpcdis   = false,
            // Counter clock stopped with RC compare.
            .cpcstop  = false,
            // Burst signal selection.
            .burst    = false,
            // Clock inversion.
            .clki     = false,
            // Internal source clock 3, connected to fPBA / 8.
            .tcclks   = TC_CLOCK_SOURCE_TC3
    };
    // Options for enabling TC interrupts
    static const tc_interrupt_t tc_interrupt = {
            .etrgs = 0,
            .ldrbs = 0,
            .ldras = 0,
            .cpcs  = 1, // Enable interrupt on RC compare alone
            .cpbs  = 0,
            .cpas  = 0,
            .lovrs = 0,
            .covfs = 0
    };
    // Initialize the timer/counter.
    tc_init_waveform(tc, &waveform_opt);
    tc_write_rc(tc, EXAMPLE_TC_CHANNEL, (sysclk_get_pba_hz() / 8 / 600));
    // configure the timer interrupt
    tc_configure_interrupts(tc, EXAMPLE_TC_CHANNEL, &tc_interrupt);
    // Start TC
    tc_start(tc, EXAMPLE_TC_CHANNEL);
}

int main(void)
{
    volatile avr32_tc_t *tc = EXAMPLE_TC;
    sysclk_init();
    gpio_local_init();
    // Enable GPIO pins for ADC
    gpio_enable_module(ADC_GPIO_MAP, sizeof(ADC_GPIO_MAP) / sizeof(ADC_GPIO_MAP[0]));
    sysclk_enable_peripheral_clock(EXAMPLE_TC);
    // Initialize the USART module for trace messages
    init_dbg_rs232(sysclk_get_pba_hz());

    AVR32_ADC.mr |= 0x1 << AVR32_ADC_MR_PRESCAL_OFFSET;
    adc_configure(&AVR32_ADC);

//ADC ENABLING
adc_enable(&AVR32_ADC, ADC_0_CHANNEL);
adc_enable(&AVR32_ADC, ADC_1_CHANNEL);
adc_enable(&AVR32_ADC, ADC_2_CHANNEL);
adc_enable(&AVR32_ADC, ADC_3_CHANNEL);
adc_enable(&AVR32_ADC, ADC_4_CHANNEL);
adc_enable(&AVR32_ADC, ADC_5_CHANNEL);

// Disable the interrupts
cpu_irq_disable();
    // Initialize interrupt vectors.
    INTC_init_interrupts();
```

```
    INTC_register_interrupt(&tc_irq, EXAMPLE_TC_IRQ, EXAMPLE_TC_IRQ_PRIORITY);
    INTC_register_interrupt(&ADC_irq, AVR32_ADC_IRQ, 0x00000000);
    // Enable the interrupts
    cpu_irq_enable();
    // Initialize the timer module
    tc_init(tc);

int c = 0;
int b = 0;

    usart_write_char(DBG_USART, 90);
    while(1)
    {
            c= usart_getchar(DBG_USART);
            if (c == 32)
            {
                    usart_putchar(DBG_USART, 32);
                    usart_putchar(DBG_USART, buffer[b]);
                    usart_putchar(DBG_USART, buffer[b+1]);
                    usart_putchar(DBG_USART, buffer[b+2]);
                    usart_putchar(DBG_USART, buffer[b+3]);
                    usart_putchar(DBG_USART, buffer[b+4]);
                    usart_putchar(DBG_USART, buffer[b+5]);
                    b = b + 6;

                    if (b > 6000){
                            b = 0;
                    }

                    c=0;
                    }
            }
}
```